### 12.3.1 Stacking

Stacking is a meta modelling technique introduced by Wolpert in the year 1992. Stacking model includes two types of learners called base-learners and a meta-learner. It is a process of ensembling multiple heterogeneous machine learning models or base learners to make predictions and then use these predictions as features to train the second level meta learner model whose prediction is considered as the final prediction. Base-learners use normal machine learning algorithms like Random Forests, SVM, Perceptron, etc., whereas the meta learner fits on the predictions of the base learner. The meta-learner can be meta-regressor or meta-classifier, which collects predictions from each base learner to estimate the final predictions, thereby improving the overall performance of the model. Stacking is making the second level of learner to estimate the biases and correct it. The stacking technique can be further extended to more than two levels which is called as multi-level stacking.

### 12.3.2 Cascading

A cascading ensemble model is a multistage learning technique with a sequence of classifiers ordered in terms of increasing complexity and specificity such that early classifiers are simple and general whereas later ones are more complex and specific. It moves to the next stage of learning only if the preceding learner's prediction is not confident. In cascaded models, different classifiers are coupled with their input/output function in a cascade, improving performance at each level. In a typical cascading ensemble model, the complexity of the model increases as more models are added to the cascade.

## 12.4 SEQUENTIAL ENSEMBLE MODELS

Boosting algorithms such as AdaBoost, Gradient Boosting and XGBoost are categorized as Sequential Ensemble models. The general principle of this model is to use multiple weak learners sequentially and the error exhibited by a weak learner is used to give importance to data instances which are further retrained by the next weak learner. Thus, the accuracy of prediction is improved in the ensemble model.

### 12.4.1 AdaBoost

AdaBoost, which is otherwise called as Adaptive Boosting, is a popular boosting algorithm and it is a non-linear classifier. It is a sequential ensemble model that trains multiple weak classifiers sequentially. It would train one weak classifier at a time, so that the errors made by poorly performing weak classifiers are reduced when they are trained sequentially.

AdaBoost is a greedy algorithm optimizing weights of data instances in the training dataset by adding a weak classifier at each step. Each weak classifier is trained with a random bootstrap sample drawn from the training dataset. Initially, weights are assigned to all data instances in the training dataset with equal probability. If there are 10 data instances in the training dataset, the probability of choosing a data instance to be trained by the first weak classifier would be 1/10. Hence, equal distribution of weights, that is, 1/10 is assigned to all data instances. After training by one weak classifier, the weights of all data instances are boosted based on the accuracy of classification done by it. The idea of boosting is to minimize the errors made by a weak classifier

by increasing the weights of misclassified data instances and decreasing the weights of correctly classified data instances. The updated probabilities of data instances are reflected in the training dataset and these instances get trained with the next classifier.

Each weak classifier is also assigned a weight after training, based on the accuracy of its classification. If the classifier is more accurate, it is given more weight. A 50% accurate classifier is given a weight of 0. If the classifier has less than 50% accuracy, it gets a negative weight.

There are many weak classifiers such as Decision trees, Decision stumps, and Multilayer Perceptron. Linear classifiers such as Decision stumps are one-level decision trees which perform poorly with low accuracy. The depth of decision stumps is 1 and it has two leaves. The hypothesis of a Decision Stump model can be given as follows:

$$H_{Decision\text{-}Stump}(x) = \begin{cases} 1 & \text{if } x > \theta \\ 0 & \text{else} \end{cases} \tag{12.1}$$

The final classifier makes a linear combination of all the predictions made by the weak classifiers.

Figure 12.5 illustrates the method of learning using the AdaBoost algorithm.

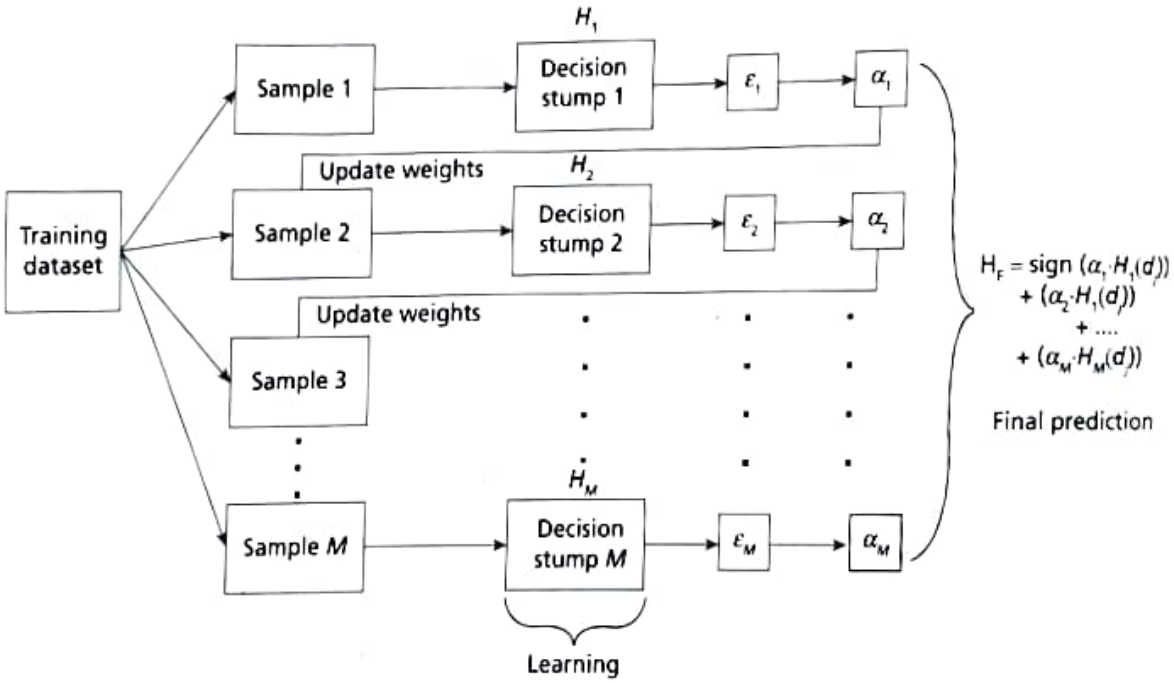


**Figure 12.5:** Working of AdaBoost Algorithm

**Algorithm 12.3: AdaBoost**

**Input:** Training dataset $T$ with $N$ data instances and $M$ weak classifiers.

**Step 1:** Assign uniform weights 1/N to all the data instances $d_j$ $(1 \le j \le N)$ in the training dataset $T$, i.e., $wt\ (d_1) = 1/N$, $wt\ (d_2) = 1/N$, ..., $wt\ (d_N) = 1/N$. Here, $wt$ implies weight.

**Step 2:** Repeat for each weak classifier.

**Step 2 (a):** Train a weak classifier $H_i$ with a random bootstrap sample from the training dataset $T$.

*(Continued)*

**Step 2 (b):** Compute the weighted error $\varepsilon_i$ of $H_i$ on current training dataset:

$$\varepsilon_i = \sum_{j=1}^{N} H_i(d_j) wt(d_j)$$

$H_i(d_j) = 0$ if prediction is correct with $H_i$

$H_i(d_j) = 1$ if prediction is wrong with $H_i$

**Step 2 (c):** Compute the weight of each weak classifier:

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

$\alpha_i$ is based on the error rate.

As the error rate increases, $\alpha_i$ grows exponentially negative.

Better classifiers get exponentially more weight.

If error rate is 50%, the classifier gets a weight of 0.

If error rate is more than 50%, the classifier gets a negative weight.

**Step 2 (d):** Calculate the normalizing factor $Z_i$. This is needed so that the sum of the weights of data instances adds up to 1.

$Z_i$ = Total weight of correctly classified instances + Total weight of incorrectly classified instances and Here, *wt* implies weight.

$$Z_i = wt(\text{correctly classified instance}) \times \text{Number of correct classifications} \times e^{-\alpha_i} + wt\begin{pmatrix}\text{incorrectly}\\ \text{classified instance}\end{pmatrix} \times \text{Number of incorrect classifications} \times e^{+\alpha_i}$$

**Step 2 (e):** Update the weight of all data instances:

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{wt(d_j)_i \text{ correct} - \text{instance} \times e^{-\alpha_i}}{Z_i}$$

$$wt(d_j)_{i+1} \text{ incorrect-instance} = \frac{wt(d_j)_i \text{ incorrect} - \text{instance} \times e^{+\alpha_i}}{Z_i}$$

where, $wt(d_j)_{i+1}$ correct-instance, is the updated weight of a correctly classified data instance to be trained by the next weak classifier.

$wt(d_j)_{i+1}$ incorrect-instance, is the updated weight of an incorrectly classified data instance to be trained by the next weak classifier.

*(Continued)*

Data instances that are wrongly classified will have their weights increased.

Data instances that are classified correctly will have their weights decreased.

**Step 3:** Compute the final predicted value for each data instance. It is the linear combination of weighted average predictions of all the classifiers multiplied by weight $\alpha_i$ computed for each classifier.

$$H_f(d_t) = \text{sign}\left(\sum_{i=1}^{M} \alpha_i \times H_i(d_j)\right)$$

**Example 12.2:** Consider a training dataset of six data instances as shown in Table 12.5.

**Table 12.5:** Training Dataset

| CGPA | Interactiveness | Practical Knowledge | Communication Skills | Job Offer |
|---|---|---|---|---|
| ≥9 | Yes | Good | Good | Yes |
| <9 | No | Good | Moderate | Yes |
| ≥9 | No | Average | Moderate | No |
| <9 | No | Average | Good | No |
| ≥9 | Yes | Good | Moderate | Yes |
| ≥9 | Yes | Good | Moderate | Yes |

Use 4 Decision Stumps for each of the 4 attributes.

The target attribute 'Job Offer' is a categorical attribute which predicts each data instance as 'Yes' or 'No'.

Number of data instances, N = 6.

Apply AdaBoost algorithm here.

**Solution:**

**Step 1:** Initial weight assigned to each item = 1/6.

**Step 2:** Iterate for each Weak classifier.

**I. Decision Stump for CGPA**

**Step 2 (a):** Train the Decision Stump $H_{CGPA}$ with a random bootstrap sample from the training dataset $T$. Since there are only 6 data instances, use the full training dataset.

The first Decision stump classifies the instances based on the CGPA attribute as shown in Table 12.6. If CGPA ≥ 9, the data instance is predicted to have 'Job Offer' as 'Yes' else 'No'.

**Table 12.6:** Decision Stump Prediction using $H_{CGPA}$

| CGPA | Predicted Job Offer | Actual Job Offer |
|---|---|---|
| ≥9 | Yes | Yes |
| <9 | No | Yes |
| ≥9 | Yes | No |
| <9 | No | No |
| ≥9 | Yes | Yes |
| ≥9 | Yes | Yes |

**Step 2 (b):** Compute the weighted error $\varepsilon_{CGPA}$ of $H_{CGPA}$ on current training dataset $T$:

$$\varepsilon_i = \sum_{j=1}^{N} H_i(d_j) wt(d_j) \qquad H_i(d_j) = 0 \text{ if prediction is correct with } H_i$$

$$H_i(d_j) = 1 \text{ if prediction is wrong with } H_i$$

$$\varepsilon_{CGPA} = 2 \times 1/6 = 0.333$$

**Step 2 (c):** Compute the weight of each weak classifier:

$$\alpha_{CGPA} = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_{CGPA}}{\varepsilon_{CGPA}}\right)$$

$$\alpha_{CGPA} = \frac{1}{2} \ln\left(\frac{1 - 0.333}{0.333}\right)$$

$$= 0.347$$

**Step 2 (d):** Calculate the normalizing factor $Z_{CGPA}$.

$$Z_{CGPA} = wt(\text{correctly classified instance}) \times \text{Number of correct classifications} \times e^{-\alpha_{CGPA}} + wt\begin{pmatrix}\text{incorrectly}\\ \text{classified instance}\end{pmatrix} \times \text{Number of incorrect classifications} \times e^{+\alpha_{CGPA}}$$

$$Z_{CGPA} = 1/6 \times 4 \times e^{-0.347} + 1/6 \times 2 \times e^{0.347}$$

$$= 0.1178 \times 4 + 0.2358 \times 2$$

$$= 0.9428$$

**Step 2 (e):** Update the weight of all data instances:

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{wt(d_j)_{CGPA} \text{ correct} - \text{instance} \times e^{-\alpha_{CGPA}}}{Z_{CGPA}}$$

$$wt(d_j)_{i+1} \text{ incorrect-instance} = \frac{wt(d_j)_{CGPA} \text{ incorrect} - \text{instance} \times e^{\alpha_{CGPA}}}{Z_{CGPA}}$$

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{1/6 \times e^{-0.347}}{0.9428}$$

$$= \frac{0.1178}{0.9482} = 0.1249$$

$$wt(d_j)_{i+1} \text{ incorrect-instance} = \frac{1/6 \times e^{0.347}}{0.9428}$$

$$= \frac{0.2358}{0.9482} = 0.2501$$

Table 12.7 shows the modified weights for the data instances after learning with $H_{CGPA}$.

Table 12.7: Modified Weights with $H_{CGPA}$

| CGPA | Predicted Job Offer | Actual Job Offer | Weights |
|---|---|---|---|
| ≥9 | Yes | Yes | 0.1249 |
| <9 | No | Yes | 0.2501 |
| ≥9 | Yes | No | 0.2501 |
| <9 | No | No | 0.1249 |
| ≥9 | Yes | Yes | 0.1249 |
| ≥9 | Yes | Yes | 0.1249 |

**II. Decision Stump for Interactiveness**

**Step 2 (a):** Train the Decision Stump $H_{Interactiveness}$ with the sample obtained from the previous weak classifier $H_{CGPA}$.

The second Decision stump classifies the instances based on the Interactiveness attribute as shown in Table 12.8. If Interactiveness = 'Yes', the data instance is predicted to have 'Job Offer' as 'Yes', else if Interactiveness = 'No' the data instance is predicted to have 'Job Offer' as 'No'.

Table 12.8: Decision Stump Prediction using $H_{Interactiveness}$

| Interactiveness | Predicted Job Offer | Actual Job Offer | Weights |
|---|---|---|---|
| Yes | Yes | Yes | 0.1249 |
| No | No | Yes | 0.2501 |
| No | No | No | 0.2501 |
| No | No | No | 0.1249 |
| Yes | Yes | Yes | 0.1249 |
| Yes | Yes | Yes | 0.1249 |

**Step 2 (b):** Compute the weighted error $\varepsilon_{Interctiveness}$ of $H_{Interactiveness}$.

Only one instance is misclassified by this decision stump:

$$\varepsilon_{Interactiveness} = 1 \times 0.2501 = 0.2501$$

**Step 2 (c):** Compute the weight of each weak classifier:

$$\alpha_{\text{Interactiveness}} = \frac{1}{2}\ln\left(\frac{1-\varepsilon_{Interactiveness}}{\varepsilon_{Interactiveness}}\right)$$

$$\alpha_{Interactiveness} = \frac{1}{2}\ln\left(\frac{1-0.2501}{0.2501}\right)$$

$$= 0.5490$$

**Step 2 (d):** Calculate the normalizing factor $Z_{Interactiveness}$.

$$Z_{Interactiveness} = wt(\text{correctly classified instance}) \times \text{Number of correct classifications} \times e^{-\alpha_{Interactiveness}}$$

$$+ wt\begin{pmatrix}\text{incorrectly}\\ \text{classified instance}\end{pmatrix} \times \text{Number of incorrect classifications} \times e^{+\alpha_{Interactiveness}}$$

$$Z_{Interactiveness} = 0.1249 \times 4 \times e^{-0.5490} + 0.2501 \times 1 \times e^{0.5490} + 0.2501 \times 1 \times e^{-0.5490}$$
$$= 0.2885 + 0.4331 + 0.1444$$
$$= 0.866$$

**Step 2 (e):** Update the weight of all data instances:

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{wt(d_j)_{Interactiveness} \text{ correct} - \text{instance} \times e^{-\alpha_{Interactiveness}}}{Z_{Interactiveness}}$$

$$wt(d_j)_{i+1} \text{ incorrect-instance} = \frac{wt(d_j)_{Interactiveness} \text{ incorrect} - \text{instance} \times e^{\alpha_{Interactiveness}}}{Z_{Interactiveness}}$$

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{0.1249 \times e^{-0.5490}}{0.866}$$
$$= \frac{0.072}{0.866} = 0.0832$$

$$wt(d_j)_{i+1} \text{ incorrect-instance} = \frac{0.2501 \times e^{0.5490}}{0.866}$$
$$= \frac{0.4331}{0.866} = 0.5001$$

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{0.2501 \times e^{-0.5490}}{0.866}$$
$$= \frac{0.1444}{0.866} = 0.1667$$

Table 12.9 shows the modified weights for the data instances after learning with $H_{Interactiveness}$.

**Table 12.9:** Modified Weights with $H_{Interactiveness}$

| Interactiveness | Predicted Job Offer | Actual Job Offer | Updated Weights |
|---|---|---|---|
| Yes | Yes | Yes | 0.0832 |
| No | No | Yes | 0.5001 |
| No | No | No | 0.1667 |
| No | No | No | 0.0832 |
| Yes | Yes | Yes | 0.0832 |
| Yes | Yes | Yes | 0.0832 |

### III. Decision Stump (Practical Knowledge)

**Step 2 (a):** Train the Decision Stump $H_{Practical\ Knowledge}$ with the sample obtained from the previous weak classifier $H_{Interactiveness}$.

The third Decision stump classifies the instances based on the Practical Knowledge attribute as shown in Table 12.10. If Practical Knowledge = 'Good' the data instance is predicted to have 'Job Offer' as 'Yes', else if Practical Knowledge = 'Average' the data instance is predicted to have 'Job Offer' as 'No'.

**Table 12.10:** Decision Stump Prediction using $H_{Practical\ Knowledge}$

| Practical Knowledge | Actual Job Offer | Predicted Job Offer | Weights |
|---|---|---|---|
| Good | Yes | Yes | 0.0832 |
| Good | Yes | Yes | 0.5001 |
| Average | No | No | 0.1667 |
| Average | No | No | 0.0832 |
| Good | Yes | Yes | 0.0832 |
| Good | Yes | Yes | 0.0832 |

No instances are misclassified by this Decision Stump. So, don't change the weights of the data instances.

### IV. Decision Stump (Communication Skills)

**Step 2 (a):** Train the Decision Stump $H_{Communication\ Skills}$ with the sample obtained from the previous weak classifier $H_{Interactiveness}$.

The fourth Decision stump classifies the instances based on the Communication Skills attribute as shown in Table 12.11. If Communication Skills = 'Good' the data instance is predicted to have 'Job Offer' as 'Yes', else if Communication Skills = 'Moderate' the data instance is predicted to have 'Job Offer' as 'No'.

**Table 12.11:** Decision Stump Prediction using $H_{Communication\ Skills}$

| Communication Skills | Predicted Job Offer | Actual Job Offer | Weights |
|---|---|---|---|
| Good | Yes | Yes | 0.0832 |
| Moderate | No | Yes | 0.5001 |
| Moderate | No | No | 0.1667 |
| Good | Yes | No | 0.0832 |
| Moderate | No | Yes | 0.0832 |
| Moderate | No | Yes | 0.0832 |

**Step 2 (b):** Compute the weighted error $\varepsilon_{Communication\ Skills}$ of $H_{Communication\ Skills}$.

Four instances are misclassified by this decision stump:

$$\varepsilon_{Communication\ Skills} = 1 \times 0.5001 + 3 \times 0.0832 = 0.7497$$

**Step 2 (c):** Compute the weight of each weak classifier:

$$\alpha_{\text{Communication Skills}} = \frac{1}{2}\ln\left(\frac{1-\varepsilon_{Communication\ Skills}}{\varepsilon_{Communication\ Skills}}\right)$$

$$\alpha_{\text{Communication Skills}} = \frac{1}{2}\ln\left(\frac{1-0.7497}{0.7497}\right)$$

$$= -0.5485$$

**Step 2 (d):** Calculate the normalizing factor $Z_{\text{Communication Skills}}$.

$$Z_{Communication\ Skills} = wt(\text{correctly classified instance}) \times \text{Number of correct classifications} \times e^{-\alpha_{Communication\ Skills}}$$

$$+ wt\left(\begin{matrix}\text{incorrectly}\\ \text{classified instance}\end{matrix}\right) \times \text{Number of incorrect classifications} \times e^{+\alpha_{Communication\ Skills}}$$

$$Z_{\text{Communication Skills}} = 0.0832 \times 1 \times e^{-(-0.5485)} + 0.1667 \times 1 \times e^{-(-0.5485)} + 0.0832 \times 3 \times e^{+(-0.5485)} + 0.5001 \times 1 \times e^{+(-0.5485)}$$
$$= 0.1440 + 0.2885 + 0.1442 + 0.2889$$
$$= 0.866$$

**Step 2 (e):** Update the weight of all data instances:

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{wt(d_j)_{\text{Communication Skills}} \text{ correct} - \text{instance} \times e^{-\alpha_{\text{Communication Skills}}}}{Z_{\text{Communication Skills}}}$$

$$wt(d_j)_{i+1} \text{ incorrect-instance} = \frac{wt(d_j)_{\text{Communication Skills}} \text{ incorrect} - \text{instance} \times e^{+\alpha_{\text{Communication Skills}}}}{Z_{\text{Communication Skills}}}$$

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{0.0832 \times e^{-(-0.5485)}}{0.866} = 0.1663$$

$$wt(d_j)_{i+1} \text{ correct-instance} = \frac{0.1667 \times e^{-(-0.5485)}}{0.866} = 0.3331$$

$$wt(d_j)_{i+1} \text{ incorrect-instance} = \frac{0.5001 \times e^{+(-0.5485)}}{0.866} = 0.3337$$

$$wt(d_j)_{i+1} \text{ incorrect-instance} = \frac{0.0832 \times e^{+(-0.5485)}}{0.866} = 0.0555$$

Table 12.12 shows the modified weights for the data instances after learning with $H_{\text{Communication Skills}}$.

**Table 12.12:** Modified Weights with $H_{\text{Communication Skills}}$

| Communication Skills | Predicted Job Offer | Actual Job Offer | Updated Weights |
|---|---|---|---|
| Good | Yes | Yes | 0.1663 |
| Moderate | No | Yes | 0.3337 |
| Moderate | No | No | 0.3331 |
| Good | Yes | No | 0.0555 |
| Moderate | No | Yes | 0.0555 |
| Moderate | No | Yes | 0.0555 |

We can observe that the weights of misclassified instances are increased.

**Step 3:** Compute the final predicted value for each data instance:

$$H_F(d_j) = \text{sign}\left(\sum_{i=1}^{M} \alpha_i \times H_i(d_j)\right)$$

The sign function classifies instances as positive if greater than 0 and negative if less than 0. A value of 0 is considered neither positive nor negative.

The final prediction for data instance 1 is computed as shown below,

$$H_f(d_{j-1}) = \alpha_{CGPA} \times \text{Yes} + \alpha_{Interactiveness} \times \text{Yes} + \alpha_{Communication\ Skills} \times \text{Yes}$$

$$= 0.347 \times 1 + 0.5490 \times 1 + -0.5485 \times 1 = 0.3475$$

$$H_f(d_{j-1}) = \text{Yes}$$

This instance is classified with 'Job Offer = Yes' since the weighted average is positive.

Table 12.13 shows the final predicted value for all the data instances in the training dataset. It can be observed that the algorithm does not over-fit with the training dataset.

**Table 12.13:** Final Prediction

| Instances | $\alpha_{CGPA} = 0.347$ | $\alpha_{Interactiveness} = 0.5490$ | $\alpha_{Communiction\ Skills} = -0.5485$ | Weighted Average | Final Prediction |
|---|---|---|---|---|---|
| 1. | Yes | Yes | Yes | 0.3475 | Yes |
| 2. | No | No | No | 0 | No |
| 3. | Yes | No | No | 0.347 | Yes |
| 4. | No | No | Yes | −0.5485 | No |
| 5. | Yes | Yes | No | 0.896 | Yes |
| 6. | Yes | Yes | No | 0.896 | Yes |

## *Advantages of AdaBoost Algorithm*

1. Fast
2. Simple to implement

## *Disadvantages of AdaBoost Algorithm*

1. From empirical evidence and theoretical evidence, AdaBoost may overfit
2. Sensitive to noise data and outliers

Scan for information on *'Gradient Tree Boosting' and 'XGBoost'*

## Summary

1. Ensemble learning is a machine learning model which combines the predictions of several classifiers in order to improve the prediction accuracy of using a single classifier.
2. Bagging and Boosting are two popular models of ensemble learning.
3. A simple way of ensemble learning is majority voting or taking an average or weighted average of predictions of multiple machine learning models or base learners.
4. Ensembling machine learning is an approach to minimize bias and variance, and to build accurate models that avoid over-fitting and under-fitting of learning models.
5. Bagging is otherwise called as bootstrap aggregation, which combines both bootstrapping and aggregation.