Supervised Term-Category Feature Weighting for Improved Text Classification

Joseph Attieh and Joe Tekli*
Electrical and Computer Engineering Department
Lebanese American University (LAU), 36 Byblos, Lebanon

Abstract. Text classification is a central task in Natural Language Processing (NLP) that aims at categorizing text documents into predefined classes or categories. It requires appropriate features to describe the contents and meaning of text documents, and map them with their target categories. Existing text feature representations rely on a weighted representation of the document terms. Hence, choosing a suitable method for term weighting is of major importance and can help increase the effectiveness of the classification task. In this study, we provide a novel text classification framework for Category-based Feature Engineering titled CFE. It consists of a supervised weighting scheme defined based on a variant of the TF-ICF (Term Frequency-Inverse Category Frequency) model, embedded into three new lean classification approaches: i) IterativeAdditive (flat), ii) GradientDescentANN (1-layered), and iii) FeedForwardANN (2-layered). The IterativeAdditive approach augments each document representation with a set of synthetic features inferred from TF-ICF category representations. It builds a term-category TF-ICF matrix using an iterative and additive algorithm that produces category vector representations and updates until reaching convergence. GradientDescentANN replaces the iterative additive process mentioned previously by computing the term-category matrix using a gradient descent ANN model. Training the ANN using the gradient descent algorithm allows updating the term-category matrix until reaching convergence. FeedForwardANN uses a feed-forward ANN model to transform document representations into the category vector space. The transformed document vectors are then compared with the target category vectors, and are associated with the most similar categories. We have implemented CFE including its three classification approaches, and we have conducted a large battery of tests to evaluate their performance. Experimental results on five benchmark datasets show that our lean approaches mostly improve text classification accuracy while requiring significantly less computation time compared with with their deep model alternatives.

Keywords. Text Classification; Document and Text Processing; Feature Engineering; Supervised Term Weighting; Inverse Category Frequency; TF-IDF; Text Representation.

1. Introduction

With the rapid growth of online information, text classification (also known as text categorization or text labelling) has become a central task in Natural Language Processing (NLP) [1, 2], and has been widely studied in many application domains ranging over information retrieval (e.g., categorizing customer complaints represented as support tickets [3, 4]), information filtering (e.g., detecting and filtering spam emails to improve user experience [5, 6]), and sentiment analysis (e.g., classifying target texts into different polarities, e.g., *positive*, *negative*, or affective categories, e.g., *happy*, *angry*, *sad* [7, 8]). Text classification for sentiment analysis has been heavily used by content recommendation and e-commerce companies like Netflix and Amazon to evaluate and adapt their content and product recommendations accordingly [9, 10].

Text classification consists of two main phases: i) feature representation phase, and ii) classification phase. State-of-the-art text feature representations mainly rely on a weighted representation of the terms in the target documents. The underlying idea is that terms that are more important in describing a given document are assigned a higher weight. The weighted document representations are then run through a trained classifier to categorize the documents against a set of target classes or categories. Hence, choosing a suitable method for term weighting is of major importance and can have a big impact on the effectiveness of the classification task. Here, we distinguish between two kinds of weighting schemes: i) unsupervised, where the document representations rely solely on the distribution of the terms across the input documents, and ii) supervised, where document representations are augmented with knowledge regarding the target categories.

* Corresponding author. Tel.: +961-9-547-262; E-mail address: joe.tekli@lau.edu.lb. Joe Tekli is also an adjunct researcher with the SPIDER research team, LIUPPA Laboratory, University of Pay and Pays Adour (UPPA), 64600, Anglet, Aquitaine, France.

In this study, we introduce a new text classification framework for Category-based Feature Engineering titled CFE, which aims to improve classification quality by integrating term-category relationships in document and category representations. Our solution consists of a supervised weighting scheme based on a variant of the TF-ICF (Term Frequency-Inverse Category Frequency) model [11]. Different from existing approaches which are designed for document representation, e.g., [12-14], we adapt TF-ICF to produce weighted representations for the target categories. We then embed the new weighting scheme in three novel text classification approaches: i) the main IterativeAdditive approach, and two neural variants: ii) GradientDescentANN, and iii) FeedForwardANN. The IterativeAdditive approach augments each document representation with a set of synthetic features inferred from the TF-ICF category representations. It builds a term-category TF-ICF matrix using an iterative and additive algorithm that produces category vector representations and updates until reaching convergence. GradientDescentANN replaces the iterative additive process mentioned previously by computing the term-category matrix using a gradient descent ANN model. Training the ANN using the gradient descent algorithm allows updating the term-category matrix until reaching convergence. FeedForwardANN uses a feed-forward ANN model to transform the document representations into the category vector space. The transformed document vectors are then compared with the target category vectors using cosine similarity in order to associate them with their most similar categories.

Compared with the existing literature in text classification (cf. Section 2), the main contributions of this study are summarized as follow: i) to our knowledge, this is the first proposal to use a variation of the TF-ICF weighting scheme for representing the target text categories, while existing solution are designed for input document representation rather than target category representation; ii) we introduce a new set of features inferred from the proposed TF-ICF weighting scheme including both document and category vector representations, to better distinguish the categories during classification; and iii) we introduce three new classification models providing lean architectures consisting of flat (i.e., IterativeAdditive), 1-layered (i.e., GradientDescentANN), and 2-layered (i.e., FeedForwardANN) structures compared with their more complex deep learning and deep attention model alternatives. Experimental results on five benchmark datasets show that our classifiers mostly improve text classification accuracy, while requiring significantly less model parameters and computation time compared with with their their deep learning and deep attention alternatives.

The remainder of the paper is organized as follows. Section 2 briefly reviews the related works. Section 3 introduces our supervised TF-ICF weighting scheme. Sections 4 and 5 describe our CFE framework and its classification approaches. Section 6 presents the complexity analysis. Section 7 describes our experimental evaluation, before concluding in Section 8.

2. Related Works

This section briefly reviews text classification, covering feature representation and classification techniques.

2.1 Feature Representation

Text documents usually represent unstructured data sets. However, these unstructured text sequences must be converted into a structured feature space to be processed by the classifier model. First, the data is cleaned to omit unnecessary characters and terms by performing tokenization, removal of stop words, removal of capitalization and punctuation, and stemming and/or lemmatization. The main goal of these data cleaning steps is to normalize the tokens (different forms of the same word are mapped to the same representation) and filter-out tokens with little significance to the classification task. Consequently, feature extraction techniques are applied to produce suitable representations of the text documents. In this context, state-of-the-art text features mainly rely on a weighted representation of the terms in the target documents, e.g., [13, 15]. The underlying

idea is that terms that are more important in describing a given document are assigned a higher weight. Here, we distinguish between two kinds of weighting schemes: i) unsupervised and ii) supervised.

2.1.1 Unsupervised Term Weighting

Unsupervised term weighing approaches rely on the distribution of terms across their containing documents and the document collection. The standard TF-IDF (Term Frequency – Inverse Document Frequency) approach (and its variants) of the Vector Space Model (VSM) [16, 17] are usually used as typical unsupervised weighting schemes. Documents are indexed in a vector space which dimensions represent, each, a distinct indexing unit t_i. An indexing unit usually stands for a single term or phrase. The coordinate of a given document d_k on dimension t_i, is noted $w_{dk}(t_i)$ and stands for the weight of t_i in document d_k within a document collection. $w_{dk}(t_i)$ is computed using a score of the TF-IDF family, taking into consideration both document and collection statistics. TF represents the number of times a term t_i occurs in a document d_k, e.g., $TF = tf(t_i , d_k)$. The underlying idea with TF is that the importance of t_i in describing d_k increases with the frequent use of t_i in d_k. IDF represents the fraction of documents in the corpus containing term t_i, e.g., IDF = $log(N/df(t_i, d_k))$ where N is the number of documents in the corpus, and $df(ti, dk)$ is the number of documents containing t_i [18, 19]. The underlying idea with IDF is that the importance of t_i in describing d_k decreases with the frequent use of t_i in the corpus [18, 19]. In a recent study [4], the authors compare the performance of TF-IDF and linguistic features-based text representations applied on different supersived classifiers. Results show that even simple algorithms like k-neatrest neighbor (KNN) and decision trees can deliver high-quality prediction when using appropriate features.

The main drawback of using unsupervised term weighting approaches is that they only focus on the term distributions within the source documents and the document collection, without considering any information about the relationship or membership of the source terms/documents with the target categories or classes.

2.1.2 Supervised Term Weighting

Supervised term weighing approaches aim to augment source document features with knowledge regarding document categories using statistical information from the text documents belonging to these categories. Various supervised term-weighting schemes have suggested to replace the IDF factor of TF-IDF, including schemes like chi-squared ($\chi 2$) [20], information gain (IR) [20], and odd ratio (OR) [21]. More recently, the authors in [12] introduced ICF (Inverse Category Frequency) highlighting the importance of terms in describing target categories, e.g., [13, 22], and allowing each document to have a different representation based on its associated category. This representation is computed using term-category statistics denoted α, β, δ, and θ [15]. Parameter α denotes the number of documents that belong to category c_j on the condition that term t_i occurs at least once in each document (i.e., $|\{\forall d_k \in D, (d_k \in c_j) \wedge (t_i \in d_k)\}|$). β denotes the number of documents that belong to category c_j such that term t_i does not occur in the documents (i.e., $|\{\forall d \in D, (d_k \in c_j) \wedge (t_i \notin d_k)\}|$). δ denotes the number of documents that do not belong to category c_j whereas term t_i occurs at least once in each document (i.e., $|\{\forall d_k \in D, (d_k \notin c_j) \wedge (t_i \in d_k)\}|$). θ denotes the number of documents that do not belong to category c_j whereas term t_i does not occur in the documents (i.e., $|\{\forall d_k \in D, (d_k \notin c_j) \wedge (t_i \notin d_k)\}|$). The total number of input (training) documents N comes down to the sum of four elements: $N = \alpha+\beta+\delta+\theta$.

Experimental results in [13, 14, 22] highlight the quality of ICF weighting schemes compared with existing supervised and unsupervised weighting approaches. In this study, we adopt ICF weighting and introduce a supervised scheme based on a variant of the TF-ICF model.

2.2 Classification Techniques

Following the feature representation phase, text classification consists in performing the classification task. The weighted document representations obtained from the feature representation phase are used to train and execute a text classifier in order to categorize the input documents against a set of target categories. Text classifiers use machine learning algorithms adapted to deal with textual features and weighting schemes. They can be organized in two main categories: i) non-parametric, and ii) deep learning. *Non-parametric* algorithms perform classification based on the data contents, without making preliminary assumptions or adding constraints about the form of the classification function. This contrasts with parametric learners like Artificial Neural Networks (ANNs), which mapping function needs to comply with a fixed set of parameters (e.g., number of layers, number of cells). Non-parametric learners include Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree (DT), and the Rocchio classifier, which are often used as a baseline for text classification tasks [1, 23]. *Deep learning* algorithms are parametric learners which make use of artificial neural structures to learn the mappings between input and output patterns. They usually require a set of input parameters to perform the learning process (including number of layers, number of cells per layer, network connectivity, activation functions, and activation thresholds, among others). Typical deep learners include ANNs, Recurrent Neural Networks (RNNs), Long Short-Term Memory models (LSTM), and Convolutional Neural Networks (CNNs) [1]. They have been increasingly investigated for text classification in the past years, surpassing the quality of their non-parametric counterparts [24, 25].

Note that most of the above algorithms use information about the target categories only in the classification phase, totally ignoring their role in the document representation phase. Therefore, recent approaches propose to replace the one-hot vectors representing the target categories by an embedding vector which integrates information about the term/document/category relationships, e.g., [26-28]. In [26], the authors introduce LEAM (Label Embedding Attentive Model) by formulating the text classification problem as a category-term joint embedding problem in which the representation of the category labels is learned in the same space as the terms using an attention mechanism derived from text-category compatibility. The attention is learned on a training set of categorized samples to ensure that relevant terms in a certain text sequence have a higher weight than irrelevant ones. This is done by generating the text sequence representation using a weighted average of the term embeddings, where the weights correspond to the category-based attention scores. In a similar study, the authors in [27] introduce a vector matching solution: they use an input encoder to generate a semantic embedding for every textual input, and a category label encoder to generate a semantic embedding for every target category. Then, the embedding vector of the input text is fed with each category vector into a matcher that computes a matching score used for classification. In [28], the authors first compute a category-based text representation from both input term and target category label embeddings. After that, a CNN is used to compute the weights of the terms from the convolution operation of both the category-based text representation and the term-based text representation (this is similar to the attention score obtained by the attention mechanism from [26]). Then, fully connected softmax layers are used to perform the classification task. In [25], the authors introduce a Text Graph Convolutional Network (Text GCN), which embeds the document corpus into a single graph, where nodes represent documents and terms, and edges represent document-word and word-word weights. TextGCN is first trained on the graph, and is then used to infer the categories of input documents. In other terms, it jointly learns the embeddings for both words and documents, as supervised by the known category labels for the training documents. In [24], the authors introduce another graph-based approach, GraphStar, which adds a virtual "star" node to propagate global information to all nodes and learn better representations by introducing topological modifications of the original graph. In [29], the authors suggest that most existing text classification and embedding solutions suffer from partial semantic loss, since they ignore the interaction between terms and sentences in the source text when generating the category-augmented text representations. In an attempt to solve the problem, they introduce LAHAN (Label-Attentive Hierarchical Attention Network) which generates

category-attentive embeddings using joint features captured from text and category labels. The authors show that their model extracts better text representations using a hierarchical architecture integrating category information at both term and sentence levels.

2.3 Discussion

Traditional text classification methods use information about the target categories only in the classification phase, and mainly ignore their role in the document representation phase, e.g., [1, 23]. Therefore, recent approaches have proposed supervised term weighting solutions which consider information about the relationships of input texts with the target categories, using ICF (Inverse Category Frequency) weighting schemes and category label embeddings to augment and transform input text vectors accordingly, e.g., [12, 15, 22]. They have been shown to produce better results compared with their traditional and unsupervised counterparts [13, 28, 29].

In this study, we adopt a supervised weighting scheme and introduce three novel text classification solutions based on a variant of the ICF model. Different from existing approaches, i) we introduce a variation of the TF-ICF weighting scheme to describe the target classes rather than the input documents, ii) we introduce a set of new synthetic features inferred from TF-ICF with their adapted classification models, and iii) we introduce three lean classifiers (i.e., flat, 1 layered, and 2 layered) compared with their (multilayered) deep learning and deep attention model alternatives.

3. Overview of TF-ICF Weighting Scheme

The general architecture of our framework is shown in Figure 1. In addition to representing each document by its own TF-IDF vector, we represent each category by a TF-ICF vector where the dimensions represent distinctive terms and the weight of each dimension reflects the frequency of occurrence of the term in all the documents belonging to the category. We then embed the new weighting scheme in CFE's text classification models to capture the relationships between document terms and target categories.

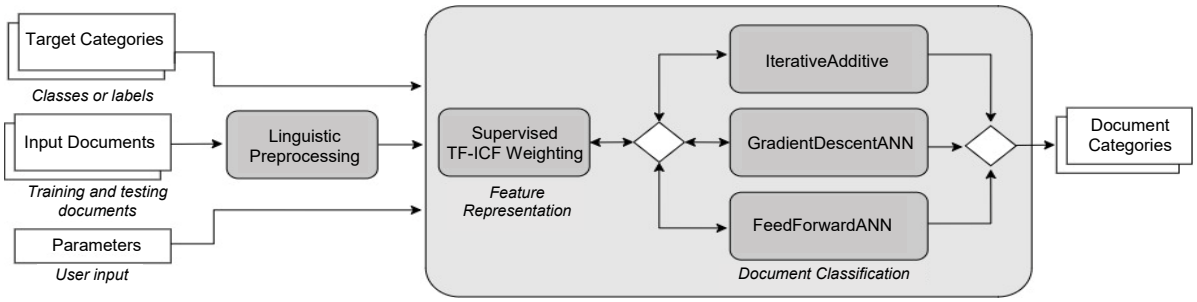


Figure 1. Simplified diagram describing our CFE framework.

More formally, we denote by $D = \{d_1, d_2, \ldots, d_{|D|}\}$ the set of training documents, $T = \{t_1, t_2, \ldots, t_{|T|}\}$ the set of terms that occur in the documents in D (i.e., the vocabulary of D), and $C = \{c_1, c_2, \ldots, c_{|C|}\}$ the set of predefined target categories (i.e., classes or labels). We compute the TF-ICF of a term $t_i \in T$ in category $c_j \in C$ as shown in Table 1. TF represents the frequency of a term inside the set of documents corresponding to the category c_j, where more frequent terms are assigned higher TF scores. ICF represents the fraction of categories that contain term t_i, where rarer terms are assigned higher ICF scores. In other words, the less categories term t_i occurs in,

the more descriptive it will be in distinctively identifying the categories it occurs in, and vice versa (i.e., the more categories term t_i occurs in, the less descriptive it will be in distinguishing its containing categories).

Table 1. Variant of the TF-ICF model adopted in our study.

Variable	Description
TF	$TF_i^j = freq_{c_j}(t_i) = \sum_{d_k \in c_j} freq_{d_p}(t_i)$
CF	$CF(t_i) = freq_{t_i}(C)$
ICF	$ICF_i = \log\left(\frac{\lvert C \rvert + 1}{CF(t_i) + 1}\right) + 1$
TF-ICF	$TF - ICF_i^j = TF_i^j \times ICF_i$

4. CFE Iterative-Additive Classification Approach

We suggest three novel solutions to perform text classification using the TF-ICF weighting scheme described in Section 3. The main approach titled CFE IterativeAdditive aims at augmenting each document representation by a set of synthetic features inferred from the TF-ICF category representations. It builds a term-category TF-ICF matrix using an iterative and additive algorithm that produces category vector representations and updates until reaching convergence. The overall architecture of IterativeAdditive is shown in Figure 2.

First, the term-category relationships are represented through two matrices computed using the *term-category matrix computation* module. The first matrix denoted M is populated with the vector weights of the TF-ICF variant introduced in Section 3. The second matrix denoted M' consists of a dynamic version of the first matrix that is iteratively updated to better fit the document dataset. Aggregate features are extracted using the *feature extraction* module and are appended to the document representations. These augmented document representations are then used to train the *classifier module*. We describe the problem formulation and modules in the following subsections.

4.1 Problem Formulation

We aim to identify a feature vector specific to every document that can help infer the correct category for the document. This feature vector is of size $|C|$ where every feature dimension represents the likelihood of the document being assigned to a category $c_j \in C$. More formally, we aim to find features that satisfy the following:

$$F(d_k) = \{f_{c_1}, ..., f_{c_{|C|}}\} \ / \ \arg\max_{c_j \in C}(F(d_k)) = c_k \quad \textbf{(1)}$$

where $F(d_k)$ represents the set of features representing document d_k, f_{c_i} is the feature representation of category c_i, and C is the set of target categories. We simplify this problem by assuming that the features we seek to infer for every document are an aggregation of the features of every term in the document. This is a reasonable assumption following the bag-of-words model, which can be expressed as an arithmetic addition of the features of every term in the document. Therefore, the problem can be simplified as follows:

$$F(d_k) = \{\sum_{t_i \in d_k} f_{c_1}^{t_i}, ..., \sum_{t_i \in d_k} f_{c_{|C|}}^{t_i}\} \ / \ \arg\max_{c_j \in C}(F(d_k)) = c_k \quad \textbf{(2)}$$

As a result, we aim to identify a feature vector of size $|C|$ for every term in the vocabulary set, i.e., a term-category matrix M of size $|T|\times|C|$ where every value in the matrix corresponds to the TF-ICF weight of a term $t_i \in T$ w.r.t. a category $c_j \in C$. We introduce two term-category matrices: a seed matrix M computed using our TF-ICF weighting scheme described earlier; and a dynamic version of M denoted M', computed based on M using a supervised iterative and additive process. The matrix computation process consists of: i) linguistic preprocessing, ii) matrix initialization, iii) matrix evaluation, iv) category inference, and v) matrix update.

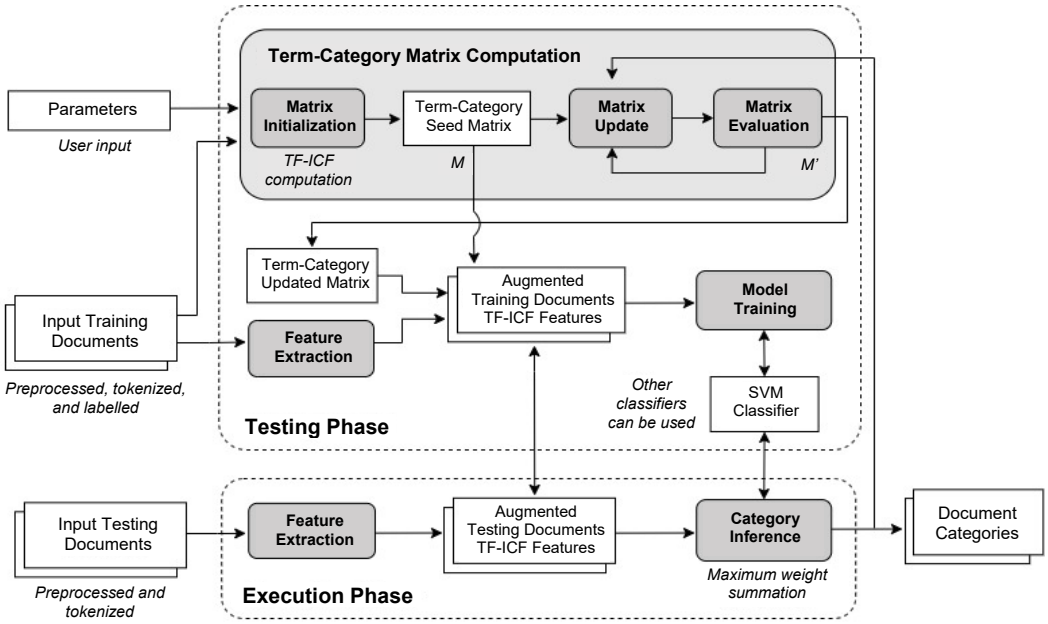


Figure 2. Simplified diagram describing our CFE IterativeAdditive approach.

4.2 Linguistic Preprocessing

Linguistic preprocessing cleans the documents' textual contents by performing tokenization, removal of stop words, removal of capitalization and punctuation, and stemming. It aims to identify and normalize the documents' representative terms, forming the initial document vocabulary. The latter are also associated with the categories assigned to the training documents.

Table 2. Sample training documents used in our running example

a. Training documents and their category labels

Documents D		Categories C	
d_1	"*He ate a green apple and then cooked an apple pie*"	c_1	*Food*
d_2	"*He wrote it on his green book*"	c_2	*Study*
d_3	"*She cooked one apple pie and then another apple pie*"	c_1	*Food*
d_4	"*The grass was green*"	c_3	*Nature*
d_5	"*The apple cook book*"	c_2	*Study*

b. Preprocessed documents and their term-frequency vectors

Preprocessed Documents D	Term-document vector dimensions							
	t_1 *eat*	t_2 *green*	t_3 *apple*	t_4 *write*	t_5 *book*	t_6 *cook*	t_7 *pie*	t_8 *grass*
<"*eat*", "*green*", "*apple*", "*cook*", "*apple*", "*pie*">	1	1	2	0	0	1	1	0
<"*write*", "*green*", "*book*">	0	1	0	1	1	0	0	0
<"*cook*", "*apple*" ,"*pie*", "*apple*" ,"*pie*">	0	0	2	0	0	1	2	0
<"*grass*", "*green*">	0	1	0	0	0	0	0	1
<"*apple*", "*cook*", "*book*">	0	0	1	0	1	1	0	0

Running example: Consider a set of training documents $D = \{d_1, d_2, d_3, d_4, d_5\}$ and their desired categories $C = \{c_1, c_2, c_3\}$ shown in Table 2.a. The preprocessed documents and their TF vector representations are shown in Table 2.b. The vector dimensions represent the set of terms $T = \{t_1, \ldots, t_8\}$ which will be utilized to compute the seed TF-ICF matrix *M*.

4.3 Matrix Initialization

Matrix initialization computes the TF-ICF seed matrix *M* based on the training documents' TF vector representations. Using the documents' desired categories, *M* provides an initial description of the impact of each term $t_i \in T$ on every category in $c_j \in C$ (cf. running example in Table 3).

Table 3. TF-ICF seed matrix *M* computed based on our running example from Table 2.

a. Term-Frequency (TF) weight matrix.

Categories C		Term-category vector dimensions							
		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
		eat	*green*	*apple*	*write*	*book*	*cook*	*pie*	*grass*
c_1	*Food*	1	1	4	0	0	2	3	0
c_2	*Study*	0	1	1	1	2	1	0	0
c_3	*Nature*	0	1	0	0	0	0	0	1

b. Inverse Category Frequency (ICF) weight vector.

Category collection	Term-category vector dimensions							
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
	eat	*green*	*apple*	*write*	*book*	*cook*	*pie*	*grass*
C	1.7	1	1.29	1.7	1.7	1.29	1.7	1.7

c. Combined TF-ICF seed matrix *M*.

Categories C		Term-category vector dimensions							
		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
		eat	*green*	*apple*	*write*	*book*	*cook*	*pie*	*grass*
c_1	*Food*	1.7	1	5.16	0	0	2.58	5.1	0
c_2	*Study*	0	1	1.29	1.7	3.4	1.29	0	0
c_3	*Nature*	0	1	0	0	0	0	0	1.7

4.4 Matrix Evaluation

After producing the TF-ICF seed matrix *M*, the quality of its weights is evaluated to decide whether a *matrix update* phase is required to improve the category inference process. The *matrix evaluation* algorithm is provided in Figure 3. It starts by splitting the set of training documents *D* into i) a reference subset (70%), and ii) a validation subset (30%, cf. Figure 3, lines 1-2)[2]. The reference subset is run through the *category inference* component (line 3, cf. Section 4.5) and is subsequently processed for *matrix update* (line 4, cf. Section 4.6). The validation subset is used to evaluate the quality of the updated matrix and decide whether the *matrix evaluation* process should be repeated or not, until reaching convergence. Quality is evaluated using category inference accuracy, by comparing the categories inferred for every document in the evaluation subset with the expected document categories (lines 4-7). Convergence is achieved when accuracy reaches a certain (user or system

[2] Here, we adopt a random 70/30 split (yet other forms of cross-validation splitting can be used, like *k*-fold or Monte-Carlo).

specified) threshold (lines 8-9). The iterative process ends when reaching convergence, or when reaching a maximum number of (user or system specified) iterations.

```
Algorithm Matrix Evaluation
Input: D, C_Desired      // Set of training documents and their desired categories
       ratio             // Separate validation ratio
       M'                // Updated TF-ICF matrix, initially equal to Seed TF-ICF matrix M, i.e, M' = M
       NbOcc             // Number of occurrences, initially: NbOcc = 0
       Max_Nb_Iterations // Maximum number of iterations
       Thresh_Conv       // Convergence threshold
Output: M'               // Updated TF-ICF matrix
Begin
   Create (D1, C_D1_Desired)= subset((D, C_Desired), ratio)   // Split D to form a reference subset (we use ratio = 70%)   1
   Create (D2, C_D2_Desired ) = subset((D, C_Desired), 1-ratio)   // and a validation subset (1-ratio = 30%)   2

   C_D1_Inferred = categoryInference(D1,M')   // Run category inference process on reference subset D1   3
   M' = matrixUpdate(D1, C_D1_Desired , M')   // Launch matrix update process on M   4
   C_D2_Inferred = categoryInference(D2, M')   // Run category inference process on evaluation subset D2   5
   Nb_Iterations ++   // Increment number of iterations by 1   6

   Acc = evaluateAccuracy(C_D2_Desired, C_D2_Inferred)   // Evaluate accuracy of category inference on D2   7

   while ( Acc < Thresh_Conv AND Nb_Iterations < Max_Nb_Iterations)   8
      matrixEvaluation((D, C_Desired), ratio, M', Nb_Iterations, Max_Nb_Iterations, Thresh_Conv)   // Recursive call   9

   Return M'   // Return updated matrix M'   10
End
```

Figure 3. Pseudo-code for *matrix evaluation* process.

4.5 Category Inference

Category inference allows identifying the category that best describes a given document. It is utilized twice in our approach: i) as part of the training phase to predict the categories of training documents (leading to their update and evaluation, cf. section 4.4[3]), and ii) as part of the execution phase to predict the categories of new (uncategorized) documents (cf. Figure 2). We adopt the single-category classification model in our approach where each document d_k is assigned a unique category $c_j \in C$. Following our TF-ICF weighting scheme, the weights inside matrix M reflect the likelihood of occurrence of each term in each category. Hence, performing category inference comes down to finding the category that is most described by the terms inside the document. This is achieved by computing the summation of the TF-ICF weights of the terms occurring in the document, for each category in C. The category with the maximum summation score is assigned to the document:

$$Category_{Inference}(d_k, C) = arg\ max_{c_j \in C} \left(\sum_{t_i \in d_k} \left(TF - ICF_i^j \right) \right) \quad \textbf{(3)}$$

where d_k is the document being processed for category inference, and c_j is the category which is assigned to d_k as a result of the category inference process.

[3] When applied in the training phase, the result of the *category inference* process is evaluated through the *matrix evaluation* component (described in Section 4.4) and helps guide the *matrix update* process (described in Section 4.6).

Running example: Consider training document d_5 = *"The apple cook book"* and its desired category c_2 = *Study*. Using seed matrix M from our running example (cf. Table 3), the initial *category inference* process applied on d_5 produces category c_1 = *Food* (cf. inference computation in Table 4). This assignment is incorrect since the document is supposed to be assigned with its desired category c_2 = *Study*. This discrepancy in category assignment is detected through the *matrix evaluation* component (described in the previous subsection), and is subsequently updated through the *matrix update* component (described in the following subsection).

Table 4. Category inference computation based on the TF-ICF seed matrix *M* from Table 3.c.

Categories *C*		Term-Category vector dimensions								Σ
		t_1 *eat*	t_2 *green*	t_3 *apple*	t_4 *write*	t_5 *book*	t_6 *cook*	t_7 *Pie*	t_8 *grass*	
c_1	*Food*	1.7	1	5.16	0	0	2.58	5.1	0	**7.74**
c_2	*Study*	0	1	1.29	1.7	3.4	1.29	0	0	5.98
c_3	*Nature*	0	1	0	0	0	0	0	1.7	0

Algorithm *Matrix Update*

Input: D // Set documents
C // Set of categories
M′ // Updated TF-ICF matrix

Output: M′ // Updated TF-ICF matrix

```
Begin
    For each term in tᵢ ∈ D                                          1
        For each category cⱼ ∈ C                                     2
            If tᵢ ∉ M′.cⱼ                                            3
                ICF(tᵢ) = Δ_ICF(tᵢ)   // Update ICF, cf. formula (4)  4
            TF(tᵢ) = Δ_TF(tᵢ)         // Update TF, cf. formula (4)  5
    Return M′                         // Return updated matrix M′    6
End
```

Figure 4. Pseudo-code for *matrix update* process.

4.6 Matrix Update

Matrix update aims at enhancing the weights of certain terms in the TF-ICF matrix to improve their descriptiveness w.r.t. certain categories. The *matrix update* algorithm is provided in Figure 4. It takes as input the TF-ICF seed matrix *M* and produces an updated version of it denoted *M'* where the weights of the desired categories are adjusted in the target documents. Weight adjustment is performed as follows: i) we increase the TF weight by adding term occurrences for terms present in the desired category (line 5), and ii) we decrease the ICF weight for terms not present in the category (i.e., TF=0) by subtracting term occurrences for terms not present in the desired category (lines 3-4):

$$\Delta_{TF} = TF + 1 \quad \text{and} \quad \Delta_{ICF} = ICF_i' - ICF_i = \left(\log\left(\frac{|C|+1}{CF_i+1+1}\right)+1\right) - \left(\log\left(\frac{|C|+1}{CF_i+1}\right)+1\right) = -\log\left(\frac{CF_i+2}{CF_i+1}\right) \qquad \textbf{(4)}$$

where $CF_i = Freq_{t_i}(C)$ is the current number of categories in C having a non-zero weight for t_i.

i) Updating TF: For the desired category to be predicted by the *category inference* process, it should maximize the result of formula (3), compared with the other categories. Therefore, the *matrix update* process keeps incrementing TF for every term in the target category until the document is classified correctly. We apply a linear increment =1 unit for every term dimension, as a basic form of TF increment (other forms of logarithmic or sublinear increments can be used).
ii) Updating ICF: For terms which are not present in the desired category, we decrease their ICF weights by applying a simple arithmetic subtraction operation to decrease every weight score by 1 denominator unit, as a basic form of ICF reduction (other forms of logarithmic or sublinear weight reductions can be used).

Table 5. Updated TF-ICF matrix *M'* computed based the seed matrix *M* from Table 3.

a. Updated Term-Frequency (TF) weights.

Categories C		Term-category vector dimensions							
		t_1 *eat*	t_2 *green*	t_3 *apple*	t_4 *write*	t_5 *book*	t_6 *cook*	t_7 *pie*	t_8 *grass*
c_1	*Food*	1	1	4	0	0	2	3	0
c_2	*Study*	0	1	1+**1**	1	2+**1**	1+**1**	0	0
c_3	*Nature*	0	1	0	0	0	0	0	1

b. Updated TF-ICF matrix *M'*.

Categories C		Term-category vector dimensions							
		t_1 *eat*	t_2 *green*	t_3 *apple*	t_4 *write*	t_5 *book*	t_6 *cook*	t_7 *pie*	t_8 *grass*
c_1	*Food*	1.7	1	5.16	0	0	2.58	5.1	0
c_2	*Study*	0	1	**2.58**	1.7	**5.1**	**2.58**	0	0
c_3	*Nature*	0	1	0	0	0	0	0	1.7

Running example: Consider training document d_5 = *"The apple cook book"* and the discrepancy produced between its desired category c_2 = *Study* and its inferred category c_1= *Food*, following the application of the *category inference* process (cf. Section 4.5). The *matrix update* process allows updating the weights of d_5's terms starting from the TF-ICF seed matrix *M*, until the sum of the weights of the terms are maximized for the desired category c_2 = *Study*. This is done by iteratively incrementing the terms in every category by 1. After computing the new TF scores, the TF-ICF seed matrix *M* is updated accordingly, resulting in matrix *M'* as shown in Table 5. Considering document d_5, applying the *category inference* process using updated TF-ICF matrix *M'* produces c_2 = *Nature*, which is the correct category assignment for document d_5 (cf. computation example in Table 6). In this case, the model is successfully trained after one single iteration of the matrix computation process, and there is no need to further proceed with the *matrix update* process.

Table 6. Category inference computation based on the updated TF-ICF matrix *M'* from Table 5.b.

Categories *C*		Term-Category vector dimensions								Σ
		t_1 *eat*	t_2 *green*	t_3 *apple*	t_4 *write*	t_5 *book*	t_6 *cook*	t_7 *pie*	t_8 *grass*	
c_1	*Food*	1.7	1	5.16	0	0	2.58	5.1	0	7.74
c_2	*Study*	0	1	2.58	1.7	35.1	2.58	0	0	**10.26**
c_3	*Nature*	0	1	0	0	0	0	0	1.3	0

4.7 Augmented TF-ICF Document Features for Classification

After computing the TF-ICF seed matrix *M* and generating its updated version *M'*, we produce the augmented TF-ICF document features needed for the classification phase. The overall process is visualized in Figure 5. From both *M* and *M'*, we extract the following aggregate feature vectors for each document:

- The summation of the weights of the terms occurring in the document per category:

$$\sum_{t_i \in d_k} TF\text{-}ICF_i^k \tag{5}$$

- The maximum of the weights of the terms occurring in the document per category:

$$\max_{t_i \in d_k} \left(TF\text{-}ICF_i^k \right) \tag{6}$$

The above features are concatenated to form the new aggregate TF-ICF feature vector for the document, which length is equal to four times the number of categories (since each feature vector has one dimension per category). We extract the aggregate TF-ICF features for all the documents in the training set, where each document is now associated with: i) a traditional TF-IDF vector representation, and ii) the aggregate TF-ICF vector representation described above. Both features are used concurrently to train the classifier model (cf. Figure 5). We currently adopt a Linear Support Vector Machine (SVM) for our classifier model due to its quality in performing text classification (SVM is specifically designed to handle sparse feature vectors, which is the case with high-dimensional text data). Nonetheless, other classifiers can be used following the system designer's preferences (cf. Background in Section 2.4). Once trained, the classifier predicts the category of a new input document based on its traditional TF-IDF and learned TF-ICF feature vectors.

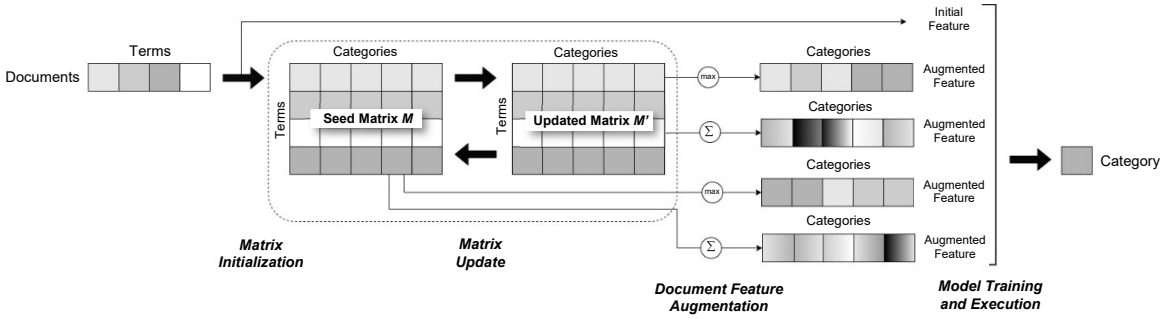


Figure 5. Simplified diagram describing the document features computation, and the features' usage within the classification process.

5. CFE Neural Variants: Gradient Descent and Feed-Forward ANNs

In addition to the main IterativeAdditive approach describe above, we suggest two neural variants to perform text classification using our supervised TF-ICF weighting scheme: GradientDescentANN and FeedForwardANN. We describe them in the following sub-sections.

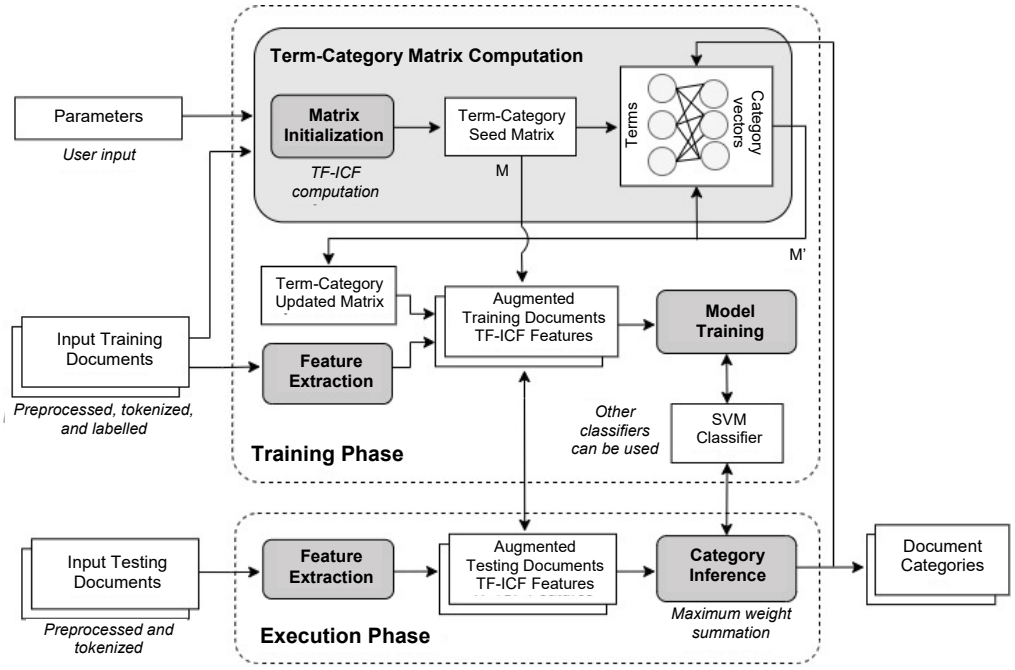


Figure 6. Simplified diagram describing the CFE GradientDescentANN approach.

5.1. CFE Gradient Descent ANN approach

While IterativeAdditive adopts an iterative and additive approach to perform TF-ICF matrix computation, nonetheless, this process can be handled using other optimization solutions such as gradient descent or evolutionary-developmental algorithms, e.g., [30, 31]. To showcase the extensibility of our approach, we introduce GradientDescentANN which computes the term-category weight matrix M using a 1-layer ANN, where the inputs correspond to the terms in the vocabulary, and the outputs correspond to the predefined categories (cf. Figure 6). The ANN has a weight matrix of size $|T|\times|C|$ which represents term-category matrix M. We adopt here a 1-layered ANN as the simplest possible solution to the problem, yet deeper neural structures can be considered. We utilize softmax as the activation function of the final layer (i.e., the only layer in our current 1-layered network), which is suitable with our decision function (i.e., the category with the highest weight in the output determines the category of the document). As a result, training the ANN using the gradient descent algorithm allows updating matrix M-into-M' in order to satisfy our problem formulation.

5.2. CFE Feed-Forward ANN approach

Our second neural variant, titled FeedForwardANN uses a feed-forward ANN to transform input document representations into the category feature space, where document vector representations from the same category are highly correlated and similar to their category vectors. The transformed document vectors are then compared with the target category vectors using cosine similarity for model training and categorization, in order to associate the input documents with their most similar categories (cf. Figure 7).

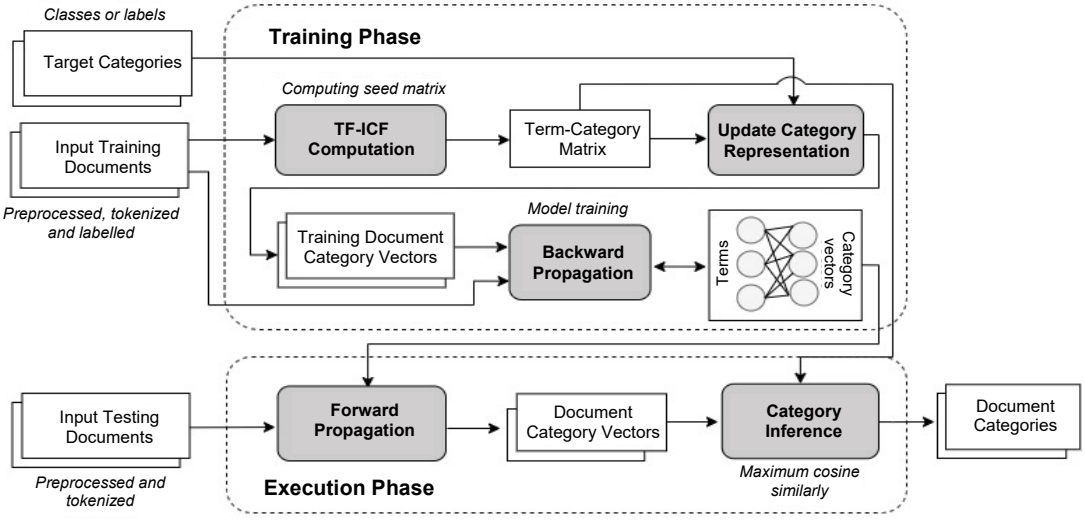


Figure 7. Simplified diagram describing the CFE FeedForwardANN approach.

To produce the transformed category feature space, we approximate the direction shared by the documents within each category by computing the centroid of this category. The centroid is the entry in the TF-ICF matrix corresponding to the category. The mapping from document space to category space is computed as follows:

i) The TF-ICF matrix M is computed through the *matrix initialization* component (similarity to our previous CFE approaches),

ii) Matrix M maps each category to a category vector representation, where documents having the same category reflect similar term distributions. Then, each document and category pair from the training set are mapped to a document and category vector pair, following the category's entry in the TF-ICF matrix. The outcome of this phase corresponds to the training data of the model (cf. Figure 8).

iii) The mapping between the documents and the new category vectors is learned through the classifier model. We adopt a feed-forward ANN with one hidden layer to handle non-linearly separable solutions as the simplest possible solution to the problem (deeper neural structures can be considered). The ANN is trained to convert the input document vector into a vector that is similar to the representative vector corresponding to the category of the document. We adopt the cosine measure to evaluate the similarity/dependence between the document vector and category vector directions (other correlation measures like Pearson coefficient or Dice can be used)[4]. This is achieved through the use of a cosine loss function, allowing to quantify the direction of the output vectors:

$$L_{Cosine}\left(V_{c_j}, V_{d_k}\right) = \frac{\left(V_{c_j} - V_{d_k}\right)^2}{\left|V_{c_j} - V_{d_k}\right|^2} + Cosine\left(V_{c_j}, V_{d_k}\right) \quad \textbf{(7)}$$

[4] Cosine similarity has been shown effective in various classification scenario, especially when dealing with high-dimensional data, e.g., [32, 33], which is the case with document and category text representations.

where V_{d_k} is the document vector produced by the ANN model, and V_{c_j} is the desired category vector. The training process aims at minimizing the cosine loss between V_{d_k} and V_{c_j}.

iv) Cosine similarity is also utilized to perform the document classification task, allowing to identify the document vector that is most similar to the category vector:

$$Category_{Inference}(d_k, C) = arg\ max_{c_j \in C}\left(Sim_{\cos ine}(V_{d_k}, V_{c_j})\right) \quad \textbf{(8)}$$

where d_k is the document being processed for category inference, $c_j \in C$ is the category which is assigned to d_k as a result of the category inference process, V_{d_k} and V_{c_j} are their vectors respectively.

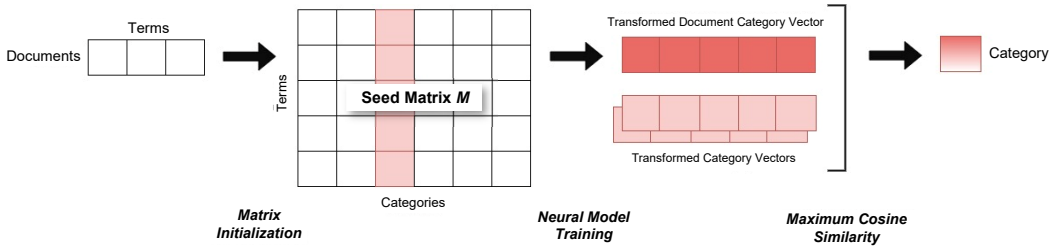


Figure 8. Simplified diagram describing the document category vector feature transformation, and the features' usage within the classification process.

6. Complexity Analysis

The computational complexity of the CFE framework, including its three classification solutions, comes down to O($t \times c$) where t is the number of textual tokens in the vocabulary (i.e., number of distinctive terms forming the document-term vector dimensions, $t = |T|$), and c is the number of target categories (forming the document-category matrix dimentions, i.e., $c = |C|$). This comes down to the complexity of the oprations performed when executing the models. Having a TF-ICF matrix of size $t \times c$, we extract from this matrix one category representation per token in the sentence, and we aggregate them to compute the additional features in the case of IterativeAdditive and GradientDescentANN, before performing category inference (which is linear in the number of categories c). In the case of FeedForwardANN, we map every sentence to a category vector before computing cosine (which is linear in vector size), thus requiring the same computational complexity albeit with additional linear overhead due to cosine similarity computation.

To put things into perspective, Table 7 compares our framework's complexity with existing solutions in the literature, including the number of compositional parameters required by each model. On the one hand, SWEM [46] requires the least number of parameters, followed by our framework and LEAM [26] requiring $t \times c$ and $c \times p$ respectively, where p is the number of dimensions of the sequence representation (i.e., length of the embedding). On the other hand, it is clear that our framework requires the least amount of computational complexity, since the number of target categories is usually much smaller than the size of the embedding vector compared with existing solutions including SWEM, i.e., $c << p$. This is also evident in the time performance results reported in Section 6.5, where our framework is clearly more efficient than existing solutions.

Table 7. Comparing the number of parameters and computational complexity with existing solutions.
t represents the number of tokens in the vocabulary (i.e. *t*=|*T*|), *c* the number of target categories (*c* = |*C*|), *f* the number of filters in the CNN, *s* the size of the CNN filter, *h* as the number of dimensions of hidden units in the LSTM, *n* and *e* respectively represent the number of nodes and the number of edges in both GCN and HyperGAT-dual, and *p* the number of dimensions of the sequence representation (i.e., length of the embedding).

Approach	Parameters	Computational Complexity
CNN-non-static [45]	$s \times f \times p$	$O(s \times f \times p \times t)$
Bi-LSTM [25]	$4h \times (h + p)$	$O(t \times h^2 + t \times h \times p)$
SWEM [46]	$\emptyset$	$O(t \times p)$
TextGCN [25]	$p^2 + n^2$	$O(n \times p^2 + e \times p)$
HyperGAT-dual [49]	$n^2 + 2e^2 + e \times n$	$O(2n^2 \times e + 2e^2 \times n)$
LEAM [26]	$c \times p$	$O(c \times t \times p)$
CFE	$t \times c$	$O(t \times c)$

7. Experimental Evaluation

We have implemented our CFE framework and its three classification solutions to test and evaluate their performance, and compare them with recent alternatives in the literature. Written in Python, our implementation comprises CFE's main modules and components, in addition to a linguistic pre-processing component to perform tokenization, stop word removal, and stemming. The experimental prototype, test data, and test results are available online[5].

In the following, we first describe the experimental data and set-up in Section 6.1. Internal evaluation experiments and results comparing the three CFE classification approaches against each other are described in Section 6.2. External evaluation and comparison results comparing the CFE approaches with existing state-of-the-art solutions are described in Section 6.3. A discussion of the difficulties in classifying similar categories is provided in Section 6.4. In summary, results show the IterativeAdditive and FeedForwardANN approaches outperform GradientDescentANN, and are on a par with and mostly improve text classification accuracy compared with their recent alternatives.

7.1 Experimental Data and Set-up

We utilized multiple benchmark datasets from the text classification literature: i) R8 is a subset of Reuters-21,578[6] including 7,674 documents organized in 8 categories (e.g., earn, acq, trade, crude), ii) R52 is another subset of Reuters-21,578 including 9,100 documents organized in 52 categories (e.g, earn, trade, fuel, coffee), iii) Ohsumed[7] is an extract of the MEDLINE database where every document represents an abstract of a medical paper from the database, and is categorized in one of 23 diseases (e.g., endocrine diseases, eye diseases, and virus diseases), iv) 20 News-Group[8] consists of 18,846 short news post organized into 20 categories (e.g.,

[5] http://sigappfr.acm.org/Projects/CFE/
[6] https://ana.cachopo.org/datasets-for-single-label-text-categorization
[7] http://disi.unitn.it/moschitti/corpora.htm
[8] http://qwone.com/~jason/20Newsgroups/

comp.graphics, sci.med, rec.autos, misc.forsale), and v) AG News[9] is a collection of short news articles collected from more than 2,000 news sources, and organized into 4 categories (e.g., world, sports, business, and sci/Tec). We adopted the standard split between training and testing sets as provided by the datasets. The dataset characteristics are shown in Table 8.

Table 8. Characteristics of experimental datasets used in our study.

Dataset	Training set (# of docs)	Testing set (# of docs)	Avg. size of doc (# of terms)	# of categories
R8	5,485	2,189	106	8
R52	6,532	2,568	113	52
Ohsumed	3,357	4,043	185	23
20NewsGroup	11,314	7,532	318	20
AGNews	120,000	7,600	39	4

We used the following CFE parameter set-up to run our experiments including the above mentioned datasets:

- Main approach – IterativeAdditive: we set the maximum number of iterations of the *matrix update* process to 400, with an early stopping rule when average accuracy reaches 0.98 on the validation set (considering a 75/25% split)[10].
- Second approach – GradientDescentANN: we train the 1-layer gradient descent ANN considering a maximum 100 epochs, with an early stopping rule if the validation accuracy converges and is stable for 10 consecutive epochs. We use softmax as the activation function.
- Third approach – FeedForwardANN: we consider a 2-layered ANN, including one hidden layer made of $\sqrt{2 \times InputLayerSize \times OutputLayerSize} = \sqrt{2 \mid T \mid}$. We empirically set the number of hidden neurons based on best practices in the literature, e.g., [37, 38]. We train the model for 10 epochs, using reLu and softmax as the hidden layer and outer layer activation functions respectively.

All models are trained with a maximum number of features = 5000. Note that optimizing parameter set-up for the different models is outside the scope of this manuscript and will be addressed in a future dedicated study.

6.2. Classification Quality

6.2.1. Comparing the CFE Classification Approaches

This first experiment evaluates the quality of our CFE classification approaches against each other. We include four variants for each approach: i) one trained on our supervised document-category TF-ICF weighting scheme, and three others trained on ii) unsupervised Boolean TF [18, 39], ii) unsupervised TF-IDF [25, 40], and iv) supervised document TF-ICF [13, 14]. Mean accuracy results are provided in Table 9.

---

[9] https://www.kaggle.com/amananandrai/ag-news-classification-dataset

[10] We set the number of iterations to 400 after multiple empirical trials with 50, 100, 200, 400, 500, and 800 iterations. In most experiments, the IterativeAdditive model converged at 400. We conducted similar empirical trials with the GradientDescentANN and FeedForwardANN models, varying the number of epochs between 10, 50, 100, 200, and 300 and selecting the number which minimized network loss and achieved optimal results. Note that parameter set-up can be further improved using dedicated parameter tuning techniques, using linear programming and machine learning solutions to identify the best weights for a given problem class, e.g., [34-36].We report this investigation to a dedicated future study.

Table 9. Mean accuracy results for CFE classification approaches.
Red color refers to the best score, and green color refers to the second best for each dataset.
We run all models 10 times and report mean accuracy and standard deviation results (between parentheses)

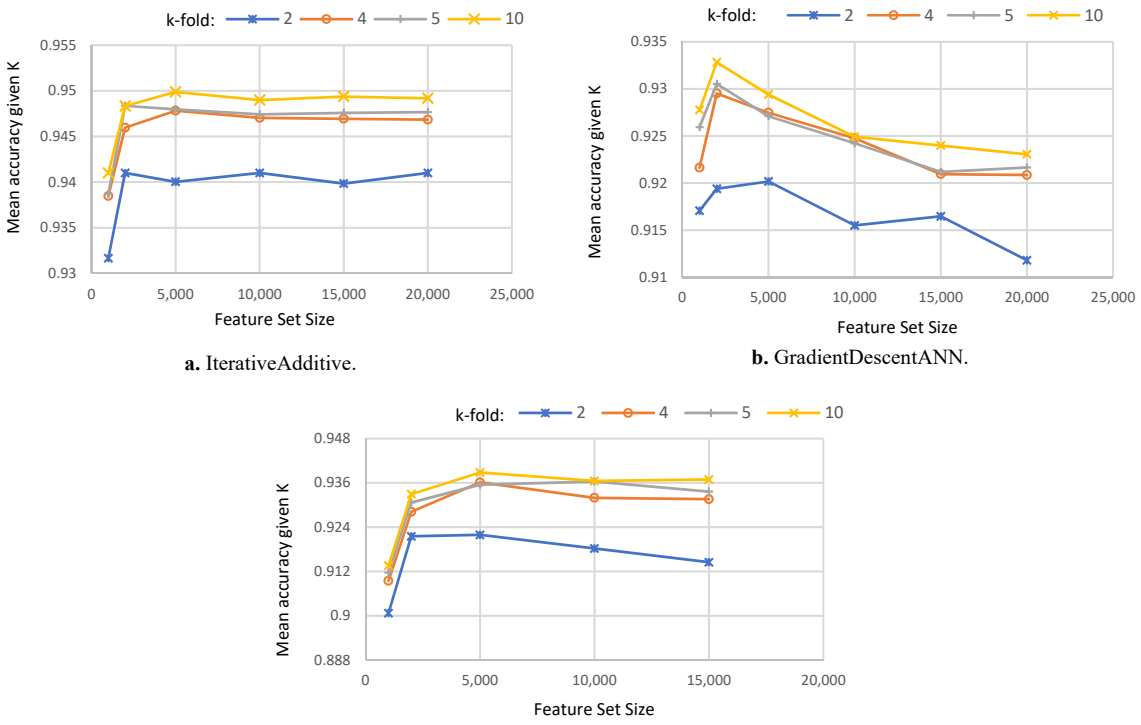
Classification Approach	Weight Scheme	Benchmark Dataset				
		R8	R52	Ohsumed	20NewsGroup	AGNews
IterativeAdditive	Unsupervised Boolean TF [18, 39]	96.21 (0.00)	92.37 (0.00)	56.79 (0.00)	79.07 (0.01)	85.80 (0.11)
GradientDescentANN		96.21 (0.00)	92.37 (0.00)	56.79 (0.00)	79.07 (0.01)	85.80 (0.11)
FeedForwardANN		95.31 (0.30)	92.13 (0.29)	62.10 (0.69)	83.93 (0.14)	89.57 (0.27)
IterativeAdditive	Unsupervised TF-IDF [25, 40]	95.66 (0.01)	93.02 (0.25)	68.65 (0.31)	78.13 (1.23)	88.10 (0.32)
GradientDescentANN		96.2 (0.2)	90.16 (0.47)	50.10 (2.9)	78.11 (0.90)	88.00 (0.06)
FeedForwardANN		96.2 (0.2)	70.16 (0.47)	50.10 (2.9)	78.11 (0.90)	88.00 (0.06)
IterativeAdditive	Supervised Document TF-ICF [13, 14] (original scheme)	96.57 (0.00)	92.7 (0.06)	66.9 (0.00)	79.14 (0.01)	85.61 (0.34)
GradientDescentANN		96.57 (0.00)	92.7 (0.06)	66.9 (0.00)	79.14 (0.01)	85.61 (0.34)
FeedForwardANN		96.22 (0.43)	94.17 (0.21)	67.31 (0.21)	83.49 (0.17)	89.75 (0.11)
IterativeAdditive	Supervised Document-Category TF-ICF (our scheme)	**97.94** (0.02)	**95.13** (0.04)	**68.90** (0.02)	**85.51 (0.04)**	**91.78** (0.02)
GradientDescentANN		96.57 (0.02)	93.85 (0.00)	**68.79** (0.05)	**86.02 (0.05)**	**91.75** (0.01)
FeedForwardANN		**97.71** (0.55)	**96.98** (1.53)	67.51 (0.03)	83.19 (0.35)	91.54 (0.68)

Results highlight the following observations:

- All CFE approaches consistently produce improved results when coined with our supervised TF-ICF scheme, compared with existing weighting models. This highlights the positive impact of our supervised document-category weighting scheme on classification quality, compared with existing unsupervised Boolean TF, unsupervised TF-IDF, and supervised document TF-ICF weighting models.
- IterativeAdditive produces the best results on 2 out of 4 evaluation datasets (i.e., R8, and R52), compared with GradientDescentANN and FeedForwardANN. This shows that our iterative and additive process using SVM with a linear kernel, is on a par with and sometimes surpasses the quality of our ANN-based solutions.
- FeedForwardANN produces the best results on 2 out of 4 evaluation datasets (i.e., 20NewsGroup and AGNews). Also, GradientDescentANN produces the second best result on one of the datasets (i.e., R8). This highlights the potential of our neural approaches compared with their iterative and additive counterpart, and the extensibility of our framework to different kinds of classification models.
- Overall, GradientDescentANN falls behind its two counterparts on all datasets. On the one hand, using a local ANN to substitute the iterative and additive *matrix update* process did not improve (and rather decayed) classification quality. Recall that the neural model in GradientDescentANN is designed to perform feature augmentation only, while maintaining the main SVM classification module. On the other hand, substituting the whole IterativeAdditive architecture with an ANN architecture in FeedForwardANN produced improved results, surpassing the quality of IterativeAdditive on two 2 of 4 evaluation datasets (i.e., 20NewsGroup and AGNews). This highlights the positive impact of using full-fledged neural architectures to improve classification quality.

6.2.2. Varying Dimensionality and Training Data Size

To shed more light on the behavior of our CFE approaches, we analyze the effect of varying: i) the number of dimensions representing the document feature vectors, and ii) the size of the training data. We vary dimensionality by considering the top weighted feature dimensions ranging over: 1,000, 2,000, 5,000, 10,000 and 15,000 dimensions per vector. We also vary the size of the training data by performing k-fold cross validation, for k= 2, 4, 5 and 10. For every model trained with a specific number of folds k, we compute and report average accuracy on the corresponding testing dataset. Results are shown in Figure 9.



a. IterativeAdditive.

b. GradientDescentANN.

c. FeedForwardANN.

Figure 9. Mean accuracy results with varying dimensionality and training data size applied on the R52 dataset (similar results were produced with the other datasets, cf. technical report in [41]).

We highlight the following observations:

- IterativeAdditive and FeedForwardANN approaches maintain good accuracy levels when increasing the number of feature dimensions, while GradientDescentANN suffers from a drop in accuracy levels. This implies that the features generated by IterativeAdditive and FeedForwardANN are better than the ones generated by GradientDescentANN. This can be explained by the fact that the *matrix update* phase in GradientDescentANN updates coefficients through gradient-descent even when it's not needed, resulting in non-optimal aggregate features that might not be good representatives of the categories. This is different from the two aforementioned approaches which only update the coefficients when needed: through the iterative and additive process with IterativeAdditive, and through the neural training process with FeedForwardANN. Training the GradientDescentANN network with a larger number of data or with a higher number of epochs might alleviate the problem. This makes IterativeAdditive and FeedForwardANN more favorable since they both managed to keep a consistent performance with the same parameters regardless of dimensionality.
- Considering all three approaches, results clearly show that classification accuracy improves when increasing the number of folds *k*. This means they were able to learn more document-category mappings with larger training data size.

6.2.3. Discussion

To sum up, we compare our classification approaches and highlight their limitations. On the one hand, IterativeAdditive and GradientDescentANN augment every document with a set of inferred features that are easily interpretable, where every feature provides information regarding the document's correlation with the target categories. While this provides improved classification results in many cases, yet the quality of the inferred features depends on the quality of the TF-ICF matrix and on the size of the feature space. This might be a limitation in a vector space made of a smaller set of document and features where the matrix might have entries that are similar with each other, especially when documents have similar vector representations. On the other hand, FeedForwardANN maps the distances between the documents more appropriately, where document vectors belonging to the same category are closer to a central category vector and are closer to each other in terms of cosine similarity. This is desirable as the resulting vector space reflects more natural distances between document and category representations. Nonetheless, this might be limiting if the category representations are similar with each other. Here, the mapping between document vectors and their most similar category vectors might cause some confusion in the classification task especially among highly similar categories. We further discuss and evaluate the issue of classifying similar categories in Section 6.4.

6.3. Comparative Study

In this section, we compare our CFE approaches against state-of-the-art text classification solutions. We briefly describe the baselines used in this experiment in Section 6.3.1, and then describe the results in Section 6.3.2.

6.3.1. Baseline Approaches

Following our literature review (cf. Section 2), we selected 16 recent approaches that achieved state-of-the art results on the benchmark datasets used in our study[11]:

- *Non-parametric models*: **TF-IDF+Linear Regression** [25], **FastText** [40], **FastTextBigrams** [40], and **CosineSVM** [44]. TF-IDF+Linear Regression follows the bag-of-words model with traditional TF-IDF weighting, and makes use of Logistic Regression to perform the classification task. FastText and FastTextBigrams are variations of TF-IDF+Linear Regression which apply TF-IDF on word/character n-grams (unigram in the case of FastText and bigram in the case of FastTextBigrams), by averaging the word embedding representations of n-grams occurring in every document. They use Logistic Regression to perform the classification task. CosineSVM applies cosine similarity on the document TF-IDF vector representations to compute support vectors for every target category using a support vector machine. It performs classification using maximum support vector similarity.
- *Deep learning models*: **CNN-non-static** [45], **Bi-LSTM** [25]**, SWEM** [46], **TextGCN** [25], **GraphCNN** [47], and **GraphStar** [24]. CNN-non-static is a Convolutional Neural Network classifier which adopts GloVe[12] as a pre-trained word embeddings model. Bi-LSTM also uses the GloVe as a pre-trained model and feeds the resulting word embeddings to a bidirectional LSTM. SWEM is a Simple Word Embedding Model which employs pooling methods (mainly max-pooling and hierarchical pooling) over GloVe word

[11] Note that transformer-based methods, e.g., [42, 43], are not included in our comparative study since they rely on transfer learning and provide additional knowledge that is extrinsic to the dataset. They are usually pre-trained for certain tasks using a big dataset by performing masked language modeling and next sentence prediction. Consequently, they are fine-tuned on the downstream task by training them on the dataset of interest. For this reason, it would be unfair to compare them with our classification methods which infer derived features from the dataset itself without including any external knowledge or transfer learning.

[12] http://nlp.stanford.edu/data/glove.6B.zip

embeddings, and then feeds them to a feedforward ANN to perform text classification. Text GCN is a Graph Convolutional Network which embeds the text corpus into a graph where documents and words are represented as nodes, having weighted document-word and word-word edges. The GCN model is trained on the graph to jointly learn the mappings for both words and documents. GraphCNN employs a graph CNN model on a word embedding similarity graph and a Chebyshev filter. GraphStar is a variant of graph neural approaches which represents i) words as graph nodes and ii) documents as sub-graphs including their constituent word nodes. It introduces topological modifications, including virtual "star" nodes to propagate document sub-graph weights to individual word nodes. Text classification is conduced using a graph network model.

- *Deep attention models*: **PV-DM** [48], **PV-DBOW** [48], **HyperGAT** [49], **LEAM** [26], and **LAHAN** [29]. PV-DM is a paragraph vector model that randomly samples consecutive terms from a document (paragraph) and learns to predict a center term using these terms and a paragraph identifier. PV-DBOW is a Paragraph Vector Distributed Bag of Words model that ignores the context terms from the input document (paragraph), and trains to predict terms that are randomly sampled from the paragraph. HyperGAT introduces a hypergraph network model for document (paragraph) representation, and learns to represent documents as document-level hypergraphs using a dual attention mechanism for sequential (HyperGAT-seq) and semantic (HyperGAT-dual) hyper-edges. LEAM performs a joint term-category embedding where category and term representations are learned in the same space, using an attention mechanism derived from text-category compatibility. It replaces the one-hot vectors representing target categories by embedding vectors which integrate information about term-category relationships. LAHAM is a variant of LEAN which generates embeddings by means of joint term and sentence features, using a hierarchical architecture integrating category information at both term and sentence levels.

6.3.2. Experimental Results

Table 10 shows the mean accuracy and standard deviation results for our CFE approaches and their alternatives. We utilize the same parameter settings described in Section 6.1, and we set the number of feature dimensions to be 15,000 for all models. Results highlight the following observations:

- CFE's IterativeAdditive approach ranks best and second best on 4 out of 5 benchmark datasets (i.e., R8, R52, Ohsumed, and AGNews). The quality of IterativeAdditive can be attributed to its augmented features which are designed and generated to directly satisfy the classification problem. Also, CFE's FeedForwardANN approach ranks best and second best on 2 out of 5 datasets (i.e., R8 and R52). FeedForwardANN replaces the iterative and additive process by a feedforward ANN architecture, and produces transformed features comparable with the augmented features produced by IterativeAdditive.

Table 10. Comparative mean accuracy results on benchmark datasets.
Red color refers to the best score, green color refers to the second best, and blue refers to the third best for each dataset.
Results for alternative approaches are verified from their respective papers. Standard deviation results are shown between parentheses.

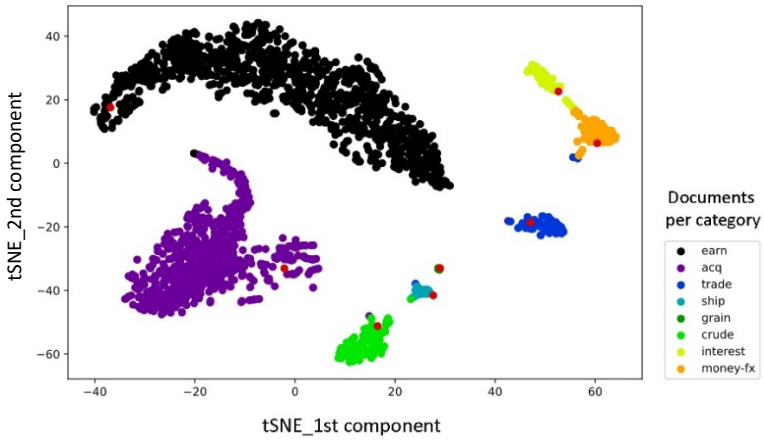
Approach		Benchmark Dataset				
		R8	R52	Ohsumed	20NewsGroup	AGNews
Non-parametric	TF-IDF+LR	93.74 (0.00)	86.95 (0.00)	54.66 (0.00)	83.19 (0.00)	-
	FastText	96.13 (0.21)	92.81(0.09)	57.70 (0.49)	79.40 (0.30)	-
	FastTextBigrams	94.74 (0.11)	90.99 (0.05)	55.69 (0.39)	79.67 (0.29)	-
	CosineSVM	97.48 (0.00)	94.50 (0.00)	-	82.64 (0.00)	-

Deep Learning	CNN-non-static	95.71 (0.52)	87.59 (0.48)	58.44 (1.06)	82.15 (0.52)	-
	Bi-LSTM	96.31 (0.33)	90.54 (0.91)	49.27 (1.07)	73.18 (0.18)	-
	SWEM	95.32 (0.26)	92.94 (0.24)	63.12 (0.55)	85.16 (0.29)	-
	GraphCNN	96.99 (0.12)	92.75 (0.22)	63.89 (0.53)	81.42 (0.32)	-
	TextGCN	97.07 (0.10)	93.56 (0.18)	68.36 (0.56)	**86.34** (0.09)	-
	GraphStar	97.40 (0.20)	**95.00** (0.30)	63.86 (0.53)	**86.90** (0.30)	-
Deep Attention	PV-DM	52.07 (0.04)	74.92 (0.05)	51.14 (0.22)	29.50 (0.07)	
	PV-DBOW	85.87 (0.10)	78.29 (0.11)	46.65 (0.19)	74.36 (0.18)	-
	HyperGAT-seq	97.14 (0.26)	94.15 (0.32)	68.48 (0.45)	86.02 (0.31)	-
	HyperGAT-dual	**97.35** (0.12)	94.72 (0.23)	**69.13** (0.23)	**86.62** (0.16)	91.24 (0.56)
	LEAM	93.31 (0.24)	91.84 (0.23)	58.58 (0.79)	81.91 (0.24)	**91.75** (0.24)
	LAHAN	-	-	-	-	**92.45** (0.00)
CFE	IterativeAdditive	**97.94** (0.02)	**95.13** (0.04)	**68.90** (0.02)	85.51 (0.04)	**91.78** (0.02)
	GradientDescentANN	96.57 (0.02)	93.85 (0.00)	**68.79** (0.05)	86.02 (0.05)	**91.75** (0.01)
	FeedForwardANN	**97.71** (0.55)	**96.98** (1.53)	67.51 (0.03)	83.19 (0.35)	91.54 (0.68)

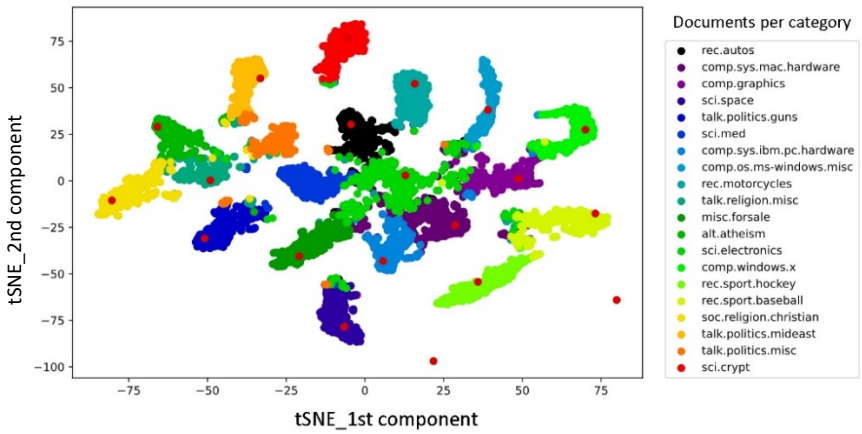
- The CFE methods compute *term-category* relationships using supervised TF-ICF weighting. This produces more accurate classification results compared with i) FastText and FastTextBigrams which make use of the n-gram model to extract *term-term* relationships, ii) PV-DM and PV-DBOW which are trained to predict terms that are randomly sampled from the document, and iii) TextGCN, GraphStart, and HyperGAT which make use of *term-document* relationships through their weighted graph and hypergraph structures.
- The CFE approaches utilize shallow architectures consisting of flat (i.e., IterativeAdditive), 1-layered (i.e., GradientDescentANN), or 2-layered (i.e., FeedForwardANN) structures which are significantly easier to train and manipulate. This is mainly due to our supervised TF-ICF scheme which is the first proposal to use a variation of TF-ICF weighting for representing the target text categories, while existing solutions are designed for input document representation rather than target category representation. Our lean classification models were designed around our term-category TF-ICF scheme, producing results which are on a par with and surpass many deep learning and supervised weighting solutions.
- CFE's IterativeAdditive and FeedForwardANN are ranked third and fourth best on the 20NewsGroup. Results in Table 10 clearly show that CFE was outperformed by its deep learning counterparts, namely TextGCN and GraphStar. This can be attributed to the semantic similarities between 20NewsGroup's categories (e.g., "talk.religion.misc.txt" and "soc.religion.christian.txt"), compared with the other datasets where categories are more distinctive. A careful analysis of the results showed that similarities between the target categories in 20NewsGroup produced similarities between the TF-ICF category feature vectors, resulting in confusion and misclassification for CFE's inference processes. We further investigate this issue in the following subsection.

6.4. Difficulty in Classifying Similar Categories

To shed more light on CFE's performance with 20NewsGroup and our approaches' difficulty in classifying similar categories, we attempt to visualize the similarities between CFE's document feature vectors and their category feature vectors in a two-dimensional space. We use the T-distributed stochastic neighborhood embedding (t-SNE) metho d [50]: a non-deterministic non-linear dimensionality reduction technique that embeds vectors in a lower dimension in which the neighborhoods of the vectors are preserved. This allows us to visualize whether document vectors are close to or far from their corresponding category vectors.



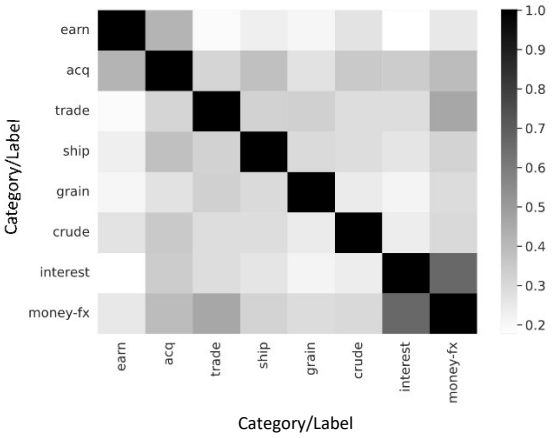
b. R8 dataset (similar visualizations for R52 and AGNews datasets are reported in [41]).



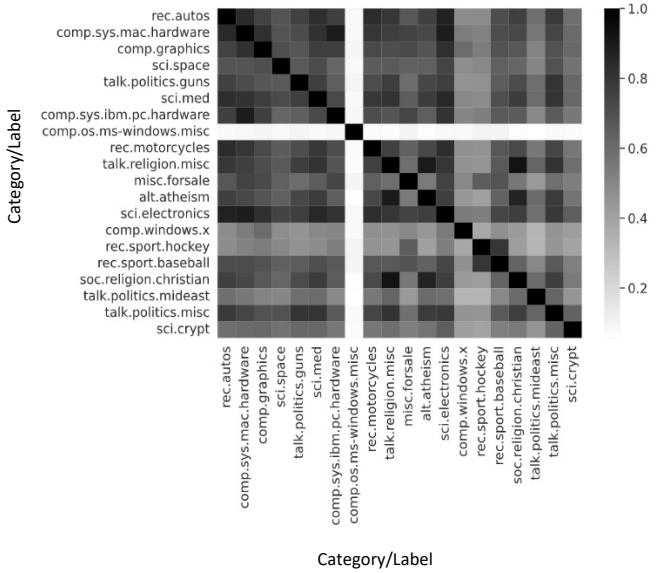
a. 20NewsGroup dataset

Figure 10. t-SNE plots of CFE's document vector and category vector mappings.

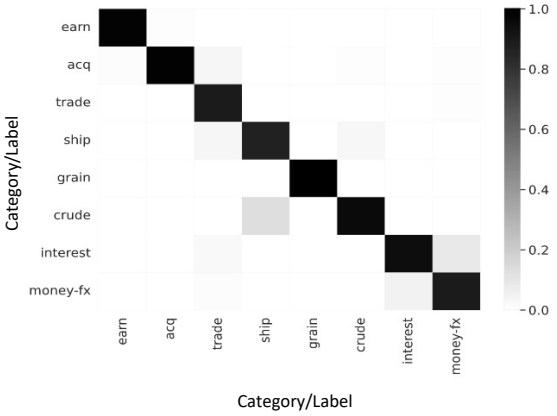
Figure 10 shows the t-SNE visualizations for 20NewsGroup compared with the R8 dataset used in our empirical study (R52 and AGNews visualizations are similar to R8, and are reported in [41]). One can see that document vectors from 20NewsGroup are less separable and more distributed across the reduced 2-D feature space, compared with document vectors from R8 which are clearly more separated and well aggregated in distinct globular areas. Furthermore, most document vectors from R8 tend to revolve around their category centers, which is not always the case with 20NewsGroup where document vectors appear to be stretched-out across large intersecting areas between neighboring category centers. We believe these intersecting areas of document vectors, which are spread midway between neighboring (i.e., similar) categories, result in confusion and misclassification for CFE's inference processes.



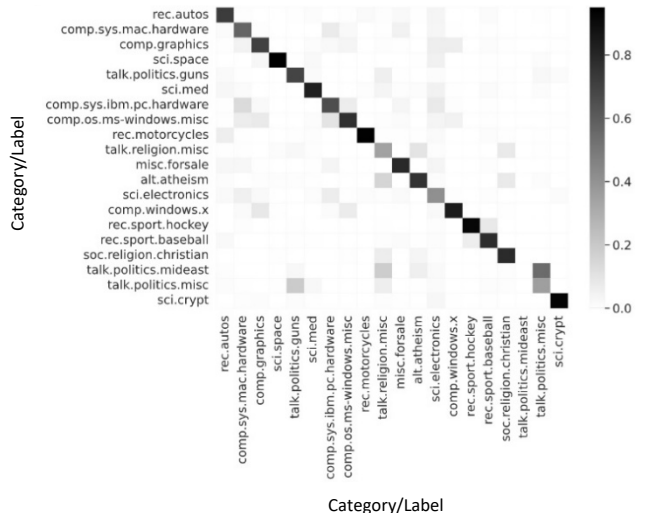
a. Cosine similarity value between representative category vectors of the R8 dataset.



b. Cosine similarity value between representative category vectors of the 20NewsGroup dataset.



c. Confusion matrix on the test set of the R8 dataset.



d. Confusion matrix on the test set of the 20NewsGroup dataset.

Figure 11. Cosine similarity and confusion matrices between target categories of the test datasets.

Furthermore, we report and visualize the similarities between CFE's category feature vectors, by plotting their pair-wise cosine similarity and classification confusion matrices in Figure 11. Here, one can clearly notice that category pairs with high cosine similarity levels in Figures 11.a and b also share high confusion levels in Figures 11.c and d respectively. This is specifically apparent with 20NewsGroup where more categories share higher similarities (e.g., "talk.politics.guns" and "talk.politics.misc", and "comp.graphics" and

"comp.windows.x" in 20NewsGroup), compared with R8 having less similar and thus less confusing categories. In other words, given that some of 20NewsGroup's categories are quite similar with each other, their CFE feature vectors inferred through the TF-ICF weighting process were also similar with each other, which resulted in confusion and misclassifications during CFE's category inference process. A possible solution would be to fine-tune and calibrate the parameters of CFE's classifiers according to the target categories of each dataset. This can be handled automatically as an optimization problem and can be solved using a number of known techniques that apply linear programming and machine learning to identify the best weights for a given problem class, e.g., [34-36]. We report this investigation to a dedicated future study.

6.5. Time Performance

We also evaluate CFE's time performance and compare it with existing solutions. Time experiments were carried out on 12th Gen Intel(R) Core(TM) i7-1260P processor with 2.66 GHz processing speed and 16 GB of RAM. Table 11 reports computation time as the wall-clock time for 1000 iterations, applied on the Yahoo Answers dataset. Results were scaled and averaged across multiple runs, to ensure consistency across the different models. Resuls show that our CFE solutions, namely IterativeAdditive and GradientDescentANN, use much less parameters and require significantly less computation time compared with the existing solutions. FeedforwardANN requires relatively more time than its two CFE alternatives due to the overhead added through its ANN architecture, yet its performance remains on a par with the fastest deep learning models (namely SWEM and LEAM).

Table 7. Comparing the number of parameters and computation time with existing solutions.

Approach	Parameters	Computation time (s)
CNN-non-static [45]	541K	171
Bi-LSTM [25]	1.8M	598
SWEM [46]	61K	63
LEAM [26]	65K	65
CFE - IterativeAdditive	20K	19
CFE - GradientDescentANN	20K	17
CFE - FeedForwardANN	115K	86

7. Conclusion

In this paper, we introduce a new text classification framework for Category-based Feature Engineering titled CFE, which aims to improve classification quality by integrating term-category relationships in document and category representations. Our solution consists of a supervised weighting scheme based on a variant of the TF-ICF (Term Frequency-Inverse Category Frequency) model [11]. Different from existing approaches which are designed for document representation, e.g., [12-14], we adapt TF-ICF to produce weighted representations for the target categories. We introduce a set of new synthetic features inferred from TF-ICF including both document vectors and target category vectors, and we design three novel classification approaches to capture the relationships between document terms and target categories. Our classification models provide lean architectures, compared with their more complex deep learning and deep attention model alternatives, while mostly producing improved and comparable classification results.

We are currently conducting more experiments, considering different feature selection [51], feature transformation [52], and latent semantic and embedding approaches [53, 54], to address the issue of classifying

similar categories. We also plan to consider external corpora to augment the descriptiveness of target classes, by integrating corpus-based statistics (e.g., distributional thesaurus construction [55], association rule mining [56], and unsupervised clustering [57]). We also aim to consider semantic data augmentation [58, 59] and semantic indexing capabilities [60, 61] to augment the target feature vectors. We aim to integrate a human tailored knowledge base such as WordNet [62, 63] and DBPedia [64, 65], and evaluate the quality of corpus-based versus knowledge-based feature vector augmentation, and their impact on the classification task.

References

[1] Pham P., Nguyen L., Pedrycz W. and Vo B. *Deep Learning, Graph-based Text Representation and Classification: a Survey, Perspectives and Challenges*. Artificial Intelligence Review, 2022, https://doi.org/10.1007/s10462-022-10265-7.

[2] Mironczuk M. and Protasiewicz J., *A Recent Overview of the State-of-the-Art Elements of Text Classification.* Expert Systems and Applications 2018. 106: 36-54.

[3] Han J. and Akbari M., *Vertical Domain Text Classification: Towards Understanding IT Tickets Using Deep Neural Networks.* AAAI Conference on Artificial Intelligence (AAAI'18), 2018. pp. 8202-8203.

[4] Revina A., et al., *IT Ticket Classification: The Simpler, the Better.* IEEE Access, 2020. 8:193380-193395.

[5] Ahmed H., et al., *Detecting Opinion Spams and Fake News using Text Classification.* Security & Privacy, 2018. 1(1).

[6] Kaddoura S., et al., *A Spam Email Detection Mechanism for English Language Text Emails Using Deep Learning Approach.* Enabling Technologies: Infrastracture for Collaborative Enterprises (WETICE'20) 2020. pp. 193-198.

[7] Fares M., et al., *Unsupervised Word-level Affect Analysis and Propagation in a Lexical Knowledge Graph.* Elsevier Knowledge-Based Systems, 2019. 165: 432-459.

[8] Fares M., et al., *Difficulties and Improvements to Graph-based Lexical Sentiment Analysis using LISA* IEEE International Conference on Cognitive Computing (ICCC'19), 2019.

[9] Chauhan U., et al., *A Comprehensive Analysis of Adverb Types for Mining User Sentiments on Amazon Product Reviews.* World Wide Web, 2020. 23(3): 1811-1829.

[10] Daniel D. and Meena M., *A Novel Sentiment Analysis for Amazon Data with TSA based Feature Selection.* Scalable Computing: Practice and Experience, 2021. 22(1): 53-66.

[11] Tang Z., et al., *Several Alternative Term Weighting Methods for Text Representation and Classification.* Knowledge Based Systems, 2020. 207:106399.

[12] Wang D. and Zhang H., *Inverse-Category-Frequency based Supervised Term Weighting Schemes for Text Categorization.* Journal of Information Science and Engineering, 2013. 29(2): 209-225.

[13] Domeniconi G., et al., *A Comparison of Term Weighting Schemes for Text Classification and Sentiment Analysis with a Supervised Variant of TF-IDF.* Inter. Conf. on Data Technologies and Applications (DATA'16) 2016. pp. 39-58.

[14] Tang Z., et al., *An Improved Supervised Term Weighting Scheme for Text Representation and Classification.* Expert Systems and Applications, 2022. 189: 115985.

[15] Alsaeedi A., *A Survey of Term Weighting Schemes for Text Classification.* International Journal of Data Mining, Modelling and Management, 2020. 12(2): 237-254.

[16] Salton G., *Automatic text processing: the transformation, analysis, and retrieval of information by computer.* Addison-Wesley Longman, Boston, MA, USA 1989. pp. 530.

[17] McGill M., *Introduction to Modern Information Retrieval.* 1983. McGraw-Hill, New York.

[18] Salton G. and Buckley C., *Term-weighting approaches in automatic text retrieval.* Information Processing and Management, 1988. 24(5):513 -523.

[19] Salton G. and Mcgill M.J., *Introduction to Modern Information Retrieval,.* 1983. McGraw-Hill, Tokio.

[20] Debole F. and Sebastiani F., *Supervised Term Weighting for Automated Text Categorization.* Text Mining and Its Applications, Springer, Berlin, Heidelberg., 2004. pp .81–97.

[21] Mladenic D. and Grobelnik M., *Feature Selection for Classification based on Text Hierarchy.* Conference on Automated Learning and Discovery (CONALD'98), 1998.

[22] Domeniconi G., et al., *A Study on Term Weighting for Text Categorization: A Novel Supervised Variant of TF-IDF.* International Conference on Data Technologies and Applications (DATA'15) 2015. pp. 26-37.

[23] Kadhim A., *Survey on Supervised Machine Learning Techniques for Automatic Text Classification.* Artificial Intelligence Review, 2019. 52(1): 273-292.

[24] Lu H., et al., *Graph Star Net for Generalized Multi-Task Learning.* Computing Research Repository, 2019. CoRR abs/1906.12330 (2019).

[25] Yao L., et al., *Graph Convolutional Networks for Text Classification.* AAAI Conference on Artificial Intelligence (AAAI'19), 2019. pp. 7370-7377.

[26] Wang G., et al., *Joint Embedding of Words and Labels for Text Classification.* Annual Meeting of the Association for Computational Linguistics (ACL'18), 2018. 2321-2331.

[27] Zhang H., et al., *Multi-Task Label Embedding for Text Classification.* Conference on Empirical Methods in Natural Language Processing (EMNLP'18), 2018. pp. 4545-4553.

[28] Wang C. and Tan C., *Label-Based Convolutional Neural Network for Text Classification.* International Conference on Control Engineering and Artificial Intelligence (CCEAI'2021) 2021. pp. 136–140.

[29] Li X., et al., *Label-Attentive Hierarchical Attention Network for Text Classification.* Proceedings of the 2020 5th International Conference on Big Data and Computing (ICBDC'20), 2020. pp. 90–96.

[30] Byerly A. and Kalganova T., *Homogeneous Vector Capsules Enable Adaptive Gradient Descent in Convolutional Neural Networks.* IEEE Access, 2021. 9: 48519-48530.

[31] Abboud R. and Tekli J., *Integration of Non-Parametric Fuzzy Classification with an Evolutionary-Developmental Framework to perform Music Sentiment-based Analysis and Composition* Soft Computing, 2019. 24(13): 9875-9925

[32] Thongtan T. and Phienthrakul T., *Sentiment Classification Using Document Embeddings Trained with Cosine Similarity.* Annual Meeting of the Association for Computational Linguistics (ACL'19), 2019. (2): 407-414.

[33] Tekli J., et al., *An Overview of XML Similarity: Background, Current Trends and Future Directions.* Elsevier Computer Science Review, 2009. 3(3):151-173.

[34] Hopfield J. J., *The Effectiveness of Neural Computing.* IFIP World Computer Congress (WCC'89), 1989. 402-409.

[35] Zou F., et al., *A Reinforcement Learning Approach for Dynamic Multi-objective Optimization.* Information Sciences, 2021. 546: 815-834.

[36] Azar D., et al., *A Combined Ant Colony Optimization and Simulated Annealing Algorithm to Assess Stability and Fault-Proneness of Classes Based on Internal Software Quality Attributes.* Inter. Journal of Artificial Intelligence, 2016. 14:2.

[37] Hinton G. E., et al., *A Fast Learning Algorithm for Deep Belief Nets.* Neural Computation, 2006. 18 (7): 1527–1554.

[38] Hornik K., *Approximation Capabilities of Multilayer Feedforward Networks.* Neural Networks, 1991. 4(2), 251–257.

[39] Lee J. H., *Properties of Extended Boolean Models in Information Retrieval.* Proceedings of the ACM SIGIR Conference, 1994. Springer-Verlag New York, pp.182-190.

[40] Joulin A., et al., *Bag of Tricks for Efficient Text Classification.* Conference of the European Chapter of the Association for Computational Linguistics (EACL'17), 2017. pp. 427-431.

[41] Attieh J. and Tekli J., *Fast, Simple, and Effective Frequency-based Text Classification.* Technical Report - LAU School of Engineering, E.C.E. Dept., 2021. http://sigappfr.acm.org/Projects/CFE/

[42] Acheampong F., et al., *Transformer Models for Text-based Emotion Detection: a Review of BERT-based Approaches.* Artificial Intelligence Review 2021. 54(8): 5789-5829.

[43] Lin X., et al., *ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification.* World Wide Web Conference (WWW'22) 2022. pp. 633-642.

[44] Park K., et al., *A Methodology Combining Cosine Similarity with Classifier for Text Classification.* Applied Artificial Intelligence, 2020. 34(5): 396-411.

[45] Kim Y., *Convolutional Neural Networks for Sentence Classification.* Conference on Empirical Methods in Natural Language Processing (EMNLP'14), 2014. pp. 1746–1751.

[46] Shen D., et al., *Baseline Needs More Love: On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms.* Annual Meeting of the Association for Computational Linguistics (ACL'18), 2018. pp. 440-450.

[47] Defferrard M., et al., *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.* Conference on Neural Information Processing Systems (NeurIPS'16), 2016. pp. 3844–3852.

[48] Le Q. and Mikolov T., *Distributed Representations of Sentences and Documents.* International Conference on Machine Learning (ICML'14), 2014. pp. 1188-1196.

[49] Ding K., et al., *Be More with Less: Hypergraph Attention Networks for Inductive Text Classification.* Conference on Empirical Methods in Natural Language Processing (EMNLP'20), 2020. pp. 4927-4936.

[50] Flexa C., et al., *Polygonal Coordinate System: Visualizing high-dimensional data using geometric DR, and a deterministic version of t-SNE.* Expert Systems and Applications, 2021. 175: 114741.

[51] Pintas J., et al., *Feature Selection Methods for Text Classification: a Systematic Literature Review.* Artificial Intelligence Review, 2021. 54(8): 6149-6200 (2021).

[52] Raghavan A. K., et al., *Label Frequency Transformation for Multi-Label Multi-Class Text Classification.* Conference on Natural Language Processing (KONVENS'19), 2019.

[53] Moreo A., et al., *Word-class Embeddings for Multiclass Text Classification.* Data Mining and Knowledge Discovery, 2021. 35(3): 911-963.

[54] Ma Y., et al., *Hybrid Embedding-based Text Representation for Hierarchical Multi-label Text Classification.* Expert Systems and Applications, 2022. 187:115905.

[55] Sarkissian S. and Tekli J., *Unsupervised Topical Organization of Documents using Corpus-based Text Analysis.* Inter. ACM Conference on Management of Emergent Digital EcoSystems (MEDES'21), 2021. pp. 87-94.

[56] Haraty R. & Nasrallah R., *Indexing Arabic Texts using Association Rule Data Mining.* Library Hi Tech, 2019. 37(1): 101-117.

[57] Haraty R., et al., *An Enhanced k-Means Clustering Algorithm for Pattern Discovery in Healthcare Data.* Intelligent Journal on Distributed Sensor Networks, 2015. 11: 615740:1-615740:11.
[58] Wei J. and Zou K., *EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks.* Conference on Empirical Methods in Natural Language Processing (EMNLP'19), 2019. (1) 2019: 6381-6387.
[59] Cai L., et al., *A Hybrid BERT Model That Incorporates Label Semantics via Adjustive Attention for Multi-Label Text Classification.* IEEE Access, 2020. 8:152183-152192.
[60] Tekli J., et al., *SemIndex+: A Semantic Indexing Scheme for Structured, Unstructured, and Partly Structured Data.* Elsevier Knowledge-Based Systems, 2019. 164: 378-403.
[61] Tekli J., et al., *Full-fledged Semantic Indexing and Querying Model Designed for Seamless Integration in Legacy RDBMS.* Data and Knowledge Engineering, 2018. 117: 133-173.
[62] Zhu X., et al., *An Improved Class-Center Method for Text Classification Using Dependencies and WordNet.* Natural Language Processing and Chinese Computing (NLPCC'19), 2019. (2): 3-15.
[63] Poostchi H. and Piccardi M., *Cluster Labeling by Word Embeddings and WordNet's Hypernymy.* Australasian Language Technology Association Workshop (ALTA'18) 2018. pp. 66-70.
[64] Mouriño-García M., et al., *Wikipedia-based Hybrid Document Representation for Textual News Classification.* Soft Computing, 2018. 22(18): 6047-6065.
[65] Flisar J. and Podgorelec V., *Improving Short Text Classification using Information from DBpedia Ontology.* Fundamenta Informaticae, 2020. 172(3): 261-297.