KLE TECH. KLE Technological University

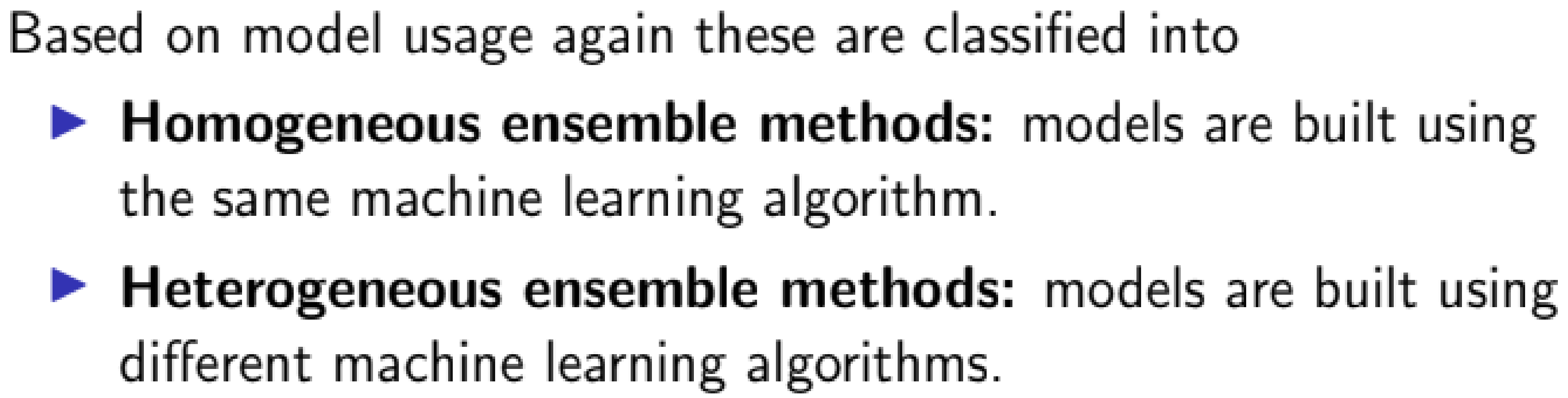
Creating Value
Leveraging Knowledge

Ensemble Modelling

Bagging
Boosting
Random Forest

What is Ensemble Learning?

- Using multiple learning algorithms together for the same task.
- Better predictions than individual learning models.



Based on model usage again these are classified into

- ▶ **Homogeneous ensemble methods:** models are built using the same machine learning algorithm.
- ▶ **Heterogeneous ensemble methods:** models are built using different machine learning algorithms.

What is Ensemble Learning?

Data Set

- Labelled Images of Samosa and Other objects
- Images of Samosa Labelled as 1
- Images of other Labeled as 0

	Test Images	Learning Algorithms	Predictions
Features and Labels		Decision Tree (75%)	Samosa
		SVM (80%)	Samosa
		Logistic Regression(45%)	Not Samosa

What is Ensemble Learning?

Data Set

- Labelled Images of Samosa and Other objects
- Images of Samosa Labelled as 1
- Images of other Labeled as 0

Test Images	**Learning Algorithms**	**Predictions**		
	Decision Tree	Samosa		
	SVM	Samosa	**Voting** →	Samosa
	Logistic Regression	Not Samosa		

What is Ensemble Learning?

Test Images	Learning Algorithms	Predictions		
	Decision Tree	Samosa		
	SVM	Samosa	Voting	Samosa
	Logistic Regression	Not Samosa		

Why to use Ensemble Models?

- Better Accuracy (Low Error)
- Higher Consistency (Avoid Over Fitting)
- Reduces Bias and Variance Errors

When and Where to use Ensemble Models?

- Single Model Over fits
- Results worth the Extra Training
- Can be used for Classification as well as Predictions

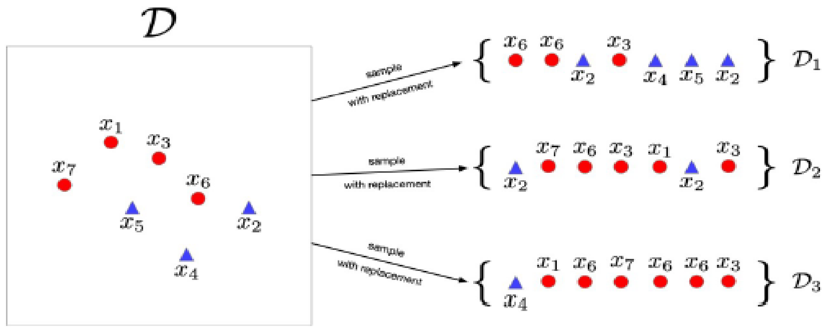
Bagging:

- Bagging stands for Bootstrap Aggregation.
- This is very useful method for the models with high variance and noisy data.
- Bagging introduces randomness into the rows of the data.
- It assigns equal weight to models created, thus helps in reducing the variance associated with classification.
- Here, random subsets of a dataset are created using replacement, meaning that the same data point may be present in several subsets.

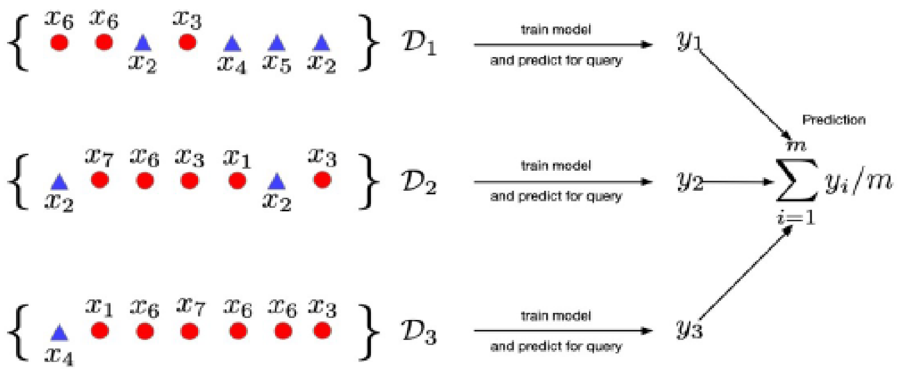
Bagging:

- Consider dataset D with n examples.
- Create M new datasets from D using sampling with replacement technique.
- Each dataset D_m has the same number of examples as in data set D (i.e. $(D_m)_{m=1}^{M}$).
- Train models $Y_1 \ldots Y_M$ using $D_1 \ldots D_M$ respectively.
- Combine all the models to find the final model $Y = 1/m \sum_{i=1}^{M} Y_M$

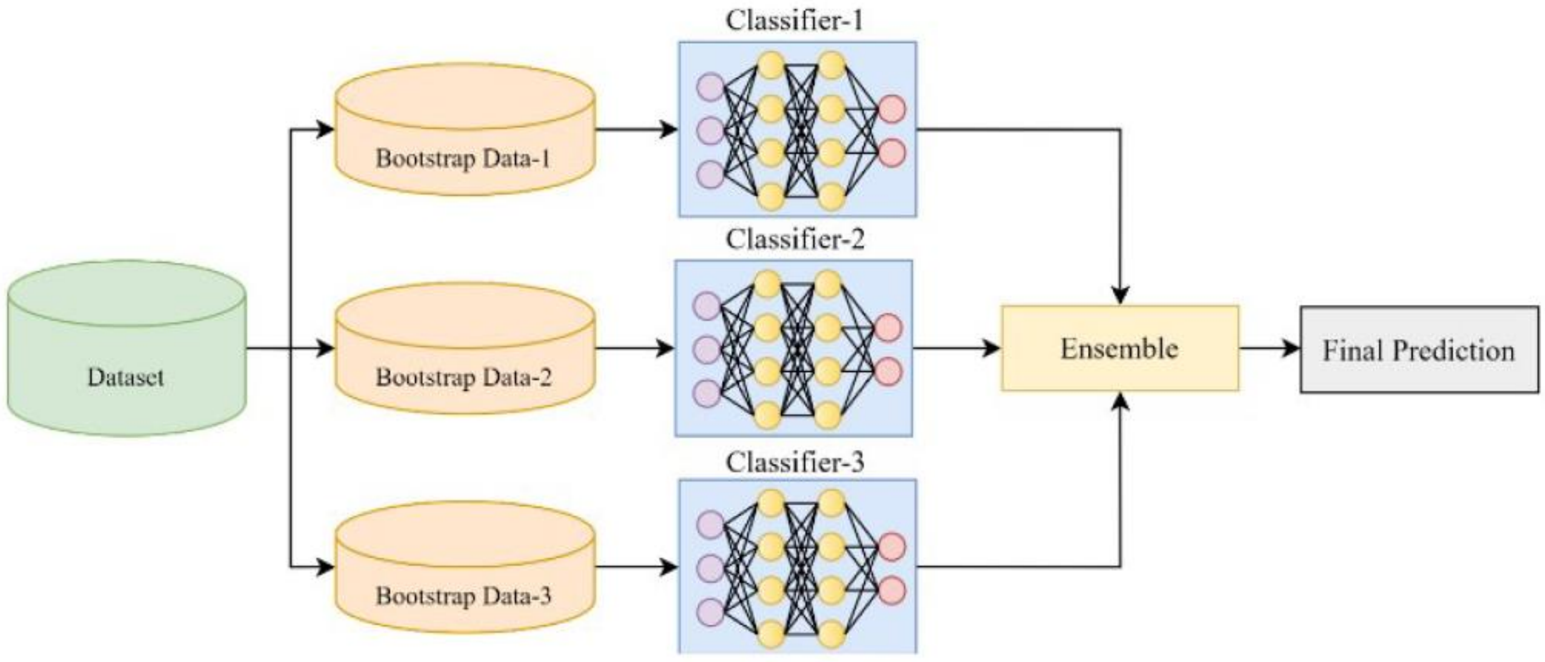
Bagging:


$\mathcal{D}$
x_1 x_3 x_7 x_6 x_5 x_2 x_4
sample with replacement
$\{x_6\ x_6\ x_2\ x_3\ x_4\ x_5\ x_2\}\ \mathcal{D}_1$
$\{x_2\ x_7\ x_6\ x_3\ x_1\ x_2\ x_3\}\ \mathcal{D}_2$
$\{x_4\ x_1\ x_6\ x_7\ x_6\ x_6\ x_3\}\ \mathcal{D}_3$

(a) Creation of multiple datasets from the original dataset (Bootstrap)


$\mathcal{D}_1$ train model and predict for query y_1
$\mathcal{D}_2$ train model and predict for query y_2
$\mathcal{D}_3$ train model and predict for query y_3
Prediction
$\sum_{i=1}^{m} y_i/m$

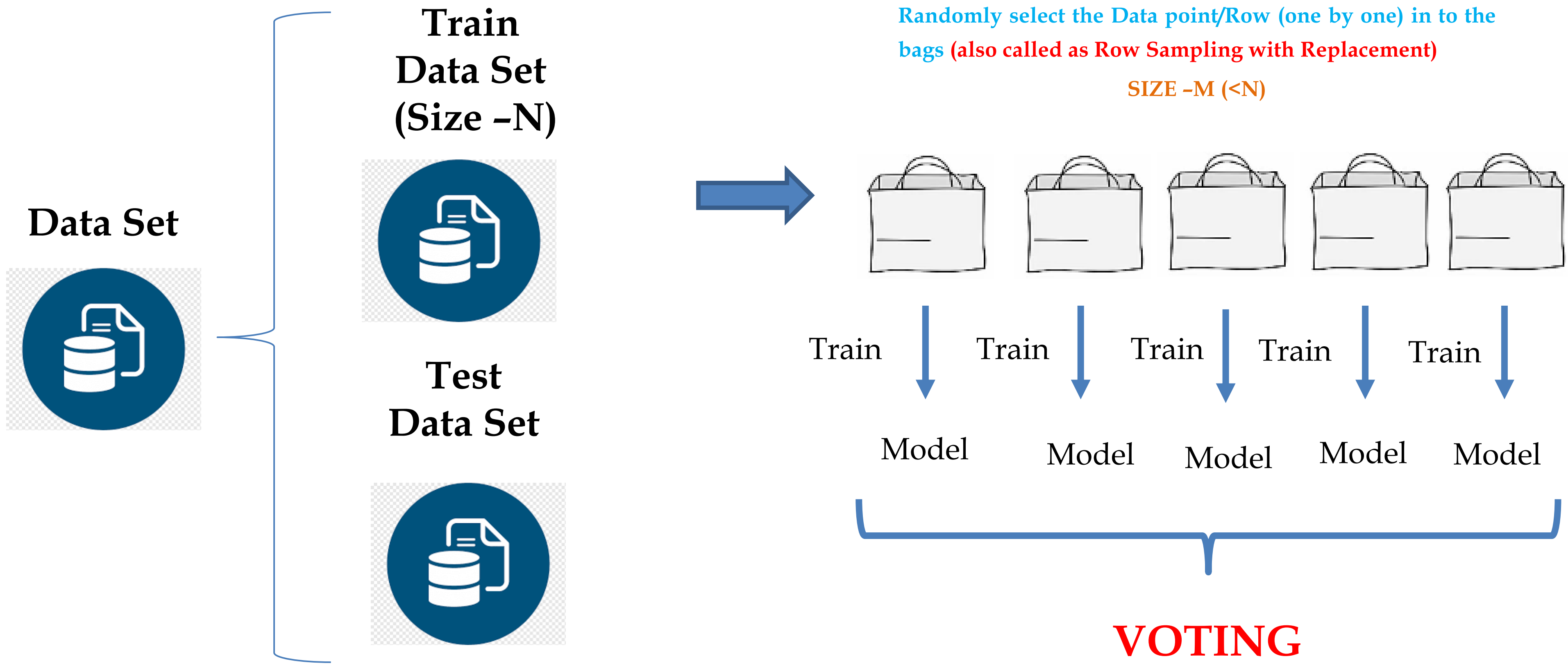
(b) Creation of final model



Bagging:

BOOTSTRAP AGGREGATING (BAGGING)

Multiple models of same learning algorithm trained with subsets of dataset randomly picked from the training dataset.



Randomly select the Data point/Row (one by one) in to the bags (also called as Row Sampling with Replacement)

SIZE –M (<N)

Bagging: Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Bootstrap sampling: sampling with replacement.

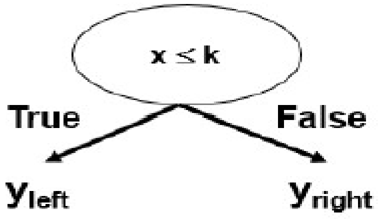
2

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

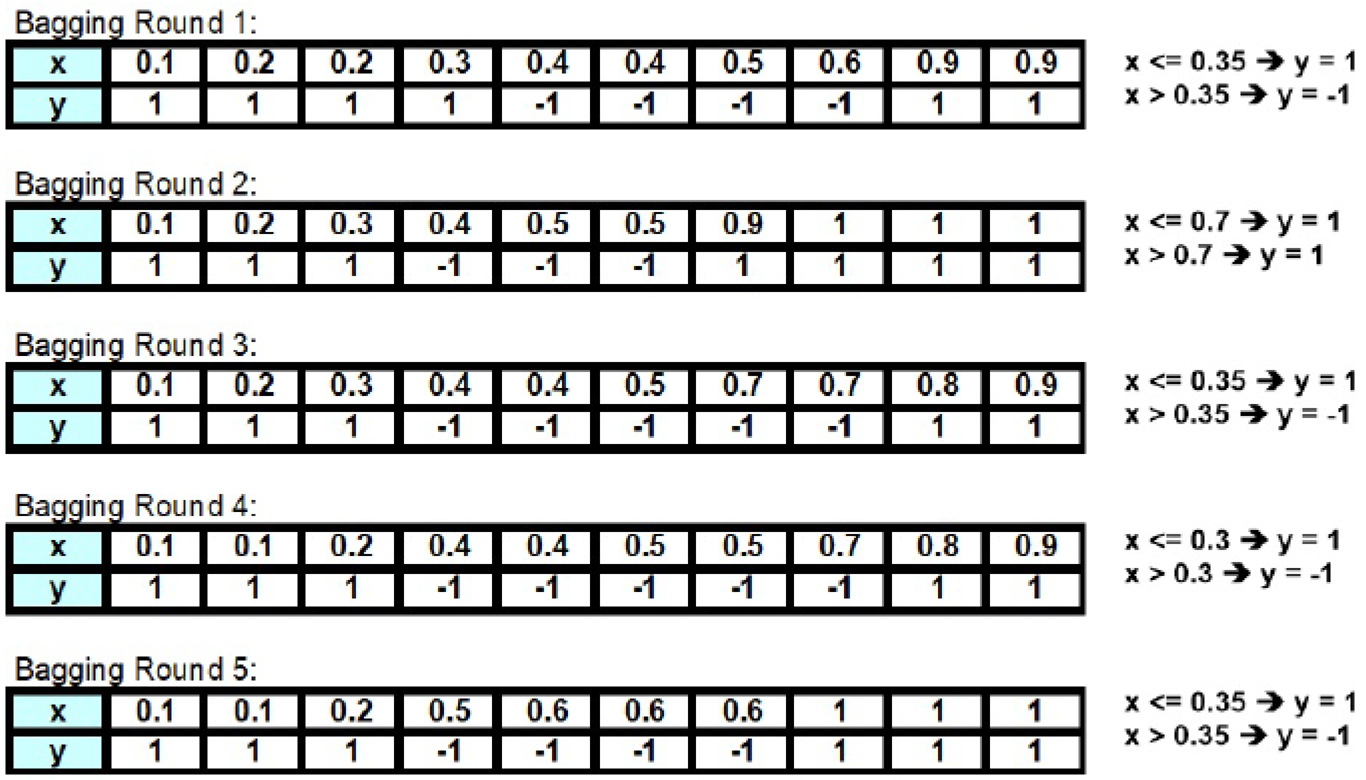
Here, decision tree classifier of size 1 is used (decision stump).

Decision rule for decision stump is as follows:

- Decision rule: $x \leq k$ versus $x > k$
- Split point k is chosen based on entropy



Bagging: Example



Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 → y = 1
x > 0.35 → y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 → y = 1
x > 0.7 → y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 → y = 1
x > 0.35 → y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 → y = 1
x > 0.3 → y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 → y = 1
x > 0.35 → y = -1

Bagging: Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

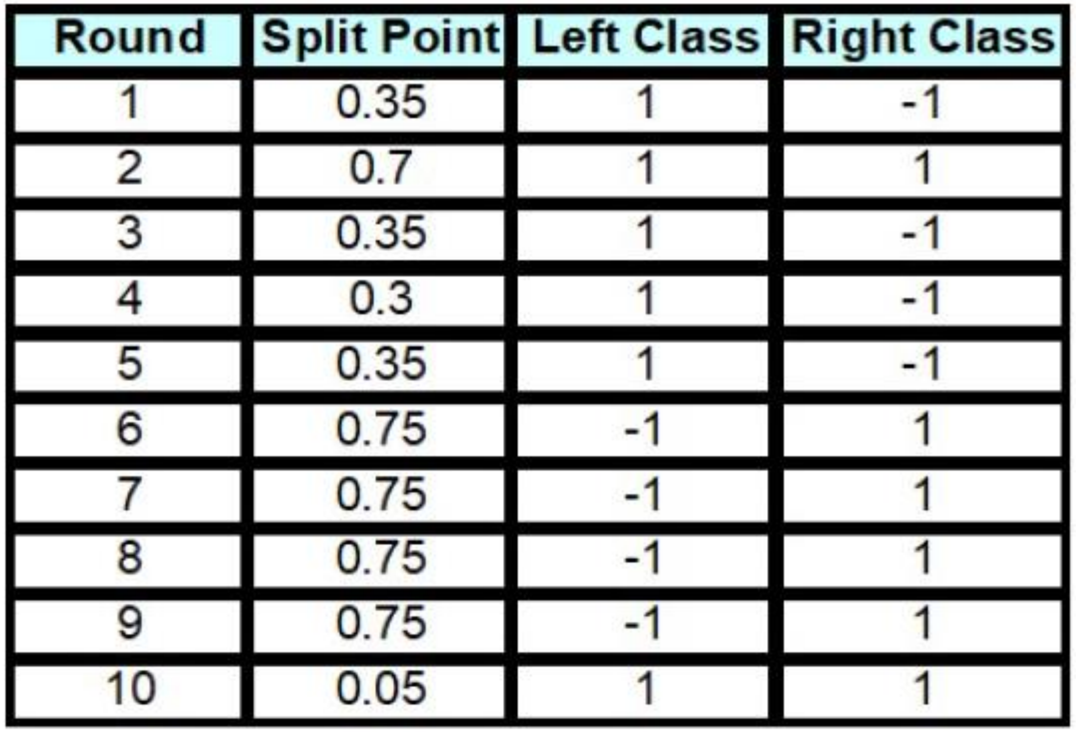
Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging: Example

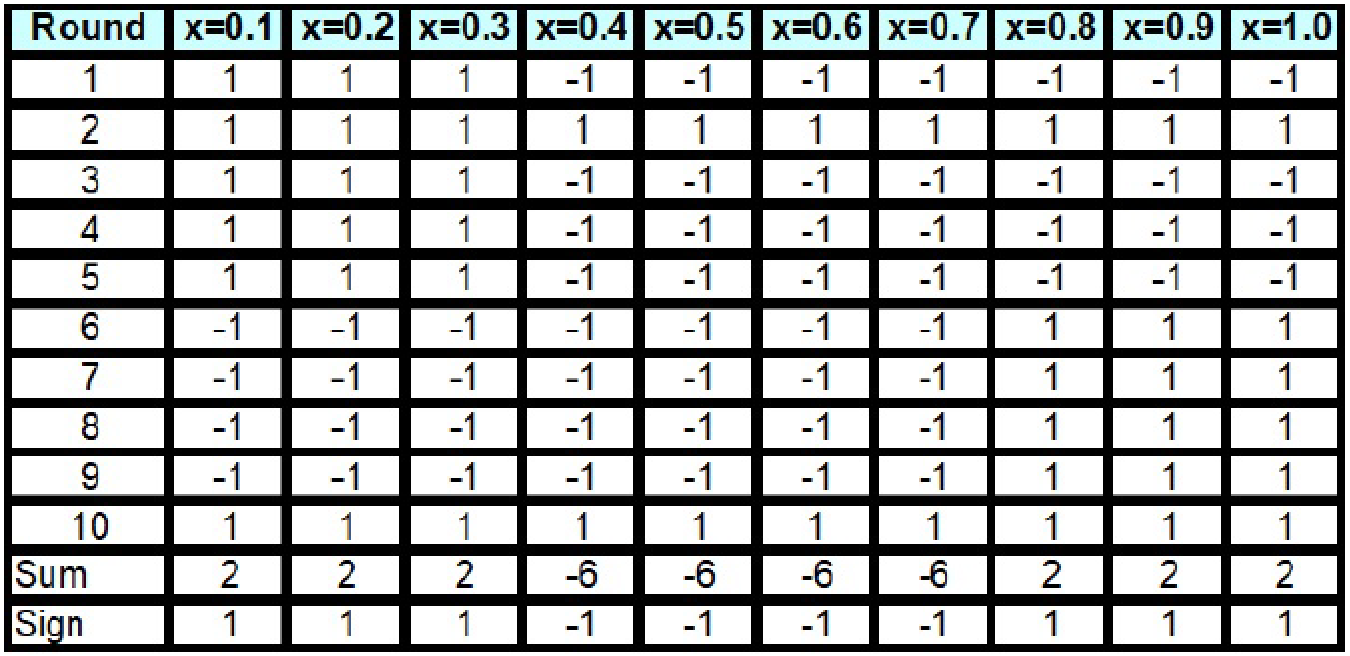
Summary of Trained Decision Stumps:



Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging: Example

Use majority vote (sign of sum of predictions) to determine class of ensemble classifier



Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Here, **Sign** indicates predicted class.

Bagging: Issues

The problem with decision trees is that they don't create smooth boundaries between different classes unless we break them down into too many branches, it is popularly known as "overfitting" problem.
This occurs when a machine learning model performs very well on training data but poorly on test examples from the real world.

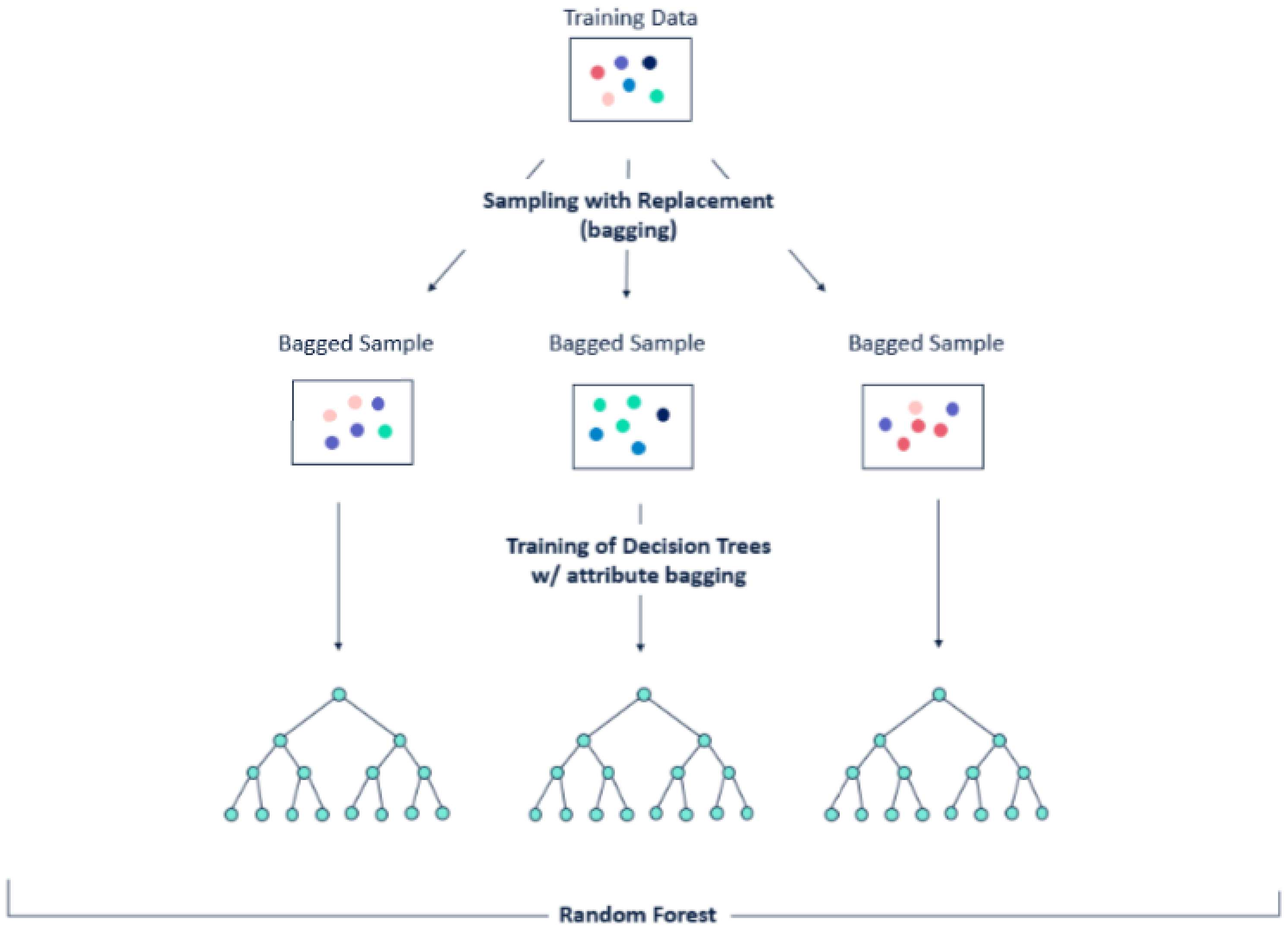
Random forest classifier

- ✓ Random forest classifier, an extension to bagging which uses *de-correlated* trees.
- ✓ To *de-correlate* the trees that make up a random forest, a process called bootstrap aggregating (also known as bagging) is conducted.
- ✓ **Two major limitations of decision trees are,**
  - ✓ **That they are prone to overfitting**
  - ✓ **That they tend to be non-robust, meaning a small change in the training data results in a very different tree.**

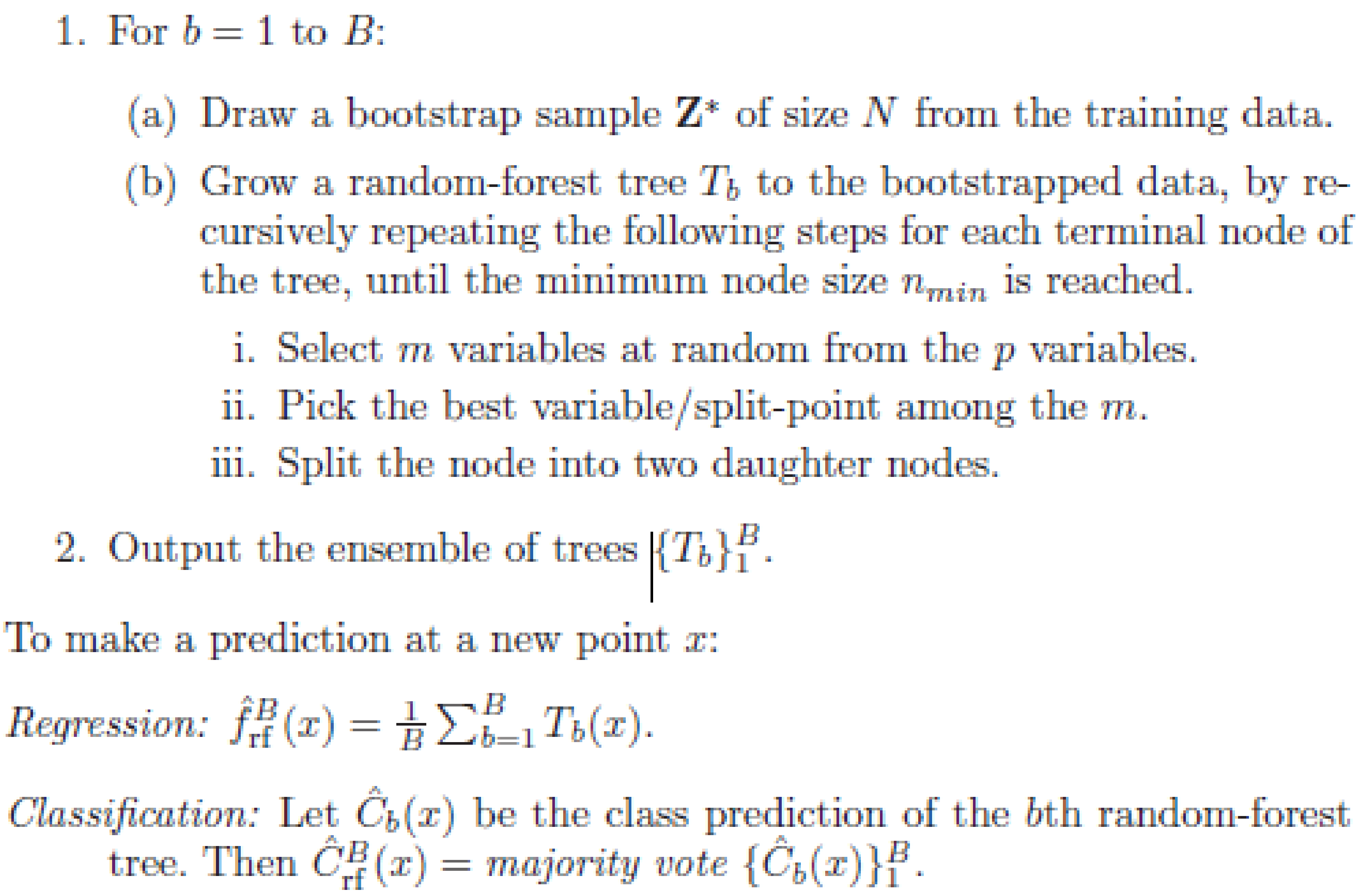
Bagging: Random Forest

- ✓ Bagging generates new training data sets from an original data set by sampling the original training data with replacement (bootstrapping).
- ✓ This is repeated for as many decision trees that that will make up the random forest.
- ✓ Each individual bootstrapped data set is then used to construct a tree.
- ✓ This process effectively decreases the variance (error introduced by random noise in the training data, i.e., overfitting) of the model without increasing the bias (underfitting).
- ✓ On its own, bagging the training data to generate multiple trees creates what is known as a bagged trees model.

Bagging: Random Forest


Training Data
Sampling with Replacement (bagging)
Bagged Sample
Bagged Sample
Bagged Sample
Training of Decision Trees w/ attribute bagging
Random Forest

Random Forest Algorithm



```
1. For $b = 1$ to $B$:
    (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by re-cursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
        i. Select $m$ variables at random from the $p$ variables.
        ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

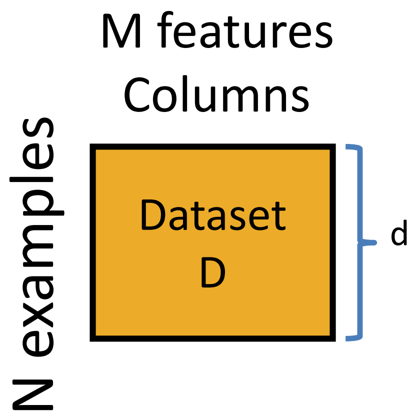
To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote } \{\hat{C}_b(x)\}_1^B$.

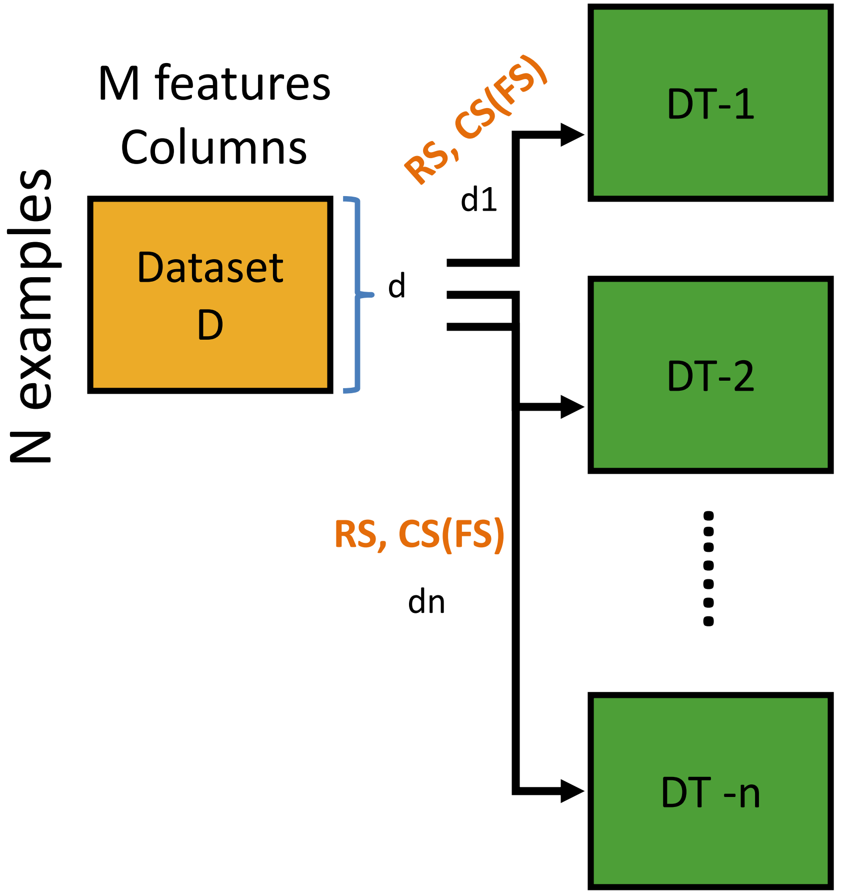
Random Forest Classifier Algorithm

Training Data



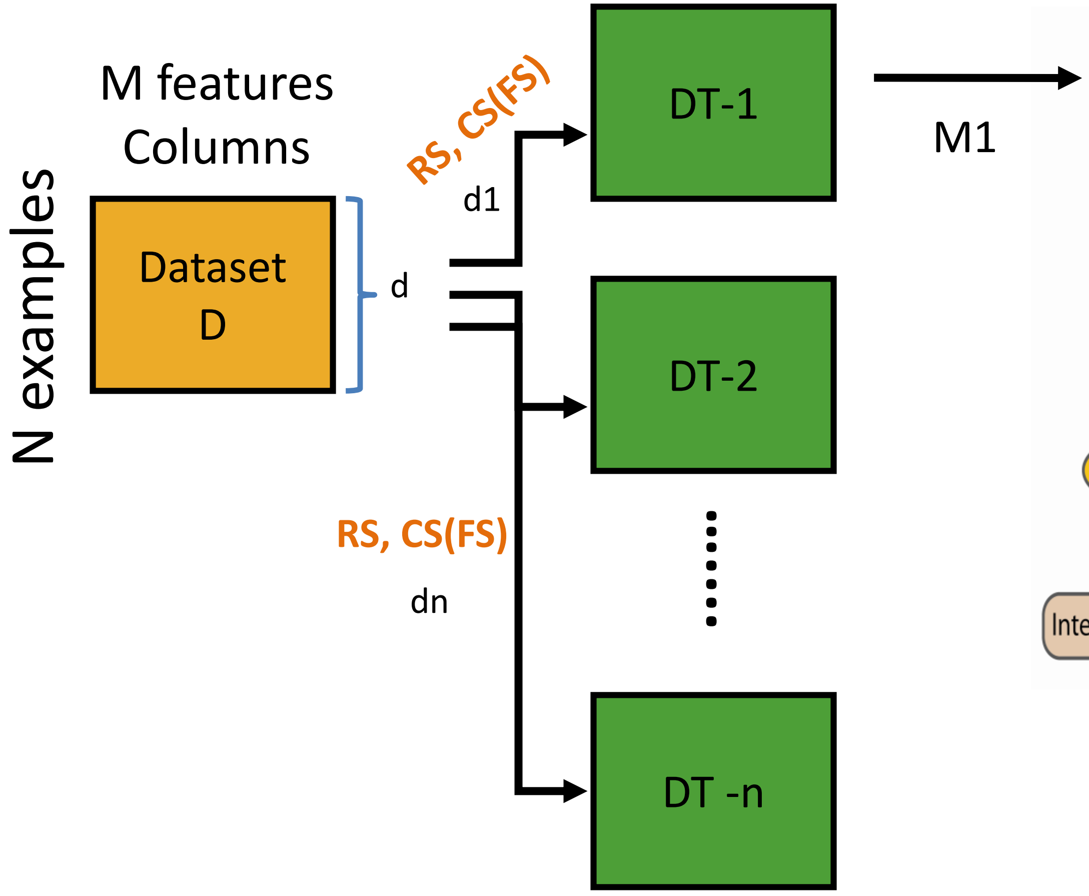
Random Forest Classifier Algorithm

Create bootstrap samples
from the training data
Base Learners
(RS+FS+Replacement)

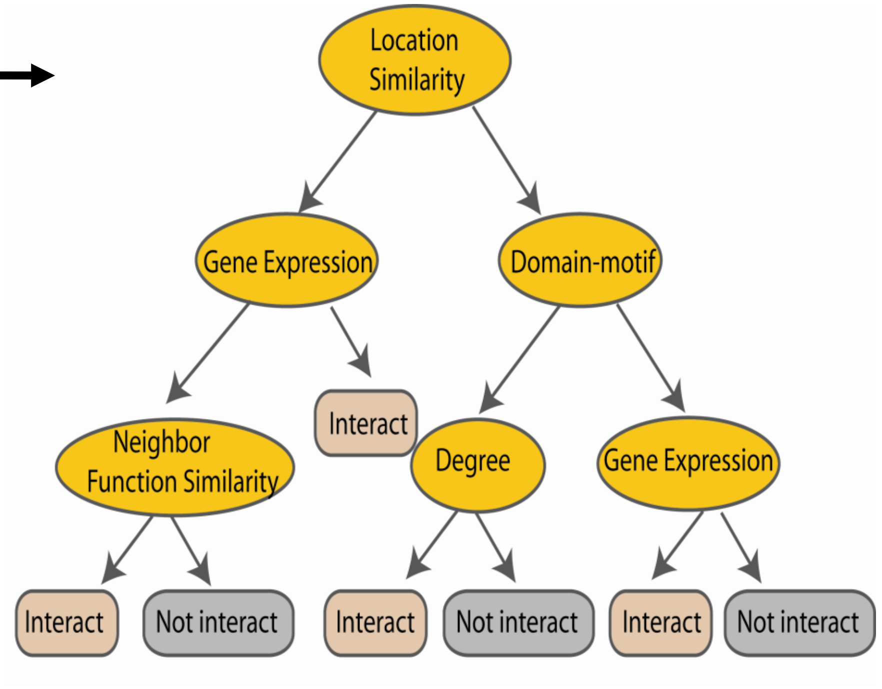


RS- Row Space
CS(FS) – Column/Feature Space

Random Forest Classifier Algorithm

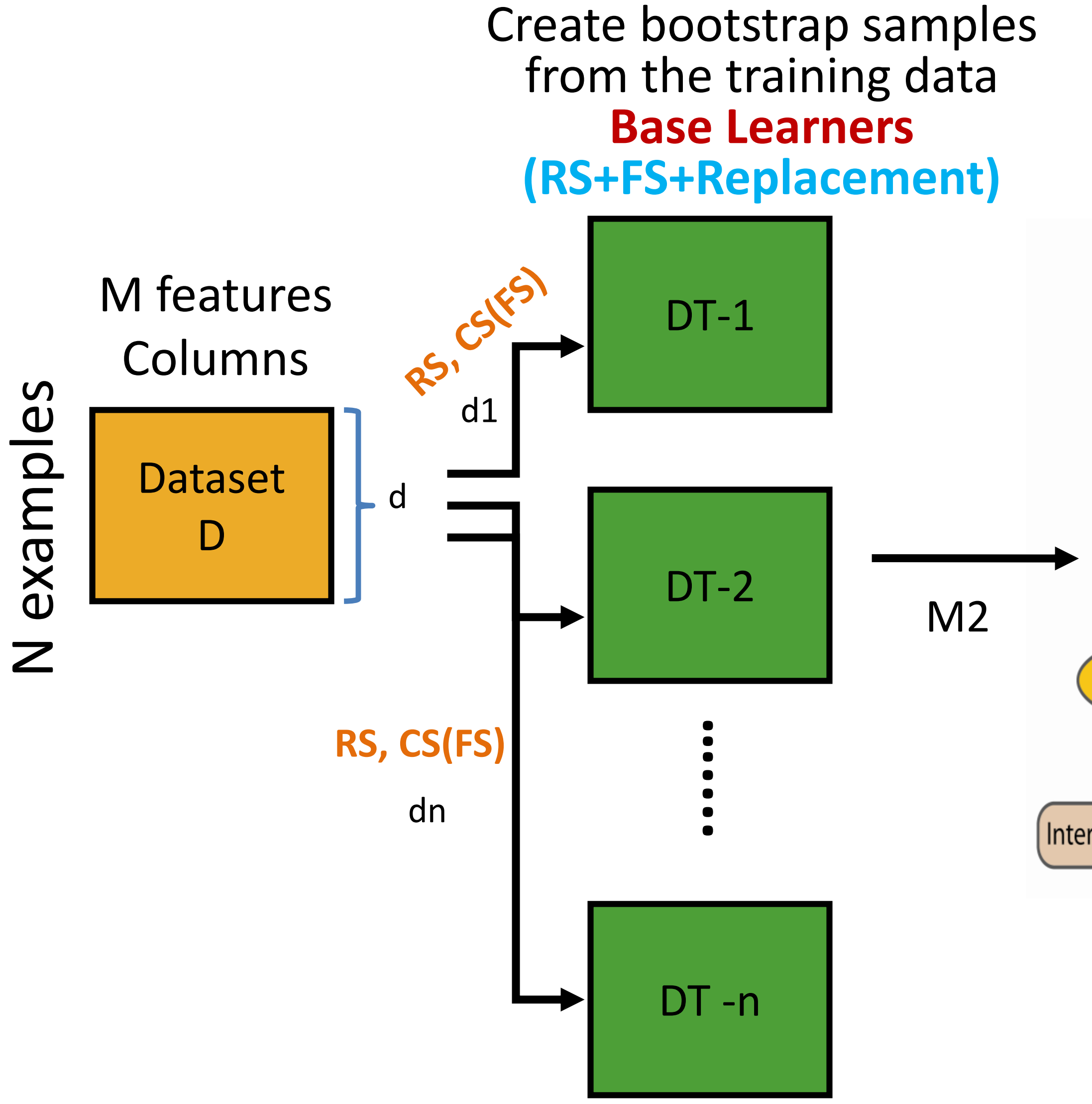


Construct a decision tree

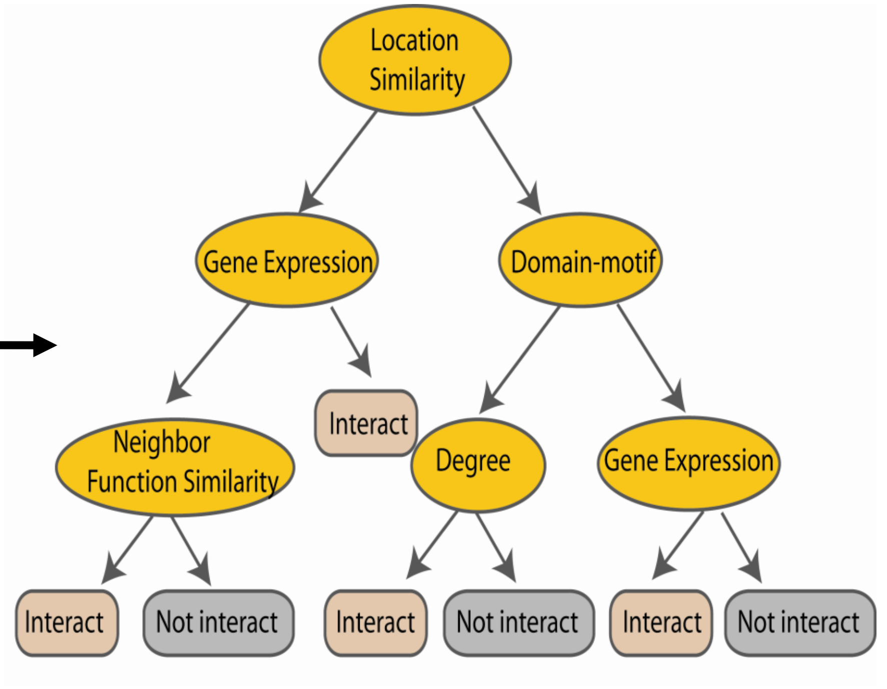


RS- Row Space
CS(FS) – Column/Feature Space

Random Forest Classifier Algorithm

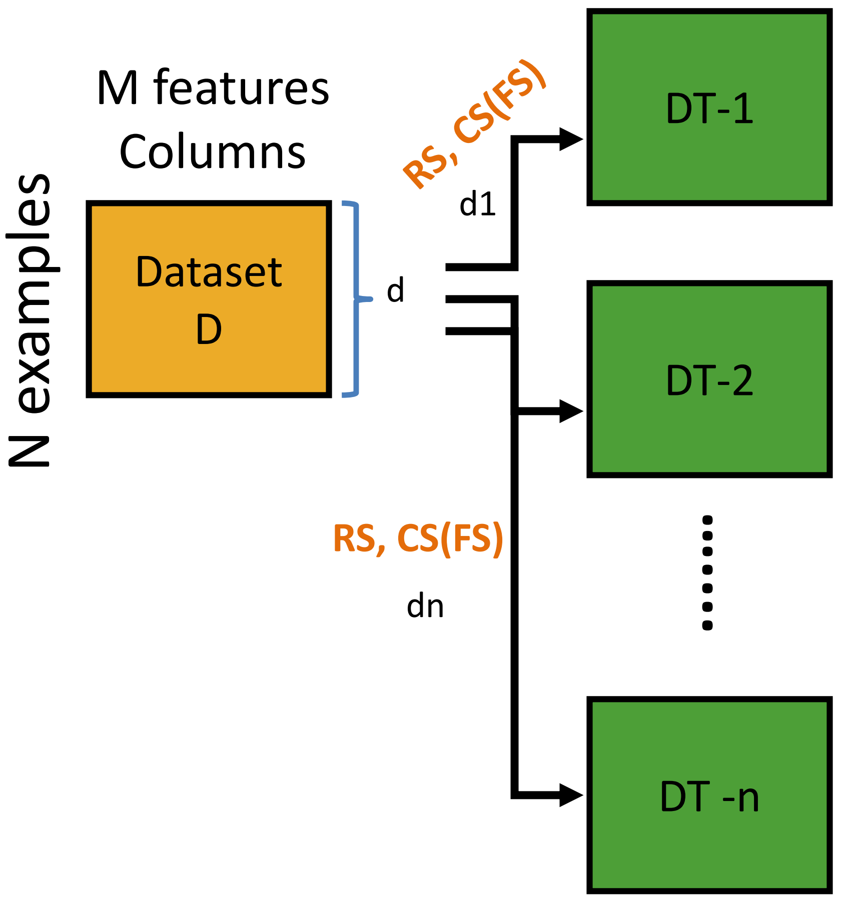


Construct a decision tree

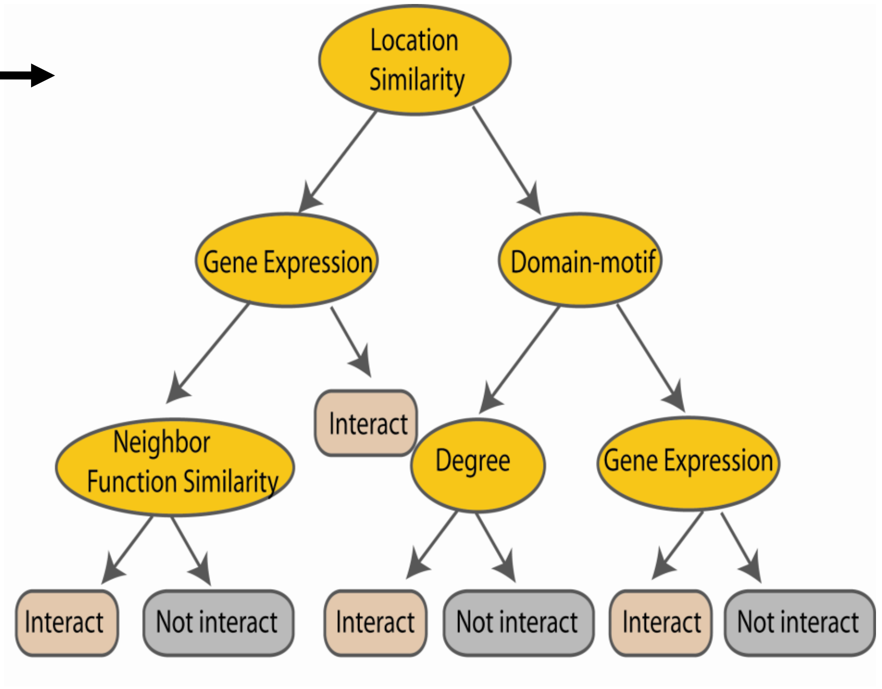


RS- Row Space
CS(FS) – Column/Feature Space

Random Forest Classifier Algorithm



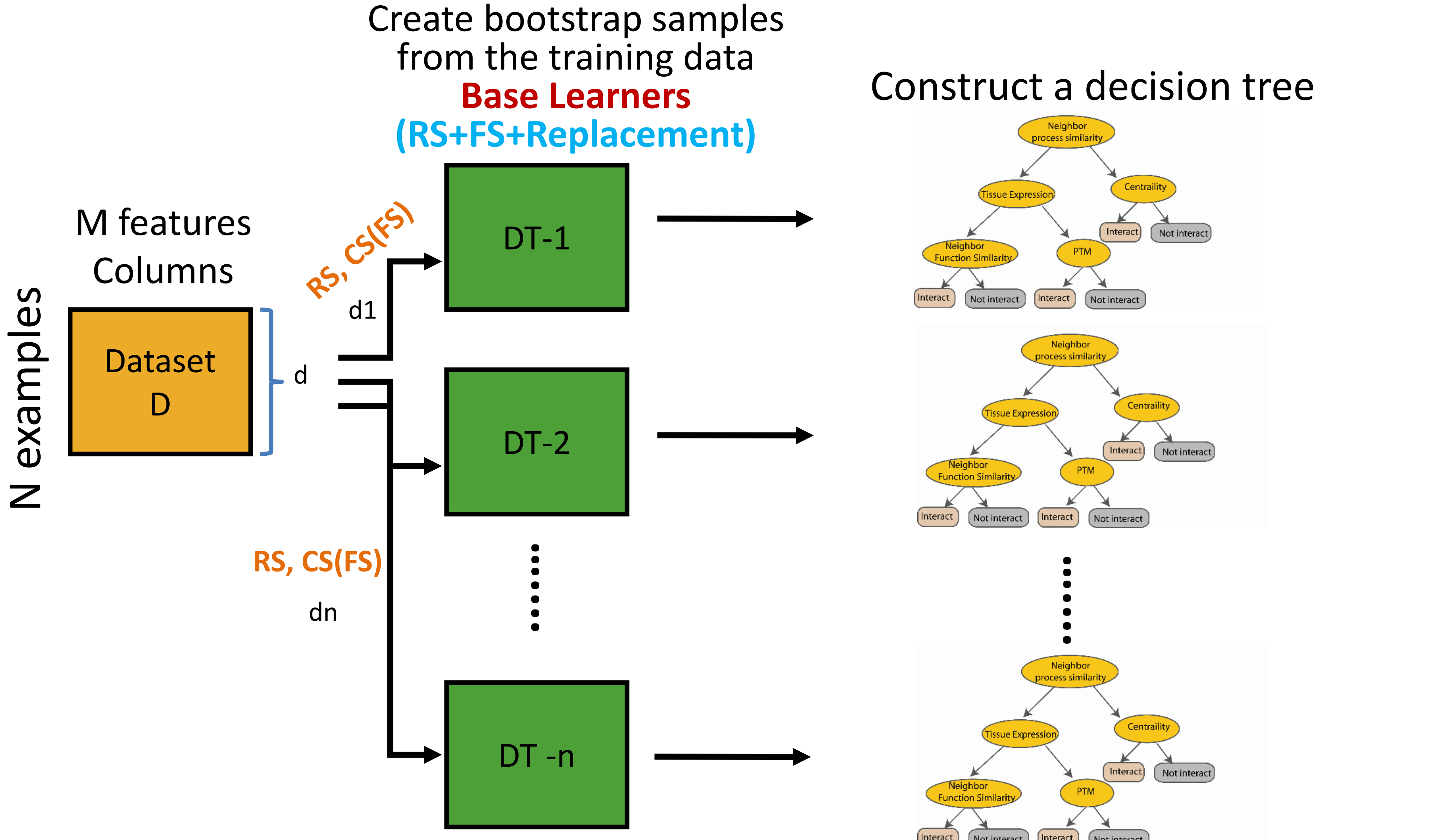
Construct a decision tree



At each node in choosing the split feature choose only among *m<M* features

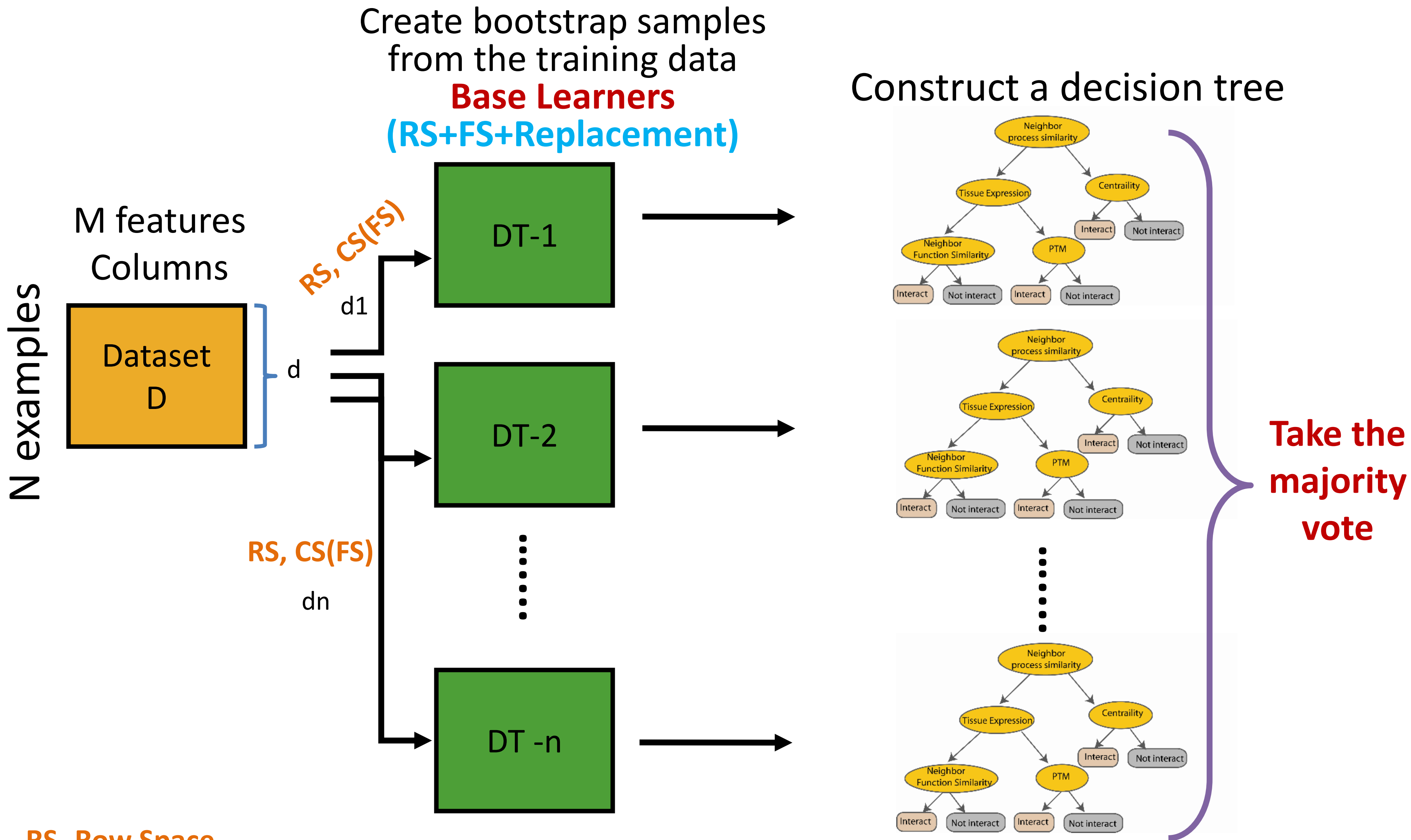
RS- Row Space
CS(FS) – Column/Feature Space

Random Forest Classifier Algorithm



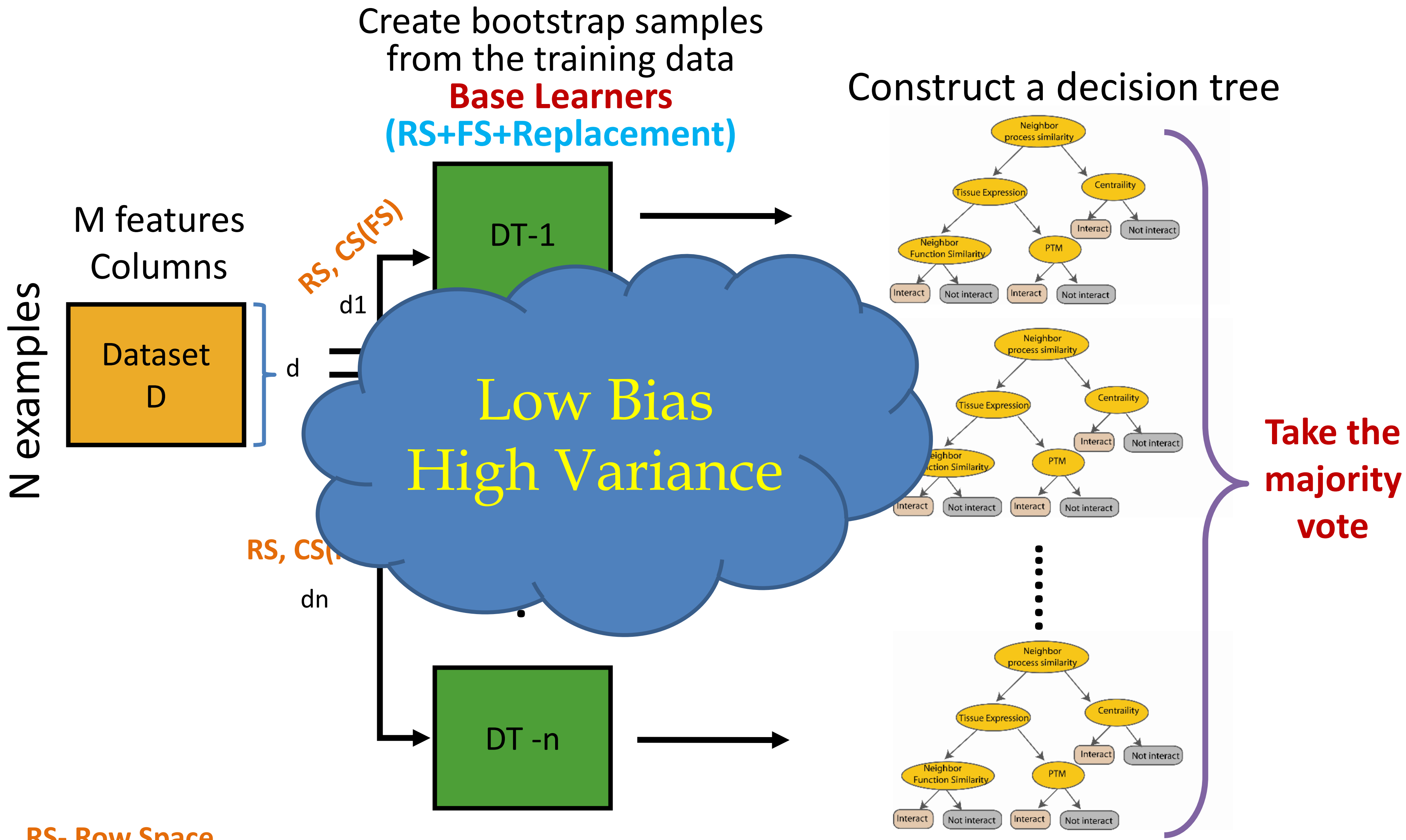
RS- Row Space
CS(FS) – Column/Feature Space

Random Forest Classifier Algorithm



RS- Row Space
CS(FS) – Column/Feature Space

Random Forest Classifier Algorithm



RS- Row Space
CS(FS) – Column/Feature Space

Comparison

- When we create DT to its complete depth then it will properly trained for train dataset and error will be less. **(Low Bias)**
- Whenever we get new test dataset they will give larger Error **(High Variance)**

Decision trees	Random Forest
1. Decision trees normally suffer from the problem of overfitting if it's allowed to grow without any control.	1. Random forests are created from subsets of data and the final output is based on average or majority ranking and hence the problem of overfitting is taken care of.
2. A single decision tree is faster in computation.	2. It is comparatively slower.
3. When a data set with features is taken as input by a decision tree it will formulate some set of rules to do prediction.	3. Random forest randomly selects observations, builds a decision tree and the average result is taken. It doesn't use any set of formulas.

Random Forest Classifier Algorithm

Create bootstrap samples
from the training data

Base l

ision tree

M features
Columns

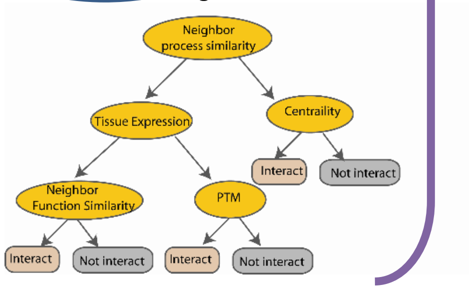
N examples

Da

Regression: Mean/Median of output
Classification: Majority of Vote

Take the majority vote

DT -n



RS- Row Space
CS(FS) – Column/Feature Space

Boosting

- Boosting does not involve bootstrap sampling.
- Trees are grown sequentially: Each tree is grown using information from previously grown trees.

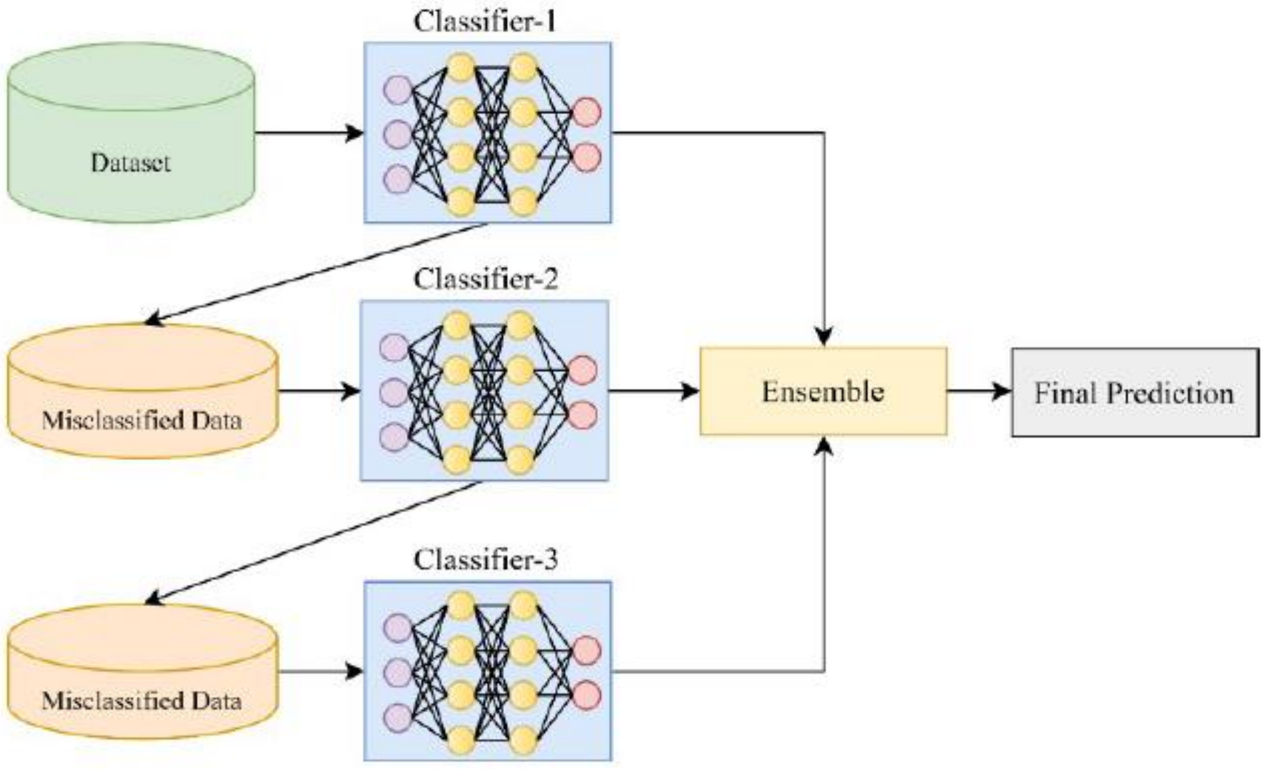
An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records

- Initially, all N records are assigned equal weights (for being selected for training)
- Unlike bagging, weights may change at the end of each boosting round
- Records that are wrongly classified will have their weights increased in the next round
- Records that are classified correctly will have their weights decreased in the next round

Boosting

- Given a weak learner, run it multiple times on (reweighted) training data, then let the learned classifiers vote
- On each iteration t :
  - Weight each training example by ”how incorrectly” it was classified
  - Learn a weak hypothesis – h_t
  - A strength for this hypothesis – α_t
  - A linear combination of the votes of the different classifiers weighted by their strength i.e $H(x) = \text{sgn} \sum(\alpha_t h_t(x))$

Boosting : Graphical Representation



Boosting : AdaBoost

- AdaBoost, short for Adaptive Boosting, is a machine learning algorithm formulated by Yoav Freund and Robert Schapire.
- Follows a decision tree model with a depth equal to one.
- Nothing but the forest of stumps rather than trees.
- It works by putting more weight on difficult to classify instances and less on those already handled well.
- AdaBoost algorithm is developed to solve both classification and regression problem.

Boosting : AdaBoost

Idea behind AdaBoost:

1. Stumps (one node and two leaves) are not great in making accurate classification so it is nothing but a week classifier/ weak learner.
2. Combination of many weak classifier makes a strong classifier and this is the principle behind the AdaBoost algorithm.
3. Some stumps get more performance or classify better than others.
4. Consecutive stump is made by taking the previous stumps mistakes into account.

Boosting : AdaBoost Algorithm

Step 1: **Assign Equal Weights to all the observations**

Step 2: **Classify random samples using stumps**

Step 3: **Calculate Total Error**

Step 4: **Calculate Performance of the Stump**

Step 5: **Update Weights**

Step 6: **Update weights in iteration**

Step 7: **Final Predictions**

Boosting : AdaBoost Algorithm

Dataset Preparation:

```
#considering only two classes
example = iris[(iris['Species'] == 'versicolor') | (iris['Species'] == 'virginica')]
```

Step 1: Assign Equal Weights to all the observations

- Initially assign same weights to each record in the dataset.
- Sample weight = 1/N
    - Where N = Number of records

```
#Initially assign same weights to each records in the dataset
example['probR1'] = 1/(example.shape[0])
```

Boosting : AdaBoost Algorithm

KLE Technological University
Creating Value
Leveraging Knowledge

Step 2: Classify random samples using stumps

- Draw random samples with replacement from original data with the probabilities equal to the sample weights and fit the model.
- Here the model (base learners) used in AdaBoost is decision tree.
- Decision trees are created with one depth which has one node and two leaves also referred to as stumps.
- Fit the model to the random samples and predict the classes for the original data.

```
#simple random sample with replacement
random.seed(10)
example1 = example.sample(len(example), replace = True, weights = example['probR1'])
```

```
#fitting the DT model with depth one
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=1)
clf = clf_gini.fit(X_train, y_train)
```

Pred1
Newly
predicted
class

SepalLength	SepalWidth	PetalLength	PetalWidth	Label	probR1	pred1
7	3.2	4.7	1.4	1	0.01	1
5.9	3.2	4.8	1.8	1	0.01	-1

Boosting : AdaBoost Algorithm

Step 3: Calculate Total Error

Total error is nothing but the sum of weights of misclassified record.

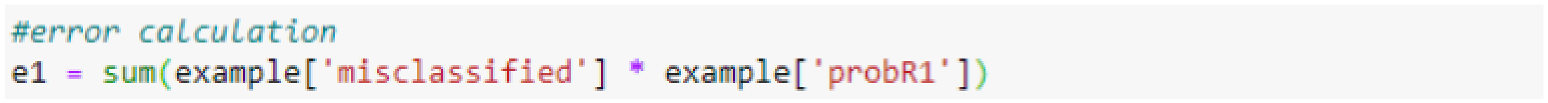
Total Error = Weights of misclassified records

Total error will be always between 0 and 1.

0 represents perfect stump (correct classification)

1 represents weak stump (misclassification)

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^{N} w_j^{(i)} \, \delta(C_i(x_j) \neq y_j)$$



```
#error calculation
e1 = sum(example['misclassified'] * example['probR1'])
```

Boosting : AdaBoost Algorithm

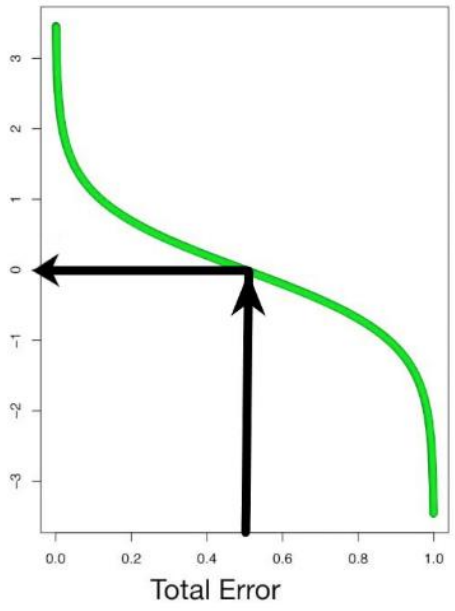
Step 4: Calculate Performance of the Stump

- Using the Total Error, determine the performance of the base learner.
- The calculated performance of stump(α) value is used to update the weights in consecutive iteration and also used for final prediction calculation.

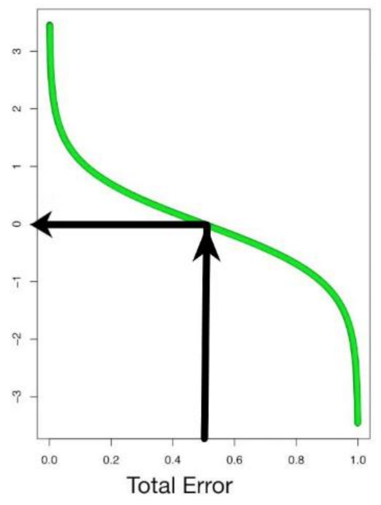
Performance of the stump(α) = ½ ln (1 – Total error/Total error)

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

```
#calculation of alpha (performance)
alpha1 = 0.5*log((1-e1)/e1)
```



Boosting : AdaBoost Algorithm



Cases:

- If the total error is 0.5, then the performance of the stump will be zero.
- If the total error is 0 or 1, then the performance will become infinity or -infinity respectively.
- When the performance(α) is relatively large, the stump did a good job in classifying the records.
- When the performance(α) is relatively low, the stump did not do a good job in classifying the records.
- **Using the performance parameter(α), we can increase the weights of the wrongly classified records and decrease the weights of the correctly classified records.**

Boosting : AdaBoost Algorithm

Step 5: Update Weight

- Based on the performance of the stump(α) update the weights.
- The next stump to correctly classify the misclassified record by increasing the corresponding sample weight and decreasing the sample weights of the correctly classified records.

New weight = Weight * $e^{(\text{performance})}$ → misclassified records
New weight = Weight * $e^{-(\text{performance})}$ → correctly classified records

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \times \begin{cases} e^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ e^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

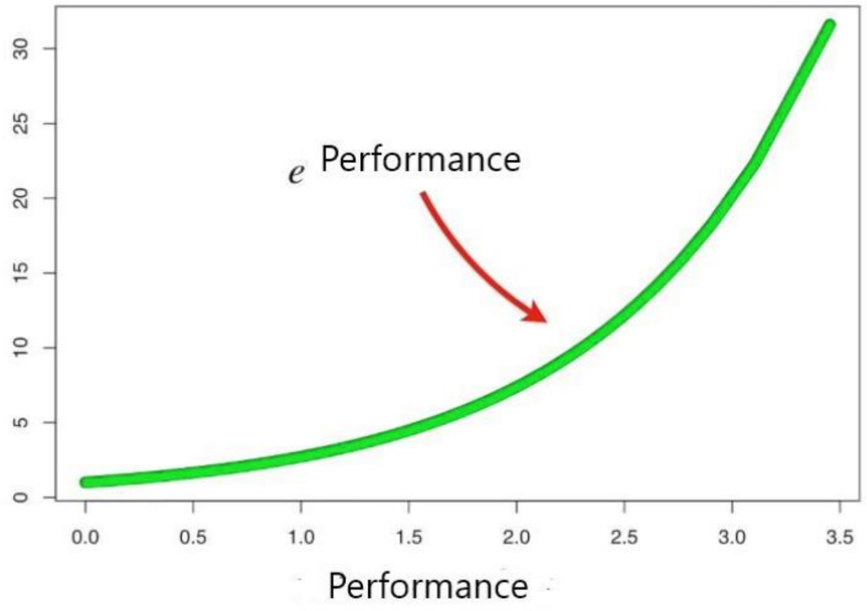
Where Z_i is the normalization factor

If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated

Boosting : AdaBoost Algorithm

Note on E ^ Performance

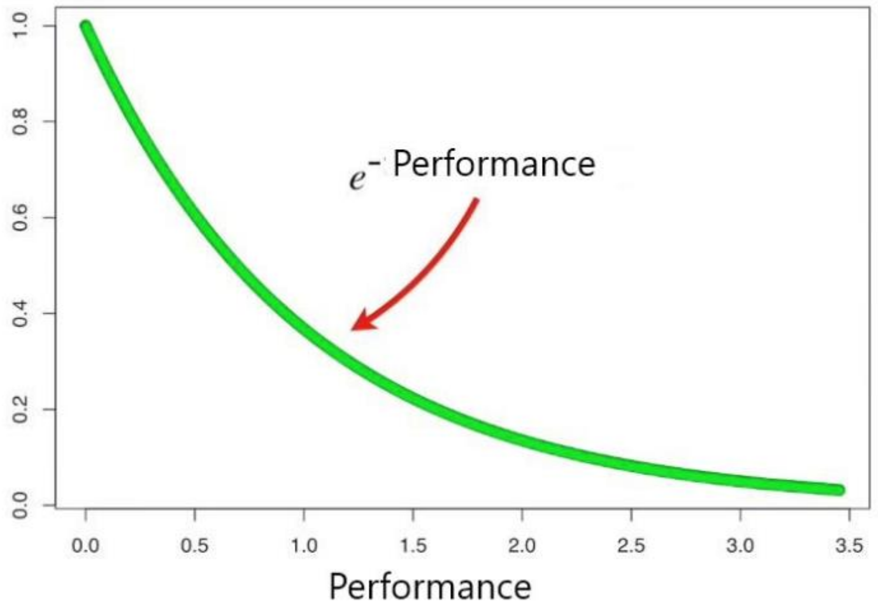
- When the performance is relatively large the last stump did a good job in classifying the records now the new sample weight will be much larger than the old one.
- When the performance is relatively low the last stump did not do a good job in classifying the records now the new sample weight will only be little larger than the old one.



Boosting : AdaBoost Algorithm

Note on E ^ - Performance

- When the performance is relatively large the last stump did a good job in classifying the records now the new sample weight will be very small than the old one.
- When the performance is relatively small the last stump did not do a good job in classifying the records now the new sample weight will only be little smaller than the old one.



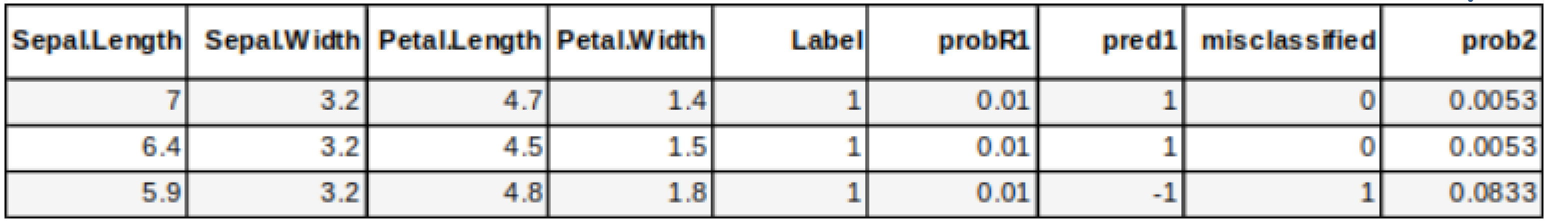
Boosting : AdaBoost Algorithm

- Here the sum of the updated weights is not equal to 1.
- whereas in case of initial sample weight the sum of total weights is equal to 1.
- So, to achieve this we will be dividing it by a number which is nothing but the sum of the updated weights (normalizing constant).

Normalizing constant = ∑ New weight
Normalized weight = New weight / Normalizing constant
Now the sum of normalized weight is equal to 1.

```
#normalized weight
z = sum(new_weight)
normalized_weight = new_weight/sum(new_weight)
```

Pred2 Newly predicted class

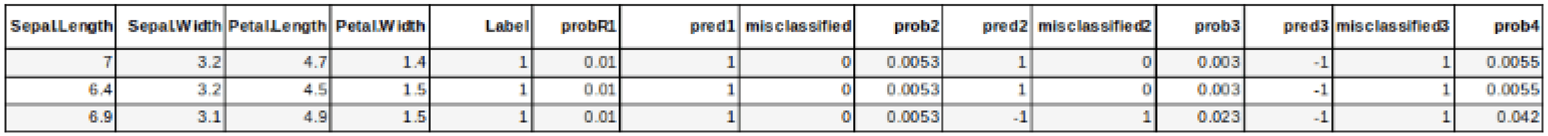


SepalLength	SepalWidth	PetalLength	PetalWidth	Label	probR1	pred1	misclassified	prob2
7	3.2	4.7	1.4	1	0.01	1	0	0.0053
6.4	3.2	4.5	1.5	1	0.01	1	0	0.0053
5.9	3.2	4.8	1.8	1	0.01	-1	1	0.0833

Boosting : AdaBoost Algorithm

Step 6: Update Weights in the iteration

- Use the normalized weight and make the second stump in the forest. Create a new dataset of same size of the original dataset with repetition based on the newly updated sample weight.
- So that the misclassified records get higher probability of getting selected. Repeat step 2 to 5 again by updating the weights for a particular number of iterations.



SepalLength	SepalWidth	PetalLength	PetalWidth	Label	probR1	pred1	misclassified	prob2	pred2	misclassified2	prob3	pred3	misclassified3	prob4
7	3.2	4.7	1.4	1	0.01	1	0	0.0053	1	0	0.003	-1	1	0.0055
6.4	3.2	4.5	1.5	1	0.01	1	0	0.0053	1	0	0.003	-1	1	0.0055
6.9	3.1	4.9	1.5	1	0.01	1	0	0.0053	-1	1	0.023	-1	1	0.042

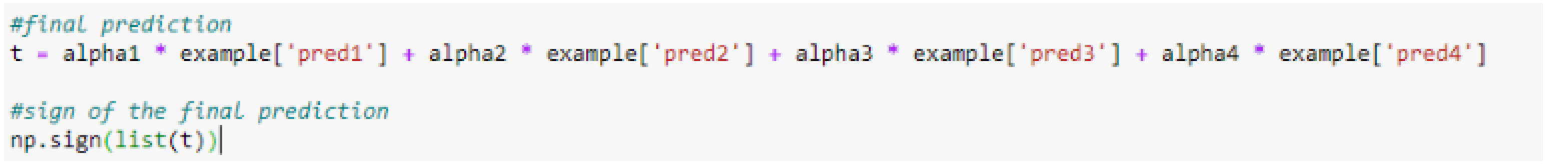
Pred4
Final
Weights

Boosting : AdaBoost Algorithm

Step 7: Final Predictions

Final prediction is done by obtaining the sign of the weighted sum of final predicted value.

Final prediction/sign (weighted sum) = $\sum$ (α_i* (predicted value at each iteration))



```
#final prediction
t = alpha1 * example['pred1'] + alpha2 * example['pred2'] + alpha3 * example['pred3'] + alpha4 * example['pred4']

#sign of the final prediction
np.sign(list(t))
```

Boosting : AdaBoost Algorithm

Advantages of AdaBoost Algorithm:

- One of the many advantages of the AdaBoost Algorithm is it is fast, simple and easy to program.
- Boosting has been shown to be robust to overfitting.
- It has been extended to learning problems beyond binary classification (i.e.) it can be used with text or numeric data.

Drawbacks:

- AdaBoost can be sensitive to noisy data and outliers.
- Weak classifiers being too weak can lead to low margins and overfitting.

Boosting : AdaBoost Algorithm

Algorithm 4.6 AdaBoost algorithm.

```
1:  $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$.   {Initialize the weights for all $N$ examples.}
2:  Let $k$ be the number of boosting rounds.
3:  for $i = 1$ to $k$ do
4:      Create training set $D_i$ by sampling (with replacement) from $D$ according to $\mathbf{w}$.
5:      Train a base classifier $C_i$ on $D_i$.
6:      Apply $C_i$ to all examples in the original training set, $D$.
7:      $\epsilon_i = \frac{1}{N}\left[\sum_j w_j \, \delta\big(C_i(x_j) \neq y_j\big)\right]$   {Calculate the weighted error.}
8:      if $\epsilon_i > 0.5$ then
9:          $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$.   {Reset the weights for all $N$ examples.}
10:         Go back to Step 4.
11:     end if
12:     $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13:     Update the weight of each example according to Equation 4.103.
14: end for
15: $C^*(\mathbf{x}) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y)\big)$.
```

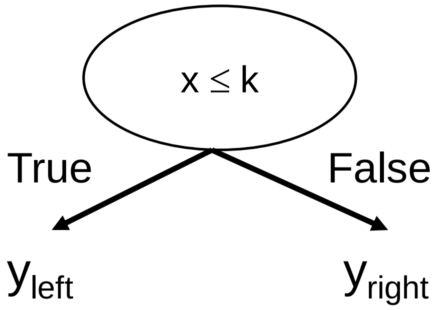
Boosting : AdaBoost Algorithm

- Consider 1-dimensional data set:

Original Data:

x	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1**
y	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**	**1**	**1**	**1**

- Classifier is a decision stump
  - Decision rule: $x \le k$ versus $x > k$
  - Split point k is chosen based on entropy



Boosting : AdaBoost Algorithm

- Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

Boosting : AdaBoost Algorithm

- Weights

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

- Classification

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Motivation for Stacking

- For each of our ensemble methods, we have:
  1. Fit the base model on the same type.
  2. Combined the predictions in a naïve way.

Stacking

Stacking is a way to generalize the ensembling approach to combine outputs of various types of model.

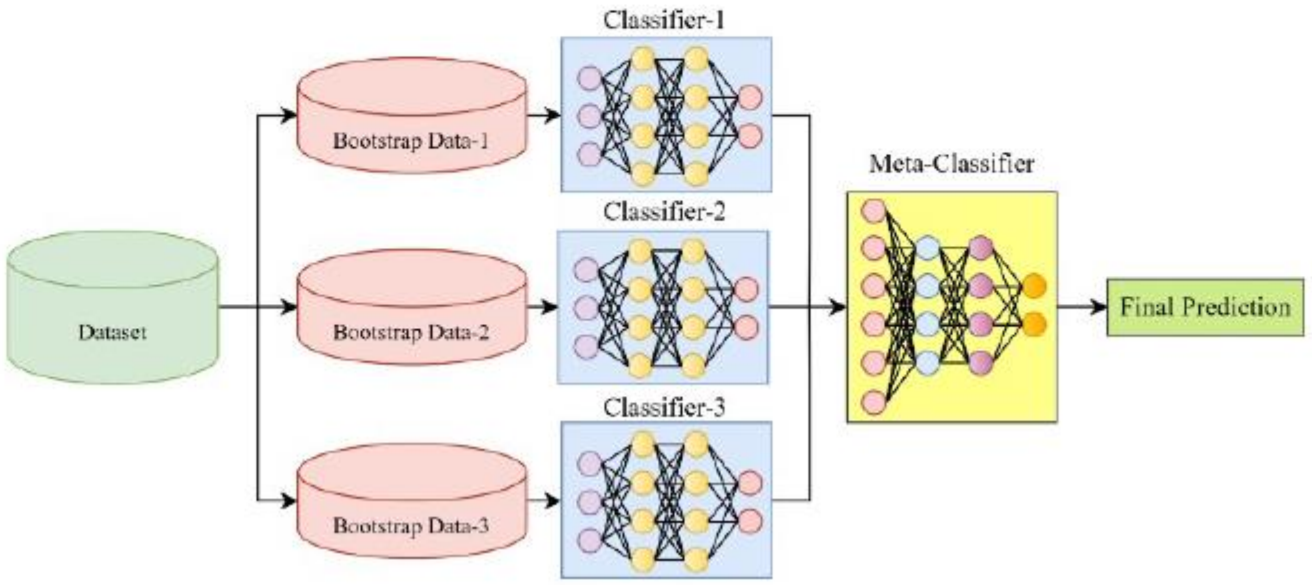
Idea

- Train L number of models, T_L on the training data $\{X_1, Y_1, \ldots, X_N, Y_N\}$.
- Train a meta-learner($\hat{T}$) on the predictions of the ensemble of models $\{(T_X, \ldots, T_L(X_1), Y_1), \ldots (T_1(X_N), \ldots, T_L(X_N), Y_N)\}$
- Here, $T_L(X_N)$ is generated by training T_L on $\{(X_1, Y_1), \ldots, (X_{n-1}, Y_{n-1}), (X_{n+1}, Y_{n+1}), \ldots, (X_N, Y_N)\}$

Stacking

It is widely used due to its flexibility but difficult to analyze theoretically. Some general rules have been found through empirical studies:

- Models in the ensemble should be diverse, i.e. their errors should not be correlated.
- it’s better to train the meta-learner on probabilities rather than predictions.
- Apply regularization to the meta-learner to avoid overfitting.



Challenges of Ensemble Learning

It is widely used due to its flexibility but difficult to analyze theoretically. Some general rules have been found through empirical studies:

▶ Takes more time to train.

▶ Explainability: A single machine learning model is easy to trace, but when you have hundreds of models contributing to an output, it is much more difficult to make sense of the logic behind each decision.

Resampling

1.Resampling: Drawing repeated samples from the original data samples, using the observer/generated data.

2. It produce new hypothetical situations/samples that mimic the underlying population, which can then be analyzed.

Why resampling methods?

1. Collecting data is expensive.
2. There is not enough data available, or there is insufficient information about the distribution.
3. Overfitting problem.

Resampling

1. Resampling methods are considered one of the most commonly used methods to deal with imbalanced datasets.
2. Resampling techniques include removing examples from the majority class (undersampling) or duplicating examples from the minority class (oversampling), as shown in Figure 7 [?].

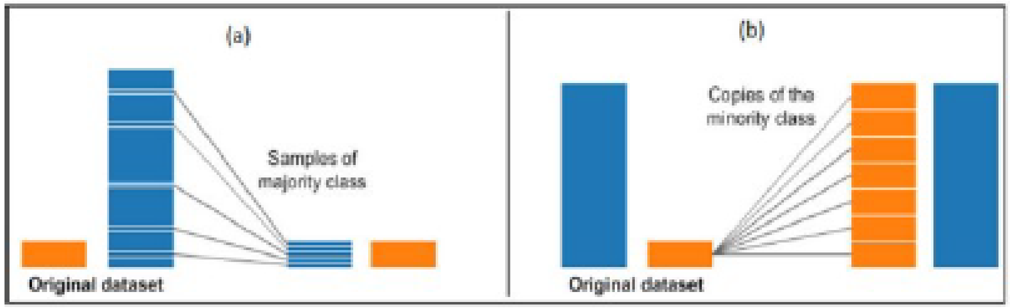


Figure 7: A general example for resampling techniques:(a) undersampling, (b) oversampling.

Types of Resampling

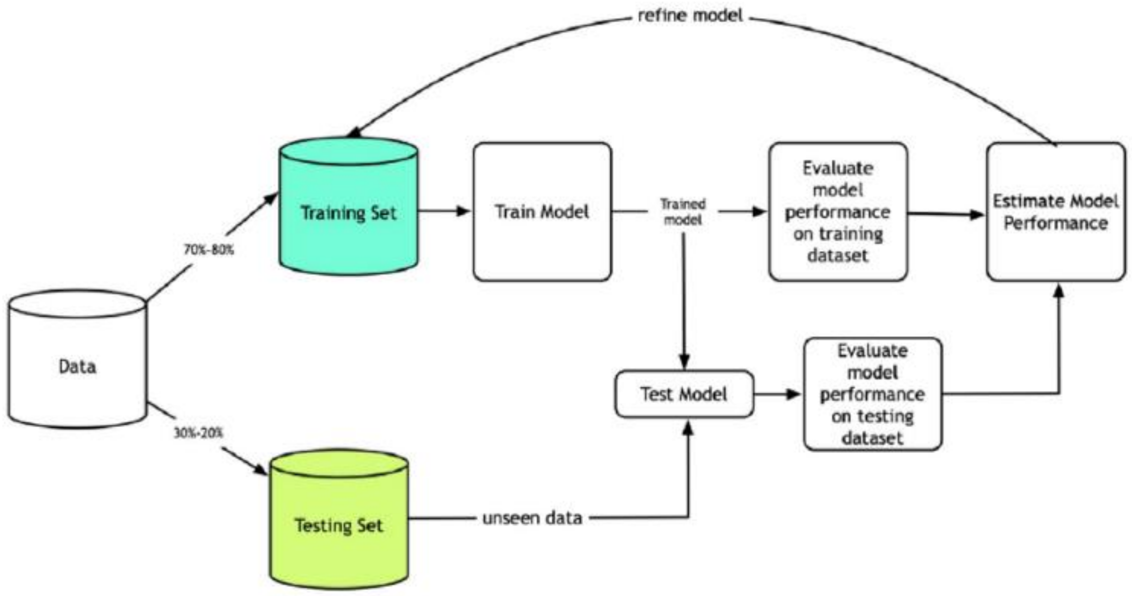
1. Bootstrap
2. Cross validation method

Bootstrap Method

- Bootstrap is a statistical method for estimating the sampling distribution of an estimator by sampling with replacement from the original sample.
- The bootstrap is a flexible and powerful
- It is used to quantify the uncertainty associated with a given estimator or statistical learning method.
- It consists in taking multiples samples out of our original sample and study the resulting distribution.

Validation set Approach

- Here we randomly divide the available set of samples into two parts: a training set and a validation or hold-out set
- The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set.



refine model
Training Set
Train Model
Trained model
Evaluate model performance on training dataset
Estimate Model Performance
70%-80%
Data
30%-20%
Test Model
Evaluate model performance on testing dataset
Testing Set
unseen data

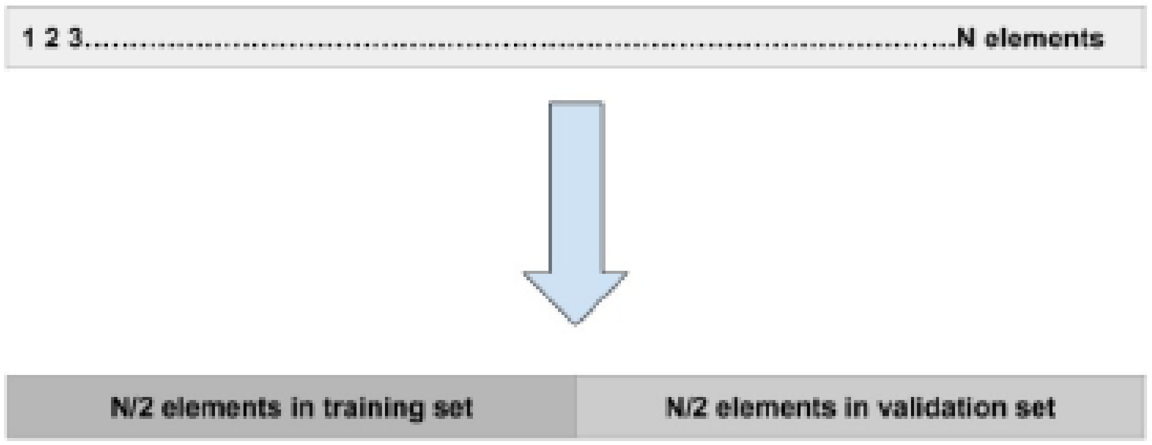
Validation set Approach

The validation set approach is simple. However, it comes with its own set of drawbacks.

- Model learns based on the training data is highly dependent on the observations included in the training set.
- If an outlier observation is included in the training set, the model will tend to learn from outlier observations which may not be relevant in actual data.

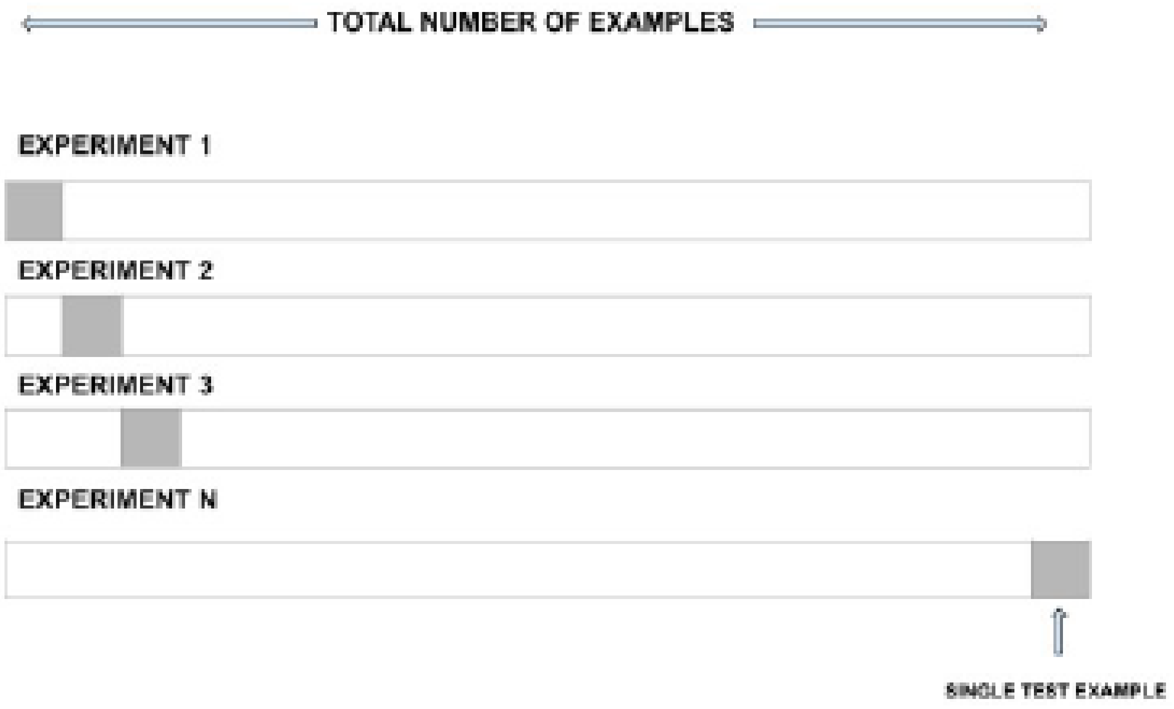
Cross Validation Method

- Widely used approach for estimating test error.
- Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.
- Popular cross validation methods are:
  (i).Leave-one-out-cross-validation (ii). k-fold cross-validation


1 2 3...N elements
N/2 elements in training set
N/2 elements in validation set

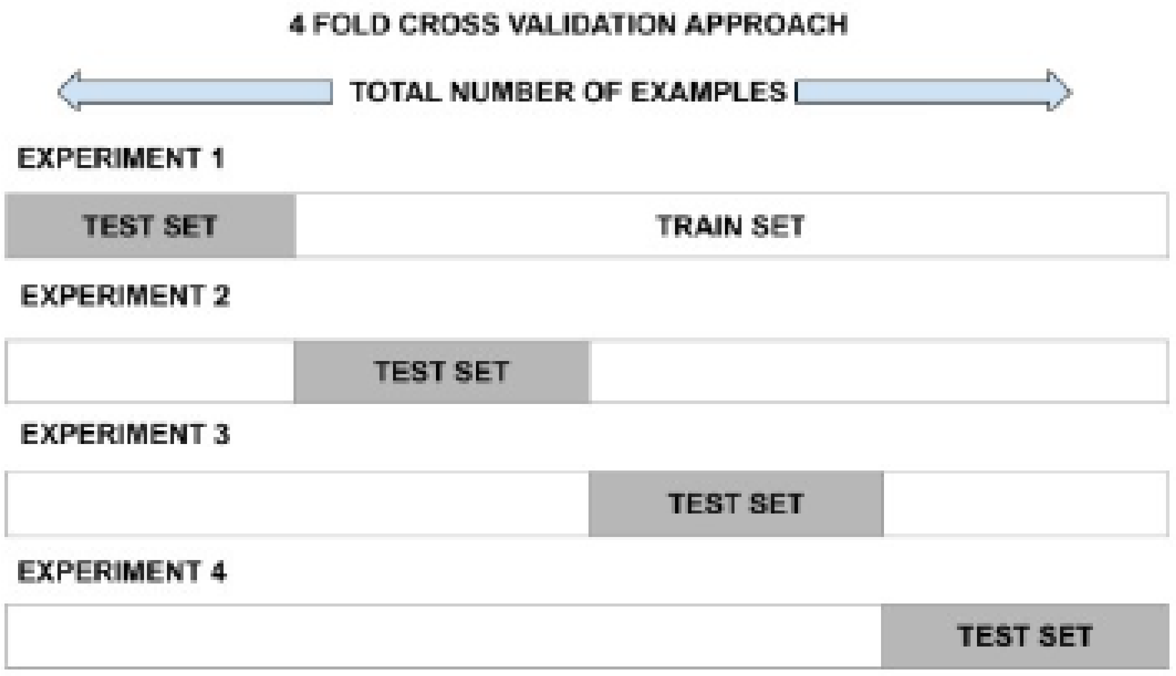
Leave one out cross validation

LOOCV is a better option than the validation set approach. Instead of splitting the entire dataset into two halves only one observation is used for validation and the rest is used to fit the model.



K-fold Cross Validation Method

- This approach involves randomly dividing the set of observations into k folds of nearly equal size.
- Here, the first fold is treated as a validation set and the model is fit on the remaining folds.
- This procedure is repeated till k times, where a different group, each time is treated as the validation set.


4 FOLD CROSS VALIDATION APPROACH
TOTAL NUMBER OF EXAMPLES
EXPERIMENT 1
TEST SET
TRAIN SET
EXPERIMENT 2
TEST SET
EXPERIMENT 3
TEST SET
EXPERIMENT 4
TEST SET