

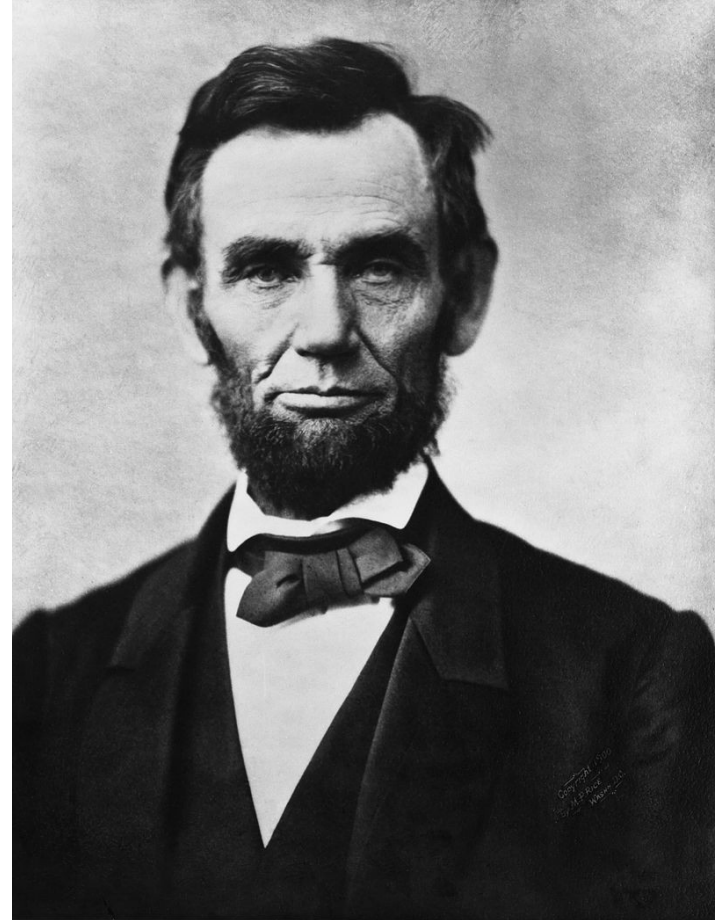
Data Mining II

Data Preprocessing



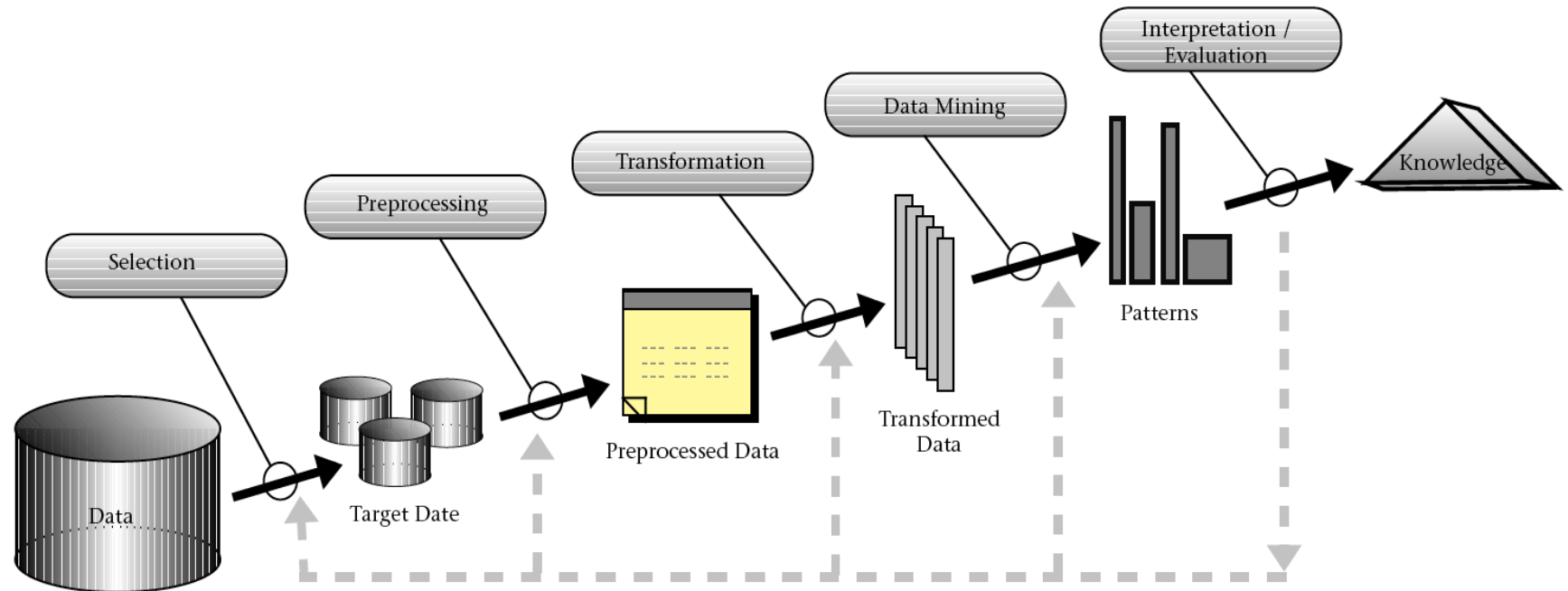
Introduction

- “Give me six hours to chop down a tree and I will spend the first four sharpening the axe.”



Abraham Lincoln, 1809-1865

Recap: The Data Mining Process



Source: Fayyad et al. (1996)

Recap: The Data Mining Process



Data Preprocessing

- Your data may have some problems
 - i.e., it may be problematic for the subsequent mining steps
- Fix those problems before going on
- Which problems can you think of?



Data Preprocessing

- Problems that you may have with your data
 - Errors
 - Missing values
 - Unbalanced distribution
 - False predictors
 - Unsupported data types
 - High dimensionality

Errors in Data

- Sources
 - malfunctioning sensors
 - errors in manual data processing (e.g., twisted digits)
 - storage/transmission errors
 - encoding problems, misinterpreted file formats
 - bugs in processing code
 - ...



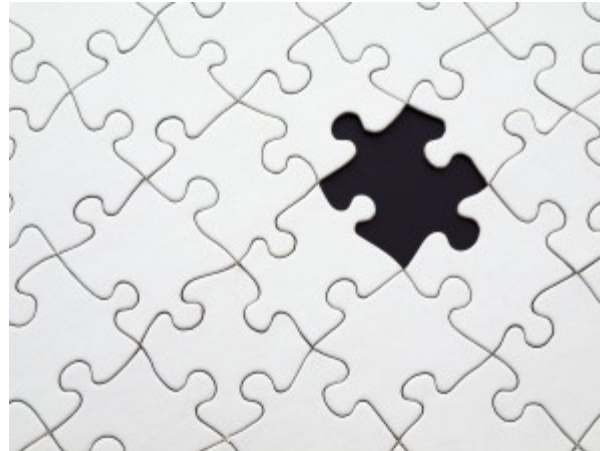
Image: <http://www.flickr.com/photos/16854395@N05/3032208925/>

Errors in Data

- Simple remedy
 - remove data points outside a given interval
 - this requires some domain knowledge
- Typical Examples
 - remove temperature values outside -30 and +50 °C
 - remove negative durations
 - remove purchases above 1M Euro
- Advanced remedies
 - automatically find suspicious data points
 - see lecture “Anomaly Detection”

Missing Values

- Possible reasons
 - Failure of a sensor
 - Data loss
 - Information was not collected
 - Customers did not provide their age, sex, marital status, ...
 - ...



Missing Values

- Treatments

- Ignore records with missing values in training data
- Replace missing value with...
 - default or special value (e.g., 0, “missing”)
 - average/median value for numerics
 - most frequent value for nominals

```
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
```

- Try to predict missing values:
 - handle missing values as learning problem
 - target: attribute which has missing values
 - training data: instances where the attribute is present
 - test data: instances where the attribute is missing

```
imp = imputer = KNNImputer(n_neighbors=2, weights="uniform")
```

Missing Values

- Note: values may be missing for various reasons
 - ...and, more importantly: at random vs. not at random
- Examples for not random
 - Non-mandatory questions in questionnaires
 - “how often do you drink alcohol?”
 - Values that are only collected under certain conditions
 - e.g., final grade of your university degree (if any)
 - Sensors failing under certain conditions
 - e.g., at high temperatures
- In those cases, averaging and imputation causes information loss
 - In other words: “missing” can be information!

Unbalanced Distribution

- Example:
 - learn a model that recognizes HIV
 - given a set of symptoms
- Data set:
 - records of patients who were tested for HIV
- Class distribution:
 - 99.9% negative
 - 0.01% positive



Unbalanced Distribution

- Learn a decision tree
- Purity measure: Gini index
- Recap: Gini index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j|t)]^2$$

– (NOTE: $p(j|t)$ is the relative frequency of class j at node t).

- Here, Gini index of the top node is

$$1 - 0.999^2 - 0.001^2 = 0.002$$

- It will be hard to find any splitting that significantly improves the purity

Decision tree learned:

false

Unbalanced Distribution

- Model has very high accuracy
 - 99.9%
- ...but 0 recall/precision on positive class
 - which is what we were interested in
- Remedy
 - re-balance dataset for training
 - but evaluate on unbalanced dataset!

Decision tree learned:

false

- Balancing:

```
df_majority_downsampled = resample(df_majority,  
                                   replace=False,  
                                   n_samples=100)
```

Resampling Unbalanced Data

- Two conflicting goals
 1. use as *much* training data as possible
 2. use as *diverse* training data as possible
- Strategies
 - Downsampling larger class
 - conflicts with goal 1
 - Upsampling smaller class
 - conflicts with goal 2

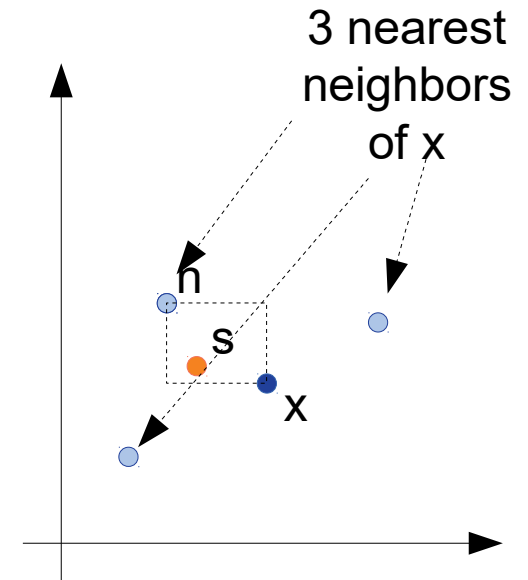
Resampling Unbalanced Data

- Consider an extreme example
 - 1,000 examples of class A
 - 10 examples of class B
- Downsampling
 - does not use 990 examples
- Upsampling
 - creates 100 copies of each example of B
 - likely for the classifier to simply *memorize* the 10 B cases

Resampling

- SMOTE (Synthetic Minority Over Sampling Technique)
 - creates synthetic examples of minority class
- Given an example x
 - create synthetic example s
 - choose n among the k nearest neighbors (w/in same class) of x
 - for each attribute a
 - $s.a \leftarrow x.a + \text{rand}(0,1) * (n.a - x.a)$
- Python has >80 variants of SMOTE

```
import smote_variants as sv
```



False Predictors

- ~100% accuracy are a great result
 - ...and a result that should make you suspicious!
- A tale from the road
 - working with our Linked Open Data extension
 - trying to predict the world university rankings
 - with data from DBpedia
- Goal:
 - understand what makes a top university



False Predictors

- The Linked Open Data extension
 - extracts additional attributes from Linked Open Data
 - e.g., DBpedia
 - unsupervised (i.e., attributes are created fully automatically)
- Model learned: $THE < 20 \rightarrow TOP = true$
 - false predictor: target variable was included in attributes
- Other examples
 - $mark < 5 \rightarrow passed = true$
 - $sales > 1000000 \rightarrow bestseller = true$



Recognizing False Predictors

- By analyzing models
 - rule sets consisting of only one rule
 - decision trees with only one node
- Process: learn model, inspect model, remove suspect, repeat
 - until the accuracy drops
 - Tale from the road example: there were other indicators as well
- By analyzing attributes
 - compute correlation of each attribute with label
 - correlation near 1 (or -1) marks a suspect
- Caution: there are also strong (but not false) predictors
 - it's not always possible to decide automatically!



Unsupported Data Types

- Not every learning operator supports all data types
 - some (e.g., ID3) cannot handle numeric data
 - others (e.g., SVM) cannot nominal data
 - dates are difficult for most learners
- Solutions
 - convert nominal to numeric data
 - convert numeric to nominal data (discretization, binning)
 - extract valuable information from dates

Conversion: Binary to Numeric

- Binary fields
 - E.g. student=yes,no
- Convert to Field_0_1 with 0, 1 values
 - student = yes → student_0_1 = 0
 - student = no → student_0_1 = 1

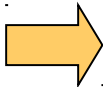
Conversion: Ordered to Numeric

- Some nominal attributes incorporated an *order*
- Ordered attributes (e.g. grade) can be converted to numbers preserving natural order, e.g.
 - A → 4.0
 - A- → 3.7
 - B+ → 3.3
 - B → 3.0
- Using such a coding schema allows learners to learn valuable rules, e.g.
 - `grade>3.5 → excellent_student=true`

Conversion: Nominal to Numeric

- Multi-valued, unordered attributes with small no. of values
 - e.g. Color=Red, Orange, Yellow, ..., Violet
 - for each value v , create a binary “flag” variable C_v , which is 1 if Color= v , 0 otherwise

ID	Color	...
371	red	
433	yellow	



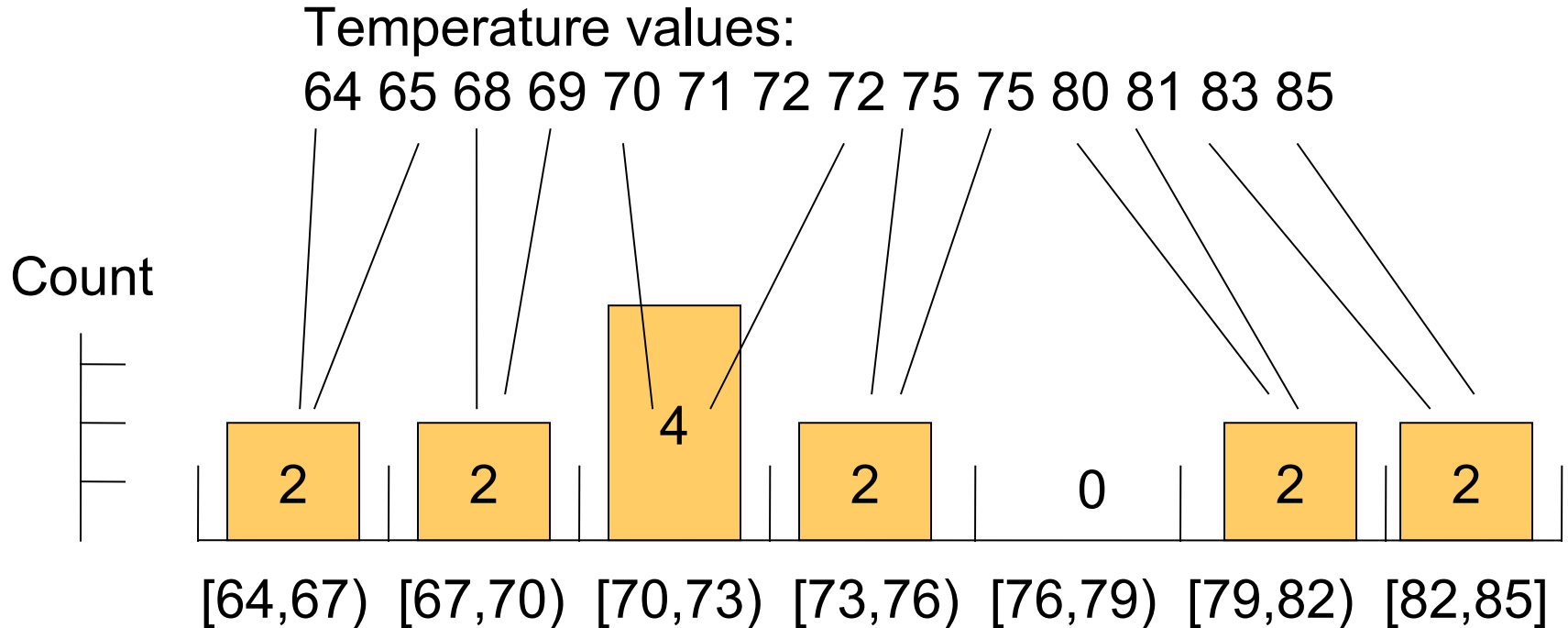
ID	C_red	C_orange	C_yellow	...
371	1	0	0	
433	0	0	1	

Conversion: Nominal to Numeric

- Many values:
 - US State Code (50 values)
 - Profession Code (7,000 values, but only few frequent)
- Approaches:
 - manual, with background knowledge
 - e.g., group US states
- Use binary attributes
 - then apply dimensionality reduction (see later today)

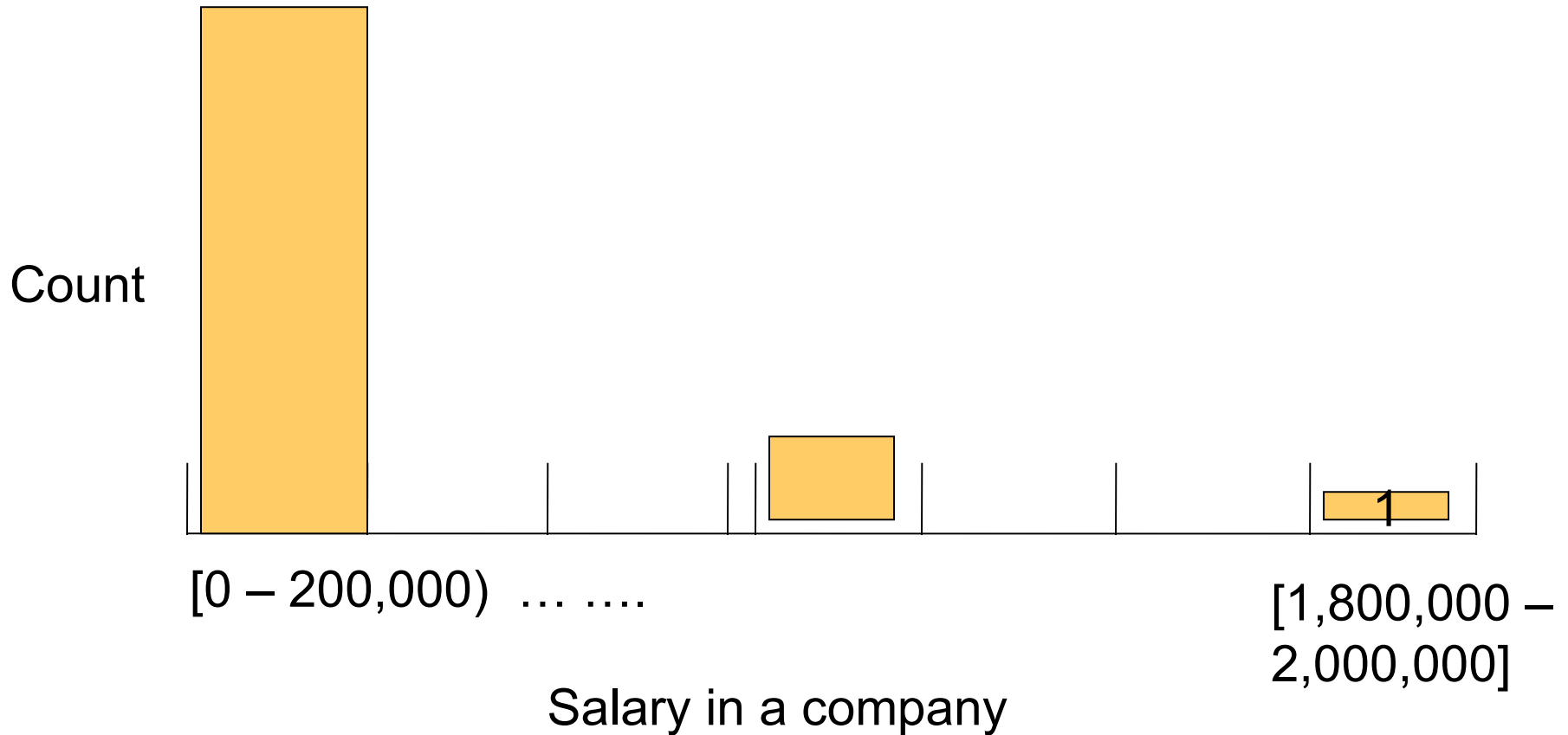


Discretization: Equal-width



Equal Width, bins $\text{Low} \leq \text{value} < \text{High}$

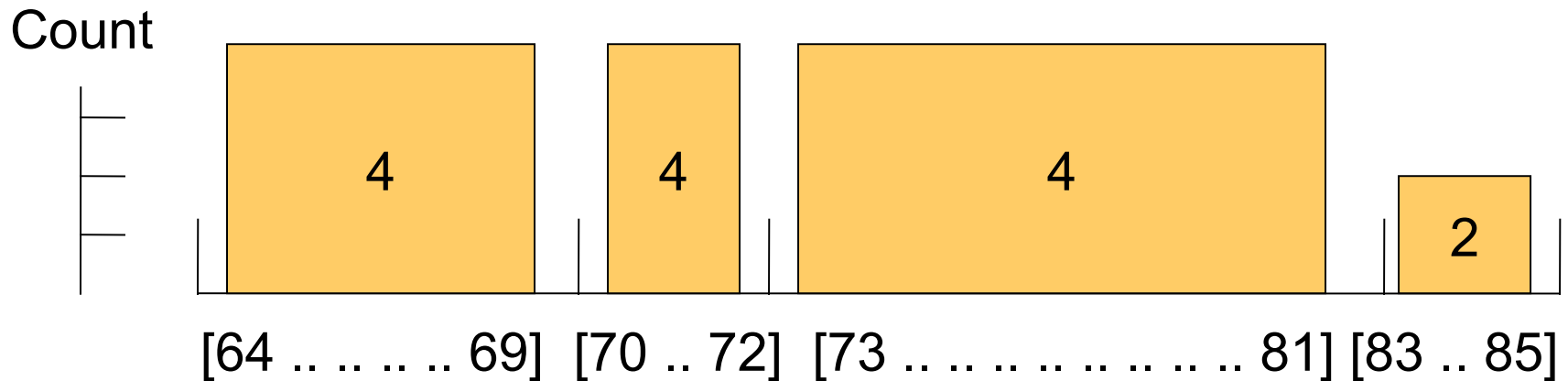
Discretization: Equal-width



Discretization: Equal-height

Temperature values:

64 65 68 69 70 71 72 72 75 75 80 81 83 85



Equal Height = 4, except for the last bin

Discretization by Entropy

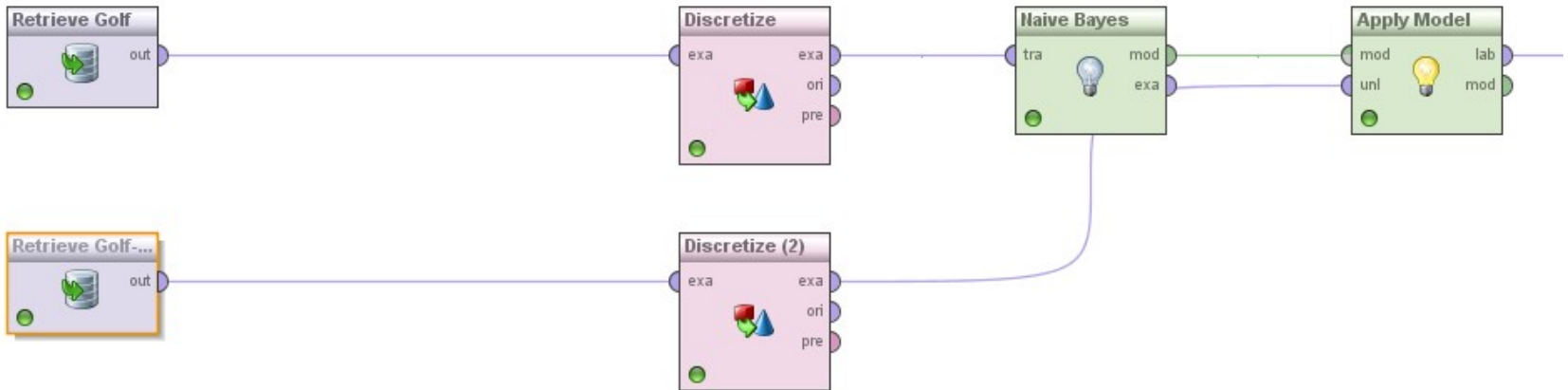
- Top-down approach
- Tries to minimize the entropy in each bin
 - Entropy: $-\sum p(x)\log(p(x))$
 - where the x are all the attribute values
- Goal
 - make intra-bin similarity as high as possible
 - a bin with only equal values has entropy=0
- Algorithm
 - Split into two bins so that overall entropy is minimized
 - Split each bin recursively as long as entropy decreases significantly

Discretization: Training and Test Data

- Training and test data have to be equally discretized!
- Learned rules:
 - income=high → give_credit=true
 - income=low → give_credit=false
- Applying rules
 - income=low has to have the same semantics on training and test data!
 - Naively applying discretization will lead to different ranges!

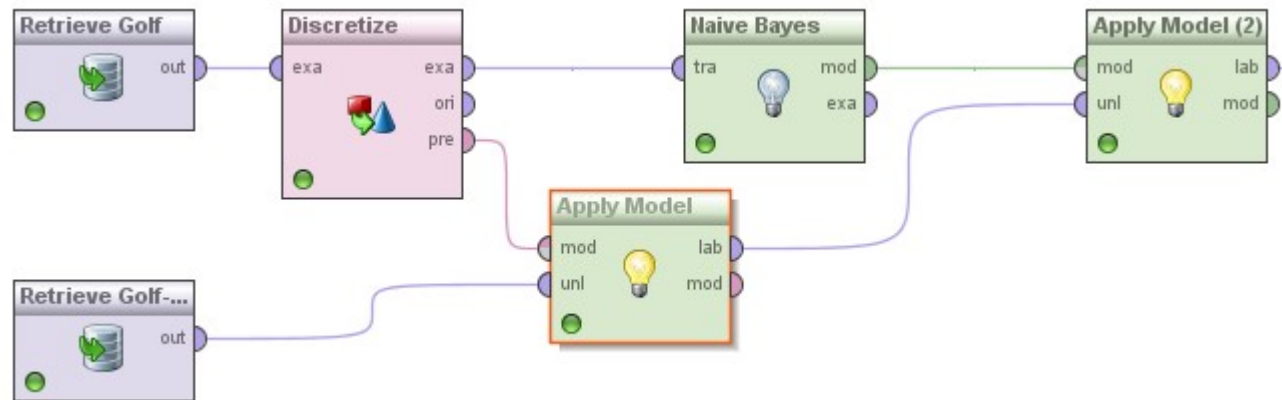
Discretization: Training and Test Data

- Wrong:



Discretization: Training and Test Data

- Right:



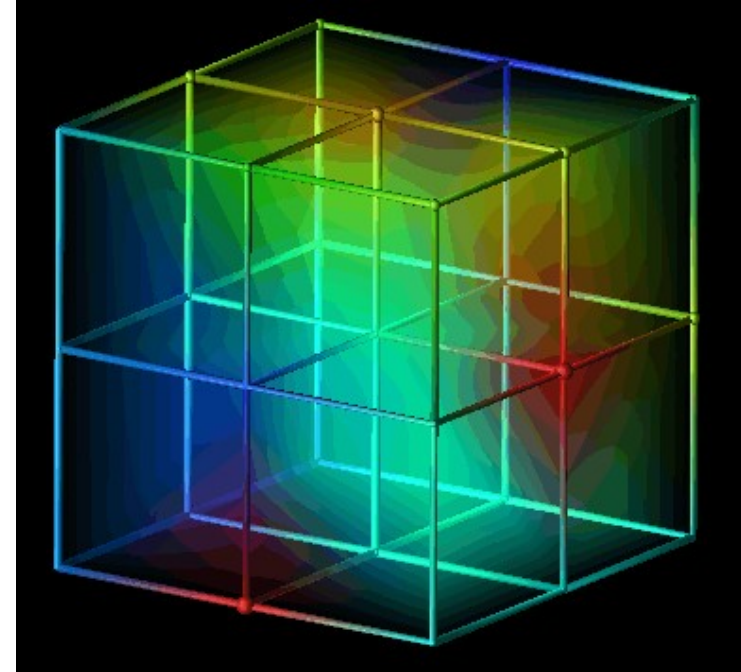
- Accuracy in this example, using equal frequency (three bins):
 - wrong: 42.7% accuracy
 - right: 50% accuracy
- Python: `fit` discretizer on training set, `transform` test set

Dealing with Date Attributes

- Dates (and times) can be formatted in various ways
 - first step: normalize and parse
- Dates have lots of interesting information in them
- Example: analyzing shopping behavior
 - time of day
 - weekday vs. weekend
 - begin vs. end of month
 - month itself
 - quarter, season
- Python: use, e.g., `datetime`

High Dimensionality

- Datasets with large number of attributes
- Examples:
 - text classification
 - image classification
 - genome classification
 - ...
- (not only a) scalability problem
 - e.g., decision tree: search all attributes for determining one single split



Curse of Dimensionality

- Learning models gets more complicated in high-dimensional spaces
- Higher number of observations are needed
 - For covering a meaningful number of combinations
 - “Combinatorial Explosion”
- Distance functions collapse
 - i.e., all distances converge in high dimensions
 - Nearest neighbor classifiers are no longer meaningful

$$\textit{euclidean distance} = \sqrt{\sum_{k=1}^n (p_k - q_k)^2}$$

Feature Subset Selection

- Preprocessing step
- Idea: only use valuable features
 - “feature”: machine learning terminology for “attribute”
- Basic heuristics: remove nominal attributes...
 - which have more than $p\%$ identical values
 - example: millionaire=false
 - which have more than $p\%$ different values
 - example: names, IDs
- Basic heuristics: remove numerical attributes
 - which have little variation, i.e., standard deviation $< s$

Feature Subset Selection

- Basic Distinction: Filter vs. Wrapper Methods
- Filter methods
 - Use attribute weighting criterion, e.g., Chi², Information Gain, ...
 - Select attributes with highest weights
 - Fast (linear in no. of attributes), but not always optimal
- Example:
- `X_f = SelectKBest(chi2, k=20).fit_transform(X, y)`

Feature Subset Selection

- Remove redundant attributes
 - e.g., temperature in °C and °F
 - e.g., textual features “Barack” and “Obama”
- Method:
 - compute pairwise correlations between attributes
 - remove highly correlated attributes
- Recap:
 - Naive Bayes requires independent attributes
 - Will benefit from removing correlated attributes

Feature Subset Selection

- Wrapper methods
 - Use classifier internally
 - Run with different feature sets
 - Select best feature set
- Advantages
 - Good feature set for given classifier
- Disadvantages
 - Expensive (naively: at least quadratic in number of attributes)
 - Heuristics can reduce number of classifier runs

Feature Subset Selection

- Forward selection:

```
start with empty attribute set
do {
  for each attribute {
    add attribute to attribute set
    compute performance (e.g., accuracy)
  }
  use attribute set with best performance
} while performance increases
```

- An learning algorithm is used for computing the performance
 - cross validation is advised

Feature Subset Selection

- Searching for optimal attribute sets

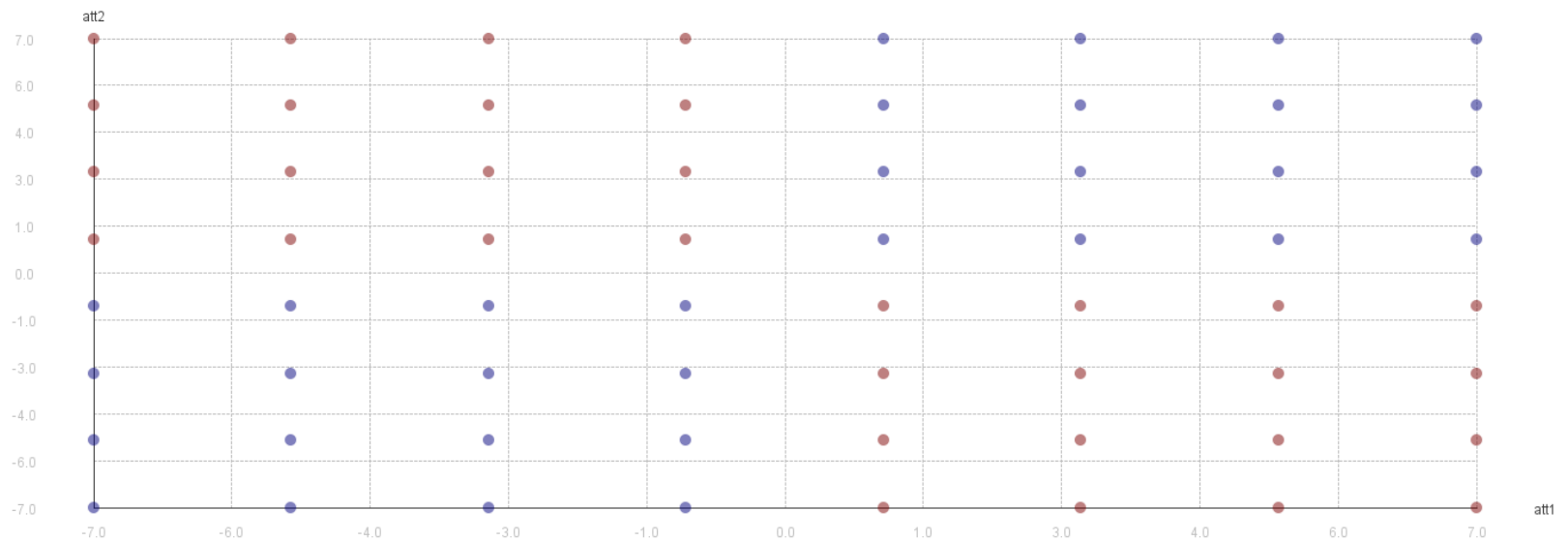
- Backward elimination:

```
start with full attribute set
do {
  for each attribute in attribute set {
    remove attribute to attribute set
    compute performance (e.g., accuracy)
  }
  use attribute set with best performance
} while performance increases
```

- An learning algorithm is used for computing the performance
 - cross validation is advised

Feature Subset Selection

- The checkerboard example revisited
 - Recap: Rule learners can perfectly learn this!
 - But what happens if we apply forward selection here?

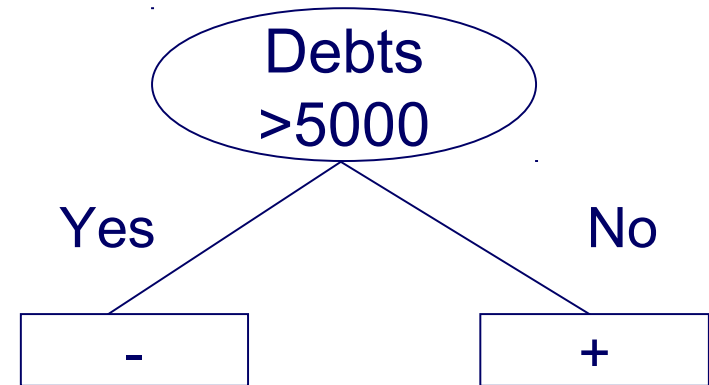


Feature Subset Selection

- Further approaches
 - Brute Force search
 - Evolutionary algorithms
(will be covered in parameter optimization session)
- Trade-off
 - simple heuristics are fast
 - but may not be the most effective
 - brute-force is most effective
 - but the slowest
 - forward selection, backward elimination, and evolutionary algorithms
 - are often a good compromise

Recap: Overfitting

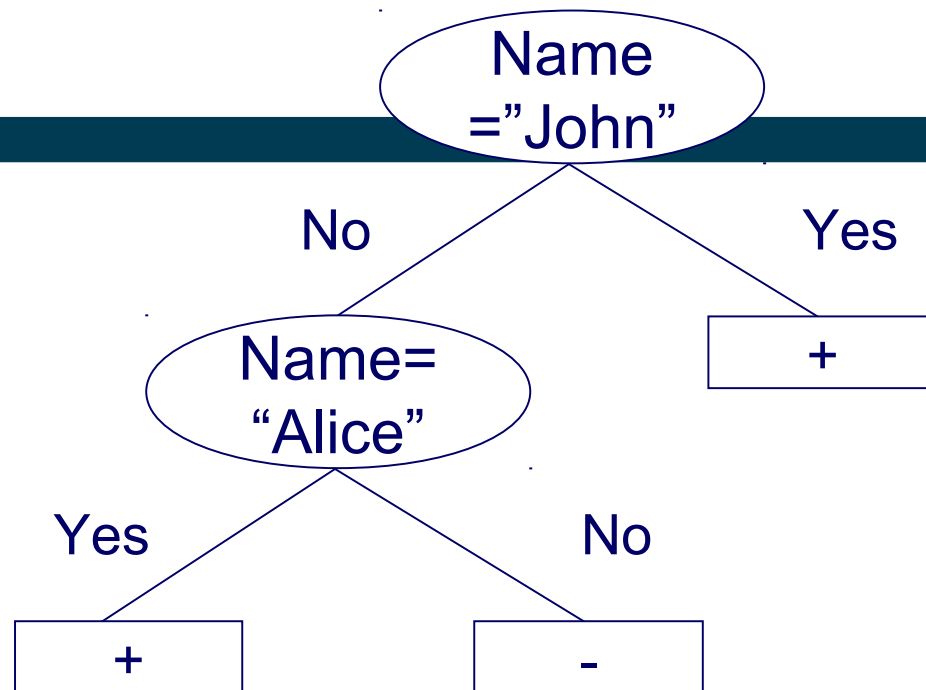
- Example: predict credit rating
 - possible decision tree:



Name	Net Income	Job status	Debts	Rating
John	40000	employed	0	+
Mary	38000	employed	10000	-
Stephen	21000	self-employed	20000	-
Eric	2000	student	10000	-
Alice	35000	employed	4000	+

Recap: Overfitting

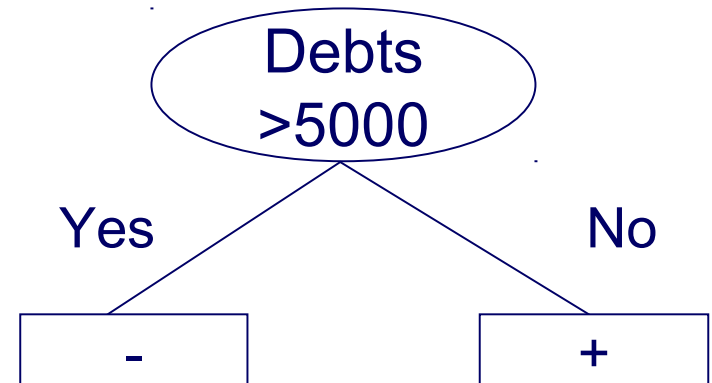
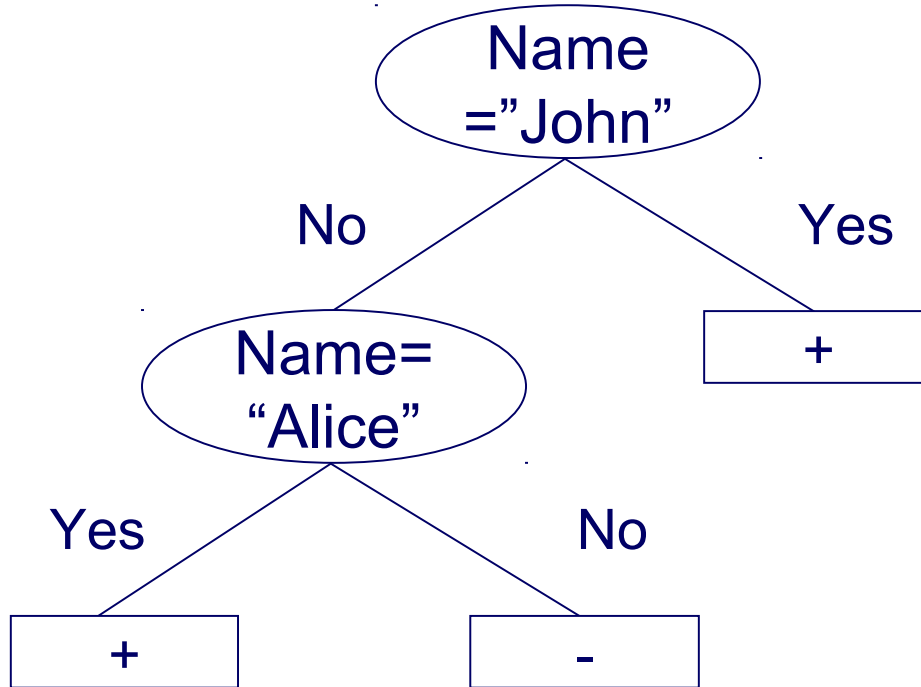
- Example: predict credit rating
 - alternative decision tree:



Name	Net Income	Job status	Debts	Rating
John	40000	employed	0	+
Mary	38000	employed	10000	-
Stephen	21000	self-employed	20000	-
Eric	2000	student	10000	-
Alice	35000	employed	4000	+

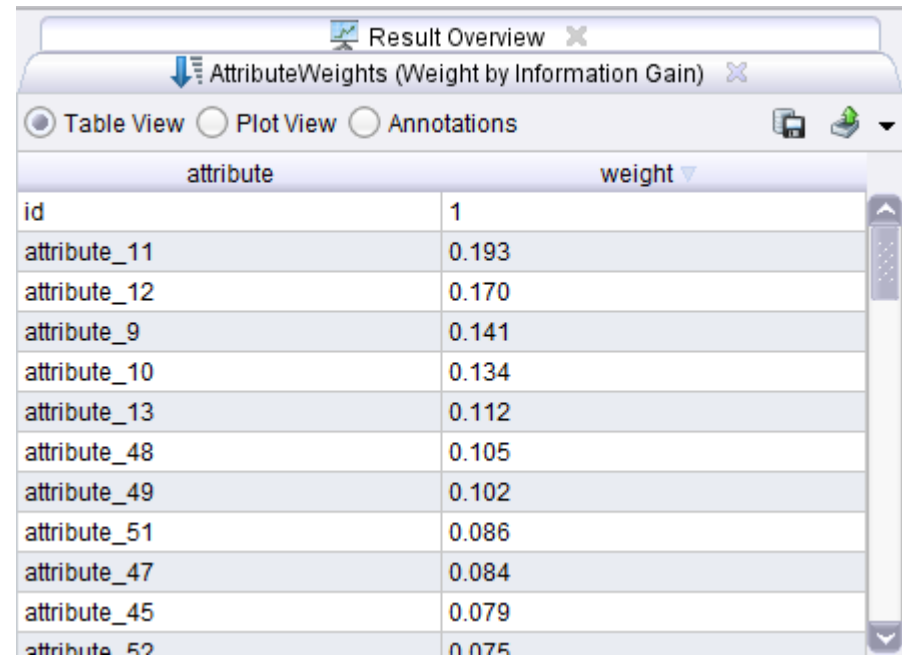
Recap: Overfitting

- Both trees seem equally good
 - Classify all instances in the training set correctly
 - Which one do you prefer?



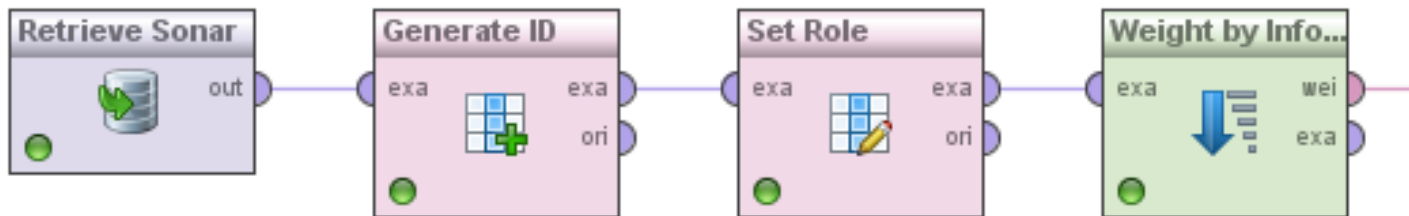
Recap: Overfitting

- Overfitting can happen with feature subset selection, too
 - Here, *name* seems to be a useful feature
 - ...but is it?
- Remedies
 - Hard for filtering methods
 - e.g., *name* has highest information gain!
 - Wrapper methods:
 - use cross validation inside!



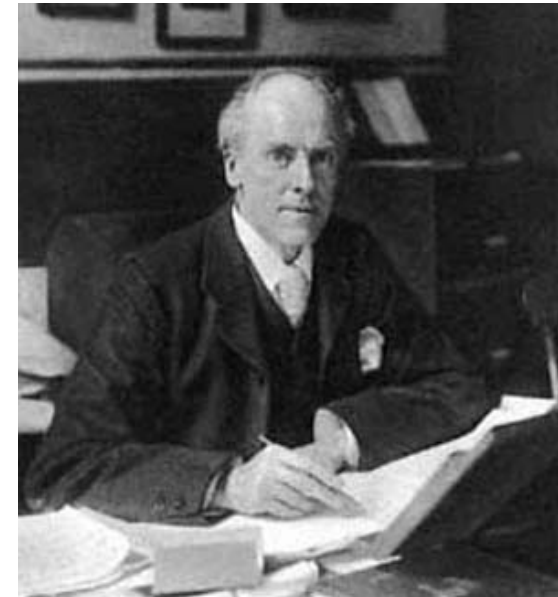
The screenshot shows a window titled 'Result Overview' with a sub-window 'AttributeWeights (Weight by Information Gain)'. The sub-window has three view options: 'Table View' (selected), 'Plot View', and 'Annotations'. Below the view options is a table with two columns: 'attribute' and 'weight'. The table lists various attributes and their corresponding weights.

attribute	weight
id	1
attribute_11	0.193
attribute_12	0.170
attribute_9	0.141
attribute_10	0.134
attribute_13	0.112
attribute_48	0.105
attribute_49	0.102
attribute_51	0.086
attribute_47	0.084
attribute_45	0.079
attribute_52	0.075



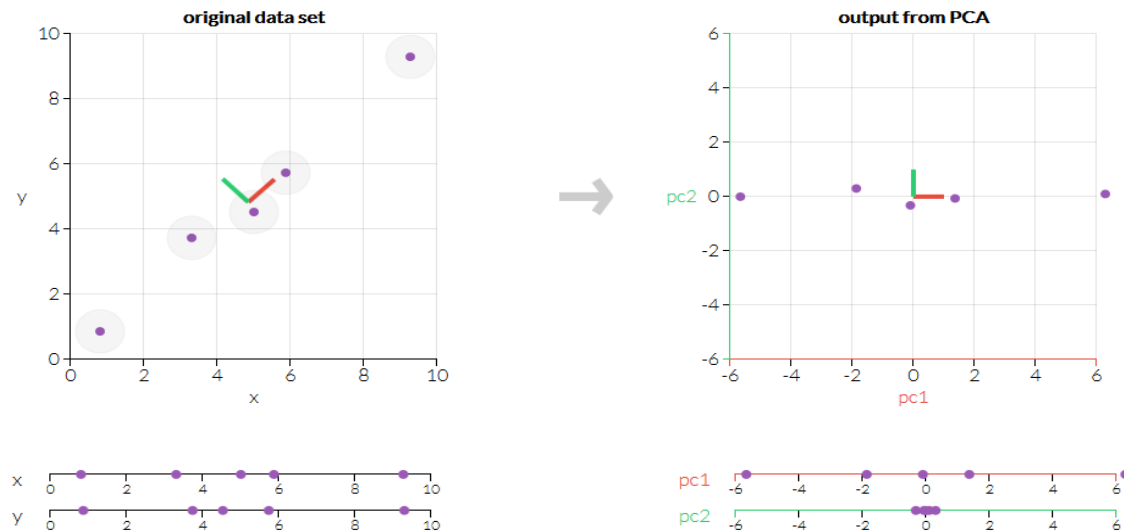
Principal Component Analysis (PCA)

- So far, we have looked at feature selection methods
 - we select a subset of attributes
 - no new attributes are created
- PCA creates a (smaller set of) new attributes
 - artificial linear combinations of existing attributes
 - as expressive as possible
- Dates back to the pre-computer age
 - invented by Karl Pearson (1857-1936)
 - also known for Pearson's correlation coefficient



Principal Component Analysis (PCA)

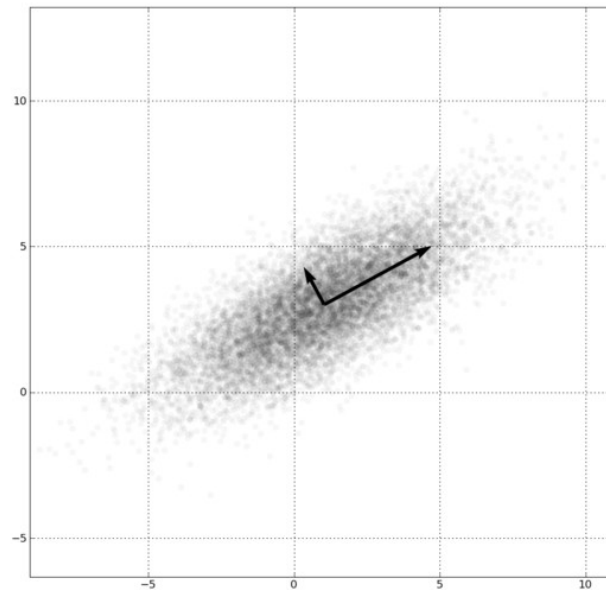
- Idea: transform coordinate system so that each new coordinate (principal component) is as expressive as possible
 - expressivity: variance of the variable
 - the 1st, 2nd, 3rd... PC should account for as much variance as possible
 - further PCs can be neglected



<http://setosa.io/ev/principal-component-analysis/>

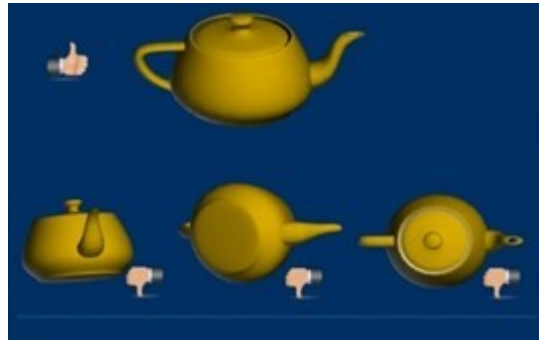
Principal Component Analysis

- Method used for computation:
 - Compute covariance matrix
 - Perform eigenvector factorization
 - See lecture: “Data Mining and Matrices”



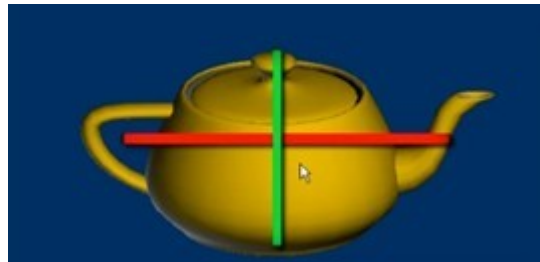
Principle Component Analysis illustrated

- Example by James X. Li, 2009
- Which 2D projection conveys most information about the teapot?



Approach:

- find longest axis first
 - in practice: use average/median diameter to limit effect of outliers
- fix that axis, find next longest



Sampling revisited

- Feature Subset Selection reduces the *width* of the dataset
- Sampling reduces the *height* of the dataset
 - i.e., the number of instances
- Trade-off
 - Maximum usage of information
 - Fast computation
- Notes
 - *Stratified* sampling respects class distribution
 - *Kennard-Stone* sampling tries to select heterogenous points

Kennard-Stone Sampling

- 1) Compute pairwise distances of points
 - 2) Add points with largest distance from one another
 - 3) While target sample size not reached
 - 1) For each candidate, find smallest distance to any point in the sample
 - 2) Add candidate with largest smallest distance
- This guarantees that heterogeneous data points are added
 - i.e., sample gets more diverse
 - includes more corner cases
 - but potentially also more outliers
 - distribution may be altered

Sampling Strategies and Learning Algorithms

- There are interaction effects
- Some learning algorithms rely on distributions
 - e.g., Naive Bayes
 - usually, stratified sampling works better
- Some rely less on distributions
 - and may work better if they see more corner cases
 - e.g., Decision Trees

Titanic Dataset
Filter: 50 training examples

	Decision Tree	Naive Bayes
Stratified	.727	.752
Kennard Stone	.742	.721

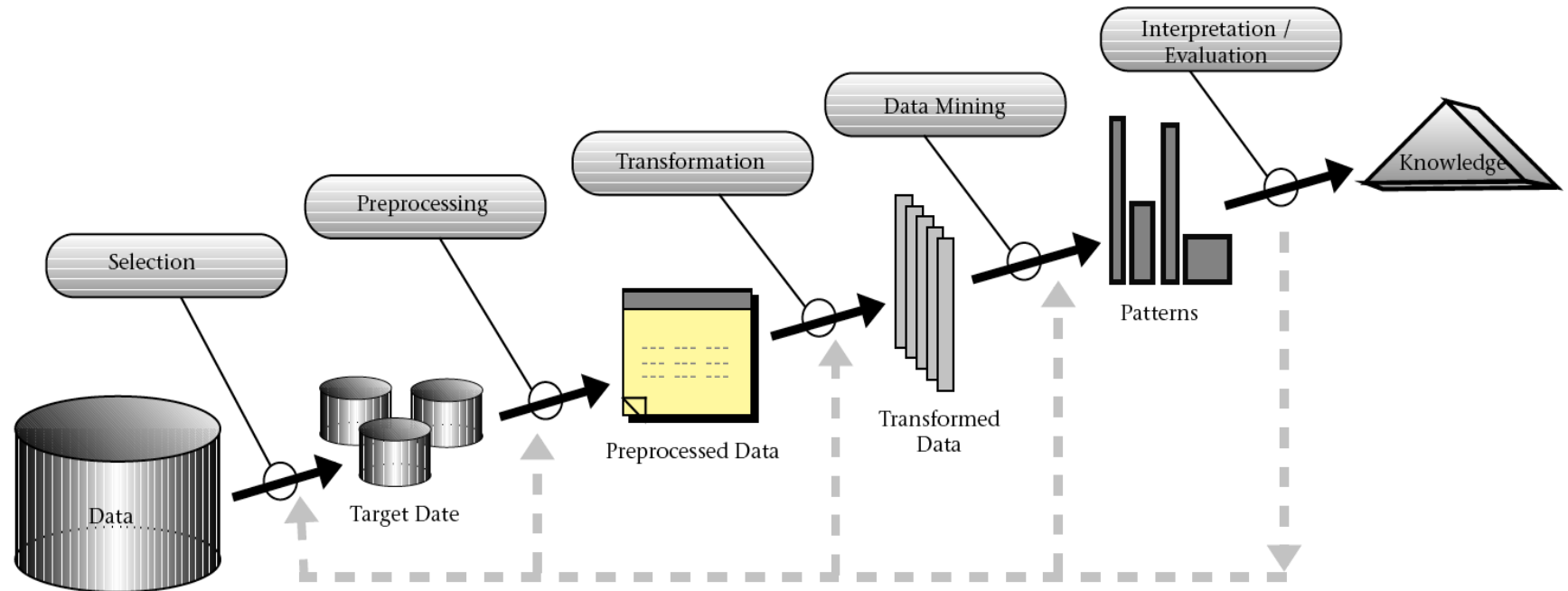
A Note on Sampling

- Often, the training data in a real-world project is already a sample
 - e.g., sales figures of last month
 - to predict the sales figures for the rest of the year
- How representative is that sample?
 - What if last month was December? Or February?
- Effect known as *selection bias*
 - Example: phone survey with 3,000 participants, carried out Monday, 9-17
 - Thought experiment: effect of selection bias for prediction, e.g., with a Naive Bayes classifier

Summary Data Preprocessing

- Raw data has many problems
 - missing values
 - errors
 - high dimensionality
 - ...
- Good preprocessing is essential for good data mining
 - one of the first steps in the pipeline
 - requires lots of experimentation and fine-tuning
 - often the most time consuming step of the pipeline

Recap: The Data Mining Process



Source: Fayyad et al. (1996)

Questions?

