ifis
Institut für Informationssysteme
Technische Universität Braunschweig

Data Warehousing & Data Mining

Wolf-Tilo Balke
Silviu Homoceanu
Institut für Informationssysteme
Technische Universität Braunschweig
http://www.ifis.cs.tu-bs.de

13. Meta-Algorithms for Classification

13.0 Meta-Algorithms Detour

- **Upper layer family of algorithms**
  - **Are not problem-specific**
  - May make use of domain-specific knowledge in the form of heuristics that are controlled by the **upper level strategy**
    - For this reason, also know as **meta-heuristics**
- Around since the early 1950s

13.0 Meta-Algorithms Detour

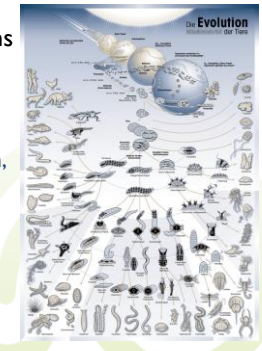
- Meta-algorithms express common knowledge
  - Usually the **last resort** before giving up and using random or brute-force search
  - Used for problems where you **don't know how to find a good solution**
    - But if shown a candidate solution, you can assign a grade
  - The algorithmic family includes genetic algorithms, hill-climbing, ant/bee colony optimization, simulated annealing, etc.

13.0 Meta-Algorithms Detour

- Genetic algorithms
  - Search algorithms based on the mechanics of biological evolution
  - Developed by John Holland, University of Michigan (1970's)
    - To understand the adaptive processes of natural systems
      - In time, organisms adapt to the environment
    - To design artificial systems software that retains the robustness of natural systems

13.0 Genetic Algorithms Detour

- A way of solving problems by **mimicking** the same processes **nature** uses
  - They use the same combination of **selection**, **recombination** and **mutation** to evolve a solution to a problem



13.0 Genetic Algorithms Detour

- Typical structure of genetic algorithms

```
initialize population;
evaluate population;
while TerminationCriteriaNotSatisfied
{
        select parents for reproduction;
        perform recombination and mutation;
        evaluate population;
}
```

- Components of genetic algorithms
  - Encoding technique: **genes** and **chromosomes**
  - Initialization procedure or **creation**
  - Evaluation function: the **environment**
  - Selection of parents: **reproduction**
  - Genetic operators: **mutation** and **recombination**

13.0 Genetic Algorithms Detour

- Problem: Given the digits 0 through 9 and the operators +, -, * and /, find a sequence that will represent a given target number
  - So, given the target number 23, the sequence 6+5*4/2+1 would be one possible solution
    - The operators will be applied sequentially from left to right
  - We need to encode a possible solution as a string of bits… a **chromosome**
    - Represent all the different characters available to the solution... that is 0 through 9 and +, -, * and /
      - This will represent a **gene**
    - Each chromosome will be made up of several genes

13.0 Genetic Algorithms Detour

- Four bits are required to represent the range of characters used
  - Then 0000 for example will be a **gene**
  - 1110 and 1111 will remain unused and therefore ignored by the algorithm
  - A solution for 23 would then form the following chromosome

6	+	5	*	4	/	2	+	1
0110	1010	0101	1100	0100	1101	0010	1010	0001

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
+	1010
-	1011
*	1100
/	1101

13.0 Genetic Algorithms Detour

- Initialization: At the beginning of a run of a genetic algorithm a large population of random chromosomes is created
  - Each one, when decoded will represent a different solution to the problem at hand

- Evaluation of the population:
  - Let's say there are N chromosomes in the initial population
  - Test each chromosome to see how good it is at solving the problem at hand and assign a **fitness score** accordingly

13.0 Genetic Algorithms Detour

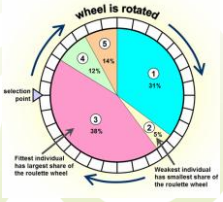
- Fitness score
  - The fitness score is a measure of how good that chromosome is at solving the problem to hand
    - **It is problem dependent**
  - If we assume the target number 42, the chromosome

6	+	5	*	4	/	2	+	1
0110	1010	0101	1100	0100	1101	0010	1010	0001

  which represents number 23, has a fitness score of has a fitness score of 1/(42-23) or 1/19

13.0 Genetic Algorithms Detour

- Select two members from the current population
  - The chance of being selected is proportional to the chromosomes **fitness (survival of the fittest)**
  - **Roulette wheel selection** is a commonly used method
    - It **does not guarantee** that the fittest member goes through to the next generation, merely that it has a very good chance of doing so



13.0 Genetic Algorithms Detour

- Recombination
  - Crossover the bits from each chosen chromosome at a randomly chosen point
  - E.g. given two chromosomes, choose a random bit along the length, say at position 9, and swap all the bits after that point

- Mutation
  - Step through the chosen chromosomes bits and flip dependent on the mutation rate (usually a very low value for binary encoded genes, say 0.001)

13.0 Genetic Algorithms Detour

- Repeat **selection**, **recombination** and **mutation** until a new population of N members has been created
  - Then evaluate the new population with fitness scores
- Stop when a chromosome from the population solves the problem

13.0 Application in Data Mining

- Example: classification problems can be solved using **multi-classifier combination** (MCC)
  - Basic classifiers may individually achieve a precision just better than random classification on difficult training data
  - But if independent classifiers are used **together**, they **strengthen each other**
  - The main idea originates from a technique called **bootstrapping**

13.1 Bootstrapping

- The idea of **bootstrapping** comes from the stories of Baron von Münchausen
  - He tries to pull himself and his horse from a swamp by his hair

  - But why is it called **bootstrapping**?

13.1 Bootstrapping

- The Baron's story inspired a metaphoric phrase in the early 19th century United States
  - "To pull oneself over a fence by one's bootstraps"
- Bootstrapping in computer science
  - Software bootstrapping
    - Development of successively more complex, faster programming environments
    - E.g. start with *vim* and *assembler* and build iteratively graphical *IDEs* and high-level programming languages

13.1 Bootstrapping

- Bootstrapping in **classification tasks**
  - Iteratively improve a classifier's performance
  - **Seed AI** is a strong artificial intelligence capable of improving itself
    - Having improved itself it would become better at improving itself, potentially leading to an exponential increase in intelligence
    - **No such system exists**

13.1 Bootstrapping

- Check the accuracy of **sample estimates**
  - Assume we are interested in the average height of all people in the world
  - We can only measure a maximum of N people and calculate their average

  - Is this average value good? We need a sense of **variability**

13.1 Bootstrapping

- **Bootstrap:** take the N values and build another sample also of length N
  - By **sampling with replacement** (i.e. each element may appear multiple times in the same sample) from the N original values
    - Example: choose a person, measure it, return it to the pool, choose again,…
    - The same person may be measured twice
- Repeat the bootstrapping 1000 times
- Calculate the 1000 averages and their variance

13.1 Bagging

- **Bagging (B**ootstrap **agg**regat**ing**) for classification problems

  - Leo Breiman, *Bagging Predictors*, Machine Learning Journal, 1994
  - Starting from a training set, draw *n* samples with replacement
    - Usually n is larger than the number of records in the training set
  - Train a classifier on the resulting sample

  - Repeat the process *m* times to learn *m* classifiers

13.1 Bagging

- Classifying a new record
  - Perform a **majority vote** over all trained classifiers
- Advantages:
  - Increases classifier stability
  - Reduces variance

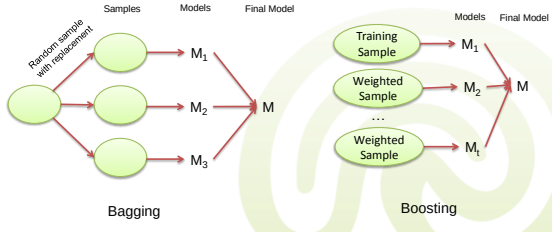
13.2 Boosting

- Boosting is based on the idea of bootstrap aggregating

  - Michael Kearns, *Thoughts on hypothesis boosting*, unpublished manuscript, 1988
  - Build a **series of classifiers**
    - Each classifier in the series pays **more attention** to the examples **misclassified** by its predecessor
  - May use any basic classifier from decision trees to support vector machines

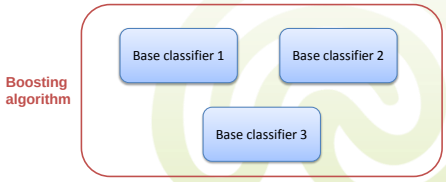
13.2 Boosting

- Bootstrap aggregating vs. Boosting



13.2 Boosting

- Basically, a boosting algorithm is a **blueprint** of how to combine a set of "real" classification algorithms to yield a single combined (and hopefully better) classifier



13.2 Boosting

- Each different classification algorithm comes with individual strengths and weaknesses
  - "There ain't no such thing as a free lunch"
- For hard classification problems, the usual classifiers tend to be **weak learners**
  - Weak learner = only slightly better than random guessing
- Question:
  - Can a set of weak learners create a single **strong learner?**
- Answer: **YES!**

13.2 Boosting

- **Naïve approach** to boosting: **majority vote!**
  1. Train base classifiers independently on the training set
  2. For each new object to be classified, independently ask each base classifier and return the answer given by the majority
- Problems:
  - Does only work if the majority is right very often
  - The base algorithms cannot take advantage of their individual strengths
  - Should expert votes really have the same weight like any other vote?

13.3 Boosting

- Better approach: **Adaptive boosting**
  - Yoav Freund, Robert E. Schapire: *A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting*, 1995.

- Major steps:
  1. Train a **first base classifier** on the training set
  2. Check which training examples **cannot be explained** by the first case classifier's underlying model ("errors")

13.3 Adaptive Boosting

3. Assign a **weight** to each training example
   - Low weight = Example perfectly fits into the first classifier's model
   - High weight = Example does not fit into the first classifier's model
4. Train a **new base classifier** on the weighted training set
   - Fitting training examples with high weights is more important than fitting those with low weights
5. **Reweight** as in step (3)
6. **Repeat** the steps (4) and (5) to create a set of base classifiers

13.3 Adaptive Boosting

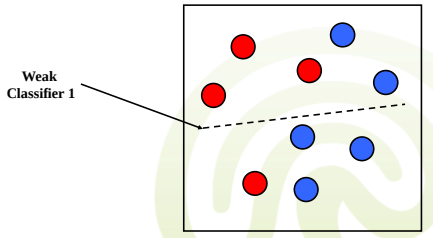
- **Adaptive boosting (cont.)**
  - In addition, assign an **importance weight** to each base classifier, depending on how many training examples fit its model
    - High importance, if errors occur only on training examples with low weight
    - Low importance, if errors occur on training examples with high weight

13.3 Adaptive Boosting

- How does the **combined classifier** work?
  1. Classify the new example with each base classifier
  2. Use **majority vote** weighing the individual classifier's answers by their **importance weights**
     - Also incorporate each classifier's confidence, whenever this information is available
- Typically, the importance weights and the weights of the individual training examples are chosen to be **balanced,** such that **the weighted majority now is right very often**

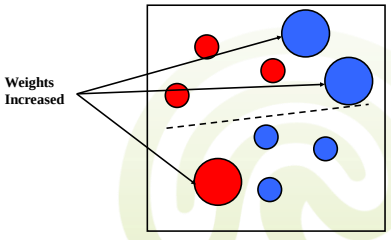
13.3 AdaBoost - Example

- Example: first weak classifier



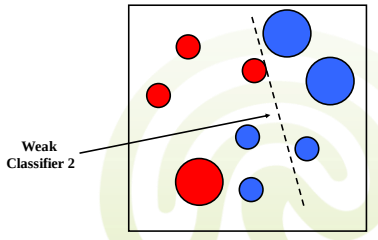
13.3 AdaBoost - Example

- Weights of misclassified objects are increased



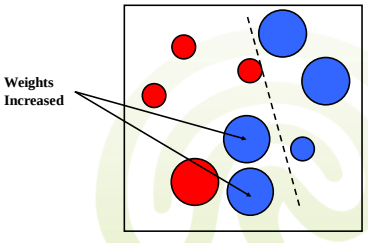
13.3 AdaBoost - Example

- Second weak classifier



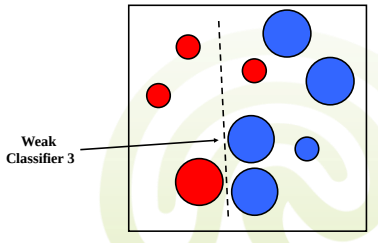
13.3 AdaBoost - Example

- Weights of newly misclassified objects are increased



13.3 AdaBoost - Example

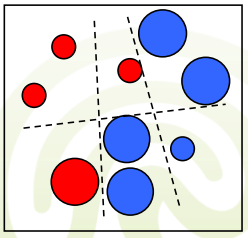
- Third weak classifier



13.3 AdaBoost - Example

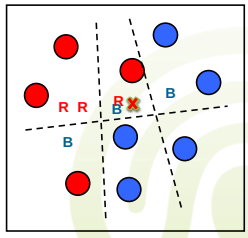
- …and we could go on like this, to improve the classification precision

Final classifier is a combination of weak classifiers



13.3 AdaBoost - Example

- Classify new data
  - C_1 and C_2 classify the new data object as red



13.3 Formal Description

- Let X denote the instance space and Y the set of class labels
  - Assume binary classification Y = {-1, +1}
  - Given a weak or base learning algorithm and a training set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where $x_i \in X$ and $y_i \in Y$ $(i = 1,\ldots, m)$
  - Let T be the number of iterations
  - D_t denotes the distribution of weights at the t-th learning round

13.3 Formal Description

- We assign equal weights to all training examples
  - $D_1(i) = \frac{1}{m}$, $i = 1, \ldots, m$
- From the training set and D_t the algorithm generates a weak learner $h_t: X \to Y$
  - $h_t = \text{argmax}\,|0.5 - \varepsilon_t|$ where $\varepsilon_t = \sum_{i=1}^{m} D_t(i) * I_{(yi \neq ht(xi))}$ and I is the indicator function
  - Put into words, h_t maximizes the absolute value of the difference of the corresponding weighted error rate from 0.5 with respect to the distribution D_t

13.3 Formal Description

- h_t is then tested on the training examples and the weights of the incorrect classified examples are increased
- Thus, an updated weight distribution D_{t+1} is obtained
  - $D_{t+1}(i) = \frac{D_t(i)^{(-\alpha_t y_i h_t(xi))}}{Z_t}$, where $\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t}$ represents the weight of h_t and Z_t is a normalization factor
- The process is repeated T rounds

13.3 Formal Description

- After T rounds we have trained weak hypotheses $h_1, \ldots, h_T$
- The combined hypothesis H is a **weighted** majority vote of the T weak hypotheses
  - Since each hypothesis h_t has weight α_t

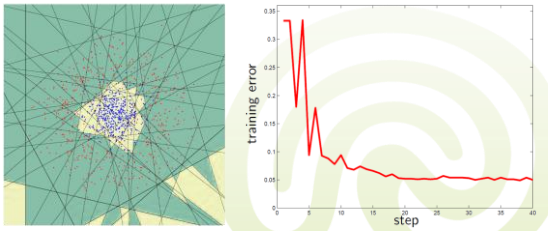
$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

- If finding a suitable number T of hypotheses to train is difficult, stop training when the last trained hypothesis is good enough
  - Stop condition: $|0.5 - \epsilon_t| \leq \beta$, where β is a quality threshold

13.3 AdaBoost Evaluation

- AdaBoost example
  (Jan Šochman, Center for Machine Perception)

$t = 40$



13.3 Advantages

- Why is adaptive boosting better than "pure" majority vote?
  - Later weak learners focus more on those training examples previous weak learners had problems with
  - Individual weaknesses can be compensated
  - Individual strengths can be exploited

Summary

- Basic classifiers **alone** achieve a precision of just better than random classification on difficult training data
- When more classifiers are used **together**, they strengthen each other out
  - Bootstrap aggregating introduces the **voting** principle
  - Boosting introduces **weights** for the falsely classified objects
  - Adaptive Boosting introduces **weights** also for the classifiers

Next week

- Guest: Mr. Andreas Engelke
  - Head of Consulting Corporate Performance Management Braunschweig at ISR Information Products AG
  - An insight on data warehousing and data mining from an **industry perspective**

13 Thank You!

- I hope you enjoyed the lecture and learned some interesting stuff…
- Next semester's master courses:
  - Relational Databases 2
  - Deductive Databases and Knowledge-Based Systems
  - Lab: Integrity Constraints
  - Seminar: Advances in Digital Libraries