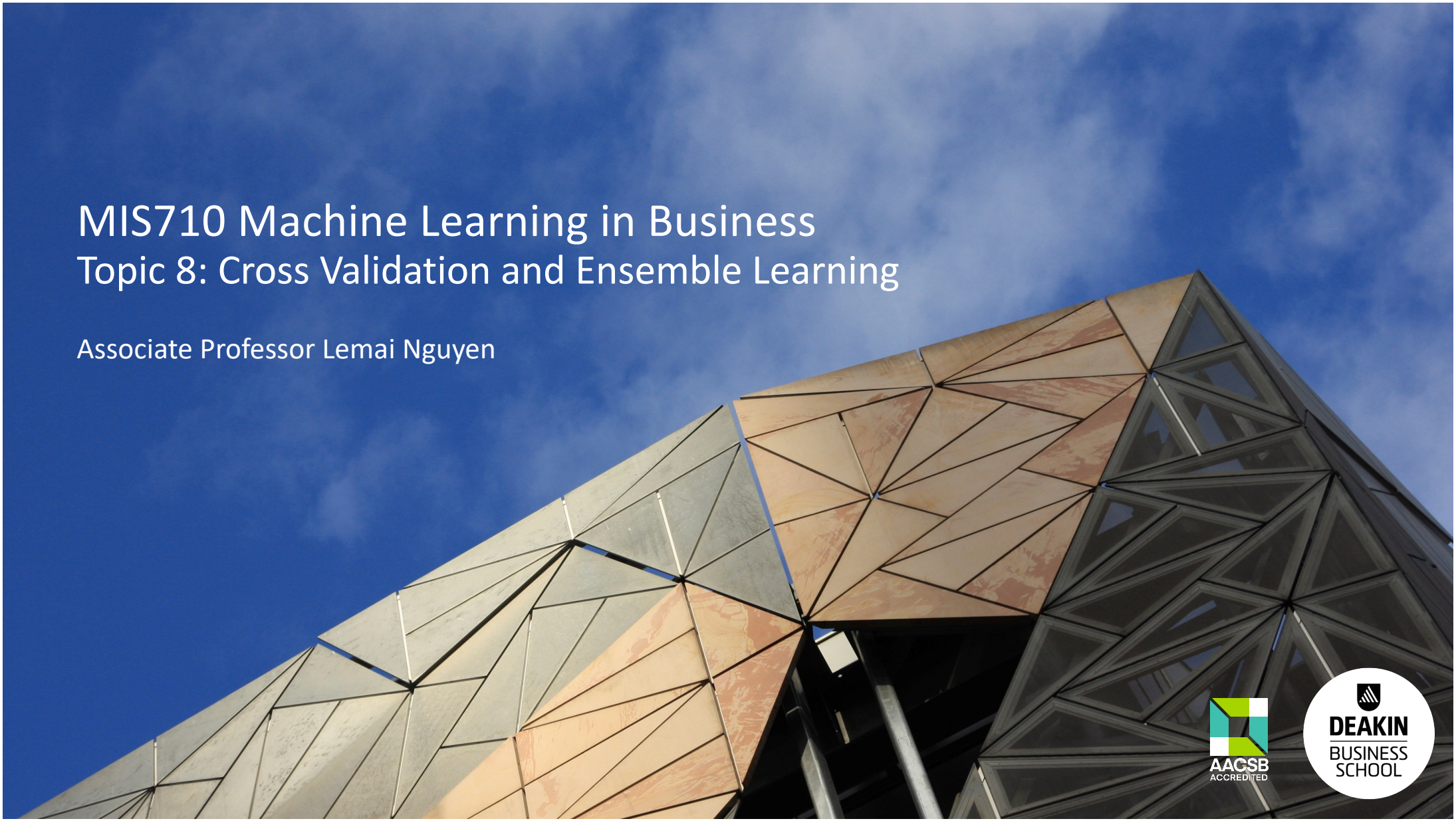
MIS710 Machine Learning in Business
Topic 8: Cross Validation and Ensemble Learning

Associate Professor Lemai Nguyen



A1 feedback – Well done!

- A clear majority of students passed.
- We were impressed by many excellent submissions!
- There was a good understanding of business requirements using BACCM.
- Many students gained strong data skills, with some showing impressive exploration, visualisation, and interpretation.
- In most submissions, fundamental ML knowledge and skills were demonstrated through model development and evaluation.

Feedforward for A2- A1 Areas for improvement

- Business Understanding: **The BACCM framework** should be used to analyse the situation.
- Data Understanding:
  - EDA:
    - Visualisation is needed to support the findings
    - Results and interpretation are required **in the report**, in addition to stats and visualisation in Python.
    - Address the **Client's requests**
    - Need to **apply critical thinking** – select variables to analyse and what to report.
  - Note for A2: Analyse the **target variable** and use an explicit function to encode the target for classification
- Model Development and Evaluation:
  - An explanation of ML approach (ML type, problem, model selection, target, dataset and process)
  - Need to discuss feature selection, why and how
  - Results must be interpreted and explained in business language, incl. trade-off among the metrics, choice of metrics (e.g., **in A2** recall for positive class).
- Recommendations for business should be aligned with specific EDA insights and model feature coefficients, contexts
- Follow the assessment description and **rubric** for requirements. **Check the HD column**
- Submission requirements: Word, PDF, and Python notebook and PDF)

Learning Analytics for Primary Schools

A2

Case Study

A2 Tasks

A2 Deliveries

- **Six** EDA questions
- **Two** supervised ML models and evaluation comparison
- **One** clustering model and cluster profiling

- **Two** reports
  - Dr. Alok Sinha, Data2Intel Director of **Data and Insights** and team
  - Sally Tran, Data2Intel Director of Education and Engagement - **business** audience

- **Two** Python files
  - ipynb and PDF version of the Python notebook

Ethical recommendations – extended FAT framework

Learning Analytics for Primary Schools

A2

Case Study

A2 Tasks

A2 Deliveries

Fairness

- Sample size of a protected sub-population
- Does the protected sub-population have same distribution of predictions as others

Accountability

- Document how you select features, handle missing data, and choose model algorithms, and impacts on model performance
- Potential implications of the model, particularly to vulnerable sub-populations, suggest ethical and privacy guidelines

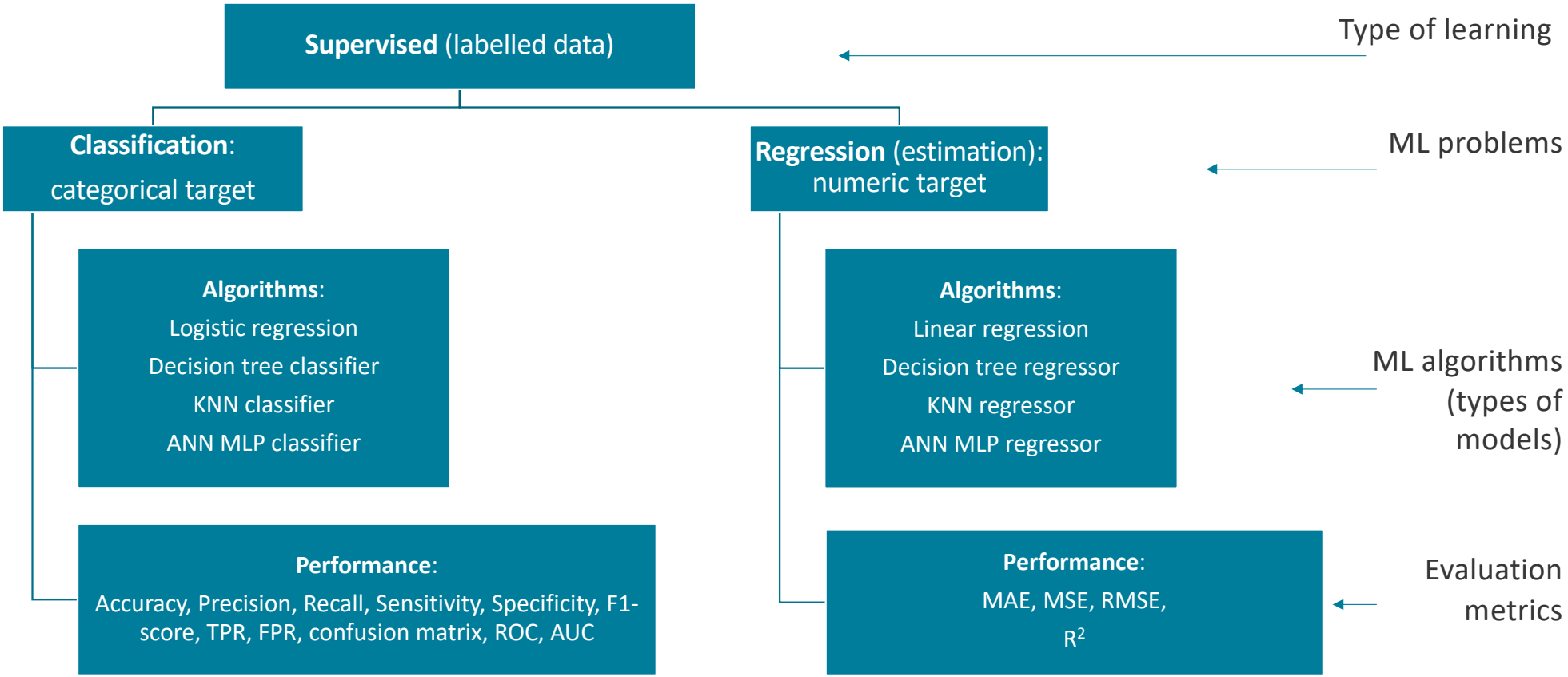
Transparency

- Interpret and explain how the model works and its performance metrics and trade off
- Ensure comprehensive documentation including data sources, preprocessing steps, model choices, and validation processes.

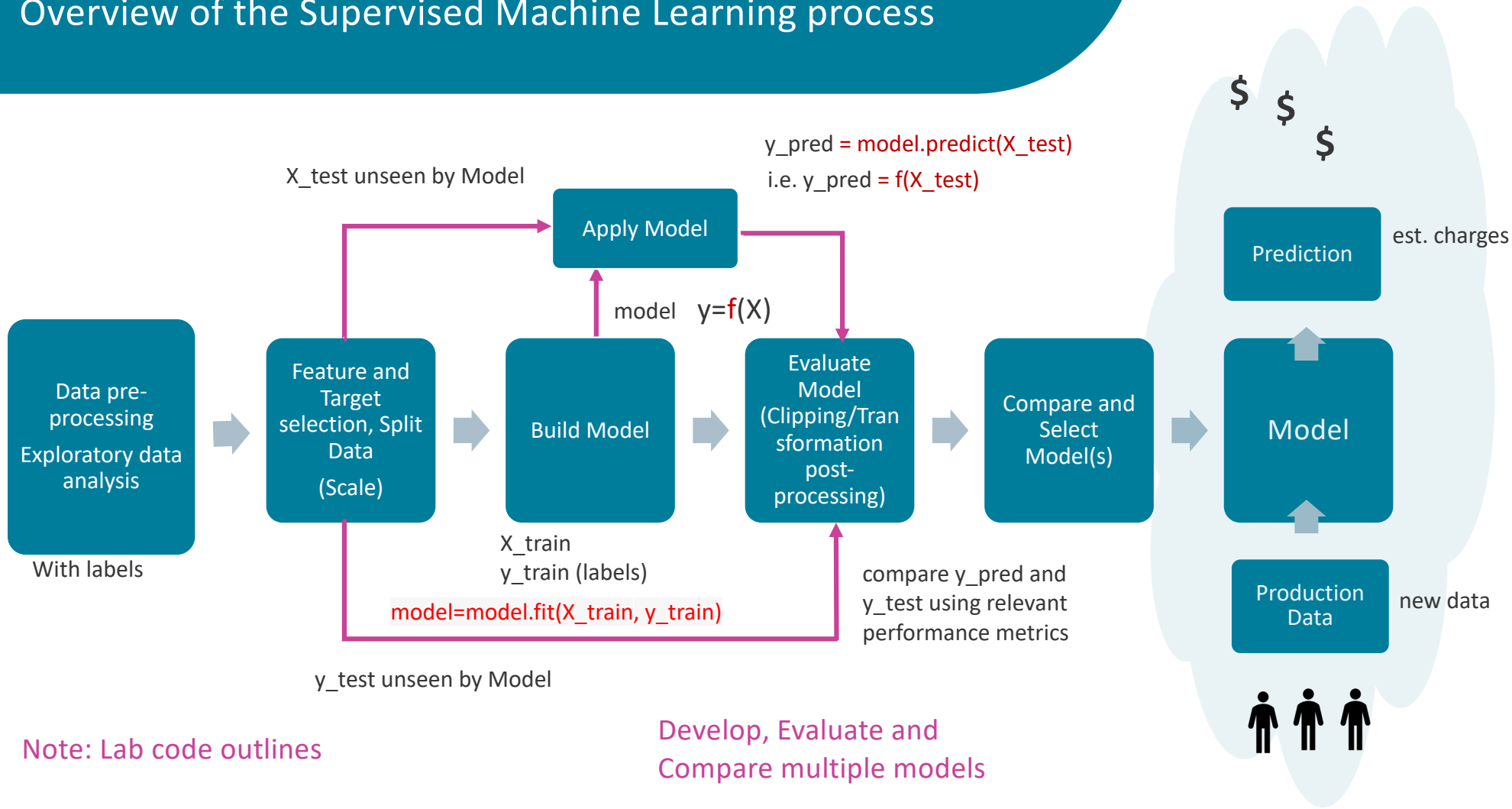
Privacy

- Data minimisation
- Access control

Recap


Supervised (labelled data)
Type of learning
Classification:
categorical target
Regression (estimation):
numeric target
ML problems
Algorithms:
Logistic regression
Decision tree classifier
KNN classifier
ANN MLP classifier
Algorithms:
Linear regression
Decision tree regressor
KNN regressor
ANN MLP regressor
ML algorithms (types of models)
Performance:
Accuracy, Precision, Recall, Sensitivity, Specificity, F1-score, TPR, FPR, confusion matrix, ROC, AUC
Performance:
MAE, MSE, RMSE,
R^2
Evaluation metrics

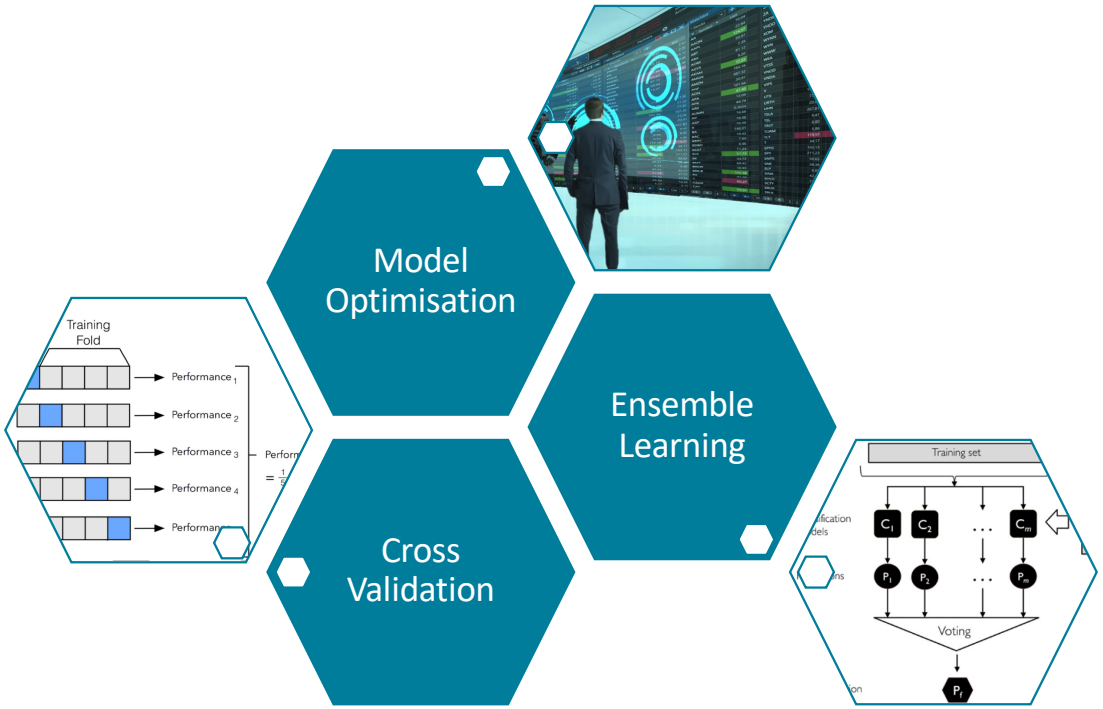
Overview of the Supervised Machine Learning process



Note: Lab code outlines

Develop, Evaluate and Compare multiple models

Predictive Machine Learning with Decision Trees



Model Optimisation

Model fitting

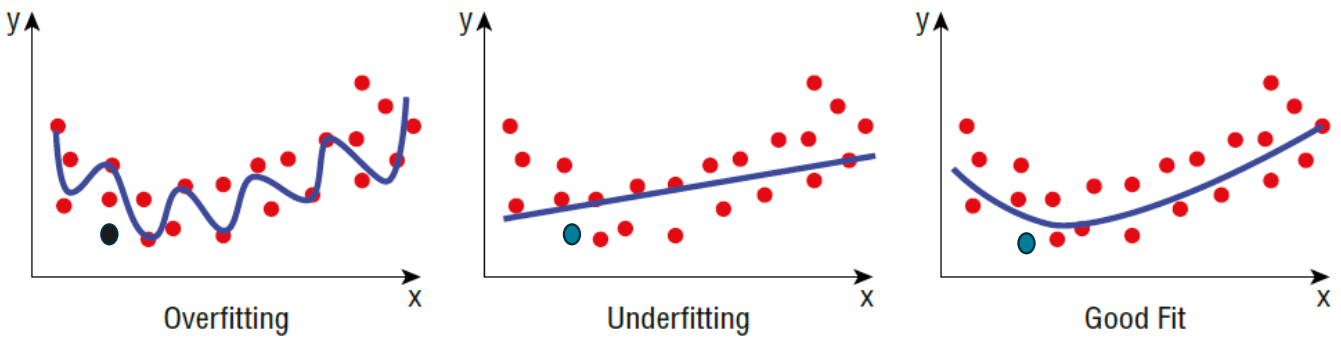
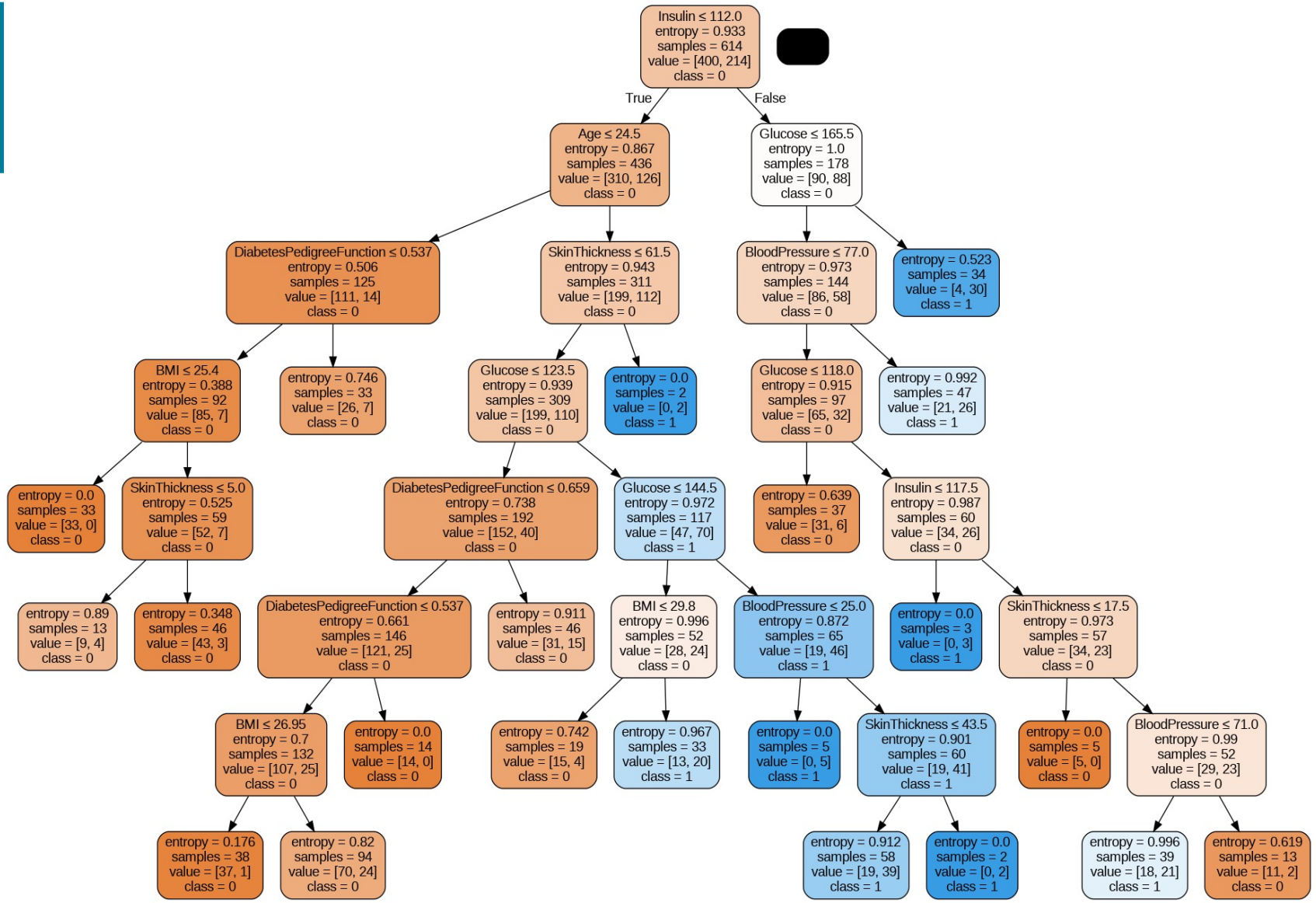


Figure 9.6: Understanding the concept of overfitting, underfitting, and a good fit

Lee, 2019

Two types of parameters

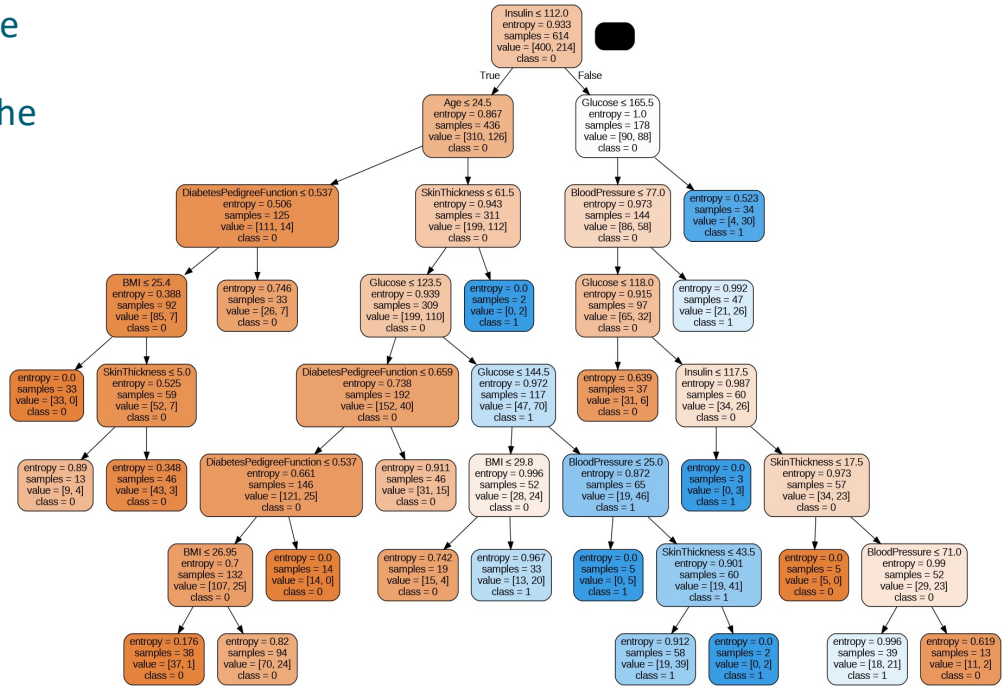
- Model parameters are internal parameters of the model that are **learned from the training process**. They define the **model's structure and how it makes predictions**, e.g., splitting rules in the nodes and leave note values in decision trees
  - The model itself
  - Learned automatically from the data during training.
  - Change with different training datasets.


Insulin ≤ 112.0
entropy = 0.933
samples = 614
value = [400, 214]
class = 0
True
False
Age ≤ 24.5
entropy = 0.867
samples = 436
value = [310, 126]
class = 0
Glucose ≤ 165.5
entropy = 1.0
samples = 178
value = [90, 88]
class = 0
DiabetesPedigreeFunction ≤ 0.537
entropy = 0.506
samples = 125
value = [111, 14]
class = 0
SkinThickness ≤ 61.5
entropy = 0.943
samples = 311
value = [199, 112]
class = 0
BloodPressure ≤ 77.0
entropy = 0.973
samples = 144
value = [86, 58]
class = 0
entropy = 0.523
samples = 34
value = [4, 30]
class = 1
BMI ≤ 25.4
entropy = 0.388
samples = 92
value = [85, 7]
class = 0
entropy = 0.746
samples = 33
value = [26, 7]
class = 0
Glucose ≤ 123.5
entropy = 0.939
samples = 309
value = [199, 110]
class = 0
entropy = 0.0
samples = 2
value = [0, 2]
class = 1
Glucose ≤ 118.0
entropy = 0.915
samples = 97
value = [65, 32]
class = 0
entropy = 0.992
samples = 47
value = [21, 26]
class = 1
entropy = 0.0
samples = 33
value = [33, 0]
class = 0
SkinThickness ≤ 5.0
entropy = 0.525
samples = 59
value = [52, 7]
class = 0
DiabetesPedigreeFunction ≤ 0.659
entropy = 0.738
samples = 192
value = [152, 40]
class = 0
Glucose ≤ 144.5
entropy = 0.972
samples = 117
value = [47, 70]
class = 1
entropy = 0.639
samples = 37
value = [31, 6]
class = 0
Insulin ≤ 117.5
entropy = 0.987
samples = 60
value = [34, 26]
class = 0
entropy = 0.89
samples = 13
value = [9, 4]
class = 0
entropy = 0.348
samples = 46
value = [43, 3]
class = 0
DiabetesPedigreeFunction ≤ 0.537
entropy = 0.661
samples = 146
value = [121, 25]
class = 0
entropy = 0.911
samples = 46
value = [31, 15]
class = 0
BMI ≤ 29.8
entropy = 0.996
samples = 52
value = [28, 24]
class = 0
BloodPressure ≤ 25.0
entropy = 0.872
samples = 65
value = [19, 46]
class = 1
entropy = 0.0
samples = 3
value = [0, 3]
class = 1
SkinThickness ≤ 17.5
entropy = 0.973
samples = 57
value = [34, 23]
class = 0
BMI ≤ 26.95
entropy = 0.7
samples = 132
value = [107, 25]
class = 0
entropy = 0.0
samples = 14
value = [14, 0]
class = 0
entropy = 0.742
samples = 19
value = [15, 4]
class = 0
entropy = 0.967
samples = 33
value = [13, 20]
class = 1
entropy = 0.0
samples = 5
value = [0, 5]
class = 1
SkinThickness ≤ 43.5
entropy = 0.901
samples = 60
value = [19, 41]
class = 1
entropy = 0.0
samples = 5
value = [5, 0]
class = 0
BloodPressure ≤ 71.0
entropy = 0.99
samples = 52
value = [29, 23]
class = 0
entropy = 0.176
samples = 38
value = [37, 1]
class = 0
entropy = 0.82
samples = 94
value = [70, 24]
class = 0
entropy = 0.912
samples = 58
value = [19, 39]
class = 1
entropy = 0.0
samples = 2
value = [0, 2]
class = 1
entropy = 0.996
samples = 39
value = [18, 21]
class = 1
entropy = 0.619
samples = 13
value = [11, 2]
class = 0

Two types of parameters

- Model parameters are internal parameters of the model that are **learned from the training process**. They define the **model's structure and how it makes predictions**, e.g., splitting rules in the nodes and leave note values in decision trees
  - The model itself
  - Learned automatically from the data during training.
  - Change with different training datasets.

- Hyperparameters are **configuration settings** used to control the training process, e.g., K in KNN, max depth and min samples split of a decision tree, number of layers in ANN
  - Set by us prior to the training process.
  - Influence the model's structure, complexity, and how it learns from the data.
  - Do not change during training; but can be optimised.



Hyperparameters in scikit learn

Linear Regression

Logistic Regression

Decision Tree Classifier

Decision Tree Regressor

KNN Classifier

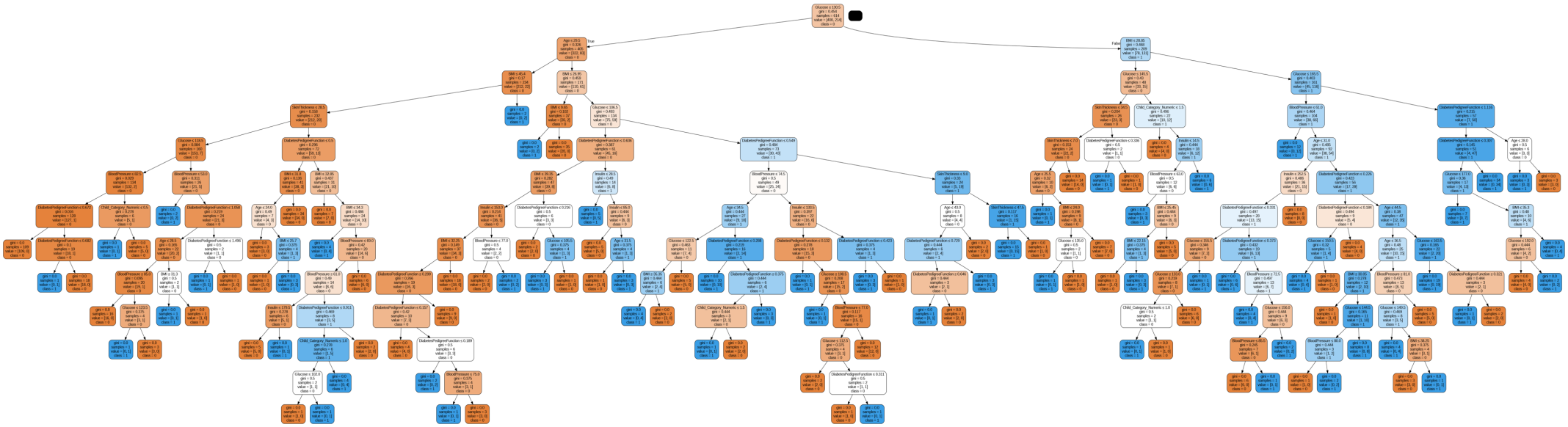
KNN Regressor

Multi Layer Perceptron Classifier

Multi Layer Perceptron Regressor

```
Confusion Matrix - Training set:
[[403   0]
 [  0 211]]
Classification Report - Training set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       403
           1       1.00      1.00      1.00       211

    accuracy                           1.00       614
   macro avg       1.00      1.00      1.00       614
weighted avg       1.00      1.00      1.00       614
```

```
[[82 18]
 [24 30]]
              precision    recall  f1-score   support

           0       0.77      0.82      0.80       100
           1       0.62      0.56      0.59        54

    accuracy                           0.73       154
   macro avg       0.70      0.69      0.69       154
weighted avg       0.72      0.73      0.72       154
```



A base decision tree with default options

Pruning decision trees

Pre-Pruning:

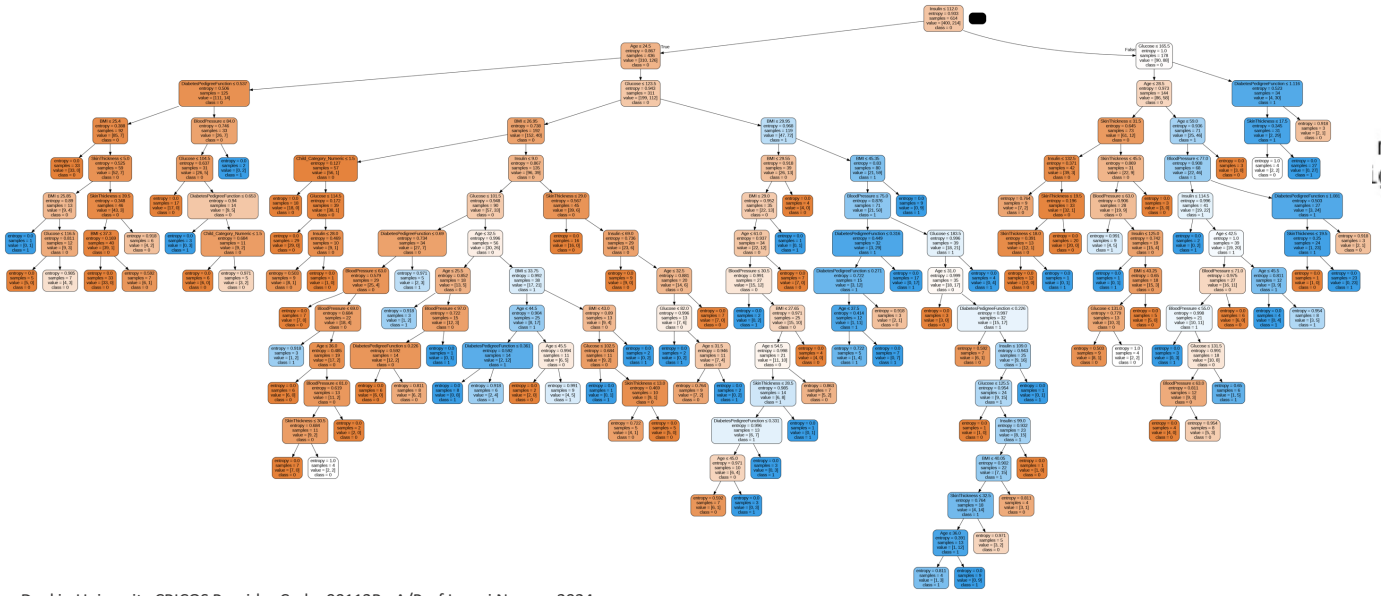
- Avoid overfitting by keeping the tree smaller and simpler by specifying maximum depth limit, minimum number of samples required to split a node.
- Quick and efficient model training
- Or when computational resources are limited.

Post-Pruning:

- Grow the decision tree fully depth then remove branches with little predictive power
- Where model accuracy is important
- Computational resources are sufficient to handle the full tree growth and pruning process.

A pre-pruned decision tree

```
# Create Decision Tree classifer object
pre_pruned_clf = DecisionTreeClassifier(max_depth=15,
        max_features='sqrt', criterion='entropy',
        splitter='best', min_samples_split=10,
        random_state=2024)

# Train Decision Tree Classifer
pre_pruned_clf = pre_pruned_clf.fit(X_train, y_train)
```



Train performance

```
[[380  20]
 [ 36 178]]
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       400
           1       0.90      0.83      0.86       214

    accuracy                           0.91       614
   macro avg       0.91      0.89      0.90       614
weighted avg       0.91      0.91      0.91       614
```

Test performance

```
[[91  9]
 [30 24]]
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       100
           1       0.73      0.44      0.55        54

    accuracy                           0.75       154
   macro avg       0.74      0.68      0.69       154
ighted avg         0.74      0.75      0.73       154
```

A post-pruned decision tree

- **ccp_alpha** represents tree complexity and accuracy.
- Larger **ccp_alpha** leads to more aggressive pruning, resulting in smaller trees with fewer nodes.
- Smaller **ccp_alpha** allows the tree to retain more nodes, potentially leading to overfitting.

```
# Create a fully grown decision tree classifier
clf_full = DecisionTreeClassifier(max_depth=None, min_samples_split=2, min_samples_leaf=1,random_state=2024)
clf_full.fit(X_train, y_train)

# Now get the cost-complexity pruning path
path = clf_full.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas # Different alpha values

# Train decision trees with different alpha values (post-pruning)
decision_trees = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=2024, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    decision_trees.append(clf)

# Evaluate all decision trees and select the best one based on accuracy score
accuracy_scores = [accuracy_score(y_test, dt.predict(X_test)) for dt in decision_trees]
post_pruned_clf = decision_trees[accuracy_scores.index(max(accuracy_scores))]

# Make predictions on the testing data using the best decision tree
y_pred_post_pruned = post_pruned_clf.predict(X_test)
```

A post-pruned decision tree

Train performance

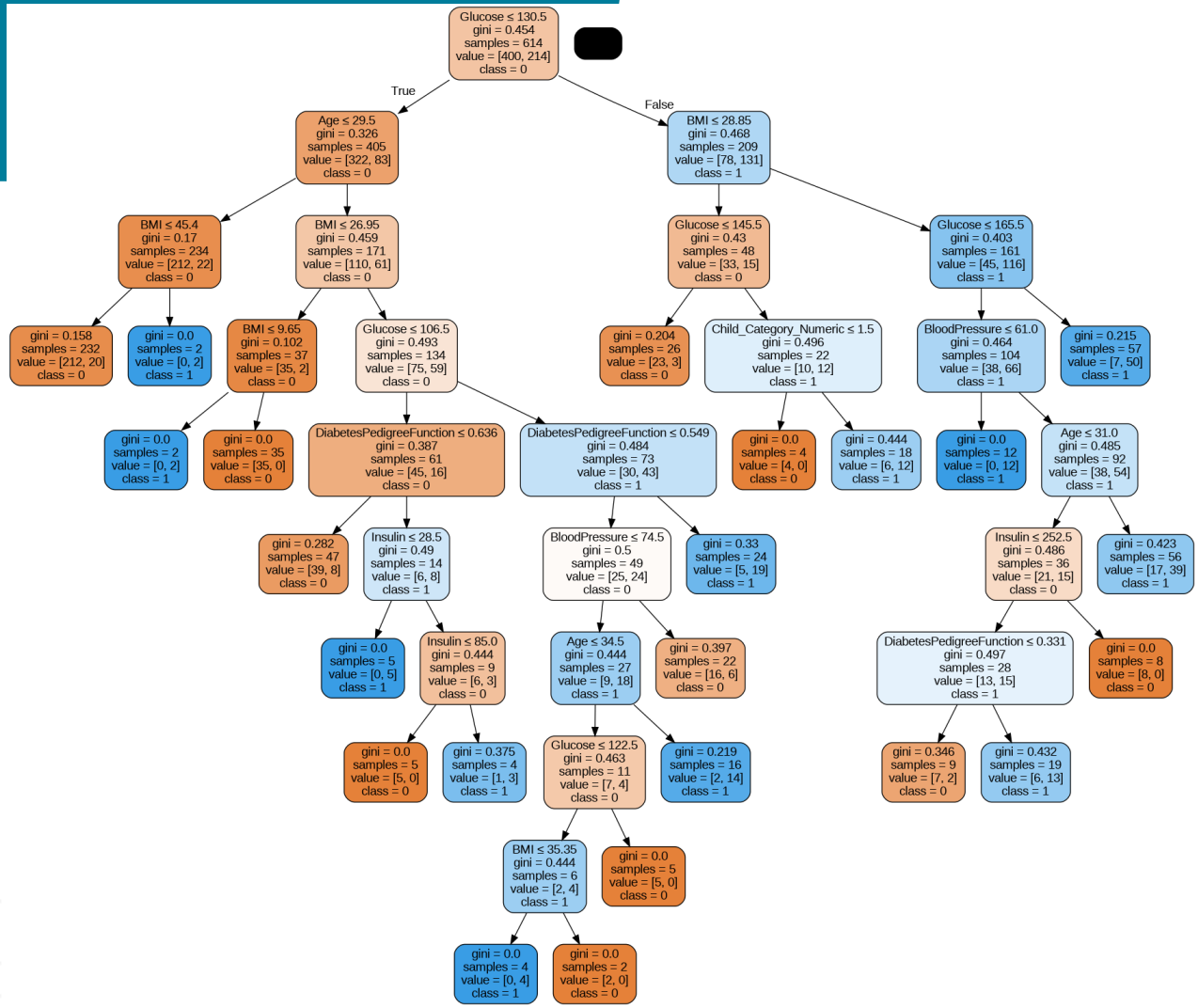
```
[[356  44]
 [ 39 175]]
```

	precision	recall	f1-score	support
0	0.90	0.89	0.90	400
1	0.80	0.82	0.81	214
accuracy			0.86	614
macro avg	0.85	0.85	0.85	614
weighted avg	0.87	0.86	0.87	614

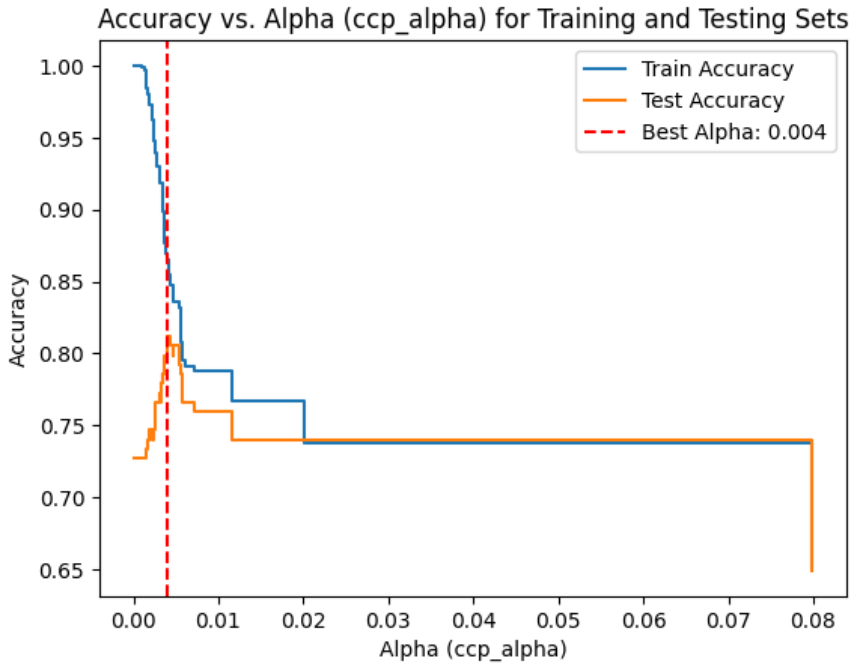
Test performance

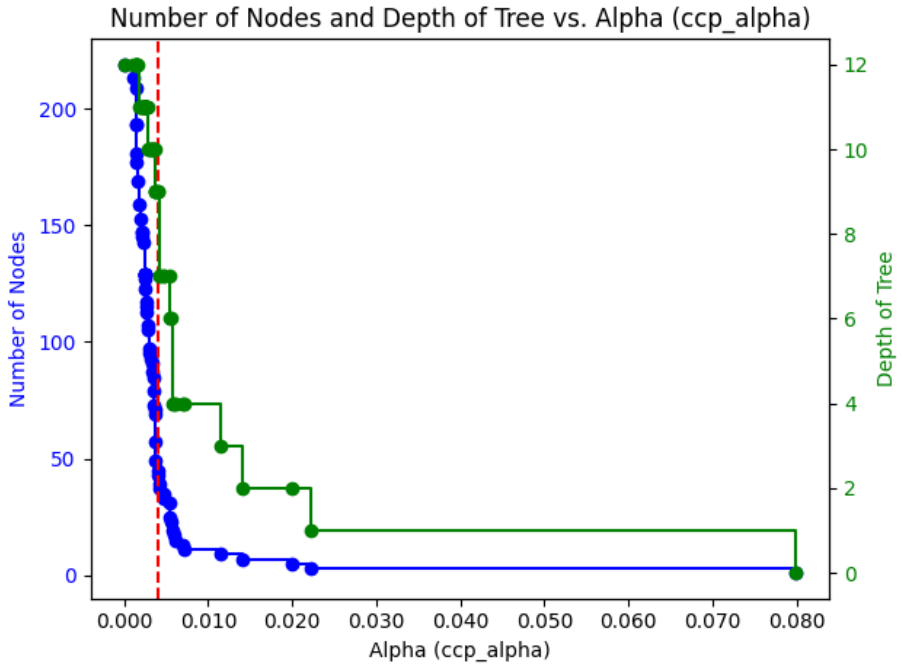
```
[[85 15]
 [14 40]]
```

	precision	recall	f1-score	support
0	0.86	0.85	0.85	100
1	0.73	0.74	0.73	54
accuracy			0.81	154
macro avg	0.79	0.80	0.79	154
weighted avg	0.81	0.81	0.81	154



Post-pruning effects

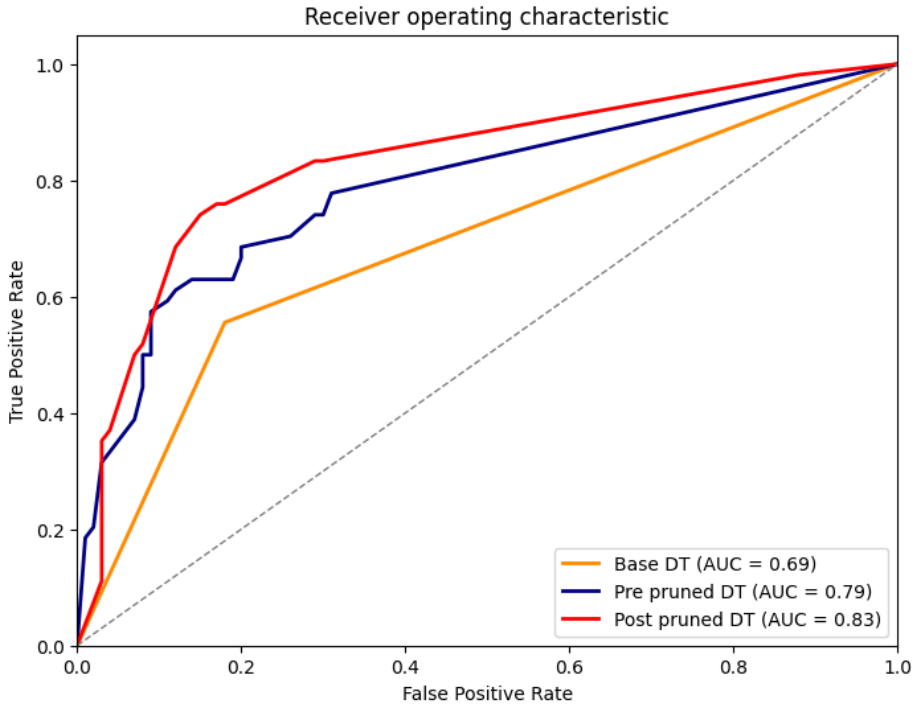





```
Best ccp_alpha: 0.0040
Accuracy_test: 0.81
Number of nodes for best ccp_alpha: 45
Depth of tree for best ccp_alpha: 9
```

Model comparison

Test Performance Metrics	Base DT	Pre-pruned DT	Post-pruned DT
Accuracy	0.73	0.79	0.81
Precision	0.62	0.77	0.73
Recall	0.56	0.56	0.74
F1 score	0.59	0.65	0.73
AUC	0.69	0.79	0.83



Base DT: random_state=2024
Pre-pruned DT: max_depth=10, max_features='sqrt', criterion='entropy', splitter='best', min_samples_split=10, random_state=2024
Post-pruned DT: full-tree: max_depth=None, min_samples_split=2, min_samples_leaf=1,random_state=2024; accuracy-based optimisation: ccp_alpha = 0.0040

Imbalance class problems

Assign penalties: Increase the penalty for incorrect predictions on the minority class using class_weight='balanced' to make the model more sensitive to the minority class.

Resampling techniques:

- Up-sample the minority class
- Down-sample the majority class
- Synthetic data generation: e.g., use SMOTE (Synthetic Minority Over-sampling Technique) to create synthetic examples for the minority class.

$$x_{new} = x_i + \lambda \times (x_{nn} - x_i)$$

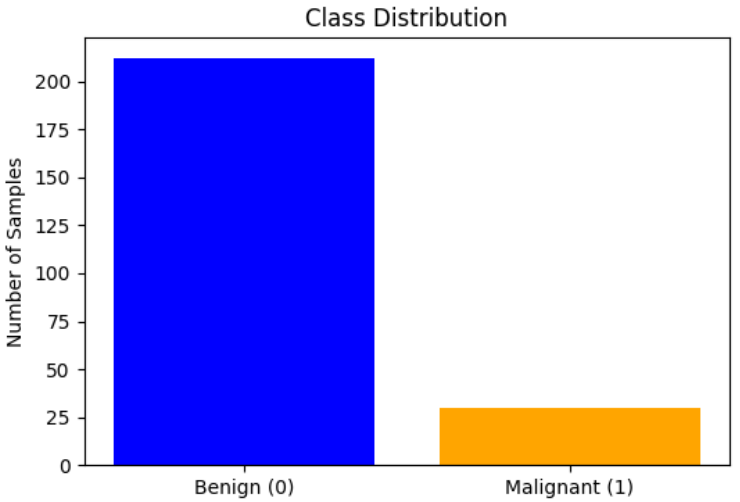
Stratified Splitting: Use stratified sampling when splitting the data, stratify=y

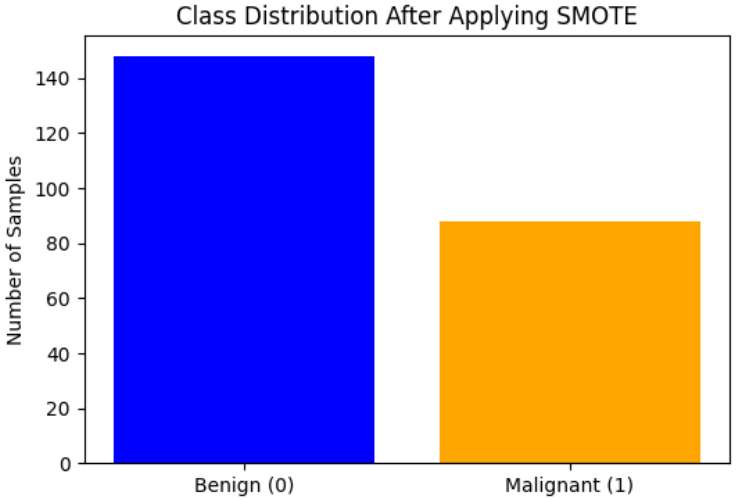
Evaluate different strategies: Experiment with various techniques and evaluate their impact on model performance. Models like Random Forests and Gradient Boosting often perform well with imbalanced data.

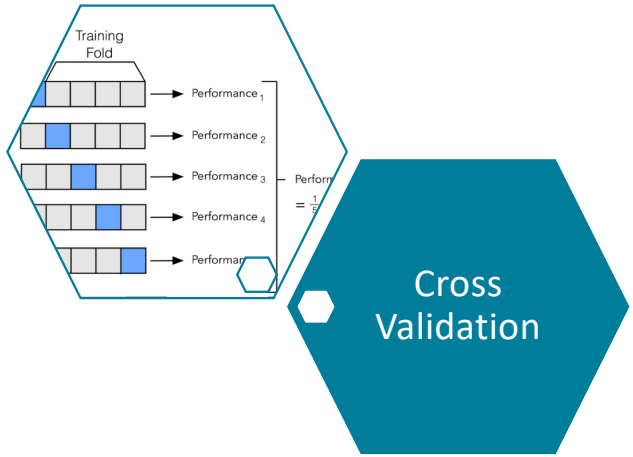
Imbalance class problems

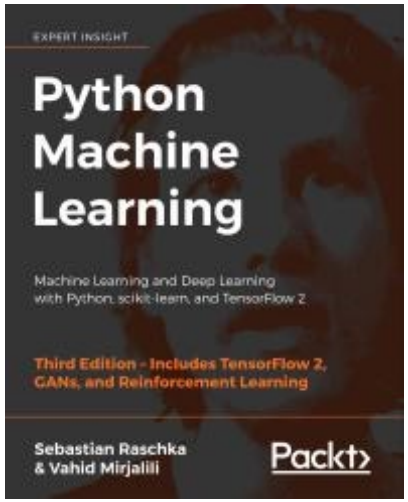
```
# install the imblearn library
pip install imbalanced-learn

from sklearn.utils import resample
from imblearn.over_sampling import SMOTE

# maintain the original class distribution.
X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size=0.3,
        random_state=2024, stratify=y)

# Step 4: Apply SMOTE to generate synthetic samples of
the minority class
smote = SMOTE(sampling_strategy=0.6, random_state=2024)
X_train_resampled, y_train_resampled =
        smote.fit_resample(X_train, y_train)
```

Training on the **resampled data**.
Evaluation on the original imbalanced test -> realistic assessment of its performance.





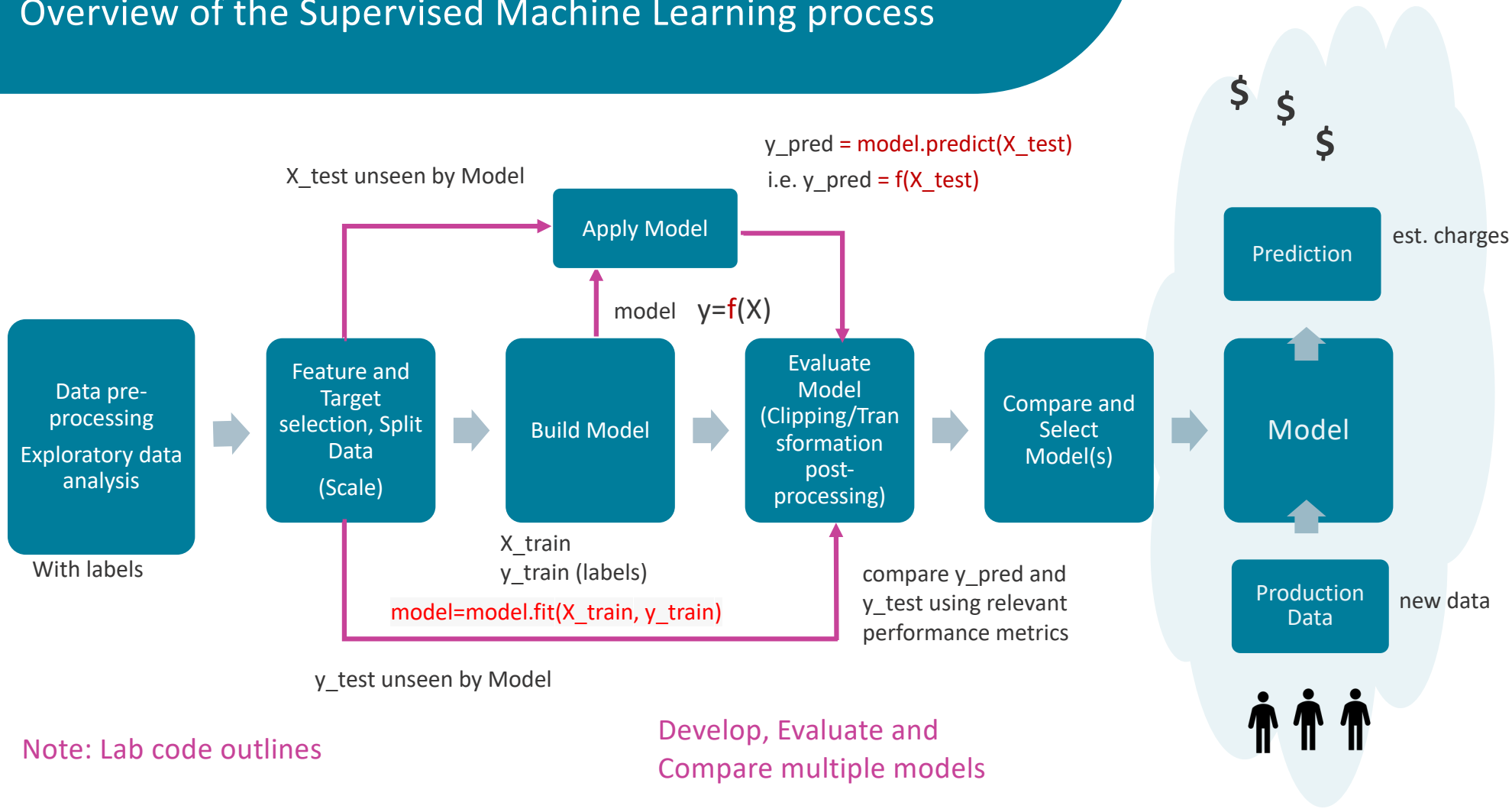




Raschka and Mirjalili, 2019

https://www.analyticsvidhya.com/blog/2021/11/top-7-cross-validation-techniques-with-python-code/

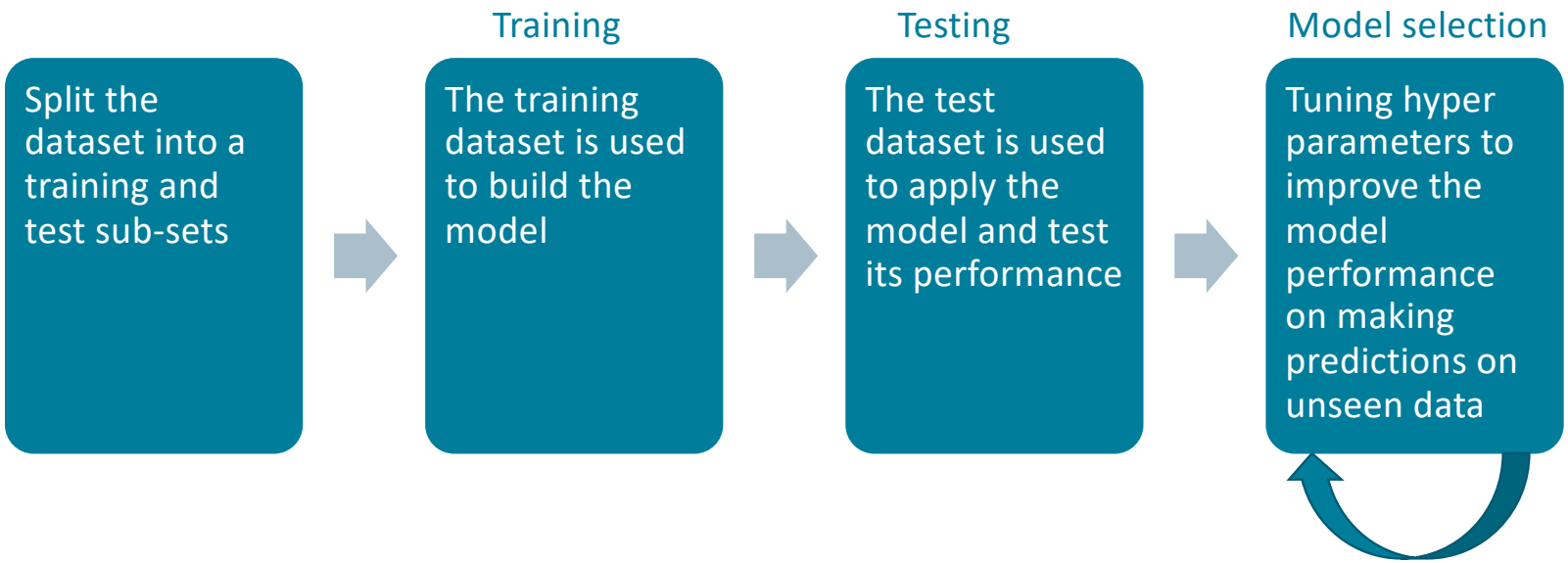
Overview of the Supervised Machine Learning process



Note: Lab code outlines

Develop, Evaluate and Compare multiple models

So far…

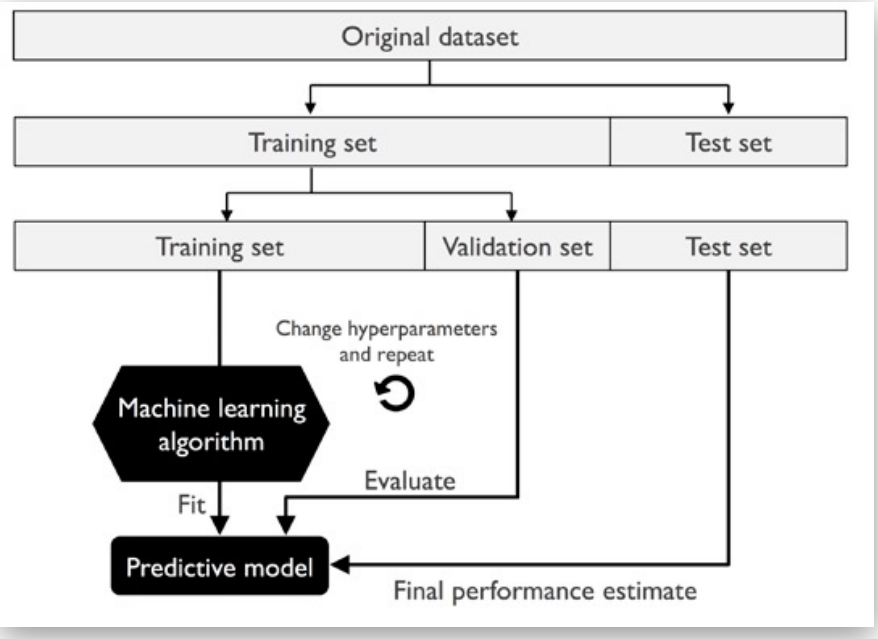


But we repeat the model selection process by training the model again and again on the same X_train and y_train, then testing it with the same X_test and y_test

How do we know the model works as well on totally unseen data?

The hold-out cross validation

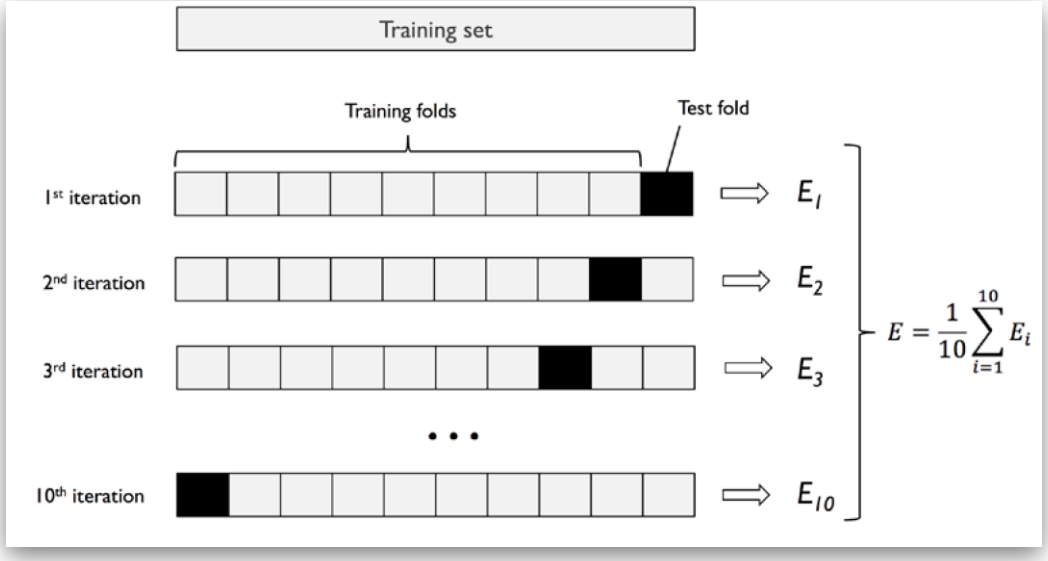
- Advantage: Less biased evaluation of model's ability to generalise to unseen data
- Disadvantage: the final performance evaluation may be sensitive to how data are partitioned into training, validation and test sets.
- A more robust approach is needed.



Raschka and Mirjalili, 2019

K-fold cross validation

- Randomly split the training dataset into k folds without replacement
- In each iteration i, k – 1 folds are used for the model training, and one fold is used for performance evaluation. Hence for each iteration, we have evaluation results E_i, e.g., accuracy /error.
- In the end, there are k models and k performance results.
- Calculate the average error (E) or accuracy. It will be less sensitive to data partitioning.
- Hyperparameter tuning applies to the performance on all k test folds.
- Then evaluate the fine tuned model on an independent test dataset.



Raschka and Mirjalili, 2019

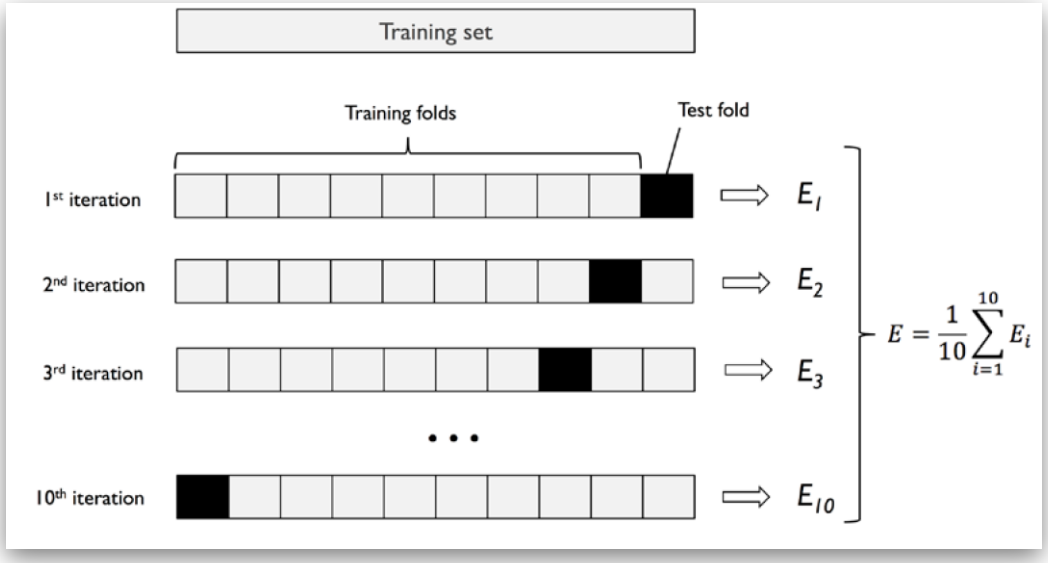
K-fold cross validation

- More accurate and robust:
  - Increased training examples
  - Each datapoint (example) is included in a test fold **once**
- Runtime may be high if k is high
- Gold standard k =10
  - Small datasets: set the value of k high to get – the more data each training iteration, and average is taken from a high number of k test results, thus less bias (less underfitting).
    - However, k is too high then training datasets will be too similar, and high computational time.
  - Large datasets: k can be low, e.g., 5, still get accuracy results and reducing computational time.

Note: imbalanced dataset then some training sets may have only one class

Stratified K-fold cross validation

- Same as K-fold cross validation
- Each fold will have the same ratio of datapoints of target variable as in the (whole) original dataset
- Works for imbalanced class classification



Raschka and Mirjalili, 2019

10-fold cross validation implementation

```
#Import cross validation function
from sklearn.model_selection import cross_val_score

# Create a new Decision Tree classifier object
pre_pruned_clf = DecisionTreeClassifier(max_depth=10, max_features='sqrt',
        criterion='entropy', splitter='best', min_samples_split=10,
        random_state=2024)

# Perform 10-fold cross-validation
accuracy= cross_val_score(pre_pruned_clf, X, y, cv=10)
f1 = cross_val_score(pre_pruned_clf, X, y, cv=10, scoring=('f1'))

# Print the mean accuracy scores and 95%CI of the scores
print("Accuracy scores: %0.3f (+/- %0.3f)" % (accuracy.mean(), get_95ci(accuracy)))

# Print the mean F1 scores and 95%CI of the scores
print("F1 scores: %0.3f (+/- %0.3f)" % (f1.mean(), get_95ci(f1)))
```

```
Accuracy scores: 0.727 (+/- 0.027)
F1 scores: 0.579 (+/- 0.053)
```

LeavePOut and Leave one out cross validation

- **LeavePOut cross-validation**
  - Train on n-1 datapoints, use p left for testing
  - Repeat the process n times
- **Leave One Out cross-validation: p=1 (LOOCV)**

https://www.analyticsvidhya.com/blog/2021/11/top-7-cross-validation-techniques-with-python-code/

- Pros: all data are included for training and testing
- Cons: high computational time for a large dataset

Leave one out validation implementation

```
#import leave one out
from sklearn.model_selection import LeaveOneOut

# Create a new Decision Tree classifer object
pre_pruned_clf = DecisionTreeClassifier(max_depth=10, max_features='sqrt',
        criterion='entropy', splitter='best', min_samples_split=10, random_state=2024)

loocv = LeaveOneOut()
accuracy = cross_val_score(pre_pruned_clf, X, y, cv=loocv, scoring='accuracy')

# Print the mean accuracy and 95%CI of the scores
print("Accuracy: %0.3f (+/- %0.3f)" % (accuracy.mean(), get_95ci(accuracy)))
```

```
Accuracy: 0.715 (+/- 0.037)
```

Leave p out validation implementation

```
#import LeavePOut
from sklearn.model_selection import LeavePOut

# Create a new Decision Tree classifer object
pre_pruned_clf = DecisionTreeClassifier(max_depth=10, max_features='sqrt',
        criterion='entropy', splitter='best', min_samples_split=10, random_state=2024)

lpo = LeavePOut(p=2)
accuracy = cross_val_score(pre_pruned_clf, X_test, y_test, cv=lpo)
```

```
Accuracy: 0.706 (+/- 0.007)
```

We use X_test and y_test for the demonstration purpose because this CV is computationally expensive!!

Reflections on Cross Validation

Method	Bias	Variance	Computational Cost	Best Use Cases
10-fold CV	Moderate	Moderate	Moderate (10 times)	General-purpose use; balanced datasets; reasonable computational resources.
LOO-CV	Low (max use of data for training)	High (sensitive to outliers)	High (n times)	Very small datasets where maximising training data is important.
LpO-CV	Depends on p	Depends on p	Extremely High (n!/(p! (n-p)!)	Thorough evaluation with small datasets; when computational resources are sufficient.

Hyperparameter optimisation

for any algorithms

```
#import GridSearchCV
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for hyperparameter optimisation
param_grid = {
        'max_depth': [None, 5, 10, 15, 20],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4, 6, 8],
        'max_leaf_nodes': [None, 5, 10, 15, 20, 25]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(
        DecisionTreeClassifier(random_state=2024), # Base model
        param_grid, # Parameter grid
        cv=5,
        scoring='accuracy', # evaluation metric
        n_jobs=-1 # Use all available CPU cores to speed up the search
)
```

Hyperparameter optimisation

for any algorithms

```
Classification Report for Test Data:
              precision    recall  f1-score   support

           0       0.80      0.86      0.83       100
           1       0.70      0.59      0.64        54

    accuracy                           0.77       154
   macro avg       0.75      0.73      0.73       154
weighted avg       0.76      0.77      0.76       154
```

```
# Fit grid search to the training data
grid_search.fit(X_train, y_train)

# Select the best decision tree classifier based on grid search results
best_clf = grid_search.best_estimator_

# Make predictions on testing data using the best decision tree classifier
y_pred = best_clf.predict(X_test)

# Print the best hyperparameters found by grid search
print(f"Best Hyperparameters: {grid_search.best_params_}")
```

```
Best Hyperparameters: {'max_depth': 5, 'max_leaf_nodes': 20, 'min_samples_leaf': 6,
'min_samples_split': 2}

Test Accuracy of the best decision tree: 0.77
```

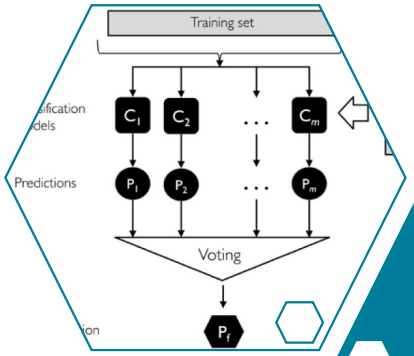
Reflections on Cross Validation

Pross

- More robust and reliable (less biased) validation compared to a single train-test split.
- Provides insights into how the model will perform on an independent dataset.
- Efficient use of data where the dataset is small.
- Model comparison: cross-validation allows you to compare the performance of different predictive models in a more robust manner.

Cons

- Computationally Intensive especially with LOO LPO
- Time-consuming with large datasets
- High variance especially in LOO and LPO, the model's evaluation can vary significantly depending on which subsets of the data are used for training and testing.
- Increased complexity, implementing cross-validation can be more complex and error-prone than a simple train/test split.



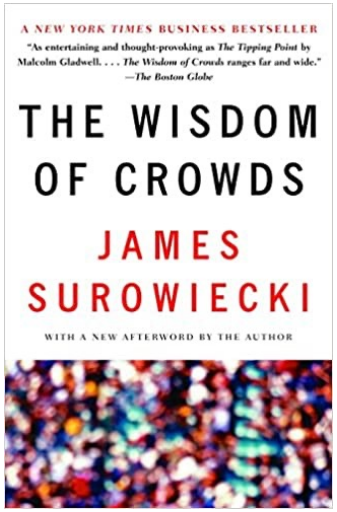
Ensemble Learning

Ensemble learning - the wisdom of crowds

Goals: improve accuracy and robustness, and avoid over fitting and underfitting

Motivation: Applying the wisdom of crowds to Machine Learning

How: Develop multiple predictive models and use the majority principle to make predictions



Ensemble learning - the wisdom of crowds

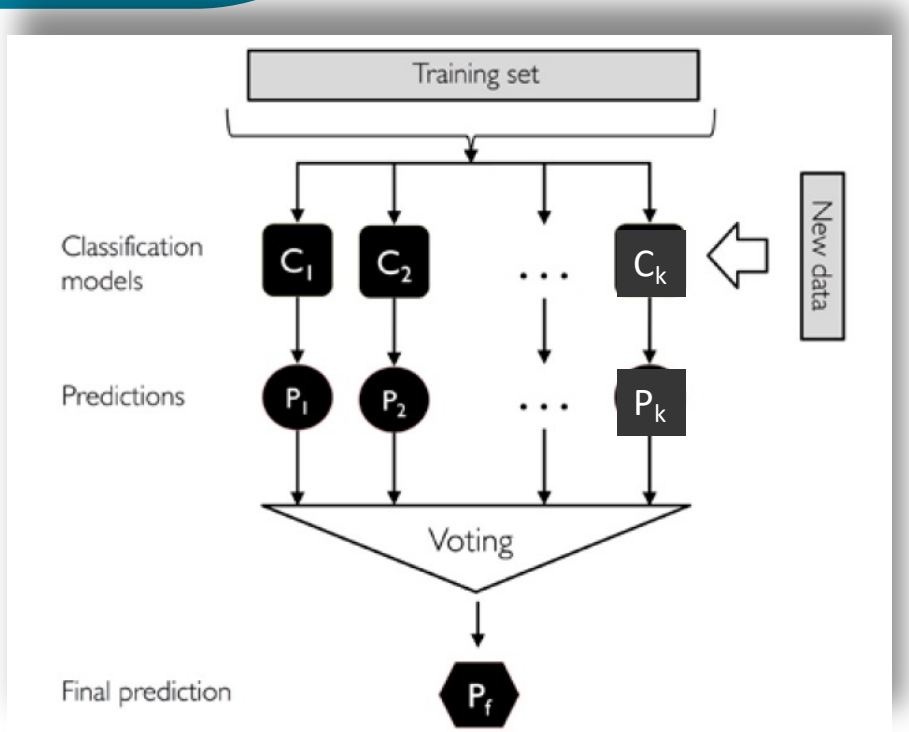
Classification problems:

- Build multiple base classifiers $C_{1..k}$
- Each giving a prediction $P_{1..k}$
- Simple majority voting (binary classification) or plurality voting (multiclass classification)

$\hat{y}$ = mode{$C_1(x)$, $C_2(x)$,…,$C_k(x)$}

i.e. selecting the class with the most votes

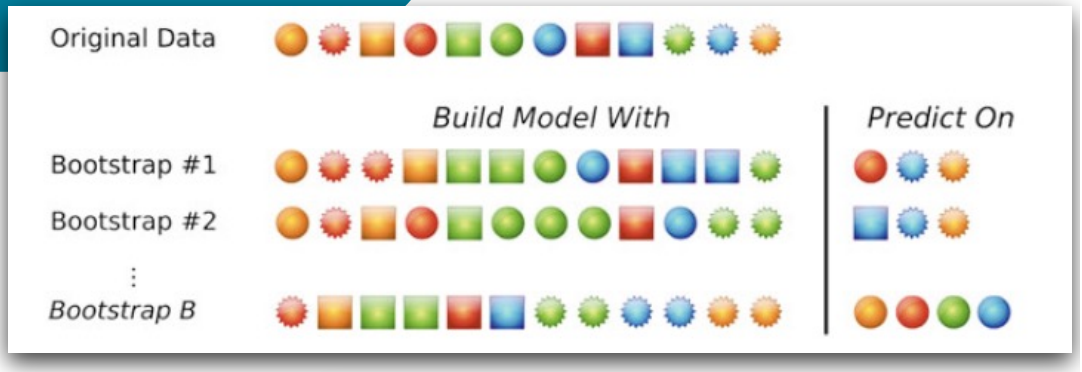
Estimation problems: same principle, take average for estimation



Adapted from Raschka and Mirjalili (2019)

Bootstrap aggregation (Bagging)

Source of Figure: Kuhn and Johnson, 2013, p..73



Given a dataset with M records and N attributes

Select the number of models to build, k

Choose a base model (e.g., Decision Tree, Neural Network)

For i = 1 to k do

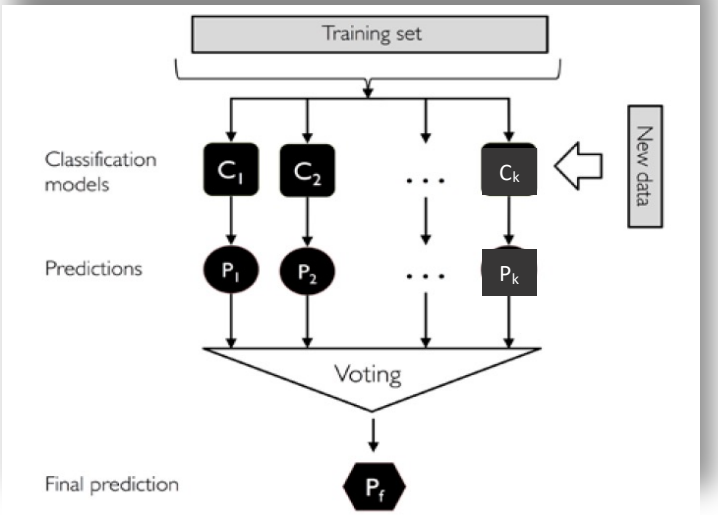
- Create a bootstrap sample by randomly sampling M records with replacement
- Train the base model on this bootstrap sample (store the model)
- For each OOB sample:
    - Predict the target using the model trained on the bootstrap sample
    - Store the OOB prediction and the actual target value

For each record in the dataset:

- Combine OOB predictions from all models where the record was an OOB sample and Calculate an OOB error metrics

Aggregate the predictions:

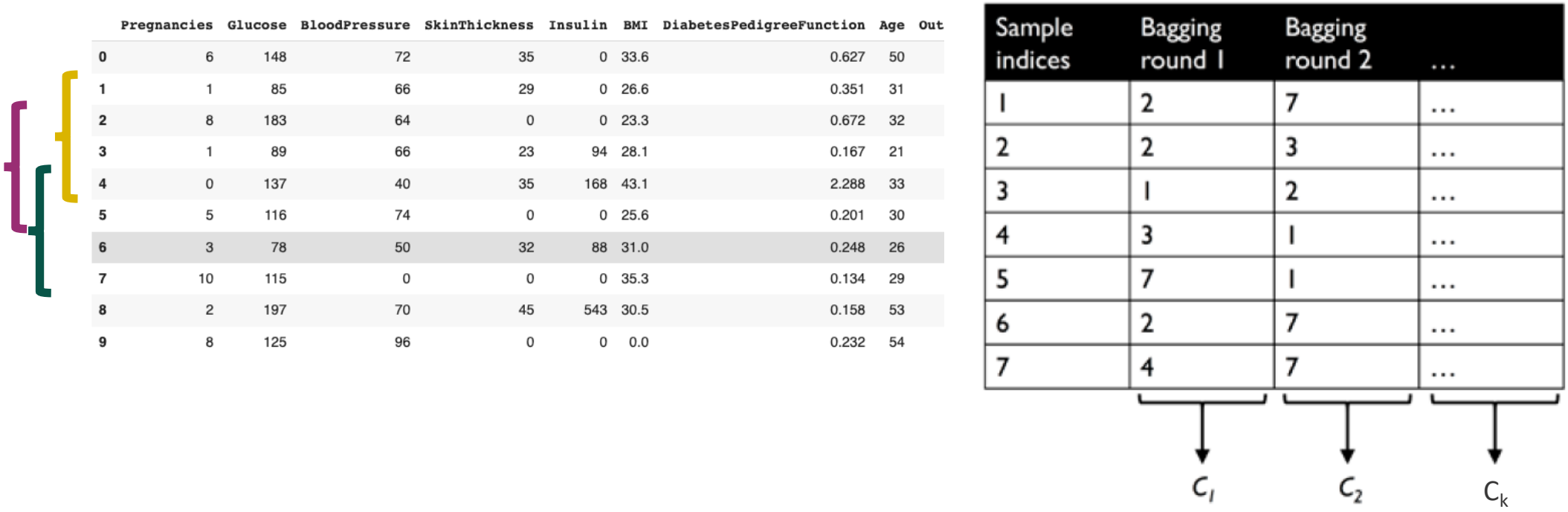
- For classification: Use majority voting
- For regression: Use averaging



Adapted from Raschka and Mirjalili (2019)

Bagging

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Out
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	
5	5	116	74	0	0	25.6	0.201	30	
6	3	78	50	32	88	31.0	0.248	26	
7	10	115	0	0	0	35.3	0.134	29	
8	2	197	70	45	543	30.5	0.158	53	
9	8	125	96	0	0	0.0	0.232	54	



Adapted from Raschka and Mirjalili (2019)

Implementation example

```
#Import BaggingClassifier
from sklearn.ensemble import BaggingClassifier


# Define the ensemble classifier using bagging
ensemble_classifier = BaggingClassifier(estimator=post_pruned_clf, n_estimators=20, random_state=2024)

# Train the ensemble classifier on the training data
ensemble_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_baggedtrees = ensemble_classifier.predict(X_test)
```

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred_baggedtrees))
```

```
[[88 12]
 [18 36]]
              precision    recall  f1-score   support

           0       0.83      0.88      0.85       100
           1       0.75      0.67      0.71        54

    accuracy                           0.81       154
   macro avg       0.79      0.77      0.78       154
weighted avg       0.80      0.81      0.80       154
```

Random forests

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

Given a dataset with M records and N attributes

Select the number of Decision Trees to build, k

Select the number of features to consider at each split, d (d < N)

For i = 1 to k:

- Create a bootstrap sample by randomly sampling M records with replacement
- Train a Decision Tree on this bootstrap sample:
  - Randomly select d features from the N features
  - Find the best split among the d selected features
  - Repeat until the stopping criteria is met (e.g., max depth, min samples per leaf)
- Store the trained Decision Tree model

For each sample, make predictions from all k Decision Trees

Final prediction:

- For classification: Use majority voting
- For regression: Use averaging

Random forests

Given a dataset with M records and N attributes

Select the number of Decision Trees to build, k

Select the number of features to consider at each split, d (d < N)

For i = 1 to k:

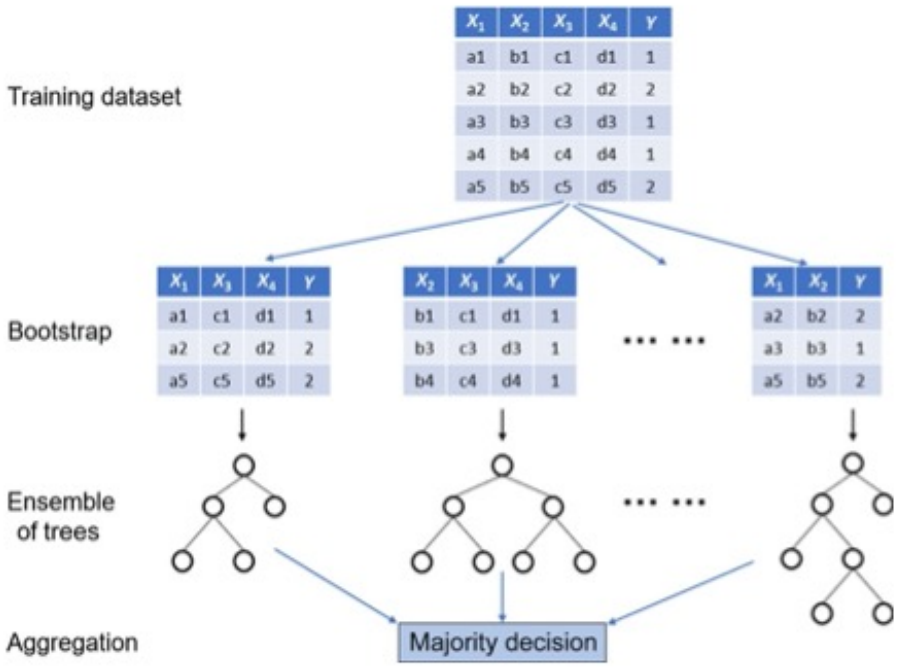
- Create a bootstrap sample by randomly sampling M records w
- Train a Decision Tree on this bootstrap sample:
  - Randomly select d features from the N features
  - Find the best split among the d selected features
  - Repeat until the stopping criteria is met (e.g., max depth, m
- Store the trained Decision Tree model

For each sample, make predictions from all k Decision Trees

Final prediction:

- For classification: Use majority voting
- For regression: Use averaging

https://www.sciencedirect.com/science/article/pii/B9780128177365000090



Random forest: example

```
#Import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# Structure a random forest model with 20 decision trees
rf_clf = RandomForestClassifier(n_estimators=20, random_state=2024)

# Train the model on the training set
rf_clf.fit(X_train, y_train)

# Evaluate the model on the test set
y_pred = rf_clf.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred_baggedtrees))

# Evaluate the model using cross-validation
cv_scores = cross_val_score(rf_clf, X, y, cv=10)
print("Cross-validation scores:", cv_scores)
print(f"Mean CV accuracy: {cv_scores.mean(): .3f}")
print(f"Std. dev. of CV accuracy: {cv_scores.std(): .3f}")
```

```
[[90 10]
 [20 34]]
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       100
           1       0.77      0.63      0.69        54

    accuracy                           0.81       154
   macro avg       0.80      0.76      0.78       154
weighted avg       0.80      0.81      0.80       154
```

```
Cross-validation scores: [0.7012987 0.71428571
0.76623377 0.7012987 0.72727273 0.83116883
0.79220779 0.80519481 0.63157895 0.77631579]

Mean CV accuracy: 0.745
Std. dev. of CV accuracy: 0.057
```

Boosting - Boosted trees

focus on difficult data points

Given a dataset of M records and N attributes

Assign the same weight to all the data points

Select the number of models to build - k

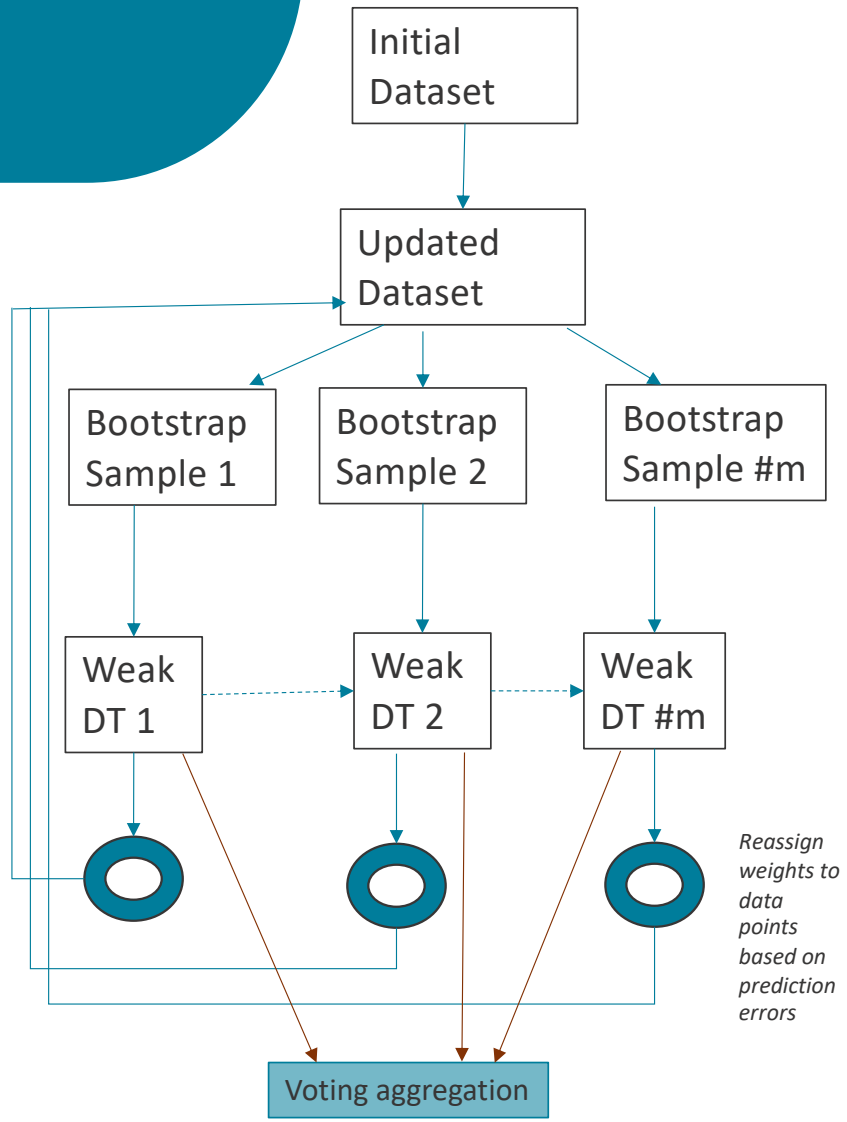
For i = 1 to k do

- Take a random sample with replacement (bootstrap sample)
- Sample are of the same size as the original dataset (M)
- The datapoints not selected are called “out-of-bag”
- Train a DT model on the bootstrap sample
- Use the DT model to predict the out-of-bag
- **Evaluate the DT model and increase weights to incorrectly classified data points**

Final prediction:

- voting for classification
- averaging for estimation

End



AdaBoost - Boosted trees

focus on difficult data points and weak learners (DTs)

Given a dataset of M records and N attributes

Assign the same weight to all the data points

Select the number of models to build - k

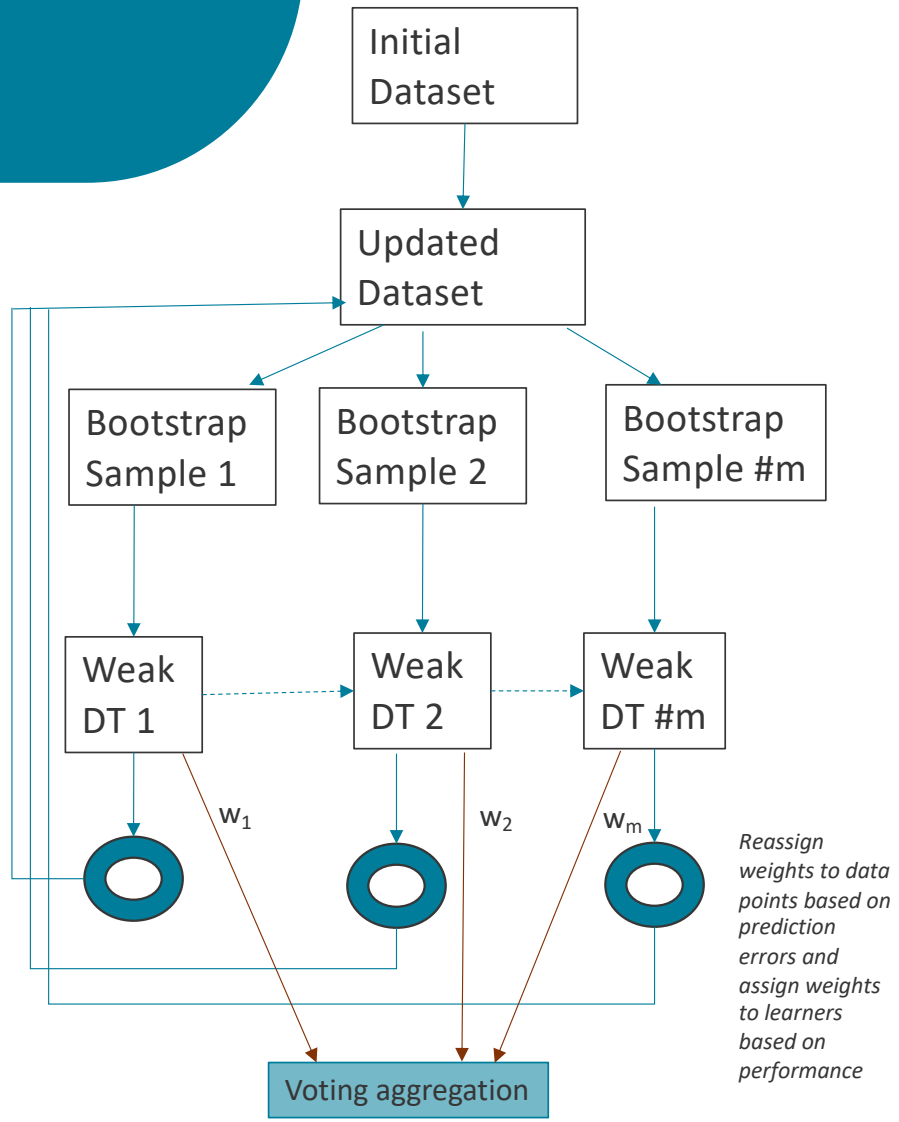
For i = 1 to k do

- Take a random sample with replacement (bootstrap sample)
- Sample are of the same size as the original dataset (M)
- The datapoints not selected are called “out-of-bag”
- Train a DT model on the bootstrap sample;
- Use the DT model to predict the out-of-bag
- **Evaluate the DT model, increase weights to incorrectly classified data points, and assign weights to the DT model**

Final prediction:

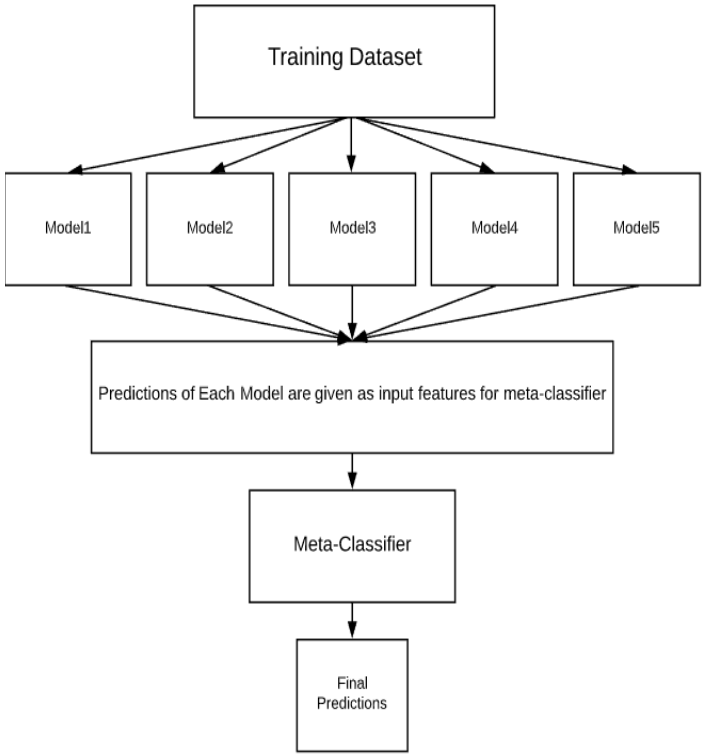
- voting for classification
- averaging for estimation

End



Stacking

- **Base Models:**
  - Multiple diverse models (e.g., Decision Trees, SVMs, Neural Networks) are trained independently on the training set.
- **Meta-Model:**
  - Trained using the predictions from the base models as input features and the true training labels.
  - Meta-model uses predictions by base models on the test set to produce the final predictions.
- **Advantages:**
  - Combines strengths of different models.
  - Effective for classification and regression.
- **Disadvantages:**
  - Increased complexity and computational cost.
  - Risk of overfitting; requires careful tuning.

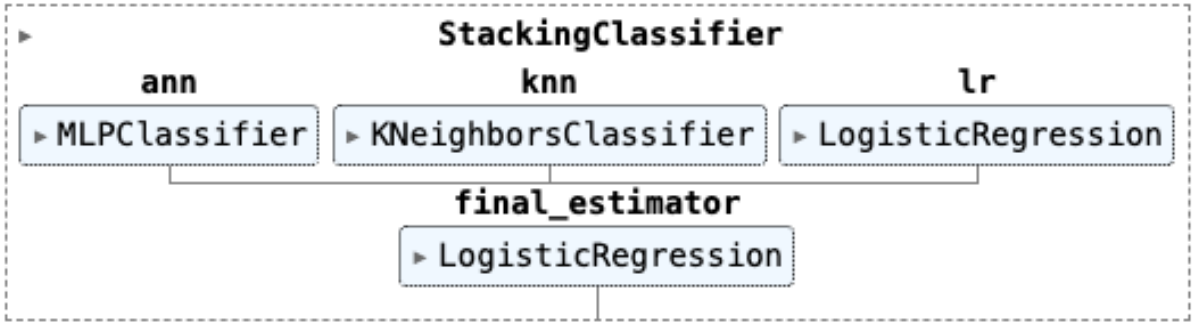


Stacking ensemble model: example

```
#Import classes and functions
from sklearn.model_selection import train_test_split

from sklearn.ensemble import StackingClassifier

from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, classification_report
```

Remember to scale data

Stacking ensemble model: example




```
# Define base models
estimators = [
    ('ann', MLPClassifier(hidden_layer_sizes=(32,16), max_iter=1000)),
    ('knn', KNeighborsClassifier(n_neighbors=5)),
    ('lr', LogisticRegression(max_iter=1000))
]

# Define the meta-classifier for the stacking ensemble
meta_classifier = LogisticRegression()


# Build the stacking ensemble model
stacking_model = StackingClassifier(estimators=estimators, final_estimator=meta_classifier)


# Train the stacking ensemble model on the scaled training set
stacking_model.fit(X_train_scaled, y_train)
```

Stacking ensemble model: example

```
# Evaluate the stacking ensemble model on the testing set
y_pred = stacking_model.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[95  5]
 [22 32]]
              precision    recall  f1-score   support

           0       0.81      0.95      0.88       100
           1       0.86      0.59      0.70        54

    accuracy                           0.82       154
   macro avg       0.84      0.77      0.79       154
weighted avg       0.83      0.82      0.82       154
```

```
# Evaluate the stacking model using 10-fold cross validation
scores = cross_val_score(stacking_model, X, y, cv=10,
scoring='accuracy')
print("Accuracy: {:.3f} (+/- {:.3f})".format(np.mean(scores),
np.std(scores) * 2))
```

```
Accuracy: 0.764 (+/- 0.067)
```

Reflections on Ensemble Learning

Pros

- Higher Predictive Power, statistically better performance than single models, offering a more accurate and robust prediction.
- Robustness to Overfitting, less likely to overfit
- Diverse Algorithms, combining different types of models, thus accommodating a diverse range of data types and structures.

Cons

- Computationally expensive and time-consuming as ensembles require the training, storage, and prediction of multiple models -> real-time prediction.
- While single models may be easier to interpret, the complexity of ensemble methods makes them harder to understand and explain.
- If the individual models are overfit, the ensemble model is likely to overfit as well.

Model Optimisation

Strategies to Improve Machine Learning Models (1)

1. Data pre-processing, for example

- **Handle missing values:** using mean, median, mode, or KNN imputation.
- **Outliers**: Use techniques like z-scores, IQR (Interquartile Range), to detect and manage outliers.
- **Normalize data:** Standardization and Min-Max Scaling
- **Handle imbalanced data:** Use SMOTE or under-sampling to balance datasets for classification tasks.

2. Feature selection (relevant variables), for example

- **Remove irrelevant features**: e.g. recursive feature elimination, or relationship analysis to select the most relevant features.
- **Dimensionality reduction**: e.g. Principal Component Analysis to reduce the number of features

3. Feature Engineering, for example

- **Create new features**: Use domain knowledge to derive new, predictive features.
- **Combine features**: Summarise data or use PCA for enhanced data representation.

4. Collect more data

Strategies to Improve Machine Learning Models (2)

1. Try Different Algorithms:

- Each algorithm has its own assumptions about data distributions / relationships
- Experiment with various algorithms (e.g., Decision Trees, Random Forests, KNN, Neural Networks) to identify the best-performing model for your data.

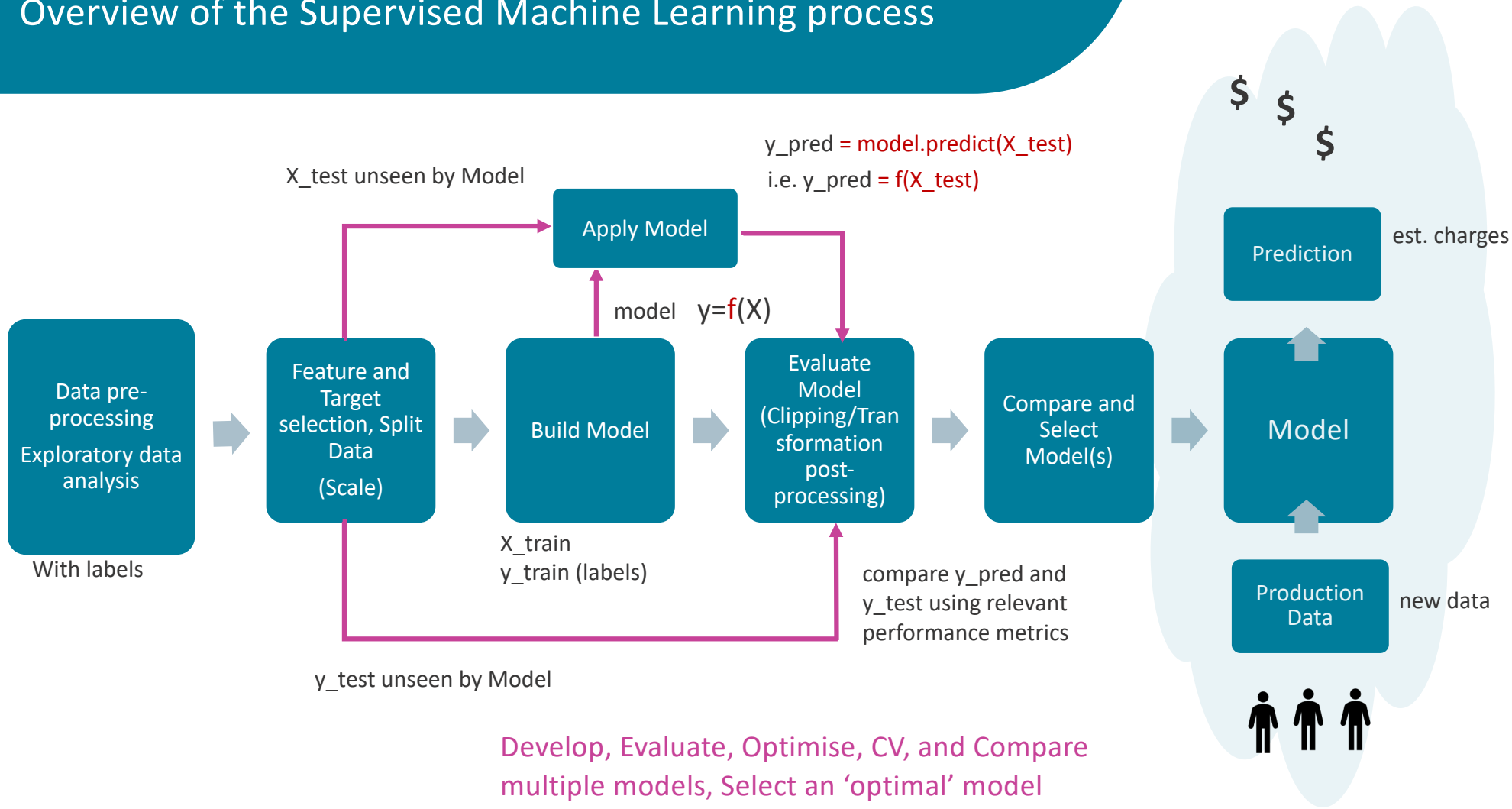
2. Ensemble Methods:

- Combine multiple models to enhance performance (e.g., Bagging, Boosting, Stacking) by leveraging their collective strengths.

3. Model Optimisation:

- Threshold optimisation: Adjust classification thresholds to balance precision and recall based on the specific evaluation metrics.
- Hyperparameter tuning: Use techniques like Grid Search to explore different hyperparameter combinations and select the best model using cross-validation.

Overview of the Supervised Machine Learning process

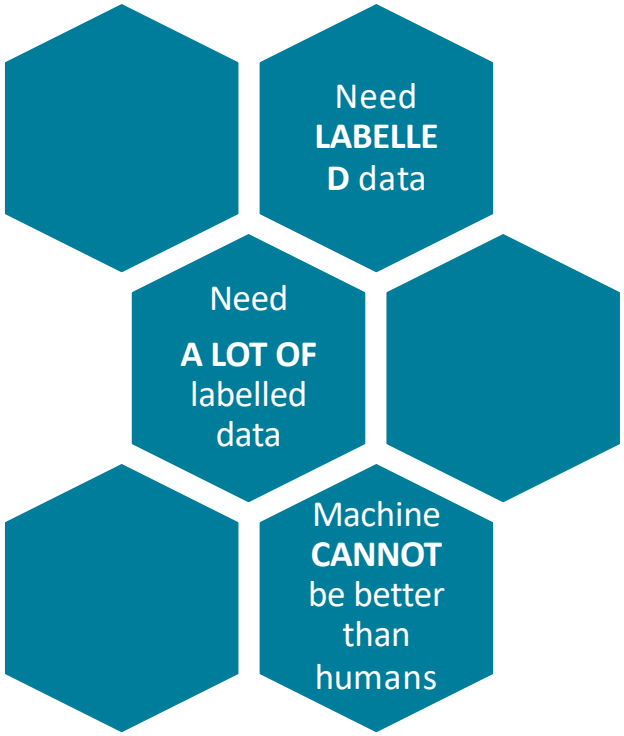


Develop, Evaluate, Optimise, CV, and Compare multiple models, Select an 'optimal' model

What you have learned today

- Model optimisation by tuning hyperparameters
- Cross validation to better estimate the model's performance on unseen data
- Create ensemble learners to make robust and accurate predictions

Problems with supervised machine learning



Topic 9 will introduce Unsupervised Machine Learning to uncover hidden structures without labels

Kotu and Deshpande, 2019, chapter 4