Knowledge-Based Systems 139 (2018) 13–22

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A Siamese Deep Forest

Lev V. Utkin*, Mikhail A. Ryabinin

Department of Telematics, Peter the Great St.Petersburg Polytechnic University, St.Petersburg, Russia

ARTICLE INFO

Article history:
Received 28 May 2017
Revised 23 September 2017
Accepted 4 October 2017
Available online 5 October 2017

Keywords:
Classification
Random forest
Decision tree
Siamese
Deep learning
Metric learning
Quadratic optimization

ABSTRACT

A Siamese Deep Forest (SDF) is proposed in the paper. It is based on the Deep Forest or gcForest proposed by Zhou and Feng and can be viewed as a gcForest modification. It can be also regarded as an alternative to the well-known Siamese neural networks. The SDF uses a modified training set consisting of concatenated pairs of vectors. Moreover, it defines the class distributions in the deep forest as the weighted sum of the tree class probabilities such that the weights are determined in order to reduce distances between similar pairs and to increase them between dissimilar points. We show that the weights can be obtained by solving a quadratic optimization problem. The SDF aims to prevent overfitting which takes place in neural networks when only limited training data are available. The numerical experiments illustrate the proposed distance metric method.

1. Introduction

One of the important machine learning tasks is to compare pairs of objects, for example, pairs of images, pairs of data vectors, etc. There are a lot of approaches for solving the task. One of the approaches is based on computing a corresponding pairwise metric function which measures a distance between data vectors or a similarity between the vectors. This approach is called the metric learning [2,17,30]. It is pointed out by Bellet et al. [2] in their review paper that the metric learning aims to adapt the pairwise real-valued metric function, for example, the Mahalanobis distance or the Euclidean distance, to a problem of interest using the information provided by training data. A detailed description of the metric learning approaches is also represented by Le Capitaine [6] and by Kulis [17]. The basic idea underlying the metric learning solution is that the distance between similar objects should be smaller than the distance between different objects.

There are many approaches and methods which take into account the above condition. One of the most important and popular approaches is to use the Mahalanobis distance as a distance metric which assumes some linear structure of data. However, this assumption significantly restricts the applicability of the Mahalanobis distance for comparing pairs of objects. Therefore, in order to overcome this restriction, the kernelization of linear methods is one of the possible ways for solving the metric learning problem. Bellet et al. [2] review several approaches and algorithms to deal with nonlinear forms of metrics. In particular, these are the Support Vector Metric Learning algorithm provided by Xu et al. [29], the Gradient-Boosted Large Margin Nearest Neighbors method proposed by Kedem et al. [14], the Hamming Distance Metric Learning algorithm provided by Norouzi et al. [22].

A powerful implementation of the metric learning dealing with non-linear data structures is the so-called Siamese neural network (SNN) introduced by Bromley et al. [5] in order to solve signature verification as a problem of image matching. This network consists of two identical sub-networks joined at their outputs. The two sub-networks extract features from two input examples during training, while the joining neuron measures the distance between the two feature vectors. The Siamese architecture has been exploited in many applications, for example, in face verification [8], in the one-shot learning in which predictions are made given only a single example of each new class [15], in constructing an inertial gesture classification [3], in deep learning [26], in extracting speaker-specific information [7], for face verification in the wild [13]. This is only a part of successful applications of SNNs. Many modifications of SNNs have been developed, including fully-convolutional SNNs [4], SNNs combined with a gradient boosting classifier [18], SNNs with the triangular similarity metric [30].

One of the difficulties of the SNN as well as other neural networks is that limited training data lead to overfitting when training neural networks. Many different methods have been developed to prevent overfitting, for example, dropout methods [24] which are based on combination of the results of different networks by randomly dropping out neurons in the network. A very interesting

* Corresponding author.
E-mail addresses: lev.utkin@list.ru (L.V. Utkin), mihail-ryabinin@yandex.ru (M.A. Ryabinin).

https://doi.org/10.1016/j.knosys.2017.10.006
0950-7051/

new method which can be regarded as an alternative to deep neural networks is the deep forest proposed by Zhou and Feng [31] and called gcForest. In fact, this is a multi-layer structure where each layer contains many random forests, i.e., this is an ensemble of decision tree ensembles. Zhou and Feng [31] point out that their approach is highly competitive to deep neural networks. In contrast to deep neural networks which require great effort in hyperparameter tuning and large-scale training data, gcForest is much easier to train and can perfectly work when there are only small-scale training data. The deep forest solves tasks of classification as well as regression. Therefore, by taking into account its advantages, it is important to modify it in order to develop a structure solving the metric learning task. We propose the so-called Siamese Deep Forest (SDF) which can be regarded as an alternative to the SNNs and which is based on gcForest proposed by Zhou and Feng [31] and can be viewed as its modification. Three main ideas underlying the SDF can be formulated as follows:

1. We propose to modify training set by using concatenated pairs of vectors.
2. We define the class distributions in the deep forest as the weighted sum of the tree class probabilities where the weights are determined in order to reduce distances between semantically similar pairs of examples and to increase them between dissimilar pairs. The weights are training parameters of the SDF.
3. We apply the greedy algorithm for training the SDF, i.e., the weights are successively computed for every layer or level of the forest cascade.

We consider the case of the weakly supervised learning [2] when there are no information about the class labels of individual training examples, but only information in the form of sets of semantically similar or dissimilar pairs of training data is provided, i.e., we know only semantic similarity of examples. However, the case of the fully supervised learning when the class labels of individual training examples are known can be considered in the same way.

It should be noted that the SDF cannot be called Siamese in the true sense of the word. It does not consist of two gcForests like the SNN. However, its aim coincides with the SNN aim. Therefore, we give this name for the gcForest modification.

The paper is organized as follows. A formal statement of the metric learning problem can be found in Section 2. Section 3 gives a very short introduction into the SNNs. A short description of gcForest proposed by Zhou and Feng [31] is given in Section 4. The ideas underlying the SDF are represented in Section 5 in detail. A modification of gcForest using the weighted averages, which can be regarded as a basis of the SDF is provided in Section 6. Algorithms for training and testing the SDF are considered in Section 7. Numerical experiments with real data illustrating cases when the proposed SDF outperforms gcForest are given in Section 8. Concluding remarks are provided in Section 9.

2. A formal statement of the metric learning problem

Suppose there is a training set $S = \{(\mathbf{x}_i, \mathbf{x}_j, y_{ij}),\ \ (i, j) \in K\}$ consisting of N pairs of examples $\mathbf{x}_i \in \mathbb{R}^m$ and $\mathbf{x}_j \in \mathbb{R}^m$ such that a binary label $y_{ij} \in \{0, 1\}$ is assigned to every pair $(\mathbf{x}_i, \mathbf{x}_j)$. If two data vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are semantically similar or belong to the same class of objects, then y_{ij} takes the value 0. If the vectors correspond to different or semantically dissimilar objects, then y_{ij} takes the value 1. This implies that the training set S can be divided into two subsets. The first subset is called the similar or positive set and is defined as

$$\mathcal{S} = \{(\mathbf{x}_i, \mathbf{x}_j) : \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are semantically similar and } y_{ij} = 0\}. \quad (1)$$

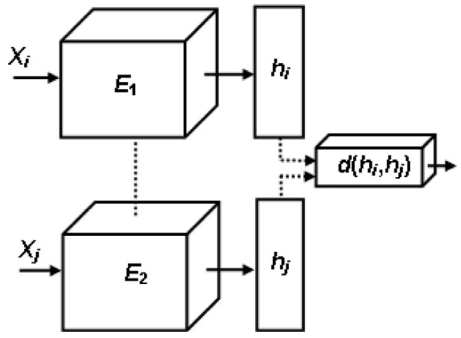

Fig. 1. An architecture of the SNN.

The second subset is the dissimilar or negative set. It is defined as

$$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{x}_j) : \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are semantically dissimilar and } y_{ij} = 1\}. \quad (2)$$

If we have two observation vectors $\mathbf{x}_i \in \mathbb{R}^m$ and $\mathbf{x}_j \in \mathbb{R}^m$ from the training set, then the distance $d(\mathbf{x}_i, \mathbf{x}_j)$ should be minimized if $\mathbf{x}_i$ and $\mathbf{x}_j$ are semantically similar, and it should be maximized between dissimilar $\mathbf{x}_i$ and $\mathbf{x}_j$. It has been mentioned that the most general and popular real-valued metric function is the squared Mahalanobis distance $d^2_M(\mathbf{x}_i, \mathbf{x}_j)$ which is defined for vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ as

$$d^2_M(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} M (\mathbf{x}_i - \mathbf{x}_j). \quad (3)$$

Here $M \in \mathbb{R}^{m \times m}$ is a symmetric positive semi-defined matrix. If $\mathbf{x}_i$ and $\mathbf{x}_j$ are random vectors from the same distribution with covariance matrix C, then $M = C^{-1}$. If M is the identity matrix, then $d^2_M(\mathbf{x}_i, \mathbf{x}_j)$ is the squared Euclidean distance. Learning the Mahalanobis distance metric M implicitly corresponds to seeking a linear transformation which projects data points into a low-dimensional subspace such that the Euclidean distance in the transformed space is equal to the Mahalanobis distance in the original space.

Given subsets $\mathcal{S}$ and $\mathcal{D}$, the metric learning optimization problem can be formulated as follows:

$$M^* = \arg\min_M [J(M, \mathcal{D}, \mathcal{S}) + \lambda \cdot R(M)], \quad (4)$$

where $J(M, \mathcal{D}, \mathcal{S})$ is a loss function that penalizes violated constraints; $R(M)$ is some regularizer on M; $\lambda \geq 0$ is the regularization parameter.

3. Siamese neural networks

Before studying the SDF, we consider the SNN which is an efficient and popular tool for dealing with data of the form $\mathcal{S}$ and $\mathcal{D}$. It will be a basis for constructing the SDF.

A standard architecture of the SNN given in the literature (see, for example, [8]) is shown in Fig. 1. Let $\mathbf{x}_i$ and $\mathbf{x}_j$ be two data vectors corresponding to a pair of elements from a training set, for example, images. Suppose that f is a map of $\mathbf{x}_i$ and $\mathbf{x}_j$ to a low-dimensional space such that it is implemented as a neural network with the weight matrix W. At that, parameters W are shared by two neural networks $f(\mathbf{x}_1)$ and $f(\mathbf{x}_2)$ denoted as E_1 and E_2 and corresponding to different input vectors, i.e., they are the same for the two neural networks. The property of the same parameters in the SNN is very important because it defines the corresponding training algorithm. By comparing the outputs $\mathbf{h}_i = f(\mathbf{x}_i)$ and $\mathbf{h}_j = f(\mathbf{x}_j)$ using the Euclidean distance $d(\mathbf{h}_i, \mathbf{h}_j)$, we measure the compatibility between $\mathbf{x}_i$ and $\mathbf{x}_j$.

If we assume for simplicity that the neural network has one hidden layer, then there holds

$$\mathbf{h} = \sigma(W\mathbf{x} + b). \quad (5)$$

Here $\sigma(z)$ is an activation function; W is the weight $p \times M$ matrix such that its element w_{ij} is the weight of the connection between unit j in the input layer and unit i in the hidden layer, $i = 1, \ldots, p$, $j = 1, \ldots, M$; $b = (b_1, \ldots, b_p)$ is a bias vector; $\mathbf{h} = (h_1, \ldots, h_p)$ is the vector of neuron activations, which depends on the input vector $\mathbf{x}$.

The Siamese neural network is trained on pairs of observations by using specific loss functions, for example, the following contrastive loss function:

$$l(\mathbf{x}_i, \mathbf{x}_j, y_{ij}) = \begin{cases} \left\|\mathbf{h}_i - \mathbf{h}_j\right\|_2^2, & y_{ij} = 0, \\ \max(0, \tau - \left\|\mathbf{h}_i - \mathbf{h}_j\right\|_2^2), & y_{ij} = 1, \end{cases} \quad (6)$$

where τ is a predefined threshold.

Hence, the total error function for minimizing is defined as

$$J(W, b) = \sum\nolimits_{i,j} l(\mathbf{x}_i, \mathbf{x}_j, y_{ij}) + \mu R(W, b). \quad (7)$$

Here $R(W, b)$ is a regularization term added to improve generalization of the neural network, μ is a hyper-parameter which controls the strength of the regularization. The above problem can be solved by using the stochastic gradient descent scheme.

4. Deep forest

According to [31], gcForest generates a deep forest ensemble, with a cascade structure. Representation learning in deep neural networks mostly relies on the layer-by-layer processing of raw features. The gcForest representational learning ability can be further enhanced by the so-called multi-grained scanning. Each level of cascade structure receives feature information processed by its preceding level, and outputs its processing result to the next level. Moreover, each cascade level is an ensemble of decision tree forests. We do not consider in detail the Multi-Grained Scanning where sliding windows are used to scan the raw features because this part of the deep forest is the same in the SDF. However, the most interesting component of gcForest from the SDF construction point of view is the cascade forest.

Given an example, each forest produces an estimate of class distribution by counting the percentage of different classes of examples at the leaf node where the concerned example falls into, and then averaging across all trees in the same forest. The class distribution forms a class vector, which is then concatenated with the original vector to be input to the next level of cascade. The usage of the class vector as a result of the random forest classification is very similar to the idea underlying the stacking method [27]. The stacking algorithm trains the first-level learners using the original training data set. Then it generates a new data set for training the second-level learner (meta-learner) such that the outputs of the first-level learners are regarded as input features for the second-level learner while the original labels are still regarded as labels of the new training data. In fact, the class vectors in gcForest can be viewed as the meta-learners. In contrast to the stacking algorithm, gcForest simultaneously uses the original vector and the class vectors (meta-learners) at the next level of cascade by means of their concatenation. This implies that the feature vector is enlarged and enlarged after every cascade level. The architecture of the cascade proposed by Zhou and Feng [31] is shown in Fig. 2. It can be seen from the figure that each level of the cascade consists of two different pairs of random forests which generate 3-dimensional class vectors concatenated each other and with the original input. After the last level, we have the feature representation of the input feature vector, which can be classified in order to get the final prediction. Zhou and Feng [31] propose to use different forests at every level in order to provide the diversity which is an important requirement for the random forest construction.

5. Three ideas underlying the SDF

The SDF aims to function like the standard SNN. This implies that the SDF should provide large distances between semantically similar pairs of vectors and small distances between dissimilar pairs. We propose three main ideas underlying the SDF:

1. Denote the set indices of all pairs $\mathbf{x}_i$ and $\mathbf{x}_j$ as $K = \{(i, j)\}$ We train every tree by using the concatenation of two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ such that the class $y_{ij} \in \{0, 1\}$ is defined by the semantical similarity of the vectors. In fact, the trees are trained on the basis of two classes and reflect the semantical similarity of pairs, but not classes of separate examples. With this concatenation, we define a new set of classes such that we do not need to know separate classes for $\mathbf{x}_i$ or for $\mathbf{x}_j$. As a result, we have a new training set $R = \{(\mathbf{x}_i, \mathbf{x}_j), y_{ij}), \ (i, j) \in K\}$ and exploit only the information about the semantical similarity. The concatenation is not necessary when the classes of training elements are known, i.e., we have a set of labels $\{y_1, \ldots, y_n\}$. In this case, only the second idea can be applied.
2. We partially use some modification of ideas provided by Xiong et al. [28] and Dong et al. [10]. In particular, Xiong et al. [28] considered an algorithm for solving the metric learning problem by means of the random forests. The proposed metric is able to implicitly adapt its distance function throughout the feature space. Dong et al. [10] proposed a random forest metric learning (RFML) algorithm, which combines semi-multiple metrics with random forests to better separate the desired targets and background in detecting and identifying target pixels based on specific spectral signatures in hyperspectral image processing. A common idea underlying the metric learning algorithms in [28] and [10] is that the distance measure between a pair of training elements $\mathbf{x}_i$, $\mathbf{x}_j$ for a combination of trees is defined as average of some special functions of the training elements. For example, if a random forest is a combination of T decision trees $\{f_t(\mathbf{x}), t = 1, \ldots, T\}$, then the distance measure is

$$d(\mathbf{x}_i, \mathbf{x}_j) = T^{-1} \sum_{t=1}^{T} f_t(\psi(\mathbf{x}_i, \mathbf{x}_j)). \quad (8)$$

Here $\psi(\mathbf{x}_i, \mathbf{x}_j)$ is a mapping function which is specifically defined in [28] and [10]. We combine the above ideas with the idea of probability distributions of classes provided in [31] in order to produce a new feature vector after every level of the cascade forest. According to [31], each forest of a cascade level produces an estimate of the class probability distribution by counting the percentage of different classes of training examples at the leaf node where the concerned example falls into, and then averaging across all trees in the same forest. Our idea is to define the forest class distribution as a weighted sum of the tree class probabilities. At that, the weights are computed in an optimal way in order to reduce distances between similar pairs and to increase them between dissimilar points.

The obtained weights are very similar to weights of the neural network connections between neurons, which are also computed during training the neural network. The trained values of weights in the SDF are determined in accordance with a loss function defining properties of the SDF or the neural network. Due to this similarity, we will call levels of the cascade as layers sometimes.

It should be also noted that the first idea can be sufficient for implementing the SDF because the additional features (the class vectors) produced by the previous cascade levels partly reflect the semantical similarity of pairs of examples. However, in order to enhance the discriminative capability of the SDF, we modify the corresponding class distributions.

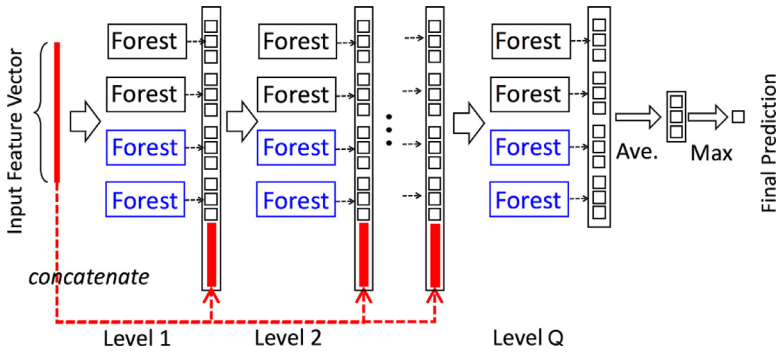

Fig. 2. The architecture of the cascade forest [31].

Table 1
Notations for indices.

Type	Index
cascade level	$q = 1, \ldots, Q$
forest	$k = 1, \ldots, M_q$
tree	$t = 1, \ldots, T_{k,q}$
class	$c = 0, 1$

3. We apply the greedy algorithm for training the SDF that is we train separately every level starting from the first level such that every next level uses results of training at the previous level. In contrast to many neural networks, the weights considered above are successively computed for every layer or level of the forest cascade.

6. The SDF construction

Let us introduce notations for indices corresponding to different deep forest components. The indices and their sets of values are shown in Table 1. One can see from Table 1, that there are Q levels of the deep forest or the cascade, every level contains M_q forests such that every forest consists of $T_{k,q}$ trees. If we use the concatenation of two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ for defining new classes of semantically similar and dissimilar pairs, then the number of classes is 2. It should be noted that the class c corresponds to label $y_{ij} \in \{0, 1\}$ of a training example from the set R.

Suppose we have trained trees in the SDF. One of the approaches underlying the deep forest is that the class distribution forms a class vector which is then concatenated with the original vector to be an input to the next level of the cascade. Suppose a pair of the original vectors is $(\mathbf{x}_i, \mathbf{x}_j)$, and the $p_{ij,c}^{(t,k,q)}$ is the probability of class c for the pair $(\mathbf{x}_i, \mathbf{x}_j)$ produced by the tth tree from the kth forest at the cascade level q. Below we use the triple index (t, k, q) in order to indicate that the element belongs to the tth tree from the kth forest at the cascade level q. The same can be said about subsets of the triple. Then, according to [31], the element $v_c^{(k,q)}$ of the class vector corresponding to class c and produced by the kth forest in gcForest is determined as

$$v_{ij,c}^{(k,q)} = T_{k,q}^{-1} \sum_{t=1}^{T_{k,q}} p_{ij,c}^{(t,k,q)}. \tag{9}$$

Denote the obtained class vector as $\mathbf{v}_{ij}^{(k,q)} = (v_{ij,0}^{(k,q)}, v_{ij,1}^{(k,q)})$. Then the concatenated vector $\mathbf{x}_{ij}^{(1)}$ after the first level of the cascade is

$$\mathbf{x}_{ij}^{(1)} = \left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{v}_{ij}^{(1,1)}, \ldots, \mathbf{v}_{ij}^{(M_1,1)}\right) = \left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{v}_{ij}^{(k,1)}, k = 1, \ldots, M_1\right). \tag{10}$$

It is composed of the original vectors $\mathbf{x}_i$, $\mathbf{x}_j$ and M_1 class vectors obtained from M_1 forests at the first level. In the same way, we can write the concatenated vector $\mathbf{x}_{ij}^{(q)}$ after the qth level of the cascade as

$$\mathbf{x}_{ij}^{(q)} = \left(\mathbf{x}_i^{(q-1)}, \mathbf{x}_j^{(q-1)}, \mathbf{v}_{ij}^{(1,q)}, \ldots, \mathbf{v}_{ij}^{(M_q,q)}\right) = \left(\mathbf{x}_i^{(q-1)}, \mathbf{x}_j^{(q-1)}, \mathbf{v}_{ij}^{(k,q)}, \ k = 1, \ldots, M_q\right). \tag{11}$$

In order to reduce the number of indices, we omit the index q below because all derivations will concern only level q, where q may be arbitrary from 1 to Q. We also replace notations M_q and $T_{k,q}$ with M and T_k, respectively, assuming that the number of forests and numbers of trees strongly depend on the cascade level.

The vector $\mathbf{x}_{ij}$ in (11) has been derived in accordance with the gcForest algorithm [31]. However, in order to implement the SDF, we propose to change the method for computing elements $v_{ij,c}^{(k)}$ of the class vector, namely, the averaging is replaced with the weighted sum of the form:

$$v_{ij,c}^{(k)} = \sum_{t=1}^{T_k} p_{ij,c}^{(t,k)} w^{(t,k)}. \tag{12}$$

Here $w^{(t,\,k)}$ is a weight for combining the class probabilities of the tth tree from the kth forest at the cascade level q. The weights play a key role in implementing the SDF. An illustration of the weighted averaging is shown in Fig. 3, where we partly modify a picture from [31] (the left part is copied from [31, Fig. 2]) in order to show how elements of the class vector are derived as a simple weighted sum. It can be seen from Fig. 3 that two-class distribution is estimated by counting the percentage of different classes ($y_{ij} = 0$ or $y_{ij} = 1$) of new training concatenated examples $(\mathbf{x}_i, \mathbf{x}_j)$ at the leaf node where the concerned example $(\mathbf{x}_i, \mathbf{x}_j)$ falls into. Then the class vector of $(\mathbf{x}_i, \mathbf{x}_j)$ is computed as the weighted average. It is important to note that we weigh trees belonging to one of the forests, but not classes, i.e., the weights do not depend on the class c. Moreover, the weights characterize trees, but not training elements. This implies that they do not depend on the vectors $\mathbf{x}_i$, $\mathbf{x}_j$ too. One can also see from Fig. 3 that the augmented features $v_{ij,0}^{(k)}$ and $v_{ij,1}^{(k)}$ (the class vector) corresponding to the kth forest are obtained as weighted sums, i.e., there hold

$$v_{ij,0}^{(k)} = 0.5 \cdot w^{(1,k)} + 0.4 \cdot w^{(2,k)} + 1 \cdot w^{(3,k)}, \tag{13}$$

$$v_{ij,1}^{(k)} = 0.5 \cdot w^{(1,k)} + 0.6 \cdot w^{(2,k)} + 0 \cdot w^{(3,k)}. \tag{14}$$

The weights are restricted by the following obvious condition:

$$\sum_{t=1}^{T_k} w^{(t,k)} = 1. \tag{15}$$

In other words, we have the weighted averages for every forest, and the corresponding weights can be regarded as trained parameters in order to decrease the distance between semantically similar

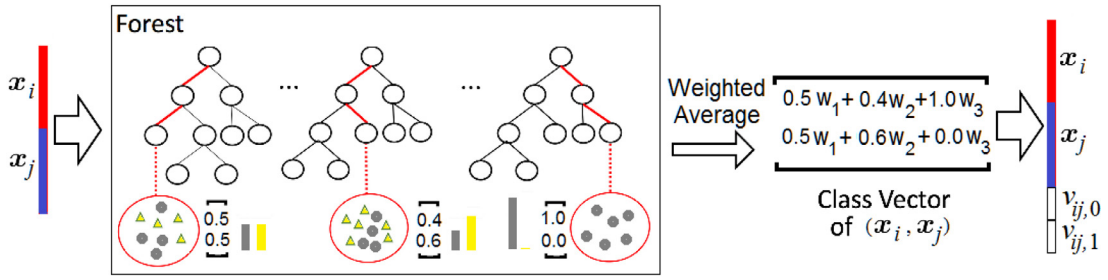

Fig. 3. An illustration of the class vector generation taking into account the weights.

$\mathbf{x}_i$ and $\mathbf{x}_j$ and to increase the distance between dissimilar $\mathbf{x}_i$ and $\mathbf{x}_j$. Therefore, we have to develop a way for training the SDF, i.e., for computing the weights for every forest and for every cascade level.

Now we have some values $v_{ij,c}^{(k)}$ of the corresponding class vector for every class. Let us analyze these values from the point of the SDF aim view by considering different cases of $(\mathbf{x}_i, \mathbf{x}_j)$.

Case 1. First, we consider the case when $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S}$ and $y_{ij} = 0$. However, we may have non-zero $v_{ij,c}^{(k)}$ for both classes. It is obvious that $v_{ij,0}^{(k)}$ (the average probability of class $c = 0$) should be as large as possible because $c = y_{ij} = 0$. Moreover, $v_{ij,1}^{(k)}$ (the average probability of class $c = 1$) should be as small as possible because $c \neq y_{ij} = 0$.

Case 2. We can similarly write conditions for the case when $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D}$ and $y_{ij} = 1$. In this case, $v_{ij,0}^{(k)}$ should be as small as possible because $c \neq y_{ij} = 1$, and $v_{ij,1}^{(k)}$ should be as large as possible because $c = y_{ij} = 1$.

In sum, we should increase (decrease) $v_{ij,c}^{(k)}$ if $c = y_{ij}$ ($c \neq y_{ij}$). In other words, we have to find the weights maximizing (minimizing) $v_{ij,c}^{(k)}$ when $c = y_{ij}$ ($c \neq y_{ij}$). The ideal case is when $v_{ij,c}^{(k)} = 1$ by $c = y_{ij}$ and $v_{ij,c}^{(k)} = 0$ by $c \neq y_{ij}$. However, the vector of weights has to be the same for every class, and it does not depend on a certain class. At first glance, we could find optimal weights for every individual forest separately from other forests. However, we should analyze simultaneously all forests because some vectors of weights may compensate those vectors which cannot efficiently separate $v_{ij,0}^{(k)}$ and $v_{ij,1}^{(k)}$.

7. The SDF training and testing

We apply the greedy algorithm for training the SDF, namely, we train separately every level starting from the first level such that every next level uses results of training at the previous level. The training process at every level consists of two parts. The first part aims to train all trees by applying all pairs of training examples. This part does not significantly differ from the training of the original deep forest proposed by Zhou and Feng [31]. The difference is that we use pairs of concatenated vectors $(\mathbf{x}_i, \mathbf{x}_j)$ and two classes corresponding to semantic similarity of the pairs. The second part is to compute the weights $w^{(t,k)}$, $t = 1, \ldots, T_k$. This can be done by minimizing the following objective function over M unit (probability) simplices in $\mathbb{R}^{T_k}$ denoted as Δ_k, i.e., over non-negative vectors $\mathbf{w}^{(k)} = (w^{(1,k)}, \ldots, w^{(T_k,k)}) \in \Delta_k$, $k = 1, \ldots, M$, that sum up to one:

$$\min_{\mathbf{w}} J_q(\mathbf{w}) = \min_{\mathbf{w}} \sum_{i,j} l(\mathbf{x}_i, \mathbf{x}_j, y_{ij}, \mathbf{w}) + \lambda R(\mathbf{w}). \tag{16}$$

Here $\mathbf{w}$ is a vector produced as the concatenation of vectors $\mathbf{w}^{(k)}$, $k = 1, \ldots, M$, $R(\mathbf{w})$ is a regularization term, λ is a hyper-parameter which controls the strength of the regularization. Let us denote the set of vectors $\mathbf{w}$ as Δ. We define the regularization term as

$$R(\mathbf{w}) = \|\mathbf{w}\|^2. \tag{17}$$

The loss function $l(\mathbf{x}_i, \mathbf{x}_j, y_{ij}, \mathbf{w})$ must satisfy the condition of the semantical similarity or dissimilarity for the pair $(\mathbf{x}_i, \mathbf{x}_j)$ taking into account the augmented features $v_{ij,0}^{(k)}$ and $v_{ij,1}^{(k)}$. In other words, it has to increase values of augmented features $v_{ij,0}^{(k)}$ corresponding to the class $c = 0$ and to decrease features $v_{ij,1}^{(k)}$ corresponding to the class $c = 1$ for semantically similar pairs $(\mathbf{x}_i, \mathbf{x}_j)$. Moreover, the loss function has to increase values of augmented features $v_{ij,1}^{(k)}$ corresponding to the class $c = 1$ and to decrease features $v_{ij,0}^{(k)}$ corresponding to the class $c = 0$ for dissimilar pairs $(\mathbf{x}_i, \mathbf{x}_j)$.

There are many loss functions taking into account conditions of the semantical similarity, for example, the hinge loss function, the squared hinge loss function, (see reviews provided by Bellet et al. [2] and Kulis [17]). However, our aim is to simplify the optimization problem (16) by looking for a convex loss function with respect to $\mathbf{w}$. Therefore, the next subsection describes one of the possible loss functions which satisfies the convexity condition and the condition of the semantical similarity.

7.1. Convex loss function

In order to efficiently solve the problem (16), the condition of the convexity of $J_q(\mathbf{w})$ in the domain of $\mathbf{w}$ should be fulfilled. One of the ways for determining the loss function l is to consider a distance $d(\mathbf{x}_i, \mathbf{x}_j)$ between two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ at the qth level. The use of the distance in the loss function is similar to the use of distances in the loss function for SNNs (see (6)). However, we do not have separate vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. We have one vector whose parts correspond to vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. Therefore, this is a distance between elements of the concatenated vector $(\mathbf{x}_i^{(-1)}, \mathbf{x}_j^{(-1)})$ obtained at $q-1$ level and between elements of augmented feature vectors $\mathbf{v}_{ij}^{(k)}$, $k = 1, \ldots, M$, of a special form. Let us consider the expression for the distance $d(\mathbf{x}_i, \mathbf{x}_j)$ in detail. The distance can be represented as a sum of $M+1$ terms. The first term denoted as $X_{ij}^{(q)}$ is the Euclidean distance between two parts of the output vector obtained at the previous level

$$X_{ij} = \sum_{l=1}^{m} \left(x_{i,l}^{(-1)} - x_{j,l}^{(-1)}\right)^2. \tag{18}$$

Here $x_{i,l}$ is the lth element of $\mathbf{x}_i$, m is the length of the input vector for the qth level or the length of the output vector for the level with the number $q-1$.

Let us consider elements $v_{ij,0}^{(k)}$ and $v_{ij,1}^{(k)}$ of the kth class vector now. We have to provide the distance between these elements as large as possible taking into account y_{ij}. In particular, if $y_{ij} = 0$, then we should decrease the difference $v_{ij,1}^{(k)} - v_{ij,0}^{(k)}$. If $y_{ij} = 1$, then we should decrease the difference $v_{ij,0}^{(k)} - v_{ij,1}^{(k)}$. Let us introduce the variable $z_{ij} = -1$ if $y_{ij} = 0$, and $z_{ij} = 1$ if $y_{ij} = 1$, i.e., $z_{ij} = 2y_{ij} - 1$.

Then the following expression characterizing the augmented features $v_{ij,0}^{(k)}$ and $v_{ij,1}^{(k)}$ can be written:

$$\left[\max\left(0, z_{ij}\left(v_{ij,0}^{(k)} - v_{ij,1}^{(k)}\right)\right)\right]^2. \tag{19}$$

It is interesting to point out that (19) coincides with the squared hinge loss function. It satisfies the condition of the semantical similarity or dissimilarity of augmented features.

Substituting (12) into (19), we get next M terms

$$\left[\max\left(0, \sum_{t=1}^{T_k} P_{ij}^{(t,k)} w^{(t,k)}\right)\right]^2, \quad k = 1, \ldots, M, \tag{20}$$

where

$$P_{ij}^{(t,k)} = z_{ij}\left(p_{ij,0}^{(t,k)} - p_{ij,1}^{(t,k)}\right). \tag{21}$$

Finally, we can write

$$d\left(\mathbf{x}_i, \mathbf{x}_j\right) = X_{ij} + \sum_{k=1}^{M}\left[\max\left(0, \sum_{t=1}^{T_k} P_{ij}^{(t,k)} w^{(t,k)}\right)\right]^2. \tag{22}$$

So, we have to maximize $d(\mathbf{x}_i, \mathbf{x}_j)$ with respect to $w^{(t,k)}$ under constraints (15). Since X_{ij} does not depend on $w^{(t,k)}$, then we consider the following objective function

$$J_q(\mathbf{w}) = \sum_{i,j}\sum_{k=1}^{M}\left[\max\left(0, \sum_{t=1}^{T_k} P_{ij}^{(t,k)} w^{(t,k)}\right)\right]^2 + \lambda\|\mathbf{w}\|^2. \tag{23}$$

The function $d(\mathbf{x}_i, \mathbf{x}_j)$ is convex in the interval [0, 1] of $w^{(t,k)}$. Then the objective function $J_q(\mathbf{w})$ as the sum of the convex functions is convex too with respect to weights.

The first important peculiarity of the optimization problem with the objective function (23) and constraints (15) is that it can be represented in the form of a standard quadratic optimization problem. Its second peculiarity is that it can be decomposed into M simple problems. Therefore, the next subsection illustrates how the considered peculiarities are realized.

7.2. Quadratic optimization problem

Let us consider the problem (23) under constraints (15) in detail. Introduce a new variable $\xi_{ij}^{(k)}$ defined as

$$\xi_{ij}^{(k)} = \max\left(0, \sum_{t=1}^{T_k} P_{ij}^{(t,k)} w^{(t,k)}\right). \tag{24}$$

Then problem (23) can be rewritten as

$$J_q(\mathbf{w}) = \min_{\xi_{ij}^{(k)}, \mathbf{w}} \sum_{i,j}\sum_{k=1}^{M}\left(\xi_{ij}^{(k)}\right)^2 + \lambda\|\mathbf{w}\|^2, \tag{25}$$

subject to (15)

$$\xi_{ij}^{(k)} \geq \sum_{t=1}^{T_k} P_{ij}^{(t,k)} w^{(t,k)}, \quad \xi_{ij}^{(k)} \geq 0, \quad (i,j) \in K, \ k = 1, \ldots, M. \tag{26}$$

We have obtained the standard quadratic optimization problem with linear constraints and variables $\xi_{ij}^{(k)}$ and $w^{(t,k)}$. It can be solved by using the well-known standard methods.

It is interesting to note that the optimization problem (25) and (26) can be decomposed into M problems of the form:

$$J_q(\mathbf{w}^{(k)}) = \min_{\xi_{ij}, \mathbf{w}^{(k)}} \sum_{i,j}\xi_{ij}^2 + \lambda\left\|\mathbf{w}^{(k)}\right\|^2, \tag{27}$$

subject to (15)

$$\xi_{ij} \geq \sum_{t=1}^{T_k} P_{ij}^{(t,k)} w^{(t,k)}, \quad \xi_{ij} \geq 0, \quad (i,j) \in K, \ k = 1, \ldots, M. \tag{28}$$

Indeed, by returning to problem (25) and (26), we can see that the subset of variables $\xi_{ij}^{(k)}$ and $w^{(t,k)}$ for a certain k and constraints for these variables do not overlap with the subset of similar variables for another k and the corresponding constraints. This implies that (25) can be rewritten as

$$J_q(\mathbf{w}) = \sum_{k=1}^{M}\min_{\xi_{ij}, \mathbf{w}}\sum_{i,j}\left(\xi_{ij}^{(k)}\right)^2 + \lambda\|\mathbf{w}\|^2, \tag{29}$$

and the problem can be decomposed.

So, we solve the problem (27) and (28) for every $k = 1, \ldots, M$ and get M vectors $\mathbf{w}^{(k)}$ which form the vector $\mathbf{w}$. The above means that the optimal weights are separately determined for individual forests.

7.3. A general algorithm for training and the SDF testing

In sum, we can write a general algorithm for training the SDF (see Algorithm 1). Its complexity mainly depends on the number of levels.

Algorithm 1 A general algorithm for training the SDF.

Require: Training set $S = \{(\mathbf{x}_i, \mathbf{x}_j, y_{ij}), \ (i,j) \in K\}$ consisting of N pairs; number of cascade levels Q

Ensure: $\mathbf{w}^{(q)}$, $q = 1, \ldots, Q$

```
Concatenate x_i and x_j for all pairs ofindices (i, j) ∈ K
Form the training set R = {(x_i, x_j), y_ij), (i, j) ∈ K} consisting of concatenated pairs
for q = 1, q ≤ Q do
    Train all trees at the qth level
    for k = 1, k ≤ M_q do
        Compute the weights w^(k) at theqth level from the kth quadratic optimization problem with the objectivefunction (27) and constraints (15) and (28)
    end for
    Concatenate w^(k), k = 1, ..., M, to get w at theqth level
    For every x_ij, compute v_ij^(k) at theqth level by using (12), k = 1, ..., M
    For every x_ij, form the concatenated vector x_ij for the next level by using (11)
end for
```

Having the trained SDF with computed weights $\mathbf{w}$ for every cascade level, we can make decision about the semantic similarity of a new pair of examples $\mathbf{x}_a$ and $\mathbf{x}_b$. First, the vectors make to be concatenated. By using the trained decision trees and the weights $\mathbf{w}$ for every level q, the pair is augmented at each level. Finally, we get

$$\mathbf{x}_{ab}^{(Q)} = \left(\mathbf{x}_a^{(Q)}, \mathbf{x}_b^{(Q)}\right) = \mathbf{v}_{ab}. \tag{30}$$

Here $\mathbf{v}_{ab}$ is the augmented part of the vector $\mathbf{x}_{ab}^{(Q)}$ consisting of elements from subvectors $\mathbf{v}_0$ and $\mathbf{v}_1$ corresponding to the class $c = 0$ and to the class $c = 1$, respectively. The original examples $\mathbf{x}_a$ and $\mathbf{x}_b$ are semantically similar if the sum of all elements from $\mathbf{v}_0$ is larger than the sum of elements from $\mathbf{v}_1$, i.e., $\mathbf{v}_0 \cdot \mathbf{1}^{\mathrm{T}} > \mathbf{v}_1 \cdot \mathbf{1}^{\mathrm{T}}$, where $\mathbf{1}$ is the unit vector. In contrast to the similar examples, the condition $\mathbf{v}_0 \cdot \mathbf{1}^{\mathrm{T}} < \mathbf{v}_1 \cdot \mathbf{1}^{\mathrm{T}}$ means that $\mathbf{x}_a$ and $\mathbf{x}_b$ are semantically dissimilar and $y_{ab} = 1$. We can introduce a threshold τ for a more robust decision making. The examples $\mathbf{x}_a$ and $\mathbf{x}_b$ are classified as semantically similar and $y_{ab} = 0$ if $\mathbf{v}_0 \cdot \mathbf{1}^{\mathrm{T}} - \mathbf{v}_1 \cdot \mathbf{1}^{\mathrm{T}} \geq \tau$. The case $0 \leq \mathbf{v}_0 \cdot \mathbf{1}^{\mathrm{T}} - \mathbf{v}_1 \cdot \mathbf{1}^{\mathrm{T}} \leq \tau$ can be viewed as undeterminable.

Table 2
A brief introduction about datasets.

Data set	Abbreviation	m	n	C
Parkinson	Park	23	197	2
Ecoli	Ecoli	8	336	8
Yeast	Yeast	8	1484	10
Ionosphere	Ion	34	351	2
Mammographic masses	MM	5	961	2
Haberman's Survival	HS	3	306	2
Breast Tissue	BT	9	106	6
Lung Cancer	LC	56	32	3
Indian Liver Patient	ILP	10	583	2
Seeds	Seeds	7	210	3

It is important to note that the identical weights, i.e., gcForest can be regarded as a special case of the SDF.

8. Numerical experiments

We compare the SDF with gcForest and other classifiers, including Bagging, Random Forest (RF) and the SNN, whose inputs are concatenated examples from several datasets. By using gcForest, we compare the SDF having computed (trained) weights with the SDF having identical weights. The SDF has the same cascade structure as the standard gcForest described in [31]. Each level (layer) of the cascade structure consists of 2 complete-random tree forests and 2 random forests. Three-fold cross-validation is used for the class vector generation. The number of cascade levels is automatically determined.

A software in Python implementing gcForest is available at https://github.com/leopiney/deep-forest. We modify this software in order to implement the procedure for computing optimal weights and weighted averages $v_{ij.c}^{(k)}$. Moreover, we use pairs of concatenated examples composed of individual examples as training and testing data. A version of the modified software in Python is available at http://pml.spbstu.ru/wp-content/uploads/2017/02/SDF.zip.

Every accuracy measure A used in numerical experiments is the proportion of correctly classified cases on a sample of data. To evaluate the average accuracy, we perform a cross-validation with 100 repetitions, where in each run, we randomly select N training data and $N_{\text{test}} = 2N/3$ test data. Different values for the regularization hyper-parameter λ have been tested, choosing those leading to the best results.

First, we compare the SDF with gcForest by using some public datasets from UCI Machine Learning Repository [20]. Table 2 is a brief introduction about these datasets, while more detailed information can be found from, respectively, the data resources. Table 2 shows the number of features m for the corresponding dataset, the number of examples n and the number of classes C.

In order to investigate how the number of decision trees impact on the classification accuracy, we study the SDF by different number of trees, namely, we take $T_k = T = 100,\ 400,\ 700,\ 1000$. It should be noted that Zhou and Feng [31] used 1000 trees in every forest.

For every analyzed dataset, we form a training set consisting of concatenated pairs of examples. If a pair is concatenated from examples belonging to the same class, then it is supposed that the corresponding pair is similar, otherwise it is dissimilar.

Results of numerical experiments for the Parkinsons dataset are shown in Table 3. It contains the accuracy measures obtained for gcForest (denoted as gcF) and the SDF as functions of the number of trees T in every forest and the number $N = 100, 500, 1000, 2000$ of pairs in the training set. It can be seen from Table 3 that the accuracy of the SDF exceeds the same measure of gcForest in most cases. At that, the difference is rather large for the small amount of training data. In particular, the largest differences between accuracy measures of the SDF and gcForest are observed by $T = 400$, 1000 and $N = 100$. Similar results of numerical experiments for the Ecoli dataset are given in Table 4. It is interesting to point out that the number of trees in every forest significantly impacts on the difference between accuracy measures of the SDF and gcForest. It follows from Table 4 that this difference is smallest by the large number of trees and by the large amount of training data. If we look at the last row of Table 4, then we see that the accuracy 0.915 obtained for the SDF by $T = 100$ is reached for gcForest by $T = 1000$. The largest difference between accuracy measures of the SDF and gcForest is observed by $T = 100$ and $N = 100$. The same can be seen from Table 3. This implies that the proposed modification of gcForest allows us to reduce the training time. Table 5 provides accuracy measures for the Yeast dataset. We again can see that the proposed SDF outperforms gcForest for most cases. It is interesting to note from Table 5 that the increasing number of trees in every forest may lead to reduced accuracy measures. If we look at the row of Table 5 corresponding to $N = 500$ pairs in the training set, then we can see that the accuracy measures by 100 trees exceed the same measures by larger numbers of trees. Moreover, the largest difference between accuracy measures of the SDF and gcForest is observed by $T = 1000$ and $N = 100$. Numerical results for the Ionosphere dataset are represented in Table 6. It follows from

Table 3
Dependence of the accuracy measures on the number of pairs N and on the number of trees T in every forest for the Parkinsons dataset.

T	100		400		700		1000	
N	gcF	SDF	gcF	SDF	gcF	SDF	gcF	SDF
100	0.530	0.545	0.440	0.610	0.552	0.575	0.440	0.550
500	0.715	0.733	0.651	0.673	0.685	0.700	0.700	0.730
1000	0.761	0.763	0.778	0.786	0.803	0.804	0.773	0.790
2000	0.880	0.881	0.884	0.895	0.875	0.891	0.887	0.893

Table 4
Dependence of the accuracy measures on the number of pairs N and on the number of trees T in every forest for the Ecoli dataset.

T	100		400		700		1000	
N	gcF	SDF	gcF	SDF	gcF	SDF	gcF	SDF
100	0.439	0.530	0.515	0.545	0.590	0.651	0.621	0.696
500	0.838	0.847	0.814	0.823	0.836	0.845	0.821	0.837
1000	0.844	0.853	0.890	0.917	0.888	0.891	0.863	0.865
2000	0.908	0.915	0.895	0.921	0.913	0.915	0.915	0.915

Table 5
Dependence of the accuracy measures on the number of pairs N and on the number of trees T in every forest for the Yeast dataset.

T	100		400		700		1000	
N	gcF	SDF	gcF	SDF	gcF	SDF	gcF	SDF
100	0.454	0.500	0.484	0.511	0.469	0.515	0.469	0.575
500	0.682	0.694	0.661	0.673	0.640	0.658	0.622	0.628
1000	0.708	0.711	0.713	0.723	0.684	0.710	0.735	0.737
2000	0.727	0.734	0.714	0.716	0.727	0.739	0.713	0.722

Table 6
Dependence of the accuracy measures on the number of pairs N and on the number of trees T in every forest for the Ionsphere dataset.

T	100		400		700		1000	
N	gcF	SDF	gcF	SDF	gcF	SDF	gcF	SDF
100	0.515	0.515	0.535	0.555	0.530	0.555	0.535	0.540
500	0.718	0.720	0.723	0.758	0.713	0.740	0.715	0.760
1000	0.820	0.830	0.837	0.840	0.840	0.860	0.885	0.895
2000	0.916	0.920	0.905	0.905	0.895	0.895	0.910	0.910

Table 7
Accuracy measures for different datasets by $N = 1000$ and $N = 2000$.

	$N = 1000$		$N = 2000$	
	gcF	SDF	gcF	SDF
MM	0.730	0.757	0.765	0.783
HS	0.533	0.580	0.555	0.594
BT	0.832	0.859	0.875	0.885
ILP	0.520	0.530	0.572	0.575
Seeds	0.886	0.892	0.956	0.968

Table 6 that the largest difference between accuracy measures of the SDF and gcForest is observed by $T = 1000$ and $N = 500$.

The numerical results for the above analyzed datasets show that the SDF significantly outperforms gcForest by small number of training data ($N = 100$ or 500). This is an important property of the SDF which are especially efficient when the amount of training data is rather small. At the same time, the largest accuracy measures are obtained when the amount of training data is large. Therefore, other datasets from Table 2 will be analyzed by $N = 1000$ or 2000. Moreover, we compare the SDF and gcForest by taking $T = 700$ trees in every random forest.

Accuracy measures for several datasets are given in Table 7 as functions of the number of training pairs $N = 1000$ and $N = 2000$. The Lung Cancer dataset is not represented in Table 7 because its number of original examples is 32, and $N = 496$ pairs of examples for training and testing can be obtained. The accuracy measures for the LC dataset are 0.642 by using gcForest and 0.686 by using the SDF.

It should be noted that the multi-grained scanning proposed in [31] was not applied to investigating the above datasets having relatively small numbers of features. The above numerical results have been obtained by using only the forest cascade structure.

In order to compare the SDF with other classifiers, including Bagging, the RF and the SNN, we take the best results from Tables 3–7 and compare them with Bagging the RF and the SNN under the same numbers of examples. We have to point out that the RF can be viewed as a special case of gcForest when its cascade consists of a single level, and the level contains a single RF. We cannot use the RF in its original form because we deal with the weakly supervised learning. Therefore, we apply the same technique of the paired examples in order to train and to test the RF. The same can be said about Bagging. The main difference between Bagging and the RF is that only a subset of features is randomly selected from all features in the RF, while all features are used for classification in Bagging.

In contrast to Bagging and the RF, the SNN aims to solve the weakly supervised learning problems, and pairs of training elements are inputs for the SNN. A software in Python implementing the SNN and using Tensorflow is available at https://github.com/ywpkwon/siamese_tf_mnist. Every subnetwork of the SNN consists of two hidden layers. The number of units in the first layer is defined by the number of features m of an analyzed dataset and is about $3m/2$. The number of units in the second hidden layer is about $m/3$. For example, the number of units for the Ionosphere dataset in the first hidden layer is 50. The second layer consists of 10 units. The parameter τ in the contrastive loss function of the SNN (see (6)) is taken in order to obtain the best results. For example, $\tau = 5$ for the Ionosphere dataset. The learning rate in the gradient descent scheme is 0.01.

Accuracy measures for several datasets by using gcForest, the RF, the SNN and the SDF are given in Table 8. It can be seen from Table 8 that the SDF outperforms all classifiers for all datasets. However, we have to point out that small training data have been considered. This is a reason why the SNN shows the worse results.

Table 8
Accuracy measures for different datasets by using the SDF, the gcForest, the RF and the SNN.

	gcF	Bagging	RF	SNN	SDF
Park	0.887	0.587	0.695	0.572	0.893
Ecoli	0.915	0.877	0.867	0.710	0.921
Yeast	0.735	0.712	0.702	0.652	0.739
Ion	0.916	0.886	0.912	0.690	0.920
MM	0.765	0.508	0.592	0.619	0.783
HS	0.555	0.494	0.488	0.564	0.594
BT	0.875	0.698	0.769	0.683	0.885
LC	0.642	0.580	0.583	0.595	0.686
ILP	0.572	0.502	0.510	0.582	0.575
Seeds	0.956	0.944	0.946	0.789	0.968

Table 9
Results of computing the t-test.

	p-value	mean	95% confidence interval
D_1	0.015	0.015	[0.004, 0.025]
D_2	0.006	0.118	[0.043, 0.192]
D_3	0.002	0.090	[0.043, 0.137]
D_4	0.006	0.118	[0.043, 0.192]

In the case of large training data, the SNN may be superior to the SDF and gcForest. Zhou and Feng [31] have pointed out the following two important factors which impact on the performance of gcForest and on the corresponding comparison of the algorithm with the neural networks:

1. The performance of gcForest can be significantly improved via task-specific tuning by selecting special structures of the forest cascade and the Multi-Grained Scanning substructure. Moreover, according to [31, Supplement to the paper], there are various types of gcForest, for example, gcForest (5grains) or gcForest (gbdt), which could improve the performance.
2. Deep neural networks as well as the SNNs have been studied for many years and a lot of architectures have been developed for improving their performance. Moreover, a huge number of methods and recommendations for tuning neural networks have been proposed. A lot of efficient software implementations of neural networks have been developed. The gcForest algorithm as well as its modifications are at an initial stage of investigation, and it is difficult to expect that we can obtain the superior results by using the SDF for all datasets.

In order to formally show the outperformance of the proposed SDF, we apply the t-test which has been proposed and described by Demsar [9] for testing whether the average difference in the performance of two classifiers is significantly different from zero. Since we use the differences between accuracy measures of the SDF and gcForest (D_1), the SDF and Bagging (D_2), the SDF and the RF (D_3), the SDF and the SNN (D_4), then we compare them with 0 which represents gcForest, Bagging, the RF and the SNN, respectively. The t statistics in this case is distributed according to the Student distribution with $10 - 1$ degrees of freedom. Results of computing the t-test for D_1, D_2, D_3, D_4, including a p-value, the 95% confidence interval for the mean, are shown in Table 9.

The t-tests demonstrate the clear outperforming of the SDF in comparison with Bagging the RF, gcForest, the SNN. It is interesting also to note that Bagging is superior to the RF for some datasets when concatenated pairs of examples are used. One can see from Table 2 that most datasets, for which Bagging outperforms the RF, have small numbers of features. Therefore, randomly selected subsets of features in the RF may lead to worse results.

In order to perform comparisons of multiple classifiers (the SDF, gcForest, Bagging, the RF, the SNN) we also apply the Friedman test and the Bonferroni–Dunn test for multiple comparison

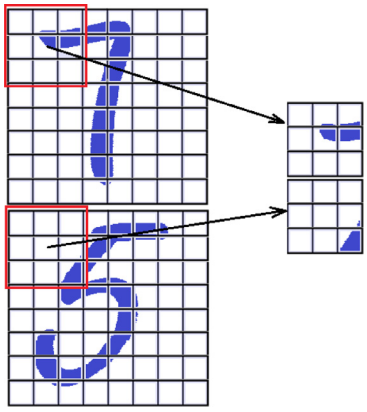

Fig. 4. The multi-grained scanning scheme for concatenated examples.

Table 10
Dependence of the accuracy measures on the number of pairs N and on the number of trees T in every forest for the MNIST dataset.

T	100		400		700		1000	
N	gcF	SDF	gcF	SDF	gcF	SDF	gcF	SDF
100	0.470	0.490	0.520	0.520	0.570	0.585	0.530	0.570
500	0.725	0.735	0.715	0.715	0.695	0.700	0.670	0.670
1000	0.757	0.770	0.755	0.760	0.775	0.780	0.830	0.840

with one control. According to the first test, we get the identical p-values 0.0016 for pairwise comparisons D_1, D_2, D_3 and the p-value 0.0114 for D_4. The Friedman tests also demonstrate the clear outperforming of the SDF in comparison with other classifiers. In contrast to the Friedman test, the Bonferroni–Dunn test shows that there is no significant difference between the SDF and the other methods except for the pair D_4 (the corresponding p-value is 0.028). This can be explained by the fact that the Bonferroni–Dunn test is too conservative.

When we deal with the large-scale data, the multi-grained scanning scheme should be use. In particular, for analyzing the well-known MNIST dataset, we used the same scheme for window sizes as proposed in [31], where feature windows with sizes $\lfloor d/16 \rfloor$, $\lfloor d/9 \rfloor$, $\lfloor d/4 \rfloor$ are chosen for d raw features. We study the SDF by applying the MNIST database which is a commonly used large database of 28×28 pixel handwritten digit images [19]. It has a training set of 60,000 examples, and a test set of 10,000 examples. The digits are size-normalized and centered in a fixed-size image. The dataset is available at http://yann.lecun.com/exdb/mnist/. The main problem in using the multi-grained scanning scheme is that pairs of the original examples are concatenated. As a result, the direct scanning leads to scanning windows covering some parts from every example belonging to a concatenated pair, which do not correspond the images themselves. Therefore, we apply the following modification of the multi-grained scanning scheme. Two identical windows simultaneously scan two concatenated images such that pairs of feature windows are produced due to this procedure, which are concatenated for processing by means of the forest cascade. Fig. 4 illustrates the used procedure. Results of numerical experiments for the MNIST dataset are shown in Table 10. It can be seen from Table 10 that the largest difference between accuracy measures of the SDF and gcForest is observed by $T = 1000$ and $N = 100$. It is interesting to note that the SDF as well as gcForest provide good results even by the small amount of training data. At that, the SDF outperforms gcForest in the most cases.

Another dataset used for comparison the SDF with the well-known classifiers is CIFAR-10 [16]. It is a dataset of 60,000 natural color images of size 32×32 pixels divided into 10 classes. The original dataset has 50,000 images for training and 10,000 images for testing. We form $N = 1500$ concatenated pairs of images such that 1000 pairs are used for training and 500 pairs for testing. The small number of pairs for experiments is caused by problems with solving the quadratic optimization problem (27) and (28). The number of trees in every random forest is 1500. The accuracy measures for the modified CIFAR-10 dataset are 0.538 by using gcForest and 0.552 by using the SDF. It can be seen that the SDF is superior to gcForest for this dataset. However, it should be noted that the number of training pairs is very small for comparison the SDF with the deep neural networks. Moreover, the SDF as well as gcForest in their recent form may be inferior to the well-known deep neural networks, for example, ResNet and AlexNet, due to reasons indicated by Zhou and Feng [31] and formulated above.

The main aim of the SDF is not to outperform the deep neural networks and their modifications, but to provide a competitive tool for dealing with small training samples when the neural networks may meet the overfitting problem due to a huge number of training parameters. At the same time, we expect that the intensive investigation of the deep forest will lead to highly efficient deep forest structures outperforming deep neural networks for large datasets.

We used PC Intel Core I5 CPUs (4 cores) for numerical experiments. For the MNIST dataset with 2000 pairs of examples and 1000 trees in every random forest, the training time is about 5 min. For the CIFAR-10 dataset with 2000 pairs of examples and 1000 trees in every random forest, the training time is about 14 min.

Generally, the computational complexity of the proposed SDF can be expressed through two parts. The first part totally coincides with the complexity of gcForest, which is considered by Zhou and Feng [31, Subsection 3.1] in detail. The second part is determined by the time required for solving the standard quadratic optimization problems. The complexity of the quadratic optimization problem is $O(H^3)$, where H^3 is the number of active constraints. The value of H is defined by the number N of pairs of training data and by the number of weights. If we suppose that all random forests have T trees, then the number of weights is T. It follows from (15) and (28) that there holds $H = 2N + T + 1$. If we take into account that the number of random forests in a level of the forest cascade is M, and the number of levels is Q, then we can write the computational complexity of solving all quadratic problems for computing the weights, which is equal to $O(MQH^3)$.

An interesting observation has been also made during numerical experiments. We have discovered that the variable z_{ij}, initially taking the values -1 for $y_{ij} = 0$ and 1 for $y_{ij} = 1$, can be viewed as a tuning parameter in order to control the number of the cascade levels used in the training process and to improve the classification performance of the SDF. One of the great advantages of gcForest is its automatic determination of the number of cascade levels. It is shown by Zhou and Feng [31], that the performance of the whole cascade is estimated on validation set after training a current level. The training procedure in gcForest terminates if there is no significant performance gain. It turns out that the value of z_{ij} significantly impact on the number of cascade levels if to apply the termination procedure implemented in gcForest. Moreover, we can adaptively change the values of z_{ij} with every level. It has been revealed that one of the best change of z_{ij} is $z_{ij}^{(q)} = 2z_{ij}^{(q-1)}$, where $z_{ij}^{(1)} = -1$ for $y_{ij} = 0$ and 1 for $y_{ij} = 1$. Of course, this is an empirical observation. However, it can be taken as a direction for further improving the SDF.

9. Conclusion

One of the implementations of the SDF has been represented in the paper. It should be noted that other modifications of the

SDF can be obtained. First of all, we can improve the optimization algorithm by applying a more complex loss function and computing optimal weights, for example, by means of the Frank–Wolfe algorithm [11]. We can use a more powerful optimization algorithm, for example, an algorithm proposed by Hazan and Luo [12]. Moreover, we do not need to search for the convex loss function because there are efficient optimization algorithms, for example, a non-convex modification of the Frank–Wolfe algorithm proposed by Reddi et al. [23], which allows us to solve the considered optimization problems. The trees and forests can be also replaced with other classification approaches, for example, with SVMs and boosting algorithms. However, the above modifications can be viewed as directions for further research.

It is important to point out that the SDF algorithm in its recent form does not allows us to investigate large datasets when the training set is very large. This difficulty stems from the use of the quadratic optimization problem (27) and (28). On the one hand, the quadratic programming allows us to get a unique solution. Moreover, there are many efficient methods for its solving. On the other hand, numbers of variables and constraints are limited in most algorithms of the quadratic optimization problem solving. Therefore, in order to consider large datasets, it is necessary to apply special algorithms which weakly depend on numbers of variables and constraints. This is also a direction for further research.

The linear combinations of weights for every forest have been used in the SDF. However, this class of combinations can be extended by considering non-linear functions of weights. Moreover, it turns out that the weights of trees can model various machine learning peculiarities and allow us to solve many machine learning tasks by means of gsForest. This is also a direction for further research. Numerical experiments have shown that it will be also very interesting to investigate classifiers different from the random forest, for example, Bagging, Forest CERN [1], etc. as components of the SDF.

It should be noted that the weights have been restricted by constraints of the form (15), i.e., the weights of every forest belong to the unit simplex whose dimensionality is defined by the number of trees in the forest. However, numerical experiments have illustrated that it is useful to reduce the set of weights in some cases. Moreover, this reduction can be carried out adaptively by taking into account the classification error at every level. One of the ways for adaptive reduction of the unit simplex is to apply imprecise statistical models, for example, the linear-vacuous mixture or imprecise ε-contaminated models proposed by Walley [25]. This study is also a direction for further research.

We have considered a weakly supervised learning algorithm when there are no information about the class labels of individual training examples, but we know only semantic similarity of pairs of training data. It is also interesting to extend the proposed ideas on the case of fully supervised algorithms when only the class labels of individual training examples are known. The main goal of fully supervised distance metric learning is to use discriminative information in distance metric learning to keep all the data samples in the same class close and those from different classes separated [21]. Therefore, another direction for further research is to adapt the proposed algorithm for the case of available class labels.

Acknowledgment

The authors would like to express their appreciation to the anonymous referees whose very valuable comments have improved the paper. The reported study was partially supported by RFBR, research project No. 17-01-00118.

References

[1] M.N. Adnan, M.Z. Islam, Forest CERN: a new decision forest building technique, in: Advances in Knowledge Discovery and Data Mining: 20th Pacific-Asia Conference, PAKDD 2016, Auckland, New Zealand, Part I, Springer International Publishing, 2016, pp. 304–315. Springer International Publishing.
[2] A. Bellet, A. Habrard, M. Sebban, A survey on metric learning for feature vectors and structured data, 2013, arXiv:1306.6709.
[3] S. Berlemont, G. Lefebvre, S. Duffner, C. Garcia, Siamese neural network based similarity metric for inertial gesture classification and rejection, in: Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on, volume 1, IEEE, 2015, pp. 1–6.
[4] L. Bertinetto, J. Valmadre, J.F. Henriques, A. Vedaldi, P.H.S. Torr, Fully-convolutional siamese networks for object tracking, 2016, arXiv:1606.09549v2.
[5] J. Bromley, J.W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Sackinger, R. Shah, Signature verification using a siamese time delay neural network, Int. J. Pattern Recognit Artif Intell. 7 (4) (1993) 737–744.
[6] H. Le, Capitaine. constraint selection in metric learning, 2016, arXiv:1612.04853v1.
[7] K. Chen, A. Salman, Extracting speaker-specific information with a regularized siamese deep network, in: Advances in Neural Information Processing Systems 24 (NIPS 2011), Curran Associates, Inc., 2011, pp. 298–306.
[8] S. Chopra, R. Hadsell, Y. LeCun, Learning a similarity metric discriminatively, with application to face verification, in: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, IEEE, 2005, pp. 539–546.
[9] J. Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[10] Y. Dong, B. Du, L. Zhang, Target detection based on random forest metric learning, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 8 (4) (2015) 1830–1838.
[11] M. Frank, P. Wolfe, An algorithm for quadratic programming, Naval Res. Logistics Q. 3 (31–2) (1956) 95–110.
[12] E. Hazan, H. Luo, Variance-reduced and projection-free stochastic optimization, in: Proceedings of the 33rd International Conference on Machine Learning, volume 48 of ICML'16, 2016, pp. 1263–1271.
[13] J. Hu, J. Lu, Y.-P. Tan, Discriminative deep metric learning for face verification in the wild, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2014, pp. 1875–1882.
[14] D. Kedem, S. Tyree, K. Weinberger, F. Sha, G. Lanckriet, Non-linear metric learning, in: F. Pereira, C.J. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 2582–2590.
[15] G. Koch, R. Zemel, R. Salakhutdinov, Siamese neural networks for one-shot image recognition, in: Proceedings of the 32nd International Conference on Machine Learning, volume 37, 2015, pp. 1–8. Lille, France
[16] A. Krizhevsky, G. Hinton, Learning Multiple Layers of Features from Tiny Images, Tech. Rep, Computer Science Department, University of Toronto, 2009.
[17] B. Kulis, Metric learning: a survey, Found. Trends Mach. Learn. 5 (4) (2012) 287–364.
[18] L. Leal-Taixe, C. Canton-Ferrer, K. Schindler, Learning by tracking: Siamese cnn for robust target association, 2016, arXiv:1604.07866.
[19] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[20] M. Lichman, UCI machine learning repository, University of California, School of Information and Computer Science, Irvine, CA, 2013 [http://archive.ics.uci.edu/ml].
[21] Y. Mu, W. Ding, Local discriminative distance metrics and their real world applications, in: 2013 IEEE 13th International Conference on Data Mining Workshops (ICDMW), IEEE, 2013, pp. 1145–1152.
[22] M. Norouzi, D. Fleet, R. Salakhutdinov, K.Q. Weinberger, Hamming distance metric learning, in: P. Bartlett, F.C. Pereira, C. Burges, L. Bottou (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1070–1078.
[23] S.J. Reddi, S. Sra, B. Poczos, A. Smola, Stochastic frank-wolfe methods for non-convex optimization, 2016, arXiv:1607.08254v2.
[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[25] P. Walley, Statistical reasoning with imprecise probabilities, Chapman and Hall, London, 1991.
[26] B. Wang, L. Wang, B. Shuai, Z. Zuo, T. Liu, C.K. Luk, G. Wang, Joint learning of convolutional neural networks and temporally constrained metrics for tracklet association, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, IEEE, 2016, pp. 1–8.
[27] D.H. Wolpert, Stacked generalization, Neural Netw. 5 (2) (1992) 241–259.
[28] C. Xiong, D. Johnson, R. Xu, J.J. Corso, Random forests for metric learning with implicit pairwise position dependence, 2012, arXiv:1201.0610v1.
[29] Z. Xu, K.Q. Weinberger, O. Chapelle, Distance metric learning for kernel machines, 2012, arXiv:1208.3422.
[30] L. Zheng, S. Duffner, K. Idrissi, C. Garcia, A. Baskurt, Siamese multi-layer perceptrons for dimensionality reduction and face identification, Multimed. Tools Appl. 75 (9) (2016) 5055–5073.
[31] Z.-H. Zhou, J. Feng, Deep forest: towards an alternative to deep neural networks, 2017, arXiv:1702.08835v2.