Name: ..... Neptun: .....

PPCU FIT • fall semester 2017

DATA MINING

ORAL EXAM

4 January 2018

PÁZMÁNY PÉTER CATOLICH UNIVERSITY

FACULTY OF INFORMATION TEHCNOLOGY AND BIONICS

Important information

Dear Candidate!

This booklet is made for the Oral Exam in Data Mining. The booklet contains the titles of the exam topics as well as the detailed form of these. In some cases you may find errors or typos in the text. If this happens please, feel free to report them on my website.

We wish you a successful preparation!

Edited by:
Márton Bese NASZLADY– 2018

Table of Contents

Topic Titles

1. **Introduction to Data Mining –** motivation, application areas, definition – Fayyad, KDD process, goals, major classes of DM tasks, related fields
2. **Input for data mining –** concepts, instances, attributes, nominal/ordinal/interval/ratio quantities, transformation between data types, missing values
3. **1R algorithm and Naïve Bayes, Bayesian network model/interpretation –** basic considerations, assumptions, form of knowledge representation produced, algorithms learning / prediction, numeric attributes and missing data
4. **Basic decision tree building, ID3 –** basic considerations, goals, information gain/entropy – properties, algorithm, highly branching attributes, limitations
5. **Basic rule construction, PRISM –** basic considerations, algorithm, limitations
6. **Mining association rules –** form of association rules, support, confidence, challenge, item sets, apriori algorithm
7. **Linear models –** linear regression, linear models for classification, logistic regression, logit transformation, perceptron
8. **Performance measures –** basic considerations, resubstitution error, training and testing, stratification, confusion matrix, Kappa statistics, considering costs, three-way data split
9. **Limited data problem –** holdout, cross-validation, LOO-VC, bootstrapping. Lift charts, ROC curves, Evaluating numeric prediction
10. **Industrial strength algorithms –** noisy data, numeric attributes, missing values, scalability. **C4.5** – pruning: prepruning, postpruning, pruning operations: subtree replacement, subtree raising, pruning decision, numeric attributes, missing values
11. **Support vector machines –** basic considerations, linear case, support vectors, maximum margin hyperplane, nonlinear case, kernel functions
12. **Learning Bayesian networks –** Bayesian network, predicting, learning – conditional probabilities, structure (K2, TAN,..)
13. **Instance-based learning –** distance measures, kd-Tree, ball-tree, speed, noise, weights of attributes, generalization
14. **Clustering –** k-means, limitations of k-means, number of clusters, hierarchical clustering, incremental clustering, density-based clustering, EM
15. **Extending numeric prediction –** regression and model trees, nominal attributes
16. **Data transformations –** attribute selection: filter (...)/wrapper, PCA, discretizing numeric attributes (unsupervised, supervised); data cleansing, projections
17. **Ensemble learning –** basic idea, general advantage/disadvantage, bias-variance decomposition, bagging, randomization, random forests, boosting, (stacking)

Exam Topics

Topic 1 Introduction to Data Mining

Motivation and application areas

Nowadays, most of the transactions (e.g. shopping, banking etc.) and measurements (e.g. car sensors, health data etc.) are carried out with digital equipment. Since the storage of the digital data is not a big problem, we can preserve this information for later.

Understanding the underlying patterns in the data is good (e.g. banks can get more money, which is good (for banks), or in healthcare a new connection can be found between a disease and smoking etc.). The aim of Data Mining is to identify such patterns in data.

Typical application areas: Banking, Marketing, Retail, Telecom, Insurance, Healthcare…

Definition – Fayyad

Definition *Knowledge Discovery* is the non-trivial **process** of identifying **valid**, **novel**, potentially **useful** and ultimately **understandable patterns** in **data**.

[Osama Fayyad et al., 1996]

The bold terms above are also defined as the followings:

process – a multi-step process involving data preparation, pattern searching, knowledge evaluation and refinement with iteration after modification.

***valid* –** discovered patterns should be true on new data with some degree of certainty. Generalize for the future (other data).

***novel* –** patterns must be novel (should not be previously known).

***useful* –** actionable; patterns should potentially lead to some useful actions.

***understandable* –** The process should lead to human insight. Patterns must be made understandable in order to facilitate a better understanding of the underlying data.

***pattern* –** an expression in a language describing facts in a subset of the data or in a model.

***data* –** set of facts (e.g. instances in a DB).

KDD process

KDD stands for *Knowledge Discovery in Databases*. The steps are the following:

1. Acquiring background domain knowledge
2. Setting goals
3. Selection, reduction, cleansing of data
4. Model selection – data mining
5. Interpretation

Goals

Data mining has two major goals:

Description: human-interpretable patterns, gain "insights" → pattern/rule + quality

Prediction: future of unknown values supporting, automating, improving, decision making.

Major classes of DM tasks

The four major classes of tasks are Clustering, Classification, Numeric prediction (regression) and Association rule mining.

Clustering
Divide a set of objects (data instances) into subsets, based on similarity. The meaning of subsets is not defined. Also, the number of subsets and the similarity measure itself are questions too. This is **unsupervised learning.**

Classification
Given a set of classified (labelled) data. The 'class' is a nominal attribute (just a label, has no mathematical meaning). The goal of classification is to learn patterns allowing to classify (label) previously unclassified data. This is **supervised learning.**

Numeric prediction
Also called "regression". Similar to classification but a class attribute is now numeric instead of nominal. The goal of numeric prediction is to learn patterns allowing to order a numeric value to unclassified data. This is **supervised learning.**

Association rule mining
The goal of association rule mining is to find association rules which means that if $a_1, \dots, a_n$ occurs in a set, then b is also part of the set with a probability of $> p$. This is similar to classification rules but there is no selected attribute, and the attributes are Boolean (contained or not contained). This is **unsupervised learning.**

Related fields

Data mining is a discipline in which statistics, databases, machine learning, artificial intelligence, data visualization … come together.

Statistics
Statistics is an "old" discipline based on classical mathematical methods while DM focuses on exploratory analysis, finding patterns in data.

Machine learning
Machine learning is about to improve performance according to some quality measure of a computer program automatically, from experience. In ML efficiency and scalability are important. ML often focuses on incomplete / dirty real world data.

Topic 2 Input for data mining

Concepts

An input to a learning scheme is a set of instances / dataset. It is represented as a single relation / flat file. The most common form in practical data mining when there are no relationships between objects, rather a restricted form of input: analog signal, time series etc.

Instance

An instance is a specific type of example:

- a thing to be classified, associated or clustered
- an individual, independent example
- a thing which is characterized by a predetermined set of attributes

Attributes

An instance is described by a fixed predefined set of features, its "attributes". The possible attribute types (the levels of measurement) are: *nominal*, *ordinal*, *interval* and *ratio*.

Nominal $(=, \neq)$
Values are distinct symbols. The values themselves serve only as labels or names. No relation is implied among nominal values (no ordering or distance measure). Only equality test can be performed.
E.g. male / female; married / single / divorced

Ordinal $(=, \neq, <, >)$
Impose order on values. Comparison is possible, but no distance between values is defined. Also, addition and subtraction does not make sense. (Distinction between nominal and ordinal type is not always clear.)
E.g. cool $<$ mild $<$ hot; small $<$ medium $<$ big

Interval $(=, \neq, <, >, -)$
Interval quantities are not only ordered but measured in fixed and equal units. Therefore, difference of two values makes sense. However, product or sum is usually meaningless. The accent is on that there is no fixed zero point in this case.
E.g. year

Ratio $(=, \neq, <, >, -, +, \times, \ldots)$
Ratio quantities are ones for which the measurement scheme defines a zero point. Ratio quantities are treated as real numbers, and therefore all mathematical operations are allowed.
E.g. size in cm, person's age in years

Transformation between data types

Many algorithms (schemes) accommodate just two levels of measurement: nominal and numeric attributes. Simple transformations allows to code ordinal attribute with n values using $n-1$ Boolean attributes. E.g. cold $<$ medium $<$ hot values can be coded into two different attributes: 'at least medium' and 'at least hot'. With these attributes cold is false-false, medium is true-false and hot is true-true.

Missing values

There can be inaccurate or missing values. The reason for *inaccurate data* can be typographical errors, measurement errors, duplicates etc. A value can be missing because of it is unknown, unrecorded or irrelevant. Reasons can be malfunctioning equipment, changes in experimental design, collation of different datasets, impossible measurement etc.

A missing value may have significance in itself (e.g. missing test in a medical examination). However, most schemes assume that this is not the case and 'missing' may need to be coded as additional value.

Topic 3 1R algorithm and Naïve Bayes, Bayesian network

1R algorithm

Form of knowledge representation produced
1R learns a 1-level decision tree. In other words, generates a set of rules that all test one particular attribute. The rule is generated by running the learning algorithm on all of the attributes of the training data; but for prediction only one attribute of the instance is used.

Basic considerations and assumptions
The basic version of the 1R algorithm assumes *nominal attributes* and *no missing values.*

The learning algorithm
The algorithm creates one branch for each attribute's value. Each branch assigns most frequent class. The error rate of a branch is proportion of the instances that don't belong to the majority class of their corresponding branch. Finally, the attribute with the lowest error rate is chosen.

```
For each attribute
   For each value of the attribute, make a rule as follows:
      count how often each class appears
      find the most frequent class
      make the rule assign that class to this attribute-value
   Calculate the error rate of the rules
Choose the rules with the smallest error rate
```

The prediction algorithm
The prediction is very simple; just apply the learned 1-level decision tree. Examine the selected attribute's value and assign the corresponding class to the instance.

Numeric attributes and missing data
Numeric attributes are discretized: the range of the attribute is divided into a set of intervals. Instances are sorted according to attribute's values. The breakpoints are places where the class changes (so the breakpoints are placed in a way that the total error is minimized). If there are equal values that can belong to different classes, then majority voting tells the final class.

Discretization procedure is very sensitive to noise. A single instance with an incorrect class label will most likely result in a separate interval. A simple solution for that is to enforce minimum number of instances in majority class per interval.

Missing values are always treated as a separate attribute value.

Naïve Bayes

Form of knowledge representation produced
Naïve Bayes produces a table of probabilities / a set of mathematical formulas / a Bayesian network that gives the probabilities of all the possible outcomes for an input instance. The model takes every attribute into account both during learning and prediction.

Basic considerations
This modeling procedure assumes that all the attributes are statistically independent. Although, this assumption almost never correct, this scheme works well in practice. This assumption leads to a mathematical simplification: the E evidence in the term $\mathbb{P}(H|E)$ can be split into independent parts, and therefore:

$$\mathbb{P}(H|E) = \frac{\mathbb{P}(E_1|H)\mathbb{P}(E_2|H)\cdots\mathbb{P}(E_n|H)\mathbb{P}(H)}{\mathbb{P}(E)}$$

Furthermore, the basic version assumes that the attributes are nominal and there are no missing values.

The learning algorithm
The learning algorithm builds a table (matrix) in which every attribute and for an attribute every possible value is represented. For an attribute the columns are the possible outcomes, the rows are the possible attribute values and the cells contain the frequency of the corresponding value-outcome pair. The values of the frequencies are normalized within a column.

Note, that until then no decisions were made, it is just filling the required matrices. It is not a problem; the predictions are going to be carried out with the help of this table.

The prediction algorithm
In the prediction phase the probabilities of all the possible outcomes are calculated in the following way:

$$\mathbb{P}(H|E) = \prod_i \mathbb{P}(A_i = E_i|H) \cdot \frac{\mathbb{P}(H)}{\mathbb{P}(E)}$$

where
H is the outcome (i.e. the case that the instance belongs to a certain class)
E is the evidence (i.e. the instance with the missing class attribute)
E_i is the i-th evidence (i-th attribute's value of the instance)
A_i is the i-th attribute's name.
The term in the product operator is telling us the probability of that A_i is having the value E_i knowing that the outcome is H.

According to the formula above, all the likelihoods are calculated. The highest value is the most likely. If it is required, the likelihoods can be normalized into probabilities.

The "zero/frequency problem"
If an attribute value does not occur with every class value the probability and also the posterior probability will be zero, regardless to the other likelihood values. To prevent *this Laplace estimator* can be used. The idea is to add a μ constant to the count of every value/class combination for such problematic attribute (in practice μ usually equals 1).

Missing values
During training the instance with a missing value is not included in frequency count for class-value combination.
During prediction the missing attribute will be omitted from the calculation.

Numeric attributes
Some kind of probability distribution is assumed (usually a Gaussian distribution). Instead of value-class pairs the parameters of the distribution (e.g. mean value and standard deviation) are calculated during learning and used in the prediction.

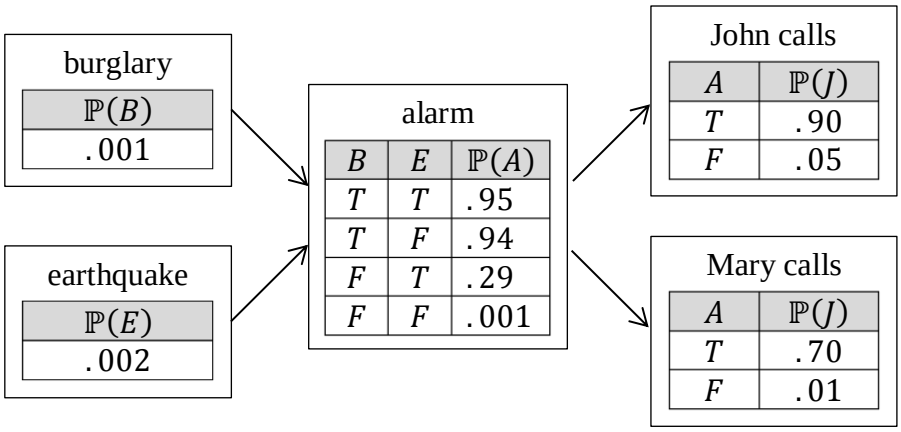
Other properties
The Naïve Bayes method works well in practice. Irrelevant attributes are OK, but many redundant attributes will cause problems.

Bayesian network model / interpretation

Bayes' rule of Conditional Probability
The probability of event H given evidence E is

$$\mathbb{P}(H|E) = \frac{\mathbb{P}(E|H)\mathbb{P}(H)}{\mathbb{P}(E)}$$

The Bayesian network
The network is represented by an acyclic, directed graph. The prior probabilities can be only found in the roots. The lower level elements contain conditional probabilities.



Using the Bayes rule, all type of questions can be answered.

Topic 4 Basic decision tree building, ID3

Basic considerations and goals

ID3 builds a basic decision tree: the nodes involve testing a particular nominal attribute, the leaves assign class value. Let us assume that there are no missing values.

Algorithm

The basic decision tree building algorithm normally works from a top-down recursive *divide-and-conquer* fashion:

- *first*, attribute is selected for root node and branch is created for each possible attribute value.
- *then*, the instances are split into subsets (one for each branch extending from the node)
- *finally*, procedure is repeated recursively for each branch, using only instances that reach the branch.

The process stops if all instances have the same class (or it is better to say that it stops when no further split makes them purer).

Information gain, entropy

For a split, we need a criterion for attribute selection. The best attribute is the one which will result in the smallest tree. There is a heuristic for this, chose the attribute that produces the purest nodes. A popular impurity criterion is *information gain.*

Information gain
Information gain increases with the average purity of the subsets that an attribute produces.

The information gain is

$$\text{gain}(\text{Attribute}) = \text{info}(\text{before split}) - \text{info}(\text{after split})$$

E.g. if a three-way split on attribute *Outlook* gives from [9,5] three groups: [2,3], [4,0], [3,2], then the information gain is $\text{gain}(\text{Outlook}) = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2])$.

In the formulas above

$$\text{info}([a_1,b_1],\dots,[a_n,b_n]) = \sum_{i=1}^{n}\left(\frac{a_i+b_i}{\sum_{j=1}^{n}(a_j+b_j)}\,\text{entropy}\left(\frac{a_i}{a_i+b_i},\frac{b_i}{a_i+b_i}\right)\right)$$

Properties of the information gain
A purity measure should have the following properties:

- its value is 0 if and only if the node is pure,
- its value is maximal (e.g. for binary case: 1) if and only if the impurity is maximal,
- it has the multistage property: $\text{m}([a,b,c]) = \text{m}([a,b+c]) + \left(\frac{b+c}{a+b+c}\right)\cdot\text{m}([b,c])$

The only function that satisfies all three properties is:

Entropy
Given a probability distribution, the info required to predict an event is the distribution's entropy. The formula for computing its value is

$$\text{entropy}(p_1,p_2,\dots,p_n) = -p_1\log p_1 - p_2\log p_2 - \dots - p_n\log p_n$$

Highly branching attributes

Attributes with a large number of values are problematic (extreme case: ID code). Subsets are more likely to be pure if there is a large number of values. This may result in overfitting due to the fact that a non-optimal attribute was chosen because of the high information gain on the many-valued attribute. A possible solution can be the introduction of the *gain ratio.*

Gain ratio

The gain ratio is a modification of the information gain that reduces its bias. Gain ratio takes the number and size of branches into account when choosing an attribute. It corrects the information gain by taking the intrinsic information (entropy of distributing instances into branches) of a split into account. The intrinsic info tells us how much info we need to tell which branch an instance belongs to.

$$\text{gainRatio}(\text{Attribute}) = \frac{\text{gain}(\text{Attribute})}{\text{intrinsicInfo}(\text{Attribute})}$$

Limitations

ID3 realizes a top-down induction of decision tree making. This algorithm cannot deal with numeric attributes, missing values and noisy data. Also, the problem of highly branching attributes is present; its effect can be reduced by using the gain ratio or other modifications. (There are many other attribute selection criteria – with almost no difference in accuracy of the result.)

The ID3 algorithm led to the development of the C4.5 algorithm, which can deal with numeric values, missing attributes and noisy data.

Topic 5 Basic rule construction, PRISM

Basic considerations

Rules
Classification rules are popular alternative to decision trees. A **rule** is built from an antecedent and a consequent part.

Antecedent (pre/condition): a series of test; usually AND-ed together but may also be general logic expressions.

```
IF person's gender = female AND person's number of children > 0
```

Consequent (conclusion): classes, or set of classes, or probability distribution assigned by rule.

```
THEN person = mother
```

The individual rules are often logically OR-ed together.

Set of rules
A set of rules can be ordered or unordered.

Ordered set of rules (also called decision list) has an order: rules should be checked in sequential, one after another. So the order of the rules is important for interpretation.

Unordered set of rules is just a pile of rules. These can overlap and lead to different conclusions for the same instance (depending on the order of interpretation).

The issue of conflicting rules can be solved in case of Boolean attributes: assuming that if instance does not belong to class A then it belongs to class B we can learn rules for class A and use the default rule for class B. Then the order of rules is not important.

Rules with exceptions
Idea is to allow rules to have exceptions:

```
IF person's gender = female AND person's number of children > 0
THEN person = mother
EXCEPT IF person's gender = male THEN person = father
```

They are logically equivalent to **IF** … **THEN** … **ELSE** rules, but exceptions offer a psychological advantage: defaults and tests early on apply more widely than exceptions further down. Exceptions reflect special cases.

Algorithm

A popular idea for rule set generation is that first a decision tree is learned and then it is converted into a set of rules. The other approach which generates a rule set directly is called *covering approach:* for each class in turn, find rule set that covers all instances in it (excluding instances not in the class).

Simple covering algorithm
A simple covering algorithm generates a rule by adding test (new terms) that maximize rule's accuracy. This is similar to situation in decision trees: problem of selecting an attribute to split on.

The goal is to maximize accuracy. For this task we introduce some variables:

- t is the total number of instances covered by rule,
- p is the number of positive examples of the class covered by rule

From t and p two (useful) quantities can be calculated:

$t - p$ is the number of errors made by the rule, and

p/t is the ratio of the positive examples. If $p/t = 1$ then the rule is perfect.

The task of test selection is finished when the p/t ratio is maximal (it is 1 or the set of instances can't be split any further).

If it is necessary to break ties then the test with greater coverage (p) is chosen.

PRISM pseudo code

```
For each class C
  Initialize E to the instance set
  While E contains instances in class C
    Create a rule R with an empty left-hand side that predicts class C
    Until R is perfect (or there are no more attributes to use) do
      For each attribute A not mentioned in R, and each value v,
        Consider adding the condition A = v to the left-hand side of R
      Select A and v pair that maximizes the accuracy p/t
          (break ties by choosing the condition with the largest p)
      Add A = v to R
    Remove the instances covered by R from E
```

Discussion of PRISM

The PRISM algorithm with the outer loop removed generates a decision list for one class. The subsequent rules are designed for instances that are not covered by previous rules; but the order doesn't matter because all rules predict the same class.

Methods like PRISM (for dealing with one class) are separate-and-conquer algorithms:

- first, a rule is defined,
- then all instances covered by the rule are separated out,
- finally, the remaining instances are "conquered"

The difference to divide-and-conquer methods is that a subset covered by rule doesn't need to be explored any further.

Limitations

???

Topic 6 Mining association rules

Form of association rules

The typical form of an association rule is

```
IF A and B occurs in the set THEN Q is also part of the set.
```

They are separate rules and similar to classification with rules, except that there is no selected attribute and *belonging to a class* is a Boolean value.

Support, confidence

Support (or coverage)
The number of instances predicted correctly. This is the same as the number of instances covered by all tests in the rule (both LHS and RHS).

Confidence (or accuracy)
The number of correct predictions, as proportion of all instances that rule applies to. So if the rule is **`IF X THEN Y`** then confidence is $\text{conf} = \frac{\text{supp}(\text{X } and \text{ Y})}{\text{supp}(\text{X})}$

Challenge

How to mine association rules? The naïve method is that we use separate-and-conquer method and treat every possible combination of attribute values as a separate class. This approach has two problems: computationally complex and the resulting number of rules will be hughe.

We can look for high support rules directly: a minimum support and confidence is pre-specified (e.g. support ≥ 2 and confidence $\geq 95\%$)

Item sets

Item: one test (an attribute-value pair)
Item set: all items occurring in a rule (both LHS and RHS)

The goal is to find only that kind of rules that exceed pre-defined support. We do it by finding all item sets with the given minimum support (these are the frequent item sets) and generating rules from them.

Apriori algorithm

Once all item sets with minimum support have been generated, we can turn them into rules. There are $(2^N - 1)$ potential rules.

Example: given the 3-item set **`Humidity = Normal, Windy = False, Play = Yes`**. The possible rules:

```
IF Humidity = Normal AND Windy = False    THEN Play = Yes
IF Humidity = Normal AND Play = Yes       THEN Windy = False
IF Windy = False AND Play = Yes           THEN Humidity = Normal

IF Humidity = Normal                      THEN Windy = False AND Play = Yes
IF Windy = False                          THEN Humidity = Normal AND Play = Yes
IF Play = Yes                             THEN Windy = False AND Humidity = Normal

IF true                                   THEN Humidity = Normal AND
                                               Windy = False AND
                                               Play = Yes
```

Generating item sets efficiently

Question is how can we efficiently find all frequent item sets? (Item sets with support higher than threshold.)

Since finding one-item sets it easy, we can use the one-item sets to generate two-item sets etc. If $(A\ B)$ is a frequent item set, then (A) and (B) have to be frequent item sets as well. In general, if X is a frequent k-item set, then all $(k-1)$-item subsets of X are also frequent.

The algorithm

- Find frequent (support larger than threshold) one-item sets.
- Compute candidate frequent k-item sets by merging frequent $(k-1)$-item sets.
- Check (real) support of candidate frequent k-item sets by reading through the whole database.
- Get rid of those item sets that are below threshold and repeat the process starting from the set merging (second step).

This algorithm reduces the number of sets the support of which has to be checked by reading the whole database.

Topic 7 Linear models

Linear regression

Basic concept
In case of **linear regression** all of the attribute values are *numeric* (both input and output). The basic linear regression gives the output as the sum of weighted input attribute values:

$$x = w_0 + w_1 a_1 + w_2 a_2 + \cdots + w_k a_k$$

where
x output
w_n is the n-th weight
a_n is the n-th attribute value

The question is how to find good weight values.

Training
In the training phase the weights are calculated from the training data. The predicted value for the first training instance $a^{(1)}$ is

$$\hat{x}^{(1)} = w_0 + w_1 a_1^{(1)} + \cdots + w_k a_k^{(1)} = w_0 + \sum_{j=1}^{k} w_j a_j^{(1)}$$

The $k + 1$ coefficients then chosen so that the squared error on the training data is minimized:

$$\sum_{i=1}^{n} \left(x^{(i)} - \hat{x}^{(i)}\right)^2 \to \min$$

The coefficients can be derived using matrix inversion. Finding good arguments is possible if there are more instances than attributes (roughly speaking).

Prediction
To get a predicted $\hat{x}$ value, one has to simply plug in the a_j values into the formula with the learned weights.

Linear models for classification

In case of **binary classification** the output of the classifier is a Boolean class index (0 or 1). In this case the line separates the two classes; the line is treated as a decision boundary: the predicted class is 1 if the output > 0, else the output class is 0.

If there are more than two possible classes the boundary becomes a higher dimension hyper-plane.

Logistic regression
Instead of the meaningless linear classification method, the linear model is transformed. Assuming that we have two classes, the probability of $\mathbb{P}(1|a_1, a_2, \ldots, a_k)$ can be replaced with

$$\log\left(\frac{\mathbb{P}(1|a_1, a_2, \ldots, a_k)}{1 - \mathbb{P}(1|a_1, a_2, \ldots, a_k)}\right)$$

This logit transformation maps the $[0,1]$ interval to the whole $\mathbb{R}$. In this model the probability of belonging to a specific class is described by the formula

$$\mathbb{P}(1|a_1, a_2, \ldots, a_k) = \frac{1}{1 + e^{-w_0 - w_1 a_1 - \cdots - w_k a_k}}$$

Parameters are found from training data using maximum likelihood method. This choses the w_j weights (iteratively) so that the log-likelihood of the model is maximal:

$$\sum_{i=1}^{n}\left[\left(1-x^{(i)}\right)\log\left(1-\mathbb{P}(1|a_1^{(i)},\ldots,a_k^{(i)})\right)+\left(x^{(i)}\right)\log\left(\mathbb{P}(1|a_1^{(i)},\ldots,a_k^{(i)})\right)\right]\rightarrow \max$$

where $x^{(i)}$ are either 1 or 0.

Prediction is made by calculating the value of $\mathbb{P}(1|a_1,a_2,\ldots,a_k)$. The decision boundary is at 0.5, which occurs when $-w_0-w_1a_1-\cdots-w_ka_k=0$.

Perceptron

If all we want to do is just classification, we do not need the exact probability estimates. This different approach is on learning the separating hyperplane. Assuming, that the data is linearly separable, we can consider the formula of the linear regression extended with a learning algorithm called *perceptron learning rule.*

The hyperplane is described by

$$0=w_0+w_1a_1+\cdots+w_ka_k$$

If the sum is greater than zero, we predict class A, else class B.

The perceptron learning algorithm

```
Set all weights to zero
Until all instances in the training data are classified correctly
  For each instance I in the training data
    If I is classified incorrectly by the perceptron
      If I belongs to the first class add it to the weight vector
      else subtract it from the weight vector
```

This works, because if we consider a situation where instance a belongs to class A, it is added to the weight vector, then the new output is

$$w_0+(w_1+a_1)a_1+\cdots+(w_k+a_k)a_k$$

which means that the output for a has increased by

$$a_1a_1+a_2a_2+\cdots+a_ka_k$$

This number is always positive and thus the hyperplane has moved to the correct direction. In the similar way we can show that the output decreases for instances of other class.

Topic 8 Performance measures

Basic considerations

The general question is how predictive is the model we learned. **Resubstitution error** is the error of the model on the training data. This is *not* a good indicator of performance on future data. Resubstitution error is (hopelessly) optimistic.

Training and testing

A simple solution that can be used if lots of data is available is to split the data into training and test sets. Then the model is trained on the training data and the predictions given on the testing set will tell the accuracy of the model.

The test set is a set of independent instances that have played no part in formation of the classifier. Generally, the larger the training data the better the classifier; and the larger the test data the more accurate the error estimate.

In practice, the amount of data is usually limited, so some more sophisticated methods are necessary.

Stratification
In case of splitting the data, we assume that bot training data and validation data are representative samples of the underlying problem. To make sure this assumption holds a modification of the data is uses which is called *stratification*. This ensures that each class is represented with approximately equal proportions in both subsets.

Three-way data split
If model selection and true error estimates are to be computed simultaneously, the data should be divided into three disjoint sets:

- *Training set:* used for learning, i.e. to fit the parameters of the classifier
- *Validation set:* used to select among several trained classifiers
- *Test set:* used only to assess the performance of a fully-trained classifier

The error rate of the final model on validation data will be biased since the validation set is used to select the final model. After assessing the final model on the test set, you must not tune the model any further.

Performance measures

Choice of performance measure
A meaningful performance measure can be:

- the number of correct classifications
- the accuracy of the probability estimates
- the error in numeric predictions

A natural performance measure for classification is the error rate: the number of correctly classified instances over the number of all instances.

Confusion matrix
In case of binary classification the actual class and predicted class can be written along the axes of a table called confusion matrix. In such an arrangement the matrix has four values, true-positives, true-negatives, false-positives and false-negatives.

		predicted class: Yes	predicted class: No
actual class	Yes	true positive	false negative
	No	false positive	true negative

Representative values

Name	Formula	Description
success rate	$\frac{TP + TN}{TP + TN + FP + FN}$	Proportion of correctly classified items.
precision	$\frac{TP}{TP + FP}$	Proportion of retrieved items that are relevant.
recall	$\frac{TP}{TP + FN}$	Proportion of relevant items that are returned.
sensitivity		Proportion of positives that are correctly identified
specificity	$\frac{TN}{FP + TN}$	Proportion of negatives that are correctly identified
F-measure	$\frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$	Harmonic mean of precision and recall

Kappa statistics

Kappa statistics work on multi-class confusion matrices ($\# \geq 2$). For a model, two confusion matrices are created. One is made by the actual predictor, the other one is created by a random predictor. The sum of diagonal elements in a matrix A is denoted by D_A. Then:

$$\kappa = \frac{D_{\text{observed}} - D_{\text{random}}}{D_{\text{perfect}} - D_{\text{random}}}$$

The kappa statistic measures relative improvement over random predictor.

Considering costs

In practice, different types of classification errors often incur different costs. E.g. in case of terrorist profiling the assumption “not a terrorist” is correct 99% of the time, but a false negative could be worse than a false positive alert.

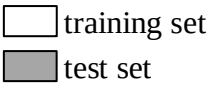
Cost sensitive classification

This method can take costs into account when making predictions. The basic idea is to only predict high-cost class when very confident about the prediction. So, given the predicted class probabilities, the predictor should minimize the expected cost.

Cost sensitive learning

It is the opposite of the cost sensitive classification. The cost matrix is considered during training.

Topic 9 Limited data problem

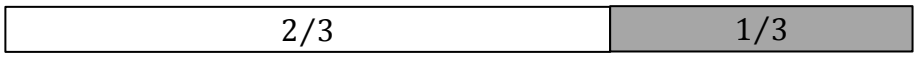


Limited data techniques

Since data is usually limited, using a separate training and test set is not the best idea. There are more sophisticated methods that make it possible to use the whole data for training and validation.

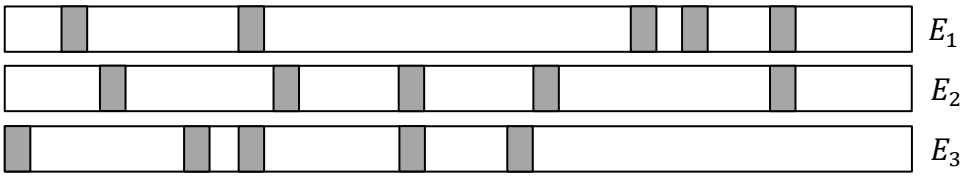
Holdout

This is the basic method: it separates the dataset into two parts. The holdout method reserves a certain amount for testing and uses the remainder for training. Usually, one third of the data is for testing, the rest is for training.



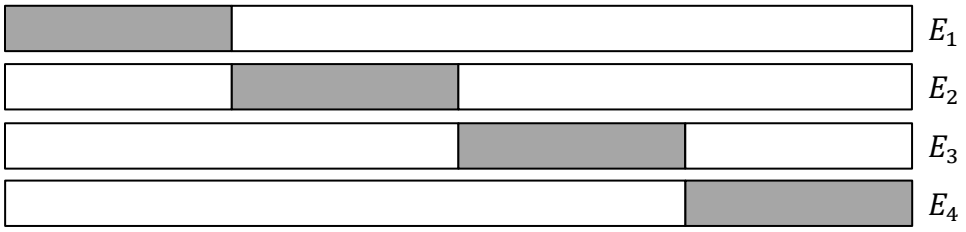
Repeated holdout method; random subsampling

There are K data splits of the entire dataset. Each split selects a fixed number of examples without replacement. For each data split we retrain the classifier from scratch with the training examples and the estimate E_i with the test examples. The true error estimate is an average of the separate estimates. This method is still not optimal as the test sets may overlap.



Cross-validation

First, the data is split into k subsets of equal size. Next, each subset in turn is used for testing and the remainder for training. The error estimates are averaged. Often the subsets are stratified before cross-validation is performed.



A standard method for evaluation is stratified ten-fold cross validation (empirical result).

Leave-one-out cross-validation

This method is a degenerate case of the k-fold cross-validation, where k equals the number of instances in the dataset. The advantage of it is that it makes maximum use of data and no random subsampling is involved. The disadvantages are that stratification is impossible and this method is computationally very expensive.

Bootstrapping

This method uses sampling with replacement to form the training set. First, a dataset of n instances is sampled n times with replacement to form a new dataset of n instances. Then this data is used for training, and those instances that don't occur in the new training set are used for testing.

A particular instance has a probability of $(1 - 1/n)$ of not being picked. Thus its probability of ending up in the test data is

$$\left(1 - \frac{1}{n}\right)^n \approx e^{-1} = 0.368$$

This means that the training data will contain approximately 63.2% of the instances. The error estimate on the test data is very pessimistic, thus it is combined with the resubstitution error:

$$\text{err} = 0.632\ \text{err}_{\text{test}} + 0.368\ \text{err}_{\text{training}}$$

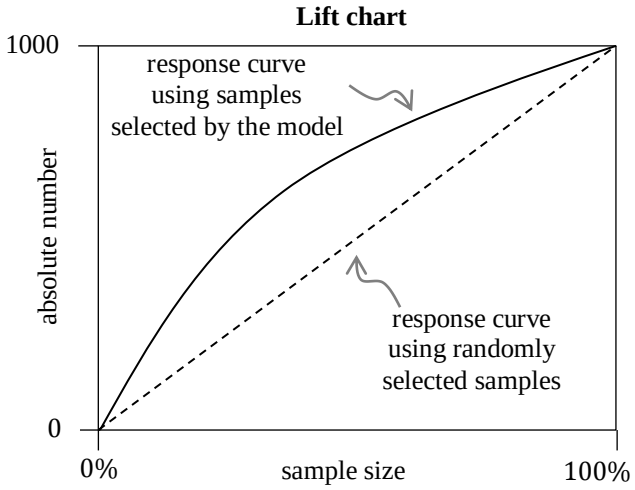
This method is probably the best way of estimating performance for very small datasets.

Charts and curves

Lift chart

In practice, costs are rarely known. Decisions are usually made by comparing possible scenarios. *Example:* promotional mailout to 1,000,000 households. Mail to all result in 0.1% response rate (1,000 responses). A data mining tool identifies subset of 100,000 most promising targets, 0.4% respond (400 responses). In this example reaching 40% of the responses while the costs are just only 10% may pay off.

A lift chart allows a visual comparison.



ROC curve

ROC stands for "receiver operating characteristic". ROC curves are similar to lift charts. They are used in signal detection to show tradeoff between hit rate and false alarm rate over noisy channel. The y axis shows the percentage of true positives in sample, while the x axis shows the percentage of false positives in sample.

Evaluating numeric prediction

For evaluating numeric predictions the same strategies are used: independent test set, cross-validation, significance tests etc. The difference is in the error measure.

Error measures for numeric prediction

The most popular error measure is the *mean squared error*:

$$\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2$$

where a_i is an actual target value and p_i is a prediction for that; n is the number of values. Other measures are *root mean squared error*:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2}$$

mean absolute error:

$$\frac{1}{n}\sum_{i=1}^{n}|p_i - a_i|$$

or *relative error values*, i.e. 10% for an error of 50 when predicting 500.

Often we want to know how much the scheme improves on simply predicting the average ($\bar{a}$). The *relative squared error* and *relative absolute error* are good measure for it:

$$\sum_{i=1}^{n}\frac{(p_i - a_i)^2}{(\bar{a} - a_i)^2} \qquad\qquad \sum_{i=1}^{n}\frac{|p_i - a_i|}{|\bar{a}_i - a_i|}$$

The correlation coefficient measures the statistical correlation between the predicted values and the actual values:

$$\frac{S_{PA}}{\sqrt{S_P S_A}}$$

where

$$S_{PA} = \frac{1}{n-1}\sum_{i=1}^{n}(p_i - \bar{p})(a_i - \bar{a}),\quad S_P = \frac{1}{n-1}\sum_{i=1}^{n}(p_i - \bar{p})^2,\quad S_A = \frac{1}{n-1}\sum_{i=1}^{n}(a_i - \bar{a})^2$$

This measure is scale independent and produces values in the $[-1, +1]$ interval.

Topic 10 Industrial strength algorithms, C4.5

Requirements

In order to be useful in a wide range of real-world applications, an algorithm has to be:

- robust in a presence of noise (avoids overfitting)
- able to deal with numeri attributes
- able to deal with missing values
- scalable.

Noisy data
The data contains noise for several reasons (error of recordation, mislabeled item in the training set etc.). A robust algorithm must deal with the problem of overfitting → *pruning.*

Numeric attributes
Not only nominal attributes are possible. A good algorithm has to deal with numeric attributes by splitting then into ranges (e.g. → binary split in case of C4.5)

Missing values
A missing value should be handled both during training and prediction. Several methods are possible, take in count the probability distribution of the attribute that has the missing value.

Scalability
Algorithms should be feasible on small and huge datasets as well.

C4.5

C4.5 is an industrial strength tree building algorithm. It provides solutions for the problems mentioned before.

Numeric attributes
The standard method of dealing with numeric attributes is binary splitting. Every numeric attribute offer many possible split points. To select the "best" split point, the information gain is evaluated on every possible split point of the attribute. This is computationally expensive, but the best approach.

This method works well (fast) if the data is sorted beforehand. The sorting should be carried out once, and after the order is created, if it is kept, then the splits can be made easily.

Further modification of the method is possible with multi-way splitting (i.e. trinary etc.).

Missing values
During *training* the C4.5 algorithm splits instances with missing values into pieces. A piece going down a particular branch receives a weight proportional to the popularity of the branch. The assigned weights sum up to1. Info gain etc. can be used with fractional instances using sums of weights instead of counts.

During *classification*, the same process is used to split instances into pieces. The results are merged using weights.

Pruning
Pruning simplifies a decision tree to prevent overfitting to noise in the data. Two main pruning strategies are *postponing* (pruning a fully-grown tree) and *prepruning* (stops growing a branch when no more improvement are possible. Postpruning is preferred in practice because of early stopping in prepruning.

Prepruning
Usually, it is based on significance test. This method stops growing the tree when there is no statistically significant association between any attribute and the class at a particular node. The problem of this method is that it may stop the growth process prematurely. Prepruning is faster than postpruning.

Postpruning
First, the tree is fully built. Then we prune it, since some subtrees are might exists due to noise or random effects. The two pruning operations in postpruning are subtree replacement and subtree raising. The decision of applying a pruning operation is based on error estimation.

Pruning decision
Error on the training data is not a useful estimator. Hold-out method can be used to generate a set for pruning. However, C4.5 uses a more complex method. It derives a confidence interval from the training data and uses a heuristic limit derived from this.

The error estimate for the subtree is the weighted sum of error estimates for all its leaves. Error estimate for a node is:

$$\text{err} = \frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N}}}{1 + \frac{z^2}{N}}$$

where f is the error on the training data, N is the number of instances contained by the leaf and z is related to confidence level through normal distribution.

Discussion of C4.5
C4.5 can be slow for large and noisy datasets. It offers two parameters: the confidence value and a threshold on the minimum number of instances in the two most popular branches. C4.5 builds univariate decision trees (one attribute per test).

Topic 11 Support vector machines

Extending linear classificators

The linear classifiers can't model nonlinear class boundaries. Therefore we use a simple trick: map the attributes into a new space consisting of combinations of attribute values. For example, with two attributes and $n = 3$:

$$x = w_1 a_1^3 + w_2 a_1^2 a_2 + w_3 a_1 a_2^2 + w_4 a_2^3$$

Problems with this approach is the *speed* of the algorithm (with 10 attributes and $n = 5$ there are > 2000 coefficients. The second problem is the overfitting; the number of coefficients is large relative to the number of training instances.

Support vector machines

Support vector machines are algorithms for learning linear classifiers. Its aim is to find a linear hyperplane (decision boundary) that separates the data. There are many possibilities for such a line. Which line is the best?

The aim is to find a hyperplane that maximizes the margin of the separation. The instances closest to the maximum margin hyperplane are called support vectors. These vectors define the maximum margin hyperplane on their own, so all other instances can be deleted without changing the position or orientation of the line.

The hyperplane can be written as

$$x = b + \sum_{i \in \{\text{support vector indices}\}} \alpha_i y_i \mathbf{a}(i) \cdot \mathbf{a}$$

where y_i is the class, b and α_i are parameters. The SVM is resilient to overfitting because it learns a particular linear decision boundary.

Finding support vectors

Support vector is a training instance for which $\alpha_i > 0$. The method of determining α_i and b is the solving of a constrained quadratic optimization problem. There are off-the-shelf methods for solving these problems, however, special purpose algorithms are faster.

Nonlinear SVMs

They use a nonlinear transformation to a space, where the classes are linearly separable. A mathematical trick is applied: to avoid computing the pseudo-attributes we compute the dot product before doing the nonlinear mapping:

$$x = b + \sum_{i \in \{\text{support vector indices}\}} \alpha_i y_i (\mathbf{a}(i)\mathbf{a})^n$$

It corresponds to a map into the instance space spanned by all products of n attributes.

Kernel functions

This mapping is called a "kernel function". The example above shows a polynomial kernel. There are many types of available kernels:

$$x = b + \sum_i \alpha_i y_i K(\mathbf{a}(i), \mathbf{a})$$

The only requirement is that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$

Example kernels:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta \mathbf{x}_i \cdot \mathbf{x}_j + b), \qquad K(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{(\mathbf{x}_i, \mathbf{x}_j)^2}{2\sigma^2}}$$

Noise
We have assumed that the data is separable (in original or in transformed). We can apply SVM to noisy data by introducing a regularization parameter C. This C bounds the influence of any one training instance of the decision boundary. So the corresponding constraint is $0 \leq \alpha_i \leq C$. It still remains a quadratic optimization problem; and furthermore, we have to determine C by experimentation.

Advantages
SVM maximizes the margin, so it is robust. It supports kernels, so there is a computable approach for nonlinear problems. The regularization parameter makes the user think about overfitting. SVM is defined by a convex optimization problem, so there are no local minima and there are efficient methods for solving this kind of problems.

Disadvantages
It is not trivial how to choose a kernel function. Also, the hyper-parameter selection for avoiding overfitting is nontrivial. There is also a high algorithmic complexity and extensive memory requirements of the required quadratic programming in large scale tasks.

Sparse data
SVM algorithms speed up dramatically if the data is sparse (many values are 0). SVMs can process sparse datasets with ten thousands of attributes.

Applications
Machine vision (e.g. face identification, handwritten digit recognition), bioinformatics, text classification…

Regression with SVM
The maximum margin hyperplane only applies to classification. However, idea of support vectors and kernel function can be used for regression. The basic method is the same as in linear regression: we want to minimize error. The differences are that we ignore errors smaller than a predefined ε constant; use absolute error instead of squared error, and simultaneously aim to maximize the flatness of the function (avoid overfitting). The user specified ε parameter defines a "tube". The support vectors in this case are the points at the boundary or outside the tube.

Topic 12 Learning Bayesian networks

Naïve Bayes assumes that the attributes are conditionally independent. This does not hold in practice but classification accuracy is often high. However, the performance is much worse than e.g. decision tree. So we are aiming to eliminate the assumption.

Bayesian network

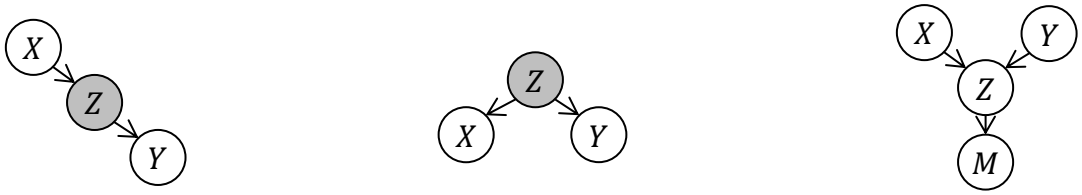
See the title Bayesian network model / interpretation in Topic 3 for a detailed description of the Bayesian networks.

What is important to mention here is that the overall probability distribution is factorized into the component distributions. The graph's nodes hold component distributions (conditional distributions).

d-separation

The nodes X and Y are *d-separated* in on any (undirected) path between X and Y there is some variable Z such that it either

- Z is in a serial or diverging connection and Z is known, or
- Z is in a converging connection and neither Z or any of Z's descendants are known.



Nodes X and Y are called *d-connected* if they are not d-separated (so there exists an undirected path between X and Y not d-separated by any node or a set of nodes).

If nodes X and Y are d-separated by Z, then X and Y are conditionally independent given Z.

Prediction: computing the class probabilities
There are two steps:

1.) Computing a product of probabilities for each class

```
For each class value
  1. Take all attribute values and class value
  2. Look up corresponding entries in conditional probability
     distribution tables
  3. Take the product of all probabilities
```

2.) Normalization

```
Divide the product for each class by the sum of the products.
```

Mathematical background
A single assumption is that values of a node's parents completely determine probability distribution for current node:

$$\mathbb{P}(\text{node}|\text{ancestors}) = \mathbb{P}(\text{node}|\text{parents})$$

It means that the node/attribute is conditionally independent of other ancestors given parents. Then we can apply the chain rule from probability theory:

$$\mathbb{P}(a_1, a_2, \dots, a_n) = \prod_{i=1}^{n} \mathbb{P}(a_i | a_{i-1}, \dots, a_1)$$

Because of our assumption, this rule applied to it is:

$$\mathbb{P}(a_1, a_2, \dots, a_n) = \prod_{i=1}^{n} \mathbb{P}(a_i | a_{i-1}, \dots, a_1) = \prod_{i=1}^{n} \mathbb{P}(a_i | a\text{'s parents})$$

Learning Bayesian Networks

There are four possible cases:

a) Known structure, full observability → maximum likelihood estimation
b) Unknown structure, full observability → search through model space
c) (Known structure, partial observability)
d) (Unknown structure, partial observability)

Likelihood vs. probability
In classification what we really want is to maximize probability of class given other attributes. We are not interested in the probability of the instances. There is no closed form solution for probabilities in nodes' tables that maximize this. However, we can easily compute conditional probability of data based on given network. It seems to work well when used for network scoring.

Known structure, full observability
We use conditional probability distributions for parameters, maximizing the likelihood of the training data.

Unknown structure, full observability
Method for evaluating the goodness of a given network is log-likelihood.
Method for searching through space of possible networks is exhaustive search through the set of edges, because the nodes are fixed.

Algorithms

The K2 algorithm
It starts with a given ordering of nodes (attributes). It processes each node in turn. Greedily tries adding edges from previous nodes to current node. It moves to next node when current node can't be optimized further. The result depends on the initial ordering of the nodes.

Extending K2 algorithm
There is no preset order of nodes. It can greedily add or delete any edges between any pair of nodes. It can also consider the direction of edges.

TAN (Three Augmented Naïve Bayes)
It starts with Naïve Bayes. This method considers adding second parent to each node (apart from class nodes). There are efficient algorithms for this approach.

AD tree
AD stands for all-dimensions. It is analogous to kD-tree for numeric data. It counts in tree. To build such a tree we assume that each attribute in the data has been assigned an index. Then, expand node for attribute i when the values of all attributes $j > i$. Two important restrictions: Most populous expansion for each attribute is omitted. Expansions with counts that are zero are also omitted. After expansions, the root node is given index zero.

Topic 13 Instance-based learning

Distance measures

Distance is the difference between the two attribute values involved (or a function thereof). For several **numeric attributes** the typical distance measure is the Euclidean distance:

$$d = \sqrt{\left(a_1^{(1)} - a_1^{(2)}\right)^2 + \left(a_2^{(1)} - a_2^{(2)}\right)^2 + \cdots + \left(a_k^{(1)} - a_k^{(2)}\right)^2}$$

Taking the square root is not necessary if we are just comparing the distances. Other metrics can be for example the Manhattan norm (adding the distances without squaring).

For **nominal attributes** distance can be defined as 0 if the values are equal and 1 else. If not all the attributes are equally important, then weighting them could be necessary.

Normalization

The different attributes are measured on different scales. They need to be normalized:

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i}$$

A common policy for missing attributes is that they are treated as that they have a maximal distance.

Weight of the attributes, noise

The question is which instance to be saved. If all the instances are preserved then no information is lost, but it needs more storage space and comparison time is higher. When talking about noise we mean that some regions of attribute space are more "stable", some are less.

kd-Tree

A simplest way of finding nearest neighbors is linear scan of the data. The nearest neighbor search can be done more efficiently using appropriate data structures, e.g. kD-tree. The kD-tree stores the instances of the space according to the size of the hyper-rectangles. The biggest hyper-rectangle is represented by the root of the tree and the levels further down are the smaller sub-rectangles.

The complexity of a kD-tree depends on the depth of the tree. The amount of backtracking required depends on the quality of the tree ("square" vs. "skinny" nodes).

To build a kD-tree we split along the direction of the greatest variance and the split point is the median value along that direction.

Incrementally updatable model

A big advantage of instance-based learning is that a classifier can be updated incrementally. We are just adding new training instances. The heuristic strategy for incrementally building a kD-tree is that

- we find a leaf ode containing the new instance,
- place instance into leaf, if leaf is empty, or
- otherwise, split leaf according to the longest dimension (to preserve squareness)

The tree should be re-built occasionally (i.e. if tree depth grows too big).

Generalization

The generalization of the instance-based learning is that the nearest neighbor rule is used outside rectangles; the rectangles are rules, and nested rectangles are rules with exceptions.

Topic 14 Clustering

k-means

The k-means algorithm produces disjoint, deterministic and flat clusters. The number of clusters is k.

Algorithm
First we chose k cluster centers (e.g. randomly). Then repeat until convergence: 1.) assign instances to clusters (based on some distance measure); and 2.) (re)compute the centroids of the clusters.

Limitations of k-means
The algorithm gives good results if the number of actual clusters is already known. The distance measure and the selected attributes also can be a problem these do not map the data well into a space where it is separable by disjoint coverings.

Recursive k-means algorithm
The recursive k-means algorithm recursively splits the dataset into $k = 2$ clusters. It uses a stopping criterion (eg. MDL). The seeds for sub-clusters can be chosen by seeding along direction if greatest variance in the cluster. The X-means algorithm is one kind of this method.

Hierarchical clustering

In case of hierarchical clustering, we recursively split the clusters; producing a hierarchy, that can be represented as a dendogram. (it could also be represented as a Venn diagram of sets and subsets (without intersections). The height of each node in the dendogram can be made proportional to the dissimilarity between its children.

Incremental clustering

Agglomerative clustering (also called bottom-up clustering) requires a distance/similarity measure. It starts by considering each instance to be a cluster, and then it finds the two closest clusters and merges them. The process is repeated until only one cluster is left. The records form a binary dendogram.

Density-based clustering

The model in this case is that the clusters (may) have high point density inside them and low point density regions separate them. The parameters of the method are the minimal density in a cluster and a distance function.
The approach is to extend the clusters by neighbors as long as the expected density holds.

Expectation Maximization algorithm

EM generalizes k-means into probabilistic setting. It is an iterative procedure:

E "expectation" step: Calculate cluster probability for each instance (improving mapping of instances to clusters).

M "maximization" step: Estimate distribution parameters from cluster probabilities (adopting, improving clusters) probability of training data given the clusters.

Stop when improvement is negligible. This algorithm finds a local maximum of the likelihood

Topic 15 Extending numeric prediction

For numeric prediction there are counterparts existing for all schemes that we previously discussed (decision trees, rule learners etc.). All classification schemes can be applied to regression problems using discretization.

The prediction is the weighted average of intervals' midpoints. Regression is more difficult than classification (i.e. not a binary error but a mean squared one.)

Regression trees

Regression trees are different from decision trees:

- splitting criterion is to minimize intra-subset variation,
- termination criterion is that the standard deviation becomes small (or the number of remaining instances is small, e.g. 4),
- pruning criterion is based on numeric error measure,
- and leaf node predict average values of training instances reaching that node (with a standard deviation).

Model trees

Model trees are regression trees with linear regression formulas (models) at each node. Linear regression applied to instances that reach a node after full regression tree has been built. Only a subset of the attributes is used for the linear regression. It is pretty fast, since the overhead for linear regression is not large because usually only a small subset of attributes is used in tree.

Smoothing and pruning

Smoothing is a factor in ancestor's predictions. The smoothing formula is

$$p' = \frac{np + kq}{n + k}$$

With smoothing, the same effect can be achieved by incorporating ancestor models into leaves. For latter we need linear regression function at each nodes.

Pruning is based on estimated absolute error of linear regression models. Model trees allow for heavy pruning: often a single linear regression model can replace a whole subtree. Pruning proceeds bottom-up: error for the linear regression model at internal node is compared to error for subtree.

Nominal attributes

Nominal attributes are converted into binary attributes (that can be treated as numeric ones). The nominal values are sorted using average class values. If there are k values, $k - 1$ so called synthetic binary attributes are generated. The i-th binary attribute is 0 if an instance's value is one of the first i in the ordering, 1 otherwise.

Topic 16 Data transformations

Attribute selection

Adding a random (i.e. irrelevant) attribute can significantly degrade C4.5's performance. The problem that we have to solve is attribute selection on lower levels based on smaller and smaller amount of data.

The "cruse of dimensionality" refers to the fact that many algorithms perform poorly if the number of attributes (dimensions) is very high (with respect to the number of instances). It is due to irrelevant, or relevant but redundant attributes (the latter is a kind of unwanted weighting).

Attributes can be selected manually (deep understanding of the learning problem or the meaning of the attributes or by plotting the data and visually select correlations); or it can be done automatically, with two methods:

- learning scheme independent method → filter method
- learning scheme dependent method → wrapper meathod.

Filter method
Attributes with many missing values, low variance or high correlation (redundancy) are filtered. Either it can use a learning scheme (e.g. C4.5, 1R) to select attributes; or can rank the attributes by instance-based learning attribute weighting, linear models.

From the set of selected and ranked attributes the ones with the lowest ranks are removed.

Correlation based feature selection (CFS)
It eliminates both redundant as well as irrelevant attributes. It creates a subset of attributes that individually correlate well with the class, have little inter-correlation and fulfil some math criteria.

Wrapper method
The wrapper method implements a wrapper around the learning scheme. The evaluation criterion is cross-validation performance. This is why this method is time consuming as it does exhaustive search in the space of all possible attribute combinations. However, it can use significance tests to stop the cross-validation for a subset early, if it is unlikely to "win". The selected attributes are not universal; they depend on the learning scheme.

E.g. Naïve Bayes is sensitive to redundant attributes; a wrapper on the learning scheme can choose the ones that are not inter-correlate. There is a forward selection algorithm that detect when a redundant attribute is about to be added.

Projections

Simple transformations can often make a large difference in performance. Some examples of the possible transformations:

- difference of two attributes,
- ratio of two attributes,
- concatenating values of nominal attributes,
- encoding
- adding noise
- removing data randomly
- etc.

Principal Component Analysis (PCA)

This is a method for identifying the important "directions" in the data. PCA can rotate the data into (reduced) coordinate system that is given by those directions.

The goal is to find the direction (axis) of the greatest variance, and find the direction if greatest variance that is perpendicular to the previous direction; then we repeat these steps.

The idea is that those directions preserve distances between instances.

To implement this, we have to find eigenvectors of covariance matrix by diagonalization. The eigenvectors are the directions of the covariance. The scale of the attributes has effect on the outcome, so the data is normally standardized for the PCA. After we have found the eigenvectors, we can transform the data into the new space.

However, PCA has some limitations as it assumes linear sub-spaces and does not consider the class attribute itself.

Discretizing numeric attributes

Some algorithms work only with nominal attributes. Therefore in case of a numeric attribute we have to discretize it first.

Discretizing can be done by crating k binary attributes for the numeric value. It lacks ordering. The other method is to have $k-1$ binary attributes and fill up with ones until i if the number was i. Also, there are other ways as well:

Unsupervised discretization
Unsupervised discretization generates intervals without looking at class labels. Two main strategies are used: equal interval binning, or equal intensity binning (histogram equalization).

Supervised discretization
It can be done as we have seen in C4.5 (sort values, split at every possible point and choose the one that maximizes information gain).

Another way is the entropy-based discretization. In this case, we build a decision tree on the attribute being discretized (with pre-pruning). The splitting criterion is the entropy; the stopping criterion is the Minimum Description Length Principle.

Also, there are an error based methods which takes the number of errors into count.

Data cleansing

Automatic data cleansing improving decision tree: relearns the tree with misclassified instances removed recursively. The performance change is not significant, but the resulting tree is usually smaller. The reason is that we consequently treat local pruning decisions globally.

Better (of course) is to check the misclassified items by a human.

Attribute noise vs class noise
Attribute noise should be left in training set (don't train on clean set and test on dirty one; decision scheme learns to cope with noise)

Systematic class noise (e.g. one class substituted for another): leave in training set

Unsystematic class noise: eliminate from training set, if possible

Topic 17 Ensemble learning

Basic idea, general advantage / disadvantage

The basic idea is that we build different "experts" for the same problem and let them vote. The advantage is that it often improves the predictive performance, but the disadvantage is that it usually produces a model that is very hard to analyze.

Bias-variance decomposition

Given several equally good but different training data sets.

A learning algorithm is *biased* for a particular input if, when trained on each of these data sets, it is systematically incorrect when predicting

A learning algorithm has high *variance* for a particular input if it predicts different output values when trained on different training sets

Bagging

Bagging stands for bootstrap-aggregating. It combines predictions by voting / averaging. This is the simplest way and very simple to implement. The idealized version trains several models on different training sets (created by bootstrapping), and build a classifier that combines the predictions. It almost always improves the performance. Usually the more classifiers the better.

The learning scheme is unstable; a small change in the data can make big change in the model. It reduces variance by voting / averaging, thus reducing the overall expected error. Theoretically it should reduce the variance without changing the bias, but for high-bias classifiers it reduces the bias (and for high-variance it reduces variance).

Randomization

Randomization randomizes the learning algorithm instead of the training data. Some models already have randomization (e.g. initial weights of a neural network). Others can be randomized e.g. if it not choosing the best opportunity in every case but selects among the three best choices.

Rotation forest

Bagging creates ensembles of accurate classifiers with relatively low diversity. Randomness in the learning algorithm increases diversity but sacrifices accuracy of individual ensemble members. Rotation forests have the goal of creating accurate and diverse ensemble members.

An iteration involves randomly dividing the input into k disjoint subsets, then apply PCA to each of the k subsets, and finally learning a decision tree from the k sets of PCA directions.

Rotation forest has the potential to improve on diversity significantly without compromising the individual accuracy.

Boosting

Boosting is about combining models that complement each other. It decides based on voting, and the classifiers are the same type, but! The training of the classifiers is carried out iteratively, the models are not built separately, and a new model is encouraged to become an expert for instances that have been classified incorrectly by the earlier models. Boosting weights a model's contribution by its confidence.

Ada Boost M1

Model Generation

```
Assign equal weight to each training instance.
For each of t iterations:
  Apply learning algorithm to weighted dataset and store resulting model.
  Compute error e of model on dataset and store error.
  If e equal to zero, or e greater or equal to 0.5
    Terminate model generation.
  For each instance in dataset:
    If instance classified correctly by model
       Multiply weight of instance by e / (1 - e).
  Normalize weight for all instances.
```

Classification

```
Assign weight of zero to all classes.
For each of the t (or less) models:
  Add -log(e/(1-e)) to weight of class model predicts.
Return class with highest weight.
```

(Stacking)

Stacking uses meta learner instead of voting to combine predictions of base learners. Predictions of base learners (level-0 models) are used as input for meta learner (level-1 model). Level-1 instance has as many attributes as there are level-0 learners. The base learners are typically different learning schemes. If base learners can output probabilities it's better to use those as input to meta learner.

The predictions on training data can't be used to generate data for level-1 model, therefore cross-validation-like scheme is employed. The resulted model is hard to analyze theoretically: "black magic".

Notes