Rakamin Academy

HOME CREDIT

Virtual Internship Experience

Machine Learning

Supervised Learning,
Unsupervised Learning,
Hyperparameter Tuning.
Ensemble Methods.

What is Supervised Learning?

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Supervised learning helps organizations solve a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.

How supervised learning works

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.
Supervised learning can be separated into two types of problems when data mining—classification and regression:

- Classification uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.
- Regression is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

Supervised learning algorithms

Various algorithms and computation techniques are used in supervised machine learning processes. Below are brief explanations of some of the most commonly used learning methods, typically calculated through use of programs like R or Python:

A. Neural networks

Primarily leveraged for deep learning algorithms, neural networks process training data by mimicking the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold), and an output. If that output value exceeds a given threshold, it “fires” or activates the node, passing data to the next layer in the network. Neural networks learn this mapping function through supervised learning, adjusting based on the loss function through the process of gradient descent. When the cost function is at or near zero, we can be confident in the model’s accuracy to yield the correct answer.

B. Naive Bayes

Naive Bayes is a classification approach that adopts the principle of class conditional independence from the Bayes Theorem. This means that the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result. There are three types of Naïve Bayes classifiers: Multinomial Naïve Bayes, Bernoulli Naïve Bayes, and Gaussian Naïve Bayes. This technique is primarily used in text classification, spam identification, and recommendation systems.

C. Linear regression

Linear regression is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes. When there is only one independent variable and one dependent variable, it is known as simple linear regression. As the number of independent variables increases, it is referred to as multiple linear regression.

For each type of linear regression, it seeks to plot a line of best fit, which is calculated through the method of least squares. However, unlike other regression models, this line is straight when plotted on a graph.

D. Logistic regression

While linear regression is leveraged when dependent variables are continuous, logistic regression is selected when the dependent variable is categorical, meaning they have binary outputs, such as "true" and "false" or "yes" and "no." While both regression models seek to understand relationships between data inputs, logistic regression is mainly used to solve binary classification problems, such as spam identification.

E. Support vector machine (SVM)

A support vector machine is a popular supervised learning model developed by Vladimir Vapnik, used for both data classification and regression. That said, it is typically leveraged for classification problems, constructing a hyperplane where the distance between two classes of data points is at its maximum. This hyperplane is known as the decision boundary, separating the classes of data points (e.g., oranges vs. apples) on either side of the plane.

F. K-nearest neighbor

K-nearest neighbor, also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average.
Its ease of use and low calculation time make it a preferred algorithm by data scientists, but as the test dataset grows, the processing time lengthens, making it less appealing for classification tasks. KNN is typically used for recommendation engines and image recognition.

G. Random forest

Random forest is another flexible supervised machine learning algorithm used for both classification and regression purposes. The "forest" references a collection of uncorrelated decision trees, which are then merged together to reduce variance and create more accurate data predictions.

Supervised learning examples

Supervised learning models can be used to build and advance a number of business applications, including the following:

- Image- and object-recognition: Supervised learning algorithms can be used to locate, isolate, and categorize objects out of videos or images, making them useful when applied to various computer vision techniques and imagery analysis.
- Predictive analytics: A widespread use case for supervised learning models is in creating predictive analytics systems to provide deep insights into various business data points. This allows enterprises to anticipate certain results based on a given output variable, helping business leaders justify decisions or pivot for the benefit of the organization.
- Customer sentiment analysis: Using supervised machine learning algorithms, organizations can extract and classify important pieces of information from large volumes of data—including context, emotion, and intent—with very little human intervention. This can be incredibly useful when gaining a better understanding of customer interactions and can be used to improve brand engagement efforts.
- Spam detection: Spam detection is another example of a supervised learning model. Using supervised classification algorithms, organizations can train databases to recognize patterns or anomalies in new data to organize spam and non-spam-related correspondences effectively.

Challenges of supervised learning

Although supervised learning can offer businesses advantages, such as deep data insights and improved automation, there are some challenges when building sustainable supervised learning models. The following are some of these challenges:

- Supervised learning models can require certain levels of expertise to structure accurately.
- Training supervised learning models can be very time intensive.
- Datasets can have a higher likelihood of human error, resulting in algorithms learning incorrectly.
- Unlike unsupervised learning models, supervised learning cannot cluster or classify data on its own.

What is unsupervised learning?

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.

Common unsupervised learning approaches

Unsupervised learning models are utilized for three main tasks—clustering, association, and dimensionality reduction. Below we'll define each learning method and highlight common algorithms and approaches to conduct them effectively.

A. Clustering

Clustering is a data mining technique which groups unlabeled data based on their similarities or differences. Clustering algorithms are used to process raw, unclassified data objects into groups represented by structures or patterns in the information. Clustering algorithms can be categorized into a few types, specifically exclusive, overlapping, hierarchical, and probabilistic.

1. Exclusive and Overlapping Clustering

Exclusive clustering is a form of grouping that stipulates a data point can exist only in one cluster. This can also be referred to as “hard” clustering. The K-means clustering algorithm is an example of exclusive clustering.

- K-means clustering is a common example of an exclusive clustering method where data points are assigned into K groups, where K represents the number of clusters based on the distance from each group’s centroid. The data points closest to a given centroid will be clustered under the same category. A larger K value will be indicative of smaller groupings with more granularity whereas a smaller K value will have larger groupings and less granularity. K-means clustering is commonly used in market segmentation, document clustering, image segmentation, and image compression.

Overlapping clusters differs from exclusive clustering in that it allows data points to belong to multiple clusters with separate degrees of membership. “Soft” or fuzzy k-means clustering is an example of overlapping clustering.

2. Hierarchical clustering

Hierarchical clustering, also known as hierarchical cluster analysis (HCA), is an unsupervised clustering algorithm that can be categorized in two ways; they can be agglomerative or divisive. Agglomerative clustering is considered a “bottoms-up approach.” Its data points are isolated as separate groupings initially, and then they are merged together iteratively on the basis of similarity until one cluster has been achieved. Four different methods are commonly used to measure similarity:

1. Ward's linkage: This method states that the distance between two clusters is defined by the increase in the sum of squared after the clusters are merged.
2. Average linkage: This method is defined by the mean distance between two points in each cluster
3. Complete (or maximum) linkage: This method is defined by the maximum distance between two points in each cluster
4. Single (or minimum) linkage: This method is defined by the minimum distance between two points in each cluster

Euclidean distance is the most common metric used to calculate these distances; however, other metrics, such as Manhattan distance, are also cited in clustering literature.

Divisive clustering can be defined as the opposite of agglomerative clustering; instead it takes a “top-down” approach. In this case, a single data cluster is divided based on the differences between data points. Divisive clustering is not commonly used, but it is still worth noting in the context of hierarchical clustering. These clustering processes are usually visualized using a dendrogram, a tree-like diagram that documents the merging or splitting of data points at each iteration.

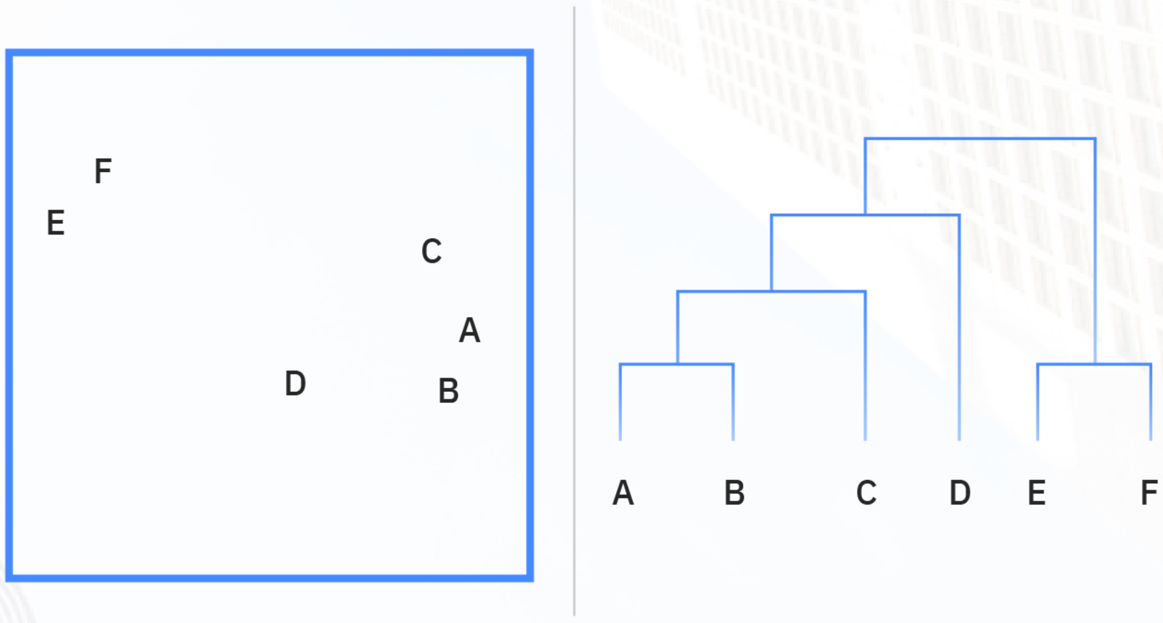
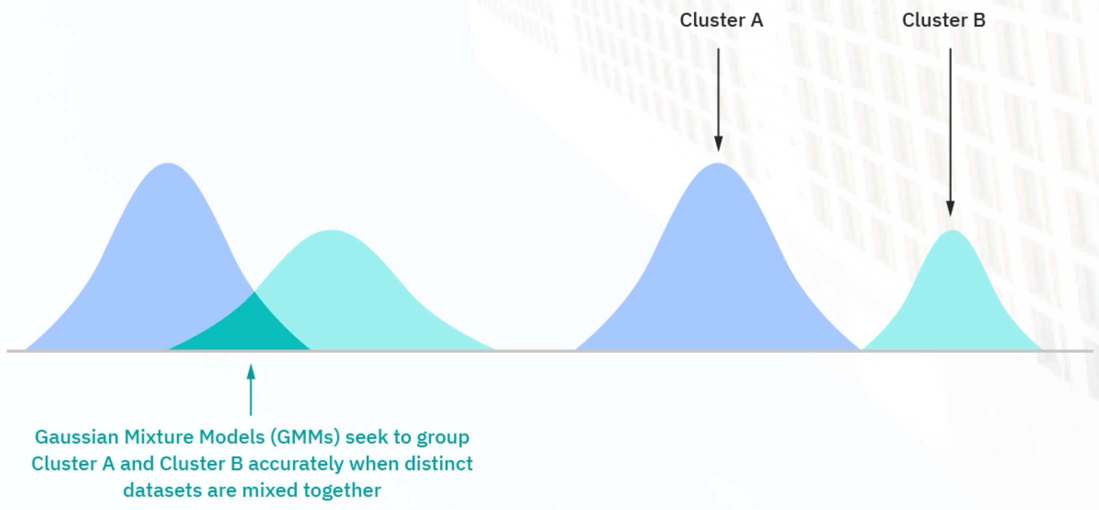


Diagram of a Dendrogram; reading the chart "bottom-up" demonstrates agglomerative clustering while "top-down" is indicative of divisive clustering.

3. Probabilistic clustering

A probabilistic model is an unsupervised technique that helps us solve density estimation or “soft” clustering problems. In probabilistic clustering, data points are clustered based on the likelihood that they belong to a particular distribution. The Gaussian Mixture Model (GMM) is the one of the most commonly used probabilistic clustering methods.

- Gaussian Mixture Models are classified as mixture models, which means that they are made up of an unspecified number of probability distribution functions. GMMs are primarily leveraged to determine which Gaussian, or normal, probability distribution a given data point belongs to. If the mean or variance are known, then we can determine which distribution a given data point belongs to. However, in GMMs, these variables are not known, so we assume that a latent, or hidden, variable exists to cluster data points appropriately. While it is not required to use the Expectation-Maximization (EM) algorithm, it is commonly used to estimate the assignment probabilities for a given data point to a particular data cluster.


Cluster A
Cluster B
Gaussian Mixture Models (GMMs) seek to group Cluster A and Cluster B accurately when distinct datasets are mixed together

B. Association Rules

An association rule is a rule-based method for finding relationships between variables in a given dataset. These methods are frequently used for market basket analysis, allowing companies to better understand relationships between different products. Understanding consumption habits of customers enables businesses to develop better cross-selling strategies and recommendation engines. Examples of this can be seen in Amazon's "Customers Who Bought This Item Also Bought" or Spotify's "Discover Weekly" playlist. While there are a few different algorithms used to generate association rules, such as Apriori, Eclat, and FP-Growth, the Apriori algorithm is most widely used.

1. Apriori algorithms

Apriori algorithms have been popularized through market basket analysis, leading to different recommendation engines for music platforms and online retailers. They are used within transactional datasets to identify frequent item sets, or collections of items, to identify the likelihood of consuming a product given the consumption of another product. For example, if I play Black Sabbath's radio on Spotify, starting with their song "Orchid", one of the other songs on this channel will likely be a Led Zeppelin song, such as "Over the Hills and Far Away." This is based on my prior listening habits as well as the ones of others. Apriori algorithms use a hash tree to count itemsets, navigating through the dataset in a breadth-first manner.

2. Dimensionality reduction

While more data generally yields more accurate results, it can also impact the performance of machine learning algorithms (e.g. overfitting) and it can also make it difficult to visualize datasets. Dimensionality reduction is a technique used when the number of features, or dimensions, in a given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the integrity of the dataset as much as possible. It is commonly used in the preprocessing data stage, and there are a few different dimensionality reduction methods that can be used, such as:

3. Principal component analysis

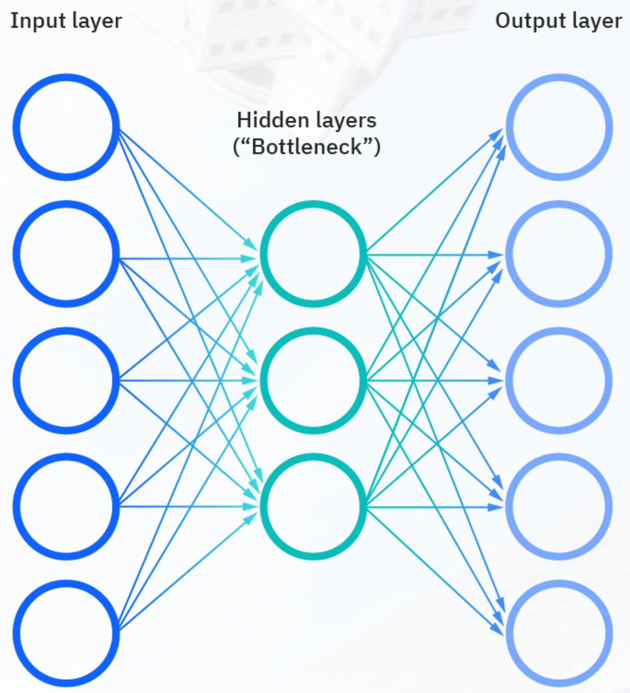
Principal component analysis (PCA) is a type of dimensionality reduction algorithm which is used to reduce redundancies and to compress datasets through feature extraction. This method uses a linear transformation to create a new data representation, yielding a set of "principal components." The first principal component is the direction which maximizes the variance of the dataset. While the second principal component also finds the maximum variance in the data, it is completely uncorrelated to the first principal component, yielding a direction that is perpendicular, or orthogonal, to the first component. This process repeats based on the number of dimensions, where the next principal component is the direction orthogonal to the prior components with the most variance.

4. Singular value decomposition

Singular value decomposition (SVD) is another dimensionality reduction approach which factorizes a matrix, A, into three, low-rank matrices. SVD is denoted by the formula, A = USVT, where U and V are orthogonal matrices. S is a diagonal matrix, and S values are considered singular values of matrix A. Similar to PCA, it is commonly used to reduce noise and compress data, such as image files.

5. Autoencoders

Autoencoders leverage neural networks to compress data and then recreate a new representation of the original data’s input. Looking at the image below, you can see that the hidden layer specifically acts as a bottleneck to compress the input layer prior to reconstructing within the output layer. The stage from the input layer to the hidden layer is referred to as “encoding” while the stage from the hidden layer to the output layer is known as “decoding.”



Applications of unsupervised learning

Machine learning techniques have become a common method to improve a product user experience and to test systems for quality assurance. Unsupervised learning provides an exploratory path to view data, allowing businesses to identify patterns in large volumes of data more quickly when compared to manual observation. Some of the most common real-world applications of unsupervised learning are:

- News Sections: Google News uses unsupervised learning to categorize articles on the same story from various online news outlets. For example, the results of a presidential election could be categorized under their label for "US" news.
- Computer vision: Unsupervised learning algorithms are used for visual perception tasks, such as object recognition.
- Medical imaging: Unsupervised machine learning provides essential features to medical imaging devices, such as image detection, classification and segmentation, used in radiology and pathology to diagnose patients quickly and accurately.
- Anomaly detection: Unsupervised learning models can comb through large amounts of data and discover atypical data points

within a dataset. These anomalies can raise awareness around faulty equipment, human error, or breaches in security.

- Customer personas: Defining customer personas makes it easier to understand common traits and business clients' purchasing habits. Unsupervised learning allows businesses to build better buyer persona profiles, enabling organizations to align their product messaging more appropriately.
- Recommendation Engines: Using past purchase behavior data, unsupervised learning can help to discover data trends that can be used to develop more effective cross-selling strategies. This is used to make relevant add-on recommendations to customers during the checkout process for online retailers.

Challenges of unsupervised learning

While unsupervised learning has many benefits, some challenges can occur when it allows machine learning models to execute without any human intervention. Some of these challenges can include:

- Computational complexity due to a high volume of training data
- Longer training times
- Higher risk of inaccurate results
- Human intervention to validate output variables
- Lack of transparency into the basis on which data was clustered

Unsupervised vs. supervised vs. semi-supervised learning

Unsupervised learning and supervised learning are frequently discussed together. Unlike unsupervised learning algorithms, supervised learning algorithms use labeled data. From that data, it either predicts future outcomes or assigns data to specific categories based on the regression or classification problem that it is trying to solve. While supervised learning algorithms tend to be more accurate than unsupervised learning models, they require upfront human intervention to label the data appropriately. However, these labeled datasets allow supervised learning algorithms to avoid computational complexity as they don’t need a large training set to produce intended outcomes.

Common regression and classification techniques are linear and logistic regression, naïve bayes, KNN algorithm, and random forest.
Semi-supervised learning occurs when only part of the given input data has been labeled. Unsupervised and semi-supervised learning can be more appealing alternatives as it can be time-consuming and costly to rely on domain expertise to label data appropriately for supervised learning.
For a deep dive into the differences between these approaches, check out "Supervised vs. Unsupervised Learning: What's the Difference?"

Summarize :

- In Supervised learning, you train the machine using data which is well "labeled."
- Unsupervised learning is a machine learning technique, where you do not need to supervise the model.
- Supervised learning allows you to collect data or produce a data output from the previous experience.
- Unsupervised machine learning helps you to find all kinds of unknown patterns in data.
- For example, you will be able to determine the time taken to reach back home based on weather conditions, Times of the day and holiday.
- For example, Baby can identify other dogs based on past supervised learning.
- Regression and Classification are two types of supervised machine learning techniques.
- Clustering and Association are two types of Unsupervised learning.
- In a supervised learning model, input and output variables will be given while with unsupervised learning model, only input data will be given

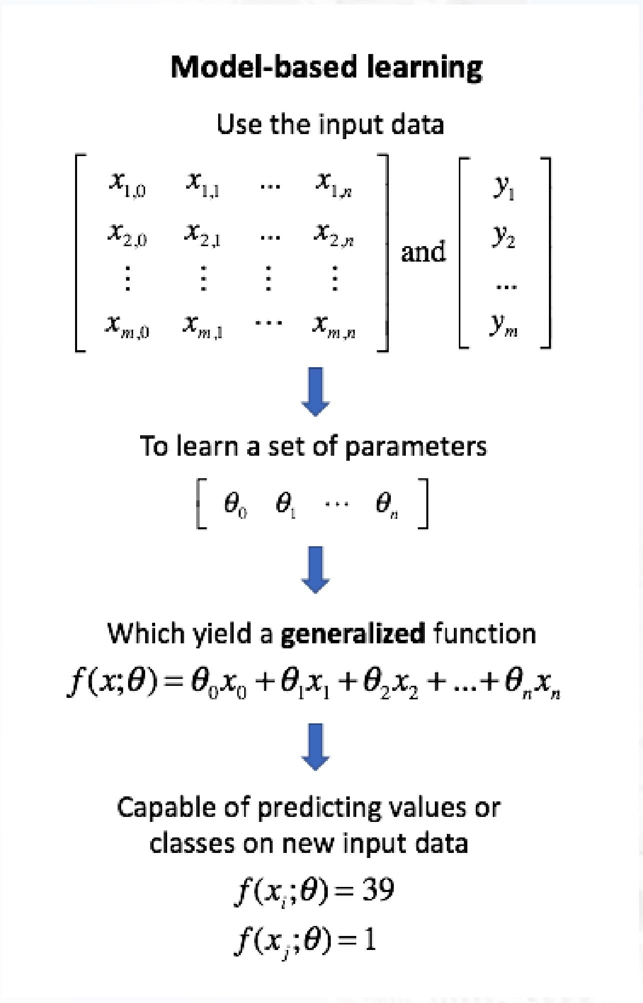
Hyperparameter Tuning

When creating a machine learning model, you'll be presented with design choices as to how to define your model architecture. Sometimes, we don't immediately know what the optimal model architecture should be for a given model, and thus we'd like to be able to explore a range of possibilities. In true machine learning fashion, we'll ideally ask the machine to perform this exploration and select the optimal model architecture automatically. Parameters which define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning.

These hyperparameters might address model design questions such as:

- What degree of polynomial features should I use for my linear model?
- What should be the maximum depth allowed for my decision tree?
- What should be the minimum number of samples required at a leaf node in my decision tree?
- How many trees should I include in my random forest?
- How many neurons should I have in my neural network layer?
- How many layers should I have in my neural network?
- What should I set my learning rate to for gradient descent?

I want to be absolutely clear, hyperparameters are not model parameters and they cannot be directly trained from the data. Model parameters are learned during training when we optimize a loss function using something like gradient descent.The process for learning parameter values is shown generally below.


Model-based learning
Use the input data
$\begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,n} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m,0} & x_{m,1} & \cdots & x_{m,n} \end{bmatrix}$ and $\begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{bmatrix}$
To learn a set of parameters
$\begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix}$
Which yield a **generalized** function
$f(x;\theta) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$
Capable of predicting values or classes on new input data
$f(x_i;\theta) = 39$
$f(x_j;\theta) = 1$

Whereas the model parameters specify how to transform the input data into the desired output, the hyperparameters define how our model is actually structured. Unfortunately, there's no way to calculate "which way should I update my hyperparameter to reduce the loss?" (ie. gradients) in order to find the optimal model architecture; thus, we generally resort to experimentation to figure out what works best.

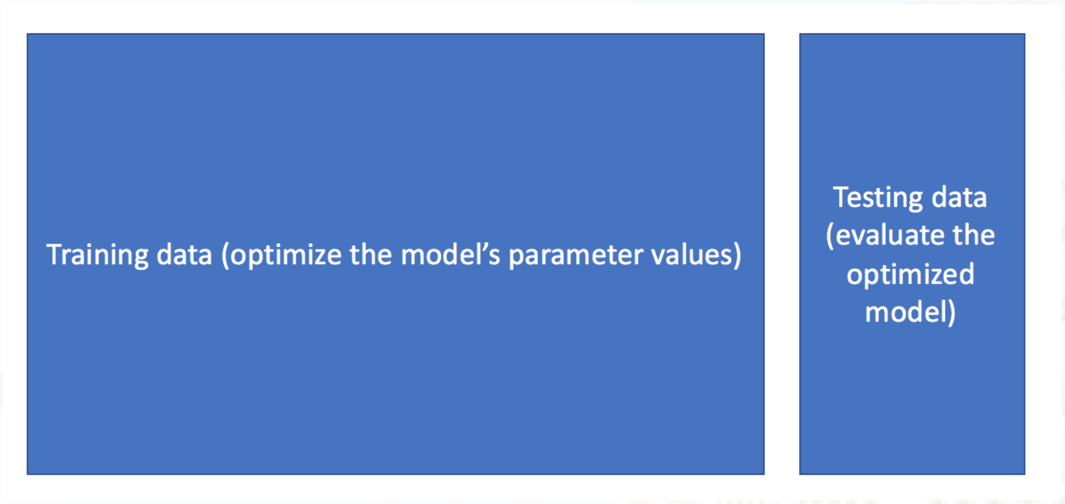
In general, this process includes:

1. Define a model
2. Define the range of possible values for all hyperparameters
3. Define a method for sampling hyperparameter values
4. Define an evaluative criteria to judge the model
5. Define a cross-validation method

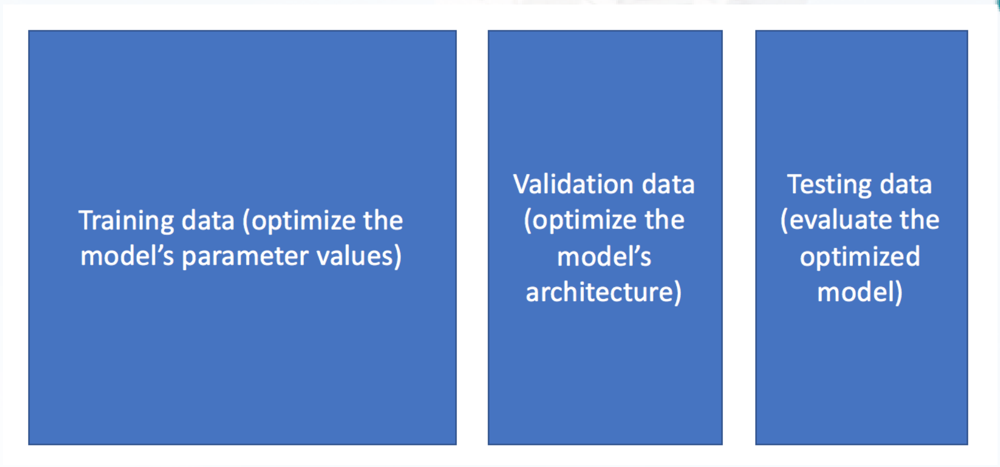
Specifically, the various hyperparameter tuning methods I'll discuss in this post offer various approaches to Step 3.

A. Model validation

Before we discuss these various tuning methods, I'd like to quickly revisit the purpose of splitting our data into training, validation, and test data. The ultimate goal for any machine learning model is to learn from examples in such a manner that the model is capable of generalizing the learning to new instances which it has not yet seen. At a very basic level, you should train on a subset of your total dataset, holding out the remaining data for evaluation to gauge the model's ability to generalize - in other words, "how well will my model do on data which it hasn't directly learned from during training?"


Training data (optimize the model's parameter values)
Testing data (evaluate the optimized model)

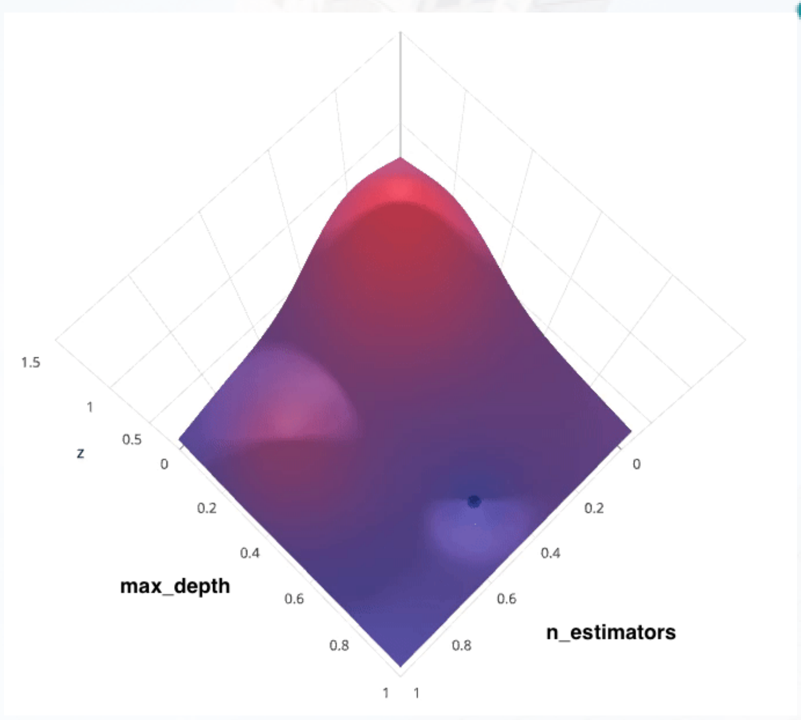
When you start exploring various model architectures (ie. different hyperparameter values), you also need a way to evaluate each model's ability to generalize to unseen data. However, if you use the testing data for this evaluation, you'll end up "fitting" the model architecture to the testing data - losing the ability to truly evaluate how the model performs on unseen data. This is sometimes referred to as "data leakage".
To mitigate this, we'll end up splitting the total dataset into three subsets: training data, validation data, and testing data. The introduction of a validation dataset allows us to evaluate the model on different data than it was trained on and select the best model architecture, while still holding out a subset of the data for the final evaluation at the end of our model development.


Training data (optimize the model's parameter values)
Validation data (optimize the model's architecture)
Testing data (evaluate the optimized model)

You can also leverage more advanced techniques such as K-fold cross validation in order to essentially combine training and validation data for both learning the model parameters and evaluating the model without introducing data leakage.

B. Hyperparameter tuning methods

Recall that I previously mentioned that the hyperparameter tuning methods relate to how we sample possible model architecture candidates from the space of possible hyperparameter values. This is often referred to as "searching" the hyperparameter space for the optimum values. In the following visualization, the x and y dimensions represent two hyperparameters, and the z dimension represents the model's score (defined by some evaluation metric) for the architecture defined by x and y.
Note: Ignore the axes values, I borrowed this image as noted and the axis values don't correspond with logical values for the hyperparameters.



If we had access to such a plot, choosing the ideal hyperparameter combination would be trivial. However, calculating such a plot at the granularity visualized above would be prohibitively expensive. Thus, we are left to blindly explore the hyperparameter space in hopes of locating the hyperparameter values which lead to the maximum score.
For each method, I'll discuss how to search for the optimal structure of a random forest classifier. Random forests are an ensemble model comprised of a collection of decision trees; when building such a model, two important hyperparameters to consider are:

- How many estimators (ie. decision trees) should I use?
- What should be the maximum allowable depth for each decision tree?

C. Grid search

Grid search is arguably the most basic hyperparameter tuning method. With this technique, we simply build a model for each possible combination of all of the hyperparameter values provided, evaluating each model, and selecting the architecture which produces the best results.

For example, we would define a list of values to try for both n_estimators and max_depth and a grid search would build a model for each possible combination.
Performing grid search over the defined hyperparameter space
n_estimators = [10, 50, 100, 200]
max_depth = [3, 10, 20, 40]

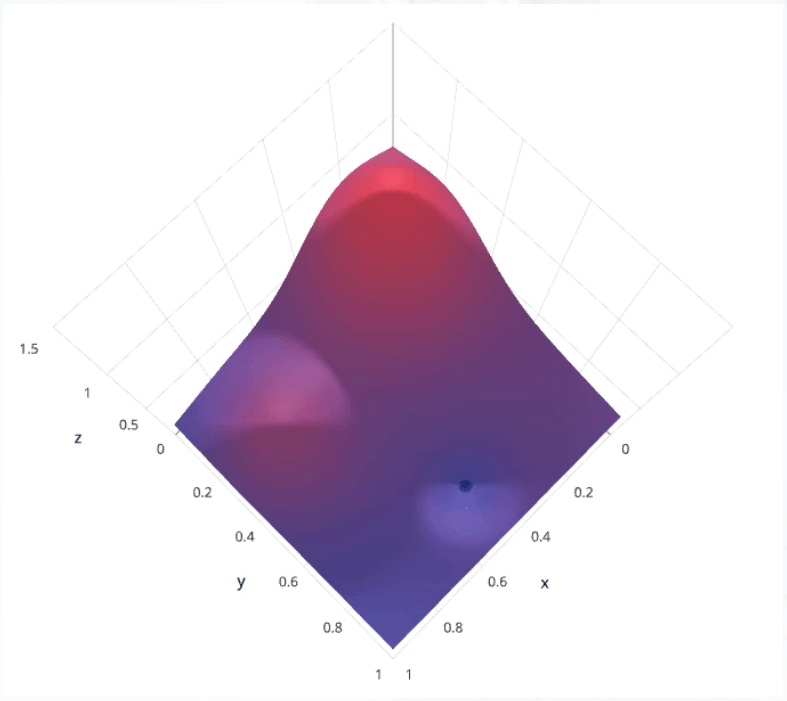
would yield the following models.
RandomForestClassifier(n_estimators=10, max_depth=3)
RandomForestClassifier(n_estimators=10, max_depth=10)
RandomForestClassifier(n_estimators=10, max_depth=20)
RandomForestClassifier(n_estimators=10, max_depth=40)

RandomForestClassifier(n_estimators=50, max_depth=3)
RandomForestClassifier(n_estimators=50, max_depth=10)
RandomForestClassifier(n_estimators=50, max_depth=20)
RandomForestClassifier(n_estimators=50, max_depth=40)

RandomForestClassifier(n_estimators=100, max_depth=3)
RandomForestClassifier(n_estimators=100, max_depth=10)
RandomForestClassifier(n_estimators=100, max_depth=20)
RandomForestClassifier(n_estimators=100, max_depth=40)

RandomForestClassifier(n_estimators=200, max_depth=3)
RandomForestClassifier(n_estimators=200, max_depth=10)
RandomForestClassifier(n_estimators=200, max_depth=20)
RandomForestClassifier(n_estimators=200, max_depth=40)

Each model would be fit to the training data and evaluated on the validation data. As you can see, this is an exhaustive sampling of the hyperparameter space and can be quite inefficient.



D. Random Search

Random search differs from grid search in that we longer provide a discrete set of values to explore for each hyperparameter; rather, we provide a statistical distribution for each hyperparameter from which values may be randomly sampled. We'll define a sampling distribution for each hyperparameter.

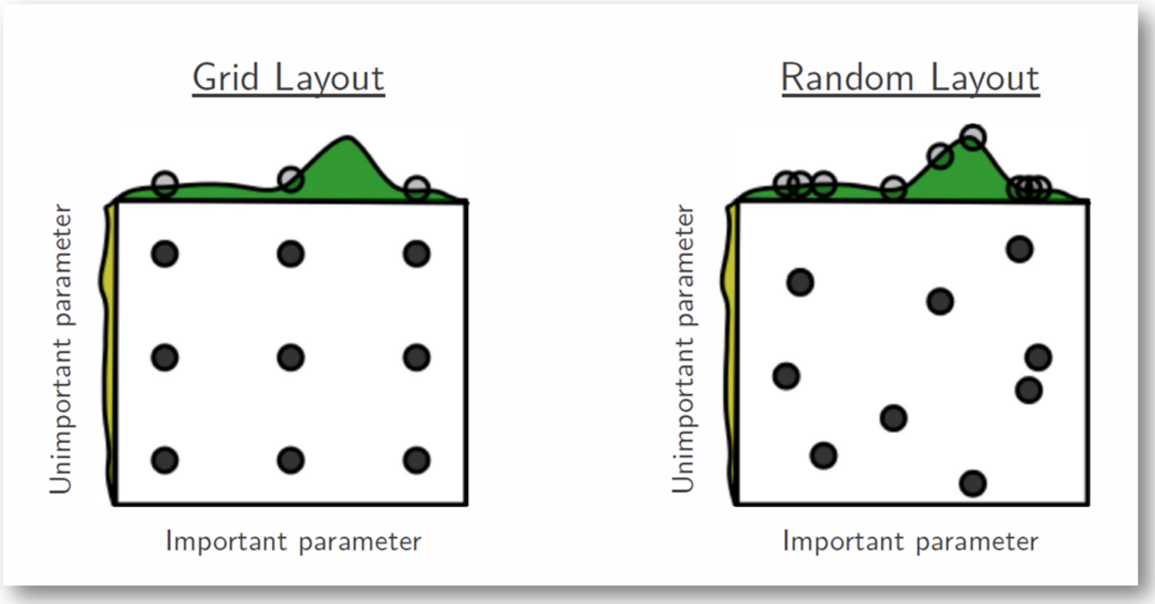
```
from scipy.stats import expon as sp_expon
from scipy.stats import randint as sp_randint

n_estimators = sp_expon(scale=100)
max_depth = sp_randint(1, 40)
```

We can also define how many iterations we'd like to build when searching for the optimal model. For each iteration, the hyperparameter values of the model will be set by sampling the defined distributions above. The scipy distributions above may be sampled with the rvs() function - feel free to explore this in Python!

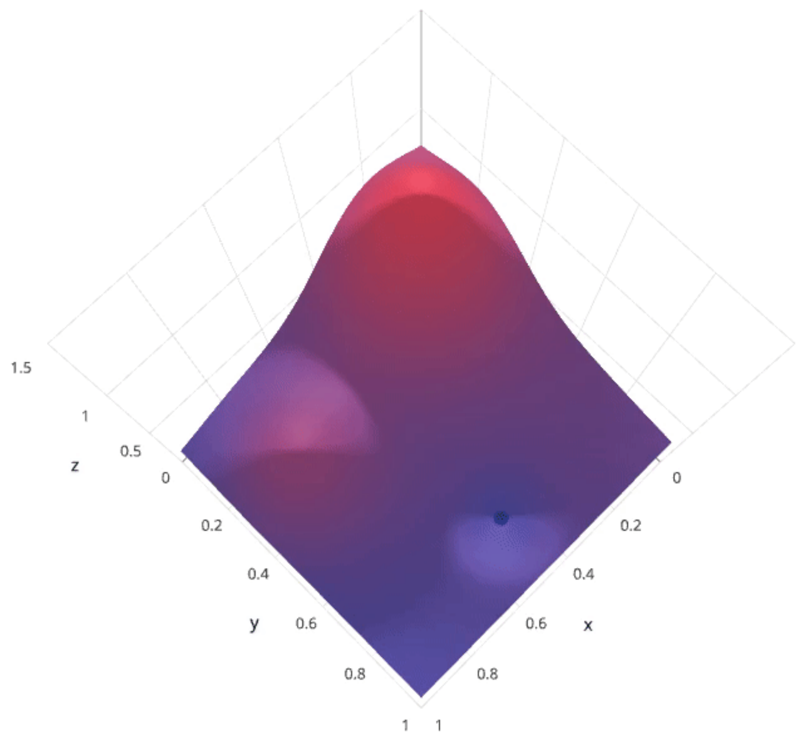
One of the main theoretical backings to motivate the use of random search in place of grid search is the fact that for most cases, hyperparameters are not equally important.

A Gaussian process analysis of the function from hyper-parameters to validation set performance reveals that for most data sets only a few of the hyper-parameters really matter, but that different hyper-parameters are important on different data sets. This phenomenon makes grid search a poor choice for configuring algorithms for new data sets. - Bergstra, 2012

In the following example, we're searching over a hyperparameter space where the one hyperparameter has significantly more influence on optimizing the model score - the distributions shown on each axis represent the model's score. In each case, we're evaluating nine different models. The grid search strategy blatantly misses the optimal model and spends redundant time exploring the unimportant parameter. During this grid search, we isolated each hyperparameter and searched for the best possible value while holding all other hyperparameters constant. For cases where the hyperparameter being studied has little effect on the resulting model score, this results in wasted effort. Conversely, the random search has much improved exploratory power and can focus on finding the optimal value for the important hyperparameter.



As you can see, this search method works best under the assumption that not all hyperparameters are equally important. While this isn't always the case, the assumption holds true for most datasets.

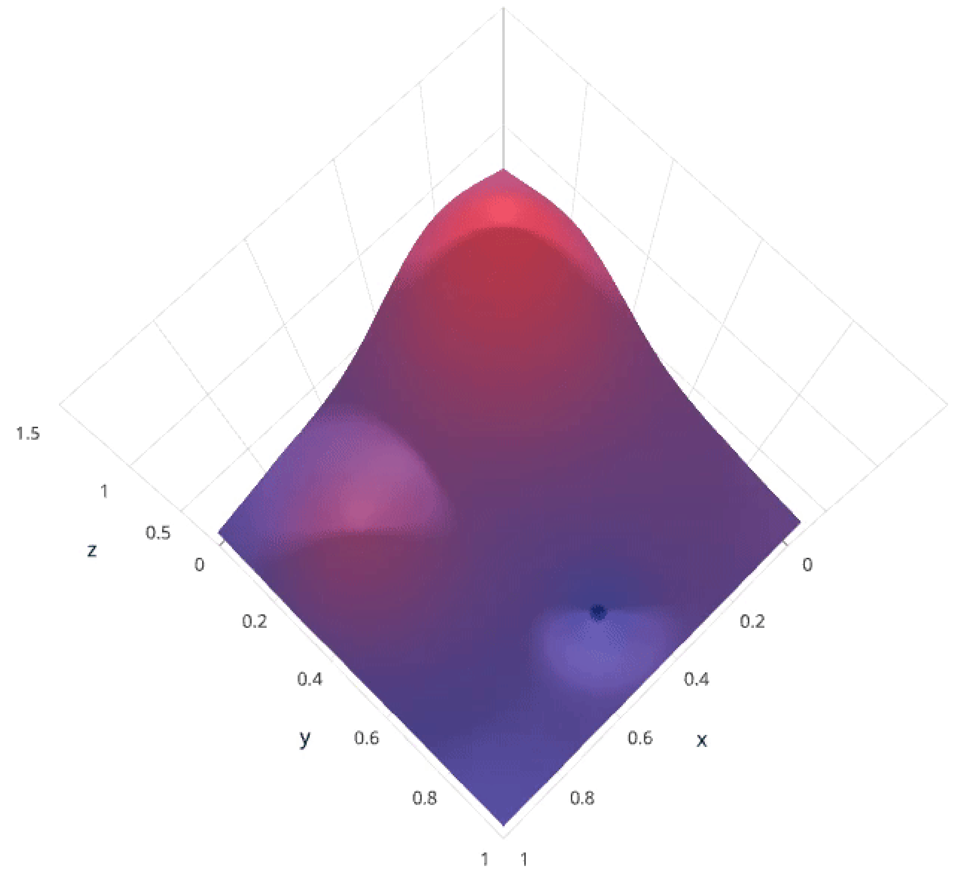


Bayesian optimization

The previous two methods performed individual experiments building models with various hyperparameter values and recording the model performance for each. Because each experiment was performed in isolation, it's very easy to parallelize this process. However, because each experiment was performed in isolation, we're not able to use the information from one experiment to improve the next experiment. Bayesian optimization belongs to a class of sequential model-based optimization (SMBO) algorithms that allow one to use the results of our previous iteration to improve our sampling method of the next experiment.

We'll initially define a model constructed with hyperparameters λ which, after training, is scored according to some evaluation metric. Next, we use the previously evaluated hyperparameter values to compute a posterior expectation of the hyperparameter space. We can then choose the optimal hyperparameter values according to this posterior expectation as our next model candidate. We iteratively repeat this process until converging to an optimum.

We'll use a Gaussian process to model our prior probability of model scores across the hyperparameter space. This model will essentially serve to use the hyperparameter values λ1,...i and corresponding scores V1,...i. we've observed thus far to approximate a continuous score function over the hyperparameter space. This approximated function also includes the degree of certainty of our estimate, which we can use to identify the candidate hyperparameter values that would yield the largest expected improvement over the current score. The formulation for expected improvement is known as our acquisition function, which represents the posterior distribution of our score function across the hyperparameter space.



What is bagging?

Bagging, also known as bootstrap aggregation, is the ensemble learning method that is commonly used to reduce variance within a noisy dataset. In bagging, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once. After several data samples are generated, these weak models are then trained independently, and depending on the type of task—regression or classification, for example—the average or majority of those predictions yield a more accurate estimate.
As a note, the random forest algorithm is considered an extension of the bagging method, using both bagging and feature randomness to create an uncorrelated forest of decision trees.

Ensemble learning

Ensemble learning gives credence to the idea of the "wisdom of crowds," which suggests that the decision-making of a larger group of people is typically better than that of an individual expert. Similarly, ensemble learning refers to a group (or ensemble) of base learners, or models, which work collectively to achieve a better final prediction. A single model, also known as a base or weak learner, may not perform well individually due to high variance or high bias. However, when weak learners are aggregated, they can form a strong learner, as their combination reduces bias or variance, yielding better model performance.
Ensemble methods are frequently illustrated using decision trees as this algorithm can be prone to overfitting (high variance and low bias) when it hasn't been pruned and it can also lend itself to underfitting (low variance and high bias) when it's very small, like a decision stump, which is a decision tree with one level. Remember, when an algorithm overfits or underfits to its training set, it cannot generalize well to new datasets, so ensemble methods are used to counteract this behavior to allow for generalization of the model to new datasets. While decision trees can exhibit high variance or high bias, it's worth noting that it is not the only

modeling technique that leverages ensemble learning to find the "sweet spot" within the bias-variance tradeoff.

How bagging works

In 1996, Leo Breiman (PDF, 829 KB) (link resides outside IBM) introduced the bagging algorithm, which has three basic steps:

1. Bootstrapping: Bagging leverages a bootstrapping sampling technique to create diverse samples. This resampling method generates different subsets of the training dataset by selecting data points at random and with replacement. This means that each time you select a data point from the training dataset, you are able to select the same instance multiple times. As a result, a value/instance repeated twice (or more) in a sample.
2. Parallel training: These bootstrap samples are then trained independently and in parallel with each other using weak or base learners.
3. Aggregation: Finally, depending on the task (i.e. regression or classification), an average or a majority of the predictions are taken to compute a more accurate estimate. In the case of regression, an average is taken of all the outputs predicted by the individual classifiers; this is known as soft voting. For classification problems, the class with the highest majority of votes is accepted; this is known as hard voting or majority voting.

Benefits and challenges of bagging

There are a number of key advantages and challenges that the bagging method presents when used for classification or regression problems. The key benefits of bagging include:

- Ease of implementation: Python libraries such as scikit-learn (also known as sklearn) make it easy to combine the predictions of base learners or estimators to improve model performance. Their documentation (link resides outside IBM) lays out the available modules that you can leverage in your model optimization.
- Reduction of variance: Bagging can reduce the variance within a learning algorithm. This is particularly helpful with

high-dimensional data, where missing values can lead to higher variance, making it more prone to overfitting and preventing accurate generalization to new datasets.

The key challenges of bagging include:

- Loss of interpretability: It's difficult to draw very precise business insights through bagging because of the averaging involved across predictions. While the output is more precise than any individual data point, a more accurate or complete dataset could also yield more precision within a single classification or regression model.
- Computationally expensive: Bagging slows down and grows more intensive as the number of iterations increases. Thus, it's not well-suited for real-time applications. Clustered systems or a large number of processing cores are ideal for quickly creating bagged ensembles on large test sets.
- Less flexible: As a technique, bagging works particularly well with algorithms that are less stable. One that is more stable or subject to high amounts of bias do not provide as much benefit as there's less variation within the dataset of the model. As noted in the Hands-On Guide to Machine Learning (link resides outside of IBM), "bagging a linear regression model will effectively just return the original predictions for large enough b."

Applications of Bagging

The bagging technique is used across a large number of industries, providing insights for both real-world value and interesting perspectives, such as in the GRAMMY Debates with Watson. Key use cases include:

- Healthcare: Bagging has been used to form medical data predictions. For example, research (PDF, 2.8 MB) (link resides outside IBM) shows that ensemble methods have been used for an array of bioinformatics problems, such as gene and/or protein selection to identify a specific trait of interest. More specifically, this research (link resides outside IBM) delves into its use to predict the onset of diabetes based on various risk predictors.

- IT: Bagging can also improve the precision and accuracy in IT systems, such as ones network intrusion detection systems. Meanwhile, this research (link resides outside IBM) looks at how bagging can improve the accuracy of network intrusion detection—and reduce the rates of false positives.
- Environment: Ensemble methods, such as bagging, have been applied within the field of remote sensing. More specifically, this research (link resides outside IBM) shows how it has been used to map the types of wetlands within a coastal landscape.
- Finance: Bagging has also been leveraged with deep learning models in the finance industry, automating critical tasks, including fraud detection, credit risk evaluations, and option pricing problems. This research (link resides outside IBM) demonstrates how bagging among other machine learning techniques have been leveraged to assess loan default risk. This study (link resides outside IBM) highlights how bagging helps to minimize risk by preventing credit card fraud within banking and financial institutions.

What is boosting?

Boosting is an ensemble learning method that combines a set of weak learners into a strong learner to minimize training errors. In boosting, a random sample of data is selected, fitted with a model and then trained sequentially—that is, each model tries to compensate for the weaknesses of its predecessor. With each iteration, the weak rules from each individual classifier are combined to form one, strong prediction rule. Before we go further, let's explore the category of ensemble learning more broadly, highlighting two of the most well-known methods: bagging and boosting.

Types of boosting

Boosting methods are focused on iteratively combining weak learners to build a strong learner that can predict more accurate outcomes. As a reminder, a weak learner classifies data slightly better than random guessing. This approach can provide robust results for prediction problems, and can even outperform neural networks and support vector machines for tasks like image retrieval (PDF, 1.9 MB) (link resides outside IBM).

Boosting algorithms can differ in how they create and aggregate weak learners during the sequential process. Three popular types of boosting methods include:

- Adaptive boosting or AdaBoost: Yoav Freund and Robert Schapire are credited with the creation of the AdaBoost algorithm. This method operates iteratively, identifying misclassified data points and adjusting their weights to minimize the training error. The model continues to optimize in a sequential fashion until it yields the strongest predictor.
- Gradient boosting: Building on the work of Leo Breiman, Jerome H. Friedman developed gradient boosting, which works by sequentially adding predictors to an ensemble with each one correcting for the errors of its predecessor. However, instead of changing weights of data points like AdaBoost, the gradient boosting trains on the residual errors of the previous predictor. The

name, gradient boosting, is used since it combines the gradient descent algorithm and boosting method.

- Extreme gradient boosting or XGBoost: XGBoost is an implementation of gradient boosting that's designed for computational speed and scale. XGBoost leverages multiple cores on the CPU, allowing for learning to occur in parallel during training.

Benefits and challenges of boosting

There are a number of key advantages and challenges that the boosting method presents when used for classification or regression problems.

The key benefits of boosting include:

- Ease of Implementation: Boosting can be used with several hyper-parameter tuning options to improve fitting. No data preprocessing is required, and boosting algorithms like have built-in routines to handle missing data. In Python, the scikit-learn library of ensemble methods (also known as sklearn.ensemble) makes it easy to implement the popular boosting methods, including AdaBoost, XGBoost, etc.
- Reduction of bias: Boosting algorithms combine multiple weak learners in a sequential method, iteratively improving upon observations. This approach can help to reduce high bias, commonly seen in shallow decision trees and logistic regression models.
- Computational Efficiency: Since boosting algorithms only select features that increase its predictive power during training, it can help to reduce dimensionality as well as increase computational efficiency.

The key challenges of boosting include:

- Overfitting: There's some dispute in the research (link resides outside IBM) around whether or not boosting can help reduce overfitting or exacerbate it. We include it under challenges because in the instances that it does occur, predictions cannot be generalized to new datasets.

- Intense computation: Sequential training in boosting is hard to scale up. Since each estimator is built on its predecessors, boosting models can be computationally expensive, although XGBoost seeks to address scalability issues seen in other types of boosting methods. Boosting algorithms can be slower to train when compared to bagging as a large number of parameters can also influence the behavior of the model.

Applications of boosting

Boosting algorithms are well suited for artificial intelligence projects across a broad range of industries, including:

- Healthcare: Boosting is used to lower errors in medical data predictions, such as predicting cardiovascular risk factors and cancer patient survival rates. For example, research (link resides outside IBM) shows that ensemble methods significantly improve the accuracy in identifying patients who could benefit from preventive treatment of cardiovascular disease, while avoiding unnecessary treatment of others. Likewise, another study (link resides out IBM) found that applying boosting to multiple genomics platforms can improve the prediction of cancer survival time.
- IT: Gradient boosted regression trees are used in search engines for page rankings, while the Viola-Jones boosting algorithm is used for image retrieval. As noted by Cornell (link resides outside IBM), boosted classifiers allow for the computations to be stopped sooner when it's clear in which way a prediction is headed. This means that a search engine can stop the evaluation of lower ranked pages, while image scanners will only consider images that actually contain the desired object.
- Finance: Boosting is used with deep learning models to automate critical tasks, including fraud detection, pricing analysis, and more. For example, boosting methods in credit card fraud detection and financial products pricing analysis (link resides outside IBM) improve the accuracy of analyzing massive data sets to minimize financial losses.

Bagging vs. boosting

Bagging and boosting are two main types of ensemble learning methods. As highlighted in this study (PDF, 248 KB) (link resides outside IBM), the main difference between these learning methods is the way in which they are trained. In bagging, weak learners are trained in parallel, but in boosting, they learn sequentially. This means that a series of models are constructed and with each new model iteration, the weights of the misclassified data in the previous model are increased. This redistribution of weights helps the algorithm identify the parameters that it needs to focus on to improve its performance. AdaBoost, which stands for "adaptive boosting algorithm," is one of the most popular boosting algorithms as it was one of the first of its kind. Other types of boosting algorithms include XGBoost, GradientBoost, and BrownBoost.

Another difference in which bagging and boosting differ are the scenarios in which they are used. For example, bagging methods are typically used on weak learners which exhibit high variance and low bias, whereas boosting methods are leveraged when low variance and high bias is observed.

Overview of stacking

Stacking mainly differs from bagging and boosting on two points. First stacking often considers heterogeneous weak learners (*different learning algorithms are combined*) whereas bagging and boosting consider mainly homogeneous weak learners. Second, stacking learns to combine the base models using a meta-model whereas bagging and boosting combine weak learners following deterministic algorithms.

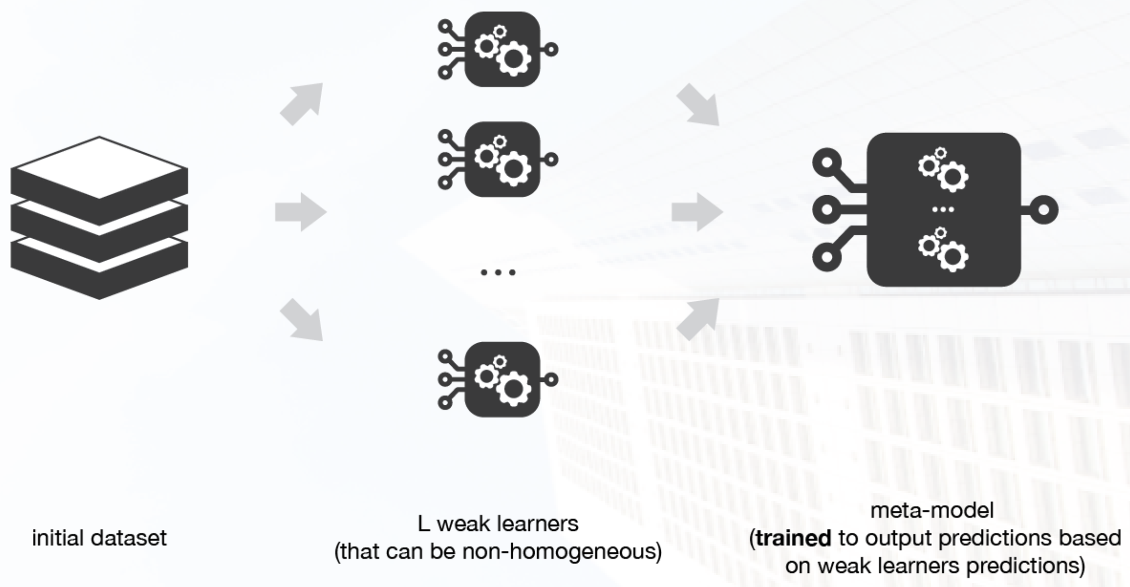
Stacking

As we already mentioned, the idea of stacking is to learn several different weak learners and combine them by training a meta-model to output predictions based on the multiple predictions returned by these weak models. So, we need to define two things in order to build our stacking model: the L learners we want to fit and the meta-model that combines them. For example, for a classification problem, we can choose as weak learners a KNN classifier, a logistic regression and a SVM, and decide to learn a neural network as a meta-model. Then, the neural network will take as inputs the outputs of our three weak learners and will learn to return final predictions based on it.

So, assume that we want to fit a stacking ensemble composed of L weak learners. Then we have to follow the steps thereafter:

- split the training data in two folds
- choose L weak learners and fit them to data of the first fold
- for each of the L weak learners, make predictions for observations in the second fold
- fit the meta-model on the second fold, using predictions made by the weak learners as inputs

In the previous steps, we split the dataset in two folds because predictions on data that have been used for the training of the weak learners are not relevant for the training of the meta-model. Thus, an obvious drawback of this split of our dataset in two parts is that we only have half of the data to train the base models and half of the data to train the meta-model. In order to overcome this limitation, we can however follow some kind of “k-fold cross-training” approach (similar to what is done in k-fold cross-validation) such that all the observations can be used to train the meta-model: for any observation, the prediction of the weak learners are done with instances of these weak learners trained on the k-1 folds that do not contain the considered observation. In other words, it consists in training on k-1 fold in order to make predictions on the remaining fold and that iteratively so that to obtain predictions for observations in any fold. Doing so, we can produce relevant predictions for each observation of our dataset and then train our meta-model on all these predictions.

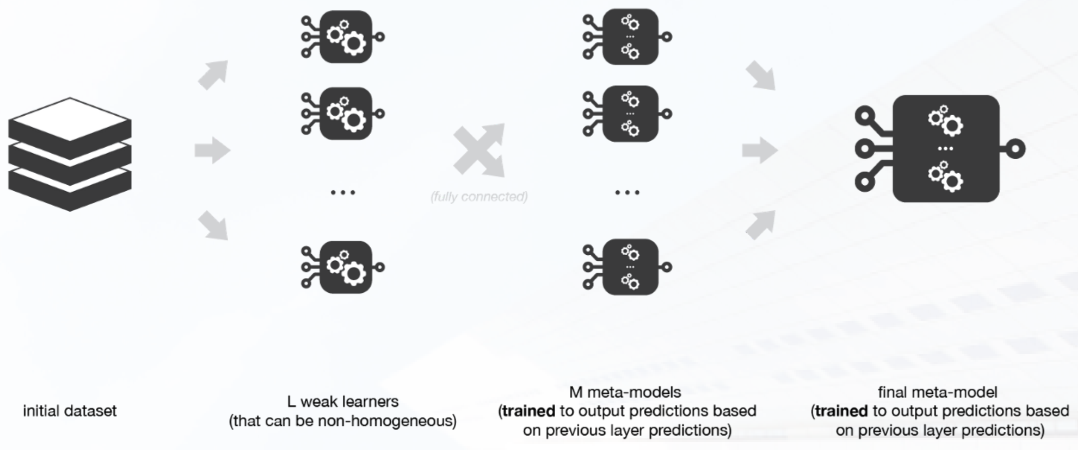

initial dataset
L weak learners
(that can be non-homogeneous)
meta-model
(**trained** to output predictions based on weak learners predictions)

Stacking consists in training a meta-model to produce outputs based on the outputs returned by some lower layer weak learners.

Multi-levels Stacking

A possible extension of stacking is multi-level stacking. It consists in doing stacking with multiple layers. As an example, let's consider 3-levels stacking. In the first level (layer), we fit the L weak learners that have been chosen. Then, in the second level, instead of fitting a single meta-model on the weak models predictions (as it was described in the previous subsection) we fit M such meta-models. Finally, in the third level we fit a last meta-model that takes as inputs the predictions returned by the M meta-models of the previous level.

From a practical point of view, notice that for each meta-model of the different levels of a multi-levels stacking ensemble model, we have to choose a learning algorithm that can be almost whatever we want (even algorithms already used at lower levels). We can also mention that adding levels can either be data expensive (if k-folds like technique is not used and, then, more data are needed) or time expensive (if k-folds like technique is used and, then, a lot of models need to be fitted).


(fully connected)
initial dataset
L weak learners
(that can be non-homogeneous)
M meta-models
(**trained** to output predictions based on previous layer predictions)
final meta-model
(**trained** to output predictions based on previous layer predictions)

Multi-level stacking considers several layers of stacking: some meta-models are trained on outputs returned by lower layer meta-models and so on. Here we have represented a 3-layers stacking model.

References :

- Joseph Rocca, "Ensemble methods: Bagging, Boosting and Stacking" from https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 [Acessed: 15/04/2022]
- Jeremy Jordan, "Hyperparameter tuning for machine learning models" from https://www.jeremyjordan.me/hyperparameter-tuning/ [Acessed : 15/04/2022]
- IBM Cloud Education, "Bagging" from https://www.ibm.com/cloud/learn/bagging?mhsrc=ibmsearch_a&mhq=ENSEMBLE [Acessed: 15/04/2022].
- IBM Cloud Educaton, "Boosting" from https://www.ibm.com/cloud/learn/boosting?mhsrc=ibmsearch_a&mhq=ENSEMBLE [Acessed : 15/04/2022].
- IBM Cloud Education, "Unsuverpised Learning" from https://www.ibm.com/cloud/learn/unsupervised-learning?mhsrc=ibmsearch_a&mhq=UNSUPERVISED [Acessed : 15/04/2022].