Principal Component Analysis (PCA)

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are ***image processing, movie recommendation system, optimizing the power allocation in various communication channels.*** It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Principal Components in PCA

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Steps for PCA algorithm

1. **Getting the dataset**

   Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.

2. **Representing data into a structure**

   Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

3. **Standardizing the data**

   In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance. If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.

4. **Calculating the Covariance of Z**

   To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.

5. **Calculating the Eigen Values and Eigen Vectors**

Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

6. **Sorting the Eigen Vectors**

   In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

7. **Calculating the new features Or Principal Components**

   Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

8. **Remove less or unimportant features from the new dataset.**

   The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Applications of Principal Component Analysis

- PCA is mainly used as the dimensionality reduction technique in various AI applications such **as computer vision, image compression, etc.**
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

Code Reference(2 Datasets)

https://www.datacamp.com/tutorial/principal-component-analysis-in-python

A Gentle Introduction to Ensemble Learning Algorithms

Ensemble learning is a general meta approach to machine learning that seeks better predictive performance by combining the predictions from multiple models. Although there are a seemingly unlimited number of ensembles that you can develop for your predictive modeling problem, there are three methods that dominate the field of ensemble learning. So much so, that rather than algorithms per se, each is a field of study that has spawned many more specialized methods.
The three main classes of ensemble learning methods are **bagging**, **stacking**, and **boosting**, and it is important to both have a detailed understanding of each method and to consider them on your predictive modeling project.

A Gentle Introduction to Ensemble Learning Algorithms

by **Jason Brownlee** on April 27, 2021 in **Ensemble Learning 17**

Ensemble learning is a general meta approach to machine learning that seeks better predictive performance by combining the predictions from multiple models.
Although there are a seemingly unlimited number of ensembles that you can develop for your predictive modeling problem, there are three methods that dominate the field of ensemble learning. So much so, that rather than algorithms per se, each is a field of study that has spawned many more specialized methods.

The three main classes of ensemble learning methods are **bagging**, **stacking**, and **boosting**, and it is important to both have a detailed understanding of each method and to consider them on your predictive modeling project.

Standard Ensemble Learning Strategies

Ensemble learning refers to algorithms that combine the predictions from two or more models.

Although there is nearly an unlimited number of ways that this can be achieved, there are perhaps three classes of ensemble learning techniques that are most commonly discussed and used in practice. Their popularity is due in large part to their ease of implementation and success on a wide range of predictive modeling problems.

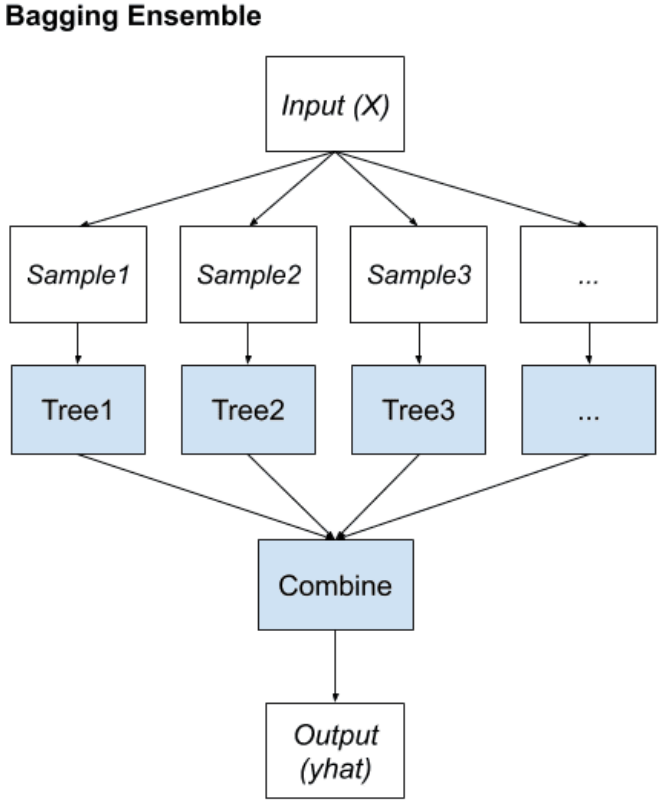
Given their wide use, we can refer to them as "*standard*" ensemble learning strategies; they are:

1. Bagging.
2. Stacking.
3. Boosting.

Bagging Ensemble Learning

Bootstrap aggregation, or bagging for short, is an ensemble learning method that seeks a diverse group of ensemble members by varying the training data. This typically involves using a single machine learning algorithm, almost always an unpruned decision tree, and training each model on a different sample of the same training dataset. The predictions made by the ensemble members are then combined using simple statistics, such as voting or averaging. Key to the method is the manner in which each sample of the dataset is prepared to train ensemble members. Each model gets its own unique sample of the dataset.
Replacement means that if a row is selected, it is returned to the training dataset for potential re-selection in the same training dataset. This means that a row of data may be selected zero, one, or multiple times for a given training dataset.

This is called a bootstrap sample. It is a technique often used in statistics with small datasets to estimate the statistical value of a data sample. By preparing multiple different bootstrap samples and estimating a statistical quantity and calculating the mean of the estimates, a better overall estimate of the desired quantity can be achieved than simply estimating from the dataset directly.



It is a general approach and easily extended. For example, more changes to the training dataset can be introduced, the algorithm fit on the training data can be replaced, and the mechanism used to combine predictions can be modified.

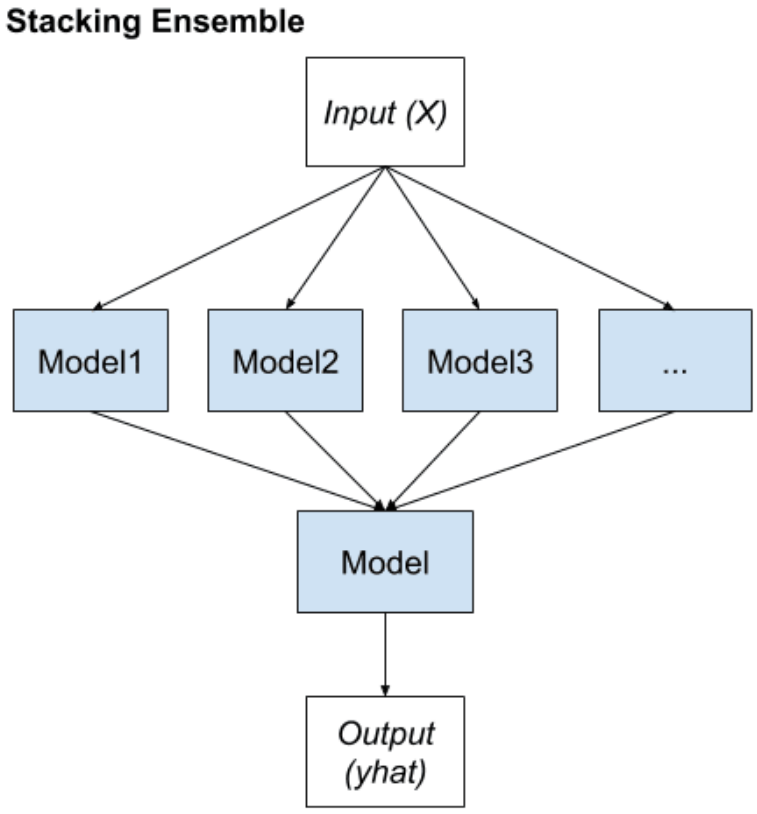
Many popular ensemble algorithms are based on this approach, including:

- Bagged Decision Trees (canonical bagging)
- Random Forest
- Extra Trees

Stacking Ensemble Learning

Stacked Generalization, or stacking for short, is an ensemble method that seeks a diverse group of members by varying the model types fit on the training data and using a model to combine predictions. Stacking has its own nomenclature where ensemble members are referred to as level-0 models and the model that is used to combine the predictions is referred to as a level-1 model.

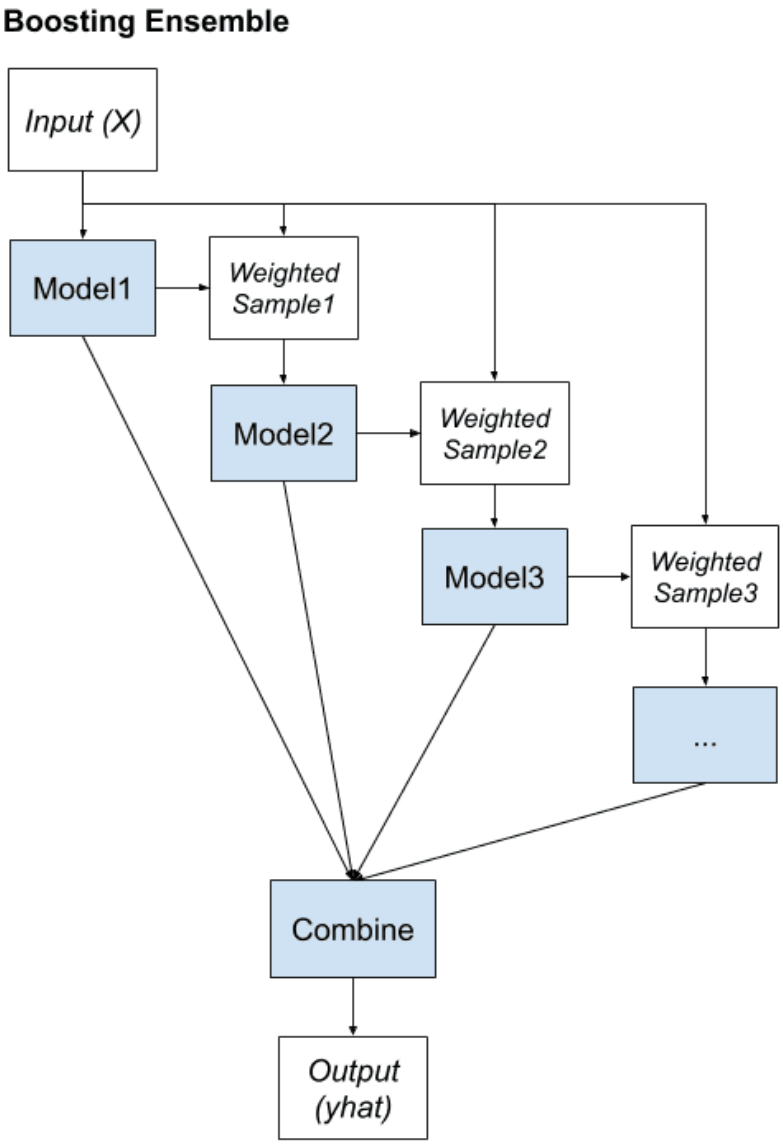
The two-level hierarchy of models is the most common approach, although more layers of models can be used. For example, instead of a single level-1 model, we might have 3 or 5 level-1 models and a single level-2 model that combines the predictions of level-1 models in order to make a prediction.



Boosting Ensemble Learning

Boosting is an ensemble method that seeks to change the training data to focus attention on examples that previous fit models on the training dataset have gotten wrong.
The key property of boosting ensembles is the idea of correcting prediction errors. The models are fit and added to the ensemble sequentially such that the second model attempts to correct the predictions of the first model, the third corrects the second model, and so on.

This typically involves the use of very simple decision trees that only make a single or a few decisions, referred to in boosting as weak learners. The predictions of the weak learners are combined using simple voting or averaging, although the contributions are weighed proportional to their performance or capability. The objective is to develop a so-called "*strong-learner*" from many purpose-built "*weak-learners*."



To summarize, many popular ensemble algorithms are based on this approach, including:

- AdaBoost (canonical boosting)
- Gradient Boosting Machines
- Stochastic Gradient Boosting (XGBoost and similar)

	Bagging	Boosting	Stacking
Purpose	Reduce Variance	Reduce Bias	Improve Accuracy
Base Learner Types	Homogeneous	Homogeneous	Heterogeneous
Base Learner Training	Parallel	Sequential	Meta Model
Aggregation	Max Voting, Averaging	Weighted Averaging	Weighted Averaging

When to use ensemble learning

You can employ ensemble learning techniques when you want to improve the performance of machine learning models. For example to increase the accuracy of classification models or to reduce the mean absolute error for regression models. Ensembling also results in a more stable model.

When your model is overfitting on the training set, you can also employ ensembling learning methods to create a more complex model. The models in the ensemble would then improve performance on the dataset by combining their predictions.

When ensemble learning works best

Ensemble learning works best when the base models are not correlated. For instance, you can train different models such as linear models, decision trees, and neural nets on different datasets or features. The less correlated the base models, the better.

The idea behind using uncorrelated models is that each may be solving a weakness of the other. They also have different strengths which, when combined, will result in a well-performing estimator. For example, creating an ensemble of just tree-based models may not be as effective as combining tree-type algorithms with other types of algorithms.

Code reference

https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

Simple Ensemble Techniques

1. Max Voting
2. Averaging
3. Weighted Averaging

Advanced Ensemble techniques

Stacking

Blending

Bagging

Boosting

References

https://www.geeksforgeeks.org/ensemble-classifier-data-mining/

https://neptune.ai/blog/ensemble-learning-guide