**Underfitting and Overfitting:**

When we talk about the Machine Learning model, we actually talk about how well it performs and its accuracy which is known as prediction errors. Let us consider that we are designing a machine learning model. A model is said to be a good machine learning model if it generalizes any new input data from the problem domain in a proper way. This helps us to make predictions about future data, that the data model has never seen. Now, suppose we want to check how well our machine learning model learns and generalizes to the new data. For that, we have overfitting and underfitting, which are majorly responsible for the poor performances of the machine learning algorithms.

**Important terms:**

* **Bias:** Assumptions made by a model to make a function easier to learn. It is actually the error rate of the training data. When the error rate has a high value, we call it High Bias and when the error rate has a low value, we call it low Bias.
* **Variance:** The difference between the error rate of training data and testing data is called variance. If the difference is high then it’s called high variance and when the difference of errors is low then it’s called low variance. Usually, we want to make a low variance for generalized our model.

**Underfitting:** A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data, i.e., it only performs well on training data but performs poorly on testing data.

Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have fewer data to build an accurate model and also when we try to build a linear model with fewer non-linear data. In such cases, the rules of the machine learning model are too easy and flexible to be applied to such minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data and also reducing the features by feature selection.

In a nutshell, Underfitting refers to a model that can neither performs well on the training data nor generalize to new data.

**Reasons for Underfitting:**

1. High bias and low variance
2. The size of the training dataset used is not enough.
3. The model is too simple.
4. Training data is not cleaned and also contains noise in it.

**Techniques to reduce underfitting:**

1. Increase model complexity
2. Increase the number of features, performing feature engineering
3. Remove noise from the data.
4. Increase the number of epochs or increase the duration of training to get better results.

**Overfitting:** A statistical model is said to be overfitted when the model does not make accurate predictions on testing data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. And when testing with test data results in High variance. Then the model does not categorize the data correctly, because of too many details and noise.

The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore, they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

In a nutshell, Overfitting is a problem where the evaluation of machine learning algorithms on training data is different from unseen data.

**Reasons for Overfitting are as follows:**

1. High variance and low bias
2. The model is too complex
3. The size of the training data

**Techniques to reduce overfitting:**

1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
4. Ridge Regularization and Lasso Regularization
5. Use dropout for neural networks to tackle overfitting.

**Good Fit in a Statistical Model:** Ideally, the case when the model makes the predictions with 0 error, is said to have a good fit on the data. This situation is achievable at a spot between overfitting and underfitting. In order to understand it, we will have to look at the performance of our model with the passage of time, while it is learning from the training dataset.

With the passage of time, our model will keep on learning, and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to the presence of noise and less useful details. Hence the performance of our model will decrease.

In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point, the model is said to have good skills in training datasets as well as our unseen testing dataset.

**Ensemble learning** helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote.

**Advantage:** Improvement in predictive accuracy.

**Disadvantage:** It is difficult to understand an ensemble of classifiers.

Diagram

Description automatically generated

**Why do ensembles work?**

**Statistical Problem –**

The Statistical Problem arises when the hypothesis space is too large for the amount of available data. Hence, there are many hypotheses with the same accuracy on the data and the learning algorithm chooses only one of them! There is a risk that the accuracy of the chosen hypothesis is low on unseen data!

**Computational Problem –**

The Computational Problem arises when the learning algorithm cannot guarantees finding the best hypothesis.

**Representational Problem –**

The Representational Problem arises when the hypothesis space does not contain any good approximation of the target class(es).

**Main Challenge for Developing Ensemble Models?**

The main challenge is not to obtain highly accurate base models, but rather to obtain base models which make different kinds of errors. For example, if ensembles are used for classification, high accuracies can be accomplished if different base models misclassify different training examples, even if the base classifier accuracy is low.

**Types of Ensemble Classifier –**

**Bagging and Boosting** are two types of Ensemble Learning. These two decrease the variance of a single estimate as they combine several estimates from different models. So the result may be a model with higher stability. Let’s understand these two terms in a glimpse.

**Bagging:** It is a homogeneous weak learners’ model that learns from each other independently in parallel and combines them for determining the model average.

**Boosting:** It is also a homogeneous weak learners’ model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

**Bagging:**

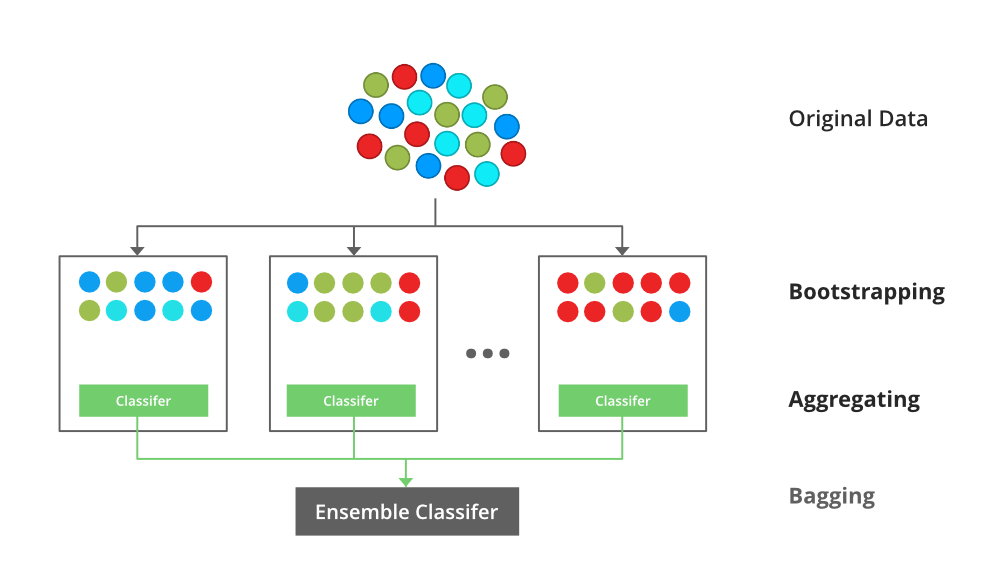
Bootstrap Aggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting. It is usually applied to decision tree methods. Bagging is a special case of the model averaging approach.

**Description of the Technique**

Suppose a set D of d tuples, at each iteration i, a training set Di of d tuples is selected via row sampling with a replacement method (i.e., there can be repetitive elements from different d tuples) from D (i.e., bootstrap). Then a classifier model Mi is learned for each training set D < i. Each classifier Mi returns its class prediction. The bagged classifier M\* counts the votes and assigns the class with the most votes to X (unknown sample).

**Implementation steps of Bagging –**

1. Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.
2. A base model is created on each of these subsets.
3. Each model is learned in parallel from each training set and independent of each other.
4. The final predictions are determined by combining the predictions from all the models.

****

**Example:**

Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples (or data) from the original training dataset – where N is the size of the original training set. Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out.

Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though.

**Original training dataset:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

**Resampled training set 1**: 2, 3, 3, 5, 6, 1, 8, 10, 9, 1

**Resampled training set 2:** 1, 1, 5, 6, 3, 8, 9, 10, 2, 7

**Resampled training set 3:** 1, 5, 8, 9, 2, 10, 9, 7, 5, 4

**Algorithm for the Bagging classifier:**

***Classifier generation:***

Let N be the size of the training set.

for each of t iterations:

sample N instances with replacement from the original training set.

apply the learning algorithm to the sample.

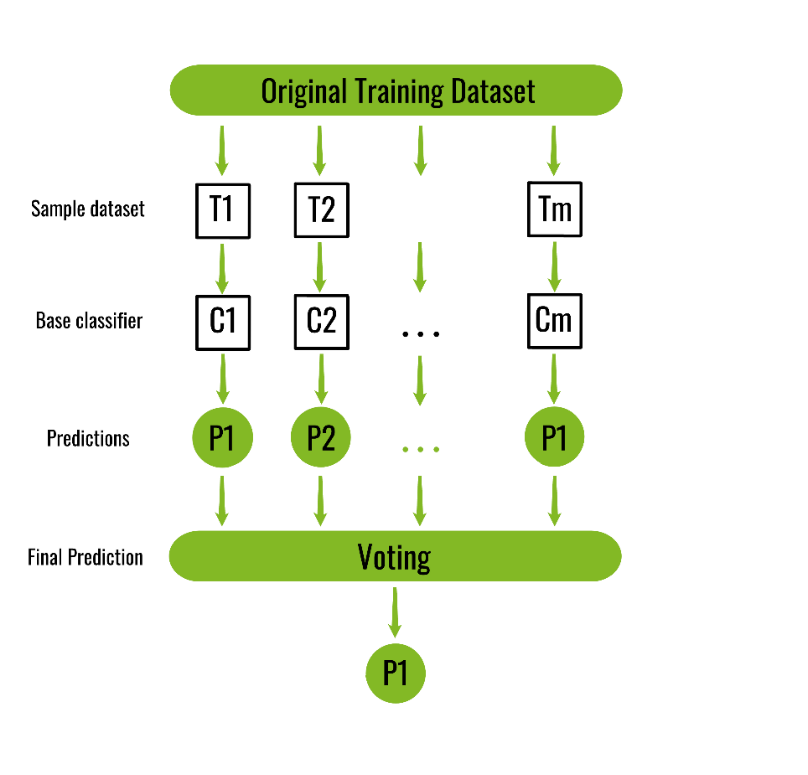
store the resulting classifier.

***Classification:***

for each of the t classifiers:

predict class of instance using classifier.

return class that was predicted most often.



**Random Forest:**

Random Forest is an extension over bagging. Each classifier in the ensemble is a decision tree classifier and is generated using a random selection of attributes at each node to determine the split. During classification, each tree votes and the most popular class is returned.

**Implementation steps of Random Forest –**

1. Multiple subsets are created from the original data set, selecting observations with replacement.
2. A subset of features is selected randomly and whichever feature gives the best split is used to split the node iteratively.
3. The tree is grown to the largest.
4. Repeat the above steps and prediction is given based on the aggregation of predictions from n number of trees.

Diagram

Description automatically generated

**Boosting:**

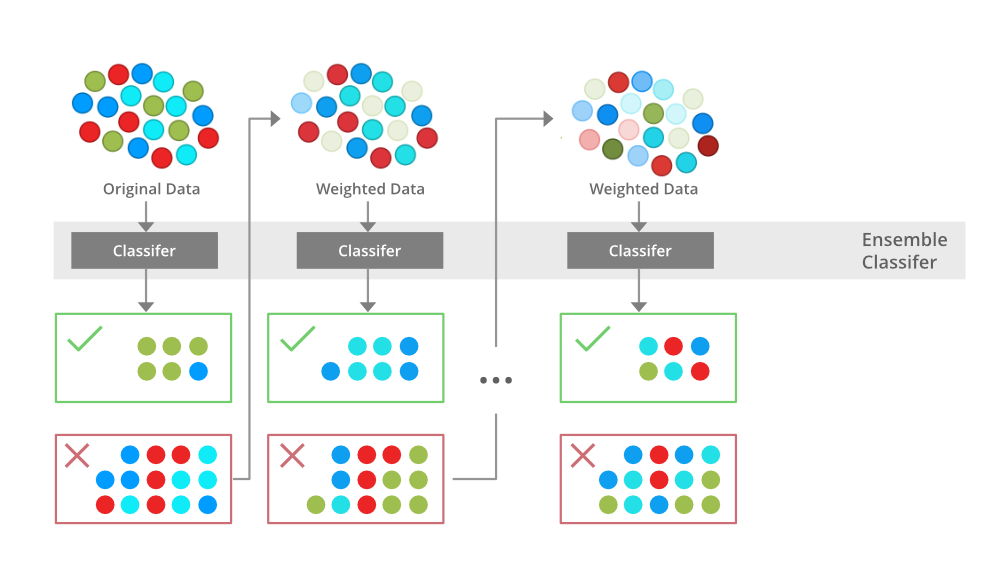
Boosting is an ensemble modelling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued, and models are added until either the complete training data set is predicted correctly or the maximum number of models is added.

**Boosting Algorithms:**

AdaBoost is short for Adaptive Boosting and is a very popular boosting technique that combines multiple “weak classifiers” into a single “strong classifier”.

**Algorithm:**

1. Initialise the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. And then normalize the weights of all data points.
4. if (got required results)  
     Goto step 5  
   else  
     Goto step 2
5. End



**Example:**

Diagram

Description automatically generated

**Explanation:**

The above diagram explains the AdaBoost algorithm in a very simple way. Let’s try to understand it in a stepwise process:

**B1** consists of 10 data points which consist of two types namely plus(+) and minus(-) and 5 of which are plus(+) and the other 5 are minus(-) and each one has been assigned **equal weight** initially. The first model tries to classify the data points and generates a vertical separator line but it wrongly classifies **3 plus(+)** as minus(-).

**B2** consists of the 10 data points from the previous model in which the **3 wrongly** classified plus(+) are weighted more so that the current model tries more to classify **these pluses(+)** correctly. This model generates a vertical separator line that correctly classifies the previously wrongly classified pluses(+) but in this attempt, it **wrongly classifies** three minuses(-).

**B3** consists of the 10 data points from the previous model in which the **3 wrongly** classified minus(-) are weighted more so that the current model tries more to classify these minuses(-) correctly. This model generates a horizontal separator line that correctly classifies the previously wrongly classified minuses(-).

**B4** combines together B1, B2, and B3 in order to build a strong prediction model which is much better than any individual model used.

**Similarities Between Bagging and Boosting**

Bagging and Boosting, both being the commonly used methods, have a universal similarity of being classified as ensemble methods. Here we will explain the similarities between them.

1. Both are ensemble methods to get N learners from 1 learner.
2. Both generate several training data sets by random sampling.
3. Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
4. Both are good at reducing variance and provide higher stability.

**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 |

**K-Fold Cross-Validation:**

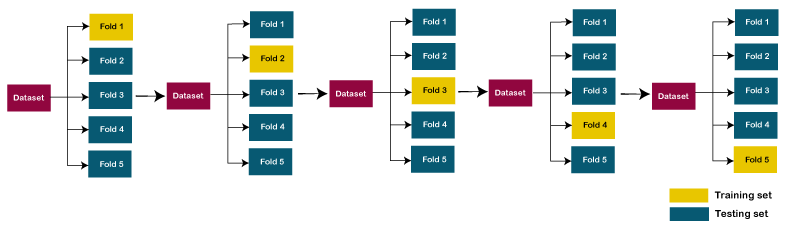
K-fold cross-validation approach divides the input dataset into K groups of samples of equal sizes. These samples are called folds. For each learning set, the prediction function uses k-1 folds, and the rest of the folds are used for the test set.

**The steps for k-fold cross-validation are:**

* Split the input dataset into K groups
* For each group:
* Take one group as the reserve or test data set.
* Use remaining groups as the training dataset
* Fit the model on the training set and evaluate the performance of the model using the test set.

Let's take an example of 5-folds cross-validation. So, the dataset is grouped into 5 folds. On 1st iteration, the first fold is reserved for test the model, and rest are used to train the model. On 2nd iteration, the second fold is used to test the model, and rest are used to train the model. This process will continue until each fold is not used for the test fold.

Consider the below diagram:



**XgBoost**

XgBoost stands for Extreme Gradient Boosting, which was proposed by the researchers at the University of Washington. It is a library written in C++ which optimizes the training for Gradient Boosting.

**Gradient Boosting**

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor’s error. In this, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

There is a technique called the Gradient Boosted Trees whose base learner is CART (Classification and Regression Trees).

**XGBoost**

XGBoost is an implementation of Gradient Boosted decision trees.

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

**Mathematics behind XgBoost**

Before beginning with mathematics about Gradient Boosting, Here’s a simple example of a CART that classifies whether someone will like a hypothetical computer game X. The example of tree is below:

The prediction scores of each individual decision tree then sum up to get If you look at the example, an important fact is that the two trees try to complement each other. Mathematically, we can write our model in the form

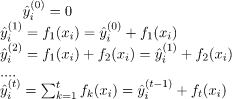


where, K is the number of trees, f is the functional space of F, F is the set of possible CARTs. The objective function for the above model is given by:



where, first term is the loss function and the second is the regularization parameter. Now, Instead of learning the tree all at once which makes the optimization harder, we apply the additive strategy, minimize the loss what we have learned and add a new tree which can be summarised below:

where, first term is the loss function and the second is the regularization parameter. Now, Instead of learning the tree all at once which makes the optimization harder, we apply the additive strategy, minimize the loss what we have learned and add a new tree which can be summarised below:



The objective function of the above model can be defined as:





Now, let’s apply taylor series expansion upto second order:



where g\_i and h\_i can be defined as:



Simplifying and removing the constant:



Now, we define the regularization term, but first we need to define the model:



Here, w is the vector of scores on leaves of tree, q is the function assigning each data point to the corresponding leaf, and T is the number of leaves. The regularization term is then defined by:



Now, our objective function becomes:



Now, we simplify the above expression:



where,



In this equation, w\_j are independent of each other, the best w\_j  for a given structure q(x) and the best objective reduction we can get is:



where, \gamma is pruning parameter, i.e the least information gain to perform split.

Now, we try to measure how good the tree is, we can’t directly optimize the tree, we will try to optimize one level of the tree at a time. Specifically we try to split a leaf into two leaves, and the score it gains is



Table

Description automatically generated with low confidence

*Calculation of Information Gain*

**Examples**

Let’s consider an example dataset:

|  |  |  |
| --- | --- | --- |
| Years of Experience | Gap | Annual salary (in 100k) |
| 1 | N | 4 |
| 1.5 | Y | 4 |
| 2.5 | Y | 5.5 |
| 3 | N | 7 |
| 5 | N | 7.5 |
| 6 | N | 8 |

First we take the base learner, by default the base model always take the average salary i.e F\_0 =6 (100k). Now, we calculate the residual values:

|  |  |  |  |
| --- | --- | --- | --- |
| Years of Experience | Gap | Annual salary (in 100k) | Residuals |
| 1 | N | 4 | -2 |
| 1.5 | Y | 4 | -2 |
| 2.5 | Y | 5.5 | -0.5 |
| 3 | N | 7 | 1 |
| 5 | N | 7.5 | 1.5 |
| 6 | N | 8 | 2 |

Now, let’s consider the decision tree, we will be splitting the data based on experience <=2 or otherwise.

Diagram

Description automatically generated

Now, let’s calculate the similarity metrices of left and right side. Since, it is the regression problem the similarity metric will be:



where, \lambda = hyperparameter

and for the classification problem:



where, P\_r = probability of either left side of right side. Let’s take \lambda = 1 ,the similarity metrics of the left side:



and for the right side.



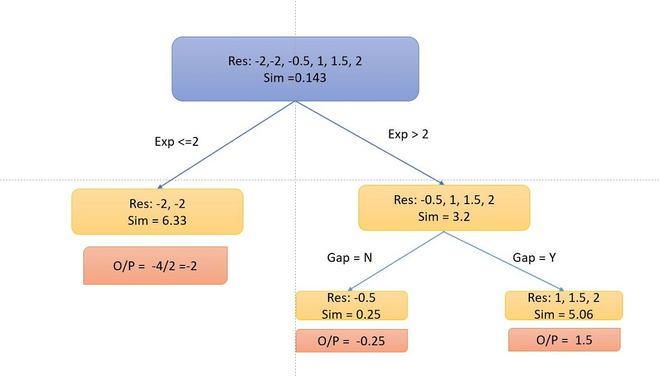
and for the top branch:



Now, the information gain from this split is:



Similarly, we can try multiple splits and calculate the information gain. We will take the split with the highest information gain. Let’s for now take this information gain. Further, we will split the decision tree if there is a gap or not.



Now, As you can notice that I didn’t split into the left side because the information Gain becomes negative. So, we only perform split on the right side.

To calculate the particular output, we follow the decision tree multiplied with a learning rate \alpha (let’s take 0.5) and add with the previous learner (base learner for the first tree) i.e for data point 1: o/p = 6 + 0.5 \*-2 =5. So our table becomes.

Similarly, the algorithm produces more than one decision tree and combine them additively to generate better estimates