04 · Supervised Learning with scikit-learn

The API

```
model = Pipeline([("pre", preprocessor()),
("clf",
LogisticRegression(max_iter=1000))])
model.fit(X_train, y_train)
# learn
model.predict(X_test)
# labels
model.predict_proba(X_test)[:, 1]
# P(positive) — use for AUC, thresholds
model.score(X_test, y_test)
# accuracy (classifiers) / R² (regressors)
joblib.dump(model, "model.joblib");
joblib.load("model.joblib")
```

Models

Model	Use	Scale?	Key hyperparameters
`DummyClassifier(strategy="most_frequent")`	baseline	no	
`LinearRegression`, `Ridge`, `Lasso`	numeric	yes	`alpha`
`LogisticRegression`	probabilities	yes	`C`, `penalty`
`DecisionTreeClassifier`	readable rules	no	`max_depth`, `min_samples_leaf`
`RandomForestClassifier`	robust default	no	`max_features`
`HistGradientBoostingClassifier`	best on most tables	no	`learning_rate`, `max_iter`, `max_depth`

Trees

- Gini = 1 − Σ p_k^2 (0 pure, 0.5 for 50/50). Split = largest weighted impurity drop.
- Forest = bagging (bootstrap rows + random features), averaged → lower variance.
- Boosting = sequential shallow trees fitting residuals, scaled by learning rate → lower bias.

Fit diagnosis

Train	Test	Diagnosis	Fix
low	low	underfitting	richer model, better features
high	much lower	overfitting	limit depth, min_samples_leaf, regularize, more data
high	high, close	good	ship after Area 05 checks

Tuning

```
grid = GridSearchCV(pipe,
{"clf__max_depth": [2, 4, 8, None]}, cv=5,
scoring="roc_auc")
grid.fit(X_train, y_train);
grid.best_params_; grid.score(X_test,
y_test)
```

- Pipeline params: `step__param`. Search on train only; test once.
- Importance: `permutation_importance(model, X_test, y_test, scoring="roc_auc")`.