Classification & Clustering

Sergej Sizov

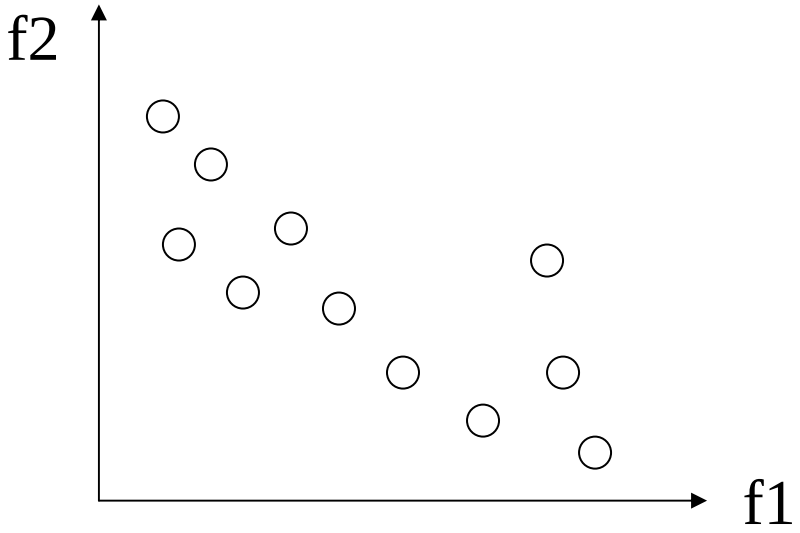
Information Retrieval

Summer term 2009

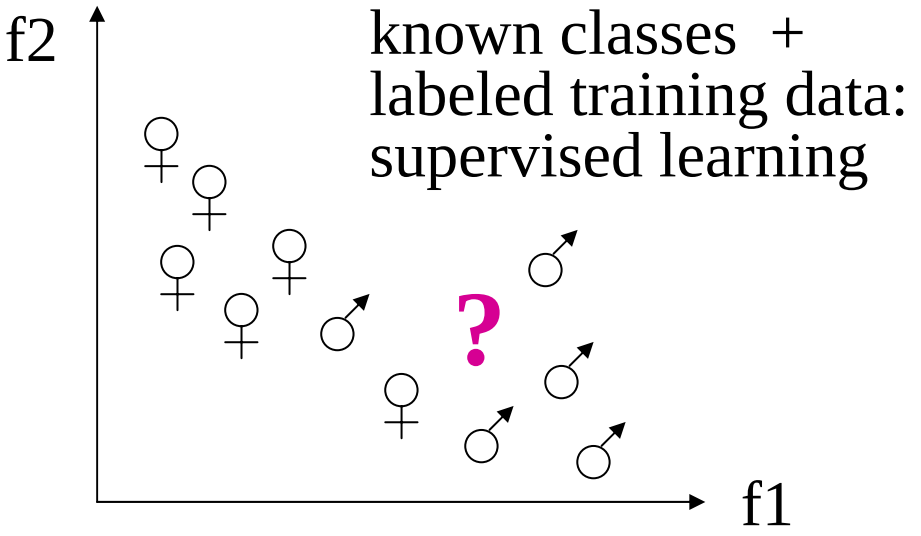
UNIVERSITÄT KOBLENZ · LANDAU

Classification Problem (Categorization)

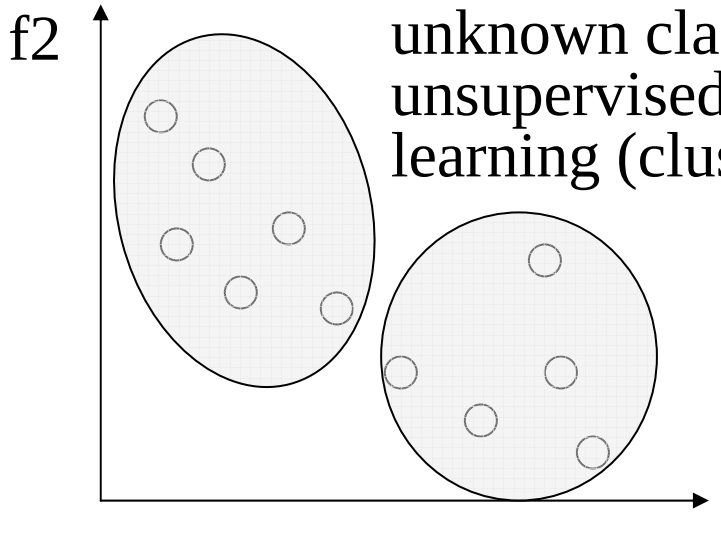
given:
feature vectors



determine class/topic
membership(s)
of feature vectors



known classes + labeled training data: supervised learning



unknown classes: unsupervised learning (clustering)

empirical by automatic classification of documents that
do not belong to the training data
(but in benchmarks class labels of test data are usually known)

For **binary classification** with regard to class C:

a = #docs that are classified into C and do belong to C
b = #docs that are classified into C but do not belong to C
c = #docs that are not classified into C but do belong to C
d = #docs that are not classified into C and do not belong to C

$$\text{Acccuracy (Genauigkeit)} = \frac{a+d}{a+b+c+d}$$ Error (Fehler) = 1–accuracy

$$\text{Precision (Präzision)} = \frac{a}{a+b}$$ $$\text{Recall (Ausbeute)} = \frac{a}{a+c}$$

$$\text{F1 (harmonic mean of precision and recall)} = \left(\frac{1}{precision}+\frac{1}{recall}\right)^{-1}$$

For **manyway classification** with regard to classes C_1, ..., C_k:

- macro average over k classes or
- micro average over k classes

use benchmark collection of completely labeled documents
(e.g., Reuters newswire data from TREC benchmark)

cross-validation (with held-out training data):

- partition training data into k equally sized (randomized) parts,
- for every possible choice of k-1 partitions
  - train with k-1 partitions and apply classifier to k^{th} partition
  - determine precision, recall, etc.
- compute micro-averaged quality measures

leave-one-out validation/estimation:

variant of cross-validation with two partitions of unequal size:
use n-1 documents for training and classify the n^{th} document

Step 1:
find among the training documents of all classes the k (e.g. 10-100) most similar documents (e.g., based on cosine similarity):
the k nearest neighbors of $\vec{d}$

Step 2:
Assign $\vec{d}$ to class Cj for which the function value

$$f(\vec{d}, C_j) = \sum_{\vec{v} \in kNN(\vec{d})} sim(\vec{d}, \vec{v}) * \begin{cases} 1 \text{ if } \vec{v} \in C_j \\ 0 \text{ otherwise} \end{cases}$$

is maximized

With binary classification assign $\vec{d}$ to class C if $f(\vec{d}, C)$ is above some threshold δ ($\delta > 0.5$)

Step 1:

Represent the training documents for class Cj
by a **prototype vector** with tf*idf-based vector components

$$\vec{c}_j := \alpha \frac{1}{|C_j|} \sum_{\vec{d} \in C_j} \frac{\vec{d}}{\|\vec{d}\|} - \beta \frac{1}{|D - C_j|} \sum_{\vec{d} \in D - C_j} \frac{\vec{d}}{\|\vec{d}\|}$$

with appropriate coefficients α and β (e.g. α=16, β=4)

Step 2:

Assign a new document $\vec{d}$ to the class Cj for which
the cosine similarity $cos(\vec{c}_j, \vec{d})$ is maximized.

Feature Selection

For *efficiency* of the classifier and to *suppress noise*
choose subset of all possible features.
→ Selected features should be

- frequent to *avoid overfitting* the classifier to the training data,
- but not too frequent in order to be characteristic.

Features should be good ***discriminators*** between classes
(i.e. frequent/characteristic in one class but infrequent in other classes).

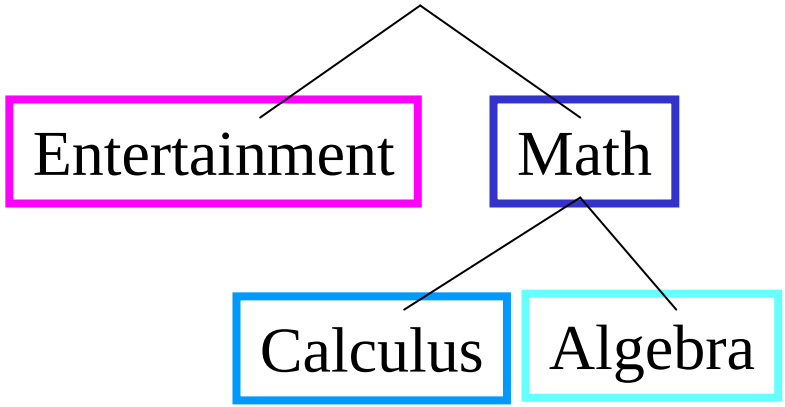
Approach:
- compute measure of discrimination for each feature
- select the top k most discriminative features in greedy manner

tf*idf is usually not a good discrimination measure,
and may give undue weight to terms with high idf value
(leading to the danger of overfitting)

Example for Feature Selection

	film	hit	chart	theorem	limit	integral	group	vector
	f1	f2	f3	f4	f5	f6	f7	f8
d1:	1	1	0	0	0	0	0	0
d2:	0	1	1	0	0	0	1	0
d3:	1	0	1	0	0	0	0	0
d4:	0	1	1	0	0	0	0	0
d5:	0	0	0	1	1	1	0	0
d6:	0	0	0	1	0	1	0	0
d7:	0	0	0	0	1	0	0	0
d8:	0	0	0	1	0	1	0	0
d9:	0	0	0	0	0	0	1	1
d10:	0	0	0	1	0	0	1	1
d11:	0	0	0	1	0	1	0	1
d12:	0	0	1	1	1	0	1	0

Class Tree:



training docs:

d1, d2, d3, d4
→ Entertainment

d5, d6, d7, d8
→ Calculus

d9, d10, d11, d12
→ Algebra

Document Frequency Thresholding:
Consider for class Cj only terms ti that occur in at least δ training documents of Cj.

Term Strength:
For decision between classes C1, ..., Ck select (binary) features Xi with the highest value of

$$s(X_i) := P[X_i \text{ occurs in doc } d \mid X_i \text{ occurs in similar doc } d']$$

To this end the set of similar doc pairs (d, d‘) is obtained
- by thresholding on pairwise similarity or
- by clustering/grouping the training docs.

+ further possible criteria along these lines

Information gain:

For discriminating classes c1, ..., ck select the binary features Xi (term occurrence) with the largest gain in entropy

$$G(Xi) = \sum_{j=1}^{k} P[c_j] \log_2 \frac{1}{P[c_j]}$$

$$- P[X_i] \sum_{j=1}^{k} P[c_j \mid X_i] \log_2 \frac{1}{P[c_j \mid X_i]}$$

$$- P[\bar{X}_i] \sum_{j=1}^{k} P[c_j \mid \bar{X}_i] \log_2 \frac{1}{P[c_j \mid \bar{X}_i]}$$

can be computed in time O(n)+O(mk)
for n training documents, m terms, and k classes

Mutual information (Kullback-Leibler distance, relative entropy): for class cj select those binary features Xi (term occurrence) with the largest value of

$$MI(X_i, c_j) = \sum_{X \in \{X_i, \bar{X}_i\}} \sum_{C \in \{c_j, \bar{c}_j\}} P[X \wedge C] \log \frac{P[X \wedge C]}{P[X]P[C]}$$

and for discriminating classes c1, ..., ck:

$$MI(X_i) = \sum_{j=1}^{k} P[c_j] MI(X_i, c_j)$$

can be computed in time O(n)+O(mk) for n training documents, m terms, and k classes

estimate: $P[\, d \in c_k \,|\, d \text{ has } \vec{X}\,] = \dfrac{P[\, d \text{ has } \vec{X} \,|\, d \in c_k\,]\, P[\, d \in c_k\,]}{P[\, d \text{ has } \vec{X}\,]}$

$\sim P[\, X \,|\, d \in c_k\,]\, P[\, d \in c_k\,]$

$= \Pi_{i=1}^{m} P[\, X_i \,|\, d \in c_k\,]\, P[\, d \in c_k\,]$

with feature independence or linked dependence:

$$\frac{P[\, X \,|\, d \in c_k\,]}{P[\, X \,|\, d \notin c_k\,]} = \Pi_i \frac{P[\, X_i \,|\, d \in c_k\,]}{P[\, X_i \,|\, d \notin c_k\,]}$$

$= \Pi_{i=1}^{m} p_{ik}^{\,Xi} (1 - p_{ik})^{1-Xi}\, p_k$

with empirically estimated $p_{ik}=P[X_i=1|c_k]$, $p_k=P[c_k]$

$$\Rightarrow \log P[c_k \,|\, d] \sim \sum_{i=1}^{m} X_i \log \frac{p_{ik}}{(1-p_{ik})} + \sum_{i=1}^{m} \log(1-p_{ik}) + \log p_k$$

for binary classification with odds rather than probs for simplification

Discriminative Classifiers: Decision Trees

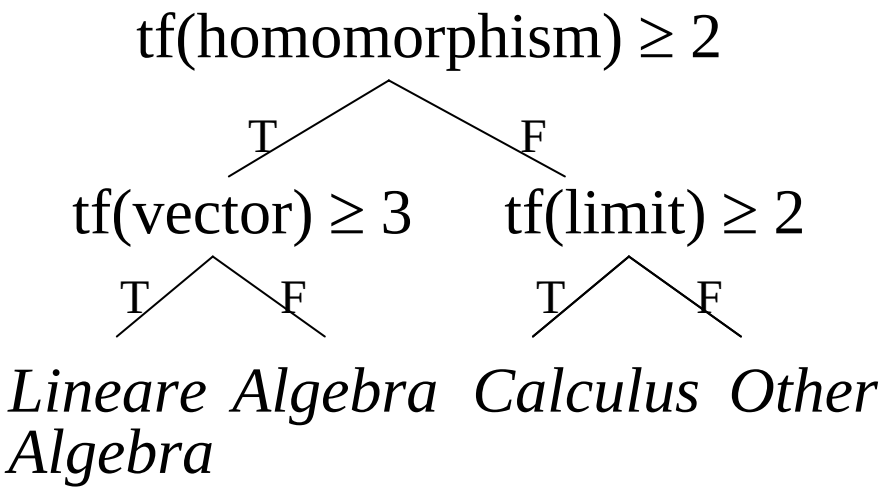
given: a multiset of **m-dimensional training data records** $\subseteq$ dom(A1) $\times$ ... $\times$ dom(Am) with numerical, ordinal, or categorial attributes Ai (e.g. term occurrence frequencies $\subseteq N_0 \times ... \times N_0$) and with **class labels**

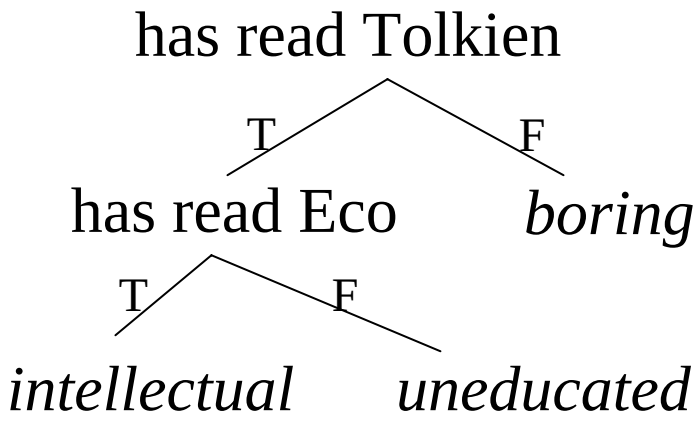
wanted: a **tree** with

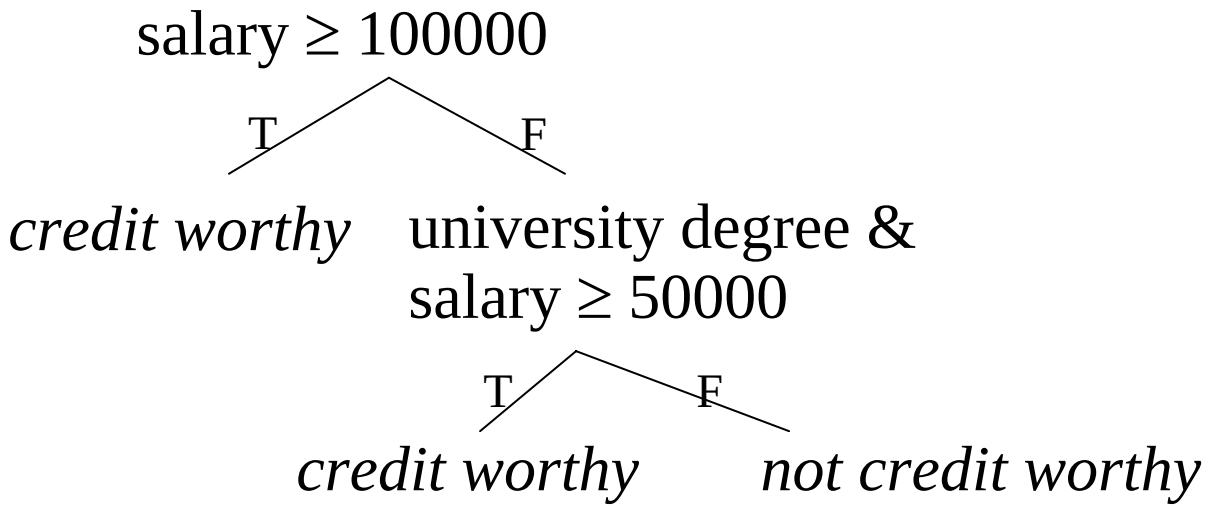
- **attribute value conditions** of the form
  - Ai $\leq$ value for numerical or ordinal attributes
    or
  - Ai $\in$ value set or Ai $\cap$ value set = $\varnothing$
    for categorial attributes
    or
  - linear combinations of this type for several numerical attributes $\sum k_i A_i \leq value$

  as inner nodes and
- **labeled classes as leaf nodes**

Examples for Decision Trees (1)







Input: decision tree node k that represents one partition D of $dom(A1) \times ... \times dom(Am)$

Output: decision tree with root k

1) BuildTree (root, $dom(A1) \times ... \times dom(Am)$)
2) PruneTree: reduce tree to appropriate size

with:

procedure BuildTree (k, D):

if k contains only training data of the same class then terminate;
determine split dimension Ai;
determine split value x for most suitable partitioning of D into
$D1 = D \cap \{d \mid d.Ai \leq x\}$ and $D2 = D \cap \{d \mid d.Ai > x\}$;
create children k1 and k2 of k;
BuildTree (k1, D1); BuildTree (k2, D2);

Split Criterion: Information Gain

Goal is to split current node such that the resulting partitions are as pure as possible w.r.t. class labels of the corresponding training data. Thus we aim to minimize the **impurity** of the partitions.

An approach to define impurity is via the entropy-based (statistical) **information gain**
(referring to the distribution of class labels within a partition)

$$G(k, k1, k2) = H(k) - (p1*H(k1) + p2*H(k2))$$

where:

n_k: # training data records in k

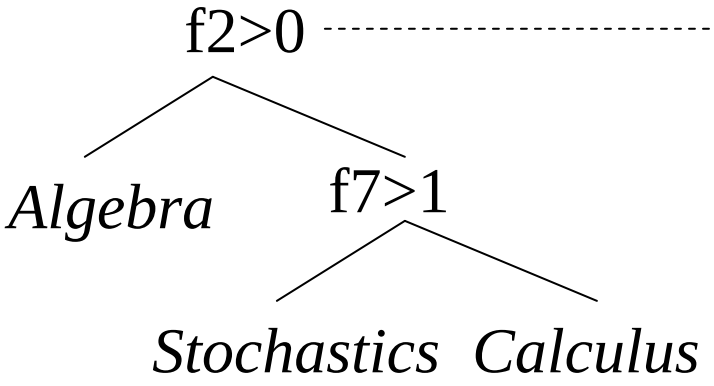
$n_{k,j}$: # training data records in k that belong to class j

$p1 = n_{k1} / n_k$ and $p2 = n_{k2} / n_k$

$$H(k) = -\sum_j \frac{n_{k,j}}{n_k} log_2 \frac{n_{k,j}}{n_k}$$

Example for Decision Tree for Text Classification

		group	homomorphism	vector	integral	limit	variance	probability.	dice
		f1	f2	f3	f4	f5	f6	f7	f8
C1: Algebra	d1:	3	2	0	0	0	0	0	1
	d2:	1	2	3	0	0	0	0	0
C2: Calculus	d3:	0	0	0	3	3	0	0	0
	d4:	0	0	1	2	2	0	1	0
C3: Stochastics	d5:	0	0	0	1	1	2	2	0
	d6:	1	0	1	0	0	0	2	2



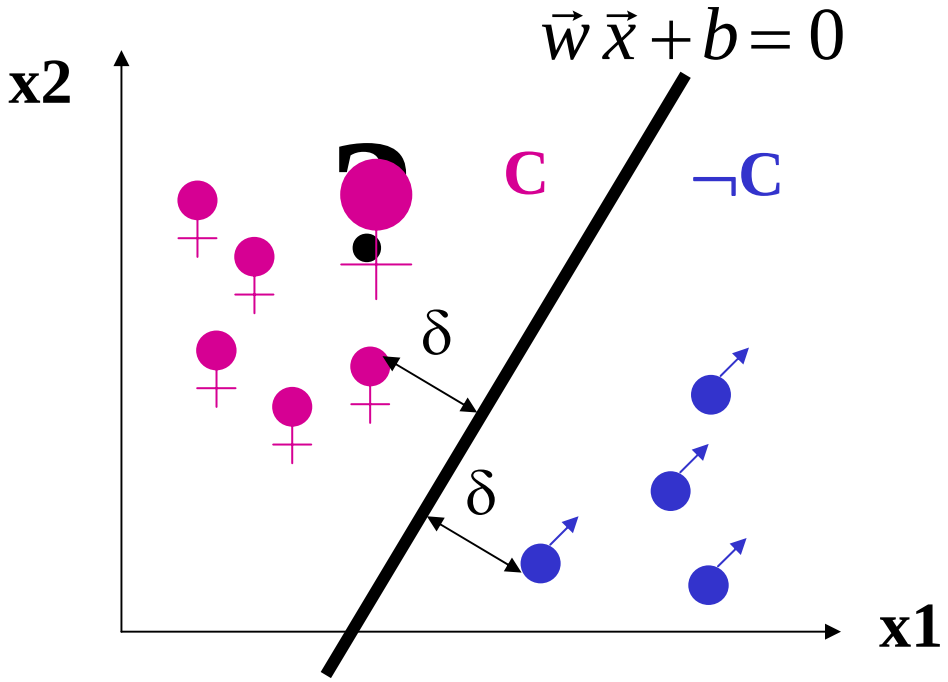
$$G = H(k) - (2/6*H(k1) + 4/6*H(k2))$$

$$H(k) = 1/3 \log 3 + 1/3 \log 3 + 1/3 \log 3$$

$$H(k1) = 1 \log 1 + 0 + 0$$

$$H(k2) = 0 + 1/2 \log 2 + 1/2 \log 2$$

$$G = \log 3 - 0 - 2/3*1 \approx 1{,}6 - 0{,}66 = 0{,}94$$



n training vectors
$(x_1, ..., x_m, C)$
with C = +1 or -1

large-margin separating hyperplane minimizes risk of classification error

Determine ***hyperplane*** $\vec{w}\,\vec{x}+b=0$ that optimally ***separates*** the training vectors in C from those not in C, such that the (Euclidean) distance δ of the (positive and negative) training samples closest to the hyperplane is maximized. (Vectors with distance δ are called ***support vectors***.)

Classify new test vector $\vec{y}$ into C if: $(\vec{w}\,\vec{y}+b)=\sum_{i=1}^{m} w_i y_i + b > 0$

SVM Engineering

+ Very efficient implementations available
  (e.g., SVM-Light at http://svmlight.joachims.org/):
  with training time empirically found to be
  ≈ quadratic in # training docs (and linear in # features)

+ SVMs can and should usually consider all possible features
(no point for feature selection unless #features intractable)

+ multi-class classification mapped to multiple binary SVMs:
  one-vs.-all or combinatorial design of subset-vs.-complement

– Choice of kernel difficult
  and highly dependent on data and application

Motivation:

- classifier can only be as good as its training data
- and training data is expensive to obtain as it requires intellectual labeling
- and training data is often sparse regarding the feature space

→ use additional unlabeled data to improve the classifier‘s implicit knowledge of term correlations

Example:

- classifier for topic „cars“ has been trained only with documents that contain the term „car“ but not the term „automobile“
- in the unlabeled docs of the corpus the terms „car“ and „automobile“ are highly correlated
- test docs may contain the term „autombobile“ but not the term „car“

Simple Iterative Labeling

Let D^K be the set of docs with known labels (training data)
and D^U the set of docs with unknown labels.

Algorithm:
train classifier with **D^K** as training data
classify docs in **D^U**
repeat
    re-train classifier with **D^K** and the now labeled docs in D^U
    classify docs in **D^U**
until labels do not change anymore (or changes are marginal)

Robustness problem:
a few misclassified docs from **D^U** could lead
the classifier to drift to a completely wrong labeling

Co-Training for Orthogonal Feature Spaces

Idea:

- start out with two classifiers A and B for „orthogonal“ feature spaces (whose distributions are conditionally independent given the class labels)
- add best classified doc of A to training set of B, and vice versa (assuming that the same doc would be given the same label by A and B)

Algorithm:

train A and B with orthogonal features of $\mathbf{D^K}$
    (e.g., text terms and anchor terms)
$\mathbf{D^U_A := D^U;\ D^U_B := D^U;\ D^K_A := D^K;\ D^K_B := D^K;}$
repeat
    classify docs in $\mathbf{D^U_A}$ by A and $\mathbf{D^U_B}$ by B
    select the best classified docs from $\mathbf{D^U_A}$ and $\mathbf{D^U_B}$: d_A and d_B
    add d_A to training set $\mathbf{D^K_B}$, add d_B to training set $\mathbf{D^K_A}$
    retrain A using $\mathbf{D^K_A}$, retrain B using $\mathbf{D^K_B}$
until results are sufficiently stable
assign docs from $\mathbf{D^U}$ to classes on which A and B agree

Combine multiple classifiers for more robust results
(usually higher precision and accuracy,
possibly at the expense of reduced recall)

Examples (with m different binary classifiers for class k):

- **unanimous decision:** $C_k(d_j) = 1 \text{ if } \sum_{\nu=1}^{m} C_k^{(\nu)} = m$

- **weighted average:** $C_k(d_j) = 1 \text{ if } \sum_{\nu=1}^{m} \tilde{p}_k^{(\nu)} C_k^{(\nu)} \geq \tau$

  with precision estimator $\tilde{p}_k^{(\nu)}$
  for classifier ν

for further info see machine learning literature
on **ensemble learning (stacking, boosting, etc.)**

Classification: Practical Issues

Gee, I'm building a text classifier for real, now!

What should I do?

How much training data do you have?

- None
- Very little
- Quite a lot
- A huge amount and its growing

Manually written rules

No training data, adequate editorial staff?

Never forget the hand-written rules solution!

- If (wheat or grain) and not (whole or bread) then
  - Categorize as grain

In practice, rules get a lot bigger than this

- Can also be phrased using tf or tf.idf weights

With careful crafting (human tuning on development data) performance is high:

- Construe: 94% recall, 84% precision over 675 categories (Hayes and Weinstein 1990)

Amount of work required is huge

- Estimate 2 days per class … plus maintenance

Very little data?

If you're just doing supervised classification, you should stick to something high bias

- There are theoretical results that Naïve Bayes should do well in such circumstances (Ng and Jordan 2002 NIPS)

The interesting theoretical answer is to explore semi-supervised training methods:

- Iterative labeling, co-training, …

The practical answer is to get more labeled data as soon as you can

- How can you insert yourself into a process where humans will be willing to label data for you?? How can I get feedback? (implicitely or explicitely)

A reasonable amount of data?

Perfect!

We can use all our clever classifiers

Roll out the SVM!

But if you are using an SVM/NB etc., you should probably be prepared with the "hybrid" solution where there is a boolean overlay

- Or else to use user-interpretable Boolean-like models like decision trees
- Users like to hack, and management likes to be able to implement quick fixes immediately

A huge amount of data?

This is great in theory for doing accurate classification…

But it could easily mean that expensive methods like SVMs (train time) or kNN (test time) are quite impractical

Naïve Bayes can come back into its own again!

- Or other advanced methods with linear training/test complexity like regularized logistic regression (though much more expensive to train)

How many categories?

A few (well separated ones)?

- Easy!

A zillion closely related ones?

- Think: Yahoo! Directory, Library of Congress classification, legal applications
- Quickly gets difficult!
  - Classifier combination is always a useful technique
    - Voting, bagging, or boosting multiple classifiers
  - Much literature on hierarchical classification
    - Mileage fairly unclear
  - May need a hybrid automatic/manual solution

How can one tweak performance?

Aim to exploit any domain-specific useful features that give special meanings or that zone the data

- E.g., an author byline or mail headers

Aim to collapse things that would be treated as different but shouldn't be.

- E.g., part numbers, chemical formulas

Does putting in "hacks" help?

You bet!

You can get a lot of value by differentially weighting contributions from different document zones:

- Upweighting title words helps (Cohen & Singer 1996)
  - Doubling the weighting on the title words is a good rule of thumb
- Upweighting the first sentence of each paragraph helps (Murata, 1999)
- Upweighting sentences that contain title words helps (Ko *et al,* 2002)

Two techniques for zones

Have a completely separate set of features/parameters for different zones like the title

Use the same features (pooling/tying their parameters) across zones, but upweight the contribution of different zones

Commonly the second method is more successful: it costs you nothing in terms of sparsifying the data, but can give a very useful performance boost

- Which is best is a contingent fact about the data

Automatic clustering

given:

n ***m-dimensional data records*** dj $\in D \subseteq \text{dom}(A1) \times ... \times \text{dom}(Am)$
with attributes Ai (e.g. term frequency vectors $\subseteq N_0 \times ... \times N_0$)
or n ***data points*** with pair-wise ***distances (similarities)*** in a ***metric space***

wanted:

k **clusters** c1, ..., ck and an assignment D $\rightarrow$ {c1, ..., ck} such that the average **intra-cluster similarity** $\frac{1}{k}\sum_k \left(\frac{1}{|c_k|} \sum_{\vec{d} \in c_k} sim(\vec{d}, \vec{c}_k) \right)$ is high and the average **inter-cluster similarity** $\frac{1}{k(k-1)} \sum_{\substack{i,j \\ i \neq j}} sim(\vec{c}_i, \vec{c}_j)$ is low,

where the ***centroid*** $\vec{c}_k$ of ck is: $\vec{c}_k = \frac{1}{|c_k|} \sum_{\vec{d} \in c_k} \vec{d}$

Hierarchical Clustering:

- detailed and insightful
- hierarchy built in natural manner from fairly simple algorithms
- relatively expensive
- no prevalent algorithm

Flat Clustering:

- data overview & coarse analysis
- level of detail depends on the choice of the number of clusters
- relatively efficient
- K-Means and EM are simple standard algorithms

Principle:

- start with each d_i forming its own singleton cluster c_i
- in each iteration combine the most similar clusters c_i, c_j into a new, single cluster

for i:=1 to n do $c_i := \{d_i\}$ od;
$C := \{c_1, ..., c_n\}$; /* set of clusters */
while $|C| > 1$ do
  determine $c_i, c_j \in C$ with maximal inter-cluster similarity;
  $C := C - \{c_i, c_j\} \cup \{c_i \cup c_j\}$;
od;

Divisive Top-down Clustering

Principle:

- start with a single cluster that contains all data records
- in each iteration identify the least „coherent“ cluster and divide it into two new clusters

$c_1 := \{d_1, ..., d_n\}$;
$C := \{c_1\}$; /* set of clusters */
while there is a cluster $c_j \in C$ with $|c_j|>1$ do
  determine c_i with the lowest intra-cluster similarity;
  partition c_i into c_{i1} and c_{i2} (i.e. $c_i = c_{i1} \cup c_{i2}$ and $c_{i1} \cap c_{i2} = \emptyset$)
  such that the inter-cluster similarity between c_{i1} and c_{i2} is minimized;
od;

For partitioning a cluster one can use another clustering method (e.g. a bottom-up method)

given: similarity on data records - sim: $D \times D \rightarrow R$ oder [0,1]
define: similarity between clusters – sim: $2^D \times 2^D \rightarrow R$ or [0,1]

Alternatives:

- **Centroid method**: sim (c,c‘) = sim(d, d‘) with centroid d of c and centroid d‘ of c‘
- **Single-Link method**: sim(c,c‘) = sim(d, d‘) with d $\in$c, d‘$\in$c‘, such that d and d‘ have the highest similarity
- **Complete-Link method**: sim(c,c‘) = sim(d, d‘) with d $\in$c, d‘$\in$c‘, such that d and d‘ have the lowest similarity
- **Group-Average method**: $\frac{1}{|c| \cdot |c'|} \sum_{d \in c, d' \in c'} sim(d, d')$

For hierarchical clustering the following axiom must hold:
max {sim(c,c‘), sim(c,c‘‘)} $\geq$ sim(c, c‘$\cup$ c‘‘) for all c, c‘, c‘‘ $\in 2^D$

With regard to **ground truth:**
known class labels $L_1, \ldots, L_g$ for data points $d_1, \ldots, d_n$:
$L(d_i) = L_j \in \{L_1, \ldots, L_g\}$

With cluster assignment $\Gamma(d_1), \ldots, \Gamma(d_n) \in c_1, \ldots, c_k$
cluster c_j has **purity** $\max_{v=1..g} |\{d \in c_j \mid L(d) = L_v\}| / |c_j|$

Complete clustering has purity $\sum_{j=1..k} purity(c_j) / k$

Alternatives:

- **Entropy** within cluster $\sum_{v=1..g} \frac{|c_j \cap L_v|}{|c_j|} \log_2 \frac{|c_j|}{|c_j \cap L_v|}$

- **MI** between cluster and classes

$$\sum_{c \in \{c_j, \bar{c}_j\}, L \in \{L_1, \ldots, L_g\}} \frac{|c \cap L| / n}{|c| \cdot |L| / n} \log_2 \frac{|c| \cdot |L| / n}{|c \cap L| / n}$$

Without any ground truth:

ratio of intra-cluster to inter-cluster similarities

$$\frac{1}{k}\sum_{k}\left(\frac{1}{|c_k|}\sum_{\vec{d}\in c_k} sim(\vec{d},\vec{c}_k)\right) \Big/ \left(\frac{1}{k(k-1)}\sum_{\substack{i,j \\ i\neq j}} sim(\vec{c}_i,\vec{c}_j)\right)$$

or other ***cluster validity measures*** of this kind
(e.g. considering variance of intra- and inter-cluster distances)

given: data records d1, ..., dn
wanted: (up to) k clusters C:={c1, ..., ck}

C := {{d1}}; /* random choice for the first cluster */
for i:=2 to n do
  determine cluster cj ∈ C with the largest value of
  sim(di, cj) (e.g. sim(di, $\vec{c}_j$) with centroid $\vec{c}_j$);
   if sim(di, cj) ≥ threshold
   then assign di to cluster cj
   else if |C| < k
        then C := C ∪ {{di}}; /* create new cluster */
        else assign di to cluster cj
        fi
   fi
od

Idea:

- determine **k prototype vectors**, one for each cluster
- **assign each data record to the most similar prototype vector**
  and compute new prototype vector
  (e.g. by averaging over the vectors assigned to a prototype)
- **iterate** until clusters are sufficiently stable

randomly choose k prototype vectors $\vec{c}_1,...,\vec{c}_k$
while not yet sufficiently stable do
  for i:=1 to n do
    assign di to cluster cj for which $sim(\vec{d}_i,\vec{c}_j)$ is minimal
  od;
  for j:=1 to k do $\vec{c}_j := \frac{1}{|c_j|}\sum_{\vec{d}\in c_j}\vec{d}$ od;
od;

Example for K-Means Clustering