Model Overfitting

Classification Errors

- Training errors (apparent errors)
  - Errors committed on the training set

- Test errors
  - Errors committed on the test set

- Generalization errors
  - Expected error of a model over random selection of records from same distribution
Example Data Set

Two class problem:

+ : 5200 instances
  • 5000 instances generated from a Gaussian centered at (10,10)
  • 200 noisy instances added

o : 5200 instances
  • Generated from a uniform distribution

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

Increasing number of nodes in Decision Trees
Which tree is better?

Decision Tree with 4 nodes

Decision Tree with 50 nodes

Which tree is better?

Model Overfitting

Underfitting: when model is too simple, both training and test errors are large

Overfitting: when model is too complex, training error is small but test error is large
Model Overfitting

- If training data is under-representative, testing errors increase and training errors decrease on increasing number of nodes.
- Increasing the size of training data reduces the difference between training and testing errors at a given number of nodes.

Reasons for Model Overfitting

- Limited Training Size
- High Model Complexity
  - Multiple Comparison Procedure
Effect of Multiple Comparison Procedure

● Consider the task of predicting whether stock market will rise/fall in the next 10 trading days

● Random guessing:
  \[ P(\text{correct}) = 0.5 \]

● Make 10 random guesses in a row:
  \[ P(\#\text{correct} \geq 8) = \frac{\binom{10}{8} + \binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0547 \]

Effect of Multiple Comparison Procedure

● Approach:
  – Get 50 analysts
  – Each analyst makes 10 random guesses
  – Choose the analyst that makes the most number of correct predictions

● Probability that at least one analyst makes at least 8 correct predictions
  \[ P(\#\text{correct} \geq 8) = 1 - (1 - 0.0547)^{50} = 0.9399 \]
Effect of Multiple Comparison Procedure

- Many algorithms employ the following greedy strategy:
  - Initial model: $M$
  - Alternative model: $M' = M \cup \gamma$
    where $\gamma$ is a component to be added to the model (e.g., a test condition of a decision tree)
  - Keep $M'$ if improvement, $\Delta(M,M') > \alpha$

- Often times, $\gamma$ is chosen from a set of alternative components, $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_k\}$

- If many alternatives are available, one may inadvertently add irrelevant components to the model, resulting in model overfitting

Effect of Multiple Comparison - Example

- Use additional 100 noisy variables generated from a uniform distribution along with $X$ and $Y$ as attributes.
- Use 30% of the data for training and 70% of the data for testing
- Using only $X$ and $Y$ as attributes
**Notes on Overfitting**

- Overfitting results in decision trees that are more complex than necessary.
- Training error does not provide a good estimate of how well the tree will perform on previously unseen records.
- Need ways for estimating generalization errors.

**Model Selection**

- Performed during model building.
- Purpose is to ensure that model is not overly complex (to avoid overfitting).
- Need to estimate generalization error:
  - Using Validation Set
  - Incorporating Model Complexity
  - Estimating Statistical Bounds
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

Model Selection:
Incorporating Model Complexity

- Rationale: Occam’s Razor
  - Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
  - A complex model has a greater chance of being fitted accidentally by errors in data
  - Therefore, one should include model complexity when evaluating a model

\[ \text{Gen. Error(Model)} = \text{Train. Error(Model, Train. Data)} + \alpha \times \text{Complexity(Model)} \]
Estimating the Complexity of Decision Trees

- **Pessimistic Error Estimate** of decision tree $T$ with $k$ leaf nodes:

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{\text{train}}}$$

- $err(T)$: error rate on all training records
- $\Omega$: trade-off hyper-parameter (similar to $\alpha$)
  - Relative cost of adding a leaf node
- $k$: number of leaf nodes
- $N_{\text{train}}$: total number of training records

---

Estimating the Complexity of Decision Trees: Example

<table>
<thead>
<tr>
<th>Decision Tree, $T_L$</th>
<th>Decision Tree, $T_R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e(T_L) = 4/24$</td>
<td>$e(T_R) = 6/24$</td>
</tr>
<tr>
<td>$\Omega = 1$</td>
<td></td>
</tr>
</tbody>
</table>

\[
e_{\text{gen}}(T_L) = \frac{4}{24} + \frac{1 \times 0}{24} = \frac{11}{24} = 0.458
\]

\[
e_{\text{gen}}(T_R) = \frac{6}{24} + \frac{1 \times 4}{24} = \frac{10}{24} = 0.417
\]
Estimating the Complexity of Decision Trees

- **Resubstitution Estimate:**
  - Using training error as an optimistic estimate of generalization error
  - Referred to as optimistic error estimate

\[
\begin{align*}
\text{e}(T_L) &= 4/24 \\
\text{e}(T_R) &= 6/24
\end{align*}
\]

- **Decision Tree, \( T_L \)**
- **Decision Tree, \( T_R \)**

Minimum Description Length (MDL)

- **Cost(Model,Data) = Cost(Data|Model) + \( \alpha \) x Cost(Model)**
  - Cost is the number of bits needed for encoding.
  - Search for the least costly model.
- **Cost(Data|Model) encodes the misclassification errors.**
- **Cost(Model) uses node encoding (number of children) plus splitting condition encoding.**
Estimating Statistical Bounds

\[
e'(N, e, \alpha) = \frac{e + \frac{\alpha}{2N} + \frac{\alpha^2}{4N^2}}{1 + \frac{\alpha}{N}}
\]

Before splitting: \( e = 2/7, \quad e'(7, 2/7, 0.25) = 0.503 \)

\[
e'(T) = 7 \times 0.503 = 3.521
\]

After splitting:

\[
e(T_L) = 1/4, \quad e'(4, 1/4, 0.25) = 0.537
\]
\[
e(T_R) = 1/3, \quad e'(3, 1/3, 0.25) = 0.650
\]
\[
e'(T) = 4 \times 0.537 + 3 \times 0.650 = 4.098
\]

Therefore, do not split

---

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 \( \chi^2 \) 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
- Subtree raising
  - Replace subtree with most frequently used branch

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!
Examples of Post-pruning

Decision Tree:

Simplified Decision Tree:

Model Evaluation

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

- Holdout
  - Reserve k% for training and (100-k)% for testing
  - Random subsampling: repeated holdout

- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n
Cross-validation Example

- 3-fold cross-validation

<table>
<thead>
<tr>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>S₂</td>
<td>S₃</td>
</tr>
</tbody>
</table>

- Run 1: S₁ (Test Set), S₂, S₃ (Training Set)
- Run 2: S₂ (Test Set), S₁, S₃ (Training Set)
- Run 3: S₁, S₂, S₃ (Test Set)