HYPERPARAMETER TUNING CHEATSHEET

Detailed explanations and guidance — Purple theme

Quick Reference Table

Hyperparameter	What it Controls	Practical Tip
n_estimators	Number of trees in ensemble models (RF, XGB).	More trees → better stability but slower training;
max_depth	Maximum depth of individual trees.	Lower depth reduces overfitting; try 3-30 or Non
min_samples_split	Min samples required to split an internal node.	Increase to prevent small splits (2,5,10).
min_samples_leaf	Min samples required at a leaf node.	Helps generalization; try 1-4.
max_features	Number or fraction of features to consider for split.	Use 'sqrt' for RF classification; reduces correlatio
learning_rate	Step size for boosting updates (GB/XGB).	Lower rates need more trees; common: 0.01-0.1
subsample	Fraction of samples used per tree in boosting.	Used to reduce overfitting; 0.6-1.0.
C	Inverse regularization strength in linear/SVM models.	Smaller C → stronger regularization; try 0.1-100
alpha	Regularization strength (Ridge/Lasso).	Larger alpha → more shrinkage; try 0.001-100.
kernel	Type of kernel function in SVM.	RBF is a good default; tune degree for poly.
n_neighbors	Number of neighbors in KNN.	Odd values avoid ties; try 3-11.
criterion	Function to measure split quality (trees).	Gini or entropy for classification; squared_error f
epsilon	Insensitive loss margin in SVR.	Larger epsilon → wider margin; tune based on n

Classification Models (Part 1)

Random Forest Classifier

```
param_grid = {
    "n_estimators": [100, 200, 300, 500],
    "max_depth": [None, 10, 20, 30, 50],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2", None],
    "bootstrap": [True, False]
}
```

Parameter meanings

n_estimators: Number of trees in the forest. Increasing improves stability and reduces variance but increases training time. Start with 100 and increase if model variance is high.

max_depth: Maximum depth of each tree. Controls model complexity; smaller values help prevent overfitting. Use None to allow full growth if you want deep trees.

min_samples_split: Minimum samples required to split a node. Higher values produce simpler trees; useful on noisy data to prevent tiny splits.

min_samples_leaf: Minimum samples at a leaf node. Prevents creation of leaves with very few samples, improving generalization.

max_features: Number of features considered for best split. 'sqrt' is typical for classification; lower values increase bias but reduce variance.

bootstrap: Whether to use bootstrapped samples when building trees. True enables bagging; False uses the whole dataset for each tree.

Logistic Regression

```
param_grid = {
```

```
    "penalty": ["l1", "l2", "elasticnet", None],
    "C": [0.01, 0.1, 1, 10, 100],
    "solver": ["lbfgs", "liblinear", "saga"],
    "max_iter": [100, 200, 500]
}
```

Parameter meanings

penalty: Type of regularization (l1, l2, elasticnet). Regularization helps prevent overfitting by shrinking coefficients; choose solver that supports the penalty.

C: Inverse of regularization strength. Smaller values = stronger regularization. Tune to balance bias-variance; common range 0.01–100.

solver: Algorithm for optimization. Some solvers (saga, liblinear) support l1/elasticnet. Choose based on penalty and dataset size.

max_iter: Max iterations for solver convergence. Increase if you see convergence warnings, especially with large datasets or complex penalties.

Support Vector Classifier (SVC)

```
param_grid = {
    "C": [0.1, 1, 10, 100],
    "kernel": ["linear", "poly", "rbf", "sigmoid"],
    "degree": [2, 3, 4],
    "gamma": ["scale", "auto"]
}
```

Parameter meanings

C: Regularization parameter (inverse of regularization strength). Larger C -> less regularization (can overfit); smaller C -> smoother decision boundary.

kernel: Kernel function transforming inputs into higher-dimensional space. 'rbf' is a good default; 'poly' allows polynomial boundaries.

degree: Degree for polynomial kernel. Tune only when using kernel='poly'; higher values increase flexibility.

gamma: Kernel coefficient for 'rbf', 'poly', and 'sigmoid'. 'scale' usually works well; higher gamma can lead to overfitting.

Classification Models (continued) & Regression Models

Gradient Boosting Classifier

```
param_grid = {
    "n_estimators": [100, 200, 300],
    "learning_rate": [0.001, 0.01, 0.1, 0.2],
    "max_depth": [3, 5, 7],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "subsample": [0.8, 1.0],
    "max_features": ["sqrt", "log2", None]
}
```

Parameter meanings

n_estimators: Number of boosting stages (trees). More trees + lower learning_rate often gives better performance but increases training time.

learning_rate: Shrinks the contribution of each tree. Lower values improve generalization but require more estimators.

max_depth: Maximum depth of each tree; controls complexity.

subsample: Fraction of samples used for fitting each base estimator.
Values <1.0 add randomness and can reduce overfitting.

XGBoost Classifier

```
param_grid = {
    "n_estimators": [100, 200, 300],
    "learning_rate": [0.01, 0.05, 0.1, 0.2],
    "max_depth": [3, 5, 7, 10],
    "subsample": [0.6, 0.8, 1.0],
    "colsample_bytree": [0.6, 0.8, 1.0],
    "gamma": [0, 0.1, 0.2],
    "reg_lambda": [0.5, 1.0, 2.0],
    "reg_alpha": [0, 0.5, 1.0]
}
```

Parameter meanings

colsample_bytree: Fraction of features used per tree; lower values help regularize and speed up training.

gamma: Minimum loss reduction required to make a split. Higher values make algorithm more conservative.

reg_lambda/reg_alpha: L2 and L1 regularization terms on weights to reduce overfitting.

K-Nearest Neighbors (Classifier)

```
param_grid = {
    "n_neighbors": [3, 5, 7, 9, 11],
    "weights": ["uniform", "distance"],
    "metric": ["euclidean", "manhattan", "minkowski"]
}
```

Parameter meanings

n_neighbors: Number of neighbors considered when making predictions. Larger values smooth predictions but may underfit.

weights: Weight function used in prediction. 'distance' gives closer neighbors more influence.

metric: Distance metric used; choose based on feature scaling and meaning.

Decision Tree Classifier

```
param_grid = {
    "criterion": ["gini", "entropy", "log_loss"],
    "max_depth": [None, 10, 20, 30, 50],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2", None]
}
```

Parameter meanings

criterion: Function to measure quality of a split. 'gini' and 'entropy' are common for classification.

max_features: Number of features to consider when looking for best split; reduces variance when smaller.

Gaussian Naive Bayes

```
param_grid = {
    "var_smoothing": [1e-9, 1e-8, 1e-7, 1e-6]
}
```

Parameter meanings

var_smoothing: Portion of the largest variance of all features added to variances for calculation stability. Increase for numerical stability on small variances.

Regression Models

Linear Regression

```
param_grid = {
    "fit_intercept": [True, False],
    "positive": [True, False]
}
```

Parameter meanings

fit_intercept: Whether to calculate the intercept for this model. Set False if data is already centered.

positive: Constrain coefficients to be positive. Use when domain requires non-negative weights.

Ridge Regression

```
param_grid = {
    "alpha": [0.01, 0.1, 1, 10, 100],
    "solver": ["auto", "svd", "cholesky", "lsqr", "saga"],
    "fit_intercept": [True, False]
}
```

Parameter meanings

alpha: Regularization strength (L2). Higher alpha increases shrinkage of coefficients and reduces variance.

solver: Algorithm to use in the optimization; some are faster for large datasets.

Lasso Regression

```
param_grid = {
    "alpha": [0.001, 0.01, 0.1, 1, 10],
    "max_iter": [1000, 5000, 10000],
    "fit_intercept": [True, False]
}
```

Parameter meanings

alpha: Regularization strength (L1). Encourages sparsity (feature selection) by driving coefficients to zero.

max_iter: Maximum iterations for coordinate descent solver; increase if convergence warnings appear.

ElasticNet Regression

```
param_grid = {
    "alpha": [0.001, 0.01, 0.1, 1, 10],
    "l1_ratio": [0.1, 0.3, 0.5, 0.7, 0.9],
    "max_iter": [1000, 5000, 10000]
}
```

Parameter meanings

l1_ratio: Mix between L1 and L2 penalties (0 = Ridge, 1 = Lasso). Choose based on whether you want sparsity or grouped shrinkage.

Random Forest Regressor

```
param_grid = {
    "n_estimators": [100, 200, 500],
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": [2, 5, 10],
```

```
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2", None],
    "bootstrap": [True, False]
}
```

Parameter meanings

n_estimators: Number of trees. More trees reduce variance but cost time.

max_depth: Controls tree size and complexity; smaller depths reduce overfitting.

Gradient Boosting Regressor

```
param_grid = {
    "n_estimators": [100, 200, 300],
    "learning_rate": [0.01, 0.05, 0.1, 0.2],
    "max_depth": [3, 5, 7],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "subsample": [0.8, 1.0],
    "max_features": ["sqrt", "log2", None]
}
```

Parameter meanings

learning_rate: Shrinks contributions of each tree; smaller rates generalize better with more trees.

XGBoost Regressor

```
param_grid = {
    "n_estimators": [100, 200, 300],
    "learning_rate": [0.01, 0.05, 0.1, 0.2],
    "max_depth": [3, 5, 7, 10],
    "subsample": [0.6, 0.8, 1.0],
    "colsample_bytree": [0.6, 0.8, 1.0],
    "gamma": [0, 0.1, 0.2],
    "reg_lambda": [0.5, 1.0, 2.0],
    "reg_alpha": [0, 0.5, 1.0]
}
```

K-Nearest Neighbors Regressor

```
param_grid = {
    "n_neighbors": [3, 5, 7, 9, 11],
    "weights": ["uniform", "distance"],
    "metric": ["euclidean", "manhattan", "minkowski"]
}
```

Parameter meanings

n_neighbors: Number of neighbors considered for predictions; more neighbors smooth predictions.

Decision Tree Regressor

```
param_grid = {
    "criterion": ["squared_error", "friedman_mse", "absolute_error", "poisson"],
    "max_depth": [None, 10, 20, 30, 50],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["sqrt", "log2", None]
}
```

Parameter meanings

criterion: Loss function to measure quality of a split; use squared_error for standard regression tasks.

Support Vector Regressor (SVR)

```
param_grid = {
    "C": [0.1, 1, 10, 100],
    "kernel": ["linear", "poly", "rbf", "sigmoid"],
    "degree": [2, 3, 4],
    "gamma": ["scale", "auto"],
    "epsilon": [0.01, 0.1, 0.2, 0.5]
}
```

Parameter meanings

epsilon: Epsilon in the epsilon-insensitive loss function; larger values make model less sensitive to small errors.

Best Practices & Tuning Tips

GridSearchCV vs RandomizedSearchCV: GridSearch exhaustively searches combinations — use for small grids. RandomizedSearch samples parameter space — faster and better for large grids.

Workflow: 1) Start with broad ranges and RandomizedSearch, 2) Narrow promising regions and use GridSearch, 3) Validate on a hold-out test set.

Cross-validation: Use stratified CV for imbalanced classification. For time-series, use time-aware CV.

Avoid overfitting: Monitor validation curves, use regularization, limit tree depth, increase min_samples_leaf, or use subsampling in boosting.

Compute cost: Balance grid size and available compute. Use n_jobs=-1 for parallelism and consider early stopping where supported (e.g., XGBoost).