Here are a few common machine learning algorithms:

* Linear Regression
* Logistic Regression
* Decision Trees
* Random Forest
* Gradient Boosting
* K-Means Clustering
* Principal Component Analysis (PCA)
* Support Vector Machines (SVMs)
* Neural Networks (including deep learning)
* Naive Bayes

This is not an exhaustive list and there are many other algorithms available depending on the use case and the type of data.

**Linear Regression**

Linear regression is a supervised learning algorithm used for predicting a continuous variable. It assumes that there is a linear relationship between the input features (also known as predictors or independent variables) and the output variable (also known as the response or dependent variable). The goal of linear regression is to find the best-fitting line that minimizes the difference between the predicted values and the actual values. Linear regression can be simple or multiple depending on the number of input features. Simple linear regression is used when there is one input feature, and multiple linear regression is used when there are multiple input features. It is a widely used algorithm for its simplicity and interpretability.

**The equation for a simple linear regression model is:**

y = beta\_0 + beta\_1 \* x

where: y is the predicted output variable (dependent variable) x is the input feature (independent variable) beta\_0 is the y-intercept (the point where the line crosses the y-axis) beta\_1 is the coefficient for x (the slope of the line)

For multiple linear regression, the equation would be:

y = beta\_0 + beta\_1 \* x1 + beta\_2 \* x2 + ... + beta\_n \* xn

Where x1, x2, ..., xn are the input features and beta\_1, beta\_2, ..., beta\_n are the coefficients for each feature. The goal of the multiple linear regression is to find the optimal values for the coefficients (beta\_0, beta\_1, ..., beta\_n) that minimize the difference between the predicted values and the actual values.

**Logistic Regression**

Logistic Regression is a supervised learning algorithm used for classification problems. The goal of logistic regression is to find the best-fitting model that describes the relationship between the input features and the output variable, where the output variable is binary (i.e. it can take on only two values, such as 0 or 1). The algorithm works by estimating the probability that the input features belong to a particular class. The predicted output class is then determined by a threshold value, usually 0.5, above which the input is classified as one class and below which it is classified as the other class.

The basic equation for logistic regression is:

p(y=1|x) = 1 / (1 + e^(-z))

where: p(y=1|x) is the probability that the input features x belong to class 1, given the input features x. z is the log-odds, which is computed as: z = beta\_0 + beta\_1 \* x1 + beta\_2 \* x2 + ... + beta\_n \* xn beta\_0, beta\_1, beta\_2, ..., beta\_n are the coefficients for each feature, which are learned by the algorithm during the training process.

This is a general equation, logistic regression can also be used for multi-class classification problems where the output variable can take on more than two values. In this case, it's called multinomial logistic regression.

**Decision Trees**

A Decision Tree is a supervised learning algorithm used for both classification and regression problems. It is a tree-based model that is used to make predictions by recursively partitioning the data into subsets based on the values of the input features.

Each internal node of the tree represents a feature or an attribute, and each leaf node represents a class label or a value. The decision of which feature to split on is made by selecting the feature that maximizes the reduction in impurity (or the increase in information gain) at each step.

The basic idea behind decision trees is to divide the data into smaller subsets, called branches, where each subset contains data points with similar characteristics. The process of splitting the data continues until the leaf nodes contain only data points of the same class or until a stopping criterion is met.

Once the decision tree is built, the prediction is made by traversing the tree from the root node to the leaf node, following the path determined by the values of the input features.

Decision trees are widely used for their interpretability and ease of use, they are also a fundamental building block for other ensemble methods such as Random Forests and Boosting which have better performance and generalization capabilities.

A decision tree algorithm doesn't have a specific equation like linear or logistic regression. Instead, it uses a set of rules that are learned from the data in the form of if-then-else statements. These rules are used to partition the data into subsets and make predictions.

The main objective of a decision tree algorithm is to find the best split of the data at each node, which is done by evaluating different splitting criteria such as Gini impurity, information gain, or gain ratio. These criteria are used to measure the homogeneity of the data within a group and the difference between different groups.

For example, Gini impurity is calculated as:

Gini Impurity = 1 - (p1^2 + p2^2 + ... + pn^2) where pi is the proportion of class i in the group.

Information Gain is calculated as:

Information Gain = Entropy(Parent) - [Weighted average] \* Entropy(Children)

where Entropy is a measure of the impurity of the data, and the weighted average is the average impurity of the children groups, weighted by their size.

The split that results in the highest reduction in impurity (or the highest increase in information gain) is chosen as the best split. The process of building the tree continues until a stopping criterion is met, such as a maximum tree depth, a minimum number of samples in a leaf node, or the absence of any useful splits.

Once the decision tree is built, the prediction is made by traversing the tree from the root node to the leaf node, following the path determined by the values of the input features. The class label or the value of the data point at the leaf node is then returned as the prediction.

**Random Forest**

Random Forest is an ensemble learning method for classification and regression. It is an extension of decision trees that builds multiple decision trees and combines their predictions to make a final prediction.

The basic idea behind Random Forest is to create multiple decision trees by selecting a random subset of the data and a random subset of the features at each node. This process is repeated many times to create a large number of decision trees, also known as the forest. The final prediction is made by averaging the predictions of all the trees in the forest, for regression problems or by taking the mode of the predictions for classification problems.

The randomness in the selection of data and features at each node helps to decorrelate the trees and avoid overfitting, which is a common problem with decision trees. This results in a more robust and accurate model.

Random Forest can be used for both classification and regression problems and it's particularly useful for high dimensional datasets, where it is able to handle thousands of input features.

The Random Forest algorithm is relatively easy to use and requires little parameter tuning, it also have the advantage of being able to give an estimate of feature importance.

Random Forest is an ensemble method, it doesn't have a specific formula like linear or logistic regression. Instead, it builds multiple decision trees and combines their predictions to make a final prediction.

The basic steps in building a Random Forest are:

1. Select a random subset of the data to train each decision tree.
2. At each node of the decision tree, select a random subset of the features to consider for the split.
3. Repeat steps 1 and 2 a specified number of times to build a forest of decision trees.
4. For a new input, make a prediction by taking the average of the predictions of all the decision trees in the forest for regression problems, or take the mode of the predictions for classification problems.

Random Forest algorithm does not require a specific formula, it uses the decision tree algorithm that was previously described. However, randomness is added by selecting the random subset of data and features at each node which helps to decorrelate the decision trees and avoid overfitting, resulting in a more robust and accurate model.

**Gradient Boosting**

Gradient Boosting is an ensemble learning method for classification and regression. It is a boosting method that uses a set of weak learners (such as decision trees) and combines their predictions to make a final prediction.

The basic idea behind Gradient Boosting is to train the weak learners sequentially, where each learner tries to correct the mistakes of the previous learner. The final prediction is made by combining the predictions of all the learners.

The algorithm begins by training a weak learner on the original dataset. Then, it iteratively trains additional weak learners, each time focusing on the samples that were misclassified by the previous learner. The predictions of the weak learners are then combined through a weighted voting scheme, where the weight of each learner is determined by its performance on the training set.

Gradient Boosting can be used for both classification and regression problems. It's particularly useful for datasets with complex patterns and non-linear relationships.

Gradient Boosting algorithms such as XGBoost and LightGBM are widely used in Kaggle competitions and other data science projects for their good performance and generalization capabilities.

Gradient Boosting is an ensemble method and it doesn't have a specific formula like linear or logistic regression. Instead, it uses a set of weak learners (such as decision trees) and combines their predictions to make a final prediction.

However, Gradient Boosting algorithm use a specific loss function that is minimized during the training process. The loss function measures the difference between the predicted values and the true values. The most commonly used loss functions for Gradient Boosting are:

* Mean Squared Error (MSE) for regression problems.
* Binary Cross-Entropy for binary classification problems.
* Multi-Class Cross-Entropy for multi-class classification problems.

The basic steps in building a Gradient Boosting model are:

1. Initialize the output of the first weak learner to be the average of the target values.
2. For each iteration, fit a weak learner to the negative gradient of the loss function with respect to the current output.
3. Update the final prediction by adding the weighted prediction of the weak learner.
4. Repeat steps 2 and 3 for a specified number of iterations or until a stopping criterion is met.

The final prediction is made by combining the predictions of all the weak learners through a weighted voting scheme, where the weight of each learner is determined by its performance on the training set.

Gradient Boosting algorithm uses the decision tree as a weak learner, but it also can use other weak learners such as linear regression, and the specific formula depends on the type of the weak learner.

**Naive Bayes**

Naive Bayes is a supervised learning algorithm used for classification problems. It is a probabilistic algorithm that is based on Bayes' theorem, which states that the probability of an event occurring is equal to the prior probability of the event multiplied by the likelihood of the event given certain conditions.

Naive Bayes assumes that all the features of the input data are independent of each other, which is known as the "Naive" assumption. Despite this assumption, Naive Bayes is known to work well in practice, particularly for text classification and spam filtering.

There are three main types of Naive Bayes:

1. Gaussian Naive Bayes: This algorithm assumes that the data is normally distributed and is used for continuous data.
2. Multinomial Naive Bayes: This algorithm is used for discrete data such as text classification, where the data is represented as a count of occurrences of each word.
3. Bernoulli Naive Bayes: This algorithm is similar to Multinomial Naive Bayes, but it is used for binary data, where the data is represented as the presence or absence of a feature.

The prediction is made by computing the probability of the input data belonging to each class and selecting the class with the highest probability.

Naive Bayes is a fast and simple algorithm, it also has the advantage of being able to work with high dimensional data and requires little data preprocessing, it also is not sensitive to irrelevant features.

Naive Bayes is a probabilistic algorithm that is based on Bayes' theorem, which states that the probability of an event occurring (in this case, the probability that a given input belongs to a certain class) is equal to the prior probability of the event multiplied by the likelihood of the event given certain conditions (in this case, the probability of the input features given the class).

Formula for Bayes' theorem:

P(A|B) = P(B|A) \* P(A) / P(B)

Where: P(A|B) is the probability of event A occurring given that event B has occurred. (in this case, the probability that the input belongs to class A given the input features) P(B|A) is the likelihood of event B occurring given that event A has occurred. (in this case, the probability of the input features given that the input belongs to class A) P(A) is the prior probability of event A. (in this case, the prior probability of the input belonging to class A) P(B) is the prior probability of event B. (in this case, the prior probability of the input features)

The steps in building a Naive Bayes model are:

1. Collect and preprocess the data.
2. Estimate the prior probabilities of each class, P(A)
3. Estimate the likelihood of the input features given each class, P(B|A)
4. Use Bayes' theorem to calculate the probability of each class given the input features, P(A|B)
5. Select the class with the highest probability as the prediction

The Naive assumption states that all the features of the input data are independent of each other, given the class, which is used to simplify the computation of the likelihood.

It's worth noting that for some types of Naive Bayes, such as Gaussian and Bernoulli, the likelihood is estimated using probability density functions or probability mass functions, which are specific to the type of data.

Hybrid machine learning models are models that combine multiple learning algorithms to improve the overall performance of the model. Examples of hybrid models include:

1. Ensemble models: These models use multiple base models and combine their predictions to make a final prediction. Examples include Random Forest and Gradient Boosting.
2. Multi-task models: These models are trained on multiple tasks and can perform multiple tasks simultaneously. An example is a model that can classify an image and also generate a caption for the image.
3. Transfer learning models: These models use pre-trained models and fine-tune them for a new task. For example, using a pre-trained image classification model as a feature extractor for a new object detection task.
4. Hybrid deep learning models: These models combine different types of deep learning architectures such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to improve performance on a specific task.
5. Multi-modal models: These models combine multiple modalities of data such as text and images to improve performance on a specific task.

Ensemble models in scikit-learn (sklearn) are a set of models that combine multiple base models to improve the overall performance of the model. Some examples of ensemble models in sklearn include:

1. RandomForestClassifier: This model creates an ensemble of decision trees and makes a final prediction by taking the mode of the predictions of the individual trees.
2. GradientBoostingClassifier: This model creates an ensemble of decision trees and makes a final prediction by taking a weighted average of the predictions of the individual trees.
3. AdaBoostClassifier: This model creates an ensemble of decision trees and makes a final prediction by adjusting the weights of the training data based on the accuracy of the individual trees.
4. VotingClassifier: This model creates an ensemble of different models and makes a final prediction by taking the mode of the predictions of the individual models.
5. BaggingClassifier: This model creates an ensemble of decision trees by training them on different subsets of the training data and makes a final prediction by taking the mode of the predictions of the individual trees.

All of these ensemble models are implemented as Python classes in the sklearn.ensemble module and can be used in a similar way as other models in sklearn, by instantiating the class, fitting to training data and then predicting on new data.

Hybrid deep learning models that combine different types of deep learning architectures, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are not directly provided in scikit-learn (sklearn). Since sklearn is a machine learning library that provides a wide range of classical machine learning models, it does not have the capability to build complex hybrid deep learning models. However, it can provide some tools to preprocess and transform the data that can be used in deep learning frameworks such as Tensorflow and Keras.

If you would like to build a hybrid deep learning model using sklearn, you can do so by using the preprocessing tools provided by sklearn to transform your data, and then use another deep learning framework such as Tensorflow or Keras to build your model. Tensorflow and Keras provide a wide range of functionalities to build and train complex deep learning models, including the ability to combine multiple architectures like CNNs and RNNs.

In summary, sklearn is not a deep learning library, it can be used for preprocessing and transforming the data that can be used in a deep learning framework. A deep learning framework like Tensorflow or Keras can be used for building and training complex hybrid deep learning models that combine multiple architectures like CNNs and RNNs.