Lecture Notes

Managing and Mining Multiplayer Online Games

Summer semester 2017

Chapter 6: Data Analytics in a Nutshell

http://www.dbs.ifi.lmu.de/cms/VO_Managing_Massive_Multiplayer_Online_Games

Overview

- What is Knowledge Discovery and Data Mining?
- KDD Process
- Supervised Learning
  - Classification
  - Prediction
- Unsupervised Learning
  - Clustering
  - Outlier Detection
- Frequent Pattern Mining
  - Frequent Itemsets

Definition: Knowledge Discovery in Databases

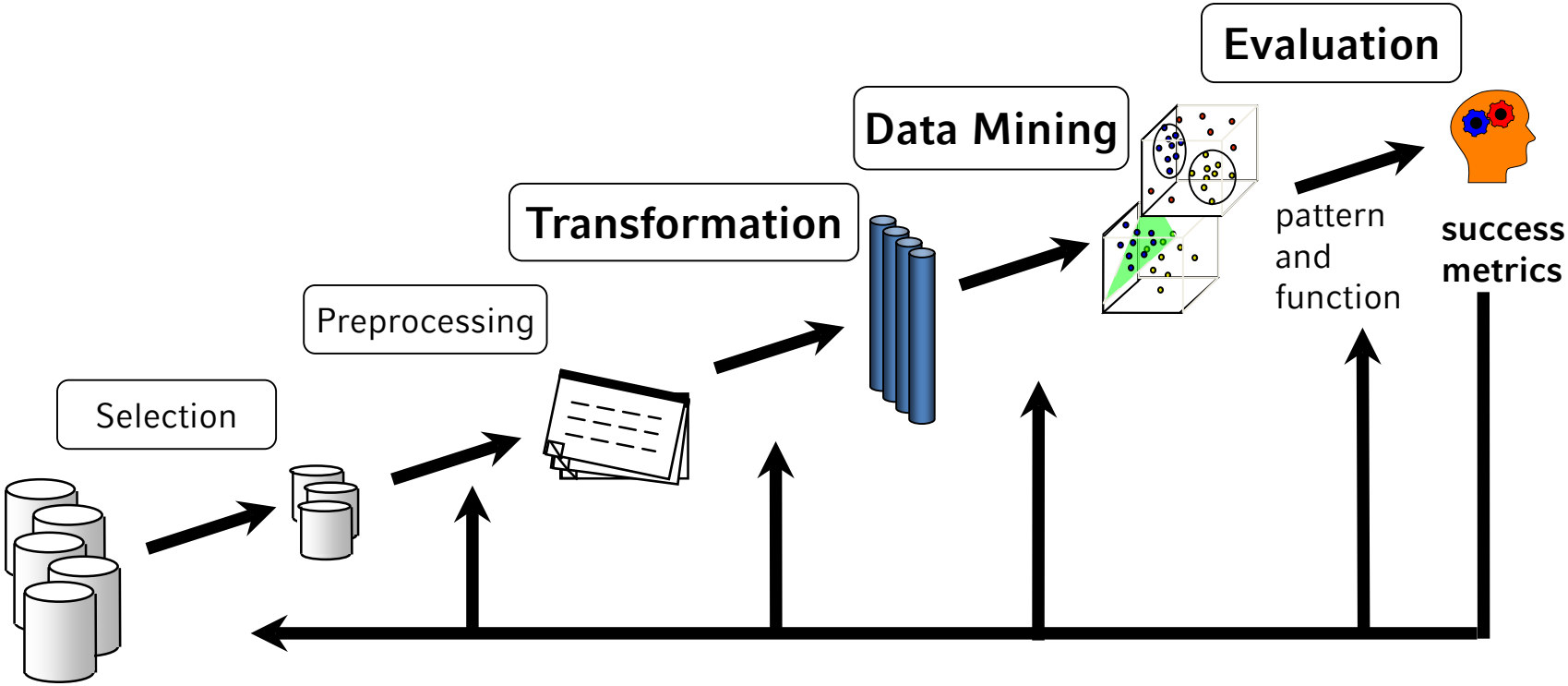
[Fayyad, Piatetsky-Shapiro & Smyth 1996]

Knowledge Discovery in Databases (KDD) is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.

Remarks:

- *valid*: in a statistic sense.
- *novel*: not explicitly known yet,
  no common sense knowledge.
- p*otentially useful*: for a given application.
- *ultimately understandable*: the end user should be able to interpret the patterns either immediately or after some postprocessing

Knowledge Discovery Process



- Knowledge Discovery is a process comprising several steps.
- The KDD process is iteratively optimized (back arrow) until the result is acceptable.
- It is important what's the purpose of the analysis.

Steps of a KDD-Process

- **Selection**: Determining a clear objective and approach.
  Example: Use of a recording of TCP-Traffic to train a prediction model, which recognizes if a player is controlled by a bot.

- **Preprocessing**: Selection, integration and ensuring consistency of data to analyze.
  Example: Saving records of normal players' and bots' network traffic. Integration of data from several servers. Elimination of too short or useless records (permanently AFK).

Steps of a KDD-Process

- **Transformation**: Transforming date into an analyzable form.
  Example: Create a vector from average package rates, length and burstiness key-figures.

- **Data Mining**: Use efficient algorithms to derive statistically significant patterns and functions from transformed data.
  Example: Training of a neural network with examples for bots and human players, to predict a new record if it is a bot.

Steps of a KDD-Process

- **Evaluation**: test the quality of the patterns and functions gained from data mining.
  - Compare expected and predicted results.
    (Rate of error)
  - Manual evaluation by experts (Does the result make sense?)
  - Evaluation based on mathematical characteristics of patterns

  Example: Testing an independent set of test-recordings on how likely the neural network predicts a bot with more than 50% confidence.

- **Conclusion**:
  - If test results are unsatisfying, the process is adapted.
  - Adaption is possible in every step: more training data, different algorithms, different parameters, …

Prerequisites for Application

- **Patterns and Frequency**
  - Patterns have to exist in some way and must be recognizable.
  - Data should be correlated to the desired outcome.
- **Generalization and Overfitting**
  - Transferring knowledge to new objects requires comparability / similarity to already analyzed data.
  - The less information describes an object, the more objects are comparable. The more properties are considered, the more different objects become.

- **Valid in a statistic sense**
  - Knowledge has room for errors => no absolute rules.
  - Useful knowledge does not need to be 100% correct, it needs to be significantly better than guessing.

Overfitting

Over adaption of models to given data objects

=> insufficient transferability to other data objects

Factors favoring overfitting:

- ***Complexity of object description***: The more information are available, the less likely two objects are similar to each other.
- ***Specificity of attribute values***: The more unique an attribute value, the less it contributes to differentiate many elements by similarities. (example: Object_ID)
- **Model complexity**: The more complex a function or a pattern, the easier it adapts to the given objects.

Goal: Model, attributes and object description should not describe one individual, but all objects belonging to the same pattern (class, cluster).

Featurespace, Distance and Similarity Measure

Similarity: Objects that are comparable within the context.

Example: 2 Players, who are controlled by the same bot are likely to create similar network-traffic.

- **Feature Space**: data mining algorithms' perspective on objects. (Features, Structure, Values range, …)
- **Similarity Measure**: calculates similarity based on feature-space. (the higher, the more similar)
- **Distance Value**: calculates difference between two object descriptions. (the higher, the more dissimilar)

IMPORTANT: Feature Space and Similarity Measure are dependent:

- Changing the feature space changes the result of the measure.
- Similarity measure may only use parts of the description or may recombine existing elements.
  (equivalent to transforming the freature space)

Formal definition of distance function

Distance function: Let ***F*** be a feature space.
$dist : F \times F \rightarrow \mathbb{R}^+_0$ is called a **distance function** if the following properties hold:

- $\forall p,q \in F,\ p \neq q : dist(p,q) > 0$ strictness
- $\forall o \in F: dist(o,o) = 0$ reflexivity
- $\forall p,q \in F: dist(p,q) = dist(q,p)$ symmetry

Additionally, if

$\forall o,p,q \in Dom : dist(o,p) \leq dist(o,q) + dist(q,p)$ triangle inequality

holds, dist ist called a **metric**.

Vectors as Object Presentation

FeatureVectors: standard representation in most algorithms

basic idea:

- **feature**: property describing an object.
  example: average packages per second
- **types of features**:
  - nominal: equality and inequality (example: name)
  - ordinal: values are ordered (example: position in ranking)
  - numerical: differences of values are quantifiable
    (level (discrete), package-rate (metric), …)

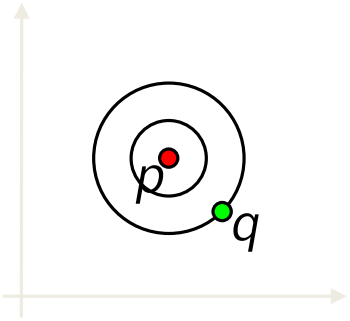
- **Feature Vector**: Set of all describing features
  example:(name, guild rank, level, package rate, package size)
- There is a variety of algebraic functions and laws usable for analysis of purely numerical data.

The L_p-Metrics

Let $p,q \in IR^d$:

Euclidian Norm (L_2):

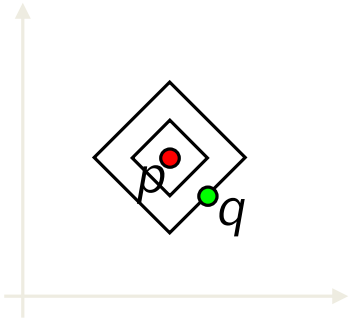
$$d_2(p,q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}$$



natural distance

Manhattan-Norm (L_1):

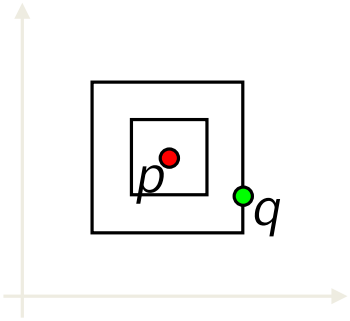
$$d_1(p,q) = \sum_{i=1}^{d} |p_i - q_i|$$



sum of the absolute differences

Maximums-Norm (L_∞):

$$d_\infty(p,q) = \max_{1 \le i \le d} |p_i - q_i|$$

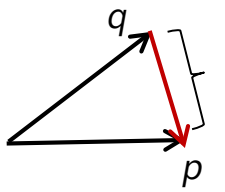


dissimilarity of the least similar feature is relevant

General formula for L_p-distance: $d_p(p,q) = \left(\sum_{i=1}^{d} |p_i - q_i|^p \right)^{\frac{1}{p}}$

Norms, Similarity and Kernel Functions

- *euklidian distance*: length of vector difference
- *length of a vector*: norm of the vector $\|q-p\|$
- *norm: square root of self* inner product
- properties of the inner product:



$$\|\vec{x}\| = \sqrt{\langle \vec{x}, \vec{x} \rangle} = \sqrt{\sum_{i=1}^{d} x_i \cdot x_i}$$

$$\langle \cdot, \cdot \rangle : F \times F \rightarrow IR,$$

$$\langle c \cdot x, y \rangle = \langle x, y \cdot c \rangle = c \langle x, y \rangle$$

$$\langle x, y \rangle = \langle y, x \rangle$$

$$\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$$

- connection between inner product, norms and metrics:

$$\|q-p\| = \langle q-p, q-p \rangle^{\frac{1}{2}} = \left(\langle q, q \rangle + \langle p, p \rangle - 2\langle q, p \rangle \right)^{\frac{1}{2}}$$

(inner products imply norms and norms imply metrics)

- inner products are often used as similarity measures.
(in this context they are called ***kernel*** functions.)

Supervised Learning

Idea: Learning from example objects to optimise a predictive function.

given:

- target variable *C*
  (*classification*: Set of nominal values, *regression*: numerical Values)
- objects: *DB* $\in$ *F*×*C*: *Object o*=(*o.v*,*o.c*) $\in$ *DB*
- training set: T $\subseteq$ DB of which *o* is fully known.

goal: function *f*: *F* $\rightarrow$ *C*, mapping object representation to values of the target variable with minimal error.

error function: *q*uantifies the quality of the model on T.

square loss/ quadratic error: $L^2(f,T) = \sum_{o \in T} \left(o.c - f(o.v)\right)^2$

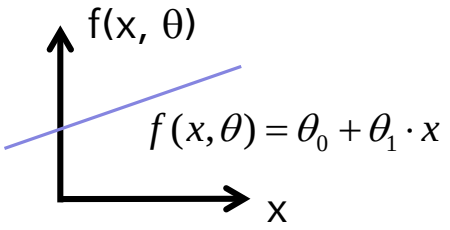
absolute error: $L_{abs}(f,T) = \sum_{o \in T} \left(o.c = f(o.v) ? 1 : 0\right)$

Training Supervised Methods

- given the type of the function *f* ,e.g., linear model
- *adapt f* to training set *T* by modifying paramter θ

example: *f* univariate linear function:

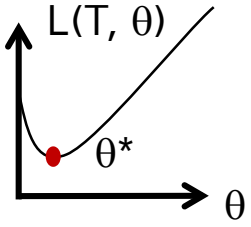
$$f(x,\theta) = \theta_0 + \sum_{i=1}^{d} \theta_i \cdot x_i$$



solution 2D case

training: minimize loss function

- Loss function *L describes the error of f for T*
- search parameters θ* minimizing *L*
- ***approach***: build the gradiet of L for θ and compute the minimum θ*.



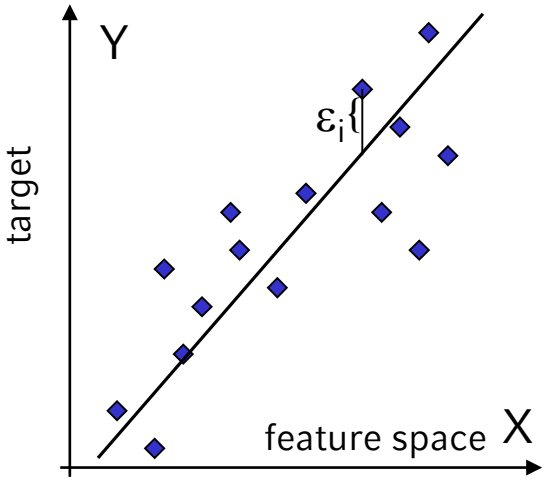
=> convex loss functions are beneficial (the only extremum is the minimum)

=> general loss functions might have multiple local minima and training can get stuck at suboptimal parameters

Example: 2D linear regression

given: Trainingsmenge $T \in IR^2$

model: $f(x,\theta) = \theta_0 + \theta_1 \cdot x$



loss function:

$$L^2(f,T) = \sum_{(x,y)\in T} \left(y - f(x,\theta)\right)^2 = \sum_{(x,y)\in T} \left(y^2 + f(x,\theta)^2 - 2y \cdot f(x,\theta)\right)$$

$$= \sum_{(x,y)\in T} \left(y^2 + (\theta_1 x)^2 + \theta_0^2 + 2\theta_0\theta_1 x - 2y\theta_0 - 2yx\theta_1\right)$$

gradient for θ_0: $\dfrac{\partial L^2}{\partial \theta_0} = \sum_{(x,y)\in T} \left(2\theta_0 - 2y + 2\theta_1 x\right) = 2\left(\theta_0 |T| - \sum_{(x,y)\in T} y + \theta_1 \sum_{(x,y)\in T} x\right)$

$$\frac{\partial L^2}{\partial \theta_0} = 0: \quad \theta_0 = \frac{\sum_{(x,y)\in T} y - \theta_1 \sum_{(x,y)\in T} x}{|T|} = E(y) - \theta_1 E(x)$$

gradient for θ_1:

$$\frac{\partial L^2}{\partial \theta_1} = \sum_{(x,y)\in T} \left(2\theta_1 x^2 - 2yx + 2\theta_0 x\right) = 2\left(\theta_1 \sum_{(x,y)\in T} x^2 + E(y) \sum_{(x,y)\in T} x - \theta_1 E(x) \sum_{(x,y)\in T} x - \sum_{(x,y)\in T} yx\right)$$

$$\frac{\partial L^2}{\partial \theta_1} = 0: \quad \theta_1 = \frac{E(y) \sum_{(x,y)\in T} x - \sum_{(x,y)\in T} yx}{\sum_{(x,y)\in T} x^2 - E(x) \sum_{(x,y)\in T} x} = \frac{Cov(x,y)}{Var(x)}$$

Further Comments on Supervised Learning

- **regularization**: Often parameters θ can grow unrestricted allowing overfitting.
  => integrate regularization term to restrict the allowed solutions

  example: linear ridge regression

regularization

$$L^2(f,T) = (1-\alpha\cdot)\sum_{o\in T}\left(o.c - \theta_0 + \sum_{i=1}^{d}\theta_i \cdot o.v_i\right)^2 + \alpha\cdot\|\theta\|^2$$

- often optimization are more complicated by considering constraints (quadratic programs, semi definite programs, …)
- loss functions do not have to be convex (neural networks)

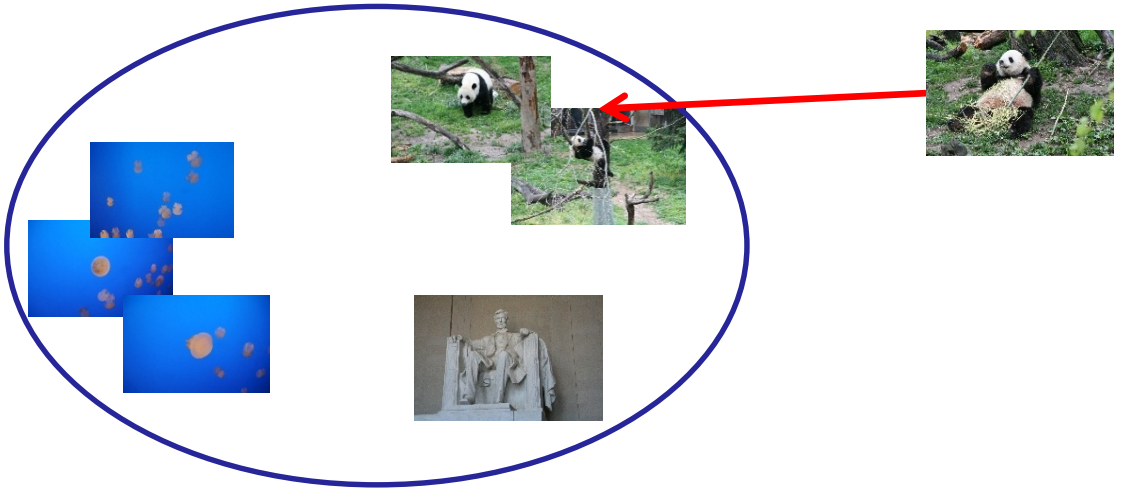
  => optimization minimizes the loss until local convergence
- there are other approaches to supervised learning not minimizing a loss function, e.g., maximize the likelihood,

Instance-based Learning

idea: search the most similar objects in training set T and a use their target variables to estimate the target value.

two components:

- decision set of similar training objects:
  - depends on similarity/distance measure (k-nearest neighbors in *T*)
  - size of decision set *k describes the generalization of the method*
- compute the predicton
  - use majority vote (classification)/ mean (regression)
  - distance weighted votes (*e.g., quadratic inverse weighting: $1/d(q,x)^2$)*



Bayesian Learning

idea: each observation is generated by a hidden statistical distribution/process.

Given a set of these distributions allows to determine the most likely explanation for any new observation.

***example**: Bot-Detection*

given: model *A*: humanplayer, model B: Bot player

observation *v* (vector describing network traffic)

assumption: *v* follows either *A* or *B*.

task: compute the likelihood that v was generated by B.

solution: compute *P(B|v) = likelihood of B given that v was already observed*

caution: do not confuse with *P(v|B)* = likelihood that B generates vector v
It might be very unlikely that B exactly generates v.

Rule of Bayes

How to compute the likelihood of B generating the given observation v.

- *we assume p(v) =p(A)·p(v|A)+p(B)·p(v|B),*
  here: *P(B), P(A) are called prior probabilities describing the general ratio of instances from B and A. (How much Bots are out there?)*

=> the above formula implies that even if *p(v|A) < p(v|B)* it might be more likely that *v* is caused by a bot because bots might be very rare.

Generally:

- rule of Bayes*:* $P(B \mid v) = \frac{P(B) \cdot P(v \mid B)}{P(v)}$

- for all distributions C and observation *v* it holds: $\sum_{c \in C} P(c \mid v) = 1$

  (the observation has to follow a known model)
- therefore, $c^* = \arg\max_{c \in C}\left(P(c \mid v)\right)$ is the most likely distribution (class/value)

Training Bayes Classifiers

- prior distribution *P(c) are approximated as the ration of class members in the training set T*

  (17 out of 100 traffic snippets were generated by bots: *P(B)* =17%)
- to compute *p(v|c)* we assume a certain type of distribution
- in the most simple case training is done be computing relative probabilties in T.

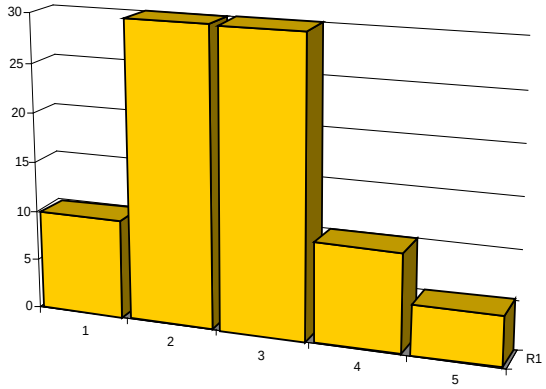
example: consider two dices

- possible results: {1, 2, 3, 4, 5, 6}
- dice D1 is uniform distributed: 1/6 for all number from 1 to 6
- distribution for dice D2 : 1: 1/12, 2: 1/12, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/3
- *p(v=1|D1) = 1/6, p(v=6|D2)=1/3*
- given: *P(D1)= 0.2* und *P(D2)=0.8:*

$$P(D2\,|\,5) = \frac{0{,}8 \cdot 0{,}1\overline{6}}{0{,}2 \cdot 0{,}1\overline{6} + 0{,}8 \cdot 0{,}1\overline{6}} = 0.8; \quad P(D\,|\,6) = \frac{0{,}8 \cdot 0{,}\overline{3}}{0{,}2 \cdot 0{,}1\overline{6} + 0{,}8 \cdot 0{,}\overline{3}} = \frac{8}{9}$$

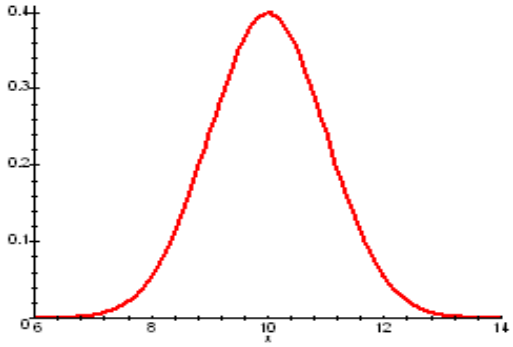
Univariate Distributions

- discrete probability spaces:
  - finite number of events
  - separate estimation for all basic events



- real valued distributions:
  - infinite number of events

    (each event has a probability 1/∞->0)
  - estimation of using density functions (e.g. normal distributions)
  - training = estimate parameters of the density function (e.g., mean and variance)
  - to compute probabilities from density functio either **integrate** over an interval of events or apply the **rule of Bayes** to determine relative densities.



Statistic Models

considering multiple features v_i requires joint estimates of $p(v_1, \ldots, v_d|c)$.

problem: How to consider correlations betweent $v_1, \ldots, v_d$?

- ***naive approach***: consider all objects as independent => naive Bayes

  pro: easy estimation and computation $P(v_1, ..., v_d \mid c) = \prod_{i=1}^{d} P(v_i \mid c)$

  con: limited expressiveness

- ***complete dependency***: estimate joint probabilities for all value combinations $(v_1, \ldots, v_d)$

  pro: any correlation might be considered

  con: number of possible events increases exponential in d
  => usually not enough training data
  => large models and slow training

- advances solutions allow to consider some correlations but not all (e.g., Bayes networks, graphical models,etc.)

Evaluating Supervised Learners

- optimization on the training data not inclusive

  $\Rightarrow$ generalization: how good does the method work on unknown data
- it is necessary to test classifiers and predictors on previously unknown and independent samples (**Train and Test**)

- **problem**: Usually, there is not enough labeled data providing a correct target value.
  => ground truth is rare
  => manual labeling is cumbersome and expensive

Testing Supervised Predictors

goal:

- apply train and test set to as many instances as possible
- avoid overfitting (train and test set are disjunctive)

- **Leave-One-Out**:
  - perform *n* tests for *n data objects*
  - each element is picked once for testing and the rest is used for training
  - results are reproducible
  - maximum test effort (requires to train n predictors)
  - only applicable to small data sets or instance-based methods

Stratified *k*-fold Cross Validation

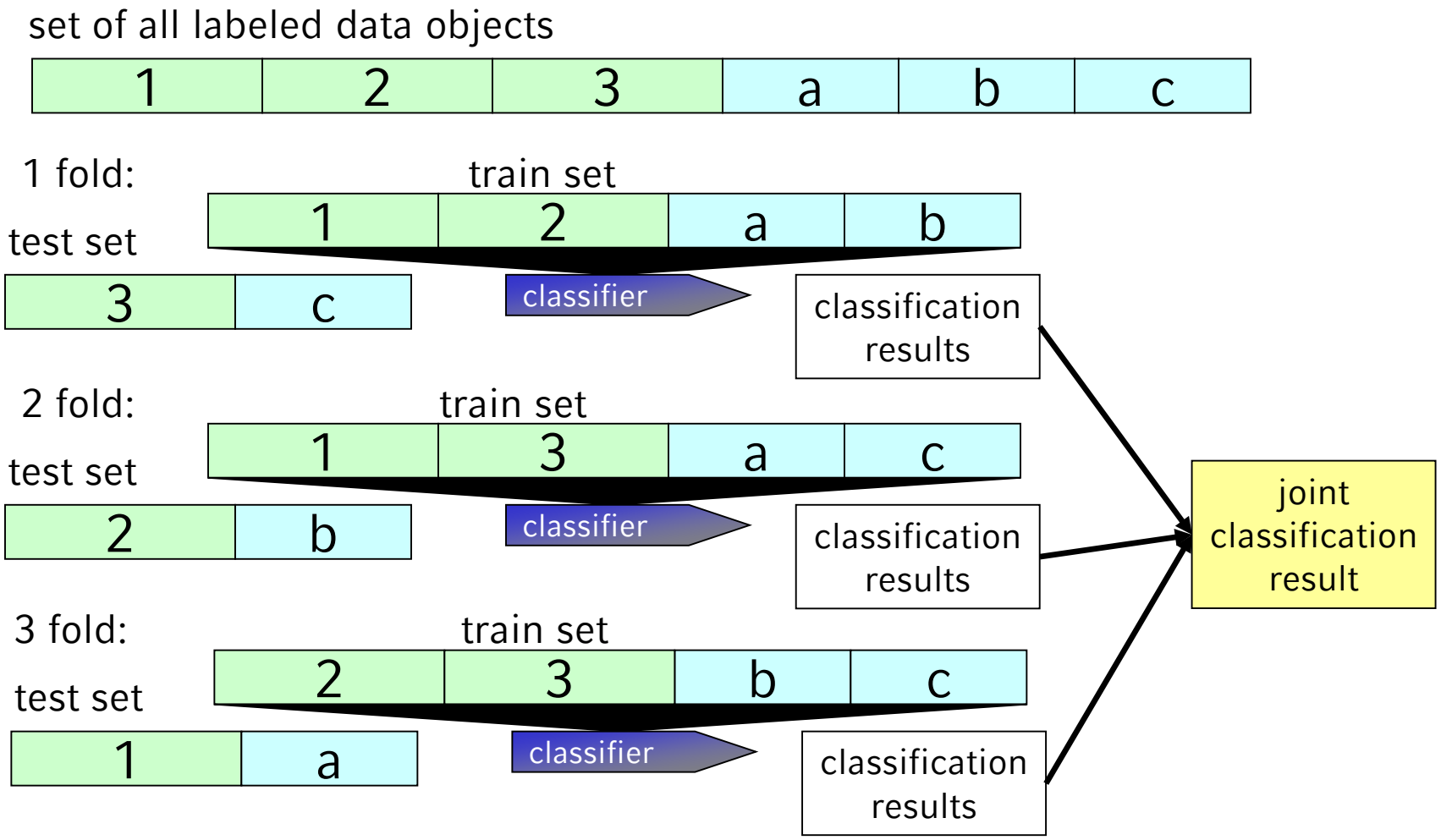
- similar to leave-one-out. Build k folds and perform leave-one-out on folds instead of instances
- **stratification**: the class distribution in each fold is the same as in the complete data set. (each class is approx. represented by the same number of objects in each fold)
- the number of folds k controls the effort (the larger the more effort)

- *result of k*-fold cross validation depends on the sampling of the folds
  => results can vary when shuffling the data

  => *k*-fold cross validation might be applied several times on different shufflings to avoid this effect

Example: Stratified 3-fold Cross Validation

green boxes: class 1 (folds:1, 2, 3)
blue boxes: class 2 (folds: a, b, c)



Evaluating Classification Results

raw test result: confusion matrix

classified as ...

real class label	class 1	class 2	class 3	classe 4	class 5
class 1	35	1	1	1	4
class 2	0	31	1	1	5
class 3	3	1	50	1	2
class 4	1	0	1	10	2
class 5	3	1	9	15	13

correct classifier objects

Based on the confusion matrix the following measures are derived:
classification accuracy, classification error, precision, recall, F1-measure

Classification Metrics

- let *f* be a classifier, *TR* be the training set, *TE* be the test set
- *o.c is the real class of o*bject *o*
- *f(o)* is the predicted class of o
- classification accuracy of f on *TE*:

$$G_{TE}(f) = \frac{|\{o \in TE | f(o) = o.c\}|}{|TE|}$$

- true classification error of *f* on *TE*:

$$F_{TE}(f) = \frac{|\{o \in TE | f(o) \neq o.c\}|}{|TE|}$$

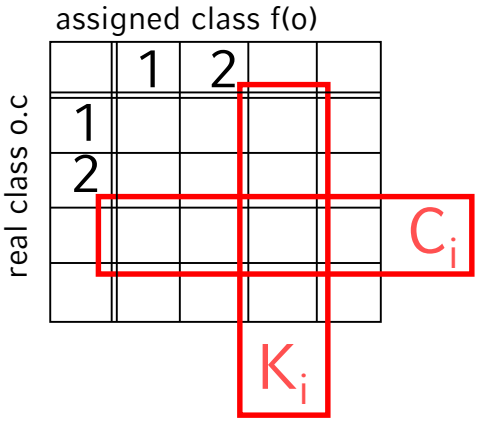
- apparent classification error on **TR** (used to determine overfitting)

$$F_{TR}(f) = \frac{|\{o \in TR | f(o) \neq o.c\}|}{|TR|}$$

Classification Metrics

- *Recall:*
  ratio of correctly classified instances of class *i*. *Let* $C_i = \{o \in TE \mid o.c = i\}$, then

$$Recall_{TE}(f,i) = \frac{|\{o \in C_i | f(o) = o.c\}|}{|C_i|}$$



- *Precision*:
  ratio of objects being correctly assigned to class i. Let $K_i = \{o \in TE \mid f(o) = i\}$, then

$$Precision_{TE}(f,i) = \frac{|\{o \in K_i | f(o) = o.c\}|}{|K_i|}$$

- *F1 score*:
  harmonic mean of precision and recall.

$$F1_{TE}(f,i) = \frac{2 \cdot Precision_{TE}(f,i) \cdot Recall_{TE}(f,i)}{Precision_{TE}(f,i) + Recall_{TE}(f,i)}$$

Unsupervised Learning

problem setting: only unlabeled objects/no classes or target values

tasks:

- find groups of similar objects. (Clustering)
- find uncommon objects. (Outlier Detection)
- find parts of objects which occure often (Pattern Mining)

pro:

- results are based on less assumptions
- no labeling required

con:

- measuring the results if often a problem (manual evaluation)
- more flexibility often implies more computational complexity
- correlating the result to the actual target is difficult without examples (how to guide the algorithm to achieve the goal of the process)

Example Applications

- **Clustering**: Determine typical tactics for a particular boss encounter.
- **Outlier Detection**: Which player might cheat?
- **Pattern Mining**: Determine standard rotations of abilitiy usage.

Clustering Methods

- identify a finite set of clusters or groups
- *similar objects should be part of the same cluster whereas dissimilar objects should be part of different clusters*
- clustering comprises finding the clusters and assigning new objects to these clusters

Clustering (formal view)

given:

- *dataset* $DB \subseteq F$ *(F is a feature space)*
- $C \subseteq IN_0$ a discrete target variable (cluster id)
- sometimes the number of clusters $|C|$ *is assumed to be known*

goal: find function $f: F \rightarrow C$ *assigning ojbects to clusters.*

find reasonable clusters(e.g. Minimize intra cluster distance and maximize distance between clusters)

quality of a clustering:

- depends on the cluster model:
  - How is an object assigned to a cluster?
  - How is decided whether two objects belong to the same custer?
- optimize:
  - compactness of clusters
  - cluster separation

Partitioning Clustering(1)

idea:

- there a k clusters and each cluster *c is represented by* o_c
- object *o* is assigned to *c* by the distance *dist*(o_c,*o*):
  $cluster(o) = \arg\min_{c \in C}(dist(o_c, o))$
- to achieve compact clusters minimize:
  - average distance of objects to the closest clusters:
    $$compact(c) = \sum_{o \in \{o \in DB | cluster(o) = c\}} dist(o_c, o)$$
  - mean squared distance to the closest cluster:
    $$sqrComp(c) = \sum_{o \in \{o \in DB | cluster(o) = c\}} dist(o_c, o)^2$$

- Quality of the comlet clustering : $TD(C) = \sum_{c \in C} compact(c)$

$$TD^2(C) = \sum_{c \in C} sqrComp(c)$$

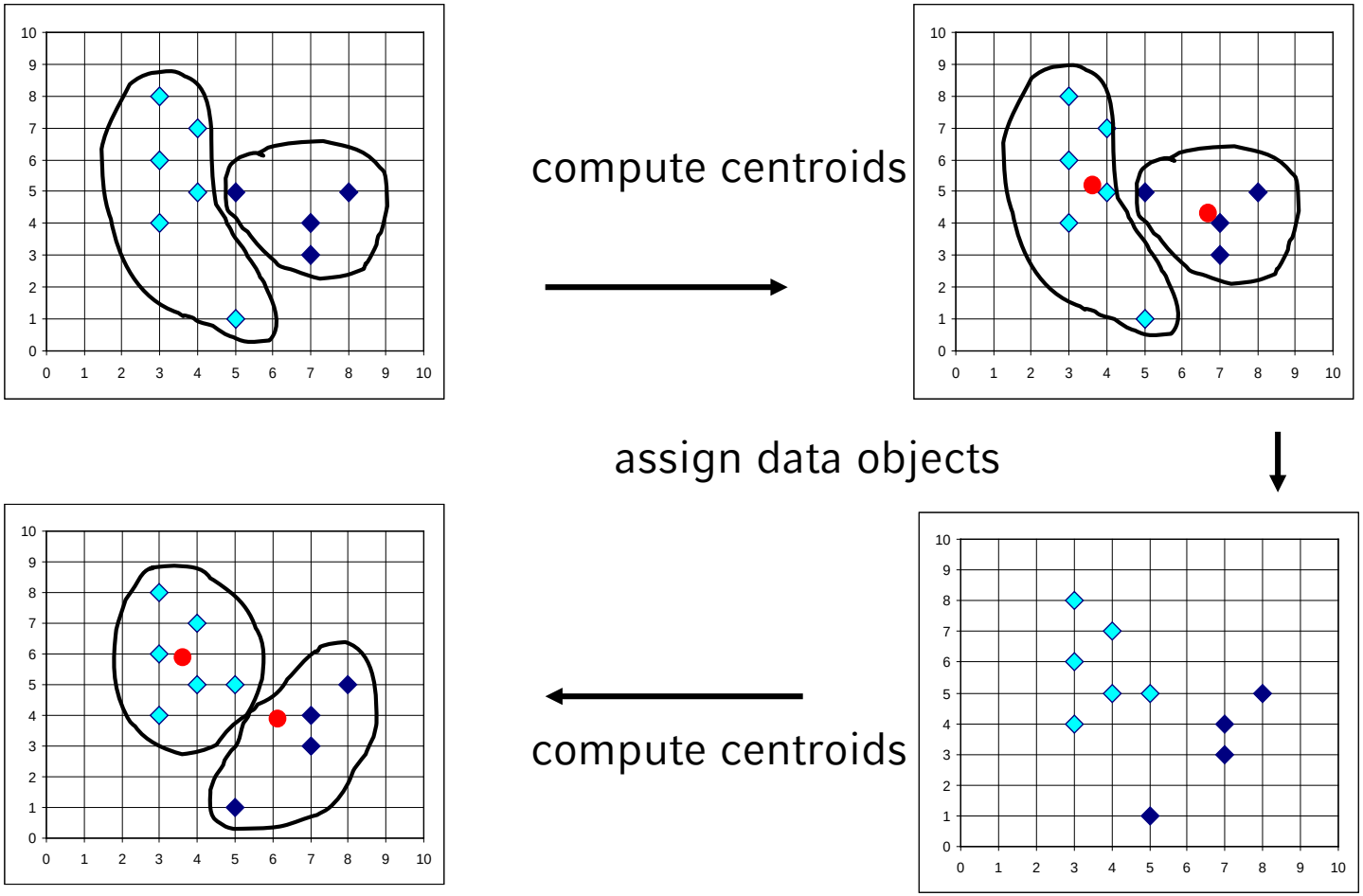
Partitionierendes Clustering (2)

- typical cluster representations:
  - centroid: $centroid(c) = \frac{1}{\left|\{o \in DB \mid cluster(o) = c\}\right|} \sum_{o \in \{o \in DB \mid cluster(o) = c\}} o$
  - medoid: $medoid(c) = \arg\min_{o \in \{o \in DB \mid cluster(o) = c\}} \left(\sum_{p \in \{p \in DB \mid cluster(o) = c\}} dist(o, p) \right)$

minimize TD or TD^2:

- *TD* and TD^2 are not konvex and might have multiple local minima
- *TD* and TD^2 are discontinuous (e.g. when switching clusters)
- apply greedy search to minimize TD/ TD^2
  1. Step: for all o ∈ DB cluster(o) is known
     => compute cluster representations $\{o_{c1}, \ldots, o_{cn}\}$
  2. Step: given the cluster representation $\{o_{c1}, \ldots, o_{cn}\}$
     => assign all objects to their closest clusters and go to step 1
  3. terminate if TD/ TD^2 do not change (no cluster switch => local minimum)

Example: Partitioning Clustering



Algorithm

```
ClusteringVarianceMinimization(Objectset DB, Integer k)
    build initial clustering by splitting DB into k Cluster;
    Compute representatives C'={C_1, ..., C_k}
    C = {};
    TD2 = sqrTD(C',DB);
    repeat
        TD2old = TD2;
        C = C';
        build k clusters by assigning each object to the next
          centroid in C;
        compute the new representatives C'={C'_1, ..., C'_k};
        TD2 = sqrTD(C',DB);
    until TD2 == TD2old;
    return C;
```

Partitioning Clustering

variants:

- *k*-Means: update a single object and then recompute affected centroids.
- Expectation Maximation Clustering (EM)

  cluster=density distribution, Bayesian model, soft-clustering
- k-Medoid Clusterings:
  - cluster representations are mediods
  - cluster adaption is done by switiching objects and medoids

properties:

- all algorithms depend on the initialization
- centroid-based are very efficient $O(i \cdot n \cdot k)$. (#Iterations i)
- medoid-based are generic but slow $O(i \cdot n^2 \cdot k)$ (#Iterationen i)