Global-Local Word Embedding for Text Classification

by

Mehran Kamkarhaghighi

A Thesis Submitted to the
School of Graduate and Postdoctoral Studies in Partial
Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Electrical and Computer Engineering

Department of Electrical, Computer and Software Engineering

Faculty of Engineering and Applied Science

University of Ontario Institute of Technology

Oshawa, Ontario, Canada
April 2019

THESIS EXAMINATION INFORMATION
Submitted by: **Mehran Kamkarhaghighi**

Doctor of Philosophy in **Electrical and Computer Engineering**

Thesis title: Global-Local Word Embedding for Text Classification

An oral defense of this thesis took place on March 27, 2019 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Walid Morsi Ibrahim
Research Supervisor	Dr. Masoud Makrehchi
Examining Committee Member	Dr. Shahryar Rahnamayan
Examining Committee Member	Dr. Qusay H. Mahmoud
University Examiner	Dr. Jeremy S. Bradbury
External Examiner	Dr. Chen (Cherie) Ding, Ryerson University

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

Only humans can understand and comprehend the actual meaning that underlies natural written language, whereas machines can form semantic relationships only after humans have provided the parameters that are necessary to model the meaning. To enable computer models to access the underlying meaning in written language, accurate and sufficient document representation is crucial. Recent word embedding approaches have drawn much attention to text mining research. One of the main benefits of such approaches is the use of global corpuses with the generation of pre-trained word vectors. Although very effective, these approaches have their disadvantages, namely sole reliance on pre-trained word vectors that may neglect the local context and increase word ambiguity. In this thesis, four new document representation approaches are introduced to mitigate the risk of word ambiguity and inject a local context into globally pre-trained word vectors. The proposed approaches, which are frameworks for document representation while using word embedding learning features for the task of text classification, are: Content Tree Word Embedding; Composed Maximum Spanning Content Tree; Embedding-based Word Clustering; and Autoencoder-based Word Embedding.

The results show improvement in the F_score accuracy measure for a document classification task applied to IMDB Movie Reviews, Hate Speech Identification, 20 Newsgroups, Reuters-21578, and AG News as benchmark datasets in comparison to using three deep learning-based word embedding approaches, namely GloVe, Word2Vec, and fastText, as well as two other document representations: LSA and Random word embedding.

Keywords: Document Representation; Word Embedding; Text Classification; Deep Learning; Neural Networks

AUTHOR'S DECLARATION

Dedicated to my wife, without whose support and encouragement none of this would have been possible.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor, Dr. Masoud Makrehchi, for offering me the opportunity to work with him and for continuously supporting my Ph.D. studies and related research, as well as for his patience, motivation, and sharing of his immense knowledge. I could not imagine having a better advisor and mentor for my doctoral studies.

In addition to my advisor, I would like to thank the rest of my thesis committee, Dr. Shahryar Rahnamayan, Dr. Qusay H. Mahmoud, and Dr. Jeremy S. Bradbury, for not only their insightful comments and encouragement, but also for all the hard questions that encouraged me to widen my research from various perspectives.

My sincere thanks also go to my friends in SciLab, Dr. Somayyeh (Bahar) Aghababaei, Mahboubeh (Tara) Ahmadalinezhad, Fateme Azimlou, Iuliia Chepurna, Dr. Eren Gultepe, Neil Seward, and Afsaneh Towhidi, without whose precious support it would not have been possible to conduct this research.

I would also like to extend my thanks to my dear friends, Dr. Roozbeh Jalali and Dr. Reza Mohammad Alizadeh, for their encouragement and support throughout my research.

I also wish to express my appreciation to Mrs. Catherine Lee, for editing and proofreading this thesis.

Finally, I would like to express my greatest gratitude to my family: my dearest wife, Solmaz, and my father and my mother for their unconditional support during this period of my life.

STATEMENT OF CONTRIBUTIONS

I hereby certify that I am the sole author of this thesis and I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis.
In all cases, I (Mehran Kamkarhaghighi) was the main investigator for each of the co-authored studies.
Part of the work described in Chapter 2 as the literature review was previously published as:

M. Kamkarhaghighi, E. Gultepe, and M. Makrehchi, "Deep Learning for Document Representation," in Handbook of Deep Learning Applications: Springer, 2019, pp. 101-110.

The other part of the work described in Chapter 3 as the CTWE approach has been published as:

M. Kamkarhaghighi and M. Makrehchi, "Content tree word embedding for document representation," Expert Systems with Applications, vol. 90, pp. 241-249, 2017.

Table of Contents

List of Tables

List of Figures

LIST OF ABBREVIATIONS AND SYMBOLS

1D-CNN	One Dimensional Convolutional Neural Network
20 NG	20 Newsgroups
AbWE	Autoencoder-based Word Embedding
b	Scalar Bias
BERT	Bidirectional Encoder Representations from Transformers
C	Cluster
CMSCT	Composed Maximum Spanning Content Tree
CNN	Convolutional Neural Network
cov	Covariance
CTWE	Content Tree Word Embedding
d	Elements in Word Vector
DAN	Deep Averaging Network
DNN	Deep Neural Network
DT	Decision Tree
e	Number of Edges
EbWC	Embedding-based Word Clustering
ELMo	Embeddings from Language Models
ESA	Explicit Semantic Analysis
EULA	End User License Agreement
FN	False Negative
FP	False Positive
GloVe	Global Vector
GNB	Gaussian Naïve Bayes
GRU	Gated Recurrent Unit
HSI	Hate Speech Identification
ICA	Independent Component Analysis
IMDB	Internet Movie Database
k-NN	k-Nearest Neighbor
LDA	Latent Dirichlet Allocation
LR	Logistic Regression

LSA	Latent Semantic Analysis
LSI	Latent Semantic Indexing
LSTM	Long Short-Term Memory
m	The Vocabulary Size of Each Document
M	Term Document Matrix
MLP	Multilayer Perceptron
MSCT	Maximum Spanning Content Tree
MST	Maximum Spanning Tree
n	The Depth of the Word in the Content Tree
N	Number of Words
NB	Naïve Bayes
NLP	Natural Language Processing
O	Algorithm Complexity (worst-case scenario)
PCA	Principal Component Analysis
PV-DBOW	Distributed Bag of Words Version of Paragraph Vector
PV-DM	Distribution Memory Model of Paragraph Vectors
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RF	Random Forest
RNN	Recurrent Neural Networks
ROC	Receiver-Operating Characteristic
S	Diagonal Matrix
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TF-IDF	Term Frequency - Inverse Document Frequency
TN	True Negative
TP	True Positive
U	Orthogonal Matrix
USE	Universal Sentence Encoder
v	Elements in EbWC Word Vector
Vec	Vector Representation

V^T	Orthogonal Matrix
W	Word
X	Word Co-Occurrence Matrix
σ	Standard Deviation
δ	Decay Factor
η	Comparison Function

Chapter 1. Introduction

1.1 Overview

Text mining is now broadly applied to a wide range of applications, including information retrieval, security, social media, and marketing. In this domain, the performance and computational cost of tasks such as classification, clustering, content analysis, and machine translation are completely dependent on document representation [1]. The bag-of-words approach is one of the first and most popular document representation models in the field of natural language processing (NLP). In this approach, each document is represented as a bag-of-words [2]. While this approach is fast and simple, and has a low-computational cost, it disregards grammar and even word order. It also suffers from the "Curse of Dimensionality" in that a short sentence for presentation needs a very high dimensional feature vector, which is also sparse. In this situation, classifiers lose their power of discrimination. This is the point where feature selection and feature transformation tools, such as Principal Component Analysis (PCA) [3], Latent Dirichlet Allocation (LDA) [4], Independent Component Analysis [5], Latent Semantic Indexing [6], and Document Frequency [7], come into account. An alternative to the bag-of-words is the word embedding approach, in which words or phrases are mapped to the low-dimensional vectors of a continuous space. Word2Vec [8], Global Vectors (GloVe) [9], and fastText [10] are three successful deep learning-based word embedding models. In order to accurately work, these models should be trained with vast corpuses. In summary, these models can present an acceptable vector for each word in training data.

Calculating the average and the summation of word vectors are the two common approaches that are used to represent a document, but these approaches do not consider the context of the document. For example, the word vector for "jaguar" as a car is equal to its word vector as an animal. The Doc2Vec model [11] presents a vector for each document or paragraph that is trained according to local data. The Doc2Vec model does not use background knowledge but it can involve the context of the document. The drawback of this model is the high computational cost of creating a model each time, while Word2Vec, GloVe, and fastText create the respective models according to the training corpus one time only. Another problem that the Word2Vec, GloVe, and fastText models cannot overcome is that of ignoring the relationship between terms that do not literally co-occur. In order to solve these problems, the use of ontology-based vectors is suggested [12]. Taxonomy is the backbone of ontology, and using ontology can solve the problem of data sparseness by replacing words with concepts [13]. The main objective of this thesis is to present low cost and accurate document representations that simultaneously use background knowledge and local data.

A named standard method for information retrieval (IR) tasks is Term Frequency - Inverse Document Frequency (TF-IDF), which is a basic method based on word count. A combination of this method and a classifier such as the Support Vector Machine (SVM) can serve as the state-of-the-art for text classification tasks [14, 15].

Similar to the bag-of-words, the TF-IDF is a high-dimensional text representation method where all the vector elements are independent whereas, in reality, the words occur in a highly correlated way. A low-dimensional representation approach is Latent Semantic Analysis (LSA) [16], which is based on the singular value decomposition (SVD) method.

Due to the variance of phrases and sentences, document representation is a more challenging problem regarding word representation.

Calculating the unweighted average of embeddings of all the words that occur in a text [17] is the most popular but not the best representation of a document from pre-trained word embedding. As an example, the representation document vector for the sentence "I like Apple computers" and "I prefer a green apple" are close to each other in the vector space.

In this study, four novel methods are described in Chapter 3: Content Tree Word Embedding (CTWE); Composed Maximum Spanning Content Tree (CMSCT); Embedding-based Word Clustering (EbWC); and Autoencoder-based Word Embedding (AbWE). These approaches create a document representation for each word in the training vocabulary based on the Word2Vec, GloVe, fastText, LSA, and Random word embedding models, then use the average of the updated word vectors for document representation.

The first approach, CTWE, employs a semi-taxonomy structure named content tree, and subsequently updates the word embedding vectors.

The second approach, MSCT, is proposed in order to select the root based on the node degree and then generate the maximum spanning tree. CMSCT, another version of this approach, which does not require a high amount of memory to generate the fully connected graph of all the words in the training vocabulary, is defined. This approach generates a small-size spanning tree for each document and then generates a training data spanning tree by combining them. In the last step, the word vectors are updated based on their location in the maximum spanning tree.

The third approach, EbWC, uses the clustering method for extracting the conceptual structure of the context. Each element of the new word embedding is the distance from the centroid of each word group cluster.

The final method, AbWE, uses autoencoder for dimensionality and noise reduction in the context. The main idea is to train an autoencoder to capture the training data concepts and then update the word vectors by encoding them.

1.2 Problem Statement

Text mining tasks that are based on a bag-of-words representation cannot guarantee satisfactory results [18]. By using the bag-of-words approach, the word order and grammar in the context are lost. The high dimension of the feature vector in the bag-of-words approach can also cause high computational complexity. In general, the use of words as the only feature involves various limitations such as:

Synonymy, when different words and phrases indicate the same concept. As an example, the words “manufacture” and “make” refer to the concept of production [19].

Polysemy, when a word has several meanings; for example, the word “apple” as a company or as a fruit [19]. In some cases, the words cannot be used independently. As an example, the phrase “passed away” refers to death, but either word on its own is not enough [19].

Some documents do not contain indicator terms. An example of this is the phrase “an armed man took the money”, referring to robbery. However, none of the words are good indicator terms [19].

Using word-level embeddings such as the Word2Vec or GloVe suffers from the out-of-vocabulary issue. In this case, the model ignores the words that do not appear in their training data. From another perspective, in order to work accurately, the Word2Vec, GloVe, and fastText models should be trained with vast corpuses. These models are able to suggest an acceptable vector for each word in the training corpus. Calculating the average and the summation of word vectors in a document are the two proposed approaches, both based on Word2Vec, GloVe, and fastText. These approaches use vectors that are calculated according to global knowledge rather than by considering the context of the document. As an example, the word vector for “blackberry” as a fruit is equal to its word vector as a company.

The Doc2Vec model presents a vector for each document or paragraph that is trained from local data. This approach does not use background knowledge but involves the context. A drawback of the Doc2Vec model is the high computational cost of model creation for each document, compared to Word2Vec, GloVe, and fastText, which create a onetime only model.

The main objective of this research is to provide document representations of the simultaneous use of global knowledge (pre-trained features) and local context for text classification, which is one of the most popular tasks in the domain of text mining. In addition, no dramatic improvement is expected since the focus of this research is to present new word embeddings that simultaneously employ global knowledge and local context.

The presented approaches, which are described in Chapter 3, are: Content Tree Word Embedding (CTWE)[20]; Composed Maximum Spanning Content Tree (CMSCT); Embedding-based Word Clustering (EbWC); and Autoencoder-based Word Embedding (AbWE).

These approaches create a document representation for each word in the training vocabulary based on the Word2Vec, GloVe, fastText, LSA, and Random word embedding models. For document representation, the average of the updated word vectors is calculated.

Figure 1.1 presents the criteria considered in this thesis and compares CTWE, AbWE, CMSCT, and EbWC to other document representation approaches.

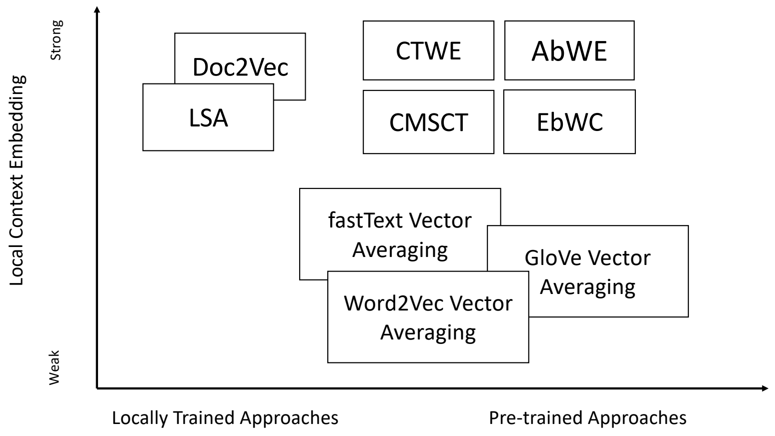

Figure 1.1. The four proposed approaches vs. other document representation approaches

1.3 Research Questions

- To what extent is the effect of the local context in the task of text classification?
- How can the local context be embedded into word embedding to improve the task of classification?

1.4 Research Objectives

Two main objectives are defined for this thesis:

- To provide document representation approaches that simultaneously use global knowledge (pre-trained features) and the local context.
- To evaluate the effectiveness of simultaneously using the local context and global knowledge in word embedding for the task of text classification.

1.5 Research Contribution

This thesis provides four different approaches to improving three word embedding-based word representations, namely: Word2Vec, GloVe, and fastText. The aim of this research is to provide approaches that can present enhanced document representation by employing information in the local context, as well as establishing the benefits of using pre-trained global models.

In summary, the main contributions of this thesis are:

- A study of word embedding, and document representation methods is provided in Chapter 2.
- In Chapter 3, the following four novel approaches are provided to generate new document representations that use global knowledge as well as the local context:
    - Content Tree Word Embedding
    - Composed Maximum Spanning Content Tree
    - Embedding-based Word Clustering
    - Autoencoder-based Word Embedding
- A description of how the four novel approaches are implemented and generate new word embeddings is provided in Chapter 4. The new word embeddings are evaluated, and the results are analysed and discussed in Chapter 5.

1.6 Thesis Outline

This thesis consists of six chapters. Following the introduction, which contains an overview, Chapter 1 presents the problem statement, research challenges, objectives and contribution of this work. A review of relevant literature is provided in Chapter 2. The proposed document representation methods are introduced in Chapter 3, while Chapter 4 describes the methodology. Experimental results are discussed in Chapter 5. Chapter 6 concludes the study by summarizing the main findings and suggesting future areas of exploration.

Chapter 2. Literature Review

2.1 Introduction

In this chapter, a comprehensive literature review of the related research in the domain of word embedding and document representation is presented. The document representation methods used before word embedding are categorized as traditional document representation methods. The Word2Vec, GloVe, fastText, LSA and Doc2Vec approaches are presented in detail. Other combination methods are later described, following which taxonomy induction approaches are reviewed.

2.2 Traditional Document Representations Methods

Using the words in a document is the most instinctive approach for document representation. Since 1954, different document representations have been introduced, the earliest of which is bag-of-words by Harris [2]. In this model, the document is presented as a fixed length vector where the length is the number of unique selected terms in the document repository. Stop word filtering, stemming, and lemmatization are optional filtering methods that can be used to improve the quality of representation. Using single words as features suffers from the "Curse of Dimensionality" in that a very high dimensional feature vector is needed to represent a short sentence. Losing the semantic meaning of the text and phrases is another limitation of the bag-of-words approach.

The n-gram document representation uses a continuous sequence of n words, which are chunks of words used as features to represent a document. Each element of the document representation vector is a set of two or more neighboring words in a document repository. Another similar approach is using a fixed chunk of letters where unique chunks of letters represent elements in the feature vector for each document. This approach has applications in language identification and spelling error detection. For documents with references, such as scholarly documents or web-pages with hyperlinks, another method can be used to represent the document, as references are related to the contents of the document. The weight of the references can be modified based on frequency of use and location in the document. In this approach, each reference is one dimension of the feature vector, which has a much lower dimensionality in comparison to the bag-of-words and n-grams approaches. [21]

Explicit Semantic Analysis (ESA) is a mixed approach based on the contents and references in a document. In ESA, the similarity of documents is calculated based on their reference sets. In the feature vector, each element is weighted in relation to specific documents within the reference sets[22].

The similarity measure based on the compression approach [23] is also used for computing representation of documents. This method is based on the hypothesis that the similarity of two files can be estimated by comparing the compressed size of the concatenated version of the two files and the summation of the compressed size of each file. The elements of the representation vector are the similarity between documents in the repository.

In a study by Maas et al. [24], a word representation was introduced to capture the semantic and sentiment meaning of words. The model created vectors by an unsupervised probabilistic-based approach. Words that come together in most documents have similar representation. In the next phase, by using a supervised learning-based method, the sentiment of words comes into account and the model is trained.

Yang et al. [25] extracted key terms from training documents based on the Gini Index, Information Gain, Mutual Information, Odds Ratio, Ambiguity Measure, and the Darmstadt Indexing Approach association factor. Extracted key terms were used for the document representation, which is known as KT-Of-Doc. In the proposed approach, each document is shown by the terms that appear in the document to enhance the effect of words and non-key terms that do not appear in the document to weaken the effect of the non-key terms. In another study by Yang et al. [26], the attention network of a document was created by detecting the more and less important parts of the content. The extracted network was used for the document representation. The proposed model had two levels: word level, which detects important words, and sentence level, which detects and relates important sentences. The introduced representation was used for text classification tasks.

Wei et al. [1] tried to solve the problem of meaningless latent representation of documents. Discriminative neighbors were defined and an autoencoder trained by minimizing the Bernoulli Cross-Entropy Error. This autoencoder was used to create a new document representation method.

In a study by Lao and Jagadeesh [27], legal questions were classified into 16 legal areas. The bag-of-words, bag-of-bigrams, TF-IDF technique, and average Word2Vec vectors of questions were used as features and compared to five different classifiers for this classification task. The Linear SVM classifier gained the best results in comparison to the other four models: Logistic Regression (LR), Multi nominal Naïve Bayes (NB), SVM with stochastic gradient descendent (SGD), and one-layer neural network.

In a study by Kim et al.[28], three approaches for document representation were introduced, based on the Word2Vec vector of content words. The first approach was average pooling, the second was class-specific Gaussian Mixture distribution and the third was Semantic Space Allocation, which uses global Gaussian Mixture Model components. Average Pooling had the best result in comparison to the other two methods and outperformed the traditional LDA method when applied to a Chinese article classification task. Bernotas et al. [12] used a tagging-based document representation method and improved a clustering task by using ontology. Based on the results, tagging-based representation had a negative impact on small scale documents, but if the document was large scale, the results were better than the word-based document representation.

Lu et al. [29] created a cluster's universe and, by segmenting documents into topics and assigning topics to the clusters, a relationship was made between each document and the clusters. The documents were assigned as members of the strongest relative cluster and associated with the second most strongly related cluster.

In a study by Socher et al. [30], a Neural Tensor Network was used to extract the relationship between entities in a knowledge base. The vector representation of words was used to calculate the average of the word vectors in an entity. The learning relation classifier and the entity representation were joined. This approach is used for knowledge-based completion tasks, which are useful in query expansion, question answering, and information retrieval.

2.3 Word2Vec

Word co-occurrence is at the heart of several machine learning algorithms, including the recently introduced Word2Vec by Mikolov et al. [8]. Word2Vec is a two-layer neural network-based approach that learns embedding for words. Negative sampling has been used in the softmax step of the output layer. The objective function maximizes the log probability of a context word ($\boldsymbol{w_O}$), given its input words ($\boldsymbol{w_I}$). By using negative sampling, the objective function is to maximize the dot product of $\boldsymbol{w_I}$ and randomly selected negative words, while minimizing the dot product of $\boldsymbol{w_I}$ and $\boldsymbol{w_O}$. The output is a vocabulary of words from the original document and one n-dimensional fixed-size vector representation. Co-occurring words in the training corpus are located adjacent to each other in a vector space. Figure 2.1 [8]illustrates how Word2Vec creates word vector representation by use of two architectures: Continuous Bag of Words (CBOW) and Skip-gram. The CBOW architecture model, which can predict a word according to the surrounding context words, works faster than Skip-gram, which predicts the surrounding words by a center word, in a fixed-length window. For infrequent words, the Skip-gram architecture works better.

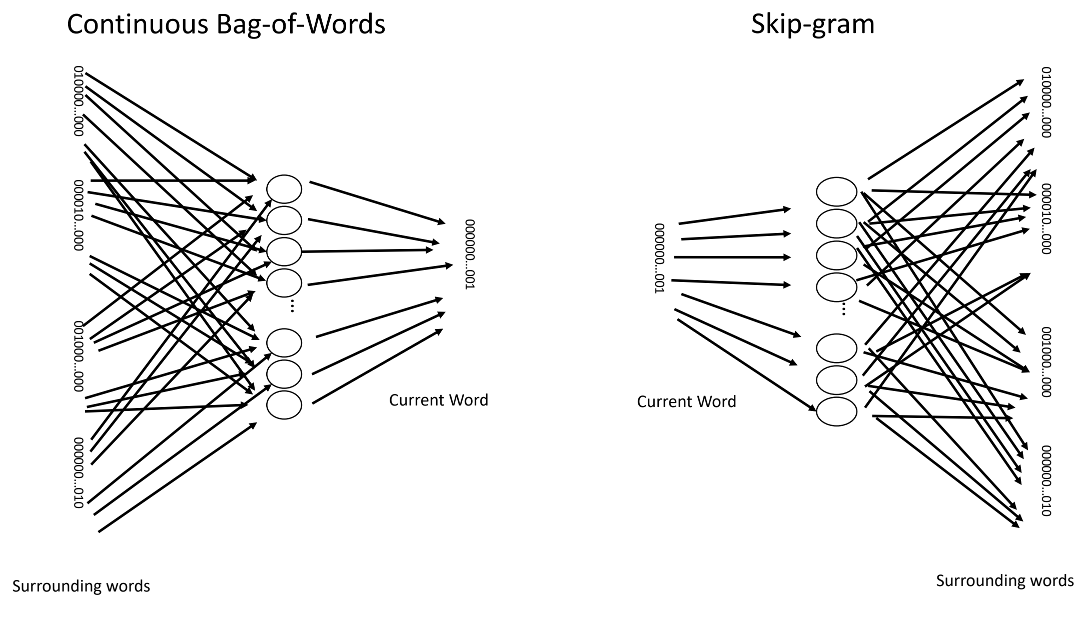

Figure 2.1. The Word2Vec architectures

Word2Vec generates vector representation only for words while, for document representation, a representation for the entire document is needed. Averaging or summation of all the word vectors of a given document can be a naive solution for creating document representation.

During the training phase, words that appear in similar contexts are grouped together in the same direction by this unsupervised learning algorithm.

In [31], it was highlighted that not only the direction but also the length of word vectors carries important information. The length of a vector merely reflects the frequency with which a word appears in the corpus and the similarity of the contexts in which the word appears. Accordingly, a word that is consistently used in a similar context will be represented by a longer vector than a word of the same frequency that is used in different contexts.

2.4 GloVe

Pennington et al. [9] introduced an unsupervised word embedding model, known as GloVe, for word representation.

GloVe tries to encode meaning as vector offsets in an embedding space. This model captures the frequency of word co-occurrences within a specific window in a large text corpus to generate linear dimensions of meaning and uses global matrix factorization and local context window methods. The model also offers a local cost function and includes a weighting function that is used to balance rare co-occurrences. Optimization methods are used to minimize the cost function.

Based on the hypothesis that similar words have similar distributions, it is expected that generally trained word vectors can be used to measure semantic similarity. Similar to Word2Vec, averaging the vectors of words in a document is an option for generating a fixed length vector for document representation.

In order to achieve GloVe word embedding, word co-occurrence information should first be collected as a word co-occurrence matrix (X). In this matrix, the value of the X_{ij} illustrates the number of times that word i was in the context (fixed window size) of the word j in the training data. The decay factor shown in Equation 2.1 is applied to reduce the weight of distant words:

$$\boldsymbol{Decay = 1/offset} \qquad \textbf{Equation 2.1}$$

In the next step, a soft constraint for each word pair is assigned by Equation 2.2:

$$\boldsymbol{Vec_i^T\, Vec_j + b_i + b_j = \log X_{ij}} \qquad \textbf{Equation 2.2}$$

where V_i represents the vector for the main word, V_j is the vector representation for the context word, and b_i and b_j are scalar biases for the main and context words.
The cost function is defined in Equation 2.3:

$$\boldsymbol{J = \sum_{i=1} \sum_{j=1} f(X_{ij})\, (Vec_i^T\, Vec_j + b_i + b_j - \log X_{ij})^2} \qquad \textbf{Equation 2.3}$$

In order to avoid learning only the form of extremely common word pairs, a weighting function f (Equation 2.4) is used:

$$\boldsymbol{f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{x_{max}})^a & if\ X_{ij} < x_{max} \\ 1 & therwise \end{cases}} \qquad \textbf{Equation 2.4}$$

The GloVe approach is different from Word2Vec. While Word2Vec is a "predictive" model, GloVe can be called a "count-based" model. A count-based model generates word vectors by applying dimensionality reduction methods over the co-occurrence matrix, while predictive models try to develop their capability to predict loss.

2.5 fastText

fastText, an extension of Word2Vec, was proposed in 2016 by Bojanowski et al. [10] from Facebook AI Research. While fastText learns the word representations, it considers the structure of the words, an approach that is useful for languages where words are morphologically similar. When it wants to represent rarely occurring words, this method is an advance on others. This feature allows the algorithm to identify prefixes, suffixes, stems, and other phonological, morphological and syntactic structures in a manner that does not rely on words being used in a similar context, thus represented in similar vector space.

The fastText approach uses the n-grams of words to train a neural network. As an example, the tri-grams for the word "hello" is hel, ell, and llo. The word embedding vector for "hello" will be the sum of all these n-grams. A word embedding for all the n-grams, given the training dataset, is generated. By having the word embedding of the n-grams, rare words can also be represented, since they may contain some of the n-grams.

In theory, fastText embeddings should work more accurately on syntactic analogies since they are mostly morphology-based. In a manner similar to Word2Vec, fastText learns word embeddings with the difference that it enriches word vectors with sub word information by using character n-grams of variable length.

It is expected that fastText will outperform Word2Vec and GloVe when the size of the dataset is small. FastText word embedding is based on continuous Skip-gram architecture, using n-gram at character level. A hashing function, called Fowler-Noll-Vo, is used to map n-grams to integers. This hashing function bonds the memory requirement of the model, which uses SGD in the negative log likelihood of solving the optimization problem.

In fastText architecture when there is a large number of classes, the hierarchical softmax [32] is used to calculate probability distribution over predefined classes. The use of linear classifiers is computationally expensive in this situation.

For word representation, a hashed version of n-gram, called the hashing trick, has been used[33]. The average of the word vectors is used as the document representation and then fed into a linear classifier, similar to the CBOW model [8].

2.6 LSA

Latent Semantic Analysis [16], or latent semantic indexing, is a document analyzing method to identify the concepts and underlying meaning of documents.

In the first step, the term document matrix ($M = n * m$ where m, the size of the row, is the number of terms and n, the column, is the number of documents while $M[i, j]$ illustrates the frequency of the term i, in the document j) is constructed. The SVD method is then applied to the matrix M to decompose it into three matrices, according to Equation 2.5. The S is the diagonal matrix while the U, $V^T$are the orthogonal matrices. Based on Equation 2.6, the k largest singular values of the three matrices will be used as the reduced dimension version of the M, which is the M_k:

$$M = U * S * V^T \qquad \textbf{Equation 2.5}$$

$$M_k = U_k * S_k * V_k^t \qquad \textbf{Equation 2.6}$$

Based on the research conducted by Naili et al. [34], the local and global weighting functions and settings have an important impact on the output of the SVD model which, in their case, was the topic segmentation task.

2.7 Doc2Vec

Le and Mikolov [11] presented Doc2Vec, also known as Paragraph2Vec. An extension of Word2Vec, the Doc2Vec model represents a document, sentence, and paragraph by a fixed-length vector. This unsupervised learning model can represent a document in two forms: Paragraph Vector with Distributed Memory (PV-DM), a complex model with more parameters that can achieve better results in comparison to the Distributed Bag of Words (PV-DBOW), which is simple and does not consider word order. The PV-DM approach considers the order of words in a paragraph and generates a vector that carries the information where other models, such as vector averaging or clustering, loses them.

PV-DBOW works in a largely similar way as Skip-gram with the difference that the input is a unique vector that represents the document and the order of words is ignored. PV-DM works in the same way as CBOW. The additional vectors used by PV-DM are a concatenation of document vectors and several target words. The objective is to use the concatenated document and word vectors to predict a context word.

The objective of a subsequent study by Hong [35] was to better the performance of Paragraph2Vec by use of two approaches: the addition of both a hidden layer and a tensor layer to the paragraph vector, such that it can interact with word vectors, both complexly and non-linearly.

The Doc2Vec model can acknowledge the context of a document but cannot benefit the background knowledge. However, the high computational cost of creating a model each time is a distinct weakness of this approach. In comparison, Word2Vec and GloVe allow pre-trained models to be used multiple times by fine-tuning the input feature vectors to a specific task.

Doc2Vec uses the Word2Vec model and adds another vector (Paragraph ID), as shown in Figure 2.2 [11]:

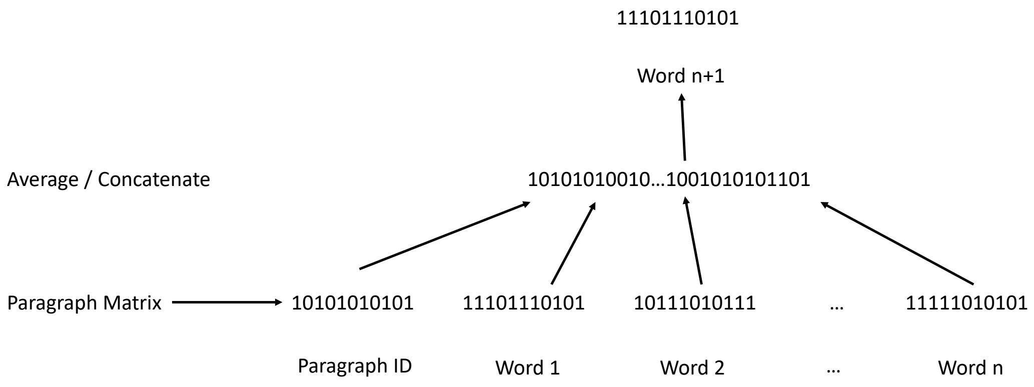

Figure 2.2. Doc2Vec PV-DM Model

The architecture of the PV-DM is similar to the CBOW model, with a new feature vector for the document. During the training phase, the document vector is also trained and can then represent the concept of the document.

Similar to Word2Vec, the other architecture, PV-DBOW, is inspired by Skip-gram, as illustrated in Figure 2.3[11].

The second architecture needs less memory and is faster than the PV-DM since it is does not save word vectors.

Le and Mikolov [11] recommended use of a combination of both architectures, although PV-DM is the superior method in most cases.

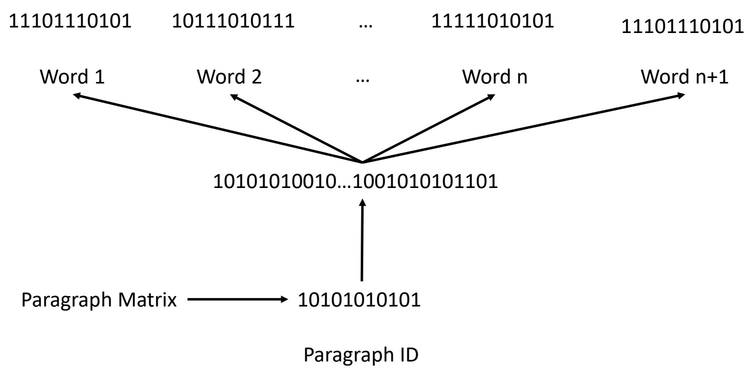

Figure 2.3. Doc2Vec PV-DBOW Model

2.8 Other Word Embedding Approaches

In 2015, Hong et al. [36] used the deep learning and Latent Dirichlet Allocation (LDA) approaches to detect anomaly sentences in End User License Agreement (EULA) legal texts. In the first step, topics were extracted by LDA from a EULA corpus; the words in the topics were then removed from the testing text. The Word2Vec vector of all the remaining words was calculated and a Word2Vec vector for each sentence was created. In the next step, agglomerative clustering (which is more successful in comparison to the K-mean clustering method) and the local outlier factor were used to detect abnormal sentences in the EULA text.

A 2015 study by Tai et al. [37] introduced a new representation for sentences that is a combination of Long Short-Term Memory (LSTM) and tree-structured network topologies. This model is known as Tree-Structured LSTM. Three classes of models represented the meaning: bag-of-words, sequence, and tree-structured models. In the bag-of-words model the model is independent from the sequence and cannot support all NLP tasks. Sequence models construct the token's sequence sensitive representation of sentences. Tree-structured models create each phrase or sentence from sub-phrases based on the given syntactic structure. This study proposes generating an alternative representation based on concepts in a document and then aggregating the results with the original word vectors.

Recently, Cer et al. [38] developed a transfer learning-based approach, called USE (Universal Sentence Encoder), for encoding text into embedding vectors. The presented encoder employs two encoding models: DAN (Deep Averaging Network), introduced for the first time by Iyyer et al. [39] in 2015, and Transformer-based sentence encoding by Vaswani et al. [40], introduced in 2017. Both models convert an English string into fixed dimensional embedding representation. To produce the embedding in DAN encoding, the embedding of the word and bi-grams are averaged together and passed through a feedforward deep neural network. This model targets efficient inference. The transformer-based sentence encoding uses an encoder sub-graph to compute context aware embedding for each word in the text and calculates an element-wise sum at each word position to produce an embedding vector for the entire document.

Zhu and Hu [41] presented a variation of doc2vec (the Distributed Bag of Words model), which is known as context aware document embedding. In their approach, each word occurrence is weighted based on its contribution in the context. This weighting allows document embedding to capture sub-topic level keywords to facilitate the learning process. Their computation and memory cost depend on the size of the text.

Mirowsky et al. [42], developed a non-linear multi-layer algorithm based on autoencoder architecture for text classification, information retrieval and topic modeling. This algorithm generates a compact document representation.

Ranzato and Szummer [43] provided an autoencoder-based document representation built on a semi-supervised approach. The authors mentioned that learning deep architecture works more efficiently with compact document representations. More compact representation needs less storage space and is computationally more efficient for indexing lookup procedures.

More recently, deeper neural architectures have been developed to generate these embeddings and to perform text classification tasks. Some of these architectures involve sequential information of text, such as LSTMs [44]. Le and Mikolov [11] developed a method to generate embeddings that outperform the traditional bag-of words [2] approach.

Extracting the list of nearest neighbors of a specific word or calculating the similarity distance between selected terms are examples of intrinsic evaluation methods. Previous studies show that different types of information can be encoded by different layers of LSTM [45, 46].

Levy et al. [47]showed that, when the hyperparameters of an algorithm are tuned, the results of all traditional and deep learning-based approaches are comparable. They claimed that the performance of word embedding and classification results are more dependent on hyperparameter optimization than the total approach. They compared four different word representations: Word2Vec's Skip-gram with negative sampling architecture; GloVe; SVD; and explicit representation, which is a Positive Pointwise Mutual Information (PPMI) matrix [48]. Based on their conclusions, for different tasks, different types of word embedding with different hyperparameter values could achieve the best results. As an example, in different experiments, Skip-gram outperformed GloVe, while the study by Pennington et al. [9] showed that completely opposite or deep learning-based word embeddings did not always outperform traditional count-based distributional methods.

In a comparative study by Altszyler et al. [49] between the LSA and Word2Vec, the results showed that, in cases with a low number of documents and low frequency of target words, the LSA works more accurately than Word2Vec.

In 2018, Peters et al. [50], introduced ELMo, a semi-supervised embedding from Language Models. By using bi-LSTM architecture, this deep contextualized word representation can capture complex characteristics of word use as well as characteristics of word use and variety across linguistic contexts. The ELMo model uses a concatenation of the vectors that are generated by the left-to-right LSTM and the right-to-left LSTM. These two vectors are independently trained. The authors showed that lower level LSTM architectures can compute syntax-based aspects of a word, while high level LSTM can capture context-dependent information of word meaning.

In 2018, Devlin et al. [51], from the Google AI lab, introduced a new word embedding named Bidirectional Encoder Representations from Transformers (BERT). Unlike ELMo, this architecture trains the vectors on the left and right contexts in all layers, so that one additional output layer can then tune the output. This architecture can be used for pre-training tasks as well as fine-tuning procedures. The authors claimed to have advanced state-of-the-art results for 11 NLP tasks, namely: question answering, Named-entity recognition, and next sentence prediction.

2.9 Taxonomy Induction

Taxonomy plays an important role in information systems, ontology learning, and the semantic web. Taxonomy is the backbone of ontology; using ontology can solve the problem of data sparseness by replacing words with concepts [13]. Ontology-based vector approach is suggested for using background knowledge and solving the problem of the high computational cost inherent in creating a model and ignoring the term relationship between terms that do not literally co-occur. [12].

The CTWE approach, which is described in 3.2.2, is inspired by creating a taxonomy of words for each document. While taxonomy is the backbone of ontology, using ontology can solve the problem of data sparseness by replacing words with concepts [13]. Significant research efforts have been made with respect to using different document representation models. In the introduction to a book edited by Buitelaar et al. [52], the focus was on ontology learning and describing a six layer model. The base layer of the model was comprised of terms. Term extraction was introduced as the first step of ontology learning. Acquisition of semantic terms in a language or between languages was described as the second layer of the model, which uses clustering-based techniques and the Latent Semantic Indexing (LSI) algorithm. The concepts layer formed the third layer, which was defined as the intentional definition of concepts, a set of instances, and a set of linguistic realizations. Taxonomy was placed as the fourth layer in the proposed model. The three paradigms of lexico-syntactic patterns, term clustering, and document-based notation of term subsumption are used to induce taxonomies. Extracting non-hierarchical relations was the fifth layer of the model, which focused on discovering new relations between known concepts. The sixth and highest layer was rule extraction, defined as: deriving lexical entailment rule extraction.

In another study, Nazar et al. [53], the authors mentioned two main automatic taxonomy strategies: the first group of strategies is lexico-syntactic, which is based on finding patterns in texts such as "is a", "consists of", and "belongs to". The second group is composed of quantitative strategies, which assume that hyponyms of a term occur in a context window, based on redundancy in texts because of the definition or characterizing statements. A strategy from the quantitative group is presented based on the asymmetry and transitivity of hypernym relations. A limited set of terms was assumed as seeds that are related to a specific domain; after searching these terms on the Internet, the co-occurrence of terms was calculated. The terms were arranged in the taxonomy, based on the asymmetric nature of the syntagmatic association of terms. Lu et al. [29] created a cluster's universe and, by segmenting documents into topics and assigning the topics into clusters, a relationship was made between each document and the clusters. The documents were assigned as members of the strongest relative cluster and associated with the second most strongly related cluster. In another study by Ben et al. [54] in 2016, "Is a" taxonomy was extracted from a Wikipedia categories graph. Two algorithms were used: an algorithm to split a "Direct Acyclic Graph" into sub graphs, and a second algorithm to merge the sub graphs. This approach uses the semantic similarity measure between terms and compares the results with WordNet.

Additional research studies that focused on cold start taxonomy induction were also conducted. In this context, the cold start procedure involves an automotive process of taxonomy induction that is purpose-built. In a very early study in 1992, Hearst [55] presented a new approach to finding relationships between words in a text corpus by finding patterns. The approach was based on bootstrapping to automatically finding patterns. In order to extract the taxonomy from an unstructured data extracted from the web, Sánchez and Moreno [56] presented a combination of three approaches: the Hearst, the noun phrase, and the web search methods. The authors mentioned that taxonomy construction is the first step in structuring domain knowledge as well as the web in comparison to other data sources such as dictionaries, databases, and news reports, which are unstructured, untrustworthy, noisy, and ambiguous. However, data from the web is vast in size and heterogeneous, as well as a real distribution of human knowledge. The authors tried to maximize the performance by a bootstrapping approach. In the first phase, the patterns were extracted by the Hearst approach and enriched by the noun verb approach. In the next phase, the most suitable candidates were selected and applied to web scale statistics by using a search engine. Human experts were also involved in the evaluation of the extracted taxonomy.

Alvarado and Arevalo [57] presented a text clustering-based approach for ontology learning. In the proposed approach, a verb-norm table was created and mutual information calculated for each pair of verbs and norms. In the next step, by using the Minipar parser, the dependency between the nouns was determined by using Clustering by Committee algorithms, and the inferred topic and hyponyms were identified. The Hearst approach, contextual information extraction, and web querying approaches were used to create a taxonomy for each topic. In a similar study by Woon and Madnick [58], a new measurable distance was introduced based on term co-occurrence, which can be used for taxonomy construction. The main characteristic of the research was the use of academic literature as the knowledge base and the use of Google scholar for obtaining the query. In the presented approach, the Kolmogorov complexity was used to calculate the distance between two terms. In a study by Meijer et al. [59], an automatic taxonomy construction framework was presented for extracting a taxonomy from a corpus in four steps: term extraction; filtering of relevant terms; the disambiguating phase; and finally, determining of hierarchical relations.

In 2007, Makrehchi and Kamel [60] used "Google distance" to find the relationship between terms to make a taxonomy. To create the term dependency and adjacency matrix, the normalized Google distance was calculated based on page count in the searched results of a term in the Google search engine. In 2006, Heymann and Garcia-Molina [61] converted two corpuses of tags into a hierarchical taxonomy of tags. For each new tag, similarity with the other tags was calculated and added under the most similar, if the cosine similarity was greater than the threshold. Otherwise, the new tag went under the taxonomy's root. This algorithm was applied to the Delicious and SiteUlike corpuses.

Bast et al. [62] used the PCA method to find types of relationships between terms. The types of relationships in the study were: unrelated; symmetrically related; asymmetrically related (when the first term is more specific than the second); and finally, asymmetrically related in the other direction. A term-term similarity matrix was created, and similarity of terms was calculated based on a co-occurrence matrix. In the next step, each term was ranked according to the optimal dimensions that were needed to best describe the similarity. The rank has been used to find the relationship between terms and to show which term covers another term. To justify the model, a normalized document-term matrix was created and the similarity between the terms was calculated. The authors mentioned that evaluation, whether automatic or manual, is difficult.

Yang and Callan [63] introduced five features for automatic taxonomy induction: (1), Contextual information: Global context Kullback–Leibler divergence (KL-Divergence), and Local context KL-Divergence; (2), Co-occurrence: Using Google search results for documents and sentences; (3), Syntactic dependency, such as the minipar syntactic distance, the modifier and the overlap for objects, subjects, and verbs; (4), Lexical syntactic patterns: Hyponym, sibling, and part of patterns; (5), Miscellanies: Word length difference, and definition overlap. Wikipedia and Google corpora have been used for training the model. The F_score was calculated for different types of relations. In another work by Yang and Callan [64], a new matrix between two terms, known as "the ontology metric", was calculated, providing a score to show the semantic distance of terms in a taxonomy. The terms were incrementally clustered, based on this new metric, using a clustering-based method with features described in [63]. The proposed approach tried to minimize changes in the taxonomy structure while inserting a new term.

Navigli et al. [13] extracted terms with a tool known as a "term extractor", and hyponym terms were then extracted by a World Class Lattices classifier. In the subsequent step, domains were filtered based on weights, and taxonomy induction was performed by using topological graph properties, which were produced in the previous steps. False roots, leaves and waiting edges were eliminated based on graph connectivity and path length. Finally, the results were compared with WordNet and two other studies.

In another effort to conduct automatic taxonomy extraction, Liu et al. [65] presented an approach to derive the taxonomy from a set of keywords by the Bayesian approach while efficiently using the knowledgebase of a search engine. The authors mentioned that manual taxonomy extraction is costly and very subjective. Finding a valid and accurate corpus for specific domains is not easy; some domains are fast changing while the data is also sparse. In a study into taxonomy extraction, Wu et al. [66] tried to discover frequent sequential patterns in a text. The output of the model, which was named the pattern taxonomy model, can demonstrate extracted sequential pattern relationships.

Garcia et al. [67] presented an approach for automatic taxonomy extraction from the Wikipedia categories structure. This approach, which is language independent, used syntactic, structural, and content based features for taxonomy extraction. In a study by Nguyen et al. [68], Wiktionary was used as the knowledgebase and the Hidden Markov Support Vector Machines approach was used for solving the problem of word ambiguity in the English, Vietnamese, and Korean languages. In another study, Saleem and Bellahsene [69] presented a tree mining-based approach for extracting mini taxonomies from existing ontologies, XML schemas, and Folksonomies with tree-based data structures. Makrehchi [70, 71] identified the dependency of terms calculated according to the inclusion index with a taxonomy extracted from a term to term matrix. By using the extracted taxonomy, sets of queries were generated and the ranking score of each generated query was calculated for each document. These calculated ranking scores were used as new feature vectors for clustering documents based on query. The results showed an improvement in comparison to the traditional bag-of-words method.

In a study by Socher et al. [30], the Neural Tensor Network was used to extract the relationship between entities in a knowledgebase. The vector representation of words was used to calculate the average of the word vectors in an entity. The learning relation classifier and the entity representation were joined. This approach is used for a knowledgebase completion task, which is useful in query expansion, question answering, and information retrieval tasks.

In 2018, Lai et al. [72] presented a combined approach that employed the pre-trained GloVe word embedding model and statistical information from an Is-A taxonomy for the task of semantic differentiation. The extracted features consisted of word frequency, co-occurrence word frequency and other statistical data from the Is-A taxonomy and word vectors, cosine distance, and L1 norm of vector difference from the GloVe pre-trained model that is passed through an SVM classifier.

2.10 Summary

In this section, various document representation and word embedding approaches are explored and reviewed. The primary approaches, namely bag-of-words and n-gram, suffer from the curse of dimensionality and are limited to the local context, while the word embedding-based approaches, which benefit from global knowledge, demand high computational power to be able to train over vast corpuses. Deep learning-based approaches benefit from global knowledge by adding an embedding layer to their complex architecture but, in terms of computational power, the cost of generating the model is high. In the following chapter, four novel approaches, which are able to inject a local context into globally pre-trained word vectors, are presented.

Chapter 3. Proposed Document Representation Approaches

3.1 Introduction

This chapter illustrates the four novel approaches to representing a document in a way that uses pre-trained word vectors as well as the document context. The document content tree, which is the foundation of the CTWE approach, is described and the CTWE is then presented. The MSCT and CMSCT are later introduced, followed by a full explanation of the EbWC and AbWC details and algorithms.

3.2 Content Tree Word Embedding

3.2.1 Document Content Tree

A document content tree is based on a semi-taxonomy representation of a document that shows the dependency between the terms in the document. In this proposal, in order to measure the dependency of terms, a correlation metric is used to evaluate the relationship between the deep learning-based generated vectors of the terms used in the document. Equation 3.1 illustrates the formula used to calculate the correlation between two 1-D arrays, X and Y:

$$Correlation\ (X,Y) = \frac{cov(X,Y)}{\sigma_X\ \sigma_Y}$$ **Equation 3.1**

where *cov* is the covariance, σ_X is the standard deviation of X, and σ_Y is the standard deviation of Y.

Figure 3.1 illustrates the process of adding a new word to the content tree in the early steps. The content tree induction algorithm assumes the first word (Word A) in the vocabulary of the content as the root and the second word (Word B) as its child. The correlation of the third word's vector (Word C) with the two previously added words in the content tree will be measured, as shown in Figure 3.1 (a), and added as the child of the word with maximum correlation, which might be the root, as shown in Figure 3.1 (b), or the second word, as shown in Figure 3.1 (c). This process continues recursively for all new words in the content in order to find their location in the content tree.

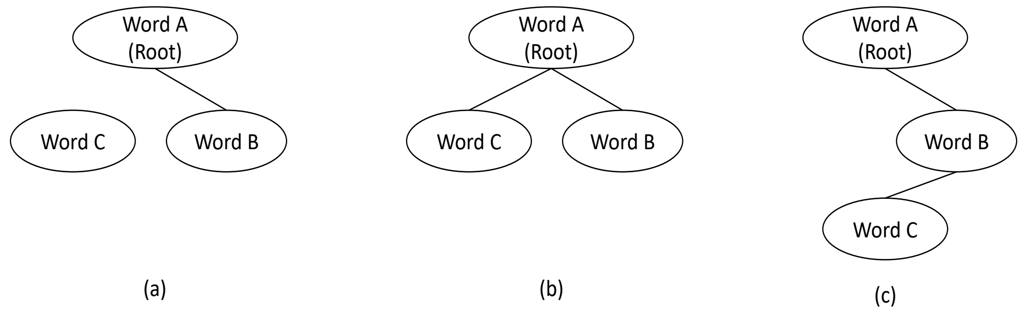

Figure 3.1. Content Tree Induction Early Process

Algorithm 3.1 shows the process of the content tree creation while Figure 3.2 shows an example of a generated Document Content Tree for a sample movie review.

```
Algorithm 3.1: Generating Document Content Tree

Input: Dataset
Output: Content Tree, Content Tree-based updated Word Vectors.

Split Training and Testing Data.
For each word in the Training Data:
    If it exists in the model:
        For each word in the Content Tree:
            Calculate the correlation between the word vectors.
        Parent ← Word with Maximum Correlation
        Add the word to the Content Tree as the child of the parent.
Return Content Tree-based updated Word Vectors.
```

For the computational cost, the worst-case scenario is when all the words in the content text are unique. In this situation, Algorithm 3.1 follows $O(n^2)$ complexity to estimate the correlation between the word vectors of each word in the content text with all the words in the updated constructed content tree, where n is the length of the content text.

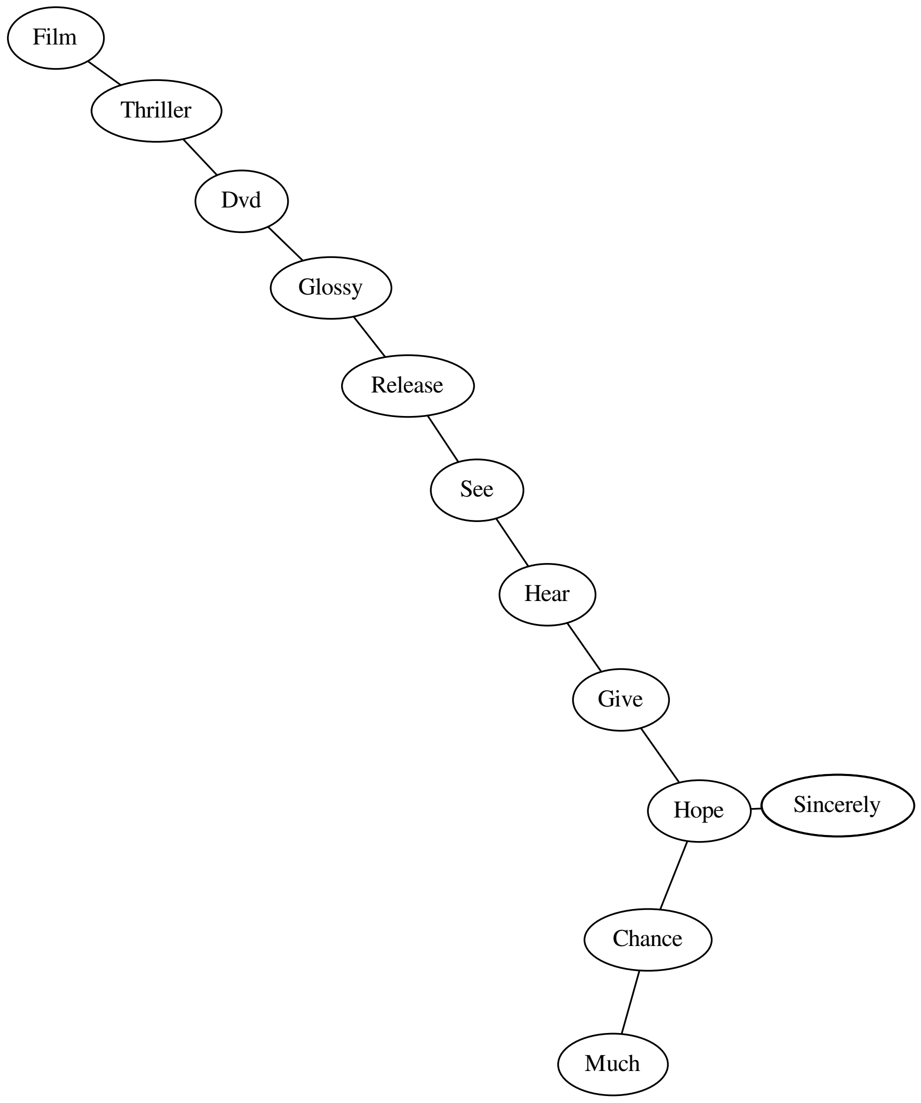

Figure 3.2. Content Tree Sample generated from an IMDB Movie Review with the Word2Vec Model

3.2.2 Content Tree Word Embedding Document Representation

In order to define the baseline, the average for the vector of words is calculated from the two deep learning-based models. The first model was trained by the Google News corpus, which generated a 300-dimensional vector for each word. The second model was trained by a corpus from open repository web crawl data, which also generated a 300-dimensional vector for each word. Furthermore, the Word2Vec model was trained and used for the classification task on the introduced dataset as another deep learning-based document representation method. The main idea of the presented approach is to create a content tree for training data. To create the content tree and find the parent-child relationship between words, the maximum correlation between the new word and all the other words in the content tree is calculated and used as the criteria. The new word will be the child of the word between the existing words (visited words) with the maximum correlation to the word's vector. For example, if the union of words: ("story", "apartment", "book") is assumed as the visited nodes in the content tree, the new correlation between the vector of the new word "chapter" should be calculated with all the words. The word with the maximum correlation will be the parent of the new word, which in this example is "book". The key idea of the approach is to update the word vector of the new word by calculating the weighted average of the word's vector and its parent's word vector. The new vector will be used to calculate the average of the word vectors in reviews for training and in testing phases of the study. This approach, illustrated in Equation 3.2, will consider the influence of the parent's word vector on the new word vector:

$$\boldsymbol{Vec_{Update}} = \frac{(1-\delta).Vec_{Parent} + \delta.Vec_{Original}}{1-\delta+\delta=1} \qquad \textbf{Equation 3.2}$$

where Vec_{Update} is the new generated vector, Vec_{Parent} is the parent of the word in the context tree, $Vec_{Original}$ is the word vector learned from the global training data, and δ is the decay factor to reduce the effect of the original word vector and increase the power of the parent's word vector. The effect of the decay factor has been investigated in [20].The following formula (Equation 3.3) shows the general form of a word vector calculation based on its location in the content tree:

$$\boldsymbol{Vec_{n(New)}} = (\mathbf{1}-\boldsymbol{\delta}).\boldsymbol{Vec_{n-1}} + \boldsymbol{\delta}.\boldsymbol{Vec_{n}} = (\mathbf{1}-\boldsymbol{\delta}).[(\mathbf{1}-\boldsymbol{\delta}).\boldsymbol{Vec_{n-2}} + \boldsymbol{\delta}.\boldsymbol{Vec_{n-1}}] + \boldsymbol{\delta}.\boldsymbol{Vec_{n}} \qquad \textbf{Equation 3.3}$$

$$= (\mathbf{1}-\boldsymbol{\delta})^2.\boldsymbol{V_{N-2}} + (\mathbf{1}-\boldsymbol{\delta}).\boldsymbol{\delta}.\boldsymbol{V_{N-1}} + \boldsymbol{\delta}.\boldsymbol{V_{N}}$$

$$= (\mathbf{1}-\boldsymbol{\delta})^2.[(\mathbf{1}-\boldsymbol{\delta}).\boldsymbol{V_{N-3}} + \boldsymbol{\delta}.\boldsymbol{V_{N-2}}] + (\mathbf{1}-\boldsymbol{\delta}).\boldsymbol{\delta}.\boldsymbol{V_{N-1}} + \boldsymbol{\delta}.\boldsymbol{V_{N}}$$

$$= (\mathbf{1}-\boldsymbol{\delta})^3.\boldsymbol{V_{N-3}} + (\mathbf{1}-\boldsymbol{\delta})^2.\boldsymbol{\delta}.\boldsymbol{V_{N-2}} + (\mathbf{1}-\boldsymbol{\delta}).\boldsymbol{\delta}.\boldsymbol{V_{N-1}} + \boldsymbol{\delta}.\boldsymbol{V_{N}}$$

$$\vdots$$

$$= (\mathbf{1}-\boldsymbol{\delta})^{n}.\boldsymbol{Vec_{Root}} + \cdots + (\mathbf{1}-\boldsymbol{\delta})^{n-m}.\boldsymbol{\delta}.\boldsymbol{Vec_{n}} + \cdots + (\mathbf{1}-\boldsymbol{\delta}).\boldsymbol{\delta}.\boldsymbol{Vec_{n-1}} + \boldsymbol{\delta}.\boldsymbol{Vec_{n}}$$

where n is the depth of the word in the content tree, Vec_n is the original word embedding and $Vec_{n(New)}$ is its updated word vector.

As shown in Equation 3.4, for each review, $Vec_{Average}$ is the average vector of all words that is calculated and used as a feature vector for training a linear SVM classifier. The same feature vector is generated for each review in the test data.

$$\boldsymbol{Vec_{Average}} = \frac{\sum_{i=1}^{N} \boldsymbol{Vec_{Update}(i)}}{\boldsymbol{N}} \qquad \textbf{Equation 3.4}$$

where N is the number of words of each review and V_{Update} is the content tree-based generated vector. Algorithm 3.2 is used for the content tree-based word vector modification and to calculate the average of the modified words for each document in the datasets. The calculated average word vector is used as a feature vector in the task of classification.

```
Algorithm 3.2: Generating CTWE
Input: Dataset
Output: CTWE-based Average Word Vectors.

Split Training and Testing Data.
For each word in the Training Data:
        If it exists in the model:
                For each word in the Content Tree:
                        Calculate the correlation between the word vectors.
                Parent ← Word with Maximum Correlation
                $Vec_{Update} \leftarrow (1-\delta).Vec_{Parent} + \delta.Vec_{Original}$
                Add the word to the Content Tree as the child of the parent.
$\mathrm{Vec}_{Average} = \frac{\sum_{i=1}^{N} Vec_{Update}(i)}{N}$
Return CTWE-based Average Word Vectors
```

Since Algorithm 3.2 employs Algorithm 3.1 to create the content tree, this algorithm follows Algorithm 3.1 complexity order, which is $O(n^2)$.

In the case of scalability for the presented approach, because creating a content tree is an exponential algorithm, a solution to reduce the size of the training data is presented. A part of the training data is randomly selected as “seed data” and used to train the content tree. The size of the seed data is reduced to 10% of the original training data.

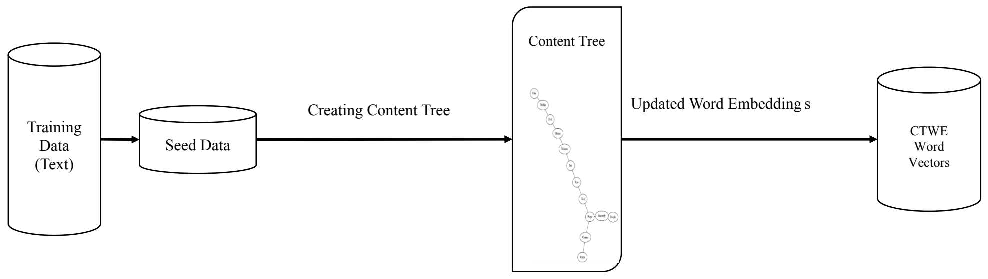

Figure 3.3 shows a block diagram of the updated word embedding process for a classification task.

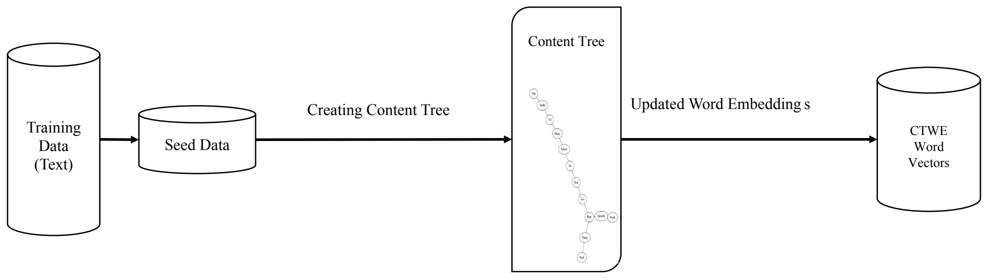

Figure 3.3. Block Diagram of the CTWE Approach

3.3 Composed Maximum Spanning Content Tree

3.3.1 Maximum Spanning Content Tree Document Representation

The maximum spanning tree (MST) is a tree with a minimum number of links and the highest possible weights that span all of one node. Kruskal's algorithm [73], which is used to calculate this tree, is applied in order to identify a graph's minimum spanning tree. It first sorts the edges in order of increasing cost and then adds edges to generate a fully connected graph by bridging the separate components. Negativizing the weights of each node allows the algorithm to compute the maximum weight spanning tree.

In the maximum spanning content tree (MSCT) approach, the same algorithm is used to generate the content tree. To select the root word, the fully connected graph of all the words in the training data should first be generated. In this weighted graph, the weight between every two nodes is the cosine similarity of their word vectors. The summation of weights for each node should then be calculated and the node with the highest weight summation is selected as the root node. The Kruskal algorithm will be applied over the graph and the MSCT will be generated for the words in the training data. In the subsequent step, the word vectors will be updated based on the parent-child relationship in the MSCT, similar to the CTWE approach. Algorithm 3.3 and Figure 3.4 illustrate the MSCT approach.

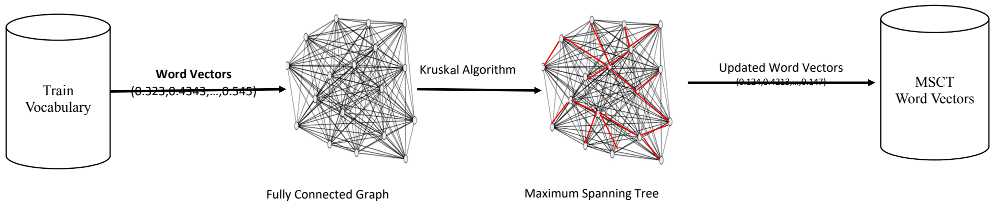

Figure 3.4. Block Diagram of the MSCT Approach

```
Algorithm 3.3: Generating MSCT

Input: Dataset
Output: MSCT-based Average Word Vectors.

Split Training and Testing Data.
For each word in the Training Data:
        If it exists in the model:
                Calculate the cosine similarity with all other word vectors as an Edge.
                Add the Edge to the fully connected graph: G.
For each node in G:
                $Weight_{Node} = \sum_{Edges} Weight_{Edge}$
Root ← Max ($Weight_{Node}$)
$G_{MSCT}$ ← Kruskal (G, Root)

For each node in $G_{MSCT}$:
        $Vec_{Update} \leftarrow (1-\delta).Vec_{Parent} + \delta.Vec_{Original}$

$\text{Vec}_{Average} = \frac{\sum_{i=1}^{N} Vec_{Update}(i)}{N}$
Return MSCT-based Average Word Vectors
```

Algorithm 3.3 follows $O(n^2)$ complexity in order to calculate the cosine similarity between all the nodes in the fully connected graph, where n is the size of the vocabulary in the training data. The Kruskal algorithm follows $O(e \log e)$, where e is the number of edges (Here e=N) in the graph.[74]

3.3.1.1 The memory challenge of the MSCT

The phase of generating the fully connected graph in the MSCT follows the $O(n^2)$ (n is the number of words). This present work implements an experiment that involves Word2Vec word embedding over the IMDB movie review dataset. During the implementation phase, for the total of 114,583 unique terms in the training data, at least 1TB memory (RAM) was needed. The only available resource at the time of the experiment was the Graham distributed system [75] in Compute Canada, which is the largest and by far the most powerful cluster among the current SHARCNET fleet of supercomputers. Due to the long processing time and queue to access the resources, another version of the MSCT, called the Composed Maximum Spanning Content Tree (CMSCT) document representation, is proposed.

3.3.2 Composed Maximum Spanning Content Tree Document Representation

In this approach, similar to the MSCT approach, a graph is generated but not the fully connected graph of the training data. The fully connected graph is generated for the first document in the training data and the MST is then extracted from it. A similar MST should also be extracted for the second document. These two graphs are then composed together. The composition in this context results in a union of the nodes and edges. For the remainder of the documents in the training data, the MST will be generated and composed to the current content tree. At the end of this process, the Kruskal algorithm will again be applied in order to remove the cycles and generate a new tree. This sequential algorithm does not suffer from the need to have an inordinately large memory. In the next step, the word vectors will be updated based on the parent-child relationship in the CMSCT, similar to the CTWE and MSCT approaches. **Error! Reference source not found.** and Algorithm 3.4 show the process of the CMSCT approach.

The evaluation result of the CMSCT in comparison to the MSCT approach is shown in Table 5.13. In terms of complexity, Algorithm 3.4 follows $O(m^2)$ to create a fully connected graph where m is the vocabulary size of each document, (m<<N) and follows the $O(e \log e)$ to create the maximum spanning tree for each document and again for the composed graph of all documents in the training data.

Algorithm 3.4: Generating CMSCT

```
Input: Dataset
Output: CMSCT-based Average Word Vectors.

Split Training and Testing Data.
For each document in the Training Data:
    For each word in the Training Data:
        If it exists in the model:
            Calculate the cosine similarity with all other word
            vectors as an Edge.
            Add the Edge to the Document's fully connected
            graph:$G_{Document}$.
    For each node in $G_{Document}$:
                    $Weight_{Node} = \sum_{Edges} Weight_{Edge}$
    $Root_{Document} \leftarrow$ Max ($Weight_{Node}$)
    $MSCT_{Document} \leftarrow$ Kruskal ($G_{Document}$, $Root_{Document}$)

For each document in the Training Data:
    $G_{MSCT}$=Composed ($G_{MSCT}$, $MSCT_{Document}$)

For each node in $G_{MSCT}$:
                    $Weight_{Node} = \sum_{Edges} Weight_{Edge}$
$Root_{MSCT} \leftarrow$ Max ($Weight_{Node}$)
$G_{CMSCT} \leftarrow$ Kruskal ($G_{MSCT}$, Root)

For each word in CMSCT:
    $Vec_{Update} \leftarrow (1-\delta).Vec_{Parent} + \delta.Vec_{Original}$

$\text{Vec}_{Average} = \frac{\sum_{i=1}^{N} Vec_{Update}(i)}{N}$
Return CMSCT-based Average Word Vectors
```

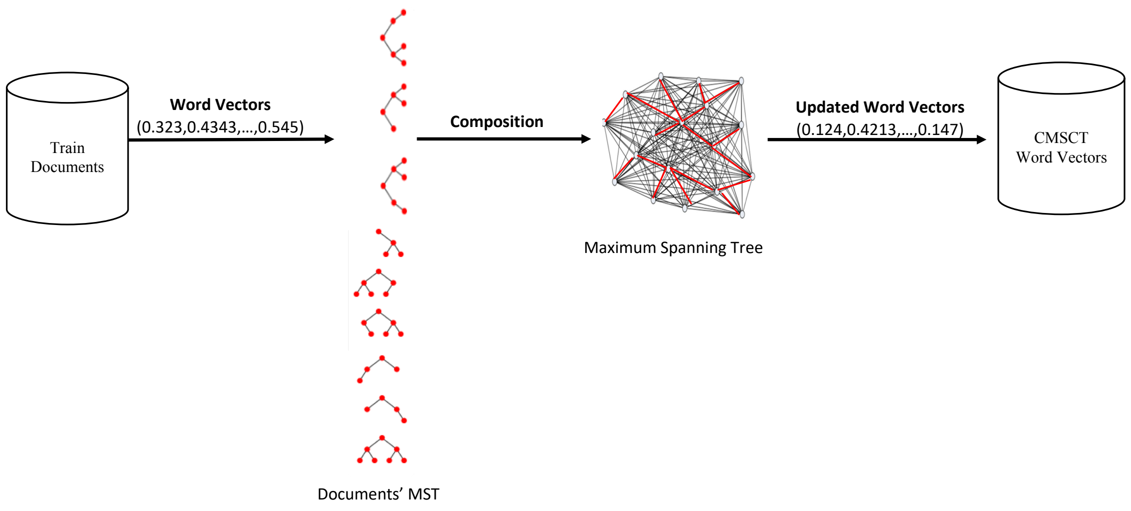

Figure 3.5. Block Diagram of the CMSCT Approach

3.4 Embedding-based Word Clustering Document Representation

The presented approach, Embedding-based Word Clustering (EbWC), consists of two phases: word clustering and word vector generation.

In the task of text clustering, the target class is not being predicted, but trying to group similar words based on different similarity measures. Groups should not ultimately be similar to each other.

In the presented approach, all the words contained in the training data are first arranged into a specified number of clusters based on their word vector from the Word2Vec, GloVe, fastText, LSA, or Random word embedding approaches. The clustering algorithm uses K-means or hierarchical clustering by which, for this study, both clustering methods build the same clusters. In the second step, the bag-of-clusters method is used for mapping each document to a vector. The distance of the word's vector to the centroid of each cluster is an element of the new representation vector. In this study, in order to be able to compare this approach with other presented word embeddings, the number of clusters in the K-means algorithm, which is the number of dimensions in the representation vector, is set to 300. Figure 3.6 illustrates the process of cluster vector creation in the EbWC approach.

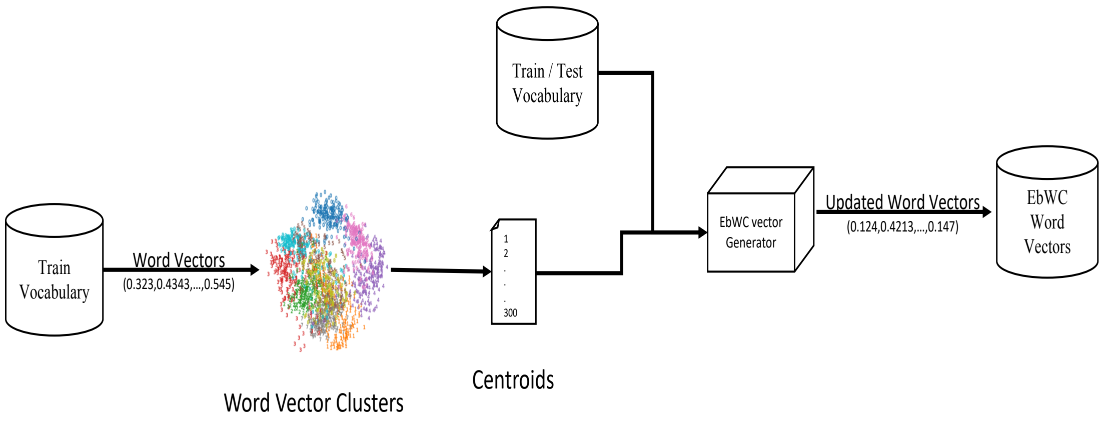

Figure 3.6. Block Diagram of the EbWC Approach

In this approach, each word ($\boldsymbol{w^i}$) is presented with a vector of distances to the clusters' centroids, as shown in Equation 3.5:

$$\boldsymbol{Vec_w} = (\boldsymbol{d^1}, \boldsymbol{d^2}, \dots, \boldsymbol{d^m}), \textbf{Clusters} = (\boldsymbol{C^1}, \boldsymbol{C^2}, \dots, \boldsymbol{C^n}) \qquad \textbf{Equation 3.5}$$

$$\boldsymbol{Vec^i_{EbWC}} = (\boldsymbol{v^1}, \boldsymbol{v^2}, \dots, \boldsymbol{v^n})$$

$$\boldsymbol{v^j} = \textbf{Cosine}(\boldsymbol{W^i}, \boldsymbol{Centroid}(\boldsymbol{C^j}))$$

where W is a word and $\boldsymbol{d^1}$ to $\boldsymbol{d^m}$ are elements in its word vector, Vec_w. $\boldsymbol{C}$ represents a cluster's centroid, while i is the number of words, and n is the number of final word vector dimensions (the number of clusters extracted from the training data). The $\boldsymbol{v^j}$ is the cosine distance of the word $\boldsymbol{W^i}$ and centroid of the $\boldsymbol{C^j}$. Algorithm 3.5 shows the process of creating EbWC representation vectors.

Algorithm 3.5 follows $O(n^2)$, according to the k-means algorithm [76] used for the clustering phase while creating new word vectors by calculating the distance from each cluster's centroid follows $O(n)$.

Algorithm 3.5: Generating EbWC

```
Input: Dataset
Output: EbWC-based Average Word Vectors.

Split Training and Testing Data.
Cluster the words in the Training Data based on their word vectors.
For each word in the Training Data:
        If it exists in the model:
                For each cluster in clusters:
                $v^j$ = cosine_distance($Vec^i, Centroid(C^j)$)
        $Vec_{Ebwc} \leftarrow (v^1, v^2, \dots, v^n)$
Vec$_{Document} = \frac{\sum_{i=1}^{N} Vec_{EbWC}^{i}}{N}$ (N is the number of words in the Document)
Return EbWC-based Average Word Vectors
```

3.5 Autoencoder-based Word Embedding

3.5.1 Autoencoder

An autoencoder is a type of neural network that tries to reconstruct inputs to outputs. This architecture uses non-linear encoder and decoder modules to provide a latent representation based on training data, which is known as "encoding the input". Deep multi-layer neural networks are used in the architecture of autoencoders.

Several studies have been conducted into model word counts by different types of autoencoders, namely: Softmax decoder [77], Poisson decoder [78], and binary stochastic hidden units.

Similar to the PCA and other dimensionality reduction methods, autoencoders compress the input into a latent-space representation and reconstruct the output from this representation. For this purpose, autoencoders use the transformation. Unlike the PCA which uses linear transformation, autoencoders use non-linear transformation.

Autoencoder architecture consists of two parts: the encoder and decoder. The encoder compresses the data in the input to a latent-space representation (h=f(x)) while the decoder reconstructs the original input from that latent-space representation (r=g(h)).

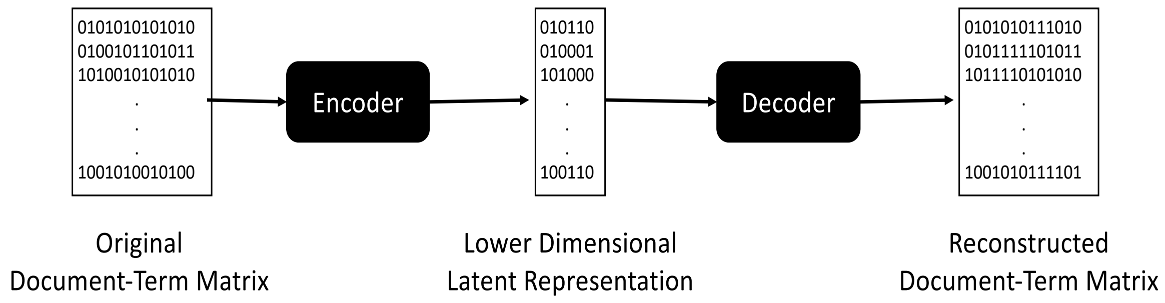

Figure 3.7. Autoencoder Block Diagram

Figure 3.7 demonstrates a two-layer vanilla autoencoder with one hidden layer. The popular types of autoencoders are: convolutional autoencoder; variational autoencoder; denoising autoencoder; and sparse autoencoder. For this study, an autoencoder that uses a single fully-connected neural layer as the encoder and decoder is implemented. An embedding layer with a size of 300 is used to generate new word vectors the size of 300.

3.5.2 Autoencoder-based Word Embedding Document Representation

The main objective of Autoencoder-based Word Embedding (AbWE) is to train the autoencoder neural network with the training data and then update (encode) the word vectors with this autoencoder. The new word representation should carry the hidden information from the training data context. Figure 3.8 illustrates the generation process of the autoencoder-based word vector representation and Algorithm 3.6 describes the intention:

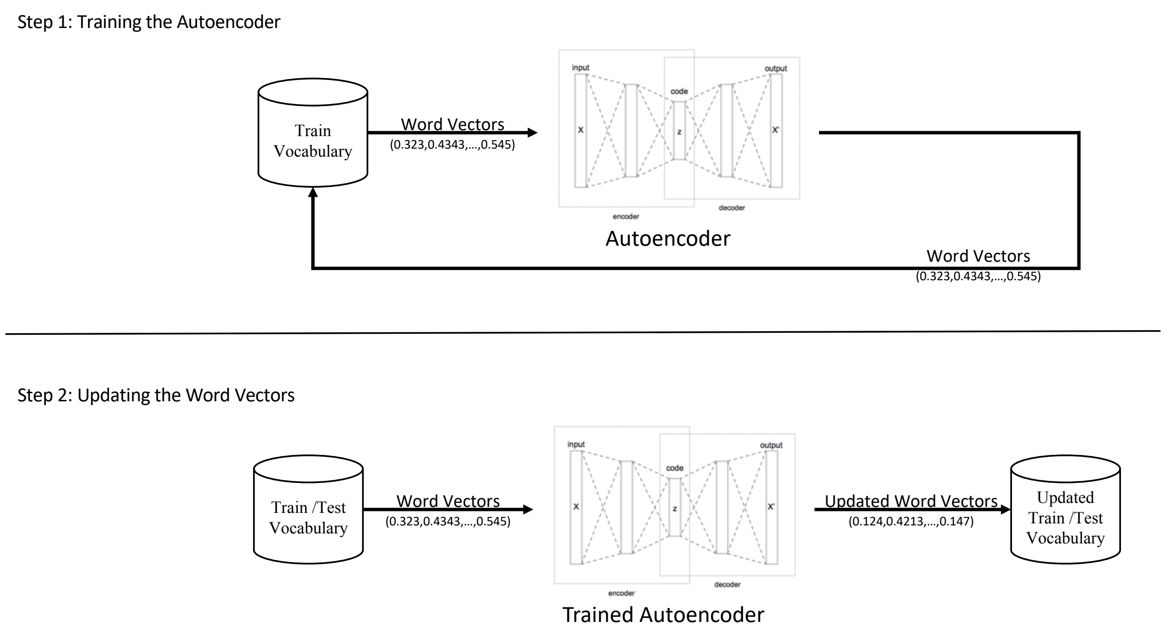
Step 1: Training the Autoencoder
Train Vocabulary
Word Vectors (0.323,0.4343,...,0.545)
input
code
output
x
z
x'
encoder
decoder
Autoencoder
Word Vectors (0.323,0.4343,...,0.545)
Step 2: Updating the Word Vectors
Train /Test Vocabulary
Word Vectors (0.323,0.4343,...,0.545)
Trained Autoencoder
Updated Word Vectors (0.124,0.4213,...,0.147)
Updated Train /Test Vocabulary

Figure 3.8. Block Diagram of the AbWE

The complexity of Algorithm 3.6 depends on the training phase that trains the autoencoder by the word vectors of the training dataset. This algorithm follows $O(n)$, as each word vector passes through the network once during each training iteration.

```
Algorithm 3.6: Generating AbWE
Input: Dataset
Output: AbWE-based Average Word Vectors.

Split Training and Testing Data.
For each word in the Training Data:
        If it exists in the model:
            $Autoencoder_{model} = train\_autoencoder(Input: W^i, Output: W^i)$
For each word in the Training and Testing Data:
        $Vec^i_{AbWE} = Autoencoder_{model}(W^i)$
$\mathrm{Vec}_{Document} = \frac{\sum_{i=1}^{N} Vec^i_{AbWE}}{N}$ (N is the number of words in the Document)
Return AbWE-based Average Word Vectors
```

3.6 Comparison with Similar Studies

The current study can be differentiated from other similar investigations due to the subject and method.

In comparison to the original versions of the Word2Vec, GloVe, and fastText, which generate word vectors in an unsupervised manner and independent from the local context, as well as to the LSA which is completely based on the local context, the presented approaches use local context information as well as global information.

Maas et al. [24] introduced a word representation to capture the semantic and sentiment meaning of words. Their approach used supervised and unsupervised learning that fit the representation into the task, not the context.

A similar study was conducted by Hong [35], who tried to improve the performance of Paragraph2Vec by adding a hidden layer and a tensor layer. This study was also different from the present work in that it fit into the task, similar to the Paragraph2Vec approach.

Kim et al. [28] introduced three approaches for document representation, based on the Word2Vec vector of content words. These approaches did not change the word level vector but calculated the document vector in a novel approach.

Bernotas et al. [12] used a tagging-based document representation method by using ontology, which was conducted at the document representation level and for a clustering task.

The study of Lu et al. [29] was similar to the EbWC approach. Here, they created a cluster's universe and, by segmenting documents into topics and assigning topics to the clusters, a relationship was made between each document and the clusters. Their approach did not change the word representation based on the context.

The study of Socher et al. [30] employed the Neural Tensor Network to extract the relationship between entities in a knowledge base. The vector representation of words was used to calculate the average of the word vectors in an entity, which made this study different from the presented approaches in the current study. Tai et al. [37] introduced a new representation for sentences that is a combination of LSTM and tree-structured network topologies. The authors first generated an alternative representation based on concepts in a document and then aggregated the results with the original word vectors. This model is also used for document representation.

In comparison with the presented approaches, the USE [38] is a transfer learning-based approach that encodes text into embedding vectors. It computes context aware embedding for each word in the text and calculates an element-wise sum at each word position to produce an embedding vector for the whole document. This representation should be calculated for each sentence and is not a one-time process.

The document representation by [41] uses weighted word occurrence based on its contribution in the context and captures sub-topic level keywords to facilitate the learning process. The computation and memory cost depend on the size of the text and is another variation of the Doc2Vec approach.

The approaches by [42] and [43] employed autoencoders similar to the AbWE approach. However, as well as not presenting a new word vector for each word in the context, these approaches worked at the document embedding level.

In comparison, ELMo [50] and BERT [51], presented LSTM based architectures that achieved state-of-the-art results in some NLP tasks, and could represent new word embeddings. Their main focus in the model is more on the architecture than the context aware word representation.

3.7 Summary

The first approach, CTWE, employs a semi-taxonomy structure, known as a content tree, and subsequently updates the word embedding vectors. The second approach, MSCT, is proposed in order to first select the root based on the node degree, then generate the maximum spanning tree. Another version of this approach, called the CMSCT, which does not require a high amount of memory to generate the fully connected graph of all the words in the training vocabulary, is defined. This approach first generates a small spanning tree for each document, then generates the training data spanning tree by combining them. In the final step, the word vectors are updated based on their location in the maximum spanning tree. The third approach, EbWC, uses the clustering method for extracting the conceptual structure of the context. Each element of the new word embedding is the distance from the centroid of each word group cluster. The fourth method, AbWE, uses an autoencoder for dimensionality and noise reduction in the context. The main idea here is to train the autoencoder to capture the training data concepts, then update the word vectors by encoding them.

Chapter 4. Experimental Evaluation Setup

4.1 Introduction

To study the effectiveness of the methods presented in Chapter 3, the design and conduct of experiments on datasets and classifiers with different characteristics is needed.

In Chapter 4, the experimental setup of the research is described. The five selected datasets are first introduced and the rationale behind this selection is described. The 11 classifiers used in this study are described and the evaluation metrics are then initiated. The experimental setup is described as Algorithm 4.1.

4.2 Datasets

The following five datasets, which are some of the most well-known in the domain of text mining and text classification, have been used as the benchmark in this study. For covering the aspect of classification type, two binary datasets and three categorical datasets are selected. Different observation sizes, unique words count, and class distributions were collected as datasets to evaluate the proposed approaches in different situations. Table 4.1 shows the statistics pertaining to the datasets.

Table 4.1. Dataset Statistics

Dataset	*# of classes*	*Training Observations*	*Testing Observations*	*# Unique Words in Training*	*# Unique Words in Testing*
IMDB	2	25,000	25,000	114,583	99,805
HSI	2	7,254	7,255	20,553	19,126
20 Newsgroups	20	11,347	7,550	261,846	189,503
Reuters-21578	74	7,769	3,019	49,240	29,397
AG News	4	120,000	7,600	182,591	38,767

4.2.1 IMDB Movie Review

The Internet Movie Database (IMDB) Movie Review [79] is a sentiment (binary) classification dataset, consisting of 25,000 training and 25,000 testing records. The information is collected from movie reviews from the website www.imdb.com. The state-of-the-art result that is found for this dataset represents an ensemble approach based on naïve base SVM and recurrent neural networks (RNNs), presented by Mesnil et al. [80], which is 92.57 in terms of accuracy.

4.2.2 Hate Speech Identification

The hate speech identification (HSI) dataset [81] contains 14,509 tweets with three classes. In their study, Almeida et al. [82] achieved a result of 96% F_score with a k-NN classifier for this dataset.

For this present study, the dataset is modified and converted to a binary classification task. The positive class is when a tweet contains hate or offensive speech and the negative class is when the tweet was not offensive.

4.2.3 20 Newsgroups

The 20 Newsgroups (20 NG) collection [83] is one of the popular datasets in text clustering and text classification. The dataset includes 18,897 posts on 20 topics, split into training and testing. The training and testing data are divided according to a specific date. Each newsgroup corresponds to a specific topic.

Lai et al. [84] employed a recurrent convolutional neural network architecture and reported state-of-the-art results for this dataset as 96.49 in terms of the F_score macro.

4.2.4 Reuters

The Reuters-21578 benchmark corpus [85] includes 10,788 news documents from the newswire service of the Reuters financial newswire service. The training data is a collection of 7,769 documents while the testing data contains 3,019 documents.

Nam et al. in [86] achieved the highest reported F_score for this dataset with 87.89%. A cross-entropy algorithm with TF-IDF document representation was used as the solution.

4.2.5 AG News

The AG news topic classification dataset [87] contains four classes with 30,000 training samples and 1,900 testing samples for each class, a total of 120,000 training and 7,600 testing records.

This dataset is collected by the ComeToMyHead academic search engine from more than 2,000 news sources during a period of almost one year. For this dataset, Conneau et al. [88], who designed a deep learning-based architecture called the Very Deep Convolutional Network, reported a state-of-the-art identified result of 7.64 in terms of error rate.

4.3 Classification

The machine learning approaches in the domain of text mining can be differentiated as supervised, unsupervised and semi-supervised learning algorithms [89, 90].

In supervised learning, the model is trained by a set of labeled input features. The set of labels are typically fixed. In the task of regression, the labels can be a continuous value vector. In supervised learning, the aim is to find the best parameters for prediction based on a loss function.

In unsupervised learning such as PCA and clustering methods, the data is not labeled, and the objective is to find hidden patterns.

Semi-supervised learning refers to methods that fall between supervised and unsupervised learning. This approach typically uses a combination of a small amount of labeled data and a large amount of unlabeled data. These methods are not as expensive as supervised learning methods that need a labeling process. Moreover, the results are usually more accurate than the unsupervised methods.

In this research, the supervised learning method is used by employing 11 classifiers, which are introduced in Sections 4.3.1 to 4.3.10.

4.3.1 Logistic Regression

Logistic regression (LR) is a classification technique in machine learning that originates from the field of statistics. It is a common method for binary classification problems, as a result of its low computational cost. The goal of LR is similar to linear regression, namely to find the weight of each input (coefficient). The difference is in the transformation function, called the logistic function [91]. This logistic function transforms the values to a range between 0 and 1, from which can be predicted the class based on the rules or probabilities. LR works better by removing the correlated attributes or those that are not related to the output.

4.3.2 Support Vector Machine

One of the most famous algorithms in machine learning is SVM, which is based on the functionality of the hyperplane, which is a line that is supposed to split the variable space based on each class of the input. The SVM calculates the coefficients for the hyperplane to most effectively split the classes.

The margin is the distance between the hyperplane and the closest bordering point of the data. An optimized hyperplane can distinguish the classes with the maximum size of margin. The points that are used to define (support) the hyperplane are classed as the support vectors. Discriminant hyperplanes are used for generalization capabilities during classification [92, 93].

Linear SVMs use linear decision boundaries. Non-linear classifiers, which are more complex, use the "Kernel Trick" by indirectly mapping the data to another higher dimensional space. The generalization power of SVMs prevents overfitting into the data, [94] and also strengthens them to face the Curse of Dimensionality [92, 93]

4.3.3 Naïve Bayes

The Naïve Bayes (NB) classifier is a powerful and simple solution for predictive modeling. The pre-assumption is the independency of each input variable. As this assumption is not realistic, the classifier is referred to as naïve. However, it is an effective solution for complex problems. The probability for each class and the conditional probability given to each value are calculated to create a probability model. For new data, the model uses the Bayes theorem for prediction.

This present study employs the Gaussian Naïve Bayes (GNB) classifier, which is a type of NB that can be used when its distribution of data follows the Gaussian (normal) distribution. For real-valued data, it is common to assume a Gaussian distribution (bell curve), so that the GNB can accurately estimate these probabilities[95].

4.3.4 Decision Tree

Tree representation is used by a decision tree (DT) to generate a predictive model by learning the decision rules. The middle nodes of the tree represent the conjunctions of the features while the leaves are the class labels.

One of the specific features of DT is the understanding level of the DT algorithm, which is easy for humans to understand, compared with other classification algorithms. In text classification, the document will start from the root and go through the query structure to reach a certain leaf that shows a certain class [96]. The classification logic can be explained to humans by simple mathematical algorithms. This is useful when we know the presence of the relevant features is a nature of the problem [14]. For tasks with a limited number of features, DT can be a good candidate due to its understandability, simplicity, and performance. A drawback of DT is the tendency to overfit the training data [97].

4.3.5 Random Forest

The Random Forest (RF) approach uses multiple DT structures that are generated by random sampling with replacement. This supervised learning classifier works according to the bagging or bootstrap ensemble machine learning algorithms. DTs can be classification or regression trees; RF can handle both classification and regression problems.

Bootstraps are used to estimate a quantity from a data sample, similar to calculating the mean of multiple samples and then calculating the average of all the mean values to determine a fine estimation of the true mean value.

The bagging approach is similar. For each sample of the training data, a model is constructed, and each model predicts for a new observation. All the predictions are averaged in order to estimate the final output [98].

Based on the sample training data, each model is accurate but different. A combination of predictions can lead to a prediction with higher quality in comparison with each individual model.

4.3.6 k-Nearest Neighbor

The k-Nearest Neighbor (k-NN) algorithm [99] is an instant-based learning algorithm that is widely used in text classification tasks. In the case of text classification, the k-NN can classify each object based on the closest document in the multi-dimensional feature space according to the training dataset. Based on the training data, the feature space is divided into partitions and each observation is assigned to one partition (class) based on the k nearest neighbor. For the regression problem, this algorithm uses the mean output variable.

While this method needs to search the entire feature space for the most similar instances, it uses all the features for computing the distance. Hence, it is computationally intensive and requires significantly large memory according to the size of the training data. Also, noise or unrelated data can reduce the accuracy of the k-NN.

The training phase consists of storing the features and categorizing them in accordance with the training data. In the classification phase, the algorithm needs to calculate the distance from the new observation's vector to all the instances in the training data in order to predict the class (the call that contains most of the neighbors). Finding similar neighbors is critical for the k-NN, which needs a proper similarity (distance) measure. The type of calculated distance impacts the k-NN efficiency. The Euclidean distance is the typical distance measure that is used with the k-NN [99]

4.3.7 Deep Neural Networks

Linear classifiers are limited when a few samples are available in a large output space because they cannot share parameters between features and classes. Factorizing to low rank metrics [8, 100] or using multilayer neural networks [101] are common solutions.

The majority of deep learning methods are formed in accordance with the neural network learning algorithms. Noteworthy research studies have been conducted on machine learning through deep neural networks. Imitating the brain's process is the significant feature of neural networks, which is applicable for image, text, and other signal processing problems. The data goes through layers, from an input layer to the final layer, in order to produce the output. Hidden layers are known as the layers between the input layer and the output layer. The network with several hidden layers is known as the deep neural network (consequently the term “deep learning”). The activations are distributed over classes in the final layer.

The parameters are calculated by the maximum likelihood method with SGD based on the dataset. SGD uses a small subset of data for each gradient update. Binary cross-entropy loss is used for binary classification and categorical cross-entropy for multi-class tasks.

The convolutional neural networks (CNNs) and the RNNs are the most popular architectures.

4.3.8 Convolutional Neural Networks

The convolution function is defined as a sliding window that is applied to a matrix. In the field of image processing, where the matrix represents the image and each element corresponds to one pixel, this sliding window is named as the filter, kernel, or feature detector. The filter multiplies over the matrix elements and calculates the total. This process is conducted for the entire matrix. In this case, finding the difference between a pixel and its neighbors can detect the edges.

CNNS are composed of several convolution layers with non-linear activations applied to the output, namely tanh or rectified linear unit (ReLU). The convolution layer is used to compute the output. Each region of the input is connected to an output neuron. Different filters are applied over each layer and the results are combined. The filters' coefficients are calculated during the training phase according to the training dataset. The network captures different aspects of the features on each layer. As an example in the field of image processing, the architecture learns the edges from the raw data in the first CNN layer, and use the edges to detect shapes in another layer. The pattern is repeated to the final layer, which can use the high-level captured features [102].

The vanilla architecture of a CNN for supervised prediction is made by a convolution layer, pooling layer, and an optional fully connected layer. More than 10 convolutional and pooling layers are regularly needed in practice to achieve acceptable results. CNNs, which need a considerably extensive amount of labeled data for training, achieve success mostly in the field of image processing (computer vision)[103].

CNN architecture is a feed-forward neural network. The generated features are convolved together in each layer since classification is applied. In image processing, the features are extracted from small 2d regions. For text documents, a one-dimensional region is used [104]. Each sentence can be presented by a sequence of the k-dimensional vectors of its words. The convolution of the words, selected by a fixed-sized window with a filter, can produce feature mapping. Feature mapping is provided by applying the filter to all the documents in the training dataset. The max-over-pooling technique is applied to the extracted features to obtain the maximum value for the filter. To obtain the output class, the softmax layer is applied as the last step in the CNNs. The input matrix is convolved with a set of kernels and the biases are added to generate a new feature map. A non-linear transformer is then applied, and this process is repeated for each convolutional layer.

In the majority of NLP tasks, the inputs are sentences of documents or their represented version with a matrix. The column (or sometimes the row) corresponds to a token, which can be a character, a word or a sequence of characters or words. Each row (or column) is a vector to represent the token. These vectors can be pre-trained word embeddings such as Word2Vec, GloVe, fastText, or even the one-hot vector to index the word into a vocabulary. For a 15-word sentence that uses a 300-dimensional word embedding, a 15 by 300 (15x300) matrix is used as the input. This matrix can be comparable with an image. In NLP, the filters regularly slide over full rows (words), unlike images which use local regions. For this reason, the width of the filters should be the same as the width of the matrix, but the height of the filter can vary. Typically, the size of a window is two to five words. A definition of the higher level is not as clear as the field of computer vision but can be interpreted as the concepts or meanings. Convolutional filters can effectively learn representation of the token (usually the words), with no need to learn the entire vocabulary. This present study uses a total 128 filters with size 5 and max pooling of 5 and 35 as the vanilla simple CNN, known as One Dimensional CNN (1D-CNN).

4.3.9 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) were developed for discrete sequence analysis. The model learns the class distribution for a sequence of inputs, rather than a single input [105]. At any one time, the output of non-linear mapping is a hidden state (latent) that is maintained by an RNN. This can be useful in applications, such as machine translation where the input is the sentence of the source, and target languages as well as a previous state. Weight matrices are shared over time. In the case of classification, the fully connected and softmax layers are needed to map the sequence to a class. For the training phase, RNNs suffer from the problems that should be dealt with by other deep learning approaches, as the gradient needs to be backpropagated through time [106]. The LSTM [107] is one of the developed popular specialized memory unit architectures of the RNN. The Gated Recurrent Unit (GRU) [108] is another architecture that is commonly used.

The RNNs have achieved success in processing streams of data [109]. One network is applied over all elements in a sequence while each output depends on the previous. To solve the gradient problem, the original RNNs only considered limited previous steps. The LSTM and GRU architectures address this problem by modeling hidden states with cells to determine what to consider, based on input value, as well as previous and current states. This can capture long-term dependencies that are essential for NLP tasks [110, 111].

4.3.10 Hybrid Approaches

4.3.10.1 Deep CNN-LSTM

A sequential combination of CNN and LSTM layers, followed by a dense layer, can define a Deep CNN-LSTM architecture. CNN models extract the features while LSTM models can interpret the features across time steps. The fully-connected layer generates the final output from a concatenation of the CNN and LSTM branches. The network uses 16 and

128 batches for training [112]. The first layer of the network, the embedding layer, converts the sequence of word indices and embeds each word into a fixed-size vector. This layer is a matrix of weights that are trained according to the training data. The multiplication of this matrix and the word index generates the word vector. Each branch, which accepts the output of the embedding layer as the input, has a one-dimensional convolution layer. Applying multiple filters with different sizes generates multiple outputs. These outputs understand a word when it appears with other words. A ReLU activation is applied to introduce the non-linear output of the CNN layer by replacing a negative result with zero. After the ReLU is activated, a one-dimensional max pooling is applied to convert the kernel size input to a single output, which is the maximum perceived number. Using max pooling will reduce overfitting with down sampling. To prevent overfitting, the dropout layer is applied by randomly replacing a part of the input to zero. This will help the generalization of the network. The next layer is the batch normalization, which normalizes the distribution of the result in each branch. This will facilitate convergence by reducing the internal covariate shift. The LSTM, as the last layer in each branch, is used according to the nature of the sequential data and will allow the previous input to impact the new input. The outputs of the branches are concatenated and generate an array. A fully-connected array will convert this array to a final output. The sigmoid activation function will adopt the output to a range between 0 and 1. A binary cross entropy loss function and an optimizer will compile the network.

4.3.10.2 AdvCNN

AdvCNN [113] is a parallel CNN based deep network with varying filter widths, able to achieve a state-of-the-art performance on sentiment analysis and question classification.

The basic idea is to use a CNN where different convolutions are used to produce different n-gram-like filters to determine the sentiment of a given text. For each convolution, 128 filters are used. The words are embedded in the first layer and convolutions with different filter sizes are applied over the vectors to generate feature maps of different lengths. A max pooling layer is applied over each feature map to generate a univariate feature vector. The concatenation of the feature vectors will pass to a softmax layer for classification. The original architecture is designed for the task of binary classification.

4.3.10.3 Boosted CNN

In a study by Gultepe et al. [104], a modified version of the AdvCNN architecture with n-gram filter sizes = (1,2,3,4) is presented. Each n-gram filter size pertains to a separate and parallel convolutional layer in the model. The core of the Boosted CNN is a 1D-CNN model [110]. These layers are finally concatenated with each other in a max-over-time pooling layer, prior to the softmax classification layer. The following are the parameters used for the network of the presented architecture:

For each n-gram, the number of filters is set to 128, which is a popular size in convolutional network models [114]. The size of the embedding dimension is set to 300 according to the size of the used word embeddings, such as the Word2Vector [8]. An L_2 parameter weight decay of $10e^{-5}$ [115, 116] is used in the convolutional layers to apply regularization over the weights. The ReLU activation function was used, according to its acceptable performance in other studies [117].

4.4 Evaluation Metrics

Measuring text classification performance is an important issue in the field of text mining. Experimental methods are more common than analytical evaluation methods.

In a 2018 survey study by Schnabel et al. [118], the evaluation methods for unsupervised word embeddings were divided into two major categories: intrinsic and extrinsic evaluations. In an intrinsic evaluation, the syntactic or semantic relationship between words is directly evaluated while, in the extrinsic method, word embedding is used as the input feature for another specific task.

Similar to evaluation methods in other fields of data mining, in text mining, the terms true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are used to compare the classifier's predicted results with the expected results.

Here, precision and recall are defined in [119] as follows:

$$Precision = \frac{TP}{TP+FP}$$ **Equation 4.1**

$$Recall = \frac{TP}{TP+FN}$$ **Equation 4.2**

Accuracy is then defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$ **Equation 4.3**

The F_score measure combines the precision and recall by calculating the harmonic mean. The balanced F-score formula [120, 121] is shown in Equation 4.4:

$$F_{\_score} = 2.\frac{Precision.Recall}{Precision + Recall}$$ **Equation 4.4**

The F_score macro individually calculates the metrics for each label and then calculates the unweighted mean. This does not take label imbalance into account. The F_score micro is calculated based on the total of true positives, false negatives and false positives.

There is a statement that suggests calculating the F_score micro for a class imbalance problem but, in this research, the F_score macro is selected to evaluate each approach and compare the results. The F_score macro is preferred over micro as it gives equal importance to each class whereas, similar to the accuracy metric, the F_score micro gives equal importance to each sample and is influenced by the number of samples in a class. For this reason, the F_score macro is consistently used in this research to evaluate each approach and compare the results.

4.5 Experimental Setup

To study the effect of the proposed document representations, the following set of experiments are designed. For each one of the five datasets, the four proposed approaches are implemented and word vectors for the dataset's vocabulary are generated based on five baseline word embeddings: Word2Vec, GloVe, fastText, LSA, and Random word embedding. The new word vectors are employed by the 11 introduced classifiers for application to the pre-defined task of the dataset. For the traditional classifiers, the average of the new word vectors is used as the document representation and, for the deep learning-based approaches, the new generated word vectors are used as the embedding layer. Algorithm 4.1 shows the steps involved in the study.

This research involved conducting a total of 1,111 experiments in order to evaluate the effect of the four proposed approaches:

```
Algorithm 4.1: Experimental Setup

Input: Datasets (IMDB Movie Reviews, Hate Speech Identification, 20
Newsgroups, Reuters-21578, and AG News)
Output: Evaluation results in terms of the F1_score Macro

For each dataset:
    Split Training and Testing Data.
        For each baseline word embedding:
            For each proposed approach:
                Generate new word embedding (Thesis contribution)
                For each classifier:
                    If Traditional Classifier:
                        $V^{Average} = \frac{\sum_{i=1}^{N} V_{Update}(i)}{N}$
                    If deep learning-based Classifier:
                        Embedding Layer ← $V_{Update}$
                    Return F1_score for the test datasets
```

5 datasets x 5 baseline word embeddings x 4 proposed approaches x 11 classifiers = 1,100 experiments plus 11 experiments that evaluated the word embedding generated by applying MSCT over the Word2Vec word embedding.

4.6 Summary

In this chapter, the experimental setup to evaluate the four novel document representation approaches is explained. Statistics for the five datasets that are used in this study are described. The theory behind the traditional and deep learning-based classifiers is specified and the evaluation metrics that are used are explained.

Chapter 5. Experimental Results and Discussions

5.1 Introduction

The results of the experiments that are conducted in Chapter 4 are presented and described in this chapter. The effect of the baseline document representations on traditional and deep learning-based classifiers is first evaluated and described. The four novel approaches, namely CTWE, CMSCT, EbWC, and AbWE are then applied to five baseline word embeddings for each dataset and new word embeddings are generated. By using these newly generated word embeddings as the new word vectors, the performance of the classifiers is compared with the corresponding baseline word embedding. Also, the word embedding that is generated by the MSCT approach for the IMDB movie review on Word2Vec word embedding is compared with the word embedding that is generated by the CMSCT approach. For each classifier, the variance of the word embedding is studied and the results, together with a discussion, are reported in the following subsections. By the end of this chapter, the total effect of the proposed approaches on the studied word embeddings, and a comparison between all the represented approaches among the different document representations, are considered and visualized as two heatmap charts.

5.2 The Effect of Document Representation on Traditional Classification Approaches

Table 5.1 shows the evaluation results for different combinations of six traditional classifiers that use different document representation methods. The bag of words, the bi-gram, and the LSA are the representatives of traditional word embeddings, while Word2Vec, GloVe, and fastText are the deep learning-based representations.

For the task of sentiment analysis in the IMDB Movie Review dataset, the best result, which is 0.8860 in terms of F_score macro, is achieved by a combination of LSA document representation and the SVM classifier.

An evaluation of classifiers shows that the LR could achieve the best result when using five out of six document representations, although it should be made clear that the main purpose of this experiment is to evaluate the document representations, not the classifiers.

Using the bi-gram enabled the LR and GNB classifiers to achieve their best results, while using the LSA produced the best results for the SVM and DT. Glove and fastText did not achieve any best result with these classification approaches.

For the task of sentiment analysis in the HSI dataset, employing the bi-gram approach showed the best results in four out of six classifiers. The highest F_score belongs to the LR, which uses bi-gram for document representation. The LSA, Word2Vec, and GloVe failed to achieve an improvement in any classifier.

For the task of document classification in the 20 NG dataset, using the LSA approach for the document representation helped four out of six classifiers to gain their best results. In addition, the best results were achieved by the SVM classifier when using the LSA document representation. The bag-of-words helped the LR and GNB to achieve their best results, while all three deep learning-based word embeddings could not produce any best result.

For the task of document representation in the Reuters dataset, the bi-gram achieved the best results when half of the classifiers used it as the document representation method. In this dataset, by using the bag-of-words document representation, the SVM classifier achieved the best result.

For the AG News dataset, the bi-gram document representation was able to achieve the best result in five out of six datasets. The best result was achieved by LR while using bi-gram as the document representation method. The second highest F_score was achieved by k-NN when using the fastText document representation.

In conclusion, it can be inferred that the bi-gram and LSA document representation approaches can work more effectively with traditional classifiers. Taking the average of the word vectors failed to achieve the best results in the majority of experiments.

Table 5.1.The Effect of Document Representation on Traditional Classification Approaches

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*
IMDB	***bag-of-words***	0.8668	0.8405	0.5819	0.7238	**0.7820**	0.6070
	bi-gram	**0.8749**	0.8563	**0.7453**	0.7262	0.7816	0.5737
	Word2Vec	0.8577	0.8452	0.3416	0.6589	0.7187	**0.7495**
	GloVe	0.8536	0.8363	0.4268	0.6787	0.7326	0.7397
	fastText	0.8660	0.8443	0.3593	0.6725	0.7296	0.7460
	LSA	0.8743	**0.8860**	0.1456	**0.7837**	0.6897	0.6780
HSI	***bag-of-words***	0.8574	0.8410	0.6522	0.8249	**0.8469**	0.7757
	bi-gram	**0.8605**	**0.8532**	0.6560	**0.8353**	0.8456	**0.7814**
	Word2Vec	0.6796	0.6528	0.6944	0.5498	0.5943	0.6261
	GloVe	0.8214	0.7985	0.8255	0.6692	0.7502	0.7534
	fastText	0.8287	0.8136	**0.8260**	0.7080	0.7711	0.7576
	LSA	0.6706	0.7377	0.4086	0.6919	0.6624	0.5483
20 NG	***bag-of-words***	**0.7781**	0.7449	**0.7119**	0.5454	0.6339	0.3016
	bi-gram	0.7719	0.7445	0.7084	0.5493	**0.6503**	0.3059
	Word2Vec	0.6596	0.6849	0.5438	0.3030	0.4050	0.5983
	GloVe	0.7205	0.7236	0.5992	0.3755	0.4887	0.6334
	fastText	0.7199	0.7247	0.6031	0.3506	0.4682	0.6502
	LSA	**0.7781**	**0.8058**	0.5104	**0.5806**	0.6472	**0.6861**
Reuters	***bag-of-words***	0.5719	**0.6082**	0.2454	0.3607	0.2884	0.3100
	bi-gram	**0.5762**	0.5992	0.2986	**0.3656**	**0.3262**	0.2809
	Word2Vec	0.2601	0.5003	0.4141	0.1854	0.2235	0.3650
	GloVe	0.3815	0.5762	0.4095	0.1757	0.2423	0.3419
	fastText	0.3096	0.5483	0.4071	0.1614	0.2296	0.3710
	LSA	0.1574	0.4208	**0.4212**	0.2480	0.2928	**0.3777**
AG News	***bag-of-words***	0.9059	0.8964	0.8609	0.8208	0.8639	0.5361
	bi-gram	**0.9087**	**0.8985**	**0.8653**	**0.8209**	**0.8668**	0.5666
	Word2Vec	0.8905	0.8838	0.8369	0.7231	0.8333	0.9020
	GloVe	0.8935	0.8423	0.8517	0.7585	0.8549	0.9044
	fastText	0.8930	0.8676	0.8463	0.7376	0.8440	**0.9068**
	LSA	0.8893	0.8913	0.8334	0.8063	0.8657	0.8995

5.3 The Effect of Word Embedding on Deep Learning-based Approaches

Table 5.2 illustrates the effect of using variant word embeddings for the embedding layer of the deep learning-based approaches that are investigated in this thesis.

LSA word embedding is used to compare the results of pre-trained word embedding approaches that carry global knowledge with the locally trained word embedding that carries context knowledge. Comparing the results of different word embeddings with the results of Random word embedding can show to what extent locally or globally trained word embeddings can help the classifier to perform more effectively.

For the task of sentiment analysis in the IMDB Movie Review dataset, the best result is achieved by the Boosted CNN architecture that uses fastText word embedding for its embedding layer. Two deep CNN architectures, AdvCNN and Boosted CNN, produced their best result with fastText word embedding. Such a similarity in results could be expected due to the similarity between the architectures. The 1D-CNN produced its best result when using the Word2Vec for the embedding layer. The other results are not significantly different, with the exception of Random word embedding, which means that using the trained word embedding can outperform the result of this architecture for approximately 3%. Using GloVe word embedding achieves the best result for the LSTM architecture, while Word2Vec produced the weakest results, even lower than using the Random word embedding.

For the task of HSI, the GloVe word embedding performed better than other word embeddings in three out of five architectures: Deep CNN-LSTM, AdvCNN, and Boosted CNN. The random architecture realized the best result in two architectures, 1D-CNN and LSTM. As this pattern is never repeated, it can be assumed that this result is not persistent. The best result in this dataset belongs to the Boosted CNN architecture that used GloVe document representation.

In the 20 NG dataset, for the task of document classification, the GloVe and the LSA word embedding each attained the best results for two architectures, while fastText performed better than others only in the 1D-CNN. In this dataset, GloVe performed more effectively with architectures that used the LSTM approach. Again, in this dataset, there was 10% up to 42% difference between Random word embedding and the trained word embedding model. Using the Boosted CNN with the LSA document representation created the best result in this task.

Table 5.2. The Effect of Word Embedding on Deep Learning-based Approaches

		1D-CNN	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Word2Vec***	**0.8892**	0.8295	0.8454	0.8983	0.8997
	GloVe	0.8815	**0.8637**	0.8432	0.8923	0.8979
	fastText	0.8881	0.8582	0.8474	**0.9051**	**0.9058**
	LSA	0.8854	0.8512	0.8378	0.8879	0.8980
	Random	0.8578	0.8550	**0.8457**	0.8521	0.8622
HSI	***Word2Vec***	0.7467	0.7657	0.6505	0.7873	0.7936
	GloVe	0.7511	0.7804	**0.6664**	**0.7913**	**0.7966**
	fastText	0.7366	0.7755	0.6508	0.7896	0.7928
	LSA	0.7236	0.7473	0.6077	0.7666	0.7792
	Random	**0.7658**	**0.7827**	0.6319	0.7866	0.7883
20 NG	***Word2Vec***	0.7707	0.7871	0.7804	0.8005	0.7794
	GloVe	0.7808	**0.8183**	**0.8099**	0.8151	0.8049
	fastText	**0.8019**	0.7825	0.7923	0.8174	0.8080
	LSA	0.7963	0.7130	0.7798	**0.8352**	**0.8309**
	Random	0.6943	0.3942	0.4786	0.5232	0.7141
Reuters	***Word2Vec***	**0.5536**	0.4882	0.2785	0.6008	**0.5756**
	GloVe	0.5439	**0.5137**	**0.3867**	0.5707	0.5482
	fastText	0.5260	0.4337	0.3446	**0.6432**	0.5575
	LSA	0.4707	0.2083	0.2372	0.5976	0.5268
	Random	0.4404	0.0908	0.0809	0.4091	0.4762
AG News	***Word2Vec***	0.9212	0.9125	0.8979	0.9181	0.9185
	GloVe	0.9237	**0.9138**	**0.9062**	**0.9243**	0.9139
	fastText	**0.9248**	0.9107	0.9034	0.9226	**0.9199**
	LSA	0.9183	0.9081	0.8990	0.9121	0.9189
	Random	0.9151	0.9055	0.9003	0.8924	0.9085

In the Reuters dataset, GloVe achieved the best results for the LSTM and Deep CNN-LSTM approaches. In contrast to the three previous approaches explained above, the AdvCNN and Boosted CNN produced their best results with different word embeddings. The highest F-score was gained by AdvCNN architecture by using fastText word embedding. The Glove word embedding attained the best result in the two LSTM-based architectures, similar to the 20 NG dataset.

Finally, in the document classification task for the AG News dataset, the fastText word embedding with the 1D-CNN architecture outperformed the other approaches. The GloVe word embedding performed better than other word embeddings in three out of five architectures.

In conclusion, in most of the experiments, neither the Random nor LSA word embedding could achieve the best result while the pre-trained word embeddings performed better with the deep learning-based architectures.

5.4 The Effect of CTWE on Word Embeddings

5.4.1 The Effect of CTWE on Word2Vec

Table 5.3 shows the effect of the CTWE approach when applied to Word2Vec word embedding and employed by different classifiers. In the task of sentiment analysis in the IMDB Movie Review dataset, the CTWE-based word embedding outperformed the original Word2Vec representation for 7 out of 11 classifiers. The highest improvement is observed in the GNB classifier from 0.3416 to 0.7601 in terms of the F_score macro. All of the traditional classifiers showed improvement with CTWE while only the LSTM architecture improved in the deep learning-based approaches. The highest result was achieved by the regular Word2Vec when used with the Boosted CNN architecture.

In the task of HSI, the CTWE approach was more effective than the regular Word2Vec in 3 out of 11 classifiers. Similar to the IMDB Movie Review dataset, the Boosted CNN achieved the best result when used with the original Word2Vec document representation.

In the task of document classification in the 20 NG dataset, the CTWE-based document representation improved the results in five of the classifiers. Also, the best result was achieved by the AdvCNN when used in the CTWE-based Word2Vec representation. The results improved in three out of 11 classifiers when the CTWE was applied to Word2Vec in the Reuters dataset while the best result was achieved by the AdvCNN, which used the original Word2Vec. In the AG News dataset, the best result was produced by the Boosted CNN when the CTEW was used as word representation. In total, six out of 11 classifiers performed better when the CTWE was applied to Word2Vec. The RF classifier showed improvement in all datasets when the CTWE modified the Word2Vec word representation.

Table 5.3. The Effect of CTWE on Word2Vec

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Word2Vec***	0.8577	0.8452	0.3416	0.6589	0.7187	0.7495	0.8892	0.8295	0.8454	0.8983	0.8997
	CTWE	**0.8585**	**0.8576**	**0.7601**	**0.6729**	**0.7289**	**0.7642**	0.8525	**0.8439**	0.8403	0.8895	0.8980
HSI	***Word2Vec***	0.6796	0.6528	0.6944	0.5498	0.5943	0.6261	0.7467	0.7657	0.6505	0.7873	0.7936
	CTWE	0.6442	**0.6844**	**0.7021**	0.5411	**0.5982**	0.6236	0.7386	0.7354	0.6400	0.7675	0.7725
20 NG	***Word2Vec***	0.6596	0.6849	0.5438	0.3030	0.4050	0.5983	0.7707	0.7871	0.7804	0.8005	0.7794
	CTWE	0.5740	0.6568	**0.5439**	**0.3060**	**0.4264**	0.5909	0.7538	0.7268	0.7663	**0.8007**	**0.7870**
Reuters	***Word2Vec***	0.2601	0.5003	0.4141	0.1854	0.2235	0.3650	0.5536	0.4882	0.2785	0.6008	0.5756
	CTWE	0.0739	0.3102	0.4035	**0.2145**	**0.2494**	**0.3659**	0.5205	0.2821	0.2483	0.5755	0.5508
AG News	***Word2Vec***	0.8905	0.8838	0.8369	0.7231	0.8333	0.9020	0.9212	0.9125	0.8979	0.9181	0.9185
	CTWE	0.8859	**0.8920**	**0.8451**	**0.7394**	**0.8387**	0.9017	0.9199	0.9096	**0.9007**	0.9142	**0.9189**

5.4.2 The Effect of CTWE on GloVe

The CTWE approach was applied to GloVe word embedding, the results for which are presented in Table 5.4. In the task of sentiment analysis for the IMDB Movie Review dataset, the CTWE improved all the traditional classifier results as well as the 1D-CNN architecture. The maximum improvement from 0.4268 to 0.7398 in terms of F_score macro is observed in the GNB classifier. The best result was achieved by the Boosted CNN when the original GloVe was used for the word embedding layer. In the HSI dataset, the CTWE approach improved the results of six out of 11 classifiers while the best result was gained by the GNB classifier with the original GloVe. The Deep CNN-LSTM is the only deep learning architecture to be improved by CTWE. Modifying GloVe with CTWE in the results of the 20 NG dataset may have led to an improvement in five classifiers, namely DT, RF, 1D-CNN, AdvCNN and Boosted CNN, with the best result coming from AdvCNN when the CTWE modified the GloVe word embedding. Similar to the 20 NG dataset, the best result in the Reuters dataset was achieved by AdvCNN when CTWE was applied to GloVe, although the CTWE improved the results of only three classifiers. In the AG News dataset, seven out of 11 classifiers were improved by CTWE, with the best result belonging to the 1D-CNN when using CTWE for its embedding layer.

Table 5.4. The Effect of CTWE on GloVe

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	*GloVe*	0.8536	0.8363	0.4268	0.6787	0.7326	0.7397	0.8815	0.8637	0.8432	0.8923	0.8979
	CTWE	**0.8557**	**0.8561**	**0.7398**	**0.6947**	**0.7463**	**0.7591**	**0.8849**	0.8466	0.8429	0.8909	0.8970
HSI	*GloVe*	0.8214	0.7985	0.8255	0.6692	0.7502	0.7534	0.7511	0.7804	0.6664	0.7913	0.7966
	CTWE	**0.8223**	**0.8186**	0.8238	**0.6799**	**0.7749**	**0.7536**	0.7486	0.7730	**0.6805**	0.7766	0.7840
20 NG	*GloVe*	0.7205	0.7236	0.5992	0.3755	0.4887	0.6334	0.7808	0.8183	0.8099	0.8151	0.8049
	CTWE	0.6617	0.7157	0.5940	**0.3777**	**0.5028**	0.6259	**0.8030**	0.7962	0.7801	**0.8204**	**0.8057**
Reuters	*GloVe*	0.3815	0.5762	0.4095	0.1757	0.2423	0.3419	0.5439	0.5137	0.3867	0.5707	0.5482
	CTWE	0.1490	0.4122	**0.4122**	0.1651	0.2353	0.3326	0.4810	0.3846	0.2875	**0.5799**	**0.5742**
AG News	*GloVe*	0.8935	0.8423	0.8517	0.7585	0.8549	0.9044	0.9237	0.9138	0.9062	0.9243	0.9139
	CTWE	0.8924	**0.8930**	**0.8556**	**0.7592**	0.8527	**0.9062**	**0.9265**	**0.9143**	0.9051	0.9219	**0.9177**

5.4.3 The Effect of CTWE on fastText

The performance of almost all the traditional classifiers improved when the CTWE modified the original fastText word embedding in the IMDB Movie Review dataset. However, in this task, none of the deep learning-based architectures showed improvement with the CTWE approach. The greatest improvement occurred in the GNB classifier for 0.3576 improvement in terms of the F_score macro. The Boosted CNN architecture with the original fastText achieved the best accuracy in terms of the F_score macro.

None of the deep learning-based approaches showed improvement when the CTWE was applied to the fastText word embedding in the HSI dataset, while four out of the six traditional classifiers were improved in the evaluation. In contrast, the best result was achieved by the SVM classifier with CTWE and fastText word embedding.

Table 5.5. The Effect of CTWE on fastText

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	*FastText*	0.8660	0.8443	0.3593	0.6725	0.7296	0.7460	0.8881	0.8582	0.8474	0.9051	0.9058
	CTWE	0.8648	**0.8654**	**0.7349**	**0.6899**	**0.7419**	**0.7656**	0.8399	0.8377	0.8416	0.8956	0.9010
HSI	*FastText*	0.8287	0.8136	0.8260	0.7080	0.7711	0.7576	0.7366	0.7755	0.6508	0.7896	0.7928
	CTWE	0.8237	**0.8374**	0.8201	**0.7254**	**0.7853**	**0.7676**	0.7279	0.7571	0.6252	0.7750	0.7822
20 NG	*FastText*	0.7199	0.7247	0.6031	0.3506	0.4682	0.6502	0.8019	0.7825	0.7923	0.8174	0.8080
	CTWE	0.6606	0.7152	0.5988	**0.3605**	**0.4846**	0.6471	0.7878	0.7661	0.7752	**0.8250**	0.8063
Reuters	*FastText*	0.3096	0.5483	0.4071	0.1614	0.2296	0.3710	0.5260	0.4337	0.3446	0.6432	0.5575
	CTWE	0.0983	0.3632	**0.4148**	0.1503	0.2095	0.3612	0.5308	0.3803	0.2436	0.6135	0.5526
AG News	*FastText*	0.8930	0.8676	0.8463	0.7376	0.8440	0.9068	0.9248	0.9107	0.9034	0.9226	0.9199
	CTWE	0.8895	**0.8932**	**0.8470**	**0.7427**	**0.8472**	**0.9071**	0.9221	0.9084	0.8990	0.9180	**0.9221**

In the 20 NG dataset, the fastText-based CTWE word embedding produced the best result when used in the embedding layer of the AdvCNN architecture. Only three out of 11 classifiers improved by applying CTWE over fastText word embedding. In the Reuters dataset, only the GNB classifier showed improvement in terms of the F_score macro when the CTWE modified the fastText word embedding. In this dataset, applying the CTWE caused a drop in the rest of the classifiers. In the AG News dataset, the SVM, GNB, DT, RF and k-NN classifiers improved by applying CTWE over the fastText word embedding. The best result was achieved by the 1D-CNN architecture, which used the original fastText in the embedding layer. Table 5.5 shows the above results in more detail.

5.4.4 The Effect of CTWE on LSA

The effect of CTWE on LSA word representation is demonstrated in Table 5.6. In the task of sentiment analysis for the IMDB Movie Review dataset, in comparison to using the original LSA word embedding, three out of the six traditional classifiers improved when CTWE was applied to the LSA word embedding, while only one of the deep learning-based approaches showed improvement. In the HSI dataset, the results of none of the traditional classifiers improved when CTWE was applied to the LSA and only the LSTM and CNN LSTM approaches showed better results in terms of the F_score macro.

Table 5.6. The Effect of CTWE on LSA

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***LSA***	0.8743	0.8860	0.1456	0.7837	0.6897	0.6780	0.8854	0.8512	0.8378	0.8879	0.8980
	CTWE	0.8261	0.8709	**0.1532**	0.7820	**0.7067**	**0.6791**	0.8695	0.8422	**0.8441**	0.8827	0.8898
HSI	***LSA***	0.6706	0.7377	0.4086	0.6919	0.6624	0.5483	0.7236	0.7473	0.6077	0.7666	0.7792
	CTWE	0.1440	0.6549	0.4068	0.6763	0.6478	0.5418	0.7046	**0.7519**	**0.6348**	0.7543	0.7611
20 NG	***LSA***	0.7781	0.8058	0.5104	0.5806	0.6472	0.6861	0.7963	0.7130	0.7798	0.8352	0.8309
	CTWE	0.6862	0.7722	0.4986	**0.5846**	**0.6474**	**0.6910**	0.7932	0.7008	0.7671	0.8048	0.8033
Reuters	***LSA***	0.1574	0.4208	0.4212	0.2480	0.2928	0.3777	0.4707	0.2083	0.2372	0.5976	0.5268
	CTWE	0.0238	0.2595	**0.4268**	0.2420	0.2879	**0.3833**	**0.4760**	**0.2194**	0.1841	0.5378	**0.5450**
AG News	***LSA***	0.8893	0.8913	0.8334	0.8063	0.8657	0.8995	0.9183	0.9081	0.8990	0.9121	0.9189
	CTWE	0.8753	0.8884	0.8332	**0.8126**	**0.8699**	**0.9028**	**0.9194**	0.9077	0.8965	0.9113	0.9127

In another experiment, none of the deep learning-based approaches improved by applying the CTWE over the 20 NG dataset and comparing the result with the original LSA word representation while three traditional classifiers, DT, RF, and k-NN, showed a slight improvement in the evaluation results.

In the Reuters dataset, five out of 11 classifiers showed an improvement in their F_score macro evaluation results. The best result was achieved by the AdvCNN classifier when the original LSA was used as the embedding layer.

5.4.5 The Effect of CTWE on Random Word Embedding

In the AG Newsgroup dataset, applying CTWE to the LSA caused the best result when the 1D-CNN was used as the classifier. Two traditional and two deep learning-based classifiers showed improvement in their evaluation results.

In Random word embedding, a random vector is assigned to a word. Applying the CTWE to random word vectors injects the local context information into the new word representation. Table 5.7 shows that, in all datasets, applying CTWE to the Random word embedding improved the results in most of the classifiers. In the IMDB Movie Review dataset, eight out of 11 classifiers showed improvement. The best result was achieved by the Boosted CNN when the CTWE was applied to its word embedding.

In the HSI dataset, five of the classifiers produced better results while 10 out of 11 classifiers showed improvement in the 20 NG dataset. The best result was achieved by the Boosted CNN, which used the CTWE approach. In the Reuters dataset, all the deep learning-based approaches were improved when CTWE was applied to the Random word embedding, although the results are not state-of-the-art. However, the best result in this setup is achieved by the Boosted CNN, which used the CTWE. In the AG News dataset, all of the classifiers showed an improvement, with the exception of the Boosted CNN. The best result belongs to the AdvCNN classifier when using the CTWE approach. THE DT, RF, and Deep CNN-LSTM showed improvement in all datasets when the CTWE was applied to the Random word embedding.

Table 5.7. The Effect of CTWE on Random Word Embedding

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Random***	0.7363	0.7359	0.6525	0.5502	0.5360	0.5836	0.8578	0.8550	0.8457	0.8521	0.8622
	CTWE	**0.7364**	**0.7399**	0.6477	**0.5560**	**0.5399**	0.5758	**0.8650**	0.8546	**0.8462**	**0.8673**	**0.8776**
HSI	***Random***	0.6871	0.6823	0.4143	0.4058	0.3595	0.6003	0.7658	0.7827	0.6319	0.7866	0.7883
	CTWE	0.6006	**0.6900**	0.4050	**0.4133**	**0.3780**	**0.6061**	0.7467	0.7562	**0.6419**	0.7701	0.7663
20 NG	***Random***	0.4596	0.4513	0.2211	0.0910	0.1151	0.3142	0.6943	0.3942	0.4786	0.5232	0.7141
	CTWE	0.3898	**0.4556**	**0.2277**	**0.1044**	**0.1232**	**0.3297**	**0.7492**	**0.5870**	**0.7034**	**0.7376**	**0.7505**
Reuters	***Random***	0.2275	0.5008	0.3122	0.0703	0.1073	0.3227	0.4404	0.0908	0.0809	0.4091	0.4762
	CTWE	0.0434	0.2672	0.3053	**0.0790**	**0.1088**	**0.3292**	**0.4632**	**0.1537**	**0.1029**	**0.4256**	**0.5117**
AG News	***Random***	0.6726	0.3984	0.6119	0.3688	0.4277	0.7853	0.9151	0.9055	0.9003	0.8924	0.9085
	CTWE	**0.6775**	**0.6767**	**0.6117**	**0.3716**	**0.4567**	**0.8155**	**0.9136**	**0.9068**	**0.9042**	**0.9177**	0.8897

5.5 The Effect of CMSCT on Word Embeddings

5.5.1 The Effect of CMSCT on Word2Vec

Table 5.8 shows the effect of the CMSCT approach when applied to the Word2Vec word embedding. In the IMDB Movie Review dataset, four out of the six traditional classifiers showed improvement when the CMSCT was applied to the regular Word2Vec word embedding and used as the word representation. Only the LSTM architecture showed a

0.2220 improvement in terms of the F_score macro. The best result was achieved by the Boosted CNN architecture when using the original Word2Vec, followed by the same architecture when the CMSCT was applied to Word2Vec. For the task of classification in the HSI dataset, the 1D-CNN and LSTM showed an improvement out of the deep learning-based approaches. The SVM, DR, and RF also improved when CMSCT was applied to Word2Vec and the average of the word vectors was used for the purpose of document representation. Similar to the IMDB Movie Review movie dataset, the Boosted CNN achieved the best result with an F_score of 0.8997. In the 20 NG dataset, the best result was achieved by the AdvCNN classifier when the CTWE modified the Word2Vec and was used as the embedding layer of the architecture. The DT, RF, LSTM, and AdvCNN were the classifiers that showed an improvement in the F_score macro when the CMSCT was applied to their original word embedding. In the Reuters dataset, only the DEEP CNN-LSTM approach was improved by the CMSCT. The AdvCNN performed better than the other classifiers in this dataset. The 1D-CNN produced the best result in the AG News dataset when using the CMSCT modified Word2Vec approach. The second highest result was achieved by the AdvCNN, again when the CMSCT was applied to Word2Vec and used as the embedding layer. In this experiment, the DT and RF improved together in four out of five datasets, which makes sense as the RF consists of several DTs.

Table 5.8. The Effect of CMSCT on Word2Vec

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Word2Vec***	0.8577	0.8452	0.3416	0.6589	0.7187	0.7495	0.8892	0.8295	0.8454	0.8983	0.8997
	CMSCT	0.8543	0.8541	**0.7416**	**0.6731**	**0.7257**	**0.7557**	0.8850	**0.8517**	0.8447	0.8966	0.8988
HSI	***Word2Vec***	0.6796	0.6528	0.6944	0.5498	0.5943	0.6261	0.7467	0.7657	0.6505	0.7873	0.7936
	CMSCT	0.6634	**0.6581**	0.6546	**0.5628**	**0.5955**	0.6130	**0.7485**	**0.7474**	0.6491	0.7758	0.7771
20 NG	***Word2Vec***	0.6596	0.6849	0.5438	0.3030	0.4050	0.5983	0.7707	0.7871	0.7804	0.8005	0.7794
	CMSCT	0.6317	0.6745	0.5278	**0.3309**	**0.4243**	0.5784	0.7525	**0.7878**	0.7633	**0.8010**	0.7779
Reuters	***Word2Vec***	0.2601	0.5003	0.4141	0.1854	0.2235	0.3650	0.5536	0.4882	0.2785	0.6008	0.5756
	CMSCT	0.1999	0.4367	0.3894	0.1655	0.2231	0.3553	0.5008	0.4572	**0.3076**	0.5996	0.5346
AG News	***Word2Vec***	0.8905	0.8838	0.8369	0.7231	0.8333	0.9020	0.9212	0.9125	0.8979	0.9181	0.9185
	CMSCT	0.8889	**0.8893**	0.8348	**0.7509**	**0.8401**	0.9008	**0.9251**	0.9118	**0.9016**	**0.9208**	0.9145

5.5.2 The Effect of CMSCT on GloVe

Table 5.9 illustrates the result of experiments when the CMSCT was applied to the GloVe word embedding. All the traditional classifiers showed an improvement when the CMSCT was applied to the GloVe word representation in the IMDB Movie Review dataset. Only the Deep CNN-LSTM architecture from the deep learning-based classifiers showed a slight improvement (+0.0065) in terms of the F_score macro. For the task of HSI, only the DT and RF showed an improvement while the evaluation results of the deep learning-based classifiers showed a drop after applying CMSCT to GloVe in comparison with using the original GloVe as the embedding layer. The exact same pattern happened for the 20 NG dataset, when the DT and RF were the only improved classifiers out of all 11 approaches. In the Reuters dataset, the DT, 1D-CNN, and the AdvCNN classifiers were enhanced by applying the GloVe-based CMSCT word representation. Also, the best result in the Reuters dataset was attained by the AdvCNN when using the GloVe-based CMSCT word embedding. For the AG News dataset, almost all of the classifiers, with the exception of the k-NN and LR, showed an increase in terms of F_score macro when the CMSCT was applied to the GloVe word embedding. The AdvCNN was recognized as the best classifier when the GloVe-based CMSCT word embedding was used.

Overall, it may be said that the DT was always enhanced when the CMSCT modified version of the GloVe word embedding was used.

Table 5.9. The Effect of CMSCT on GloVe

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***GloVe***	0.8536	0.8363	0.4268	0.6787	0.7326	0.7397	0.8815	0.8637	0.8432	0.8923	0.8979
	CMSCT	**0.8542**	**0.8548**	**0.7227**	**0.6872**	**0.7458**	**0.7603**	0.8400	0.8608	**0.8497**	0.8816	0.8971
HSI	***GloVe***	0.8214	0.7985	0.8255	0.6692	0.7502	0.7534	0.7511	0.7804	0.6664	0.7913	0.7966
	CMSCT	0.8060	0.7799	0.7906	**0.7186**	**0.7656**	0.7518	0.7448	0.7793	0.6460	0.7887	0.7887
20 NG	***GloVe***	0.7205	0.7236	0.5992	0.3755	0.4887	0.6334	0.7808	0.8183	0.8099	0.8151	0.8049
	CMSCT	0.7048	0.7231	0.5835	**0.3998**	**0.4953**	0.6105	0.7791	0.8160	0.8010	0.8066	0.7921
Reuters	***GloVe***	0.3815	0.5762	0.4095	0.1757	0.2423	0.3419	0.5439	0.5137	0.3867	0.5707	0.5482
	CMSCT	0.2995	0.5241	0.3818	**0.1891**	0.2374	0.3335	**0.5566**	0.5088	0.3332	**0.5755**	0.5333
AG News	***GloVe***	0.8935	0.8423	0.8517	0.7585	0.8549	0.9044	0.9237	0.9138	0.9062	0.9243	0.9139
	CMSCT	0.8923	**0.8931**	**0.8544**	**0.7798**	**0.8645**	0.9006	**0.9251**	**0.9167**	**0.9073**	**0.9248**	**0.9200**

5.5.3 The Effect of CMSCT on fastText

The results of applying CMSCT to the fastText word embedding is illustrated in Table 5.10. In the IMDB Movie Review dataset, all the traditional classifiers, with the exception of the LR, showed improvement, with the most significant change occurring in the GNB when its F_score macro improved from 0.3593 to 0.7155. None of the deep learning-based approaches were enhanced with the fastText-based CMSCT word representation while the best result was achieved by the Boosted CNN classifiers with the original and modified word embedding.

In the HSI dataset, two classifiers from the traditional classifiers (DT and RF) and two classifiers from the deep learning-based architectures (1D-CNN and Deep CNN-LSTM) showed an improvement when the CMSCT was applied to the original fastText word embedding.

In the task of document classification in the 20 NG dataset, six out of 11 of the examined classifiers improved with the CMSCT approach, with the best result achieved by the LSTM when using fastText-based CMSCT word embedding. In the Reuters dataset, the CMSCT approach had no significant effect over the performance of the classifiers. Only the RF and 1D-CNN improved in this experiment.

Table 5.10. The Effect of CMSCT on fastText

		LR	***SVM***	***GNB***	***DT***	***RF***	***KNN***	***1D-CNN***	***LSTM***	***Deep CNN-LSTM***	***AdvCNN***	***BOOSTED CNN***
IMDB	***FastText***	0.8660	0.8443	0.3593	0.6725	0.7296	0.7460	0.8881	0.8582	0.8474	0.9051	0.9058
	CMSCT	0.8592	**0.8603**	**0.7155**	**0.6888**	**0.7369**	**0.7541**	0.8870	0.8434	0.8463	0.9019	0.9053
HSI	***FastText***	0.8287	0.8136	0.8260	0.7080	0.7711	0.7576	0.7366	0.7755	0.6508	0.7896	0.7928
	CMSCT	0.8031	0.7929	0.7957	**0.7371**	**0.7842**	0.7412	**0.7720**	0.7643	**0.6563**	0.7824	0.7876
20 NG	***FastText***	0.7199	0.7247	0.6031	0.3506	0.4682	0.6502	0.8019	0.7825	0.7923	0.8174	0.8080
	CMSCT	0.7056	**0.7305**	0.5841	**0.3893**	**0.5045**	0.6255	0.7818	**0.8214**	**0.7935**	**0.8199**	0.7992
Reuters	***FastText***	0.3096	0.5483	0.4071	0.1614	0.2296	0.3710	0.5260	0.4337	0.3446	0.6432	0.5575
	CMSCT	0.2571	0.4787	0.3912	0.1481	**0.2315**	0.3420	**0.5521**	0.3981	0.3264	0.6059	0.5575
AG News	***FastText***	0.8930	0.8676	0.8463	0.7376	0.8440	0.9068	0.9248	0.9107	0.9034	0.9226	0.9199
	CMSCT	0.8930	**0.8926**	0.8415	**0.7688**	**0.8549**	0.9053	0.9233	**0.9185**	**0.9039**	**0.9229**	0.9226

Using fastText-based word embedding together with the AdvCNN classifier produced the best result in the AG News dataset, although the improvement was only slight in comparison to the original fastText. Three out of six traditional classifiers and three out of five deep learning-based classifiers showed an improvement when using the CMSCT version of fastText.

The RF was the only classifier that always improved when the CMSCT approach was applied to its fastText word embedding. The DT achieved similar results, with the exception of the Reuters dataset. The Boosted CNN and LR never improved when the CMSCT was applied to their word embedding.

5.5.4 The Effect of CMSCT on LSA

Table 5.11 shows the effect of applying CMSCT to an LSA word representation. GNB, RF, and Deep CNN-LSTM are the classifiers that showed an enhancement when CMSCT was applied to the LSA word representation in the IMDB Movie Review dataset. The best result was achieved by the Boosted CNN when using the LSA word representation.

In the HSI dataset, out of the 11 classifiers, the Deep CNN-LSTM is the only classifier whose results were improved by applying the CMSCT to the LSA word representation. Similar to the IMDB Movie Review dataset, the Boosted CNN architecture achieved the best F_score macro. In the 20 NG dataset, three out of five deep learning-based classifiers showed an improvement when the Boosted CMSCT word representation was used as the embedding layer. The RF is the only traditional classifier to show an improvement when using this modified word embedding. In the Reuters dataset, using the CMSCT approach enhanced six out of 11 classifiers: the GNB, RF, and k-NN from the traditional classifiers and 1D-CNN, LSTM, and Boosted CNN from the deep learning-based architectures.

In the AG News dataset, the RF, k-NN, 1D-CNN, and AdvCNN were improved by the CMSCT, while the Boosted CNN with the original LSA embedding layer produced the best result.

Table 5.11. The Effect of CMSCT on LSA

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***LSA***	0.8743	0.8860	0.1456	0.7837	0.6897	0.6780	0.8854	0.8512	0.8378	0.8879	0.8980
	CMSCT	0.8540	0.8775	**0.1845**	0.7637	**0.7061**	0.6675	0.8822	0.8269	**0.8441**	0.8822	0.8945
HSI	***LSA***	0.6706	0.7377	0.4086	0.6919	0.6624	0.5483	0.7236	0.7473	0.6077	0.7666	0.7792
	CMSCT	0.5727	0.7186	0.4062	0.6423	0.6368	0.3906	0.7293	0.7370	**0.6164**	0.7662	0.7769
20 NG	***LSA***	0.7781	0.8058	0.5104	0.5806	0.6472	0.6861	0.7963	0.7130	0.7798	0.8352	0.8309
	CMSCT	0.7639	0.7925	0.4968	0.5751	**0.6536**	0.6881	**0.8019**	**0.7203**	**0.7870**	0.8287	0.8196
Reuters	***LSA***	0.1574	0.4208	0.4212	0.2480	0.2928	0.3777	0.4707	0.2083	0.2372	0.5976	0.5268
	CMSCT	0.1384	0.3852	**0.4462**	0.2338	**0.2946**	**0.4060**	**0.5118**	**0.2633**	0.2050	0.5817	**0.5457**
AG News	***LSA***	0.8893	0.8913	0.8334	0.8063	0.8657	0.8995	0.9183	0.9081	0.8990	0.9121	0.9189
	CMSCT	0.8826	0.8883	0.8230	0.7987	**0.8658**	**0.9021**	**0.9219**	0.9066	0.8978	**0.9124**	0.9167

5.5.5 The Effect of CMSCT on Random Word Embedding

Table 5.12 illustrates the evaluation result of the CMSCT approach when applied to the Random word embedding. In the IMDB Movie Review dataset, the best result was achieved by the Boosted CNN architecture, which used the Random word embedding-based CMSCT as the embedding layer. The DT, k-NN, and 1D-CNN are the other classifiers that were improved by using the random-based CMSCT word representation. In the HSI dataset, four out of 11 classifiers improved, of which three (GNB, DR, and RF) were from the traditional classifiers. The Deep CNN-LSTM architecture was the only architecture to show an improvement in terms of F_score macro.

Table 5.12. The Effect of CMSCT on Random Word Embedding

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Random***	0.7363	0.7359	0.6525	0.5502	0.5360	0.5836	0.8578	0.8550	0.8457	0.8521	0.8622
	CMSCT	0.7336	0.7343	0.6162	**0.5542**	0.5357	**0.5958**	**0.8605**	0.8413	0.8175	0.7513	**0.8679**
HSI	***Random***	0.6871	0.6823	0.4143	0.4058	0.3595	0.6003	0.7658	0.7827	0.6319	0.7866	0.7883
	CMSCT	0.6476	0.6554	**0.4188**	**0.4271**	**0.4025**	0.5813	0.7567	0.7610	**0.6704**	0.7775	0.7505
20 NG	***Random***	0.4596	0.4513	0.2211	0.0910	0.1151	0.3142	0.6943	0.3942	0.4786	0.5232	0.7141
	CMSCT	0.4495	**0.4628**	0.2016	**0.0959**	0.1137	**0.3464**	0.6117	0.3594	0.4318	0.3879	0.7011
Reuters	***Random***	0.2275	0.5008	0.3122	0.0703	0.1073	0.3227	0.4404	0.0908	0.0809	0.4091	0.4762
	CMSCT	0.1416	0.4504	0.2269	**0.0747**	**0.1151**	**0.3237**	**0.4489**	0.0543	0.0682	0.2923	0.4346
AG News	***Random***	0.6726	0.3984	0.6119	0.3688	0.4277	0.7853	0.9151	0.9055	0.9003	0.8924	0.9085
	CMSCT	**0.6993**	**0.6985**	0.5938	**0.3812**	**0.4763**	**0.8315**	**0.9162**	**0.9075**	0.8986	**0.8982**	0.9007

In the 20 NG dataset, none of the deep learning-based classifiers showed enhancement, while 50% of the traditional classifiers improved by using the random-based CMSCT. In the Reuters dataset, 1D-CNN was the only enhanced deep learning architecture. Three traditional classifiers, DT, RF, and k-NN, showed improved evaluation results. In the AG News dataset, five out of six traditional classifiers, and three out of five deep learning architectures, were enhanced by using the random-based CMSCT approach. The 1D-CNN, which used the random-based CMSCT word embedding, achieved the best results out of all the classifiers in this dataset.

The DT is the only classifier in all the datasets that always improved when the CMSCT was applied to its word embedding.

5.6 CMSCT vs. MSCT

Table 5.13 shows a comparison between the CMSCT and MSCT approaches in the IMDB Movie Review dataset. For the traditional classifiers, the MSCT and CMSCT were always higher than the baseline and, in the LSTM approach, the MSCT worked better than the other approaches. For the LR, DT and RF, the CMSCT worked even better than the MSCT, and needed much lower memory during the graph generation process. The MSCT worked better than the CMSCT with the SVM, GNB, k-NN, and LSTM classifiers. For the deep learning-based classifiers, the MSCT worked better with almost all of the approaches, with the exception of the 1D-CNN, for which the CMSCT achieved better results.

Table 5.13. Comparison of CMSCT vs. MSCT

Classifier	*Baseline* *F_score macro*	*CMSCT* *F_score macro*	*MSCT* *F_score macro*
LR	0.8577	**0.8580**	0.8549
SVM	0.8451	0.8411	**0.8557**
GNB	0.3415	0.7045	**0.7438**
DT	0.6588	**0.7078**	0.6715
RF	0.7186	**0.7530**	0.7266
k-NN	0.7495	0.7441	**0.7539**
1D-CNN	**0.8891**	0.8646	0.8402
LSTM	0.8295	0.8352	**0.8423**
CNN+LSTM	**0.8454**	0.8057	0.8251
AdvCNN	**0.8982**	0.8706	0.8943
Boosted CNN	**0.8996**	0.8775	0.8978

5.7 The Effect of AbWE on Word Embeddings

5.7.1 The Effect of AbWE on Word2Vec

The results of applying the introduced AbWE to the Word2Vec are illustrated in Table 5.14. In the IMDB Movie Review dataset, the majority of the traditional classifiers, with the exception of the LR, showed an improvement in terms of F_score when the introduced AbWE was trained by the training data and then applied to the word vectors. The highest improvement occurred for the GNB classifier, where the F_score macro increased from 0.3416 to 0.7336. The Boosted CNN, the only deep learning-based approach to be enhanced, achieved the best result in the evaluation. Applying the AbWE to the Word2Vec word representation in the HSI dataset and AG News failed to register a significant result. Only the LSTM architecture improved in the HSI dataset, and the Deep CNN-LSTM architecture in the AG News dataset showed improvement. In the 20 NG dataset, three traditional classifiers, DT, RF, and k-NN, were slightly improved. In the Reuters dataset, the RF and k-NN were the only enhanced classifiers.

Table 5.14. The Effect of AbWE on Word2Vec

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Word2Vec***	0.8577	0.8452	0.3416	0.6589	0.7187	0.7495	0.8892	0.8295	0.8454	0.8982	0.8996
	AbWE	0.8575	**0.8584**	**0.7336**	**0.6719**	**0.7276**	**0.7779**	0.8416	0.8185	0.8390	0.8611	**0.9052**
HSI	***Word2Vec***	0.6796	0.6528	0.6944	0.5498	0.5943	0.6261	0.7467	0.7657	0.6505	0.7872	0.7935
	AbWE	0.5412	0.6285	0.6234	0.5084	0.5666	0.5567	0.7462	**0.7676**	0.6245	0.7593	0.7740
20 NG	***Word2Vec***	0.6596	0.6849	0.5438	0.3030	0.4050	0.5983	0.7707	0.7871	0.7804	0.8004	0.7794
	AbWE	0.6563	0.6787	0.5219	**0.3076**	**0.4076**	**0.5997**	0.7360	0.7620	0.7246	0.7884	**0.7972**
Reuters	***Word2Vec***	0.2601	0.5003	0.4141	0.1854	0.2235	0.3650	0.5536	0.4882	0.2785	0.6008	0.5755
	AbWE	0.2536	0.4999	0.3467	0.1806	**0.2679**	**0.3705**	0.4349	0.2962	0.1817	0.5395	0.5525
AG News	***Word2Vec***	0.8905	0.8838	0.8369	0.7231	0.8333	0.9020	0.9212	0.9125	0.8979	0.9181	0.9184
	AbWE	0.7992	0.8244	0.7293	0.6725	0.7710	0.7918	0.9202	0.9080	**0.8990**	0.9122	0.9163

5.7.2 The Effect of AbWE on GloVe

Table 5.15 shows the results of the evaluation for when the AbWE was applied to the GloVe word representation. In the IMDB Movie Review dataset, four out of the six traditional classifiers were enhanced in terms of F_score macro when the AbWE modified the GloVe word representation. None of the deep learning-based classifiers improved in this experiment. However, using the original GloVe as their embedding layer, they performed much better than the modified version. The best result was achieved by the Boosted CNN when it used the GloVe word embedding.

In the HSI dataset, the SVM was the only classifier to achieve a positive impact from the AbWE approach. Similar to the HSI dataset, in the AG News dataset, only the SVM classifier showed a better accuracy in terms of F_score. The 1D-CNN and the Boosted CNN architectures improved in the 20 NG dataset when the AbWE modified the original GloVe word embedding, while none of the traditional classifiers were enhanced in this experiment. In the Reuters dataset, the LR and DT from the traditional classifiers with AdvCNN and Boosted CNN from the deep learning-based classifiers improved by applying the AbWE to the GloVe word embedding. The best result was achieved by the Boosted CNN architecture and GloVe-based word embedding.

Table 5.15. The Effect of AbWE on GloVe

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***GloVe***	0.8536	0.8363	0.4268	0.6787	0.7326	0.7397	0.8815	0.8637	0.8432	0.8923	0.8979
	AbWE	0.8533	0.8533	**0.7345**	**0.6913**	**0.7468**	**0.7658**	0.8655	0.8395	0.8482	0.8809	0.8729
HSI	***GloVe***	0.8214	0.7985	0.8255	0.6692	0.7502	0.7534	0.7511	0.7804	0.6664	0.7913	0.7966
	AbWE	0.7859	**0.8213**	0.7740	0.6656	0.7235	0.7203	0.7349	0.7643	0.6145	0.7791	0.7848
20 Newsgro	***GloVe***	0.7205	0.7236	0.5992	0.3755	0.4887	0.6334	0.7808	0.8183	0.8099	0.8151	0.8049
	AbWE	0.7191	0.7195	0.5842	0.3744	0.4715	0.6321	**0.7926**	0.8162	0.7750	0.8150	**0.8096**
Reuters	***GloVe***	0.3815	0.5762	0.4095	0.1757	0.2423	0.3419	0.5439	0.5137	0.3867	0.5707	0.5482
	AbWE	**0.3821**	0.5596	0.3863	**0.1974**	0.2223	0.3407	0.5284	0.4452	0.2398	**0.5772**	**0.5825**
AG News	***GloVe***	0.8935	0.8423	0.8517	0.7585	0.8549	0.9044	0.9237	0.9138	0.9062	0.9243	0.9139
	AbWE	0.8525	**0.8649**	0.8046	0.7257	0.8213	0.8459	0.9196	0.9110	0.8956	0.9175	0.9027

5.7.3 The Effect of AbWE on fastText

Table 5.16 presents the evaluation results for when AbWE is applied to the fastText word embedding. Four traditional classifiers, namely SVM, GNB, RF, and k-NN, showed an improvement when the AbWE modified the fastText word representation in the IMDB

Movie Review dataset, while none of the deep learning-based architectures showed improvement. None of the classifiers were enhanced when the AbWE was applied to fastText in the HSI movie dataset, while the best result in the AG News dataset was achieved by the fastText-based AbWE word embedding when used by the Boosted CNN architecture. However, it should be noted that Boosted CNN was the only classifier to be enhanced in the AG News dataset.

In the 20 NG dataset, the Boosted CNN deep learning architecture, together with the DT and RF traditional classifiers, are the three classifiers to be enhanced by the AbWE approach. The 1D-Classifier in the Reuters dataset was the only enhanced classifier when the AbWE was applied to the fastText word embedding.

Table 5.16. The Effect of AbWE on fastText

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	*FastText*	0.8660	0.8443	0.3593	0.6725	0.7296	0.7460	0.8881	0.8582	0.8474	0.9051	0.9058
	AbWE	0.8654	**0.8671**	**0.7300**	0.6843	**0.7428**	**0.7682**	0.8448	0.8178	0.8464	0.7575	0.8726
HSI	*FastText*	0.8287	0.8136	0.8260	0.7080	0.7711	0.7576	0.7366	0.7755	0.6508	0.7896	0.7928
	AbWE	0.7293	0.8056	0.7765	0.6464	0.7121	0.6713	0.7327	0.7534	0.6502	0.7711	0.7749
20 NG	*FastText*	0.7199	0.7247	0.6031	0.3506	0.4682	0.6502	0.8019	0.7825	0.7923	0.8174	0.8080
	AbWE	0.7127	0.7218	0.5907	**0.3560**	**0.4720**	0.6460	0.7936	0.8020	0.7835	0.8039	**0.8172**
Reuters	*FastText*	0.3096	0.5483	0.4071	0.1614	0.2296	0.3710	0.5260	0.4337	0.3446	0.6432	0.5575
	AbWE	0.3067	0.5314	0.3691	0.1309	0.2166	0.3615	**0.5487**	0.3713	0.1430	0.5840	0.5435
AG News	*FastText*	0.8930	0.8676	0.8463	0.7376	0.8440	0.9068	0.9248	0.9107	0.9034	0.9226	0.9199
	AbWE	0.8070	0.8312	0.7128	0.6417	0.7445	0.8069	0.9201	0.9091	0.9013	0.9183	**0.9209**

5.7.4 The Effect of AbWE on LSA

Table 5.17 shows the evaluation results for the investigated classifiers when AbWE was applied to the LSA word representation. In the IMDB Movie Review dataset, the Deep CNN-LSTM is the only deep learning classifier to be improved by the AbWE that modified the LSA. Three out of the six traditional classifiers, namely GNB, RF, and k-NN, were enhanced by this new version of LSA. In the HSI dataset, the k-NN was the only improved classifier, while two deep learning-based architectures, 1D-CNN and Deep CNN-LSTM, achieved better results when using the modified version of LSA word embedding. In the

20 NG dataset, the k-NN was the only improved classifier out of all the traditional and deep learning-based classifiers, while the best result in this dataset was achieved by the AdvCNN using the original LSA word embedding. In the Reuters dataset, none of the deep learning-based architectures were enhanced while three out of the six traditional classifiers (DT, RF, and k-NN) showed improvement as a result of applying the AbWE to the LSA word representation. None of the classifiers showed improved results in the AG dataset. Moreover, in this dataset, using the AbWE was not effective in comparison to using the original LSA word vectors. The k-NN was enhanced in five out of the six datasets.

Table 5.17. The Effect of AbWE on LSA

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***LSA***	0.8743	0.8860	0.1456	0.7837	0.6897	0.6780	0.8854	0.8512	0.8378	0.8879	0.8980
	AbWE	0.8719	0.8841	**0.1834**	0.7617	**0.6937**	**0.6806**	0.8247	0.8130	**0.8504**	0.7838	0.8269
HSI	***LSA***	0.6706	0.7377	0.4086	0.6919	0.6624	0.5483	0.7236	0.7473	0.6077	0.7666	0.7792
	AbWE	0.2917	0.4396	0.4067	0.4764	0.4321	**0.5648**	**0.7283**	0.7464	**0.6341**	0.7614	0.7603
20 NG	***LSA***	0.7781	0.8058	0.5104	0.5806	0.6472	0.6861	0.7963	0.7130	0.7798	0.8352	0.8309
	AbWE	0.7747	0.8036	0.4222	0.5547	0.6242	**0.6906**	0.7052	0.5758	0.5843	0.7589	0.7742
Reuters	***LSA***	0.1574	0.4208	0.4212	0.2480	0.2928	0.3777	0.4707	0.2083	0.2372	0.5976	0.5268
	AbWE	0.1539	0.4198	0.3841	**0.2550**	**0.3000**	**0.3838**	0.4362	0.1840	0.1029	0.3851	0.4826
AG News	***LSA***	0.8893	0.8913	0.8334	0.8063	0.8657	0.8995	0.9183	0.9081	0.8990	0.9121	0.9189
	AbWE	0.8055	0.8121	0.6741	0.7510	0.8118	0.8475	0.9186	0.9069	0.8959	0.9077	0.9101

5.7.5 The Effect of AbWE on Random Word Embedding

The effect of applying the AbWE approach to Random Word Embedding is shown in Table 5.18. The DT and Deep CNN-LSTM are the two classifiers that showed an improvement when the random-based AbWE approach was used for word representation. In the HSI dataset, the GNB, DT, and RF showed enhancement, while the Deep CNN-LSTM was the only deep learning-based architecture to show an increase in terms of F_score macro when using the random-based AbWE word vectors as the embedding layer. The k-NN is the only improved approach in the 20 NG dataset, while the k-NN, 1D-CNN, and the LSTM architecture showed an enhancement in the Reuters dataset. The LR, 1D-CNN, and

AdvCNN showed improvement in the AG News dataset when the AbWE approach was applied to the Random word embedding. The 1D-CNN with the random-based AbWE word embedding layer produced the best result out of all the deep learning-based and traditional classifiers.

Table 5.18. The Effect of AbWE on Random Word Embedding

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Random***	0.7363	0.7359	0.6525	0.5502	0.5360	0.5836	0.8578	0.8550	0.8457	0.8521	0.8622
	AbWE	0.7344	0.7237	0.6258	**0.5516**	0.5252	0.5587	0.5166	0.8416	**0.8603**	0.7372	0.8002
HSI	***Random***	0.6871	0.6823	0.4143	0.4058	0.3595	0.6003	0.7658	0.7827	0.6319	0.7866	0.7883
	AbWE	0.4798	0.5956	**0.4258**	**0.4230**	**0.3629**	0.3932	0.7476	0.7469	**0.6516**	0.7650	0.7710
20 NG	***Random***	0.4596	0.4513	0.2211	0.0910	0.1151	0.3142	0.6943	0.3942	0.4786	0.5232	0.7141
	AbWE	0.4537	0.4460	0.2078	0.0903	0.1070	**0.3265**	0.6705	0.3835	0.4173	0.1439	0.7023
Reuters	***Random***	0.2275	0.5008	0.3122	0.0703	0.1073	0.3227	0.4404	0.0908	0.0809	0.4091	0.4762
	AbWE	**0.2304**	0.4963	0.2941	0.0699	0.0988	**0.3328**	**0.4810**	**0.4256**	0.0725	0.3875	0.4711
AG News	***Random***	0.6726	0.3984	0.6119	0.3688	0.4277	0.7853	0.9151	0.9055	0.9003	0.8924	0.9085
	AbWE	0.5134	**0.5561**	0.4256	0.3595	0.4135	0.4660	**0.9196**	0.9040	0.8968	**0.9144**	0.9052

5.8 The Effect of EbWC on Word Embeddings

5.8.1 The Effect of EbWC on Word2Vec

Table 5.19 illustrates the result of applying the EbWC approach to the Word2Vec in different classifiers and datasets.

In the IMDB Movie Review dataset, four out of the six traditional classifiers (LR, GNB, DT, and RF) showed enhancement when the EbWC was applied to the Word2Vec word vectors while the LSTM was the only improved deep learning-based classifier. The greatest increase in terms of F_score macro, from 0.3416 to 0.7045, was observed in the GNB classifier. In the HSI dataset, the SVM and RF as traditional classifiers, plus 1D-CNN and LSTM from the deep learning-based classifiers, showed an increase in terms of F_score macro. In the 20 NG dataset, the DT, RF, and Boosted CNN showed enhancement by employing the EbWC-modified Word2Vec while the best result was achieved by the AdvCNN when the original Word2Vec was used. In the Reuters dataset, only the DT and

RF showed improved results by using the EbWC. In the AG News dataset, the SVM, DT, RF, 1D-CNN, and AdvCNN showed improvement. Furthermore, the best evaluation result in this dataset was generated by the 1D-CNN when the EbWC modified the Word2Vec word representation. The RF is the only classifier to improve in all the datasets as a result of employing the EbWC approach.

Table 5.19. The Effect of EbWC on Word2Vec

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Word2Vec***	0.8577	0.8452	0.3416	0.6589	0.7187	0.7495	0.8892	0.8295	0.8454	0.8983	0.8997
	EbWC	**0.8580**	0.8411	**0.7045**	**0.7078**	**0.7530**	0.7442	0.8647	**0.8352**	0.8058	0.8707	0.8776
HSI	***Word2Vec***	0.6796	0.6528	0.6944	0.5498	0.5943	0.6261	0.7467	0.7657	0.6505	0.7873	0.7936
	EbWC	0.6363	**0.6709**	0.6058	0.5002	**0.5971**	0.5998	**0.7625**	**0.7744**	0.6423	0.7740	0.7718
20 NG	***Word2Vec***	0.6596	0.6849	0.5438	0.3030	0.4050	0.5983	0.7707	0.7871	0.7804	0.8005	0.7794
	EbWC	0.5333	0.6083	0.3900	**0.3963**	**0.4538**	0.4516	0.7421	0.7847	0.7414	0.8000	**0.7927**
Reuters	***Word2Vec***	0.2601	0.5003	0.4141	0.1854	0.2235	0.3650	0.5536	0.4882	0.2785	0.6008	0.5756
	EbWC	0.0965	0.3024	0.2173	**0.2334**	**0.2482**	0.2848	0.4747	0.3491	0.1902	0.5077	0.5595
AG News	***Word2Vec***	0.8905	0.8838	0.8369	0.7231	0.8333	0.9020	0.9212	0.9125	0.8979	0.9181	0.9185
	EbWC	0.8787	**0.8845**	0.8066	**0.7915**	**0.8597**	0.8798	**0.9252**	0.9111	0.8990	**0.9186**	0.9167

5.8.2 The Effect of EbWC on GloVe

Table 5.20 shows the evaluation results of the investigated classifiers when the EbWC was applied to the GloVe word representation. In the IMDB Movie Review dataset, almost all the traditional classifiers, with the exception of the k-NN, showed enhancement in terms of F_score macro while none of the deep learning-based investigated solutions saw improvement when the EbWC was applied to the GloVe word embedding. In the HSI dataset, the SVM was the only improved classifier; the remainder showed no increase in terms of F_score macro during the evaluation. Similar to the HSI dataset, the DT was the only enhanced classifier out of all the traditional and deep learning-based approaches. In the Reuters dataset, all the classifiers showed a drop in the evaluation while the EbWC approach failed to work in any of the classifiers. Similar to the 20 NG dataset, the DT is the only enhanced classifier in the AG News dataset. None of the deep learning-based investigated architectures showed an increase in terms of F_score macro when the EbWC approach was applied to the GloVe word embedding.

Table 5.20. The Effect of EbWC on GloVe

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***GloVe***	0.8536	0.8363	0.4268	0.6787	0.7326	0.7397	0.8815	0.8637	0.8432	0.8923	0.8979
	EbWC	**0.8554**	**0.8450**	**0.6212**	**0.7160**	**0.7458**	0.7277	0.8399	0.8380	0.8139	0.8698	0.8664
HSI	***GloVe***	0.8214	0.7985	0.8255	0.6692	0.7502	0.7534	0.7511	0.7804	0.6664	0.7913	0.7966
	EbWC	0.7911	**0.8191**	0.6864	0.7102	0.7479	0.7368	0.7471	0.7688	0.6617	0.7749	0.7792
20 NG	***GloVe***	0.7205	0.7236	0.5992	0.3755	0.4887	0.6334	0.7808	0.8183	0.8099	0.8151	0.8049
	EbWC	0.5565	0.6422	0.2715	**0.4014**	0.4425	0.4133	0.7777	0.7962	0.7194	0.8035	0.7963
Reuters	***GloVe***	0.3815	0.5762	0.4095	0.1757	0.2423	0.3419	0.5439	0.5137	0.3867	0.5707	0.5482
	EbWC	0.0738	0.2391	0.0743	0.1596	0.1794	0.1796	0.4596	0.2876	0.1708	0.4642	0.4615
AG News	***GloVe***	0.8935	0.8423	0.8517	0.7585	0.8549	0.9044	0.9237	0.9138	0.9062	0.9243	0.9139
	EbWC	0.8794	0.8869	0.7310	**0.7905**	0.8468	0.8572	0.9221	0.9131	0.9010	0.9198	0.9121

5.8.3 The Effect of EbWC on fastText

Table 5.21 shows the effect of EbWC on the fastText word representation method. In the IMDB Movie Review dataset, all of the traditional classifiers, as well as the LSTM classifier, were enhanced in terms of F_score macro. The highest increase belonged to the GNB, with an F_score that improved from 0.3593 to 0.6913. The best result in this dataset was achieved by the Boosted CNN, which used the original fastText word embedding. In the HSI dataset, none of the deep learning-based classifiers improved in the evaluation. The SVM, DT, and RF were the traditional classifiers that showed an increase in terms of F_score macro. The SVM, which used the modified (by EbWC) fastText word vectors, produced the best result in this dataset. In the 20 NG and the Reuters datasets, the DT is the only classifier to be positively affected by the EbWC approach. In both datasets, the AdvCNN classifier with the original fastText embedding layer achieved the best results. In the AG News dataset, SVM, DT, RF, and LSTM were the enhanced classifiers. The DT was the only classifier to improve in all datasets when the EbWC was applied to the fastText word representation.

Table 5.21. The Effect of EbWC on fastText

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***FastText***	0.8660	0.8443	0.3593	0.6725	0.7296	0.7460	0.8881	0.8582	0.8474	0.9051	0.9058
	EbWC	**0.8683**	**0.8527**	**0.6913**	**0.7326**	**0.7646**	**0.7548**	0.8633	**0.8637**	0.8366	0.8742	0.8762
HSI	***FastText***	0.8287	0.8136	0.8260	0.7080	0.7711	0.7576	0.7366	0.7755	0.6508	0.7896	0.7928
	EbWC	0.8079	**0.8368**	0.6614	**0.7354**	**0.7759**	0.7541	0.7338	0.7568	0.6382	0.7743	0.7795
20 NG	***FastText***	0.7199	0.7247	0.6031	0.3506	0.4682	0.6502	0.8019	0.7825	0.7923	0.8174	0.8080
	EbWC	0.5380	0.6450	0.3346	**0.4153**	0.4682	0.4501	0.7571	0.7644	0.7375	0.7710	0.7877
Reuters	***FastText***	0.3096	0.5483	0.4071	0.1614	0.2296	0.3710	0.5260	0.4337	0.3446	0.6432	0.5575
	EbWC	0.0710	0.2528	0.1408	**0.1950**	0.2109	0.2329	0.4600	0.1800	0.1349	0.4528	0.5140
AG News	***FastText***	0.8930	0.8676	0.8463	0.7376	0.8440	0.9068	0.9248	0.9107	0.9034	0.9226	0.9199
	EbWC	0.8782	**0.8891**	0.7709	**0.7956**	**0.8568**	0.8704	0.9245	**0.9114**	0.9012	0.9205	0.9112

5.8.4 The Effect of EbWC on LSA

Table 5.22 shows the evaluation result of the investigated classifiers when the EbWC modified the LSA word representation. In the IMDB Movie Review dataset, none of the deep learning-based classifiers showed improvement in evaluation, while the GNB, DT, RF, and k-NN were all improved by the EbWC. In the HSI dataset, three out of five classifiers (1D-CNN, LSTM, and Deep CNN-LSTM) were enhanced by the EbWC approach. The GNB and k-NN were the two improved traditional classifiers.

In the 20 NG dataset, four out of the six traditional classifiers (GNB, DT, RF, and k-NN) and three out of the five deep learning-based classifiers (LSTM, Deep CNN-LSTM, and AdvCNN) were enhanced when the EbWC was applied to the LSA word representation. The highest F_score macro in this dataset was achieved by the AdvCNN when the LSA-based EbWC embedding layer was used.

In the Reuters dataset, the DT, k-NN, LSTM, and Boosted CNN classifiers showed increase in terms of F_score macro.

SVM, GNB, DT, and k-NN, as the traditional classifiers, as well as 1D-CNN, AdvCNN, and Boosted CNN, achieved a higher F_score macro when the EbWC modified the fastText word representation in the AG News dataset. The highest evaluation result (0.9207 in terms of F_score macro) belonged to the Boosted CNN architecture when the modified LSA (by EbWC) word embedding layer was used.

Table 5.22. The Effect of EbWC on LSA

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***LSA***	0.8743	0.8860	0.1456	0.7837	0.6897	0.6780	0.8854	0.8512	0.8378	0.8879	0.8980
	EbWC	0.8672	0.8828	**0.6584**	**0.7976**	**0.8180**	**0.7401**	0.8399	0.8431	0.8318	0.8642	0.8592
HSI	***LSA***	0.6706	0.7377	0.4086	0.6919	0.6624	0.5483	0.7236	0.7473	0.6077	0.7666	0.7792
	EbWC	0.6480	0.7228	**0.4515**	0.6706	0.6527	**0.5610**	**0.7244**	**0.7579**	**0.6199**	0.7664	0.7714
20 NG	***LSA***	0.7781	0.8058	0.5104	0.5806	0.6472	0.6861	0.7963	0.7130	0.7798	0.8352	0.8309
	EbWC	0.7713	0.8001	**0.5877**	**0.6553**	**0.7085**	**0.7163**	0.7678	**0.8120**	**0.7916**	**0.8385**	0.8313
Reuters	***LSA***	0.1574	0.4208	0.4212	0.2480	0.2928	0.3777	0.4707	0.2083	0.2372	0.5976	0.5268
	EbWC	**0.2011**	0.4074	0.3460	**0.2858**	0.2751	**0.4089**	0.4550	**0.4749**	0.2003	0.5567	**0.5327**
AG News	***LSA***	0.8893	0.8913	0.8334	0.8063	0.8657	0.8995	0.9183	0.9081	0.8990	0.9121	0.9189
	EbWC	0.8869	**0.8923**	**0.8620**	**0.8114**	0.8598	**0.9021**	**0.9190**	0.9072	0.8954	**0.9195**	**0.9207**

5.8.5 The Effect of EbWC on Random Word Embedding

Table 5.23 illustrates the evaluation results for the investigated classifiers when the EbWC was applied to the Random word embedding. In the IMDB Movie Review dataset, as well as the 20 NG and Reuters datasets, none of the classifiers showed improvement in evaluation when the EbWC was applied to the Random Embedding. In the HSI dataset, out of all the traditional and deep learning-based architectures, the AdvCNN was the only classifier to be improved by the EbWC in comparison with using Random word embedding. It should be noted that the achieved result is the highest F_score macro in the HSI dataset. In the AG News dataset, the SVM classifier is a traditional classifier while the LSTM and the AdvCNN are the only enhanced classifiers when the EbWC was applied to the Random word embedding.

Table 5.23. The Effect of EbWC on Random Word Embedding

		LR	*SVM*	*GNB*	*DT*	*RF*	*KNN*	*1D-CNN*	*LSTM*	*Deep CNN-LSTM*	*AdvCNN*	*BOOSTED CNN*
IMDB	***Random***	0.7363	0.7359	0.6525	0.5502	0.5360	0.5836	0.8578	0.8550	0.8457	0.8521	0.8622
	EbWC	0.6434	0.0045	0.2537	0.5434	0.5066	0.5194	0.6667	0.7986	0.8385	0.3333	0.8322
HSI	***Random***	0.6871	0.6823	0.4143	0.4058	0.3595	0.6003	0.7658	0.7827	0.6319	0.7866	0.7883
	EbWC	0.0609	0.0588	0.3591	0.3678	0.3372	0.4292	0.7210	0.7714	0.6280	**0.7903**	0.7741
20 NG	***Random***	0.4596	0.4513	0.2211	0.0910	0.1151	0.3142	0.6943	0.3942	0.4786	0.5232	0.7141
	EbWC	0.0269	0.0128	0.0367	0.0799	0.0865	0.1251	0.5791	0.3116	0.4200	0.0163	0.6337
Reuters	***Random***	0.2275	0.5008	0.3122	0.0703	0.1073	0.3227	0.4404	0.0908	0.0809	0.4091	0.4762
	EbWC	0.0079	0.0079	0.0016	0.0495	0.0556	0.1271	0.4244	0.0485	0.0398	0.0339	0.3182
AG News	***Random***	0.6726	0.3984	0.6119	0.3688	0.4277	0.7853	0.9151	0.9055	0.9003	0.8924	0.9085
	EbWC	0.4882	**0.5752**	0.2169	0.3292	0.3794	0.5706	0.9018	**0.9069**	0.8991	**0.8977**	0.8897

5.9 Comparison: The Presented Document Representation Approaches over Logistic Regression Efficiency

Table 5.24 compares the presented approaches in different datasets when the LR is used as the classification solution.

In the IMDB Movie Review dataset, the CTWE was able to achieve the best result in comparison to the other three approaches when it was applied to the Word2Vec, GloVe, and Random word embedding. The EbWC more effectively improved Word2Vec, GloVe and fastText compared to other approaches, but no approach was able to improve the LSA word representation. The best result with the LR classifier was achieved by the original LSA in the IMDB Movie Review dataset.

In the HSI dataset, only the CTWE was able to improve the GloVe word representation while none of the other approaches could improve the baseline results with the LR classifier. The best result was achieved by the fastText word representation in this dataset.

None of the presented solutions was able to improve the baseline result when the LR was used as the classifier in the 20 NG dataset.

The AbWE approach was the only solution in the Reuters dataset that was able to improve the evaluation result for two of the word representations, namely GloVe and Random word embedding. In the Reuters dataset, the best F_score macro was achieved by the GloVe word representation that was modified by the AbWE approach.

Table 5.24. Comparison: The presented document representation approaches over Logistic Regression Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	0.8577	**0.8585**	0.8543	0.8575	**0.8580**
	GloVe	0.8536	**0.8557**	**0.8542**	0.8533	**0.8554**
	fastText	0.8660	0.8648	0.8592	0.8654	**0.8683**
	LSA	**0.8743**	0.8261	0.8540	0.8719	0.8672
	Random	0.7363	**0.7364**	0.7336	0.7344	0.6434
HSI	***Word2Vec***	**0.6796**	0.6442	0.6634	0.5412	0.6363
	GloVe	0.8214	**0.8223**	0.8060	0.7859	0.7911
	fastText	**0.8287**	0.8237	0.8031	0.7293	0.8079
	LSA	**0.6706**	0.1440	0.5727	0.2917	0.6480
	Random	**0.6871**	0.6006	0.6476	0.4798	0.0609
20 NG	***Word2Vec***	**0.6596**	0.5740	0.6317	0.6563	0.5333
	GloVe	**0.7205**	0.6617	0.7048	0.7191	0.5565
	fastText	**0.7199**	0.6606	0.7056	0.7127	0.5380
	LSA	**0.7781**	0.6862	0.7639	0.7747	0.7713
	Random	**0.4596**	0.3898	0.4495	0.4537	0.0269
Reuters	***Word2Vec***	**0.2601**	0.0739	0.1999	0.2536	0.0965
	GloVe	0.3815	0.1490	0.2995	**0.3821**	0.0738
	fastText	**0.3096**	0.0983	0.2571	0.3067	0.0710
	LSA	**0.1574**	0.0238	0.1384	0.1539	0.2011
	Random	0.2275	0.0434	0.1416	**0.2304**	0.0079
AG News	***Word2Vec***	**0.8905**	0.8859	0.8889	0.7992	0.8787
	GloVe	**0.8935**	0.8924	0.8923	0.8525	0.8794
	fastText	**0.8930**	0.8895	**0.8930**	0.8070	0.8782
	LSA	**0.8893**	0.8753	0.8826	0.8055	0.8869
	Random	0.6726	**0.6775**	**0.6993**	0.5134	0.4882

The CMSCT enhanced the results of using fastText and Random Word representation in the AG News dataset. Meanwhile, CTWE had a positive effect on the Random word embedding.

In general, when the LR was used as the classifier in the selected datasets, all the presented approaches were able to enhance at least one word representation.

5.10 Comparison: The Presented Document Representation Approaches over Support Vector Machine Efficiency

Table 5.25 illustrates the evaluation results of using the SVM as the classifier in the five investigated datasets. The same as the LR results in the IMDB Movie Review dataset, none of the classifiers improved the results that were achieved by the original LSA word representation. The CTWE was able to improve the Word2Vec, GloVe, fastText and Random word embedding. Using the AbWE improved the Word2Vec, GloVe and fastText in this dataset. For the IMDB Movie Review dataset, neither the CMSCT nor EbWC achieved the highest F_score macro in any of the investigated word representations. In contrast, the AbWE enhanced Word2Vec, GloVe, and fastText while EbWC was effective on the F_score macro of both Glove and FastText.

In the HSI dataset, the Word2Vec, GloVe and fastText, as well as the Random word embedding, were enhanced by the CTWE approach. The GloVe word representation was improved by using the AbWE solution while the CMSCT was only effective on the Word2Vec word representation. The EbWC improved the baseline result of the Word2Vec, GloVe, and fastText word embedding methods. The highest F_score macro in this dataset was gained when the fastText word embedding was modified by the CTWE approach.

The CMSCT and CTWE are the only solutions that are positively effective in the 20 NG dataset. These approaches improved the Random word embedding. CMSCT was also effective on the fastText.

In the Reuters dataset, applying the four presented approaches resulted in no improvement when the SVM was used as the classifier. In contrast to the evaluation results in the Reuters dataset, all of the word embeddings in the AG News are enhanced when the four presented solutions are applied to the five word representations. The CTWE and CMSCT improved the Word2Vec, GloVe, fastText and Random word embedding. Using the AbWE produced improved results for GloVe and Random word embedding, while using the EbWC enhanced all five investigated word representations.

Table 5.25. Comparison: The presented document representation approaches over Support Vector Machine Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	0.8452	**0.8576**	**0.8541**	**0.8584**	0.8411
	GloVe	0.8363	**0.8561**	**0.8548**	**0.8533**	**0.8450**
	fastText	0.8443	**0.8654**	**0.8603**	**0.8671**	**0.8527**
	LSA	**0.8860**	0.8709	0.8775	0.8841	0.8828
	Random	0.7359	**0.7399**	0.7343	0.7237	0.0045
HSI	***Word2Vec***	0.6528	**0.6844**	**0.6581**	0.6285	**0.6709**
	GloVe	0.7985	**0.8186**	0.7799	**0.8213**	**0.8191**
	fastText	0.8136	**0.8374**	0.7929	0.8056	**0.8368**
	LSA	**0.7377**	0.6549	0.7186	0.4396	0.7228
	Random	0.6823	**0.6900**	0.6554	0.5956	0.0588
20 NG	***Word2Vec***	**0.6849**	0.6568	0.6745	0.6787	0.6083
	GloVe	**0.7236**	0.7157	0.7231	0.7195	0.6422
	fastText	0.7247	0.7152	**0.7305**	0.7218	0.6450
	LSA	**0.8058**	0.7722	0.7925	0.8036	0.8001
	Random	0.4513	**0.4556**	**0.4628**	0.4460	0.0128
Reuters	***Word2Vec***	**0.5003**	0.3102	0.4367	0.4999	0.3024
	GloVe	**0.5762**	0.4122	0.5241	0.5596	0.2391
	fastText	**0.5483**	0.3632	0.4787	0.5314	0.2528
	LSA	**0.4208**	0.2595	0.3852	0.4198	0.4074
	Random	**0.5008**	0.2672	0.4504	0.4963	0.0079
AG News	***Word2Vec***	0.8838	**0.8920**	**0.8893**	0.8244	**0.8845**
	GloVe	0.8423	**0.8930**	**0.8931**	**0.8649**	**0.8869**
	fastText	0.8676	**0.8932**	**0.8926**	0.8312	**0.8891**
	LSA	0.8913	0.8884	0.8883	0.8121	**0.8923**
	Random	0.3984	**0.6767**	**0.6985**	**0.5561**	**0.5752**

5.11 Comparison: The Presented Document Representation Approaches over Naïve Bayes Efficiency

Table 5.26 shows the results of experiments on the five studied datasets, when using the NB classifier, and the four presented approaches applied to the Word2Vec, GloVe, fastText, LSA, and Random word embedding methods. In the IMDB Movie Review dataset, all of the word representations, with the exception of the Random word embedding, are enhanced by applying the CTWE, CMSCT, AbWE, and EbWC approaches. For the NB classifier, the highest result in this dataset is achieved by the Word2Vec when enhanced by the CTWE approach.

In the HSI dataset, CTWE improved the Word2Vec, while the CMSCT and AbWE approaches were only able to improve the results of the Random word embedding approach.

In the 20 NG dataset, the Word2Vec and Random word embedding showed an increase in terms of F_score macro when the CTWE was applied. The LSA word representation was improved by the EbWC approach.

The CTWE was positively effective on the GloVe, fastText, and LSA word embeddings while LSA, the remaining word embedding, was enhanced by the CMSCT approach.

The GloVe word embedding in the AG News dataset was improved by the CTWE and CMSCT approaches. Word2Vec and fastText were enhanced only by CTWE while the LSA word embedding was improved by the EbWC approach.

Table 5.26. Comparison: The presented document representation approaches over Naïve Bayes Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	*Word2Vec*	0.3416	**0.7601**	**0.7416**	**0.7336**	**0.7045**
	GloVe	0.4268	**0.7398**	**0.7227**	**0.7345**	**0.6212**
	fastText	0.3593	**0.7349**	**0.7155**	**0.7300**	**0.6913**
	LSA	0.1456	**0.1532**	**0.1845**	**0.1834**	**0.6584**
	Random	**0.6525**	0.6477	0.6162	0.6258	0.2537
HSI	*Word2Vec*	0.6944	**0.7021**	0.6546	0.6234	0.6058
	GloVe	**0.8255**	0.8238	0.7906	0.7740	0.6864
	fastText	**0.8260**	0.8201	0.7957	0.7765	0.6614
	LSA	**0.4086**	0.4068	0.4062	0.4067	0.4515
	Random	0.4143	0.4050	**0.4188**	**0.4258**	0.3591
20 NG	*Word2Vec*	0.5438	**0.5439**	0.5278	0.5219	0.3900
	GloVe	**0.5992**	0.5940	0.5835	0.5842	0.2715
	fastText	**0.6031**	0.5988	0.5841	0.5907	0.3346
	LSA	0.5104	0.4986	0.4968	0.4222	**0.5877**
	Random	0.2211	**0.2277**	0.2016	0.2078	0.0367
Reuters	*Word2Vec*	**0.4141**	0.4035	0.3894	0.3467	0.2173
	GloVe	0.4095	**0.4122**	0.3818	0.3863	0.0743
	fastText	0.4071	**0.4148**	0.3912	0.3691	0.1408
	LSA	0.4212	**0.4268**	**0.4462**	0.3841	0.3460
	Random	**0.3122**	0.3053	0.2269	0.2941	0.0016
AG News	*Word2Vec*	0.8369	**0.8451**	0.8348	0.7293	0.8066
	GloVe	0.8517	**0.8556**	**0.8544**	0.8046	0.7310
	fastText	0.8463	**0.8470**	0.8415	0.7128	0.7709
	LSA	0.8334	0.8332	0.8230	0.6741	**0.8620**
	Random	**0.6119**	0.6117	0.5938	0.4256	0.2169

5.12 Comparison: The Presented Document Representation Approaches over Decision Tree Efficiency

Table 5.27 compares the evaluation results of the presented approaches when the DT is used as the classifier. The CTWE, CMSCT, and AbWE approaches enhanced the four investigated word embeddings in the IMDB Movie Review dataset: Word2Vec, GloVe, fastText, and the Random word embedding. The EbWC was the only approach that not only improved Word2Vec, GloVe, and fastText, but also improved the LSA word embedding and produced the highest F_score macro in this dataset.

In the HSI dataset, LSA was the only word representation that failed to be improved by any of the approaches. The Word2Vec was enhanced only by the CMSCT, while the GloVe and fastText were improved by the CTWE, the CMSCT, and the EbWC approaches. The Random word embedding was positively impacted by the CTWE, CMSCT, and AbWE approaches.

In the 20 NG dataset, the CTWE approach improved all the investigated word embeddings. The CMSCT improved the results of all the word embeddings with the exception of the LSA approach. Applying the AbWE approach had a positive effect on the Word2Vec and fastText. The EbWC approach improved all the investigated word embeddings, with the exception of the Random word embedding.

In the Reuters dataset, all the word embeddings were improved by at least one of the presented approaches. Word2Vec was enhanced by both the CTWE and EbWC. The GloVe was improved by the CMSCT and the AbWE. The fastText was enhanced by the EbWC while the LSA was improved by the AbWE and the EbWC. The Random word embedding was improved by both the CTWE and CMSCT approaches.

In the AG News dataset, the CTWE improved all the word embedding results and also achieved the highest F_score in this dataset when improved by the LSA word embedding. The CMSCT enhanced all the word embedding methods with the exception of the LSA while the AbWE had no effect on any of the word representations. The EbWC increased the F_score macro of all the word embeddings, with the exception of the Random word embedding.

Table 5.27. Comparison: The presented document representation approaches over Decision Tree Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	0.6589	**0.6729**	**0.6731**	**0.6719**	**0.7078**
	GloVe	0.6787	**0.6947**	**0.6872**	**0.6913**	**0.7160**
	fastText	0.6725	**0.6899**	**0.6888**	**0.6843**	**0.7326**
	LSA	0.7837	0.7820	0.7637	0.7617	**0.7976**
	Random	0.5502	**0.5560**	**0.5542**	**0.5516**	0.5434
HSI	***Word2Vec***	0.5498	0.5411	**0.5628**	0.5084	0.5002
	GloVe	0.6692	**0.6799**	**0.7186**	0.6656	**0.7102**
	fastText	0.7080	**0.7254**	**0.7371**	0.6464	**0.7354**
	LSA	**0.6919**	0.6763	0.6423	0.4764	0.6706
	Random	0.4058	**0.4133**	**0.4271**	**0.4230**	0.3678
20 NG	***Word2Vec***	0.3030	**0.3060**	**0.3309**	**0.3076**	**0.3963**
	GloVe	0.3755	**0.3777**	**0.3998**	0.3744	**0.4014**
	fastText	0.3506	**0.3605**	**0.3893**	**0.3560**	**0.4153**
	LSA	0.5806	**0.5846**	0.5751	0.5547	**0.6553**
	Random	0.0910	**0.1044**	**0.0959**	0.0903	0.0799
Reuters	***Word2Vec***	0.1854	**0.2145**	0.1655	0.1806	**0.2334**
	GloVe	0.1757	0.1651	**0.1891**	**0.1974**	0.1596
	fastText	0.1614	0.1503	0.1481	0.1309	**0.1950**
	LSA	0.2480	0.2420	0.2338	**0.2550**	**0.2858**
	Random	0.0703	**0.0790**	**0.0747**	0.0699	0.0495
AG News	***Word2Vec***	0.7231	**0.7394**	**0.7509**	0.6725	**0.7915**
	GloVe	0.7585	**0.7592**	**0.7798**	0.7257	**0.7905**
	fastText	0.7376	**0.7427**	**0.7688**	0.6417	**0.7956**
	LSA	0.8063	**0.8126**	0.7987	0.7510	**0.8114**
	Random	0.3688	**0.3716**	**0.3812**	0.3595	0.3292

5.13 Comparison: The Presented Document Representation Approaches over Random Forest Efficiency

Table 5.28 illustrates the evaluation results of different word embeddings when the RF classifier is used.

In the IMDB Movie Review dataset, all the word embedding methods, with the exception of the Random word embedding, were improved by applying any one of the presented approaches. The CTWE was the only approach that was able to improve the F_score macro of the Random word embedding. The highest F_score macro was achieved by fastText when the EbWC modified the word representations.

In the HSI dataset, both the CTWE and CMSCT improved the Word2Vec, GloVe, fastText, and Random word embedding methods. The original LSA approach was not improved in this dataset. The AbWE approach was only effective on the Random word embedding while the EbWC improved the results of the Word2Vec and fastText methods.

In the 20 NG dataset, the CTWE improved all the studied word representations. The Random word embedding was the only word representation method that was not improved by the CMSCT in this dataset. The AbWE had a positive effect on the Word2Vec and fastText approaches while the EbWC improved the results of the Word2Vec and the LSA approaches, with the most recent being the highest F_score macro to be achieved in this dataset.

In the Reuters dataset, none of the approaches improved the GloVe word embedding while the Word2Vec was improved by the CTWE, AbWE, and EbWC approaches. CMSCT was the only approach to improve the fastText word embedding while the LSA approach was positively affected by the CMSCT and AbWE approaches. The Random word embedding was improved by applying the CTWE and CMSCT approaches to this dataset.

In the AG News dataset, the CMSCT approach improved all the investigated word embedding methods. In contrast, the AbWE had no positive effect. The CTWE improved all the word embeddings with the exception of the GloVe. The EbWC was effective on both the Word2Vec and fastText word representations in this dataset.

Table 5.28. Comparison: The presented document representation approaches over Random Forest Efficiency

		Baseline	***CTWE***	***CMSCT***	***AbWE***	***EbWC***
IMDB	***Word2Vec***	0.7187	**0.7289**	**0.7257**	**0.7276**	**0.7530**
	GloVe	0.7326	**0.7463**	**0.7458**	**0.7468**	**0.7458**
	fastText	0.7296	**0.7419**	**0.7369**	**0.7428**	**0.7646**
	LSA	0.6897	**0.7067**	**0.7061**	**0.6937**	**0.8180**
	Random	0.5360	**0.5399**	0.5357	0.5252	0.5066
HSI	***Word2Vec***	0.5943	**0.5982**	**0.5955**	0.5666	**0.5971**
	GloVe	0.7502	**0.7749**	**0.7656**	0.7235	0.7479
	fastText	0.7711	**0.7853**	**0.7842**	0.7121	**0.7759**
	LSA	**0.6624**	0.6478	0.6368	0.4321	0.6527
	Random	0.3595	**0.3780**	**0.4025**	**0.3629**	0.3372
20 NG	***Word2Vec***	0.4050	**0.4264**	**0.4243**	**0.4076**	**0.4538**
	GloVe	0.4887	**0.5028**	**0.4953**	0.4715	0.4425
	fastText	0.4682	**0.4846**	**0.5045**	**0.4720**	0.4682
	LSA	0.6472	**0.6474**	**0.6536**	0.6242	**0.7085**
	Random	0.1151	**0.1232**	0.1137	0.1070	0.0865
Reuters	***Word2Vec***	0.2235	**0.2494**	0.2231	**0.2679**	**0.2482**
	GloVe	**0.2423**	0.2353	0.2374	0.2223	0.1794
	fastText	0.2296	0.2095	**0.2315**	0.2166	0.2109
	LSA	0.2928	0.2879	**0.2946**	**0.3000**	0.2751
	Random	0.1073	**0.1088**	**0.1151**	0.0988	0.0556
AG News	***Word2Vec***	0.8333	**0.8387**	**0.8401**	0.7710	**0.8597**
	GloVe	0.8549	0.8527	**0.8645**	0.8213	0.8468
	fastText	0.8440	**0.8472**	**0.8549**	0.7445	**0.8568**
	LSA	0.8657	**0.8699**	**0.8658**	0.8118	0.8598
	Random	0.4277	**0.4567**	**0.4763**	0.4135	0.3794

5.14 Comparison: The Presented Document Representation Approaches over k-Nearest Neighbor Efficiency

Table 5.29 shows the evaluation results for when the k-NN classifier is used. In the IMDB Movie Review dataset, with the exception of the Random word embedding, the CTWE and the AbWE improved the results of all word embeddings. The CMSCT was positively effective on the Word2Vec, GloVe, fastText, and Random word embedding. The EbWC enhanced both fastText and LSA. With fastText, the AbWE achieved the highest F_score macro in the IMDB Movie Review dataset.

In the HSI dataset, none of the approaches improved the Word2Vec. In contrast, the LSA was improved by both the AbWE and EbWC approaches. The CMSCT failed to improve any of the investigated word embeddings, while the CTWE enhanced the results for the GloVe, fastText, and Random word embedding.

In the 20 NG dataset, the CTWE and CMSCT approaches were able to improve the results of the LSA and Random word embedding. The AbWE improved the Word2Vec as well as the LSA and Random word embedding while the EbWC had a positive effective only on the LSA Word Embedding.

Experiments for the Reuters dataset show that the Word2Vec, LSA, and Random word embedding were improved by both the CTWE and AbWE approaches. Similar to the 20 NG dataset, the GloVe and the fastText word embeddings consistently failed to be enhanced by the approaches. The CMSCT was effective over the LSA and Random word embedding while the EbWC was only able to improve the LSA word representation.

In the AG News dataset, similar to the HSI dataset, the Word2Vec did not improve while the CTWE improved all the other word representation methods. The CMSCT enhanced the LSA and Random representations while the EbWC was effective only for the LSA embedding. The AbWE failed to increase the F_score macro of any of the investigated embeddings.

Table 5.29. Comparison: The presented document representation approaches over k-Nearest Neighbor Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	0.7495	**0.7642**	**0.7557**	**0.7779**	0.7442
	GloVe	0.7397	**0.7591**	**0.7603**	**0.7658**	0.7277
	fastText	0.7460	**0.7656**	**0.7541**	**0.7682**	**0.7548**
	LSA	0.6780	**0.6791**	0.6675	**0.6806**	**0.7401**
	Random	0.5836	0.5758	**0.5958**	0.5587	0.5194
HSI	***Word2Vec***	**0.6261**	0.6236	0.6130	0.5567	0.5998
	GloVe	0.7534	**0.7536**	0.7518	0.7203	0.7368
	fastText	0.7576	**0.7676**	0.7412	0.6713	0.7541
	LSA	0.5483	0.5418	0.3906	**0.5648**	**0.5610**
	Random	0.6003	**0.6061**	0.5813	0.3932	0.4292
20 NG	***Word2Vec***	0.5983	0.5909	0.5784	**0.5997**	0.4516
	GloVe	**0.6334**	0.6259	0.6105	0.6321	0.4133
	fastText	**0.6502**	0.6471	0.6255	0.6460	0.4501
	LSA	0.6861	**0.6910**	**0.6881**	**0.6906**	**0.7163**
	Random	0.3142	**0.3297**	**0.3464**	**0.3265**	0.1251
Reuters	***Word2Vec***	0.3650	**0.3659**	0.3553	**0.3705**	0.2848
	GloVe	**0.3419**	0.3326	0.3335	0.3407	0.1796
	fastText	**0.3710**	0.3612	0.3420	0.3615	0.2329
	LSA	0.3777	**0.3833**	**0.4060**	**0.3838**	**0.4089**
	Random	0.3227	**0.3292**	**0.3237**	**0.3328**	0.1271
AG News	***Word2Vec***	**0.9020**	0.9017	0.9008	0.7918	0.8798
	GloVe	0.9044	**0.9062**	0.9006	0.8459	0.8572
	fastText	0.9068	**0.9071**	0.9053	0.8069	0.8704
	LSA	0.8995	**0.9028**	**0.9021**	0.8475	**0.9021**
	Random	0.7853	**0.8155**	**0.8315**	0.4660	0.5706

5.15 Comparison: The Presented Document Representation Approaches over 1D-CNN Efficiency

Table 5.30 shows the result of the proposed approaches when the investigated word representation methods were used as the embedding layer of the 1D-CNN architecture for classification purposes.

In the IMDB Movie Review dataset, the EbWC and the AbWE had no positive effect over the word embeddings. The CMSCT was only able to improve the Random word embedding while the CTWE was effective for the GloVe and Random word embedding.

In the HSI dataset, neither the Glove nor Random word embedding were improved. Further, the CTWE was not effective for any of the investigated word embeddings. The CMSCT enhanced the Word2Vec, fastText and LSA. The AbWE improved the LSA while the EbWC enhanced the LSA as well as the Word2Vec.

The Word2Vec and fastText showed no improvement in the 20 NG dataset when the four presented approaches were applied. The GloVe was enhanced by the CTWE and the AbWE approaches while the LSA was improved by the CMSCT. The CTWE was the only approach to enhance the Random word embedding.

In the Reuters dataset, none of the presented approaches was able to improve the Word2Vec word representation while the CMSCT improved the remaining word representation methods. The CTWE was effective when applied to the fastText, LSA, and Random word embedding. Unlike the EbWC, which was not successful in this dataset, the AbWE improved the fastText and Random word embedding.

In the AG News dataset, all the word embeddings, with the exception of fastText, were enhanced with the CMSCT approach. The CTWE improved the GloVe and LSA while the AbWE was effective for the LSA and Random word embedding. Both the Word2Vec and LSA were enhanced by the EbWC approach.

Table 5.30. Comparison: The presented document representation approaches over CNN Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	**0.8892**	0.8525	0.8850	0.8416	0.8647
	GloVe	0.8815	**0.8849**	0.8400	0.8655	0.8399
	fastText	**0.8881**	0.8399	0.8870	0.8448	0.8633
	LSA	**0.8854**	0.8695	0.8822	0.8247	0.8399
	Random	0.8578	**0.8650**	**0.8605**	0.5166	0.6667
HSI	***Word2Vec***	0.7467	0.7386	**0.7485**	0.7462	**0.7625**
	GloVe	**0.7511**	0.7486	0.7448	0.7349	0.7471
	fastText	0.7366	0.7279	**0.7720**	0.7327	0.7338
	LSA	0.7236	0.7046	**0.7293**	**0.7283**	**0.7244**
	Random	**0.7658**	0.7467	0.7567	0.7476	0.7210
20 NG	***Word2Vec***	**0.7707**	0.7538	0.7525	0.7360	0.7421
	GloVe	0.7808	**0.8030**	0.7791	**0.7926**	0.7777
	fastText	**0.8019**	0.7878	0.7818	0.7936	0.7571
	LSA	0.7963	0.7932	**0.8019**	0.7052	0.7678
	Random	0.6943	**0.7492**	0.6117	0.6705	0.5791
Reuters	***Word2Vec***	**0.5536**	0.5205	0.5008	0.4349	0.4747
	GloVe	0.5439	0.4810	**0.5566**	0.5284	0.4596
	fastText	0.5260	**0.5308**	**0.5521**	**0.5487**	0.4600
	LSA	0.4707	**0.4760**	**0.5118**	0.4362	0.4550
	Random	0.4404	**0.4632**	**0.4489**	**0.4810**	0.4244
AG News	***Word2Vec***	0.9212	0.9199	**0.9251**	0.9202	**0.9252**
	GloVe	0.9237	**0.9265**	**0.9251**	0.9196	0.9221
	fastText	**0.9248**	0.9221	0.9233	0.9201	0.9245
	LSA	0.9183	**0.9194**	**0.9219**	**0.9186**	**0.9190**
	Random	0.9151	0.9136	**0.9162**	**0.9196**	0.9018

5.16 Comparison: The Presented Document Representation Approaches over LSTM Efficiency

Table 5.31 demonstrates the evaluation results for when different word representations are used as the embedding layer of the LSTM architecture. In the IMDB Movie Review dataset, the Word2Vec method is enhanced by the CTWE, CMSCT, and EbWC approaches. The fastText word embedding was improved by the EbWC while the GloVe, LSA, and Random word embedding showed no positive response to any of the presented approaches.

In the HSI dataset, the Word2Vec was enhanced by the AbWE and EbWC approaches and the LSA was improved by the CTWE and EbWC approaches. The GloVe, fastText, and Random word embedding did not change in this dataset, nor did the CMSCT have any positive effect.

In the 20 NG dataset, the CTWE method was effective for the Random word embedding only while the CMSCT enhanced the Word2Vec, fastText, and LSA word representations. The AbWE increased the fastText and the LSA was enhanced by the EbWC approach. The highest score in this dataset was achieved by the fastText when modified by the CMSCT approach.

In the Reuters dataset, the LSA method was enhanced by the CTWE, CMSCT, and EbWC approaches while the Word2Vec, GloVe, and fastText showed no improvement as a result of any of the presented approaches. The Random word embedding showed a positive response to both the CTWE and AbWE approaches.

Finally, in the AG News dataset, the CTWE enhanced the GloVe and Random word embedding, the CMSCT was effective for the GloVe, fastText, and Random word embedding, and the EbWC enhanced both the fastText and Random word embedding. The AbWE was not effective in this dataset and Word2Vec and LSA were not improved by any of the presented approaches.

Table 5.31. Comparison: The presented document representation approaches over LSTM Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	0.8295	**0.8439**	**0.8517**	0.8185	**0.8352**
	GloVe	**0.8637**	0.8466	0.8608	0.8395	0.8380
	fastText	0.8582	0.8377	0.8434	0.8178	**0.8637**
	LSA	**0.8512**	0.8422	0.8269	0.8130	0.8431
	Random	**0.8550**	0.8546	0.8413	0.8416	0.7986
HSI	***Word2Vec***	0.7657	0.7354	0.7474	**0.7676**	**0.7744**
	GloVe	**0.7804**	0.7730	0.7793	0.7643	0.7688
	fastText	**0.7755**	0.7571	0.7643	0.7534	0.7568
	LSA	0.7473	**0.7519**	0.7370	0.7464	**0.7579**
	Random	**0.7827**	0.7562	0.7610	0.7469	0.7714
20 NG	***Word2Vec***	0.7871	0.7268	**0.7878**	0.7620	0.7847
	GloVe	**0.8183**	0.7962	0.8160	0.8162	0.7962
	fastText	0.7825	0.7661	**0.8214**	**0.8020**	0.7644
	LSA	0.7130	0.7008	**0.7203**	0.5758	**0.8120**
	Random	0.3942	**0.5870**	0.3594	0.3835	0.3116
Reuters	***Word2Vec***	**0.4882**	0.2821	0.4572	0.2962	0.3491
	GloVe	**0.5137**	0.3846	0.5088	0.4452	0.2876
	fastText	**0.4337**	0.3803	0.3981	0.3713	0.1800
	LSA	0.2083	**0.2194**	**0.2633**	0.1840	**0.4749**
	Random	0.0908	**0.1537**	0.0543	**0.4256**	0.0485
AG News	***Word2Vec***	**0.9125**	0.9096	0.9118	0.9080	0.9111
	GloVe	0.9138	**0.9143**	**0.9167**	0.9110	0.9131
	fastText	0.9107	0.9084	**0.9185**	0.9091	**0.9114**
	LSA	**0.9081**	0.9077	0.9066	0.9069	0.9072
	Random	0.9055	**0.9068**	**0.9075**	0.9040	**0.9069**

5.17 Comparison: The Presented Document Representation Approaches over Deep CNN-LSTM Tree Efficiency

Table 5.32 shows the effect of the CTWE, CMSCT, AbWE, and EbWC approaches over the investigated word representations when the Deep CNN-LSTM is used as a classification solution in the five selected datasets.

In the IMDB Movie Review dataset, the CTWE was able to improve the LSA and Random word embedding and the CMSCT was effective for the GloVe and LSA. The EbWC was unable to help any of the word representations to improve their evaluation results while the AbWE enhanced the GloVe, LSA, and Random word embedding.

In the HSI dataset, the LSA word representation method was enhanced by all the presented approaches, while none of the approaches proved effective for the Word2Vec method. The GloVe was enhanced by the CTWE and produced the highest F_score macro. The CMSCT improved only the fastText method and the Random word embedding was positively impacted by the CTWE, CMSCT, and AbWE approaches. In the 20 NG dataset, the CTWE approach enhanced the Random word embedding, the CMSCT improved the fastText and LSA, and the EbWC increased the F_score macro of the LSA method. The AbWE was not effective in this dataset and the Word2Vec and GloVe were not improved by the presented approaches.

In the Reuters dataset, Random word embedding and Word2Vec were the only word representations to be enhanced by the CTWE and the CMSCT approaches, respectively.

The Word2Vec method was enhanced by all the presented approaches in the AG News dataset. The GloVe and fastText were enhanced by the CMSCT approaches while the Random word embedding was improved by the CTWE. Meanwhile, the LSA in this dataset could not reach a higher F_score when the presented approaches were applied.

Table 5.32. Comparison: The presented document representation approaches over Deep CNN STM Tree Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	**0.8454**	0.8403	0.8447	0.8390	0.8058
	GloVe	0.8432	0.8429	**0.8497**	**0.8482**	0.8139
	fastText	**0.8474**	0.8416	0.8463	0.8464	0.8366
	LSA	0.8378	**0.8441**	**0.8441**	**0.8504**	0.8318
	Random	0.8457	**0.8462**	0.8175	**0.8603**	0.8385
HSI	***Word2Vec***	**0.6505**	0.6400	0.6491	0.6245	0.6423
	GloVe	0.6664	**0.6805**	0.6460	0.6145	0.6617
	fastText	0.6508	0.6252	**0.6563**	0.6502	0.6382
	LSA	0.6077	**0.6348**	**0.6164**	**0.6341**	**0.6199**
	Random	0.6319	**0.6419**	**0.6704**	**0.6516**	0.6280
20 NG	***Word2Vec***	**0.7804**	0.7663	0.7633	0.7246	0.7414
	GloVe	**0.8099**	0.7801	0.8010	0.7750	0.7194
	fastText	0.7923	0.7752	**0.7935**	0.7835	0.7375
	LSA	0.7798	0.7671	**0.7870**	0.5843	**0.7916**
	Random	0.4786	**0.7034**	0.4318	0.4173	0.4200
Reuters	***Word2Vec***	0.2785	0.2483	**0.3076**	0.1817	0.1902
	GloVe	**0.3867**	0.2875	0.3332	0.2398	0.1708
	fastText	**0.3446**	0.2436	0.3264	0.1430	0.1349
	LSA	**0.2372**	0.1841	0.2050	0.1029	0.2003
	Random	0.0809	**0.1029**	0.0682	0.0725	0.0398
AG News	***Word2Vec***	0.8979	**0.9007**	**0.9016**	**0.8990**	**0.8990**
	GloVe	0.9062	0.9051	**0.9073**	0.8956	0.9010
	fastText	0.9034	0.8990	**0.9039**	0.9013	0.9012
	LSA	**0.8990**	0.8965	0.8978	0.8959	0.8954
	Random	0.9003	**0.9042**	0.8986	0.8968	0.8991

5.18 Comparison: The Presented Document Representation Approaches over AdvCNN Tree Efficiency

Table 5.33 compares the evaluation results of the different approaches when the AdvCNN architecture was used as the classifier.

In the IMDB Movie Review dataset, almost none of the approaches were able to enhance the evaluation results of the baseline methods. In fact, only the CTWE enhanced the Random word embedding from 0.8521 to 0.8673 in terms of F_score macro.

A similar pattern occurred for the HSI dataset and the only improved result was for the Random word embedding that was improved by the EbWC approach.

In the 20 NG dataset, the CTWE enhanced all the word embeddings, with the exception of the LSA. The CMSCT improved the Word2Vec and fastText. The EbWC, which was the only approach to enhance the LSA, also produced the highest F_score macro in this dataset. None of the methods were improved by the AbWE, which was also the case for the IMDB Movie Review dataset, HSI dataset, and the AG News dataset.

In the Reuters dataset, the Word2Vec, GloVe, and LSA were not improved by the presented approaches while the GloVe was enhanced by the CTWE, the CMSCT, and the AbWE approaches. The Random word embedding was enhanced only by the CTWE method.

In the AG News dataset, the CMSCT enhanced all of the word embedding methods in this study and achieved the highest F_score macro with the GloVe word embedding. The CTWE and AbWE was only able to improve the Random word embedding. In contrast, the EbWC was effective in improving the results of the Word2Vec, LSA and Random word embedding.

Table 5.33. Comparison: The presented document representation approaches over AdvCNN Tree Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	**0.8983**	0.8895	0.8966	0.8612	0.8707
	GloVe	**0.8923**	0.8909	0.8816	0.8809	0.8698
	fastText	**0.9051**	0.8956	0.9019	0.7575	0.8742
	LSA	**0.8879**	0.8827	0.8822	0.7838	0.8642
	Random	0.8521	**0.8673**	0.7513	0.7372	0.3333
HSI	***Word2Vec***	**0.7873**	0.7675	0.7758	0.7593	0.7740
	GloVe	**0.7913**	0.7766	0.7887	0.7791	0.7749
	fastText	**0.7896**	0.7750	0.7824	0.7711	0.7743
	LSA	**0.7666**	0.7543	0.7662	0.7614	0.7664
	Random	0.7866	0.7701	0.7775	0.7650	**0.7903**
20 NG	***Word2Vec***	0.8005	**0.8007**	**0.8010**	0.7884	0.8000
	GloVe	0.8151	**0.8204**	0.8066	0.8150	0.8035
	fastText	0.8174	**0.8250**	**0.8199**	0.8039	0.7710
	LSA	0.8352	0.8048	0.8287	0.7589	**0.8385**
	Random	0.5232	**0.7376**	0.3879	0.1439	0.0163
Reuters	***Word2Vec***	**0.6008**	0.5755	0.5996	0.5396	0.5077
	GloVe	0.5707	**0.5799**	**0.5755**	**0.5772**	0.4642
	fastText	**0.6432**	0.6135	0.6059	0.5840	0.4528
	LSA	**0.5976**	0.5378	0.5817	0.3851	0.5567
	Random	0.4091	**0.4256**	0.2923	0.3875	0.0339
AG News	***Word2Vec***	0.9181	0.9142	**0.9208**	0.9122	**0.9186**
	GloVe	0.9243	0.9219	**0.9248**	0.9175	0.9198
	fastText	0.9226	0.9180	**0.9229**	0.9183	0.9205
	LSA	0.9121	0.9113	**0.9124**	0.9077	**0.9195**
	Random	0.8924	**0.9177**	**0.8982**	**0.9144**	**0.8977**

5.19 Comparison: The Presented Document Representation Approaches over Boosted CNN Efficiency

The impact of the presented approaches on the word embeddings when the Boosted CNN is used as the classifier is presented in Table 5.34.

In the IMDB Movie Review dataset, only the Random word embedding and Word2Vec showed improvement. The Random word embedding was enhanced by the CTWE and CMSCT, while the AbWE was able to enhance the Word2Vec method.

None of the presented approaches were successful in the HSI dataset and the highest result was achieved by the baseline GloVe word embedding.

In the 20 NG dataset, the CTWE successfully increased the F_score macro in the Word2Vec, GloVe, and Random word embedding. The AbWE approach enhanced the results of the Word2Vec, GloVe, and fastText methods. The EbWC was successful in enhancing the Word2Vec and LSA word embeddings. The CMSCT was unable to improve any of the studied word embedding methods in this dataset.

The CTWE approach was successful in enhancing all the word embedding methods in the Reuters dataset, with the exception of the GloVe word embedding for which none of the presented approaches could increase the F_score macro. In this dataset, the only enhanced word embedding method with the CMSCT was the fastText. The Word2Vec and Random word embedding were enhanced by the AbWE method, producing the highest F_score macro in this dataset. In the Reuters dataset, the fastText was improved by the EbWC.

In the AG News dataset, the CTWE approach enhanced three word embedding methods: the Word2Vec, GloVe, and fastText. The CMSCT increased the F_score macro for the GloVe and fastText word embeddings. The LSA is the only word representation method in this dataset to be enhanced by the EbWC, the only approach that successfully enhanced it. The fastText was also enhanced by the AbWE. The highest F_score to be achieved in this dataset belongs to the fastText, which was modified by the CMSCT approach.

Table 5.34. Comparison: The presented document representation approaches over Boosted CNN Efficiency

		Baseline	*CTWE*	*CMSCT*	*AbWE*	*EbWC*
IMDB	***Word2Vec***	0.8997	0.8980	0.8988	**0.9053**	0.8776
	GloVe	**0.8979**	0.8970	0.8971	0.8729	0.8664
	fastText	**0.9058**	0.9010	0.9053	0.8726	0.8762
	LSA	**0.8980**	0.8898	0.8945	0.8269	0.8592
	Random	0.8622	**0.8776**	**0.8679**	0.8002	0.8322
HSI	***Word2Vec***	**0.7936**	0.7725	0.7771	0.7741	0.7718
	GloVe	**0.7966**	0.7840	0.7887	0.7848	0.7792
	fastText	**0.7928**	0.7822	0.7876	0.7749	0.7795
	LSA	**0.7792**	0.7611	0.7769	0.7603	0.7714
	Random	**0.7883**	0.7663	0.7505	0.7710	0.7741
20 NG	***Word2Vec***	0.7794	**0.7870**	0.7779	**0.7973**	**0.7927**
	GloVe	0.8049	**0.8057**	0.7921	**0.8096**	0.7963
	fastText	0.8080	0.8063	0.7992	**0.8172**	0.7877
	LSA	0.8309	0.8033	0.8196	0.7742	**0.8313**
	Random	0.7141	**0.7505**	0.7011	0.7023	0.6337
Reuters	***Word2Vec***	0.5482	**0.5742**	0.5333	**0.5825**	0.4615
	GloVe	**0.5575**	0.5526	0.5575	0.5435	0.5140
	fastText	0.5268	**0.5450**	**0.5457**	0.4826	**0.5327**
	LSA	0.4762	**0.5117**	0.4346	0.4711	0.3182
	Random	0.5482	**0.5742**	0.5333	**0.5825**	0.4615
AG News	***Word2Vec***	0.9185	**0.9189**	0.9145	0.9163	0.9167
	GloVe	0.9139	**0.9177**	**0.9200**	0.9027	0.9121
	fastText	0.9199	**0.9221**	**0.9226**	**0.9209**	0.9112
	LSA	0.9189	0.9127	0.9167	0.9101	**0.9207**
	Random	**0.9085**	0.8897	0.9007	0.9052	0.8897

5.20 Comparison between All the Represented Approaches among the Different Document Representations

Figure 5.1 is a heatmap chart that compares the different presented document representation approaches. Each cell indicates the summation of the times where the approach in the column header outperforms the approach in the row title among the different datasets and classifiers. Equation 0.1 shows how each cell is calculated:

$$\mathbf{Cell}(\mathbf{x},\mathbf{y}) = \sum_{\mathbf{i}=\mathbf{0}}^{\mathbf{p}} \boldsymbol{\eta}\left(\mathbf{F}_{\mathbf{score}}(\mathbf{DR}_{\mathbf{y}}), \mathbf{F}_{\mathbf{score}}(\mathbf{DR}_{\mathbf{x}})\right) \qquad \textbf{Equation 0.1}$$

$$\boldsymbol{\eta}(\boldsymbol{\alpha},\boldsymbol{\beta}) = \begin{cases} \mathbf{1}\ \boldsymbol{\alpha} \geq \boldsymbol{\beta} \\ \mathbf{0}\ \boldsymbol{\alpha} < \boldsymbol{\beta} \end{cases}$$

where P (=55, 11 classifiers multiply by five datasets) is the total number of experiments for which the document representation (DR) x or y is used and $\boldsymbol{\eta}$ is the comparison function. The values from 0 to 55 are divided into four levels that are arranged from light to dark according to value.

Out of all the approaches, the baseline methods were most improved by the CMSCT. The CTWE, CMSCT, and AbWE were more successful in improving the GloVe and fastText compared to the Word2Vec, LSA, and Random word embedding. As could have been predicted, the Random word embedding is the weakest embedding. This means that all the investigated word embeddings carry information and that the CTWE and CMSCT more effectively enhanced the baseline word embedding compared to the AbWE and EbWC. The EbWC better improved the baseline word embedding compared to the AbWE. According to Figure 5.1, the presented approaches can be ranked as follows: the CMSCT, CTWE, EbWC and, finally, the AbWE.

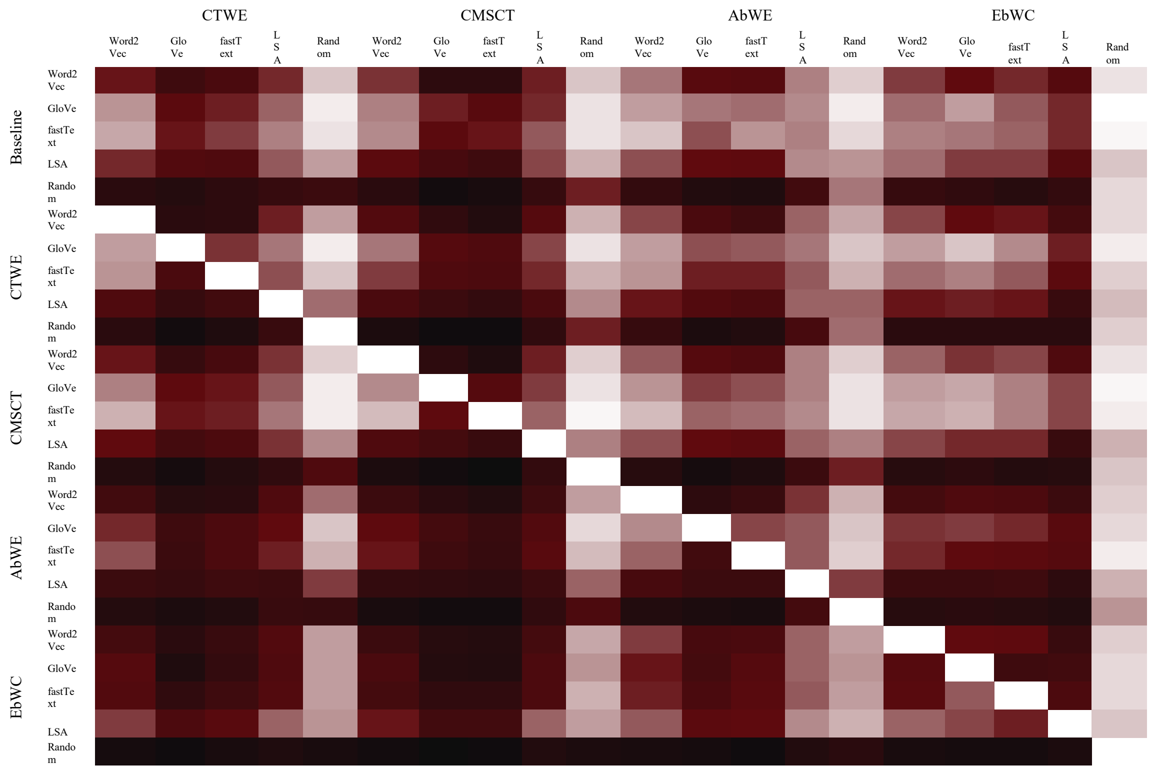

Figure 5.1. Heatmap Chart Comparing Presented Approaches

In summary, the CTWE improved the GloVe, the CMSCT was more successful in improving the fastText, the AbWE enhanced the GloVe, and the EbWC worked more effectively with the LSA in comparison to the other word embedding approaches.

5.21 The Total Effect of the Proposed Approaches Over Word Embeddings

The heatmap chart in Figure 5.2 shows the effect of the presented approaches over different baseline word embeddings. This heatmap concludes the results from Table 5.3 to Table 5.12 and Table 5.14 to Table 5.23. Each cell indicates the summation of the times where the proposed approach (in the column) enhanced the word embedding method in the row among the different datasets and classifiers. Similar to Figure 5.1, the same formula (Equation 0.1) is used to calculate each cell score. The values from 0 to 55 are divided into three levels that are arranged from light to dark according to the value.

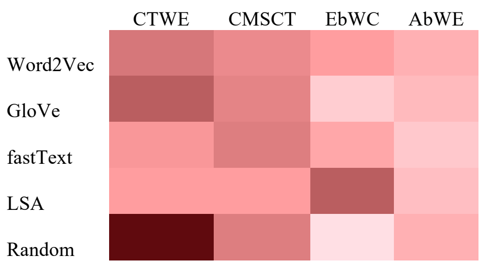

Figure 5.2. Heatmap Chart Comparing the Effect of Proposed Approaches over Word Embeddings

According to Figure 5.2, the Word2Vec is enhanced by CTWE, CMSCT, EbWC and AbWE, respectively. For the GloVe word embedding, the CTWE and CMSCT could be more beneficial approaches. For the fastText, the CMSCT was able to enhance the results more effectively, while the EbWC was the best approach to use for the LSA. According to the results of the random word embedding, the CTWE and CMSCT were able to improve the random results in a more efficient way.

5.22 Comparison with the State-Of-the-Art Results

In this present work, comparing state-of-the-art results may not be regarded as peer to peer since this evaluation calculates the F_score macro while other studies either reported other metrics or failed to clearly mention their metrics. For the IMDB Movie Reviews dataset, the highest reported result by Mesnil et al. is 92.57 in terms of accuracy [80]. As it is a binary classification with equal observations from each class in training and testing, it can be assumed that the accuracy and F_score macro are almost equal. In this present study, the maximum achieved result for the IMDB Movie Reviews dataset was 90.52, which was obtained by applying the MSCT over the fastText word embedding and employing the Boosted CNN architecture.

For the HSI dataset, the highest reported result, by Almeida et al., is 96 in terms of F_score [82]. This present study used a modified version of the dataset and changed the number of classes from three to two by removing one of the classes. The highest result in terms of F_score is 83.73, obtained by applying the CTWE approach to the fastText and using the SVM classifier.

For the 20 NG dataset, an F_score of 96.49 was reported by Lai et al. [84]. In this present study, the highest result in terms of F_score macro, 83.84, was obtained by applying the EbWC approach to the LSA and using AdvCNN architecture.

For the Reuters-21578 dataset, an F_score of 87.89 was reported by Nam et al. in [86]. In this present study, the highest reported result is 61.35 in terms of F_score, which was obtained by applying the CTWE approach to the fastText and employing the AdvCNN architecture.

The state-of-the-art result for the AG News dataset is 7.64 in terms of error rate, as reported by Conneau et al. [88]. In the present study, a 92.64 F_score macro was achieved by applying the CTWE approach to the GloVe word embedding and using the 1D-CNN architecture. This result is very close to the state-of-the-art results.

5.23 Summary

In this section, four novel approaches, namely CTWE, CMSCT, EbWC, and AbWE, are applied to five baseline word embedding methods: Word2Vec, GloVe, fastText, LSA, and Random. These five popular NLP task datasets, together with 11 classifiers, are used to evaluate the presented approaches. A total of 1,111 experiments are conducted. The effectiveness of the four novel approaches are evaluated and discussed over each word embedding as well as each classifier. All the newly generated word representations are ultimately compared and ranked against each other, as well as the baseline word embeddings. In summary, the GloVe word embedding is improved by CTWE, the CMSCT is successful in improving the fastText, the AbWE slightly enhances the GloVe, while the EbWC works more effectively with the LSA when compared to the other word embedding approaches. From a classifier perspective, when the SVM is used, the CTWE and CMSCT improve the Word2Vec, GloVe, fastText and Random Word Embeddings. The AbWE improves the results for the GloVe and Random Word Embedding while the EbWC enhances all five word representations. By using the NB classifier, all four approaches on the IMDB Movie dataset are improved. When the DT and RF are used, CTWE and CMSCT show consistent improvement. CTWE, CMSCT and AbWE improve the baseline results in the case of the KNN. When the deep learning architectures of CNN, Deep CNN-LSTM, and AdvCNN are used for classification, CMSCT and CTWE outperform the two other approaches and the baseline in the majority of the experiments while the LSTM architecture fails to benefit from the use of updated word representations.

Chapter 6. Conclusion and Future Works

6.1 Summary and Conclusion

In this thesis, four novel approaches are presented in an attempt to combine local context information with the global knowledge that is pre-trained over large corpuses of text. The main application of these word embeddings is to provide more accurate document representation for the task of text classification.

The presented approaches attempt to improve three well-known word embedding methods, namely: Word2Vec, GloVe, and fastText. In order to compare the evaluation results, two other methods are considered as the baseline: the locally trained method of LSA and Random Word Embedding, which assigns a random vector to each word in the training dataset.

A comprehensive literature review of the related research in the domain of word embedding and document representation is presented in Chapter 2.

The four presented methods, which are described in Chapter 3, are: Content Tree Word Embedding (CTWE); Composed Maximum Spanning Content Tree (CMSCT); Embedding-based Word Clustering (EbWC): and Autoencoder-based Word Embedding (AbWE).

These approaches create a document representation for each word in the training vocabulary based on the Word2Vec, GloVe, fastText, LSA, and Random word embedding models, then use the average of the updated word vectors for document representation.

The first approach, CTWE, employs a semi-taxonomy structure named content tree, and subsequently updates the word embedding vectors.

The second approach, MSCT, is proposed in order to select the root based on the node degree and then generate the maximum spanning tree. Another version of this approach, called CMSCT, is defined. This approach does not require a high level of memory to generate a fully connected graph of all the words in the training vocabulary. In addition, CMSCT generates a small-size spanning tree for each document, then generates a training data spanning tree by combining them. In the final step, the word vectors are updated based on their location in the maximum spanning tree.

The third approach, EbWC, uses the clustering method for extracting the conceptual structure of the context. Each element of the new word embedding is the distance from the centroid of each word group cluster.

The AbWE method uses autoencoder for dimensionality and noise reduction in the context. The main intention of this present work was to train an autoencoder to capture the training data concepts and then update the word vectors by encoding them.

In order to evaluate the presented approaches, each approach is applied over the five mentioned word embeddings. The new word vectors are then used for the evaluation task over five well-known datasets in the domain of NLP and text mining. The five datasets are: IMDB Movie Reviews; Hate Speech Identification; 20 NG; Reuters-21578; and AG News.

In order to show the effect of each approach over the different types of classifiers, 11 different classifiers are employed, six of which are traditional classifiers: LR, SVM, GNB, DT, RF, and KNN. Five other classifiers were designed based on deep learning solutions (1D-CNN, LSTM, Deep CNN-LSTM, AdvCNN, and Boosted CNN). Overall, 1,111 experiments are executed, for which the evaluation results are presented and described in Chapter 4.

According to the experiments with the original (non-modified) document representation methods and the traditional classifiers that are shown in Table 5.1, it can be inferred that the bi-gram and LSA document representation approaches can work more effectively with traditional classifiers. In contrast, taking an average of the word vectors failed to achieve the best results in the majority of experiments.

As shown in

Table 5.2, which illustrates the effect of using variant word embedding for the embedding layer of the deep learning-based approaches, the Random and the LSA Word Embeddings could not achieve the best results in the majority of experiments, while the pre-trained word embeddings performed better with the deep learning-based architectures. These results confirm that using pre-trained word embeddings that use global knowledge can be employed as a successful approach for text mining challenges.

The results of applying the CTWE approach to the studied word embeddings are illustrated in Tables 5.3 to 5.7, shown in different patterns depending on the dataset and the used classifier. The CTWE mostly improves the results when a traditional classifier is used rather than a deep learning-based solution. It should be considered that the deep learning-based classifiers outperformed the traditional classifiers. In those cases where CTWE improved the results of a deep learning-based approach, it was the best result among all the traditional and deep learning-based approaches. As an example, this pattern is observed in the IMDB movie, 20 NG, Reuters, and AG news datasets when the Word2Vec that was employed by the boosted CNN or the AdvCNN was improved by CTWE and achieved the highest F_score macro.

It can be concluded that the CTWE is more effective on the traditional classifiers, with a lower performance level than the deep learning approaches while, when the classifier is powerful, there is no remaining capacity for improvement.

Tables 5.8 to 5.12 present the effect of CMSCT on the investigated word embeddings. Similar to CTWE, the CMSCT was more effective with the traditional approaches that use the average of the words in the document as the feature vector rather than the deep learning-based solutions that use the new word representation in their embedding layer.

The effect of the AbWE on word embeddings is presented in Tables 5.14 to 5.18, and the effect of the EbWC approach is presented in Tables 5.19 to 5.23. However, these two approaches did not have as much influence over the results as the CTWE and CMSCT. Greater improvement occurred for the traditional word embedding areas than situations where the deep learning-based solutions were used.

In order to compare all the different presented document representation approaches, for each two approaches where one has outperformed the other, a summation of the times among the different datasets and different classifiers is calculated and shown in Figure 5.1. Compared to the other approaches, the CMSCT improved the baseline methods to a higher level. The CTWE, CMSCT, and AbWE were more successful in improving the GloVe and fastText compared to the Word2Vec, LSA, and Random word embedding. The Random word embedding is the weakest embedding, which could have been predicted. This means that all the studied word embeddings carry information and that the CTWE and the CMSCT more effectively enhanced the baseline word embedding than either AbWE or EbWC. The EbWC more successfully improved the baseline word embedding than the AbWE. The CTWE and AbWE improved the GloVe, the CMSCT was more successful in improving the fastText, and the EbWC works better with the LSA in comparison with other word embedding approaches.

Based on this comparison, the presented approaches can be ranked as follows: the CMSCT, the CTWE, the EbWC and, finally, the AbWE.

Tables 5.24 to 5.34 show the effect of the presented approaches on each investigated classifier.

For the LR, none of the approaches could show significant or persistent improvement. Applying the presented approaches works more efficiently when using the SVM classification. The CTWE and CMSCT improved the Word2Vec, GloVe, fastText and Random word embedding. Using the AbWE improved the results of GloVe and Random word embedding, while using the EbWC enhanced all five investigated word representations.

Although its results were not promising in general, the Naïve Bayes results were improved by applying the four presented approaches in the IMDB movie dataset, but not significantly in the other datasets.

When the DT and RF were used as the classifier, the presented approaches showed improvement in the majority of the datasets, particularly CTWE and CMSCT, which showed consistent improvement in the results. The KNN works much better with CTWE, CMSCT and AbWE versus the EbWC in a different dataset.

The CMSCT and CTWE improved the results of 1D-CNN in the majority of the experiments, while the LSTM approach worked more efficiently with the baseline word embeddings, although in some cases one approach could improve the results.

The effect of the approaches was not consistent for the Deep CNN-LSTM classifier, the LSA and the Random word embedding, which produced better results when the CTWE or

CMSCT was applied over their baseline word vectors. However, improvements to the Word2Vec, GloVe, and fastText were inconsistent.

Depending on the dataset, the AdvCNN showed better results with different word embeddings. In 20 NG, the CTWE boosted the results and in the AG News dataset, the CMSCT consistently showed improvement. In the other datasets, the baseline word representation produced the better results in the majority of experiments. Similar to the AdvCNN, the boosted CNN architecture was not always improved, and in different datasets, different approaches could or could not improve the results.

In general, it can be concluded that the best combination of classification approach and word embedding differed according to the nature of the data and the task.

For a binary and balanced dataset, similar to the IMDB movie reviews, the SVM, NB, DT, and RF classifiers can benefit from all four presented approaches. For the KNN classifier, although CTWE, CMSCT, and AbWE showed enhancement, the EbWC could not improve results.

For a binary dataset which is unbalanced, similar to the HSI dataset, the SVM with CTWE and RF with CTWE and CMSCT can create appropriate combinations.

For categorical datasets that have a balanced distribution between their classes, similar to the AG News and 20 NG datasets, when the number of categories is limited, the SVM or NB with CTWE can be a good combination as well as the DT, RF, and AdvCNN with CTWE and CMSCT. When the number of classes is higher, similar to the 20 NG dataset, the AdvCNN with CTWE can be a good option.

For categorical and unbalanced datasets, such as the Reuters-21578, although the results failed to show a consistent pattern, it can be concluded that the combination of the CNN family with CTWE could be a good option.

6.2 Future Works

The following issues are the most important in terms of requiring further investigation:

- In order to fairly compare the results of different approaches, all the parameters among the traditional classifiers and deep learning-based architectures are used with their default values. While the parameter tuning can change, the final results for each combination of document representation and classifier should be further investigated in another independent environment.
- Similar to the above issue, further studies could be conducted into how word representation can be changed by different parameters such as vector length, number of learning iterations, and training text corpuses. The investigated tasks that are defined for this study are classification approaches. A similar study should be conducted for other text mining and NLP tasks such as clustering.
- The present study was conducted only for English documents. However, it would be valuable to repeat such a study for other languages.

References

[1] C. Wei, S. Luo, X. Ma, H. Ren, J. Zhang, and L. Pan, "Locally Embedding Autoencoders: A Semi-Supervised Manifold Learning Approach of Document Representation," *PloS one,* vol. 11, no. 1, 2016.

[2] Z. S. Harris, "Distributional structure," *Word,* vol. 10, no. 2-3, pp. 146-162, 1954.

[3] K. Pearson, "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science,* vol. 2, no. 11, pp. 559-572, 1901.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research,* vol. 3, no. Jan, pp. 993-1022, 2003.

[5] A. Jung, "An introduction to a new data analysis tool: Independent Component Analysis," in *Proceedings of Workshop GK" Nonlinearity"-Regensburg*, 2001.

[6] S. Deerwester, "Improving information retrieval with latent semantic indexing," 1988.

[7] E. E. Milios, M. M. Shafiei, S. Wang, R. Zhang, B. Tang, and J. Tougas, "A systematic study on document representation and dimensionality reduction for text clustering," Technical report, Faculty of Computer Science, Dalhousie University2006.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781,* 2013.

[9] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global Vectors for Word Representation," in *EMNLP*, 2014, vol. 14, pp. 1532-43.

[10] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606,* 2016.

[11] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *arXiv preprint arXiv:1405.4053,* 2014.

[12] M. Bernotas, K. Karklius, R. Laurutis, and A. Slotkienė, "The peculiarities of the text document representation, using ontology and tagging-based clustering technique," *Information Technology And Control,* vol. 36, no. 2, 2015.

[13] R. Navigli, P. Velardi, and S. Faralli, "A graph-based algorithm for inducing lexical taxonomies from scratch," in *IJCAI*, 2011, pp. 1872-1877.

[14] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European conference on machine learning*, 1998, pp. 137-142: Springer.

[15] F. Debole and F. Sebastiani, "An analysis of the relative hardness of Reuters-21578 subsets," *Journal of the American Society for Information Science and technology,* vol. 56, no. 6, pp. 584-596, 2005.

[16] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science,* vol. 41, no. 6, pp. 391-407, 1990.

[17] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, "Towards universal paraphrastic sentence embeddings," *arXiv preprint arXiv:1511.08198,* 2015.

[18] J. Jiang, "Information extraction from text," in *Mining text data*: Springer, 2012, pp. 11-41.

[19] E. Riloff and W. Lehnert, "Information extraction as a basis for high-precision text classification," *ACM Transactions on Information Systems (TOIS),* vol. 12, no. 3, pp. 296-333, 1994.

[20] M. Kamkarhaghighi and M. Makrehchi, "Content Tree Word Embedding for Document Representation," *Expert Systems with Applications,* 2017.

[21] J. SzymańSki, "Comparative analysis of text representation methods using classification," *Cybernetics and Systems,* vol. 45, no. 2, pp. 180-199, 2014.

[22] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using Wikipedia-based explicit semantic analysis," in *IJcAI*, 2007, vol. 7, pp. 1606-1611.

[23] M. Li and P. Vitányi, *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media, 2009.

[24] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, 2011, pp. 142-150: Association for Computational Linguistics.

[25] J. Yang, Z. Liu, and Z. Qu, "Text Representation Based on Key Terms of Document for Text Categorization," *International Journal of Database Theory and Application,* vol. 9, no. 4, pp. 1-22, 2016.

[26] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016.

[27] B. Lao and K. Jagadeesh, "Classifying Legal Questions into Topic Areas Using Machine Learning," 2014.

[28] H. K. Kim, H. Kim, and S. Cho, "Distributed Representation of Documents with Explicit Explanatory Features," 2014.

[29] Q. Lu, J. G. Conrad, K. Al-Kofahi, and W. Keenan, "Legal document clustering with built-in topic segmentation," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 383-392: ACM.

[30] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Advances in Neural Information Processing Systems*, 2013, pp. 926-934.

[31] A. M. Schakel and B. J. Wilson, "Measuring word significance using distributed representations of words," *arXiv preprint arXiv:1508.02297,* 2015.

[32] J. Goodman, "Classes for fast maximum entropy training," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, 2001, vol. 1, pp. 561-564: IEEE.

[33] K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. Smola, "Feature hashing for large scale multitask learning," *arXiv preprint arXiv:0902.2206,* 2009.

[34] M. Naili, A. H. Chaibi, and H. H. B. Ghezala, "Comparative study of word embedding methods in topic segmentation," *Procedia Computer Science,* vol. 112, pp. 340-349, 2017.

[35] S. Hong, "Improving Paragraph2Vec," 2016.

[36] Y. Hong and T. Zhao, "Automatic Hilghter of Lengthy Legal Documents," 2015.

[37] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075,* 2015.

[38] D. Cer *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175,* 2018.

[39] M. Iyyer, V. Manjunatha, J. Boyd-Graber, and H. Daumé III, "Deep unordered composition rivals syntactic methods for text classification," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015, vol. 1, pp. 1681-1691.

[40] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998-6008.

[41] Z. Zhu and J. Hu, "Context Aware Document Embedding," *arXiv preprint arXiv:1707.01521,* 2017.

[42] P. Mirowski, M. Ranzato, and Y. LeCun, "Dynamic auto-encoders for semantic indexing," in *Proceedings of the NIPS 2010 Workshop on Deep Learning*, 2010, vol. 2.

[43] M. A. Ranzato and M. Szummer, "Semi-supervised learning of compact document representations with deep networks," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 792-799: ACM.

[44] H. Palangi *et al.*, "Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP),* vol. 24, no. 4, pp. 694-707, 2016.

[45] Y. Belinkov, N. Durrani, F. Dalvi, H. Sajjad, and J. Glass, "What do Neural Machine Translation Models Learn about Morphology?," *arXiv preprint arXiv:1704.03471,* 2017.

[46] A. Søgaard and Y. Goldberg, "Deep multi-task learning with low level tasks supervised at lower layers," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2016, vol. 2, pp. 231-235.

[47] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," *Transactions of the Association for Computational Linguistics,* vol. 3, pp. 211-225, 2015.

[48] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography," *Computational linguistics,* vol. 16, no. 1, pp. 22-29, 1990.

[49] E. Altszyler, M. Sigman, S. Ribeiro, and D. F. Slezak, "Comparative study of LSA vs Word2vec embeddings in small corpora: a case study in dreams database," *arXiv preprint arXiv:1610.01520,* 2016.

[50] M. E. Peters *et al.*, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365,* 2018.

[51] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805,* 2018.

[52] P. Buitelaar, P. Cimiano, and B. Magnini, *Ontology learning from text: methods, evaluation and applications*. IOS press, 2005.

[53] R. Nazar, J. Vivaldi, and L. Wanner, "Automatic taxonomy extraction for specialized domains using distributional semantics," *Terminology,* vol. 18, no. 2, pp. 188-225, 2012.

[54] M. B. Aouicha, M. A. H. Taieb, and M. Ezzeddine, "Derivation of "is a" taxonomy from Wikipedia Category Graph," *Engineering Applications of Artificial Intelligence,* vol. 50, pp. 265-286, 2016.

[55] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proceedings of the 14th conference on Computational linguistics-Volume 2*, 1992, pp. 539-545: Association for Computational Linguistics.

[56] D. Sánchez and A. Moreno, "Pattern-based automatic taxonomy learning from the Web," *Ai Communications,* vol. 21, no. 1, pp. 27-48, 2008.

[57] A. B. Rios-Alvarado and I. Lopez-Arevalo, "Ontology Learning by using text clustering techniques," 2010.

[58] W. L. Woon and S. Madnick, "Asymmetric information distances for automated taxonomy construction," *Knowledge and information systems,* vol. 21, no. 1, pp. 91-111, 2009.

[59] K. Meijer, F. Frasincar, and F. Hogenboom, "A semantic approach for extracting domain taxonomies from text," *Decision Support Systems,* vol. 62, pp. 78-93, 2014.

[60] M. Makrehchi and M. S. Kamel, "Automatic taxonomy extraction using google and term dependency," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, 2007, pp. 321-325: IEEE Computer Society.

[61] P. Heymann and H. Garcia-Molina, "Collaborative creation of communal hierarchical taxonomies in social tagging systems," 2006.

[62] H. Bast, G. Dupret, D. Majumdar, and B. Piwowarski, "Discovering a term taxonomy from term similarities using principal component analysis," in *Semantics, Web and Mining*: Springer, 2006, pp. 103-120.

[63] H. Yang and J. Callan, "Feature selection for automatic taxonomy induction," in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, 2009, pp. 684-685: ACM.

[64] H. Yang and J. Callan, "A metric-based framework for automatic taxonomy induction," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International*

Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1, 2009, pp. 271-279: Association for Computational Linguistics.

[65] X. Liu, Y. Song, S. Liu, and H. Wang, "Automatic taxonomy construction from keywords," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 1433-1441: ACM.

[66] S.-T. Wu, Y. Li, Y. Xu, B. Pham, and P. Chen, "Automatic pattern-taxonomy extraction for web mining," in *Web Intelligence, 2004. WI 2004. Proceedings. IEEE/WIC/ACM International Conference on*, 2004, pp. 242-248: IEEE.

[67] R. D. García, S. Schmidt, C. Rensing, and R. Steinmetz, "Automatic taxonomy extraction in different languages using wikipedia and minimal language-specific information," in *Computational Linguistics and Intelligent Text Processing*: Springer, 2012, pp. 42-53.

[68] K.-H. Nguyen and C.-Y. Ock, "Using wiktionary to improve lexical disambiguation in multiple languages," in *Computational Linguistics and Intelligent Text Processing*: Springer, 2012, pp. 238-248.

[69] K. Saleem and Z. Bellahsene, "Automatic extraction of structurally coherent mini-taxonomies," in *Conceptual Modeling-ER 2008*: Springer, 2008, pp. 341-354.

[70] M. Makrehchi, "Taxonomy-based Document Clustering," *JDIM,* vol. 9, no. 2, pp. 79-86, 2011.

[71] M. Makrehchi, "Query-relevant document representation for text clustering," in *Digital Information Management (ICDIM), 2010 Fifth International Conference on*, 2010, pp. 132-138: IEEE.

[72] S. Lai, K. S. Leung, and Y. Leung, "SUNNYNLP at SemEval-2018 Task 10: A Support-Vector-Machine-Based Method for Detecting Semantic Difference using Taxonomy and Word Embedding Features," in *Proceedings of The 12th International Workshop on Semantic Evaluation*, 2018, pp. 741-746.

[73] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society,* vol. 7, no. 1, pp. 48-50, 1956.

[74] N. Patel and D. K. Patel, "A Survey on: Enhancement of Minimum Spanning Tree," *Journal of Research and Application,* vol. 5, no. 1, pp. 06-10, 2015.

[75] C. Canada. (2018, May 2018). *Sharcnet Graham Cluster*. Available: https://www.sharcnet.ca/help/index.php/Graham

[76] M. K. Pakhira, "A linear time-complexity k-means algorithm using cluster shifting," in *Computational Intelligence and Communication Networks (CICN), 2014 International Conference on*, 2014, pp. 1047-1051: IEEE.

[77] G. E. Hinton and R. R. Salakhutdinov, "Replicated softmax: an undirected topic model," in *Advances in neural information processing systems*, 2009, pp. 1607-1614.

[78] P. V. Gehler, A. D. Holub, and M. Welling, "The rate adapting poisson model for information retrieval and object recognition," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 337-344: ACM.

[79] IMDB. (2011, January 2018). *IMDB movie Review*. Available: https://data.world/crowdflower/hate-speech-identification

[80] G. Mesnil, T. Mikolov, M. A. Ranzato, and Y. Bengio, "Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews," *arXiv preprint arXiv:1412.5335,* 2014.

[81] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," *arXiv preprint arXiv:1703.04009,* 2017.

[82] T. G. Almeida, B. À. Souza, F. G. Nakamura, and E. F. Nakamura, "Detecting Hate, Offensive, and Regular Speech in Short Comments," in *Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web*, 2017, pp. 225-228: ACM.

[83] K. Lang. (2008, January 2018). *20 Newsgroups*. Available: http://qwone.com/~jason/20Newsgroups/

[84] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent Convolutional Neural Networks for Text Classification," in *AAAI*, 2015, vol. 333, pp. 2267-2273.

[85] Reuters. (1999, January 2018). *Reuters-21578*. Available: http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html

[86] J. Nam, J. Kim, E. L. Mencía, I. Gurevych, and J. Fürnkranz, "Large-scale multi-label text classification—revisiting neural networks," in *Joint european conference on machine learning and knowledge discovery in databases*, 2014, pp. 437-452: Springer.

[87] ComeToMyHead. (2004, January 2018). *AG's corpus of news articles*. Available: https://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

[88] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for text classification," *arXiv preprint arXiv:1606.01781,* 2016.

[89] P. S. S. S. Bhumika and P. A. Nayyar, "A review paper on algorithms used for text classification," *International Journal of Application or Innovation in Engineering & Management,* vol. 2, no. 3, pp. 90-99, 2013.

[90] V. Korde and C. N. Mahender, "Text classification and classifiers: A survey," *International Journal of Artificial Intelligence & Applications,* vol. 3, no. 2, p. 85, 2012.

[91] S. B. Kommina, P. Sandeep, and V. S. Raj, "Mining Social Media Data to Identify the Sentiments of," 2017.

[92] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery,* vol. 2, no. 2, pp. 121-167, 1998.

[93] K. P. Bennett and C. Campbell, "Support vector machines: hype or hallelujah?," *Acm Sigkdd Explorations Newsletter,* vol. 2, no. 2, pp. 1-13, 2000.

[94] A. K. Jain, R. P. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Transactions on pattern analysis and machine intelligence,* vol. 22, no. 1, pp. 4-37, 2000.

[95] K. S. Divya, P. Bhargavi, and S. Jyothi, "Machine Learning Algorithms in Big data Analytics," 2018.

[96] A. Khan, B. Baharudin, and L. H. Lee, "Khairullah khan,(2010)“A Review of Machine Learning Algorithms for Text-Documents Classification, journal of advances in information technology, vol. 1, no. 1," ed: February, 2010.

[97] R. Greiner and J. Schaffer, "AIxploratorium-decision trees," *Department of Computing Science, University of Alberta, Edmonton, AB T6G 2H1, Canada.,* 2001.

[98] S. Ali, M. Khan, and A. Anjum, "Time Frequency Feature Extraction Scheme based on MUAP for classification of Neuromuscular Disorders using EMG signals," *International Journal on Recent and Innovation Trends in Computing and Communication,* vol. 5, no. 7, pp. 249–257-249–257, 2017.

[99] V. Tam, A. Santoso, and R. Setiono, "A comparative study of centroid-based, neighborhood-based and statistical approaches for effective document categorization," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, 2002, vol. 4, pp. 235-238: IEEE.

[100] H. Schütze, "Dimensions of meaning," in *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, 1992, pp. 787-796: IEEE Computer Society Press.

[101] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160-167: ACM.

[102] A. Moreno and T. Redondo, "Text analytics: the convergence of big data and artificial intelligence," *IJIMAI,* vol. 3, no. 6, pp. 57-64, 2016.

[103] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in bioinformatics,* 2017.

[104] K. M. Gultepe E, Makrehchi M., "Latent Semantic Analysis Boosted Convolutional Neural Networks for Document Classification," presented at the The 5th International Conference on Behavioral, Economic, and Socio-Cultural Computing (BESC), National Univerisity of Kaohsoung, Taiwan, 2018.

[105] G. Litjens *et al.*, "A survey on deep learning in medical image analysis," *Medical image analysis,* vol. 42, pp. 60-88, 2017.

[106] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks,* vol. 5, no. 2, pp. 157-166, 1994.

[107] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[108] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078,* 2014.

[109] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, and M. Nielsen, "Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network," in *International conference on medical image computing and computer-assisted intervention*, 2013, pp. 246-253: Springer.

[110] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research,* vol. 12, no. Aug, pp. 2493-2537, 2011.

[111] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104-3112.

[112] A. Yenter and A. Verma, "Deep CNN-LSTM with combined kernels from multiple branches for IMDb review sentiment analysis," in *Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), 2017 IEEE 8th Annual*, 2017, pp. 540-546: IEEE.

[113] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882,* 2014.

[114] F.-F. Li, A. Karpathy, and J. Johnson, "Cs231n: Convolutional neural networks for visual recognition," *University Lecture,* 2015.

[115] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," *arXiv preprint arXiv:1412.1058,* 2014.

[116] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675-678: ACM.

[117] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv preprint arXiv:1510.03820,* 2015.

[118] T. Schnabel, I. Labutov, D. Mimno, and T. Joachims, "Evaluation methods for unsupervised word embeddings," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 298-307.

[119] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 233-240: ACM.

[120] Y. Sasaki, "The truth of the F-measure," *Teach Tutor mater,* vol. 1, no. 5, pp. 1-5, 2007.

[121] C. Van Rijsbergen, "Information retrieval. dept. of computer science, university of glasgow," *URL: citeseer. ist. psu. edu/vanrijsbergen79information. html,* vol. 14, 1979.