Representation of Texts into String Vectors for Text Categorization

Taeho Jo
School of Computer and Information Engineering, Inha University
tjo018@inha.ac.kr

Received 7 August 2009; Revised 5 February 2010; Accepted 16 February 2010

In this study, we propose a method for encoding documents into string vectors, instead of numerical vectors. A traditional approach to text categorization usually requires encoding documents into numerical vectors. The usual method of encoding documents therefore causes two main problems: huge dimensionality and sparse distribution. In this study, we modify or create machine learning-based approaches to text categorization, where string vectors are received as input vectors, instead of numerical vectors. As a result, we can improve text categorization performance by avoiding these two problems.

Categories and Subject Descriptors: Text Mining **[Data Mining]**:

General Terms: Text Categorization

Additional Key Words and Phrases: String Vectors, Text Categorization, Neural Text Categorizer

1. INTRODUCTION

Text categorization refers to the process of assigning one or more predefined categories to each unseen document. It requires two preliminary manual tasks: predefinition of categories and preparation of sample-labeled documents. As a learning process, the classification capacity is built automatically using the manually prepared labeled documents. Using the built classification capacity, unseen documents are classified automatically. Therefore, text categorization is carried out with three steps: predefinition of categories and preparation of sample-labeled documents, learning process, and classification of unseen documents.

Documents should be encoded into numerical vectors as the requirement for a traditional approach to text categorization. Words are extracted from a corpus as feature candidates. Among them, some are selected as features by a criterion or a combination of criteria. Various criteria were proposed. TF-IDF, frequency, information

Copyright(c)2010 by The Korean Institute of Information Scientists and Engineers (KIISE). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Permission to post author-prepared versions of the work on author's personal web pages or on the noncommercial servers of their employer is granted without fee provided that the KIISE citation and notice of the copyright are included. Copyrights for components of this work owned by authors other than KIISE must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires an explicit prior permission and/or a fee. Request permission to republish from: JCSE Editorial Office, KIISE. FAX +82 2 521 1352 or email office@kiise.org. The Office must receive a signed hard copy of the Copyright form.

gain, and chi-square are typical measures. Note the citation of two problems, that is, huge dimensionality and sparse distribution, which are inevitable in encoding documents into numerical vectors.

The first problem, which is inherent in encoding documents into numerical vectors, is the huge dimensionality. This problem refers to the phenomena, where many features are required for representing documents into numerical vectors more robustly. The number of feature candidates extracted from a corpus usually reaches several ten thousands. Even if selecting only some of them using advanced features selection criteria, the number of the selected ones reach several hundreds. The number of features is much smaller than that of feature candidates, but it is still large with respect to the size of input data.

The second problem inherent in encoding is sparse distribution. It refers to the phenomena where zero values dominate each numerical vector. In other words, zero values take more than 90% of each numerical vector representing a document. The reason is that each word used as a feature has very small coverage in the given corpus. The computation of the inner product of two sparse numerical vectors becomes very fragile to this problem.

The idea of this research is to represent documents into string vectors as an alternative form. A string vector refers to an ordered finite set of words in this study. In each string vector, a word is given as an element or a value, whereas it is given as an attribute or a feature in each numerical vector. The features of string vectors may be defined as posting, statistical, and grammatical properties of words. However, in this study, for simplicity and ease of implementation, we define the feature of each string vector as the first-ranked frequent word, the second-ranked one, and so on; each string vector consists of words in a descending order of their frequencies in the given document.

We propose an operation necessary for the learning and classification of string vectors-based machine-learning algorithms. The proposed operation is called average semantic similarity. It corresponds to the inner product, which is the operation on numerical vectors and is used for computing a similarity between them. In other words, the operation of average semantic similarity measures the similarity of two-string vectors. It is described in detail in Section 3.4.

The second idea proposed in this study is to modify the existing machine-learning algorithms for their string vector-based versions. The operation on string vectors, which corresponds to the inner product in the context of numerical vectors, is defined for this purpose. In this paper, the operation is called average semantic similarity between two string vectors. A similarity matrix is built automatically from the given corpus, and it is used as the basis for carrying out the operation. We modify the K Nearest Neighbors (KNN) and the Support Vector Machine (SVM) into their string vector-based versions as the approaches to text categorization, and describe them in Sections 4.1 and 4.2, respectively.

The final idea of this paper is to create a new machine-learning algorithm as well as to modify existing ones. In this research, the new machine-learning algorithm is called Neural Text Categorizer (NTC), which receives string vectors as its input vector. With respect to its architecture, it consists of three layers: input layer, learning

layer, and output layer. It follows the Perceptron with respect to the learning process in that the weights are updated only in each misclassification and the Categorical Score Values (CSVs) are computed by the linear combinations of weights and input values.

However, the NTC is essentially different from the Perceptron in that its input vector is given as a string vector, and it will be described in detail in Section 4.3.

This article is organized into six sections including this section. In Section 2, we will explore previous works relevant to this research. In Section 3, we will describe string vectors and the involved operation on them with respect to its definitions and properties. In Section 4, we present the string vector-based approaches to text categorization. In Section 5, we will validate empirically the performance of string vector-based approaches to text categorization by comparing them with numerical vector-based ones. In Section 6, we will mention the significance of our results as a conclusion.

2. PREVIOUS WORKS

This section reviews previous works relevant to this research. In Section 2.1, we explore the existing cases of using machine-learning algorithms for text categorization tasks. Even if there are various approaches to text categorization, we focus only on four representative ones: Naive Bayes (NB), KNN, SVM, and Multilayer Perceptron (MLP) with Back Propagation (BP). In Section 2.2, we explore previous solutions to two main problems in encoding documents into numerical vectors.

2.1 Previous Approaches to Text Categorization

This subsection reviews previous approaches to text categorization. Rule-based approaches may be regarded as traditional and popular. However, machine learning-based approaches have already replaced the rule-based ones because of their greater flexibility and scalability. We review no rule-based approach, but only four representative machine learning-based approaches: NB, KNN, SVM, and MLP with BP. Therefore, we present each of the four machine-learning algorithms in terms of its initial proposal, initial application, and its successive applications to text categorization.

One of the typical approaches to text categorization is KNN. It was initially proposed by [Cover and Hart 1967] and was applied to text categorization by [Masand et al. 1992]. It was recommended by Yang as one of the best approaches in her evaluation of more than ten approaches to text categorization [Yang 1999]. Sebastiani regarded KNN as an approach comparable to SVM, which showed its best performance for text categorization on the standard test bed, Reuter 21578 in 2002 [Sebastiani 2002].

Another typical approach to text categorization is NB. It is based on Bayes rule, and it was first proposed by Kononenko [Kononenko 1989]. In 1997, Mitchell mentioned NB as a typical approach to text categorization in his textbook [Mitchell 1997]. In 1999, Mladenic and Grobelink attempted to find an optimal feature-selection method under the environment, where NB was used for text categorization [Mladenic and Grobelink 1999]. Some of the applications of NB to text categorization were mentioned

again by [Eyheramendy et al. 2003].

SVM became a popular approach to text categorization in the late 1990s. Hearst first introduced SVM in her magazine article in 1998 [Hearst 1998]. In the same year, Joachims [Joachims 1998] applied it to text categorization. In 1999, Drucker proposed SVM as an approach to spam-mail filtering, which is a practical instance of text categorization [Drucker et al. 1999]. Similar to the case of NB, in 2000, SVM was also presented as a typical approach to text categorization by Cristianini and Shawe-Taylor in their textbook [Cristianini and Shawe-Taylor 2000].

Among neural networks, the most popular model is MLP with a back-propagation algorithm, subsequently applied to text categorization. The MLP with back propagation was initially created by McClelland and Rumelhart in 1986 [McClelland and Rumelhart 1986]. Winer initially applied it to text categorization in 1995 [Wiener 1995]. He validated empirically that the MLP with back propagation worked better than KNN on the standard test bed, Reuter 21578 [Wiener 1995]. Later, Ruiz and Srinivasan proposed the hierarchical combination of MLPs in 2002 [Ruiz and Srinivasan 2002].

2.2 Previous Solutions to Huge Dimensionality and Sparse Distribution

It is required to encode documents into numerical vectors if one chooses to use any of the previous approaches to text categorization, including those mentioned in Section 2.1. Such encoding leads to two main problems: huge dimensionality of each numerical vector and sparse distribution within it. Huge dimensionality means that each document should be represented into a huge-dimensional numerical vector for preventing its information loss, though this takes much time for processing and many training examples are required for the robust classification proportional to the dimension. Sparse distribution refers to the phenomenon where zero values are dominant in each numerical vector, which degrades the performance of text categorization because of the loss of discrimination among numerical vectors. Therefore, previous researches attempted to find solutions to the two problems, and we review the solutions in this section.

As a first attempt, in 2002, Lodhi et al. proposed string kernel for the use of SVM for text categorization [Lodhi et al. 2002]. The string kernel refers to a kernel function of two raw texts, which returns their syntactical similarity. Owing to the string kernel, documents did not need to be represented into numerical vectors. However, it took much time to perform the string kernel, and the performance of text categorization failed to improve.

Because of the cost of encoding documents into string vectors for the solution, it was proposed that documents should be encoded into tables. In 2007, Jo and Cho claimed that documents should be encoded for text categorization [Jo and Cho 2007]. They proposed an approach that classified documents by matching tables representing unseen documents with those representing a collection of sample-labeled documents. It was validated that their approach worked better than NB, KNN, and SVM. However, since all words except stop words were used for representing documents into tables, it took more time to classify each document than other earlier approaches.

3. STRING VECTORS

This section presents string vectors and operations on them. In Section 3.1, we describe definitions and properties of string vectors. In Section 3.2, we cover how to encode documents into string vectors. In Section 3.3, similarity matrix is defined as the basis for the semantic operations on string vectors. Finally, in Section 3.4, we will define and describe the semantic operation on string vectors, called 'average semantic similarity'.

3.1 Definition of String Vectors

A string vector is defined as an ordered finite set of strings. String vectors are structured data representing raw data and should be distinguished from other structured data such as numerical vectors and bags of words. In this paper, we propose string vectors as an alternative representation of documents to numerical vectors. We define string vectors formally and informally, and compare them with numerical vectors and bags of words.

A string vector is denoted as an ordered finite set of strings by $\mathbf{str}=[str_1, str_2, ..., str_n]$. A string, which is an element of the string vector, is a combination of alphabets of a given language and signifies a word or a vocabulary in natural language. Since elements are ordered in a string vector, two string vectors with identical elements, but different orders are treated as different ones. The finite number of elements in the string vector is called the dimension of the string vector. For example, [computer information system], [business company industry], and [biology RNA DNA] are instances of three-dimensional string vectors.

The differences between string vectors and numerical vectors are illustrated in Table I. As a similarity measure between the structured data, cosine similarity or Euclidean distance is used in numerical vectors, whereas the average semantic similarity is used in string vectors. In numerical vectors, words become features, which decide elements, while in string vectors, statistical, linguistic, and posting information of words become features. With respect to the relation between string vectors and numerical vectors, if strings as elements only replace numbers, a numerical vector is equivalent to a string vector.

The differences between string vectors and bags of words are illustrated in Table II. Both types of structured data have strings as their elements. As a similarity measure, cardinality of intersection of two bags of words is used, while the average semantic similarity is used in string vectors. A bag of words is defined as an unordered variable set of words, while a string vector is defined as an ordered finite set of words.

Table I. Comparison of Numerical Vectors and String Vectors.

	Numerical Vector	String Vector
Element	Numerical Value	String
Similarity Measure	Inner Products Euclidean Distance	Semantic Similarity
Attributes	Words	Property of Words

Table II. Comparison of Bag of Words and String Vectors.

	Bag of Words	String Vector
Element	Word	String
Similarity Measure	Number of Shared Words	Semantic Similarity
Set	Unordered Infinite Set	Ordered Finite Set

Although a bag of words and a string vector look similar to each other, they should be distinguished, based on Table II.

There are three advantages in representing documents into string vectors. The first advantage is to avoid the aforementioned two main problems completely: the huge dimensionality and the sparse distribution. The second advantage is that string vectors are characterized as more transparent representations of documents than numerical vectors; it is easier to guess the content of documents only from their representations. The third advantage is that there is the potential possibility of tracing more easily, as to why documents are classified so. However, in order to use string vectors more freely, it is necessary to set foundations that are more mathematical.

3.2 Encoding Documents into String Vectors

This subsection shows the process of encoding documents into string vectors. The definition of the features of each string vector is based on the frequencies of words in its source document. The document is indexed into a list of words. The words are sorted in a descending order of their frequencies, and the highest frequent word is taken as the first element of the string vector. Therefore, in this section, we will describe the process of encoding documents into string vectors based on frequencies.

Actually, we can define features in encoding documents into string vectors in various ways. In spite of that, the reason for adopting only frequencies as the features is due to a relatively easy implementation of the proposed system. Let us assume that the dimension of the string vector representing a document is d. The first element and the last element of the string are the first-ranked frequent word and the dth-ranked frequent word, respectively; in each string vector, words are arranged in a descending order of their frequencies. If the features of string vectors are defined as others such as a noun in the given title, a noun in the first sentence, a verb in the first sentence, and so on, the implementation of the module of encoding documents into string vectors becomes complicated.

A document or documents may be indexed into a list of words through the three steps: tokenization, stemming, and stop word removal. Tokenization refers to the process of segmenting a long text into tokens by the white space or the punctuation marks. Stemming means the mapping of each token into its root form; for example, it maps plural nouns and verbs in their passive forms into singular nouns and verbs in their root forms, respectively. Stop words are the words, which function only grammatically, irrelevant to the content; they need to be removed in the third step. As output of the indexing, a list of words and their frequencies is generated.

We may derive the string vector from the list. The words in the list are sorted in a descending order of their frequencies. We select the d highest frequent words as the elements of the string vector. Hence, a string vector is made by selecting the highest words from the list.

We defined the features of string vectors very clearly for simple implementation, in spite of the availability of various types of features. The first type of feature belongs to the statistical class, which includes the highest frequent word in the entire text, the most highly weighted words, and the highest frequent word in a particular paragraph. The second type of feature belongs to the posting class where feature values are given depending on the positions of the given full text. The third type of feature belongs to the grammatical class to which a subjective noun, a verb, and an objective noun in a particular sentence belong. Features of string vectors may be defined in the combination of several types.

3.3 Similarity Matrix

This subsection presents similarity matrix, which is given as the basis for carrying out the operation on string vectors. The similarity matrix is built from a particular corpus. The similarity matrix refers to a word-by-word square matrix where each entry indicates a semantic similarity between two words corresponding to rows and columns. The similarity matrix has two properties: its diagonal elements are 1.0 and it is a symmetry matrix.

The similarity matrix is written as follows:

$$\begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1N} \\ s_{21} & s_{22} & \cdots & s_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N1} & s_{N2} & \cdots & s_{NN} \end{pmatrix}$$

In the above matrix, N indicates the total number of words, and the similarity matrix is a $N \times N$ matrix. In similarity matrix, its columns and rows correspond to words; the ith column and the ith row correspond to the identical word. The entry of the similarity matrix, s_{ij} indicates the semantic similarity between the word which corresponds to ith column or ith row and the word which corresponds to jth column or jth row, and it is computed by equation (1),

$$sim(str_i, str_j)=s_{ij}=\frac{2\times df(w_i, w_j)}{df(w_i)+df(w_j)} \tag{1}$$

where in the corpus, $df(w_i, w_j)$ indicates the number of documents including both words, w_i and w_j; $df(w_i)$ indicates the number of documents including the word, w_i; and $df(w_j)$ indicates the number of documents including the word, w_j. Therefore, in the given corpus, the more the documents include both words, the higher the semantic similarity between the words becomes.

The first property of the similarity matrix is that it is symmetric. The following expression is applicable to every entry of the similarity matrix.

$$s_{ij}=s_{ji},\ 1 \le i, j \le N$$

The following shows that the similarity matrix is a symmetry matrix.

$$s_{ij}=\frac{2\times df(w_i, w_j)}{df(w_i)+df(w_j)}=\frac{2\times df(w_j, w_i)}{df(w_i)+df(w_j)}=s_{ji}$$

From the expression above, since both $df(w_i, w_j)$ and $df(w_j, w_i)$ signify the number of documents including both words, w_i and w_j, the commutative law is applicable as follows:

$$df(w_i, w_j)=df(w_j, w_i)$$

Therefore, the applicability of the commutative law to the computation of the semantic similarity between two words characterizes the similarity matrix.

The second property of the similarity matrix is that its diagonal elements are given as 1.0s. In other words, the value of s_{ii} is 1.0. The second property of the similarity matrix can be shown as follows:

$$s_{ii}=\frac{2\times df(w_i, w_i)}{df(w_i)+df(w_i)}=\frac{2\times df(w_i)}{2\times df(w_i)}=1.0$$

In the expression above, $df(w_i, w_i)$ is identical to $df(w_i)$ since both $df(w_i, w_i)$ and $df(w_i)$ signify identically the number of documents including w_i. This property shows that although two identical words may have different meanings depending on their context, they are treated in this paper as the same, both syntactically and semantically.

We need to consider the construction of the similarity matrix from a corpus. The corpus is indexed into a list of words. For each word, a list of documents including the word is built. Using the equation (1), semantic similarities of all possible pairs are computed as elements of the similarity matrix. Once the similarity matrix is built, it can be used continually for the text categorization as long as the given domain is not changed, but note that it costs very much for building the similarity matrix with respect to time and system resources.

3.4 Average Semantic Similarity between Two String Vectors

This subsection defines the operation on string vectors involved in the proposed version of KNN. The operation described in this subsection is called average semantic similarity between two string vectors. There are four properties in the operation and they will be mentioned in the subsequent paragraph. The operation is used as the similarity measure between a training example and an unseen example in the proposed version of KNN. We describe the operation in terms of its definition, computation, and properties.

Before computing the average semantic similarity between two string vectors, let us discuss the concept of semantic similarity between two strings or words. It is assumed that the content of a document tends to aim at its consistency. Words, which are collocated in same documents, tend to have similar or relevant meanings. The more documents that include two words in a given corpus there are, the greater the similarity between the two words becomes. A semantic similarity between two words may be different depending on the given corpus.

Two string vectors are denoted as follows:

$$\mathbf{str_1}=[str_{11}, str_{12}, ..., str_{1d}], \mathbf{str_2}=[str_{21}, str_{22}, ..., str_{2d}]$$

The similarities of 1:1 pairs of two string vectors denoted by $sim(str_{11}, str_{21})$, $sim(str_{12}, str_{22})$, ..., $sim(str_{1d}, str_{2d})$ are computed by getting elements crossing in the rows and columns corresponding to the given strings or words. If there is no such word which corresponds to its row or column, the similarity of the word becomes zero. The average semantic similarity between the two string vectors is computed, using equation (2).

$$sim(\mathbf{str_1}, \mathbf{str_2})=\frac{1}{2}\sum_{i=1}^{d} sim(str_{1i}, str_{2i})$$

where $sim(str_{1i}, str_{2i})$ indicates the similarity between two strings, str_{1i} and str_{2i} and obtained from the similarity matrix. Since semantic similarities of one-to-one pairs are given as normalized values, the average value is computed as a normalized value.

The properties of the average semantic similarity between two string vectors are as follows:

- Average semantic similarity between two identical string vectors is 1.0, since all one to one pairs of elements are 1.0.
- The commutative law is applicable to this operation as expressed

$$sim(\mathbf{str_1}, \mathbf{str_2})=sim(\mathbf{str_2}, \mathbf{str_1})$$

  because the semantic similarity is given as a symmetry matrix.
- This operation involves only two string vectors, since the similarity matrix is given as a two-dimensional array.
- The operation generates a normalized value, since only normalized values are averaged.

The operation on string vectors, called average semantic similarity, corresponds to the inner product, which is a typical operation on numerical vectors. When two vectors are identical in terms of their elements and order, whether they are numerical vectors or string vectors, the similarity between them becomes largest. The difference from the inner product is that the value resulted from the inner product is given as a real value, while the value resulted from the average semantic similarity is given as a normalized value between zero and one. The inner product between numerical vectors is very fragile due to their sparse distributions; it generates a zero value easily. Therefore, in the proposed version of KNN, documents are represented into string vectors and the inner product is replaced by the average semantic similarity as the similarity measure between training and an unseen example.

4. String Vector-based Approaches

This section is concerned with the string vector-based approaches to text categorization, and it is composed of three subsections. In Section 4.1, we describe in detail the string vector-based version of KNN. In Section 4.2, we define the string vector kernel of the SVM. In Section 4.3, we clarify the NTC which is specialized for any task of text categorization.

4.1 The Proposed Version of KNN

This subsection is concerned with the proposed version of KNN, as the approach to text categorization. Sample-labeled documents are encoded into string vectors, but they are not learned in advance. When an unseen document or documents are given its or their nearest neighbors among sample-labeled string vectors are selected. KNN decides its or their categories by voting the target categories of the selected nearest neighbors. We describe the proposed version of KNN where documents are encoded into string vectors and the average semantic similarity is used as the similarity measure.

Before discussing the proposed version of KNN, let us consider its learning process. Since KNN does not learn sample-labeled examples until any unseen example is given, it is called lazy learning algorithm [Mitchell 1997]. When any unseen example is available, it starts the learning. It computes the similarities of the given unseen example with the sample-labeled examples, and selects several nearest ones as neighbors of the unseen example. Therefore, KNN does both learning and classification after any unseen example is given.

The classification of KNN refers to deciding a category or categories of the given unseen example based on the selected neighbors. The K is given as a parameter and indicates the number of neighbors, which are selected for the classification. The K is usually set as an odd number such as one and three. The majority of categories of the selected neighbors decides a category of an unseen example. However, in the initial version, the discrimination among selected neighbors is overlooked.

We need to consider the discrimination among the neighbors based on their distance from the given unseen example. Different weights should be assigned to neighbors depending on the similarity of the unseen example; most nearest neighbor should be assigned a highest value, while least nearest one should be assigned a lowest one. To each category, the sum of weights is computed. We can decide the category of the unseen document corresponding to the maximum sum of weights. Therefore, various schemes for assigning weights to neighbors may be considered.

Table III illustrates the comparison between traditional and proposed version of KNN. In the traditional version, numerical vectors are given as input data, while in the proposed version, string vectors are given so. As for the similarity between a sample-labeled example and an unseen one, the cosine similarity or Euclidean distance is used in the traditional version, while the average semantic similarity is used in the proposed version. In the traditional version, two main problems, which are inherent in encoding documents into numerical vectors, still exist. Therefore, the proposed version is aimed to improve the performance of text categorization due to the absence of the two main problems.

4.2 String Vector-based SVM

The SVM is characterized as a mapping from a particular feature space into another space. Since the inner product of two vectors generates a scalar value in any space, the kernel function is used for generating the result from the inner product in the mapped space without mapping the vectors implicitly. In this section, we will describe

Table III. Comparison of Numerical Vectors and String Vectors.

	Traditional Version	Proposed Version
Input Data	Numerical Vectors	String Vectors
Similarity Measure	Cosine Similarity Euclidean Distance	Semantic Similarity
Dimensionality	Usually Several Hundreds Dimensions	Several Tens Dimensions
Sparse Distribution	Frequent	Never

a version of SVM, which is modified by replacing a kernel function of numerical vectors by one of the string vectors.

The output value of a kernel function of two vectors indicates the value, which results from the inner product of the vectors in the mapped space, and there are typically three types of kernel functions. The first type is a linear kernel function, which is an inner product with its own bias. One in the second type is a polynomial kernel function, which makes a polynomial equation of two vectors. The third type is called a Gaussian kernel function, which is based on the Gaussian distribution based on the difference between two vectors. The value, which results from a kernel function in any type, indicates the similarity between the two vectors in the mapped space.

The general form of the equation of two hyperplanes defined by SVM is defined as equation (3).

$$f(x)=\sum_{i=1}^{d} \alpha_i k(x_i, x) \geq \pm 1 \tag{3}$$

where N is the number of training examples, x_i, i=1, ..., N is a training example, and $k(\cdot)$ is a kernel function. In equation (3), α_i called Lagrange multiplier corresponds to each training example, and learning of SVM means the optimization of the Lagrange multipliers. Almost, all Lagrange multipliers have zero values and only a few have nonzero values; training examples corresponding to nonzero Lagrange multipliers are called support vectors. There are various optimization algorithms; the Sequential Minimization Optimization (SMO) is the-most popular among them.

SVM was originally designed to a binary classification task. If the value of the function, $f(x)$ is more than one, it belongs to the positive class, but otherwise, it belongs to the negative class. Different from any other linear classifier, SVM defines the two equations as the result from its learning: $f(x)=\Sigma_{i=1}^{N} \alpha_i k(x_i, x)=-1$. The area within $-1 \leq \Sigma_{i=1}^{N} \alpha_i k(x_i, x) \leq +1$ is called the marginal area, and the SVM pursues the maximization of the marginal area for better classification of unseen examples. In order to treat real problems, SVM defines the equations with the error tolerance as

Table IV. Comparison of Two Versions of SVM

	Traditional Version	Modified Version
Input Data	Numerical Vectors	String Vectors
Kernel Function	Inner Products	Semantic Similarity
Sparse Distribution	Available	Not Available

$\Sigma_{i=1}^{N} \alpha_i k(x_i, x) \leq -1+\zeta$ and $\Sigma_{i=1}^{N} \alpha_i k(x_i, x) \geq 1-\zeta$.

The differences of the two versions of SVM are illustrated in Table IV. In the traditional version, an input data is given as a numerical vector whereas in the modified version, an input data is given as a string vector. In the traditional version, three types of kernel functions are available, while currently only the average semantic similarity itself is given as the kernel function in the proposed version. The traditional version of SVM is tolerance to the huge dimensionality of numerical vectors, but fragility to the sparse distribution of each numerical vector. The modified version becomes tolerant to both problems by encoding documents into string vectors in which the two problems can never be inherent.

4.3 Neural Text Categorizer

This section is concerned with architecture, learning algorithm, and classification of the proposed neural network called NTC. Documents are encoded into string vectors for using NTC. In the neural network, the special layer, called the learning layer, exists. With respect to the architecture of NTC, the input nodes are directly connected to the output nodes, and the weights between the input and output nodes are decided by the learning nodes.

The architecture of NTC is illustrated in Figure 1. The input layer receives an input vector given as a string vector and the number of nodes in the layer is consistent with the dimension of the string vector. The output layer generates categorical scores, which indicates the likelihood of the given input vector to each category, and the number of nodes in the layer is consistent with the number of the predefined categories or classes. The learning layer decides the weights between the input and output layers differently depending on the given input vector, and the number of nodes in the layer is also consistent with the number of categories or classes. Therefore, with respect to its architecture, NTC has the three layers as shown in Figure 1.

In context of the learning process, the first step of NTC is to initialize the weights between the input and output layers. Let us assume that NTC is applied to text categorization without decomposing the task into binary classification tasks. A set of the training string vectors is partitioned category-by-category. Each learning node has its own table, which consists of words and their weights. Frequencies of elements of string vectors within each category are assigned in the table as the initial weights.

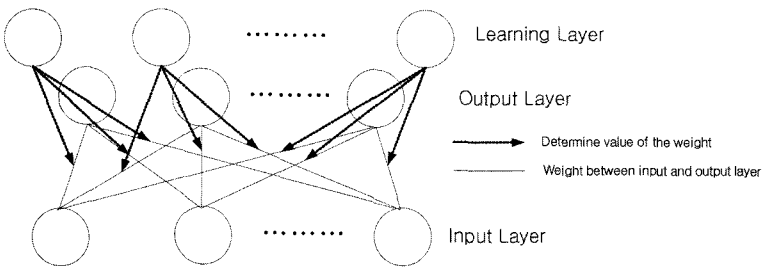

Figure 1. Overall Architecture of the NTC.

Therefore, the initial step is to set up the tables in learning nodes.

The learning process of the NTC refers to the process of optimizing the weights in the tables of the learning nodes. Each training example is classified by summing the initial weights and selecting the category corresponding to the maximal sum. If the training example is classified correctly, the weights are not updated. Otherwise, the weights are incremented toward the target category and are decremented toward the classified category. The optimized weights are generated as an output of this process.

In NTC, each example is classified by summing the optimized weights, whether it is a training or unseen example. Each output node generates the summation of weights connected to itself from the input nodes as its categorical score. The weights are decided by referring the table, which is owned by its corresponding learning node. The category corresponding to the output node, which generates its maximum categorical score, is decided as the category of the given example. Therefore, the output of this process is one of the predefined categories, assuming that NTC is applied to text categorization without the decomposition.

The property which characterizes NTC exists. It is that the learning layer exists inherently in NTC, and it has its own table as the reference for deciding weights, which consists of words and their weights. Each input node receives a string as an element of a string vector, and learning nodes decide the weights connected from the input node by referring to their own tables. In the current version of NTC, if the word is not registered in a table, its weight is assigned as zero. The issue in this case will be considered in further research.

5. COMPARISON OF STRING VECTORS AND NUMERICAL VECTORS

We will compare the above representations of documents in three sets of experiments. In Section 5.1, we compare two versions, the string vector-based version and the numerical vector-based version of KNN with each other. In Section 5.2, we compare the two kernel functions, the inner product of two numerical vectors, and the average semantic similarity of two string vectors in using SVMs for a task of text categorization. In Section 5.3, we compare neural network, called NTC, with other traditional machine-learning algorithms. The goal of these sets of experiments is to compare the performance of string vector-based approaches and numerical vector-based approaches to text categorization.

5.1 Numerical Vector-based Version vs. String Vector-based Version of KNN

This subsection is concerned with a set of experiments for comparing the two versions of the KNN. Reuter21578, which is known as the standard test bed for evaluating approaches to text categorization, is used in this set of experiments. Documents are encoded into string vectors and numerical vectors for two versions. It may be interpreted that within the KNN, two types of representations of documents are compared with each other. The goal of this set of experiments is to observe the performance of the two versions of the KNN [Sebastiani 2002].

In this set of experiments, the collection of news articles named °×Reuter21578°± is used as the test bed. The partition of the collection into the training set and the test set is illustrated in Table V. Actually; there are more than 100 categories in the test

Table V. Training and Test Sets in the Test Bed: Reuter21578

Category Name	Training Set	Test Set	Total
Acq	1452	672	2124
Corn	152	57	209
Crude	328	203	531
Earn	2536	954	3490
Grain	361	162	523
Interest	296	135	431
Money-Fx	553	246	799
Ship	176	87	263
Ship	335	160	495
Wheat	173	76	249
Total	6362	2752	9114

bed. However, only ten most frequent categories are involved in this set of experiments. Other literature also use only ten categories as well for this research.

The number of nearest neighbors given as the parameter of the KNN is fixed to three. In the traditional version, documents are encoded into 100-, 250-, and 500-dimensional numerical vectors[1]. In the proposed version, they are encoded into 10-, 25-, 50-dimensional string vectors. In order to present the robustness of the proposed version with the smaller sized input data, the input size of the proposed version is one-tenth of the traditional version.

In this set of experiments, words with their highest total frequencies are selected as features. The given text classification is decomposed into binary classification tasks in as many categories as possible; each binary classification corresponds to each category. In each binary classification task, training documents belonging to the corresponding category are labeled with the positive class, and some of the others, which do not belong to the corresponding category, are selected at random and are labeled with the negative class. The collection of the training documents, which are relabeled so, are indexed into a collection of words called feature candidates, among which words with their highest frequencies are selected as features. In this set of experiments, only frequency-based feature selection is used, and we will consider other advanced schemes for selecting features in our next research.

The results from this set of experiments are presented in Figure 2. In the left side of the figure, microaveraged F1 measures are given. In the right side, macroaveraged F1 measures are provided. With respect to F1 measure, both versions are similar to each other, but with respect to macroaveraged F1 measure, the proposed version is slightly better. Therefore, through this set of experiments, we can conclude that the

[1]In previous literatures, documents were usually encoded into approximately 300-dimensional numerical vectors, so the dimensions presented in this set of experiments are based on facts [Sebastiani 2002].

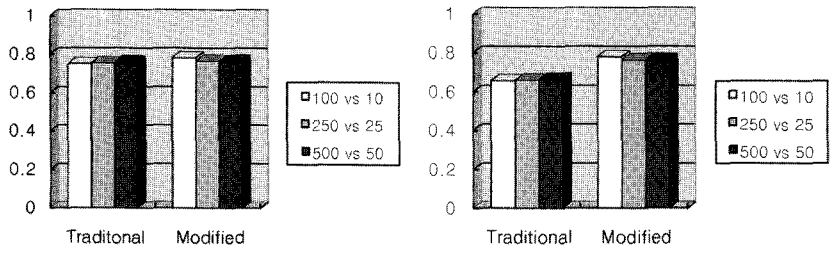

Figure 2. The Results from Comparing the Two Versions of KNN.

proposed version is robust with only smaller-sized input data, and more tolerant to small-sized categories.

5.2 Numerical vector vs. String Vector in SVM

This section is concerned with a set of experiments for comparing two kernel functions of SVM in context of the text categorization. The standard collection of news articles, called Reuter21568, is used as the test bed. Like the previous set of experiments, the documents are encoded into high-dimensional numerical vectors and low-dimensional ones. In the traditional version of SVM, the inner product is used as its kernel function, whereas the average semantic similarity is used as the kernel function in the proposed version. The goal of this set of experiments is to observe the text categorization performances of the two kernel functions of the SVM.

The results of this set of experiments are illustrated in Figure 3. Like Figure 1, the left side shows the results with respect to microaveraged F1 measure, whereas the right side shows those with respect to macroaveraged F1 measure. As shown in Figure 3, the proposed version is better than the traditional version with respect to both F1 measures. Accordingly, because of the better results evaluated by the microaveraged F1 measure, the proposed version is more robust than the traditional version. Based on the better results with respect to the macroaveraged F1 measure, the proposed version is more tolerant to the sparse category where only a few documents are available.

5.3 NTC vs. Traditional Machine-Learning Algorithms

This subsection is concerned with a set of experiments for validating empirically the performance of the string vector-based neural network. In the previous sections, two

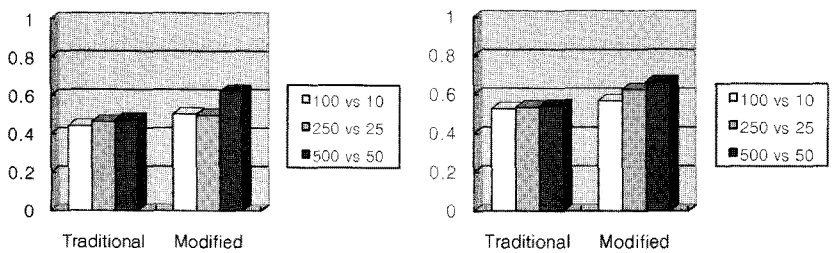

Figure 3. The Results from Comparing the Two Kernel Functions of SVM.

Table VI. The Most Frequent Categories in Reuter21378.

Category Name	#Training	#Test	#Total
Earn	2536	954	3490
Acq	1452	672	2114
Money-Fx	553	246	799
Crude	328	203	531
Grain	361	162	523
Trade	335	160	495
Interest	296	135	431
Ship	176	87	263
Wheat	173	76	249
Corn	152	57	209

kinds of representations of documents are compared with each other within using a particular machine-learning algorithm. In this set of experiments, the neural network, called NTC, is compared with the several traditional and popular approaches to text categorization. Documents are also encoded into the two kinds of representations for this set of experiments.

Table VI illustrates the predefined categories and the number of news articles per category in the test bed, Reuter 21578. The ten most frequent categories are selected among more than 100 categories in this set of experiments. As illustrated in Table VI, the collection of news articles is partitioned into two sets: the training and the test set. The selection of ten most frequent categories and the partition are subject to the literature [Sebastiani 2002]. The Reuter 21578 is popularly used as a standard test bed for evaluating approaches to text categorization [Sebastiani 2002].

The configurations of the approaches are illustrated in Table VII. The parameters of SVM and KNN, the capacity and the number of nearest neighbors are set as four and three, respectively, but the NB has no parameter. The parameters of the NNBP such as the number of hidden nodes and the learning rate are arbitrarily set as shown

Table VII. The Configurations of Participating Approaches.

Approaches to Text Categorization	Parameter Settings
SVM	Capacity = 4.0
KNN	#nearest number = 3
Naive Bayes	N/A
NNBP with Back Propagation	Hidden Layer: 10 hidden nodes Learning rate: 0.3 #Training Epochs: 1000
NTC	Learning rate: 0.3 #Training Epochs: 100

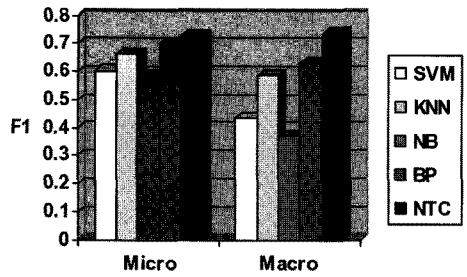

Figure 4. The Results of This Set of Experiments.

in Table VII. News articles are encoded into 500-dimensional numerical vectors and 50-dimensional string vectors. Therefore, the configurations of the involved approaches are set as shown in Table VII.

The results of comparing the involved approach with each other are presented in Figure 4. Among the five bars, the black bar indicates the performance of the proposed approach. The left group indicates microaveraged F1 measure of five approaches. The right group does macroaveraged F1 measure. The proposed approach, NTC, shows its best performance among the five approaches.

Let us discuss the comparison of the NTC with the traditional approaches in this set of experiments. The four typical machine-learning algorithms, NB, traditional version of SVM, traditional version of KNN, and MLP with back propagation. As shown in the figure, NTC outperforms the four approaches. However, NTC is very slightly better than MLP with back propagation. Nevertheless, NTC is a more recommendable approach to text categorization with respect to both performance and learning speed, compared with MLP.

6. CONCLUSIONS

In this paper, we proposed new representations of documents in favor of the text categorization. It was discovered that the string vector-based version is better in using KNN. In addition, the string vector-based kernel function is better than the numerical vector-based one in using SVM. NTC worked significantly better than NB, a traditional version of KNN, and a traditional version of SVM, but slightly better than MLP. Finally, we conclude that it is more recommendable to encode documents into string vectors than into numerical vectors.

REFERENCES

COVER, T. AND P. HART. 1967. Nearest Neighbor Pattern Classification. *IEEE Transaction on Information Theory, 1*, 13, 21–27.

CRISTIANINI, N. AND SHAWE-TAYLOR, J. 2000. Support Vector Machines and Other Kernel-based Learning Methods. *Cambridge University Press.*

DRUCKER, H., D. WU, AND V. N. VAPNIK. 1999. Support Vector Machines for Spam Categorization. *IEEE Transaction on Neural Networks, 10*, 5, 1048–1054.

EYHERAMENDY, A., D. LEWIS, AND D. MADIGAN. 2003. On the Naive Bayes Model for Text Categorization. *In Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, 165–171.

HEARST, M. 1998. Support Vector Machines. *IEEE Intelligent Systems 13* 4, 18–28.

JO, T. AND D. CHO. 2007. Index Based Approach for Text Categorization. *International Journal of Mathematics and Computers in Simulation, 2*, 1, 127–132.

JOACHIMS, T. 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. *In Proceedings of 10th European Conference on Machine Learning*, 143–151.

KONONENKO, I. 1989. Id3, Sequential Bayes, Naive Bayes and Bayesian Neural Networks. *In Proceedings of 4th European Working Session on Learning*, 91–98.

LODHI, H., C. SAUNDERS, J. SAHWE-TAYLOR, N. CRISTIANINI, AND C. WATKINS. 2002. Text classification with string kernels. *Journal of Machine Learning Research 2*, 2, 419–444.

MASAND, B., L. GORDON, AND D. WALTS. 1992. Classifying News Stories using Memory Based Reasoning. *In Proceedings of 15th ACM International Conference on Research and Development in Information Retrieval*, 59–65.

MITCHELL, T. M. 1997. *Machine Learning.* McGraw-Hill.

MLADENIC, D. AND GROBELINK, M. 1999. Feature Selection for Unbalanced Class Distribution and Naive Bayes. *In Proceedings of International Conference on Machine Learning*, 256–267.

RUMELHART, D. E. AND J. L. MCCLELLANDC. 1986. A General Framework for Parallel Distributed Processing. In Rumelhard, D.E., Parallel Distributed Processing. *MIT Press.*

RUIZ, M. AND P. SRINIVASAN. 2002. Hierarchical Text Categorization using Neural Networks. *Information Retrieval, 5*, 1, 87–118.

SEBASTIANI, F. 2002. Machine Learning in Automated Text Categorization. *ACM Computing Survey* 34, 1–47.

WIENER, E. D. 1995. A Neural Network Approach to Topic Spotting in Text. Thesis of Master of University of Colorado.

YANG, Y. 1999. An Evaluation of Statistical Approaches to Text Categorization. *Information Retrieval* 67–88.

Taeho Jo received PhD degree from University of Ottawa in 2006, Master degree from POSTECH (Pohang Institute of Science and Technology) in 1997, and Bachelor from Korea University in 1994. Currently, he works for the School of Computer and Information Engineering at Inha University as a Professor. Previously, he had worked for Samsung SDS, ETRI (Electronic Telecommunication Research Insitute), KISTI (Korea Insitute of Science and Technology Information), Nstein Technologies, BK21 Chonbuk National University, and IT Convergence Insitute for KAIST. He has published and submitted more than 100 research papers since 1996. His research interests are text mining, neural networks, machine learning, and information retrieval.