2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Self-learning Recursive Neural Networks for Structured Data Classification

Abdelhamid Bouchachia and Alexander Ortner

Abstract— **Automatic classification of structured data is a challenging task and its relevance to many domains is evident. However, collecting labeled data may turn to be a quite expensive task and sometimes even prone to mislabeling. A technical solution to this problem consists in combining few labeled data samples and a significant amount of unlabeled data samples to train a classifier. Likewise, the present paper deals with the classification of partially labeled tree-like structured data. To carry on this task, we suggest an adapted variant of recursive neural networks (RNNs) that is equipped with semi-supervision mechanisms capable of learning from labeled and unlabeled tree-like data. Accordingly RNNs rely on self-learning to actively pre-label data which will be combined with originally labeled one during the learning process. The semi-supervised RNNs approach is presented and evaluated on real-world eXtensible Markup Language (XML) collection of documents in the context of digital libraries. The initial empirical experiments show high quality results.**

I. Introduction

are very efficient in various applications, particularly in classification tasks. NNs are inductive machines capable of learning from examples. Apart from reinforcement learning, there are two main types of learning dedicated to neural networks: supervised and unsupervised. In supervised learning (classification and prediction), the decision function sought is learned from training pairs (input vector, class label). Among others, multilayer perceptron, radial basis functions and learning vector quantization are NNs that use supervised training during which the squared distance to the target class value is stepwise minimized. In unsupervised learning (clustering), on the other hand, the decision function sought is learned from only input vectors by generally minimizing the intra-cluster distance and maximizing the between-clusters distance. Examples of NNs based on unsupervised learning are self-organizing maps and adaptive resonance theory.

It has been recognized the value of combining both classification and clustering in what is known as semi-supervised learning (SSL)[8]. The semi-supervised learning paradigm aims at using both labeled and unlabeled data for learning classification problems. While unsupervised learning algorithms are relatively cheap to build, their classification accuracy may not be high. On the other hand, supervised learning algorithms relying on fully labeled training data provide high classification accuracy. The difficulty here is the cost of obtaining labeled training data. In many applications, acquiring labels is very expensive and prone to mislabeling errors, leading to data-size unbalance between available labeled and unlabeled data, especially when labeled data is scarce. Think, for instance, about applications where to label one data point you need to run expensive experiments or you need to hire an expert to do that. Labeled data can be plentiful for some applications, but in other applications such as medical imaging, web classification, biomedical data applications, the correct class labels can not be easily obtained for a significant part of the vast amount of training data available. Usually the difficulty in obtaining class labels may arise due to incomplete knowledge or limited resources.

While many machineries have been used to deal with semi-supervised learning (e.g. including generative models, ensemble learning, graph-based models, boundary classification, and semi-supervised clustering), negligible work has been done with respect to neural networks. Known work in this context relies on pre-labeling techniques [5], [3] and focuses on radial basis function networks [2] and MLP [17], [20]. The problem of such NNs is that supervised learning approaches relying on the error backpropagation have no direct way to incorporate unlabeled data and, therefore, discard them. The present contribution aims at investigating ways to combine labeled and unlabeled data to train a particular type of neural networks, that is RNNs.

We are investigating semi-supervised learning relying on recursive neural networks due to the following reasons:

1) Novelty of the proposed study, since this type of neural networks has never been investigated in the context of semi-supervised learning despite its relevance to many domains where data are structured (tree-like data, graphical data, chemical data, etc.)
2) The application of RNNs in the context of XML document classification allows testing their capability on complex and partially labeled data. Such an experimental setting has not been investigated sofar to the best of our knowledge. The study is thus significant in terms of scale compared to other work on smaller and `fully` labeled data sets [14].

RNNs have strong theoretical background and are efficient in a wide range of pattern classification. However, they need mechanisms to handle hybrid data.

In this study we propose and investigate the following:

1) Proposal of recursive neural networks for the classification of structured data (XML documents).
2) Adaptation of a self-learning algorithm to handle partly labeled structured data. RNNs are equipped with sleflearning mechanisms to accommodate unlabeled data.

Abdelhamid Bouchachia is with Bournemouth University, Faculty of Science and Technology, UK. (email: abouchachia@bournemouth.ac.uk). Alexander Ortner is with the University of Klagernfurt, Austria. (email: a3ortner@edu.uni-klu.ac.at).

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

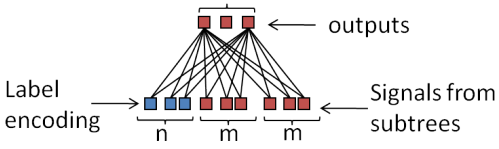
3) A first empirical evaluation of the proposed self-learning RNNs on XML documents including sensitivity analysis.

In the following, recursive neural networks are introduced in Sec. II. Details on the training procedure are also provided in Sec. II-A. The mapping of XML documents onto RNN is outlined in Sec. III. Next, an overview of semi-supervised learning algorithm and description of the self-learning algorithm applied in this study are presented in Sec. IV. Section V provides an initial empirical evaluation of the approach. Finally, Sec. VI concludes the paper.

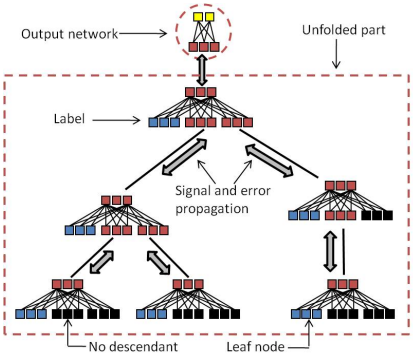
II. Recursive Neural Networks

Being interested in a particular type of data, namely trees and graphs, RNNs seem very appropriate for classification purposes. RNNs have been used in some applications including medical and technical diagnoses, molecular biology and chemistry, geometrical and spatial reasoning, speech and language processing [12], [18]. Figure 1(a) illustrates the structure of the generalized recursive neuron. A neuron in the RNN is an extension of a neuron in one-level recurrent neural network, where instead of just considering the output of the neuron in the previous time step (like with the recurrent neuron), the recursive node gets signals from its subtrees.

The encoding of a node (for instance the root node in the figure) of the tree shown at the bottom of the figure admits as input the signals emanating from the children nodes (pointed to by the current node). Functionally speaking, each node in the tree will be represented as a standard neural network. In other terms, the combination of generalized neurons results in recursive neural networks as shown in Fig. 1(b).



(a) The structure of a recursive neuron



(b) A generic recursive neural network

Fig. 1. Structure of RNNs

Figure 1(b) shows the architecture of an RNN consists of:

- `Folding part`: It encodes the information of the tree structure and is called the `encoding part`.
- `Classification part`: This part classifies the trees with the encoded information of the folding part as input. It is also called the `transformation part`.

Each tree's node will be represented as an individual network in the folding structure consisting of at least two layers. The first layer consists of two components:

- Units for encoding the label (tag) of the node (represented as blue units). The number of units is n.
- Units for encoding the descendants of the node. Assuming the maximum outdegree of all the trees from the collection is k and the number of input units used to encode each descendant is m, the total number of units for representing the subtree rooted at the node is $k \cdot m$. If some nodes have less than the maximum k of children, the missing information will be set to nil (units are represented as black units in Fig 1(a)). An output neuron assembles information of the label encoding part and from the subtree to be transmitted to the node's parent.

A detailed view of an RNN is shown in Fig. 2. It consists of $r+s+1$ layers, where layers $l_0, \ldots, l_r$ belongs to the folding part and layers $l_r, \ldots, l_{r+s}$ belong to the classification part. All layers are fully connected. The symbols appearing in Fig.1(a) and Fig. 2 are defined as follows:

n	... the representation length of the node's label
m	... the number of input units
k	... the maximum outdegree of the tree's nodes
l_0	... the first input layer consisting of $(n+k \cdot m)$ units
l_{r+s}	... the last layer, where the i-th output corresponds to the i-th class label, for $0 \leq i < q$
q	... the number of class labels
$\lambda(t)$	... the label of the node t

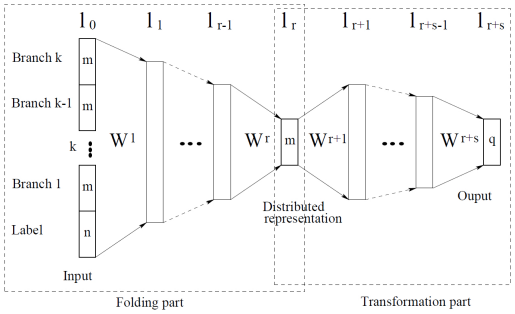


Fig. 2. The architecture of a recursive neural network proposed in [12]

The first proposal for using RNNs for (supervised) classification purposes developed by Sperduti and Starity [18]. In the present work, RNNs are used to classify XML documents which can be seen as trees/graphs. Figure 3 illustrates the structure of an XML document.

Given a tree to be processed, the folding part of the RNN will represent the nodes of the tree such that the signal flows from the leaves and finishes with the root node. In other terms, the bottom-up order has to be kept so that the signal emanating from the descendant nodes is transmitted as input

(a) An XML source file (b) The encoding tree

Fig. 3. Structure of XML data

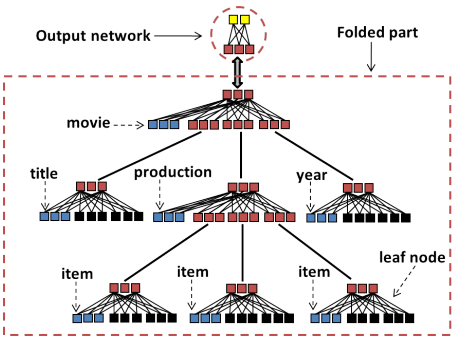


Fig. 4. Equivalent recursive neural network

to their parent nodes. Figure 4 illustrates an example of a folding part with two layers.

Specifically it shows how an XML document with 6 nodes can be represented as a neural network, where the folding part encodes the XML tree. Note that the nodes $\{movie, title, production, year, item\}$ in Fig. 4 use the same elementary coding structure of the folding part. The classification part admits the output of the folding part of the root node as input.

A. Backpropagation Through Structure

To train this neural architecture, Kuechler and Goller [12] introduced an algorithm called *backpropagation through structure (BPTS)* inspired from the well-known backpropagation through time. BPTS consists of two phases:

- Forward phase: information flows from the leaves to the root which represents the classification part.
- Backward phase: the whole neural network is updated backwards, starting by the classification part and going through the folding part down to the leaves.

An XML document is presented to the network node by node, starting from the leaves. As mentioned earlier, each node is represented as a folding part. Only after the signal from all subtrees have reached the root node of the tree, this later can then transmit the signals to the classification part.

Let the output of a neuron i in layer l for a node t be denoted as $y_i^{(l)}(t)$. The first layer l_0 consists of $n + k \cdot m$ neurons. The first n neurons are for encoding the node's label and the remaining $k \cdot m$ neurons are for encoding the input signals of the subtrees. Note that k is the maximum outdegree of all trees of the collection and m is the number of neurons for representing the information of the current node's subtrees. The value of m is the same for the whole tree. The output for layer l_0 is defined as follows:

$$y_i^{(0)}(t) = \begin{cases} [C(\lambda(t))]_i & 0 \le i < n \\ y^{(r)}_{((i-n) \bmod m)}\left(t_{\lfloor \frac{i-n}{m} \rfloor}\right) & n \le i < n + dm \\ [nil]_{((i-n) \bmod m)} & n + dm \le i < n + km \end{cases} \quad (1)$$

where C denotes a coding function which maps the label $\lambda(t)$ of the node t to a numeric code. The variable d represents the number of descendants of a node t, $0 \le d \le k$ and $t_i \in \{t_0, t_1, \ldots,\}$ the children of node t. For nodes with less then k children, the missing data will be interpreted as nil information. The output for the layers l_j, $j > 0$ are defined as follows:

$$\begin{aligned} y_j^{(l+1)}(t) &= f(net_j^{(l+1)}(t)) = f(v_j^{(l+1)}(t)) \\ &= f(\sum_i y_i^{(l)}(t) \cdot \omega_{ij}^{(l+1)} + \theta_j^{(l+1)}) \\ &\quad \text{for } r \le l < r + s, \text{ if } t \text{ is a root node} \quad (2) \\ &\quad \text{for } 0 \le l < r, \text{ if } t \text{ is any node} \quad (3) \end{aligned}$$

where $\theta_j^{(l+1)}$ denotes the bias associated with neuron j at layer $l + 1$, $w_{ij}^{(l+1)}$ is the weight of the connection between neuron i of layer l and neuron j of layer $l + 1$ and $net_j = v_j$. The function f represents the activation function of the neural network. The output signals of all other nodes are transmitted to their parents starting from the leave nodes. For instance, in Fig. 4 the `production` node receives signals from `item`'s. The signal obtained from the `production`, `title` and `year` nodes are sent to the `movie` node, whose output is in turn transmitted to the classification network.

III. Preprocessing and Training

A. Encoding of Labels

The first step to be taken in order to classify XML documents using recursive neural networks is to analyze the set of documents. In particular, the common supertree of the collection is constructed by merge as shown in Fig. 5. The resulting supertree serves to determine the architecture of the recursive neural network, the different labels and the maximum number of descendant elements occurring in the collection.

In the second step, the labels of the supertree nodes are encoded. In this work, we apply a simple binary mapping function that generates distinct codes for the available labels (tags). Figure 6 illustrates the encoding idea.

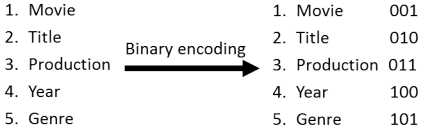


Fig. 6. Generation of binary codes for the nodes' labels

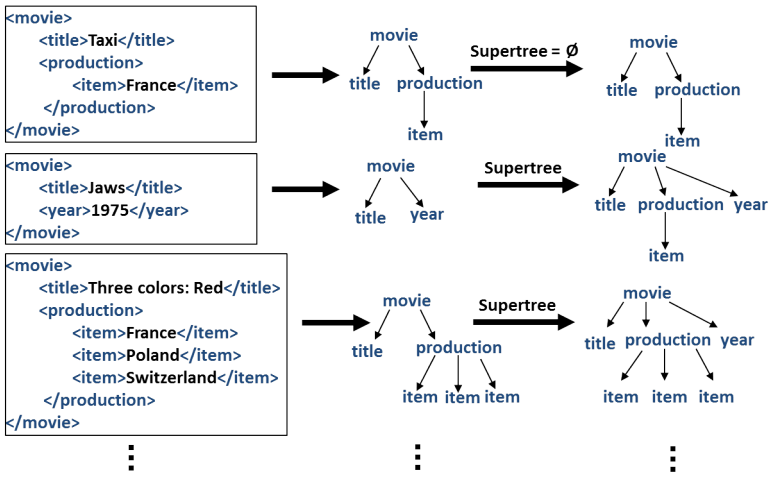


Fig. 5. Building a common supertree stepwise

B. Learning Algorithm

Algorithm 1 illustrates the preprocessing and main learning steps of the Backpropagation through structure algorithm for classifying structured XML documents.

Algorithm 1 : Training algorithm

- Given the number of classes, the maximum number of descendant nodes of the XML trees and the number of different labels in the XML tree.
- Generate the recursive neural network (i.e., folding and classification parts of the net)

repeat

for each document in the training set **do**

- **Forward phase:** starts from the leaves to the root and transmits the information to the classification part. The output of each layer are calculated.
- **Backward phase:** updates the whole neural network backwards, starting by the classification part and going through the folding part down to the leaves.
  - Calculate the total error of the classification.
  - Calculate the gradient of each output layer of a node. starting from the root back to the leaves.
  - Update the synaptic weights and the bias.

end for

until The classification error obtained falls under a threshold or the number exceeds some given number.

IV. Semi-supervised via Self-learning

A. Semi-supervised Learning

As outlined in Sec. I, semi-supervised learning is a very interesting learning scheme. It aims at devising mechanisms of learning from labeled and unlabeled data in a symbiotic way. The underlying motivation is to boost the accuracy of the classifier using the available labeled data augmented with the unlabeled one. The scenario is portrayed in Fig. 7 which shows that labeled and unlabeled data can be processed

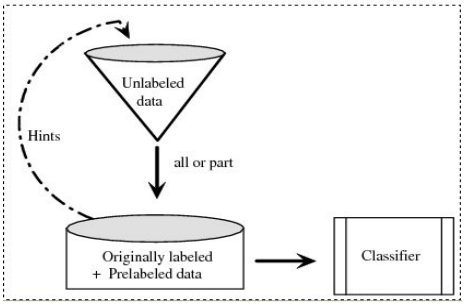


Fig. 7. The process of pre-labeling

according to two fundamental alternatives learning via pre-labeling and pure semi-supervised learning:

1) Usage of labeled data (or some knowledge hints extracted from it) to estimate the label of unlabeled data before initiating a fully supervised classifier. This scheme is known as pre-labeling which can be either active (by human being) or passive pre-labeling (by an algorithm)
2) Complete merge of both labeled and unlabeled data to train a partially supervised classifier. This scheme corresponds to pure semi-supervised learning (which is also known as learning with partial supervision)

So far many approaches have been suggested to deal with learning from hybrid data. Broadly speaking, the approaches can be classified into: generative [7], [15], ensemble learning [10], [21], graph-based [1], [19], boundary classification [13], and semi-supervised clustering models [5]. All of the available methods fall in one of the two schemes mentioned above.

Considering self-learning which relies on offline or online pre-labeling [4], [3], [16] to enlarge the training data set in order to enable a fairly consistent training of the classifier. In the present work, we aim at incorporating unlabeled data in the training of the recursive neural network described earlier. Self-learning based on a feedforward neural network has

been adopted in [17] and [20]. The neural network model can be used to estimate a posterior probability function for each class in the classification domain, and assigning an unlabeled example to the class for which the posterior probability of membership is highest [17]. In [20], the unlabeled example is assigned to the class (represented as fuzzy set) for which the fuzzy membership degree is the highest. This approach will be adapted in the present work to use both labeled and unlabeled XML documents to train the recursive neural network.

Assuming that the training set T consists of two subsets $T = \{T_l, T_u\}$, where $T_l = \left\{(t^1, c^1), (t^2, c^2), \ldots, (t^{N_l}, c^{N_l})\right\}$ represents the subset of labeled trees and $T_u = \left\{t^{N_l+1}, t^{N_l+2}, \ldots, t^{N_l+N_u}\right\}$ the subset of unlabeled trees. The symbol t^i denotes the ith tree, c^i denotes the class label such that $i \in I = \{1, 2, \ldots Q\}$, Q is the number of classes. N_l and N_u are the number of labeled and unlabeled trees respectively, $N = N_l + N_u$. The estimated labels for the unlabeled subset $o^1, \ldots, o^{N_l}$ are obtained according to the scheme *1-of-Q*, i.e.

$$o_k^n = \begin{cases} 1 & \text{if } c^n = k \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

B. Label Estimation for Unlabeled Data

According to Verikas et al. [20], the neural network output values provide membership degrees to the fuzzy sets which represent the classes. The membership degree $\mu_{c_j}(t^n)$ of the nth XML tree t^n to the jth fuzzy set is then given by the output signal $y_j^{(r+s)}(t^n)$ of the jth output neuron of the neural network. Applying the intensification operation which increases the contrast between the elements of the fuzzy set, increasing $\mu_{c_j}(t^n)$ which are larger than 0.5 as follows:

$$\eta_{c_j}(t^n) = \begin{cases} 2\left[\mu_{c_j}(t^n)\right]^2 & \text{for } 0 \le \mu_{c_j}(t^n) \le 0.5 \\ 1 - 2\left[1 - \mu_{c_j}(t^n)\right]^2 & \text{otherwise} \end{cases}$$
$$j = 1, 2, \ldots, Q \tag{5}$$

Verikas et al. used k-nearest neighbors (k_{nn}) for their label estimation. The k_{nn} criterion is given as:

$$\left\|x^n - x^k\right\| < \left\|x^n - x^i\right\|, \quad \forall k \in M^n \text{ and } \forall i \notin M^n \tag{6}$$

where M^n is the set of indices of the nearest neighbors of x^n.

To adapt the algorithm proposed by Verikas et al. to structured data like XML trees, we need to define a distance function between two labeled trees. There are several approaches to compute such a distance. In this study, we rely on the tree edit distance proposed in [9]. The tree edit distance consists of three edit operations:

1) *insertion:* inserts a child to a node in the tree. The cost for this operation is 1. The number of occurrences is denoted with ins.
2) *deletion:* deletes a node from the tree. The cost for this operation is 1. The number of occurrences is denoted with del.
3) *update:* updates the label of a node in the tree. The cost for this operation is 1. The number of occurrences is denoted with upd.

The idea is to count the steps of a tree a to be transformed into tree b. We can now create a distance measure between two nodes.

$$TreeDist(t^i, t^j) = \frac{ins + del + upd}{|t^i| + |t^j|} \tag{7}$$

where $|t^i|$ denotes the number of nodes of tree t^i and $|t^i|+|t^j|$ is the upper limit of edit steps to transform t^i into t^j. Note that $TreeDist$ is symmetric and standardized. The following pseudo code illustrates the determination of $ins + del + upd$.

Algorithm 2 : Computation of the distance between two nodes

```
// CalculateDistance(TreeNode s, TreeNode t)
Let D a matrix of size: numOfChildren(s)+
1 × numOfChildren(t) + 1;
D[0,0]=UpdateCost(LabelOf(s), LabelOf(t));
for i = 1 to numOfChildren (s) do
  D[i,0] = [i − 1, 0]+numOfNodes(s_i);
end for
for j = 1 to numOfChildren(t) do
  D[0, j] = D[0, j − 1]+numOfNodes(t_j);
end for
for i = 1 to numOfChildren (s) do
  for j = 1 to numOfChildren (t) do
    D[i, j] = Min{CalculateDistance(s_i, t_j) +
    D[i−1, j−1], D[i, j−1]+numOfNodes(t_j), D[i−
    1, j]+numOfNodes(s_i) };
  end for
end for
Return(D);
```

The quantities used are:

- s_i, t_j are the ith and jth subtrees of node s and t respectively.
- `numOfChildren(s)` denotes the number of the sth node's descendants.
- `numOfNodes(s)` denotes the number of nodes of the subtree rooted at s.
- `UpdateCost(LabelOf(s), LabelOf(t))` returns 1 if labels of s and t are equal or 0 otherwise.
- `D[i][j]` stores the edit distance ($ins + del + upd$) between the subtree a tree T_1 rooted at i and the subtree of the tree T_2 rooted at j.

After introducing $TreeDist$, the $k-$nearest neighbors for the unlabeled trees can then be identified as follows:

$$TreeDist(t^n, t^k) < TreeDist(t^n, t^i), \ \forall k \in M^n \text{ and} \tag{8}$$
$$\forall i \notin M^n \quad N_l + 1 \le n \le N; \quad 1 \le k, i \le N \tag{9}$$

Then, we calculate the target vector o^n for the unlabeled tree t^n as follows:

$$o^n = \frac{\sum_{z \in M^n} o^z}{k_{nn}} \tag{10}$$

where

$$o_j^z = \begin{cases} \eta_{c_j}(t^z) & \text{if } N_l + 1 \leq z \leq N \\ 1 \text{ or } 0 & \text{if } 1 \leq z \leq N_l \end{cases} \quad (11)$$
$$\forall j = 1, \ldots, Q; \qquad \forall n = N_{l+1}, \ldots, N$$

The self-learning algorithm is then given in Alg. 3.

Algorithm 3 : Training algorithm

Train the network using labeled XML trees only, as proposed in section III-B.
repeat
Calculate target values for the unlabeled data using Eq. 10.
Retrain the network using both the labeled and unlabeled XML trees.
until the classification accuracy does not change after a certain number of subsequent iterations or a predefined number of iterations is reached

V. Empirical Evaluation

To illustrate the application of self-supervised RNN for the classification of XML documents by structure, we will focus on four aspects:

- Effect of labeled data on the accuracy of RNN
- Effect of unlabeled data on the accuracy
- Comparison of fully supervised RNN against semi-supervised RNN
- Effect of increasing the number of classes

A movie database obtained from INEX[1] is applied. A subset of 2063 XML documents stemming of three classes are considered in the first set of experiments. The classes consists of 594, 701 and 768 documents respectively. This XML data set is randomly split into a training set (60%) and a testing set (40%). Moreover to conduct the experiments of this self-supervised learning approach, the training set is randomly divided into two subsets: labeled set and unlabeled set. Because of the assumption that the unlabeled set is usually larger than the labeled set, we consider the 70% (unlabeled)-30% (labeled) rule.

Since the time computational complexity of training the RNN on random structure is very high, we attempt to enhance the training efficiency. To achieve that, we cluster the unlabeled set and only use a certain proportion thereof that consists of the typical examples. We use the k-medoids algorithm partitioning around medoids (PAM) [11] to cluster the unlabeled XML documents. The choice of PAM is dictated by the nature of data, we do not need to compute a prototype like in K-means and Fuzzy C-means.

PAM operates on a dissimilarity matrix for the given data set and a pre-specified number of clusters k. It searches for k representative documents (i.e., medoids which are the most centrally located document in a cluster). After finding a set of k medoids, k clusters are constructed by assigning each document to the nearest medoid. Note that PAM aims at finding k medoids $M = (m_1, \ldots, m_k)$ which minimize the sum of the dissimilarities of the documents to their closest medoid, that is:

$$argmin_M \sum_i min_k \, d(x_i, m_k) \quad (12)$$

where function $d(x, y)$ is the distance between object x and y. Note that the distance between two documents is calculated according to Eq. 7. The main steps of the PAM algorithm are shown in Alg. 4.

Algorithm 4 : PAM steps

Randomly choose k documents that stand for initial medoids.
repeat
Associate each data point to the nearest medoid (distance between two documents is calculated according to Eq. 7).
for For each medoid m **do**
for For each non-medoid document o **do**
Swap m and o and compute the total cost (summation of distances) of the configuration.
end for
end for
Select the configuration with the lowest cost (summation of distances).
until there is no change in the medoid set.

The quality of clustering is measured using the entropy of clusters [6]. The optimal number of clusters k the weighted average entropy of the generated clusters. The entropy of cluster i is defined as

$$E_i = -\frac{1}{log(H)} \sum_{j=1}^{H} \frac{n_{ij}}{n_i} \, log\left(\frac{n_{ij}}{n_i}\right) \quad (13)$$

where n_i is the total number of XML documents in the cluster i and n_{ij} is the number of XML documents of class j positioned in cluster i.

Once the clusters are obtained, the most typical samples around the medoids are collected to form the unlabeled set. An XML document of a cluster is selected if the distance between this document and a medoid does not exceed a certain threshold which is set to 0.5. For 70% of the training data (=865 unlabeled documents), we have found that the optimal number is 3. The quality of the clusters measured by Eq. 13 and the medoids are shown in Tab. I and Tab. II respectively. Then the pre-selection of the unlabeled set allowed to retain 651 documents among the initial set consisting of 865 documents.

The main goal is the evaluation of the proposed learning algorithm by measuring the classification accuracy. In all of the experiments we use the following network parameters:

- $\eta = 0.1$, learning rate
- $\alpha = 0.01$, momentum rate

[1] http://www.inex.otago.ac.nz/tracks/strong/strong.asp

TABLE I

CLUSTER QUALITY

cluster	size	quality
1	237	0
2	292	0.037283
3	336	0

TABLE II

MEDOIDS OF THE CLUSTERS

cluster	medoid
1	`"Lively_Set_The_1964.xml"`
2	`"Book_Revue_1946.xml"`
3	`"People_Will_Talk_1951.xml"`

TABLE IV

ACCURACY AFTER 30 RUNS (EFFECT OF UNLABELED DOCUMENTS)

Proportion	30%	50%	70%	90%
Accuracy mean	0.93805	0.98370	0.95892	0.98640
Stan. deviation	0.10732	0.06388	0.12642	0.05480

- $H = 3$, hidden layer size
- $m = 4$, number of nodes in context layer
- $k_{nn} = 5$, number of nearest neighbors (for label estimation)

These settings have been determined based on initial runs on a subset of data. Tab. III illustrates just an excerpt of configurations among a great number of configurations we tested. The best parameter choice corresponds to configuration 4. To ensure a fair evaluation of the accuracy of the proposed model, each experiment is repeated 30 times. While the parameters remain the same, the labeled and unlabeled change - but their proportions remain the same. That is, the random split of the XML documents has to be renewed for each run. Consequently, the clustering of the new unlabeled XML documents will be executed in each run.

A. Effect of Unlabeled Data

In the following the effect of increasing the number of unlabeled documents is observed. The experiments are carried out using 30% (260 docs), 50% (433 docs), 70% (607 docs) and 90% (780 docs) of the unlabeled data (70% of the training data = 867 docs). The subsets of the unlabeled data are randomly sampled. Table IV shows the results of the 30 runs for each configuration. The accuracy of the classifier increases as the number of unlabeled documents increases, except in the case of 70% of unlabeled data. On a closer look at the accuracy value obtained from 70% can be explained with the standard deviation that is relatively high.

B. Effect of Labeled Data

To check the effect of changing the number of labeled documents. Various proportions of the labeled set (30% of the whole training data) are considered: 10% (67 docs), 30% (111 docs), 50% (185 docs), 70% (260 docs) and 90% (334 docs). The unlabeled size is 70% of the training data. Note that the subsets are chosen randomly. Table V depicts the classification results of the thirty runs for each of the proportion. As expected, the accuracy of the classifier increases as the number of labeled documents increases.

TABLE VI

ACCURACY OF SUPERVISED AND SEMI-SUPERVISED CLASSIFIER

	Supervised	Self-supervised
Accuracy mean	0.97764	0.98414
Stan. deviation	0.05278	0.05643

C. Supervised vs. Self-supervised Learning

In the following a comparison between supervised and self-supervised is shown. In these experiments, we depart from the same baseline, that is, the supervised and self-supervised algorithms uses the same labeled set of documents. However, the self-supervised algorithm additionally involves unlabeled documents in the training phase.

Table VI depicts the classification results of the thirty runs using the labeled 30% of the training data set for training the supervised RNN and using the whole training set (30% labeled + 70% unlabeled) for training the self-supervised RNN. The conclusion drawn from the previous set of experiments related to the positive effect of unlabeled data is confirmed by the direct comparison of supervised RNN and self-supervised RNN. The accuracy of the semi-supervised classifier is higher than the accuracy of the supervised classifier.

D. Effect of the Number of Classes

To check the effect of increasing the number of classes, we have considered 5 classes instead of 3 classes, the results are shown in Tab. VII and Tab. VIII. Again, we observe theevolution of the self-learning RNN when increasing the proportion of labeled and unlabeled documents respectively. We have considered only 5 classes due to the high time complexity of training RNN. The results after 30 runs show similar behavior of the supervised RNN and self-supervised RNN as the one that emerged with 3 classes. The positive effect of labeled and unlabeled data is clearly confirmed despite some of the insignificant fluctuations that can take place with some proportions.

VI. CONCLUSION

This paper proposes a self-learning version of recursive neural network which aims at classifying partially labeled tree-like structured data. We have focused on XML data as an instance of structured data. The main steps how to preprocess structured data and how training using hybrid data are illustrated. Moreover a detailed empirical study to show the effect of labeled and unlabeled data on the efficiency of the self-learning RNN are discussed. The results show that equipping RNN with semi-supervision mechanisms can enhance the classification accuracy of structured data.

TABLE III

PARAMETER SETTINGS

Parameter	Conf1	Conf2	Conf3	Conf4	Conf5	Conf6	Conf7	Conf8
η	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
α	0.01	0.01	0.00	0.01	0.00	0.01	0.00	0.00
H	3	4	4	4	4	5	5	5
m	3	3	3	3	3	3	3	3
n	8	8	8	8	8	8	8	8
k_{nn}	5	3	3	5	4	3	4	5
Clas. rate	92.98%	97.94%	61.50%	99.80%	96.13%	89.83%	97,22%	99.76%

TABLE V

ACCURACY AFTER 30 RUNS (EFFECT OF LABELED DOCUMENTS)

Proportion	10%	30%	50%	70%	90%
Accuracy mean	0.86634	0.94988	0.94306	0.95993	0.99072
Stan. deviation	0.18105	0.13557	0.16374	0.09485	0.01419

TABLE VII

EFFECT OF INCREASING THE NUMBER OF CLASSES (5 CLASSES) - CHANGING THE PROPORTION OF LABELED DOCUMENTS)

% of total labeled documents	10%	30%	50%	70%	90%
Accuracy mean	0.8663	0.9498	0.9430	0.9599	0.9907
Standard deviation	0.0219	0.0072	0.0115	0.0030	0.0032
Variance coefficient	0.2089	0.1427	0.1736	0.0988	0.0143

TABLE VIII

EFFECT OF INCREASING THE NUMBER OF CLASSES (5 CLASSES) - CHANGING THE PROPORTION OF UNLABELED DOCUMENTS)

Proportion of unlabeled documents	0%	30%	50%	70%	90%
Accuracy mean	0.8375	0.9880	0.9837	0.9589	0.9864
Standard deviation	0.1491	0.0172	0.0628	0.1264	0.0548
Variance coefficient	0.1780	0.0174	0.0638	0.1318	0.0555

REFERENCES

[1] A. Blum, J. Lafferty, M. Rwebangira, and R. Reddy. Semi-supervised Learning using Randomized Mincuts. In *Proc. of the* 21^{th} *Int. Conf. on ML*, pages 92–100, 2004.

[2] A. Bouchachia. Learning with Hybrid Data. In *Proc. of the* 5^{th} *Int. Conf. on Hybrid Intelligent Systems*, pages 193–198, 2005.

[3] A. Bouchachia. RBF Networks for Learning from Partially Labeled Data. In *Proc. of the workshop on Learning with Partially Classified Training Data at the* 22^{nd} *Int. Conf. on Machine Learning*, pages 10–18, Bonn, Germany, 2005.

[4] A. Bouchachia. An evolving classification cascade with self-learning. *Evolving Systems*, 1:143–160, 2010.

[5] A. Bouchachia and W. Pedrycz. Data clustering with partial supervision. *International Journal of Data Mining and Knowledge Discovery*, 12(1):47–78, 2006.

[6] A. Bouchachia and W. Pedrycz. Enhancement of fuzzy clustering by mechanisms of partial supervision. *Fuzzy Sets and Systems*, 157:1733–1759, 2006.

[7] A. Bouchachia and C. Vanaret. GT2FC : An online growing interval type-2 self-learning fuzzy classifier. *IEEE Transactions on Fuzzy Systems*, In press, 2014.

[8] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[9] T. Dalamagas, T. Cheng, K. Winkel, and T. Sellis. A methodology for clustering xml documents by structure. *Information Systems*, 31:187–228, 2006.

[10] R. Ghani. Combining Labeled and Unlabeled Data for MultiClass Text Categorization. In *Proc. of the* 19^{th} *Internaltional Conf. on Machine Learning*, pages 187–194, 2002.

[11] L. Kaufman and P. Rousseeuw. *Finding groups in data: An introduction to cluster analysis*. Wiley, New York, 1990.

[12] A. Küchler and C. Goller. Inductive learning in symbolic domains using structure-driven recurrent neural networks. In *KI-96: Advances in Artificial Intelligence*, pages 183–197. Springer, 1996.

[13] D. Lawrence and I. Jordan. Semi-supervised learning via Gaussian processes. In *NIPS*, pages 753–760, 2005.

[14] A. Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Trans. on Neural Networks*, 20(3):498–511, 2009.

[15] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.

[16] R. Raina, A. Battle, H. Lee, B. Packer, and A. Ng. Self-taught learning: transfer learning from unlabeled data. In *The 24th International Conference on Machine Learning*, pages 759–766. ACM, 2007.

[17] A. Skabar. Augmenting supervised neural classifier training using a corpus of unlabeled data. In *Proc. of the 25th Annual German Conference on AI*, pages 174–185, 2002.

[18] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8:714–735, 1997.

[19] M. Szummer and T. Jaakkola. Information Regularization with Partially Labeled Data. *Advances in Neural Information Processing Systems*, 15:1025–1032, 2002.

[20] A. Verikas, A. Gelzinis, and K. Malmqvist. Using unlabeled data to train a multilayer perceptron. *Neural Processing Letters*, 14:179–201, 2001.

[21] Y. Zhou and S. Goldman. Democratic Co-Learning. In *Proc. of the* 16^{th} *IEEE Int. Conf. on Tools with Artificial Intelligence*, pages 1082–3409, 2004.