THOR: Algorithms' Parameters and Ranges

1 Predictive Algorithms Parameters

KNeighborsClassifier (n_neighbors = 5, weights = 'uniform', algorithm = 'auto', p = 2, metric = 'minkowski')

- **n_neighbors**: Int, optional (default = 5) Number of neighbors to use by default for kneighbors queries.
- **weights**: String or callable, optional (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.
- **algorithm**: 'auto', 'ball_tree', 'kd_tree', 'brute', optional 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.
- **p**: Integer, optional (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.
- **metric**: String or callable, (default 'minkowski') the distance metric to use for the tree. The default metric is minkowski, and with p = 2 is equivalent to the standard Euclidean metric. 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.

SVC(C = 1.0, kernel = 'rbf', degree = 3, gamma = 'scale', coef0 = 0.0, shrinking = True, probability = True, tol = 1e-3, cache_size = 200, class_weight = None, verbose = False, max_iter = -1, decision_function_shape = 'ovr')

- **C**: Float, optional (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.
- **kernel**: String, optional (default = 'rbf') Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to precompute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).
- **degree**: Int, optional (default = 3) Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
- **gamma**: {'scale', 'auto'} or float, optional (default='scale') Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma='scale' (default) then it uses $\frac{1}{n\_features * X.Var()}$ as value of gamma, if 'auto', uses $\frac{1}{n\_features}$.
- **coef0**: Float, optional (default=0.0) Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.
- **shrinking**: Boolean, optional (default=True) Whether to use the shrinking heuristic.
- **probability**: Boolean, optional (default=False) Whether to enable probability estimates.
- **tol**: Float, optional (default = 1e-3) Tolerance for stopping criterion.
- **cache_size**: Float, optional Specify the size of the kernel cache (in MB).
- **class_weight**: {dict, 'balanced'}, optional 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 $\frac{n\_samples}{n\_classes * np.bincount(y)}$
- **verbose**: Boolean, (default: False) Enable verbose output.
- **max_iter**: Int, optional (default=-1) Hard limit on iterations within solver, or -1 for no limit.
- **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, $\frac{n\_classes * (n\_classes - 1)}{2}$). However, one-vs-one ('ovo') is always used as multi-class strategy.

DecisionTreeClassifier (criterion = "gini", max_depth = None, min_samples_split = 2, min_samples_leaf = 1, max_features = None, max_leaf_nodes = None)

- **criterion**: {"gini", "entropy"}, (default = "gini") The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
- **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.
- **min_samples_split**: Int or float, (default = 2) The minimum number of samples required to split an internal node.
- **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.
- **max_features**: Int, float or {"auto", "sqrt", "log2"}, (default = None) The number of features to consider when looking for the best split:
  - **int** Consider 'max_features' features at each split.
  - **float** 'max_features' is a fraction and 'int$(max_features * n_features)$' features are considered at each split.
  - **"auto"** '$max\_features = n\_features^2$'.
  - **"sqrt"** '$max\_features = n\_features^2$'.
  - **"log2"** '$max\_features = log_2(n\_features)$'.
  - **None** '$max\_features = n\_features$'.
- **max_leaf_nodes**: Int, (default = None) Grow a tree 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.

LogisticRegression (C = 1.0, fit_intercept = True,solver = 'lbfgs')

- **C**: Float, (default = 1.0) Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.
- **fit_intercept**: Boolean, (default = True) Specifies if a constant (a.k.a., bias or intercept) should be added to the decision function.

- **solver**: {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, (default = 'lbfgs') Algorithm to use in the optimization problem. For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones. For multi-class problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss; 'liblinear' is limited to one-versus-rest schemes.

RandomForestClassifier (n_estimators = 100, criterion = "gini", max_depth = None, min_samples_split = 2, max_features = "auto", bootstrap = True)

- **n_estimators**: Int, optional (default=100) The No. of trees in the forest.
- **criterion**: String, optional (default = "gini") The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
- **max_depth**: Integer or None, optional (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.
- **min_samples_split**: Int, float, optional (default=2) The minimum number of samples required to split an internal node
- **max_features**: Int, float, string or None, optional (default = "auto") The number of features to consider when looking for the best split:
  - **int** Consider 'max_features' features at each split.
  - **float** 'max_features' is a fraction and 'int($max\_features * n\_features$)' features are considered at each split.
  - **"auto"** '$max\_features = n\_features^2$'.
  - **"sqrt"** '$max\_features = n\_features^2$'.
  - **"log2"** '$max\_features = log_2(n\_features)$'.
  - **None** '$max\_features = n\_features$'.
- **bootstrap**: Bool, optional (default=True) Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

2 BayesSearchCV Parameters

BayesSearchCV(estimator, search_spaces, n_iter=50)

- **estimator**: Estimator object. An object of that type is instantiated for each search point. This object is assumed to implement the scikit-learn estimator API.
- **search_spaces**: Dict, list of dict or list of tuple containing (dict, int). One of these cases:

- **dictionary** Keys are parameter names (strings) and values are skopt.space.Dimension instances (Real, Integer or Categorical) or any other valid value that defines skopt dimension. Represents search space over parameters of the provided estimator.
- **list of dictionaries**: A list of dictionaries, where every dictionary fits the description given in case 1 above. If a list of dictionary objects is given, then the search is performed sequentially for every parameter space with maximum number of evaluations set to self.n_iter.
- **list of (dict, int > 0)**: An extension of case 2 above, where first element of every tuple is a dictionary representing some search subspace, similarly as in case 2, and second element is a number of iterations that will be spent optimizing over this subspace.

- **n_iter**: Int, (default = 50) Number of parameter settings that are sampled. n_iter trades off runtime vs quality of the solution.

The search_spaces of each algorithm is listed in Table 1.

Algorithm	Parameter	Search Range
KNN	n_neighbors	Integer (low=2 , high=30)
	algorithm	Categorical([‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’])
	weight	Categorical([‘distance’ , ‘uniform’])
	p	Integer(low=1 , high=6)
SVC	C	Real(low=1e-6, high=1e+6, prior=‘log-uniform’)
	kernel	Categorical([‘linear’ , ‘poly’ , ‘rbf’])
	degree	Integer(low=1, high=8)
	gamma	Real(low=1e-6, high=1e+1, prior=‘log-uniform’)
	probability	Categorical([True])
DT	max_depth	Integer(low=1, high=5)
	criterion	Categorical([‘gini’,‘entropy’])
	max_features	Integer(low=1, high=40)
	max_leaf_nodes	Integer(low=1, high=20)
	min_samples_leaf	Integer(low=1, high=20)
LR	C	Real(low=0.5, high=1)
	fit_intercept	Categorical([True, False])
	solver	Categorical([‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’])
RF	n_estimators	Integer(low=10, high=100)
	criterion	Categorical([‘gini’,‘entropy’])
	max_depth	Integer(low=1, high=40)
	max_features	Integer(low=1, high=40)
	bootstrap	Categorical([True , False])

Table 1: BSCV search_spaces for different classifier algorithms