Chapter 6
Classification & Clustering

Sergej Sizov
Information Retrieval
Winter semester 2006/2007

Classification Problem (Categorization)
given:
- feature vectors
  \( f_1 \), \( f_2 \)
- determine class/topic membership(s) of feature vectors

\( \Omega^* \)
unknown classes: unsupervised learning (clustering)

\( \Omega^\dagger \)
known classes + labeled training data: supervised learning

Classification with Training Data: Overview

Science
Mathematics
Algebra
Probabilities
and Statistics
Hypotheses
Testing
Large
Deviation
intelligent
assignment

\( \{ c_k \}^m \)
feature space: term frequencies \( f_i \) \((i = 1, \ldots, m)\)

\( \mathcal{G} \)
new documents

\( \mathcal{G} \)
automatische Zuweisung

training data

estimates
and assign document to the class
with the highest probability

e.g. with Bayesian method:

\[ P(d \in c_j | f) = \frac{P(f | d \in c_j) P(d \in c_j)}{P(f)} \]

Assessment of Classification Quality

For binary classification with regard to class \( C \):

\[ a = \# \text{docs that are classified into } C \text{ and do belong to } C \]
\[ b = \# \text{docs that are classified into } C \text{ but do not belong to } C \]
\[ c = \# \text{docs that are not classified into } C \text{ but do belong to } C \]
\[ d = \# \text{docs that are not classified into } C \text{ and do not belong to } C \]

Accuracy (Genauigkeit) = \[ \frac{a + d}{a + b + c + d} \]

Precision (Präzision) = \[ \frac{a}{a + b} \]

Recall (Ausbeute) = \[ \frac{a}{a + c} \]

\( F_1 \) (harmonic mean of precision and recall) = \[ \left( \frac{1}{\text{precision}} + \frac{1}{\text{recall}} \right)^{-1} \]

For manyway classification with regard to classes \( C_1, \ldots, C_k \):

- macro average over \( k \) classes or
- micro average over \( k \) classes

Estimation of Classifier Quality

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

Distance-based Classifiers: k-Nearest-Neighbor Method (kNN)

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 \( d \)

Step 2:
Assign \( d \) to class \( C_j \) for which the function value

\[ f(d, C_j) = \sum_{\hat{\varphi} \in \text{NNN}(d)} \text{sim}(d, \hat{\varphi}) \]

is maximized

With binary classification assign \( d \) to class \( C \) if

\[ f(d, C_j) \text{ is above some threshold } \delta \text{ (e.g. } \delta > 0.5) \]
Distance-based Classifiers: Rocchio Method

Step 1: Represent the training documents for class Cj by a prototype vector with tf*idf-based vector components

\[ \overline{c}_j = \alpha \sum_{d \in C_j} \frac{d}{\|d\|} - \beta \sum_{d \notin C_j} \frac{d}{\|d - C_j\|} \]

with appropriate coefficients \( \alpha \) and \( \beta \) (e.g. \( \alpha = 16, \beta = 4 \)).

Step 2: Assign a new document \( d \) to the class Cj for which the cosine similarity \( \text{cos}(\overline{c}_j, d) \) is maximized.

Feature Selection

For efficiency of the classifier and to suppress noise, choose a 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

Feature Selection Based on Information Gain

Information gain:

For discriminating classes \( c_1, ..., c_k \) select the binary features \( X_i \) (term occurrence) with the largest gain in entropy

\[
G(X_i) = \sum_{j=1}^{k} P(X_i|c_j) \log_2 \frac{1}{P(X_i|c_j)}
- P(X_i) \sum_{j=1}^{k} P(X_i|c_j) \log_2 \frac{1}{P(X_i|c_j)}
- P(X_i) \sum_{j=1}^{k} P(X_i|c_j) \log_2 \frac{1}{P(X_i|c_j)}
\]

and for discriminating classes \( c_1, ..., c_k \):

\[
MI(X_i, c_j) = \sum_{X \in \{X_i, X_j\}} P(X \land c_j) \log_2 \frac{P(X \land c_j)}{P(X \land c_j)}
\]

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

Feature Selection Based on Mutual Information

Mutual information (Kullback-Leibler distance, relative entropy):

For class \( c_j \) select those binary features \( X_i \) (term occurrence) with the largest value of

\[
MI(X_i, c_j) = \sum_{X \in \{X_i, X_j\}} P(X \land c_j) \log_2 \frac{P(X \land c_j)}{P(X \land c_j)}
\]

and for discriminating classes \( c_1, ..., c_k \):

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

can be computed in time \( O(n) + O(mk) \) for \( n \) training documents, \( m \) terms, and \( k \) classes.
Distribution-based Classifiers: Naive Bayes with Binary Features \( X_i \)
estimate:
\[
P \left( d \in c_k \mid d \right) = \frac{P \left( d \right) \times P \left( d \in c_k \mid d \right)}{P \left( d \right)} = \frac{1}{\sum_{k} n_k} \prod_{i=1}^{m} \frac{P \left( X_i \mid d \in c_k \right)}{P \left( X_i \right)}
\]
where:
- \( n_k \): number of training data records in \( k \)
- \( n_k,j \): number of training data records in \( k \) that belong to class \( j \)
- \( p_1 = n_{k1} / n_k \) and \( p_{2} = n_{k2} / n_k \)
- \( H(k) = -\sum_{j=1}^{n} \frac{n_k,j}{n_k} \log \left( \frac{n_k,j}{n_k} \right) \)

Discriminative Classifiers: Decision Trees
given:
a multiset of \( m \)-dimensional training data records
\[
\subseteq \text{dom}(A_1) \times \ldots \times \text{dom}(A_m)
\]
with
- numerical, ordinal, or categorial attributes \( A_i \)
e.g. term occurrence frequencies \( n_1 \times \ldots \times n_m \)
and with class labels
wanted:
a tree with
- attribute value conditions of the form
  - \( A_i \leq \text{value} \) for numerical or ordinal attributes or
  - \( A_i \in \text{value set} \) for categorial attributes
- linear combinations of this type
  \[
  \sum_{i=1}^{m} a_i A_i \leq \text{value}
  \]
as inner nodes and
- labeled classes as leaf nodes

Examples for Decision Trees (1)

- has read Tolkien
- has read Eco
- intellectual

Top-Down Construction of Decision Tree
Input: decision tree node \( k \) that represents one partition \( D \) of dom(A1) \( \times \ldots \times \text{dom}(A_m) \)
Output: decision tree with root \( k \)
1) BuildTree (root, dom(A1) \( \times \ldots \times \text{dom}(A_m) \))
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 \( A_i \);
  - determine split value \( x \) for most suitable partitioning of \( D \) into \( D_1 = D \cap \{ d \mid A_i \leq x \} \) and \( D_2 = D \cap \{ d \mid A_i > x \} \);
  - create children \( k_1 \) and \( k_2 \) of \( k \);
  - BuildTree (\( k_1 \), \( D_1 \)) and BuildTree (\( k_2 \), \( D_2 \))

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)

\[
G(k, k_1, k_2) = H(k) - (p1 \times H(k_1) + p2 \times H(k_2))
\]
where:
- \( n_k \): # training data records in \( k \)
- \( n_{k,j} \): # training data records in \( k \) that belong to class \( j \)
- \( p_1 = n_{k1} / n_k \) and \( p_{2} = n_{k2} / n_k \)
- \( H(k) = -\sum_{j=1}^{n} \frac{n_k,j}{n_k} \log \left( \frac{n_k,j}{n_k} \right) \)

Example for Decision Tree for Text Classification

\[
G = \text{H}(k) - (f7 > 1 \times 2/6 \times \text{H}(k_1) + 4/6 \times \text{H}(k_2))
\]

\[
\begin{align*}
\text{Algebra} & : f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8 \\
C1: \text{Algebra} & : d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8 \\
C2: \text{Stochastics} & : d_5, d_6, d_7, d_8 \\
\end{align*}
\]
**Discriminative Classifiers: Support Vector Machines (SVM), Binary Classification**

![Diagram of hyperplane](image)

**Computation of the Optimal Hyperplane**

Find \( \hat{w} \in \mathbb{R}^d \) and \( b \in \mathbb{R} \) such that:

1. \( \hat{\delta} \in \mathbb{R} \) is maximal and
2. \( C \frac{1}{n} \sum_{i=1}^{n} (\hat{w} \cdot x_i) + b \) for all \( i=1, \ldots, n \)

This is (w.l.o.g. with the choice \( \lambda = 1/\hat{\delta} \)) equivalent to (V. Vapnik: Statistical Learning Theory, 1998):

Find \( \alpha_1, \ldots, \alpha_n \in \mathbb{R} \) such that:

1. \( \sum_{i=1}^{n} \alpha_i = 1 \), \( \sum_{i=1}^{n} \alpha_i C \cdot C_i = 0 \) and \( \sum_{i=1}^{n} \alpha_i = 0 \)
2. \( \hat{w} = \sum_{i=1}^{n} \alpha_i C_i \cdot x_i \)

Optimal vector \( \hat{w} \) is a linear combination of support vectors:

\[
\hat{w} = \sum_{i=1}^{n} \alpha_i C_i \cdot x_i
\]

**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 and soft-margin parameter \( \lambda \): difficult and highly dependent on data and application: high \( \lambda \) minimizes training error, but leads to poor generalization (smaller separation, thus higher risk)

**Classifiers with Semisupervised Learning**

**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 “automobile” 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:**
1. train classifier with \( D^k \) as training data
2. classify docs in \( D^u \)
3. repeat 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 $D^k$
(e.g., text terms and anchor terms)
$D^A := D^k_A$; $D^B := D^k_B$
repeat
classify docs in $D^A$ by A and $D^B$ by B
select the best classified docs from $D^A$ and $D^B$; $d_A$ and $d_B$
add $d_A$ to training set $D^A_B$, add $d_B$ to training set $D^B_A$
retrain A using $D^A_B$, retrain B using $D^B_A$
until results are sufficiently stable
assign docs from $D^A$ to classes on which A and B agree

More Meta Strategies

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**: $C_k(d_j) = 1$ if $\sum_{\nu=1}^{m} C\nu_k(d_j) = m$
- **weighted average**: $C_k(d_j) = 1$ if $\sum_{\nu=1}^{m} P\nu_k C\nu_k(d_j) \geq \tau$
  with precision estimator $P\nu_k$
  for classifier $\nu$

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

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? (implicitly or explicitly)

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)

A huge amount of data?

With enough data the choice of classifier may not matter much, and the best choice may be unclear
- Data: Brill and Banko on context-sensitive spelling correction
- But the fact that you have to keep doubling your data to improve performance is a bit unpleasant

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

Hierarchical vs. Flat Clustering

<table>
<thead>
<tr>
<th>Hierarchical Clustering:</th>
<th>Flat Clustering:</th>
</tr>
</thead>
<tbody>
<tr>
<td>detailed and insightful</td>
<td>detailed and insightful</td>
</tr>
<tr>
<td>hierarchy built</td>
<td>hierarchy built</td>
</tr>
<tr>
<td>in natural manner</td>
<td>in natural manner</td>
</tr>
<tr>
<td>from fairly simple algorithms</td>
<td>from fairly simple algorithms</td>
</tr>
<tr>
<td>relatively expensive</td>
<td>relatively expensive</td>
</tr>
<tr>
<td>no prevalent algorithm</td>
<td>no prevalent algorithm</td>
</tr>
</tbody>
</table>

Clustering: Classification based on Unsupervised Learning

given:
a m-dimensional data records $d_j \in D \subseteq \text{dom}(A_1) \times \ldots \times \text{dom}(A_m)$
with attributes $A_i$ (e.g. term frequency vectors $N_0 \times \ldots \times N_0$) or n data points with pair-wise distances (similarities) in a metric space

wanted:
k clusters $c_1, \ldots, c_k$ and an assignment $D \rightarrow \{c_1, \ldots, c_k\}$ such that the average intra-cluster similarity

$\sum_{k \in C} \frac{1}{|k|} \sum_{d \in k} \sum \text{sim}(d, \bar{c}_k)$

is high and the average inter-cluster similarity

$\frac{1}{k(k-1)} \sum e \neq f \sum \text{sim}(\bar{c}_e, \bar{c}_f)$

is low,

where the centroid $\bar{c}_k$ of $c_k$ is

$\bar{c}_k = \frac{1}{|k|} \sum \bar{d}_{c_k}$

Hierarchical Clustering: Agglomerative Bottom-up Clustering (HAC)

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 $i=1 \rightarrow n$ do $c_i := \{d_i\}$ od;

$C := \{c_i, \ldots, c_j\}^* \text{ set of clusters }$

while $|C| > 1$ do

determine $c_i, c_j \in C$ with maximal inter-cluster similarity;

$C := C \rightarrow \{c_i, \ldots, c_j\}$

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.

$c_1 := \{d_1, \ldots, d_n\}$; $C := \{c_1\}$; */ 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)

---

Alternative Similarity Metrics for Clusters

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

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

For hierarchical clustering the following axiom must hold:

$max \{\text{sim}(c, c'), \text{sim}(c, c'')\} \geq \text{sim}(c, c' \cup c'')$ for all $c, c', c'' \in 2^D$

---

Cluster Quality Measures (1)

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** $\max_{\Gamma(d_i)} |\{d \in c_j | L(d) = L_j\}| / |c_j|$ Complete clustering has purity $\sum_{j=1}^{k} \text{purity}(c_j) / k$

Alternatives:
- **Entropy**: within cluster $\sum_{c_j} |c_j \cap L_j| \log \frac{|c_j \cap L_j|}{|c_j|}$
- **MI** between cluster and classes $\sum_{c_j \subseteq L} \frac{|c_j \cap L|}{n} \log \frac{|c_j \cap L|}{|c_j|}$

---

Cluster Quality Measures (2)

Without any ground truth:

ratio of intra-cluster to inter-cluster similarities

$\frac{1}{k} \sum_{j=1}^{k} \left( \frac{1}{|c_j|} \sum_{d \in c_j} \text{sim}(d, \bar{c}_j) \right) \leq \left( \frac{1}{k(k-1)} \sum_{j \neq j'} \text{sim}(\bar{c}_j, \bar{c}_{j'}) \right)$

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

---

Flat Clustering: Simple Single-Pass Method

given: data records $d_1, \ldots, d_n$

wanted: (up to) $k$ clusters $C := \{c_1, \ldots, c_k\}$

$C := \{d_1\}$; */ random choice for the first cluster */

for $i := 2$ to $n$ do

- determine cluster $c_j \in C$ with the largest value of $\text{sim}(d_i, c_j)$ (e.g. $\text{sim}(d_i, \bar{c}_j)$ with centroid $\bar{c}_j$);
- if $\text{sim}(d_i, \bar{c}_j) \geq \text{threshold}$ then assign $d_i$ to cluster $c_j$
- else if $|C| < k$

- then $C := C \cup \{d_i\}$; /* create new cluster */
- else assign $d_i$ to cluster $c_j$
- fi

od

---

K-Means Method for Flat Clustering (1)

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 $\bar{c}_1, \ldots, \bar{c}_k$

while not yet sufficiently stable do

for $i := 1$ to $n$ do

- assign $d_i$ to cluster $c_j$ for which $\text{sim}(d_i, \bar{c}_j)$ is minimal
- od;

for $j := 1$ to $k$ do

- $\bar{c}_j := \frac{1}{|d_j|} \sum_{d \in d_j} d$
- od;

od;
Example for K-Means Clustering

K=2

K-Means Method for Flat Clustering (2)

- run-time is $O(n)$ (assuming constant number of iterations)
- a suitable number of clusters, $K$, can be determined experimentally or based on the MDL principle
- the initial prototype vectors could be chosen by using another – very efficient – clustering method (e.g., bottom-up clustering on random sample of the data records).
- for sim any arbitrary metric can be used