Ensemble learning

Mithilesh Singh,
Learnbay

When making crucial decisions, everyone should consider enlisting the help of numerous specialists rather than relying on the advice of a single person.

Ensemble learning is the same way. Ensemble learning is the process of combining numerous individual learners to produce a better learner.

Ensemble Learning

Ensemble Learning is a method of reaching a consensus in predictions by fusing the salient properties of two or more models.

For example—

Different models which make incorrect predictions on different sets of samples from the dataset. If two statistically similar models are ensemble the resulting model will only be as good as the contributing models.

Ensemble learning combines the mapping functions learned by different classifiers to generate an aggregated mapping function.

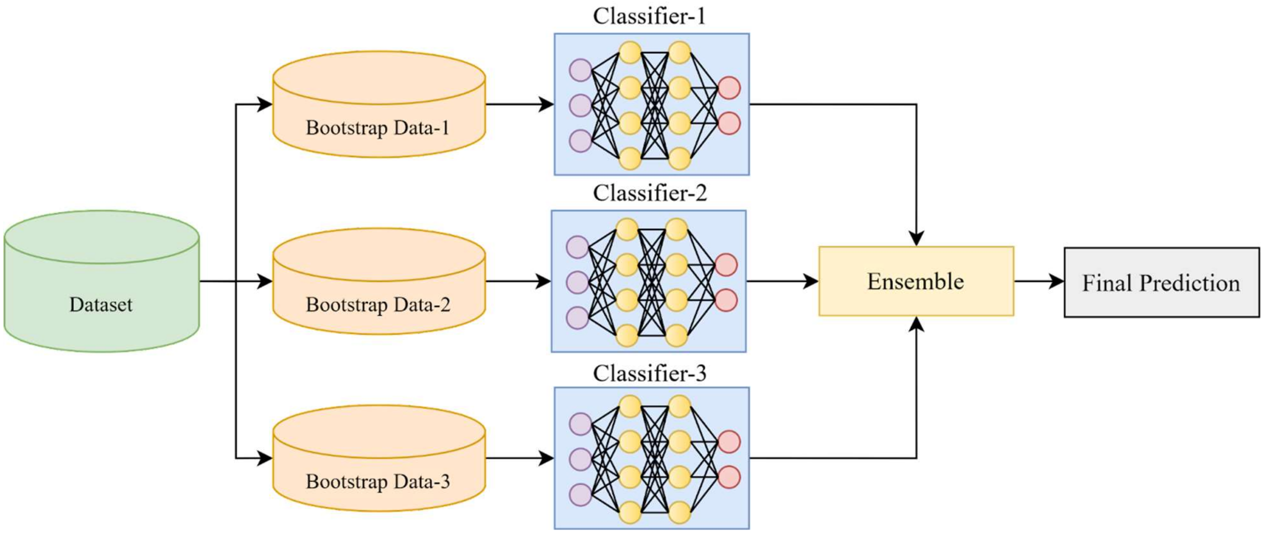
1. Bagging

The Bagging ensemble technique (“bootstrap aggregating”)

subsamples from a dataset are created and they are called “bootstrap sampling.”

Random subsets of a dataset are created using replacement, (meaning that the same data point may be present in several subsets).

These subsets are now treated as independent datasets, on which Machine Learning models will be fit. There is an aggregation mechanism used to compute the final prediction (like averaging, weighted averaging, etc.).



Bagging ensemble mechanism.

Parallel stream of processing occurs. The main aim of the bagging method is to reduce variance in the ensemble predictions.

Popular ensemble methods:

- Bagged Decision Trees
- Random Forest Classifiers
- Extra Trees

While trying to make a better predictive model, we come across a famous ensemble technique, known as Random Forest in Machine Learning. The Random Forest algorithm comes along with a concept of OOB_Score.

Random Forest, is a powerful ensemble technique for machine learning, but most people tend to skip the concept of OOB_Score while learning about the algorithm and hence fail to understand the complete importance of Random forest as an ensemble method.

Though, Decision trees are easy to understand and in interpretations. One major issue with the decision tree is:

1. If a tree is grown to its maximum depth(default setting), then it will capture all the acute details in the training dataset.
2. And applying on testing data gives high error due to High Variance (overfitting of Training data)

Hence, to have the best of both worlds, that is less variance and more interpretability. The algorithm of Random Forest was introduced.

Random Forests or Random Decision Forests are an ensemble learning method for classification and regression problems that operate by constructing a multitude of independent decision trees (using bootstrapping) at training time and outputting majority prediction from all the trees as the final output.

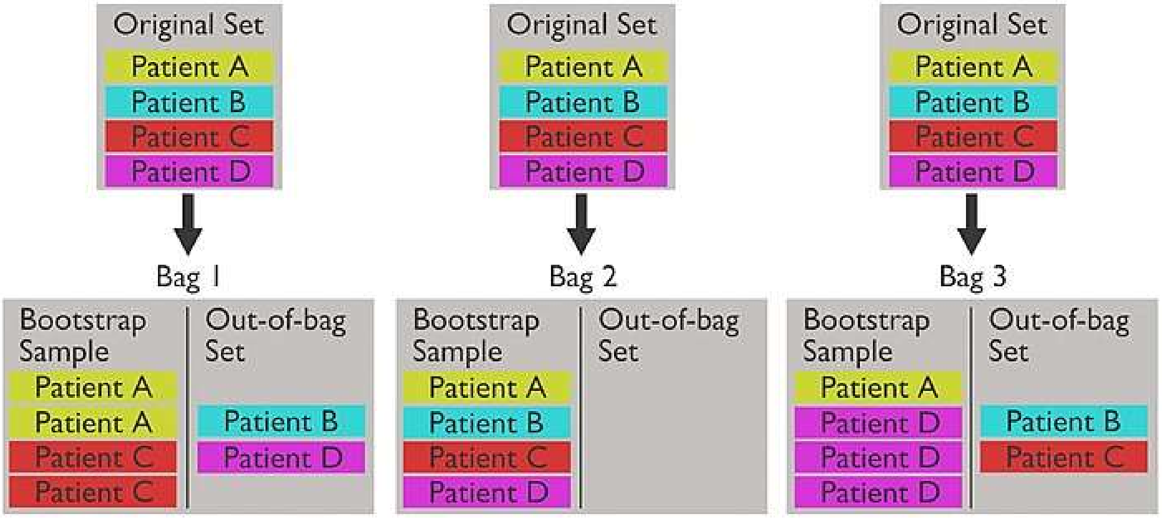
Constructing many decision trees in a Random Forest algorithm helps the model to generalize the data pattern rather than learn the data pattern and therefore, reduce the variance (reduce overfitting).

But, how to select a training set for every new decision tree made in a Random Forest? This is where **Bootstrapping** kicks in!!

3. Bootstrapping and Out-of-Bag Sample

When bootstrap aggregating is performed, two independent sets are created. One set, the bootstrap sample, is the data chosen to be "in-the-bag" by sampling with replacement. The out-of-bag set is all data not chosen in the sampling process.

When this process is repeated, such as when building a random forest, many bootstrap samples and OOB sets are created. The OOB sets can be aggregated into one dataset, but each sample is only considered out-of-bag for the trees that do not include it in their bootstrap sample.



Comparison to cross-validation

Out-of-bag error and cross-validation (CV) are different methods of measuring the error estimate of a machine learning model. Over many iterations, the two methods should produce a very similar error estimate. That is, once the OOB error stabilizes, it will converge to the cross-validation (specifically leave-one-out cross-validation) error The advantage of the OOB method is that it requires less computation and allows one to test the model as it is being trained.

3. Out-of-Bag Score (OOB_Score)

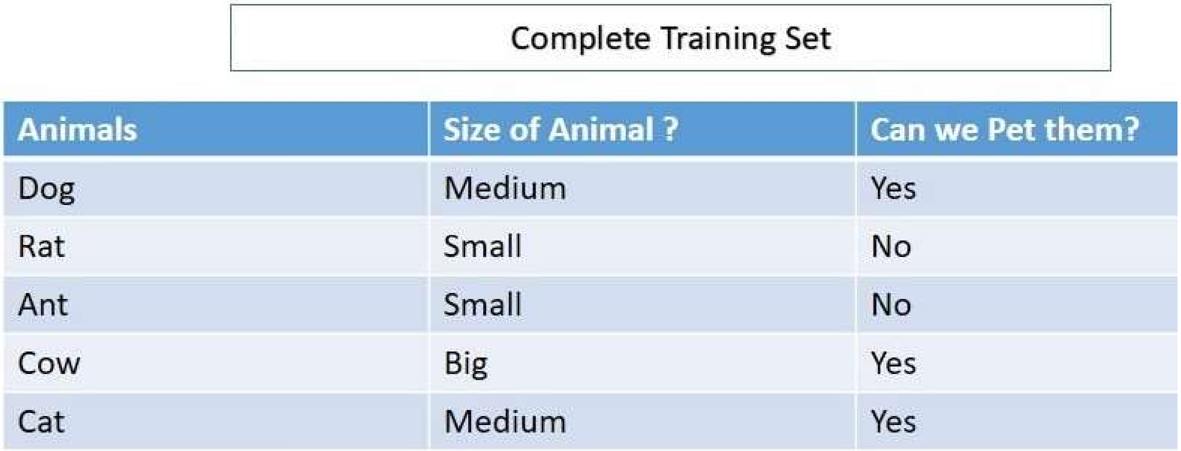
Where does OOB_Score come into the picture?? **OOB_Score is a very powerful Validation Technique used especially for the Random Forest algorithm for least Variance results.**

Note: While using the cross-validation technique, every validation set has already been seen or used in training by a few decision trees and hence there is a leakage of data, therefore more variance.

But, OOB_Score prevents leakage and gives a better model with low variance, so we use OOB_score for validating the model.

Let's understand OOB_Score through an example:

Here, we have a training set with 5 rows and a classification target variable of whether the animals are domestic/pet?



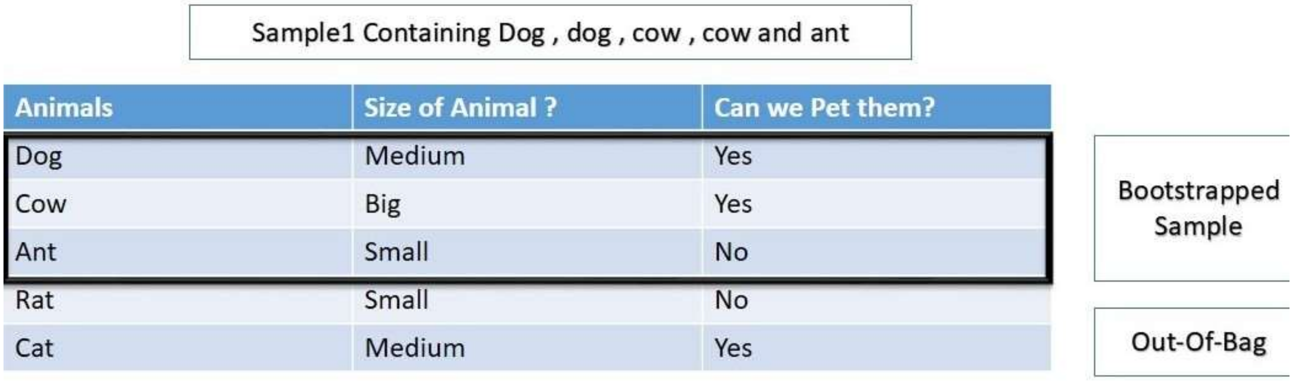
Complete Training Set

Animals	Size of Animal ?	Can we Pet them?
Dog	Medium	Yes
Rat	Small	No
Ant	Small	No
Cow	Big	Yes
Cat	Medium	Yes

Out of multiple decision trees built in the random forest, a bootstrapped sample for one particular decision tree, say DT_1 is shown below

Here, Rat and Cat data have been left out. And since, Rat and Cat are OOB for DT_1, we would predict the values for Rat and Cat using DT_1. (Note: Data of Rat and Cat hasn't been seen by DT_1 while training the tree.)

Just like DT_1, there would be many more decision trees where either rat or cat was left out or maybe both of them were left out.



Sample1 Containing Dog , dog , cow , cow and ant

Animals	Size of Animal ?	Can we Pet them?
Dog	Medium	Yes
Cow	Big	Yes
Ant	Small	No
Rat	Small	No
Cat	Medium	Yes

Bootstrapped Sample

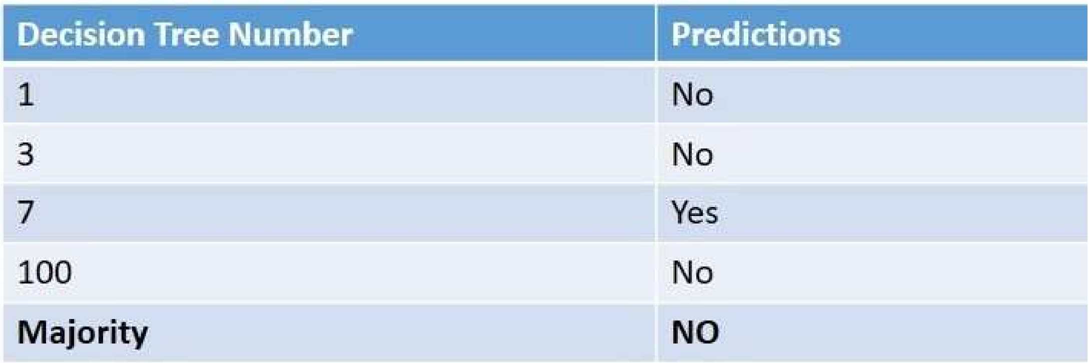
Out-Of-Bag

Say 3rd, 7th, and 100th decision trees also had Rat as an OOB datapoint, which means “Rat” data wasn’t seen by any of them, before predicting the value for Rat.

So, we recorded all the predicted values for “Rat” from the trees DT_1, Dt_3, DT_7, and DT_100.

And saw that aggregated/majority prediction is the same as the actual value for “Rat”.

(**<u>To Note:</u> None of the models had seen data before, and still predicted the values for a data point correctly)**



Decision Tree Number	Predictions
1	No
3	No
7	Yes
100	No
Majority	**NO**

<u>Similarly</u>, every data point is passed for prediction to trees where it would be behaving as OOB and an aggregated prediction is recorded for each row.

OOB Score and OOB Error:-

And

OOB Error is the number of wrongly classifying the OOB Sample.

# Obtain the OOB error

```
# Create a random forest
# classifier and fit it to the data
clf = RandomForestClassifier(n_estimators=100,
                             oob_score=True,
                             random_state=0)
clf.fit(X, y)

# Obtain the OOB error
oob_error = 1 - clf.oob_score_

# Print the OOB error
print(f'OOB error: {oob_error:.3f}')
```

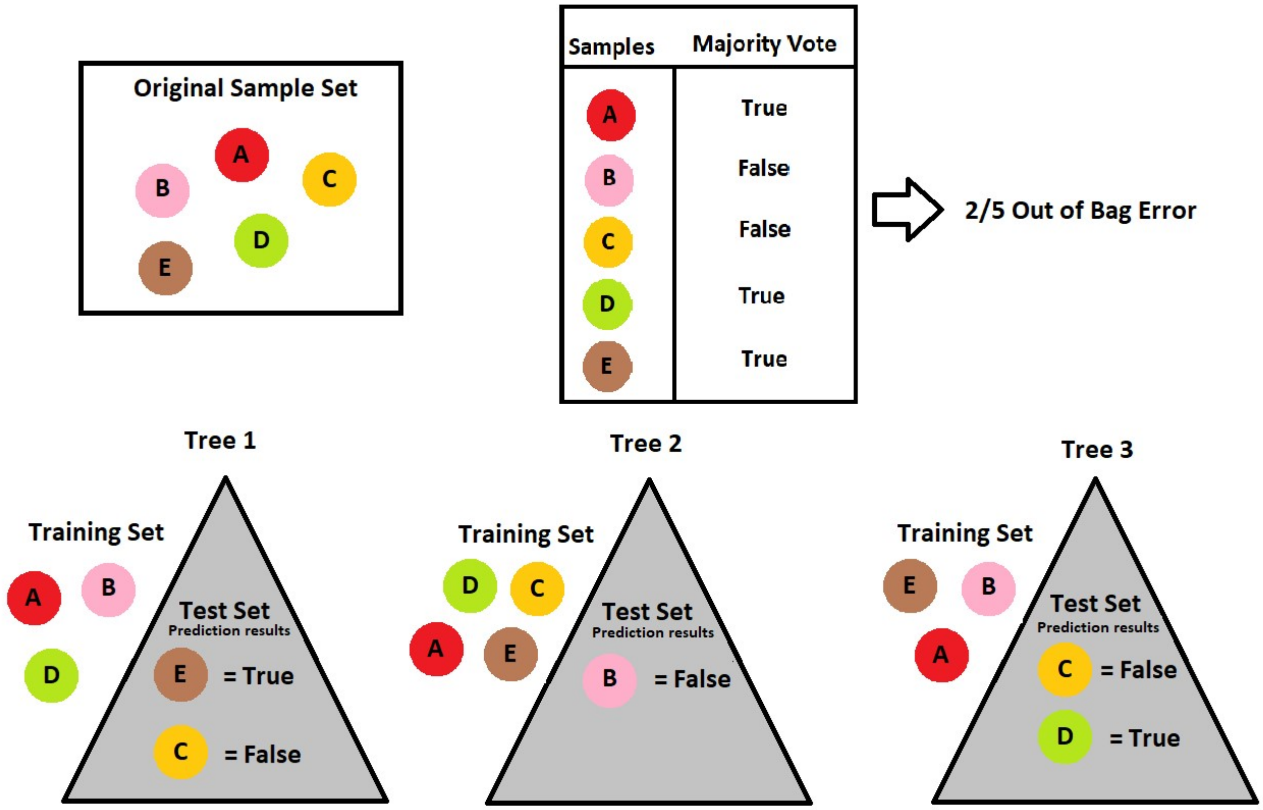
The oob_score_ attribute will only be available if the oob_score parameter was set to True when creating the classifier.

Interpretation of OOB Error: -

Yes, out-of-bag error is an estimate of the error rate (which is 1 - accuracy) that this training approach has for new data from the same distribution. This estimate is based on the predictions that you get for each data point by using only averaging those trees, for which the record was not in the training data.

If you have a low number of trees the OOB error might be not a good estimate of the error rate that training an algorithm like this has on new data, because each tree in RF tends to be underfit and only once you combine enough trees the RF gets better (so if there's only a small number of trees per record, it may underestimate performance). On the other hand, once you have a huge number of trees it becomes a pretty good estimate like you get from a train-validation split with a lot of data (or cross-validation).

What makes it difficult to interpret in your particular case is that the OOB error is estimated on the training data distribution. When you do SMOTE, you change the training data distribution vs. the true distribution, which affects accuracy (which is of course a performance measure that is affected by class prevalence). So, I would not compare OOB error between the two scenarios you have for that reason.



Samples	Majority Vote
A	True
B	False
C	False
D	True
E	True

4. Advantages of using OOB_Score:

1. **No leakage of data:** Since the model is validated on the OOB Sample, which means data hasn't been used while training the model in any way, so there isn't any leakage of data and henceforth ensures a better predictive model.
2. **Less Variance** : **[**More Variance ~ Overfitting due to more training score and less testing score]. Since OOB_Score ensures no leakage, so there is no over-fitting of the data and hence least variance.
3. **Better Predictive Model:** OOB_Score helps in the least variance and hence it makes a much better predictive model than a model using other validation techniques.
4. **Less Computation:** It requires less computation as it allows one to test the data as it is being trained.

Disadvantages of using OOB_Error :

1. **Time Consuming:** The method allows to test the data as it is being trained, but the overall process is a bit time-consuming as compared to other validation techniques.
2. **Not good for Large Datasets:** As the process can be a bit time-consuming in comparison with the other techniques, so if the data size is huge, it may take a lot more time while training the model.
3. **Best for Small and medium-size datasets:** Even if the process is time-consuming, but if the dataset is medium or small sized, OOB_Score should be preferred over other techniques for a much better predictive model.

What are some of the use cases of the OOB error?

- One of the main use cases of the OOB error is to evaluate the performance of an ensemble model such as a random forest. Because the OOB error is calculated using out-of-bag samples, which are samples that are not used in the training of the model, it provides an unbiased estimate of the model’s performance.

- Another use case of the OOB error is to tune the hyper parameters of a model. By using the OOB error as a performance metric, the hyper parameters of the model can be adjusted to improve its performance on unseen data.

- Additionally, the OOB error can be used to diagnose whether a model is overfitting or under fitting. If the OOB error is significantly higher than the validation score, it may indicate that the model is overfitting and not generalizing well to unseen data. On the other hand, if the OOB error is significantly lower than the validation score, it may indicate that the model is underfitting and not learning the underlying patterns in the data.

- Overall, the OOB error is a useful tool for evaluating the performance of an ensemble model and for diagnosing issues such as overfitting and underfitting.

Final Note:- Random Forest can be a very powerful technique for predicting better values if we use the OOB_Score technique. Even if OOB_Score takes a bit more time but the predictions are worth the time consumed in training the random forest model with the OOB_Score parameter set as True.

2. Boosting

The boosting ensemble mechanism works different from the bagging mechanism.

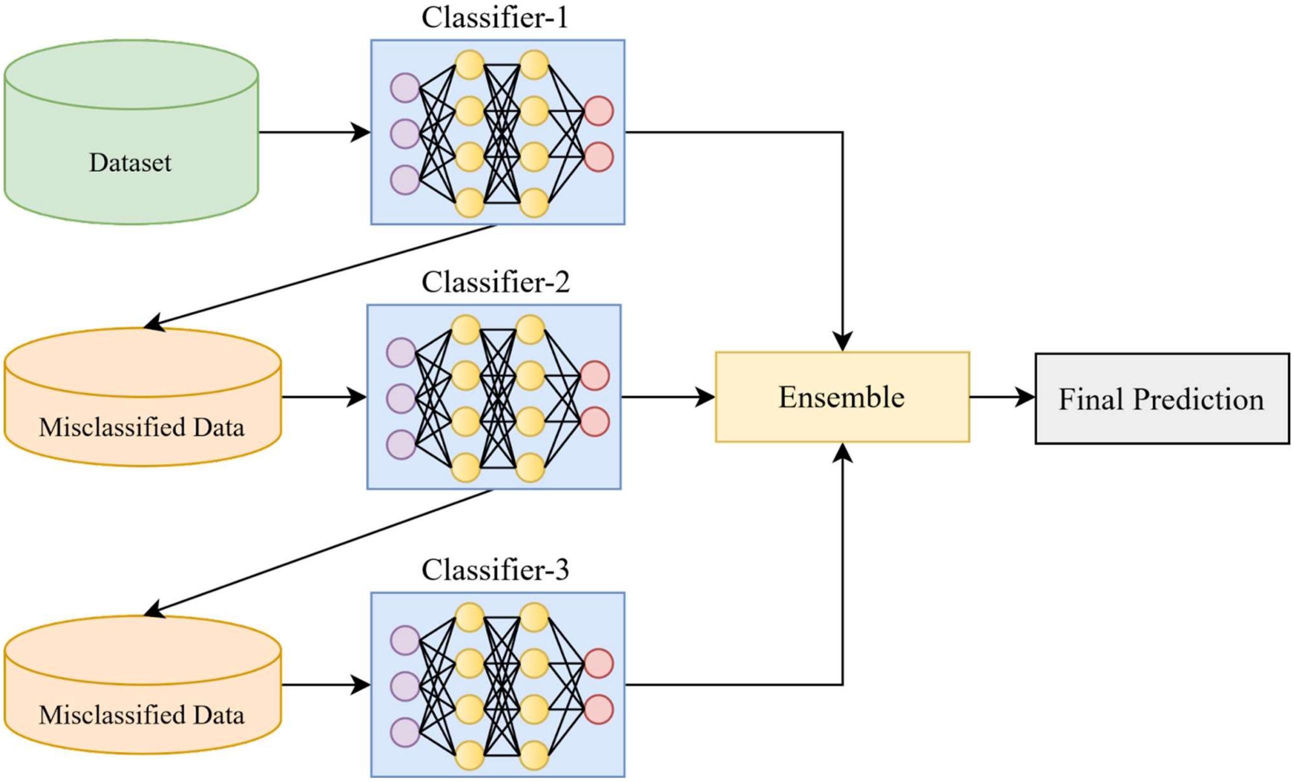
Here, instead of parallel processing of data, sequential processing of the dataset occurs.

The first classifier is fed with the entire dataset, and the predictions are analyzed.

The instances where Classifier-1 fails to produce correct predictions (that are samples near the decision boundary of the feature space) are fed to the second classifier.

This is done so that Classifier-2 can specifically focus on the problematic areas of feature space and learn an appropriate decision boundary.

Similarly, further steps of the same idea are employed, and then the ensemble of all these previous classifiers is computed to make the final prediction on the test data.

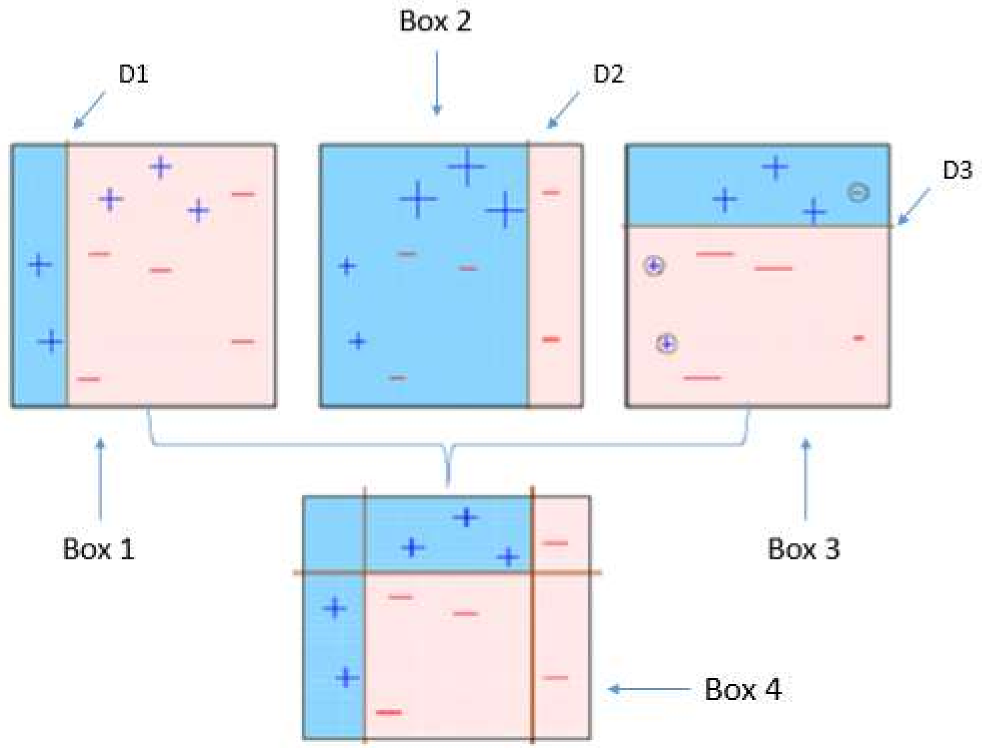


Boosting ensemble mechanism

The main aim of the boosting method is to reduce bias in the ensemble decision. Other algorithms based on this approach include:

- Adaptive Boosting
- Stochastic Gradient Boosting
- Gradient Boosting Machines

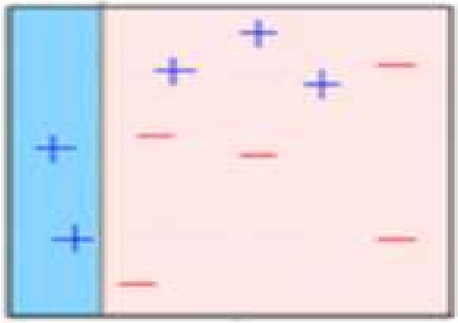
Boosting Algorithm: AdaBoost



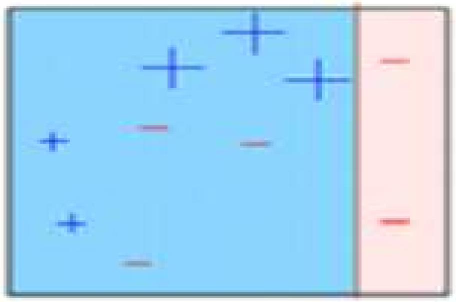
This diagram aptly explains Ada-boost. Let’s understand it closely:

<u>Box 1:</u> assigned equal weights to each data point and applied a decision stump to classify them as + (plus) or – (minus). The decision stump (D1) has generated vertical line at left side to classify the data points.

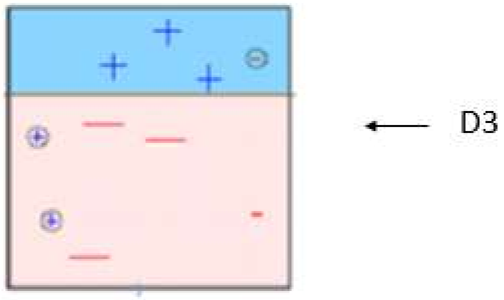
This vertical line has incorrectly predicted three + (plus) as – (minus). In such case, we’ll assign higher weights to these three + (plus) and apply another decision stump.



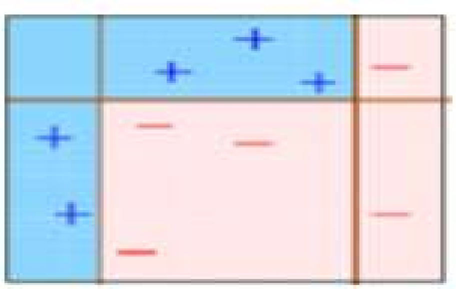
Box 2: Here, we can see that the size of three incorrectly predicted + (plus) is bigger as compared to rest of the data points. In this case, the second decision stump (D2) will try to predict them correctly. Now, a vertical line (D2) at right side of this box has classified three mis-classified + (plus) correctly. But again, it has caused mis-classification errors. This time with three -(minus). Again, we will assign higher weight to three – (minus) and apply another decision stump.



Box 3: Here, three – (minus) are given higher weights. A decision stump (D3) is applied to predict these mis-classified observation correctly. This time a horizontal line is generated to classify + (plus) and – (minus) based on higher weight of mis-classified observation.



Box 4: Here, we have combined D1, D2 and D3 to form a strong prediction having complex rule as compared to individual weak learner. You can see that this algorithm has classified these observation quite well as compared to any of individual weak learner.



AdaBoost (Adaptive Boosting) : It works on similar method as discussed above. It fits a sequence of weak learners on different weighted training data. It starts by predicting original data set and gives equal weight to each observation. If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly. Being an iterative process, it continues to add learner(s) until a limit is reached in the number of models or accuracy.

Mostly, we use decision stamps with AdaBoost.

We can use AdaBoost algorithms for both classification and regression problem.

Boosting Algorithm: Gradient Boosting

In gradient boosting, it trains many model sequentially. Each new model gradually minimizes the loss function (y = ax + b + e, e needs special attention as it is an error term) of the whole system using **Gradient Descent** method. The learning procedure consecutively fit new models to provide a more accurate estimate of the response variable.

The principle idea behind this algorithm is to construct new base learners which can be maximally correlated with negative gradient of the loss function, associated with the whole ensemble.

3. Majority Voting

Majority voting is one of the earliest and easiest ensemble schemes

In this method, an odd number of contributing classifiers are chosen, and for each sample, the predictions from the classifiers are computed. the class that gets most of the class from the classifier pool is deemed the ensemble's predicted class.

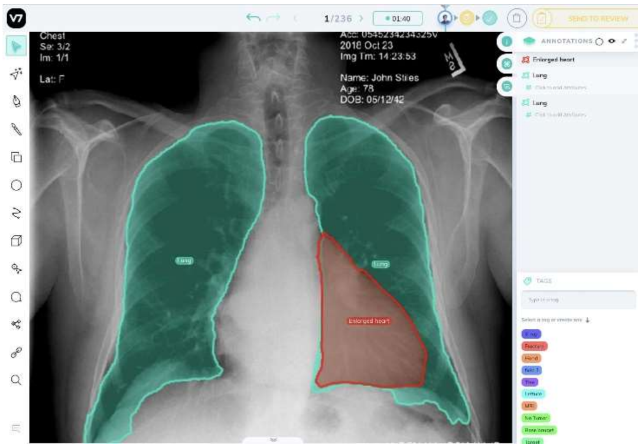
Such a method works well for binary classification problems, where there are only two candidates for which the classifiers can vote.

However, it fails for a problem with many classes since many cases arise, where no class gets a clear majority of the votes.

Applications of Ensemble Learning

1. Disease detection

cardiovascular disease detection from X-Ray and CT scans.



2. Remote Sensing

Tasks like Landslide Detection and Scene Classification

3. Fraud Detection

4. Speech emotion recognition