Semi-Supervised Self-Training Feature Weighted Clustering Decision Tree and Random Forest

ZHENYU LIU¹,², TAO WEN¹,², WEI SUN², AND QILONG ZHANG¹

¹College of Computer Science and Engineering, Northeastern University, Shenyang 110819, China
²Department of Computer Science and Technology, Dalian Neusoft University of Information, Dalian 116023, China

Corresponding author: Zhenyu Liu (liuzhenyu@neusoft.edu.cn)

This work was supported by the National Nature Science Foundation of China under Grant 61772101 and Grant 61602075.

ABSTRACT A self-training algorithm is an iterative method for semi-supervised learning, which wraps around a base learner. It uses its own predictions to assign labels to unlabeled data. For a self-training algorithm, the classification ability of the base learner and the estimation of prediction confidence are very important. The classical decision tree as the base learner cannot be effective in a self-training algorithm, because it cannot correctly estimate its own predictions. In this paper, we propose a novel method of node split of the decision trees, which uses weighted features to cluster instances. This method is able to combine multiple numerical and categorical features to split nodes. The decision tree and random forest constructed by this method are called FWCDT and FWCRF respectively. FWCDT and FWCRF have the better classification ability than the classical decision trees and forests based on univariate split when the training instances are fewer, therefore, they are more suitable as the base classifiers in self-training. What’s more, on the basis of the proposed node-split method, we also respectively explore the suitable prediction confidence measurements for FWCDT and FWCRF. Finally, the results of experiment implemented on the UCI datasets show that the self-training feature weighted clustering decision tree (ST-FWCDT) and random forest (ST-FWCRF) can effectively exploit unlabeled data, and the final obtained classifiers have better generalization ability.

INDEX TERMS Semi-supervised learning, self-training, decision tree, random forest, node splits.

I. INTRODUCTION

In the case of sufficient labeled instances, supervised learning methods are very effective. However, it is very expensive and time-consuming to obtain labeled instances in many practical applications, such as object detection, document and web-page categorization and so on. On the contrary, unlabeled data in these fields is easy to obtain, which makes semi-supervised learning method widely used in these fields [1]–[3].

Semi-supervised learning algorithms use labeled and unlabeled data to construct classifiers. The goal is to improve learning performance as much as possible by using a large amount of unlabeled data when there is only a small amount of labeled data. In recent decades, a large number of semi-supervised learning algorithms have been proposed. Among them, some algorithms use generative model for classification [4]–[6], and some extend traditional SVM for semi-supervised learning [7]–[9], and others are based on graph methods [10]–[12].

Both surveys [13] and [14] refer to that wrapper methods are among the oldest and most widely known algorithms for semi supervised learning. Wrapper methods are able to be simply divided into self-training methods [15]–[17], [21] and co-training methods [18]–[20], in terms of the classifiers constructed by single-view or multi-view data.

A self-training method constructs a classifier on single-view data and uses this classifier to select pseudo-labeled data for itself; while the co-training methods construct multiple classifiers on multi-view data and select pseudo-labeled data for each other.

The wrapper methods improve the performance of the classifiers by continuously expanding the labeled training sets. The performance of the wrapper algorithms strongly depends on the selected newly-labeled data at each iteration of the training procedure. Therefore, both self-training methods and co-training methods, it is very important for them to correctly measure the prediction confidence. In the current popular supervised learning algorithms, such as Bayesian methods and neural networks, which get the prediction results expressed by a probability distribution; the prediction results of the ensemble learning algorithms are the combination
of multiple individual predictions; other supervised learning algorithms, like decision trees, only give one class label as the prediction result. If the wrapper methods use the above two types of supervised learners as base classifiers, the probability distribution and the votes in the ensembles can be respectively used as the prediction confidence. However, because the decision tree uses a class label as the prediction of a test object, which makes it impossible to directly determine which predictions are more credible. Therefore, when decision tree is used as base classifier, it is necessary to redefine the reasonable measure of prediction confidence.

Decision trees and forests are widely used supervised learning algorithms, how to make these algorithms play important roles in semi-supervised learning has also been studied by many scholars. Such as, in paper [21], the authors used the No-pruning, Grafting, Laplacian Correction and Naive Bayes to reform the classical decision tree algorithms, and proposed a distance-based measure combined with the improved decision tree learners. Experiments showed that the decision trees and forests modified by their method are more suitable for self-training scheme than the classical algorithms. In order to overcome the shortcomings of traditional co-training needing sufficient and redundant features, Zhou and Li proposed tri-training [19] and co-forest [20], respectively. They used bootstrap method to sample training subsets from the original labeled data to train three or more decision trees as base classifiers. Experiments in [19, 20] showed that tri-training and co-forest can achieve better results even when the features are not particularly redundant. And the two methods are more suitable for common data mining scenarios than the traditional co-training methods [18]. In [34], the authors proposed a method to split the nodes of decision trees using both labeled and unlabeled data, and their semi-supervised random forest was applied to the field of computer vision.

In this paper, we propose a node-split method for feature weighted clustering decision tree, and this method-based decision tree (FWCDT) and the corresponding ensemble (FWCRF) have the better generalization abilities, compared with the classical decision trees(DTs) and the classical random forest(RF), especially when training instances are sparse. The underlying principle of all these algorithms is to give each leaf node a confidence measurement, which is used to determine whether the leaf node is split or not. The confidence measurement is usually a probability value between 0 and 1, and the decision tree uses a class label as the prediction of a test object. The confidence measure is usually a probability value between 0 and 1, and the decision tree uses a class label as the prediction of a test object. The confidence measure is usually a probability value between 0 and 1, and the decision tree uses a class label as the prediction of a test object. The confidence measure is usually a probability value between 0 and 1, and the decision tree uses a class label as the prediction of a test object. Therefore, when decision tree is used as base classifier, it is necessary to redefine the reasonable measure of prediction confidence.

The rest of the paper is organized as follows: the second section will introduce the node-split method and the construction methods of FWCDT and FWCRF; the constructions of ST-FWCDT and ST-FWCRF will be given in the third section; the fourth section will report the experiments on UCI datasets [22]; the last section is our conclusions.

II. THE PROPOSED NODE-SPLIT METHOD

The linear split methods of decision trees are generally divided into two types: axis-parallel linear splits (univariate splits) [23], [24] and oblique linear splits (multivariate splits) [27]–[29]. The axis-parallel methods use only one feature to split an internal node, and generate a hyperplane which is orthogonal to the coordinate axis, while the oblique methods combine multiple features to generate an oblique hyperplane. Theoretically, in the p-dimensional feature space with n instances, there are \( \binom{n}{p} \cdot p! \) different axis-parallel hyperplanes. However, there are \( 2^p \cdot \binom{n}{p} \) different oblique hyperplanes. This means that in the same feature space, it is more difficult to search for the proper oblique hyperplanes compared with the axis-parallel hyperplanes. Generally speaking, the construction speed of axis-parallel tree is faster, and the generalization ability of oblique decision tree is stronger.

The ensemble classifiers (e.g. random forest [25] and AdaBoost [26]) by combining multiple decision trees can obtain the better generalization performance than the single learner methods. Breiman [25] pointed out that the generalization ability of ensemble depends not only on the accuracy of individual decision trees, but also on individual diversity. And the individual diversity is usually obtained in terms of the subsampling (e.g. bagging, boosting) or the random subspace [30]. When the instances and features are sufficient, the univariate decision trees with fast construction speed are more suitable to be treated as the base classifiers of the ensembles. Instead, if the instances or features are insufficient, the multivariate split methods will bring about more diverse individuals for the ensembles. In [31] and [32], the nodes are split by linear discriminant analysis (LDA), ridge regression and MPSVM, so as to construct the oblique random forests. Their experiments show that these ensembles have exceeded the performance of classical RF in some application fields. However, the oblique split methods they adopt can only be used for numerical data, therefore, in the above paper, they contain discrete and unordered categorical features, they need to convert the categorical features into numerical features by CRIMCOORD [33] or OneHot Encoder, which may bring the new deviation to the classification problem and reduce the generalization ability of the classifier.

The feature weighted clustering split method proposed in this paper is based on the combination of multiple variables, which is suitable to deal with the data of numerical features, categorical features and mixed features. The decision trees constructed by this method not only have better generalization ability than the univariate decision trees, but also can bring more diverse individuals to the ensembles by combining the “subsampling” and “subspace”, so that it ensures the better generalization ability of the ensembles compared with the classical RF in the case of fewer training instances or features.
The proposed split method is based on clustering assumption. The clustering assumption states that the instances belonging to the same cluster belong to the same class. Different from the most multivariate split methods [27]–[29], we adopt the multi-way splits for the internal nodes, that is, in one split, multiple hyperplanes are generated simultaneously, and the feature space is divided into several disjoint regions. Our method can be simply described as: looking for k anchor points in the feature space, instances are divided into different clusters according to the nearest anchor points. Fig.1 illustrates the 5-way splits of the two-dimensional feature space.

K-means is a widely used clustering algorithm. We adopt the improved K-means to split nodes, and the cluster centers are the anchor points mentioned above.

Let \( L \) be a set of \( n \) labeled instances, which will be divided into \( k \) disjoint subsets, \( C_1, C_2, \ldots, C_k \). Firstly, class centroids are treated as the initial centers of \( k \) clusters, respectively, represented by \( \mu_1, \mu_2, \ldots, \mu_k \). Secondly, for each instance \( x_i \), we use (1) to calculate the cluster center of the instance:

\[
\text{label}_i = \arg \min_{1 \leq j \leq k} \| x_i - \mu_j \|. \quad (1)
\]

In (1), \( \| x_i - \mu_j \| \) indicates the distance between \( x_i \) and \( \mu_j \). After each instance obtains the corresponding label, we use (2) to update the center of each cluster.

\[
\mu_j = \frac{1}{n_j} \sum_{i=1}^{n_j} x_i \quad x_i \in C_j. \quad (2)
\]

Calculate repeatedly (1) and (2) until the preset number of iterations is reached or the positions of all cluster centers change no longer.

The original K-means is an unsupervised clustering algorithm, which is suitable for unlabeled data. And the optimization goal is to minimize the within groups sum of squared errors (WGSS).

The goal of node splits is to reduce the class impurity of current nodes as much as possible. Note that the two goals are not same. Therefore, we use the labels of the instances to calculate the correlation between each feature and the label. When calculating the distance from an instance to a cluster center, we give a larger weight to the feature strongly related to the label, enlarging the contribution of the feature to the distance. Otherwise, we give a smaller weight, reducing the contribution of the uncorrelated feature to the distance. In this way, the optimization goal of K-means algorithm is close to that of node splits.

Relief-F [36] is an extension of the famous feature selection algorithm Relief [35] on the multi-classification problems. The algorithm uses (3) to calculate the weight of each feature:

\[
W(A) = W(A) - \frac{k_m}{\| \text{diff}(A, R, H_j) \|} + \sum_{C \in \text{class}(R)} \left( \frac{p(C)}{1 - p(\text{class}(R))} \right) \times \frac{k_m}{\| \text{diff}(A, R, M_j(C)) \|} \quad (3)
\]

In (3), \( R \) represents a randomly selected instance, and \( H_j \) indicates the \( j \)th nearest same-class neighbor of \( R, m \) and \( k_m \) represent the number of sampling and the number of nearest neighbors respectively. \( p(C) \) represents the proportion of class \( C \) instances to the total instances. \( M_j(C) \) represents the \( j \)th nearest neighbor to \( R \) in class \( C \). \( \text{diff}(A, R_1, R_2) \) calculates the difference between two instances on the feature \( A \) in (4), as follows:

\[
\text{diff}(A, R_1, R_2) = \begin{cases} 
\frac{|R_1[A] - R_2[A]|}{\max(A) - \min(A)} & \text{if } A \text{ is numerical} \\
0 & \text{if } A \text{ is categorical And } R_1[A] = R_2[A] \\
1 & \text{if } A \text{ is categorical And } R_1[A] \neq R_2[A] 
\end{cases} \quad (4)
\]

Both Relief-F and K-means are based on the measures of distance. In this paper, we use the Relief-F to weight the features, and use (5) to calculate the distances between the instances and the cluster centers in the K-means clustering process.

\[
\| x_i - \mu_j \| = \sqrt{\sum_{l=1}^{p} w_l \cdot (x_i[l] - \mu_j[l])^2}, \quad (5)
\]

in (5), \( p \) represents the number of features, \( w_l \) represents the weight of the \( l \)th feature obtained by the Relief-F, and \( x_i[l] \) and \( \mu_j[l] \) represent the values of instance \( x_i \) and cluster center \( \mu_j \) on the \( l \)th feature, respectively.

In order to illustrate the role of feature weighting in node splits, we use dataset iris to carry out a simple experiment: 150 instances of iris belong to three classes, and each has 50 instances. We use the original K-means to cluster, where 10 instances by mistake. As is seen the confusion matrix in Table 1 for specific result.

Then, we use the Relief-F to calculate the weights of four features, which are 0.09, 0.14, 0.34 and 0.39 respectively.

| Child node 1 | 50 | 0 | 0 |
| Child node 2 | 0 | 44 | 4 |
| Child node 3 | 0 | 6 | 46 |

TABLE 1. Split results of unweighted features for IRIS.
In the process of clustering, the distances between instances and cluster centers are calculated in (5). We obtain 6 misclassified instances, and the specific results are shown in Table 2.

To sum up, our proposed method can be simply described as two steps: feature weighting and clustering. The specific process is shown in algorithm 1.

The algorithm 1 is used to construct the individual trees in the forest. If it is used to construct a single tree classifier, the second step should be deleted, and at this time, \( F \) contains all the features. In addition, \( I_{\text{max}} \) represents the maximum number of iterations in the clustering process, which is taken as a parameter of the algorithm.

In step 3 of algorithm 1, Relief-F is used to get the weights. Time complexity of Relief-F is \( O(m \cdot p \cdot n \cdot \log_2 k_{nn}) \), where \( p \) is the feature number, \( n \) is the instance number, \( m \) is the sampling number and \( k_{nn} \) is the nearest neighbor number. In this paper, \( m \) is set \( \log_2 n \), \( k_{nn} \) is set 1, \( \log_2 k_{nn} \) is negligible, so the time complexity of Relief-F in this paper is \( O(p \cdot n \cdot \log n) \).

Steps 7 to 12 are the clustering process, and the time complexity is \( O(1 \cdot p \cdot n \cdot k) \), where \( k \) is cluster number and \( I \) is iteration number. When we use algorithm 1 to split nodes, the max iterations \( I_{\text{max}} \) is assigned 6, which means that time complexity may reach \( O(6 \cdot p \cdot n \cdot k) \) in the worst case.

Considering the above two parts, the time complexity of the algorithm 1 is \( O((6k + \log_2 n) \cdot p \cdot n) \). Compared with the time complexity of the classical axis parallel splits, there’s an extra \( k \).

In OC1 [27] of classic oblique decision tree, the time complexity of split one node is \( O(p \cdot n^2 \cdot \log_2 n) \) in the worst case. In [29], the time complexities of HHHCART(A) and HHHCART(D) are \( O((p+n \cdot \log_2 n) \cdot p^2 \cdot k) \) and \( O((p+\log_2 n) \cdot p \cdot n \cdot k) \) respectively. In [28], the speed of FDT for splitting node is close to or even better than that of axis parallel split method. The time complexity of this method is about \( O(p^2 \cdot n) \), and unfortunately, it can only be applied in binary classification problems.

In summary, when \( k \) is smaller, the efficiency of the proposed split method is close to that of the classical axis parallel split method, and is better than that of most oblique split methods.

### B. CATEGORICAL FEATURES

The node split method described in the above subsection can be directly applied to the numerical features. For categorical features, Relief-F can still be used to realize the feature weighting. In the clustering, the representations of cluster centers and the calculation of the distances from instances to cluster centers need to be redefined.

\( K \)-modes [37] is a variation of \( K \)-means algorithm on the categorical features. In this algorithm, the modes of the categorical features are used to represent the cluster centers, instead of the means of the numerical features. The distances between the instances and the cluster centers on the categorical features are calculated by (4). Although \( K \)-modes is simple and easy to implement, but the distances calculated by this method are not accurate enough, and when there are multiple modes of a feature, the selections of different modes may get the opposite conclusions.

Here is an example. Suppose there are two clusters \( C_1 \) and \( C_2 \) described by two categorical features \( A_1 \) and \( A_2 \), and each cluster contains 10 instances as is shown in Table 3. The modes of \( C_1 \) and \( C_2 \) for \( A_1 \) are a11, which makes \( A_1 \) useless for distinguishing the distances between instances and the clusters. There are two modes for \( A_2 \) in \( C_1 \) and \( C_2 \) respectively. Suppose that there is an instance \( q \) = (a11, a21), if \( \mu_1 \) = (a11, a21) is selected as the center of \( C_1 \) and \( \mu_2 \) = (a11, a23) for \( C_2 \), distance between \( q \) and \( \mu_1 \) is 0 and distance between \( q \) and \( \mu_2 \) is 1, hence, \( q \) is nearer to \( C_1 \). If \( \mu_1 \) = (a11, a22) is selected as the center of \( C_1 \) and \( \mu_2 \) = (a11, a21) for \( C_2 \), distance between \( q \) and \( \mu_1 \) is 1 and distance between \( q \) and \( \mu_2 \) is 0, hence, \( q \) is nearer to \( C_2 \).
To avoid the less precision and the ambiguity of distance measure on the modes, we define “instance-cluster center” distance using the conditional probability.

Let $L$ be a set of categorical data described by $p$ categorical features. Number of instances in $L$ is $n$ and instances are partitioned into $k$ clusters. There are $d^{(k)}$ different values $\omega_1, \omega_2, \ldots, \omega_{d^{(k)}}$ for the $l$th feature $A_l$ of the $j$th cluster $C_j$.

**Definition 1:** Let $C_{j,l}$ represents the set of instances with value of $x_l$ on the feature $A_l$ in $C_j$, where $x_l \in \omega_1, \omega_2, \ldots, \omega_{d^{(l)}}$. The condition probability is as follows:

$$P(x_l|j) = \frac{|C_{j,l}|}{|C_j|}.$$  

(6)

And summary of all values of $A_l$ in $C_j$ is a $S_{l,j}$ as follows:

$$S_{l,j} = \{P(\omega_1|j), P(\omega_2|j), \ldots, P(\omega_{d^{(l)}},j)\}.$$  

(7)

**Definition 2:** We represent center of $C_j$ as a vector:

$$\mu_j = (S_{1,j}, S_{2,j}, \ldots, S_{d^{(l)},j}).$$  

(8)

**Definition 3:** Let $\text{diff}(A_l, \omega, S_{l,j})$ represent the distance between value $\omega$ and $S_{l,j}$ for $A_l$:

$$\text{diff}(A_l, \omega, S_{l,j}) = \begin{cases} 
1 - P(\omega|j) & \text{if } \omega \in \{\omega_1, \omega_2, \ldots, \omega_{d^{(l)}}, j\} \\
1 & \text{others}
\end{cases}.$$  

(9)

**Definition 4:** Let $\text{dis}_c(x_l, \mu_j)$ represent the distance between instance $x_l$ and center $\mu_j$:

$$\text{dis}_c(x_l, \mu_j) = \sum_{l=1}^{p} (w_l \cdot \text{diff}(A_l, x_{l,j}, S_{l,j})).$$  

(10)

According to (10), in the above example, the weights of two features are 1. The distances between instance $q=(a11, a21)$ and two cluster centers $\mu_1$ and $\mu_2$ are $0.7=0.1+0.6$ and $1.2=0.6+0.6$ respectively. It means that $q$ is closer to $C_1$, which is in accordance with the human’s intuition.

To cluster categorical data, we use (8) to replace (2) in step 5 and step 9 of Algorithm 1, and (10) to replace (5) in step 8.

In mixed data, each cluster center contains two parts: the means of the numerical variables, and the summaries of categorical variables. While the “instance-cluster center” distance contains two parts too. We use (11) to calculate the distances from instances to clustering centers, where, $\text{dis}_n$ and $\text{dis}_c$ are obtained by the (5) and (10), respectively.

$$\text{dis}(x_l, \mu_j) = (1 - \gamma) \cdot \text{dis}_n(x_l, \mu_j) + \gamma \cdot \text{dis}_c(x_l, \mu_j).$$  

(11)

Here, $\text{dis}_n(x_l, \mu_j)$ and $\text{dis}_c(x_l, \mu_j)$ represent the distance of numerical feature subset and the distance of categorical feature subset, respectively; $\gamma$ is a real number between $0$ and $1$, which is used to adjust the proportion of two terms in the (11). When the $\text{split}$ function is used to generate a single tree classifier, the GINI index is used as the evaluation criterion. And we choose $\gamma$ which makes GINI index decrease the most in $\{0.1, 0.2, \ldots, 0.9\}$, while split function is used to generate individual trees in forest, $\gamma$ following uniform distribution is randomly generated for each split in order to generate more diverse individual trees.

**C. CONSTRUCT FWCDT AND FWCRF**

FWCDT adopts the top-down and recursive manner to grow completely, namely, the splits stop only when all instances in the node are the same class or the feature subset currently used cannot distinguish these instances. The specific process is shown in algorithm 2.

The function $\text{create\_tree}$ in algorithm 2 is used to create FWCDT and individual trees in FWCRF. When it is used to create the FWCDT, the input training set $L_s$ contains all labeled instances in the training set. If it is used to construct an individual tree in the FWCRF, $L_s$ is a subset of the original training set $L$ via bootstrap sampling.

After sampling, the set $L$ is divided into the subset $L_o$ and $L_s$. $L_o$ composes of instances that are not sampled is called out-of-bag dataset. For an individual tree, according to (12), we use out-of-bag $L_o$ as test set to evaluate the confidence grade $\alpha$ of each leaf node.

$$\alpha = (acc + 1)/\text{(acc + err + 2)},$$  

(12)

in (12), the $acc$ and $err$ represent the number of correctly classified instances and the number of misclassified instances in $L_o$ on a leaf node, respectively. Algorithm 3 shows how to build FWCRF.

After FWCRF is constructed, the combination of individual predictions is realized by (13).

$$\text{Fo}(x) = \text{arg max}_{1 \leq j \leq k} \sum_{l=1}^{s_l} \alpha_l \cdot I(T_l(x) = j),$$  

(13)
the prediction confidence of FWCDT. Semi-supervised learning [39], here we also use it to measure.
In the process of prediction, when an unlabeled instance reaches a leaf node through assignments, we use the average of fuzziness obtained from each round of assignment as the prediction fuzziness for the instance, which is used to measure the prediction confidence, as (14). The lower the prediction fuzziness is, the higher the prediction confidence is.

\[
fuzziness(x) = \frac{1}{n_s} \sum_{i=1}^{n_s} \frac{\text{dis}(i, x, \mu_{\text{nearest1}})}{\text{dis}(i, x, \mu_{\text{nearest2}})},
\]

in (13), the \( k \) is the number of classes; \( F_0(x) \) and \( T_i(x) \) represent the predictions for instance \( x \) by the forest and the \( i \)th tree respectively, and \( \alpha_i \) represents the confidence grade of the leaf node which \( x \) falls into, and \( I(\cdot) \) is the indicator function. Compared with the majority voting, our combination prediction function only uses the confidence grade of the corresponding leaf node instead of the constant 1.

III. SELF-TRAINING SEMI-SUPERVISED FOR FWCDT AND FWCRF

Self-training algorithm wraps around a base classifier. Firstly, a classifier is trained on the current labeled data (training step); secondly, the classifier is used to predict labels for unlabeled data (prediction step); some instances with high prediction confidence are selected and marked with the pseudo labels, and they are put into the labeled dataset (selection step). Repeat the three steps (training, prediction and selection) until a stop condition reached. Among them, selection step is important, because selection of incorrect predictions will propagate to produce further classification errors. In the rest of this section, we will introduce the prediction confidence measurement and instance selection method of the corresponding self-training algorithm, considering that FWCDT and FWCRF are used as base classifiers respectively.

A. SELF-TRAINING FWCDT

Fuzziness is a common phenomenon in our daily life activities because many events are not crisp in nature rather they are fuzzy. The term fuzziness was first proposed by Lotfi A. Zadeh in 1965 in association with his great invention fuzzy set theory [40]. The theory has been used in semi-supervised learning [39], here we also use it to measure the prediction confidence of FWCDT.

Inspired by fuzzy C-means[38], we measure the membership grade of instance \( x_i \) in \( C_j \) using the distance between \( x_i \) and cluster center \( \mu_j \). And for an instance, the closer it is to the cluster center, the higher its membership grade is, at the same time, the D-values of membership grades can also reflect the fuzziness of the instance assignment. For example, in Fig.2, \( \mu_1 \), \( \mu_2 \) and \( \mu_3 \) indicate three cluster centers, and the distances between instances (\( x_1 \) and \( x_2 \)) and the center (\( \mu_1 \)) are closer than that between instances (\( x_1 \) and \( x_2 \)) and the centers (\( \mu_2 \) and \( \mu_3 \)), so from the perspective of hard clustering, both two instances belong to \( C_1 \). While from the perspective of fuzzy clustering, the distance between \( x_1 \) and \( \mu_1 \) is much closer than that between \( x_1 \) and the centers (\( \mu_2 \) and \( \mu_3 \)), and the distance between \( x_2 \) and \( \mu_1 \) is slightly closer than that between \( x_2 \) and \( \mu_2 \). This case shows that the assigning fuzziness is higher for \( x_2 \) than \( x_1 \).

When FWCDT is used to predict an unlabeled instance, the instance needs to be continuously assigned until it reaches a leaf node from the root node. In each assignment, we need to calculate the distances between the instance and the centers of all the child nodes (clusters) of the current node. The instance falls into the corresponding branch with the nearest distance. At the same time, we take the ratio of the nearest distance to second nearest distance as the assigning fuzziness. For example, in Fig.2, the distances between \( x_1 \) and cluster centers (\( \mu_1 \), \( \mu_2 \) and \( \mu_3 \)) are 0.1,1 and 2 respectively, so the assigning fuzziness is 0.1/1=0.1 for \( x_1 \); the distances between \( x_2 \) and cluster centers (\( \mu_1 \), \( \mu_2 \) and \( \mu_3 \)) are 0.4,0.5 and 1.7 respectively, so the assigning fuzziness is 0.4/0.5=0.8 for \( x_2 \).

In the process of prediction, when an unlabeled instance reaches a leaf node through assignments, we use the average of fuzziness obtained from each round of assignment as the prediction fuzziness for the instance, which is used to measure the prediction confidence, as (14). The lower the prediction fuzziness is, the higher the prediction confidence is.
and \( \text{dis}(i, x, \mu_{\text{nearest}2}) \) respectively represent the nearest cluster center distance and the second nearest cluster center distance of \( x \) in the \( i \)th round. The distance is the weighted distance expressed by (5) and (10).

In order to verify the effectiveness of the prediction confidence measurement we proposed for FWCDT, we use glass dataset to implement experiment. There are 214 instances in the glass dataset, which are divided into 6 classes. First of all, we randomly select 23 instances (about 10%), and use algorithm 2 to train a classifier for predicting the remaining 191 instances, and calculate the fuzziness of each prediction according to (14), and obtain 59.16% accuracy (113 instances were predicted correctly), the prediction fuzziness distribution is between 0.031 and 0.966. Then, we sort the instances in ascending order by the prediction fuzziness, and find that all the top-19 instances (about 10% of the total) are absolutely correct; the top-38 instances (about 20% of the total) are predicted 37 correctly, with an accuracy of 97.37%; the top-57 instances (about 30% of the total) are predicted 49 correctly, with an accuracy of 85.96%.

In iterations of self-training FWCDT, we use the current labeled data to establish a classifier to predict the current unlabeled data, and sort the instances order by the prediction fuzziness. We select the instances whose prediction fuzziness is less than the threshold \( \theta_f \) in top-\( \lceil |U| \cdot r \rceil \) instances (\( U \) represents the initial unlabeled dataset, \( r \) and \( \theta_f \) are the two parameters of the algorithm, which are taken value as \([0, 1]\)). After the selected instances are labeled, they are removed from the unlabeled dataset and added to the labeled dataset. The iterations will end when the unlabeled set is empty or no instance with the prediction fuzziness less than \( \theta_f \) in the current unlabeled set. In the proposed algorithm, our experimental results show that when the parameters \( r \) and \( \theta_f \) are 0.2 and 0.5 respectively, the good results can be obtained on most datasets. As seen in algorithm 4 for a description of self-training FWCDT.

**Algorithm 4** Self-Training FWCDT

**Input:** Labeled data \( L \), Unlabeled data \( U \), Maximum unlabeled instance selection ratio \( r = 0.2 \), Fuzziness threshold of instance prediction \( \theta_f = 0.5 \)

**Output:** Generate final hypothesis \( H \)

1. \( s_{\text{max}} = \lceil |U| \cdot r \rceil \);
2. While \( U \) is not empty Do
3. \( H \leftarrow \text{create}\_\text{tree}(L)\);
4. Use \( H \) to predict each instance in \( U \), and utilize (14) to calculate prediction fuzziness.
5. Sort Newly-Labeled instances based on the prediction fuzziness.
6. In top-\( s_{\text{max}} \) instances, construct the set \( S \) composed of the instances whose prediction fuzziness is less than \( \theta_f \).
7. If \( S \) is empty Then break While
8. \( U = U - S; L = L \cup S; \)
9. End While
10. Return \( H \)

where, \( c \) represents the class of \( x \) predicted by the ensemble, \( c \in \{1, 2, \ldots, k\} \).

In this work, we present a lexicographic method to measure the prediction confidence of FWCRF by combining individual prediction agreement with average prediction fuzziness. For \( x \) predicted as class \( c \), the average prediction fuzziness is the mean of the ones in individual trees where \( x \) is predicted as class \( c \).

For the next example, let the prediction agreement of \( x_1 \) and \( x_2 \) be \( a_1 \) and \( a_2 \), respectively, and average prediction fuzziness be \( f_1 \) and \( f_2 \); \( t_0 \) is threshold associated with prediction agreement, and is equal to 0.05 in this paper, if \( |a_1 - a_2| > t_0 \), we only consider the agreement. Otherwise, we need to compare \( f_1 \) and \( f_2 \), further.

In iterations of self-training FWCRF, we adopt the lexicographic method to sort the instances and use prediction agreement threshold \( \theta_a \) to filter instances. In addition, when building FWCRF, we use the out-of-bag data of each tree to estimate its individual accuracy. In an iteration, if the average individual accuracy of the current FWCRF is lower than that of the previous iteration, we will stop the loop and regard the FWCRF of the previous iteration as final hypothesis. Algorithm 5 describes the construction process of self-training FWCRF.

**IV. EXPERIMENTS**

In order to verify the performance of the proposed algorithms, we implement self-training FWCDT and FWCRF with C++ language, and compare them with the algorithms [21,17] in some classification tasks.

**A. DATASET**

Eighteen UCI datasets [22] are used in the experiments. Information on these datasets are tabulated in Table 4.
For comparing the performance of $k$ classifiers on $N$ datasets, we first apply Friedman’s test as a nonparametric test equivalent to the repeated measures ANOVA. Under the null hypothesis Friedman’s test states that all the algorithms are equal and the rejection of this hypothesis implies differences among the performance of the algorithms. It ranks the algorithms based on their performance for each dataset separately, it then assigns the rank 1 to the best performing algorithm, rank 2 to the second best, and so on. In case of ties average ranks are assigned to the related algorithms. Let $r_i$ represent the average rank of the $i$th algorithm, then the mean and variance of $r_i$ are $(k+1)/2$ and $(k^2 - 1)/12$, respectively. Friedman’s statistic

$$
\chi^2_F = \frac{12N}{k(k+1)} \left( \sum_{i=1}^{k} r_i^2 - \frac{k(k+1)^2}{4} \right)
$$

(17)

is distributed according to $\chi^2$ with $k-1$ degree of freedom. Because the original Friedman’s test is so conservative, the statistic

$$
F_F = \frac{(N-1)\chi^2_F}{N(k-1) - \chi^2_F}
$$

(18)

is usually used. This statistic is distributed according to the $F$-distribution with $(k-1)$ and $(k-1)(N-1)$ degrees of freedom.

If the null-hypothesis is rejected(performance of the classifiers are statistically significant), the Nemenyi post-hoc test can be used to check whether the performance of two among $k$ classifiers is significantly different. The performance of two classifiers is significantly different if the corresponding average ranks differ by at least the critical difference

$$
CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}},
$$

(19)

where critical values $q_\alpha$ are based on the Studentized range statistic divided by $\sqrt{2}$. $\alpha$ is the significance level and is equal to 0.05 in this paper.

### B. STATISTICAL TEST

For comparing the performance of $k$ classifiers on $N$ datasets, we first apply Friedman’s test as a nonparametric test equivalent to the repeated measures ANOVA. Under the null hypothesis Friedman’s test states that all the algorithms are equal and the rejection of this hypothesis implies differences among the performance of the algorithms. It ranks the algorithms based on their performance for each dataset separately, it then assigns the rank 1 to the best performing algorithm, rank 2 to the second best, and so on. In case of ties average ranks are assigned to the related algorithms. Let $r_i$ represent the average rank of the $i$th algorithm, then the mean and variance of $r_i$ are $(k+1)/2$ and $(k^2 - 1)/12$, respectively. Friedman’s statistic

$$
\chi^2_F = \frac{12N}{k(k+1)} \left( \sum_{i=1}^{k} r_i^2 - \frac{k(k+1)^2}{4} \right)
$$

(17)

is distributed according to $\chi^2$ with $k-1$ degree of freedom. Because the original Friedman’s test is so conservative, the statistic

$$
F_F = \frac{(N-1)\chi^2_F}{N(k-1) - \chi^2_F}
$$

(18)

is usually used. This statistic is distributed according to the $F$-distribution with $(k-1)$ and $(k-1)(N-1)$ degrees of freedom.

If the null-hypothesis is rejected(performance of the classifiers are statistically significant), the Nemenyi post-hoc test can be used to check whether the performance of two among $k$ classifiers is significantly different. The performance of two classifiers is significantly different if the corresponding average ranks differ by at least the critical difference

$$
CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}},
$$

(19)

where critical values $q_\alpha$ are based on the Studentized range statistic divided by $\sqrt{2}$. $\alpha$ is the significance level and is equal to 0.05 in this paper.

### C. SELF-TRAINING WITH A SINGLE CLASSIFIER

The performance of ST-FWCDT is compared with two self-training decision trees in paper [21], i.e., ST-C4G and ST-NB. ST-C4G and ST-NB use C4.4graft (C4.4graft is a combination of No-pruning and Laplace correction) and Naive Bayesian Tree as the self-training base classifiers respectively, and these methods use the probability estimates
at the leaves of decision trees to measure the prediction confidence.

Tables 5, 6 and 7 show the classification accuracy of three self-training decision trees on 18 datasets under different label ratios. The initial in the tables are the results obtained by using only labeled data, and the final are the results obtained by combining labeled and unlabeled data. The last lines in the tables are the average accuracies. Under 10% label rate, the average accuracy of the final hypothesis of ST-FWCDT is about 5.82% higher than that of the initial hypothesis, while the average accuracies of ST-C4G and ST-NB are only increased by 3.21% and 4.4%, respectively. Under 20% label rate, the results of ST-FWCDT, ST-C4G and ST-NB are improved by 3.29%, 3.01% and 3.26% respectively. In general, the three self-training decision trees can effectively exploit unlabeled data to improve the hypotheses under all the label rates. However, with the increase of label rate, the improvements of the three self-training decision trees keep decreasing.

Compared with the three self-training decision trees, ST-FWCDT has the best performance under different label rates, however, with the increase of label rate, the gaps between ST-FWCDT and the other two classifiers are becoming smaller. For example, under 10% label rate, the D-value between the average final accuracy of ST-FWCDT and that of ST-C4G is 9.18%; under 20% and 40% label rate, the D-values are 5.17% and 3.94%. Comparing the accuracies of the initial hypotheses of three self-training decision trees, similar results appear again. For example, under 10% label rate, the D-value between the average initial accuracy of ST-FWCDT and that of ST-C4G is 6.57%; under 20% and 40% label rate, the D-values are 4.89% and 3.91%. The experimental results indicate that ST-FWCDT performs better than the other two in the case of less label data. For instance, under 10% label rate, the initial hypothesis of ST-FWCDT on the dataset zoo achieves an accuracy of 89.41% and the final accuracy of 94.55%, while the initial and final accuracies of ST-C4G are only 60.89% and 67.94%, respectively. There are only 101 instances in zoo, which is a 7-classification problem, and in this case, 10% label rate means that there are only 1-2 labeled instances on each class. In the case of using few training instances, the FWCDT based on the multivariate splits generates a better initial hypothesis than C4.4graft. A better initial hypothesis ensures that we can exploit unlabeled data more effectively in the self-training process, so that a better final hypothesis can be obtained.

Table 8 shows the average ranks of initial and final accuracies obtained by three self-training decision trees on 18 datasets under different label rates, as well as the corresponding statistics \( F \). With three algorithms and 18 datasets, \( F \) follows the \( F \)-distribution with 2 and 34 degrees of freedom. The critical value of \( F(2, 34) = 0.05 \) is 3.2759. The six values in the last column of Table 8 are all greater than 3.2759, so we reject the null-hypotheses. The Nemenyi post-hoc test is carried out. According to (19), the critical difference \( CD = 0.781333 \), where \( q_{0.05} = 2.344 \). Based on the
average ranks in Table 8, under 10\% label rate, the performance of the initial hypothesis of ST-FWCDT is significantly better than that of ST-C4G and the performance of the final hypothesis of ST-FWCDT is significantly better than that of the other two. Under 20\% and 40\% label rates, the initial and final hypotheses of ST-FWCDT are significantly better than those of ST-C4G, while there was no significant difference between ST-NB and ST-FWCDT.

\section*{D. SELF-TRAINING WITH AN ENSEMBLE CLASSIFIER}

The performance of ST-FWCRF is compared with ST-Stacking [17] and ST-RNB [21]. ST-Stacking is a hybrid self-training system that combines a Support Vector Machine, a Decision Tree, a 3NN Learner and a Bayesian algorithm using a Stacking variant methodology.

The authors in the paper [17] compared 20 semi-supervised learning algorithms on 41 datasets, and the experimental results show that ST-Stacking is the best one. The ST-RNB proposed in [21] is a self-training system using RNB as a base classifier. Here, RNB is an ensemble constructed by using the random subspace method whose individuals use Naive Bayesian trees. In our experiments, the number of individuals of ST-FWCRF and ST-RNB is set to 100, and ST-RNB randomly selects half number of the features when splitting each node.

Table 9, 10 and 11 respectively report the initial and final accuracies of three self-training ensembles on 18 datasets under different label rates. Under 10\% label rate, the average accuracy of the final hypothesis of ST-FWCRF is about 6.07\% higher than that of the initial hypothesis, the average accuracies of ST-Stacking and ST-RNB are approximately increased by 3.79\% and 3.82\%. Under 20\% label rate, the results of the three methods are approximately improved by 3.38\%, 1.7\% and 2.9\%. Under 40\% label rate, the results of the three algorithms are approximately improved by 1.37\%, 0.59\% and 1.66\%. In general, all three ensembles can effectively exploit unlabeled data to improve the hypotheses under all the label rates.

Compared with the average accuracies of the initial and final hypotheses, ST-FWCRF has the best performance under different label rates. Similar to ST-FWCDT, ST-FWCRF has more advantages than the other two ensembles in the case of less training instances.

The results of the statistical tests are reported in Table 12. Under 20\% and 40\% label rates, there is no significant difference in the initial hypotheses generated by the three
ensures \((F_\nu < 3.2759)\). The Nemenyi post-hoc test is implemented on the average ranks of other four groups. The performance of the initial hypothesis of ST-FWCRF under 10% label rate and the performance of the final hypothesis of ST-FWCRF under 40% label rate are significantly better than these of ST-Stacking. The performance of the final hypotheses of ST-FWCRF outperformed significantly the other two under 10% and 20% label rates.

**V. CONCLUSION**

Semi-supervised classification methods are particularly relevant to scenarios where labeled data is scarce. At the beginning of self-training, only a few labeled instances can be used to generate initial hypothesis. The classification ability of initial hypothesis has a great influence on the one of final hypothesis. In this paper, a new node-split method of decision tree is proposed. This method consists of two steps: (I) use Relief-F algorithm to calculate the weight of each feature, and (II) use clustering algorithm based on weighted distance to split the instances. In addition, in order to directly use the categorical features, we define the “centers” of the categorical features, and propose the calculation method of the distance between “two points” regarding the categorical features. The FWCDT and FWCRF constructed by this node-split method can obtain better classification ability than the classical decision trees and random forests under the condition of the less training instances. Therefore, FWCDT and FWCRF are very suitable to be used as base classifiers for self-training.

Instance selection is very important for self-training methods, because selection of incorrect predictions will propagate to produce further classification errors. In this paper, we use the distance-based fuzziness to measure the confidence grade of the prediction in FWCDT, and the combination of individual prediction agreement and average prediction fuzziness to measure the prediction confidence in FWCRF. We use FWCDT and FWCRF as base classifiers to construct ST-FWCDT and ST-FWCRF. In the iterations, the proposed prediction confidence measurements are used to sort the predictions, and the instances with a higher confidence is selected to be added to the labeled dataset. On UCI datasets, we compare ST-FWCDT and ST-FWCRF with the self-training algorithms in [17], [21]. The experimental results show that ST-FWCDT and ST-FWCRF can effectively exploit unlabeled data, and the final classifiers have better generalization abilities.

Classification of high-dimensional data is a challenge. Feature selection and dimensionality reduction can avoid curse of dimensionality. However, some feature selection and dimensionality reduction methods may fail in the case of serious lack of labeled data. In our future work, we focus on semi-supervised learning for high dimensional data.

**REFERENCES**

[1] S. Honari, P. Molchanov, S. Tyree, P. Vincent, C. Pal, and J. Kautz, “Improving landmark localization with semi-supervised learning,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, Jun. 2018, pp. 1546–1555.

[2] M. Bahl and M. Batoche, “Drug-target interaction prediction in drug repositioning based on deep semi-supervised learning,” in *Proc. Int. Conf. Comput. Intell. Appl.* (CIA), Oran, Algeria, 2018, pp. 302–313.

[3] J. Liu, J. Yao, M. Bagheri, and R. M. Summers, “A semi-supervised CNN learning method with pseudo-class labels for vascular calcification detection on low dose CT scans,” in *Proc. Comput.-Aided Diagnosis Int. Soc. Opt. Photon.*, San Diego, CA, USA, 2019, Art. no. 109501L.

[4] B. M. Shahshahani and D. A. Landgrebe, “The effect of unlabeled samples in reducing the small sample size problem and mitigating the Hughes phenomenon,” *IEEE Trans. Geosci. Remote Sens.*, vol. 32, no. 5, pp. 1087–1095, Sep. 1994.

[5] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell, “Text classification from labeled and unlabeled data using EM,” *Mach. Learn.*, vol. 39, nos. 2–3, pp. 103–134, 2000.

[6] F. G. Cozman, J. Cohen, and M. C. Cirilo, “Semi-supervised learning of mixture models,” in *Proc. 20th Int. Conf. Mach. Learn.*, Washington, DC, USA, 2003, pp. 99–106.

[7] K. Bennett and A. Demiriz, “Semi-supervised support vector machines,” in *Proc. Conf. Adv. Neural Inf. Process. Syst. (NIPS)*, Cambridge, MA, USA, 1999, pp. 368–374.

[8] T. Joachims, “Transductive inference for text classification using support vector machines,” in *Proc. 16th Int. Conf. Mach. Learn.*, San Francisco, CA, USA, 1999, pp. 200–209.

[9] Y.-F. Li and Z.-H. Zhou, “Towards making unlabeled data never hurt,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 1, pp. 175–188, Jan. 2015.

[10] M. Belkin, P. Niyogi, and V. Sindhwani, “Manifold regularization: A geometric framework for learning from labeled and unlabeled examples,” *J. Mach. Learn. Res.*, vol. 7, pp. 2399–2434, Dec. 2006.

[11] A. Subramanya and P. P. Talukdar, “Graph-based semi-supervised learning,” *Synth. Lect. Artif. Intell. Mach. Learn.*, vol. 8, no. 4, pp. 1–125, Feb. 2014.

[12] S. Yu, X. Yang, and W. Zhang, “PKGCN: Prior knowledge enhanced graph convolutional network for graph-based semi-supervised learning,” *Int. J. Mach. Learn. Cybern.*, vol. 10, no. 11, pp. 3115–3127, Aug. 2019.

[13] X. Zhu, “Semi-supervised learning literature survey,” *Dept. Comput. Sci.*, Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep. TR 1530, 2008.

[14] J. van Engelen and H. Hoos, “A survey on semi-supervised learning,” *Mach. Learn.*, vol. 109, no. 2, pp. 373–440, Nov. 2019.

[15] D. Yarowsky, “Unsupervised word sense disambiguation rivaling supervised methods,” in *Proc. ACL*, Cambridge, MA, USA, 1995, pp. 189–196.

[16] B. Wang, B. Spencer, C. X. Ling, and H. Zhang, “Semi-supervised self-training for sentence subjectivity classification,” in *Proc. Conf. Can. Soc. Comput. Stud. Intell.*, Canadian AI, 2008, pp. 344–355.

[17] S. Karlos, N. Fazakis, S. Kotsiantis, and K. Sgarbas, “Self-trained stacking for semi-supervised learning,” *Int. J. Artif. Intell. Tools*, vol. 26, no. 2, Apr. 2017, Art. no. 1750001.

[18] A. Blum and T. Mitchell, “Combining labeled and unlabeled data with co-training,” in *Proc. 11th Annul. Conf. Comput. Learn. Theory*, Madison, WI, USA, 1998, pp. 92–100.

[19] Z.-H. Zhou and M. Li, “Tri-training: Exploiting unlabeled data using three classifiers,” *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1529–1541, Nov. 2005.

[20] M. Li and Z.-H. Zhou, “Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples,” *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 37, no. 6, pp. 1088–1098, Nov. 2007.

[21] J. Tanha, M. van Someren, and H. Afsharnasheh, “Semi-supervised self-training for decision tree classifiers,” *Int. J. Mach. Learn. Cybern.*, vol. 8, no. 1, pp. 355–370, Feb. 2017.

[22] M. Lichman, (2013). UCI Machine Learning Repository. [Online]. Available: http://archive.ics.uci.edu/ml
[23] J. R. Quinlan, *C4.5 Programs for Machine Learning*, San Mateo, CA, USA: Morgan Kaufmann, 1992.

[24] W. Buntine, "Learning classification trees," *Statist. Comput.*, vol. 2, no. 2, pp. 63–73, Jun. 1992.

[25] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[26] F. Provost and R. E. Schapire, "A decision-theoretic generalization of online learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[27] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *J. Artif. Intell. Res.*, vol. 2, no. 1, pp. 1–32, Aug. 1996.

[28] A. López-Chau, J. Cervantes, L. López-García, and P. García-Lamont, "Fisher’s decision tree," *Expert Syst. Appl.*, vol. 40, no. 16, pp. 6283–6291, Nov. 2013.

[29] D. C. Wickramarachchi, B. L. Robertson, M. Reale, C. J. Price, and J. Brown, "HHCART: An oblique decision tree," *Comput. Statist. Data Anal.*, vol. 96, pp. 12–23, Apr. 2016.

[30] T. Kam Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, Aug. 1998.

[31] R. Katuwal and P. N. Suganthan, "Enhancing multi-class classification of random forest using random vector functional neural network and oblique decision surfaces," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Rio, Brazil, Jul. 2018, pp. 5–10.

[32] W. Y. Loh and Y. S. Shih, "Split selection methods for classification trees," *Statistica Sinica*, vol. 7, no. 4, pp. 815–840, May 1999.

[33] X. Liu, M. Song, D. Tao, Z. Liu, L. Zhang, C. Chen, and J. Bu, "Random forest construction with robust semisupervised node splitting," *IEEE Trans. Image Process.*, vol. 24, no. 1, pp. 471–483, Jan. 2015.

[34] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proc. 9th Int. Workshop Mach. Learn. (ML)*, Aberdeen, U.K., 1992, pp. 249–256.

[35] L. A. Zadeh, "Probability measures of fuzzy events," *J. Math. Anal. Appl.*, vol. 23, no. 2, pp. 421–427, Aug. 1968.