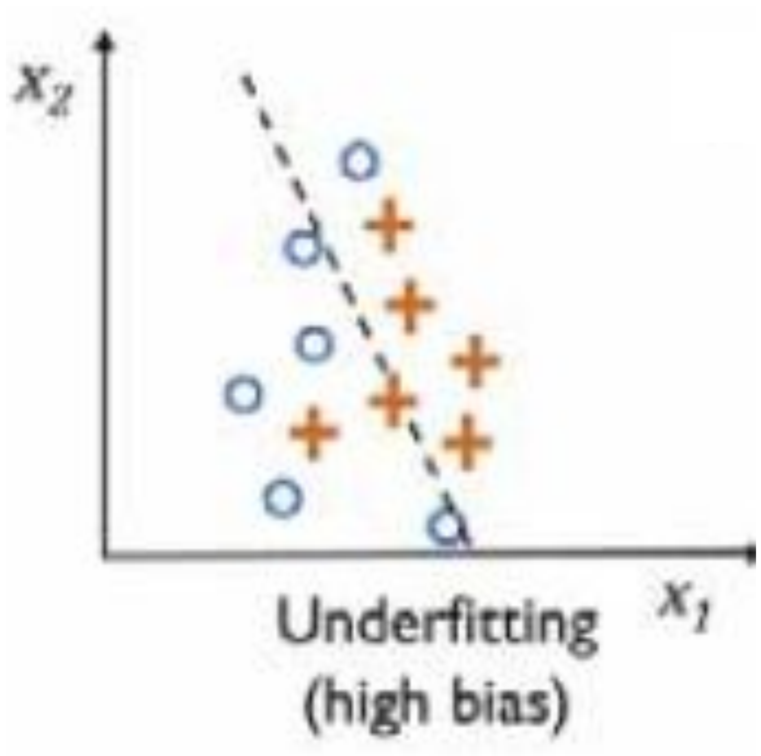
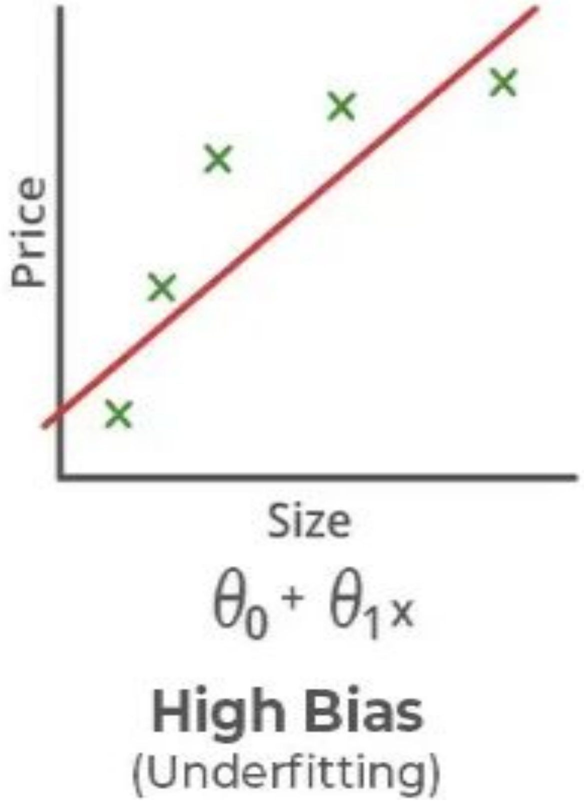
Overfitting and Underfitting

Errors of Machine Learning Models

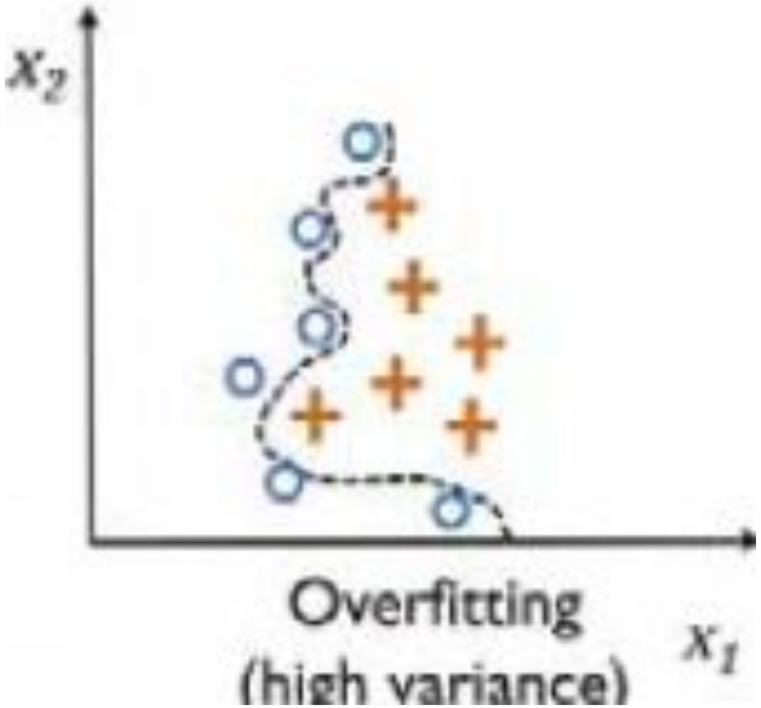
- ***Bias*** – errors due to overly simplistic assumption in the learning algorithms
  - Inability to represent the true relationship between features and target variable
  - *High bias* means that the model has poor performance on both training and test data
- ***Variance*** – errors due to the model's sensitivity to fluctuations in the training data
  - Inability to generalize the model to different training data set
  - *High variance* means that the model performs well for the training data but performs poorly for unseen data

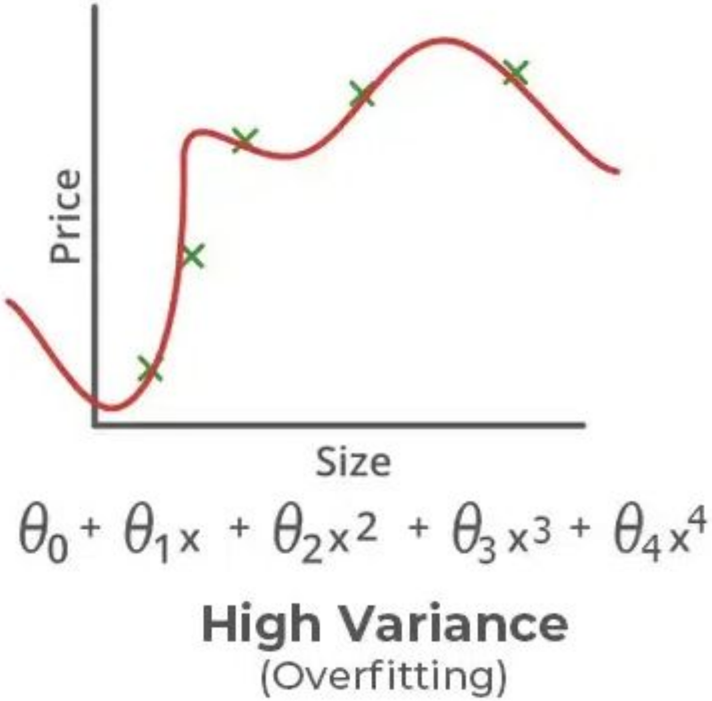
Underfitting



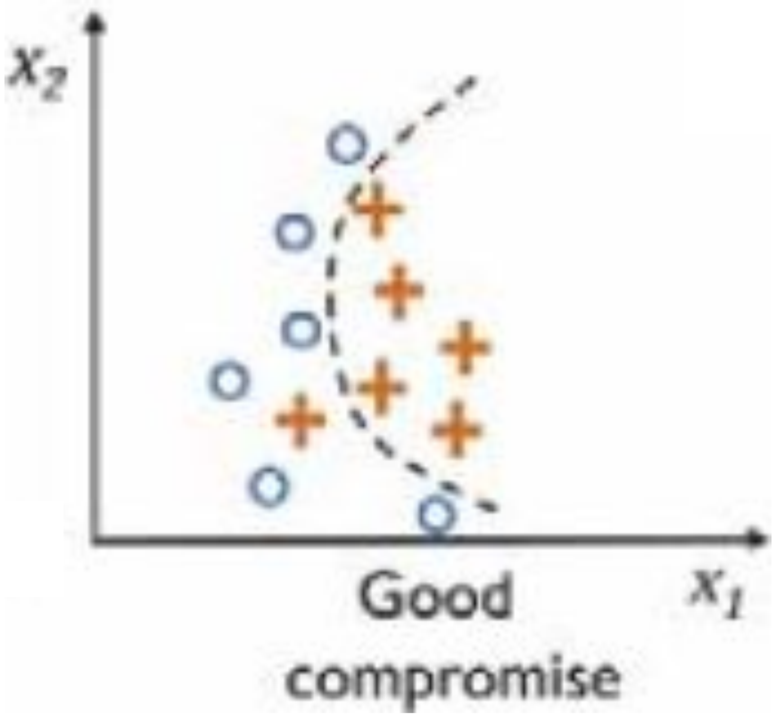


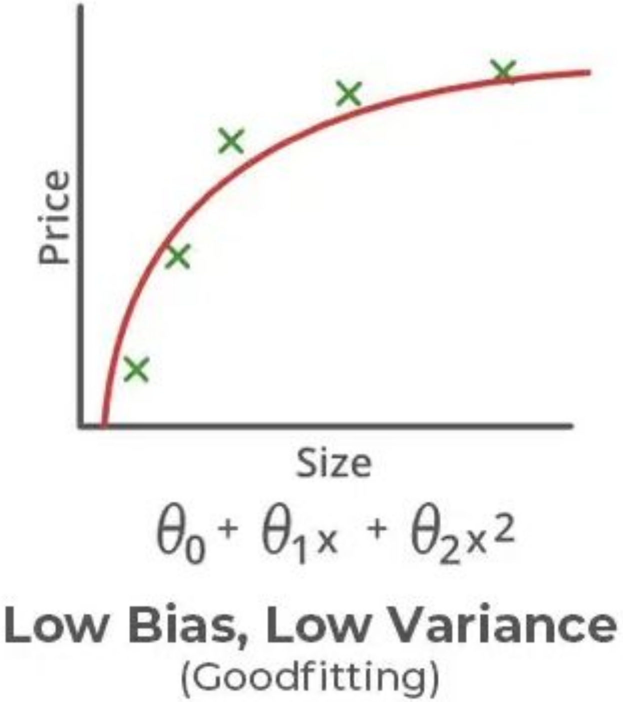
Overfitting





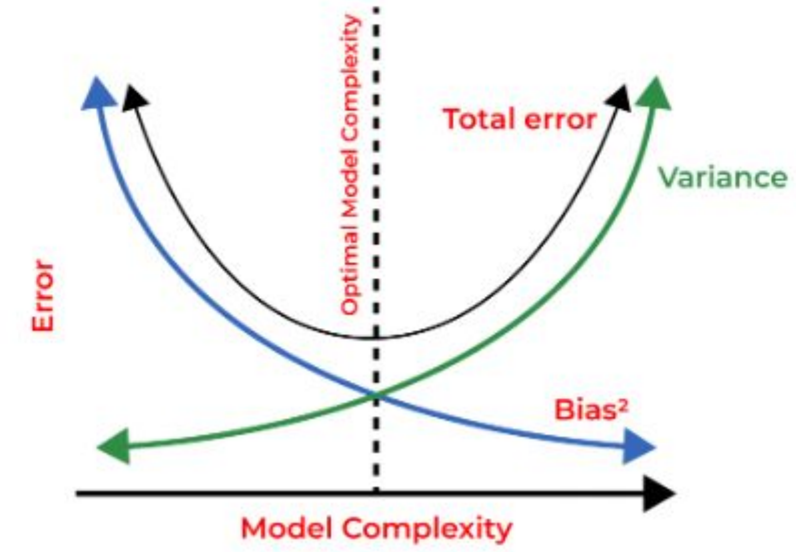
Good Fitting





Reasons for Underfitting/Overfitting

- Model complexity
  - Model too complex
    - Too many features (parameters)
    - Overfitting
  - Model too simple
    - Linear separable model
    - Underfitting
- Noisy data
- Insufficient training data



Remedies for Overfitting/Underfitting

- Too many features
  - Feature selection
  - Regularization
- Linear separability
  - Kernel method
- Noisy data
  - Decision tree pruning
  - SVM soft margin

Regularization

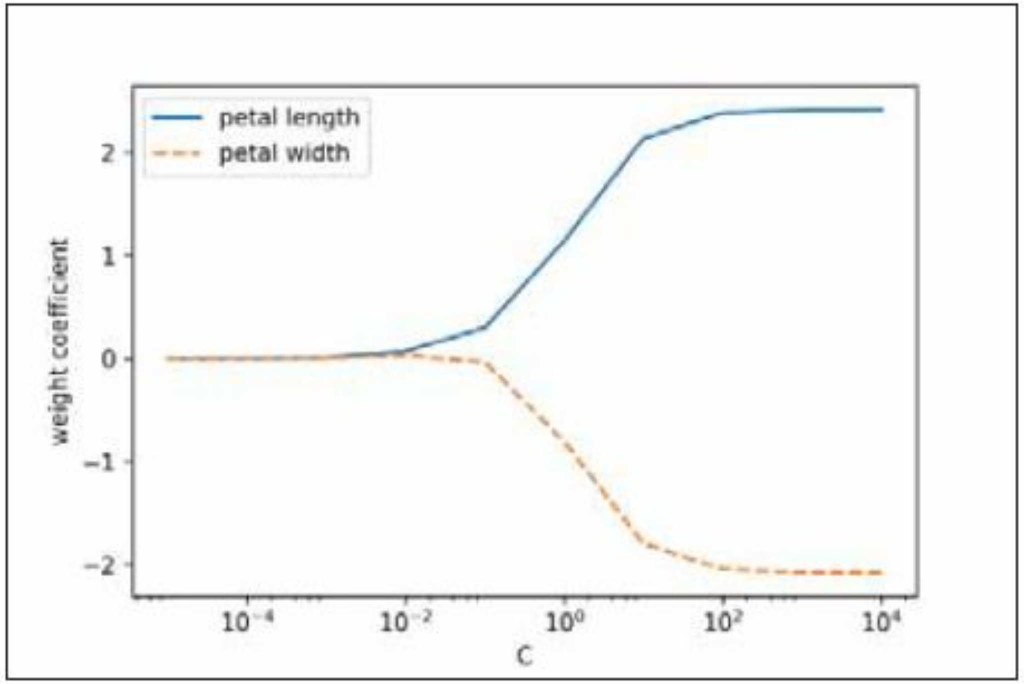
- Add a *regularization term* to the objective function of linear regression, logistic regression algorithms

$$\frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$$

- λ is the so-called **regularization parameter** (hyperparameter)
- Regularization can help
  - Reduce the coefficients of less important features to zero
  - Prevent excessive weighting of outliers or irrelevant features
  - Handle high correlation between features (multicollinearity)
  - Achieve the right balance between bias and variance

Logistic Regression with Regularization

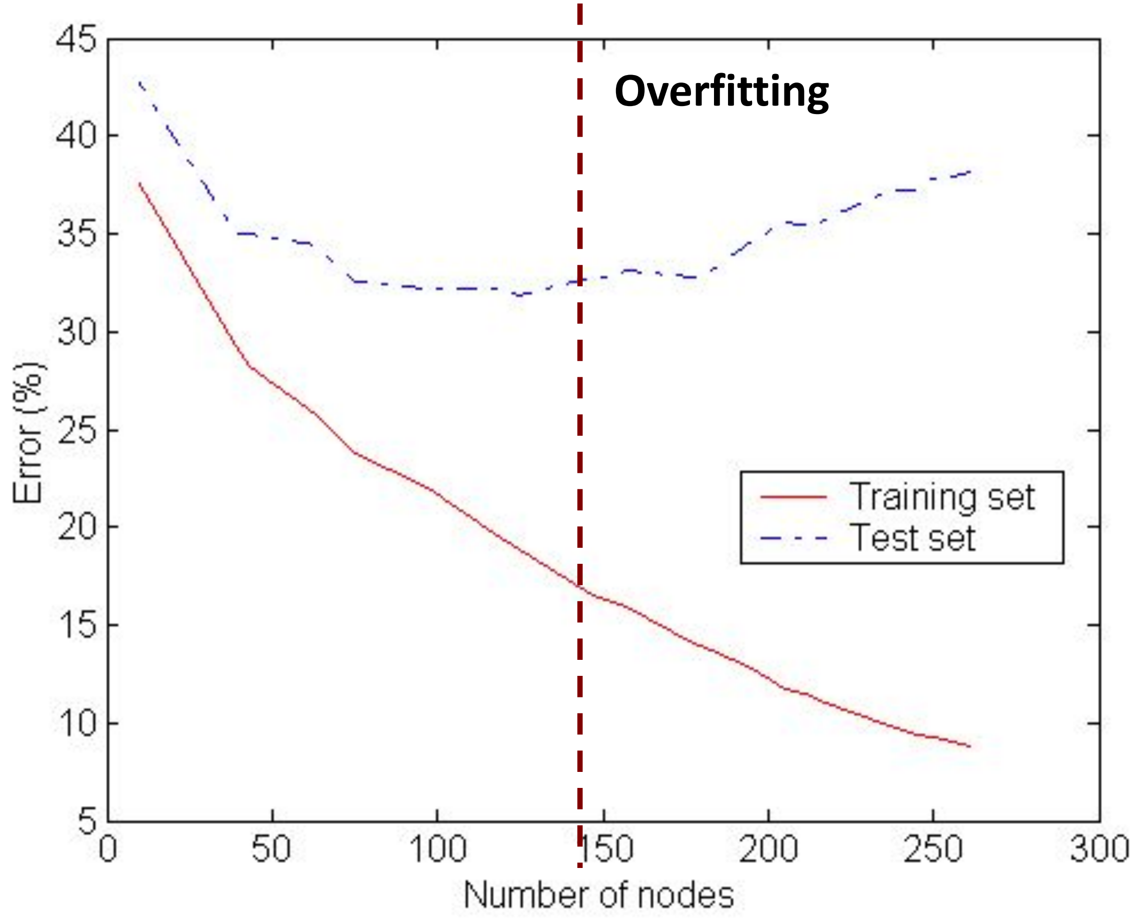
```
>>> weights, params = [], []
>>> for c in np.arange(-5, 5):
...     lr = LogisticRegression(C=10.**c, random_state=1)
...     lr.fit(X_train_std, y_train)
...     weights.append(lr.coef_[1])
...     params.append(10.**c)
>>> weights = np.array(weights)
>>> plt.plot(params, weights[:, 0],
...          label='petal length')
>>> plt.plot(params, weights[:, 1], linestyle='--',
...          label='petal width')
>>> plt.ylabel('weight coefficient')
>>> plt.xlabel('C')
>>> plt.legend(loc='upper left')
>>> plt.xscale('log')
>>> plt.show()
```



Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *Remove branches* from a “fully grown” tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the “best pruned tree”

Number of Nodes and Overfitting



Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary

- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records

- Need new ways for estimating errors

How to Avoid Overfitting

- 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).

How to Avoid Overfitting…

- Post-pruning
  - Grow decision tree to its entirety
  - 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

Estimating Generalization Errors

- Re-substitution errors: error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))

- Methods for estimating generalization errors:
  - Optimistic approach: e'(t) = e(t)
  - Pessimistic approach:
    - For each leaf node: e'(t) = (e(t)+0.5)
    - Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
    - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
      Training error = 10/1000 = 1%
      Generalization error = (10 + 30×0.5)/1000 = 2.5%
  - Reduced error pruning (REP):
    - uses validation data set to estimate generalization error

Example of Post-Pruning

Class = Yes	20
Class = No	10
Error = 10/30	

Training Error (Before splitting) = 10/30

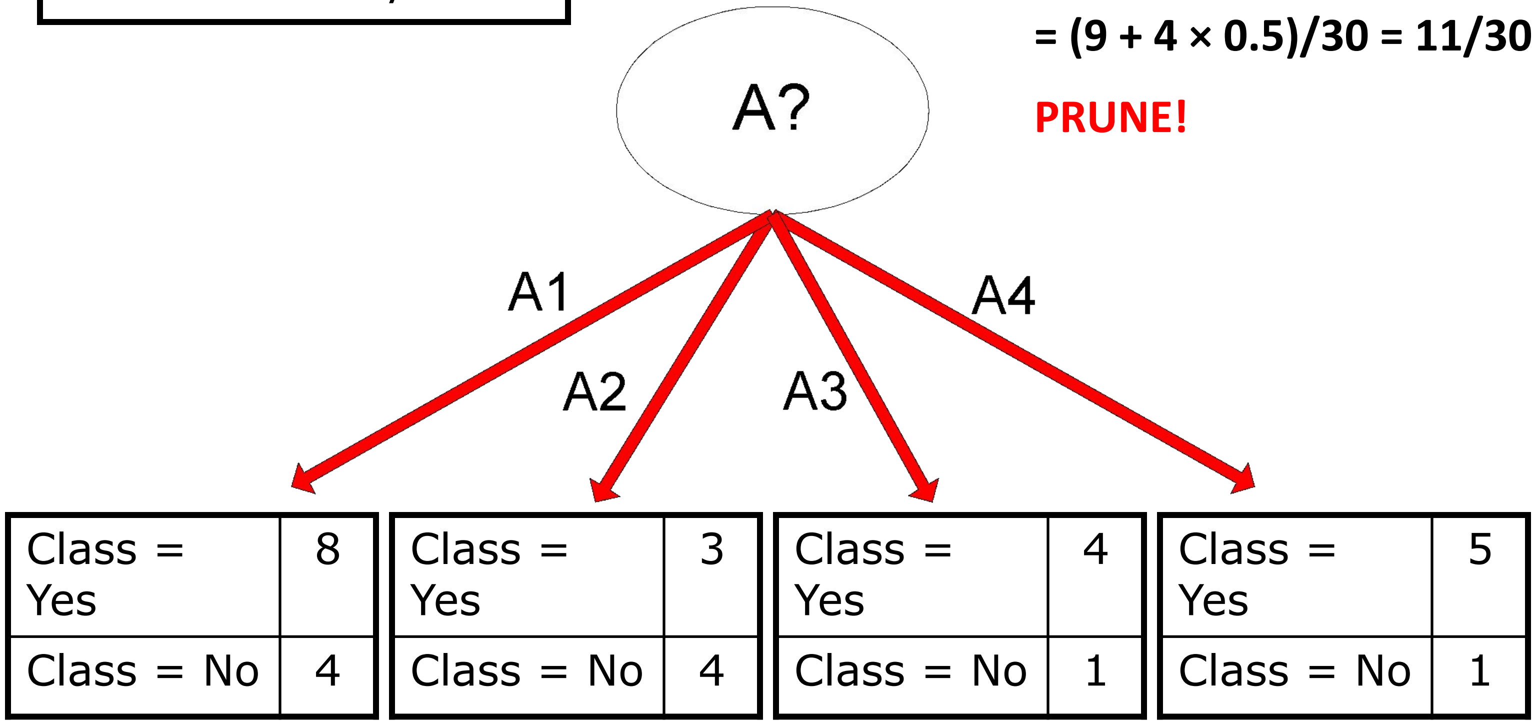
Pessimistic error = (10 + 0.5)/30 = 10.5/30

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= (9 + 4 × 0.5)/30 = 11/30

PRUNE!



Reduced Error Pruning (REP)

Use pruning set to estimate accuracy of sub-trees and accuracy at individual nodes

Let T be a sub-tree rooted at node v

Define:

$$\text{Gain from prunning at } v = \#\text{misclassification in } T - \#\text{misclassification at } v$$

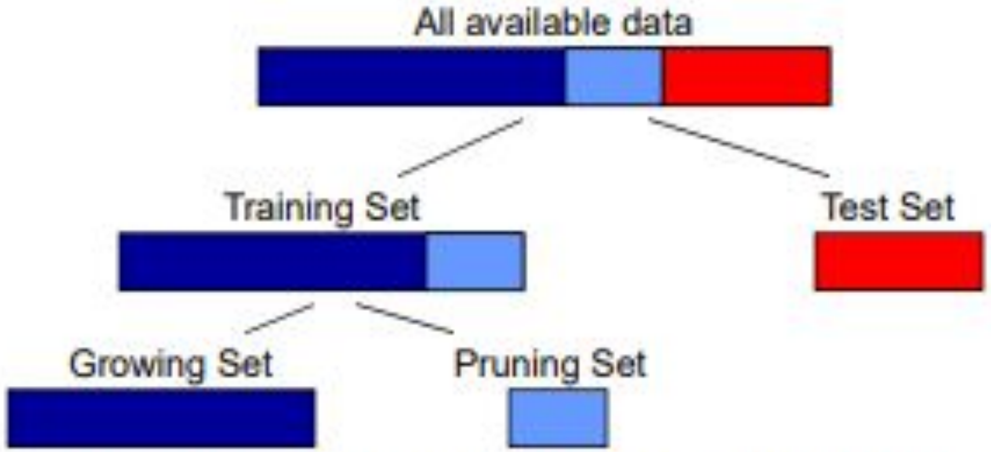
Repeat: prune at node with largest gain until until only negative gain nodes remain

"Bottom-up restriction": T can only be pruned if it does not contain a sub-tree with lower error than T

Partition Data in Tree Induction

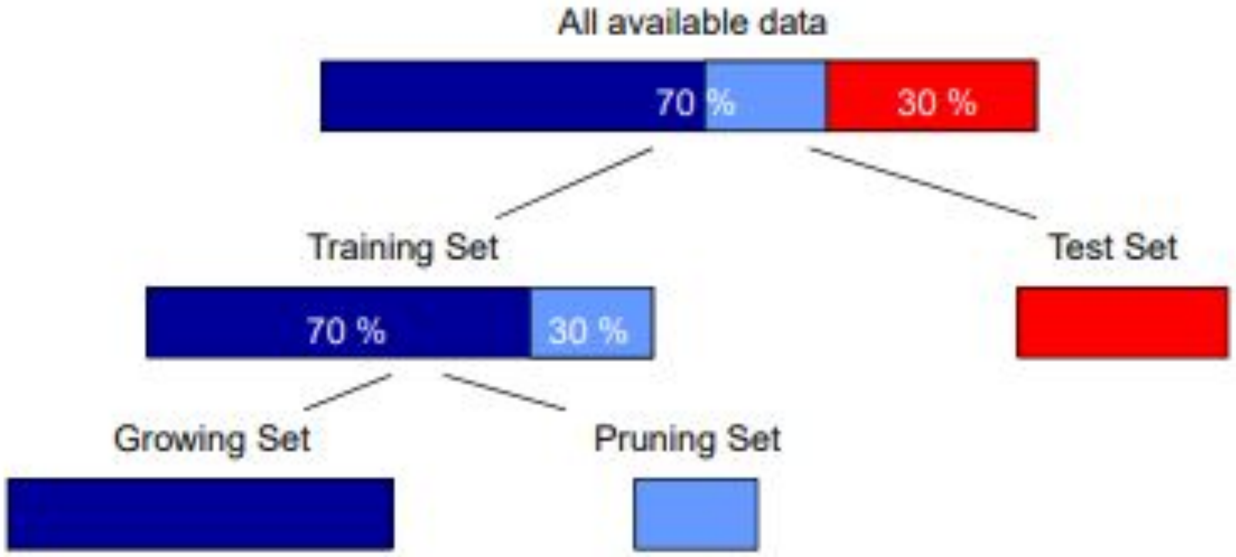
Estimating accuracy of a tree on new data: "Test Set"

Some post pruning methods need an independent data set: "Pruning Set"

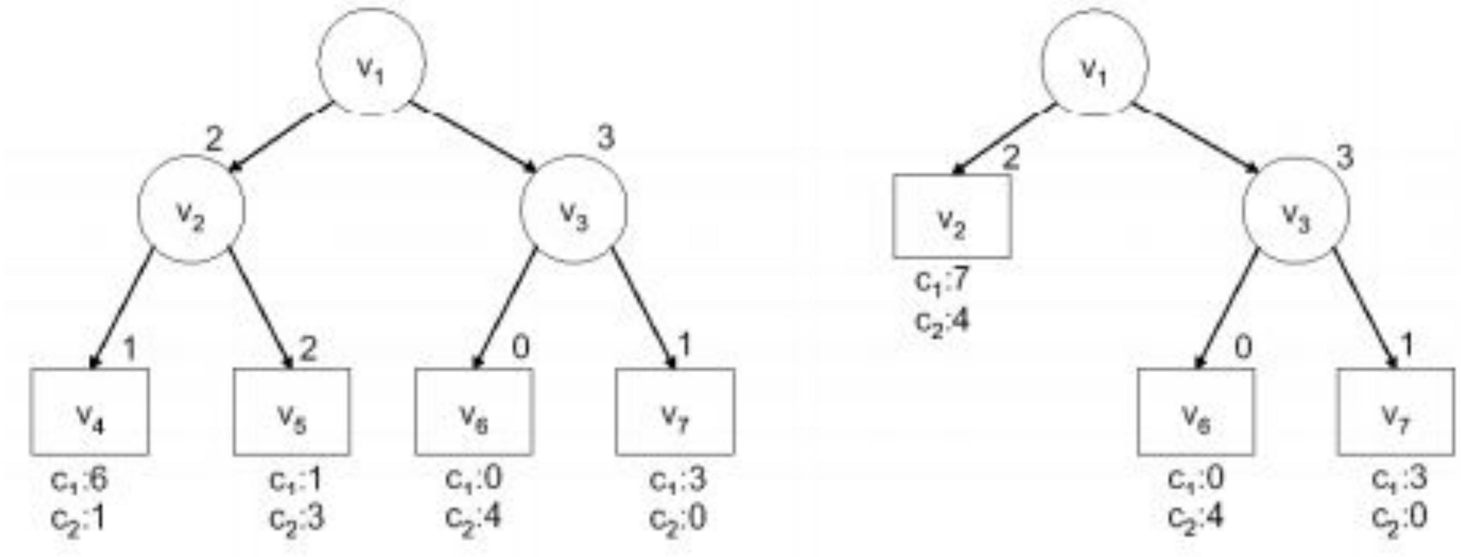


To evaluate the classification technique, experiment with repeated random splits of data

Typical Proportions

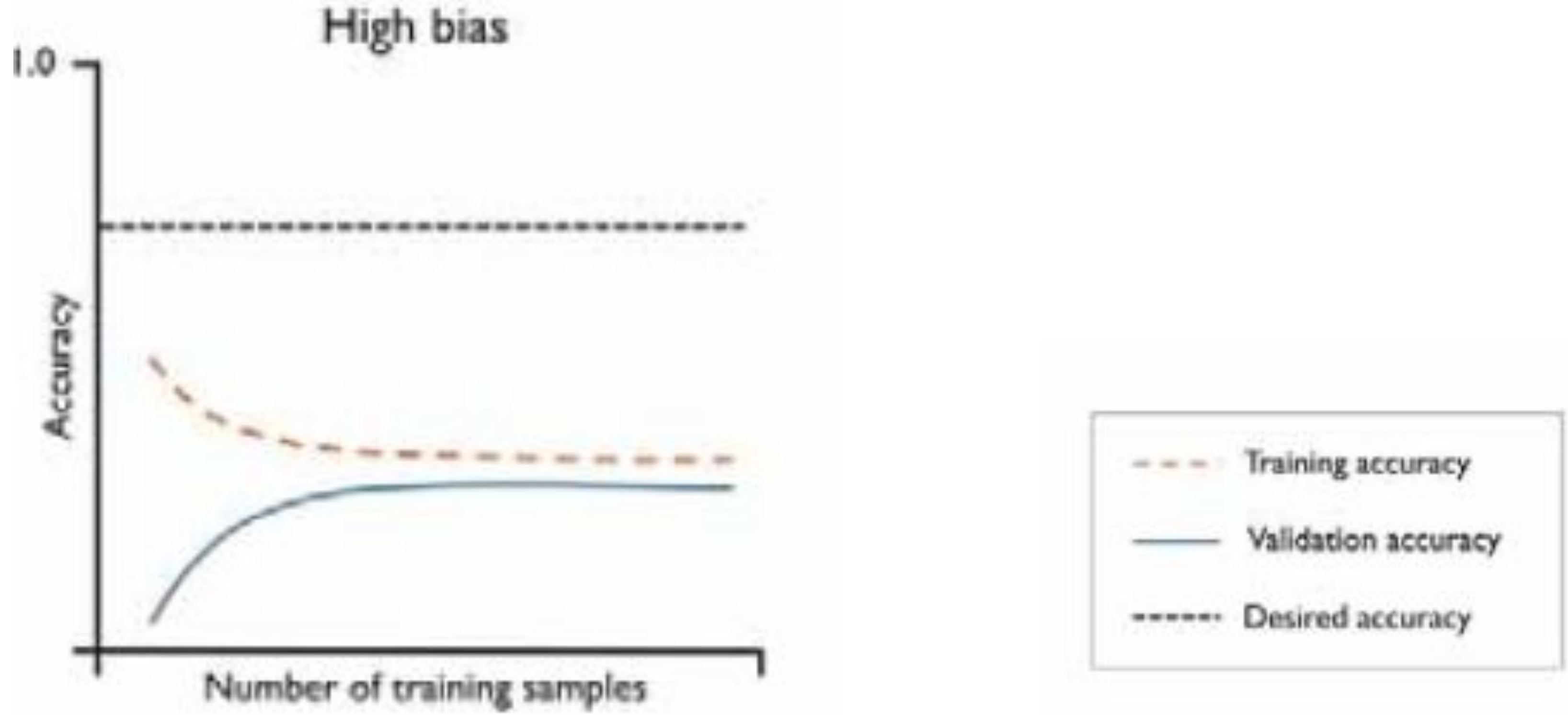


REP Example

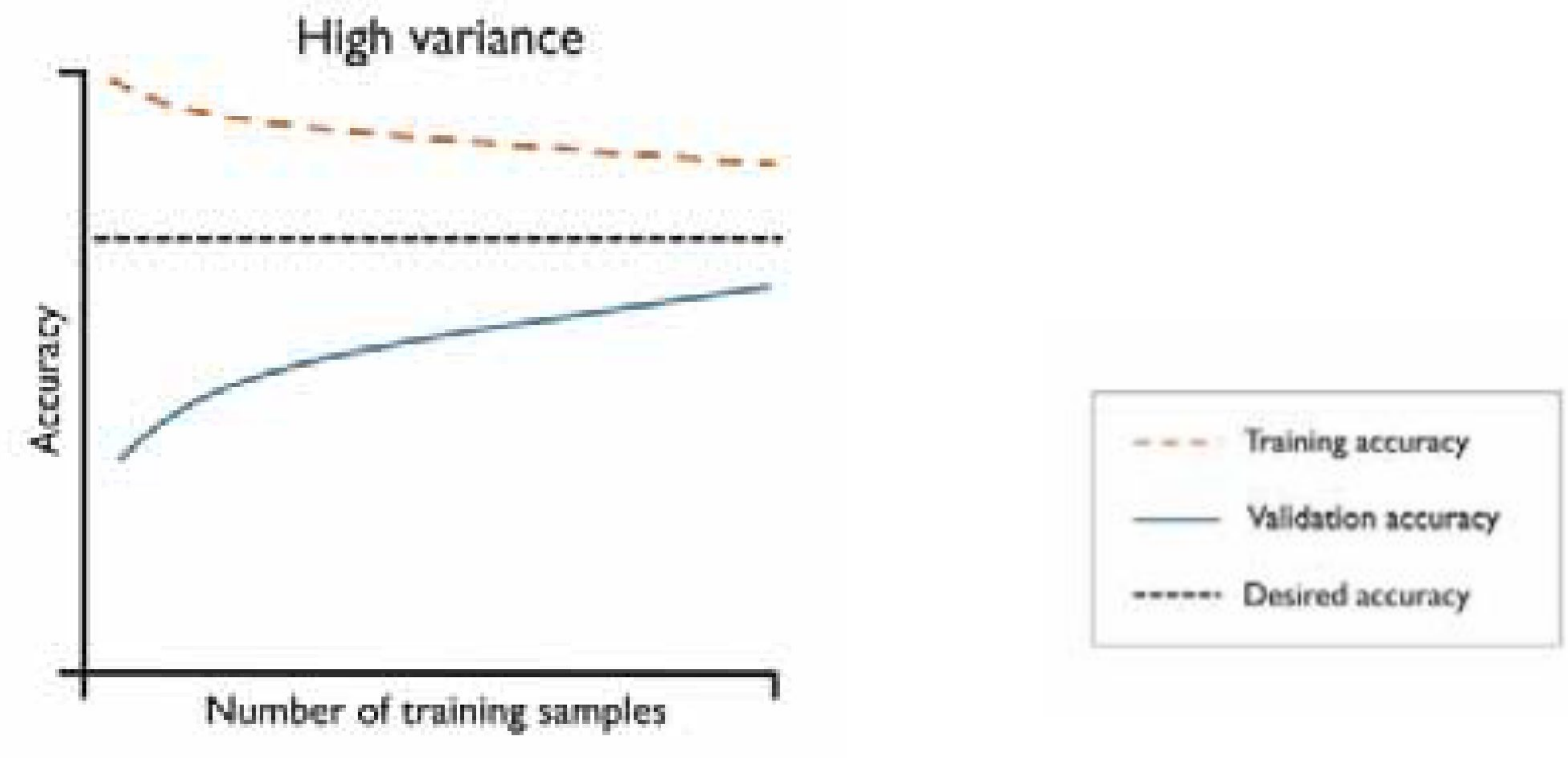


$E(T_{v_2}) = 3, E(v_2) = 2, E(T_{v_3}) = 1, E(v_3) = 3.$

Learning Curve and Underfitting



Learning Curve and Overfitting



Validation Curve and Overfitting/Underfitting