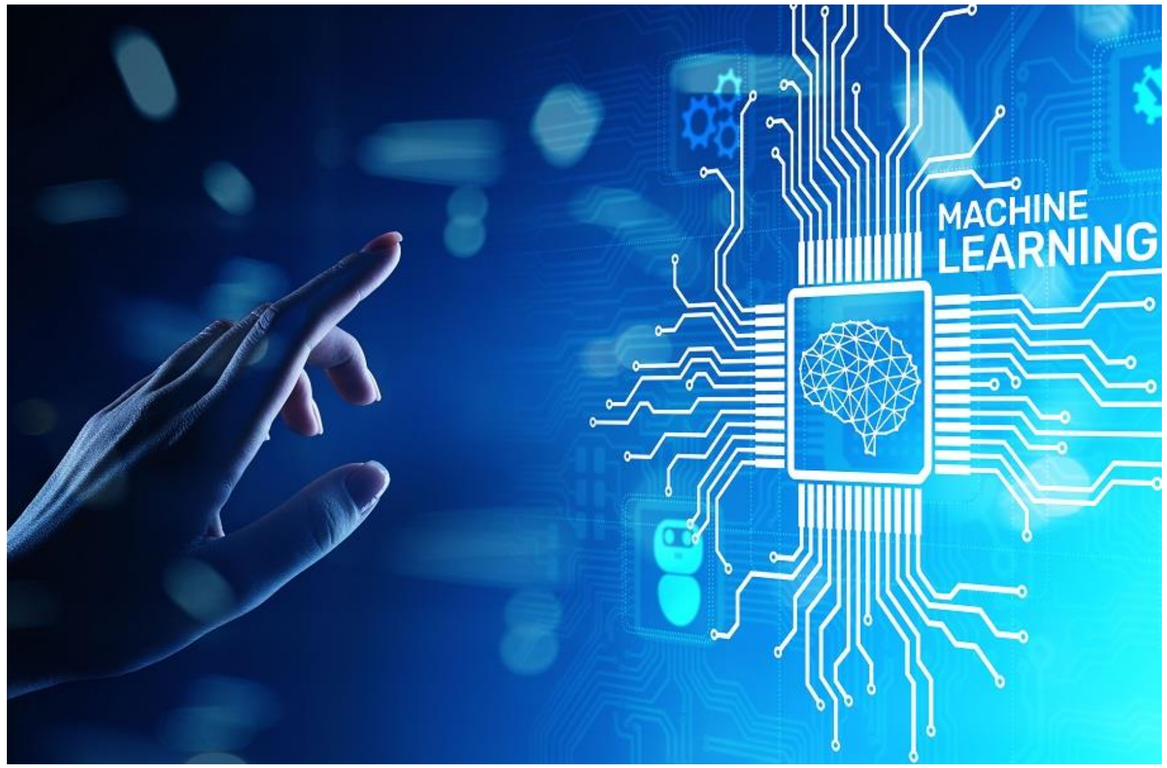
Hyperparameter Tuning



Learning for Today (20.1.2026)

- Hyperparameters in ID3
- Hyperparameters in CART (Classification And Regression Trees)
- Why Hyperparameter Tuning?
- Methods
- Performance Metrics of ML models

Hyperparameters

- Settings you choose before training that control how the tree is grown, its complexity, and its stopping behavior

- Not directly learned from data, but strongly affect overfitting vs underfitting

Hyperparameters in ID3 (Iterative Dichotomiser 3)

- Splitting Criterion
  - Based on Entropy → fixed
- Maximum Tree Depth
  - Helps prevent overfitting
- Minimum Samples to Split
  - Stops growth on very small subsets
- Minimum Information Gain
  - Split only if information gain ≥ threshold
- Handling Continuous Features
  - Discretization strategy
  - Number of bins / split points
- Pruning Parameters (if post-pruning is used)
  - Confidence threshold
  - Validation-set size

Hyperparameters in CART (Classification And Regression)

- Splitting Criterion
  - Based on Gini
- Maximum Tree Depth
  - Helps prevent overfitting
- Minimum Samples to Split
  - Stops growth on very small subsets
- Minimum Samples at Leaf
  - Split only if information gain ≥ threshold
- Maximum Number of Leaf Nodes
- Maximum Features
- Pruning Hyperparameter (if post-pruning is used)

Why Hyperparameters needs to be tuned?

- Deep Tree → Overfitting
- Shallow Tree → Underfitting
- Important hyperparameters considered here

Hyperparameter	Effect
max_depth	Controls tree depth
min_samples_split	Minimum samples to split a node
min_samples_leaf	Minimum samples in a leaf
criterion	Gini / Entropy (classification)
ccp_alpha	Post-pruning strength

Techniques for hyperparameter tuning

- Try different hyper parameter
  - Eg.. max_depth → choose the one with best validation score → completed last lab (lab 4)
- k-Fold Cross-Validation
- Grid Search (Systematic search)
- Random Search (Efficient for large grids)
- Cost-Complexity Pruning (CART-specific)

What is k-fold cross validation?

- Split data into k folds

For each hyperparameter set:

  - Train on k−1 folds
  - Validate on remaining fold

- Average performance

- Select best hyperparameters

Sample 1: K-fold validation

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold,
cross_val_score
import numpy as np

/* Load iris dataset into X and y */

/* Create decision tree classifier with Gini Index*/

kfold = KFold(
n_splits=5,
shuffle=True,
random_state=42
)
```

```
from sklearn.datasets import load_iris
scores = cross_val_score(
dt,
X,
y,
cv=kfold,
scoring="accuracy"
)
/*print accuracy, mean accuracy and standard
deviation*/
```

Exercise 1

1. Apply different depths to the above code calculate the mean cross validation accuracy. Display the below content

   depth -- Mean cross validation accuracy

Sample 2 :Building Decision Tree – Regression and MSE

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
model = DecisionTreeRegressor(max_depth=5)
scores = cross_val_score(
    model, X, y,
    cv=5,
    scoring="neg_mean_squared_error"
)
mse = -scores.mean()
print("Mean MSE:", mse)
```

Exercise 2

Create a decision Regression tree for the below dataset and display MSE for different depths

https://www.kaggle.com/c/house-prices-advanced-regression-techniques

What is Grid Search?

- Exhaustive search method
- Tries all combinations of chosen hyperparameter values using a validation metric
- Pros:
  - Easy
  - Systematic
  - Work with any model and metric
- Cons:
  - Computationally expensive

Sample 3: Grid Search

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, KFold

/* Load iris dataset into X and y */

dt = DecisionTreeClassifier(random_state=42)

param_grid = {
'max_depth': [2, 3, 4, 5, 6, None], # depth of tree
'min_samples_split': [2, 5, 10], # min samples to split a node
'min_samples_leaf': [1, 2, 4], # min samples at a leaf node
'criterion': ['gini', 'entropy'] # splitting criterion
}

kfold = KFold(n_splits=5, shuffle=True, random_state=42)

grid_search = GridSearchCV(
estimator=dt,
param_grid=param_grid,
cv=kfold,
scoring='accuracy', # can change to F1, precision, etc.
n_jobs=-1 # use all CPU cores
)
grid_search.fit(X, y)
print("Best hyperparameters:", grid_search.best_params_)
print("Best cross-validation accuracy:", grid_search.best_score_)
```

Sample 4: Grid Search - Regression Tree

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import numpy as np

# Load data (update the path to where you saved train.csv)
train_df = pd.read_csv('train.csv')

# Target
y = train_df['SalePrice']

# Features (drop target and ID)
X = train_df.drop(columns=['SalePrice', 'Id'])

# Identify categorical and numerical columns
cat_cols = X.select_dtypes(include=['object']).columns
num_cols = X.select_dtypes(exclude=['object']).columns

# Preprocess: OneHotEncode categories + pass through numeric unchanged
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols),
    ('num', 'passthrough', num_cols)
])
# Decision Tree Regressor
dt_reg = DecisionTreeRegressor(random_state=42)

# Hyperparameter grid to search
param_grid = {
    'model__max_depth': [3, 5, 7, 10, None],
    'model__min_samples_split': [2, 5, 10, 20],
    'model__min_samples_leaf': [1, 2, 5, 10]
}
```

Sample 4: Grid Search - Regression Tree

```
# Pipeline: preprocessing + model
pipeline = Pipeline([
    ('preprocess', preprocessor),
    ('model', dt_reg)
])
# 5-fold cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    scoring='neg_root_mean_squared_error',  # minimize RMSE
    cv=kfold,
    verbose=2,
    n_jobs=-1
)
grid_search.fit(X, y)
print("Best hyperparameters:", grid_search.best_params_)
print("Best CV RMSE:", -grid_search.best_score_)
```

Exercises

4.

https://www.kaggle.com/competitions/titanic/data

For the above dataset, Predict whether a passenger survived using decision tree classification and hyper parameter tuning with GridSearch

5.

https://www.kaggle.com/datasets/uciml/adult-census-income/code

For the above dataset, Predict Predict whether income exceeds $50K using decision tree classification and hyper parameter tuning with GridSearch

Other Searches

- Randomized Search → randomly samples hyperparameter combinations
  - Not guaranteed to find the absolute best combination
  - Not as computationally intensive as Exhaustive search
- Bayesian Optimization → Uses a probabilistic model (like Gaussian Process) to model the relationship between hyperparameters and performance
- Genetic Algorithms / Evolutionary Search → Treats hyperparameters like a “genome” and uses selection, crossover, mutation to evolve the best combination.

Optional code

```
/*Build the decisiontree */

from scipy.stats import uniform

param_dist = {

    'ccp_alpha': uniform(0.0, 0.05)

/ *define kfold*/

rand_search = RandomizedSearchCV(

    estimator=dt,

    param_distributions=param_dist,

    n_iter=10,              #  10 random values of ccp_alpha

    cv=kfold,

    scoring='neg_mean_squared_error',  # minimize MSE

    random_state=42,

    n_jobs=-1

)

rand_search.fit(X, y)


print("Best ccp_alpha:", rand_search.best_params_['ccp_alpha'])

print("Best CV RMSE:", np.sqrt(-rand_search.best_score_))
```