Lecture Notes for

Managing and Mining Multiplayer Online Games

Summer Term 2019

Chapter 5: Game Analytics

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

Overview

- What is Game Analytics?
- Reasons for Fraud in Computer Games
- Types of fraud and countermeasures
- Monitoring player behavior
- Typical balancing tasks
- Game Analytics and KDD

Design Goals

Factors influencing sustainability of player experience:

- Games should be challenging, but not frustrating
- Games should guarantee a fair competition
- Game accomplishments should be persistent
- Games should allow/encourage social interaction
- Achievements should be visible for other players (Rankings, Title, Items, …)
- Games should be expanded and modified regularly
- Games must adapt to growing player skill

Game Analytics

Data Mining and statistical analysis of observed player behavior to gain knowledge about the manner in which a game is being played:

- Creating a game does not make you it's best player.
  How is a game played most effectively?
- Thousands of players cannot be monitored manually.
- How much time do players spend with the game?
  What aspect of the game occupies players most of the time?
- Game difficulty is relative to player skill:
  - *Who is playing, and what motivates players?*
  - *How capable are players with respect to different skills?*

Knowledge Discovery on Game Data

Objectives:

1. **Fraud Detection**

Fraud influences a MMMO's long term success:

- micro-Transactions are no longer necessary
- wrecks other player's game experience

2. **Evaluating Game Balance**

- controlling difficulty level and player progress
- balancing power for different kinds of factions, classes, avatars…
- analysis of player resources necessary for success:
  time, skill and money.

E-Sports Analytics

How can players improve:

- How good do I play?
- How can I improve my performance?
- Which tactics and counter tactics exist?
- Which units are actually the best?
- What is the best team composition?
- What is the superior skill setup?

How strong are E-Sports team:

- How well will my team perform?
- Which new players should I recruit?
- How is my pool developing?
- How do I scout talents?

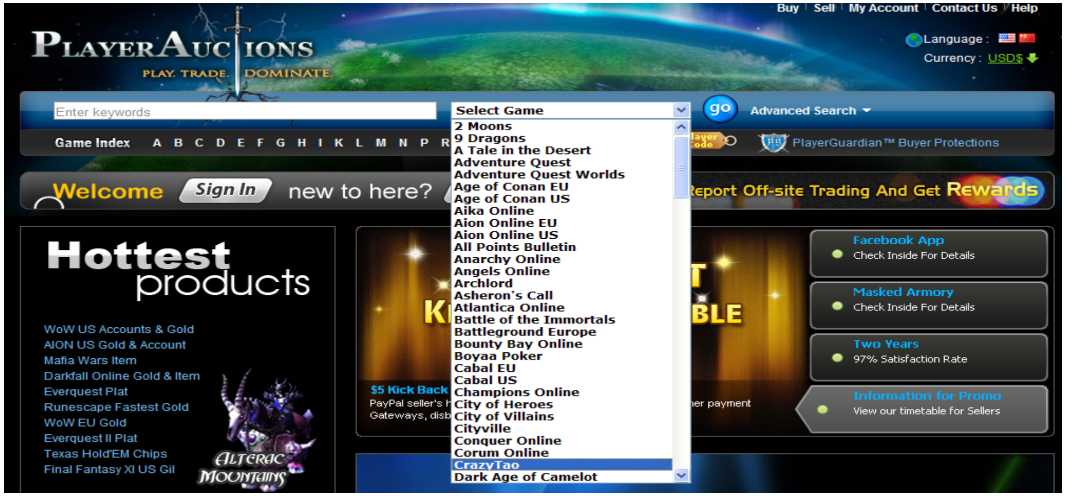
Why are Players committing fraud?

- ***Economic reasons***

  In-game currencies, goods or whole accounts have a real equivalent value: Poker Bots, Gold Farming, Account Trading, Item Trading, …

Example: Portals for trading game goods exist (numbers 2013) (playerauctions.com)

- More than 1 B USD accumulated player-to-player commercial value
- On average 25,000 daily transactions (ca. 20 pro Minute)
- More than 700 Massively Multi-Player Online Games are supported
- More than 30 MM completed transactions



Why are Players committing fraud?

- ***Saving time***

  example: AFK Bots (autonomous programs to control the game for completing task that do not require much skill (gathering, …) in player absence
- ***Prestige***

  in social games success is coupled with a reputation.

  example: titles, rankings, ladders..
- ***Fun***

  beating other players is fun, even when you play unfair

Any motivation is problematic

- Providers directly loose money (micro transactions)
- Games loose sustainability and players loose interest
 (unfair competition, no performance comparability, fair players get ftustrated)

Technical ways to cheat

- ***Exploits:*** taking advantage of bugs and bad design in the game
- ***Client Modifications (Hacks)***
  - Information Hack: player gets more information than intended.
    (e.g. Map-Hacks, Wall-Hacks, …)
  - circumventing game-physics or other rules (e.g. Teleportation Hack)
- ***Modifying other system components***
  - manipulation the network by manipulating latency, protocol headers or time-stamps(e.g. Protocol Hacks)
  - manipulating device drivers (e.g. Wall-Hacks with transparent textures)
- ***Botting***
  - control of an avatar to save time (Farm Bots)
  - control of an avatar to increase game power (Poker Bots, Aim Bots)
- ***Macros, scripts, programmable I/O-devices***
  - partly automating control of an avatar to simplify complex actions (Gaming Macros)
  - programs that optimize user input

Other ways of Cheating

- ***Win-Trading***
  losing deliberately to speed up the opponents progress

- ***Account Kidnapping:***
  takeover of a player account for a limited time:
  - selling the victims virtual possessions and in game currency
  - obstructing the opponent during deciding game stages

- ***Illicit trade with virtual goods***
  - often in combination with account kidnapping
  - external trade might trade ingame achievement with spending real money
    (If weak players control very successful avatars, it undermines the sustainability of fairly acquired successes)
  - trade may decrease the revenue of the game
    (less micro transactions)

Countermeasures: Prevention

Cheat/Fraud Prevention

- important operations are calculated on server side
- use of checksum methods on client programs
- clients receive software that checks for fraudulent behavior
  (e.g. Punkbuster, Warden, …)

Advantage:

- prevents fraud before it affects other players

Disadvantages:

- control software has access to the client computer
  (endangers privacy)
- server side computations cost expensive resources and cut down the in game response time (waiting for a RTT)
- client computer are always under player control
  (virtual machines, Roots Kits, Code and DLL Insertion, …)

Countermeasure: Detection

Cheat/Fraud Detection:

- server side surveillance of players
- recognition of suspicious or fraudulent behavior
- players get sanctioned if caught
  (temporary ban players from the game)

Advantages:

- fraud is detected on server side
  - $\Rightarrow$ cheaters are unable to analyze the detection mechanism
  - $\Rightarrow$ no breach of privacy
- flexible approach, capable of detecting new kinds of fraud without adjustment

Disadvantage:

- reactive approach
  (fraud must occur before it can be sanctioned)

Monitoring player behavior

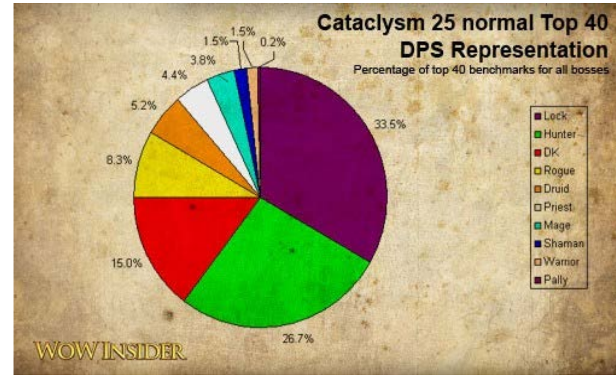
Challenges and Problems with Player surveillance:

- A chain of events is necessary for analysis.
  (saving the course of play is crucial)
- Reviewing every player action constitutes a very large calculation effort.
- Reviews should be as unspecific as possible, so variations and new possibilities can be detected without additional effort.
- Sanctioning fraud has to consider all impacts and player perceptions
  (Not every minor cheat/exploit should lead to a ban)

Monitoring Game Balance

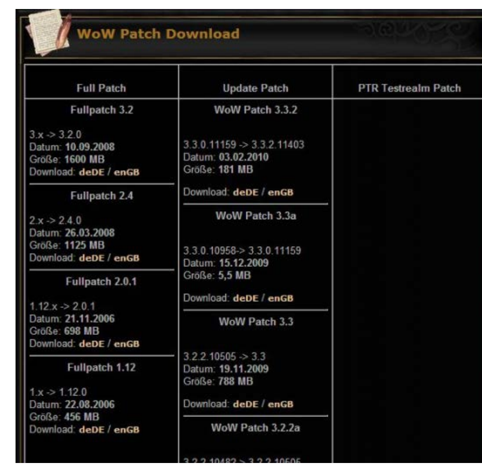
Game Balance describes:

- the difficulty level of the game
  (challenging, but not frustrating)
- the fairness of the game
  (Do all players have a fair chance to win?)
- the influence of skill, money and time
  (Should top players invest less or more money? How much time can/must be invested into the game?)



How are games balanced:

- defining design-goals
- establish mechanics to implement the goals
- observe the impact on game play



Monitoring Game Balance

Problems with Beta Tests:

- The more is seen during the beta-test, the less “fresh content” is left for the actual game. (Spoiler)
- Beta Tests are usually too small to include all group compositions, circumstances and possible tactics.
- New content should be released regularly

  => limited time frame for tests

⇒ Beta Tests require game analytics to be as comprehensive and effective as possible.

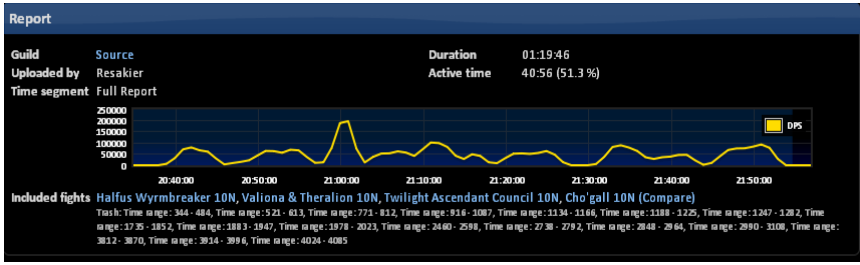
⇒ Control over current events and hot fixing problems are daily tasks of most MMOs.

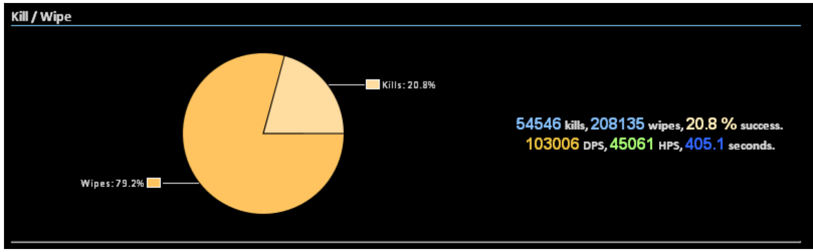
Typical Game Balancing Tasks

- **Predicting player skill and match making**
  - Which teams should compete against each other?
    => dependent on the skill level and the players currently queuing
  - How are new teams ranked?
  - How should the player ranking be modified after the game?
- **Analysis of character classes and units**
  - Is the choice of faction or class a deciding factor for success?
  - What are the reasons behind this observation?
    => dependent on game-situations and player skill
- **What kinds of players are there?**
  - Which players are most profitable?
  - What are specific player groups' needs?
  - Which kind of players are necessary for sustainable success of the game?

Methods for Game Analyzing Balance

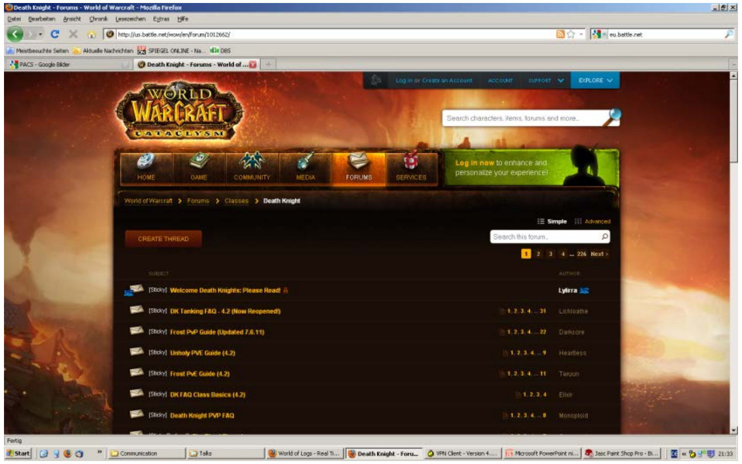
- Event Detection in data streams
- Monitoring of encounter-results
- Estimating player strength, to remove data bias





- Identification and description of characteristic strategies
  => *the more diverse, the more interesting is an encounter*

- Analysis of social media (e.g. Forum) and specifically including player opinions



The Analytical Process in Games

1. define the goal of the analysis (Focussing)

2. extract relevant data
   (players, events, ...) (Selection)

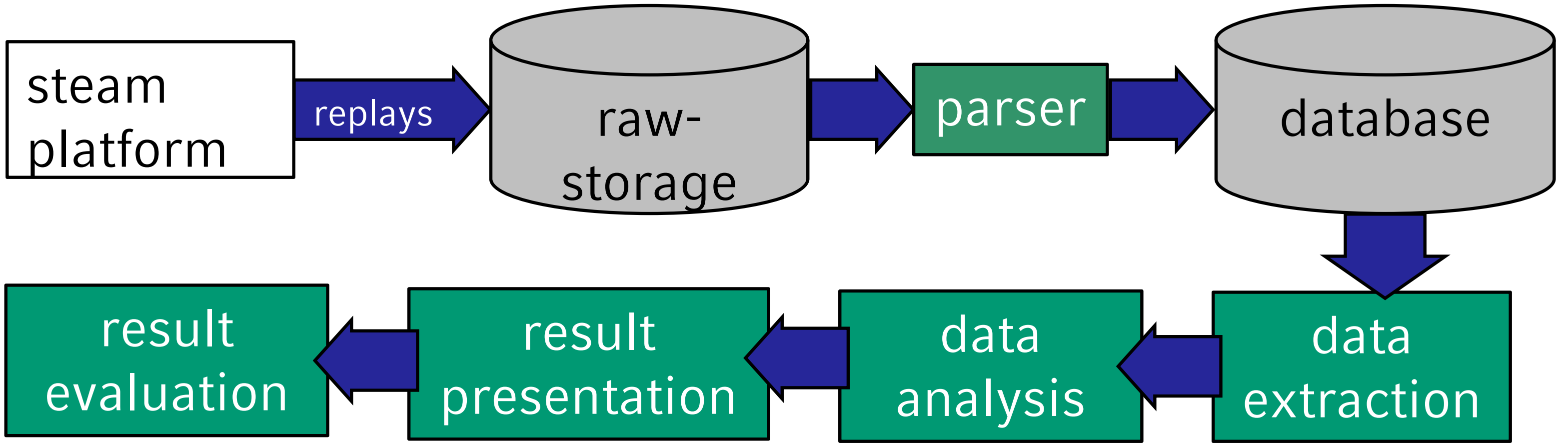
3. model player behavior (Data Transformation)

4. apply data mining/machine learning (Mining)

5. evaluate the found patterns (Evaluation)

6. use results to develop measures to reach design-goals
   (How do you use the results?)

Example: Analysis of DOTA2



- define question: At which time is it possible to predict the outcome of game?
- data acquisition and processing: download/parse replays
- model player behavior: accumulate gold/XP progress
- generating knowledge: build classifier for each time step
- evaluation: How good is the model? Is the result useful?

When does Game Analytics work?

- **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 is available, the less likely two objects are similar to each other.
- ***Specificity of attribute values***: The more unique an attribute value, the smaller is its contribution to distinguish data objects. (example: Object_ID)
- **Model complexity**: The more complex a function or a pattern is, the easier it adapts to the given training objects and does not generalize well.

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

Feature Space, Distance and Similarity Measure

In how far do objects behave in an comparable way?

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

Formalizing Similarity:

- **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 Function**: calculates difference between two object descriptions. (the higher, the more dissimilar)

Note: 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 feature space)

Formal definition of distance function

Distance function: Let F be a feature space.
$dist : F \times F \rightarrow \mathrm{IR}^{+}_{0}$ 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

Feature Vectors: 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.

Supervised Learning

Idea: Learn from example objects to optimize a predictive function.

given:

- target variable C
  (*classification*: set of nominal values, *regression*: numerical values)
- objects: $DB \in F \times C$: *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: $L^2(f,T) = \sum_{(x,y)\in T} \big(y - f(x)\big)^2$

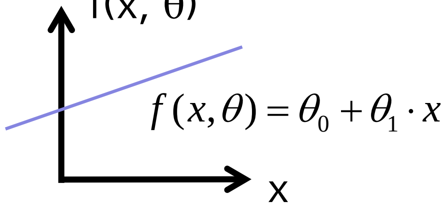
hinge loss: $L_{hinge}(f,T) = \sum_{(x,y)\in T} \max(0,1 - t \cdot f(x))$

Training Supervised Methods

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

example: *f* univariate linear function:

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

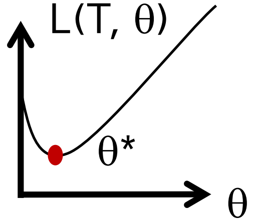


solution 2D case

training: minimize loss function

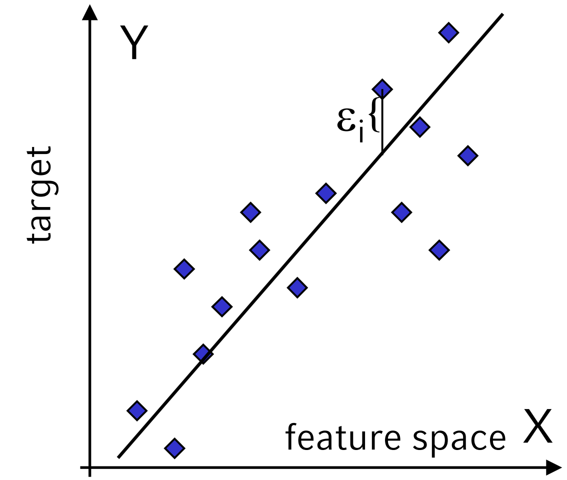
- Loss function *L describes the error of f for T*
- search parameters θ* minimizing *L*
- ***approach***: build the gradient of L for θ
  and compute the minimum θ*.

  => 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:

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

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

gradient for θ_0:

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

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

gradient for θ_1:

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

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

Further Comments on Supervised Learning

- **regularization**: Often parameters θ can grow unrestricted
  => 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)

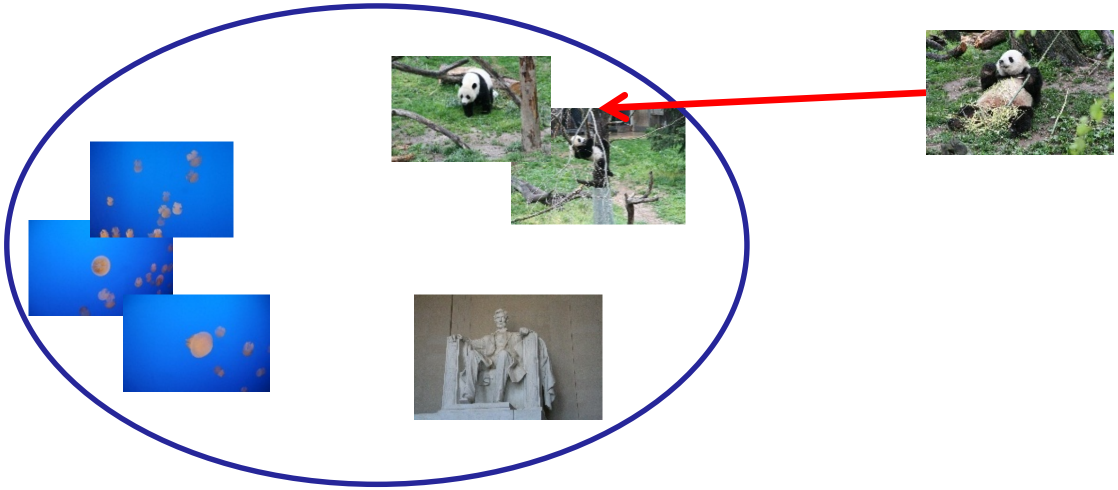
  => optimization minimizes the loss until local convergence
- there are other approaches to supervised learning not minimizing a loss function, e.g., maximize the likelihood

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 prediction
  - use majority vote (classification)/ mean (regression)
  - distance weighted votes *(e.g., quadratic inverse weighting: $1/d(q,x)^2$)*



Rule of Bayes

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

- *we assume $p(v) = p(A) \cdot p(v|A) + p(B) \cdot p(v|B)$,*
  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*:* $P(B|v) = \frac{P(B) \cdot P(v|B)}{P(v)}$
- for all distributions C and observation *v* it holds: $\sum_{c \in C} P(c|v) = 1$

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

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*: human player, 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.

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 probabilities in T.

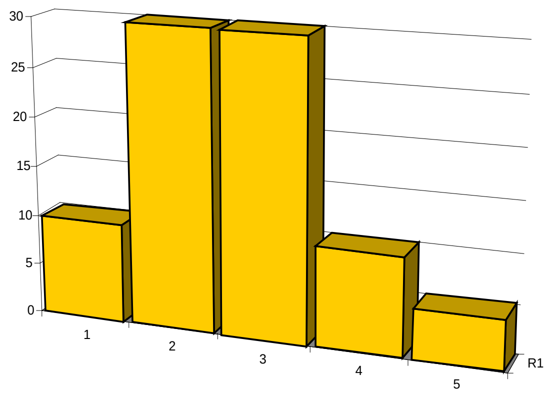
example: consider two dices

- 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:*

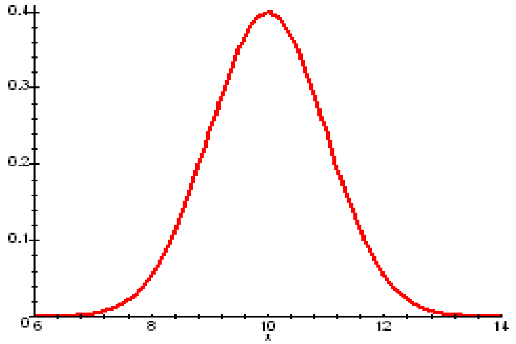
$$P(D2|5) = \frac{0{,}8 \cdot \frac{1}{6}}{0{,}2 \cdot \frac{1}{6} + 0{,}8 \cdot \frac{1}{6}} = 0{,}8 \qquad P(D2|6) = \frac{0{,}8 \cdot \frac{1}{3}}{0{,}2 \cdot \frac{1}{6} + 0{,}8 \cdot \frac{1}{3}} = \frac{8}{9}$$

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 $1/\infty \rightarrow 0$)
  - estimation of using probability density functions (e.g. Gaussian distributions)
  - training = estimate parameters of the density function (e.g., mean and variance)
  - to compute probabilities from density function either **integrate** over an interval of events or apply the **rule of Bayes** to determine relative densities.



Statistical Models

considering multiple features v_i requires joint estimates of

$p(v_1, \ldots, v_d | c)$.

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

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

  pro: easy estimation and computation

  $$P(v_1, \ldots, v_d \mid c) = \prod_{i=1}^{d} P(v_i \mid c)$$

  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
- advanced 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 very 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:

- train and test on as many instances as possible
- train and test set must be 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

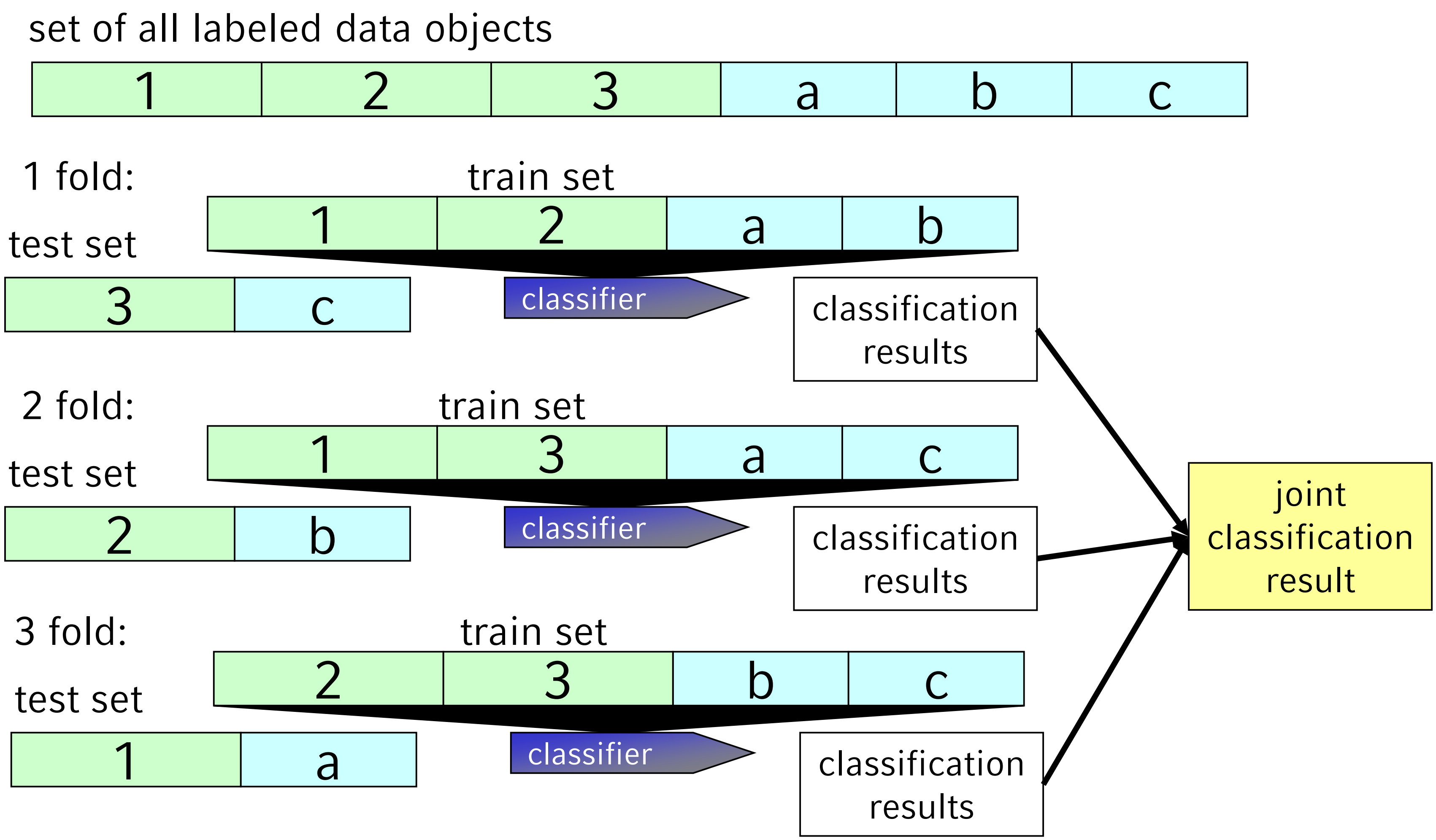
- similar to leave-one-out.
- build *k* folds and performs leave-one-out on folds instead of instances

- **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 k the more effort, but the larger the training sets)

- *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: 3-fold stratified 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:

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

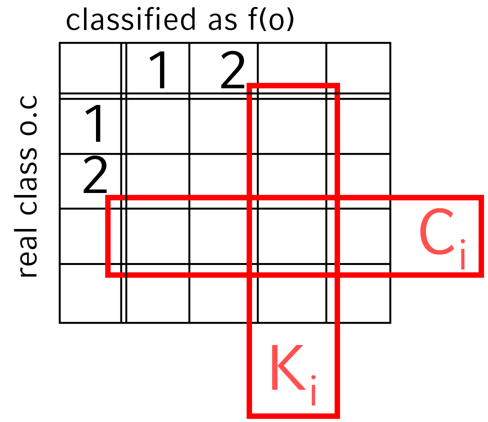
- apparent classification error on **TR** (used to determine overfitting)

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

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.

$$\mathrm{F1}_{TE}(f,i) = \frac{2 \cdot \mathrm{Precision}_{TE}(f,i) \cdot \mathrm{Recall}_{TE}(f,i)}{\mathrm{Precision}_{TE}(f,i) + \mathrm{Recall}_{TE}(f,i)}$$

Learning goals

- What is game analytics?
- Motivations for fraud
- Methods of fraud and counter measures
- Game Balance and Balancing Tasks
- Game Analytics as process
- Generalization and Overfitting
- Supervised Learning
- Minimizing Loss Functions
- Instance Based Learning
- Baysian Prediction
- Evalution of supervised methods

Bibliography

- J. Yan, B. Randell
  A systematic classification of cheating in online games
  In Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games, 2005.

- Greg Hoglund, Gary McGraw:
  Exploiting Online Games: Cheating Massively Distributed Systems
  Software Security Series, Addison Wesley, 2007.

- script KDD I:
  http://www.dbs.ifi.lmu.de/cms/Knowledge_Discovery_in_Databases_I_(KDD_I)

- Han J., Kamber M., Pei J.:
  Data Mining: Concepts and Techniques
  3rd edition, Morgan Kaufmann Publishers, March 2011.