18CSE751 – Introduction to Machine Learning
Lecture 20-25: RULE EXTRACTION FROM TREES, LEARNING RULES FROM DATA, SUPPORT VECTOR MACHINE IN DETAIL
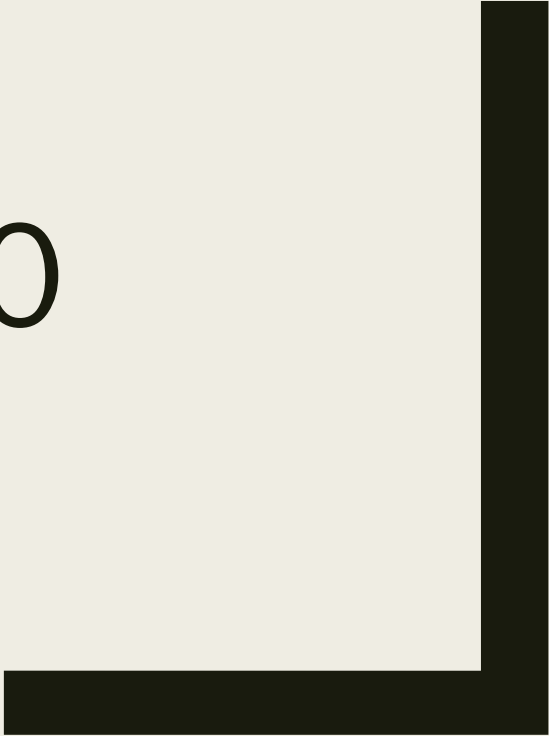
Dr.Vani Vasudevan

Professor –CSE, NMIT

LECTURE 20

Dr.Vani V

Rule-Based Classifier

■ Classify records by using a collection of "if...then..." rules

■ Rule: (*Condition*) → *y*
  – *where*
    ■ *Condition* is a conjunctions of attributes
    ■ *y* is the class label
  – *LHS: rule antecedent or condition*
  – *RHS: rule consequent*
  – *Examples of classification rules:*
    ■ (Blood Type=Warm) ∧ (Lay Eggs=Yes) → Birds
    ■ (Taxable Income < 50K) ∧ (Refund=Yes) → Evade=No

Rule-based Classifier (Example)

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
human	warm	yes	no	no	mammals
python	cold	no	no	no	reptiles
salmon	cold	no	no	yes	fishes
whale	warm	yes	no	yes	mammals
frog	cold	no	no	sometimes	amphibians
komodo	cold	no	no	no	reptiles
bat	warm	yes	yes	no	mammals
pigeon	warm	no	yes	no	birds
cat	warm	yes	no	no	mammals
leopard shark	cold	yes	no	yes	fishes
turtle	cold	no	no	sometimes	reptiles
penguin	warm	no	no	sometimes	birds
porcupine	warm	yes	no	no	mammals
eel	cold	no	no	yes	fishes
salamander	cold	no	no	sometimes	amphibians
gila monster	cold	no	no	no	reptiles
platypus	warm	no	no	no	mammals
owl	warm	no	yes	no	birds
dolphin	warm	yes	no	yes	mammals
eagle	warm	no	yes	no	birds

R1: (Give Birth = no) $\land$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\land$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\land$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\land$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

Application of Rule-Based Classifier

■ A rule *r* covers an instance x if the attributes of the instance satisfy the condition of the rule

R1: (Give Birth = no) ∧ (Can Fly = yes) → Birds

R2: (Give Birth = no) ∧ (Live in Water = yes) → Fishes

R3: (Give Birth = yes) ∧ (Blood Type = warm) → Mammals

R4: (Give Birth = no) ∧ (Can Fly = no) → Reptiles

R5: (Live in Water = sometimes) → Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
hawk	warm	no	yes	no	?
grizzly bear	warm	yes	no	no	?

The rule R1 covers a hawk => Bird

The rule R3 covers the grizzly bear => Mammal

Rule Coverage and Accuracy

- Coverage of a rule:
  - *Fraction of records that satisfy the antecedent of a rule*

- Accuracy of a rule:
  - *Fraction of records that satisfy both the antecedent and consequent of a rule*

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

(Status=Single) $\rightarrow$ No

Coverage = 40%, Accuracy = 50%

How does Rule-based Classifier Work?

R1: (Give Birth = no) ∧ (Can Fly = yes) → Birds

R2: (Give Birth = no) ∧ (Live in Water = yes) → Fishes

R3: (Give Birth = yes) ∧ (Blood Type = warm) → Mammals

R4: (Give Birth = no) ∧ (Can Fly = no) → Reptiles

R5: (Live in Water = sometimes) → Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
lemur	warm	yes	no	no	?
turtle	cold	no	no	sometimes	?
dogfish shark	cold	yes	no	yes	?

A lemur triggers rule R3, so it is classified as a mammal

A turtle triggers both R4 and R5

A dogfish shark triggers none of the rules

Characteristics of Rule-Based Classifier

■ Mutually exclusive rules

   – *Classifier contains mutually exclusive rules if the rules are independent of each other*

   – *Every record is covered by at most one rule*

■ Exhaustive rules

   – *Classifier has exhaustive coverage if it accounts for every possible combination of attribute values*

   – *Each record is covered by at least one rule*

From Decision Trees To Rules
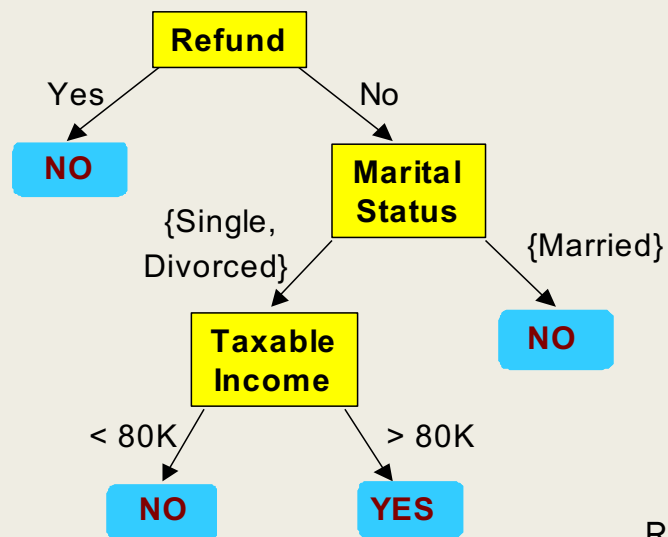
Classification Rules

(Refund=Yes) ==> No

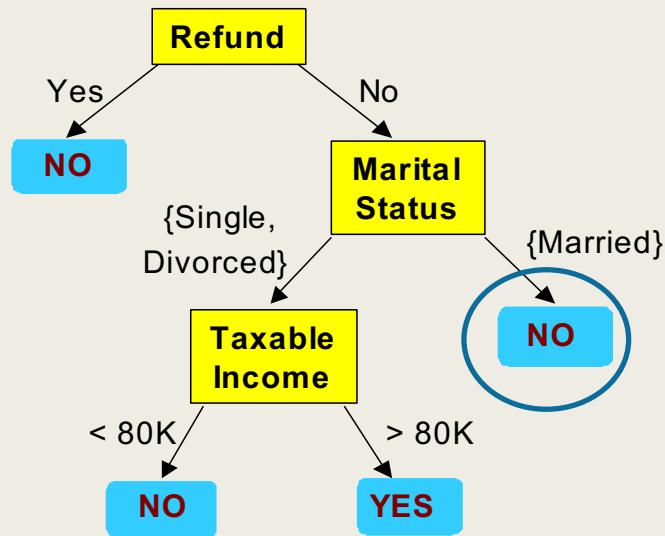(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

(Refund=No, Marital Status={Married}) ==> No

Rules are mutually exclusive and exhaustive

Rule set contains as much information as the tree

Rules Can Be Simplified

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	**Married**	100K	No
3	No	Single	70K	No
4	Yes	**Married**	120K	No
5	No	Divorced	95K	Yes
6	No	**Married**	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	**Married**	75K	No
10	No	Single	90K	Yes

Initial Rule: (Refund=No) $\wedge$ (Status=Married) $\rightarrow$ No

Simplified Rule: (Status=Married) $\rightarrow$ No

Effect of Rule Simplification

- Rules are no longer mutually exclusive
  - *A record may trigger more than one rule*
  - *Solution?*
    - Ordered rule set
    - Unordered rule set – use voting schemes

- Rules are no longer exhaustive
  - *A record may not trigger any rules*
  - *Solution?*
    - Use a default class

Rule Ordering Schemes

- **Rule-based ordering**
  - *Individual rules are ranked based on their quality*
- **Class-based ordering**
  - *Rules that belong to the same class appear together*

Rule-based Ordering
(Refund=Yes) ==> No
(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No
(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes
(Refund=No, Marital Status={Married}) ==> No

Class-based Ordering
(Refund=Yes) ==> No
(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No
(Refund=No, Marital Status={Married}) ==> No
(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

Building Classification Rules
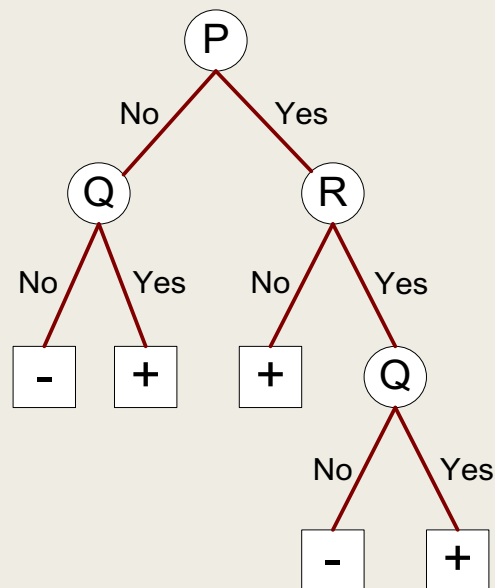
- Direct Method:
    - Extract rules directly from data
    - e.g.: RIPPER, CN2, Holte's 1R

- Indirect Method:
    - Extract rules from other classification models (e.g. decision trees, neural networks, etc).
    - e.g: C4.5 rules

Indirect Methods

Rule Set

r1: (P=No,Q=No) ==> -
r2: (P=No,Q=Yes) ==> +
r3: (P=Yes,R=No) ==> +
r4: (P=Yes,R=Yes,Q=No) ==> -
r5: (P=Yes,R=Yes,Q=Yes) ==> +

Indirect Method: C4.5rules

- Extract rules from an unpruned decision tree

- For each rule, r: A $\rightarrow$ y,
    - *consider an alternative rule r': A' $\rightarrow$ y where A' is obtained by removing one of the conjuncts in A*
    - *Compare the pessimistic error rate for r against all r's*
    - *Prune if one of the r's has lower pessimistic error rate*
    - *Repeat until we can no longer improve generalization error*

Indirect Method: C4.5rules

- Instead of ordering the rules, order subsets of rules (class ordering)
  - *Each subset is a collection of rules with the same rule consequent (class)*
  - *Compute description length of each subset*
    - Description length = L(error) + g L(model)
    - g is a parameter that considers the presence of redundant attributes in a rule set (default value = 0.5)

Sequential Covering Algorithm...

1. Used to extract rules directly from data.

2. Rules are grown in a greedy fashion based on a certain evaluation measure.

3. The algorithm extracts the rules one class at a time for data sets that contain more than two classes.

   - *For the vertebrate classification problem,*

   - *the sequential covering algorithm may generate rules for classifying birds first, followed by rules for classifying mammals, amphibians, reptiles, and finally, fishes*

Class-Based Ordering

(Skin Cover=feathers, Aerial Creature=yes)
    ==> Birds

(Body temperature=warm-blooded,
Gives Birth=no) ==> Birds

(Body temperature=warm-blooded,
Gives Birth=yes) ==> Mammals

(Aquatic Creature=semi)) ==> Amphibians

(Skin Cover=none) ==> Amphibians

(Skin Cover=scales, Aquatic Creature=no)
    ==> Reptiles

(Skin Cover=scales, Aquatic Creature=yes)
    ==> Fishes

Direct Method: Sequential Covering...

1. Start from an empty rule

2. Grow a rule using the Learn-One-Rule function

3. Remove training records covered by the rule

4. Repeat Step (2) and (3) until stopping criterion is met

Sequential Covering Algorithm...

The criterion for deciding which class should be generated

1. Depends on a number of factors, such as the **class prevalence** (i.e., fraction of training records that belong to a particular class) or

2. The **cost of misclassifying records** from a given class.

Algorithm 5.1 Sequential covering algorithm.

1: Let E be the training records and A be the set of attribute-value pairs, $\{(A_j, v_j)\}$.
2: Let Y_o be an ordered set of classes $\{y_1, y_2, \ldots, y_k\}$.
3: Let $R = \{\ \}$ be the initial rule list.
4: **for** each class $y \in Y_o - \{y_k\}$ **do**
5:    **while** stopping condition is not met **do**
6:       $r \leftarrow$ Learn-One-Rule (E, A, y).
7:       Remove training records from E that are covered by r.
8:       Add r to the bottom of the rule list: $R \longrightarrow R \vee r$.
9:    **end while**
10: **end for**
11: Insert the default rule, $\{\} \longrightarrow y_k$, to the bottom of the rule list R.
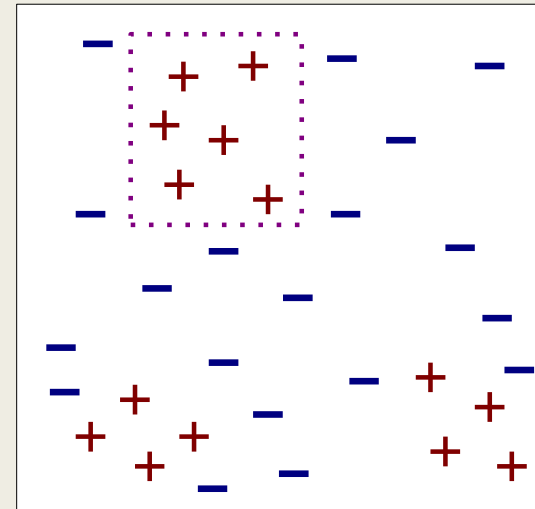
Sequential Covering Algorithm...

1. The algorithm **starts with an empty decision list, R.**

2. The **Learn-One- Rule function** is then used to **extract the best rule for class y that covers the current set of training records.**

3. During rule extraction, **all training records for class y are considered to be positive examples,** while **those that belong to other classes are considered to be negative examples.**

4. A **rule is desirable if it covers most of the positive examples and none (or very few) of the negative examples.** Once such a rule is found, **the training records covered by the rule are eliminated.**

5. **The new rule is added to the bottom of the decision list R.**

6. This procedure is **repeated until the stopping criterion is met.**

7. The algorithm then proceeds to generate rules for the next class.

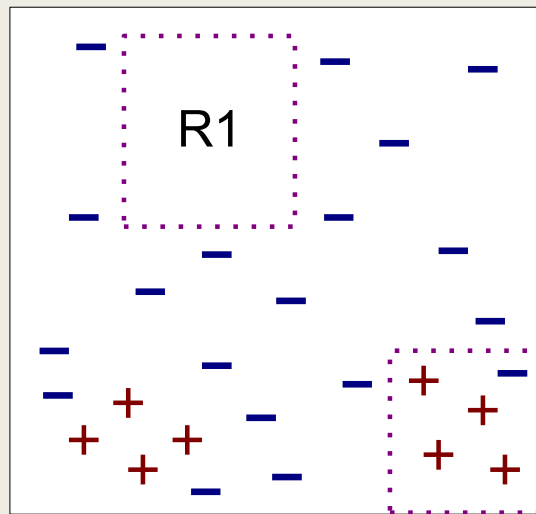# Example of Sequential Covering...
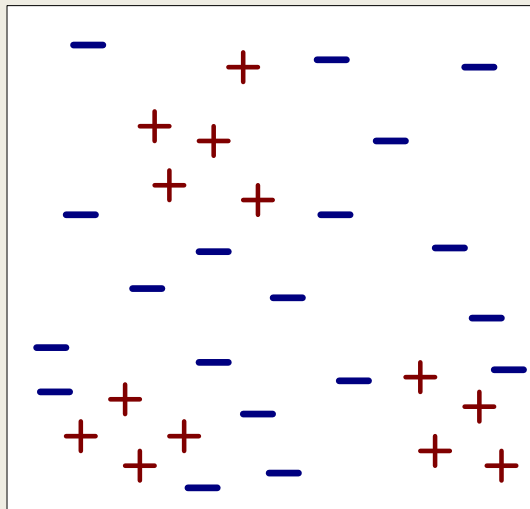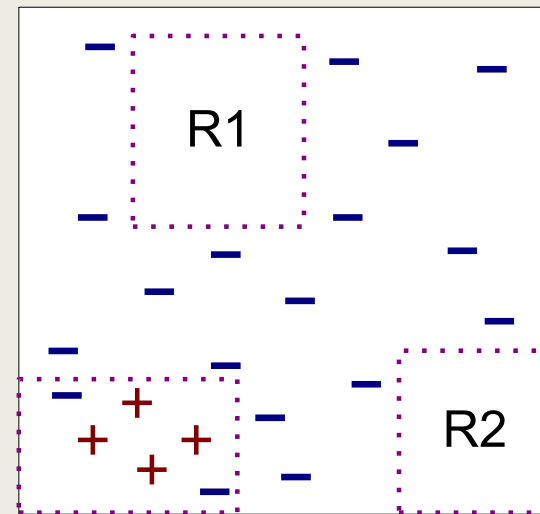

(i) Original Data

(ii) Step 1

Example of Sequential Covering

(iii) Step 2

(iv) Step 3

Learn-One Rule Function

- The **objective of the Learn-One-Rule function is to extract a classification rule that covers many of the positive examples** and none (or very few) of the negative examples in the training set.
- Finding an optimal rule is computationally expensive given the exponential size of the search space.
- The **Learn-One-Rule function addresses the exponential search problem by growing the rules in a greedy fashion.**
- It generates an **initial rule r and keeps refining the rule until a certain stopping criterion is met.**
- The rule is then pruned to improve its generalization error.

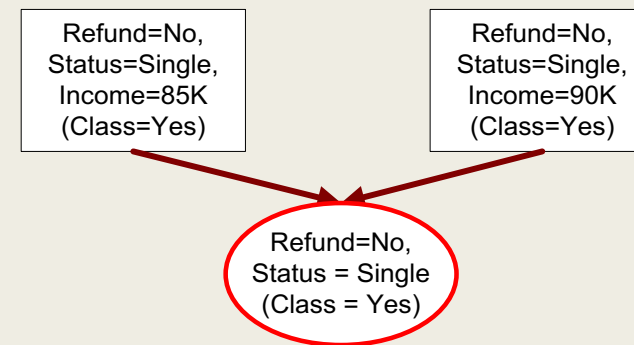# Aspects of Sequential Covering

- Rule Growing

- Instance Elimination

- Rule Evaluation

- Stopping Criterion

- Rule Pruning

Rule Growing

- Two common strategies
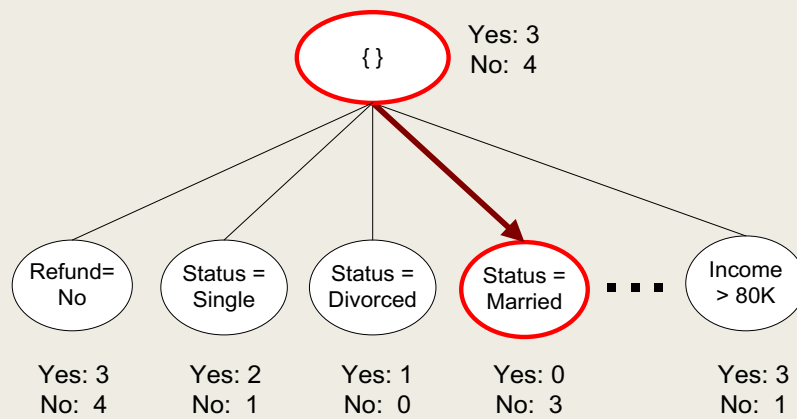
(a) General-to-specific

(b) Specific-to-general

Rule Evaluation…

- ■ Metrics:
  - – *Accuracy* $= \dfrac{n_c}{n}$

  - – *Laplace* $= \dfrac{n_c + 1}{n + k}$

  - – *M-estimate* $= \dfrac{n_c + kp}{n + k}$

n : Number of instances covered by rule

n_c *:* Number of c instances covered by rule

k : Number of classes

p : Prior probability

Rule Evaluation...

■ An evaluation metric is needed to determine which conjunct should be added (or removed) during the rule-growing process.

■ Accuracy is an obvious choice because it explicitly measures the fraction of training examples classified correctly by the rule.

■ However, **a potential limitation of accuracy is that it does not consider the rule's coverage.**

■ For example, consider a training set that contains 60 positive examples and 100 negative examples. Suppose we are given the following two candidate rules:

– *Rule r1: covers 50 positive examples and 5 negative examples,*

– *Rule r2: covers 2 positive examples and no negative examples.*

■ The accuracies for r1 and r2 are 90.9% and 100%, respectively.

– *r1 is the better rule despite its lower accuracy. The high accuracy for r2 is potentially spurious because the coverage of the rule is too low.*

Rule Evaluation...

■ The following approaches can be used to handle this problem.

1. A statistical test can be used to prune rules that have poor coverage.

■ Likelihood ratio statistic:

$$R = 2\sum_{i=1}^{k} f_i \log(f_i/e_i),$$

where k is the number of classes,

f_i is the observed frequency of class i examples that are covered by the rule, and

e_i is the expected frequency of a rule that makes random predictions.

Rule Evaluation…

Rule r1: covers 50 positive examples and 5 negative examples,

Rule r2: covers 2 positive examples and no negative examples.

■ For example, since *r1* covers 55 examples,

– $e_+ = 55 \times 60/160 = 20.625,$

$e_- = 55 \times 100/160 = 34.375.$

■ The likelihood ratio for r1 is $R(r_1) = 2 \times [50 \times \log_2(50/20.625) + 5 \times \log_2(5/34.375)] = 99.9.$

■ Similarly, $e_+ = 2 \times 60/160 = 120/160 = \frac{3}{4} = 0.75,$ $e_- = 0$, the likelihood ratio for r2 is

$$R(r_2) = 2 \times [2 \times \log_2(2/0.75) + 0 \times \log_2(0/1.25)] = 5.66.$$

■ This statistic therefore suggests that **r1 is a better rule than r2.**

Rule Evaluation...

Rule r1: covers 50 positive examples and 5 negative examples,

Rule r2: covers 2 positive examples and no negative examples.

2. An **evaluation metric that considers the rule coverage can be used.**

$$\text{Laplace} = \frac{f_+ + 1}{n + k},$$

$$\text{m-estimate} = \frac{f_+ + kp_+}{n + k},$$

where , n is the number of examples covered by the rule,

f_+ is the number of positive examples covered by the rule,

k is the total number of classes, and

p_+ is the prior probability for the positive class.

Note : m-estimate is equivalent to the Laplace measure by choosing p_+ = 1/k.

The Laplace measure for r1 is 51/57 = 89.47%, r2 is 3/4=75%

LECTURE 21

Dr.Vani V

Rule Evaluation...

3. An evaluation metric that considers the **support count** of the rule can be used. One such metric is the **FOIL's information gain.**

- The **support count of a rule corresponds to the number of positive examples covered by the rule.**

- Suppose the rule r : A ⟶ +
  - *covers p_0 positive examples and n_0 negative examples.*
- After adding a new conjunct B, the extended rule r : A ∧ B ⟶ +
  - *covers p1 positive examples and n1 negative examples.*
- Given this information, the FOIL's information gain of the extended rule is defined as follows:

$$\text{FOIL's information gain} = p_1 \times \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right).$$

Rule Growing (Examples)

- **CN2 Algorithm:**
  - *Start from an empty conjunct: {}*
  - *Add conjuncts that minimizes the entropy measure: {A}, {A,B}, …*
  - *Determine the rule consequent by taking majority class of instances covered by the rule*

- **RIPPER Algorithm:**
  - *Start from an empty rule: {} => class*
  - *Add conjuncts that **maximizes FOIL's information gain measure:***
    - R0: {} => class   (initial rule)
    - R1: {A} => class (rule after adding conjunct)
    - $Gain(R0, R1) = t [\ log (p1/(p1+n1)) – log (p0/(p0 + n0))]$
    - where   t: number of positive instances covered by both R0 and R1
      p0: number of positive instances covered by R0
      n0: number of negative instances covered by R0
      p1: number of positive instances covered by R1
      n1: number of negative instances covered by R1

Instance Elimination

- **Why do we need to eliminate instances?**
  - *Otherwise, the next rule is identical to previous rule*

- **Why do we remove positive instances?**
  - *Ensure that the next rule is different*

- **Why do we remove negative instances?**
  - *Prevent underestimating accuracy of rule*
  - *Compare rules R2 and R3 in the diagram*
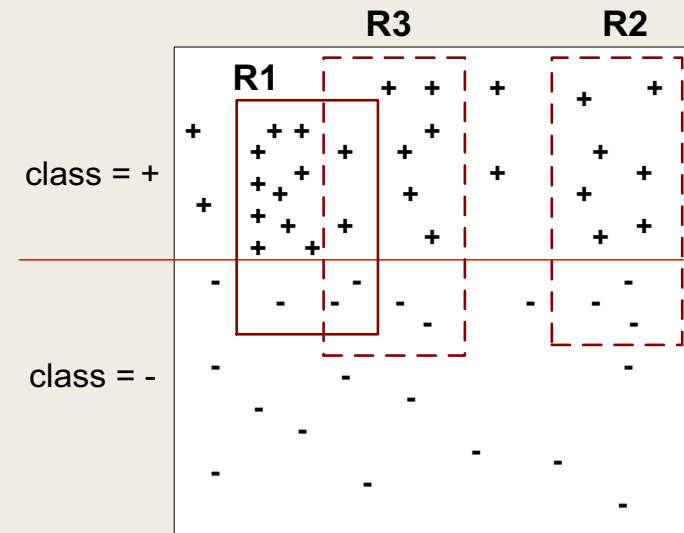
Stopping Criterion and Rule Pruning

- Stopping criterion
  - *Compute the gain*
  - *If gain is not significant, discard the new rule*

- Rule Pruning
  - *Reduced Error Pruning:*
    - Remove one of the conjuncts in the rule
    - Compare error rate on validation set before and after pruning
    - If error improves, prune the conjunct

Summary of Direct Method

- Grow a single rule

- Remove Instances from rule

- Prune the rule (if necessary)

- Add rule to Current Rule Set

- Repeat

Direct Method: RIPPER

■ For 2-class problem, choose one of the classes as positive class, and the other as negative class
  – *Learn rules for positive class*
  – *Negative class will be default class*

■ For multi-class problem
  – *Order the classes according to increasing class prevalence (fraction of instances that belong to a particular class)*
  – *Learn the rule set for smallest class first, treat the rest as negative class*
  – *Repeat with next smallest class as positive class*

Direct Method: RIPPER

- Growing a rule:
  - *Start from empty rule*
  - *Add conjuncts if they improve FOIL's information gain*
  - *Stop when rule no longer covers negative examples*
  - *Prune the rule immediately using incremental reduced error pruning*
  - *Measure for pruning: $v = (p-n)/(p+n)$*
    - p: number of positive examples covered by the rule in the validation set
    - n: number of negative examples covered by the rule in the validation set
  - *Pruning method: delete any final sequence of conditions that maximizes v*

Direct Method: RIPPER

- Building a Rule Set:
  - *Use sequential covering algorithm*
    - Finds the best rule that covers the current set of positive examples
    - Eliminate both positive and negative examples covered by the rule
  - *Each time a rule is added to the rule set, compute the new description length*
    - stop adding new rules when the new description length is d bits longer than the smallest description length obtained so far

Direct Method: RIPPER

- Optimize the rule set:
  - *For each rule r in the rule set **R***
    - Consider 2 alternative rules:
      - *Replacement rule (r*): grow new rule from scratch*
      - *Revised rule(r'): add conjuncts to extend the rule r*
    - Compare the rule set for *r* against the rule set for r* and r'
    - Choose rule set that minimizes MDL (Minimum Description Length) principle
  - *Repeat rule generation and rule optimization for the remaining positive examples*

Example

Name	Give Birth	Lay Eggs	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	no	yes	mammals
python	no	yes	no	no	no	reptiles
salmon	no	yes	no	yes	no	fishes
whale	yes	no	no	yes	no	mammals
frog	no	yes	no	sometimes	yes	amphibians
komodo	no	yes	no	no	yes	reptiles
bat	yes	no	yes	no	yes	mammals
pigeon	no	yes	yes	no	yes	birds
cat	yes	no	no	no	yes	mammals
leopard shark	yes	no	no	yes	no	fishes
turtle	no	yes	no	sometimes	yes	reptiles
penguin	no	yes	no	sometimes	yes	birds
porcupine	yes	no	no	no	yes	mammals
eel	no	yes	no	yes	no	fishes
salamander	no	yes	no	sometimes	yes	amphibians
gila monster	no	yes	no	no	yes	reptiles
platypus	no	yes	no	no	yes	mammals
owl	no	yes	yes	no	yes	birds
dolphin	yes	no	no	yes	no	mammals
eagle	no	yes	yes	no	yes	birds

C4.5 versus C4.5rules versus RIPPER
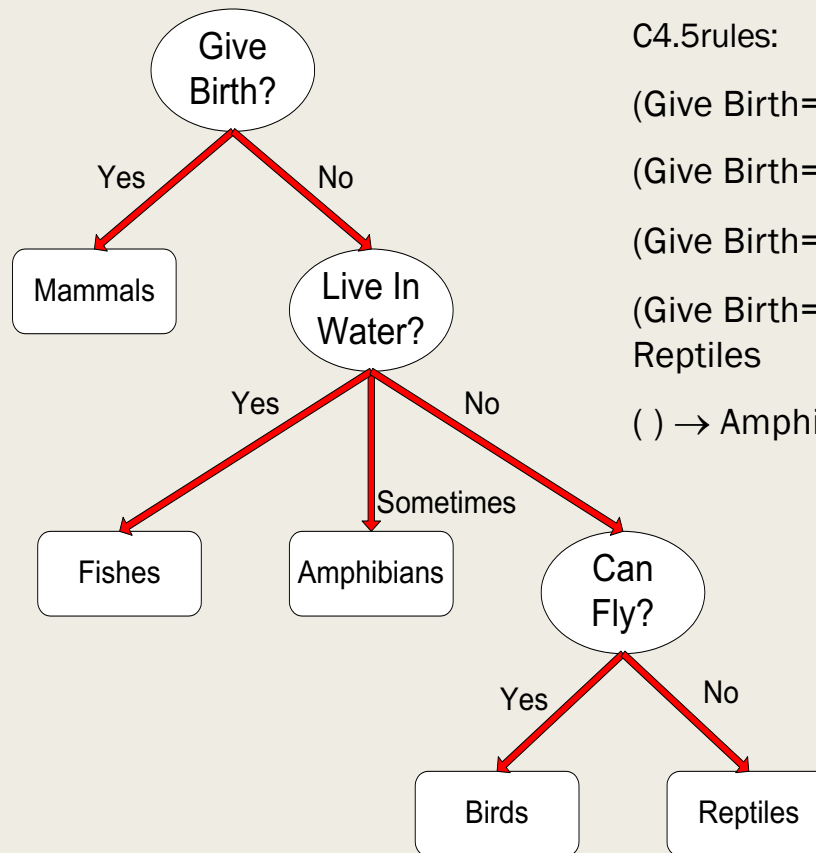
C4.5rules:

(Give Birth=No, Can Fly=Yes) → Birds

(Give Birth=No, Live in Water=Yes) → Fishes

(Give Birth=Yes) → Mammals

(Give Birth=No, Can Fly=No, Live in Water=No) → Reptiles

RIPPER:

( ) → Amphibians

(Live in Water=Yes) → Fishes

(Have Legs=No) → Reptiles

(Give Birth=No, Can Fly=No, Live In Water=No)
        → Reptiles

(Can Fly=Yes,Give Birth=No) → Birds

() → Mammals

Advantages of Rule-Based Classifiers

- As highly expressive as decision trees
- Easy to interpret
- Easy to generate
- Can classify new instances rapidly
- Performance comparable to decision trees

Characteristics of Rule-Based Classifiers

A rule-based classifier has the following characteristics:

- The expressiveness of a rule set is almost equivalent to that of a decision tree

- **If the rule-based classifier allows multiple rules to be triggered for a given record, then a more complex decision boundary can be constructed.**

- **Rule-based classifiers are generally used to produce descriptive models that are easier to interpret but gives comparable performance to the decision tree classifier.**

- **The class-based ordering approach adopted by many rule-based classifiers (such as RIPPER) is well suited for handling data sets with imbalanced class distributions.**

Selected Exercises: 1...

- Consider a binary classification problem with the following set of attributes and attribute values:

  - Air Conditioner = {Working, Broken}

  - Engine = {Good, Bad}

  - Mileage = {High, Medium, Low}

  - Rust = {Yes, No}

- Suppose a rule-based classifier produces the following rule set:

  r1: Mileage = High $-\rightarrow$ Value = Low

  r2: Mileage = Low $-\rightarrow$ Value = High

  r3: Air Conditioner = Working, Engine = Good $-\rightarrow$ Value = High

  r4: Air Conditioner = Working, Engine = Bad $-\rightarrow$ Value = Low

  r5: Air Conditioner = Broken $-\rightarrow$ Value = Low

Selected Exercises:1

(a) Are the rules mutually exclusive?

No

(b) Is the rule set exhaustive?

Yes

(c) Is ordering needed for this set of rules?

Yes, because a test instance may trigger more than one rule.

(d) Do you need a default class for the rule set?

No, because every instance is guaranteed to trigger at least one rule.

Selected Exercises:2...

Given:

Suppose R1 is covered by 350 positive examples and 150 negative examples, while R2 is covered by 300 positive examples and 50 negative examples. Compute the FOIL's information gain for the rule R2 with respect to R1.

Solution:

$$\text{FOIL's information gain} = p_1 \times \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right).$$

For this problem, p0 = 350, n0 = 150, p1 = 300, and n1 = 50. Therefore, the FOIL's information gain for R2 with respect to R1 is:

$$Gain = 300 \; x \; [log_2 \frac{300}{350} - log_2 \frac{350}{500}] \quad = 87.65$$

2. The RIPPER algorithm (by Cohen [170]) is an extension of an earlier algorithm called IREP (by Fürnkranz and Widmer [184]). Both algorithms apply the **reduced-error pruning** method to determine whether a rule needs to be pruned. The reduced error pruning method uses a validation set to estimate the generalization error of a classifier. Consider the following pair of rules:

$$R_1: \quad A \longrightarrow C$$
$$R_2: \quad A \wedge B \longrightarrow C$$

R_2 is obtained by adding a new conjunct, B, to the left-hand side of R_1. For this question, you will be asked to determine whether R_2 is preferred over R_1 from the perspectives of rule-growing and rule-pruning. To determine whether a rule should be pruned, IREP computes the following measure:

$$v_{IREP} = \frac{p + (N - n)}{P + N},$$

where P is the total number of positive examples in the validation set, N is the total number of negative examples in the validation set, p is the number of positive examples in the validation set covered by the rule, and n is the number of negative examples in the validation set covered by the rule. v_{IREP} is actually similar to classification accuracy for the validation set. IREP favors rules that have higher values of v_{IREP}. On the other hand, RIPPER applies the following measure to determine whether a rule should be pruned:

$$v_{RIPPER} = \frac{p - n}{p + n}.$$

Selected Exercises:2...

Given : Consider a validation set that contains 500 positive examples and 500 negative examples. For R1, suppose the number of positive examples covered by the rule is 200, and the number of negative examples covered by the rule is 50. For R2, suppose the number of positive examples covered by the rule is 100 and the number of negative examples is 5.

(1) Compute $vIREP$ for both rules. Which rule does $IREP$ prefer?

Selected Exercises:2...

(1) For the given problem, P = 500, and N = 500.

For rule R1, p = 200 and n = 50. Therefore,

$$V_{IREP}(R1) = \frac{p + (N - n)}{P + N}$$

$$= \frac{200 + (500 - 50)}{1000} = 0.65$$

For rule R2, p = 100 and n = 5.

$$V_{IREP}(R2) = \frac{p + (N - n)}{P + N}$$

$$= \frac{100 + (500 - 5)}{1000} = 0.595$$

Thus, IREP prefers rule R1.

Selected Exercises:2

(2) Compute v_{RIPPER} for the given problem. Which rule does RIPPER prefer?

For rule R1, p = 200 and n = 50.

$$V_{RIPPER}(R1) = \frac{p - n}{p + n}$$

$$= \frac{200 - 50}{200 + 50} = 0.6$$

For rule R2, p = 100 and n = 5.

$$V_{RIPPER}(R2) = \frac{p + n}{p + n}$$

$$= \frac{100 - 5}{100 + 5} = 0.9$$

Thus, RIPPER prefers rule R2.

Selected Exercises:3...

C4.5rules is an implementation of an indirect method for generating rules from a decision tree. RIPPER is an implementation of a direct method for generating rules directly from data.

(a) Discuss the strengths and weaknesses of both methods.

The C4.5 rules algorithm generates classification rules from a global perspective. This is because the rules are derived from decision trees, which are induced with the objective of partitioning the feature space into homogeneous regions, without focusing on any classes. In contrast, RIPPER generates rules one-class-at-a-time. Thus, it is more biased towards the classes that are generated first.

Selected Exercises:3

C4.5 rules is an implementation of an indirect method for generating rules from a decision tree. RIPPER is an implementation of a direct method for generating rules directly from data.

(b) Consider a data set that has a large difference in the class size (i.e., some classes are much bigger than others). Which method (between C4.5 rules and RIPPER) is better in terms of finding high accuracy rules for the small classes?

The class-ordering scheme used by C4.5 rules has an easier interpretation than the scheme used by RIPPER.

Selected Exercises:4...

Consider a training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

R1: A → + (covers 4 positive and 1 negative examples),

R2: B → + (covers 30 positive and 10 negative examples),

R3: C → + (covers 100 positive and 90 negative examples),

Determine which is the best and worst candidate rule according to:

(a) Rule accuracy.

(b) FOIL's information gain.

(c) The likelihood ratio statistic.

(d) The Laplace measure.

(e) The m-estimate measure (with k = 2 and p+ = 0.2).

Selected Exercises:4...

(a) Rule accuracy

Solution: R1 : 4/5

R2: 30/40

R3:100/190

The accuracies of the rules are 80% (for R1), 75% (for R2), and 52.6% (for R3), respectively. Therefore, R1 is the best candidate and R3 is the worst candidate according to rule accuracy

Selected Exercises:4...

(b) Foil's information gain

Solution: Assume the initial rule is $\emptyset \rightarrow +$. This rule covers $p0 = 100$ positive examples and $n0 = 400$ negative examples. The rule R1 covers $p1 = 4$ positive examples and $n1 = 1$ negative example. Therefore, the FOIL's information gain for this rule is

$$\text{FOIL's information gain} = p_1 \times \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right).$$

$$Gain(R1) = 4 \; x \; [log_2 \frac{4}{5} - log_2 \frac{100}{500}] \; = 8$$

$$Gain(R2) = 57.2$$

$$Gain(R3) = 139.6$$

Therefore, R3 is the best candidate and R1 is the worst candidate according to FOIL's information gain.

Selected Exercises:4...

(c)The likelihood ratio statistic:

$$R = 2 \sum_{i=1}^{k} f_i \log(f_i/e_i),$$

Solution:

The given training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

R1: A $\rightarrow$ + (covers 4 positive and 1 negative examples),

For R1, the expected frequency for the positive class is 5 x 100/500 = 1 and the expected frequency for the negative class is 5 x 400/500 = 4. Therefore, the likelihood ratio for R1 is

R(R1) = $2 \times [4x \, log_2\frac{4}{1} + 1\text{x} \, log_2\frac{1}{4}]$ = 12

Selected Exercises:4...

(c)The likelihood ratio statistic:

$$R = 2\sum_{i=1}^{k} f_i \log(f_i/e_i),$$

Solution:

The given training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

R2: B → + (covers 30 positive and 10 negative examples),

For R2, the expected frequency for the positive class is 40x 100/500 = 8 and the expected frequency for the negative class is 40 x400/500 = 32. Therefore, the likelihood ratio for R2 is

R(R2) = $2 x [30x \ log_2 \frac{30}{8} + 10x \ log_2 \frac{10}{32}]$ = 80.85

Selected Exercises:4...

(c)The likelihood ratio statistic:

$$R = 2\sum_{i=1}^{k} f_i \log(f_i/e_i),$$

Solution:

The given training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

R3: C → + (covers 100 positive and 90 negative examples),

For R3, the expected frequency for the positive class is 190 x 100/500 = 38 and the expected frequency for the negative class is 190x 400/500 = 152. Therefore, the likelihood ratio for R3 is

R(R3) = $2\,x\,[100\,x\,log_2\dfrac{100}{38} + 90x\,log_2\dfrac{90}{152}]$ = 143.09

Therefore, R3 is the best candidate and R1 is the worst candidate according to the likelihood ratio statistic.

Selected Exercises:4...

(d) The Laplace measure.

$$\text{Laplace} = \frac{f_+ + 1}{n + k},$$

> where, n is the number of examples covered by the rule,
>
> f_+ is the number of positive examples covered by the rule,
>
> k is the total number of classes, and
>
> p_+ is the prior probability for the positive class.

Given: Consider a training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

R1: A → + (covers 4 positive and 1 negative examples),

R2: B → + (covers 30 positive and 10 negative examples),

R3: C → + (covers 100 positive and 90 negative examples),

Solution:

- The Laplace measure of the rules are 71.43% (for R1), 73.81% (for R2), and 52.6% (for R3), respectively. **Therefore, R2 is the best candidate and R3 is the worst candidate according to the Laplace measure.**

Selected Exercises:4

e) The m-estimate measure (with k = 2 and p+ = 0.2).

$$\text{m-estimate} = \frac{f_+ + kp_+}{n + k},$$

Given: Consider a training set that contains 100 positive examples and 400 negative examples. For each of the following candidate rules,

R1: A → + (covers 4 positive and 1 negative examples),

R2: B → + (covers 30 positive and 10 negative examples),

R3: C → + (covers 100 positive and 90 negative examples),

Solution:

The m-estimate measure of the rules are 62.86% (for R1), 73.38% (for R2), and 52.3% (for R3), respectively. Therefore, R2 is the best candidate and R3 is the worst candidate according to the m-estimate measure.

LECTURE 22

Dr.Vani V

Selected Exercises:5

Consider the one-dimensional data set shown in below table.

x	0.5	3.0	4.5	4.6	4.9	5.2	5.3	5.5	7.0	9.5
y	−	−	+	+	+	−	−	+	−	−

(a) Classify the data point x = 5.0 according to its 1-, 3-, 5-, and 9-nearest neighbors (using majority vote).

(b) Repeat the previous analysis using the distance-weighted voting approach

Solution:

Majority Voting: $y' = \text{argmax}_{v} \sum_{(x_i, y_i) \in D_z} I(v = y_i),$

Distance-Weighted Voting: $y' = \text{argmax}_{v} \sum_{(x_i, y_i) \in D_z} w_i \times I(v = y_i).$

(a)
1-nearest neighbor: +,
3-nearest neighbor: −,
5-nearest neighbor: +,
9-nearest neighbor: −.

(b)
1-nearest neighbor: +,
3-nearest neighbor: +,
5-nearest neighbor: +,
9-nearest neighbor: +.

Example: PEBLS

- PEBLS: Parallel Examplar-Based Learning System (Cost & Salzberg)
  - *Works with both continuous and nominal features*
    - For nominal features, distance between two nominal values is computed using Modified Value Difference Metric (MVDM)
  - *Each record is assigned a weight factor*
  - *Number of nearest neighbor, k = 1*

Example: PEBLS

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Distance between nominal attribute values:

d(Single,Married)

= | 2/4 – 0/4 | + | 2/4 – 4/4 | = 1

d(Single,Divorced)

= | 2/4 – 1/2 | + | 2/4 – 1/2 | = 0

d(Married,Divorced)

= | 0/4 – 1/2 | + | 4/4 – 1/2 | = 1

d(Refund=Yes,Refund=No)

= | 0/3 – 3/7 | + | 3/3 – 4/7 | = 6/7

Class	Marital Status		
	Single	Married	Divorced
Yes	2	0	1
No	2	4	1

Class	Refund	
	Yes	No
Yes	0	3
No	3	4

$$d(V_1, V_2) = \sum_i \left| \frac{n_{1i}}{n_1} - \frac{n_{2i}}{n_2} \right|$$

Example: PEBLS

Tid	Refund	Marital Status	Taxable Income	Cheat
X	Yes	Single	125K	**No**
Y	No	Married	100K	**No**

Distance between record X and record Y:

$$\Delta(X,Y) = w_X w_Y \sum_{i=1}^{d} d(X_i, Y_i)^2$$

where:

$$w_X = \frac{\text{Number of times X is used for prediction}}{\text{Number of times X predicts correctly}}$$

$w_X \cong 1$ if X makes accurate prediction most of the time

$w_X > 1$ if X is not reliable for making predictions

Selected Exercises:6...

Consider the training set for the loan classification problem shown in Figure
Use the Modified Value Difference Metric (MVDM) measure to compute the distance between every pair of attribute values for the **Home Owner and Marital Status attributes.**

	binary	categorical	continuous	class
Tid	**Home Owner**	**Marital Status**	**Annual Income**	**Defaulted Borrower**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Selected Exercises:6

Tid	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Class	Marital Status		
	Single	Married	Divorced
Yes	2	0	1
No	2	4	1

Class	Home Owner	
	Yes	No
Yes	0	3
No	3	4

$d(\text{Single, Married}) = 1$

$d(\text{Single, Divorced}) = 0$

$d(\text{Married, Divorced}) = 1$

$d(Home\ Owner\ =\ yes, Home\ Owner\ =\ no) = 6/7$
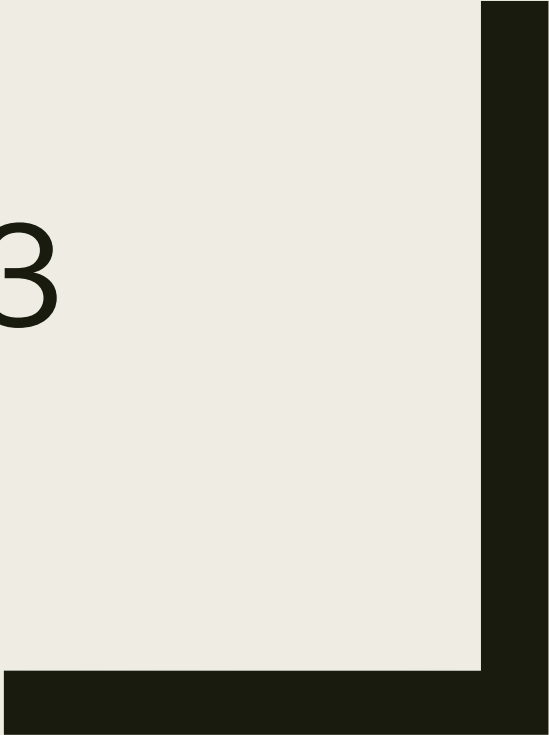
References

TEXTBOOKS :
1. Pang-Ning Tan, Vipin Kumar, Michael Steinbach: **Introduction to Data Mining**, Pearson, 2012.
2. Jiawei Han and Micheline Kamber: **Data Mining - Concepts and Techniques**, 3rd Edition, MorganKaufmann Publisher, 2014
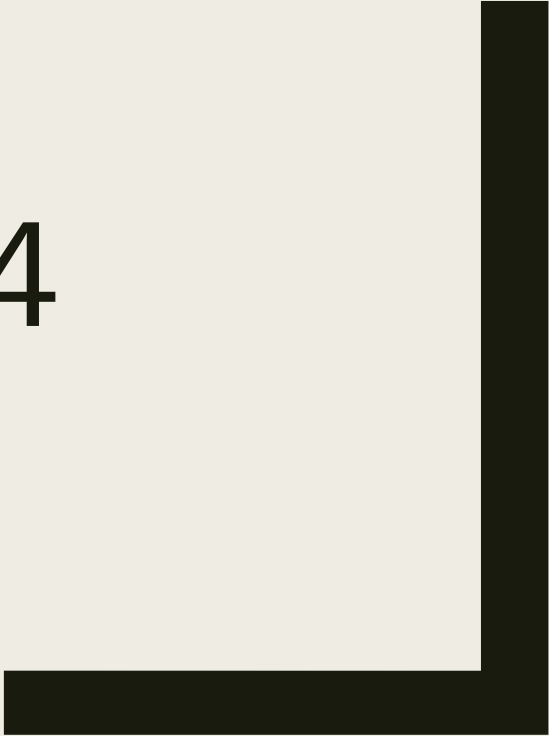
LECTURE 23

Support Vector Machine (SVM)

Support Vector Machine

- *Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labelled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.*

LECTURE 24

Support Vector Machine (SVM)

Support Vector Machine

- The Support Vector Machine (SVM) , which is one of the most popular algorithms in modern machine learning.

- It was introduced by Vapnik in 1992

- It often (but not always) provides very impressive classification performance on reasonably sized datasets

- SVMs do not work well on extremely large datasets,

Optimal Separation

- Figure shows a simple classification problem with three different possible linear classification lines.

- All three of the lines that are drawn separate out the two classes, so in some sense they are 'correct', and the Perceptron would stop its training if it reached any one of them.

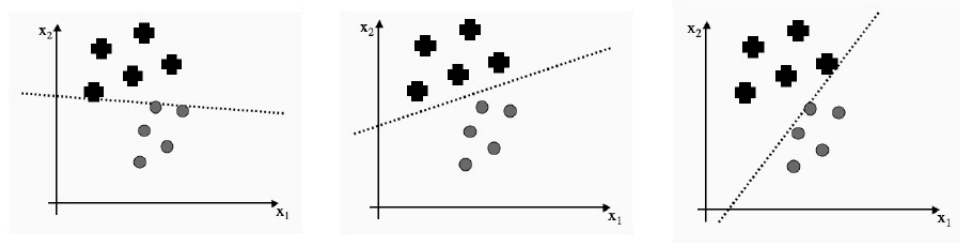- why one of the lines should be any better than the others?

FIGURE 8.1 Three different classification lines. Is there any reason why one is better than the others?

The Margin and Support Vectors

- Measure the distance to travel away from the line (in a direction perpendicular to the line) before we hit a datapoint.

- Imagine to put a 'no-man's land' around the line (shown in Figure 8.2), so that any point that lies within that region is declared to be too close to the line to be accurately classified.

- This region is symmetric about the line, so that it forms a cylinder about the line in 3D and a hyper-cylinder in higher dimensions.

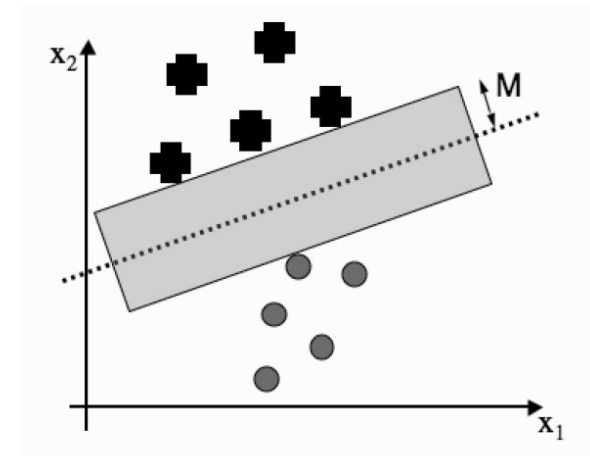- **This largest radius is known as the margin, labelled M.**

FIGURE 8.2 The margin is the largest region we can put that separates the classes without there being any points inside, where the box is made from two lines that are parallel to the decision boundary.

The Margin and Support Vectors

- The classifier in the middle of Figure 8.1 has the largest margin of the three. It has the imaginative name of **the maximum margin (linear) classifier.**

- **The datapoints in each class that lie closest to the classification line are called support vectors.**

- **first that the margin should be as large as possible,** and

- **second that the support vectors are the most useful datapoints because they are the ones that might get wrong.**
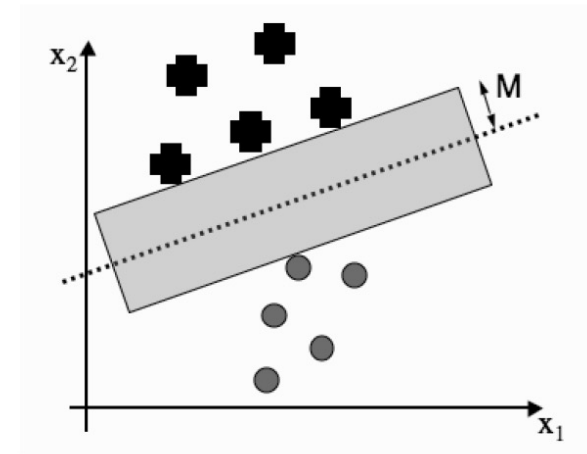

FIGURE 8.2 The margin is the largest region we can put that separates the classes without there being any points inside, where the box is made from two lines that are parallel to the decision boundary.

The Margin and Support Vectors

■ This leads to an interesting feature of these algorithms: **After training , throw away all the data except for the support vectors, and use them for classification, which is a useful saving in data storage.**
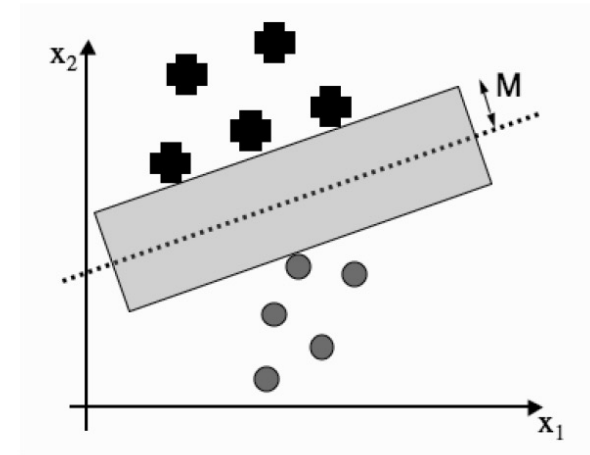

FIGURE 8.2 The margin is the largest region we can put that separates the classes without there being any points inside, where the box is made from two lines that are parallel to the decision boundary.

Optimal Decision Boundary

Points to recollect :

- a weight vector (a vector, not a matrix, since there is only one output) and an input vector x.

- The output used was $y = \mathbf{w} \cdot \mathbf{x} + b$, with b being the contribution from the bias weight.

- Used the classifier line: any x value that gives a positive value for $\mathbf{w} \cdot \mathbf{x} + b$ is above the line, and so is an example of the '+' class, while any x that gives a negative value is in the '-' class.

- In the new version, include no-man's land,
  - *Instead of checking whether the value of w · x + b is positive or negative, Check whether the absolute value is less than margin M.*

- Remember that $\mathbf{w} \cdot \mathbf{x}$ is the inner or scalar product, $\mathbf{w} \cdot \mathbf{x} = \sum_i w_i x_i$. This can also be written as $w^T x$.

Optimal Decision Boundary

- For a given margin value M we can say that any point x where $W^T X + b \geq$ M is a plus, and any point where $W^T X + b \leq$ −M is a circle.

- The actual separating hyperplane is specified by $W^T X + b = 0$.

- Suppose, we pick a point X^+ that lies on the '+' class boundary line, so that $W^T X^+ =$ M. This is a support vector.

- If we want to find the closest point that lies on the boundary line for the '*o*' class, then we travel perpendicular to the '+' boundary line until we hit the '*o*' boundary line. The point that we hit is the closest point, and we'll call it X^-.

- How far did we have to travel in this direction?

Optimal Decision Boundary

- Figure 8.2 shows that **the distance travelled is M to get to the separating hyperplane**, and then M from there to the opposing support vector.

- The weight vector w is perpendicular to the classifier line.

- If it is perpendicular to the classifier line, then it is obviously perpendicular to the '+' and 'o' boundary lines too, so the direction we travelled from x^+ to x^- is along w.

- Make w a unit vector w/‖w‖, and so we see that the margin is 1/ ‖ w ‖.
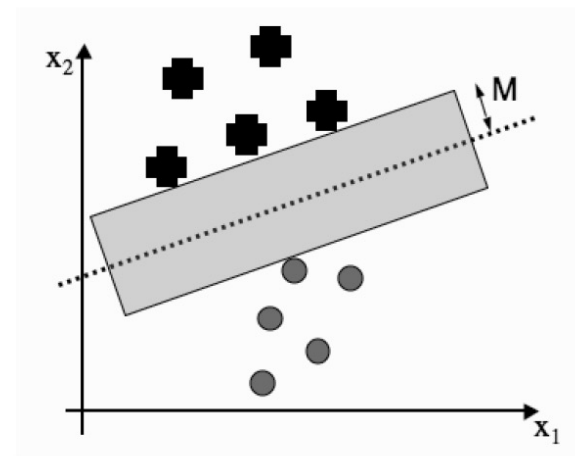

FIGURE 8.2   The margin is the largest region we can put that separates the classes without there being any points inside, where the box is made from two lines that are parallel to the decision boundary.

Requirements

- Find the w and b that give the biggest possible value of M.

- Our knowledge that the width of the margin is 1/ ‖ w ‖ tells that making M as large as possible is the same as making w^Tw as small as possible.

- If that was the only constraint, then set w = 0, and the problem would be solved, but the classification line to separate out the '+' data from the 'o', is needed.

- So, satisfy two problems simultaneously:
  - *find a decision boundary that classifies well,*
  - *while also making w^Tw as small as possible.*

- Mathematically, write these requirements as: **minimise** $\frac{1}{2}w^Tw$ (where the half is there for convenience as in so many other cases) subject to some constraint that says that the data are well matched

A Constrained Optimisation Problem...

- How do we decide whether a classifier is any good?
  - *Write down a set of constraints that say that the classifier should get the answer right.*
  - *Set the target answers for our two classes be ±1, rather than 0 and 1.*
  - *Then write down t_i x y_i, the target multiplied by the output, and this will be positive if the two are the same and negative otherwise.*
  - *Then, write down the equation of the straight line again, which is how we computed y, to see $t_{i(}w^T x + b_)} \geq 1$.*
  - *This means that the constraints just need to check each datapoint for this condition. So , the full problem that we wish to solve is:*
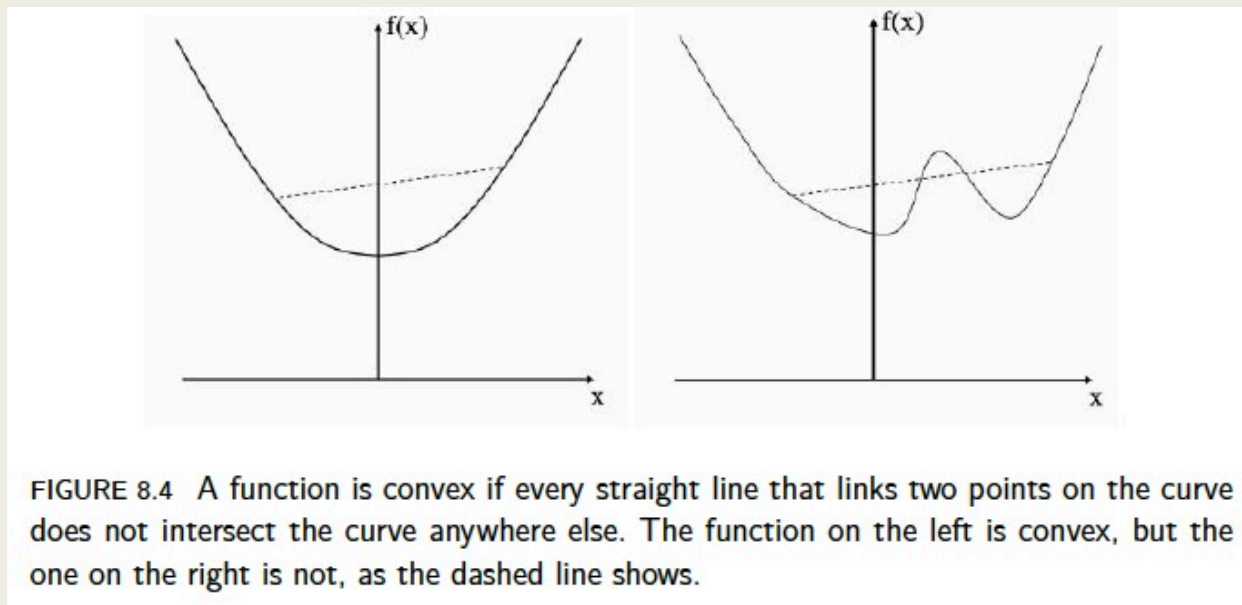
$$\text{minimise } \frac{1}{2}\mathbf{w}^T\mathbf{w} \text{ subject to } t_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \text{ for all } i = 1, \ldots n.$$

A Constrained Optimisation Problem...

minimise $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ subject to $t_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ for all $i = 1, \dots n.$

- To solve this equation, there is a method that is much better suited over gradient descent, which is quadratic programming,

- It takes advantage of the fact that the problem described is quadratic and therefore convex and has linear constraints.

- A convex problem is one where , if we take any two points on the line and join them with a straight line, then every point on the line will be above the curve. Figure 8.4 shows an example of a convex and a non-convex function.

- Convex functions have a unique minimum, which is easy to see in one dimension, and remains true in any number of dimensions.

A Constrained Optimisation Problem...

FIGURE 8.4 A function is convex if every straight line that links two points on the curve does not intersect the curve anywhere else. The function on the left is convex, but the one on the right is not, as the dashed line shows.

- Figure 8.4 shows an example of a convex and a non-convex function.

- Convex functions have a unique minimum, which is easy to see in one dimension, and remains true in any number of dimensions.

A Constrained Optimisation Problem..

- The types of problem that we are interested in can be solved directly and efficiently (i.e., in polynomial time).

- Since the problem is quadratic, there is a unique optimum.

- When we find that optimal solution, the **Karush–Kuhn–Tucker (KKT)** conditions will be satisfied.

- These are (for all values of i from 1 to n, and where the ***** denotes the optimal value of each parameter):

$$
\begin{aligned}
\lambda_i^*(1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*)) &= 0 \\
1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*) &\leq 0 \\
\lambda_i^* &\geq 0,
\end{aligned}
$$

A Constrained Optimisation Problem...

$$\lambda_i^*(1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*)) = 0$$
$$1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*) \leq 0$$
$$\lambda_i^* \geq 0,$$

where the λ_i are positive values known as Lagrange multipliers, which are a standard approach to solving equations with equality constraints.

- The first of these conditions tells us that if $\lambda_i \neq 0$ then $(1 - ti(wTxi + b)) = 0$.

- This is only true for the support vectors (the SVMs provide a sparse representation of the data),

- So, consider only them, and can ignore the rest.

A Constrained Optimisation Problem…

■ In the jargon, the support vectors are those vectors in the **active set** of constraints.

■ For the support vectors the constraints are equalities instead of inequalities. We can therefore solve the Lagrangian function:

$$\mathcal{L}(\mathbf{w}, b, \lambda) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n}\lambda_i(1 - t_i(\mathbf{w}^T\mathbf{x}_i + b)), \qquad (8.5)$$

We differentiate this function with respect to the elements of $\mathbf{w}$ and b:

$$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{i=1}^{n}\lambda_i t_i \mathbf{x}_i, \qquad (8.6)$$

and

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n}\lambda_i t_i. \qquad (8.7)$$

If we set the derivatives to be equal to zero, so that we find the saddle points (maxima) of the function, we see that:

$$\mathbf{w}^* = \sum_{i=1}^{n}\lambda_i t_i \mathbf{x}_i, \quad \sum_{i=1}^{n}\lambda_i t_i = 0. \qquad (8.8)$$

We can substitute these expressions at the optimal values of $\mathbf{w}$ and b into Equation (8.5) and, after a little bit of rearranging, we get (where λ is the vector of the λ_i):

$$\mathcal{L}(\mathbf{w}^*, b^*, \lambda) = \sum_{i=1}^{n}\lambda_i - \sum_{i=1}^{n}\lambda_i t_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j t_i t_j \mathbf{x}_i^T\mathbf{x}_j, \qquad (8.9)$$

A Constrained Optimisation Problem...

- Notice that using the derivative with respect to b we can treat the middle term as 0.

- **This equation is known as the dual problem,** and the aim is to maximise it with respect to the λ_i variables.

- The constraints are $that \quad \lambda_i >= 0$ for all i, and $\sum_{i=1}^{n} \lambda_i t_i = 0$

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \lambda_i(1 - t_i(\mathbf{w}^T\mathbf{x}_i + b)), \qquad (8.5)$$

We differentiate this function with respect to the elements of $\mathbf{w}$ and b:

$$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i, \qquad (8.6)$$

and

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \lambda_i t_i. \qquad (8.7)$$

If we set the derivatives to be equal to zero, so that we find the saddle points (maxima) of the function, we see that:

$$\mathbf{w}^* = \sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i, \quad \sum_{i=1}^{n} \lambda_i t_i = 0. \qquad (8.8)$$

We can substitute these expressions at the optimal values of $\mathbf{w}$ and b into Equation (8.5) and, after a little bit of rearranging, we get (where $\boldsymbol{\lambda}$ is the vector of the λ_i):

$$\mathcal{L}(\mathbf{w}^*, b^*, \boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i t_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j t_i t_j \mathbf{x}_i^T \mathbf{x}_j, \qquad (8.9)$$

A Constrained Optimisation Problem...

■ Equation (8.8) gives an expression for w* , but we also want to know what b* is.

$$\mathbf{w}^* = \sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i, \quad \sum_{i=1}^{n} \lambda_i t_i = 0. \tag{8.8}$$

We can substitute these expressions at the optimal values of $\mathbf{w}$ and b into Equation (8.5) and, after a little bit of rearranging, we get (where $\boldsymbol{\lambda}$ is the vector of the λ_i):

$$\mathcal{L}(\mathbf{w}^*, b^*, \boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i t_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j t_i t_j \mathbf{x}_i^T \mathbf{x}_j, \tag{8.9}$$

■ We know that for a support vector $t_i(w^T x_i + b) = 1$,

■ We can substitute the expression for w* into there and substitute in the (x, t) of one of the support vectors.

A Constrained Optimisation Problem

- In case of errors, it is not very stable, so it is better to average it over the whole set of N_s support vectors:

$$b^* = \frac{1}{N_s} \sum_{\text{support vectors } j} \left(t_j - \sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i^T \mathbf{x}_j \right). \tag{8.10}$$

We can also use Equation (8.8) to see how to make a prediction, since for a new point $\mathbf{z}$:

$$\mathbf{w}^{*T} \mathbf{z} + b^* = \left(\sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i \right)^T \mathbf{z} + b^*. \tag{8.11}$$

- This means that **to classify a new point**, we just need to **compute the inner product between the new datapoint and the support vectors.**

Slack Variables for Non-Linearly Separable Problems...

- So far, it is assumed that the dataset is linearly separable.

- If we have a non-linearly separable dataset, then we cannot satisfy the constraints for all the datapoints.

- The solution is to **introduce some slack variables $\boldsymbol{\eta_i} \geq 0$** so that the constraints become $\boldsymbol{t_{i(}w^T xi + b)} \geq \boldsymbol{1 - \eta_i}$. For inputs that are correct, we set $\boldsymbol{\eta_i = 0}$.

Slack Variables for Non-Linearly Separable Problems...

- The **slack variables** tells that
  - *when comparing classifiers, we should consider the case where one classifier makes a mistake by putting a point just on the wrong side of the line, and*
  - *another puts the same point a long way onto the wrong side of the line.*

- The first classifier is better than the second, because the mistake was not as serious, so we should include this information in our minimisation criterion. We can do this by modifying the problem

Slack Variables for Non-Linearly Separable Problems...

- **Add a term into the minimisation problem so that we will now minimise w^Tw + C x** (distance of misclassified points from the correct boundary line).

- Here, **C is a trade off parameter** that decides how much weight to put onto each of the two criteria:
    - *Small C means we prize a large margin over a few errors, while large C means the opposite.*

- This transforms the problem into a **soft-margin classifier**, since we are allowing for a few mistakes.

- Writing this in a more mathematical way, the function that we want to minimise is:

$$L(\mathbf{w}, \epsilon) = \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \epsilon_i.$$

Slack Variables for Non-Linearly Separable Problems...

- The derivation of the dual problem worked out earlier still holds, except that

- $0 \leq \lambda_i \leq C$, and the support vectors are now those vectors with $\lambda_i > 0$. The KKT conditions are slightly different, too:
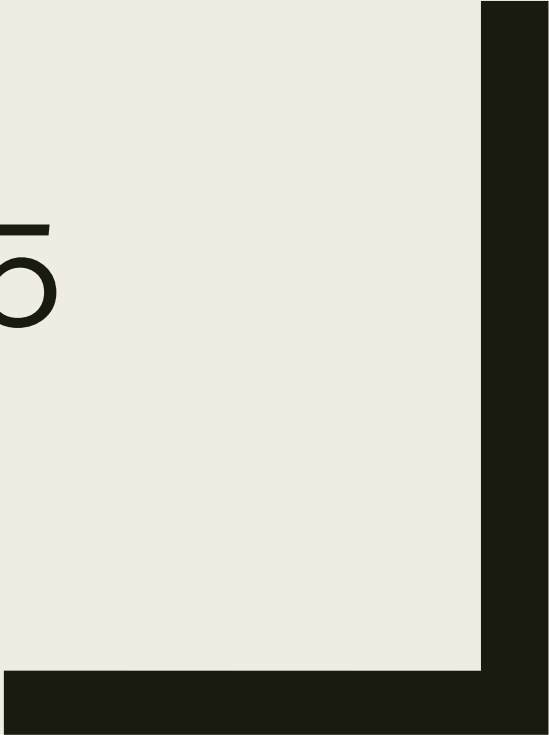
$$
\begin{aligned}
\lambda_i^*(1 - t_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*) - \eta_i) &= 0 \\
(C - \lambda_i^*)\eta_i &= 0 \\
\sum_{i=1}^{n} \lambda_i^* t_i &= 0.
\end{aligned}
$$

- The second condition tells us that, if $\lambda_i < C$, then $\eta_i = 0$, which means that these are the support vectors. If $\lambda_i = C$, then the first condition tells us that if $\eta_i > 1$ then the classifier made a mistake.

- The problem with this is that it is not as clear how to choose a limited set of vectors, and so most of our training set will be support vectors.

LECTURE 25

Support Vector Machine (SVM)
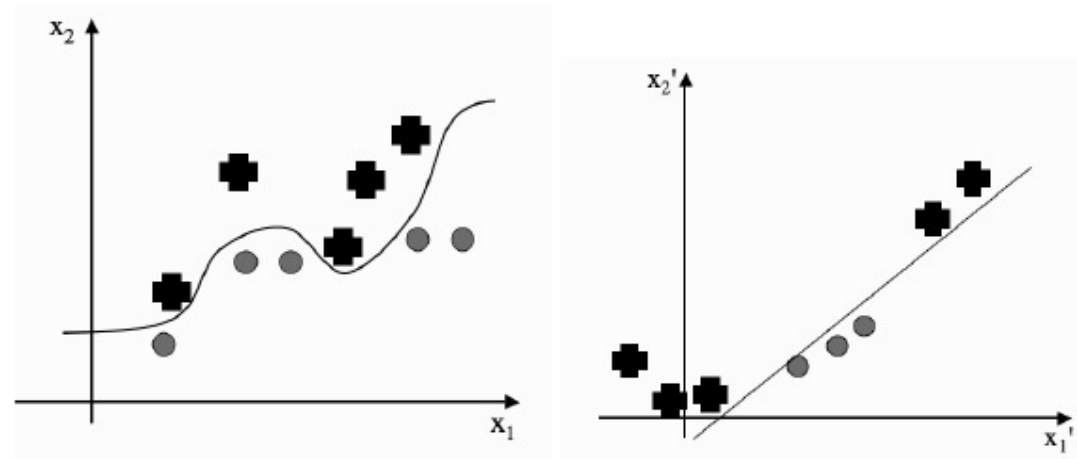
Kernels...

- Transformation of the data

FIGURE 8.5 By modifying the features we hope to find spaces where the data are linearly separable.

Kernels...

- If the features are modified in some way, then able to linearly separate the data, as it was done in XOR problem

- use more dimensions, to find a linear decision boundary that separates the classes.

- So, work out what extra dimensions to use.

- Can't invent new data, so the new features will have to be derived from the current ones in some way.

- Introduce new functions $\phi(x)$ of our input features.

Kernels...

- Just transforming the data, to make some function $\phi(x_i)$ from input x_i.

- Use the SVM algorithm, particularly Equation

$$\mathbf{w}^{*T}\mathbf{z} + b^* = \left(\sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i\right)^T \mathbf{z} + b^*.$$

- Replace xi by $\phi(x_i)$ (and z by $\phi(z)$) and get a prediction quite easily:

$$\mathbf{w}^T\mathbf{x} + b = \left(\sum_{i=1}^{n} \lambda_i t_i \phi(\mathbf{x}_i)\right)^T \phi(\mathbf{z}) + b.$$

Kernels...

- Pick what functions to use?

- Consider the basis that consists of the polynomials of everything up to degree 2.

- For the case d = 3, with a set of $\sqrt{2}$s in there

$$\Phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, x_1^2, x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3). \qquad (8.17)$$

- If there was just one feature, x_1, then change one-dimensional problem into a three-dimensional one as $(1, x_1, x_1^2)$.

Kernels...

- Computational time: the function $\phi(x_i)$ has $d^2/2$ elements, and it must be multiplied with another one of the same size, This must be done many times .

- It is computationally expensive, and it becomes worser if powers greater than 2 is used with input vector.

- Last trick: to avoid computing $\phi(x_i)^T\phi(x_j)$.

$$\Phi(\mathbf{x})^T\Phi(\mathbf{y}) = 1 + 2\sum_{i=1}^{d} x_i y_i + \sum_{i=1}^{d} x_i^2 y_i^2 + 2\sum_{i,j=1;i<j}^{d} x_i x_j y_i y_j.$$

- The above equation can be factorized as $(1+x^Ty)^2$

- The dot product is in the original space, so it only requires d multiplications, which is obviously much better—this part of the algorithm has now been reduced from $O(d^2)$ to $O(d)$.

Kernels

■ It holds true for the polynomials of any degree s , where the **cost of the naïve algorithm is $O(d^s)$.**

■ The important thing is that we remove the problem of computing the dot products of all the extended basis vectors, which is expensive, with the computation of a **kernel matrix (also known as the Gram matrix)**

■ **K** that is made from the dot product of the original vectors, which is only linear in cost. This is sometimes known as the **kernel trick.**

■ These **kernels** are the fundamental reason why these methods work, and the reason why we went for **dual formulation of the problem.**

Choosing Kernels...

How do we go about finding a suitable kernel?

- **Any symmetric function that is positive definite** (meaning that it enforces positivity on the integral of arbitrary functions) **can be used as a kernel.** This is a result of **Mercer's theorem,**

- Also, **convolve kernels together and the result will be another kerne**l.

Choosing Kernels...

■ **There are three different types of basis functions that are commonly used,** and they have nice kernels that correspond to them:

- polynomials up to some degree s in the elements x_k of the input vector (e.g., x_3^3 or $x_1 \times x_4$) with kernel:

$$K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T \mathbf{y})^s \qquad (8.19)$$

  For $s = 1$ this gives a **linear kernel**

- sigmoid functions of the x_ks with parameters κ and δ, and kernel:

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} - \delta) \qquad (8.20)$$

- radial basis function expansions of the x_ks with parameter σ and kernel:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-(\mathbf{x} - \mathbf{y})^2 / 2\sigma^2\right) \qquad (8.21)$$

Choosing Kernels

- Choosing which kernel to use and the parameters in these kernels is a tricky problem.

1. Vapnik–Chernik dimension can be applied,

2. Experiment with different values and find one that works, using a validation set as done for the MLP

Example: XOR

1. Need to modify the problem to have targets -1 and 1 rather than 0 and 1,

2. Introduce a basis of all terms up to quadratic in our two features:

$1, \sqrt{2x_1}, \sqrt{2x_2}, x_1, x_2, x_1^2 \ x_2^2$ where the $\sqrt{2}$ is to keep the multiplications simple.

$$\mathcal{L}(\mathbf{w}^*, b^*, \lambda) = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i t_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j t_i t_j \mathbf{x}_i^T \mathbf{x}_j, \qquad (8.9)$$

The above equation becomes

$$\sum_{i=1}^{4} \lambda_i - \sum_{i,j} \lambda_i \lambda_j t_i t_j \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j),$$

SVM algorithm...

1. Choose a kernel

2. For the given data, **assemble the relevant quadratic problem and its constraints as matrices**,

3. Pass them to the solver, which **finds the decision boundary and necessary support vectors.**

4. These are then used to **build a classifier** for that training data.

SVM algorithm

1. Choose a kernel

2. For the given data, **assemble the relevant quadratic problem and its constraints as matrices**,

3. Pass them to the solver, which **finds the decision boundary and necessary support vectors.**

4. These are then used to **build a classifier** for that training data.

The Support Vector Machine Algorithm

- **Initialisation**

  - for the specified kernel, and kernel parameters, compute the kernel of distances between the datapoints
    * the main work here is the computation $\mathbf{K} = \mathbf{X}\mathbf{X}^T$
    * for the linear kernel, return $\mathbf{K}$, for the polynomial of degree d return $\frac{1}{\sigma}\mathbf{K}^d$
    * for the RBF kernel, compute $\mathbf{K} = \exp(-(\mathbf{x} - \mathbf{x}')^2/2\sigma^2)$

- **Training**

  - assemble the constraint set as matrices to solve:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T t_i t_j \mathbf{K}\mathbf{x} + \mathbf{q}^T\mathbf{x} \text{ subject to } \mathbf{G}\mathbf{x} \leq \mathbf{h}, \mathbf{A}\mathbf{x} = b$$

  - pass these matrices to the solver

- identify the support vectors as those that are within some specified distance of the closest point and dispose of the rest of the training data

- compute b^* using equation (8.10)

- **Classification**

  - for the given test data $\mathbf{z}$, use the support vectors to classify the data for the relevant kernel using:

    * compute the inner product of the test data and the support vectors
    * perform the classification as $\sum_{i=1}^n \lambda_i t_i \mathbf{K}(\mathbf{x}_i, \mathbf{z}) + b^*$, returning either the label (hard classification) or the value (soft classification)

Computational Complexity of SVM Algorithm

- Computing the kernel (which is $O(m^2n)$, where m is the number of datapoints and n is the dimensionality), and

- The second part is inside the solver, which must factorise a sum of the kernel matrix and a test matrix at each iteration.

- Factorisation costs $O(m^3)$ in general, and therefore the SVM is very expensive to use for large datasets.

Example :

- In order to see the SVM working, and to identify the differences between the kernels, simple 2D datasets with two classes are considered
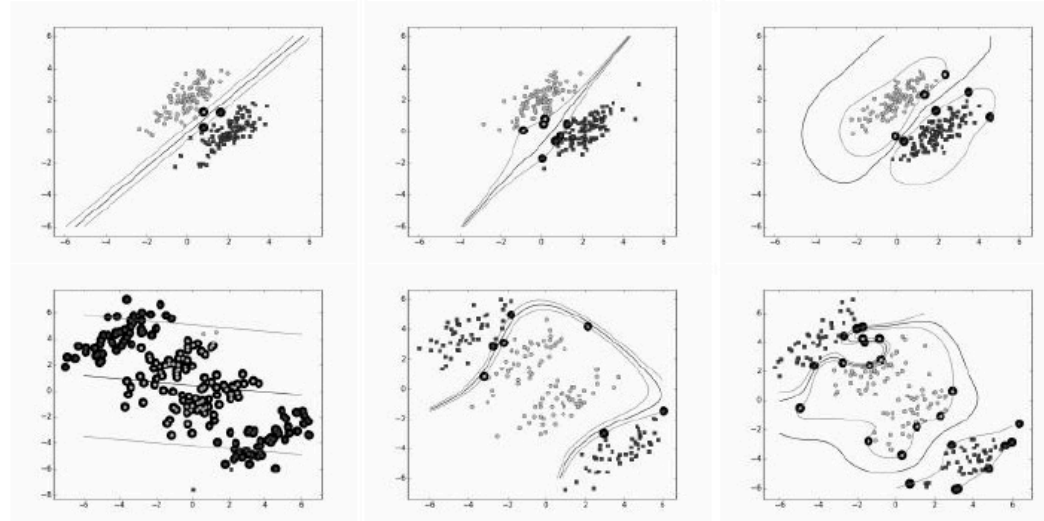
FIGURE 8.7 The SVM learning about a linearly separable dataset (*top row*) and a dataset that needs two straight lines to separate in 2D (*bottom row*) with *left* the linear kernel, *middle* the polynomial kernel of degree 3, and *right* the RBF kernel. $C = 0.1$ in all cases.

Multi-Class Classification

- **How to use SVM for more classes ?**

- The SVM only works for two classes.

- For the problem of N-class classification,
    - *train an SVM that learns to classify class one from all other classes,*
    - *then another that classifies class two from all the others.*
    - *So, for N-classes, we have N SVMs.*

- This still leaves one problem: How do we decide which of these SVMs is the one that recognises the input?
    - *The answer is just to choose the one that makes the strongest prediction*
    - *Classifier return either the class label (as the sign of y) or the value of y*
    - *The value of y is telling us how far away from the decision boundary it is, and clearly it will be negative if it is a misclassification.*
    - *Therefore, **use the maximum value of this soft boundary as the best classifier.***

SVM Regression...

■ It is also possible to use the SVM for regression.

■ The key is to take the usual least-squares error function (with the regulariser that keeps the norm of the weights small):

$$\frac{1}{2} \sum_{i=1}^{N} (t_i - y_i)^2 + \frac{1}{2}\lambda \|\mathbf{w}\|^2,$$

■ and transform it using ϵ-insensitive error function (E_ϵ) that gives 0 if the difference between the target and output is less than ϵ (and subtracts ϵ in any other case for consistency).

SVM Regression...

■ Error Function :

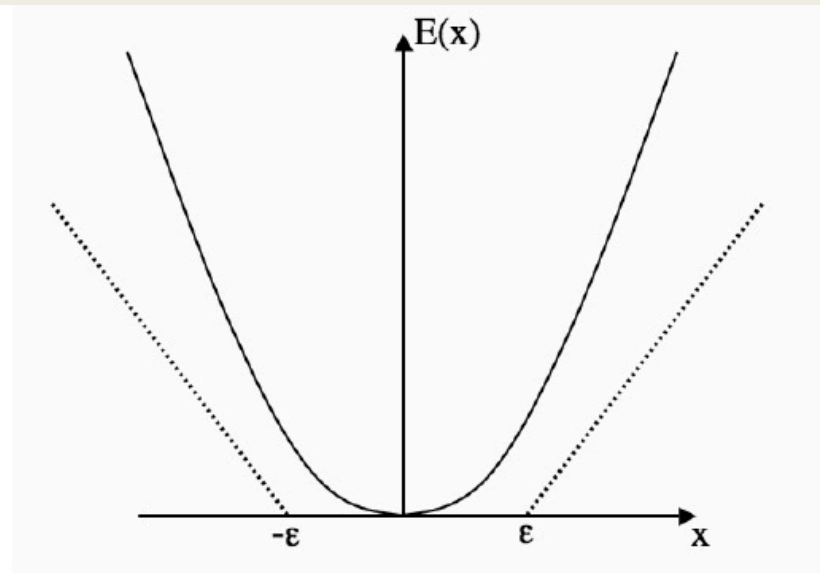$$\sum_{i=1}^{N} E_\epsilon(t_i - y_i) + \lambda\frac{1}{2}\|\mathbf{w}\|^2.$$

FIGURE 8.10 The ϵ-insensitive error function is zero for any error below ϵ.

SVM Regression

- Prediction for test point z is :

$$f(\mathbf{z}) = \sum_{i=1}^{n} (\mu_i - \lambda_i K(\mathbf{x}_i, \mathbf{z}) + b),$$

- where μ_i and λ_i are two sets of constraint variables.

Other Advances

- There is a lot of advanced work on kernel methods and SVMs.

- This includes lots of work on the optimisation, I
  - *Sequential Minimal Optimisation,*
  - *and extensions to compute posterior probabilities instead of hard decisions, such as the Relevance Vector Machine.*

- There are several SVM implementations available via the Internet that are more advanced than the implementation presented in textbook.

- Some common choices are SVMLight, LIBSVM, and scikit-learn(https://scikit-learn.org/stable/modules/svm.html) .
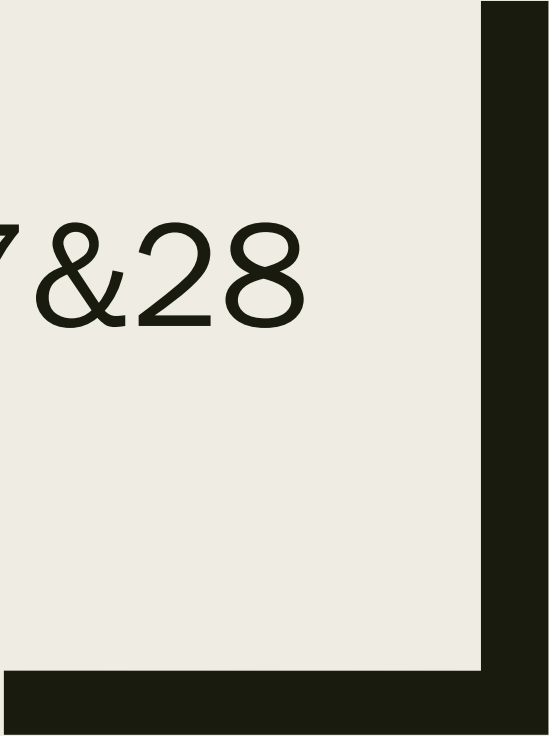
Reference

- Stephan Marsland, **Chapter 8, Machine Learning, An algorithmic Perspective**, CRC Press Second Edition, 2015.

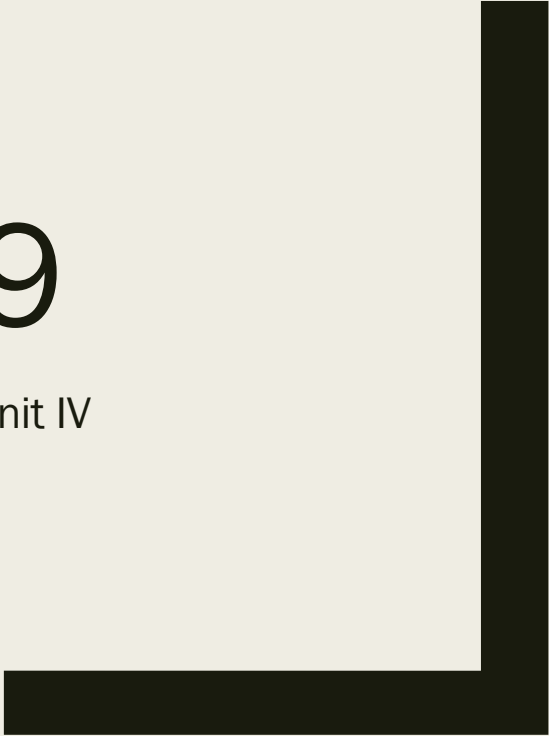# LECTURE 26,27&28

Course Project Development

LECTURE 29

MSE II QP Discussion and Wrapping up Unit IV

Dr.Vani V

Recap

■ **Support Vector Machines :**

- *Optimal Separation*

- *Kernels, SVM Algorithm*

- *Multiclass Classification*

- *SVM Regression*

■ Worthy Read:

https://www.analyticsvidhya.com/blog/2021/05/support-vector-machines/

https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

Recap

- Decision Trees :
  - *Learning with Trees*
  - *Using Decision Trees*
  - *Classification Trees*
  - *Pruning*
  - *Rule Extraction from Trees*
  - *Learning Rules from Data*
  - *Examples*

Univariate Trees

■ Each internal node, the test uses only one of the input dimensions.

■ If the used input dimension, x_j, is discrete, taking one of n possible values, the decision node checks the value of x_j and takes the corresponding branch, implementing an n-way split.

■ Example, if an attribute is color $\in$ {red, blue, green}, then a node on that attribute has three branches, each one corresponding to one of the three possible values of the attribute.

$$f_m(x) : x_j > w_{m0}$$

Univariate Trees

- A decision node has discrete branches, and a numeric input should be discretized. If x_j is numeric (ordered), the test is a comparison

$$f_m(\mathbf{x}) : x_j > w_{m0}$$

w_{m0} - threshold value.

The decision node divides the input space into two: $L_m = \{x | xj > wm_0\}$ and $Rm = \{x | xj \leq w_{m0}\}$; this is called a binary split
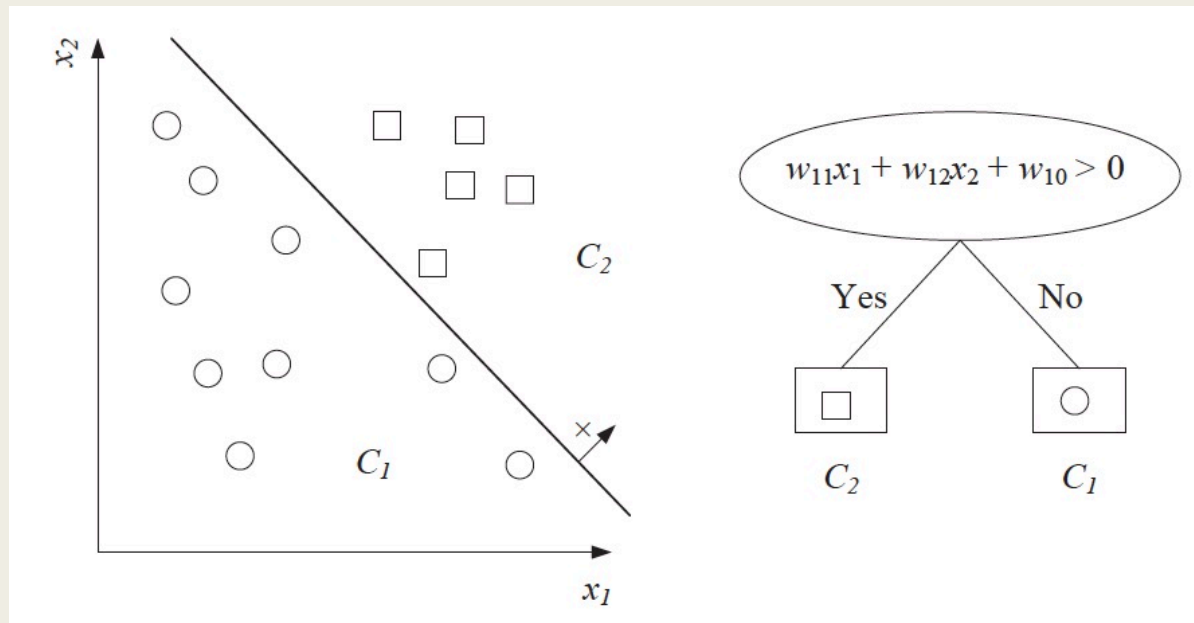
Multivariate Trees

- In the case of a **univariate tree**, only **one input dimension** is used at a split.

- In **a multivariate tree**, at a decision node, **all input dimensions** can be used and thus it is more general.

- When all inputs are numeric, a binary linear multivariate node is defined as

$$f_m(\boldsymbol{x}) : \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} > 0$$

- Because the linear multivariate node takes a weighted sum, discrete attributes should be represented by 0/1 dummy numeric variables.

- The equation defines a hyperplane with arbitrary orientation

Multivariate Trees

Example of a linear multivariate decision tree. The linear multivariate node can place an arbitrary hyperplane and thus is more general, whereas the univariate node is restricted to axis-aligned splits.

Recollection of Decision Tree with impurity measures

- For all attributes, discrete and numeric, and for a numeric attribute for all split positions, we calculate the impurity and choose the one that has the minimum entropy,

- Then, tree construction continues recursively and in parallel for all the branches that are not pure, until all are pure. This is the basis of the classification and

- **Classification And Regression Tree (CART) algorithm (Breiman et al. 1984),**

- **Regression Tree ID3 algorithm (Quinlan 1986), and**

- **Its extension C4.5 (Quinlan 1993).**

Recollection of Decision Tree with impurity measures

- The ID3 , C4.5 pseudocode of the algorithm is given in Figure 1.

- It can also be said that at each step during tree construction, we choose the split that causes the largest decrease in impurity, which is the difference between the impurity of data reaching node m and the total entropy of data reaching its branches after the split

```
GenerateTree(X)
    If NodeEntropy(X)< θ_I /* equation 9.3 */
        Create leaf labelled by majority class in X
        Return
    i ← SplitAttribute(X)
    For each branch of x_i
        Find X_i falling in branch
        GenerateTree(X_i)

SplitAttribute(X)
    MinEnt← MAX
    For all attributes i = 1,...,d
        If x_i is discrete with n values
            Split X into X_1,...,X_n by x_i
            e ← SplitEntropy(X_1,...,X_n) /* equation 9.8 */
            If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i is numeric */
            For all possible splits
                Split X into X_1,X_2 on x_i
                e←SplitEntropy(X_1,X_2)
                If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

Regression Trees

- A regression tree is constructed in almost the same manner as a classification tree, except that the impurity measure that is appropriate for classification is replaced by a measure appropriate for regression.

- Let us say for node m, X_m is the subset of X reaching node m.

- It is the set of all x ∈ X satisfying all the conditions in the decision nodes on the path from the root until node m.

Additional Reading : https://arifromadhan19.medium.com/regrssion-in-decision-tree-a-step-by-step-cart-classification-and-regression-tree-196c6ac9711e

Regression Trees

- Define

$$b_m(x) = \begin{cases} 1 & \text{if } x \in X_m : x \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$$

- In regression, the goodness of a split is measured by the mean square error from the estimated value. Let us say g_m is the estimated value in node m.

$$E_m = \frac{1}{N_m} \sum_t (r^t - g_m)^2 b_m(x^t)$$

Regression Trees

- In a node, use the mean (median if there is too much noise) of the required outputs of instances reaching the node

$$g_m = \frac{\sum_t b_m(\boldsymbol{x}^t) r^t}{\sum_t b_m(\boldsymbol{x}^t)}$$

- Then equation E_m corresponds to the variance at m.

- If at a node, the error is acceptable, that is, Em < θr , then a leaf node is created, and it stores the g_m value.

- If the error is not acceptable,
  - *data reaching node m is split further such that the sum of the errors in the branches is minimum.*
  - *As in classification, at each node, we look for the attribute (and split threshold for a numeric attribute) that minimizes the error, and then we continue recursively.*

References

1. Stephan Marsland, **Chapter 8**, **Machine Learning, An algorithmic Perspective**, CRC Press Second Edition, 2015.

2. Ethem Alpaydin, **Chapter 9 Introduction to Machine Learning**, 2nd Ed., PHI Learning Pvt. Ltd., 2013.