Start Date	@8 December 2025
Weeks	Week 6

Week 6 - ML

Best Practices For Model Evaluation + Hyperparameter Tuning

Streamlining Workflows With Pipelines

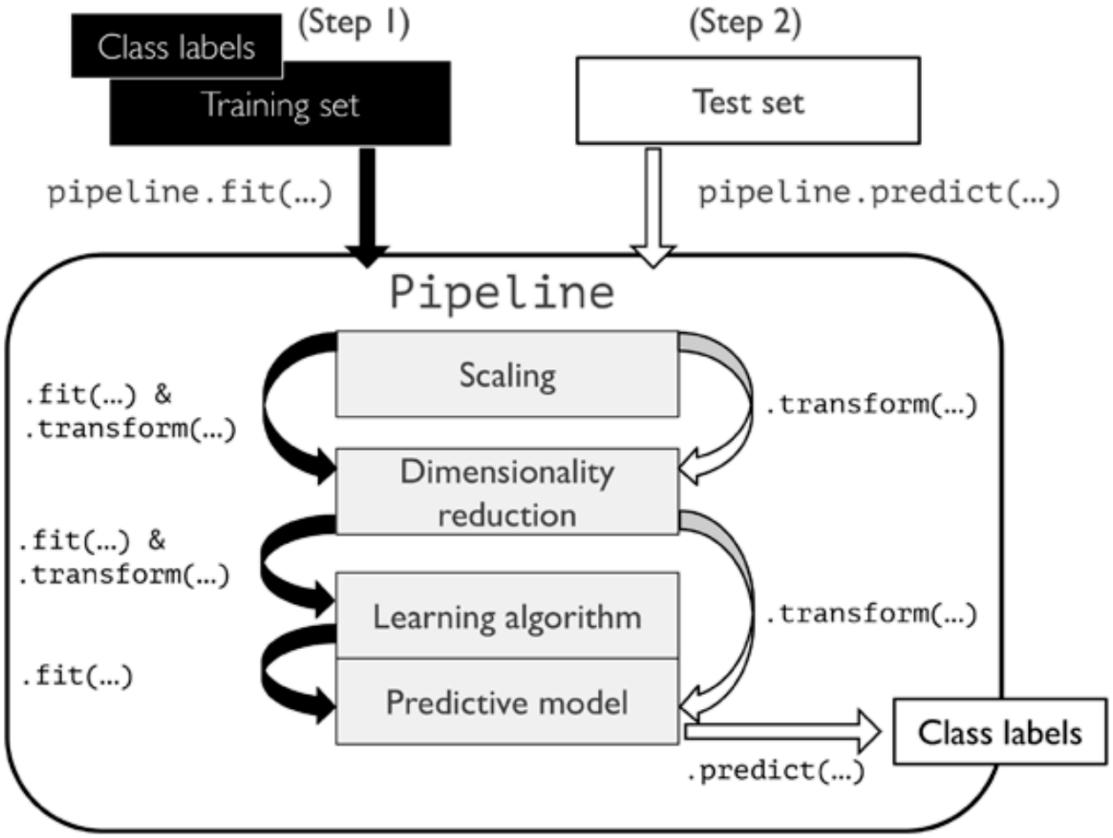
- `Pipeline` class → Scikit-learn
- Steps:
  1. Read dataset `pd.read_csv()`
  2. `LabelEncoder` object → transform class labels from string to integer
  3. Check with transform → correctly fitted
  4. Split train - test from dataset

Combining Transformer + Estimator In A Pipeline

- Breast cancer dataset
  - Need to be standardised → measured at various different scales
  - If we want to compress data to lower two-dimensional subspace (PCA) + Dimensionality reduction
    - Instead of splitting model fitting + data transformation → chain → `StandardScaler` + `PCA` + `LogisticRegression` object in pipelien
- `make_pipeline` → Takes arbitrary number of transformers (support fit + transform method as input) + estimator (fit + predict method) ← Scikit-learn
  - Meta-estimator or wrapper around individual transformers + estimators
  - Estimator fitted on transformed training data

⭐ No limit to number of steps in pipeline, but if pipeline is used for prediction, the last element must be a estimator

- Pipelines are a very useful wrapper tool



Inside of a Pipeline object

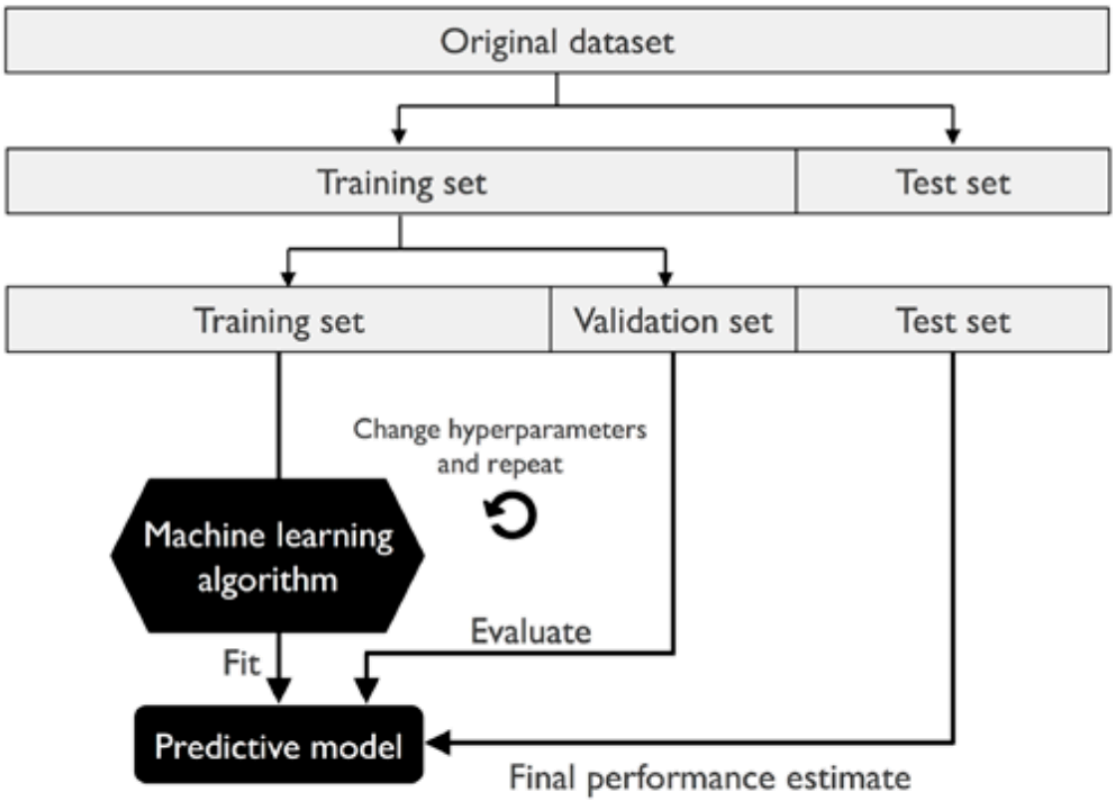
K-Fold Cross-validation To Assess Model Performance

- Common cross-validation technique → holdout cross-validation + k-fold cross-validation
  - Helps obtain reliable estimates of models' generalisation performance → how well model performs on unseen data

⭐ Hyperparameter → Tuning parameter

Holdout Method

- Classic + popular method
- Model selection → Classification problem → select optimal values for tuning hyperparameters
- Usually people split dataset into training and test, then use the same test dataset over again during model selection. Like this the model will overfit
- **Holdout Method → We split into 3 ways - training, validation, test**
    - Training → fit different models
    - Validation → model selection
    - Testing → Test
        - Data hasn't been seen during training + model selection, therefore model has less bias to generalise new data



Holdout Method

Disadvantage:

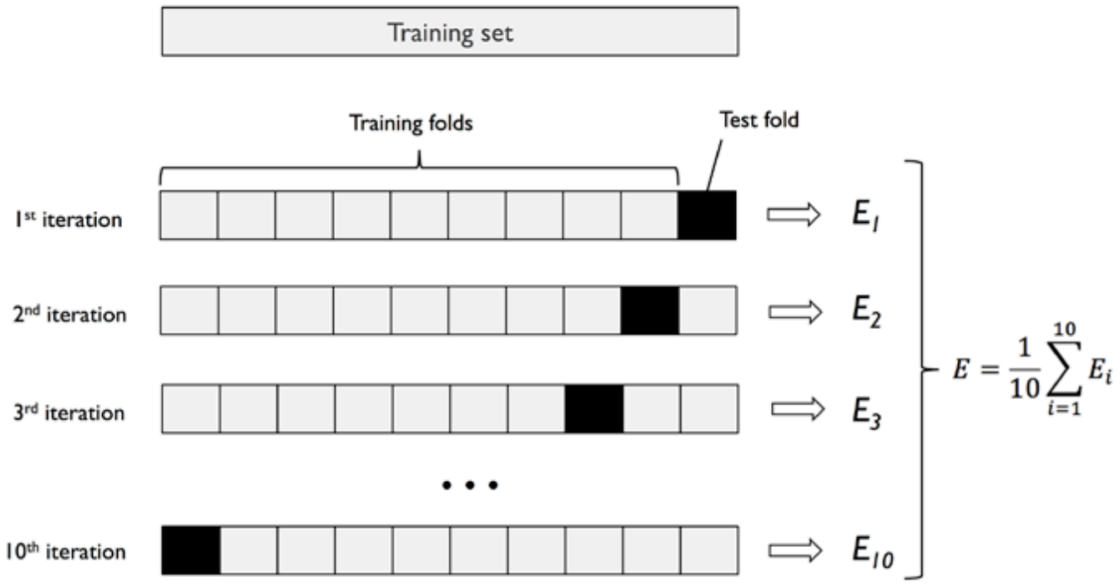
- Performance estimate may be very sensitive → This really depends on the partition of test + validation subset.

K-Fold Cross-Validation

- Training data set is split into k folds without replacemenet
- K-1 folds
  - Training folds → used for model training
  - Test fold → performance evaluation
- Procedure repeated k times → obtain k models + performance estimates
- Calculate average performance of models with different + independent test folds, which obtains performance estimates → Less sensitive to sub-partitioning of training data, compared to holdout method

K-fold typically used for model tuning → Optimal hyperparameter values → yields satisfying generalisation performance

- Retrain model with complete training dataset, after finding satisfactory hyperparameters
  - First → interested in single, final model
  - Second → more training examples for learning algorithm
- K-fold cross-validation is a resampling technique without replacement. It's advantage is that for each iteration, each example is used exactly once!
- All test folds are disjoint → No overlap between test folds



K-fold Cross-validation

How does k-fold work with, $k = 10$

- K-fold uses dataset better than holdout → all data points used for evaluation
- Good standard for k → 10

> ⭐ **Small dataset → better to use more folds.**

  - More k = more training data used in each iteration, which results in slower computation and noisier estimate

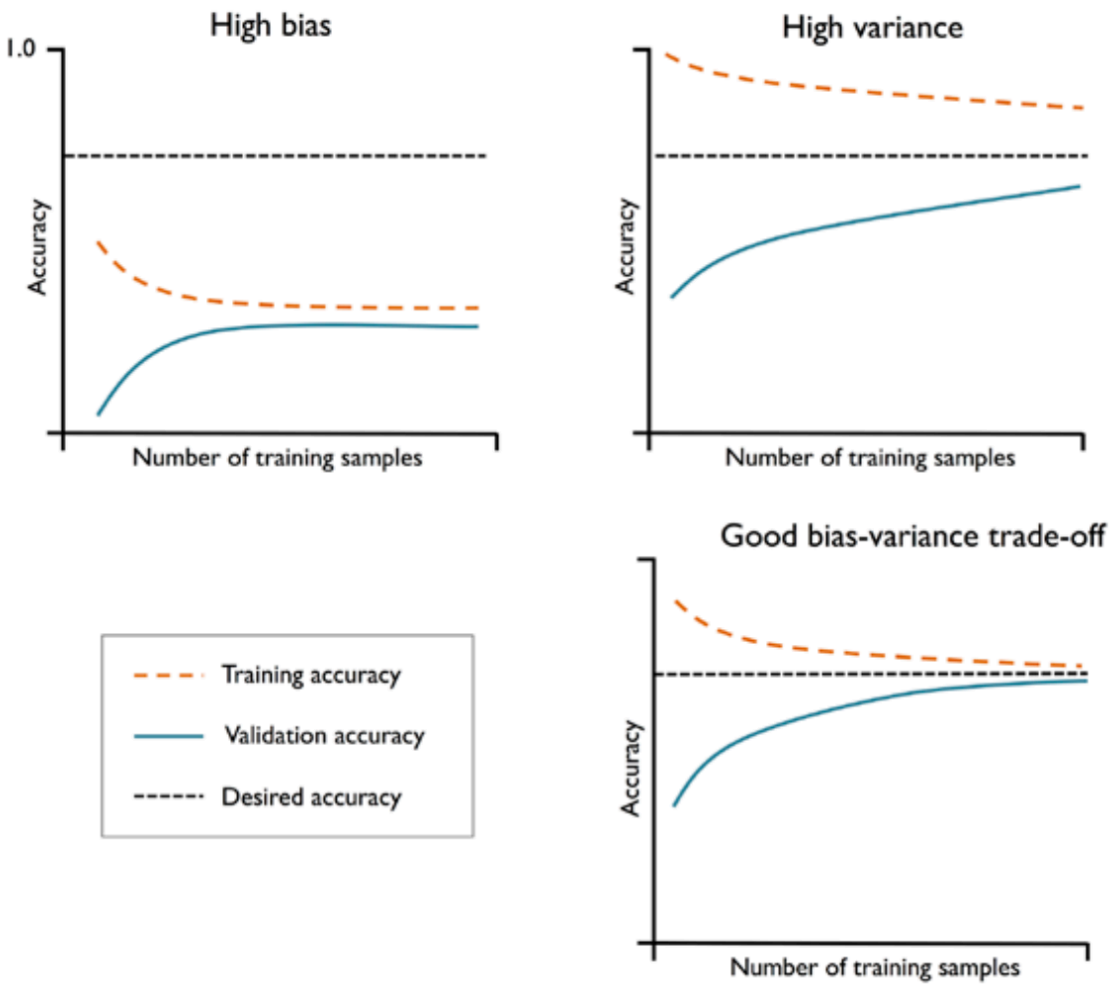
> ⭐ **Large dataset → use smaller k (e.g. 5) → enough for good performance + computation is low**

Debugging with Learning + Validation curves

- **Learning curves** → does learning algorithm have problems with overfitting (high variance) or underfitting (high bias)
- **Validation curves** → Helps address common learning algorithm problems

Bias + Variance With Learning Curves

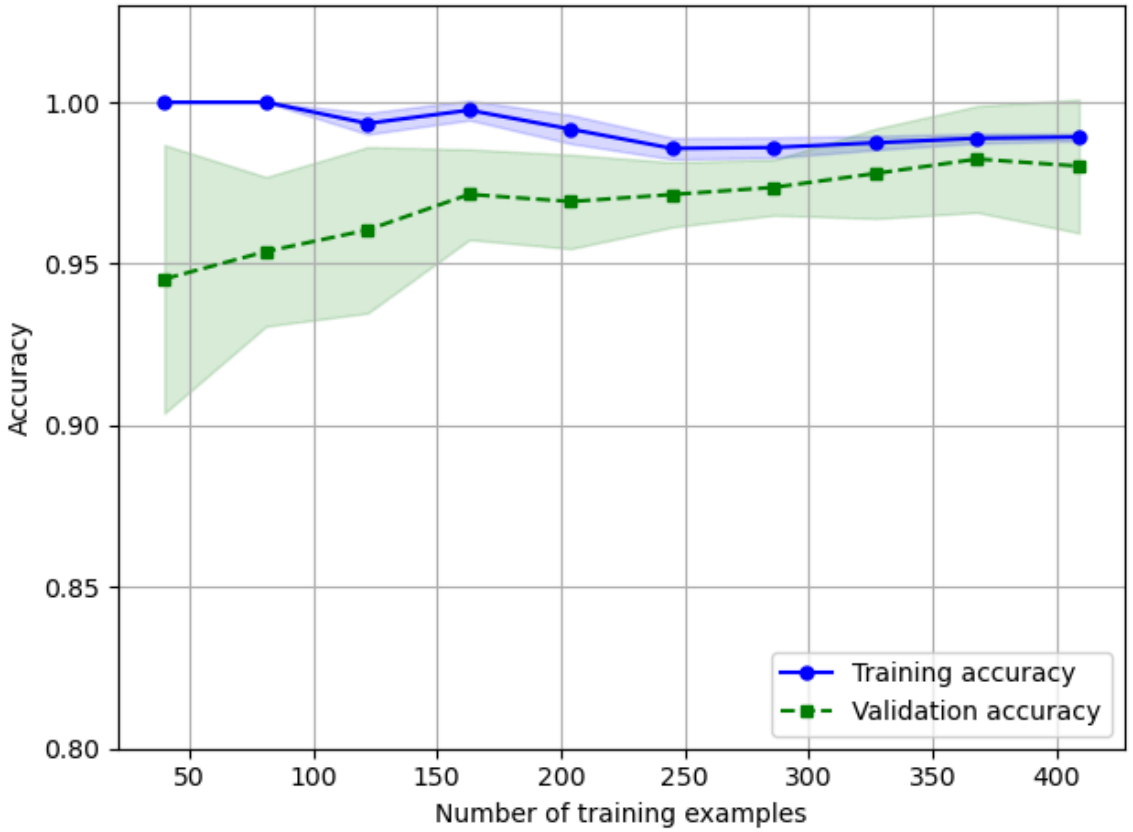
- Complex model for training data → model will tend to overfit, therefore GET MORE DATA



Model issues

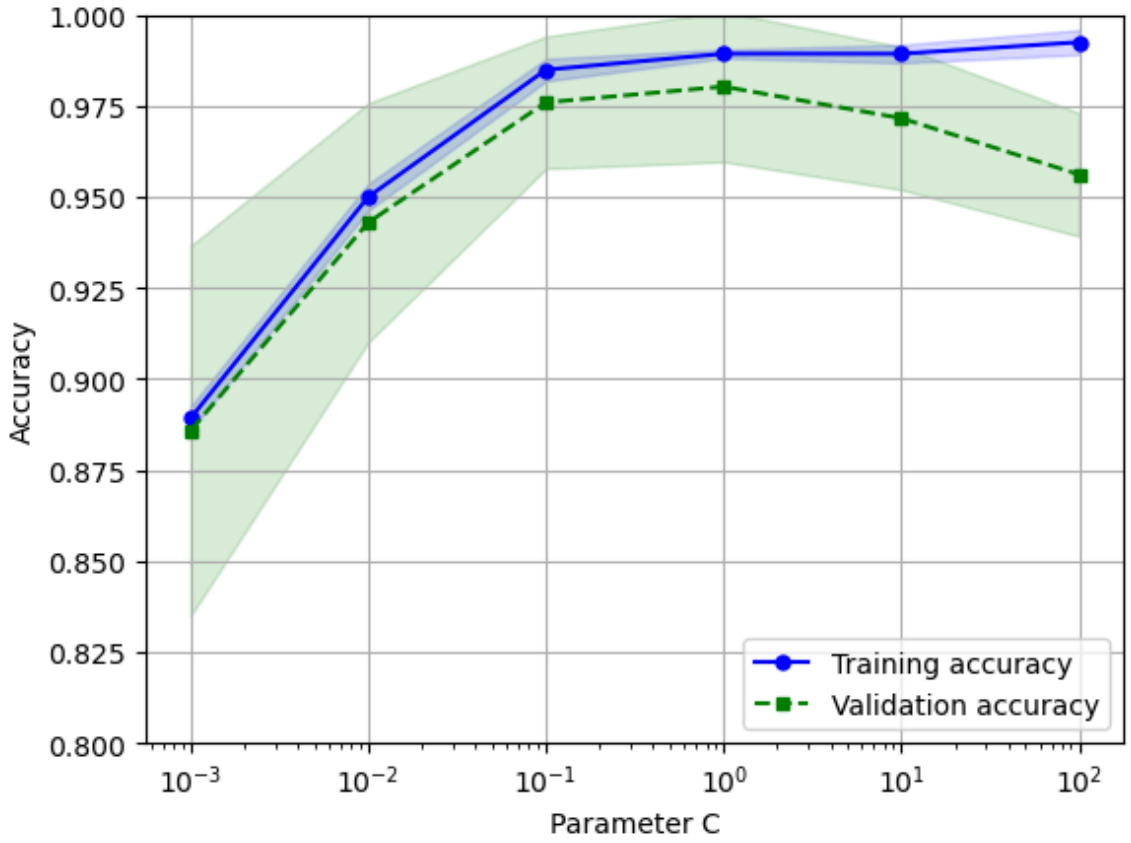
- Top left → Underfit
  - **Solution:**
    - Increase number of model parameters
    - Construct additional features
    - Decrease degree of regularisation
- Top right → Overfit
  - **Solution:**
    - Collect more training data
    - Reduce model complexity

    - Increase regularisation
- Unregularised models → Decrease number of features → Use feature selection or feature extraction



- `train_sizes = np.linspace(0.1, 1.0, 10)` = 10 evenly spaced dataset sizes
- `learning_curve = default` = K-fold cross-validation → k=10

Addressing over and underfitting with validation curve



- Validation curve = Vary the values of the model parameters
- Example above = inverse regularisation parameter, C
  - Validation curve plot for sum parameter C
  - `validation_curve` = Stratified k-fold cross-validation → default → estimate performance
  - Inverse regularisation of logistic regression → `logisticregression__C`
  - Plotted average training + cross-validation accuracies

Fine-tuning Machine Learning Model via Grid Search

- Machine Learning → We have two types:
  1. Learned from training data + learning algorithm that are optimised separately
  2. Tuning parameters
- Popular hyperparameter optimising technique → **Grid Search** → help improve performance of model

Tuning Hyperparameter via Grid Search

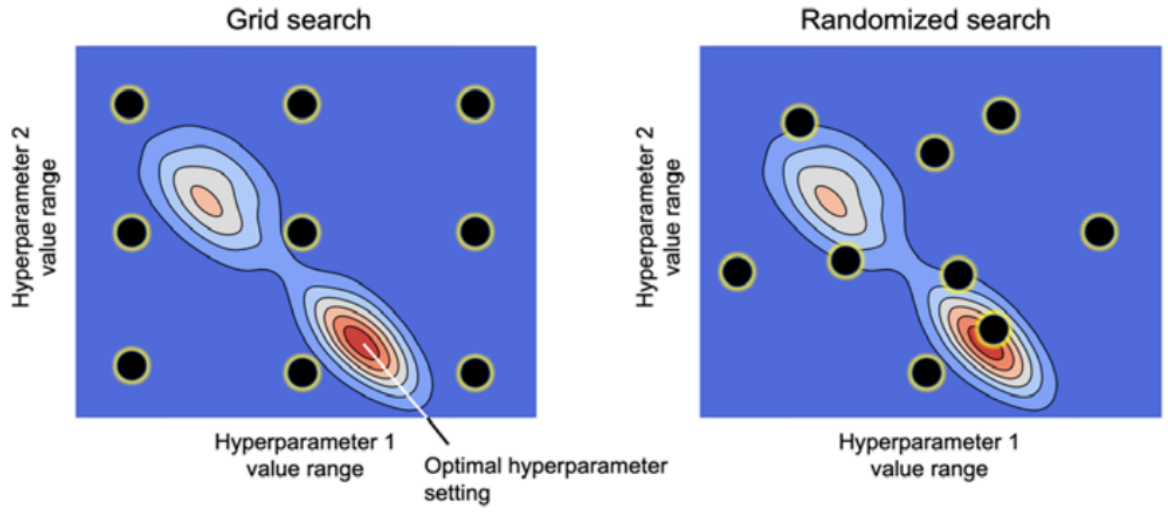
- Pretty simple. We brute-force exhaustive search paradigm
    - List values for different hyperparameters → Computer evaluates the model performance for each combinations

Hyperparameter Configuration With Randomized Search

- Grid Search → Exhaustive Search. It's guaranteed to find optimal hyperparameter configuration.
    - Large hyperparameter grids → Grid search is expensive in practice

> ⭐ An alternative approach to Grid Search is Reandomized Search

- Randomized Approach → Draw hyperparameter config randomly from distributions
    - Allows exploration of wider range of hyperparameter in more cost and time efficient manner



Comparison of Grid vs Randomized Search

- Grid search explores discrete, user-specified choice → may miss good hyperparameter configurations → if search space is too sparse
- Randomized Search for SVM

- `RandomizedSearchCV` → Difference
  - Specify distributions as part of parameter grid
  - Specify total number of hyperparameter configurations to be evaluated

Successive Halving

- Finds suitable hyperparameters more efficiently
- Throws out unpromising hyperparameter configuration until only one configuration is left
- **Summary:**
  1. Draw large set of candidate configs via random sampling
  2. Train model with limited resources → e.g. small subset of training data
  3. Discard bottom 50% → based on predictive performance
  4. Back to Step 2

> ⭐ Repeated until one hyperparameter configuration is left

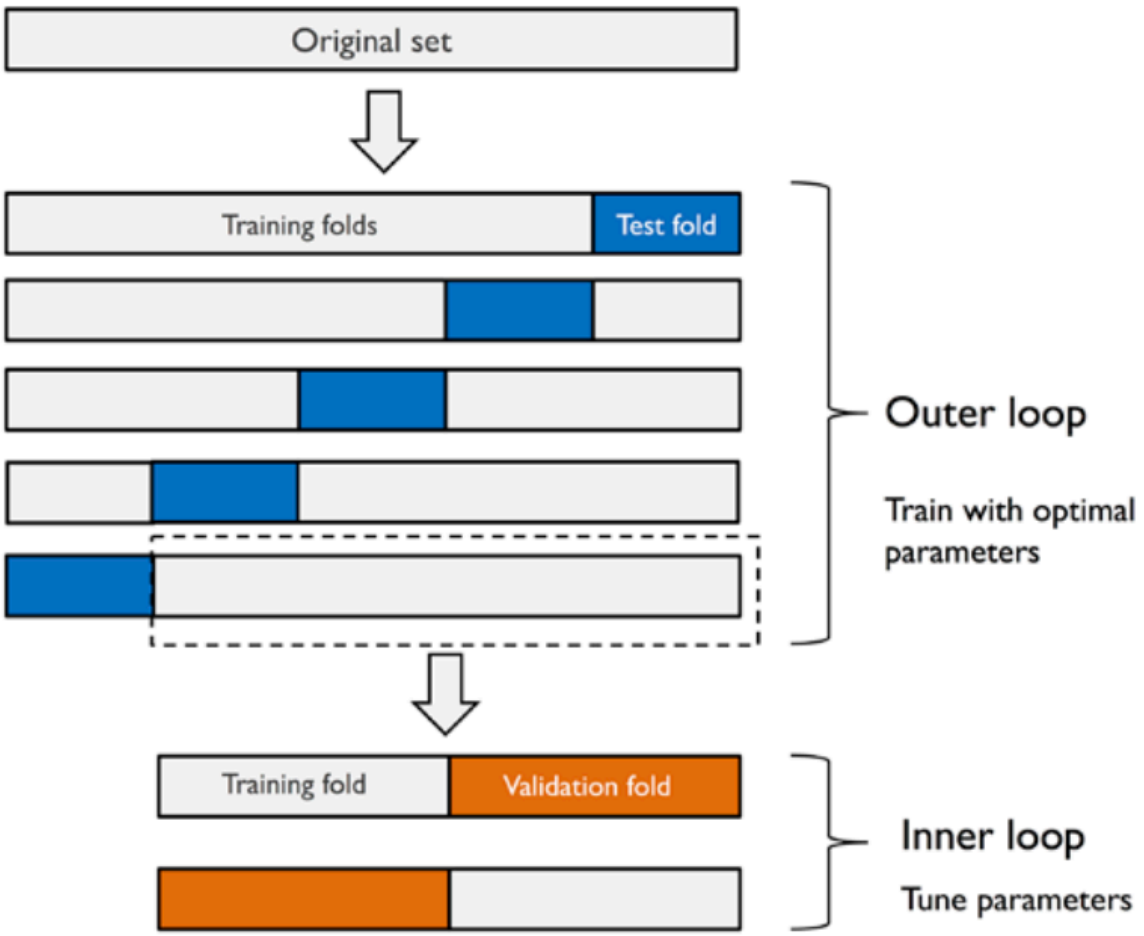
Hyperopt

- Another popular library for hyperparameter optimisation
- Implements several methods → Randomized search + Tree-Structured Parzen Estimators (TPE)
- TPE → Optimisation method → Probabilistic model → Continuously updated, based on past hyperparameter evaluation + performance

Algorithm Selection With Nested Cross-Validation

- K-fold Cross-validation + Grid/Randomized search is a useful approach for fine-tuning
- Nested Cross-validation → Outer + Inner k-fold cross-validation
  - **Outer:** Split data into training + test fold → after model selection test fold used for evaluating model performance

- **Inner:** Selects model using k-fold cross-validation



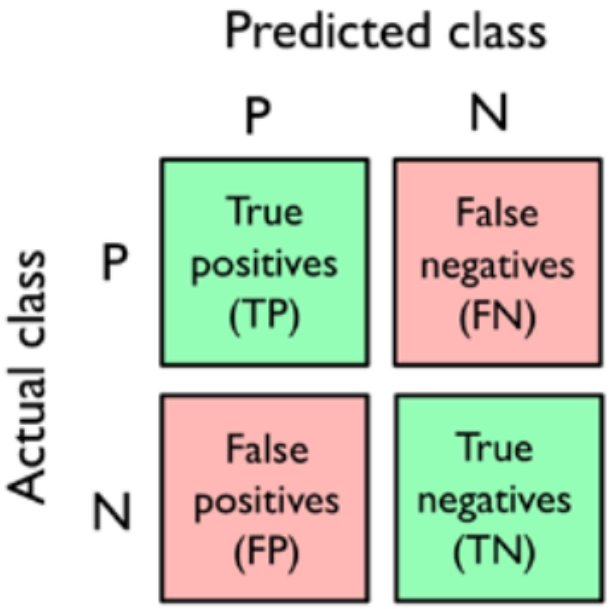
Nested Cross-validation (5×2 cross-validation)

- Average cross-validation accuracy → good estimate of what to expect if we tune hyperparameter + use on unseen data
- Works better than decision trees

Performance Evaluation Metrics

- Precision
- Recall
- F1 Score
- Matthews Correlation Coefficient (MCC)

Confusion Matrix



Confusion Matrix

- Matrix → performance of learning algorithm
- True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN)

Optimising Precision + Recall of Classification Model

- Error (ERR) + Accuracy (ACC) → Information about misclassified examples
- **Error** → Sum of false predictions / number of total predictions

$$ERR = \frac{FP + FN}{FP + FN + TP + TN}$$

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

- True Positive Rate (TPR) + False Positive Rate (FPR) → Useful for imbalanced class problems → fraction of positive examples that were correctly identified out of total pool.

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

$$TPR = \frac{TP}{N} = \frac{TP}{FN + TP}$$

- Precision (PRE) + Recall (REC)
  - **Precision** → How many records predicted as relevant are actually TP
  - **Recall** → How many relevant records are captured as TP

$$PRE = \frac{TP}{TP + FP}$$

$$REC = TPR = \frac{TP}{N} = \frac{TP}{FN + TP}$$

- **Malignant Tumor Detection:**
  - If we optimise recall → Minimise chance of not detection malignant tumor → however, increase FP
  - If we optimise precision → Emphasises correctness if we predict patient has malignant tumor → However, increases FN
- Balance Optimising PRE + REC and Mean of PRE + REC → You get **F1 Score**

$$F1 = 2\frac{PRE \cdot REC}{PRE + REC}$$

MCC → Popular in biological context

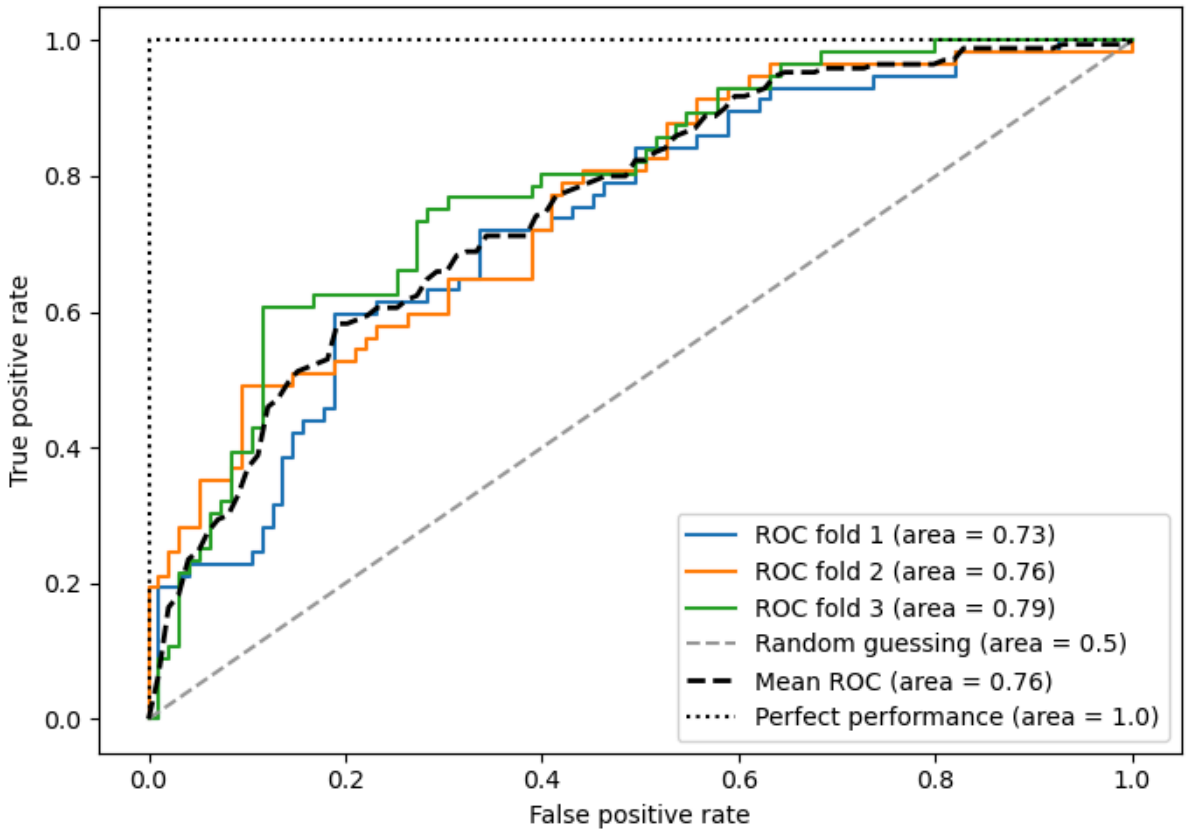
$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

- **MCC ranges between -1, 1 → others don't**
  - Takes all elements of confusion matrix into account

Plotting Receiver Operating characteristic (ROC)

- Receiver Operating characteristic (ROC) Graphs → Useful for selecting models for classification based on performance
  - Computed by shifting the decision threshold of classifier
- Diagonal ROC → Random guessing → Classification models under diagonal, are considered worse than random guessing
- **Perfect classifier** → top left-corner of graph where TPR=1 and FPR=0

- ROC area under the curve (ROC AUC) → Performance of classification model
- Precision-recall curves → for different probability thresholds



Scoring Metric for Multiclass Classification

- Scikit-learn → Macro + micro averaging methods → Extend scoring metrics to multiclass problems via OvA

$$PRE_{micro} = \frac{TP_1 + \cdots + TP_k}{TP_1 + \cdots + TP_k + FP_1 + \cdots + FP_k}$$

$$PRE_{macro} = \frac{PRE_1 + \cdots + PRE_k}{k}$$

- Micro → Useful if we want to weight each instance or prediction equally
- Macro → Weights all classes equally → evaluate overall performance of classifier with regards to most frequent class labels.

Class Imbalance

- Quite common when working with real life data
  - e.g. spam filters, fraud detection, disease screening
- Algorithm learns a model that optimises the predictions based on most abundant class in the dataset
- Imbalanced class proportions
  - Model fitting → assign larger penalty to wrong predictions on minority class
  - Upsampling minority class → downsampling majority class

Test and figure out what works for your model

- **Another option:**
  - Generation of synthetic training examples
  - **Synthetic Over-Sampling Technique (SMOTE)**
  -