Data Mining & Knowledge Discovery

Classification - 02

Md. Biplob Hosen

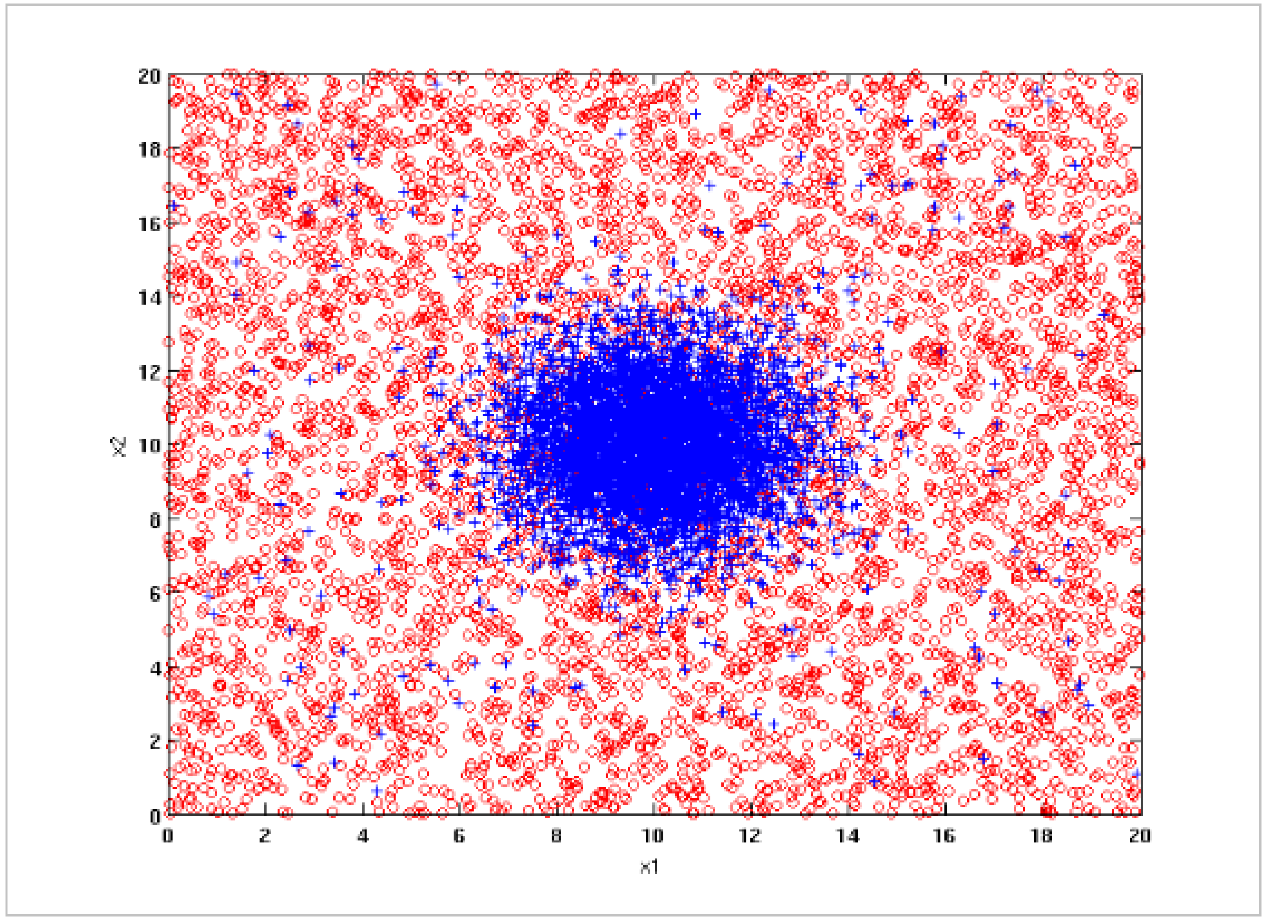
Lecturer, IIT-JU

Email: biplob.hosen@juniv.edu

Model Overfitting

- Classification models try to show the lowest error on the training set.
- However, even if a model fits well over the training data, it can still show poor generalization performance, a phenomenon known as model overfitting.
- Model overfitting occurs when a machine learning model becomes too complex.
- As a result, the model performs exceedingly well on the training data but fails to generalize to new, unseen data. In other words, the model becomes too tailored to the training dataset and loses its ability to make accurate predictions on other data points.
- Consider the two-dimensional data set shown in the following figure. The data set contains instances that belong to two separate classes, represented as + and o, respectively, where each class has 5400 instances. All instances belonging to the class were generated from a uniform distribution.
- For the + class, 5000 instances were generated from a Gaussian distribution centered at (10,10) with unit variance, while the remaining 400 instances were sampled from the same uniform distribution as the o class.

Model Overfitting



Two class problem:

+ : 5400 instances

- **5000 instances generated from a Gaussian centered at (10,10)**
- **400 noisy instances added**

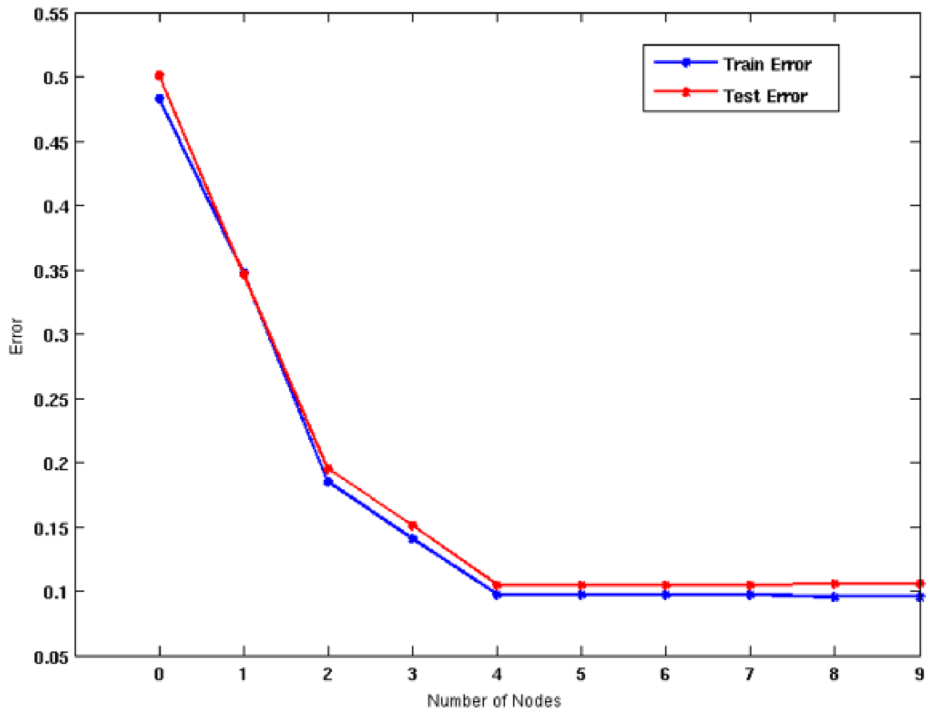
o : 5400 instances

- **Generated from a uniform distribution**

10 % of the data used for training and 90% of the data used for testing

Model Overfitting & Underfitting

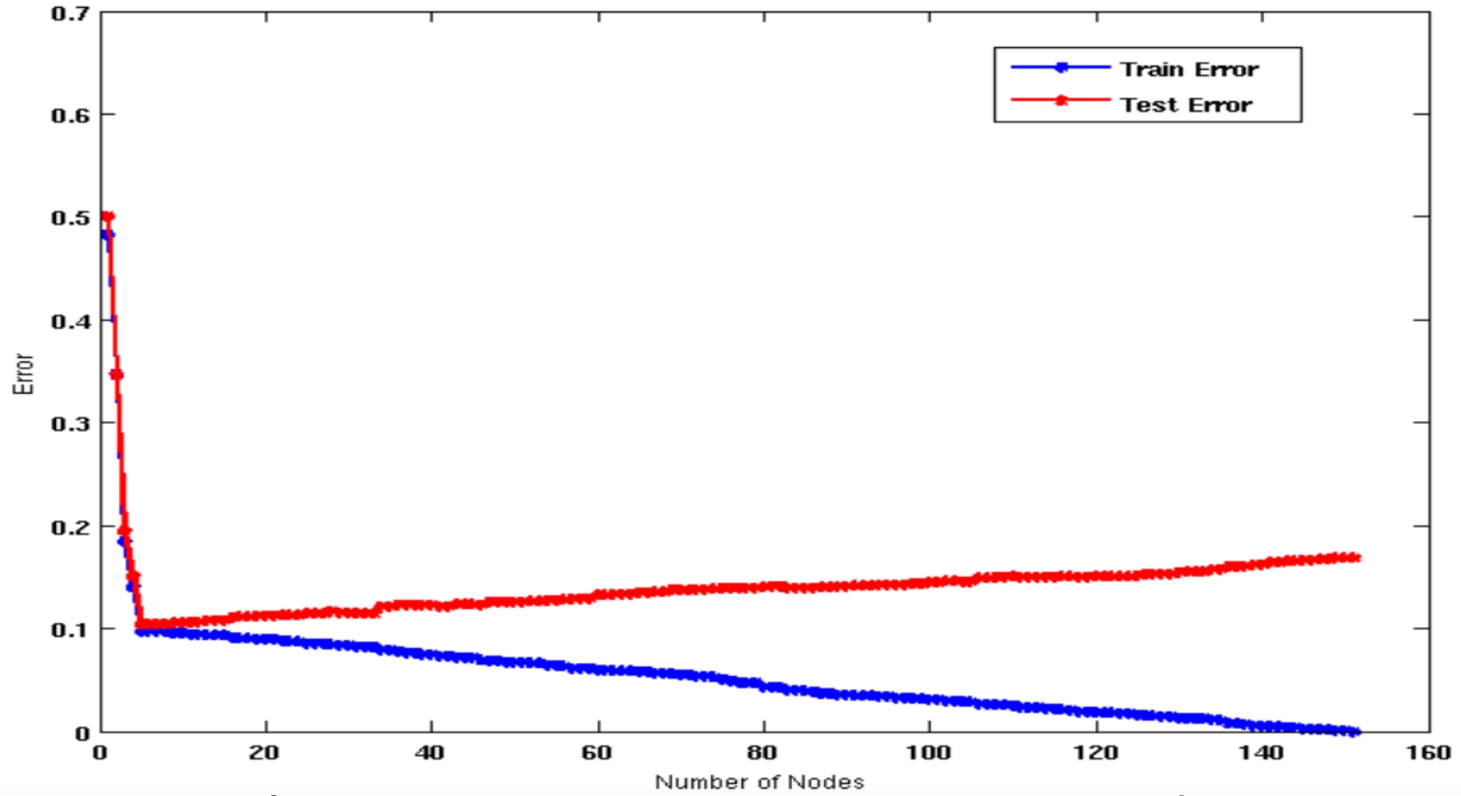
- Following figure shows changes in the training and test error rates as the size of the tree varies from 1 to 8.
- Both error rates are initially large when the tree has only one or two leaf nodes. This situation is known as **model underfitting**.



- Underfitting occurs when the learned decision tree is too simplistic, and thus, incapable of fully representing the true relationship between the attributes and the class labels.
- As we increase the tree size from 1 to 8, we can observe two effects.
- First, both the error rates decrease since larger trees are able to represent more complex decision boundaries.
- Second, the training and test error rates are quite close to each other, which indicates that the performance on the training set is fairly representative of the generalization performance.

Model Overfitting

- As we further increase the size of the tree from 8 to 150, the training error continues to steadily decrease till it eventually reaches zero.



- However, in a striking contrast, the test error rate ceases to decrease any further beyond a certain tree size, and then it begins to increase.
- The training error rate thus grossly under-estimates the test error rate once the tree becomes too large.
- Further, the gap between the training and test error rates keeps on widening as we increase the tree size.
- This behavior, which may seem counter-intuitive at first, can be attributed to the phenomena of **model overfitting**.

Reasons for Model Overfitting

- **Limited Training Size:** A training set consisting of a finite number of instances can only provide a limited representation of the overall data.
- Hence, it is possible that the patterns learned from a training set do not fully represent the true patterns in the overall data, leading to model overfitting.
- In general, as we increase the size of a training set (number of training instances), the patterns learned from the training set start resembling the true patterns in the overall data.
- Hence, the effect of overfitting can be reduced by increasing the training size, as illustrated in the following example.

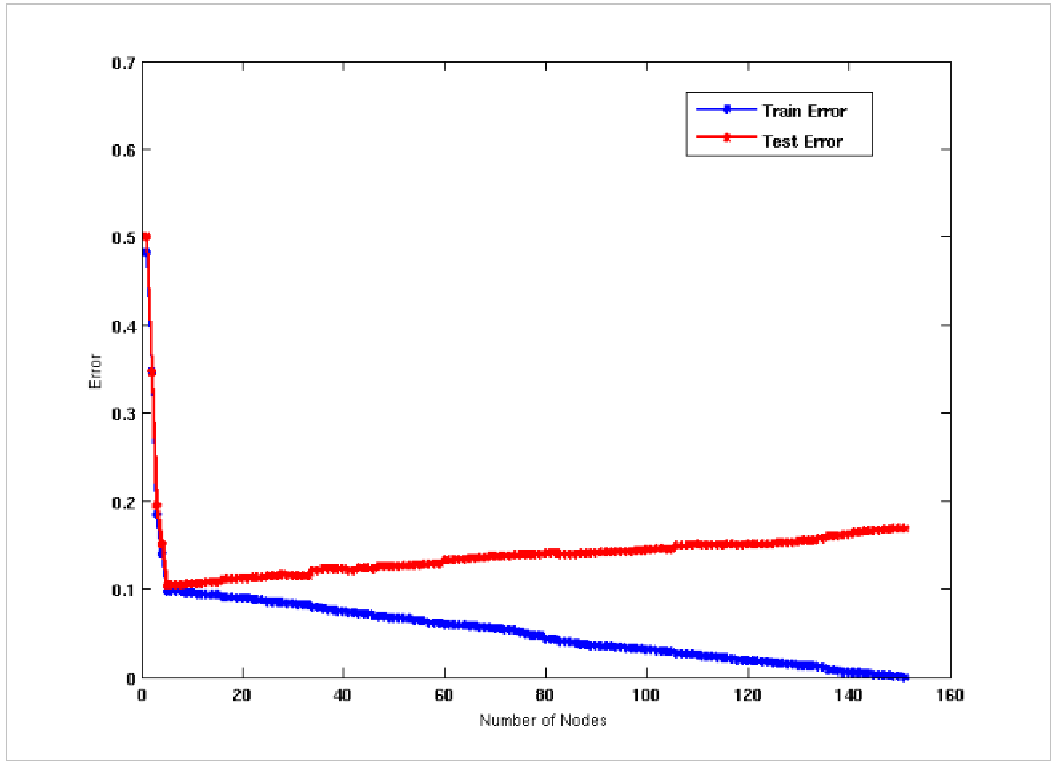
Example Effect of Training Size:

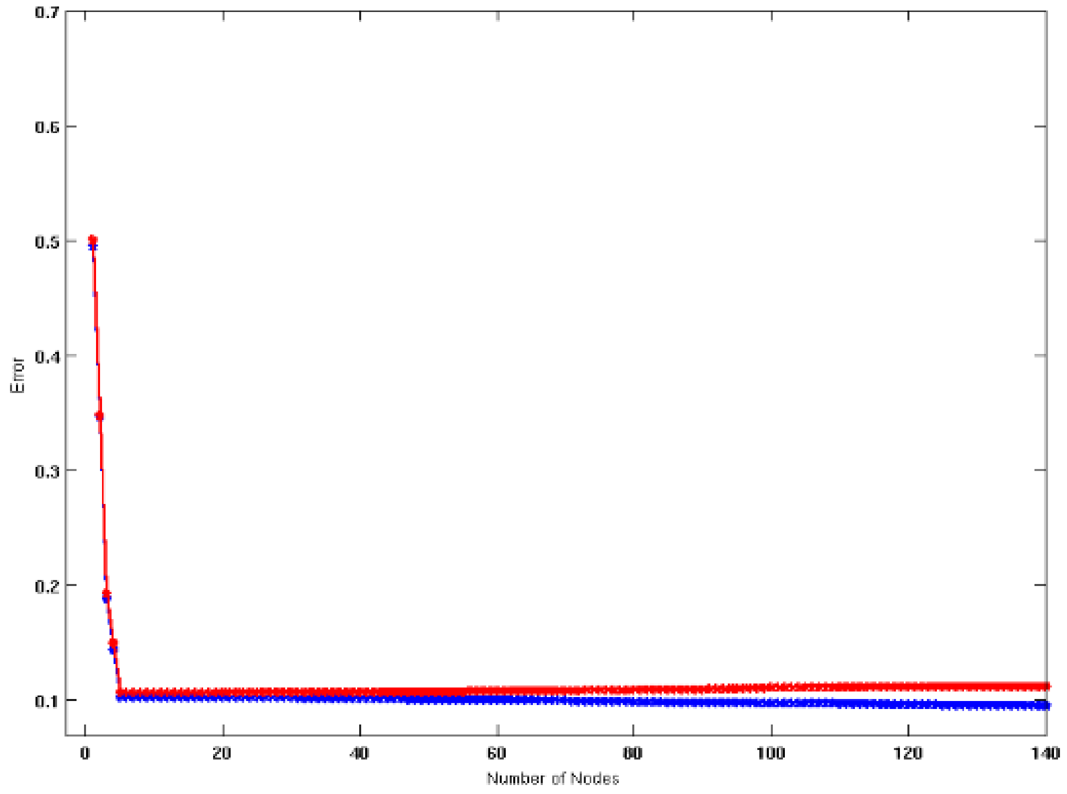
- Suppose that we use twice the number of training instances than what we had used in the previous example. Specifically, we use 20% data for training and remainder for testing.

Limited Training Size

- Following figure shows the training and test error rates as the size of the tree is varied from 1 to 150.
- There are two major differences in the trends shown in this figure and previous Figure.
- First, even though the training error rate decreases with increasing tree size in both figures, its rate of decrease is much smaller when we use twice the training size.
- Second, for a given tree size, the gap between the training and test error rates is much smaller when we use twice the training size.
- These differences suggest that the patterns learned using 20% of data for training are more generalizable than those learned using 10% of data for training.

Limited Training Size





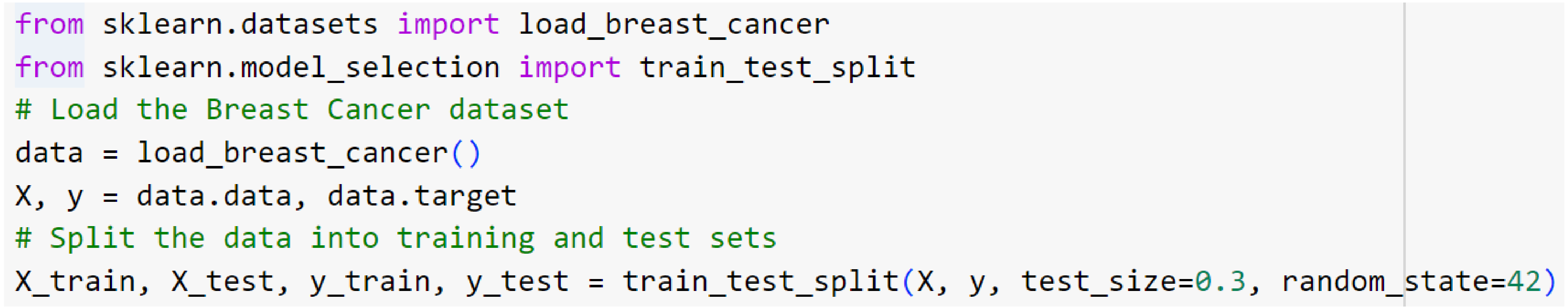
Using twice the number of data instances

Reasons for Model Overfitting

- **High Model Complexity:** Generally, a more complex model has a better ability to represent complex patterns in the data.
- For example, decision trees with larger number of leaf nodes can represent more complex decision boundaries than decision trees with fewer leaf nodes.
- However, an overly complex model also has a tendency to learn specific patterns in the training set that do not generalize well over unseen instances.
- Models with high complexity should thus be judiciously used to avoid overfitting.
- One measure of model complexity is the number of “parameters” that need to be inferred from the training set.
- For example, in the case of decision tree induction, the attribute test conditions at internal nodes correspond to the parameters of the model that need to be inferred from the training set.
- A decision tree with larger number of attribute test conditions (and consequently more leaf nodes) thus involves more “parameters” and hence is more complex.

Overfitting - Python Example

- Let's consider a classification example using the famous "Breast Cancer Wisconsin" dataset. We will use a Decision Tree classifier to demonstrate model overfitting.
- Dataset: First, we'll load the Breast Cancer dataset and split it into training and test sets:



```python
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
# Load the Breast Cancer dataset
data = load_breast_cancer()
X, y = data.data, data.target
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Continue…

- **Model 1: Simple Decision Tree Classifier** with default hyperparameters:

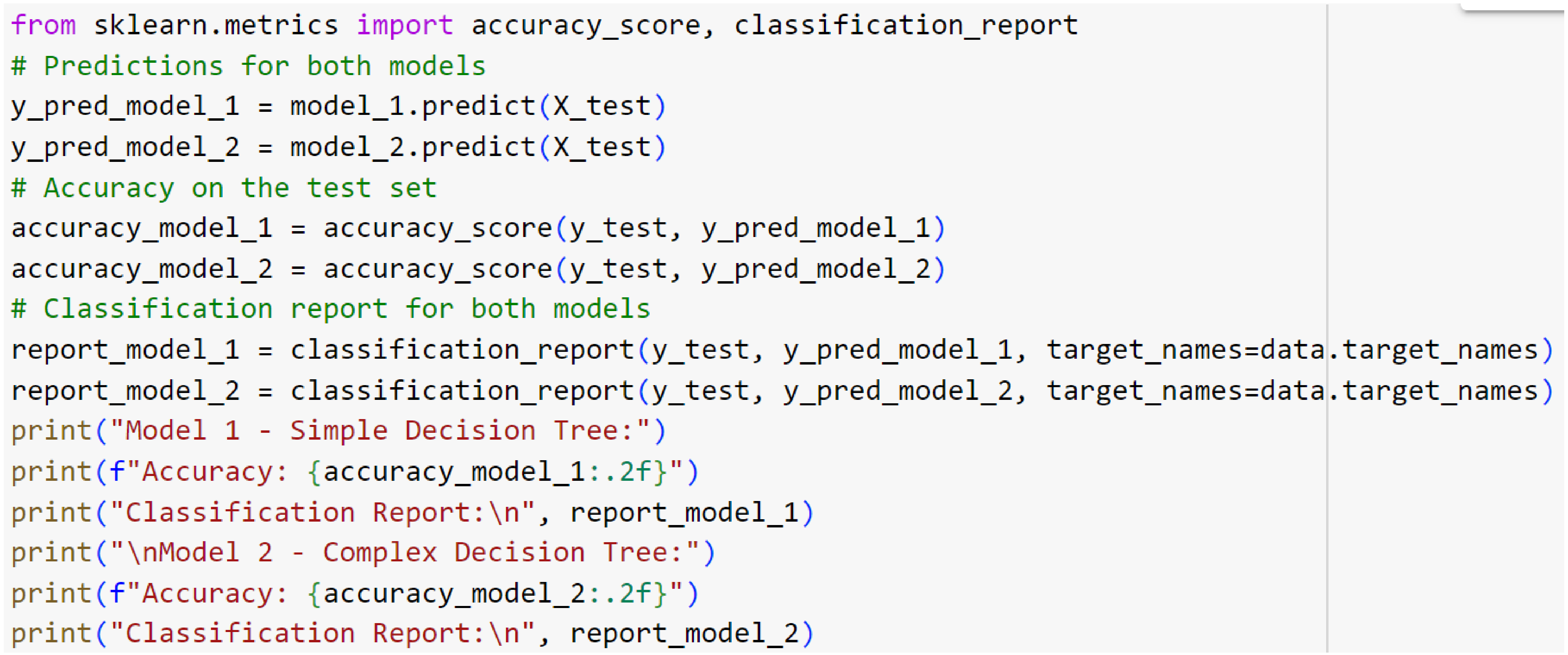
```
from sklearn.tree import DecisionTreeClassifier

# Create and fit the Decision Tree classifier
model_1 = DecisionTreeClassifier(random_state=42)
model_1.fit(X_train, y_train)
```

- **Model 2: Complex Decision Tree Classifier** with deeper trees, allowing it to learn more intricate patterns in the training data:

```
# Create and fit the complex Decision Tree classifier
model_2 = DecisionTreeClassifier(max_depth=20, random_state=42)
model_2.fit(X_train, y_train)
```

Continue…

- **Model Evaluation:** Evaluate both models on the test set and compare their performance:



```python
from sklearn.metrics import accuracy_score, classification_report
# Predictions for both models
y_pred_model_1 = model_1.predict(X_test)
y_pred_model_2 = model_2.predict(X_test)
# Accuracy on the test set
accuracy_model_1 = accuracy_score(y_test, y_pred_model_1)
accuracy_model_2 = accuracy_score(y_test, y_pred_model_2)
# Classification report for both models
report_model_1 = classification_report(y_test, y_pred_model_1, target_names=data.target_names)
report_model_2 = classification_report(y_test, y_pred_model_2, target_names=data.target_names)
print("Model 1 - Simple Decision Tree:")
print(f"Accuracy: {accuracy_model_1:.2f}")
print("Classification Report:\n", report_model_1)
print("\nModel 2 - Complex Decision Tree:")
print(f"Accuracy: {accuracy_model_2:.2f}")
print("Classification Report:\n", report_model_2)
```

Continue…

- **Result:** The output will show the accuracy and classification reports for both models. The complex Decision Tree classifier (Model 2) might achieve higher accuracy on the training set compared to the simple Decision Tree classifier (Model 1).

Summary:

- Model 1 (Simple Decision Tree Classifier) is less likely to overfit the data as it has a smaller depth, which results in a simpler decision boundary. It may have slightly lower accuracy on the training data but is expected to perform better on new, unseen data.
- Model 2 (Complex Decision Tree Classifier) might be more prone to overfitting due to the deeper tree, which allows it to memorize the training data better. It may have excellent performance on the training data but may not generalize well to new data, leading to a drop in accuracy on the test set.

Model Selection

- There are many possible classification models with varying levels of model complexity that can be used to capture patterns in the training data.
- Among these possibilities, we want to select the model that shows lowest generalization error rate.
- The process of selecting a model with the right level of complexity, which is expected to generalize well over unseen test instances, is known as model selection.
- The training error rate cannot be reliably used as the sole criterion for model selection.
- In the following, we present three generic approaches to estimate the generalization performance of a model that can be used for model selection.

Model Selection: Using Validation Set

- Divide training data into two parts:
- Training set: use for model building.
- Validation set: use for estimating generalization error.
    - Note: validation set is not the same as test set.
- Drawback: Less data available for training.
- Given a training set D.train, we can partition D.train into two smaller subsets, D.tr and D.val, such that D.tr is used for training while D.val is used as the validation set.
- For example, two-thirds of D.train can be reserved as D.tr for training, while the remaining one-third is used as D.val for computing validation error rate.
- For any choice of classification model m that is trained on D.tr, we can estimate its validation error rate on D.val, errval(m).
- The model that shows the lowest value of errval(m) can then be selected as the preferred choice of model.

Model Selection: Incorporating Model Complexity

- Since the chance for model overfitting increases as the model becomes more complex, a model selection approach should not only consider the training error rate but also the model complexity.
- This strategy is inspired by a well known principle known as **Occam's razor or the principle of parsimony**, which suggests that given two models with the same errors, the simpler model is preferred over the more complex model.
- A generic approach to account for model complexity while estimating generalization performance is formally described as follows:
- Given a training set D.train, let us consider learning a classification model m that belongs to a certain class of models, M.
- For example, if represents the M set of all possible decision trees, then m can correspond to a specific decision tree learned from the training set.
- We are interested in estimating the generalization error rate of m, gen.error(m).

Model Selection: Incorporating Model Complexity

- As discussed previously, the training error rate of m, train.error(m, D.train), can under-estimate gen.error(m) when the model complexity is high.
- Hence, we represent gen.error(m) as a function of not just the training error rate but also the model complexity of M as follows:
  - gen.error(m)=train.error(m, D.train)+α×complexity(M),
- Where α is a hyper-parameter that strikes a balance between minimizing training error and reducing model complexity.
- A higher value of α gives more emphasis to the model complexity in the estimation of generalization performance.
- To choose the right value of α, we can make use of the validation set.
- For example, we can iterate through a range of values of α and for every possible value, we can learn a model on a subset of the training set, D.tr, and compute its validation error rate on a separate subset, D.val.
- We can then select the value of α that provides the lowest validation error rate.

Model Selection: Estimating Statistical Bounds

- Instead of using previous equation to estimate the generalization error rate of a model, an alternative way is to apply a statistical correction to the training error rate of the model that is indicative of its model complexity.
- This can be done if the probability distribution of training error is available or can be assumed.
- For example, the number of errors committed by a leaf node in a decision tree can be assumed to follow a binomial distribution.
- We can thus compute an upper bound limit to the observed training error rate that can be used for model selection.

Model Selection for Decision Trees

Pre-Pruning (Early Stopping Rule)

- Stop the algorithm before it becomes a fully-grown tree
- Typical stopping conditions for a node:
  - Stop if all instances belong to the same class
  - Stop if all the attribute values are the same
- More restrictive conditions:
  - Stop if number of instances is less than some user-specified threshold
  - Stop if class distribution of instances are independent of the available features (e.g., using χ 2 test)
  - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).
  - Stop if estimated generalization error falls below certain threshold

Model Selection for Decision Trees

Post-pruning

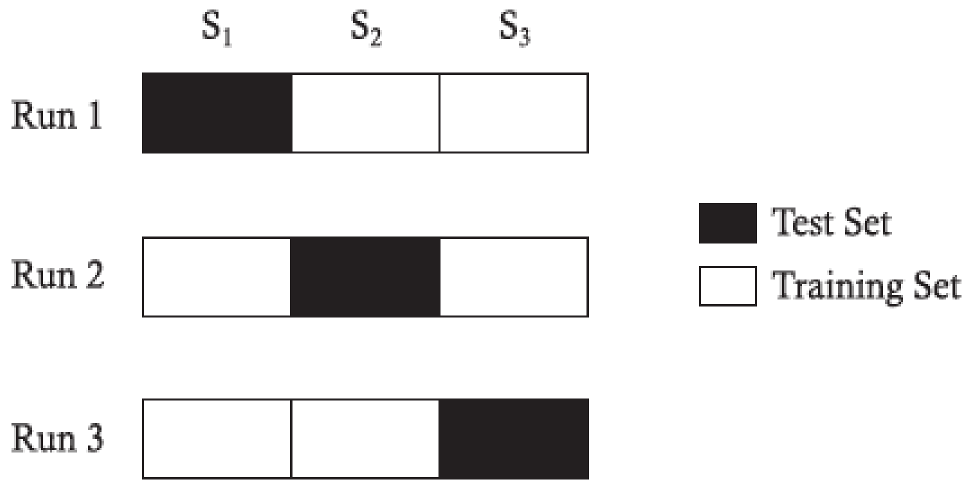
- Grow decision tree to its entirety
- Subtree replacement
  - Trim the nodes of the decision tree in a bottom-up fashion
  - If generalization error improves after trimming, replace sub-tree by a leaf node
  - Class label of leaf node is determined from majority class of instances in the sub-tree

Model Evaluation

- Purpose: To estimate performance of classifier on previously unseen data (test set).

1. Holdout: Reserve k% for training and (100-k)% for testing.
   - Random subsampling: repeated holdout.
2. Cross validation:

- Partition data into k disjoint subsets.
- k-fold: train on k-1 partitions, test on the remaining one.
- Example: 3-fold cross-validation.



Hyper-Parameters

- Hyper-parameters are parameters that are set before the model training process and are not learned from the data during training.
- They control various aspects of the learning algorithm and model architecture, influencing how the model learns from the data and generalizes to new, unseen data.
- Unlike model parameters, which are learned from data during training, hyper-parameters are set by the data scientist or machine learning engineer before the training process begins.
- Here are some common examples of hyper-parameters for various machine learning models:

1. **Learning Rate (α):** In gradient-based optimization algorithms, the learning rate determines the step size taken in the direction of the gradient during parameter updates. A small learning rate might lead to slow convergence, while a large learning rate can result in overshooting the optimal solution.
2. **Number of Hidden Layers and Units in Neural Networks:** The architecture of a neural network is determined by the number of hidden layers and the number of units (neurons) in each layer. The depth and width of the network are hyper-parameters that influence its complexity.

Hyper-Parameters

3. **Activation Functions:** Activation functions introduce non-linearity to neural network models. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh. The choice of activation function is a hyper-parameter
4. **Batch Size:** In mini-batch training, the batch size specifies the number of samples used in each training iteration. Larger batch sizes might lead to more stable updates, but they also require more memory.
5. **Regularization Strength (λ):** Regularization is used to prevent overfitting in machine learning models. The regularization strength (λ) controls the amount of regularization applied to the model's parameters.
6. **Number of Trees and Depth in Ensemble Models:** In ensemble models like Random Forest and Gradient Boosting, the number of trees and the maximum depth of each tree are hyperparameters that affect model complexity and performance.

Hyper-Parameters

7. **Kernel and Regularization in Support Vector Machines (SVM):** For SVM classifiers, the choice of kernel (linear, polynomial, radial basis function, etc.) and the regularization parameter (C) are hyperparameters that impact the decision boundary and model performance.
8. **Number of Neighbors in k-Nearest Neighbors (k-NN):** In k-NN, the number of neighbors (k) to consider for classification or regression is a hyperparameter that influences model performance.

- Hyperparameter tuning involves searching for the best combination of hyperparameter values to optimize the model's performance on a validation set. Techniques like grid search, random search, and Bayesian optimization are commonly used for hyperparameter tuning. Careful tuning of hyperparameters is crucial for building accurate and well-performing machine learning models.

Thank You!