

Classification

Alternative methods and Advanced Classification

Nicolás García-Pedrajas

Computational Intelligence and Bioinformatics Research Group

November 1, 2021



Table of contents

Rule-based classifier

Support vector machines

Other methods



Rule-based classifier



Rule-based classifier (Weiss and Indurkha, 1995)

- ➡ Classify records by using a collection of “if . . . then . . .” rules
- ➡ Rule: (Condition) $\longrightarrow y$
 - ✓ where
 - Condition is a conjunction of attributes
 - y is the class label
 - ✓ LHS: rule antecedent or condition
 - ✓ RHS: rule consequent
- ➡ Examples of classification rules:
 - ✓ (Blood Type=Warm) and (Lay Eggs=Yes) \longrightarrow Birds
 - ✓ (Taxable Income < 50K) and (Refund=Yes) \longrightarrow Evade=No



Application of Rule-Based Classifier

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

Rule set

$R_1: (\text{Give Birth} = \text{no}) \wedge (\text{Can Fly} = \text{yes}) \longrightarrow \text{Birds}$

$R_2: (\text{Give Birth} = \text{no}) \wedge (\text{Live in Water} = \text{yes}) \longrightarrow \text{Fishes}$

$R_3: (\text{Give Birth} = \text{yes}) \wedge (\text{Blood Type} = \text{warm}) \longrightarrow \text{Mammals}$

$R_4: (\text{Give Birth} = \text{no}) \wedge (\text{Can Fly} = \text{no}) \longrightarrow \text{Reptiles}$

$R_5: (\text{Live in Water} = \text{sometimes}) \longrightarrow \text{Amphibians}$

Name	Blood type	Give birth	Can fly	Live in water	Class
hawk	warm	no	yes	no	?
grizzly bear	warm	yes	no	no	?

The rule R_1 covers a hawk \longrightarrow Bird

The rule R_3 covers the grizzly bear \longrightarrow Mammal



Rule coverage and accuracy

- Rule $r : A \longrightarrow y$ in a dataset D
- Coverage: Fraction of records that satisfy the antecedent of a rule

$$\text{Coverage}(D) = \frac{|A|}{|D|}$$

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

$$\text{Accuracy}(D) = \frac{|A \cap y|}{|A|}$$

- $(\text{Status} = \text{Single}) \longrightarrow \text{No}$

✓ Coverage = 40%

✓ Accuracy = 50%

	Refund	Marital Status	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

How does Rule-based Classifier Work?

Rule set

R1: (Give Birth = no) \wedge (Can Fly = yes) \longrightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \longrightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \longrightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \longrightarrow Reptiles

R5: (Live in Water = sometimes) \longrightarrow Amphibians

Name	Blood type	Give birth	Can fly	Live in water	Class
lemur	warm	yes	no	no	?
turtle	cold	no	no	sometimes	?
dogfish	cold	yes	no	yes	?



How does Rule-based Classifier Work?

Rule set

R1: (Give Birth = no) \wedge (Can Fly = yes) \longrightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \longrightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \longrightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \longrightarrow Reptiles

R5: (Live in Water = sometimes) \longrightarrow Amphibians

Name	Blood type	Give birth	Can fly	Live in water	Class
lemur	warm	yes	no	no	Mammal
turtle	cold	no	no	sometimes	?
dogfish	cold	yes	no	yes	?

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



How does Rule-based Classifier Work?

Rule set

R1: (Give Birth = no) \wedge (Can Fly = yes) \longrightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \longrightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \longrightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \longrightarrow Reptiles

R5: (Live in Water = sometimes) \longrightarrow Amphibians

Name	Blood type	Give birth	Can fly	Live in water	Class
lemur	warm	yes	no	no	Mammal
turtle	cold	no	no	sometimes	Reptile/amphibian
dogfish	cold	yes	no	yes	?

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

➡ A turtle triggers both R4 and R5: A **criterion** must be established



How does Rule-based Classifier Work?

Rule set

R₁: (Give Birth = no) \wedge (Can Fly = yes) \longrightarrow Birds

R₂: (Give Birth = no) \wedge (Live in Water = yes) \longrightarrow Fishes

R₃: (Give Birth = yes) \wedge (Blood Type = warm) \longrightarrow Mammals

R₄: (Give Birth = no) \wedge (Can Fly = no) \longrightarrow Reptiles

R₅: (Live in Water = sometimes) \longrightarrow Amphibians

Name	Blood type	Give birth	Can fly	Live in water	Class
lemur	warm	yes	no	no	Mammal
turtle	cold	no	no	sometimes	Reptile/amphibian
dogfish	cold	yes	no	yes	Unknown

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

➡ A turtle triggers both R₄ and R₅: A **criterion** must be established

➡ A dogfish triggers none of the rules: Classification is **unknown**

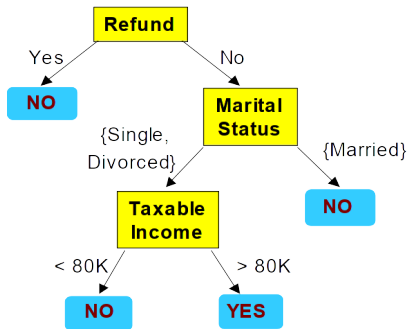


Characteristics of Rule-Based Classifier

- ➡ 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 tree to rules



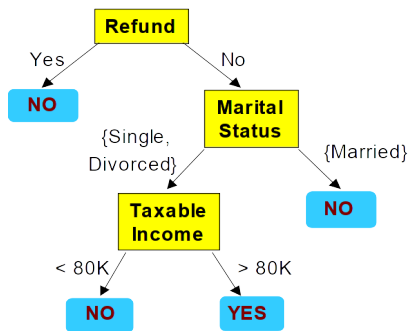
Classification Rules

1. (Refund=Yes) \longrightarrow No
2. (Refund=No, Marital Status={Single, Divorced}, Income<80K) \longrightarrow No
3. (Refund=No, Marital Status={Single, Divorced}, Income>80K) \longrightarrow Yes
4. (Refund=No, Marital Status={Married}) \longrightarrow No

⇒ Rules are mutually exclusive and exhaustive

⇒ Rule set contains as much information as the tree

Rules can be simplified



	Refund	Marital Status	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

Initial Rule: $(\text{Refund}=\text{No}) \wedge (\text{Status}=\text{Married}) \longrightarrow \text{No}$

Simplified Rule: $(\text{Status}=\text{Married}) \longrightarrow \text{No}$

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 with a **voting** scheme
- ➡ Rules are no longer exhaustive
- ➡ A record may not trigger any rules
 - ✓ Solution
 - Use a default class



From rule set to decision tree

➡ Consider the rule set

- ✓ Attributes A, B, C, and D can have values 1, 2, and 3

$A = 1 \wedge B = 1 \longrightarrow \text{Class} = Y$

$C = 1 \wedge D = 1 \longrightarrow \text{Class} = Y$

Otherwise, Class = N

➡ How to represent it as a decision tree?

- ✓ The rules need a common attribute

$A = 1 \wedge B = 1 \longrightarrow \text{Class} = Y$

$A = 1 \wedge B = 2 \wedge C = 1 \wedge D = 1 \longrightarrow \text{Class} = Y$

$A = 1 \wedge B = 3 \wedge C = 1 \wedge D = 1 \longrightarrow \text{Class} = Y$

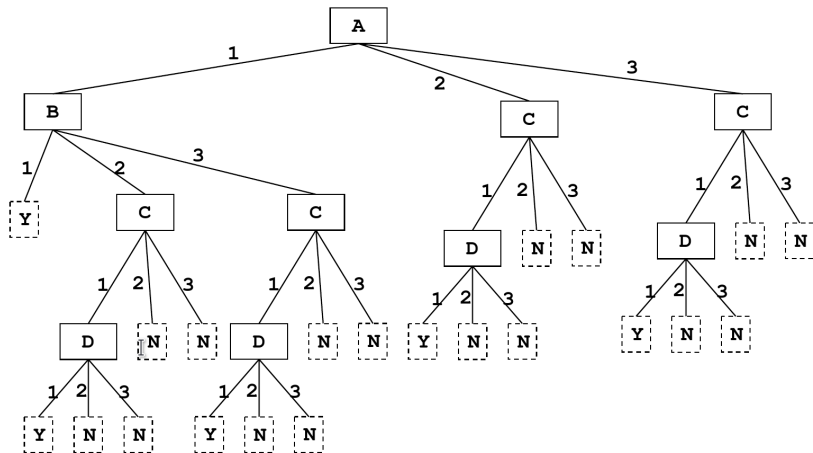
$A = 2 \wedge C = 1 \wedge D = 1 \longrightarrow \text{Class} = Y$

$A = 3 \wedge C = 1 \wedge D = 1 \longrightarrow \text{Class} = Y$

Otherwise, Class = N



From rule set to decision tree



Ordered rule set

- ➡ Rules are rank ordered according to their priority
 - ✓ An ordered rule set is known as a decision list
- ➡ When a test record is presented to the classifier
 - ✓ It is assigned to the class label of the highest ranked rule it has triggered
 - ✓ If none of the rules fired, it is assigned to the default class



Ordered rule set

Rule set

R1: (Give Birth = no) \wedge (Can Fly = yes) \longrightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \longrightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \longrightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \longrightarrow Reptiles

R5: (Live in Water = sometimes) \longrightarrow Amphibians

Name	Blood type	Give birth	Can fly	Live in water	Class
turtle	cold	no	no	sometimes	?



Ordered rule set

Rule set

R1: (Give Birth = no) \wedge (Can Fly = yes) \longrightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \longrightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \longrightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \longrightarrow Reptiles

R5: (Live in Water = sometimes) \longrightarrow Amphibians

Name	Blood type	Give birth	Can fly	Live in water	Class
turtle	cold	no	no	sometimes	R4: Reptile



Ordered rule set

Rule set

R1: (Give Birth = no) \wedge (Can Fly = yes) \longrightarrow Birds

R2: (Give Birth = no) \wedge (Live in Water = yes) \longrightarrow Fishes

R3: (Give Birth = yes) \wedge (Blood Type = warm) \longrightarrow Mammals

R4: (Give Birth = no) \wedge (Can Fly = no) \longrightarrow Reptiles

R5: (Live in Water = sometimes) \longrightarrow Amphibians

Name	Blood type	Give birth	Can fly	Live in water	Class
turtle	cold	no	no	sometimes	R5: Amphibian



Ordered rule set

Rule set

R₁: (Give Birth = no) \wedge (Can Fly = yes) \longrightarrow Birds

R₂: (Give Birth = no) \wedge (Live in Water = yes) \longrightarrow Fishes

R₃: (Give Birth = yes) \wedge (Blood Type = warm) \longrightarrow Mammals

R₄: (Give Birth = no) \wedge (Can Fly = no) \longrightarrow Reptiles

R₅: (Live in Water = sometimes) \longrightarrow Amphibians

Name	Blood type	Give birth	Can fly	Live in water	Class
turtle	cold	no	no	sometimes	Reptile

Unordered rule set

- ➡ Triggered rules cast a (possibly weighted) vote
- ➡ Most voted class is assigned to instance
- ➡ Advantages
 - ✓ Less error prone and sensitive to noise
 - ✓ Faster at training stage as rule ordering is not needed
- ➡ Disadvantages
 - ✓ Slower for the testing stage



Rule ordering schemes

➡ Rule-based ordering

- ✓ Individual rules are ranked based on their quality (a criterion needed)
- ✓ Lower-ranked rules more difficult to interpret: They imply the negation of the previous ones

➡ Class-based ordering

- ✓ Rules that belong to the same class appear together
- ✓ Rules are collectively sorted by group of classes (internal order irrelevant)
- ✓ Most common



Rule ordering schemes

Rule-based ordering

1. (Refund=Yes) \longrightarrow No
2. (Refund=No, Marital Status={Single,Divorced}, Income<80K) \longrightarrow No
3. (Refund=No, Marital Status={Single,Divorced}, Income>80K) \longrightarrow Yes
4. (Refund=No, Marital Status={Married}) \longrightarrow No

Class-based ordering

1. (Refund=Yes) \longrightarrow No
2. (Refund=No, Marital Status={Single,Divorced}, Income<80K) \longrightarrow No
3. (Refund=No, Marital Status={Married}) \longrightarrow No
4. (Refund=No, Marital Status={Single,Divorced}, Income>80K) \longrightarrow Yes



Building Classification Rules

➡ Direct Method

- ✓ Extract rules directly from data
- ✓ Usually greedy algorithms
- ✓ E.g.: RIPPER, CN2, Holte's rR

➡ Indirect Method

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

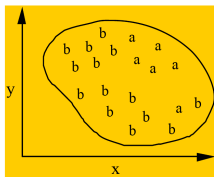


Direct Method: Sequential Covering

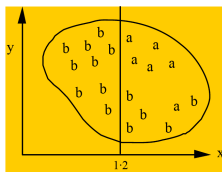
```
Let E be the training records
Let A be the set of attribute-value pairs  $\{(A_j, v_j)\}$ 
Let  $Y_0$  be an ordered set of classes  $\{y_1, y_2, \dots, y_k\}$ 
Let  $R = \{\}$  be the initial rule list
for each class  $y$  in  $Y_0 - \{y_k\}$  do
    while stopping condition is not met do
         $r = \text{Learn-One-Rule}(E, A, y)$ 
        Remove training records from E that are covered
            by  $r$ 
        Add  $r$  to the bottom of the rule list:  $R = R + r$ 
    end while
end for
Insert the default rule,  $\{\} \rightarrow y_k$ , to the bottom of the
rule list R
```



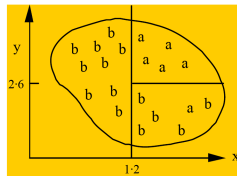
Generating a rule



$$() \longrightarrow a$$



$$(x > 1.2) \longrightarrow a$$



$$(x > 1.2 \wedge y > 2.6) \longrightarrow a$$

➡ Possible rule set for class “b”

$$(x \leq 1.2) \longrightarrow b$$

$$(x > 1.2 \wedge y \leq 2.6) \longrightarrow b$$

➡ Could add more rule, get “perfect” rule set

Learn-one-rule algorithm

```

Learn-One-Rule(target_attribute , attributes , examples , k
)
# Returns a single rule that covers some of the Examples
best-hypothesis = the most general hypothesis
candidate-hypotheses = {best-hypothesis}
while candidate-hypothesis ;Generate the next more
specific candidate-hypotheses
    all-constraints = all "att.=val." constraints
    new-candidate-hypotheses = all specializations of
        candidate-hypotheses by adding all-constraints
    remove from new-candidate-hypotheses duplicates ,
        inconsistent , or not maximally specific
# Update best-hypothesis
best-hypothesis = argmax h in new-candidate-
    hypotheses Performance(h, examples ,
        target_attribute)
# Update candidate-hypotheses
candidate-hypotheses = the k best from new-candidate-
    -hypotheses according to Performance.
prediction = most frequent value of target_attribute
    from examples that match best-hypothesis
return IF best-hypothesis THEN prediction

Performance(h, examples , target_attribute)
h-examples = the set of examples that match h
return - Entropy(h-examples) wrt target_attribute
    
```



Rule extraction

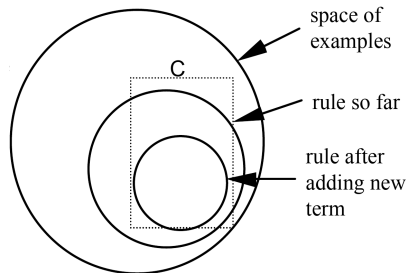
- ➡ One rule $A \longrightarrow y$ is desirable if
 - ✓ Covers most of the positive examples of class y
 - ✓ Covers none, or just a few, negative examples
- ➡ Finding an optimal rule is computationally expensive
 - ✓ The search space grows exponentially
- ➡ Approximate (greedy) solutions



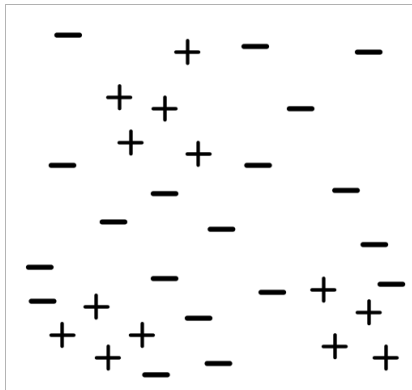
Simple covering algorithm

- ➡ Goal: Choose a test that improves a quality measure for the rules
 - ✓ E.g. maximize rule's accuracy
- ➡ Similar to situation in decision trees: problem of selecting an attribute to split on
 - ✓ Decision tree algorithms maximize overall purity
- ➡ Each new test reduces rule's coverage:

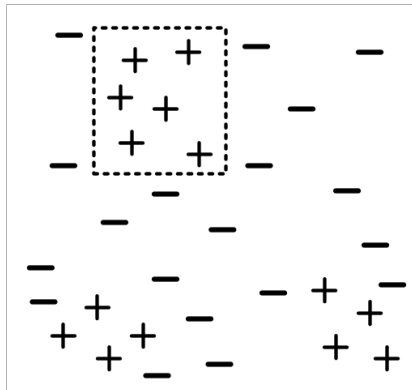
t : number of instances covered by rule
 p : positive examples of the class predicted by rule
 $t - p$: number of errors made by rule
 Rules accuracy = p/t



Sequential covering

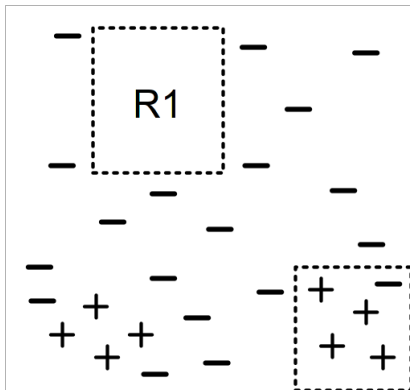


Original data

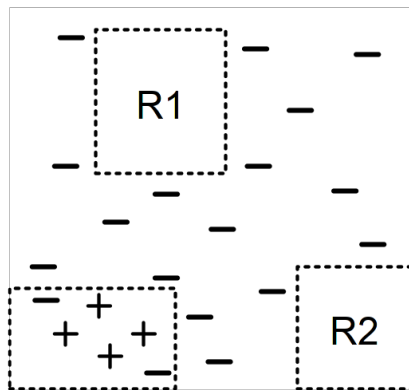


Step 1

Sequential covering



Step 3



Step 4

Aspects of Sequential Covering

- Rule Growing
- Instance Elimination
- Rule Evaluation
- Stopping Criterion
- Rule pruning



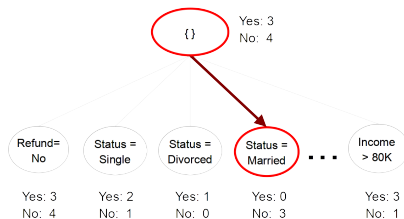
Rule growing

- ➡ Two common strategies
 - ✓ General to specific
 - ✓ Specific to general



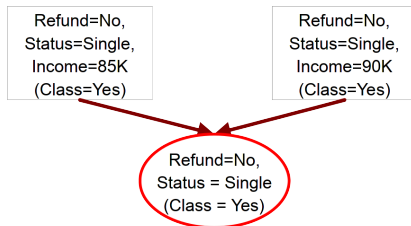
General-to-specific

- Initial rule $r : \{\} \rightarrow y$ is created
- Greedy strategy is used to add new conjuncts



Specific-to-general

- One positive example is randomly chosen
 - ✓ A new rule is created from it
- The rule is refined removing the conjunct that cover more positive examples
- The procedure is continued until the stopping criterion is met



Rule growing: CN2

1. Start from an empty conjunct: $\{\}$
2. Add conjuncts that minimizes the entropy measure: $\{A\}, \{A, B\}, \dots$
3. Determine the rule consequent by taking majority class of instances covered by the rule



Rule growing: RIPPER

Start from an empty rule: $\{\} \rightarrow \text{class}$

Add conjuncts that maximizes FOIL information gain measure:

$R_0: \{\} \rightarrow \text{class}$ (initial rule)

$R_1: \{A\} \rightarrow \text{class}$ (rule after adding conjunct)

$\text{Gain}(R_0, R_1) = t \left[\log(p_1/(p_1+n_1)) - \log(p_0/(p_0 + n_0)) \right]$

where t : number of positive instances covered by both R_0 and R_1

p_0 : number of positive instances covered by R_0

n_0 : number of negative instances covered by R_0

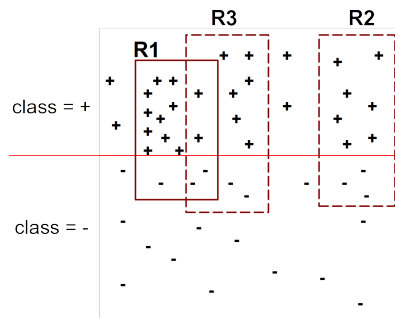
p_1 : number of positive instances covered by R_1

n_1 : number of negative instances covered by R_1



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 a bad estimation of the accuracy of a rule
 - ✓ Compare rules R2 and R3 in the diagram



Removing instances

- ➡ If positive instances are not removed
 - ✓ Next rule effective accuracy may be over-estimated
- ➡ If negative instances are not removed
 - ✓ Next rule effective accuracy may be under-estimated



Rule evaluation

$$\Rightarrow \text{Accuracy} = \frac{n_c}{n}$$

$$\Rightarrow \text{Laplace} = \frac{n_c + 1}{n + k}$$

$$\Rightarrow \text{M-estimate} = \frac{n_c + kp}{n + k}$$

where:

n : Number of instances covered by rule

n_c : Number of instances covered by rule correctly classified

k : Number of classes

p : Prior probability



Rule evaluation

- ➡ Accuracy does not consider rule's coverage
 - ✓ r_1 : covers 50 positive cases, 5 negative cases
Accuracy = $\frac{50}{55} = 0.91$
 - ✓ r_2 : covers 2 positive cases, 0 negative cases
Accuracy = $\frac{2}{2} = 1.00$



Stopping criterion

- ➡ Compute the gain
- ➡ If gain is not significant, discard the new rule



Rule pruning

- ➡ Similar to post-pruning of decision trees
- ➡ 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



RIPPER

Growing a rule:

1. Start from empty rule
2. Add conjuncts as long as they improve FOIL's information gain
3. Stop when rule no longer covers negative examples
4. Prune the rule immediately using incremental reduced error pruning
5. 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
6. Pruning method: delete any final sequence of conditions that maximizes v



Similar to information gain:

where:

R_0 denotes a rule before adding a new literal

R_1 is an extension of R_0

p_0 denotes the number of positive instances, covered by R_0

p_1 the number of positive instances, covered by R_1

n_0 and n_1 are the number of negative instances, covered by the according rule

t is the number of positive instances, covered by R_0 as well as by R_1

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

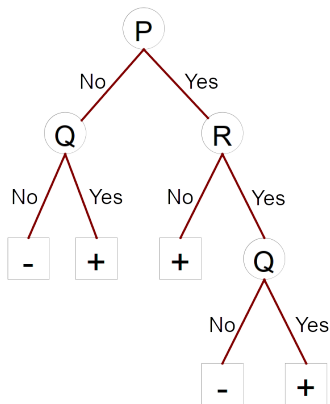


RIPPER

- ➡ 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 principle
 - ✓ Repeat rule generation and rule optimization for the remaining positive examples



Indirect methods



Rule Set

r1: (P=No,Q=No) \longrightarrow -

r2: (P=No,Q=Yes) \longrightarrow +

r3: (P=Yes,R=No) \longrightarrow +

r4: (P=Yes,R=Yes,Q=No) \longrightarrow -

r5: (P=Yes,R=Yes,Q=Yes) \longrightarrow +

Indirect Method: C4.5rules

⇒ Extract rules from an unpruned decision tree

✓ For each rule,

$$r : A \longrightarrow y$$

,

- consider an alternative rule $r' : A' \longrightarrow y$ 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: C4.5rule

- ➡ 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 = $L(error) + gL(model)$
 - g is a parameter that takes into account the presence of redundant attributes in a rule set (default value = 0.5)

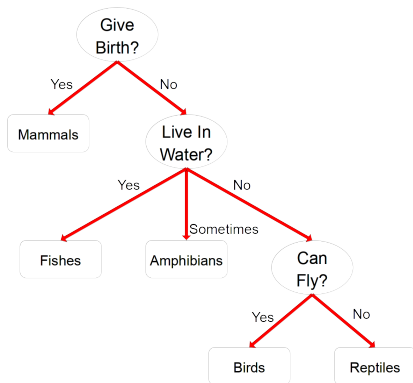


Example

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
 () → Amphibians

RIPPER

(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

C4.5 versus C4.5rules versus RIPPER

C4.5rules		PREDICTED CLASS				
		Amphibians	Fishes	Reptiles	Birds	Mammals
ACTUAL CLASS	Amphibians	2	0	0	0	0
	Fishes	0	2	0	0	1
	Reptiles	1	0	3	0	0
	Birds	1	0	0	3	0
	Mammals	0	0	1	0	6
RIPPER		PREDICTED CLASS				
		Amphibians	Fishes	Reptiles	Birds	Mammals
ACTUAL CLASS	Amphibians	0	0	0	0	2
	Fishes	0	3	0	0	0
	Reptiles	0	0	3	0	1
	Birds	0	0	1	2	1
	Mammals	0	2	1	0	4

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

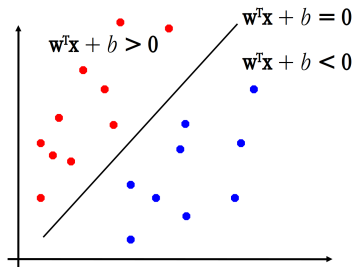


Support vector machines



Perceptron Revisited: Linear Separators

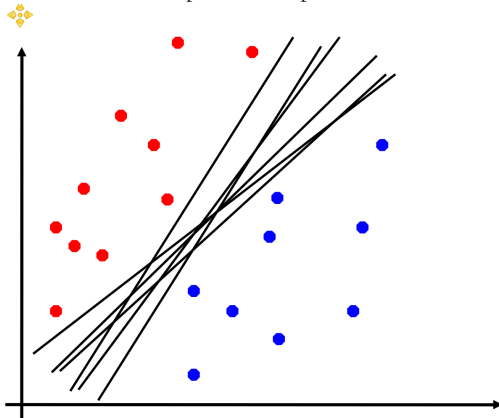
Binary classification can be viewed as the task of separating classes in feature space



$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

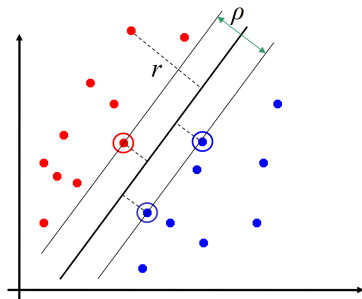
Linear Separators

Which of the linear separators is optimal?



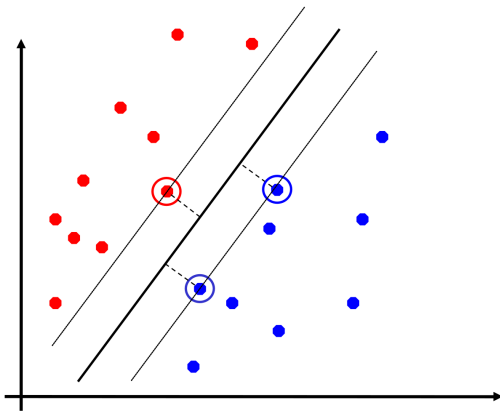
Classification margin

- Distance from example \mathbf{x}_i to the separator is: $r = \frac{w^T \mathbf{x}_i + b}{\|w\|}$
- Examples closest to the hyperplane are **support vectors**
- Margin ρ of the separator is the distance between support vectors



Maximum Margin Classification

- Maximizing the margin is good according to intuition and Probably Approximately Correct learning (PAC) theory.
- Implies that only support vectors matter; other training examples are ignorable.



Linear SVM Mathematically

- ➡ Let training set $\{(x_i, y_i)\}, i = 1, \dots, n, x_i \in R^d, y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ . Then for each training example (x_i, y_i) :

$$\begin{aligned} \mathbf{w}^T \mathbf{x}_i + b &\leq -\rho/2, & \text{if } y_i = -1 \\ \mathbf{w}^T \mathbf{x}_i + b &\leq \rho/2, & \text{if } y_i = 1 \end{aligned} \Leftrightarrow y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq \rho/2$$

- ➡ For every support vector \mathbf{x}_s the above inequality is an equality. After rescaling \mathbf{w} and b by $\rho/2$ in the equality, we obtain that distance between each \mathbf{x}_s and the hyperplane is

$$r = \frac{y_s(\mathbf{w}^T \mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \quad (1)$$

- ➡ Then the margin can be expressed through (rescaled) \mathbf{w} and b as $\rho = 2r = \frac{2}{\|\mathbf{w}\|}$



Linear SVM Mathematically

- ⇒ The we can formulate the quadratic optimization problem

Quadratic optimization problem

Find \mathbf{w} and b such that

$$\rho = \frac{2}{\|\mathbf{w}\|} \text{ is maximized}$$

$$\text{and for all } (x_i, y_i), i = 1, \dots, n : \quad y_i(\mathbf{w})^T \mathbf{x}_i + b \geq 1$$

- ⇒ Which can be reformulated as

Reformulated optimization problem

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w} \text{ is minimized}$$

$$\text{and for all } (x_i, y_i), i = 1, \dots, n : \quad y_i(\mathbf{w})^T \mathbf{x}_i + b \geq 1$$



Solving the Optimization Problem

- ➡ Need to optimize a quadratic function subject to linear constraints
- ➡ Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist
- ➡ The solution involves constructing a dual problem where a Lagrange multiplier α_i is associated with every inequality constraint in the primal (original) problem

Dual problem

Find $\alpha_1, \dots, \alpha_n$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is minimized and

$$(1) \sum \alpha_i y_i = 0$$

$$(2) \alpha_i \geq 0, \forall \alpha_i$$

The Optimization Problem Solution

- ➡ Given a solution $\alpha_1, \dots, \alpha_n$ to the dual problem, solution to the primal is:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i, \quad b = y_k - \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_k, \quad \forall \alpha_k > 0$$

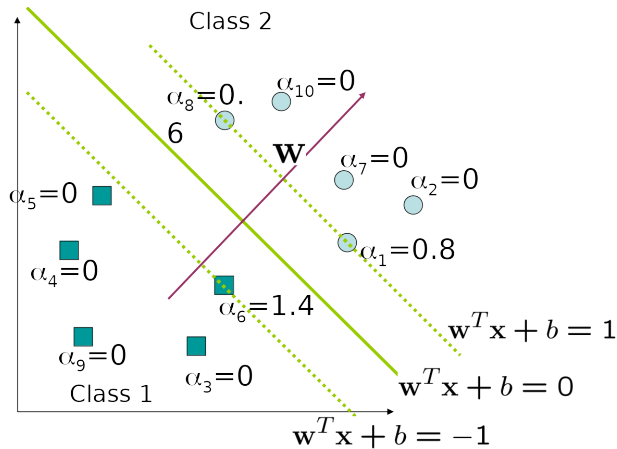
- ➡ Each non-zero α_i indicates that corresponding \mathbf{x}_i is a support vector
- ➡ Then the classifying function is (note that we don't need \mathbf{w} explicitly)

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

- ➡ Notice that it relies on an *inner product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i (we will return to this later)
- ➡ Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^T \mathbf{x}_j$ between all training points

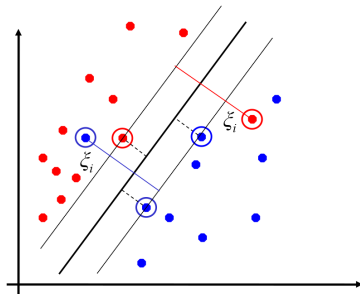


Geometrical Interpretation



Soft Margin Classification

- ➡ What if the training set is not linearly separable?
- ➡ *Slack variables*, ξ_i , can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*



Soft Margin Classification Mathematically

Optimization problem

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w} \text{ is minimized}$$

and for all $(\mathbf{x}_i, y_i), i = 1, \dots, n$: $y_i(\mathbf{w})^T \mathbf{x}_i + b \geq 1$

Modified formulation incorporates slack variables

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w} + C \sum \xi_i \text{ is minimized}$$

and for all $(\mathbf{x}_i, y_i), i = 1, \dots, n$: $y_i(\mathbf{w})^T \mathbf{x}_i + b \geq 1 - \xi_i, \quad \xi_i \geq 0$

- ➡ Parameter C can be viewed as a way to control overfitting
- ➡ “Trades off” the relative importance of maximizing the margin and fitting the training data.



Soft Margin Classification – Solution

- ➡ Dual problem is identical to separable case (would not be identical if the 2-norm penalty for slack variables $C \sum \xi_i^2$ was used in primal objective, we would need additional Lagrange multipliers for slack variables)

Dual problem

Find $\alpha_1, \dots, \alpha_n$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is minimized and

(1) $\sum \alpha_i y_i = 0$, (2) $0 \leq \alpha_i \leq C, \forall \alpha_i$

- ➡ Again, \mathbf{x}_i with non-zero α_i will be support vectors

Solution to the dual problem is

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$$

$$b = y_k(1 - \xi_k) - \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_k, \text{ s. t. } \alpha_k > 0$$

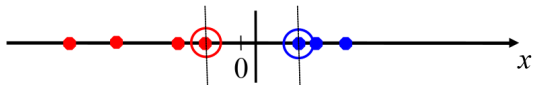
Again, we don't need to compute \mathbf{w} explicitly for classification

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

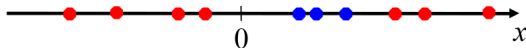


Non-linear SVMs

- ⇒ Datasets that are linearly separable with some noise work out great



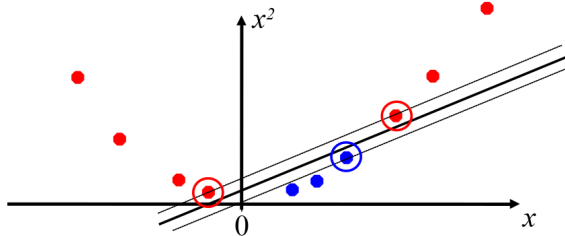
- ⇒ But what are we going to do if the dataset is just too hard?



Non-linear SVMs

How about...

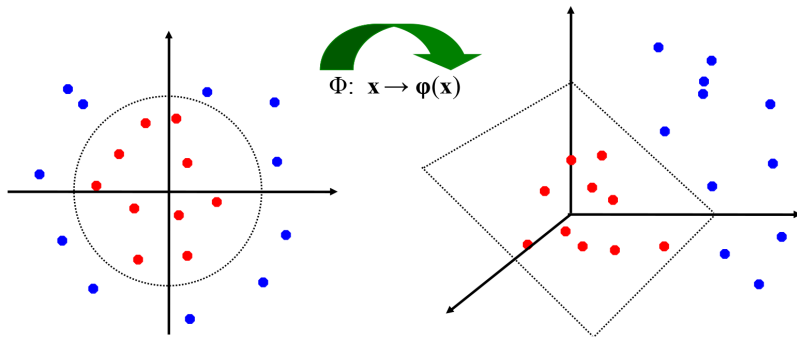
- ✓ mapping data to a higher-dimensional space



Non-linear SVMs: Feature spaces

General idea

The original feature space can always be mapped to some higher-dimensional feature space where the training set is separable



The ``Kernel trick``

- ➡ The linear classifier relies on inner product between vectors $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- ➡ If every datapoint is mapped into high-dimensional space via some transformation $\Phi : \mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$$

- ➡ A kernel function is a function that is equivalent to an inner product in some feature space
- ➡ Example: 2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$, let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$
Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$
$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2 = 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} = \\ &= [1 \ x_{i1}^2 \ \sqrt{2x_{i1}x_{i2}} \ x_{i2}^2 \ \sqrt{2x_{i1}} \ \sqrt{2x_{i2}}]^T [1 \ x_{j1}^2 \ \sqrt{2x_{j1}x_{j2}} \ x_{j2}^2 \ \sqrt{2x_{j1}} \ \sqrt{2x_{j2}}] = \\ &= \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j), \text{ where } \varphi(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2x_1x_2} \ x_2^2 \ \sqrt{2x_1} \ \sqrt{2x_2}] \end{aligned}$$
- ➡ Thus, a kernel function implicitly maps data to a high-dimensional space (without the need to compute each $\varphi(\mathbf{x})$ explicitly)



What functions are kernels?

- ➡ For some functions $K(x_i, x_j)$ checking that $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$ can be cumbersome

Mercer's theorem

Every semi-positive definite symmetric function is a kernel

- ➡ Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix \mathbf{K}

$$\mathbf{K} = \begin{Bmatrix} K(x_1, x_1) & K(x_1, x_2) & K(x_1, x_3) & \dots & K(x_1, x_n) \\ K(x_2, x_1) & K(x_2, x_2) & K(x_2, x_3) & \dots & K(x_2, x_n) \\ \dots & \dots & \dots & \dots & \dots \\ K(x_n, x_1) & K(x_n, x_2) & K(x_n, x_3) & \dots & K(x_n, x_n) \end{Bmatrix}$$

- ➡ $K(x_i, x_i) \geq 0$ and $K(x_i, x_j) = K(x_j, x_i)$



Examples of Kernel functions

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$

 - ✓ Mapping $\Phi : \mathbf{x} \rightarrow \varphi(\mathbf{x})$, where $\varphi(\mathbf{x}) = \mathbf{x}$
- Polynomial of power p : $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$

 - ✓ Mapping $\Phi : \mathbf{x} \rightarrow \varphi(\mathbf{x})$, where $\varphi(\mathbf{x})$ has $\binom{d+p}{p}$ dimensions
- Gaussian (radial-basis function): $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$

 - ✓ Mapping $\Phi : \mathbf{x} \rightarrow \varphi(\mathbf{x})$, where $\varphi(\mathbf{x})$ is infinite-dimensional: every point is mapped to a function (a Gaussian); combination of functions for support vectors is the separator
- Higher-dimensional space still has intrinsic dimensionality d (the mapping is not onto), but linear separators in it correspond to non-linear separators in original space.



Non-linear SVMs Mathematically

Non-linear dual problem

Find $\alpha_1, \dots, \alpha_n$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is minimized and

$$(1) \sum \alpha_i y_i = 0$$

$$(2) \alpha_i \geq 0, \forall \alpha_i$$

Solution

$$f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

⇒ Optimization techniques for finding α_i 's remain the same

String kernels(Leslie and Kuang, 2004; Sonnenburg et al., 2007)

- kernel is a distance function so it can be used to incorporate expert knowledge
Classification of genomic sequences
 $x_1 = TCCTCCCAAGGTAGAAACAAATAACTTCTGGGGTAGAAAGCATTTTCAG$
 $x_2 = TAGGAGAAACCCTCCTCAATTACGTTAAAATAGGACCCCCTCCCTCTGT$
- Distances such as Euclidean make no sense, the distance must be based in alignments or common subsequences
- Kernel functions can do that



String kernels

Weighted degree kernel(Rätsch, Sonnenburg, and Schäfer, 2006)

Efficiently computes similarities between sequences while taking positional information of multiple k -mers into account

The WD kernel of order d compares two sequences \mathbf{x} and \mathbf{x}' of equal length l by summing all contributions of k -mer matches of lengths $k \in \{1, \dots, d\}$, weighted by coefficients β_k

$$k(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^d \beta_k \sum_{i=1}^{l-k+1} \mathbf{I}(\mathbf{u}_{k,i}(\mathbf{x}) = \mathbf{u}_{k,i}(\mathbf{x}'))$$

$\mathbf{u}_{k,i}(\mathbf{x})$ is the string of length k starting at position i of the sequence \mathbf{x}

$\mathbf{I}(\cdot)$ is the indicator function which is equal to 1 when its argument is true and to 0 otherwise

$$k(\mathbf{s}_1, \mathbf{s}_2) = w_7 + w_1 + w_2 + w_2 + w_3$$

$\mathbf{s}_1 \rightarrow$ AGTCAGATAGAGGACATCAGTAGACAGATTAAA \rightarrow
 $\mathbf{s}_2 \rightarrow$ TTATAGATAGACAAAGACATCAGTAGACTTATT \rightarrow



Weighted degree kernel with shifts(Rätsch, Sonnenburg, and Schäfer, 2006)

$$k(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^d \beta_k \sum_{i=1}^{l-k+1} \sum_{\substack{s=0 \\ s+i \leq l}} \delta_s \mu_{k,i,s,\mathbf{x},\mathbf{x}'}$$

where $\beta_k = 2(d - k + 1)/(d(d + 1))$ and $\delta_s = 1/2(s + 1)$



SVM applications

- ➡ SVMs were originally proposed by Boser, Guyon, and Vapnik (1992) and gained increasing popularity in late 1990s
- ➡ SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- ➡ SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- ➡ SVM techniques have been extended to a number of tasks such as regression, principal component analysis, etc.
- ➡ Most popular optimization algorithms for SVMs use decomposition to hill-climb over a subset of α_i 's at a time
- ➡ Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.



SVM practical recommendations

- ➡ When hyper-parameters are appropriately tuned one of the best classifiers
- ➡ Without cross-validating parameters can be very bad
- ➡ Specially sensitive to C
- ➡ Very good for large feature spaces
- ➡ Computationally very expensive for many instances



Other methods









Other methods

- ➡ Instance-based learners
 - ✓ Rote-learner
 - ✓ k -nearest neighbors
 - ✓ PEBLS: Parallel Exemplar-Based Learning System (Cost and Salzberg, 1993)
- ➡ Bayes classifier and Naïve Bayes classifier
- ➡ Artificial neural networks

These methods are studied in other subjects



References I

-  Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). 'A training algorithm for optimal margin classifiers'. In: *Proceedings of the 5th Annual ACM Workshop on COLT*. Ed. by D. Haussler, pp. 144–152.
-  Cost, S. and S. Salzberg (1993). 'A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features'. In: *Machine Learning* 10.1, pp. 57–78.
-  Leslie, C. and R. Kuang (2004). 'Fast String Kernels using Inexact Matching for Protein Sequences'. In: *Journal of Machine Learning Research* 5, pp. 1435–1455.
-  Rätsch, G., S. Sonnenburg, and C. Schäfer (2006). 'Learning interpretable SVMs for biological sequence classification'. In: *BMC Bioinformatics* 7 (Suppl 1), S9.
-  Sonnenburg, S. et al. (2007). 'Accurate splice site prediction using support vector machines'. In: *BMC Bioinformatics* 8(Suppl 10).S7, pp. 1–16.
-  Weiss, S. M. and N. Indurkha (1995). 'Rule-based Machine Learning Methods for Functional Prediction'. In: *Journal of Artificial Intelligence Research* 3, pp. 383–403.