Machine Learning Mastery

Worked Example

We can make the bootstrap procedure concrete with a small worked example. We will work through one iteration of the procedure.

Imagine we have a dataset with 6 observations:

```
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
```

The first step is to choose the size of the sample. Here, we will use 4.

Next, we must randomly choose the first observation from the dataset. Let's choose 0.2.

```
sample = [0.2]
```

This observation is returned to the dataset and we repeat this step 3 more times.

```
sample = [0.2, 0.1, 0.2, 0.6]
```

We now have our data sample. The example purposefully demonstrates that the same value can appear zero, one or more times in the sample. Here the observation 0.2 appears twice.

An estimate can then be calculated on the drawn sample.

```
statistic = calculation([0.2, 0.1, 0.2, 0.6])
```

Those observations not chosen for the sample may be used as out of sample observations.

```
oob = [0.3, 0.4, 0.5]
```

In the case of evaluating a machine learning model, the model is fit on the drawn sample and evaluated on the out-of-bag sample.

```
train = [0.2, 0.1, 0.2, 0.6]
test = [0.3, 0.4, 0.5]
model = fit(train)
statistic = evaluate(model, test)
```

That concludes one repeat of the procedure. It can be repeated 30 or more times to give a sample of calculated statistics.

```
statistics = [...]
```

This sample of statistics can then be summarized by calculating a mean, standard deviation, or other summary values to give a final usable estimate of the statistic.

```
estimate = mean([...])
```

1. Choose a number of bootstrap samples to perform
2. Choose a sample size
3. For each bootstrap sample
    1. Draw a sample with replacement with the chosen size
    2. Fit a model on the data sample
    3. Estimate the skill of the model on the out-of-bag sample.
4. Calculate the mean of the sample of model skill estimates.

```
from sklearn.utils import resample
```

```
data = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
data
```

```
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
```

```
## sample observations
sample = resample(data, replace=True, n_samples=4, random_state=1)
sample
```

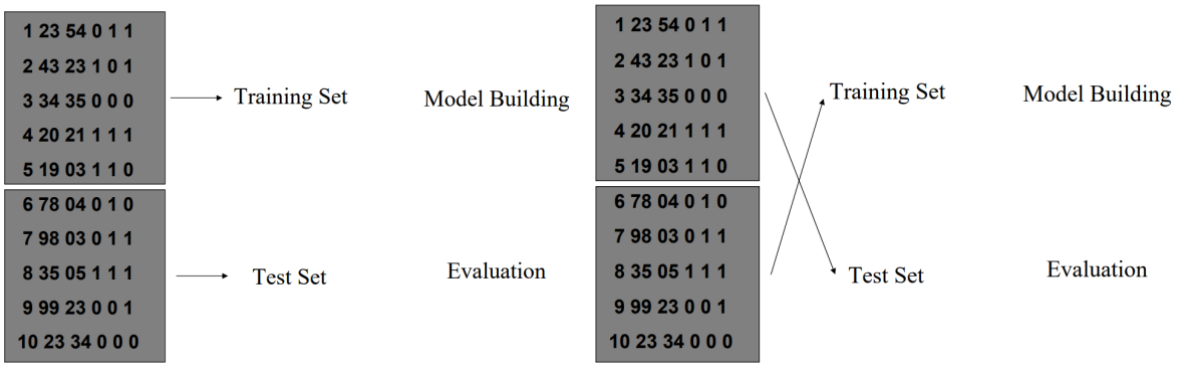
```
[0.6, 0.4, 0.5, 0.1]
```

```
# out of bag observations
oob = [x for x in data if x not in sample]
oob
```

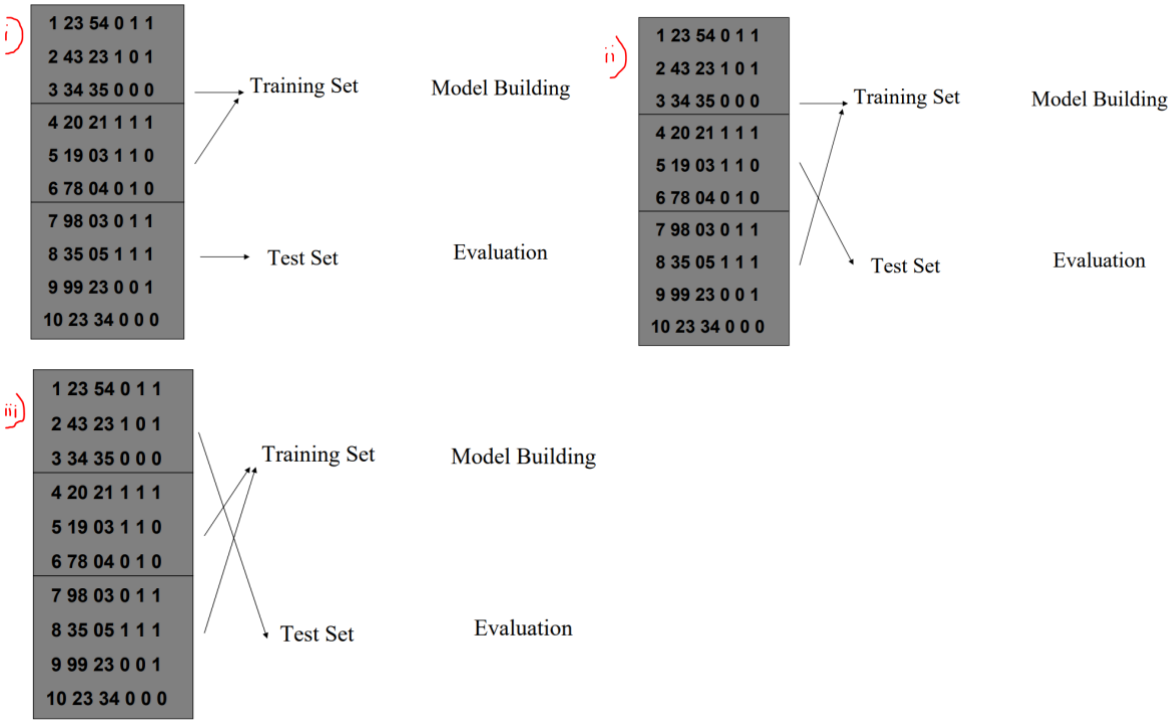
```
[0.2, 0.3]
```

Cross-Validation

Leave N/2 out

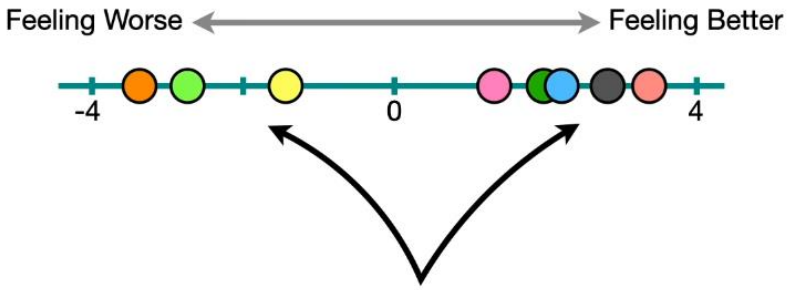


Leave N/3 out


i)
1 23 54 0 1 1
2 43 23 1 0 1
3 34 35 0 0 0
4 20 21 1 1 1
5 19 03 1 1 0
6 78 04 0 1 0
7 98 03 0 1 1
8 35 05 1 1 1
9 99 23 0 0 1
10 23 34 0 0 0
Training Set
Test Set
Model Building
Evaluation
ii)
1 23 54 0 1 1
2 43 23 1 0 1
3 34 35 0 0 0
4 20 21 1 1 1
5 19 03 1 1 0
6 78 04 0 1 0
7 98 03 0 1 1
8 35 05 1 1 1
9 99 23 0 0 1
10 23 34 0 0 0
Training Set
Test Set
Model Building
Evaluation
iii)
1 23 54 0 1 1
2 43 23 1 0 1
3 34 35 0 0 0
4 20 21 1 1 1
5 19 03 1 1 0
6 78 04 0 1 0
7 98 03 0 1 1
8 35 05 1 1 1
9 99 23 0 0 1
10 23 34 0 0 0
Training Set
Test Set
Model Building
Evaluation

Stats-Quest

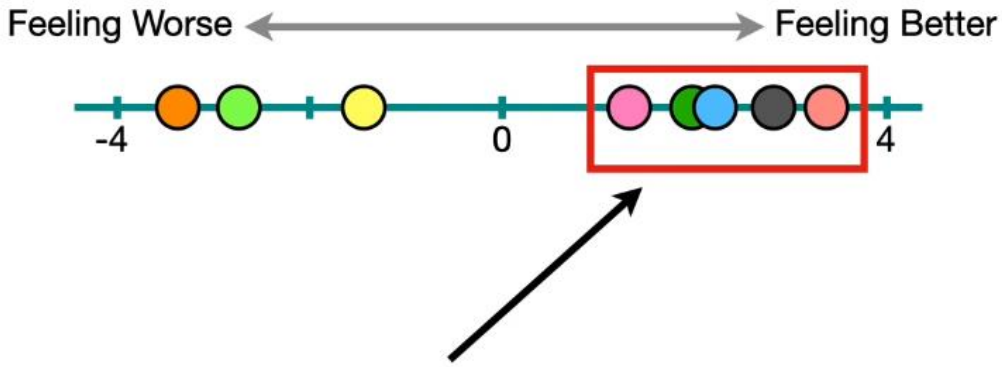
Reason for Bootstrapping



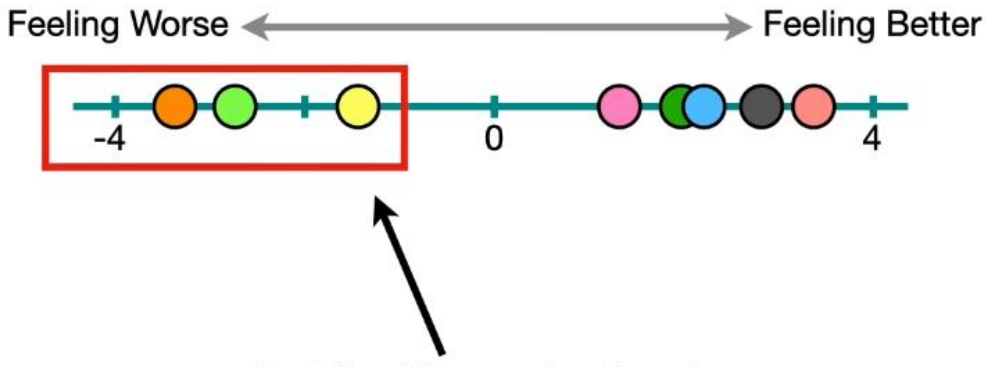
…and we gave that drug to **8** different people that had the illness.

VS

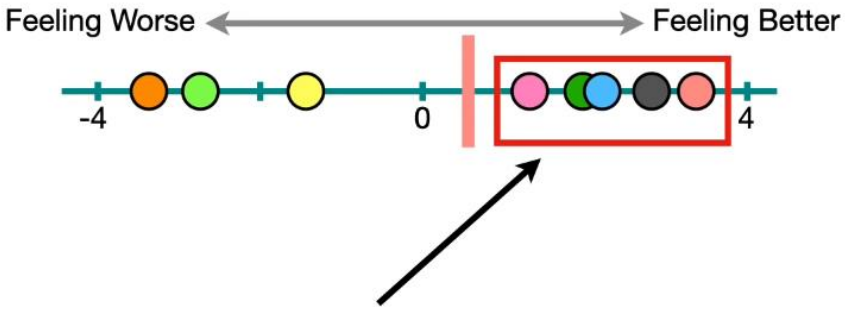
There is a new drug to treat illness.



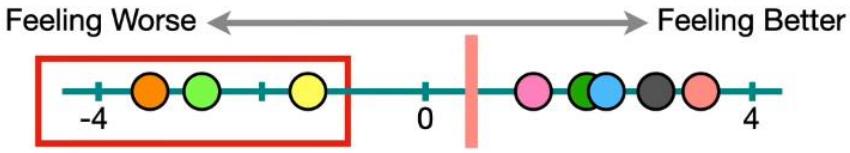
For **5** of those people, the drug appeared to help them feel better…



…but for **3** people, the drug appeared tp make them feel worse.



However, maybe these **5** people all felt better because they were healthier to begin with…



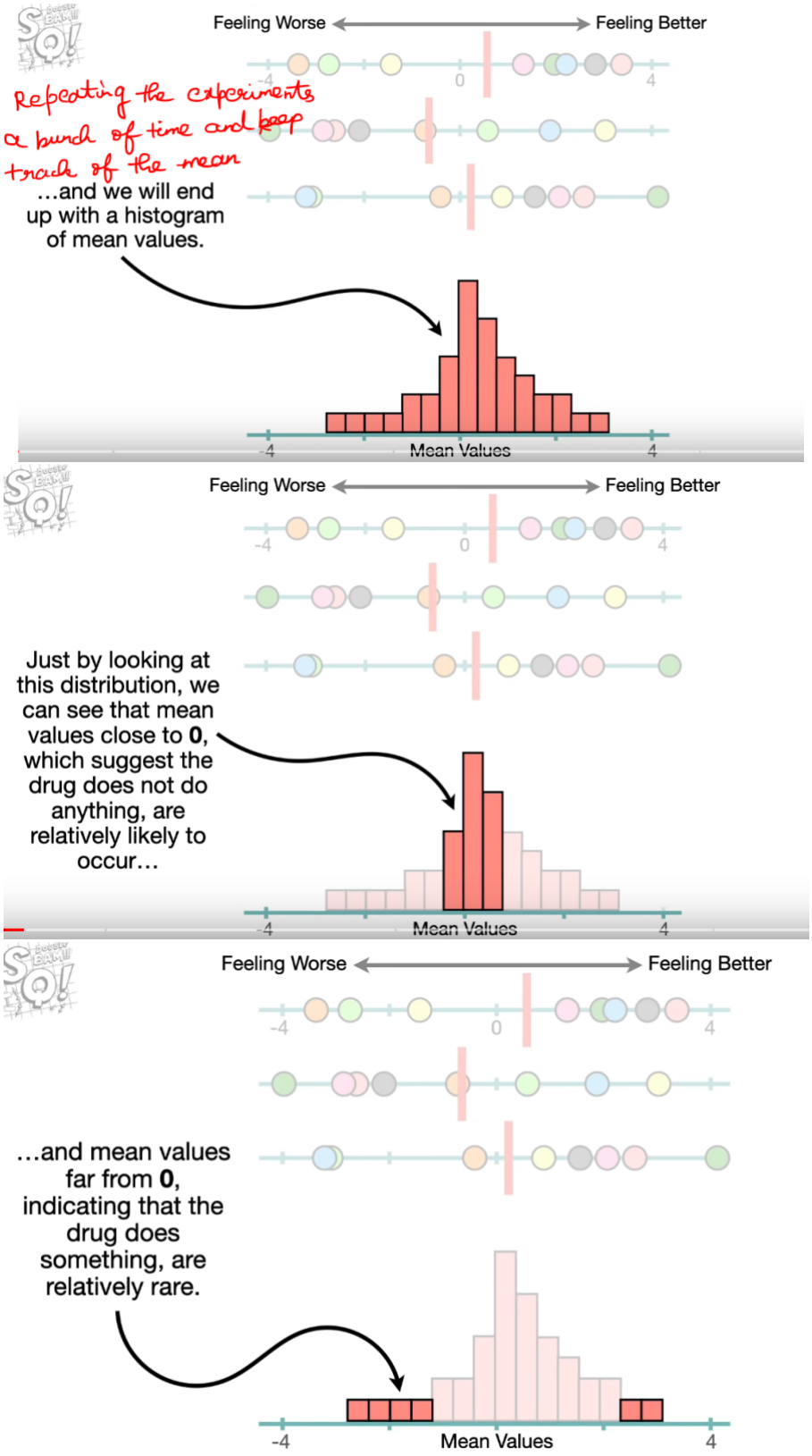
…and maybe these **3** people all felt worse because they had unhealthy lifestyles.

*) So it is possible that the reason we got a mean value = **0.5** instead of **0** is because of random things that we can't control.

*) Is that can we do to decide the drug will work / Not ?

The solution is

One ***expensive*** and ***time consuming*** option would be to replicate the experiment a bunch of times.



Repeating the experiments a bunch of time is expensive and time consuming, one optimal solution to solve this is to use Bootstrapping.

Bootstrap in action

We regularly come across many game shows on television and you must have noticed an option of “Audience Poll”. Most of the time a contestant goes with the option which has the highest vote from the audience and most of the time they win. We can generalize this in real life as well where taking opinions from a majority of people is much more preferred than the opinion of a single person. The Ensemble technique has a similar underlying idea where we aggregate predictions from a group of predictors, which may be classifiers or regressors, and most of the time the prediction is better than the one obtained using a single predictor.

Definition: — Ensemble learning is a machine learning paradigm where multiple models (often called “weak learners”) are trained to solve the same problem and combined to get better results. The main hypothesis is that when weak models are correctly combined, we can obtain more accurate and/or robust models.

Weak Learners: A ‘weak learner’ is any ML algorithm (for regression/classification) that provides an accuracy slightly better than random guessing.

In ensemble learning theory, we call weak learners (or base models) models that can be used as building blocks for designing more complex models by combining several of them. Most of the time, these basics models perform not so well by themselves either because they have a high bias or because they have too much variance to be robust. Then, the idea of ensemble methods is to try reducing bias and/or variance of such weak learners by combining several of them together to create a strong learner (or ensemble model) that achieves better performances.

Let’s suppose we have ’n’ predictors/models:

Z1, Z2, Z3, ……., Zn with a standard deviation of σ

Variance(z) = σ^2

If we use single predictors Z1, Z2, Z3, ……., Zn the variance associated with each will be σ^2 but the expected value will be the average of all the predictors.

Let’s consider the average of the predictors:

$\mu = (Z1 + Z2 + Z3 + \ldots\ldots + Zn)/n$

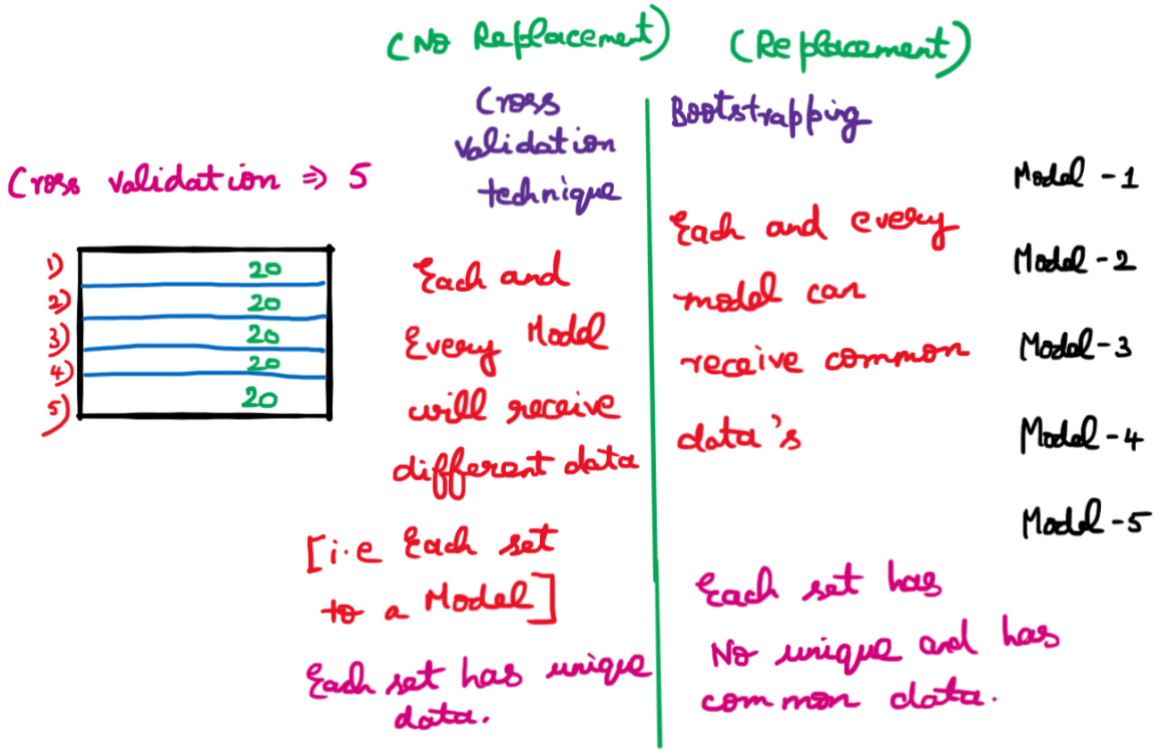
If we use μ as the predictor then the expected value remains the same but see the variance now:

variance(μ) = σ^2/n

So, the expected value remained ‘μ’ but variance decreases when we use an average of all the predictors.

This is why taking mean is preferred over using single predictors.

Difference between cross-validation and bagging

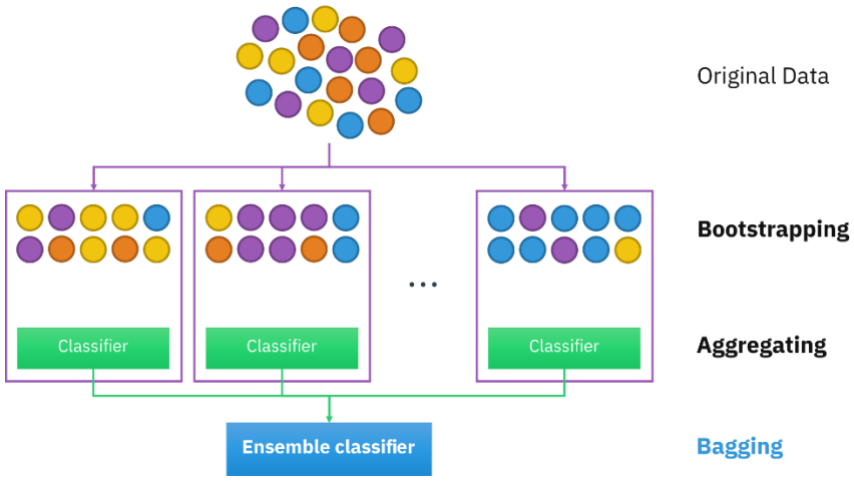


- Brief answer

Bootstrap aggregating (also called Bagging) is a method used to produce a predictive model. Cross-validation is a model validation technique. So cross-validation and bagging solve different problems. You cannot replace cross-validation with bagging as it would not have any sense.

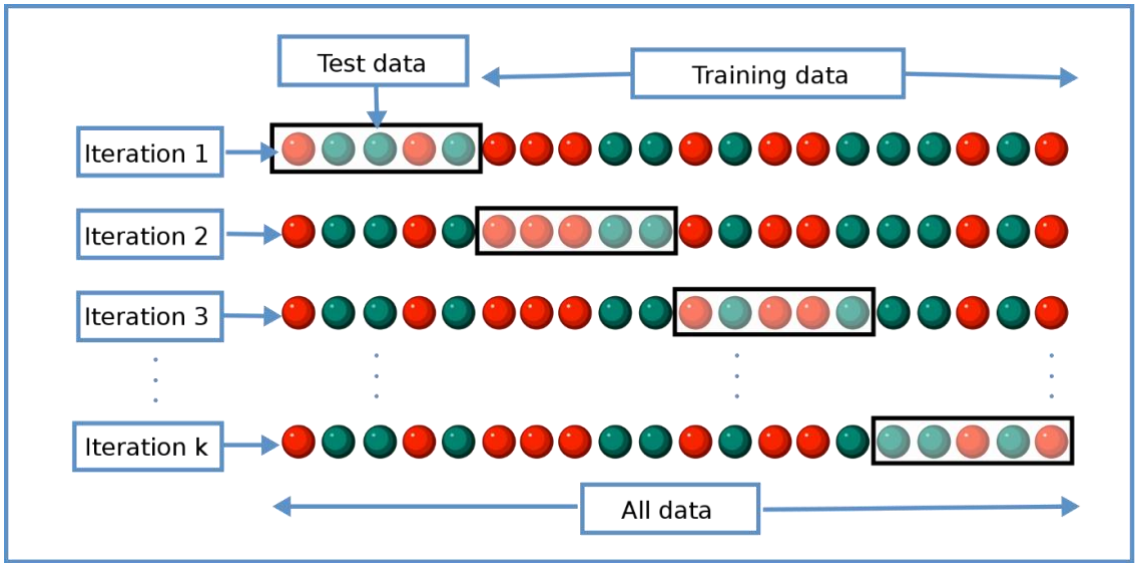
- Explanation

Bagging consists in sampling several datasets from an original training set. Then, a model (generally a classification tree or a regression tree) is trained on each new dataset. Next, the predictions of all the models are aggregated (for example, by taking their mean for a regression task or the majority vote for a classification task) to get a final model.
Below is a schema of how bagging works.



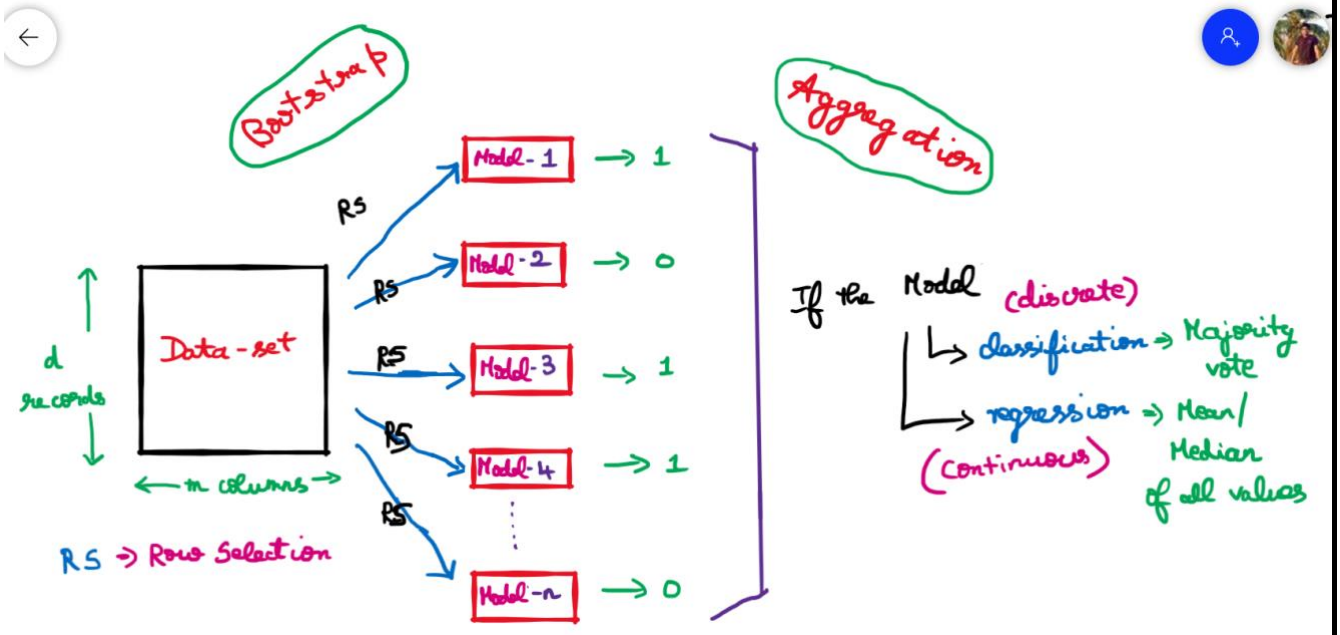
The main advantage of using bagging is indeed to reduce the variance. A model obtained by aggregating several models trained on several sampled dataset is generally less prone to overfitting.

On another side, cross-validation is a method to measure the reliability of a model once it is already created. For example, let's describe how works 5-fold cross-validation.
First, the original dataset is divided into 5 subsets. Next, the model is trained on four of them and a validation performance is computed on the fifth one. Then the operation is repeated by selecting another validation subset among the predefined subsets. At the end of the procedure, we have 5 performance scores, one per validation subset. The mean and standard deviation of the 5 performance scores can be calculated to estimate the bias and variance of the model.
Below is a schema of how k-cross-validation works (in our previous example, k = 5).



Bagging / Bootstrap Aggregation

It is a parallel process.



1. BAGGING / Bootstrap Aggregation

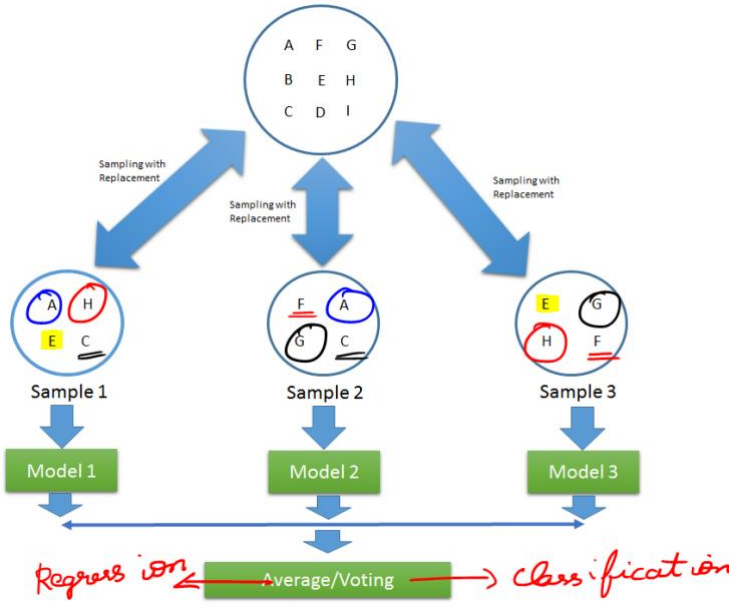
* Bagging stands for Bootstrap Aggregation.

* In real-life scenarios, we don't have multiple different training sets on which we can train our model separately and at the end combine their result. Here, bootstrapping comes into the picture.

* Bootstrapping is a technique of sampling different sets of data from a given training set by using replacement. After bootstrapping the training dataset, we train the model on all the different sets and aggregate the result. This technique is known as Bootstrap Aggregation or Bagging.

* For aggregating the outputs of base learners, bagging uses majority voting (most frequent prediction among all predictions) for classification and averaging (mean of all the predictions) for regression.

Regression ⇒ Averaging Classification ⇒ Majority



(with replacement)

since the model sees repeated data

· Advantages of a Bagging Model:

1. Bagging significantly decreases the variance without increasing bias.

2. Bagging methods work so well because of diversity in the training data since the sampling is done by bootstrapping.

3. Also, if the training set is very huge, it can save computational time by training the model on a relatively smaller data set and still can increase the accuracy of the model.

4. Works well with small datasets as well.

· Disadvantages of a Bagging Model:

1. The main disadvantage of Bagging is that it improves the accuracy of the model at the expense of interpretability i.e., if a single tree was being used as the base model, then it would have a more attractive and easily interpretable diagram, but with the use of bagging this interpretability gets lost.

2. Another disadvantage of Bootstrap Aggregation is that during sampling, we cannot interpret which features are being selected i.e., there are chances that some features are never used, which may result in a loss of important information.

Out of Bag Evaluation: - In bagging, when different samples are collected, no sample contains all the data but a fraction of the original dataset. There might be some data that are never sampled at all. The remaining data which are not sampled are called out of bag instances.

The Random Forest approach is a bagging method where deep trees (Decision Trees), fitted on bootstrap samples, are combined to produce an output with lower variance.

Pasting

Pasting is an ensemble technique similar to bagging with the only difference being that there is no replacement done while sampling the training dataset. This causes less diversity in the sampled datasets and data ends up being correlated. That's why bagging is more preffered than pasting in real scenarios.

Let's see python implementation of Bagging:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.datasets import load_breast_cancer
dataset = load_breast_cancer()
X = dataset.data
y = dataset.target
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=3
)
```

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

```
0.916083916083916
```

Bagging/bootstrap aggregation

```
## number of classifiers ⟶ 10
## % of data each classifier can fit ⟶ 50%
## bootstrap=False (i.e pasting)

bag_knn = BaggingClassifier(KNeighborsClassifier(n_neighbors=5),
                            n_estimators=10, max_samples=0.5,
                            bootstrap=True, random_state=3,oob_score=True)
```

```
bag_knn.fit(X_train, y_train)
bag_knn.score(X_test, y_test)
```

```
0.9370629370629371
```

Great! our score sginificantly improves with use of bagging.

let's not use bootstrap and see the model accuracy! Remember this is "Pasting"

Pasting

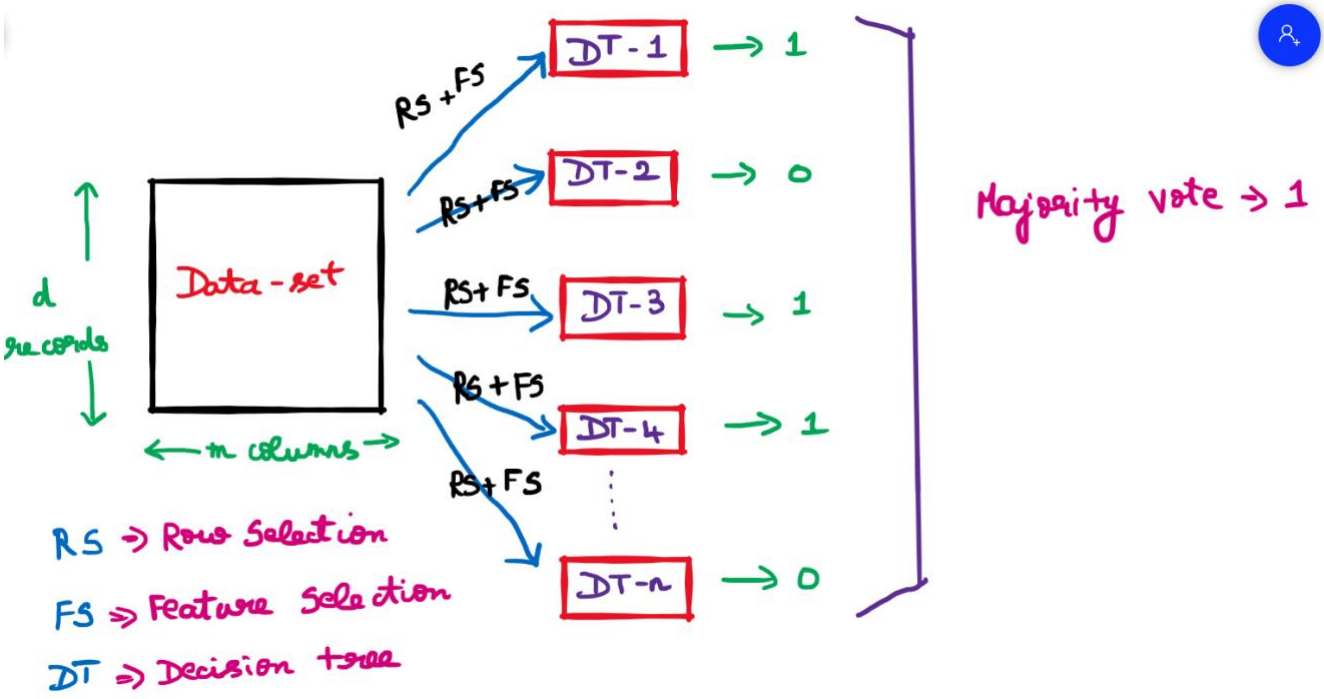
```
pasting_knn = BaggingClassifier(KNeighborsClassifier(n_neighbors=5),
                                n_estimators=10, max_samples=0.5,
                                bootstrap=False, random_state=3)
```

```
pasting_knn.fit(X_train, y_train)
pasting_knn.score(X_test, y_test)
```

```
0.9300699300699301
```

Random Forest

The base learner is decision tree.



Advantages and Disadvantages of Random Forest:

1) It can be used for both regression and classification problems.

2) Since base model is a tree, handling of missing values is easy.

3) It gives very accurate result with very low variance.

4) Results of a random forest are very hard to interpret in comparison with decision trees.

5) High computational time than other respective models.

Random Forest should be used where accuracy is up utmost priority and interpretability is not very important. Also, computational time is less expensive than the desired outcome.

```
data = pd.read_csv("winequality_red.csv")
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

The data set consists following Input variables :

1 - fixed acidity 2 - volatile acidity 3 - citric acid 4 - residual sugar 5 - chlorides
6 - free sulfur dioxide 7 - total sulfur dioxide 8 - density 9 - pH 10 - sulphates 11 - alcohol.

Output variable gives the quality of th wine based on the input variables:

12 - quality (score between 0 and 10)

```
X = data.drop(columns = 'quality')
y = data['quality']
```

```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.30, random_state= 355)
```

```
#let's first visualize the tree on the data without doing any pre processing
clf = DecisionTreeClassifier(criterion='gini',min_samples_split= 2)
clf.fit(x_train,y_train)
```

```
DecisionTreeClassifier()
```

```
# accuracy of our classification-1 tree
clf.score(x_test,y_test)
```

```
0.6291666666666667
```

```
#let's first visualize the tree on the data without doing any pre processing
clf2 = DecisionTreeClassifier(criterion = 'entropy', max_depth =24, min_samples_leaf= 1)
clf2.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=24)
```

```
# accuracy of our classification-2 tree
clf2.score(x_test,y_test)
```

```
0.6208333333333333
```

```
rand_clf = RandomForestClassifier(criterion='gini',n_estimators=100,random_state=6)
```

Random state, if given none then score will vary everytime you run the RandomForestClassifier. If we asssign a value to it, then result will remain constant.

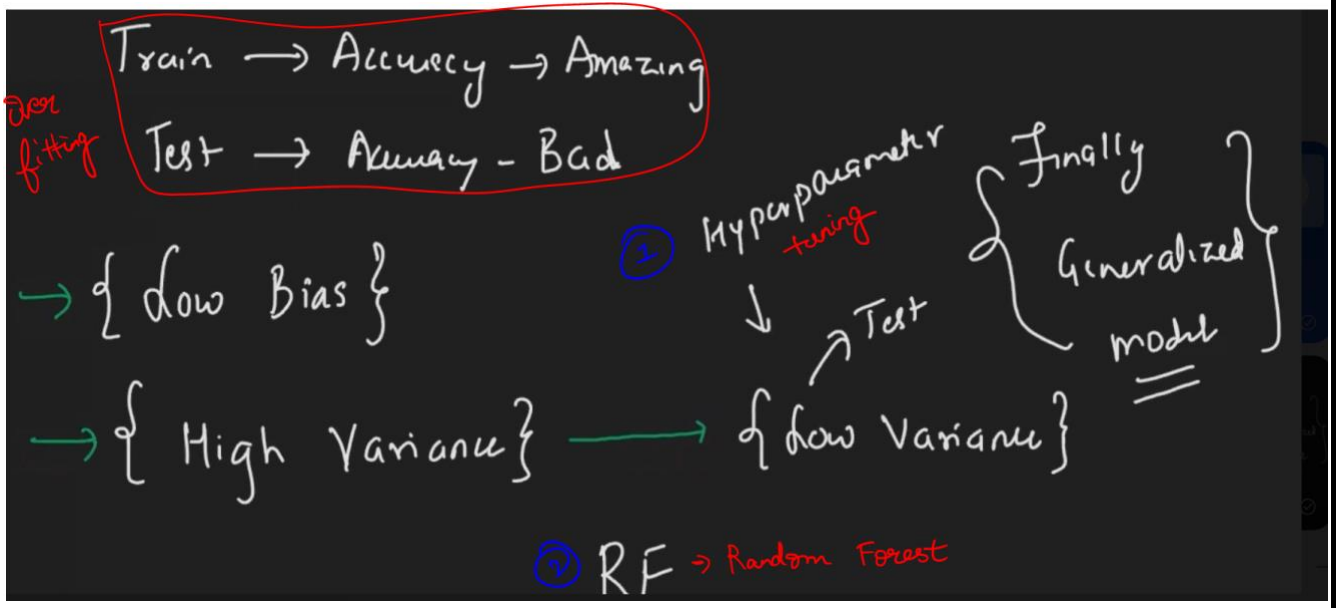
```
rand_clf.fit(x_train,y_train)
```

```
RandomForestClassifier(random_state=6)
```

```
rand_clf.score(x_test,y_test)
```

```
0.6708333333333333
```

Is Random Forest impacted by the outliers ??? No
Is Random Forest requires feature scaling ?? No
Is over-fitting takes place in Random Forest ?? No



Hold-out Method

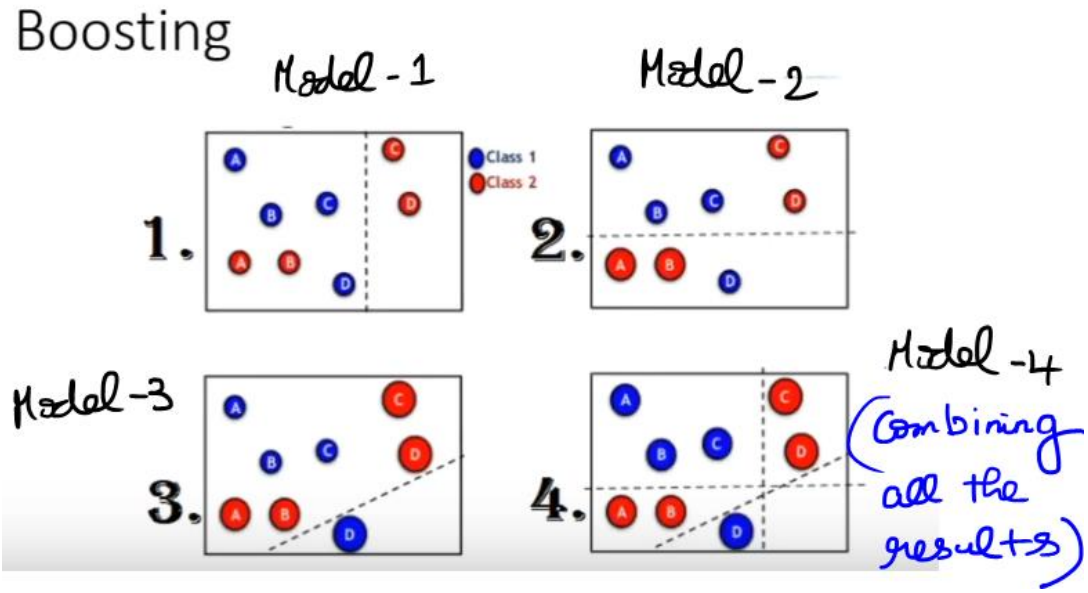
Name is different Normal train-test-split method

- Hold Out Method:

It is the most basic of the CV techniques. It simply divides the dataset into two sets of training and test. The training dataset is used to train the model and then test data is fitted in the trained model to make predictions. We check the accuracy and assess our model on that basis. This method is used as it is computationally less costly. But the evaluation based on the Hold-out set can have a high variance because it depends heavily on which data points end up in the training set and which in test data. The evaluation will be different every time this division changes.

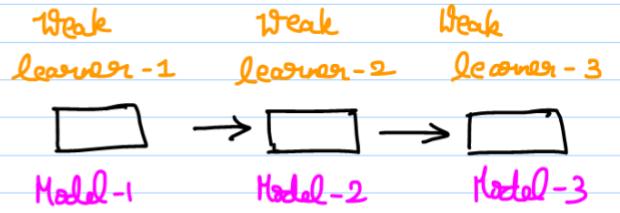
Boosting

It is a sequential process.



Boosting [Combine all the weak learners to Strong learner (sequentially)]

Bagging	Boosting
Parallely	Sequentially



Takes place in Sequential fashion

Ada-boost (Adaptive Boosting)

Steps for Adaboost Algorithm

Step 1 : initialize the weights as 1/n to every n observations

Step 2 : Select the 1 features according to Lowest Gini/Highest information Gain and calculate the Total Error

Step 3 : calculate the Performance of the stump.

Step 4: Calculate the new weights for each misclassification (increase) and right classification (decrease)

Step 5 : Normalize the new weights so that the sum of weights is 1

Step 6: Now Repeat from Step 2 and so on till the configured number of estimators reached or the accuracy achieved.

Step-1 → Pick out the 1st decision stump

Learning An AdaBoost Model From Data

AdaBoost is best used to boost the performance of decision trees on binary classification problems.

AdaBoost was originally called AdaBoost.M1 by the authors of the technique Freund and Schapire. More recently it may be referred to as discrete AdaBoost because it is used for classification rather than regression.

AdaBoost can be used to boost the performance of any machine learning algorithm. It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem.

The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only contain one decision for classification, they are often called decision stumps.

Each instance in the training dataset is weighted. The initial weight is set to:

$$weight(x_i) = 1/n$$

Where xi is the i'th training instance and n is the number of training instances.

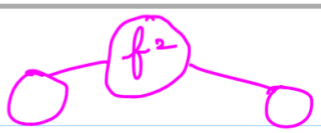
Data - set

	f1	f2	f3	output	Initial weights
1)				Yes	1/7
2)				No	1/7
3)				Yes	1/7
4)				Yes	1/7
5)				No	1/7
6)				No	1/7
7)				Yes	1/7

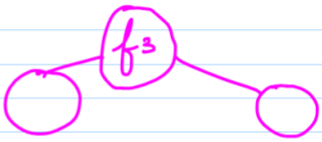
Decision tree is created only with the help of 1-depth (with many leaf Nodes) } Stumps

*) Based on f-1 

⇒ Entropy / Gini Impurity

*) Based on f2 

⇒ Entropy / Gini Impurity

*) Based on f3 

⇒ Entropy / Gini Impurity

} Take the lowest one

*) Now we selected 1st Decision tree as the 1st base learning model.
(lowest entropy)

Step-2 → Calculate the error

A weak classifier (decision stump) is prepared on the training data using the weighted samples. Only binary (two-class) classification problems are supported, so each decision stump makes one decision on one input variable and outputs a +1.0 or -1.0 value for the first or second class value.

The misclassification rate is calculated for the trained model. Traditionally, this is calculated as:

$$error = (correct - N) / N$$

Where error is the misclassification rate, correct are the number of training instance predicted correctly by the model and N is the total number of training instances. For example, if the model predicted 78 of 100 training instances correctly the error or misclassification rate would be (78-100)/100 or 0.22.

Data - set

	f1	f2	f3	output	Initial weights
1)				Yes	1/7
2)				No	1/7
3)				Yes	1/7
4)				Yes	1/7
5)				No	1/7
6)				No	1/7
7)				Yes	1/7

incorrectly classified by the 1st decision stump.

(Based on feature - 1)

Total error ⇒ $\frac{\text{correct} - N}{N}$

correct ⇒ (correctly classified samples

N ⇒ Total Number of samples

correct	Incorrect
6	1

⇒ $\frac{6-7}{7}$ ⇒ 1/7

Total error ⇒ 1/7

Performance of the stump (stage) ⇒ $\frac{1}{2}\log_e\left(\frac{1 - \text{Total error}}{\text{Total error}}\right)$

⇒ $\frac{1}{2}\log_e\left(\frac{1 - 1/7}{1/7}\right)$

⇒ $\frac{1}{2}\log_e\left(\frac{7-1/7}{1/7}\right)$

⇒ $\frac{1}{2}\log_e(6)$

Performance of the stump ⇒ 0.896

Weight will be increasing for incorrectly classified sample.
Weight will be decreasing for correctly classified sample.

Update the weighs

Correctly classified sample → New sample weight ⇒ Old sample weight * $e^{-(\text{Performance of the stump})}$

Incorrectly classified sample → New sample weight ⇒ Old sample weight * $e^{(\text{Performance of the stump})}$

Correctly classified sample ⇒ $1/7 * e^{-0.895}$ ⇒ 0.05

Incorrectly classified sample ⇒ $1/7 * e^{0.895}$ ⇒ 0.349

Step-3 Getting the Normalized weighs

Data - set

	f1	f2	f3	output	Initial weighs	Updated weighs	Normalized weighs
1				Yes	1/7	0.05	0.07
2				No	1/7	0.05	0.07
3				Yes	1/7	0.349	0.513
4				Yes	1/7	0.05	0.07
5				No	1/7	0.05	0.07
6				No	1/7	0.05	0.07
7				Yes	1/7	0.05	0.07
					1	0.68	1

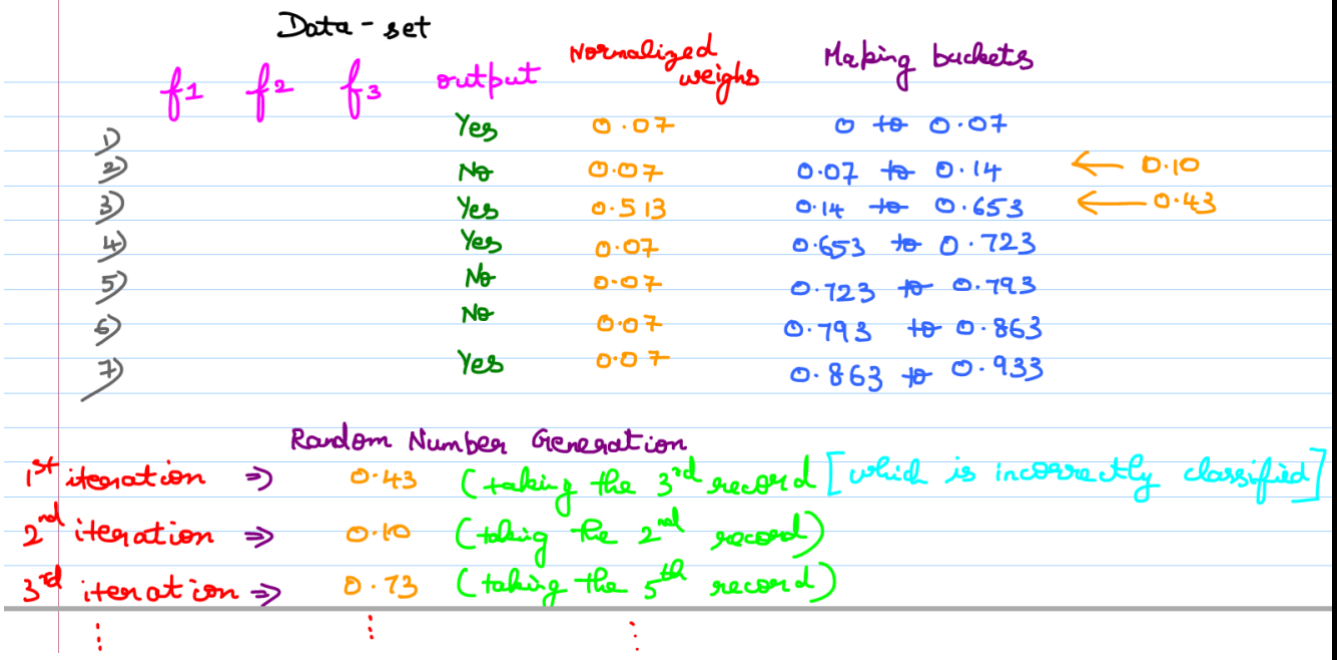
Since the sum of weighs!=1, we are normalizing the weighs.

Normalized weighs ⇒ Updated weighs / 0.68

0.05/0.68 ⇒ 0.07

0.349/0.68 ⇒ 0.513

Step-4 Creating the New data-set



Final Example

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET
1	63	1	233	150	2.3	0	1
2	37	1	250	187	3.5	0	0
3	41	0	204	172	1.4	2	0
4	56	1	236	178	0.8	2	1
5	57	0	354	163	0.6	2	0
6	57	1	192	148	0.4	1	1
7	56	0	294	153	1.3	1	1

⇨

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights
1	63	1	233	150	2.3	0	1	1/7
2	37	1	250	187	3.5	0	0	1/7
3	41	0	204	172	1.4	2	0	1/7
4	56	1	236	178	0.8	2	1	1/7
5	57	0	354	163	0.6	2	0	1/7
6	57	1	192	148	0.4	1	1	1/7
7	56	0	294	153	1.3	1	1	1/7

Initial Weight = $\frac{1}{n}$ to each observations

⇩

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights
1	63	1	233	150	2.3	0	1	0.14
2	37	1	250	187	3.5	0	0	0.14
3	41	0	204	172	1.4	2	0	0.14
4	56	1	236	178	0.8	2	1	0.14
5	57	0	354	163	0.6	2	0	0.14
6	57	1	192	148	0.4	1	1	0.14
7	56	0	294	153	1.3	1	1	0.14

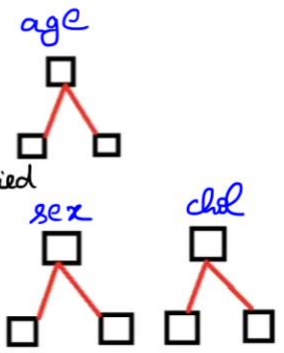
STEP1

Now decision tree ⇒ Based on age
(weak learner)

STEP 2

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights	Predicted Output	Updated Wieghts
1	63	1	233	150	2.3	0	1	0.14	1	
2	37	1	250	187	3.5	0	0	0.14	0	
3	41	0	204	172	1.4	2	0	0.14	1	
4	56	1	236	178	0.8	2	1	0.14	1	
5	57	0	354	163	0.6	2	0	0.14	0	
6	57	1	192	148	0.4	1	1	0.14	0	
7	56	0	294	153	1.3	1	1	0.14	1	

Mis classified



Total Error = 2/7 = 0.28

Step 3 [based on age]

Performance of stump = 1/2 x $\log_e$ (1 - Total Error) / Total Error

Stage-wise multimodality using Multi-class Exponential Loss function

= 1/2 x $\log_e$ (1- 0.28) / 0.28

=1/2 x log (2.5)

= 1/2 x .91 = 0.45

Age ⇒ having lowest Entropy

Correctly classified	Incorrectly classified
5	2

Old weighs

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights	Predicted Output	Updated Wieghts
1	63	1	233	150	2.3	0	1	0.14	1	
2	37	1	250	187	3.5	0	0	0.14	0	
3	41	0	204	172	1.4	2	0	0.14	1	
4	56	1	236	178	0.8	2	1	0.14	1	
5	57	0	354	163	0.6	2	0	0.14	0	
6	57	1	192	148	0.4	1	1	0.14	0	
7	56	0	294	153	1.3	1	1	0.14	1	

Performance of stump = 0.45

Step 4: Updating Weights

New Weights = old Weigths x $e^{\pm Performance}$

+ for Misclassification

- for right classification

New Weights = 0.14 x $e^{+0.45}$ = 0.14 x 1.56 = .21

Probability of choosing this incorrectly classified sample will be more

New Weights = 0.14 x $e^{-0.45}$ = 0.14 x 0.63 = 0.08

Probability of choosing correctly classified sample will be less

For the upcoming decision trees

*) Purposefully with the new weighs formulae

Decreasing the probability (correctly classified samples)

To pay less attention for the further decision tree

Increasing the probability (Incorrectly classified samples)

To pay more attention for the further decision tree

New Weights = $0.14 \times e^{+0.45}$ = 0.14 x 1.56 = .21 For mis classification

New Weights = $0.14 \times e^{-0.45}$ = 0.14 x 0.63 = 0.08 For right classification

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights	Predicted Output	Updated Wieghts
1	63	1	233	150	2.3	0	1	0.14	1	0.08
2	37	1	250	187	3.5	0	0	0.14	0	0.08
3	41	0	204	172	1.4	2	0	0.14	1	0.21
4	56	1	236	178	0.8	2	1	0.14	1	0.08
5	57	0	354	163	0.6	2	0	0.14	0	0.08
6	57	1	192	148	0.4	1	1	0.14	0	0.21
7	56	0	294	153	1.3	1	1	0.14	1	0.08
								1		0.82

Normal weights sum → 1.

Updated weighs sum is not equal to 1.

Step 5 ⇨ Normalization

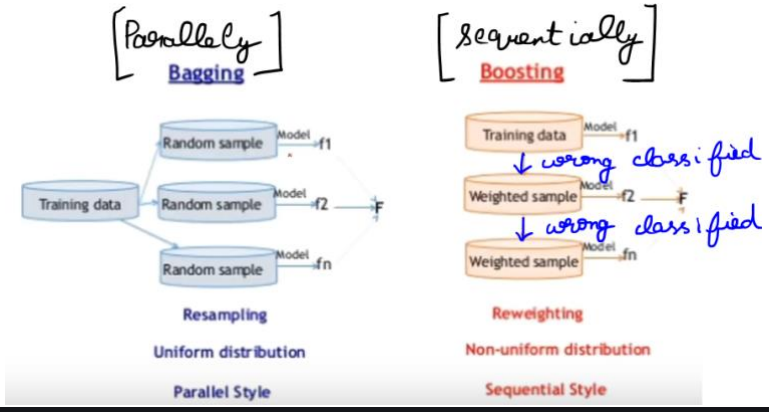
S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights	Predicted Output	Updated Wieghts	Normalized Weights
1	63	1	233	150	2.3	0	1	0.14	1	0.08	0.10
2	37	1	250	187	3.5	0	0	0.14	0	0.08	0.10
3	41	0	204	172	1.4	2	0	0.14	1	0.21	0.26
4	56	1	236	178	0.8	2	1	0.14	1	0.08	0.10
5	57	0	354	163	0.6	2	0	0.14	0	0.08	0.10
6	57	1	192	148	0.4	1	1	0.14	0	0.21	0.26
7	56	0	294	153	1.3	1	1	0.14	1	0.08	0.10
								1		0.82	1.00

Normalized weighs

Correctly classified sample → $\frac{0.08}{0.82}$ → 0.10

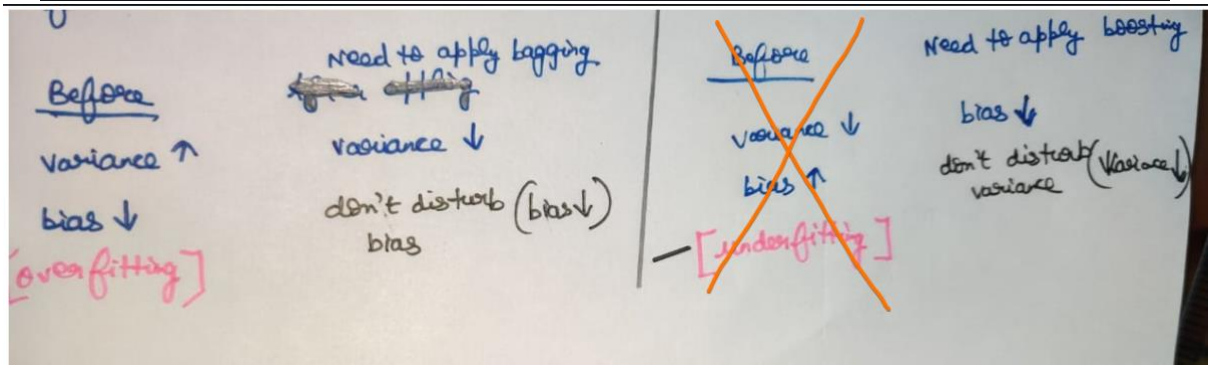
In-correctly classified sample → $\frac{0.21}{0.82}$ → 0.26

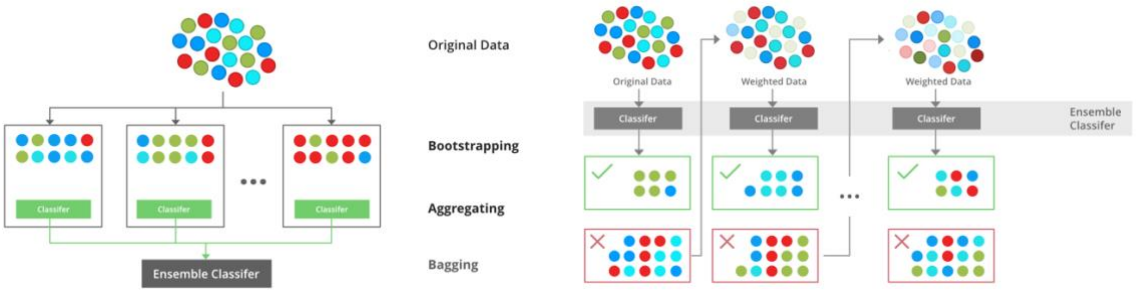
Difference between Boosting and Bagging



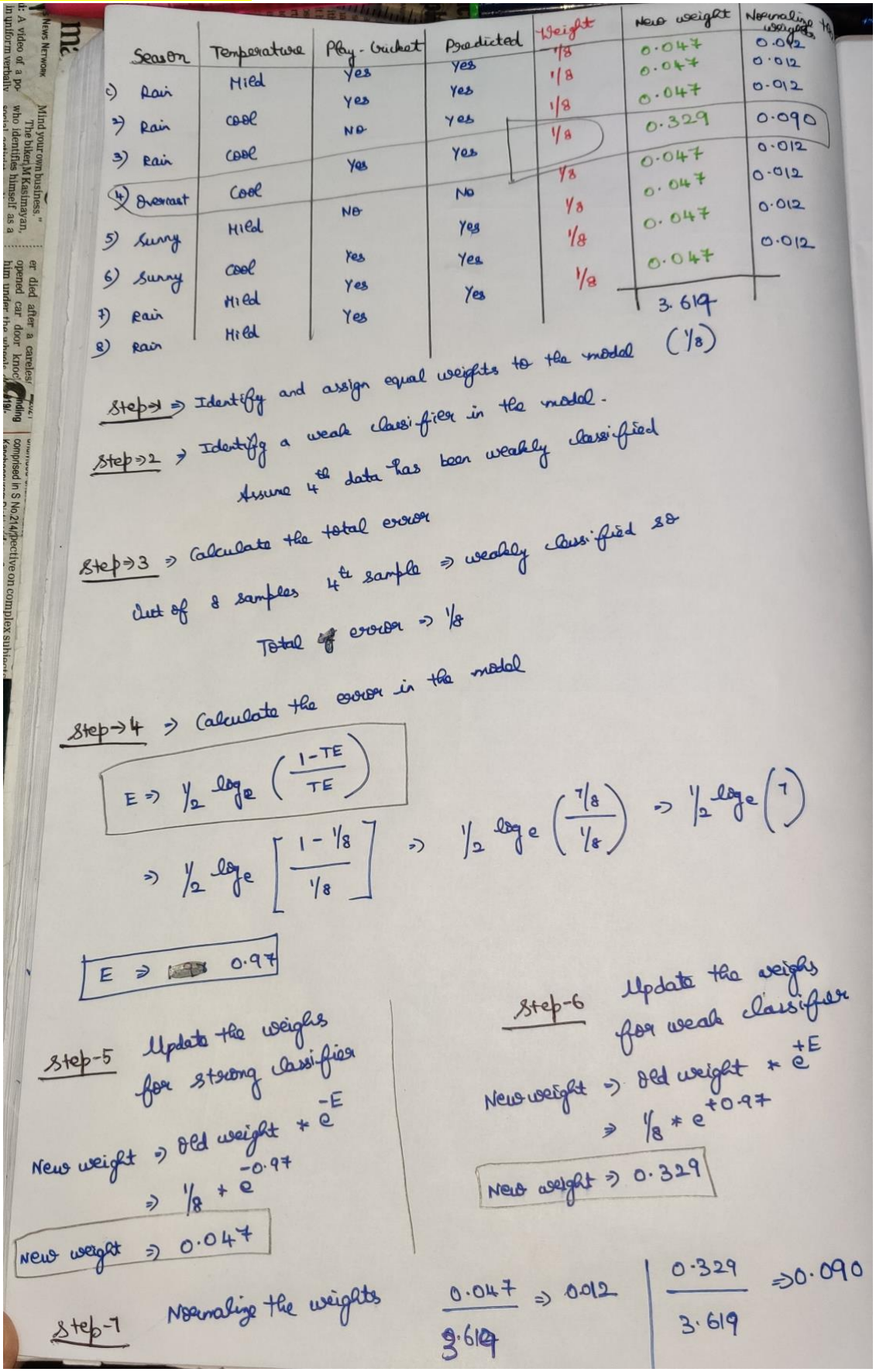
Differences Between Bagging and Boosting

S.NO	Bagging	Boosting
1.	The simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by the performance of previously built models.
5.	Different training data subsets are selected using row sampling with replacement and random sampling methods from the entire training dataset.	Every new subset contains the elements that were misclassified by previous models.
6.	Bagging tries to solve the over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	In this base classifiers are trained parallelly.	In this base classifiers are trained sequentially.
9	Example: The Random forest model uses Bagging.	Example: The AdaBoost uses Boosting techniques




Original Data
Classifer
Classifer
Classifer
Ensemble Classifer
Bootstrapping
Aggregating
Bagging
Original Data
Weighted Data
Weighted Data
Classifer
Classifer
Classifer
Ensemble Classifer

Ada-Boost Sum



	Season	Temperature	Play - Cricket	Predicted	Weight	New weight	Normalizing the weights
1)	Rain	Mild	Yes	Yes	1/8	0.047	0.012
2)	Rain	Cool	Yes	Yes	1/8	0.047	0.012
3)	Rain	Cool	No	Yes	1/8	0.047	0.012
4)	Overcast	Cool	Yes	Yes	1/8	0.329	0.090
5)	Sunny	Mild	No	No	1/8	0.047	0.012
6)	Sunny	Cool	Yes	Yes	1/8	0.047	0.012
7)	Rain	Mild	Yes	Yes	1/8	0.047	0.012
8)	Rain	Mild	Yes	Yes	1/8	0.047	0.012
						3.619	(1/8)

Step-1 ⇒ Identify and assign equal weights to the model

Step-2 ⇒ Identify a weak classifier in the model.

Assume 4th data has been weakly classified

Step-3 ⇒ Calculate the total error

Out of 8 samples 4th sample ⇒ weakly classified so

Total error ⇒ 1/8

Step-4 ⇒ Calculate the error in the model

$$E \Rightarrow \frac{1}{2}\log_e\left(\frac{1-TE}{TE}\right)$$

$$\Rightarrow \frac{1}{2}\log_e\left[\frac{1-1/8}{1/8}\right] \Rightarrow \frac{1}{2}\log_e\left(\frac{7/8}{1/8}\right) \Rightarrow \frac{1}{2}\log_e(7)$$

$$E \Rightarrow 0.97$$

Step-5 Update the weights for strong classifier

New weight ⇒ Old weight * e^{-E}

⇒ $1/8 * e^{-0.97}$

New weight ⇒ 0.047

Step-6 Update the weights for weak classifier

New weight ⇒ Old weight * e^{+E}

⇒ $1/8 * e^{+0.97}$

New weight ⇒ 0.329

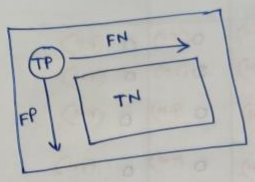
Step-7 Normalize the weights

$\frac{0.047}{3.619} \Rightarrow 0.012$

$\frac{0.329}{3.619} \Rightarrow 0.090$

Calculate the Precision, Recall and Accuracy for below confusion matrix? What is the drawback of Accuracy metric?

	0	1	2	3	4	5	6	7	8	9
0	38	0	0	0	1	0	0	0	0	0
1	0	29	0	0	0	1	0	0	1	0
2	0	0	34	0	0	0	0	0	0	1
3	1	1	0	32	0	0	0	0	0	0
4	0	0	0	0	39	0	0	2	0	0
5	0	1	0	0	0	32	0	0	0	1
6	0	0	0	0	0	0	34	0	0	0
7	0	0	0	0	0	0	0	47	1	0
8	1	1	0	0	0	0	0	0	33	0
9	0	1	0	0	0	0	0	0	1	27



Feature - 0

TP ⇒ 38

FP ⇒ 2

TN ⇒ (33 + 34 + 32 + 39 + 33 + 34 + 49 + 36 + 29) ⇒ 319

FN ⇒ 1

$TPR \Rightarrow \frac{TP}{TP + FN} \Rightarrow \frac{38}{38+1} \Rightarrow 0.97$

$FPR \Rightarrow \frac{FP}{FP + TN} \Rightarrow \frac{2}{2+319} \Rightarrow \frac{2}{321} \Rightarrow 0.00623$

(0.00623, 0.97)

Feature - 5

TP ⇒ 32

FP ⇒ 0

TN ⇒ (34 + 47 + 35 + 27) ⇒ 143

FN ⇒ 1

$TPR \Rightarrow \frac{TP}{TP + FN} \Rightarrow \frac{32}{32+1} \Rightarrow 32/33 \Rightarrow 0.96$

$FPR \Rightarrow \frac{FP}{FP + TN} \Rightarrow \frac{0}{0+143} \Rightarrow 0$

(0, 0.96)

Stacking

References

https://inblog.in/ENSEMBLE-METHODS-Bagging-Boosting-and-Stacking-WkkkvEKBse

https://www.youtube.com/watch?v=Xz0x-8-cgaQ

https://www.analyticsvidhya.com/blog/2020/02/what-is-bootstrap-sampling-in-statistics-and-machine-learning/

https://www.youtube.com/watch?v=KIOeZ5cFZ50

https://www.youtube.com/watch?v=ERBr7jAcuvE

https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/