Nearest Neighbor algorithm. Rule-Based Algorithms: 1R and PRISM

COMP5318 Machine Learning and Data Mining

semester 1, 2021, week 2

Irena Koprinska

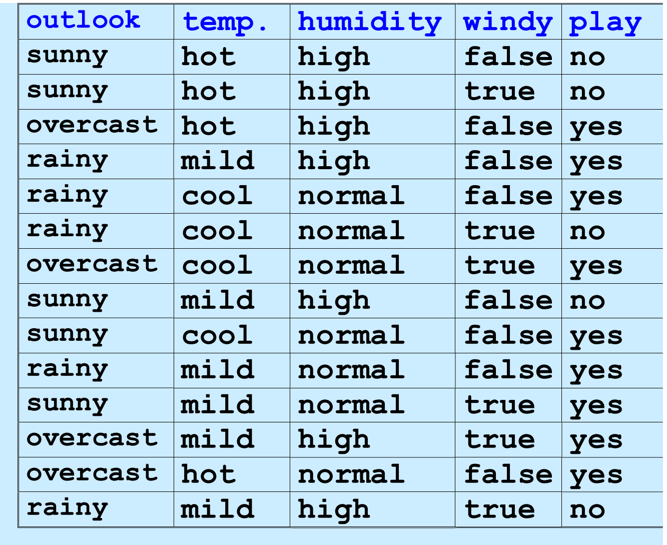
Reference: Witten ch.4: 91-96, 135-141, 115-119; Tan ch.4: 208-212

Outline

- Nearest Neighbor algorithm
- Rule-based algorithms
  - 1R
  - PRISM

Classification task – setup and an example

- Given: a set of *labelled* examples (labelled with the class values)
  - 14 examples
  - 4 attributes (features): *outlook, temperature, humidity* and *windy*
  - class: *play*
- Task: Build a model (classifier) that can be used to predict the class of new (unseen) examples
  - e.g. predict the class (yes or no) of this new example: *outlook=sunny, temp=hot, humidity=low, windy=true*
- Examples used to build the model are called *training set*
- To evaluate performance, we measure the *accuracy* on *test data*
  - Test data hasn’t been used to build the classifier; it is also labelled
  - *accuracy* – proportion of correctly classified test examples



outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Two types of attributes: categorical and numeric

- *categorical (nominal)* - their values belong to a pre-specified, finite set of possibilities
- *numeric (continuous)* - their values are numbers

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

categorical

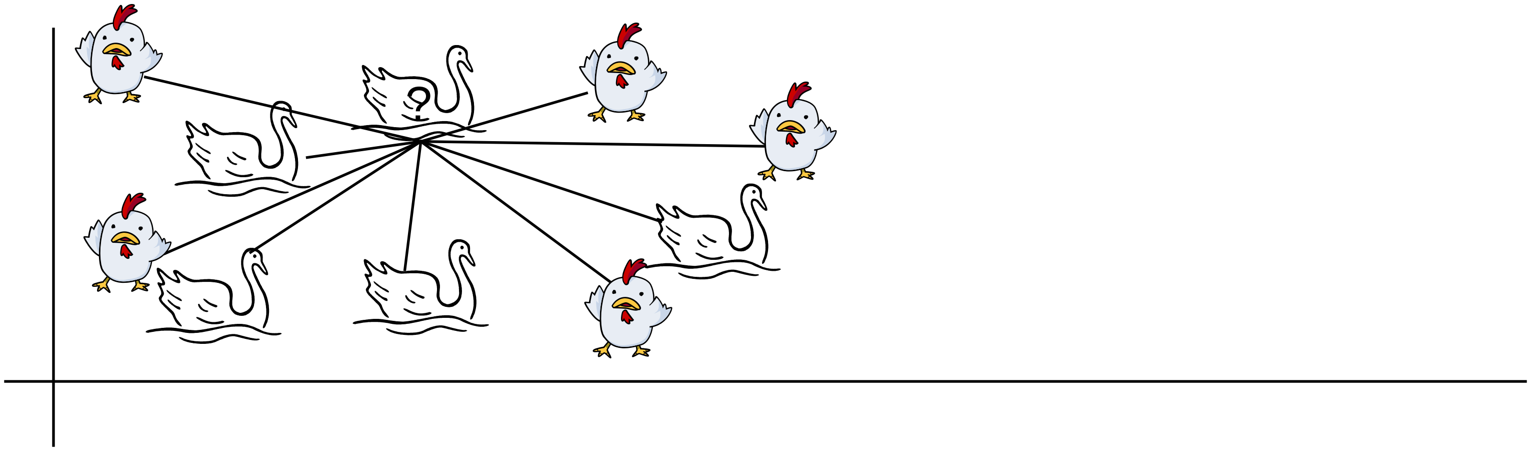
sepal length	sepal width	petal length	petal width	iris type
5.1	3.5	1.4	0.2	iris setosa
4.9	3.0	1.4	0.2	iris setosa
4.7	3.2	1.3	0.2	iris setosa
...				
6.4	3.2	4.5	1.5	iris versicolor
6.9	3.1	4.9	1.5	iris versicolor
5.5	2.3	4.0	1.3	iris versicolor
6.5	2.8	4.6	1.5	iris versicolor
6.3	3.3	6.0	2.5	iris virginica
5.8	2.7	5.1	1.9	iris virginica
...				

numeric

Nearest Neighbor Algorithm

Nearest neighbour algorithm

- Remember all training examples
- To make a prediction for a new unlabelled example:
  - Find the nearest training example to it using a distance measure
  - The class label of the nearest neighbor will be the predicted label for the new example



Distance measures - revision

- Revision from last week – distance measures for numeric attributes

  A, B – examples with attribute values $a_1, a_2, \dots, a_n$ & $b_1, b_{2,} \dots, b_n$

  E.g. A= [1, 3, 5], B=[1, 6, 9]

- Euclidean distance (L2 norm) – most frequently used

$$D(A,B)=\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+\ldots+(a_n-b_n)^2}$$

D(A,B) = sqrt ((1-1)^2+(3-6)^2+(5-9)^2)=5

- Manhattan distance (L1 norm)

$$D(A,B)=|a_1-b_1|+|a_2-b_2|+\ldots+|a_n-b_n|$$

D(A,B)=|1-1|+|3-6|+|5-9|=7

Nearest neighbor - example

- Ex.1 from the tutorial exercises (t2.pdf):

The dataset below consists of 4 examples described with 3 numeric features (a1, a2 and a3); the class has 2 values: yes and no.

What will be the prediction of the Nearest Neighbor algorithm using the Euclidian distance for the following new example: a1=2, a2=4, a3=2?

	a1	a2	a3	class
1	1	3	1	yes
2	3	5	2	yes
3	3	2	2	no
4	5	2	3	no

Exercise adapted from M. Kubat, Introduction to Machine Learning, Springer, 2017

Solution

Need for normalization - revision

- Normalization – attribute transformation to a new range, e.g. [0,1]
- Used to avoid the dominance of attributes with large values over attributes with small values when calculating distance
- Required for distance based ML algorithms such as nearest neighbor
- Example: Predict if a customer is likely to buy a product based on 2 attributes: age and income
  - *age* (in years) and annual *income* (in dollars) have different scales

    A=[20, 40 000]

    B=[40, 60 000]

    D(A,B)=|20-40| + |40 000-60 000|=20 020

    The *income* dominates, the effect of *age* is lost in the distance calculation as its scale is very small

- Solution: First normalize the attribute values, then calculate the distance

Normalization (2)

- Performed for each attribute
- Normalization is also called min-max scaling:

$$x' = \frac{x - \min(\mathrm{x})}{\max(\mathrm{x}) - \min(\mathrm{x})}$$

x – original value

x' – new value

x – all values of the attribute; a vector

min(x) and max(x) – min and max values of the attribute (of the vector x)

Before normalization:

A=[20, 40 000]

B=[40, 60 000]

C=[25, 30 000]

…

Suppose that:

for *age*: min = 0, max=100

for *income*: min=0, max=100 000

After normalization:

A=[0.20, 0.4]

B=[0.40, 0.6]

C=[0.25, 0.3]

…

D(A,B)=|0.2-0.4| + |0.4-0.6|=0.4

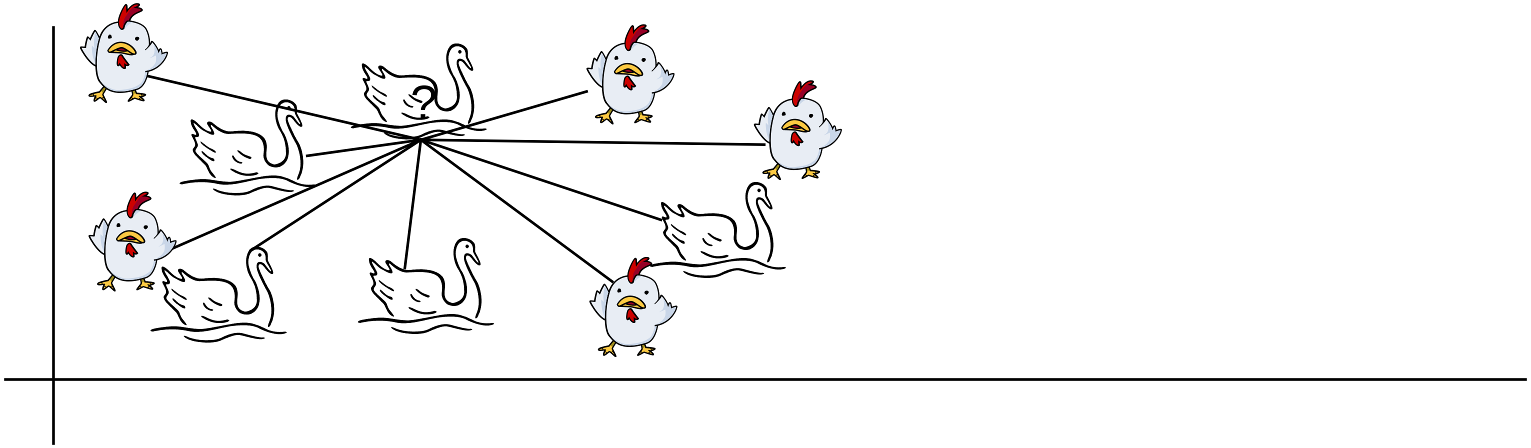
Both *income* and *age* contributing, income do not dominate age as before

Computational complexity

- Training is fast – no model is built; just storing training examples
- Classification (prediction for a new example)
  - Compare each unseen example with each training example
  - If *m* training examples with dimensionality *n* => lookup for 1 unseen example takes *m*n* computations, i.e. *O(mn)*
- Memory requirements – you need to remember each training example, *O(mn)*
- Impractical for large data due to time and memory requirements
- However, there are variations allowing to find the nearest neighbours more efficiently
  - e.g. by using more efficient data structures such as KD-trees and ball trees, see Witten p.136-141

From 1-neareast neighbor to k-nearest neighbor

- Using the closest 1 example (to predict the class of the new example) is called 1-Nearest Neighbor
- Using the k closest examples is called k-Nearest Neighbor
  - majority voting is used to take the decision
- What will be the prediction of 3-Nearest Neighbor?



k-nearest neighbor

- K-Nearest Neighbor is very sensitive to to the value of k
  - rule of thumb: k ≤ sqrt(#training_examples)
  - commercial packages typically use k=10
- Using more nearest neighbors increases the robustness to noisy examples

- K-Nearest Neighbor can be used not only for classification, but also for regression
  - The prediction will be the average value of the class values (numerical) of the k nearest neighbors

k-nearest neighbor - example

- Ex.1 from the tutorial exercises (t2.pdf):

The dataset below consists of 4 examples described with 3 numeric features (a1, a2 and a3); the class has 2 values: yes and no.

What will be the prediction of the 3-Nearest Neighbor algorithm using the Euclidian distance for the following new example: a1=2, a2=4, a3=2?

	a1	a2	a3	class
1	1	3	1	yes
2	3	5	2	yes
3	3	2	2	no
4	5	2	3	no

Solution:

Calculating distance for nominal attributes

- Revision from last week
- Various options depending on the task and type of data type; requires domain expertise, e.g.:
  - difference = 0 if attribute values are the same
  - difference = 1 if they are not

  - Example: 2 attributes = *temperature* and *windy*

    temperature values: low and high

    windy values: yes and no

    ex.1= {high, no}

    ex.2 = {high, yes}

    d(A,B) =$(0+1)^{1/2}$=1 (Euclidean distance)

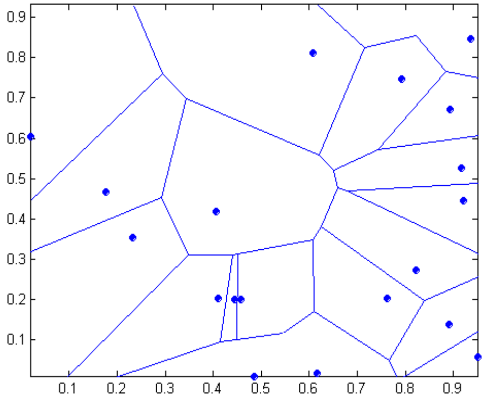
Weighted nearest neighbor

- Idea: Closer neighbors should count more than distant neighbors
- Distance-weighted nearest-neighbor algorithm
  - Find the k nearest neighbors
  - Weight their contribution based on their distance to the new example
    - bigger weight if they are closer
    - smaller weight if they are further
    - e.g. the vote can be weighted according to the distance – weight $w = 1/d^2$

Decision boundary of 1-nearest neighbor

- Nearest neighbor classifiers produced decision boundaries with an arbitrary shape
- The 1-nearest neighbor boundary is formed by the edges of the Voronoi diagram that separate the points of the two classes
- Voronoi region:
  - Each training example has an associated Voronoi region; it contains the data points for which this is the closest example

Voronoi diagram



K-nearest neighbor - discussion

- Often very accurate
- Has been used in statistics since 1950s
- Slow for big datasets
  - requires efficient data structures such as KD-trees
- Distance-based - requires normalization
- Produces arbitrarily shaped decision boundary defined by a subset of the Voronoi edges
- Not effective for high-dimensional data (data with many features)
  - The notion of “nearness” is not effective in high dimensional data
  - Solution – dimensionality reduction and feature selection
- Sensitive to the value of k

Rule-Based Algorithms: 1R

1R

- 1R stands for “1-rule"
- Generates *1 rule* that tests the value of *a single attribute*
  - e.g. a rule that tests the value of outlook

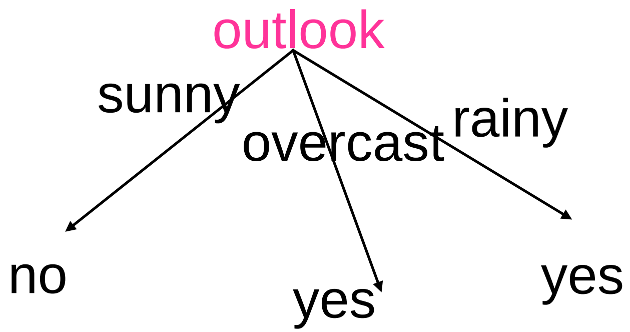
    if outlook=sunny then class=no

    elseif outlook=overcast then class=yes

    elseif outlook=rainy then class=yes

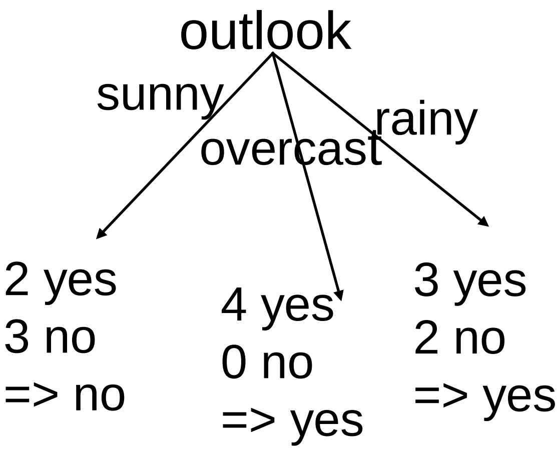
outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

- The rule can be represented as a 1-level decision tree (decision stump)
  - At the root: test the attribute value
  - Each branch corresponds to a value
  - Each leaf corresponds to a class



Q1: How to determine the class label for the leaves?

- Take the majority class (the class that occurs most often)
  - i.e. we are minimizing the errors on the training set



outlook

sunny

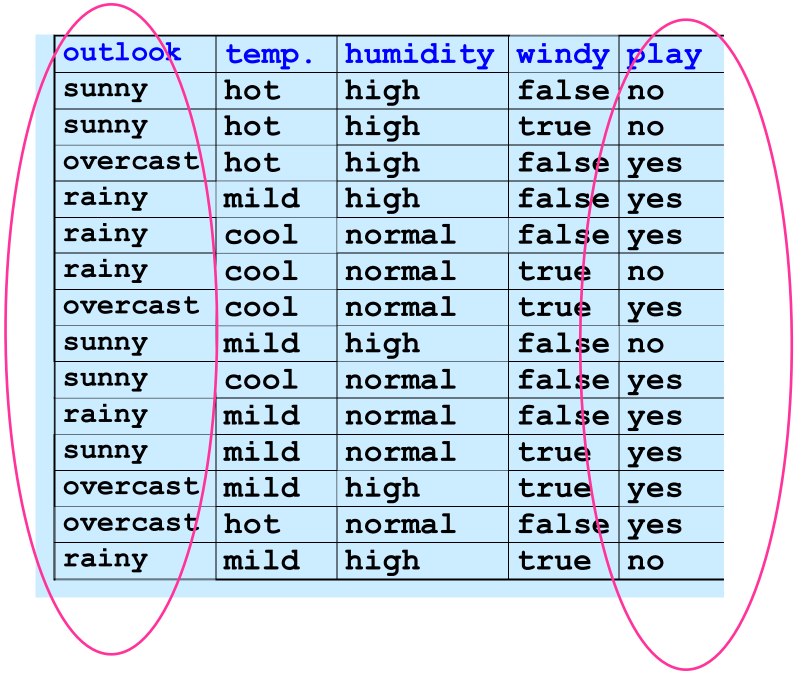
overcast

rainy

2 yes
3 no
=> no

4 yes
0 no
=> yes

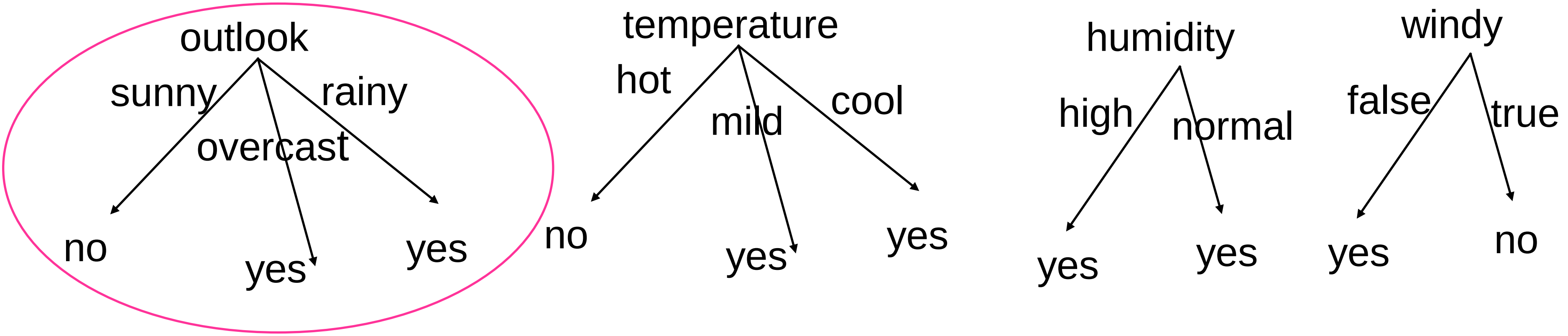
3 yes
2 no
=> yes



outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

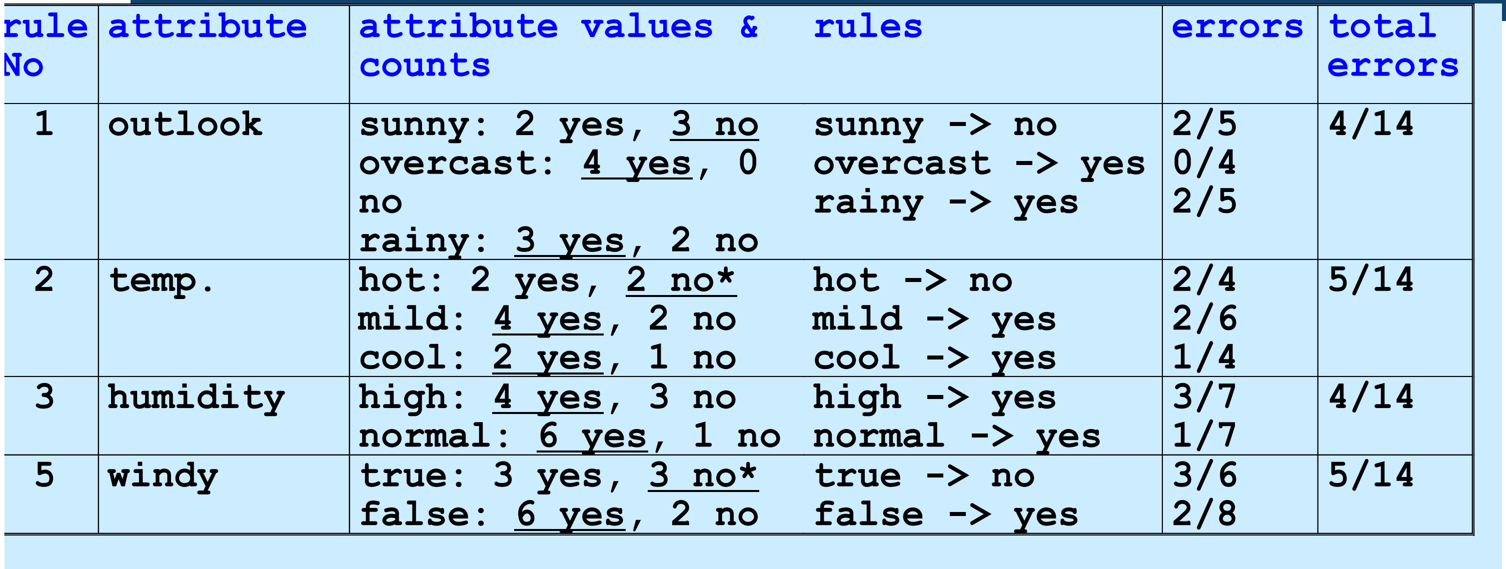
Q2: How to select the best rule?

- A rule is generated for each attribute
- Which is the *best rule* (i.e. the *best attribute*)?
  - The one with the smallest error rate (i.e. with the *highest accuracy*) on training data
- 1R algorithm
  - Generate a rule (decision stump) for each attribute
  - Evaluate each rule on the training data and calculate the number of errors
  - Choose the one with the smallest number of errors



Errors on training data: 4 5 4 5

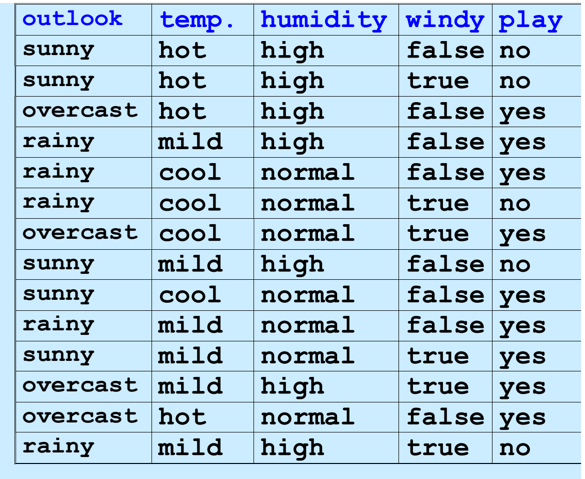
Applying 1R



rule No	attribute	attribute values & counts	rules	errors	total errors
1	outlook	sunny: 2 yes, 3 no overcast: 4 yes, 0 no rainy: 3 yes, 2 no	sunny -> no overcast -> yes rainy -> yes	2/5 0/4 2/5	4/14
2	temp.	hot: 2 yes, 2 no* mild: 4 yes, 2 no cool: 2 yes, 1 no	hot -> no mild -> yes cool -> yes	2/4 2/6 1/4	5/14
3	humidity	high: 4 yes, 3 no normal: 6 yes, 1 no	high -> yes normal -> yes	3/7 1/7	4/14
5	windy	true: 3 yes, 3 no* false: 6 yes, 2 no	true -> no false -> yes	3/6 2/8	5/14

* - random choice

Final rule - rule 1:
if outlook=sunny then play=no
elseif outlook=overcast then play=yes
elseif outlook=rainy then play=yes



outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

1R pseudocode

```
For each attribute,
    For each value of that attribute, make a rule as follows:
        - count how often each class appears
        - find the most frequent class
        - make the rule assign that class to this attribute value.
    Calculate the error rate of the rules.
Choose the rule with the smallest error rate.
```

1R - discussion

- Simple and computationally efficient algorithm
- Produces rules that are easy to understand
  - In contrast to “black-box” models such as neural networks
- Sometimes produces surprisingly good results
- It is useful to try the simple algorithms first and compare their performance with more sophisticated algorithms

- Numerical datasets require discretization
  - 1R has an in-built procedure to do this

Rule-Based Algorithms: PRISM

Covering rule-based algorithm

- PRISM is an example of rule-based covering algorithm
- This type of algorithms:
  - consider each class in turn and
  - construct a set of if-then rules that cover all examples from this class and do not cover any examples from the other classes.
- Called covering algorithms because:
  - At each stage of the algorithm, a rule is identified that “covers” some of the examples
- if part of the rules
  - contains conditions that test the value of an attribute
  - initially empty, tests are added incrementally in order to create a rule with maximum accuracy

if ? then class a

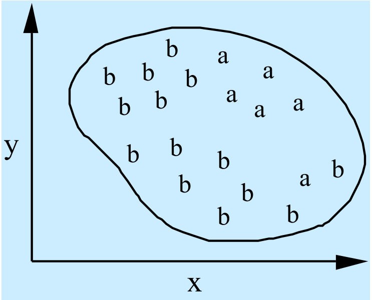
if $x1>2$ then class a

if $x1>2$ & $x2<5$ then class a

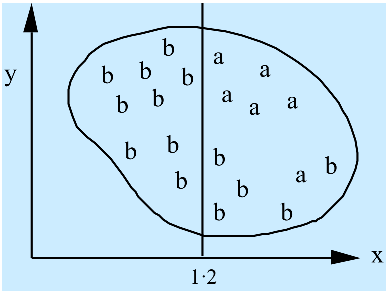
if $x1>2$ & $x2<5$ & $x3<3$ then class a

Example – generating a rule

- Creating a rule for class a
- “perfect” rule – covers only examples from the same class

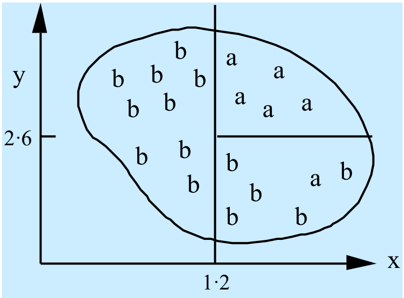


if ? then class a



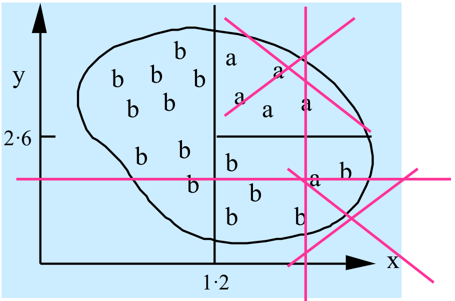
if x>1.2 then class a

Not perfect – covers not only examples from class a but also from class b
=> add another test



if x>1.2 and y>2.6 then class a

Perfect – covers only examples from class a. But it doesn’t cover all class a examples => Stop adding tests to this rule; generate a new rule to cover the uncovered class a examples



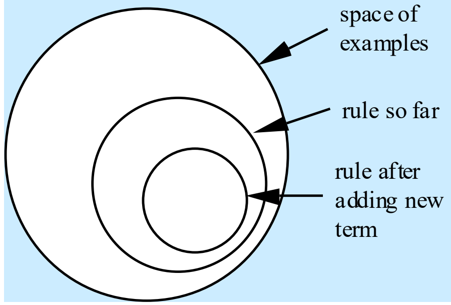
If x>1.4 & y<2.4 then class a

perfect; stop adding tests
All examples from class a are covered, stop generating rules for class a
Generate rules for class b

Accuracy on training data always 100%

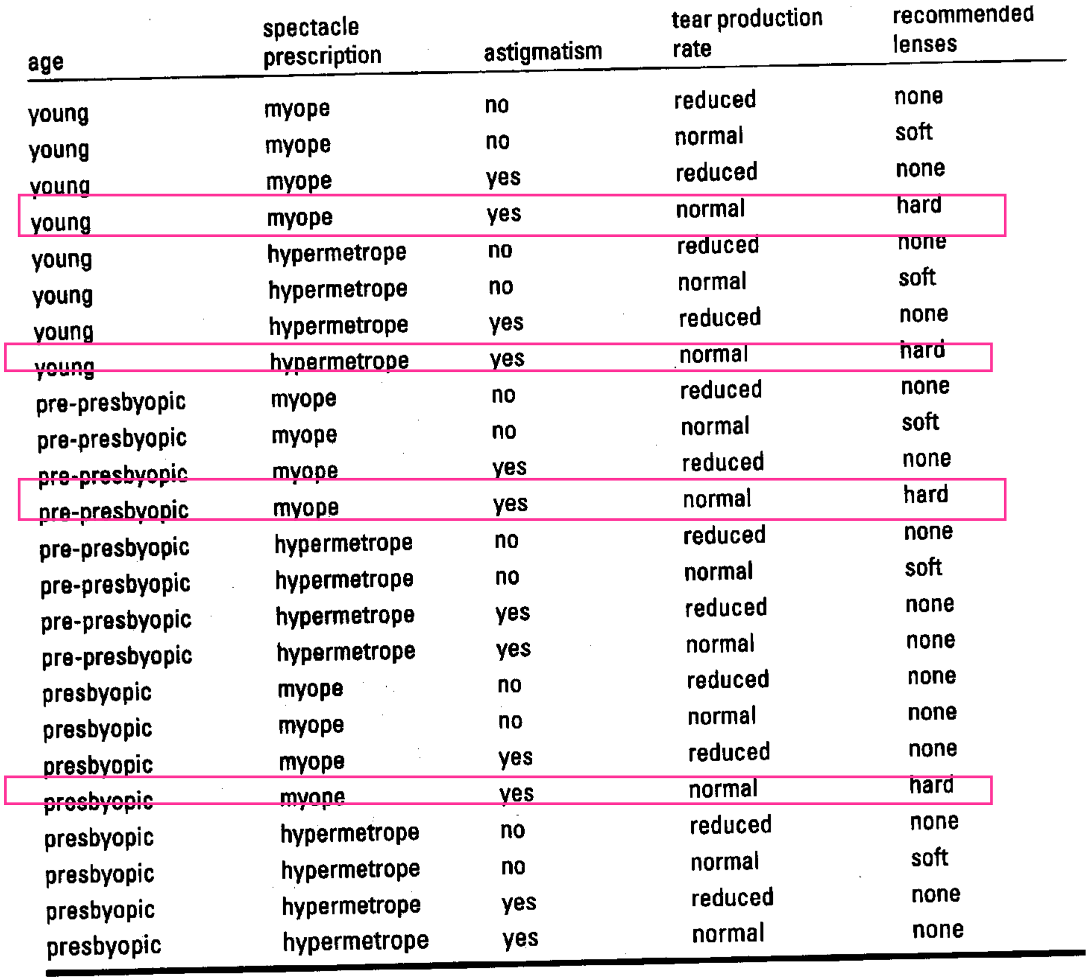
PRISM algorithm

- Idea: Generate a rule by adding tests that maximize the rule's accuracy
- For each class:
  - Start with an empty rule and gradually add conditions
    - Each condition tests the value of 1 attribute
    - By adding a new test, the rule coverage is reduced (i.e. the rule becomes more specific)
  - Finish when *p/t=1* (i.e. the rule covers only examples from the same class, i.e. is "perfect")



- Which test to add at each step? The one that maximizes accuracy p/t:
  - *t*: total number of examples (from all classes) covered by the rule (t comes from *total*)
  - *p*: examples from the class under consideration, covered by the rule (p comes from *positive*)
  - *t-p*: number of errors made by the rule
  - Select the test that maximises the accuracy *p/t*

Example – contact lenses data



age	spectacle prescription	astigmatism	tear production rate	recommended lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	no	reduced	none
pre-presbyopic	myope	no	normal	soft
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	no	reduced	none
pre-presbyopic	hypermetrope	no	normal	soft
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	no	reduced	none
presbyopic	myope	no	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	no	reduced	none
presbyopic	hypermetrope	no	normal	soft
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Let’s start with creating a rule for class “hard”!

Example – contact lenses data (1)

- Start with an empty rule: `if ? then recommendation = hard`

- 9 possible tests for the 4 attributes based on num. attribute values (3+2+2+2):

	p/t (accuracy)
`age = young`	2/8
`age= pre-presbyoptic`	1/8
`age = presbyoptic`	1/8
`spectacle prescription = myope`	3/12
`spectacle prescription = hypermetrope`	1/12
`astigmatism = no`	0/12
`astigmatism = yes`	**4/12**
`tear production rate = reduced`	0/12
`tear production rate = normal`	4/12

age=young in 8 ex. and in 2 of them class=hard

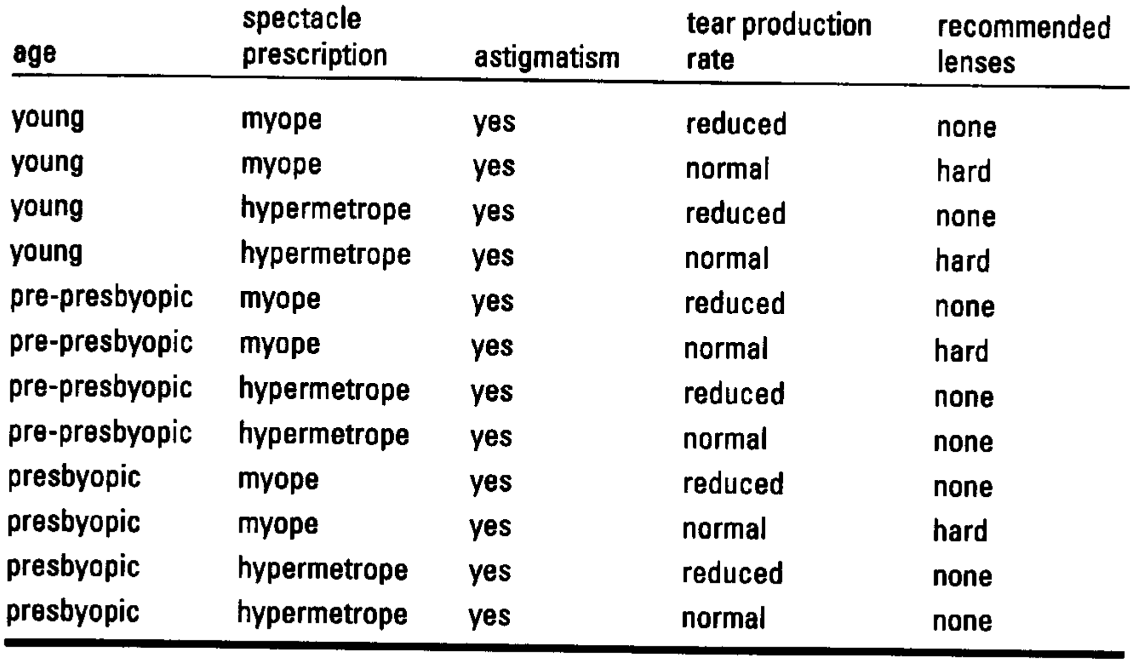
- Best test (highest accuracy): `astigmatism = yes`
- Note that there is a tie: both `astigmatism = yes` and `tear production rate = normal` have the same accuracy 4/12; we choose the first one randomly

Example – contact lenses data (2)

- Current rule

  `if astigmatism = yes then recommendation = hard`

- Not “perfect” - covers 12 examples but only 4 of them are from class `hard` => refinement is needed



age	spectacle prescription	astigmatism	tear production rate	recommended lenses
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Example – contact lenses data (3)

- Further refinement by adding tests:

  `if astigmatism = yes and ? then recommendation = hard`

- Possible tests:

age = young	2/4
age= pre-presbyoptic	1/4
age = presbyoptic	1/4
spectacle prescription = myope	3/6
spectacle prescription = hypermetrope	1/6
tear production rate = reduced	0/6
tear production rate = normal	**4/6**

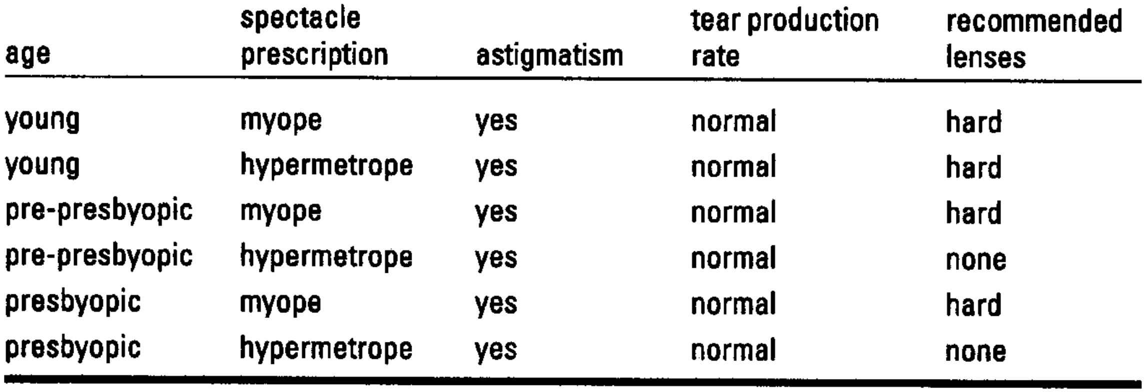
- Best test: `tear production rate = normal`

Example – contact lenses data (4)

- Current rule:

```
if astigmatism = yes & tear production = normal then
recommendation = hard
```

- Examples covered by the rule



age	spectacle prescription	astigmatism	tear production rate	recommended lenses
young	myope	yes	normal	hard
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	yes	normal	none

- The rule is again not “perfect” – 2 examples classified as `none` => further refinement is needed

Example – contact lenses data (5)

- Further refinement:

```
if astigmatism = yes & tear production = normal and ? then
recommendation = hard
```

- Possible tests

age = young	2/2
age= pre-presbyoptic	1/2
age = presbyoptic	1/2
spectacle prescription = myope	3/3
spectacle prescription = hypermetrope	1/3

- Best test: tie between the 1st and 4th; choose the one with the greater coverage (4th)
- New rule:

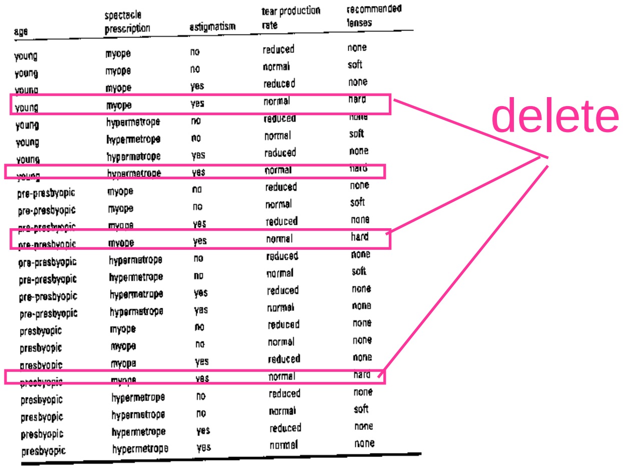
```
if astigmatism = yes & tear production = normal & spectacle
prescription = myope then recommendation = hard
```

Example – contact lenses data (6)

- This rule is perfect – all examples it covers are from class `hard`
- Thus, we stop adding new conditions to the current rule
- Does the rule cover all `hard` examples? No, only 3/4, so we will need another rule
- Delete these 3 examples and start again

```
if ? then recommendation = hard

...
```



age	spectacle prescription	astigmatism	tear production rate	recommended lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	no	reduced	none
pre-presbyopic	myope	no	normal	soft
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	no	reduced	none
pre-presbyopic	hypermetrope	no	normal	soft
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	no	reduced	none
presbyopic	myope	no	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	no	reduced	none
presbyopic	hypermetrope	no	normal	soft
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

delete

- Final second rule

```
if age = young and astigmatism = yes and tear production =
normal then recommendation = hard
```

- Stop as all examples from class `hard` are covered
- Follow the same process for the other 2 classes (`soft` and `none`)

PRISM – pseudocode

```
For each class C
    Initialize E to the instance set
    While E contains instances in class C
        Create a rule R with an empty LHS that predicts C
        Until R is perfect do
            For each attribute A not mentioned in R and each value V
                Consider adding the condition A=v to the LHS of R
            Select A and v to maximize p/t
            (break ties by choosing the condition with the largest p)
            Add A=v to R
        Remove the instances covered by R from E
```

PRISM – discussion

- When applying the PRISM rules to classify a new example:
  - Does the order in which the rules for a given class are applied matter?
  - No, the rules are order-independent

- Problem – some test examples may not be covered by the PRISM rules and will not receive classification
  - Default rule is needed - assign them to the class with the most training examples
- Dealing with numeric attributes – discretization is needed

Covering vs divide-and-conquer algorithms

- PRISM is an example of the *covering* approach for constructing classification rules
  - Takes each class in turn and generates a rule that covers all examples in it while excluding the examples not in the class
  - It is a sequential covering algorithm
- Decision Trees (DT) – we will study them later - are an example of the *divide-and-conquer* approach
  - They are top-down method; start with all examples, at each stage find the attribute that best *separates* the classes; then recursively processing the sub-problems that result from the attribute split.
- Both PRISM and DT select the best test to add to the current rule or tree but the criterion is different
  - PRISM maximizes the accuracy p/t
  - DT maximizes the separation between classes (i.e. maximizes Information Gain)