Data Preparation

Paulo Trigo Silva

(The Key) Steps of the Data Mining Process

- Choose the data to analyze
  - usually from an “operational” (relational) database
- Assemble the data in the format needed for data mining
  - usually a text file
- (Possibly) Apply some preprocessing actions
  - data transform(s), e.g. discretize, binarize, rescale, standardize, normalize
  - e.g., remove of one (or more) attributes
  - e.g., apply some filter to eliminate undesired examples
- Execute the knowledge extraction method
  - e.g., use 1R method to get a model (i.e., classification rules)
- Evaluate the results
  - e.g., compute the expected of the model in a test dataset

De-normalization of the “Data Source”

- When designing a database the goal is to avoid redundancies
  - by normalizing the data
- … as a result, the data for an entity (e.g., a patient) may get
  - spread over multiples tables; or, over multiple tuples of a table
- To prepare for data mining (and even data warehousing)
  - we often need to de-normalize (relax normalization assumptions)
  - multiple tables $\mapsto$ 1 (one) single table (named “dataset”)
- Example: “find associations between prescriptions and patients”
  - … clinic with 3 relevant tables: Patient. Doctor and Prescription
  - we may need to combine data from all the three tables in order to create the necessary training examples (the desired “dataset”)

Paulo Trigo Silva

Assemble the Data

- We may also need to assemble, or (re)format, the data
  - we can use SQL (or a Python script) to do the (re)formatting
- Usually the data must include some metadata
  - the name of each attribute (when the dataset has a tabular format)
  - the domain type of each attribute (e.g., discrete, continuous, string)
  - the target attribute (when we want to apply a classification method)

Country	Zone	Area	Population	Language	Bars	Stripes	Colors	Red	Green	Blue	Gold	White	Black	Orange	Circles	Crosses	Sunstars	Crescent	Triangle	Icon	Animate	TopLeft	BotRight
string	d	c	c	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d	d
i																							
Afghanistan	1	648	16	10	0	3	5	1	1	0	1	1	1	0	0	0	1	0	0	1	0	black	green
Albania	1	29	3	6	0	0	3	1	0	0	1	0	1	0	0	0	1	0	0	0	1	red	red
Algeria	1	2388	20	8	2	0	3	1	1	0	0	1	0	0	0	0	1	1	0	0	0	green	white
Angola	2	1247	7	10	0	2	3	1	0	0	1	0	1	0	0	0	1	0	0	1	0	red	black
Portugal	4	92	10	6	0	0	5	1	1	1	1	1	0	0	1	0	0	0	0	1	0	green	red
Puerto-Rico	4	9	3	2	0	5	3	1	0	1	0	1	0	0	0	0	1	0	1	0	0	red	red
Spain	4	505	38	2	0	3	2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	red	red
Sri-Lanka	1	66	15	10	2	0	4	0	1	0	1	0	0	1	0	0	0	0	0	1	1	gold	gold

- … an example (an excerpt of the “flag” dataset)
  - metadata includes: d - discrete | c - continuous | i - ignore

Paulo Trigo Silva

Data Transform(s) – preprocessing action(s)

The goal is to **prepare the data** to better **align the algorithm's assumptions** with the **structure of the data**.

… different algorithms make different assumptions so we need different "data transforms".

General Recommendation *– create different data transforms (views); then exercise a handful of algorithms on each view; identify the better transforms.*

- … "data transform" techniques (most used) include
  - **discretize**, i.e., from numeric to nominal
  - **binarize**, i.e., a "way of" discretization
  - **rescale**, i.e., the same scale to different domains
  - **standardize**, i.e., Gaussian attribute to standard domain (μ=0 and σ=1)
  - **normalize**, i.e., each observation (row) to unit norm (vector_length=1)

Paulo Trigo Silva

… Discretization (a preprocessing action)

- The discretization is a process that converts
  - a numeric attribute into
  - a nominal/categorical attribute
- This involves segmenting (dividing) the range (domain) into
  - sub-ranges called “buckets” or “bins”
- … example “an *age* attribute may be divided into bins such as:”
  - `child:  ]   .. 12]`
  - `teen:   [13 .. 17]`
  - `young:  [18 .. 35]`
  - `middle: [36 .. 59]`
  - `senior: [60 ..   ]`

Paulo Trigo Silva

… why the Discretization?

*Some **classification** and **clustering** algorithms deal with **nominal attributes only** and cannot handle ones measured on a numeric scale*.

To use them on general datasets, numeric attributes must first be "discretized" into a small number of distinct ranges.

Statistical clustering methods often assume that **numeric** attributes have a normal distribution – often not a very plausible assumption in practice. Also, the standard extension of the Naïve Bayes classifier to handle numeric attributes adopts the same assumption.

Most decision tree and decision rule learners can handle numeric attributes. But, some implementations work much slower when numeric attributes are present because they repeatedly sort the attribute values.

… that's why we may consider **discretization before learning takes place**!

Paulo Trigo Silva

Discretization Approaches

What if we do not know which sub-ranges make sense?

e.g., how to divide (segment) the whole range of country areas?

- **Unsupervised** discretization
  - quantize each attribute in the absence any knowledge of the classes of the instances in the training set
  - … this is the only possibility when dealing with clustering problems in which the classes are unknown or nonexistent
  - e.g., equal-width binning
  - e.g., equal-frequency binning
- **Supervised** discretization
  - take classes into account when discretizing
  - e.g., the 1R discretization method
  - e.g., entropy-based discretization (details later with decision tree classification)

(Unsupervised) Discretization – “equal-width binning”

divide the range into a predetermined number of equal intervals:

a fixed, data-independent yardstick

equal-width binning

divide the range of all possible values into **N** sub-ranges of the same size.

`bin_width = ( maximum value – minimum value ) / N`

Example

if the observed values (of attribute A) are all between 10 and 110

what would be a 5 bins discretization (of attribute A)?

Paulo Trigo Silva

example – “equal-width binning”

Example

if the observed values (of attribute A) are all between 10 and 110
what would be a 5 bins discretization (of attribute A)?

`bin_width = ( 110 – 10 ) / 5 = 20`

so, the **bins** are:

[10 .. 30],]30 .. 50],]50 .. 70],]70 .. 90],]90 .. 110]

usually the first and last bins are extended to allow for values outside the range of observed values

] – infinity .. 30],]30 .. 50],]50 .. 70],]70 .. 90],]90 .. + infinity [

Paulo Trigo Silva

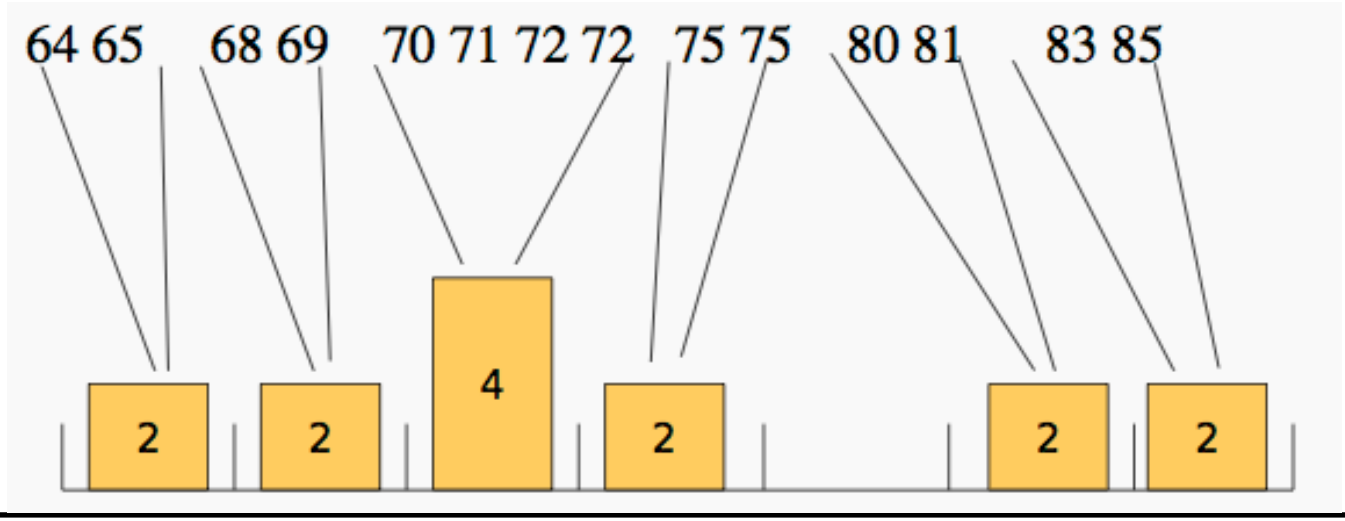
another example – “equal-width binning”

Example

in the weather dataset, the observed values of temperature are all between 64 and 85; what would be a 7 bins discretization?

`bin_width = ( 85 – 64 ) / 7 = 3`

so, create 7 **bins** each of which is between x and x+3:



Paulo Trigo Silva

“equal-width binning” – some characteristics

- This is a simple method and fast to compute
  - only needs to search for the maximum and the minimum values
- But, it requires the N to be provided (e.g., by a human user)
  - how does one knows which is the most appropriate N value?
  - e.g., via repetitive **evaluation** of results with several N values
- But, it does not consider the distribution of observed values
  - e.g., room temperature is *not* equally distributed between -5º .. 40º
  - … most observed values would be in the range of 20 to 30 centigrade
  - e.g., salaries in a big corporation may range from 10k to 500k year
  - … but most will be at the bottom of such scale
- … some bins contain many instances, and others contain none
  - may destroy distinctions possibly useful for the learning process

(Unsupervised) Discretization – “equal-frequency binning”

intervals of different sizes with the same number of examples in each one:

number of instances in each bin is the same and the ranges covered are different.

equal-frequency (or equal-height, or histogram) binning

divide the range of all possible values into **N** bins each of which holds approximately the same number of training examples.

`bin_freq ≈ #examples / N`

Example

if we have 14 observed values (of attribute A) what would be a 4 bins discretization (of attribute A), given that the observed values are:

75, 5, 17, 19, 15, 20, 44, 65, 22, 18, 5, 22, 1, 60

Paulo Trigo Silva

example – “equal-frequency binning”

Example

if we have 14 observed values (of attribute A) what would be a 4 bins discretization (of attribute A), given that the observed values are:

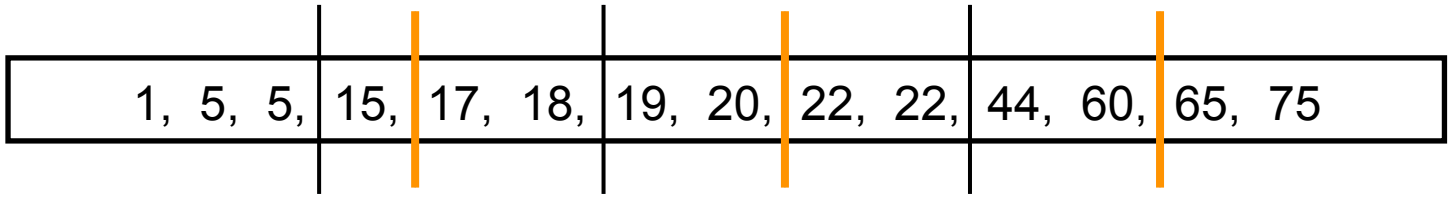
75, 5, 17, 19, 15, 20, 44, 65, 22, 18, 5, 22, 1, 60

`bin_freq ≃ 14 / 4 = 3`x*4* **+** *2*

option1: have **3 bins** with *4* examples and **1 bin** with *2* examples

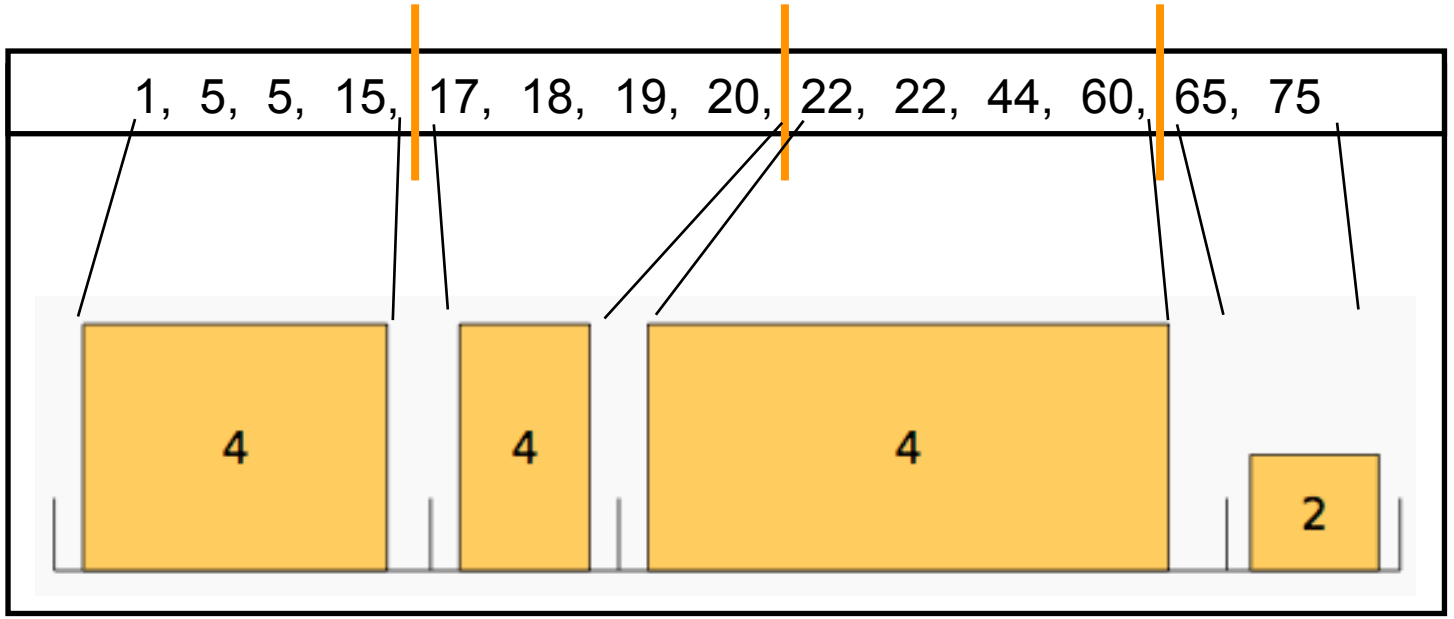
option2: distribute the *2* examples among the bins (e.g., the last)

first sort the examples and then set the bin breakpoints



Paulo Trigo Silva

… example – “equal-frequency binning”



This method is sometimes called **histogram equalization**,

because if we take a histogram of the contents of the resulting bins, such histogram will be (almost) completely flat.

if we view the number of bins as a resource, this method makes best use of it; i.e., **completely fills the resources with the data**!

“equal-frequency binning” – some characteristics

- This is a simple method and fast to compute
  - only needs to get the cardinality of the dataset (number of examples)
- But, it requires the N to be provided (e.g., by a human user)
  - how does one knows which is the most appropriate N value?
  - e.g., via repetitive evaluation of results with several N values
- But, it does not consider the relation with the class attribute
  - e.g., if all instances in a bin have one class, and all instances in the next higher bin have another except for the first, which has the original class, surely it makes sense to respect the class divisions and include that first instance in the previous bin
  - … sacrifice the equal-frequency property for the sake of homogeneity
- … nevertheless, equal-frequency binning yields good results
  - once found a proper number, N, of intervals

… heuristics to choose the number of intervals

The **number of bins** (intervals), **N**, should be **larger than the number of class values** in order to be possible to retain the mutual information between class values end intervals.

A useful heuristic ("rule of thumb") formula,

$$\mathbf{N} = \max[\, C,\ E / (3 \times C) \,]$$

where:

C: number of class values

E: number of examples

Statistics also tell us **that no fewer than 5 points** should be in any bin (interval); interval with few than 5 points do not have statistics relevance.

Paulo Trigo Silva

… heuristics – example

A useful heuristic (“rule of thumb”) formula,

N = max[C, E / (3 x C)]

where:

C: number of class values

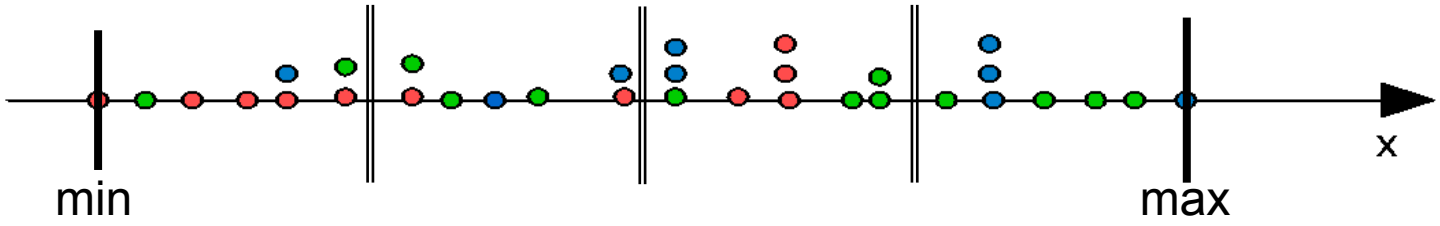
E: number of examples

C = 3 (red, green, blue)

E = 33

N = max[3, 33 / (3 x 3)] = max[3, 4] = 4

example using the “equal-width binning” method



Paulo Trigo Silva

(Supervised) Discretization – the 1R method

Idea: take classes into account during the discretization process.

1R (main) criteria: if two consecutive values, of the class, are different, then break in the middle point of the corresponding values of the numeric attribute

The “temperature” attribute discretization (example previously explored):

Pontos de Partição: ***17,5*** ***19*** ***?*** ***?*** ***25*** ***26,5*** ***28,5***

Temperatura	17	18	20	20	21	21	22	22	24	24	26	27	28	29
Jogar?	sim	não	sim	sim	sim	não	não	sim	sim	sim	não	sim	sim	não
Temperatura	C1	C2	C3			C4		C5			C6	C7		C8

Pontos de Partição: *17,5* *19* *25* *26,5* *28,5*

Temperatura	17	18	20	20	21	21	22	22	24	24	26	27	28	29
Jogar?	sim	não	sim	sim	sim	não	não	sim	sim	sim	não	sim	sim	não
Temperatura	C1	C2	C3								C4	C5		C6

Paulo Trigo Silva

(Supervised) Discretization – “entropy-based” method

Idea: use criterion for splitting a numeric attribute in decision tree formation.

decision tree (main) criteria:

1. sort instances by attribute’s value and consider, **for each possible splitting point**, the “**information gain**” of the resulting split.

2. to discretize the attribute, once the first split is determined the process can be repeated in the upper and lower parts of the range, and so on, recursively.

… to be detailed in a later lesson (about decision tree construction)

The “temperature” attribute discretization (example previously explored):

64	65	68	69	70	71	72	75	80	81	83	85
yes	no	yes	yes	yes	no	no yes	yes yes	no	yes	yes	no

… "entropy-based" method – main idea

The "temperature" attribute discretization (example previously explored):

64	65	68	69	70	71	72	75	80	81	83	85
yes	no	yes	yes	yes	no	no yes	yes yes	no	yes	yes	no

For example, the information value of the test "**temperature < 71.5**", which splits the range into 4 yes's and 2 no's versus 5 yes's and 3 no's, is:

info([4; 2], [5; 3]) =

= (6/14) × info([4, 2]) + (8/14) × info([5, 3]) =

= (6/14) × entropy([4/6, 2/6]) + (8/14) × entropy([5/8, 3/8]) =

= (6/14) × [– 4/6 $\log_2$(4/6) – 2/6 $\log_2$(2/6)] +

(8/14) × [– 5/8 $\log_2$(5/8) – 3/8 $\log_2$(3/8)]

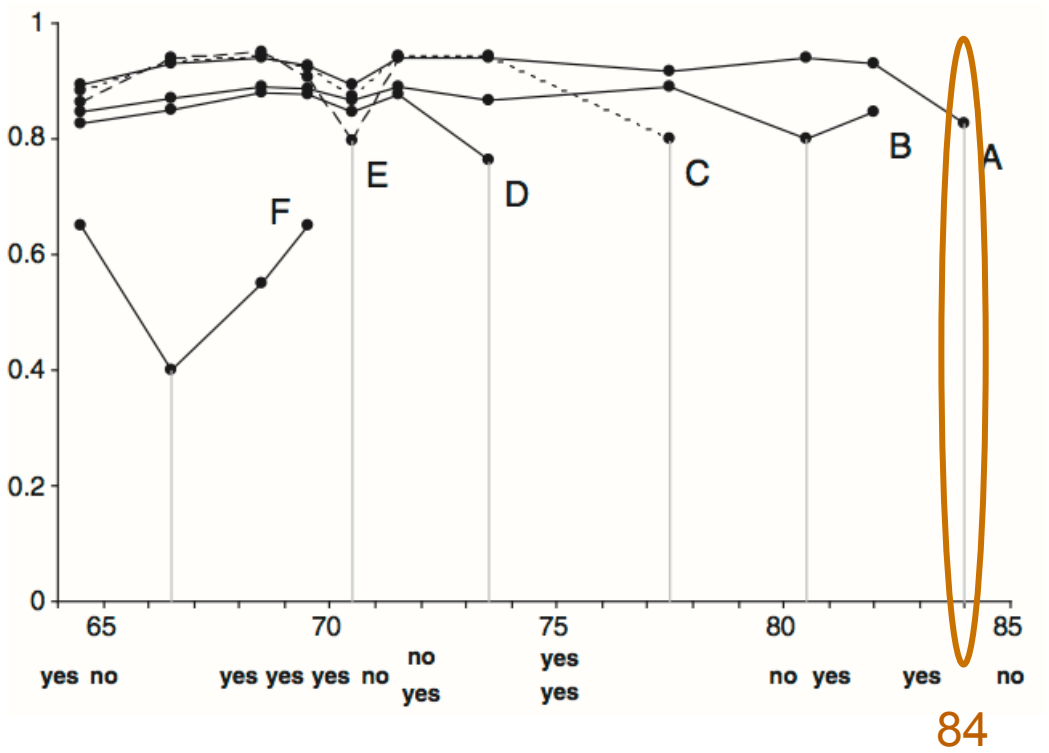
= **0.939** bits

note: the lower the information value the "more cleaner" the division; e.g.,

e.g., info([0; 2], [5; 0]) = (2/2) × info([0, 2]) + (5/5) × info([5, 0]) = 0

… “entropy-based” method – example (1st split)

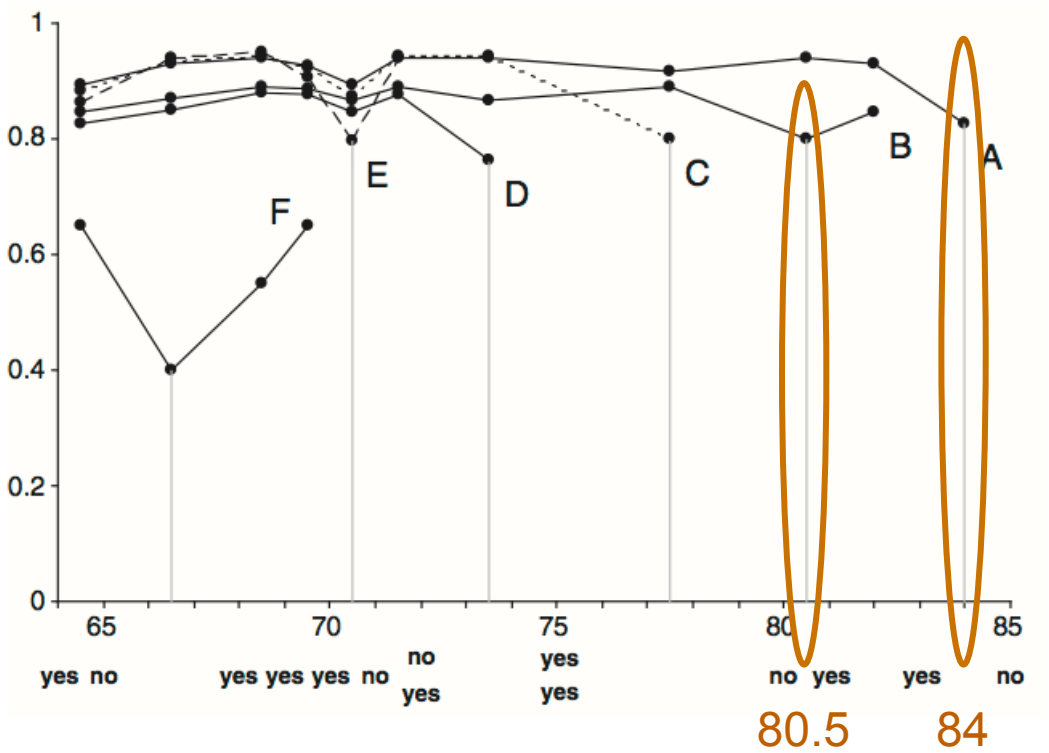
*Graph A: shows information values at each possible cut point at the first stage. The cleanest division – smallest information value – is at a temperature **84** (0.827 bits); separates just the final value, “no”, from the preceding list.*



Paulo Trigo Silva

… “entropy-based” method – example (2nd split)

*Graph B: invoking the algorithm again on the lower range of temperatures (from 64 to 83) yields the graph B, which has a minimum at **80.5** (0.800 bits), which splits off the next two values both “yes” instances.*

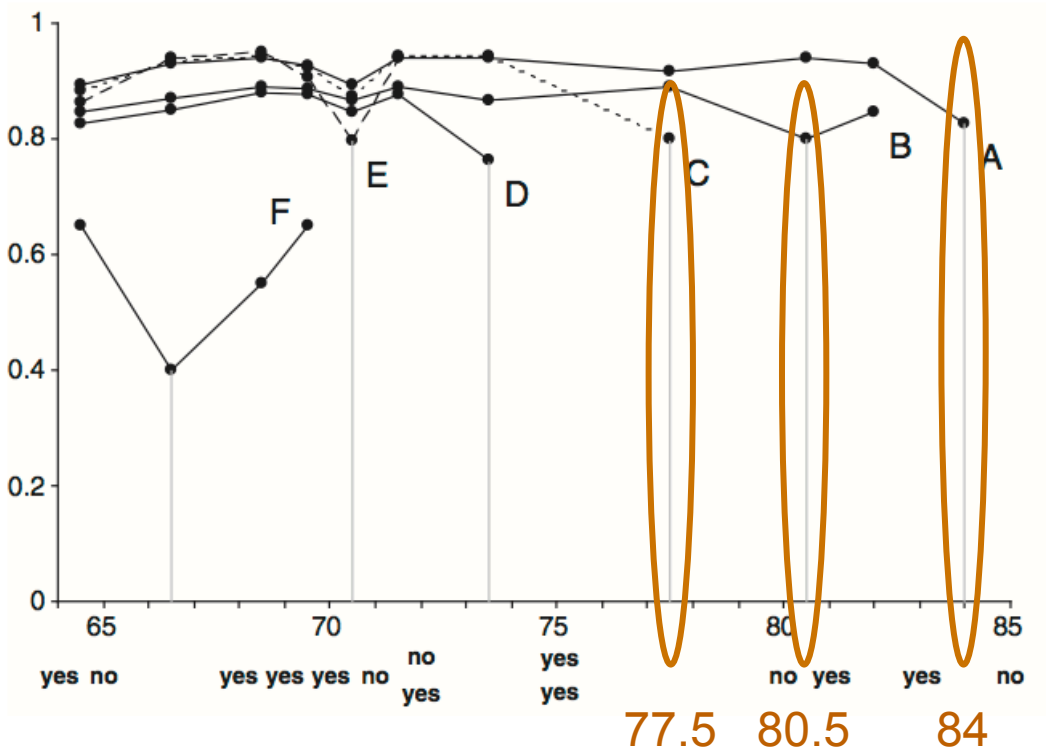


Paulo Trigo Silva

… “entropy-based” method – example (3rd split)

*Graph C: again invoking the algorithm on the lower range of temperatures (now from 64 to 80) yields the graph C, which has a minimum at **77.5** (0.801 bits), which splits off another “no” instance.*

this process repeats and yields graphs D, E, F.



Paulo Trigo Silva

… “entropy-based” method – example (final result)

The result of discretizing the “temperature” attribute.

64	65	68	69	70	71	72	75	80	81	83	85
yes	no	yes	yes	yes	no	no yes	yes yes	no	yes	yes	no

F E D C B A

66.5 70.5 73.5 77.5 80.5 84

The fact that recursion only ever occurs in the first interval of each split is an artifact of this example: in general, both the upper and the lower intervals will have to be split further.

There are some possible stopping criteria for the recursion. It can stop whenever a “pure” partition was found. It can also follow the principle that the “best hypothesis is the one with minimal description length (MDL)” (*see later*).

The “ordering information” of a discretized attribute

- A discretized attribute is derived from a numeric one
  - so its values are ordered
- … treating an interval of numeric value as a nominal one
  - discards the numeric ordering (possibly) valuable information
- If the learning method **can** handle ordered attributes directly
  - the solution is simple
  - … just declare each discretized attribute of the type “ordered”
- If the learning method **can not** handle ordered attributes
  - if the discretized attribute has ***k*** values, transform that attribute into a set of ***k -1*** binary attributes … set the $(i - 1)^{th}$ binary attribute value to represent whether the discretized attribute is less than i.
  - … if a decision tree learner splits on this attribute, it implicitly uses the ordering information it encodes.

Paulo Trigo Silva

… exploit the ordering information of a discrete attribute

- Exploit the ordering information of a discrete attribute
  - even when the learning method does not account for such information
- Given a discrete attribute with ***k*** values
  - transform that attribute into a set of ***k -1*** binary attributes
  - the value of the first *i - 1* attributes are set to *false*, whenever the i^{th} value of the discretized attribute is present in the data; *true* otherwise
  - the remaining attributes are set to *true*.
- … or, in other words, the $(i - 1)^{th}$ binary attribute
  - represents whether the discretized attribute value is higher or equal than *i*.

Example

suppose that attribute "`dimension`" has values: `low`, `medium`, `high`, `huge`

apply the above technique to exploit the fact that:

`low < medium < high < huge`

Paulo Trigo Silva

… example – ordering information of a discrete attribute

Example

suppose that attribute "`dimension`" has values: `low`, `medium`, `high`, `huge`

apply the above technique to exploit the fact that:

`low < medium < high < huge`

Technique

transform `dimension`, **4** valued attribute, into **3** attributes: `low`, `medium`, `high`

transform each value in the original dataset into a tuple:

`< true|false, true|false, true|false >`

original dataset → **transformed dataset**

illustrative scenario

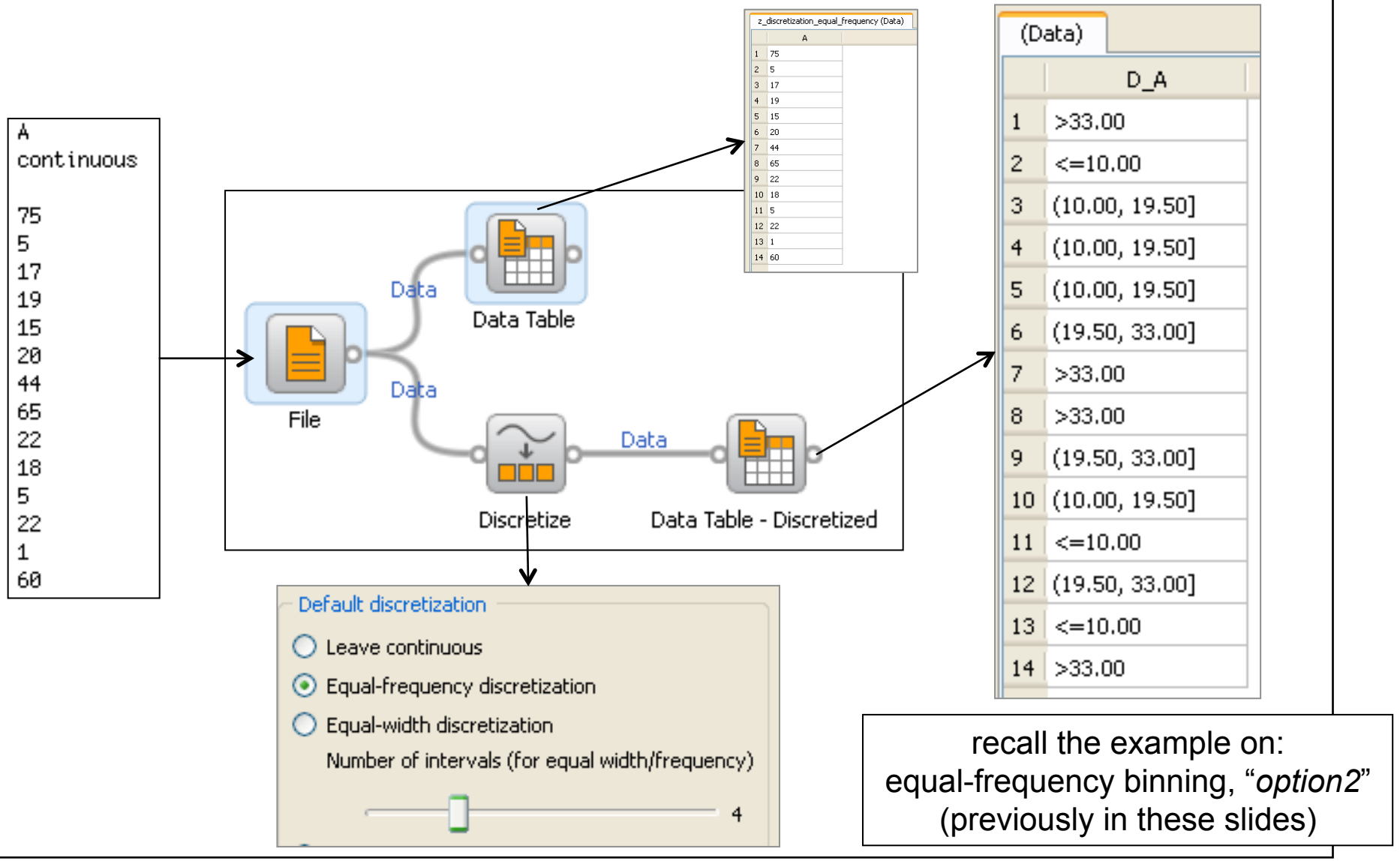
dimension
huge
high
medium
low

the $(i - 1)^{th}$ binary attribute represents whether the discrete attribute value is higher or equal than *i*.

low	medium	high
FALSE	FALSE	FALSE
FALSE	FALSE	TRUE
FALSE	TRUE	TRUE
TRUE	TRUE	TRUE

Paulo Trigo Silva

Discretization in “Orange Data Mining”



recall the example on:
equal-frequency binning, “*option2*”
(previously in these slides)

(recall the) Data Transform(s) – preprocessing action(s)

- … "data transform" techniques (most used) include
  - *discretize, i.e., from numeric to nominal*
  - **binarize**, i.e., a "way of" discretization
  - **standardize**, i.e., mean removal and variance scaling (e.g., Gaussian μ=0, σ=1)
  - **rescale**, i.e., the same scale to different domains
  - **normalize**, i.e., each observation (row) to unit norm (vector_length=1)

- we explore each technique with the following "data-transform-recipe":
  - **load** a dataset, D,
  - **split** D into features, X, and class, y, attributes
  - **apply** a "data transform" to the feature (X) attributes (and get Dtransf)
  - **summarize** the data, Dtransf, to show the change

… implement the “data-transform-recipe” using libraries

- … the Python `scikit-learn` library provides two functions
  - `fit()`, to prepare **once** the parameters of the **data** transform, and
  - `transform()`, to apply on the **same-or-other-samples** of that **data**.
- … the Python `pandas` library helps to create/manipulate a dataset
  - `read_csv()`,returns a `DataFrame` structure
  - … `DataFrame` enables *sql-like* manipulation

```
from pandas import read_csv, DataFrame

from sklearn.preprocessing import
     Binarizer, MinMaxScaler, StandardScaler, Normalizer
```

dataTransform.py

… the recipe implementation

```
def data_transform_recipe( file_feature_name, func_transformer, *args ):
  (fileName, list_featureName) = file_feature_name
  D = load_dataset( fileName, list_featureName )
  (X, y) = split_dataset( D )
  Dtransf = apply_data_transform( X, func_transformer, *args )
  summarize_data( Dtransf )

# the list of functions for “data-transform”
list_func_transformer = \
  [ (binarize,    (0.0, )     ),  #threshold
    (standardize, (True, True)),  #with_mean, with_std
    (rescale,     (0, 1)      ),  #min_range, max_range
    (normalize,   (“l2”, )    ) ] #norm (“l1” or “l2”); we see this later
fileName = "pima-indians-diabetes.data.csv”
list_featureName = ['preg', 'plas', 'pres', 'skin’,
                    'test’, 'mass', 'pedi', 'age', 'class']

def main():
  for (f, args) in list_func_transformer:
    data_transform_recipe( (fileName, list_featureName), f, *args )

#_____________________________
# the "main" of this module
if __name__ == "__main__": main()
```

dataTransform.py

Paulo Trigo Silva

… about the dataset

```
fileName = "pima-indians-diabetes.data.csv"
list_featureName = ['preg', 'plas', 'pres', 'skin',
                    'test', 'mass', 'pedi', 'age', 'class']
```

Dataset originally from *National Institute of Diabetes and Digestive and Kidney Diseases*. Goal is to predict, based on diagnostic measurements, whether a patient has diabetes.

… several medical predictor (independent) variables and one target (dependent) variable.

pregnancies: *number of times pregnant*

plasmaGlucose: *glucose concentration a 2 hours in an oral glucose tolerance test*

blood**Pres**sure: *diastolic blood pressure (mm Hg)*

skinThickness: *triceps skin fold thickness (mm)*

insulin**Test**: *2-hour serum insulin (mu U/ml)*

body**Mass**Index: *BMI, weight in kg/(height in m)^2*

diabetes**Pedi**greeFunction: *diabetes pedigree function*

age: *age (in years)*

outcome: ***class*** *variable (0 or 1); i.e., the "target" variable*

Paulo Trigo Silva

… the recipe implementation (cont.)

```
def load_dataset( fileName, list_featureName=None ):
  D = read_csv( fileName, names=list_featureName )
  return D


# split feature, X, from class, y, attributes
# (assume that class is the "last column")
def split_dataset( D ):
  # all rows and all columns except last column
  X = D.values[:, 0:-1]

  # all rows and just the last column
  y = D.values[:, -1]
  return ( X, y )


# apply data transform
def apply_data_transform( data, func_transformer, *args ):
  transformer = func_transformer( data, *args )
  data_transformed = transformer.transform( data )
  return data_transformed
```

dataTransform.py

… binarize – a preprocessing action

```
def binarize( data, threshold ):
  # 1, if value>threshold; 0, otherwise
  transformer = Binarizer( threshold=threshold ).fit( data )
  return transformer
```

dataTransform.py

transform data using a **binary threshold**

all values above the threshold are marked 1 (one)

`if value>threshold then value=1`

all values below or equal the threshold are marked 0 (zero)

`if value<=threshold then value=0`

useful when we have probabilities and we want to make them crisp values

useful when we want to add a new feature that indicate something meaningful, e.g., if body-mass-index indicates obesity or not

… standardize – a preprocessing action

```
def standardize( data, with_mean, with_std ):
  # X = (X – mean) / std, if with_mean=with_std=True
  transformer = StandardScaler( with_mean=with_mean, with_std=with_std ).fit( data )
  return transformer
```

dataTransform.py

transform data by mean removal and standard deviation (std) scaling

in practice we often ignore the shape of the distribution and just transform data to center it by removing the mean of each feature, then scale by dividing by standard deviation

transform into a distribution (e.g., standard Gaussian) with `mean=0` and `std=1`

`X=(X–mean)/std`

useful for algorithms that assume that features are centered around zero and have variance in the same order (e.g., linear and logistic regression)

if a feature has a variance orders of magnitude larger than others, it may dominate and make estimator unable to learn from other features correctly as expected

… rescale – a preprocessing action

```
def rescale( data, min_range, max_range ):
  # X_std = (X - X.min) / (X.max - X.min)
  # X_scaled = X_std * (max_range – min_range) + min_range
  transformer = MinMaxScaler( feature_range=(min_range, max_range) ).fit( data )
  return transformer
```

dataTransform.py

transform data to lie between a `min_range` and `max_range` value (often `0` and `1`)

rescale is an alternative to standardize; a motivation to rescale is the robustness to very small standard deviations of features

transform into `0..1`: `X = (X–X.min)/(X.max–X.min)`
then, into `min_range..max_range`: `X=X*(max_range–min_range)+min_range`

useful for optimization algorithms that are (often) in the core of some algorithms such as gradient descent

useful for algorithms that use distance measures (e.g., LVQ or KNN) and for algorithms that weight attributes (e.g., regression or neural networks)

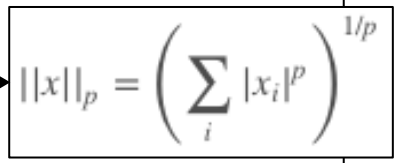
… normalize – a preprocessing action

```
def normalize( data, norm ):
  # Xi / sum(|Xi|^p)^1/p, where i is each value of the individual sample X
  # norm="l1" uses p=1; norm="l2" uses p=2
  transformer = Normalizer( norm=norm ).fit( data )
  return transformer
```

dataTransform.py

transform data for each observation (row, or individual sample) to have a length of 1 (called a unit norm or a vector with the length of 1 in linear algebra)

this unit-norm assumption is the base of the vector-space-model often used in information retrieval, text classification and clustering contexts



transform using norm "L1" (p=1) or "L2" (p=2)
i.e., divide each value of X by its norm; $Xi / ||X||_1$ for "L1" or $Xi / ||X||_2$ for "L2"

useful for sparse datasets (lots of zeros) with attributes of varying scales when using algorithms that use distance measures (e.g., LVQ or KNN) and for algorithms that weight attributes (e.g., regression or neural networks)

Paulo Trigo Silva

… (finally) the summarize of each transform

```
(binarize,
#threshold
(0.0, ))
```

```
>> Summarize Data:
[[ 1.  1.  1.  1.  0.  1.  1.  1.]
 [ 1.  1.  1.  1.  0.  1.  1.  1.]
 [ 1.  1.  1.  0.  0.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.]
 [ 0.  1.  1.  1.  1.  1.  1.  1.]]
```

```
(standardize,
#with_mean, with_std
(True, True))
```

```
>> Summarize Data:
[[ 0.64   0.848  0.15   0.907 -0.693  0.204  0.468  1.426]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.684 -0.365 -0.191]
 [ 1.234  1.944 -0.264 -1.288 -0.693 -1.103  0.604 -0.106]
 [-0.845 -0.998 -0.161  0.155  0.123 -0.494 -0.921 -1.042]
 [-1.142  0.504 -1.505  0.907  0.766  1.41   5.485 -0.02 ]]
```

```
(rescale,
#min_range, max_range
(0, 1))
```

```
>> Summarize Data:
[[ 0.353  0.744  0.59   0.354  0.     0.501  0.234  0.483]
 [ 0.059  0.427  0.541  0.293  0.     0.396  0.117  0.167]
 [ 0.471  0.92   0.525  0.     0.     0.347  0.254  0.183]
 [ 0.059  0.447  0.541  0.232  0.111  0.419  0.038  0.   ]
 [ 0.     0.688  0.328  0.354  0.199  0.642  0.944  0.2  ]]
```

```
(normalize,
#norm
("l2", ))
```

```
>> Summarize Data:
[[ 0.034  0.828  0.403  0.196  0.     0.188  0.004  0.28 ]
 [ 0.008  0.716  0.556  0.244  0.     0.224  0.003  0.261]
 [ 0.04   0.924  0.323  0.     0.     0.118  0.003  0.162]
 [ 0.007  0.588  0.436  0.152  0.622  0.186  0.001  0.139]
 [ 0.     0.596  0.174  0.152  0.731  0.188  0.01   0.144]]
```

Paulo Trigo Silva

… and the Next (Key) Steps of the Data Mining Process

- Execute the knowledge extraction method
  - next we explore additional data mining methods
  - … classification, rule extraction, clustering
- Evaluate the results
  - next we explore techniques to validate results
  - … build the training, validation and test datasets
  - … use stratified holdout, cross-validation, leave-on-out, bootstrap