Exposition on Random Forest: A stock example

Technical Report · March 2017

DOI: 10.13140/RG.2.2.16412.08329

2 authors:

Suryoday Basak
University of Texas at Arlington
35 PUBLICATIONS 110 CITATIONS
SEE PROFILE

Snehanshu Saha
People's Education Society
170 PUBLICATIONS 366 CITATIONS
SEE PROFILE

Random Forests: An Example of Classification Using Stocks' Data

Suryoday Basak and Snehanshu Saha

March 23, 2017

Some of the main advantages of using Random Forests of Classification and Regression Trees (CART) are:

1. Tree based learning methods are *non-metric.* This means that there are no inherent assumptions of distributions in data.
2. Random Forests are recommended to be used with *bagging*, i.e., test data is sampled with replacement. Bagging ensures that with the increase in the number of tree estimators in a random forest, the chance of error decreases. A simple assertion of this is made (using the binomial distribution stuff) and is proved using Chebyshev's Inequality in (enter section number).
3. CART can handle categorical and continuous values naturally.
4. It is easy to understand and quick to fit, even if the data is cluttered and/or if the problem is inherently large.
5. The accuracy of random forests compares well with traditional methods in ML, such as Naïve Bayes', etc.
6. Random forests are *stable.* A slight change in the input data may affect individual trees, but the characteristics of the forest remains largely unchanged.

In random forests, decision tree learners are constructed by randomly selecting m out of M features and n out of N attributes. Here, we illustrate the working of random forests by randomly considering 20 samples from the data set as the training set and 5 samples as the test set; the training and test sets are mutually exclusive. They are gshown in Tables 1 and 2 respectively.

Let us take $m = \sqrt{M}$, so that in each tree constructed, $\sqrt{M}$ features are randomly considered. In the data set, since there are 6 features, $\sqrt{M} = \sqrt{6} \approx 2.45$. Hence, in each tree, 2 features will be considered. We shall take the subsample size to be equal to the sample size, i.e., 20 in each tree constructed here.

The best *split* for a node in a decision tree is determined by sorting the samples in a node with respect to a feature (considered by the respective tree) and then partitioning it into two parts: the left and right children of that node. The partitioning is done by selecting the best threshold (hence, the sorting) for the corresponding feature and calculating the Gini gain (insert equation number). For example, let us consider a tree constructed by using the first two features in the training set, *fersi* and *dpsch*, the training set as seen by the decision tree, after being sorted based on fersi, looks as shown in Table 3.

The root node of the decision tree will perform the first partition of the data in Table 3. Hence, the constituents of the root node of the tree (or any tree) will have all the samples in the training set with 2 features selected at random. Let us consider arbitrarily the first 10 samples in Table 3 to constitute the *left child* and the last 10 samples to constitute the *right child* of the root node. The tree, after just the first split, will look as shown in Figure 1.

Here, the impurity of the parent node (in this case, at level 0; the root node) is 0.5. This is easily understandable as 10 samples belonging to Class 1. and Class -1. So the Gini Impurity is calculated as shown in Equation 1.

S.Id	Class Label	fersi	dpsch	dpwill	MACD	pricerate60	OBV
1	-1	91.02	74.0	-54.07	1.12	0.03	259336600.0
2	1	11.09	17.12	-35.42	-0.8	-0.05	23149700.0
3	1	75.95	77.76	-54.6	0.64	-0.02	263929400.0
4	-1	88.25	85.8	-50.08	1.02	0.01	369776000.0
5	-1	16.71	14.19	-35.08	-0.67	0.04	1434446700.0
6	1	9.55	11.48	-35.28	-1.5	-0.02	1264042100.0
7	1	66.21	77.38	-48.78	-0.23	-0.02	198021100.0
8	-1	67.82	77.98	-50.25	0.36	0.02	276163600.0
9	-1	71.88	49.71	-50.82	0.58	0.08	560679400.0
10	1	4.66	11.6	-44.17	-1.56	-0.06	135643900.0
11	1	1.52	16.01	45.27	-0.33	-0.13	-2263609600.0
12	-1	17.41	24.08	-44.97	-0.12	0.02	-3417917300.0
13	1	22.27	31.57	-40.62	-0.22	-0.03	-3265559400.0
14	-1	70.79	77.25	-45.93	0.09	0.02	-2846701500.0
15	1	77.74	84.71	-72.5	0.36	-0.08	-3516514200.0
16	-1	57.5	73.5	-44.46	-0.12	0.09	-2926120100.0
17	1	42.81	54.22	-52.63	-0.26	-0.02	-3213555400.0
18	1	61.83	72.97	-66.66	-0.09	-0.1	-3904360800.0
19	-1	18.86	16.0	-36.27	-0.13	0.03	-3645587700.0
20	-1	63.08	52.02	-50.11	-0.03	0.02	-2182828500.0

Table 1: Sample Training Set

S.Id	Class Label	fersi	dpsch	dpwill	MACD	pricerate60	OBV
1	-1	24.97	15.54	-43.74	-0.57	0.15	-1373395100.0
2	-1	85.06	75.77	-59.75	0.37	0.08	-1711049700.0
3	-1	86.88	70.44	-66.76	0.78	0.03	-1910292300.0
4	1	0.69	13.22	-36.48	-1.35	-0.19	-2573216900.0
5	1	78.9	80.95	-47.66	1.71	-0.01	-2371874000.0

Table 2: Sample Test Set

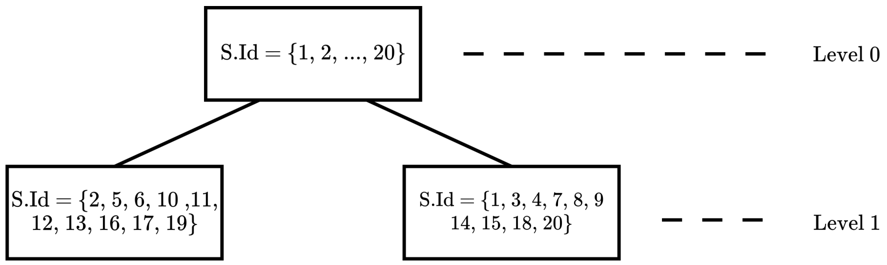

S.Id = {1, 2, ..., 20}
Level 0
S.Id = {2, 5, 6, 10 ,11, 12, 13, 16, 17, 19}
S.Id = {1, 3, 4, 7, 8, 9 14, 15, 18, 20}
Level 1

Figure 1: Tree creation by arbitrarily using the first 10 samples from Table 3 as the left child and the last 10 samples as the right child

S.Id	Class Label	fersi	dpsch
11.0	1	1.52	16.01
10.0	1	4.66	11.6
6.0	1	9.55	11.48
2.0	1	11.09	17.12
5.0	-1	16.71	14.19
12.0	-1	17.41	24.08
19.0	-1	18.86	16.0
13.0	1	22.27	31.57
17.0	1	42.81	54.22
16.0	-1	57.5	73.5
18.0	1	61.83	72.97
20.0	-1	63.08	52.02
7.0	1	66.21	77.38
8.0	-1	67.82	77.98
14.0	-1	70.79	77.25
9.0	-1	71.88	49.71
3.0	1	75.95	77.76
15.0	1	77.74	84.71
4.0	-1	88.25	85.8
1.0	-1	91.02	74.0

Table 3: Training set sorted in increasing order of *fersi*

$$\begin{aligned} G(N) &= 1 - (P_1)^2 - (P_{-1})^2 \\ &= 1 - \left(\frac{10}{20}^2\right) - \left(\frac{10}{20}^2\right) \\ &= 1 - 0.25 - 0.25 \\ &= 0.5 \end{aligned} \tag{1}$$

However, as the quality of the split depends on the *GiniGain*, the Gini impurities of the left and right children also need to be calculated. They are calculated in a manner similar to Equation 1.

$$\begin{aligned} G(N_l) &= 0.48 \\ G(N_r) &= 0.48 \end{aligned} \tag{2}$$

Note that the left child has 6 samples belonging to Class 1 and 4 samples belonging to Class -1, and the right child has 6 samples belonging to Class -1 and 4 samples belonging to Class 1. Hence, in this case, $G(N_l)$ and $G(N_r)$ are equal. It is not always necessary for them to be equal!

The *GiniGain* hence is calculated as in Equation 3.

$$GiniGain = 0.5 - 0.48 - 0.48 = 0.02 \tag{3}$$

This is how a node in a tree is split. At the next level of the tree, in each child node, the best split is determined again in a similar way. But now the partitions of the training sample space belonging to the children nodes are considered for partitioning each respective child node. This kind of partitioning is done recursively till *pure* nodes are reached. In decision trees, all leaf nodes need to be pure nodes (a leaf node is a node that is not split further). A node is said to be *pure* if all the samples in that node belong to the same

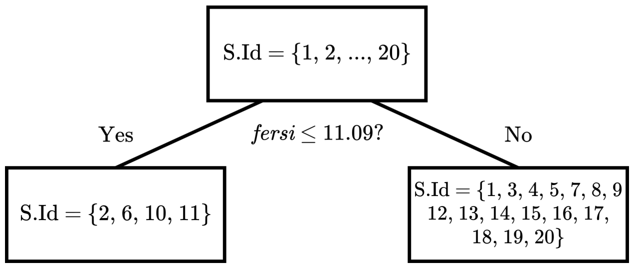


Figure 2: The best split for the root node when considering *fersi* and *dpsch*

class such that it does not require any further splitting. When an element needs to be classified, through a series of partitioning criteria, it reaches a leaf node and is assigned the class label of all the elements in the respective leaf node.

Returning to the illustration, it should be noted that a 50:50 split on the root node, with the first 10 samples from Table 3 constituting the left child node and the last 10 samples constituting the right child node is not the best split for that node. In fact, the best split is when the left child node has the first 4 samples and the right child node has the last 16 samples, such that the split as shown in Figure 2.

For this split, the *GiniGain* is 0.125, which is greater than that for the split shown in Figure 1.

Now, how is this number 11.09, as the splitting threshold, determined? Looking at Table 3, we see that it is the value of the *fersi* attribute of the fourth sample. Reiterating, the samples in this table are sorted in the increasing order of *fersi*. As the partition till the 4^{th} sample for the left child node and the last 16 samples for the right child node results in the best GiniGain, the *fersi* value of the last sample comprising the left partition is used as a boundary value, or the split value (notice how the first four samples all belong to Class 1: this should help develop the readers' intuition on the working of *GiniGain* for determining the best split). Thus, the split point for the root node becomes (*fersi*, 11.09). When an element is being classified using this tree, it essentially needs to reach a leaf node by going through a series of splits. Hence, the first split will be done using (*fersi*, 11.09): if the value if *fersi* ≤ 11.09) for an element to be classified, it will traverse the left branch of the root node (which, in this case, is a pure node), and if the value of *fersi* > 11.09, it will traverse the right branch, and consequently, a series of more splits, till it reaches a leaf node.

The complete tree, built recursively such that all leaf nodes are pure nodes, is shown in Figure 3.

The geometric interpretation of a decision tree is that of partitions in the feature space corresponding to each class. Using the thresholds determined using the *GiniGain* for each split, the feature space can be considered to be divided into partitions or *pockets*, with each partition corresponding to one class. The partitions of the feature space for Tree 1 are shown in Figure 4. In this graph, all the partitions in blue colour correspond to Class 1, and all the partitions in red colour correspond to Class 2. On this graph, if we were to plot each sample in the test set (Table 2) by considering only (*fersi*, *dpsch*), then the *colour* of the partition a sample belonged to would indicate which class the respective sample is a part of (as the colour of the partition represents the Class label)! Thus, random forests can be used to effectively handle non-linear trends in data or separability of data.

Now that a tree is constructed using the training set, the obvious task at hand is to classify the samples in the test set using this tree. Reiterating, in order to classify a test sample, we need to trace a test sample down the tree until a leaf node is reached. As leaf nodes are pure, the class of the leaf node will be assigned to the test sample being classified.

From Table 4, we can see that **Tree T1** misclassifies 2 out of 5 test samples, samples 1 and 5. Hence, it can be said that **T1** has a classification accuracy of 60%. Generally, a 60% accuracy for a classifier is not considered to be good enough. So in order to have a complete understanding of how a random forest might work, we shall construct two more trees and consider the majority vote for each test sample.

Let us construct the next tree in the forest by considering the features *dpwill* and *dpsch*. Notice that we're resampling the values of *dpsch*. This is the principle of bootstrap aggregation, more commonly known as *bagging*: to sample data with replacement. Bagging has certain advantages. While growing a tree, a feature may or may not be chosen to be a feature using which the tree grows. This is especially relevant for

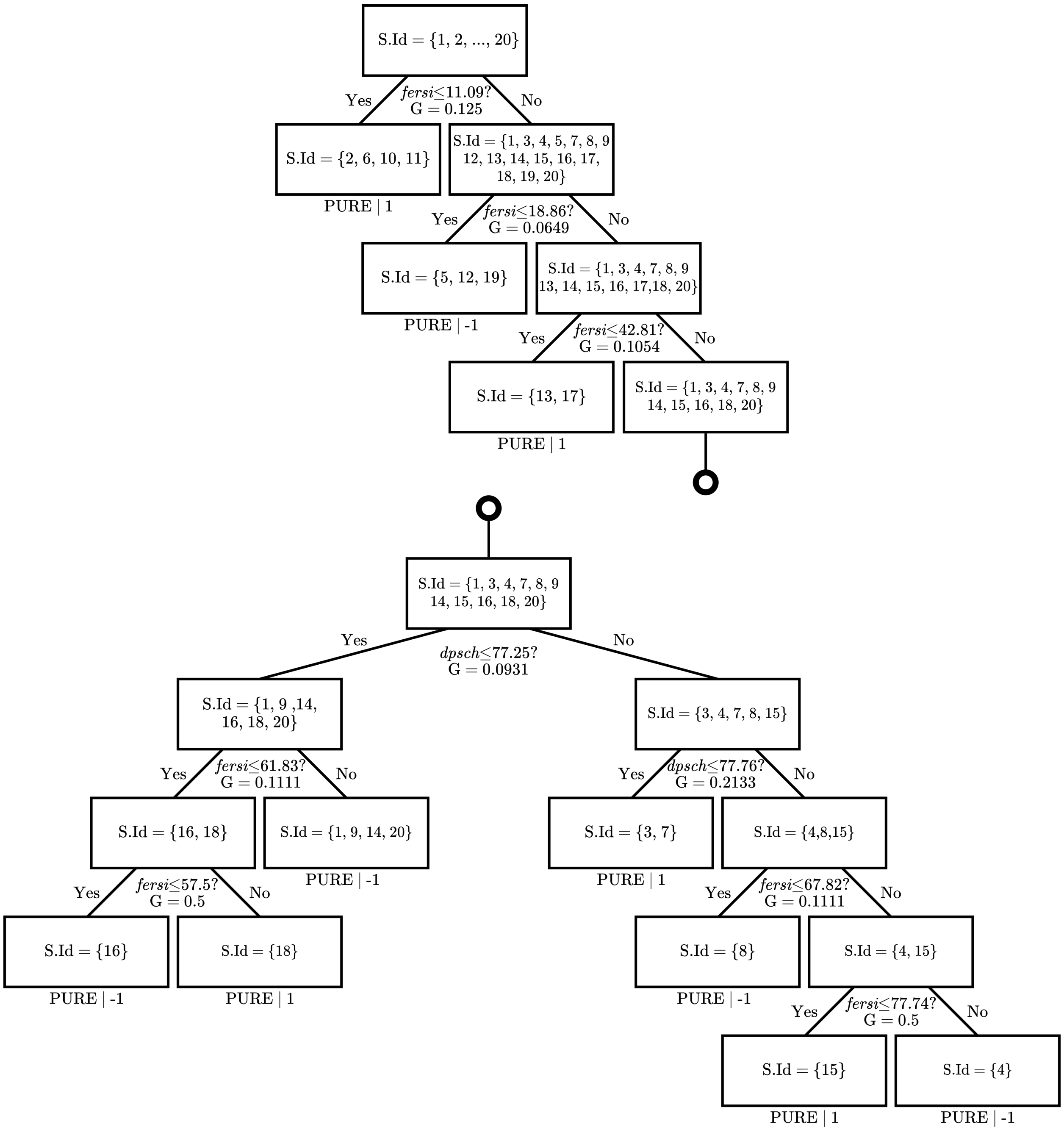


Figure 3: **Tree 1**: Random tree constructed considering *fersi* and *dpsch*

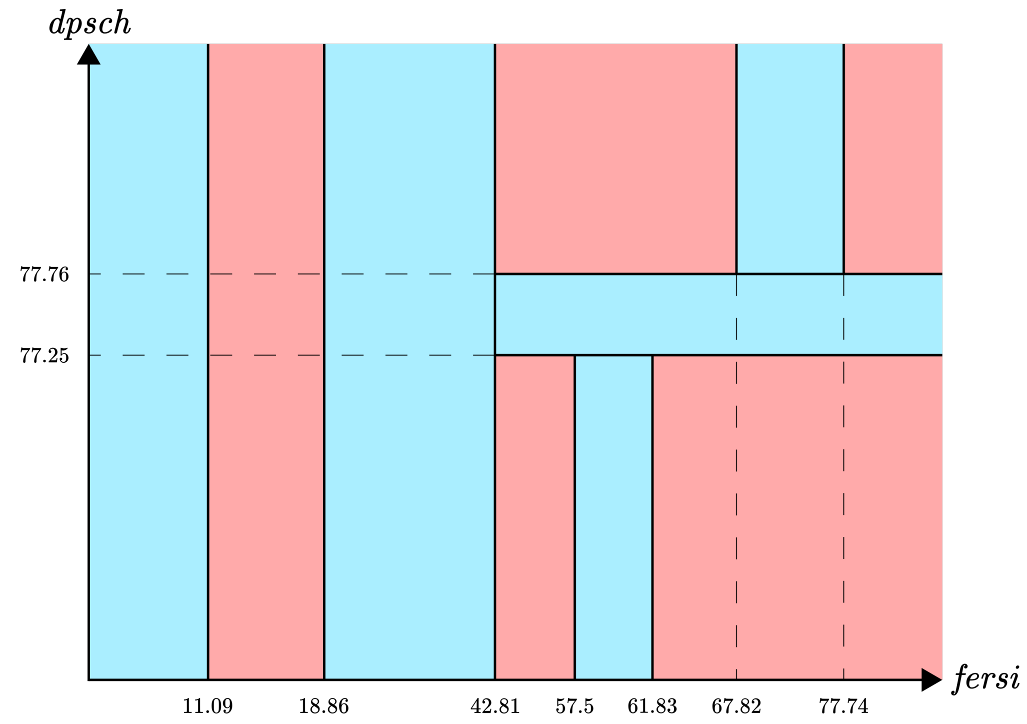


Figure 4: Partitions of sample space of Tree 1 (graph not drawn to scale)

S.Id	fersi	dpsch	Class Label	Predicted Label
1	24.97	15.54	-1	1
2	85.06	75.77	-1	-1
3	86.88	70.44	-1	-1
4	0.69	13.22	1	1
5	78.9	80.95	1	-1

Table 4: Results of classification the sample test set using **Tree 1**

S.Id	dpwill	dpsch	Class Label	Predicted Label
1	-43.74	15.54	-1	-1
2	-59.75	75.77	-1	1
3	-66.76	70.44	-1	1
4	-36.48	13.22	1	-1
5	-47.66	80.95	1	1

Table 5: Results of classification the sample test set using **Tree 2**

high-dimensional datasets. Since each feature has a certain probability of success or failure of being chosen when a tree is being grown, it can be said that the probability of a feature being used in a random forest depends on the number of trees being constructed, with the probability being *binomially distributed*.

In the dataset used for the current work, as we have already mentioned, there are 6 features and for each tree being constructed, 2 features are considered. This implies that the probability of *success* (success being the condition that a feature is chosen to grow a decision tree) is $P(success) = 2/6 = 1/3 = 0.33$. The probability of *failure* (failure being the condition that a feature is not chosen to grow a decision tree) is $P(failure) = 1 - P(success) = 0.67$. Let us represent the probability of success as p and the probability of failure as q. Then the probability that a feature x_a is used in growing k out of n trees is given by Equation 4.

$$P(k\ successes) = \binom{n}{k} p^k q^{n-k} \tag{4}$$

It is easy to see that the LHS of Equation 4 increases as the value of n increases. This can be used to understand the idea of *stability* of a random forest: that upon increasing the number of trees in a random forest, the expected results are stabilized by continuous re-sampling of data so that the forest as a whole is representative of the class to which a sample to be classified belongs.

From Table 5, we can see that 2 out of 5 samples are classified correctly. This is performance, as an individual tree, is not good. Upon comparing the predicted results from **Tree 1** and **Tree 2**, we see that the results from both trees combined are accurate only 50% of the time. The system as a whole is not accurate enough to predict the class of a previously unobserved sample accurately. To move a step closer towards an accurate system, let us construct another tree. We shall construct **Tree 3** using *dpsch* and *pricerate60*.

The decision tree constructed using *dpsch* and *pricerate60* is a rather interesting one: there's only one split in the tree and the children of the root nodes are pure! The results of classification using just this tree is given in Table 6. This single tree gives us a 100% classification accuracy! The question arises: *why not use just this tree to classify the sample?*. The reasons go back to the notions of stability: that if there's some change in the data, some of the trees might not perform well, but the results of the forest should largely remain unchanged. In general, a forest classifier can be used to reduce the effect of muddy data. As this is an illustrative example, we are dealing with only 20 samples. However, the trend in millions of samples may not be as easy to deal with a single threshold. Besides, more recent innovations in tree based classifiers (such as AdaBoost, XGBoost) can discern or decide how to grow trees based on the performance of previously grown trees in the forest.

Now that there are three trees in our forest, we can perform a majority voting. The result of the entire tree using **Tree 1**, **Tree 2** and **Tree 3** is given in Table 7. By combining the performance of all the trees in the forest, the results are perfect, and also stable.

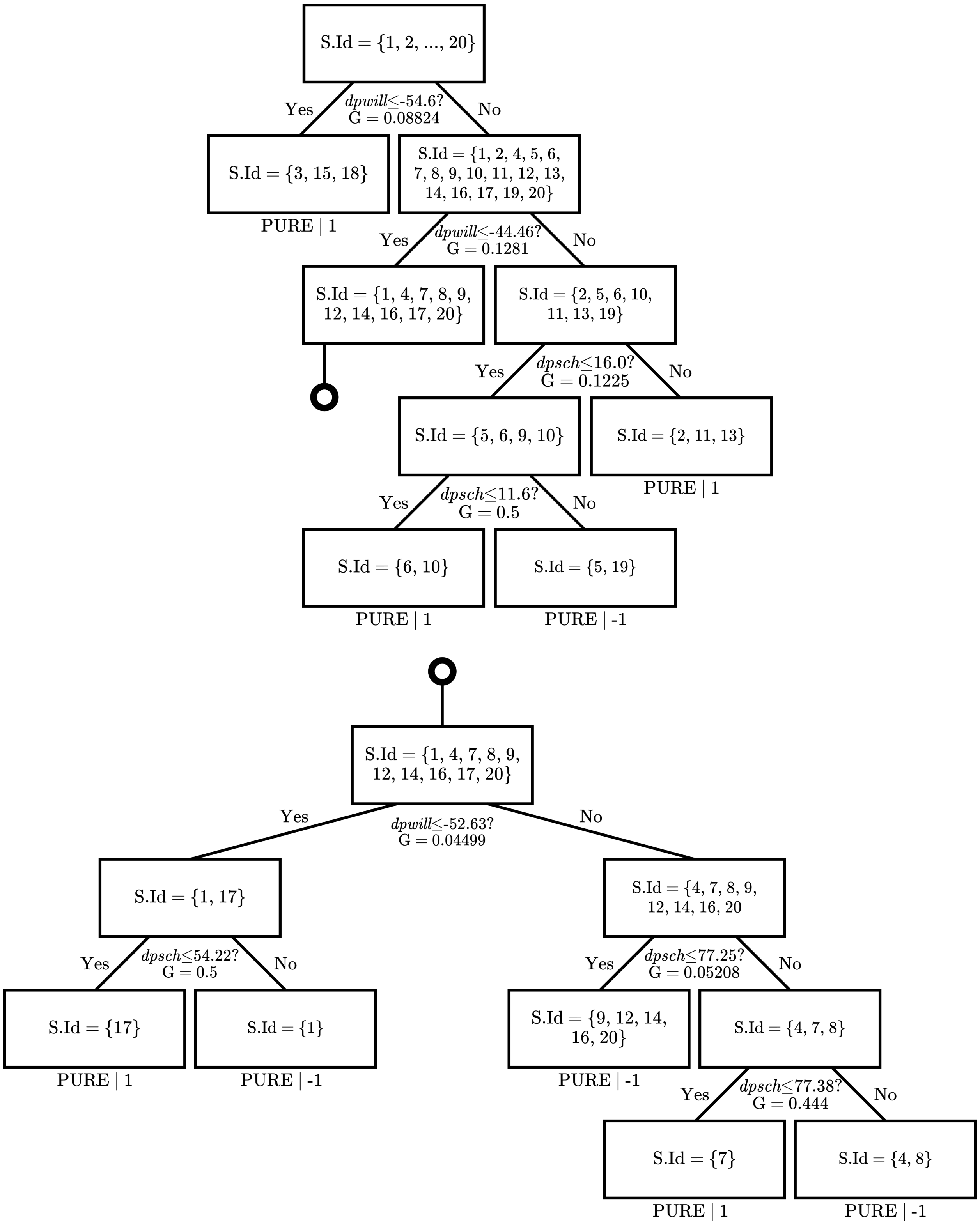


Figure 5: **Tree 1**: Random tree constructed considering *dpwill* and *dpsch*

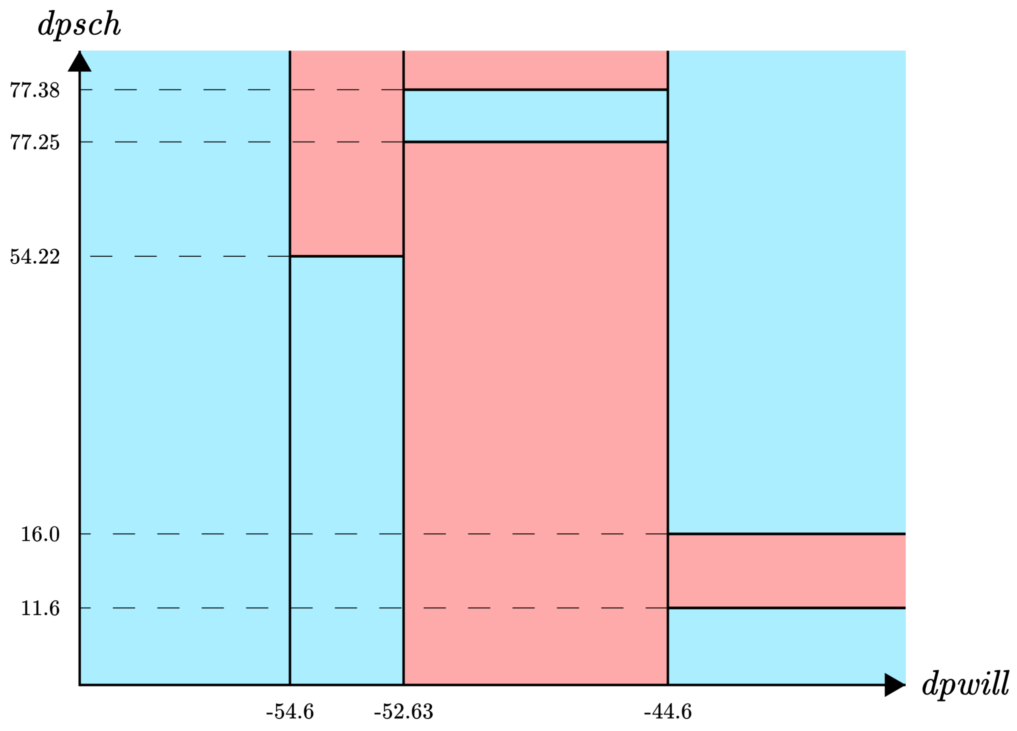
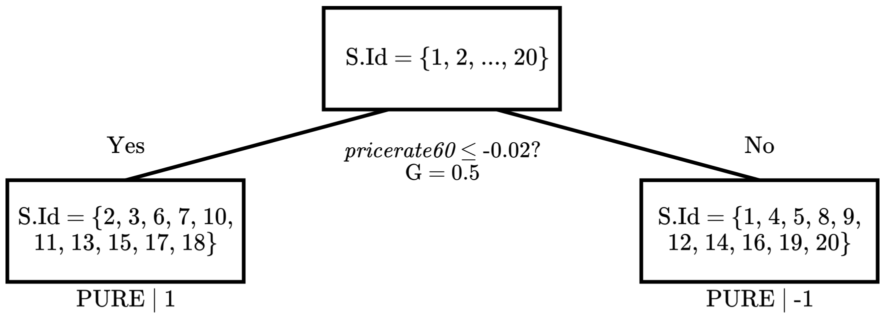


Figure 6: Partitions of sample space of Tree 2 (graph not drawn to scale)



S.Id = {1, 2, ..., 20}
Yes
No
pricerate60 ≤ -0.02?
G = 0.5
S.Id = {2, 3, 6, 7, 10, 11, 13, 15, 17, 18}
PURE | 1
S.Id = {1, 4, 5, 8, 9, 12, 14, 16, 19, 20}
PURE | -1

Figure 7: **Tree 3**: Random tree constructed considering *dpsch* and *pricerate60*

S.Id	dpsch	pricerate60	Class Label	Predicted Label
1	15.54	0.15	-1	-1
2	75.77	0.18	-1	-1
3	70.44	0.03	-1	-1
4	13.22	-0.19	1	1
5	80.95	-0.10	1	1

Table 6: Results of classification the sample test set using **Tree 3**

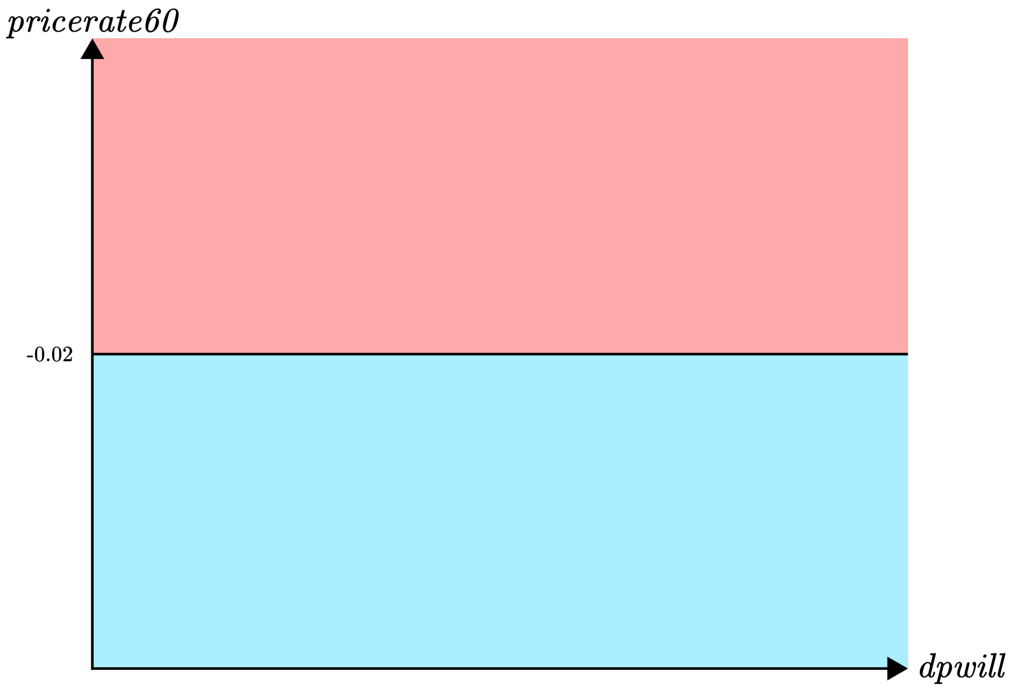


Figure 8: Partitions of sample space of Tree 2 (graph not drawn to scale)

S.Id	Class Label	Tree 1	Tree 2	Tree 3	Predicted Label
1	-1	1	-1	-1	-1
2	-1	-1	1	-1	-1
3	-1	-1	1	-1	-1
4	1	1	-1	1	1
5	1	-1	1	1	1

Table 7: Results of classification of the entire forest using majority voting