Decision Trees from large Databases: SLIQ

- C4.5 often iterates over the training set
  - How often?
  - If the training set does not fit into main memory, swapping makes C4.5 unpractical!
- SLIQ:
  - Sort the values for every attribute
  - Build the tree “breadth-first“, not “depth-first“.
- Original reference:

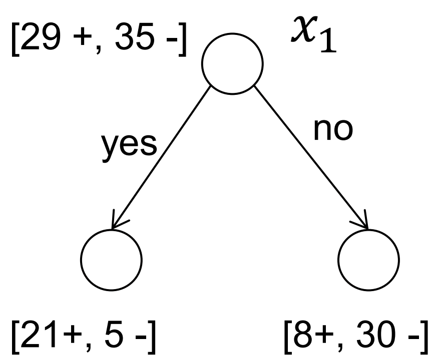
  M. Mehta et. al.: ”SLIQ: A Fast Scalable Classifier for Data Mining”. 1996

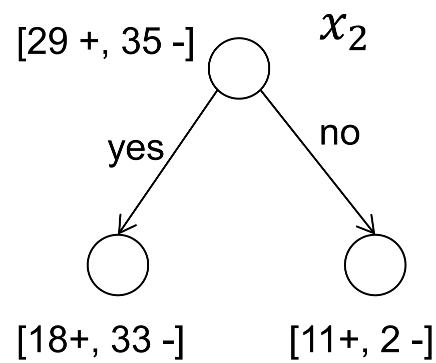
SLIQ: Gini Index

- To determine the best split, SLIQ uses the Gini-Index instead of Information Gain.
- For a training set L with n distinct classes:
  - $Gini(L) = 1 - \sum_{j=1\ldots n} p_j^2$
    - p_j is the relative frequency of value j
- After a binary split of the set L into sets L_1 and L_2 the index becomes:
  - $Gini_{split}(L) = \frac{|L_1|}{|L|} Gini(L_1) + \frac{|L_2|}{|L|} Gini(L_2)$
- Gini-Index behaves similarly to Information Gain.

Compare: Information Gain vs. Gini Index

- Which split is better?





- $H(L,y) = -\left(\frac{29}{64}\log_2\frac{29}{64} + \frac{35}{64}\log_2\frac{35}{64}\right) = 0.99$
- $IG(L,x_1) = 0.99 - \left(\frac{26}{64}H\left(L_{x_1=yes},y\right) + \frac{38}{64}H\left(L_{x_1=no},y\right)\right) \approx 0.26$
- $IG(L,x_2) = 0.99 - \left(\frac{51}{64}H\left(L_{x_2=yes},y\right) + \frac{13}{64}H\left(L_{x_2=no},y\right)\right) \approx 0.11$

$$\left(1 - \left(\left(\frac{8}{38}\right)^2 + \left(\frac{30}{38}\right)^2\right)\right) \approx 0.33$$

- $Gini_{x_1}(L) = \frac{26}{64}Gini\left(L_{x_1=yes}\right) + \frac{38}{64}Gini\left(L_{x_1=no}\right) \approx 0.32$
- $Gini_{x_2}(L) = \frac{51}{64}Gini\left(L_{x_2=yes}\right) + \frac{13}{64}Gini\left(L_{x_2=no}\right) \approx 0.42$

SLIQ – Algorithm

1. Start **Pre-sorting** of the samples.
2. As long as the stop criterion has not been reached
   1. For every attribute
      1. Place all nodes into a class histogram.
      2. Start **evaluation** of the splits.
   2. Choose a split.
   3. Update the decision tree; for each new node **update its class list** (nodes).

SLIQ (1st Substep)

- Pre-sorting of the samples:

1. For each attribute: create an *attribute list* with columns for the value, sample-ID and class.
2. Create a *class list* with columns for the sample-ID, class and leaf node.
3. Iterate over all training samples:
   - For each attribute
     - Insert its attribute values, sample-ID and class (sorted by attribute value) into the attribute list.
     - Insert the sample-ID, the class and the leaf node (sorted by sample-ID) into the class list.

SLIQ: Example

TRAINING DATA

Age	Salary	Class
30	65	G
23	15	B
40	75	G
55	40	B
55	100	G
45	60	G

AFTER PRE-SORTING

Age	Class List Index

Age List

Salary	Class List Index

Salary List

	Class	Leaf
1	G	
2		
3		
4		
5		
6		

Class List

SLIQ: Example

TRAINING DATA

Age	Salary	Class
30	65	G
23	15	B
40	75	G
55	40	B
55	100	G
45	60	G

AFTER PRE–SORTING

Age	Class List Index
23	2
30	1
40	3
45	6
55	5
55	4

Age List

Salary	Class List Index
15	2
40	4
60	6
65	1
75	3
100	5

Salary List

	Class	Leaf
1	G	N1
2	B	N1
3	G	N1
4	B	N1
5	G	N1
6	G	N1

Class List

SLIQ – Algorithm

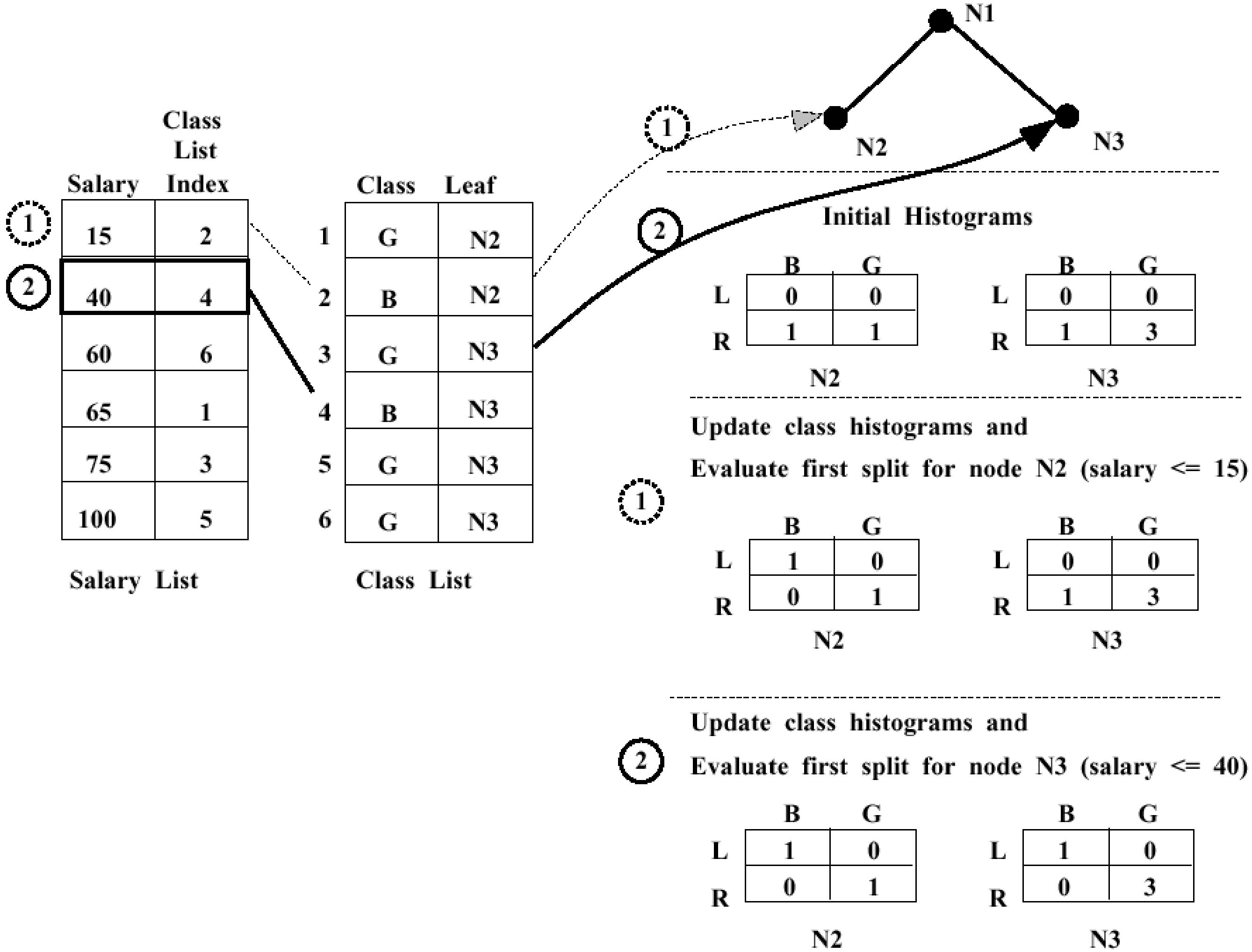
1. Start **Pre-sorting** of the samples. ✓
2. As long as the stop criterion has not been reached
   1. For every attribute
      1. Place all nodes into a class histogram.
      2. Start **evaluation** of the splits.
   2. Choose a split.
   3. Update the decision tree; for each new node **update its class list** (nodes).

SLIQ (2nd Substep)

- Evaluation of the splits.

1. For each node, and for all attributes
   1. Construct a histogram (for each class the histogram saves the count of samples before and after the split).
2. For each attribute A
   1. For each value v (traverse the attribute list for A)
      1. Find the entry in the class list (provides the class and node).
      2. Update the histogram for the node.
      3. Assess the split (if its a maximum, record it!)

SLIQ: Example



SLIQ – Algorithm

1. Start **Pre-sorting** of the samples. ✓
2. As long as the stop criterion has not been reached
   1. For every attribute
      1. Place all nodes into a class histogram.
      2. Start **evaluation** of the splits. ✓
   2. Choose a split.
   3. Update the decision tree; for each new node **update its class list** (nodes).

SLIQ (3rd Substep)

- Update the Class list (nodes).

1. Traverse the attribute list of the attribute used in the node.
2. For each entry (value, ID)
3. Find the matching entry (ID, class, node) in the class list.
4. Apply the split criterion emitting a new node.
5. Replace the corresponding class list entry with (ID, class, new node).

SLIQ: Example

Age	Class List Index
23	2
30	1
40	3
45	6
55	5
55	4

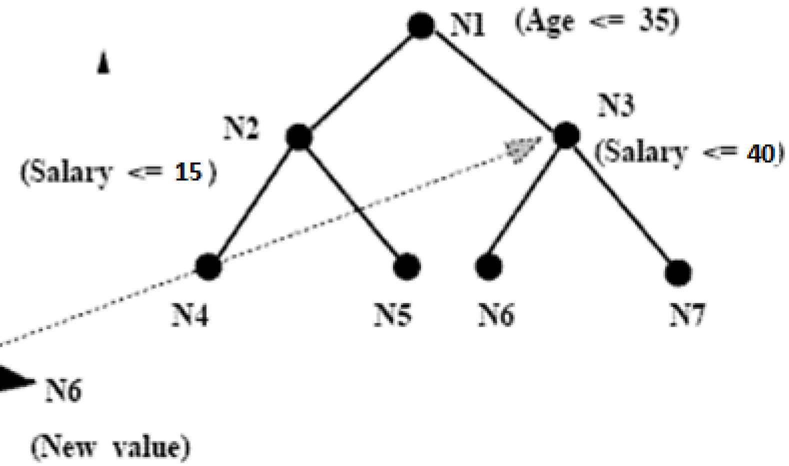
Age List

Salary	Class List Index
15	2
40	4
60	6
65	1
75	3
100	5

Salary List

	Class	Leaf
1	G	N2
2	B	N4
3	G	N3
4	B	N3
5	G	N3
6	G	N3

Class List



SLIQ: Data Structures

- Data structures in memory?
- Swappable data structures?
- Data structures in a database?

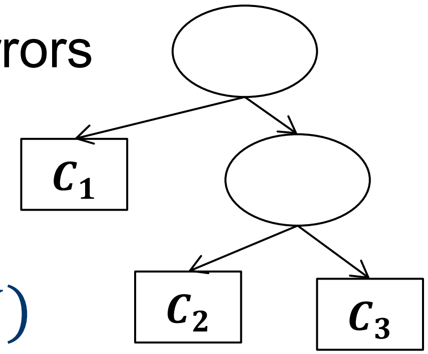
SLIQ- Pruning

- Minimum Description Length (MDL): the best model for a given data set minimizes the sum of the length the encoded data by the model plus the length of the model.
  - $cost(M, D) = cost(D|M) + cost(M)$
- $cost(M)$ = cost of the model (length).
  - How large is the decision tree?
- $cost(D|M)$ = cost to describe the data with the model.
  - How many classification errors are incurred?

Pruning – MDL Example I

- Assume: 16 binary attributes and 3 classes

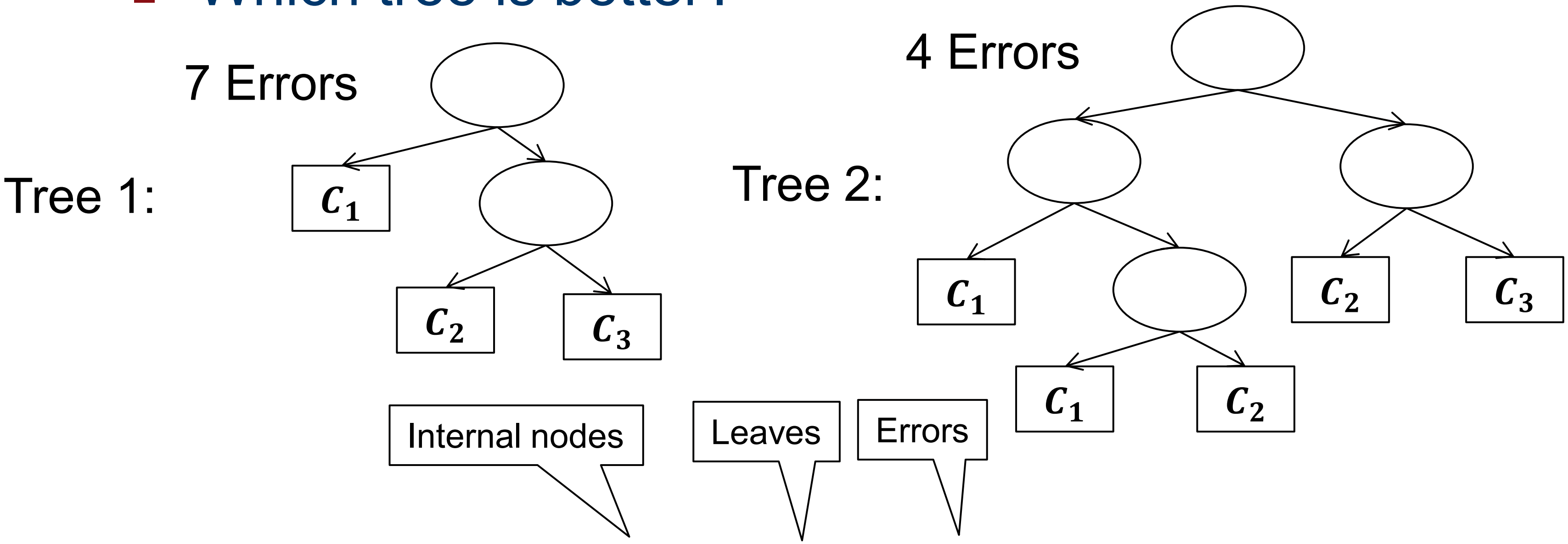
7 Errors



- $cost(M, D) = cost(D|M) + cost(M)$
  - Cost for the encoding of an internal node:
    - $\log_2(m) = \log_2(16) = 4$
  - Cost for the encoding of a leaf:
    - $\lceil \log_2(k) \rceil = \lceil \log_2(3) \rceil = 2$
  - Cost for the encoding of a classification error for n training data points:
    - $\log_2(n)$

Pruning – MDL Example II

- Which tree is better?



- Cost for tree 1: $2 * 4 + 3 * 2 + 7 * \log_2 n = 14 + 7 \log_2 n$
- Cost for tree 2: $4 * 4 + 5 * 2 + 4 * \log_2 n = 26 + 4 \log_2 n$
- If $n < 16$, then tree 1 is better.
- If $n > 16$, then tree 2 is better.

SLIQ – Properties

- Running time for the initialization (pre-sorting):
  - $O(n \log n)$ for each attribute
- Much of the data must not be kept in main memory.
- Good scalability.
  - If the class list does not fit in main memory then SLIQ no longer works.
  - Alternative: SPRINT
- Can handle numerical and discrete attributes.
- Parallelization of the process is possible.

Regression Trees

- ID3, C4.5, SLIQ: Solve classification problems.
  - Goal: low error rate + small tree
- Attribute can be continuous (except for ID3), but their prediction is discrete.
- Regression: the prediction is continuously valued.
  - Goal: low quadratic error rate + simple model.
  - $SSE = \sum_{j=1}^{n}\left(y_j - f\left(x_j\right)\right)^2$
- Methods we will now examine:
  - Regression trees,
  - Linear Regression,
  - Model trees.

Regression Trees

- Input: $L = \langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\rangle$, continuous y.
- Desired result: $f: X \rightarrow Y$
- Algorithm CART
  - It was developed simultaneously & independently from C4.5.
  - Terminal nodes incorporate continuous values.
  - Algorithm like C4.5. For classification, there are slightly different criteria (Gini instead of IG), but otherwise little difference.
  - Information Gain (& Gini) only work for classification.
  - Question: What is a split criterion for Regression?
- Original reference:

  L. Breiman et. al.: ”Classification and Regression Trees”. 1984

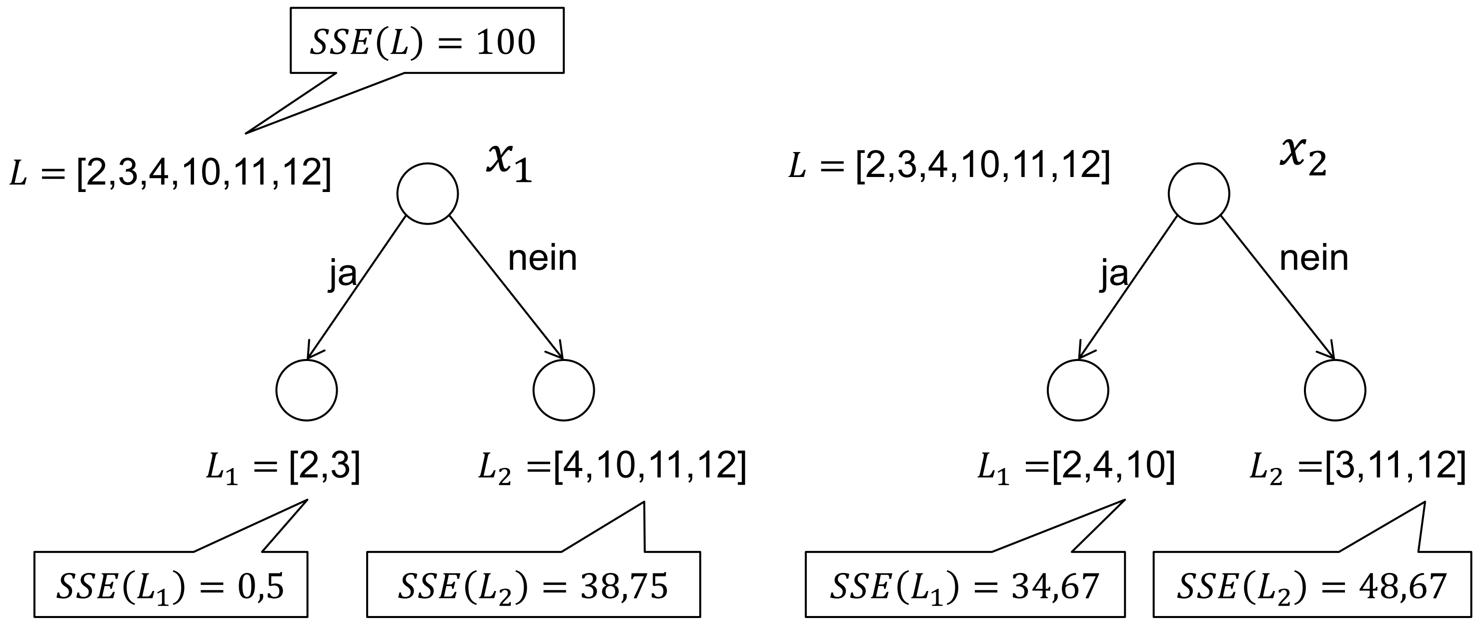
Regression Trees

- Goal: small quadratic error (SSE) + small tree.
- Examples L arrive at the current node.
- SSE at the current node: $SSE_L = \sum_{(\mathbf{x},y)\in L}(y-\bar{y})^2$ where $\bar{y} = \frac{1}{|L|}\sum_{(\mathbf{x},y)\in L} y$.
- With which test should the data be split?
- Criterion for test nodes $[x \leq v]$:
  - $SSE-Red(L,[x \leq v]) = SSE_L - SSE_{L[x\leq v]} - SSE_{L[x>v]}$
  - SSE-Reduction through the test.
- Stop Criterion:
  - Do not make a new split unless the SSE will be reduced by at least some minimum threshold.
  - If so, then create a terminal node with mean m of L.

CART- Example

$SSE_L = \sum_{(\mathbf{x},y)\in L}(y-\bar{y})^2$ where $\bar{y} = \frac{1}{|L|}\sum_{(\mathbf{x},y)\in L} y$

- Which split is better?



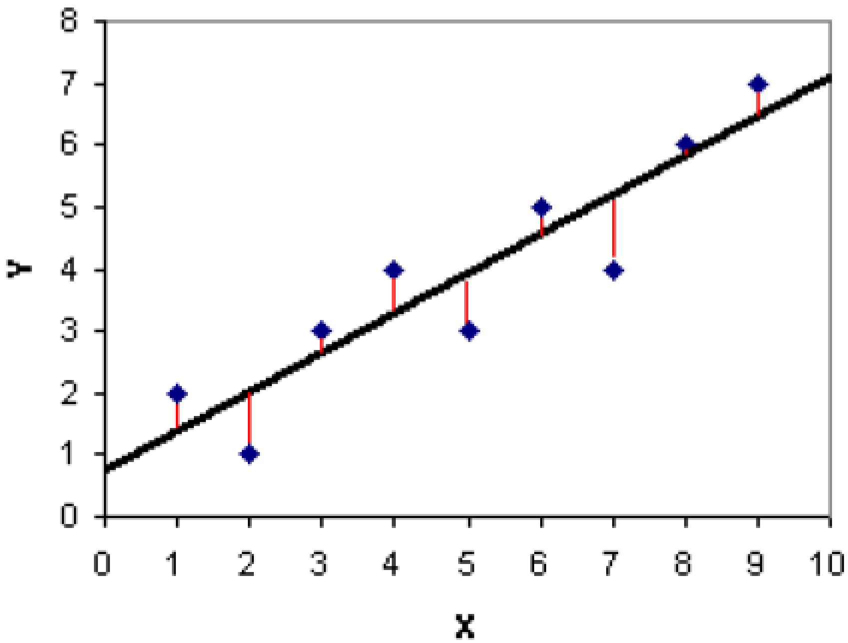
Linear Regression

- Regression trees: constant prediction at every leaf node.

- Linear Regression: Global model of a linear dependence between $\mathbf{x}$ and y.
- Standard method.
- Useful by itself and it serves as a building block for Model trees.

Linear Regression

- Input: $L = \langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\rangle$
- Desired result: a linear model, $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + c$



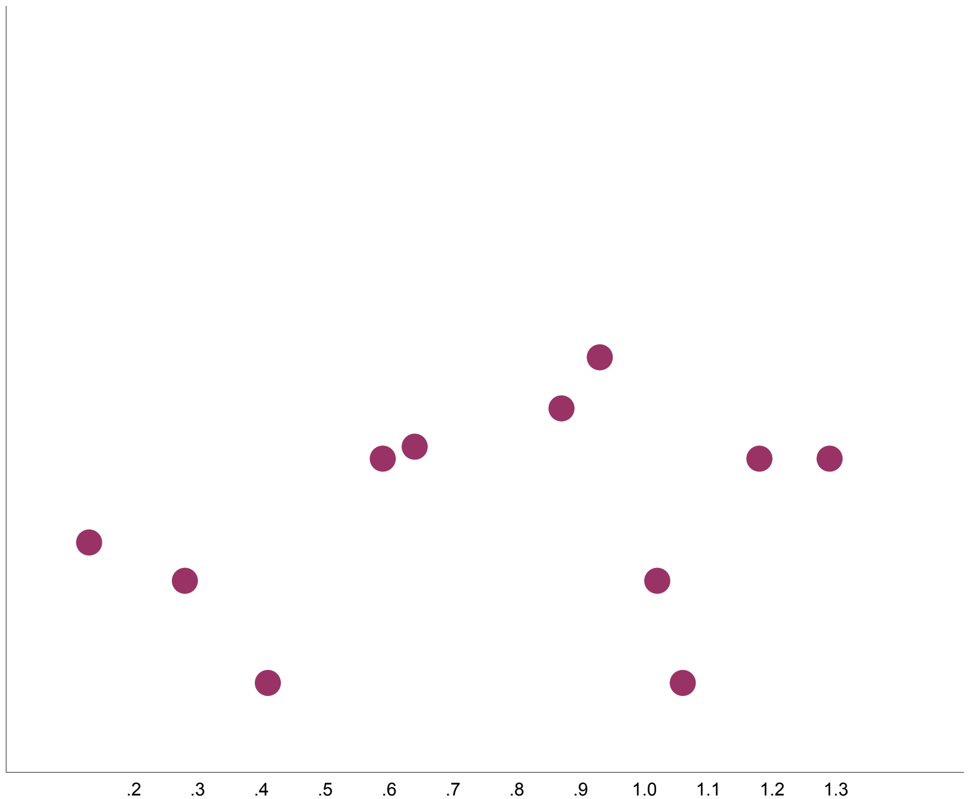
Normal vector

Residuals

Points along the Regression line are perpendicular to the normal vector.

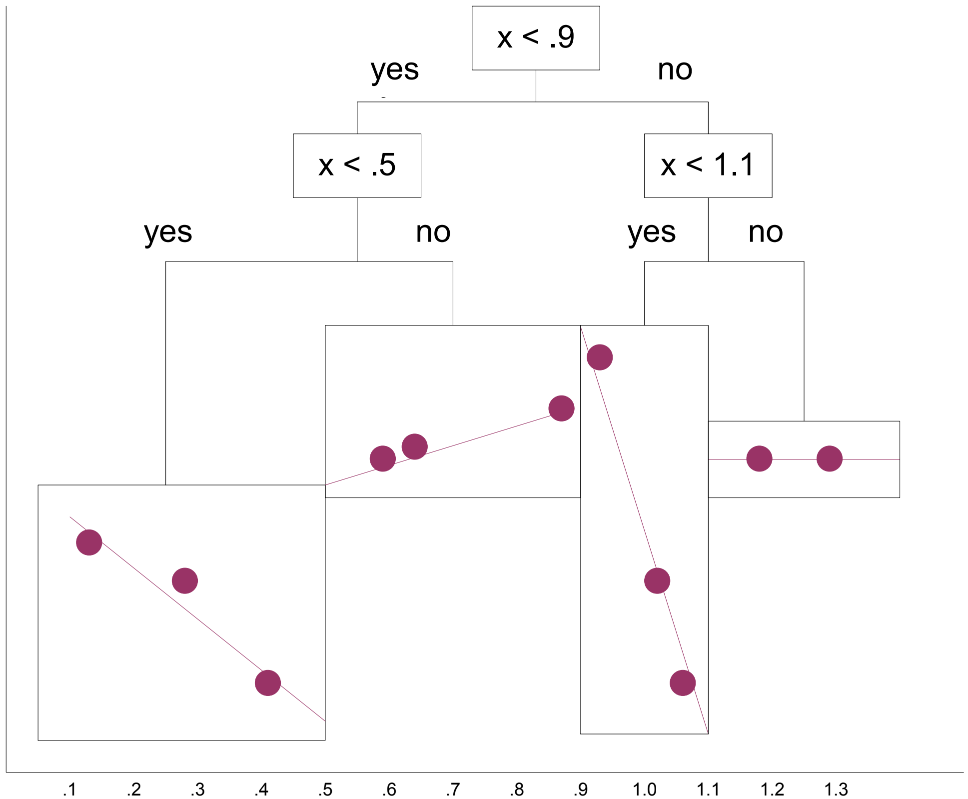
Model Trees

- Decision trees but with a linear regression model at each leaf node.



Model Trees

- Decision trees but with a linear regression model at each leaf node.



Model Trees: Creation of new Test Nodes

- Fit a linear regression of all samples that are filtered to the current node.
- Compute the regression‘s SSE.
- Iterate over all possible tests (like C4.5):
  - Fit a linear regression for the subset of samples on the left branch, & compute SSE_{left}.
  - Fit a linear regression for the subset of samples on the right branch, & compute SSE_{right}.
- Choose the test with the largest reduction $SSE - SSE_{left} - SSE_{right}$.
- Stop-Criterion: If the SSE is not reduced by at least a minimum threshold, don‘t create a new test.

Missing Attribute Values

- Problem: Training data with missing attributes.
- What if we test an attribute A for which an attribute value does not exist for every sample?
  - These training samples receive a substitute attribute value which occurs most frequently for A among the samples at node n.
  - These training samples receive a substitute attribute value, which most samples with the same classification have.

Attributes with Costs

- Example: Medical diagnosis
  - A blood test costs more than taking one's pulse
- Question: How can you learn a tree that is consistent with the training data at a low cost?
- Solution: Replace Information Gain with:
  - Tan and Schlimmer (1990): $\frac{IG^2(L,x)}{Cost(A)}$
  - Nunez (1988): $\frac{2^{IG(L,x)}-1}{(Cost(A)+1)^w}$
  - Parameter $w \in [0,1]$ controls the impact of the cost.

Bootstrapping

- Simple resampling method.
- Generates many variants of a training set L.
- Bootstrap Algorithm:
  - Repeat $k = 1 \ldots M$ times:
    - Generate a sample dataset L_k of size n from L:
      - The sampled dataset is drawn uniformly with replacement from the original set of samples.
    - Compute model θ_k for dataset L_k
  - Return the parameters generated by the bootstrap:
    - $\theta^* = (\theta_1, \ldots, \theta_M)$

Bootstrapping - Example

$$x = \begin{pmatrix} 3.12 \\ 0 \\ 1.57 \\ 19.67 \\ 0.22 \\ 2.20 \end{pmatrix}, \bar{x} = 4{,}46$$

$$x = \begin{pmatrix} 1.57 \\ 0.22 \\ 19.67 \\ 0 \\ 0.22 \\ 3.12 \end{pmatrix}, \bar{x} = 4{,}13$$

$$x = \begin{pmatrix} 0 \\ 2.20 \\ 19.67 \\ 2.20 \\ 2.20 \\ 1.57 \end{pmatrix}, \bar{x} = 4{,}64$$

$$x = \begin{pmatrix} 0.22 \\ 3.12 \\ 1.57 \\ 3.12 \\ 2.20 \\ 0.22 \end{pmatrix}, \bar{x} = 1{,}74$$

Bagging – Learning Robust Trees

- Bagging = **B**ootstrap **agg**regat**ing**
- Bagging – Algorithm:
  - Repeat $k = 1 \dots M$ times :
    - Generate a sample dataset L_k from L.
    - Compute model θ_k for dataset L_k.

e.g. a model tree

- Combine the M learned predictive models:
  - Classification problems: Voting
  - Regression problems: Mean value

- Original reference:
  L.Breiman: "Bagging Predictors". 1996

Random Forests

Bootstrap

- Repeat:
  - Draw randomly n samples, uniformly with replacement, from the training set.
  - Randomly select $m' < m$ features
  - Learn decision tree (without pruning)

- Classification: Maximum over all trees (Voting)
- Regression: Average over all trees

- Original reference:

  L. Breiman: "Random Forests". 2001

Usages of Decision Trees

- Medical Diagnosis.
- In face recognition.
- As part of more complex systems.

Decision Tree - Advantages

- Easy to interpret.
- Can be efficiently learned from many samples.
  - SLIQ, SPRINT
- Many Applications.
- High Accuracy.

Decision Tree - Disadvantages

- Not robust against noise
- Tendency to overfit
- Unstable

Summary of Decision Trees

- Classification: Prediction of discrete values.
  - Discrete attributes: ID3.
  - Continuous attributes: C4.5.
  - Scalable to large databases: SLIQ, SPRINT.
- Regression: Prediction of continuous values.
  - Regression trees: constant value predicted at each leaf.
  - Model trees: a linear model is contained in each leaf.