Two-Pair Splits Resampling in Decision Trees

Tzu-Cheng Chuang and Saul B. Gelfand

November 12, 2009

Abstract

A new type of decision tree, two pair splits tree, is introduced and applied in classification problem. Usually there are two types of split, univariate split and multivariate split. In previous research, some researchers chose to use either univariate split or multivariate split for the internal node based on 5x2 cross validation F test. In our design, some of the internal nodes have two different kinds of split, and some of them have only one type of split. When we encounter two types of split at that node, we double the copies of data samples. During testing, one data sample might fall in several decision regions. The final distribution of the data sample is obtained by aggregating the distribution of the regions that it lands in. We compare our method with CART, J48, and Random Forest Tree. Among 6 datasets, our proposed method outperforms than CART and J48. It also performs better than Random Forest on one data set. We further improve our algorithm by including bootstarapping and random subspace.

1 Introduction

Decision trees are one of the most frequently used data mining tools. They have been widely used in a lot of applications, such as remote sensing[18], handwritten recognition[17] and star/galaxy discrimination[13]. The main idea of decision tree is divide-and-conquer, and it is constructed by recursively partitioning the feature space. This procedure involves three steps: splitting data into two or more groups at the internal node, determining when to stop splitting, and assigning class label to the terminal nodes.

Several different types of decision trees have been developed so far. We can have two splits or multiway splits[7, 12] at one node. For the tree having only two splits at each node, we call it binary tree. Having multiway split can make the tree become shallow. The splitting method can be based on a single coordiante(univariate split) or oblique hyperplane (multivariate split). If the instance space is like Figure 1, it apparently shows that multivariate split needs only one splitting rule. This generates smaller tree size than univariate split tree does. Suppose the m^{th} featuer of a data sample is x_m. Let's say $s_1$is the univariate split, and s_2 is the multivariate split. The data sample is sent to either left child node or right child node based on the answer from the following:

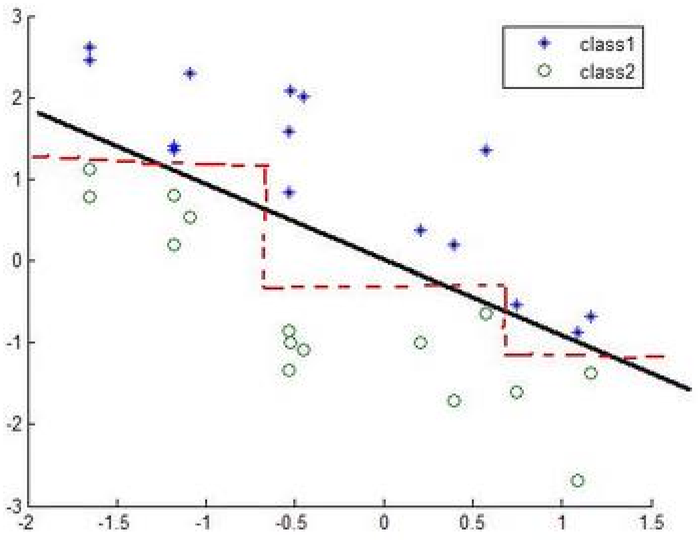


Figure 1: An example of two dimensional data space; "*":class 1,"o":class 2.

s_1 :Is $x_m \le c$?

s_2:Is $\sum a_m x_m \le c$?

Some trees have applied the mixed type of univariate split and multivariate split. These types of trees are called omnivariate trees[19, **?**]. They use some measures to decide whether the node should adopt either univariate split or multivariate split. In omnivariate trees developed by Yildiz, they use 5x2 cross validation F test[1] to decide which model should be used at that node. If the multivariate split statistically significantly generates lower classification error than univariate split does, multivariate split is used at that node. The rationale behind this is that we should adopt easier univariate split if there is no huge improvement for multivariate split.

One of the first and best-known examples is CART system[5]. Breiman uses Gini index or twoing rule as the splitting criterion. The other well-known one is C4.5[14] which uses information gain or gain ratio as the splitting criterion. Both methods use the idea of impurity measure. They compute the initial impurity at one node. By using greedy search along the coordinate of the features, they find the splitting point which maximizes the reduction of impurity. The reduction of impurity is defined by the following:

$\Delta Impurity = Impurity(initial) - \sum_i \frac{|T_i|}{|T|} \times Impurity(T_i)$

where $T_i$is one of the branch from node T, and $|T_i|$ is the number of data samples in T_i.

While computing the splitting criteria, the above methods are impurity based. There are also some other different methods[15], such as distance measure, orthogonality criterion or Kolmogorov-Smirnov Criterion. The splitting point is chosen when it minimizes or maximizes the criterion.

When we decide to use the multivariate split, we need to find the best coefficients which satisfy a criterion. There are several ways[6] to do it, such as recursive least square, pocket algorithm, thermal training, explicit reduction of impurity[5], and Fisher's linear discriminat[19]. The previous methods except for Fisher's linear use iterative method to update the coefficients. This kind of search needs to avoid local minimum and consumes a lot of time. However, Fisher's linear discriminant is only good for two class problems. When we deal with multi-class problems, we usually need to use selection method[20] or exchange method[8] to divide the classes into two groups. From the two groups, we can compute the mean, scatter matrices and the coefficients. This analytical method makes the computation faster.

Significant improvements in classification accuracy have resulted from voting among different independent classifiers. In order to grow an ensemble of trees, we have several methods to reshuffle the data set and generate new decision trees. Later on, the aggregation of the classification from the new resampled data is done by consensus among several classifiers. Bagging[3] is the abbreviation of Bootstrap aggregating. This method is done by choosing one data sample from the training data one at a time until the number of chosen data samples is the same as the original training size. Some data samples might be duplicated in the new generated data set. After several independent training sets are generated, trees are grown by those resampled training sets. Boosting[16] updates weights for each data smaple each iteration and several trees are grown on the updated training sets. Random subspace[10] method randomly chooses some of the features as a new input. Random forest[4] combines the idea of bagging and random subspace. It generates new input which is done by randomly selecting features from original features and also applies bootstrapping to shuffle the training instances. The final consensus can be done by simple majority voting, least square error weighting, Bayesian average or double-layer hierarchical combining[11].

The ensemble method can be used at an internal node [2]. The author constructs several splitting models with one type. The type of splitting can be multilayer perceptron, linear multivariate perceptron or Fisher's linear discriminant function. The data sample sent to either left child node or right child node is based on the ensemble result, positive or negative sign.

2 Consensus on Decision Regions

For a basic univariate binary tree, the data space is split into two regions along one coordinate at one internal node. After a series of splits, the regions are partioned into several regions. The data is represented as $L = \{\overrightarrow{x_i}, y_i\}, \overrightarrow{x_i} = [x_{i1}, ...x_{im}, ..., x_{iM}]^T, m \in \{1, ..., M\}, i \in \{1, ..., N\}$Each region is assigned with one class label $j(j \in \{1, ..., J\})$. Suppose there are total T terminal nodes, then there are total $T-1$ splits. The region for each terminal node is indicated as

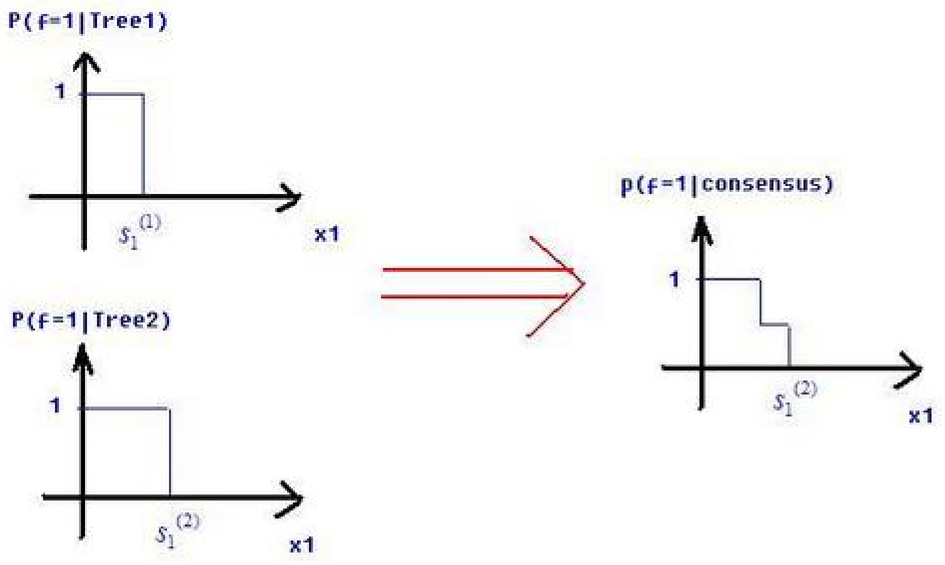


Figure 2: The distribution for the data sample classified by 2 different trees.

follows:

$R_1 = \{x_1 \leqslant s_1, x_2 \leqslant s_2, ..., x_M \leqslant s_M\}$

$R_2 = \{x_1 > s_1, x_2 \leqslant s_2, ..., x_M \leqslant s_M\}$

continue this process until we get R_T. After the training is done, these decision regions $R_1, ..., R_T$are mutually exclusive and collectively exhaustive. This means that when a testing data sample needs to be classified, it will only land in one of these T regions. Based on the region it lands in, we assume this testing data sample has the same distribution as the trained data which lands in the same region. In this region, we assign the class which gives us maximum probability. However, the testing data sample might have slightly different distribution as the distribution of the training data.

Let's focus on a simple binary-class problem and there is only one variable in the data samples. The predicted output of class 1 is $f = 1$ and the predicted output of class 2 is $f = 2$. We can also focus on one testing data sample $\overrightarrow{x_i}$ which is classified by Tree1 and Tree2. The region $R_x^{(1)} = \left\{x_{i1} \leq s_1^{(1)}\right\}$is labeled as class 1 for data sample $\overrightarrow{x_i}$ classified by Tree1. The region $R_x^{(2)} = \left\{x_{i1} \leq s_1^{(2)}\right\}$is labeled as class 1 for the data sample $\overrightarrow{x_i}$ classified by Tree2.

When the number of trees used for consensus becomes larger, one data sample may fall in several distinct regions. After averaging, the distribution of $p(f = 1|consensus)$ becomes smoother. In decison trees, we try to construct an algorithm which can minimized the risk. The risk function is defined as follows:

$R(x) = L(f = 1|y = 2)P_{y=2}(x) + L(f = 2|y = 1)P_{y=1}(x)$

From Figure2, if the loss function $L(f = 1|y = 2)$and $L(f = 2|y = 1)$ are

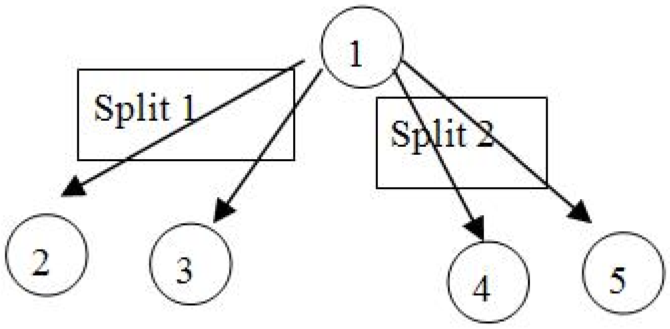


Figure 3: Two different splits at one node.

equal, the best splitting point would be the poinet where $P_{y=1}(x) = P_{y=2}(x)$. If the decision regions of decision trees really reflect the true distribution of data samples, the best splitting point is the middle point of $s_1^{(1)}$ and $s_1^{(2)}$.

Suppose from the decision trees, we can construct K different decision regions. Along $x_1$axis, $s_1^{(1)}, ..., s_1^{(K)}$are splitting points for separately individual decision tree. The best splitting point is s_1^*which minimizes the risk. Of course, $\min s_1^{(k)} \leq s_1^* \leq \max s_1^{(k)}$. Therefore, the risk of each tree $R(Tree1)$is larger than or equal to $R(Tree_{consensus})$.

This is obvious for 2-class problem. The better we construct $p(f = 1)$ and $p(f = 2)$ to minimize the risk, the better splitting point we can find by using Bayes rule. For multi-class problems, we can construct better $p(f = j|x)$ from consensus of distribution and pick the final output $f(x) = \arg \max_j p(f = j|x)$.

3 Two Pair Splits Decision Tree

We can use two different rules for splitting at one node so that different decision regions are constructed and integrated in a tree. It is shown in Figure 3. We also wish two different splitting criteria could generate two dramatically different decision regions. The more diverse decion regions, the better decision boundaries we can construct. We call this Two-Pair Splits Tree (TPST).

3.1 The splitting criteria

We can choose these two splits both univariate splits. However, using univariate split for two-class problem, gini index and twoing criteria give the same splitting situation. Omnivariate tree also shows that there is no big difference from impurity based splitting rules. In order to maximize the difference of distribution between two splits, we adopt univariate split and multivariate split at the internal node.

In our design, the splitting criterion of univariate split is similar to what C4.5 does. Entropy is defined by

$Entropy = -\sum_j \frac{|S_{y=c_j}|}{|S|} \times log_2(\frac{|S_{y=c_j}|}{|S|})$

where $|S|$ is the number data samples in the node and $|S_{y=c_j}|$ is the number of samples belonging to class c_j. The information gain is defined as follows:

$\Delta Info = Entropy(initial) - \sum_i \frac{|T_i|}{|T|} \times Entropy(T_i)$

Gain ratio measures the"normalized" information gain: {Gain Ratio}={Information gain}/ {Entropy}

The best splitting point along one coordinate is the one which gives us the maximal gain ratio.

Linear discriminant is an analytical way to find the best coefficient at one shot. The other methods take longer time to search for best coefficients and try to avoid the local minimum. Fisher's linear discriminant is for finding the best separation hyperplane for two class problem. We can segment the data in to two groups based on their classes, and then find the mean and scatter matrix in that group. Suppose $S_L$is the scatter matrix for the left group and $S_R$is the scatter matrix for the right group. The total within-class scatter matrix is $S_T = S_L + S_R$. $m_L$is the mean of the left group, and m_R is the mean for the right group.

Scatter matix is defined as follows:

$S = \sum_{j=1}^{n}(x_j - \bar{x})(x_j - \bar{x}))^T$

Fisher's linear discriminant is

$w = S_T^{-1}(m_L - m_R)$

If we assume the groups are normally distributed with equal variance, one can solve for the optimal threshold $w_0$as

$w_0 = -\frac{1}{2}(m_L + m_R)^T S_T^{-1}(m_L - m_R) - \ln \frac{n_L}{n_R}$

For multi-class problems, we search through possible groups by assigning only one class to one group. For J classes, we will have at most J different grouping ways. We can also call this method one-against-all-the-others. Among these grouping ways, we pick the one which maximizes the information gain ratio. While computing the inverse of a matrix, it is possible to have a singular matrix. To avoid this scenario, when the number of data samples left in one group is less than the number of features, we skip computing the linear discriminant. If all of the grouping ways generate singular matrix, we will not have a multivariate split at this node.

3.2 Grwoing phase

It is possible that while computing two different splitting criteria, and we decide to have only univariate split. Multivariate split might be dismissed due to the singularity of matrix or similar splitting situation as univariate split. Similar distributions after splits mgiht be obtained between two different spliting criterion. In this situation, there is no need to have two similar splits. The tree keeps growing until the number of remaining data samples is less than a threshold or the remaining data samples is pure enough. We can define whether the node is

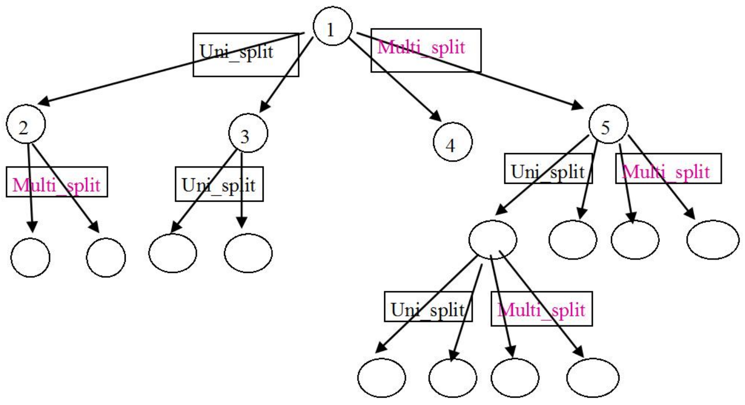


Figure 4: Two pair split tree structure

pure enough by using Gini index[5] or entropy measure. An example of a TPST tree structure is given in Figure 4.

3.3 Labeling scheme

After the tree is constructed, a data sample might fall in several terminal nodes. There are two ways to label this data sample, majority vote and Bayesian average. When there is a terminal node that the data sample lands in, we find out the majority class in the terminal node. The clas assined to is according to this function $\arg\max_j P(C = j|x \in t_o)$, where t_o is a terminal node. After collecting all the votes from the terminal nodes, we find out which class gets the maximal number of votes, and then we assign this class to this data sample. There is a little bias for this majority voting rule. The terminal node might not be pure, but we just simply get rid of the distribution information for other minor classes.

For the Baysian average method, we need to integrate every distribution from the terminal nodes that the data sample lands in. The structure of a generalized TPST is illustrated in Figure 5. $S_{i,k}$ means the k^{th} split at the i^{th} node. In our case, we have two different kinds of split at one node. Hence, $k \in \{1, 2\}$.

We define the child nodes of the root node t_0 by the followings.

$Child(t_0, S_{0,1}) = \{t_1, t_2\}$

$Child(t_0, S_{0,2}) = \{t_3, t_4\}$

By using recursive method, we can find the probability of a data point x.

$P(C = j|x \in t_0(S_{0,1}, S_{0,2})) = P(S_{0,1})P(C = j|S_{0,1}) + P(S_{0,2})P(C = j|S_{0,2})$

We assume $P(S_{0,1}) = P(S_{0,2}) = 0.5$. It means we put equal weight on two different splits.

$P(C = j|x \in t_0(S_{0,1}, S_{0,2})) = \frac{1}{2}P(C = j|S_{0,1}) + \frac{1}{2}P(C = j|S_{0,2})$

Given a split, the probability of class j is

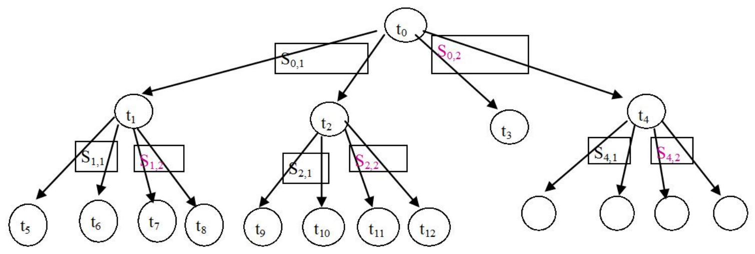


Figure 5: A generalized TPST structure

$$P(C=j|S_{0,1}) = \begin{cases} P(C=j|x \in t_1) & if \quad x \in \{t_1\} \\ P(C=j|x \in t_2) & if \quad x \in \{t_2\} \end{cases}$$

Similarly,

$$P(C=j|S_{0,2}) = \begin{cases} P(C=j|x \in t_3) & if \quad x \in \{t_3\} \\ P(C=j|x \in t_4) & if \quad x \in \{t_4\} \end{cases}$$

If t_i is not a terminal node, we can represent it in another form.
$P(C=j|x \in t_i) = P(C=j|x \in t_i(S_{i,1}, S_{i,2}))$
else t_i is a terminal node, we can compute its probability.
$P(C=j|x \in t_i)$

The final labeling for x is done after computing $P(C=j|x \in t_0)$recursively till the terminal nodes. The class is assined by$\arg\max_j P(C=j|x \in t_0)$. From the equation, we can notice that the weighting for the distribution of a terminal node is dependent on the level of the tree. Suppose H is the level of the terminal node, then the wighting for that node is $(1/2)^H$. This is not contradict to our heuristics that we should put larger weight for the node which stops in the early stage. It stops earlier is because the node is pure. For the nodes which are far away from the root node, it means those data samples are harder to classify so that they pass through a number of splitting rules to reach the terminal nodes.

By doing this, the probability for different classes and different terminal nodes can be integrated into the mathmatical equation. We can overcome the shortcoming of majority vote which ignores the distribution from minor classes. During testing, we can change how much weight should be put on the probability obtained from univariate split or multivariate split. The default is to put equal weight for two different kinds of split.

3.4 Resampling to further strengthen TPST

In our research, we used bagging and random subspace to resample the dataset. A number of classifiers are trained by these new generated datasets. The new resampled datasets are independent with each other so aggregating the results can strengthen the classifier performance.

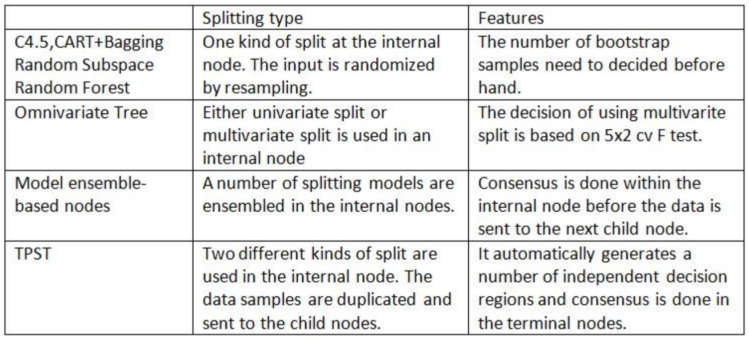
Bagging was first coined by Breiman [3] which means bootstrap aggregating. In this procedure, the original training set TR=$(\overrightarrow{x_i}, y_i), i = 1, 2, \ldots, N$ is used to generate K different new datasets, where the y's are the class labels for the corresponding inputs. Bootstrapping means randomly choosing one data sample each time from the original dataset. After recording this data sample, we just put it back to the original bag. Then we draw the next data sample from the bag again until the size of the new data samples is the same as the size of the original training set. It's like resampling with replacement. These K different new datasets are independent to each other. Some instances may appear more than once, and some instances may not be in a new resampled training set. By using these new resampled datasets, a number of independent classifiers are trained. The final decision is done by doing consensus among these classifiers.

Random subspace means randomly choosing a subset of features to form a new feature space. Suppose the i^{th} sample in the data set is $\overrightarrow{x_i} = [x_{i1}, x_{i2}, \ldots, x_{iM}]^T$. By randomly extracting q features from total M features, a new data sample is constructed. The new dataset is: $[x_{i1}, x_{i2}, \ldots, x_{iq}]^T$ and $q < M$. For each new feature space, the new training samples are formed and used to train a classifier. The final prediction is done by consensus between these different classifiers. While most classifiers try to avoid the curse of dimensionality, this method takes advantage of high dimensionality. The author also mentioned that this method generates 100% training accuracy. The random feature components are chosen to form a new feature space and this gives total 2^M possible combinations. This doesn't mean that the algorithm uses all of the combinations. In Ho's research[10], she suggested using $q = 0.5M$ to obtain the best results.

In decision trees, even removing a data sample from the training set can affect the decision boundaries a lot. Since there is no optmization involved in building a tree, the way to find out the best separation hyperplane is to randomize the data set as much as possible. This is like perturbing the decision boundaries among different trees. Bagging and random space have been shown to further improve classification accuracy. In our research, we applied these methods on top of TPST, and it indeed further increased the performance.

3.5 Comparison with other similar methods

There are other methods which use resampling techniques in decision treess. Omnivariate tree uses either univariate split or multivariate split at the internal node. They choose which splitting rule should be used base on 5x2 cross validation. This is still a binary tree. Model ensemble-based nodes tree generates a number of splitting models at the internal node, the data sample is sent to either the left child node or the right child node based on the aggregation results from a number of splitting models. Bagging, random subspace and random forest shuffle the initial dataset to generate a number of independent new data sets, and then these new data sets are used for training a number of independent classifiers. The final result is aggregated by majority vote or Baysian average. Our proposed TPST mthod adopts both univariate split and multivariate split to maximize the randomization of decision boundaries. The summarized comparison table is shown in Fig. 6.



	Splitting type	Features
C4.5,CART+Bagging Random Subspace Random Forest	One kind of split at the internal node. The input is randomized by resampling.	The number of bootstrap samples need to decided before hand.
Omnivariate Tree	Either univariate split or multivariate split is used in an internal node	The decision of using multivarite split is based on 5x2 cv F test.
Model ensemble-based nodes	A number of splitting models are ensembled in the internal nodes.	Consensus is done within the internal node before the data is sent to the next child node.
TPST	Two different kinds of split are used in the internal node. The data samples are duplicated and sent to the child nodes.	It automatically generates a number of independent decision regions and consensus is done in the terminal nodes.

Figure 6: Comparisons with other existing methods.

4 Computation Complexity

The computation time of TPST grows with the number of data samples, features, and classes. The data samples are duplicated at each internal node because there are two splitting rules integrated at that node. Each splitting rule requires one copy of data samples. For univariate split, the number of computation depends on the feature size and samples size at that node. Brute-force search is executed for finding all possibilities of features and splitting points. The best set of feature and splitting point is used when it provides the largest information gain ratio. For multivariate split, the number of computation depends on the computation of scatter matrix, inverse of matrix, and matrix multiplication.

During testing, we aggregate all the distribution of terminal nodes that contain the data sample x. Suppose L is the level of the constructed TPST tree. From Fig. 5, A tree containing node $\{t_0, t_1, t_2, t_3, t_4\}$ is a 1-level TPST, we call this $L = 1$. Let's consider the number of terminal nodes that contain the sample x in a full-grown tree.

$L = 1$, there will be 2 terminal nodes that contain x.

$L = 2$, there will be 2^2 terminal nodes that contain x.

$L = 3$, there will be 2^3 terminal nodes that contain x.

In this two-pair splits tree, the estimated probability $P(C = j|x)$ is aggregated through different weightings on the terminal nodes that contain x. For the full-grown tree, it is like consensus on 2^L nodes. The better $P(C = j|x)$ we can obtain, the more accurately we can predict.

Since there are two types of splitting rule integrated in TPST and each splitting rule seperates the data into two groups. We can compute the number of sub-binary trees in TPST. By looking at Fig. 5, we can see that the approximate number of sub-binary trees is $2 \times (4 \times 2)^{L-1}$.

$L = 1$. there are two sub-binary trees. They are $\{t_0, t_1, t_2\}$ and $\{t_0, t_3, t_4\}$.

$L = 2$ and including $S_{0,1}$. There are 8 sub-binary trees. They are $\{t_0, S_{0,1}, t_1, t_2, t_5, t_6\}$,$\{t_0, S_{0,1}, t_1, t_2, t_7, t_8\}$,$\{t_0, S_{0,1}, t_1, t_2, t_9, t_{10}\}$,$\{t_0, S_{0,1}, t_1, t_2, t_{11}, t_{12}\}$,$\{t_0, S_{0,1}, t_1, t_2, t_5, t_6, t_9, t_{10}\}$

Dataset Name	# of samples	# of features	# of classes
bc_wisconsin	683	9	2
breast_cancer	286	9	2
sonar	208	60	2
iris	150	4	3
vehicle	846	18	4
vowel*	990	10	11

Table 1: The summary of the datasets that we used in our experiment.

,$\{t_0, S_{0,1}, t_1, t_2, t_5, t_6, t_{11}, t_{12}\}$, $\{t_0, S_{0,1}, t_1, t_2, t_7, t_8, t_9, t_{10}\}$, and $\{t_0, S_{0,1}, t_1, t_2, t_7, t_8, t_{11}, t_{12}\}$
.

This computation complexity grows so dramatically for large dataset. In order to reduce this problem, at some nodes we might just use one kind of split. The other way to solve this problem is to include pruning algorithm to avoid overfitting and overly large trees. The best separation boundaries are obtained from shuffling and resampling the data samples, but the estimated distribution and class probability are obtained from aggregation of distributions from a number of terminal nodes. There should be an optimal binary tree embedded in this big TPST. If we could find some ways to find out this optimal binary tree, we would reduce the computation complexity a lot in advance.

5 Experimental Results

We use six datasets to test our algorithm. The datasets are downloaded from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/). The summary of the datasets are described in Table 1. Three of the six datasets are multi-class problem. Because we use one-against-all-the-others for multi-variate split, it takes longer time to compute. In the vowel dataset, speaker number and sex are removed from the features because they are nominal attributes. For multivarite split, we need to compute scatter matrix and mean. Nominal attributes are not suitable in TPST.

Alpaydin suggested using 5x2 cross validation[1] to compare the performance among different classifiers. He claimed this gives smaller type I error and larger power. 5x2 cross validation means repeating the 2-fold cross validation for 5 times. Each time, the dataset is divided into two different groups, one for training and the other one for validation. We get one classification accuracy from this set-up. Then we swap the training and validation set to get another classification accuracy. Totally, we have 10 classification accuracies. By averaging those 10 numbers, we get an averaging classication accuracy. We record this number in the table.

We implemented our algorithm in Weka[9]. In TPST, we can put different weighting on univariate split and multivariate split. In order to satisfy with probability theory, we let the sum of weighting of univarite and multivariate

	W_uni=0.2	W_uni=0.4	W_uni=0.5	W_uni=0.6	W_uni=0.8
bc_wisconsin	95.81%	96.28%	96.31%	96.40%	95.73%
breast_cancer	68.39%	68.67%	68.81%	69.65%	69.86%
sonar	69.52%	69.52%	69.52%	69.52%	69.52%
iris	96.93%	96.93%	95.60%	93.87%	93.73%
vehicle	78.37%	80.14%	80.31%	80.17%	77.52%
vowel*	80.08%	84.20%	84.28%	83.31%	77.66%

Table 2: Classification peroformance of TPST from different weighting of univariate split and multivariate split.

Dataset	J48	Bagging+J48	CART	Bagging+CART	RF	TPST
bc_wisconsin	94.64%	96.19%	94.64%	95.75%	96.28%	96.31%
breast_cancer	70.28%	71.47%	71.05%	71.61%	68.53%	68.81%
sonar	69.33%	74.23%	69.33%	71.15%	74.04%	69.52%
iris	92.93%	93.47%	93.47%	93.87%	94.80%	95.60%
vehicle	69.72%	72.34%	68.61%	71.11%	72.88%	80.31%
vowel*	70.44%	81.15%	68.38%	79.43%	85.64%	84.28%

Table 3: Comparison of accuracies of different decision trees.

equal to one. Classification performance of TPST with different weighings are obtained from 5x2 cross validation. From Table 2, we can see that the performance of 2-class dataset (bc_wisconsin, breast_cancer, and sonar) varies slightly with different weighting. Classification accuracy of sonar dataset is not changed when the weighting is varied. Iris is a dataset with 4 features and 3 classes. More weighting should be put on multivariate split to better classify iris dataset. For the remaining vehicle and vowel dataset, equal weighting should be put between univariate and multivariate in order to obtain higher accuracy. For the convenience of later research, we put weighting 0.5 for univariate split and 0.5 for multivariate split.

There are several well-known univariate decision trees, such as C4.5 and CART. In Weka, the implemetation version of C4.5 is called J48. Resampling such as bagging (bootstrap aggregating) can be used to further improve the classification accuracy. We also applied baggin on J48 and CART. Random forest (RF) combines bootstrapping and random subspace. From the new generated independent training data, a number of classifiers are generated. During testing, the final decision is done by consensus among these classifiers. The classification accuracies from J48, Bagging+J48, CART, Bagging+CART, Random Forest, and TPST are obtained by 5x2 cross validation. In the setting of the experiments, the number of iterations for Bagging and Random Forest is set to 10. The results are shown in Table 3.

From Table 3, we can see that mostly TPST outperforms than basic classifiers, such as J48 and CART. For vehicle dataset, TPST even largely outperforms than Random Forest. Since resampling from bootstrapping and RF can

Dataset	TPST	Boot.+TPST	Rand.Sub.+TPST	Boot.+Rand.Sub.+TPST
bc_wisconsin	96.31%	96.98%	97.31%	97.22%
breast_cancer	68.81%	70.98%	72.10%	73.22%
sonar	69.52%	73.46%	76.35%	76.06%
iris	95.60%	95.73%	94.40%	94.80%
vehicle	80.31%	80.90%	78.46%	77.90%
vowel*	84.28%	86.22%	86.71%	83.78%

Table 4: Resampling to further increase the classification accuracy of TPST. Boot. is the abbreviation of bootstrapping. Rand.Sub. is the abbreavation of Random Subspace.

further increase the classification, we apply these two methods onTPST. We can apply only bootstrapping, only random subspace, or both bootstrapping and random subspace. In Random Subspace method, we follow Ho's suggestion to randomly choose half of the features as new feature space. In order to make the comparison fair, we fix the number of iterations for resampling, 10, for all the datasets, except for vowel dataset. For vowel dataset, the computation easily runs out of the memory, 1.5GB. When running classification with this daset, we reduce the number of iterations to 5. The results are shown in Table 4.

After applying resampling techniques, we can see some of the results are better. In iris and vehicle dataset, the resampling with Random Subspace makes the performance worse a little bit. However, resampling with Bootstrapping can improve the performance more constantly.We also notice that combining bootstrapping and random subspace doesn't increase the pefromance in vehicle and vowel dataset.

6 Discussion

We use analytical way to find the coefficients for multivariate split. This makes the computation faster. Avoiding the singularity of matrix let the type of the nodes which are closer to leaf nodes be only univariate splits. It makes sense that the remaining data samples in the nodes which are closer to leaf doesn't require complicated multivariate split. However, the computation of matrix makes it not able to handle nominal attributes as CART or C4.5. We might walk around this problem by converting the nominal attributes to numeric attributes.

From Table 3, we can see that our proposed method performs better in some data sets. Especially for vehicle and vowel datasets, they are statistically better than J48 and CART. The performance of TPST is similar to that of Random Forest tree. TPST is even much better than Random Forest in vehicle dataset. This illustrates that two pair splits integrated in a tree can aggreate the distribution as Random Forest does.

We have discovered that randomization in decision trees can further improve classsification accuracy. There are three ways used in this paper, two-pair splits, bootstrapping and random subspace. Usually, the more randomness we integrate into a decision tree, the better performance we can get.

The computation complexity grows with exponent, $O(x^L)$. L is the levels of the tree. It's better to find some ways to reduce using two different splits at one node while it still increases the diversity of the distributions. When we run large datasets (the number of data samples is large), it takes much much longer time. If we want to apply resampling techniques, it takes even more time to run several iterations. We consider to use pruning algorithm to reduce the tree dimension. We can select a split type at a node using deterministic or randomized rules. The randomized rules are particularly important since they construct a global approach to the problem.

References

[1] Ethem Alpaydin. Combined 5 x 2 cv f test for comparing supervised classification learning algorithms. *Neural Computation*, 11(8):1885, 1999.

[2] Hakan Altincay. Decision trees using model ensemble-based nodes. *Pattern recognition*, 40(12):3540–3551, 2007.

[3] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman Hall, 1984.

[6] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine learning*, 19:45–77, 1995.

[7] Eibe Frank and Ian H. Witten. Selecting multiway splits in decision trees, 1996.

[8] Heng Guo and Saul B. Gelfand. Classification trees with neural network feature extraction. *IEEE transactions on neural networks*, 3(6):923–933, 1992.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[10] Tin Kam Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, 1998.

[11] Hyun-Chul Kim, Shaoning Pang, Hong-Mo Je, Daijin Kim, and Sung Yang Bang. Constructing support vector machine ensemble. *Pattern recognition*, 36(12):2757–2767, 2003.

[12] Hyunjoong Kim and Wei-Yin Loh. Classification trees with unbiased multiway splits. *Journal of the American Statistical Association*, 96(454):589–604, 2001.

[13] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *The journal of artificial intelligence research*, 2:1–32, 1994.

[14] J. Ross Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning).* Morgan Kaufmann, 1993.

[15] Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers - a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(4):476–487, 2005. 1094-6977.

[16] Robert E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999.*, 1999.

[17] Kenneth Joseph Wilder and Donald Geman. *Decision tree algorithms for handwritten digit recognition.* PhD thesis, University of Massachusetts Amherst, 1998. Director - Donald Geman.

[18] Min Xu, Pakorn Watanachaturaporn, Pramod K. Varshney, and Manoj K. Arora. Decision tree regression for soft classification of remote sensing data. *Remote Sensing of Environment*, 97(3):322–336, 2005. 10.1016/j.rse.2005.05.008.

[19] Olcay Taner Yildiz and Ethem Alpaydin. Omnivariate decision trees. *Neural Networks, IEEE Transactions on*, 12(6):1539–1546, 2001.

[20] Olcay Taner Yildiz and Ethem Alpaydin. Linear discriminant trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(3):323–353, 2005.