Evolving Recurrent Linear-GP for Document Classification and Word Tracking

Xiao Luo, A. Nur Zincir-Heywood, *Member, IEEE*

***Abstract*— In this paper, we propose a novel document classification system where the recurrent linear Genetic Programming is employed to classify the documents that are represented in encoded word sequences. During this process, word sequences of documents are tracked, frequent patterns are detected and document is classified. We describe the word encoding model and the recurrent linear Genetic Programming based classification mechanism. The performance results on benchmark data set Reuters 21578 show that this system can analyze the temporal sequence patterns of a document and get competitive performance on classification. We expect that it can be easily applied to other application areas, where the temporal sequences are very significant.**

I. Introduction

Automatic text document analysis and management such as classification is more and more important to the information management area. Text representation and analysis are the two necessary and important steps for this purpose. The most widely used technique to represent a text document is so called 'bag of words', which considers only term frequencies or word occurrences in documents or categories and therefore ignores the significance of the sequences in which they occur. However, understanding and classifying a document can change based on different paragraphs or even sentences. Some of these changes come from the temporal sequence of words learned through the document. Thus, analyzing the temporal sequence of words becomes very important and unique for the context tracking and classification.

In this paper, we describe a text classification system where the recurrent linear Genetic Programming is employed to analyze and classify the documents that are represented in encoded word sequences. We consider the temporal word sequence of a document as words occur one after another in time throughout the whole document. In this work, word sequence tracking is only based on the order of words without considering the semantics. The core of the approach is to employ machine-learning techniques to automate the identification of temporal sequences within a document. This in return automates the process of word tracking, frequent word co-occurrence mining and document classification. There are two major parts of the proposed system. The first is the new architecture for document representation. To this end, the hierarchical Self-Organizing Map (SOM) encoding architecture is used to first encode the characters, then words. The similarities between words are discovered by the topology of the SOM, and then the whole document is represented by the neurons of the SOM [14]. The second is the classifier to maximally utilize the encoded word sequence information for word tracking, and classification. Recurrent page-based Linear Genetic Programming (RLGP) is used for this part. There are two advantages with RLGP. First, the registers in the recurrent mechanism are used to remember and analyze the word sequences. Second, Genetic Programming creates classification rules to track the word sequence and to classify the document. The overall performance results on data set Reuters 21578, which has been a standard benchmark for the text classification tasks throughout the last ten years, show that the whole system can analyze the temporal sequence patterns of a document and get competitive performance on classification.

The rest of this paper is organized as follows. Section 2 presents the related work and methodologies on the text data representation, analysis and classification; Section 3 gives an overview of the proposed system. Section 4 describes the document preprocessing and encoding techniques. The RLGP algorithm applied to the document classification problem is given in Section 5. In Section 6, results from our experimental study are presented. Section 7 draws the conclusions and discusses the future work.

II. Related Work

In the automatic text classification area, a text document needs to be uniformly represented before directly interpreted by an analysis or classification algorithm. The conventional representation is 'bag of words' where a document is represented by a vector of words. However, with this representation, all word order information is lost. Moreover, word combinations, co-occurrences or multi-word units can not be identified since the method is only based on frequencies. Thus, other representations of the documents are investigated. In particular, those representations are based on the phrases in addition to individual words. Two kinds of phrase investigation approaches exist. One is based on the syntactical phrase where the phrase is defined according to a grammar of the language [13]. The other is based on the statistical phrase where the phrase is not grammatically such, but is composed of a set/sequence of words whose patterns of contiguous occurrence in the collection are statistically significant. An example of this representation is N-gram representation [3] [10]. Although the above methods inject positional information using phrases or multi-word units or local co-occurrence statistics, none of the representations

Xiao Luo is with the Faculty of Computer Science, Dalhousie University, Halifax, NS B3H1W5, CANADA (phone: 902-425-6669; fax: 902-492-1517; email: luo@cs.dal.ca).

A. Nur Zincir-Heywood is with the Faculty of Computer Science, Dalhousie University, Halifax, NS B3H1W5, CANADA (phone: 902-494-3157; fax: 902-492-1517; email: zincir@cs.dal.ca).

above considers the significant sequences of words or phrases within the whole document.

In the area of automatic text analysis and classification, many techniques such as decision trees, Naïve Bayesian, k-Nearest Neighbor, and Support Vector Machines [19] have been researched. The tree based GP classifier was also utilized for this task in 2005 [10]. In that work, authors used N-grams as the input to their GP classifier. Several designed GP functions were used to operate on the N-grams to generate classification rules. On the other hand, in the late 90's, some word sequence analysis techniques have been introduced, such as the recurrent neural network and the word-sequence kernel based Support Vector Machine. Farkas [7] and Wermter et al. [21] introduced the recurrent neural network for text classification in 1995 and 1999 respectively. In their system, a word is represented by a vector, which includes the information of the word to each category. This could mislead the classification process according to the category sequences instead of the actual word sequences. In 2003, Nicola Cancedda et al. [2] invented the word-sequence kernel based Support Vector Machine. Using this kernel, the similarity is assessed by the number of (possibly non-contiguous) matching sub-sequences shared by two word sequences. In that work, there is a predefined length for the sub-sequences. Thus, the classifier does not work on the word sequence of a document as a whole where the length is dynamic.

III. Overview of the System

As discussed above, in this work, we designed and developed a text classification system and evaluated it on the Reuters 21578 news collection(over 10000 documents). The proposed system is composed of two major components:

- Document Encoding
- Document Classification

In the document encoding component, we make use of the ability of the unsupervised learning system - Self-Organizing Map - to provide approximations of a high dimensional input into a lower dimensional space [16]. Thus, the Self-Organizing Map acts as an encoder to represent a larger input space as finding a smaller set of prototypes. The results of this encoding are then input to the GP classifier. We developed binary classifiers for each category, so the classification rules were evolved for each category. Divide and conquer approach is employed in the process of classification. We separate the training and test set into four bins. Each bin contains documents of different length regarding to the number of words. Four GP classifiers are evolved, one for each bin. Figure 1 gives an overview of the system.

Hierarchical document classification has been researched previously [5] [11] [18]. By utilizing a known hierarchical structure, the classification problem can be decomposed into a set of smaller problems corresponding to hierarchical splits in the tree. In order to increase the efficiency and accuracy, we employed a hierarchical classification architecture as well. Part of the architecture is as shown in Figure 2. Each

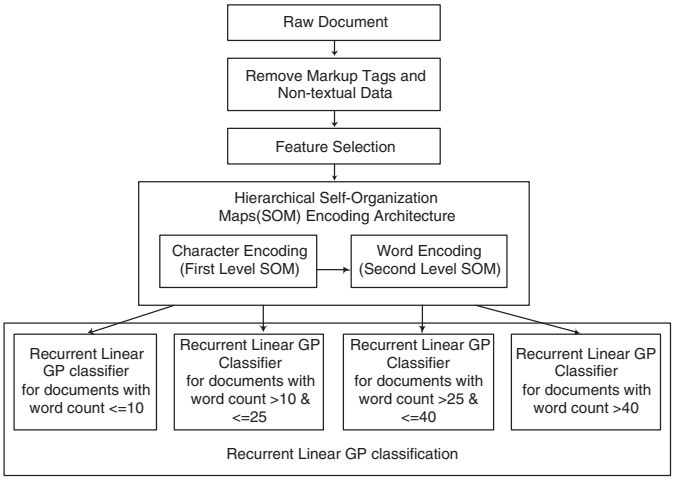

Fig. 1. An overview of the system

layer classifier takes the in class and out class documents of the current biggest class and separate them. If there is a class that is a subclass of its parent class, we train a classifier to separate them within the parent class.

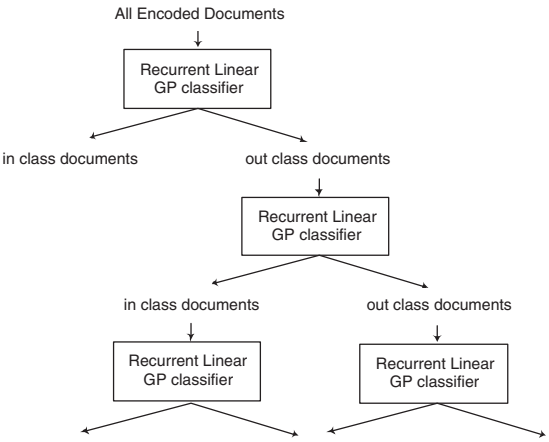

Fig. 2. Hierarchical classification

IV. Feature Selection and Document Encoding

A. Feature Selection with Mutual Information

For reasons of both efficiency and efficacy, different feature selection techniques are widely used in document analysis and management, mutual information has been used in several text classification experiments [6] [1]. We used it as a feature selection method in our experiments as well. The mutual information, $MI(w_i, C)$, of a feature w_i (in this case, a word) corresponding to a category C is computed by using Equation 1, given below.

$$MI(w_i, C) = \sum_{\{\overline{w_i}, w_i\}} \sum_{\{\overline{C}, C\}} P(w_i, C) \log \frac{P(w_i, C)}{P(w_i)P(C)} \quad (1)$$

We select the 300 features (words) for each category with largest mutual information value.

1) Hierarchical Self-Organizing Map Encoding(SOM) Architecture: As discussed previously, we employed a hierarchial Self-Organizing Map architecture for the process of encoding.

The Self-Organizing Map is a vector quantization method with topology preservation with the dimension of the map matching the true dimension of the input space. In brief, topology preservation means that input patterns close in the input space are mapped to the units close on the Self-Organizing Map lattice. We took the advantage of this characteristic of Self-Organizing Map, and designed a two-level hierarchical Self-Organizing Map architecture for the process of character and word encoding. The first level of the Self-Organizing Map hierarchy is employed to discover the patterns of characters and the second level is employed to discover the patterns of words. The hierarchical nature of the model is shown in Figure 3.

1) Input for the First-Level SOMs: The assumption at this level is that the SOM forms a codebook for the patterns in characters that occur in a specific document category. In order to train an SOM to recognize patterns in characters, the document data must be formatted in such a way as to distinguish characters and highlight the relationships between them. Characters can easily be represented by their ASCII representations. However, for simplicity, we enumerated them by the numbers 1 to 26, i.e. no differentiation between upper and lower case. The relationships between characters are represented by a character's position, or time index, in a word. For example, in the word "*cost*": "*c*" appears at time index 1, "*o*" appears at time index 2, "*s*" appears at time index 3, etc. It should be noted that it is important to repeat these words as many times as they occur in the documents, so that the neurons on the SOM will be more excited by those frequent words and their information will be more accurately encoded. The overall pre-processing process for the first-level SOM is therefore:

- Convert the word's characters to numerical representations between 1 and 26.
- Give the time index to the characters in a word. It is the actual time index plus 2, except the first character in the word.

The indices of the characters is altered in this way so that when the list is input to an SOM, both data features (enumerated characters and indices) are spread out over a similar range. There is no bias on either of the features. The inputs to the first level SOM are all the characters in the training set. In our experiments, the size of the first-level SOM is 7 by 13.

2) Input for the Second-Level SOMs: The assumption at this level is that the SOM forms a codebook for the patterns in words that occur in a specific document category. When a character and its index are run through a trained first-level SOM, the closest neurons (in the Euclidian sense), or *Best Matching Units* (*BMU*s), are used to represent the input space. A two-step process is used to create a vector for each word, *k*, which is input to the first-level SOM of each document:

- Form a vector of size equal to the number of neurons (r) in the first-level SOM, where each entry of the vector corresponds to a neuron on the first-level SOM, and initialize each entry of the vector to 0.
- For each character of a word,
  - Observe which neurons $n_1, n_2, \ldots, n_r$ are affected the most.
  - Increase entries in the vector corresponding to the 3 most affected *BMU*s by $1/j, 1 \leq j \leq 3$.

Hence, each vector represents a word through the sum of its characters. The result given by the second-level SOM is encoding of words. It is worth to mention that we trained different second level SOMs for each document category, where the size of each second-level SOM is 6 by 6.

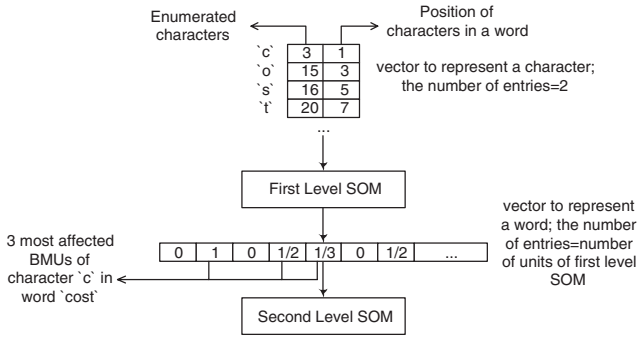

Fig. 3. An overview of the hierarchical SOMs encoding architecture

2) Document Representation : We train the second level SOMs based on words from all in class documents. The hypothesis here is that it captures the characteristic distribution and relationships of words of the in class documents. Each word space(in this case, input to the SOM) has a *BMU* neuron. A document is then represented by neurons instead of words. After training the second level SOM, we noticed that some of the neurons have greater number of hits than the others(An input hits a neuron if the neuron is the *BMU* for the input). For the efficiency of the training process of RLGP classification, we choose a set of neurons with greater number of hits to construct the document representation. The threshold here is that each in class document has at least one word hitting the chosen neurons. Recall the training nature of the SOM, each neuron is the cluster center of the inputs hit, so the weight of a neuron is the mean of inputs hit. We normalized the Euclidian distance between the word w_i and the corresponding *BMU* neuron N by using Equation 2

$$G(w_i, N) = \frac{1}{\sigma\sqrt{2\Pi}} e^{\frac{-(w_i - n)^2}{(2\sigma^2)}} \tag{2}$$

Where:

w_i : A word that is represented in the format of input into SOM as shown in Figure 3

n : Weight of the neuron

σ^2: Variance of words that hit neuron N

We select the words that hit one of the chosen neurons N if $G(w_i, N) \geq min(G(w_{1...NumOfTotalWordsHitN}, N))$to construct the document representation. Hence, each in class and out class document is represented by a two-dimensional vector. The first dimension is the normalized distance to the *BMU* of the word, and the second dimension is the normalized index of the *BMU* of the word as shown in Figure 4.

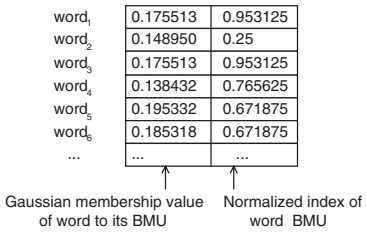

Fig. 4. An example of an encoded document representation

V. Recurrent Linear Genetic Programming

A. Linear Genetic Programming

Linearly Structured Genetic Programming is based on a representation closely related to that employed by Genetic Algorithms. Specifically, individuals are constructed from a (linear) sequence of integers each of which has to be decoded into a valid instruction (syntactic closure). The decoding process effectively translates each integer into an equivalent binary string, separates the string into a series of fields based on the addressing mode and maps each field into a valid value. Typical fields include mode, opcode, source and destination, Figure 5. The mode bit distinguishes between different instruction types, for example instructions detailing a constant or an operation performed on a register or on an input. The source and destination fields detail specific registers or input ports. Programs now take the form of a register level transfer language in which all operations operate on general purpose registers or read values from input ports (features from the current example). In this work a 2-address instruction format is employed e.g.,$R1 = R1 + IP3$, where R1 denotes a register with index '1' and IP3 is a reference to the 3rd feature of the current pattern.

As with the case of Tree structured GP many instances of Linear GP (L-GP) have been developed over a considerable period of time [8], [4], [17], [12]. The specific form of Linearly structured GP (L-GP) employed by this work utilizes the page-based L-GP developed in an earlier work [9]. Such a scheme enforces a fixed length representation (crossover only exchanges an equal number of instructions), the basic components of which are defined as follows.

- *Representation*: Individuals take the form of a 2-address instruction format, Figure 5. Individuals are described in terms of a (uniform) randomly selected number of pages, where each page has the same fixed number of instructions.
- *Initialization*: Individuals are described in terms of the number of pages and instructions, where instructions are selected from a valid set of integers denoting the instruction set. The number of pages per individual is determined through uniform selection over the interval [1, ..., maxPages]. That is to say the initial population is initialized over the entire range of program lengths. Defining an instruction is a two-stage process in which the mode bit is first defined (instruction type) using a roulette wheel (user specifies the proportions of the three instruction types). Secondly the content of the remaining fields is completed with uniform probability. Such a scheme is necessary in order to avoid half of the instructions denoting constants i.e. effect of the mode field, Figure 5.
- *Selection Operators*: A steady-state tournament is employed. In this case all such tournaments are conducted with 4 individuals selected from the population with uniform probability. The two fittest individuals are retained and reproduce. The children over-write the worst two individuals from the same tournament using their respective position in the population.
- *Variation Operators*: Three variation operators are utilized, each with a corresponding probability of application, where such tests are applied additively (i.e., the resulting children might be the result of all three variation operators). Crossover selects one page from each offspring and swaps them. The pages need not be aligned, but always consist of the same number of instructions. Mutation has two forms. The first case - hereafter referred to as 'Mutation' - merely Ex-OR's an instruction with a new instruction. No benefits were observed in making such a mutation operator 'field specific', where this is undoubtedly a factor of the addressing format [9]. The second mutation operator - hereafter denoted 'Swap' - identifies two instructions with uniform probability in the same individual and interchanges them. The basic motivation being that an individual might possess the correct instruction mix, but have the instruction order incorrect.

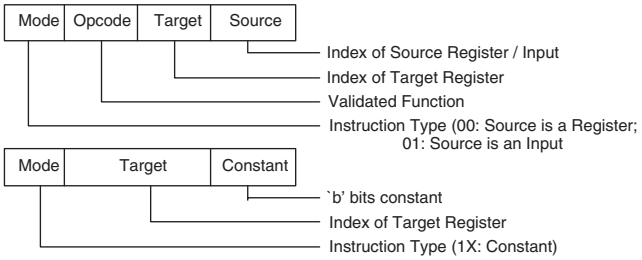

Fig. 5. Instruction Format (2-register addressing)

This represents the basic page-based L-GP scheme. However, the selection of page size is problem specific. As a consequence the Dynamic Page based L-GP algorithm was introduced to modify the number of instructions per page

dynamically during the course of the training cycle [9]. In this case the user merely defines the maximum page size as an order of 2. The page size is then doubled for each plateau in the fitness function, beginning with a page size of 1 and finishing at 'max page size' and returning to a page size of 1 once a plateau following 'max page size' is encountered. Plateaus are defined in terms of consecutive non-overlapping windows of 10 tournaments. For each of the 10 tournaments the (tournament's) best-case individual's fitness is summed. If the total over both windows is the same then a plateau is 'defined'. Such a scheme was demonstrated to be much more robust than that of a fixed page size over a range of benchmark problems (2 boxes, 6 parity, UCI classification benchmarks) [9]. In this work, we employed the dynamic page-based L-GP.

B. Recurrent L-GP

Recurrent L-GP(RLGP) is based on the standard page based L-GP, the only modification necessary to change a standard page based L-GP model into a recurrent model is to retain register values between sequential pattern presentations. Thus within the context of a prediction problem, the registers are never reset. The prediction is read from the predefined 'output' register(s) [15].

In the case of document classification problem, the objective is to predict the class label for each document. Since a document can be represented by an encoded word sequence, the document classification can be treated as a word sequence label prediction problem. Encoded words of a document are input into RLGP one after another, registers are never reset until the last word is input into RLGP. The prediction of the class label for a document is then read from the predefined 'output' register after the last word is input.

C. Dynamic Subset Selection (DSS) – Increase the efficiency of training RLGP

Since we are applying RLGP to the document classification problem, Reuters 21578 data set which is a large well known benchmarking data set is selected for our experiments. In order to efficiently train RLGPs on this large data set, we employed the dynamic subset selection algorithm. The basic approach of this algorithm is based on the concept of selecting a subset of the training data instead of using all the training data [20]. The DSS algorithm consists of two basic stages, Figure 6:

- Selecting a subset of the training patterns from all of the training patterns - step 2.a.iii, Figure 6. Subset selection is performed in proportion to a predefined ratio of age and difficulty (P(age) in Figure 6) and the relative age and difficulty of patterns with respect to each other. The implication being that two 'roulette wheels' exists, one for age and one for difficulty. Age is updated after the content of a subset is refreshed - step 2.a.iv, Figure 6. Pattern difficulty is updated with respect to each GP individual participating in an evolutionary phase using a suitable distance metric. In the case of classification problems, this reflects whether the training pattern was correctly classified or not, if pattern was correctly classified the difficulty increases by 0 otherwise increases by 1 - step 2.b.vi, Figure 6.
- Training the learning algorithm on a subset - step 2.b, Figure 6. At this point only a subset of the whole training data is used over which an 'evolutionary phase' takes place. The number of such evolutionary phases performed is defined by the number of DSS iterations - step 2, Figure 6. The whole training data used to evaluate the fitness of the individual at the frequency of "j mod (*DSSRefeshFrequency*)" - stage 2.c, Figure 6. In other words, we evaluate the fitness of the best individual from the 'evolutionary phase' on the whole training data when we change the subset. This is termed as the (DSS) refresh frequency. A refresh frequency of 1 naturally implies that the subset content changes for every evolutionary phase. In this work, the refresh frequency is set at 10% of the population size (or every 6 'evolutionary phase');

TABLE I
DSS PARAMETERS

Parameter	Value
DSSRefreshFrequency	6
P(age)	0.3
subset-size	20% of whole training data

```
1. TotalError=MaxError;
2. FOR j=0; ((j<Generations) ||(TotalError<=0)); j++
        2.a IF (j mod (DSSRefreshFrequency) == 0) THEN
                2.a.i SubsetError=subsetSize+1;
        2.a.ii Update Age and Difficulty 'roulette Wheel'
                2.a.iii FOR k=0; k<subsetSize; k++
                        1. IF rand < P(age)
                           THEN Subset(k)=pattern with large age
                           ELSE Subset(k)=pattern with large difficulty
                2.a.iv pattern  Subset, pattern.age++;
        2.b Perform an 'evolutionary phase' over patterns defined by PatternSubset;
                2.b.i Choose individuals with uniform probability from population;
                2.b.ii Evaluate fitness of individuals over Subset;
                2.b.iii IF (Error(TourBestIndividual,Subset)<SubsetError)
                        THEN BestIndividual=TourBestIndividual;
                             SubsetError=Error(TourBestIndividual,Subset);
                2.b.iv Apply variation operators to the selected individuals;
                2.b v Update population with children of selected individuals;
                2.b.vi  pattern.difficulty = pattern.difficulty+ 0/1;
        2.c IF  (j mod (DSSRefreshFrequency) == 0)  THEN
                  IF (Error(BestIndividual, TrainingSet)<TotalError) THEN
                     FinalBestIndividual=BestIndividual;
```

Fig. 6. Algorithm of DSS (modified from [18])

The DSS parameters used in our experiments are summarized in Table I.

D. Inputs and outputs of RLGP

The number of inputs to GP is the number of values used to represent the words. In this case, a word is represented by a two-dimensional vector as shown in Figure 4, so the number of inputs to GP is 2. The output of the GP at the end of a document is the classification label of the input document. We can track the 'output' register value after each input word. Binary classifiers are built up for each category. The RLGP classifier classifies documents to in class or out class. Hence,

TABLE II
GP PARAMETERS

Parameter	Value
Selection type	Tournament
Tournament size	4
Functional Set	+, -, ×, ÷
Instruction Type and Ratio	Constants (0), Internal(4), External(1)
Node Limit	256
Population Size	125
Generations	48000
Number of Register	8
P(Xover)	0.9
P(Mutate)	0.5
P(Swap)	0.9

the output of GP is one value corresponding to the label. In the actual experiments, we assign value 1 as the correct label to the in class documents and value -1 as the correct label to the out class documents. The real 'output' register value was projected into the data range [-1,1] by using the following Equation 3:

$$GPoutNew = \frac{2}{1 + e^{-GPout}} - 1 \tag{3}$$

E. Fitness function and parameters of RLGP

GP sets the task of forming classification rule by using a set of instructions operating on registers and inputs with operators +, -, × and ÷. The classification rule is then evaluated against the documents in the training set. Each rule can be tested against any document and will return a value indicating whether the rule is true for that document or not.

A classification rule must be evolved for each category C on each bin (Figure 1). When evolving a rule for a particular set, the fitness depends on the number of documents classified to the correct category or not. In this work, we simply used the sum square error of all in class and out class documents as fitness function shown in Equation 4:

$$fitness = \sum_{p=1}^{P} (CorrectLabel(p) - GPout(p))^2 \tag{4}$$

Where:

P is the number of total examples in the training set

Finally, the GP parameters used in our experiments are summarized in Table II

VI. EXPERIMENTS AND RESULTS

In this work, the experiments involve documents from the Reuters 21578 collection, which has been a standard benchmark for the text classification tasks throughout the last ten years. There are a total of 12612 news stories in this collection. These stories are in English, where 9603 of them are in the training set, and 3299 are in the test set. We used the top 10 categories which are widely used in the area of document analysis and classification.

There are two main objectives in our experiments:

- To evolve effective classifiers to classify the documents in the data set
- To track the word sequence in order to detect the context changing within a document

A classification rule for each category and each bin was evolved using 10 different GP initializations. We selected the best rule for all cases. Because of the space limitations, we can not show all the best rules produced by RLGP, however, an example of a rule produced by a RLGP evolving a classifier for category 'Earn' of the Reuters 21578 shown as follows:

Register1=Register1-Input1;
Register0=Register0×Input1;
Register1=Register1-Input1;
Register0=Register0+Input1;
Register1=Register1-Input1;
Register0=Register0-Register1;
Register0=Register0-Input0;
Register1=Register1-Input1;
Register0=Register0-Register1;
Register0=Register0-Register1;
Register0=Register0-Input0;
Register0=Register0÷Input1;
Register0=Register0-Input0;
Register0=Register0+Input1;
Register1=Register1÷Input1;

In this rule, only two registers among eight is utilized. The predefined 'output' register of GP is Register0. Given a document (in the following example, there are 19 words left after feature selection), Figure 7 shows the change of the input words (one after another into GP) and the corresponding change of the 'output' register value of GP. Recall that value '1' is the in class label, and '-1' is the out class label. When the 'output' register value reduces, it implies that the context is moving towards the out class, while when the 'output' register value increases, it implies that the context is moving towards the in class.

In information retrieval and text classification, the classical effectiveness measurements are Recall(R), Precision(P) and F1-measure(F1), which are given by Equations 5 , 6 and 7 respectively.

$$R = \frac{TP}{TP + FN} \tag{5}$$

$$P = \frac{TP}{TP + FP} \tag{6}$$

$$F1 = \frac{2 \times R \times P}{R + P} \tag{7}$$

Where:

TP:In class documents, classified to be in class.

FN:In class documents, classified to be out class.

FP:Out class documents, classified to be in class.

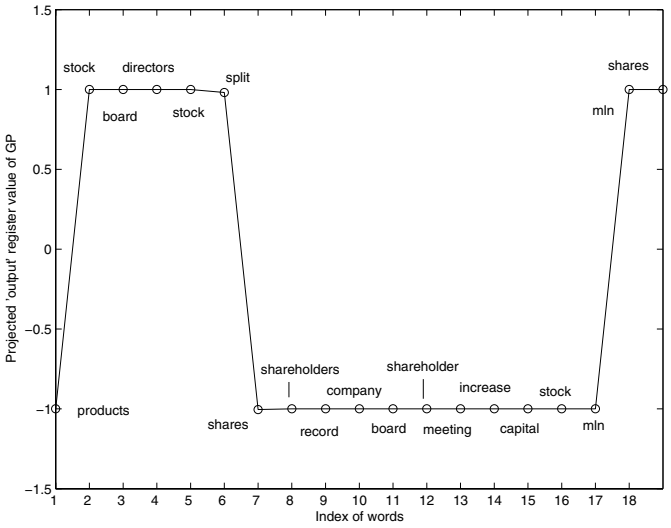

Fig. 7. Classification label changes as the input word changes

TABLE III
F1-MEASURE FOR TOP 10 CATEGORIES OF OUR SYSTEM AND THE SYSTEM DESCRIBED IN PAPER [10]

Category	F1 of our system	F1 of system in [10]
Earn	0.9506	0.857
Acq	0.7766	0.755
Money	0.6917	0.612
Grain	0.8927	0.550
Crude	0.9196	0.826
Trade	0.9005	0.761
Interest	0.7642	0.569
Wheat	0.9481	0.663
Ship	0.7039	0.745
Corn	0.8081	0.835

In this work, a label given by GP to a test document is a value between -1 and 1. Thresholds need to be chosen to separate the in class and out class examples for each category. In these experiments, a threshold(T) is calculated as the median between the median value of in class and out class examples of the training data, shown in Equation 8

$$T = median(median(inClass), median(outClass)) \quad (8)$$

If the 'output' value of GP of a test document is greater than the threshold, it is classified to be in class, otherwise it is classified to be out class. For each test document, we run it through all classifiers in the hierarchy, so that we can identify multi-labelled documents as well.

Table III shows the classification results of our system and the tree based GP system [10] mentioned in Section 2 based on the returned F1-measure.

The results show that our proposed system performs better than the tree based GP classifier in [10], which worked on N-gram representation, although the information preprocessing is different and not a strictly controlled comparison. Indeed our intention at this point is not to produce the best classifier in terms of accuracy but to produce a good classifier which can analyze the word sequences of documents for classification while tracking the context changes associated with words and the categories. The results show that our system performs much better than the tree based GP on 8 of the 10 categories. However, it does not perform well on 2 categories namely 'Ship' and 'Corn'. We hypothesize that the reason behind this is that there is no consistent co-occurrent words in those categories to represent the system to be tracked and to be utilized for categorization. Finally, it is worth to note that the RLGP classifier generates really simple rules as shown in the evolved rule for category "Earn". The rules produced can be reformulated and easily fed into a database or an Internet search engine to retrieve similar text based on the context and/or the structure of documents or to perform online text analysis.

VII. CONCLUSIONS AND FUTURE WORK

We have produced a system capable of encoding documents into sequences and discovering rules by utilizing these sequences. The encoding process is based on a hierarchical SOM architecture. Recurrent Linear GP was used at the stage of document classification. The returned results are competitive with other systems [10]. However, our system is the first system that directly analyze the word sequence of a document without giving any weight or probability value to each individual word. The rules generated by RLGP is relatively simple and can be easily stored in a database or embedded in programs. Thus, it is useful for online document analysis and word tracking associated with categories and also useful for document structure analysis where we have tags to define the structure of a document like HTML files. Moreover, this system can efficiently work on both textual and non-textual data. The idea of sequence analysis using recurrent GP is new for text analysis, the future work will investigate the following issues:

- Developing different fitness functions which would give a more natural fitness measure for information retrieval, such as including F1 measure in the fitness function
- Developing different forms of DSS algorithm where subset is selected based on other rules associated with the nature of categories instead of simple age and difficulty
- Exploring other feature selection and encoding techniques

We believe that the system described here with further improvement may be of value when used in some text classification or data analysis application, especially where temporal sequence information is important and significant.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Malcolm Heywood for his helpful discussions. This work was supported by the grant of NSERC, Canada

REFERENCES

[1] P. N. Bennett, S. T. Dumais, and E. Horvitz, "Probabilistic Combination of Text Classifiers Using Reliability Indicators: Models and Results," *Proc. 25th ACM International Confernece on Research and Development in Information Retrieval(SIGIR2002)*, 2002, pp. 207–215.

[2] N. Cancedda, E. Gaussier, C. Goutte, J. Renders, "Word-sequence Kernels," Journal of Machine Learning Research, vol. 3, 2003, pp. 1059-1082.

[3] M. F. Caropreso, S. Matwin, and F. Sebastiani, "A Learner-Independent Evaluation of the Usefulness of Statistical Phrases for Automated Text Categorization," Text Databases and Document Management: Theory and Practice, 2001, pp. 78-102.

[4] N.L. Cramer, "A Representation for the Adaptive Generation of Simple Sequential Programs," *Proc. International Conference on Genetic Algorithms and Their Application*,1985,pp. 183-187.

[5] S. Dumais, H. Chen, "Hierarchical Classification of Web Cocument," *Proc. 23rd ACM International Conference on Research and Development in Information Retrieval(SIGIR2000)*, 2000.

[6] S. Dumais, J. Platt, and D. Heckerman, "Inductive Learning Algorithms and Representations for Text Categorization," *Proc. 7th International Conference on Information and Knowledge Management*, 1998, pp. 148–155.

[7] J. Farkas, "Document Classification and Recurrent Neural Networks," *Proc. Conference of the Centre for Advanced Studies on Collaborative research*, 1995, pp. 21-27.

[8] R.M. Friedberg, "A Learning Machine: Part I," IBM Journal of Research and Development,1958, vol. 2, no. 1, pp. 2-13.

[9] M.I. Heywood, A.N. Zincir-Heywood,"Dynamic Page-Based Linear Genetic Programming", IEEE Transactions on Systems, Man and Cybernetics - PartB: Cybernetics, 2002, vol. 32, no. 3, pp. 380-388.

[10] L. Hirsch, M. Saeedi, R. Hirsch, "Evoloving Rules for Document Classification," *Proc. 8th European Conference, EuroGP 2005*, Switzerland, 2005, pp. 85-95.

[11] C.-C. Huang, S.-L. Chuang, and L.-F. Chien, "Liveclassifier: Creating Hierarchical Text Classifiers Through Web Corpora," *Proc. 13th International Conference on World Wide Web*, 2004, pp. 184-192.

[12] L. Huelsbergen,, "Finding General Solutions to the Parity Problem by Evolving Machine-Language Representations," *Proc. 3rd Conference on Genetic Programming*, 1998, pp. 158-166.

[13] D. D. Lewis, "An Evaluation of Phrasal and Clustered Pepresentations on a Text Categorization Task," *Proc. 15th International ACM Conference on Research and Development in Information Retrieval*, 2002, pp. 37–50.

[14] X. Luo, A. N. Zincir-Heywood, "Analyzing the Temporal Sequences for Text Categorization," *Proc. the 8th International conference on Knowledge-Based Intelligent Information and Engineering Systems*, Springer-Verlag LNAI 3215, 2004, pp. 498-505.

[15] X. Luo, M. Heywood, and N. Zincir-Heywood, "Evolving Recurrent Models Using Linear GP," *Proc. 2005 conference on Genetic and evolutionary computation*, 2005, pp. 1787-1788.

[16] X. Luo, A. N. Zincir-Heywood, "A Comparison of SOM Based Document Categorization Systems," *Proc. Joint Conference on Neural Network*, 2003, pp. 1786-1791.

[17] P. Nordin, "Evolutionary Program Induction of Binary Machine Code and its Applications," Vrehl Verlag, Mnster, 1999(Corrected Edition).

[18] M. Ruiz, P. Srinivasan, "Hierarchical Text Categorization using Neural Networks," Information Retrieval, vol. 5, pp. 87-118

[19] F. Sebastiani, "Machine Learning in Automated Text Categorization," ACM Computing Surveys, vol. 34, no. 1, 2002, pp. 1-47.

[20] D. Song, M.I. Heywood, and A.N. Zincir-Heywood, "Training Genetic Programming on Half a Million Patterns: an Example from Anomaly Detection," IEEE Transactions on Evolutionary Computation, 2005, vol. 9, no. 3, pp. 225-239.

[21] S. Wermter, G. Arevian and C. Panchev, "Recurrent Neural Network Learning for Text Routing," *Proc. 9th International Conference on Artificial Neural Networks*, 1999, pp. 470-475.