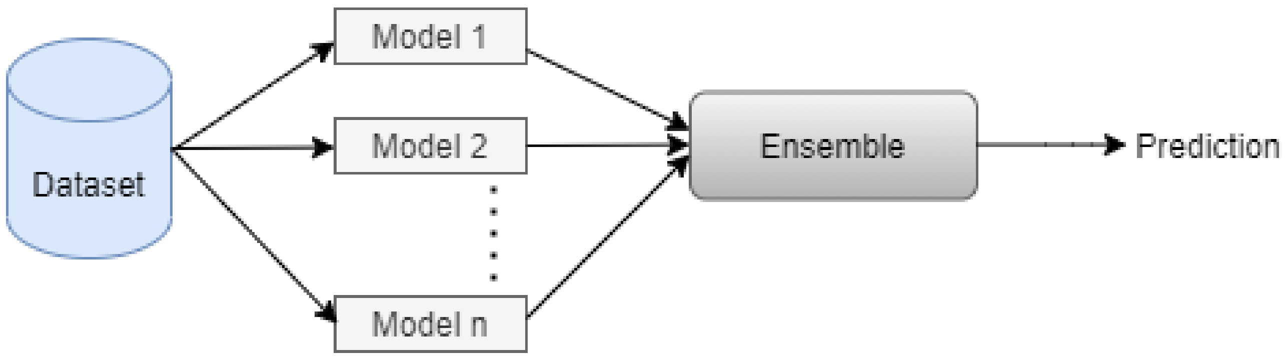
Ensemble Learning

Suppose you ask a complex question to thousands of random people, then aggregate their answers. In many cases, you will find that this aggregated answer is better than an expert's answer. This is called the *wisdom of the crowd*.

Wisdom of Crowd: Brain Games how many gumballs? [click on the link above to see an example of wisdom of crowd]

With this example, you can infer that a diverse group of people are likely to make better decisions as compared to individuals.

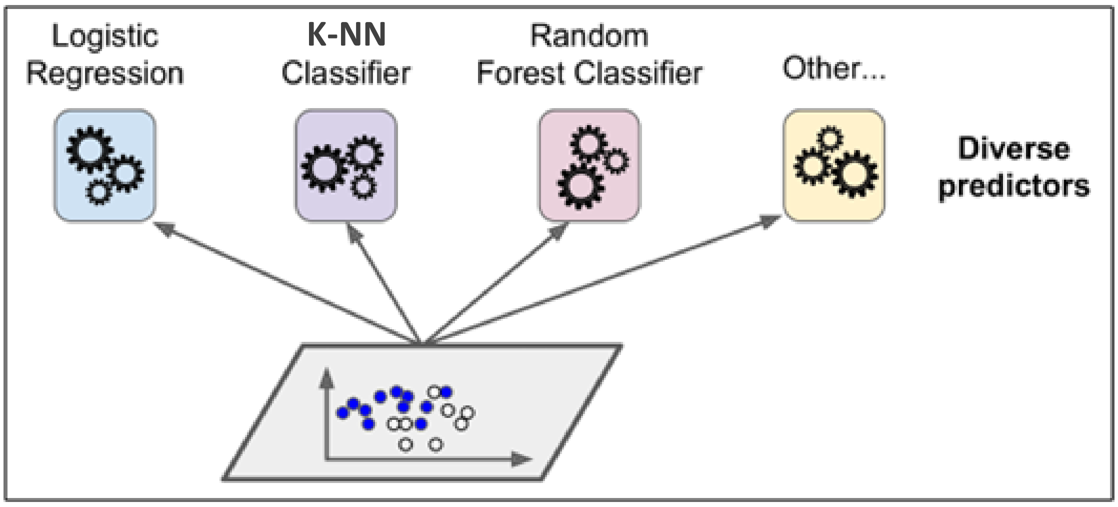
Similarly, if you aggregate the predictions of a group of predictors (such as classifiers or regressors), you will often get better predictions than with the best individual predictor. A group of predictors is called an ensemble; thus, this technique is called Ensemble Learning, and an Ensemble Learning algorithm is called an Ensemble method.



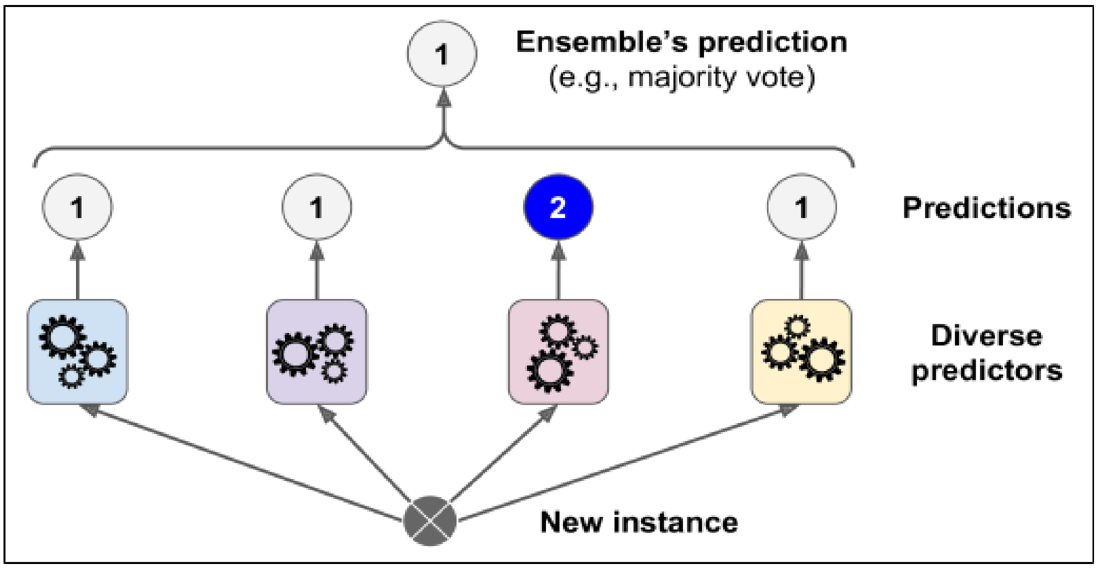
For example, you can train a group of Decision Tree classifiers, each on a different random subset of the training set. To make predictions, you just obtain the predictions of all individual trees, then predict the class that gets the most votes (as in the last session). Such an ensemble of Decision Trees is called a Random Forest, and despite its simplicity, this is one of the most powerful Machine Learning algorithms available today.

Voting Classifiers

Suppose you have trained a few classifiers, each one achieving about 70% accuracy. You may have a Logistic Regression classifier, a Random Forest classifier, a K-Nearest Neighbors classifier, and perhaps a few more (see figure)



A very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes. This majority-vote classifier is called a hard voting classifier (see figure)



Somewhat surprisingly, this voting classifier often achieves higher accuracy than the best classifier in the ensemble.

In fact, even if each classifier is a **weak learner** (meaning it does only slightly better than random guessing), the ensemble can still be a strong learner (achieving high accuracy), provided there are a sufficient number of weak learners and they are sufficiently diverse.

If all classifiers are able to estimate class probabilities (i.e., they have a `predict_proba()` method), then you can tell Scikit-Learn to predict the class with the highest class probability, averaged over all the individual classifiers. This is called **soft voting**. It often achieves higher performance than hard voting because it gives more weight to highly confident votes. All you need to do is replace `voting="hard"` with `voting="soft"` and ensure that all classifiers can estimate class probabilities.

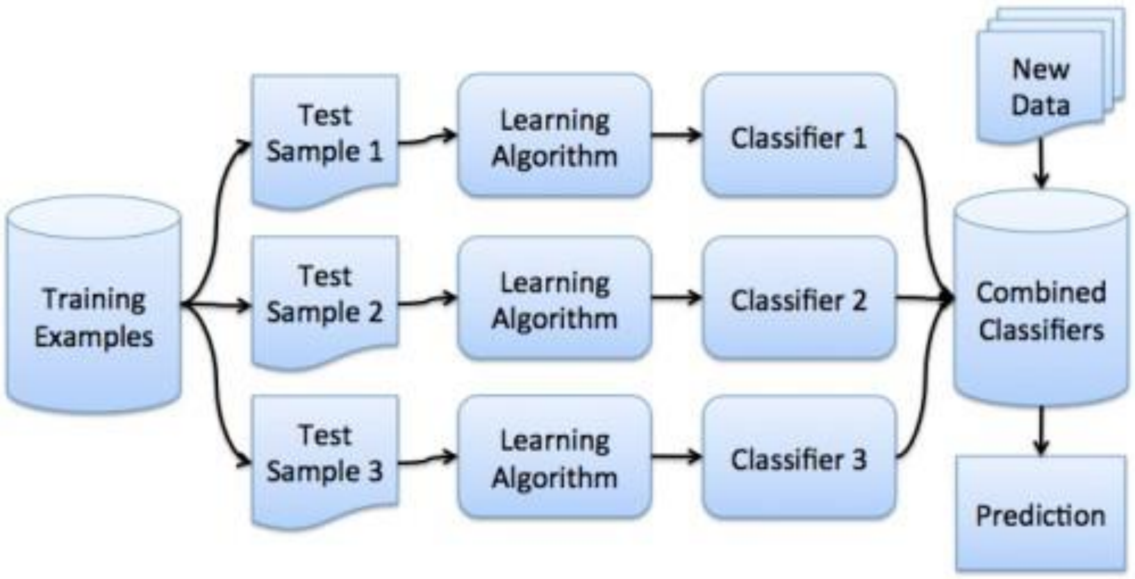
Bagging

Ensemble methods work best when the predictors are as independent from one another as possible.

<u>Here's a question:</u> *If you create all the models on the same set of data and combine it, will it be useful?* There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem?

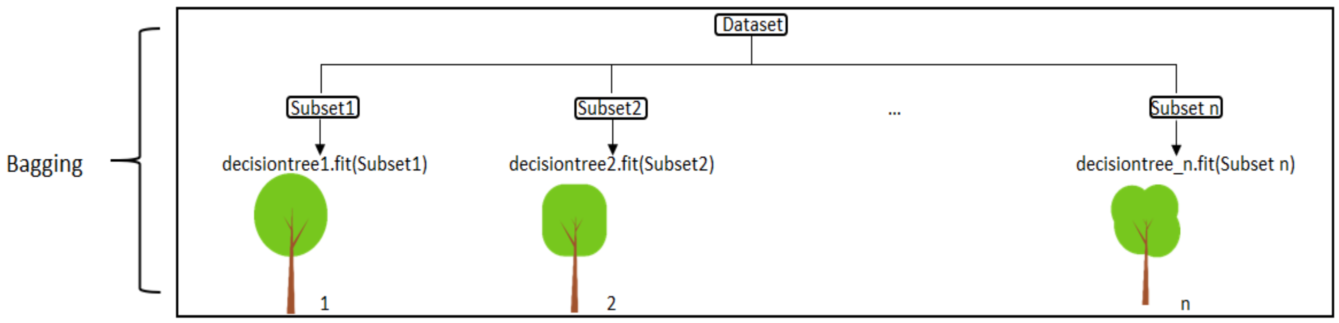
One way to get a diverse set of classifiers is to use very different training algorithms, as just discussed. Another approach is to use the same training algorithm for every predictor, but to train them on different random subsets of the training set. When sampling is performed with replacement, this method is called bagging (short for **b**ootstrap **agg**regating).

In other words, bagging allow training instances to be sampled several times across multiple predictors and also allows training instances to be sampled several times for the same predictor.

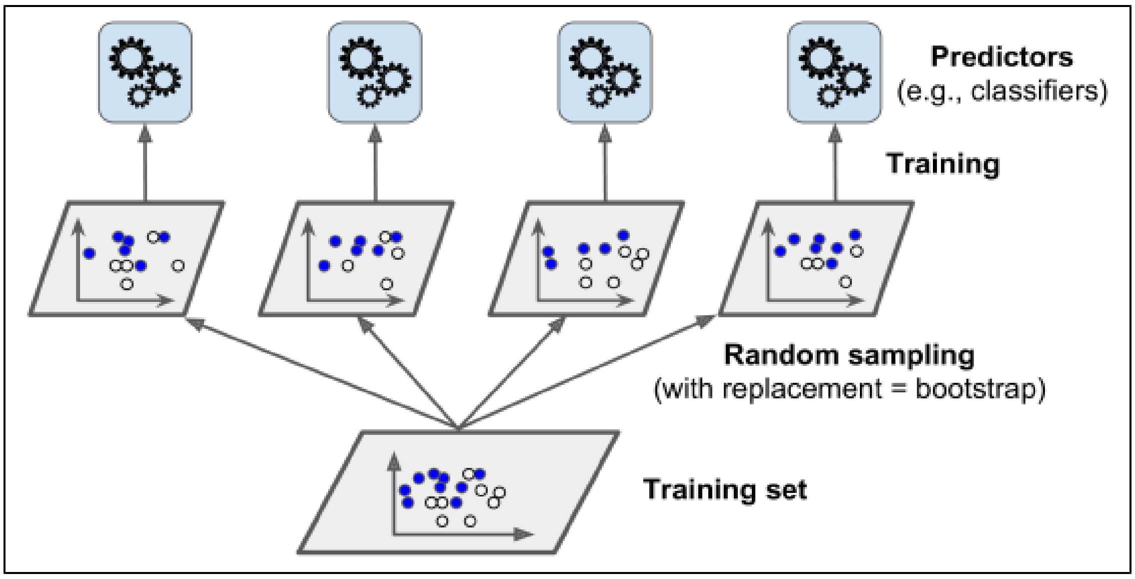


Once all predictors are trained, the ensemble can make a prediction for a new instance by simply aggregating the predictions of all predictors. The aggregation function is typically the statistical mode (i.e., the most frequent prediction, just like a hard voting classifier) for classification, or the average for regression.

For example, we have discussed Random Forest which is a Bagging algorithm and can be used for both classification and regression problems.


Dataset
Subset1
Subset2
...
Subset n
Bagging
decisiontree1.fit(Subset1)
decisiontree2.fit(Subset2)
decisiontree_n.fit(Subset n)
1
2
n

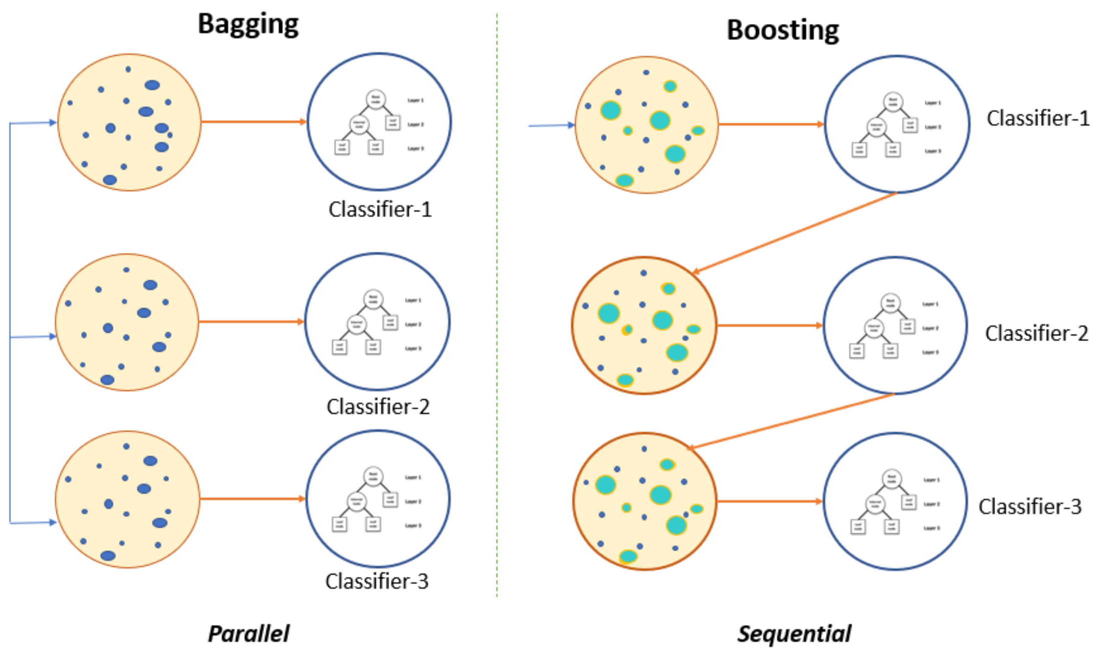
In the bagging method, all the individual models are built parallel, each individual model is different from one other. In this method, all the observations in the bootstrapping sample will be treated equally. In other words, all the observations will have equal at zero weightage.



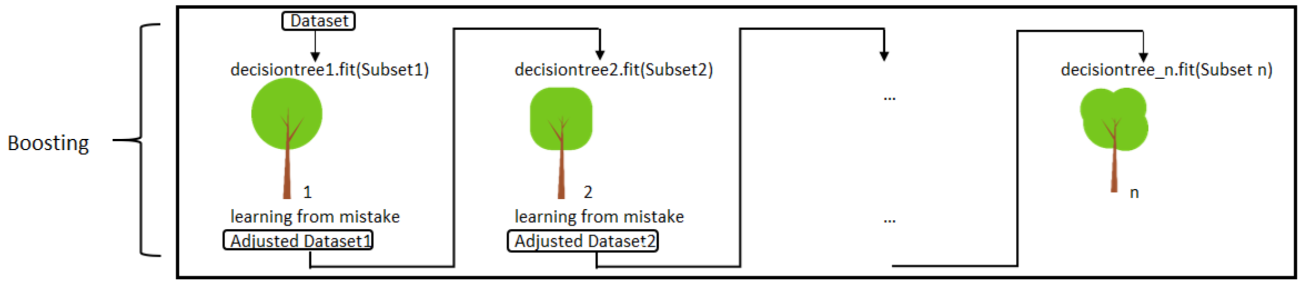
Boosting

Just as humans learn from their mistakes and try not to repeat them further in life, the Boosting algorithm tries to build a strong learner (predictive model) from the mistakes of several weaker models.

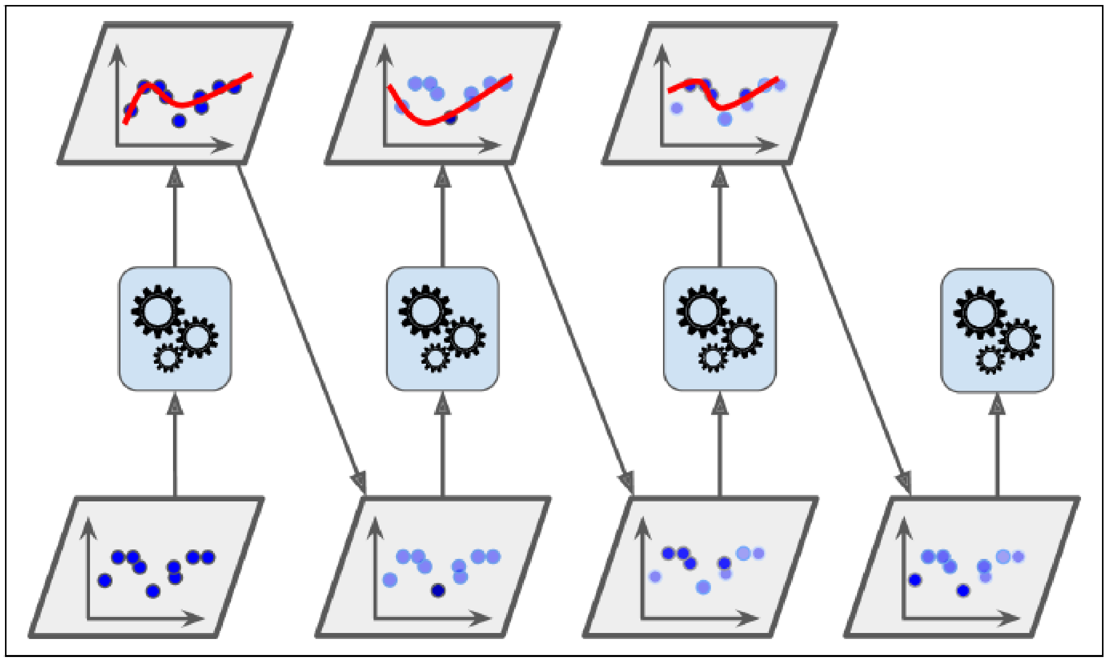
Boosting (originally called hypothesis boosting) refers to any Ensemble method that can combine several weak learners into a strong learner. The general idea of most boosting methods is: *You start by creating a model from the training data. Then, you create a second model from the previous one by trying to reduce the errors from the previous model*. Models are added sequentially, each correcting its predecessor, until the training data is predicted perfectly or the maximum number of models have been added.



For example, if we consider Random Forest for boosting,



In the boosting method, all the individual models are built sequentially. Which means the outcome of the first model passes to the next model and etc. Unlike bagging all the observations in the bootstrapping sample are not equally treated in boosting. Observations will have some weightage. For a few observations, the weightage will be high for others lower.



These two are the most important terms describing the ensemble (combination) of various models into one more effective model.

- ✓ *Bagging to decrease the model's variance.*
- ✓ *Boosting to decrease the model's bias.*

AdaBoost (Adaptive Boosting)

One way for a new predictor to correct its predecessor is to pay a bit more attention to the training instances that the predecessor underfitted. This results in new predictors focusing more and more on the hard cases. This is the technique used by Ada-Boost.

For example, to build an AdaBoost classifier, a first base classifier (such as a Decision Tree) is trained and used to make predictions on the training set. The relative weight of misclassified training instances is then increased. A second classifier is trained using the updated weights and again it makes predictions on the training set, weights are updated, and so on.

Let's understand the way boosting works in the below steps.

1. Each instance weight w_i is initially set to $\frac{1}{m}$. A first model is trained and its weighted error rate r_1 is computed on the training set using formula:

$$weighted\ error\ rate\ of\ the\ j^{th}\ model, \qquad r_j = \sum_{\substack{i=1 \\ \hat{y}_j \neq y_j}}^{m} w_i \Big/ \sum_{i=1}^{m} w_i$$

Where $\hat{y}_j$ is the j^{th} model prediction for the i^{th} instance / training example.

2. The model's weight α_j is then computed using:

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

where η is the learning rate hyperparameter (defaults to 1)

> - the higher weighted error rate of a tree, ☹, the less decision power the tree will be given during the later voting
> - the lower weighted error rate of a tree, ☺, the higher decision power the tree will be given during the later voting

3. Next the instance weights are updated using the following equations so that the misclassified instances are boosted. *Misclassified item is assigned higher weight so that it appears in the training subset of next classifier with higher probability*.

$$for\ i = 1, 2, \cdots, m$$

$$w_i = \begin{cases} w_i & if\ \hat{y}_j = y_j \\ w_i\, e^{\alpha_j} & if\ \hat{y}_j \neq y_j \end{cases}$$

Then all the instance weights are normalized (i.e., divided by $\sum_{i=1}^{m} w_i$

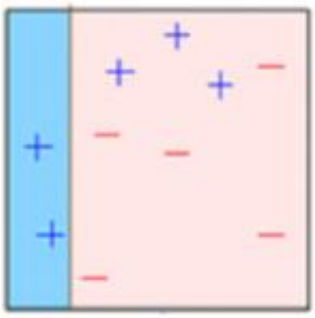
4. Finally, a new model is trained using the updated weights, and the whole process is repeated (the new model's weight is computed, the instance weights are updated, then another model is trained, and so on). The algorithm stops when the desired number of models is reached, or when a perfect model is found.

To make predictions, AdaBoost simply computes the predictions of all the models and weighs them using the model weights α_j. The predicted class is the one that receives the majority of weighted votes.

If you simply ignore the mathematical formulas in the algorithm, it becomes pretty straight forward:

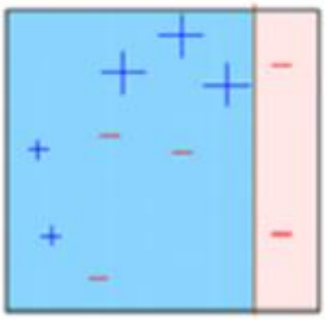
1. Initially, all data points are given equal weights.

2. A base model is created. This model is used to make predictions on the whole dataset.



3. Errors are calculated using the actual values and predicted values.

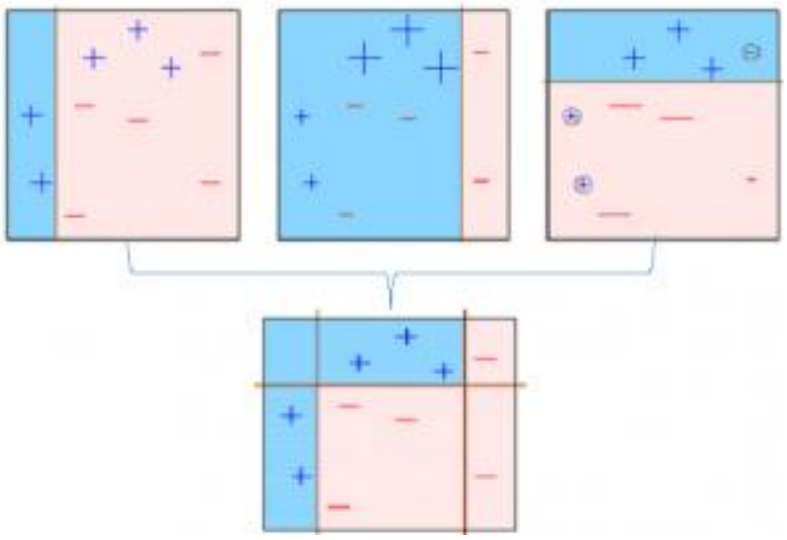
4. The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)



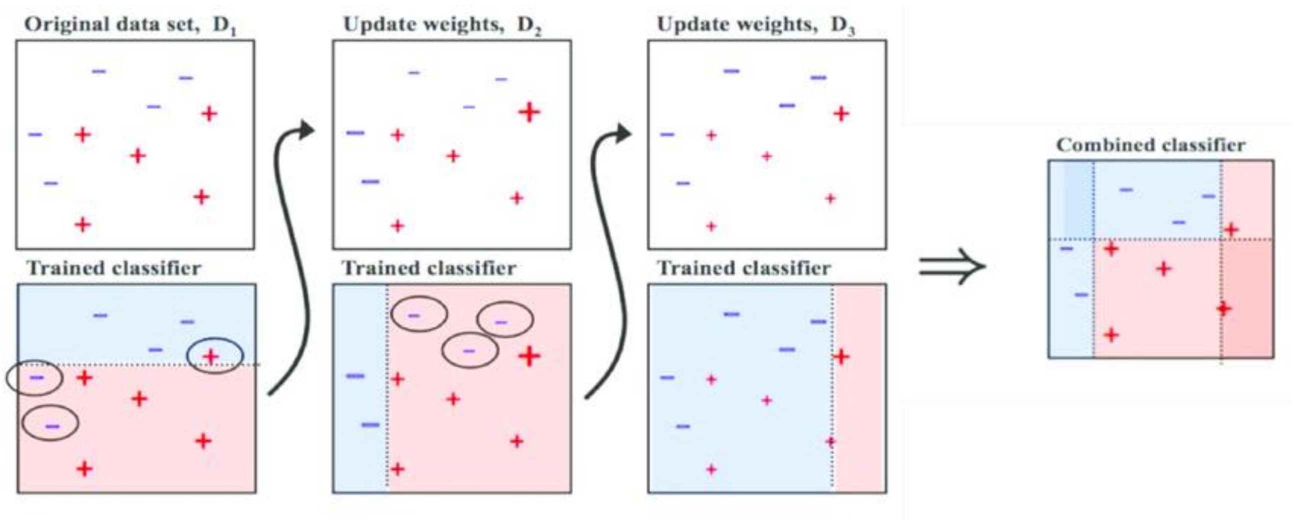
5. Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)

6. Similarly, multiple models are created, each correcting the errors of the previous model.

7. The final model (strong learner) is the weighted mean of all the models (weak learners).



Thus, the boosting algorithm combines a number of weak learners to form a strong learner. The individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.



Original data set, D_1
Update weights, D_2
Update weights, D_3
Trained classifier
Trained classifier
Trained classifier
Combined classifier