Random Forests:In random forests (see RandomForestClassifier and RandomForestRegressor classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.Furthermore, when splitting each node during the construction of a tree, the best split is found through an exhaustive search of the features values of either all input features or a random subset of size max_features. (See the parameter tuning guidelines for more details.)The purpose of these two sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to overfit. The injected randomness in forests yield decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant hence yielding an overall better model.In contrast to the original publication [B2001], the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.A competitive alternative to random forests are Histogram-Based Gradient Boosting (HGBT) models:Building trees: Random forests typically rely on deep trees (that overfit individually) which uses much computational resources, as they require several splittings and evaluations of candidate splits. Boosting models build shallow trees (that underfit individually) which are faster to fit and predict.Sequential boosting: In HGBT, the decision trees are built sequentially, where each tree is trained to correct the errors made by the previous ones. This allows them to iteratively improve the model's performance using relatively few trees. In contrast, random forests use a majority vote to predict the outcome, which can require a larger number of trees to achieve the same level of accuracy.Efficient binning: HGBT uses an efficient binning algorithm that can handle large datasets with a high number of features. The binning algorithm can pre-process the data to speed up the subsequent tree construction (see Why it's faster). In contrast, the scikit-learn implementation of random forests does not use binning and relies on exact splitting, which can be computationally expensive.Overall, the computational cost of HGBT versus RF depends on the specific characteristics of the dataset and the modeling task. It's a good idea to try both models and compare their performance and computational efficiency on your specific problem to determine which model is the best fit.

RandomForestClassifier: class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)A random forest classifier:A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Trees in the forest use the best split strategy, i.e. equivalent to passing splitter="best" to the underlying DecisionTreeRegressor. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.For a comparison between tree-based ensemble models see the example Comparing Random Forests and Histogram Gradient Boosting models.

La liste de Hyperparameter,Parameters de Random Forests : La liste des hyperparamètres et des paramètres de Random Forests comprend n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, max_leaf_nodes, min_impurity_decrease, min_impurity_split, bootstrap, oob_score, n_jobs, random_state, verbose, warm_start, class_weight, ccp_alpha et max_samples."

Hyperparameter,Parameters de Random Forests:1. n_estimators,int, default=100,Number of trees in the forest.

Hyperparameter,Parameters de Random Forests:2.criterion{“gini”, “entropy”, “log_loss”}, default=“gini”.The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “log_loss” and “entropy” both for the Shannon information gain, see Mathematical formulation. Note: This parameter is tree-specific.

Hyperparameter,Parameters de Random Forests:3.max_depth,int, default=None.The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

Hyperparameter,Parameters de Random Forests:4.min_samples_split,int or float, default=2.The minimum number of samples required to split an internal node:If int, then consider min_samples_split as the minimum number.If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.

Hyperparameter,Parameters de Random Forests:5.min_samples_leaf,int or float, default=1.The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.If int, then consider min_samples_leaf as the minimum number.If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

Hyperparameter,Parameters de Random Forests:6.min_weight_fraction_leaf,float, default=0.0.The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

Hyperparameter,Parameters de Random Forests:7.max_features{“sqrt”, “log2”, None}, int or float, default=”sqrt”.The number of features to consider when looking for the best split:If int, then consider max_features features at each split.If float, then max_features is a fraction and max(1, int(max_features * n_features_in_)) features are considered at each split.If “sqrt”, then max_features=sqrt(n_features).If “log2”, then max_features=log2(n_features).If None, then max_features=n_features.Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.

Hyperparameter,Parameters de Random Forests:8.max_leaf_nodes,int, default=None.Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

Hyperparameter,Parameters de Random Forests:9.min_impurity_decrease,float, default=0.0.A node will be split if this split induces a decrease of the impurity greater than or equal to this value.The weighted impurity decrease equation is the following:N_t / N * (impurity - N_t_R / N_t * right_impurity - N_t_L / N_t * left_impurity).where N is the total number of samples, N_t is the number of samples at the current node, N_t_L is the number of samples in the left child, and N_t_R is the number of samples in the right child.N, N_t, N_t_R and N_t_L all refer to the weighted sum, if sample_weight is passed.

Hyperparameter,Parameters de Random Forests:10.bootstrap,bool, default=True.Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

Hyperparameter,Parameters de Random Forests:11.oob_score,bool or callable, default=False.Whether to use out-of-bag samples to estimate the generalization score. By default, accuracy_score is used. Provide a callable with signature metric(y_true, y_pred) to use a custom metric. Only available if bootstrap=True.

Hyperparameter,Parameters de Random Forests:12.n_jobs,int, default=None.The number of jobs to run in parallel. fit, predict, decision_path and apply are all parallelized over the trees. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. See Glossary for more details.

Hyperparameter,Parameters de Random Forests:13.random_state,int, RandomState instance or None, default=None.Controls both the randomness of the bootstrapping of the samples used when building trees (if bootstrap=True) and the sampling of the features to consider when looking for the best split at each node (if max_features < n_features). See Glossary for details.

Hyperparameter,Parameters de Random Forests:14.verbose,int, default=0.Controls the verbosity when fitting and predicting.

Hyperparameter,Parameters de Random Forests:15.warm_startbool, default=False.When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest. See Glossary and Fitting additional trees for details.

Hyperparameter,Parameters de Random Forests:16.class_weight{“balanced”, “balanced_subsample”}, dict or list of dicts, default=None.Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of [{1:1}, {2:5}, {3:1}, {4:1}].The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))The “balanced_subsample” mode is the same as “balanced” except that weights are computed based on the bootstrap sample for every tree grown.For multi-output, the weights of each column of y will be multiplied.Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

Hyperparameter,Parameters de Random Forests:17.ccp_alphanon-negative, float, default=0.0.Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.

Hyperparameter,Parameters de Random Forests:18.max_samples,int or float, default=None.If bootstrap is True, the number of samples to draw from X to train each base estimator.If None (default), then draw X.shape[0] samples.If int, then draw max_samples samples.If float, then draw max(round(n_samples * max_samples), 1) samples. Thus, max_samples should be in the interval (0.0, 1.0].

Hyperparameter,Parameters de Random Forests:19.monotonic_cst,array-like of int of shape (n_features), default=None.Indicates the monotonicity constraint to enforce on each feature.1: monotonic increase.0: no constraint.-1: monotonic decrease.If monotonic_cst is None, no constraints are applied.Monotonicity constraints are not supported for:multiclass classifications (i.e. when n_classes > 2),multioutput classifications (i.e. when n_outputs_ > 1),classifications trained on data with missing values.The constraints hold over the probability of the positive class.

Support Vector Machines: Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.The advantages of support vector machines are:Effective in high dimensional spaces.Still effective in cases where number of dimensions is greater than the number of samples.Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.The disadvantages of support vector machines include:If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.

class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None).C-Support Vector Classification.The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using LinearSVC or SGDClassifier instead, possibly after a Nystroem transformer or other Kernel Approximation.The multiclass support is handled according to a one-vs-one scheme.

La liste de Hyperparameter,Parameters de SVM: Les paramètres et hyperparamètres de SVM incluent C, kernel, degree, gamma, coef0, shrinking, probability, tol, cache_size, class_weight, verbose, max_iter, decision_function_shape, break_ties et random_state.
Hyperparameter,Parameters de SVM:1.C:float, default=1.0
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. For an intuitive visualization of the effects of scaling the regularization parameter C, see Scaling the regularization parameter for SVCs.
Hyperparameter,Parameters de SVM:2.kernel{‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’} or callable, default=’rbf’
Specifies the kernel type to be used in the algorithm. If none is given, ‘rbf’ will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples). For an intuitive visualization of different kernel types see Plot classification boundaries with different SVM Kernels.
Hyperparameter,Parameters de SVM:3.degree:int, default=3
Degree of the polynomial kernel function (‘poly’). Must be non-negative. Ignored by all other kernels.
Hyperparameter,Parameters de SVM:4.gamma{‘scale’, ‘auto’} or float, default=’scale’
Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.if gamma='scale' (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,if ‘auto’, uses 1 / n_features.if float, must be non-negative.
Hyperparameter,Parameters de SVM:5.coef0:float, default=0.0
Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.

Hyperparameter,Parameters de SVM:6.shrinking:bool, default=True
Whether to use the shrinking heuristic. See the User Guide.
Hyperparameter,Parameters de SVM:7.probability:bool, default=False
Whether to enable probability estimates. This must be enabled prior to calling fit, will slow down that method as it internally uses 5-fold cross-validation, and predict_proba may be inconsistent with predict. Read more in the User Guide.
Hyperparameter,Parameters de SVM:8.tol:float, default=1e-3
Tolerance for stopping criterion.
Hyperparameter,Parameters de SVM:9.cache_size :float, default=200
Specify the size of the kernel cache (in MB).
Hyperparameter,Parameters de SVM:10.class_weight :dict or 'balanced', default=None
Set the parameter C of class i to class_weight[i]*C for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).
Hyperparameter,Parameters de SVM:11.verbose :bool, default=False
Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.
Hyperparameter,Parameters de SVM:12.max_iter :int, default=-1
Hard limit on iterations within solver, or -1 for no limit.
Hyperparameter,Parameters de SVM:13.decision_function_shape{'ovo', 'ovr'}, default='ovr'
Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, note that internally, one-vs-one ('ovo') is always used as a multi-class strategy to train models; an ovr matrix is only constructed from the ovo matrix. The parameter is ignored for binary classification.
break_tiesbool, default=False
If true, decision_function_shape='ovr', and number of classes > 2, predict will break ties according to the confidence values of decision_function; otherwise the first class among the tied classes is returned. Please note that breaking ties comes at a relatively high computational cost compared to a simple predict.
Hyperparameter,Parameters de SVM:14.random_state :int, RandomState instance or None, default=None
Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False. Pass an int for reproducible output across multiple function calls.

KNeighborsClassifier: class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None) Classifier implementing the k-nearest neighbors vote.
La liste de Hyperparameter,Parameters de KNN: La liste des hyperparamètres et des paramètres de KNN (k plus proches voisins) comprend n_neighbors, weights, algorithm, leaf_size, p, metric et metric_params.

Hyperparameter,Parameters de KNN:1.n_neighbors,int, default=5Number of neighbors to use by default for kneighbors queries.
Hyperparameter,Parameters de KNN:2.weights{'uniform', 'distance'}, callable or None, default='uniform'
Weight function used in prediction. Possible values:
'uniform' : uniform weights. All points in each neighborhood are weighted equally.
'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
[callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.
Refer to the example entitled Nearest Neighbors Classification showing the impact of the weights parameter on the decision boundary.
Hyperparameter,Parameters de KNN:3.algorithm{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'.Algorithm used to compute the nearest neighbors:'ball_tree' will use BallTree;'kd_tree' will use KDTree;'brute' will use a brute-force search.'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method.Note: fitting on sparse input will override the setting of this parameter, using brute force.
Hyperparameter,Parameters de KNN:4.leaf_size,int, default=30
Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
Hyperparameter,Parameters de KNN:5.p,float, default=2
Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used. This parameter is expected to be positive.
Hyperparameter,Parameters de KNN:6.metric,str or callable, default='minkowski'
Metric to use for distance computation. Default is "minkowski", which results in the standard Euclidean distance when p = 2. See the documentation of scipy.spatial.distance and the metrics listed in distance_metrics for valid metric values.
If metric is "precomputed", X is assumed to be a distance matrix and must be square during fit. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors.
If metric is a callable function, it takes two arrays representing 1D vectors as inputs and must return one value indicating the distance between those vectors. This works for Scipy's metrics, but is less efficient than passing the metric name as a string.
Hyperparameter,Parameters de KNN:7.metric_params,dict, default=None
Additional keyword arguments for the metric function.
Hyperparameter,Parameters de KNN:8.n_jobs,int, default=None
The number of parallel jobs to run for neighbors search. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. See Glossary for more details. Doesn't affect fit method.

XGBoost Documentation:XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Introduction to Boosted Trees,XGBoost stands for "Extreme Gradient Boosting", where the term "Gradient Boosting" originates from the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman.The gradient boosted trees has been around for a while, and there are a lot of materials on the topic. This tutorial will explain boosted trees in a self-contained and principled way using the elements of supervised learning. We think this explanation is cleaner, more formal, and motivates the model formulation used in XGBoost.

Elements of Supervised Learning,XGBoost is used for supervised learning problems, where we use the training data (with multiple features)
to predict a target variable . Before we learn about trees specifically, let us start by reviewing the basic elements in supervised learning.

AdaBoostClassifier: class sklearn.ensemble.AdaBoostClassifier(estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)[source]
An AdaBoost classifier.An AdaBoost [classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.This class implements the algorithm based on .
Hyperparameter,Parameters de AdaBoost:1.estimator,object, default=None
The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper classes_ and n_classes_ attributes. If None, then the base estimator is DecisionTreeClassifier initialized with max_depth=1.
Hyperparameter,Parameters de AdaBoost:2.n_estimators,int, default=50.The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early. Values must be in the range [1, inf).
Hyperparameter,Parameters de AdaBoost:3.learning_rate,float, default=1.0.Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the learning_rate and n_estimators parameters. Values must be in the range (0.0, inf).
Hyperparameter,Parameters de AdaBoost:4.algorithm{'SAMME', 'SAMME.R'}, default='SAMME.R'.If 'SAMME.R' then use the SAMME.R real boosting algorithm. estimator must support calculation of class probabilities. If 'SAMME' then use the SAMME discrete boosting algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.

Hyperparameter,Parameters de AdaBoost:5.random_state,int, RandomState instance or None, default=None.Controls the random seed given at each estimator at each boosting iteration. Thus, it is only used when estimator exposes a random_state. Pass an int for reproducible output across multiple function calls.

La RFID (Radio frequency identification) ou radio-identification est une technologie qui permet de mémoriser, stocker, enregistrer et récupérer des données à distance à l'aide de marqueurs : des radio-étiquettes, également appelées "Tags RFID" qui réagissent aux Définition | RFID - Puce RFID - Radio Frequency Identification des radio et transmettent des informations à distance. Ces puces RFID, étiquettes ou balises, peuvent être collées sur un produit ou être incorporées dans un produit.Un peu d'histoire : cette technologie RFID est utilisée depuis la 2nd guerre mondiale. À l'époque, elle était surtout répandue dans le secteur militaire (identification des avions etc). Plus tard, dans les 70's, cette technologie est utilisée par l'armée pour le contrôle de l'accès aux sites sensibles comme le nucléaire par exemple. C'est à partir des années 80 qu'elle est utilisée dans le quotidien de certains professionnels comme pour l'identification du bétail et lors de la chaîne de fabrication de construction. Enfin, elle est totalement popularisée dans les années 2000.Les secteurs principalement concernés par la technologie RFID sont les suivants : La logistique (traçabilité des produits) ;- La sécurité (système d'antivol ou moyen d'identification des produits en caisse) ;- Cartes et papiers d'identité (passeports, cartes d'accès aux transports en commun etc) ;- Agriculture ou secteur animalier (identification du bétail et des animaux).