CS685: Data Mining
Decision Trees and Rule-Based Learners

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

Computer Science and Engineering,
Indian Institute of Technology, Kanpur
http://web.cse.iitk.ac.in/~cs685/

1st semester, 2018-19

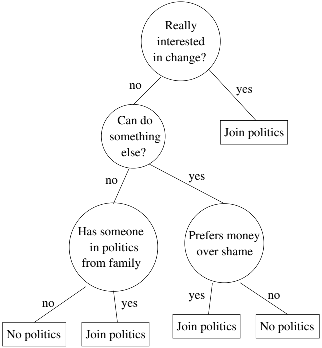
Mon, Thu 1030-1145 at RM101

Decision Trees

- A decision tree is a tree structure used for classification
- Each *internal node* represents a test on an attribute
- Each *branch* represents an outcome of the test
- Each *leaf* represents a class outcome
- For a test object, its attributes are tested and a particular path is followed to a leaf, which is deemed its class

Decision Trees

- A decision tree is a tree structure used for classification
- Each *internal node* represents a test on an attribute
- Each *branch* represents an outcome of the test
- Each *leaf* represents a class outcome
- For a test object, its attributes are tested and a particular path is followed to a leaf, which is deemed its class



Constructing a Decision Tree

- If all objects are in same class, label the leaf node with that class
  - The leaf is then pure

Constructing a Decision Tree

- If all objects are in same class, label the leaf node with that class
  - The leaf is then pure
- Else, choose the "best" attribute to split
  - Determine splitting criterion based on splitting attribute
  - Indicates split point(s) or splitting subset(s)
  - Different measures of impurity to split a node
- Separate objects into different branches according to split
- Recursively, build tree for each split

Constructing a Decision Tree

- If all objects are in same class, label the leaf node with that class
  - The leaf is then pure
- Else, choose the "best" attribute to split
  - Determine splitting criterion based on splitting attribute
  - Indicates split point(s) or splitting subset(s)
  - Different measures of impurity to split a node
- Separate objects into different branches according to split
- Recursively, build tree for each split
- Stop when either
  - Leaf becomes pure
  - No more attributes to split – assign class through majority voting

Constructing a Decision Tree

- If all objects are in same class, label the leaf node with that class
  - The leaf is then pure
- Else, choose the "best" attribute to split
  - Determine splitting criterion based on splitting attribute
  - Indicates split point(s) or splitting subset(s)
  - Different measures of impurity to split a node
- Separate objects into different branches according to split
- Recursively, build tree for each split
- Stop when either
  - Leaf becomes pure
  - No more attributes to split – assign class through majority voting
- Decision tree building is top-down and backtracking is *not* allowed

Information Gain

- Entropy impurity or information impurity

$$info(D) = -\sum_{i=1}^{k}(p_i \log_2 p_i)$$

- For n partitions into $D_1, \ldots, D_n$, denoted by S

$$info_S(D) = \sum_{j=1}^{n}(|D_j|/|D|)info(D_j)$$

- Information gain is

$$gain_S(D) = info(D) - info_S(D)$$

- More the gain, better the split
- Choose attribute and split point that maximizes gain

Gini Index

- Variance impurity for two classes

$$var(D) = p_1.p_2$$

- For k classes, generalized to Gini index or Gini impurity

$$gini(D) = \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} p_i.p_j = 1 - \sum_{i=1}^{k} p_i^2$$

- For n partitions into $D_1, \dots, D_n$, denoted by S

$$gini_S(D) = \sum_{j=1}^{n} (|D_j|/|D|) gini(D_j)$$

- Less the gini index, better the split
- Choose attribute and split point that minimizes gini index

Classification Error

- Classification error or misclassification index

$$class(D) = 1 - \max_i p_i$$

- This is the *probability of misclassification* when no more split is done and majority voting is used
- Find reduction in impurity by splitting

$$class(D) - class_S(D) = class(D) - \sum_{j=1}^{n} (|D_j|/|D|) class(D_j)$$

- More the reduction in impurity, better the split
- Choose attribute and split point that maximizes reduction

Gain Ratio

- Most impurity measures are biased towards multiway splits
- Higher chance that a node becomes purer
- Gain ratio counters it
- For n partitions into $D_1, \dots, D_n$, denoted by S
- Split information is defined as

$$splitinfo_S(D) = -\sum_{j=1}^{n} (|D_j|/|D|) \log_2(|D_j|/|D|)$$

- Similar to information measure, although just uses the number of objects in each partition and *not* any class information
- This is used to normalize information gain

$$gainratio_S(D) = gain_S(D)/splitinfo_S(D)$$

- Higher the gain ratio, better the split
- Choose attribute and split point that maximizes gain ratio

Choosing a Split Point

- If attribute is nominal

Choosing a Split Point

- If attribute is nominal
  - Each category denotes a new branch
  - If binary split is required,

Choosing a Split Point

- If attribute is nominal
  - Each category denotes a new branch
  - If binary split is required, use set membership testing
- If attribute is ordinal

Choosing a Split Point

- If attribute is nominal
  - Each category denotes a new branch
  - If binary split is required, use set membership testing
- If attribute is ordinal
  - Each category denotes a new branch
  - If binary split is required,

Choosing a Split Point

- If attribute is nominal
  - Each category denotes a new branch
  - If binary split is required, use set membership testing
- If attribute is ordinal
  - Each category denotes a new branch
  - If binary split is required, use order information
- If attribute is numeric

Choosing a Split Point

- If attribute is nominal
  - Each category denotes a new branch
  - If binary split is required, use set membership testing
- If attribute is ordinal
  - Each category denotes a new branch
  - If binary split is required, use order information
- If attribute is numeric
  - Sort all values and choose a (binary) split point
  - If multiway split is required,

Choosing a Split Point

- If attribute is nominal
  - Each category denotes a new branch
  - If binary split is required, use set membership testing
- If attribute is ordinal
  - Each category denotes a new branch
  - If binary split is required, use order information
- If attribute is numeric
  - Sort all values and choose a (binary) split point
  - If multiway split is required, choose multiple split points

Discussion

- Over-fitting can happen
- Tree needs to be *pruned*

Discussion

- Over-fitting can happen
- Tree needs to be *pruned*
- Can use criteria such as chi-square test to stop splitting
- Can use criteria such as information gain to merge

Discussion

- Over-fitting can happen
- Tree needs to be *pruned*
- Can use criteria such as chi-square test to stop splitting
- Can use criteria such as information gain to merge
- Under-fitting can also happen
- Some thresholds are always needed to control these

Discussion

- Over-fitting can happen
- Tree needs to be *pruned*
- Can use criteria such as chi-square test to stop splitting
- Can use criteria such as information gain to merge
- Under-fitting can also happen
- Some thresholds are always needed to control these
- Missing attributes

Discussion

- Over-fitting can happen
- Tree needs to be *pruned*
- Can use criteria such as chi-square test to stop splitting
- Can use criteria such as information gain to merge
- Under-fitting can also happen
- Some thresholds are always needed to control these
- Missing attributes
    - Ignored while training
    - Most frequent class assumed when testing

Discussion

- Over-fitting can happen
- Tree needs to be *pruned*
- Can use criteria such as chi-square test to stop splitting
- Can use criteria such as information gain to merge
- Under-fitting can also happen
- Some thresholds are always needed to control these
- Missing attributes
  - Ignored while training
  - Most frequent class assumed when testing
- Node decisions are based on single attribute – monothetic trees

Discussion

- Over-fitting can happen
- Tree needs to be *pruned*
- Can use criteria such as chi-square test to stop splitting
- Can use criteria such as information gain to merge
- Under-fitting can also happen
- Some thresholds are always needed to control these
- Missing attributes
    - Ignored while training
    - Most frequent class assumed when testing
- Node decisions are based on single attribute – monothetic trees
- Why not polythetic trees where decisions are based on multiple attributes?

Discussion

- Over-fitting can happen
- Tree needs to be *pruned*
- Can use criteria such as chi-square test to stop splitting
- Can use criteria such as information gain to merge
- Under-fitting can also happen
- Some thresholds are always needed to control these
- Missing attributes
  - Ignored while training
  - Most frequent class assumed when testing
- Node decisions are based on single attribute – monothetic trees
- Why not polythetic trees where decisions are based on multiple attributes?
  - Theoretically possible but practically too complex

Variants of Decision Trees

- Three main variants
- ID3 (from Iterative Dichotomiser generation 3)
  - Multiway split
  - Uses information gain

Variants of Decision Trees

- Three main variants
- ID3 (from Iterative Dichotomiser generation 3)
  - Multiway split
  - Uses information gain
- C4.5
  - Evolved from ID3
  - Multiway split
  - Uses gain ratio

Variants of Decision Trees

- Three main variants
- ID3 (from Iterative Dichotomiser generation 3)
  - Multiway split
  - Uses information gain
- C4.5
  - Evolved from ID3
  - Multiway split
  - Uses gain ratio
- CART (from Classification and Regression Trees)
  - Binary split
  - Uses gini index

Rules

- Rules are of the form

  `if` condition `then` class

- *condition* is a conjunct (i.e., logical AND) of tests on *single* attributes
- If the condition holds, then the object is said to be from *class*
- *condition* is called antecedent or precondition
- *class* is called consequent
- Example: `if` in family = yes AND can do something = no `then` politics

Rules

- Rules are of the form

  `if` condition `then` class

- *condition* is a conjunct (i.e., logical AND) of tests on *single* attributes
- If the condition holds, then the object is said to be from *class*
- *condition* is called antecedent or precondition
- *class* is called consequent
- Example: `if` in family = yes AND can do something = no `then` politics
- Two important parameters of a rule
  - Coverage: Number of objects the rule applies to

    $$coverage = |covers|/|D|$$

  - Accuracy: Number of correctly classified objects when rule is applied

    $$accuracy = |correct|/|covers|$$

Triggering and Firing of Rules

- For every tuple, a rule that satisfies it is "triggered"
- If for that tuple, it is the *only* rule, then it is "fired"

Triggering and Firing of Rules

- For every tuple, a rule that satisfies it is "triggered"
- If for that tuple, it is the *only* rule, then it is "fired"
- Otherwise, a conflict resolution strategy is devised

Triggering and Firing of Rules

- For every tuple, a rule that satisfies it is "triggered"
- If for that tuple, it is the *only* rule, then it is "fired"
- Otherwise, a conflict resolution strategy is devised
- *Size-based ordering*: Rule with larger antecedent is invoked
    - More stringent, i.e., tougher

Triggering and Firing of Rules

- For every tuple, a rule that satisfies it is "triggered"
- If for that tuple, it is the *only* rule, then it is "fired"
- Otherwise, a conflict resolution strategy is devised
- *Size-based ordering*: Rule with larger antecedent is invoked
  - More stringent, i.e., tougher
- *Class-based ordering*: Two schemes
  - Consequent class is more frequent, i.e., according to order of prevalence
  - Consequent class has less misclassification
  - Within same class, there is arbitrary ordering

Triggering and Firing of Rules

- For every tuple, a rule that satisfies it is "triggered"
- If for that tuple, it is the *only* rule, then it is "fired"
- Otherwise, a conflict resolution strategy is devised
- *Size-based ordering*: Rule with larger antecedent is invoked
  - More stringent, i.e., tougher
- *Class-based ordering*: Two schemes
  - Consequent class is more frequent, i.e., according to order of prevalence
  - Consequent class has less misclassification
  - Within same class, there is arbitrary ordering
- *Rule-based ordering*: Priority list according to some function based on coverage, accuracy and size

Triggering and Firing of Rules

- For every tuple, a rule that satisfies it is “triggered”
- If for that tuple, it is the *only* rule, then it is “fired”
- Otherwise, a conflict resolution strategy is devised
- *Size-based ordering*: Rule with larger antecedent is invoked
  - More stringent, i.e., tougher
- *Class-based ordering*: Two schemes
  - Consequent class is more frequent, i.e., according to order of prevalence
  - Consequent class has less misclassification
  - Within same class, there is arbitrary ordering
- *Rule-based ordering*: Priority list according to some function based on coverage, accuracy and size
- For a query tuple, the rule that satisfies it is invoked
- If no such rule, then a *default* rule is invoked: `if () then` class i
  - Class i is the most abundant class

Learning Rules from a Decision Tree

Learning Rules from a Decision Tree

- Every path is a rule
- As verbose or complex as the decision tree itself

Learning Rules from a Decision Tree

- Every path is a rule
- As verbose or complex as the decision tree itself
- Rules are *mutually exclusive* and *exhaustive*

Learning Rules from a Decision Tree

- Every path is a rule
- As verbose or complex as the decision tree itself
- Rules are *mutually exclusive* and *exhaustive*
- No need to order the rules

Sequential Covering Algorithm

- Sequential covering algorithm learns rules sequentially
- Rules are learnt per class one-by-one

Sequential Covering Algorithm

- Sequential covering algorithm learns rules sequentially
- Rules are learnt per class one-by-one
- When a rule is learnt, all tuples covered by it are removed
- Given a set of tuples, how is a rule learnt?

Sequential Covering Algorithm

- Sequential covering algorithm learns rules sequentially
- Rules are learnt per class one-by-one
- When a rule is learnt, all tuples covered by it are removed
- Given a set of tuples, how is a rule learnt?
- Greedy learn-one-rule method learns the “best” rule given the current set of tuples
- *General-to-specific* strategy

Sequential Covering Algorithm

- Sequential covering algorithm learns rules sequentially
- Rules are learnt per class one-by-one
- When a rule is learnt, all tuples covered by it are removed
- Given a set of tuples, how is a rule learnt?
- Greedy learn-one-rule method learns the "best" rule given the current set of tuples
- *General-to-specific* strategy
- Starts with an empty antecedent
- At each stage, every attribute (and every possible split) is considered
- If the new rule has better quality than the old rule, it is retained
- Decisions are greedy and are never backtracked

Sequential Covering Algorithm

- Sequential covering algorithm learns rules sequentially
- Rules are learnt per class one-by-one
- When a rule is learnt, all tuples covered by it are removed
- Given a set of tuples, how is a rule learnt?
- Greedy learn-one-rule method learns the "best" rule given the current set of tuples
- *General-to-specific* strategy
- Starts with an empty antecedent
- At each stage, every attribute (and every possible split) is considered
- If the new rule has better quality than the old rule, it is retained
- Decisions are greedy and are never backtracked
- Otherwise, the old rule is accepted

Sequential Covering Algorithm

- Sequential covering algorithm learns rules sequentially
- Rules are learnt per class one-by-one
- When a rule is learnt, all tuples covered by it are removed
- Given a set of tuples, how is a rule learnt?
- Greedy learn-one-rule method learns the “best” rule given the current set of tuples
- *General-to-specific* strategy
- Starts with an empty antecedent
- At each stage, every attribute (and every possible split) is considered
- If the new rule has better quality than the old rule, it is retained
- Decisions are greedy and are never backtracked
- Otherwise, the old rule is accepted
- The next rule is then learnt

Sequential Covering Algorithm

- Sequential covering algorithm learns rules sequentially
- Rules are learnt per class one-by-one
- When a rule is learnt, all tuples covered by it are removed
- Given a set of tuples, how is a rule learnt?
- Greedy learn-one-rule method learns the "best" rule given the current set of tuples
- *General-to-specific* strategy
- Starts with an empty antecedent
- At each stage, every attribute (and every possible split) is considered
- If the new rule has better quality than the old rule, it is retained
- Decisions are greedy and are never backtracked
- Otherwise, the old rule is accepted
- The next rule is then learnt
- Rules are ordered according to their order of inception

Sequential Covering Algorithm

- Sequential covering algorithm learns rules sequentially
- Rules are learnt per class one-by-one
- When a rule is learnt, all tuples covered by it are removed
- Given a set of tuples, how is a rule learnt?
- Greedy learn-one-rule method learns the "best" rule given the current set of tuples
- *General-to-specific* strategy
- Starts with an empty antecedent
- At each stage, every attribute (and every possible split) is considered
- If the new rule has better quality than the old rule, it is retained
- Decisions are greedy and are never backtracked
- Otherwise, the old rule is accepted
- The next rule is then learnt
- Rules are ordered according to their order of inception
- Variants are AQ, CN2 and RIPPER

Rule Quality

- Accuracy is the most vital concern

Rule Quality

- Accuracy is the most vital concern
  - A rule with 90% accuracy and 80% coverage may not be better than another rule with 95% accuracy and 10% coverage

Rule Quality

- Accuracy is the most vital concern
  - A rule with 90% accuracy and 80% coverage may not be better than another rule with 95% accuracy and 10% coverage
- Coverage also needs to be considered though

Rule Quality

- Accuracy is the most vital concern
  - A rule with 90% accuracy and 80% coverage may not be better than another rule with 95% accuracy and 10% coverage
- Coverage also needs to be considered though
- Old rule R_1 has a_1 as antecedent
- New rule R_2 has a_2 as antecedent
- Let the number of tuples covered by a rule be denoted by D_i
- For the particular class in question, p_i is the number of tuples correctly classified, i.e., the consequent is this class
- Correspondingly, n_i is the number of negative tuples

$$D_i = p_i + n_i$$

Rule Quality

- Accuracy is the most vital concern
  - A rule with 90% accuracy and 80% coverage may not be better than another rule with 95% accuracy and 10% coverage
- Coverage also needs to be considered though
- Old rule R_1 has a_1 as antecedent
- New rule R_2 has a_2 as antecedent
- Let the number of tuples covered by a rule be denoted by D_i
- For the particular class in question, p_i is the number of tuples correctly classified, i.e., the consequent is this class
- Correspondingly, n_i is the number of negative tuples

$$D_i = p_i + n_i$$

- Four rule quality measures

FOIL Gain

- FOIL_Gain measure proposed as part of the sequential covering algorithm First Order Inductive Learner (FOIL) used in RIPPER

$$FOIL\_Gain(R_1 \rightarrow R_2) = p_2 \times \left(\log_2 \frac{p_2}{D_2} - \log_2 \frac{p_1}{D_1} \right)$$

- Considers both coverage and accuracy

Likelihood Ratio

- Statistical test using the likelihood ratio statistic

$$LR = 2 \sum_{i=1}^{m} f_i \log \frac{f_i}{e_i}$$

  where m is the number of classes, f_i and e_i are the observed and expected frequencies of tuples in each class
- LR statistic has a chi-square distribution with $m - 1$ degrees of freedom
- The larger the statistic, the more deviated it is from the random rule, and thus, the better

Entropy

- Entropy: Rule with less entropy is better

M-Estimate

- M-estimate measure considers the number of classes as well

$$m\text{-}estimate = \frac{p_i + m.c_i}{D_i + m}$$

  where m is the number of classes and c_i is the prior probability of class C_i
- If the prior probabilities are not known, replacing it by $1/m$ yields the Laplacian estimate

$$Laplacian = \frac{p_i + 1}{D_i + m}$$

- The larger the estimate, the better is the rule

Rule Pruning

- General-to-specific strategy is susceptible to overfitting
- *Specific-to-general* strategy first learns the most specific rule and then prunes the antecedent
- This is rule pruning

Rule Pruning

- General-to-specific strategy is susceptible to overfitting
- *Specific-to-general* strategy first learns the most specific rule and then prunes the antecedent
- This is rule pruning
- Each training instance starts as a rule
- From a rule R_1, an antecedent is removed to yield rule R_2
- Measure of rule quality is FOIL_Prune

$$FOIL\_Prune = \frac{p_i - n_i}{D_i}$$

- If this measure is higher for R_2, then pruning is applied

Frequent Pattern-Based Classifier

- Uses the idea of frequent patterns from association rule mining
- Assume a frequent pattern $\{A, B, C\}$
- Suppose C is a class label
- Suppose confidence of the rule $\{A, B\} \implies C$ is high
- Then presence of $\{A, B\}$ is a good rule for classifying into class C

Frequent Pattern-Based Classifier

- Uses the idea of frequent patterns from association rule mining
- Assume a frequent pattern $\{A, B, C\}$
- Suppose C is a class label
- Suppose confidence of the rule $\{A, B\} \implies C$ is high
- Then presence of $\{A, B\}$ is a good rule for classifying into class C
- This frequent pattern-based mining can be done for each class

Discussion

- Rules can be very verbose

Discussion

- Rules can be very verbose
- Simple rules can only learn rectilinear boundaries

Discussion

- Rules can be very verbose
- Simple rules can only learn rectilinear boundaries
- Rules have an interpretation and can lead to descriptive models

Discussion

- Rules can be very verbose
- Simple rules can only learn rectilinear boundaries
- Rules have an interpretation and can lead to descriptive models
- Can handle imbalance in class distribution very well