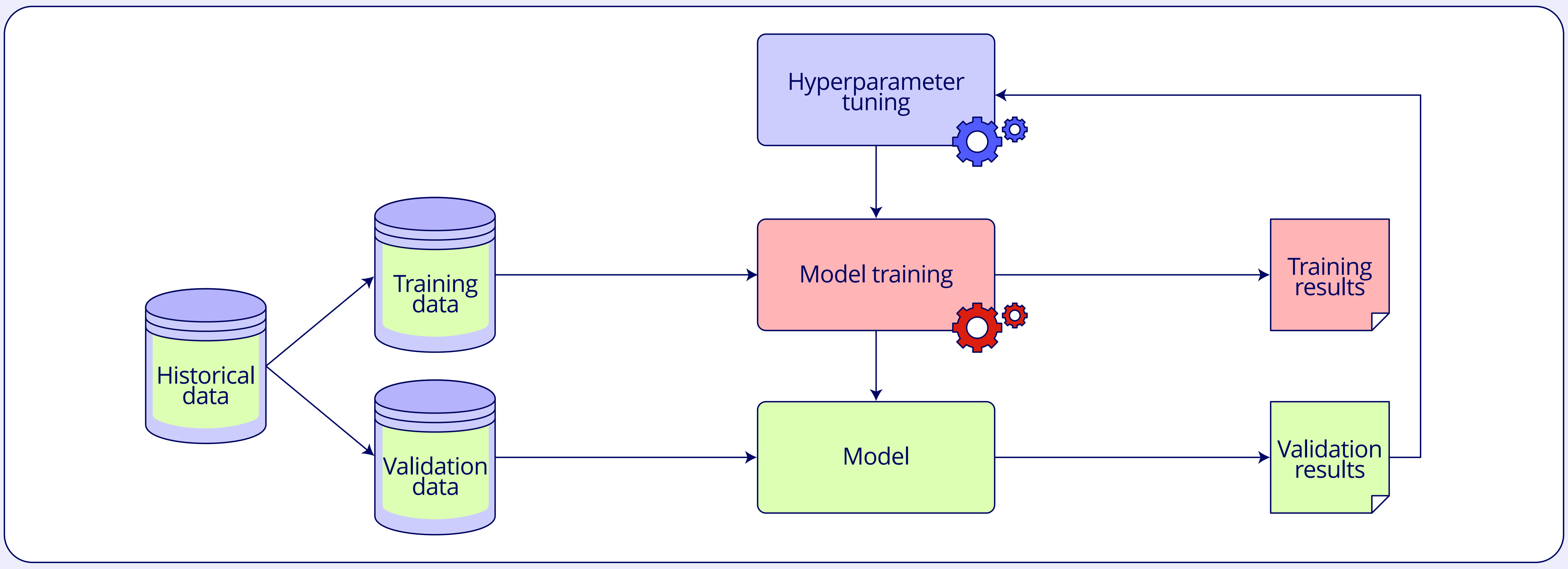
Hyperparameter Tuning

educative

Hyperparameters

- Parameters that control the training process and must be set before training.
- Examples include the number of layers in a neural network and the learning rate in gradient descent.



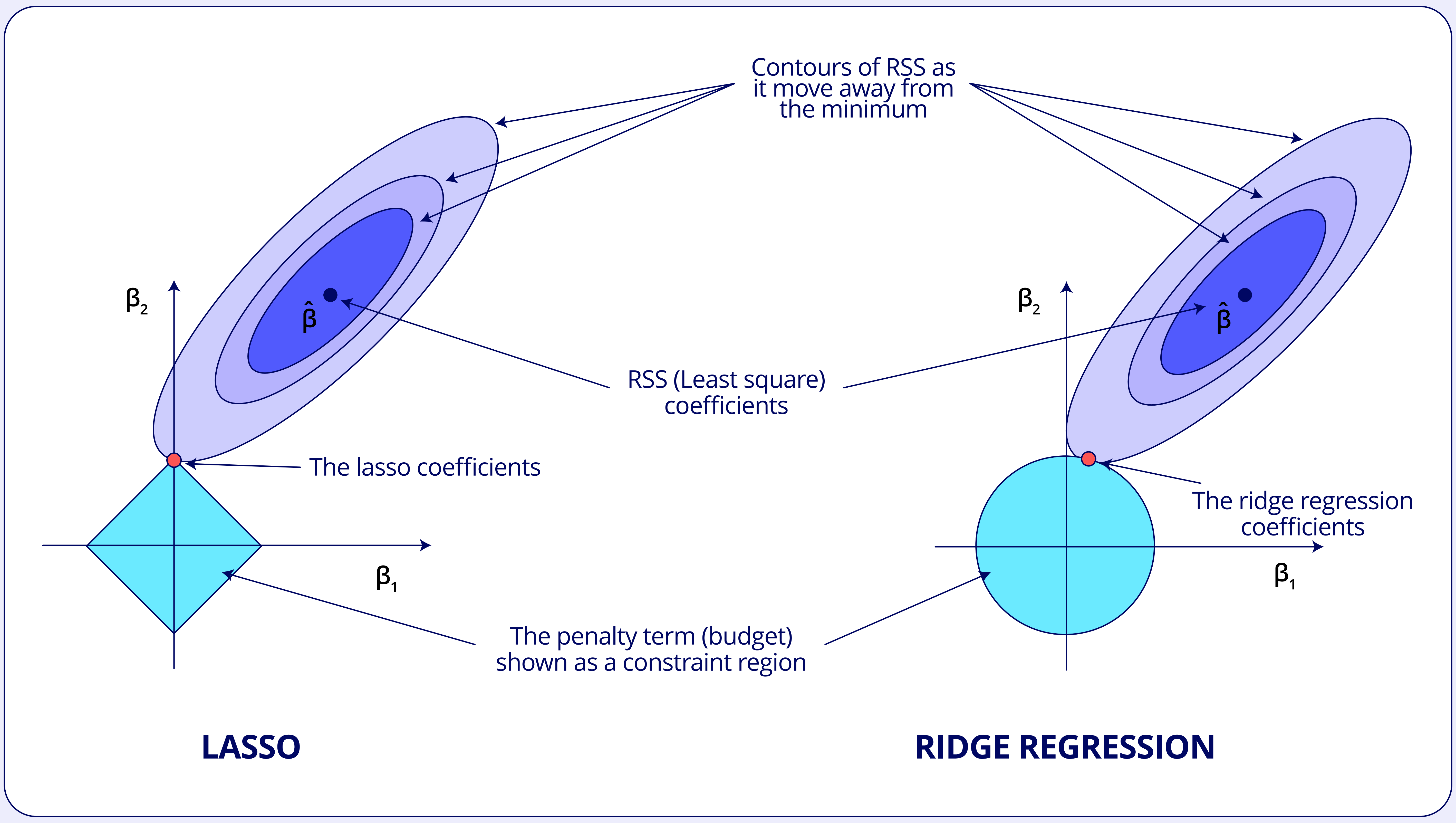
Model Parameters

- Model parameters are the elements of the model that are learned during the training process by the model itself.
- Examples include the weights in a neural network and the coefficients in a linear regression model.

Common Hyperparameters by Model

1. Linear Models (e.g., linear regression, logistic regression)

- **Learning rate:** Controls the step size taken during gradient descent optimization.
- **Regularization strength:**
  - **L1 (Lasso):** Encourages sparsity by driving some coefficients to exactly zero, useful for feature selection.
  - **L2 (Ridge):** Reduces the magnitude of coefficients, helping to prevent overfitting

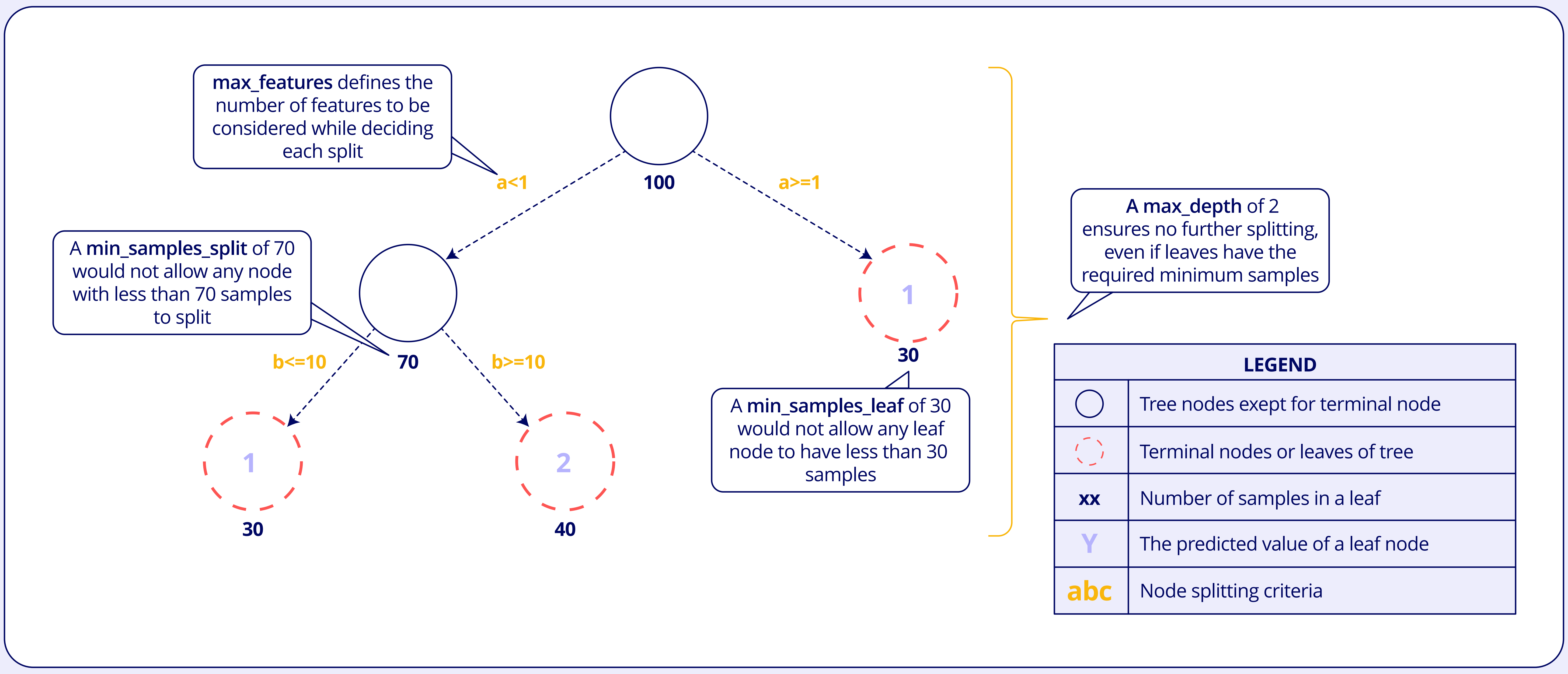


Contours of RSS as it move away from the minimum
β_2
$\hat{\beta}$
RSS (Least square) coefficients
The lasso coefficients
β_1
The ridge regression coefficients
The penalty term (budget) shown as a constraint region
LASSO
RIDGE REGRESSION

- **Number of iterations:** Determines the maximum number of passes through the training data during optimization
- **Solver:** Specifies the optimization algorithm used. Common options include:
  - **Liblinear:** Efficient for smaller datasets and L1 regularization.
  - **Saga:** Suitable for large datasets and supports both L1 and L2 regularization.

2. Decision Trees

- **Max depth:** Controls the maximum depth of the tree. Deeper trees can capture complex patterns but are more prone to overfitting.
- **Min samples split:** Minimum number of samples required to split an internal node.
- **Min samples leaf:** Minimum number of samples required to be at a leaf node.
- **Max features:** Number of features to consider when looking for the best split.


max_features defines the number of features to be considered while deciding each split
a<1
100
a>=1
A min_samples_split of 70 would not allow any node with less than 70 samples to split
70
b<=10
b>=10
1
30
2
40
1
30
A min_samples_leaf of 30 would not allow any leaf node to have less than 30 samples
A max_depth of 2 ensures no further splitting, even if leaves have the required minimum samples

LEGEND	
○	Tree nodes exept for terminal node
◌	Terminal nodes or leaves of tree
xx	Number of samples in a leaf
Y	The predicted value of a leaf node
abc	Node splitting criteria

3. Random Forests

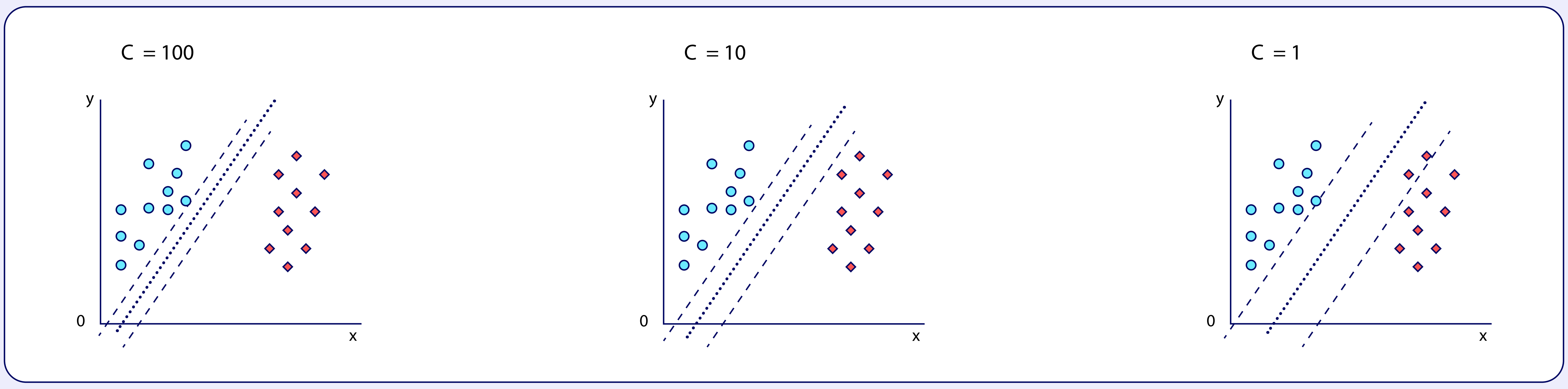
- **Number of estimators:** Number of trees in the forest.
- **Max depth:** Maximum depth of each tree.
- **Min samples split:** Minimum number of samples required to split an internal node.
- **Min samples leaf:** Minimum number of samples required to be at a leaf node.
- **Max features:** Number of features to consider when looking for the best split.
- **Bootstrap:** Indicates whether bootstrap samples are used when building trees.

4. Gradient Boosting Algorithms (e.g., XGBoost, LightGBM)

- **Learning rate:** Shrinks the contribution of each tree.
- **Number of estimators:** Number of boosting stages to perform.
- **Max depth:** Controls the maximum depth of individual trees. Deeper trees can capture complex patterns but are more prone to overfitting.
- **Min child weight:** Minimum sum of instance weight needed in a child.
- **Subsample:** Introduces randomness by randomly selecting a subset of training data for each tree.
- **Colsample_bytree:** Similar to subsample but for features, randomly selecting a subset of features for each tree.
- **Regularization:** L1 (Alpha) and L2 (Lambda) regularization help prevent overfitting by penalizing the complexity of the model.

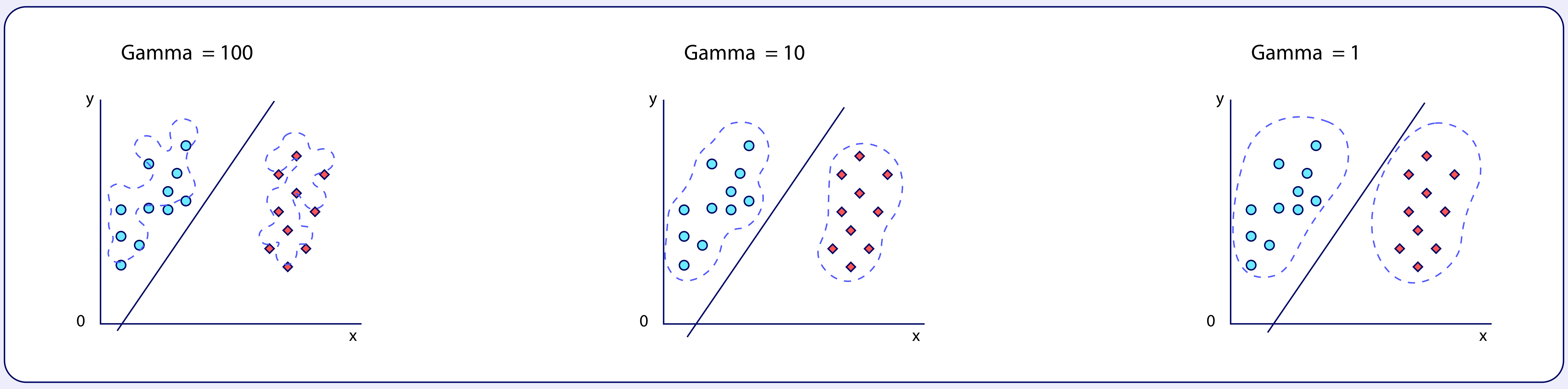
5. Support Vector Machines (SVM)

- **C:** Controls the trade-off between maximizing the margin and minimizing the classification error. A higher C value allows for less margin violation, potentially leading to overfitting.



- **Kernel:** Defines the function used to map data into a higher-dimensional space. Common kernels include:
  - **linear:** For linearly separable data
  - **poly:** Polynomial kernel, allowing for nonlinear relationships
  - **rbf:** Radial Basis Function kernel, commonly used for nonlinear data
  - **sigmoid:** Models nonlinear relationships using a function similar to the activation function in neural networks

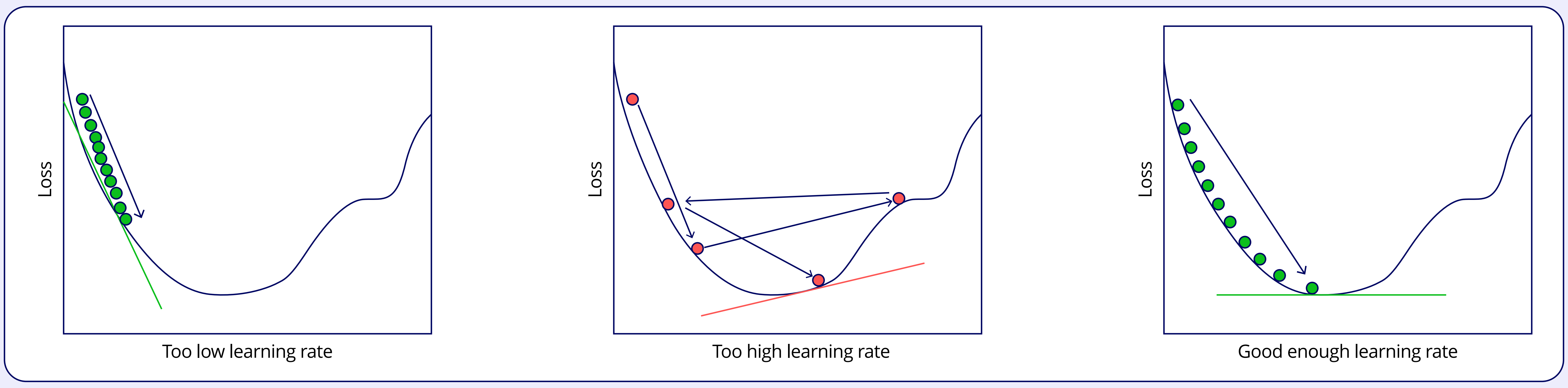
- **Gamma:** Kernel coefficient for nonlinear kernels (rbf, poly, sigmoid). It influences the shape of the decision boundary. A higher gamma leads to a more complex decision boundary.



- **Degree:** The degree of the polynomial kernel function ('poly'), where higher degrees lead to more complex models.

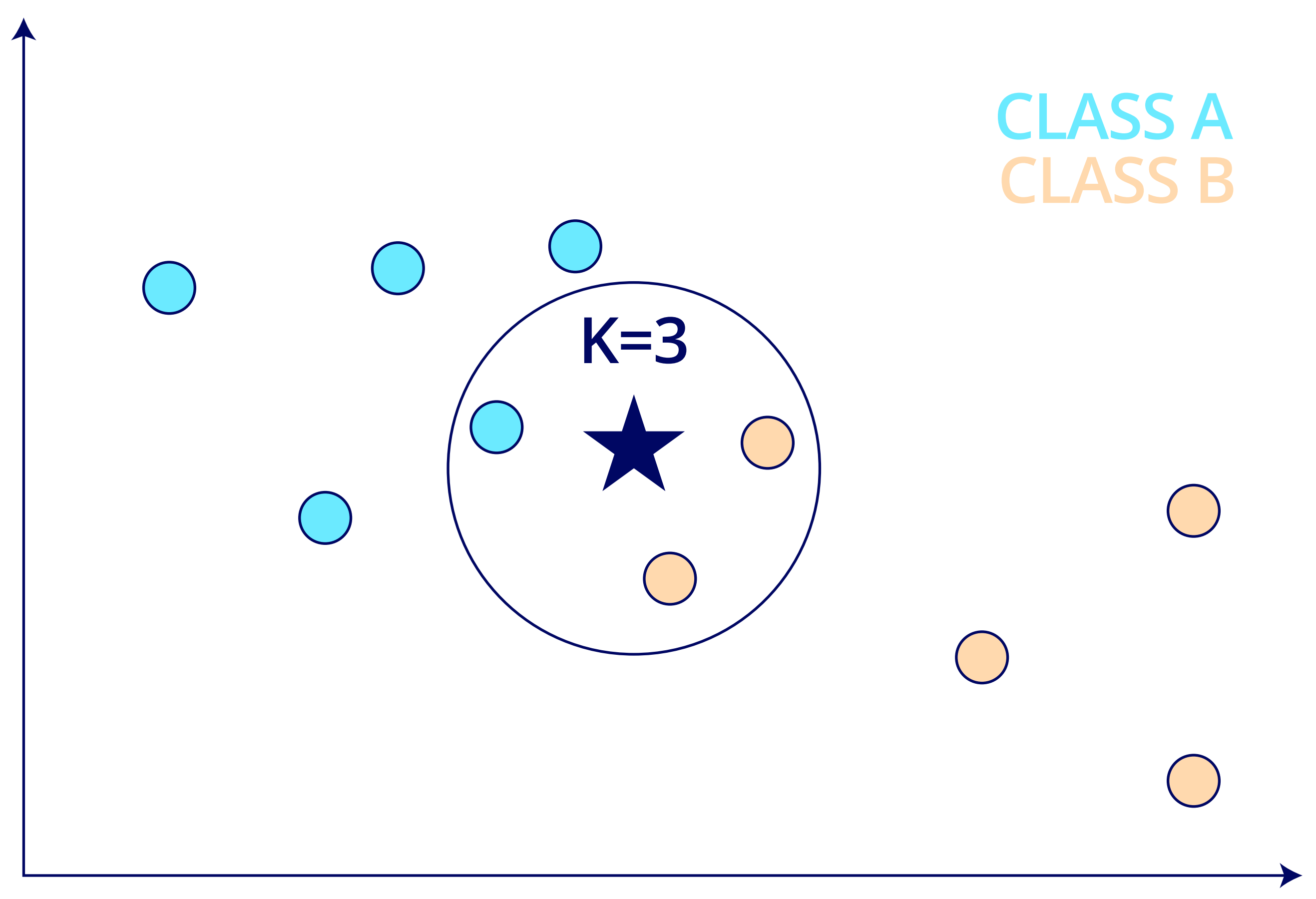
6. Neural Networks

- **Learning rate:** Step size for weight updates.
- **Batch size:** Number of samples per gradient update.
- **Epochs:** Number of passes through the training data.
- **Number of layers/units:** This refers to the architecture of the network. The number of layers and units per layer determines the model's capacity to learn and represent data.
- **Activation functions:** Functions applied to the output of each layer to introduce nonlinearity into the model.
- **Dropout rate:** Fraction of units to drop during training to prevent overfitting.
- **Optimizer:** Algorithm for optimizing the weights (e.g., SGD, Adam).



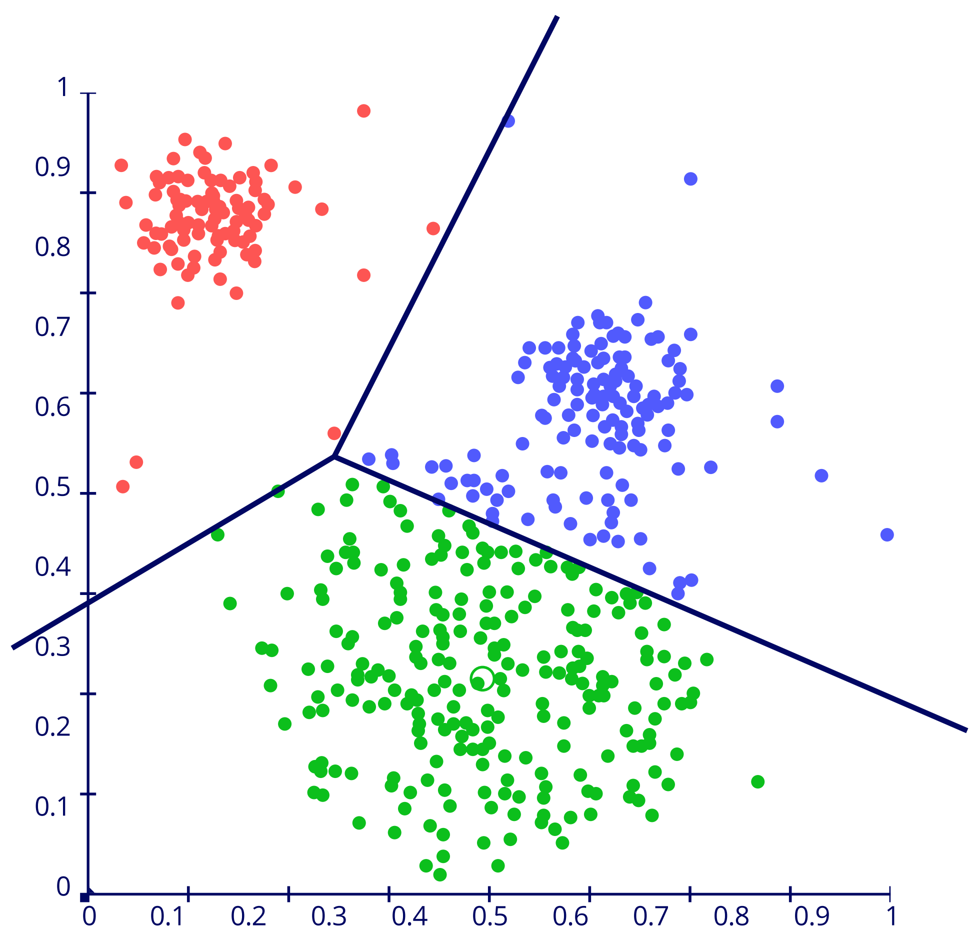
7. K-Nearest Neighbors (KNN)

- **Number of neighbors (K):** This determines the number of data points used for prediction. Lower K value is more sensitive to noise, while a higher K value smooths decision boundaries but may lead to underfitting.
- **Weights:** This specifies how to weigh the contributions of the neighbors.
  - **uniform:** All neighbors contribute equally.
  - **distance:** The weight of each neighbor is inversely proportional to its distance from the query point.
- **Metric:** This defines how distance is calculated between data points. Common metrics include:
  - **Euclidean:** Straight-line distance between points.
  - **Manhattan:** Sum of absolute differences between corresponding coordinates.
  - **Minkowski:** A generalized distance metric that includes both Euclidean and Manhattan distances as special cases.
  - **Hamming:** Measures the number of positions at which the corresponding elements are different. It is particularly useful when dealing with binary vectors.



8. K-Means Clustering

- **Number of clusters (K):** This parameter determines the number of clusters into which the data points are grouped.
- **Initialization method:**
  - **k-means++:** Uses a heuristic to place the initial centroids far apart.
  - **random:** Initial centroids are chosen randomly.
- **Distance metric:**
  - **Euclidean:** Measures the straight-line distance between points. Most common and default metric for K-means due to its sensitivity to the mean values of clusters.
- **Maximum iterations:** Sets the maximum number of iterations the algorithm will run before stopping.
- **Tolerance:** Determines the threshold for convergence. If the change in cluster centroids is less than this value, the algorithm stops.

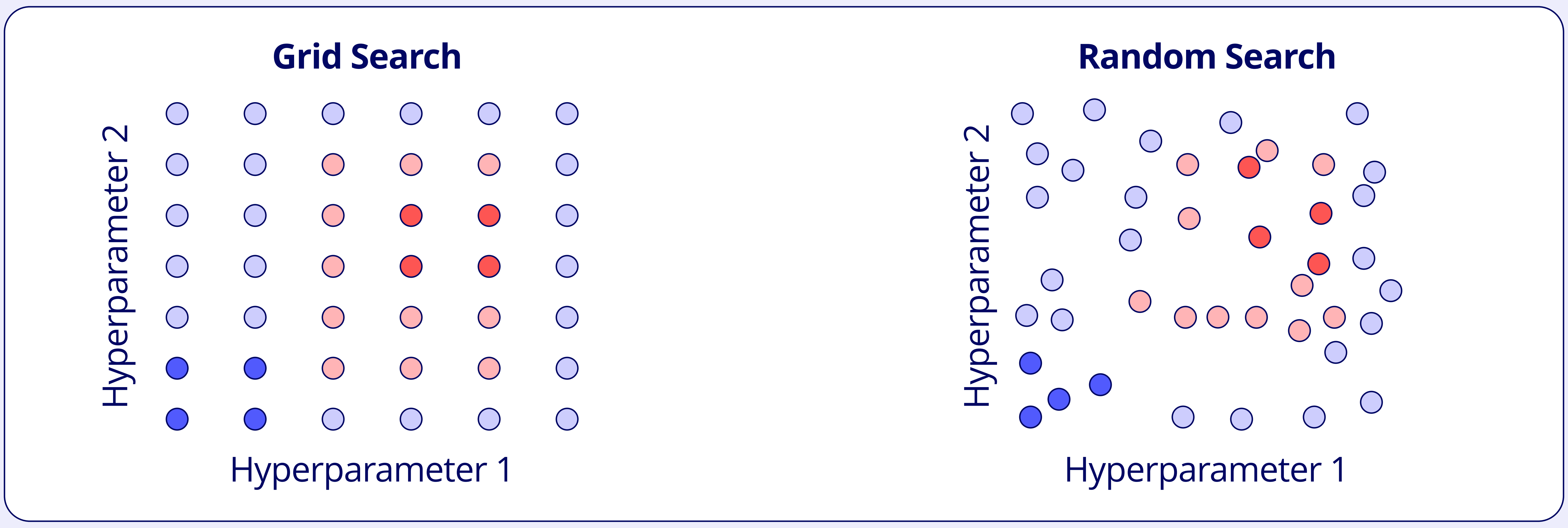


Hyperparameter Tuning

educative

Hyperparameter Tuning Methods

- **Grid search:** Uses exhaustive search over a predefined parameter grid.
  - **Pros:** It's simple to implement.
  - **Cons:** It is computationally expensive, does not scale well with large parameter spaces.
- **Random search:** Randomly samples parameters from a specified distribution.
  - **Pros:** It's more efficient than grid search, better coverage of parameter space.
  - **Cons:** It may miss optimal parameters.



- **Bayesian optimization:** Builds a probabilistic model and use it to find the best hyperparameters.
  - **Pros:** It's efficient and finds good hyperparameters with fewer evaluations.
  - **Cons:** It's more complex to implement.
- **Hyperband:** Combines random search and early stopping to quickly identify top models.
  - **Pros:** It's efficient, balances exploration and exploitation.
  - **Cons:** It may require a large initial budget.
- **Genetic algorithms:** Uses evolutionary strategies to optimize parameters.
  - **Pros:** It's good for large and complex search spaces.
  - **Cons:** It's computationally expensive.

Practical Tips

- **Start simple:** Begin with a smaller subset of hyperparameters.
- **Use cross-validation:** Ensures robustness of results.
- **Track results:** Use tools like TensorBoard, MLflow, or WandB.
- **Monitor overfitting:** Compare training and validation performance.
- **Automate tuning:** Utilize frameworks like Optuna, Hyperopt, or Keras Tuner for efficient hyper-parameter search.

Popular Libraries for Hyperparameter Tuning

- **Scikit-learn:** GridSearchCV, RandomizedSearchCV
- **Hyperopt:** For Bayesian optimization
- **Optuna:** Advanced hyperparameter optimization framework
- **Keras Tuner:** For tuning hyperparameters in Keras/TensorFlow models

Workflow Summary

Define search space:
List out all hyperparameters and their possible values.

Choose tuning method:
Select grid search, random search, Bayesian optimization, etc.

Run search:
Execute the search to find the best hyperparameters.

Evaluate results:
Analyze the performance using cross-validation scores.

Select best model:
Use the hyperparameters that yielded the best performance.

Validate model:
Confirm results on a separate validation/test dataset.