**Ensemble Methods: A combination of Machine Learning Models to improve accuracy**

Ensemble method why a combination of machine learning Models? Ensemble methods are famous machine learning techniques that work by combining several base learners (models) to produce an optimal model having an improved accuracy. They are also known as meta-algorithms.

Let us understand the concept from an example: if you want to purchase a car, it is highly unlikely you will conclude your buying decision in an instant. Most likely you will evaluate your discission through several methods. You will browse through different web portals to look for car models, features, their prices, and reviews. Most probably you will ask for opinions from your friends or colleagues. Or you will walk to the car shop and takes advice from a car dealer. This means your decision uses opinions (combination of knowledge) from many other people. The same is the case with ensemble methods. They utilize multiple models/opinions and then combine them for improved performance.

**Why they are famous:** As we know machine learning is positively impacting business efficiency, these meta-algorithms generally produce more accurate solutions than a single model depending on the dataset consistency. There exists a number of winning solutions in machine learning competitions that have used ensemble methods for winning such as collaborative filtering algorithms in the [Netflix competition](http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/) or [KDD](https://medium.com/kaggle-blog) 2009.

**When we should consider the ensemble**

The main hypothesis of the ensemble is that; combining weak models or learners to obtain a more robust, precise, and reliable model. In this way, it will help to prevent overfitting and underfitting, enhances randomization, and improves model accuracy.

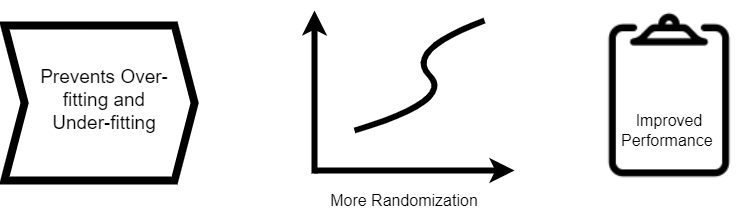


Figure 1. Need for Ensemble Methods

**Bias-Variance Trade-off**

The most important thing to achieve good results in machine learning problems like classification or regression is the selection of a model. This selection depends on many factors like distributive hypothesis, quality of dataset and dimensionality of space… etc.

The two characteristics that are frequently used to explain the errors made by ML algorithms are the Bias and the Variance.

The bias serves as a gauge of how accurately the model can represent the function that maps the inputs to the outputs. It is used to capture the rigidity of model/learners which is the strength of the model's presumption on the functional structure of the input-output mapping. Whereas the variance of the models is the degree to which model performance varies when it is fitted to various training datasets. It captures the details that the model has on the dataset. Every model's performance is connected by both bias and variance.

Ideally, a model having low bias and low variance is preferred. For a model to "solve" a problem, it should have the freedom to deal with the inherent complexities of the dataset, and also it should be robust enough to prevent excessive variance to make the model more resilient. This is known as the tradeoff between bias and variance.

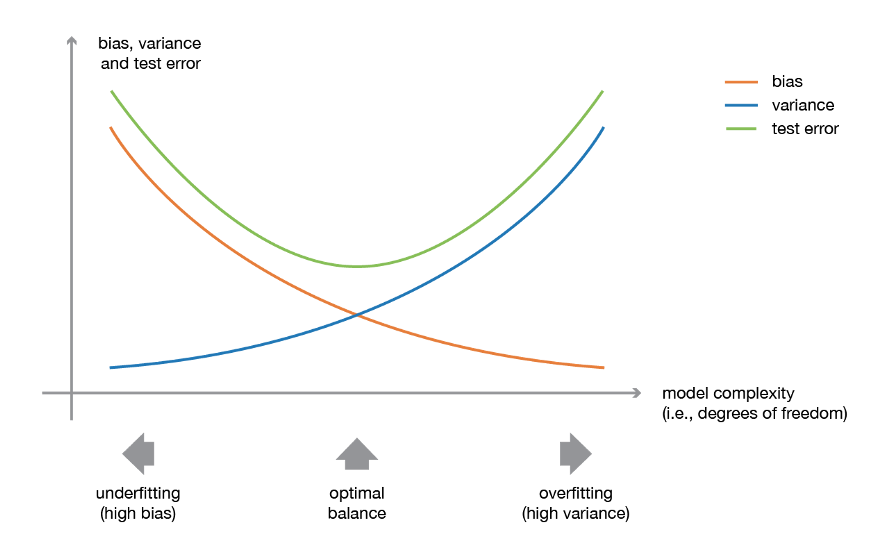


Figure 2. Illustration of Bias-Variance Trade-off ([source](https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205))

Weak learners (or base models) are most commonly used for ensemble learning as they are combined to create more complicated models. These fundamental weak models typically don't perform well on their own either because they have a strong bias (limited freedom models) or because they contain too much variance to be robust (more freedom models). To generate a strong ensemble learner that performs well, the approach aims to attempt and reduce the bias and/or variance of such weak learners by merging a number of them.

One important factor is the selection of weak models that should be consistent with the way aggregation is performed using these models. If we choose a model with high variance and low bias, the aggregating strategy should tend to minimize variance, and similarly, if we choose a model with high bias and low variance, the aggregating strategy should tend to minimize bias.  There are many popular ensemble methods, here two important models have been discussed in detail i.e. bagging and boosting.

**Bagging**

It is difficult to address the problems of ML in real life, especially when the performance accuracy on large training data is needed. In such scenarios, Bagging enters. The term "bagging" is also referred to as Bootstrap Aggregation.

Definition: Bagging is considered homogenous weak learning, it’s a type of ensemble learning in which a single training algorithm is used to independently learn on different training subsets. The subset sampling is done through bootstrap replacement algorithms. When all of the algorithms are trained the results are combined with the deterministic averaging process.

In reality, whether we are working on a classification or regression problem, we always map a function during the training process on the initial dataset that takes an input and provides us with an output. To train different subsets of the dataset, we use different models or mapping functions when talking about the bagging approach. However, these fitted models are susceptible to variability because of the theoretical variance present in the training dataset (remember that a dataset is an observed sample originating from a true unknown underlying distribution). The basic concept behind bagging is to fit a number of different models and "average" their prediction to get a model with a smaller variance.

But in reality, we are unable to construct entirely independent models to train on excessive data therefore we rely upon bootstraps samples in order to fit almost all independent models. For each of the bootstrap samples, we fit a weak learner so that we can aggregate their outputs in a way that the ensemble method will have less variance than its components.

To aggregate multiple models, there exist several approaches. For classification majority voting or hard voting is used when we need to determine the most frequent prediction among all predictions. Or soft-voting in which we consider the average of the probabilities of each class returned by all models and use the class having the highest average probability for prediction. For regression problems averaging process is used to determine the means of all output predictions. Averages or votes can either be simple or weighted unless any appropriate weights needed to be applied. Random forest is an example of bagging that uses Decision Trees (Deep tress) fitted to bootstrap samples and merged to provide an output with decreased variance.

Let us understand the bagging concept from an example. As you see different data samples are present in the training dataset. Replacement algorithms draw a random sample from training data. These samples can also be duplicated. Each sample is trained with different estimators or classifiers represented using Classifier 1, Classifier 2, …, and Classifier n. The final prediction is based on voting or averaging respectively. In last the performance of the trained model is tested using the test dataset.

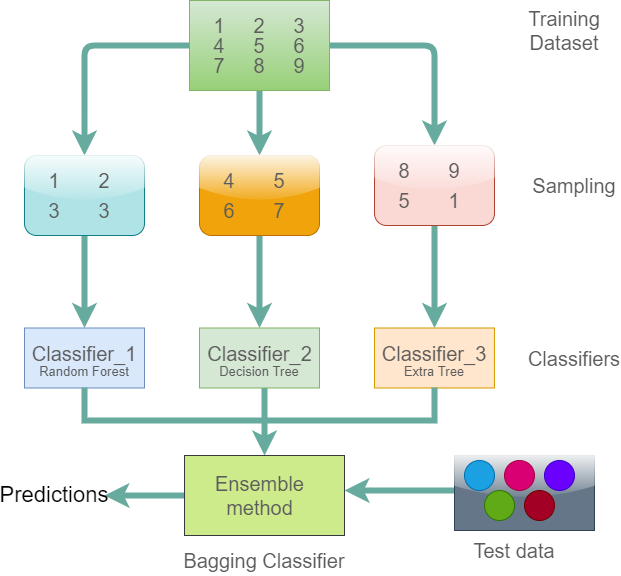


Figure 3. Illustration of Bagging Classifier

**Example Code:**

import numpy as np

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.pipeline import make\_pipeline

from sklearn.ensemble import BaggingClassifier

# Loading the dataset

BC = datasets.load\_breast\_cancer()

X = BC.data

Y = BC.target

# Create training and test split

X\_Train, X\_Test, y\_Train, y\_Test = train\_test\_split(X, Y, test\_size=0.25, random\_state=30, stratify=Y)

# Pipeline Estimator

Pipeline = make\_pipeline(StandardScaler(), LogisticRegression(random\_state=1))

# A model is fit using BaggingClassifier with base estimator as LogisticRegression

BGclassifier = BaggingClassifier(base\_estimator=Pipeline, n\_estimators=100,

max\_features=10,

max\_samples=100,

random\_state=30, n\_jobs=5)

# Fit the bagging classifier

BGclassifier.fit(X\_Train, y\_Train)

# Model scores on test and training data

print('Model test Score: %.3f, ' %BGclassifier.score(X\_Test, y\_Test),

'Model training Score: %.3f' %BGclassifier.score(X\_Train, y\_Train))

**Output:**

Model training Score: 0.965, Model test Score: 0.972

The major benefit of bagging is parallelizing bagging. Intense parallelization techniques can be applied if necessary because the various models are fitted independently from one another.

**Advantages:**

1. Works well when training set is small, but it also saves computational time when the dataset is large, by training the model on small subsets.
2. Bagging decreases variance without increasing bias.
3. Bagging also helps to deal with diverse datasets because of sampling through bootstrapping.

**Disadvantages:**

1. During bootstrapping, sampling bagging doesn’t predict features that are not selected in this way there exist a chance that certain type of features would never be used in training which results in the loss of important information.
2. Another disadvantage is the loss of interpretability when improving model accuracy e.g. if a single decision tree is used as weak model/leaners then this should be interpreted easily in the form of a diagram however this doesn’t happen in bagging.

**Boosting**

One can grasp the knowledge of boosting by comparing with bagging, as you have seen an ensemble is produced in bagging by taking several distinct samples from the training dataset and then fitting a decision tree to each of the samples. Each decision tree is unique since each sample in the training dataset is unique, which results in slightly varied predictions and prediction errors. Combining the predictions from each decision tree built reduces error compared to fitting a single tree.

Boosting works in a similar way. To get a better prediction than fitting a single decision tree, many trees are fitted on different variants of the training dataset, and the predictions from the trees are merged using averaging for regression and simple voting for classification.

Definition: boosting also frequently takes into account, homogenous weak learners. In this model learners learns data sequentially in a very adaptative manner before combining them in a deterministic manner.

In boosting every model in the sequence is fitted in a way to give more importance to the dataset observations that failed to be handled by the previous models in the sequence. We can say that in boosting every new model concentrates its efforts on the difficult observations to fit, in order to produce a powerful learner with less bias at the end of the process. The process is continued and models are added in this manner until either the whole training data set is successfully predicted or the maximum number of models is added. As our focus is on reducing bias, all base models that are taken into consideration for developing a boosting ensemble are those that have low variance but high bias. For instance, we would often select shallow decision trees with a limited number of depths if we wanted to utilize trees as our basic models.

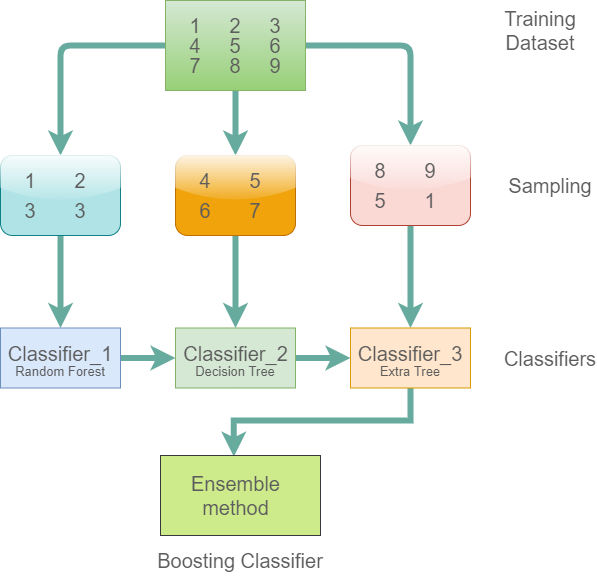


Figure 4. Illustration of Boosting Classifier

There are a few significant variations:

* Depending on the complexity, every instance in the training set is given a weight.
* Instance weights are an important consideration when determining learning algorithms.
* Ensemble learners are added sequentially.

**Advantages**

1. Prediction interpretations are easy to handle with boosting algorithms.
2. Boosting is a strong technique that readily reduces overfitting.

**Disadvantages**

1. Boosting is sensitive to outliers since every classifier is obliged to fix the errors of the predecessors.
2. As every estimator bases its correctness on the previous predictor this method is almost impossible to scale up thus making boosting difficult to streamline.

**Algorithms**

from sklearn import datasets

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.metrics import mean\_squared\_error

# Load the Boston Dataset

boston = datasets.load\_boston()

# Create Training and Test Split

X\_Train, X\_Test, y\_Train, y\_Test = train\_test\_split(boston.data, boston.target, random\_state=30, test\_size=0.25)

# Standardize the dataset

ss = StandardScaler()

X\_Train\_Std = ss.fit\_transform(X\_Train)

X\_Test\_Std = ss.transform(X\_Test)

# Hyperparameters for GradientBoostingRegressor

GBR\_params = {'n\_estimators': 1000,'max\_depth': 3,'min\_samples\_split': 5,'learning\_rate': 0.01, 'loss': 'ls'}

# Create an instance of gradient boosting regressor

GBR = GradientBoostingRegressor(\*\*GBR\_params)

# Fit the model

GBR.fit(X\_Train\_Std, y\_Train)

# Print Coefficient of determination R^2

print("Model Accuracy: %.3f" % GBR.score(X\_Test\_Std, y\_Test))

# Create the mean squared error

MSE = mean\_squared\_error(y\_Test, GBR.predict(X\_Test\_Std))

print("The mean squared error (MSE) on test set: {:.4f}".format(MSE))

**Output:**

Model Accuracy: 0.891

The mean squared error (MSE) on test set: 6.9844

**When Ensemble learning should not be used**

Ensemble learning causes a significant overhead and they are not the best option when high memory requirements, additional training and inference time requirements, or other computational costs cannot be afforded. Ten times as many resources are needed when utilizing an ensemble of ten models. The fact that the knowledge acquired by ensembles cannot be understood by the user is also one of the fundamental flaws in existing ensemble methods. The direction of increasing the [narrative coherence](https://www.lamda.nju.edu.cn/publication/aicom03.pdf) of ensembles is poorly explored. Another crucial issue is that currently there are no diversity metrics for ensemble learning while it is already said that [diversity](https://link.springer.com/article/10.1023/A:1022859003006) plays a vital role in ensembles. Ensemble learning will be able to contribute more to more applications if such problems can be successfully resolved.