Chapter 9

Classification
and
Clustering

Classification and Clustering

- Classification and clustering are classical *pattern recognition* and *machine learning* problems
- Classification, also referred to as categorization
  - Asks "what class does this item belong to?"
  - *Supervised learning* task (automatically applies *labels* to data)
- Clustering
  - Asks "how can I group this set of items?"
  - *Unsupervised learning* task (*grouping* related items together)
- Items can be documents, emails, queries, entities & images
- Useful for a wide variety of search engine tasks

Classification

- Classification is the task of automatically applying labels to items
- Useful for many search-related tasks
  - Spam detection
  - Sentiment classification
  - Online advertising
- Two common approaches
  - Probabilistic
  - Geometric

How to Classify?

- How do humans classify items?
- For example, suppose you had to classify the healthiness of a food
  - *Identify* set of features indicative of health: fat, cholesterol, sugar, sodium, etc.
  - *Extract* features from foods
    - Read nutritional facts, chemical analysis, etc.
  - *Combine evidence* from the features into a hypothesis
    - Add health features together to get "healthiness factor"
  - Finally, *classify* the item based on the evidence
    - If "healthiness factor" is above a certain value, then deem it *healthy*

Ontologies

- Ontology is a labeling or *categorization scheme*
- Examples
  - Binary (spam, not spam)
  - Multi-valued (red, green, blue)
  - Hierarchical (news/local/sports)
- Different classification tasks require different ontologies

Naïve Bayes Classifier

- Probabilistic classifier based on Bayes' rule:

$$P(C|D) = \frac{P(D|C)P(C)}{P(D)}$$

$$= \frac{P(D|C)P(C)}{\sum_{c \in \mathcal{C}} P(D|C=c)P(C=c)}$$

  - *C* (*D*) is a *random variable* corresponding to the class (input)

- Based on the term independence assumption, the Naïve Bayes' rule yields:

$$P(c \mid d) = \frac{P(d \mid c)\, P(c)}{\sum_{c \in C} P(d \mid c)\, P(c)}$$

$$= \frac{\prod_{i=1}^{n} P(w_i \mid c)\, P(c)}{\sum_{c \in C} \prod_{i=1}^{n} P(w_i \mid c)\, P(c)}$$ } (Chain rule)

Probability 101: Random Variables

- Random variables are *non-deterministic*, *probability* based
  - Can be *discrete* (finite number of outcomes) or *continues*
  - Model *uncertainty* in a variable
- $P(X = x)$ means "the probability that random variable X (i.e., its value is not *fixed*) takes on value x"
- Examples.
  - Let X be the outcome of a coin toss, $P(X = \text{heads}) = P(X = \text{tails}) = 0.5$
  - $Y = 5 - 2X$. If X is *random*/*deterministic*, then Y is *random*/*deterministic*
    - *Deterministic* variables mean not random, in the context of others

Naïve Bayes Classifier

- Documents are *classified* according to

$$\begin{aligned} \text{Class}(d) &= \arg\max_{c \in \mathcal{C}} P(c|d) \\ &= \arg\max_{c \in \mathcal{C}} \frac{P(d|c)P(c)}{\sum_{c \in C} P(d|c)P(c)} \end{aligned}$$

- Must estimate *P*(*d* | *c*) and *P*(*c*)
  - *P*(*c*) is the probability of observing class *c*
  - *P*(*d* | *c*) is the probability that document *d* is observed given the class is known to be *c*

Estimating *P*(*c*)

- *P*(*c*) is the probability of observing class *c*
- Estimated as the *proportion* of training documents in class *c*:

$$P(c) = \frac{N_c}{N}$$

- N_c is the number of *training documents* in class *c*
- *N* is the total number of *training documents*

Estimating P(*d* | *c*)

- P(*d* | *c*) is the probability that *document d* is observed given the *class* is known to be *c*
- Estimate depends on the *event space* used to represent the documents
- What is an event space?
  - The set of all possible outcomes for a given random variable
  - e.g., for a coin toss random variable the event space is *S* = { heads, tails }
- The probability of an event space *S*
  - A *probability* is assigned to each event/outcome in *S*
  - The *sum* of the *probabilities* over all the events in *S* must equal to one

Multiple Bernoulli Event Space

- Documents are represented as *binary vectors*
  - One entry for every *word* in the vocabulary
  - Entry *i* = 1, if word *i* occurs in the document; 0, otherwise
- Multiple Bernoulli distribution is a natural way to model distributions over binary vectors
- Same event space as used in the classical *probabilistic retrieval model*

Multiple Bernoulli Document Representation

Example.

document *id*	cheap	buy	banking	dinner	the	*class*
1	0	0	0	0	1	not spam
2	1	0	1	0	1	spam
3	0	0	0	0	1	not spam
4	1	0	1	0	1	spam
5	1	1	0	0	1	spam
6	0	0	1	0	1	not spam
7	0	1	1	0	1	not spam
8	0	0	0	0	1	not spam
9	0	0	0	0	1	not spam
10	1	1	0	1	1	not spam

Multiple-Bernoulli: Estimating P(*d* | *c*)

- *P*(*d* | *c*) is computed (in the Multiple-Bernoulli model) as

$$P(d|c) = \prod_{w \in \mathcal{V}} P(w|c)^{\delta(w,d)} \left(1 - P(w|c)\right)^{1-\delta(w,d)}$$

where $\delta(w, d) = 1$ iff term *w* occurs in *d*; *P*(*d* | *c*) = 0 if $\exists w \in d$ never occurred in *c* in the training set, the "data sparseness" problem, which can be solved by the "smoothing" methods.

- Laplacian smoothed estimate:

$$P(w|c) = \frac{df_{w,c} + 1}{N_c + 1}$$

where $df_{w,c}$ denotes the number of documents in *c* including term *w*
N_c is the number of documents belonged to class *c*

- Collection smoothed estimate:

$$P(w|c) = \frac{df_{w,c} + \mu \frac{N_w}{N}}{N_c + \mu}$$

where μ is a *tunable parameter* and N_w is the no. of doc. including *w*

Multinomial Event Space

- Documents are represented as vectors of term frequencies
  - One entry for every word in the vocabulary
  - Entry *i* = number of times that term *i* occurs in the document
- Multinomial distribution is a natural way to model distributions over *frequency* vectors
- Same event space as used in the language modeling retrieval model

Multinomial Document Representation

Example.

document *id*	cheap	buy	banking	dinner	the	*class*
1	0	0	0	0	2	not spam
2	3	0	1	0	1	spam
3	0	0	0	0	1	not spam
4	2	0	3	0	2	spam
5	5	2	0	0	1	spam
6	0	0	1	0	1	not spam
7	0	1	1	0	1	not spam
8	0	0	0	0	1	not spam
9	0	0	0	0	1	not spam
10	1	1	0	1	2	not spam

Multinomial: Estimating *P*(*d* | *c*)

- P(*d* | *c*) is computed as:

Probability of generating a document of length |*d*| (= number of terms in *d*)

$$P(d|c) = P(|d|)\,(tf_{w_1,d}, tf_{w_2,d}, \ldots, tf_{w_{\mathcal{V}},d})!\prod_{w\in\mathcal{V}} P(w|c)^{tf_{w,d}}$$

$$\propto \prod_{w\in\mathcal{V}} P(w|c)^{tf_{w,d}}$$

Multinomial Coefficient = $tf\,!\,/\,(tf_{w_1,d}!\times\ldots\times tf_{w_V,d}!)$

Document Dependent

- Laplacian smoothed estimate:

$$P(w|c) = \frac{tf_{w,c}+1}{|c|+|\mathcal{V}|}$$

where |*c*| is the number of *terms* in the *training documents* of class *c*

|*V*| is the number of *distinct terms* in the *training documents*

Number of Terms *w* in Class *c*

- Collection smoothed estimate:

$$P(w|c) = \frac{tf_{w,c}+\mu\frac{cf_w}{|C|}}{|c|+\mu}$$

Number of term *w* in a training set *C*

Number of terms in all training documents

Tunable parameter

Multinomial Versus Multiple-Bernoulli Model

- The Multinomial model is consistently outperform the *Multiple-Bernoulli* model
- Implementing both models is relatively *straightforward*
- Both classifiers are
  - *efficient*, since their statistical data can be stored in memory
  - *accurate* in document classification
  - *popular* and *attractive* choice as a general-purpose classifier

Support Vector Machines (SVM)

- Based on geometric principles
- Given a set of inputs labeled '+' & '-', find the best hyper-plane in an *N*-dimensional space that separates the '+'s and '-'s, i.e., a *binary-categorization* method
- Documents are represented as *N*-dimensional vectors with (non-)binary feature weights
- Questions
  - How is "best" defined?
  - What if no hyperplane exists such that the '+'s and '-'s can be perfectly separated?

"Best" Hyperplane?

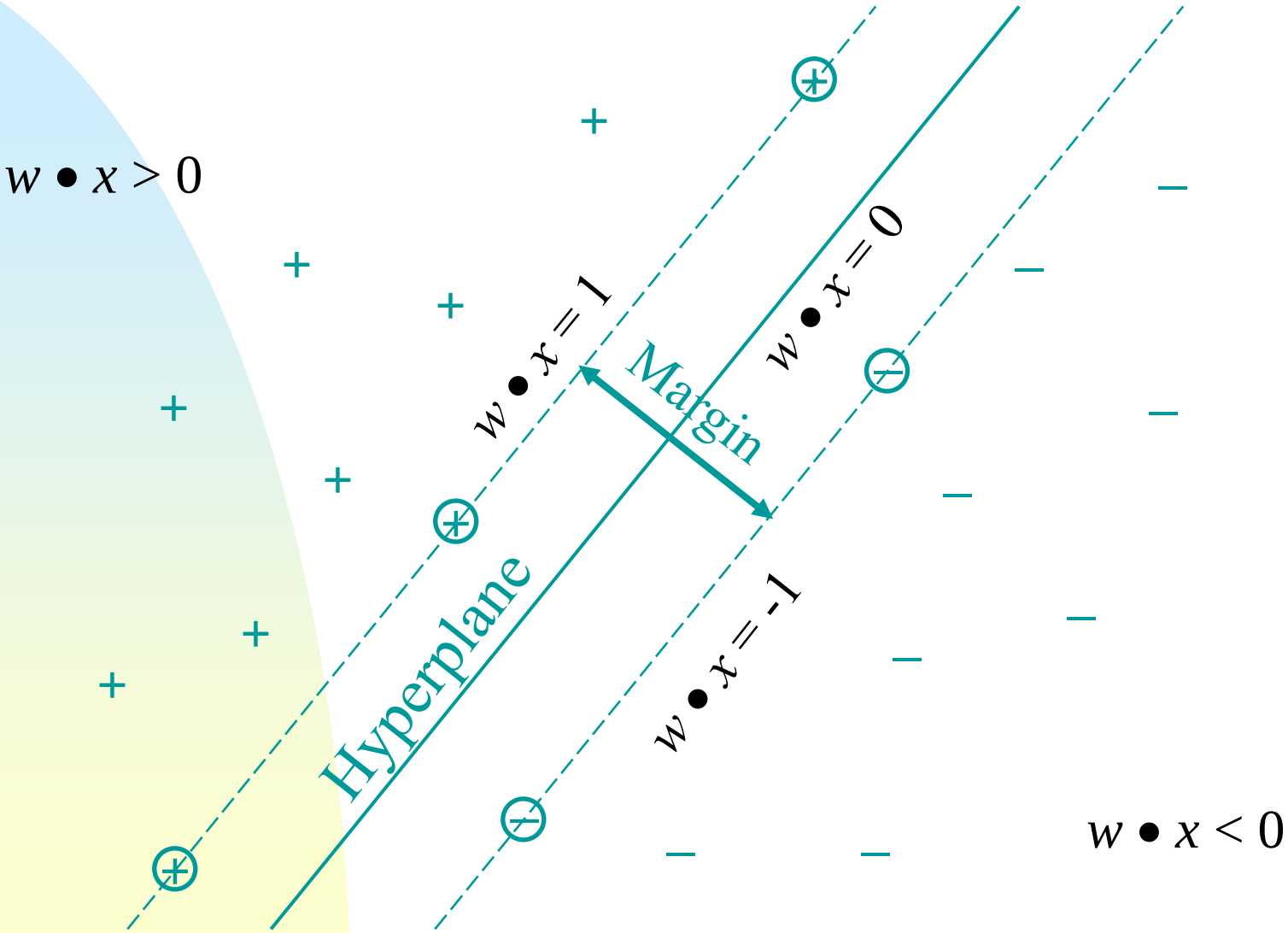
- First, what is a *hyperplane*?
  - A generalization of a line to higher dimensions
  - Defined by vector *w* that is *learned* from the training data
  - Avoiding the *overfitting* problem, i.e., working well with the training data, but fails at classifying the test data
- To avoid overfitting, SVM chooses a hyperplane with the maximum margin that separates '+'s & '-'s
- If x^+ & x^- are the *closest* '+' & '-' inputs to the hyper-plane, called *support vectors*, then the margin is:

$$\text{Margin}(w) = \frac{|w \cdot x^-| + |w \cdot x^+|}{||w||}$$

Sum of the distances w & x^- and w & x^+

Length of w

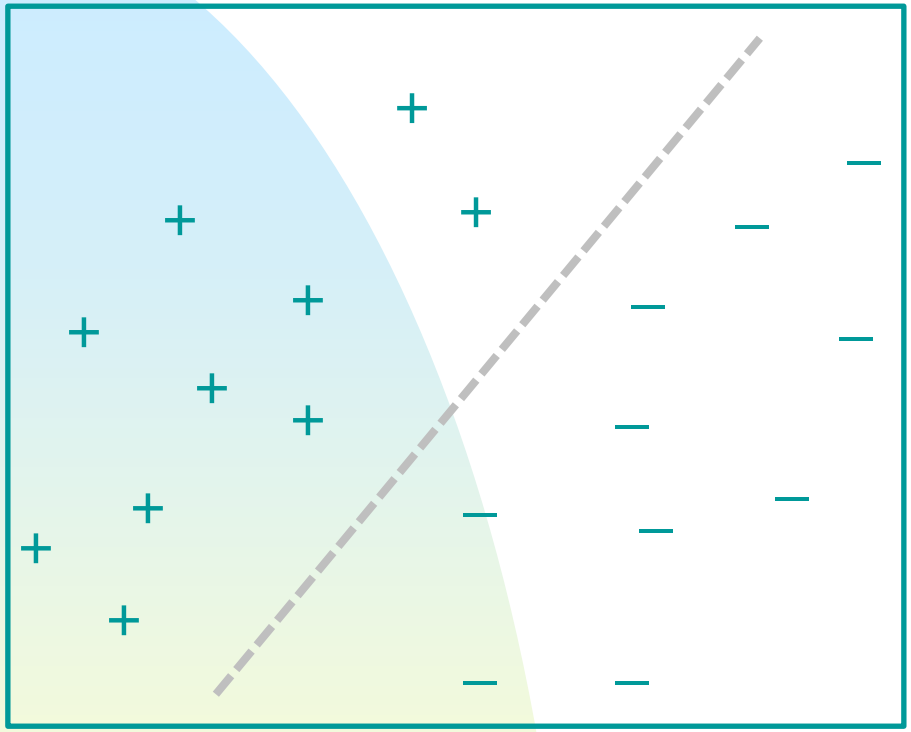
Support Vector Machines



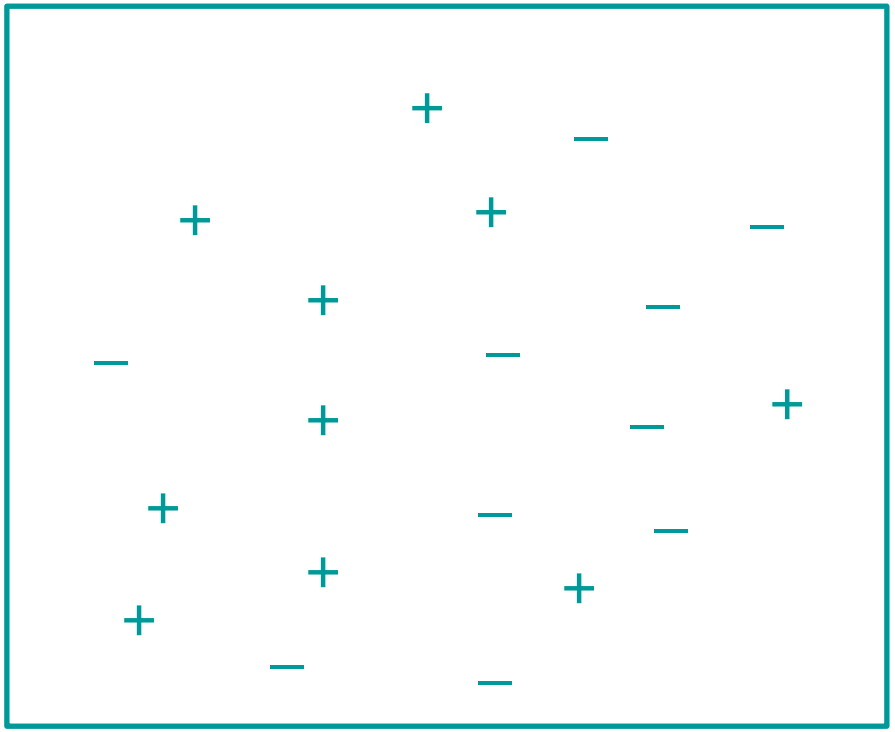
“Best” Hyperplane?

- It is typically assumed that $|w \bullet x^{-}| = |w \bullet x^{+}| = 1$, which does not change the solution to the problem
- Thus, to find the hyperplane with the *largest* margin, we must **maximize** $2 / || w ||$

Separable vs. Non-Separable Data



Separable



Non-Separable

Linear Separable Case

- In math: $minimize: \quad \frac{1}{2}||w||^2$

  $subject\ to:$

  $$w \cdot x_i \geq 1 \quad \forall i \text{ s.t. } \text{Class}(i) = +$$
  $$w \cdot x_i \leq -1 \quad \forall i \text{ s.t. } \text{Class}(i) = -$$

- In English:
  - Find the *largest margin* hyperplane that separates the '+'s and '-'s
  - Can be solved using *quadratic programming*
- An unseen document *d* can be classified using

$$\text{Class}(d) = \begin{cases} + & \text{if } w \bullet x_d > 0 \\ - & \text{otherwise} \end{cases}$$

Linearly Non-Separable Case

- In math:

$$minimize: \quad \tfrac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i$$

(Penalty factor)

often set to 1 (controls how much to penalize)

$$subject\ to:$$

$$w \cdot x_i \geq 1 - \xi_i \qquad \forall i \text{ s.t. } \mathrm{Class}(i) = +$$

$$w \cdot x_i \leq -1 + \xi_i \qquad \forall i \text{ s.t. } \mathrm{Class}(i) = -$$

$$\xi_i \geq 0 \qquad \forall i$$

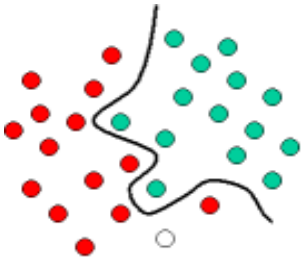
- In English:
  - ξ_i, *slack variable*, which allows the target values to be violated, denotes how misclassified instance *i* is
  - ξ_i s= 0, if no violation occurs, i.e., a *linear separable* case
  - Find a hyperplane with a large *margin* and lowest *mis-classification cost*

The Kernel Trick

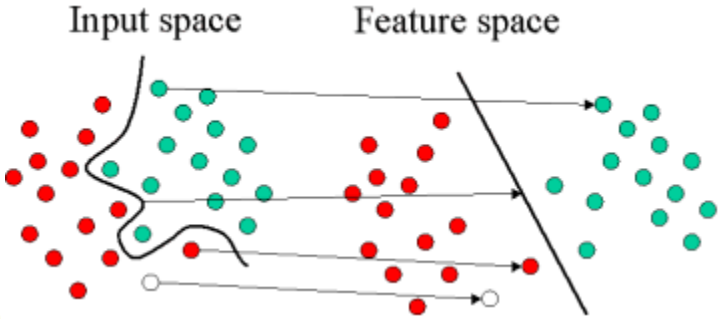
- Linearly, *non-separable* data may become linearly *separable* if transformed, or mapped, to a *higher* dimension space
- Computing vector math (i.e., dot products) in very high dimensional space is costly due to *computational time & space requirements*
- The kernel (function) trick allows very high dimensional dot products to be computed efficiently
- Allows inputs to be *implicitly mapped* to high (possibly infinite) dimensional space with little computational overhead

The Kernel Functions

- To fully separate the (green & red) objects below would require a *curve*, which is more complex than a *line*



- Mapping, i.e., rearranging, the original objects (on the left) using a set of mathematical functions, known as kernel functions, to the right



- The mapped objects are *linearly separable*, instead of constructing a complex curve

Kernel Trick Example

- The following function maps 2-vectors to 3-vectors, where $w = \langle w_1, w_2 \rangle$ and $x = \langle x_1, x_2 \rangle$:

$$\Phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

$\Phi(x)$s may be linearly separated

- Standard way to compute $\Phi(w) \bullet \Phi(x)$ is to map the inputs & compute the dot product in the higher dimensional space

- However, the dot product can be done entirely in the original 2-dimensional space:

$$\begin{aligned} \Phi(w) \cdot \Phi(x) &= w_1^2x_1^2 + 2w_1w_2x_1x_2 + w_2^2x_2^2 \\ &= (w \cdot x)^2 \end{aligned}$$

Common Kernels

- The previous example is known as the *polynomial kernel* (with $p = 2$)
- Most common kernels are linear, polynomial & Gaussian
- Each kernel performs a dot product in a higher implicit dimensional space

Kernel Type	Value	Implicit Dimension
Linear	$K(x_1, x_2) = x_1 \cdot x_2$	N
Polynomial	$K(x_1, x_2) = (x_1 \cdot x_2)^p$	$\begin{pmatrix} N+p-1 \\ N \end{pmatrix}$
Gaussian	$K(x_1, x_2) = \exp -\|\|x_1 - x_2\|\|^2/2\sigma^2$	Infinite

Non-Binary Classification with SVMs

- One versus all (OVA)
  - Train "class *c* vs. not class *c*" SVM for every class
  - If there are *K* classes, must train *K classifiers*
  - Classify items according to Class(x) = Arg Max$_c$ $(W_c \bullet x)$
- One versus one (OVO)
  - Train a binary classifier for each pair of classes, e.g., "bad", "fair", "excellent"; the class w/ most votes is chosen
  - Must train $K \times (K - 1) / 2$ classifiers
  - Computationally expensive for large values of *K*

SVM Tools

- Solving SVM optimization problem is not straightforward
- Many good software packages exist
  - SVM-Light
  - LIBSVM
  - R library
  - Matlab SVM Toolbox

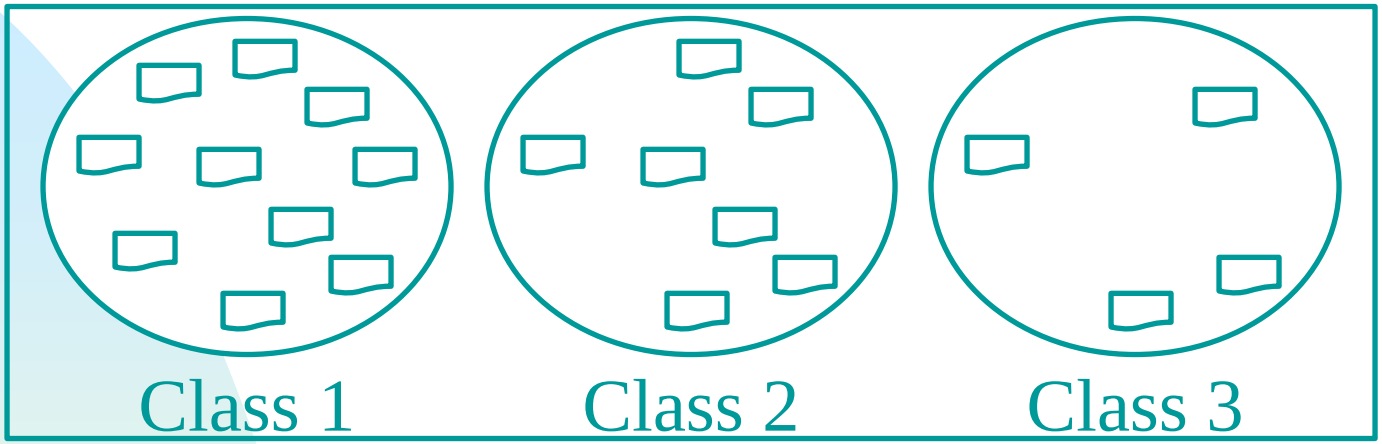
Evaluating Classifiers

- Common classification metrics
  - Accuracy (precision at rank 1)
  - Precision
  - Recall
  - F-measure
  - ROC curve analysis
- Differences from IR metrics
  - "Relevant" replaced with "classified correctly"
  - *Micro-averaging* more commonly used in classification
    - Computes a *metric* for every test instance (document) and then averages over all the instances, whereas
    - Macro-averaging computes the average of the per-class metrics

Classes of Classifiers

- Types of classifiers
  - Generative (e.g., *Naïve-Bayes*): assumes some under-lying *probability distribution* on generating (writing) both *documents* and *classes* using $P(c)$ and $P(d \mid c)$
  - Non-parametric (e.g., *Nearest Neighbor*)
- Types of learning
  - Supervised (e.g., *Naïve-Bayes*, *SVMs*)
  - Semi-supervised (e.g., *Rocchio*, *Relevance models*)
  - Unsupervised (e.g., *Clustering*)

Naïve Bayes Generative Process



Generate class according to $P(c)$

Class 2

Generate document according to $P(d \mid c)$

Feature Selection for Text Classification

- Document classifiers can have a very large number of features, such as indexed terms
  - *Not* all features are useful
  - Excessive features can increase computational *cost* of *training* and *testing*
- *Feature selection* methods *reduce* the number of features by choosing the most useful features
  - which can significantly *improve efficiency* (in terms of storage and processing time) while *not hurting* the *effectiveness* much (in addition to *eliminating noisy*)

Feature Selection for Text Classification

- Feature selection is based on entropy/information gain
  - The law of large numbers indicates that the symbol a_j will, on the average, be selected

$$n \times p(a_j)$$

times in a total of n selections

  - The average amount of information obtained from n source outputs for each a_j ($j \geq 1$) with $-\log_2 p(a_j)$ bits is

$$n \times p(a_1) \log_2 p(a_1)^{-1} + \ldots + n \times p(a_j) \log_2 p(a_j)^{-1}$$

bits. Divided by n obtain the *average* amount of information per source output symbol, which is known as *uncertainty*, or the *entropy*, $E(p)$:

$$E = -\sum_{i=1}^{\sigma} p_i \log_2 p_i$$

Information Gain

- Information gain is a commonly used feature selection measure based on information theory
  - It tells how much "information" is gained (about the class labels) if we observe some feature
  - Entropy is the expected information contained in $P(c)$
- Rank features by *information gain* and then train model using the top K (K is typically small)
- The information gain for a *Multiple-Bernoulli Naïve Bayes classifier* is computed as:

$$\begin{aligned} IG(w) &= H(C) - H(C|w) \\ &= -\sum_{c \in \mathcal{C}} P(c) \log P(c) + \sum_{w \in \{0,1\}} P(w) \sum_{c \in \mathcal{C}} P(c|w) \log P(c|w) \end{aligned}$$

Entropy of $P(c)$

Conditional Entropy

Information Gain

- Example. The information gain for the term "cheap", using $IG(w) = -\sum_{c \in C} P(c) \log P(c) + \sum_{w \in \{0,1\}} \sum_{c \in C} P(c|w) \log P(c|w)$

$$
\begin{aligned}
IG(cheap) = & -P(spam)\log P(spam) - P(\overline{spam})\log P(\overline{spam}) + \\
& P(cheap)P(spam|cheap)\log P(spam|cheap) + \\
& P(cheap)P(\overline{spam}|cheap)\log P(\overline{spam}|cheap) + \\
& P(\overline{cheap})P(spam|\overline{cheap})\log P(spam|\overline{cheap}) + \\
& P(\overline{cheap})P(\overline{spam}|\overline{cheap})\log P(\overline{spam}|\overline{cheap}) \\
= & -\frac{3}{10}\log\frac{3}{10} - \frac{7}{10}\log\frac{7}{10} + \frac{4}{10}\cdot\frac{3}{4}\log\frac{3}{4} \\
& + \frac{4}{10}\cdot\frac{1}{4}\log\frac{1}{4} + \frac{6}{10}\cdot\frac{0}{6}\log\frac{0}{6} + \frac{6}{10}\cdot\frac{6}{6}\log\frac{6}{6} \\
= & \ 0.2749
\end{aligned}
$$

where $P(\overline{cheap})$ denotes $P(cheap = 0)$,

$P(\overline{spam})$ denotes $P(not\ spam)$,

0 log 0 = 0, and

$IG(buy) = 0.0008$, $IG(banking) = 0.04$, $IG(dinner) = 0.36$, $IG(the) = 0$

Classification Applications

- Classification is widely used to enhance search engines
- Example applications
  - Spam detection
  - Sentiment classification
  - Semantic classification of advertisements, based on the semantically-related, not topically-related scope, e.g., “tropical fish”
    - Semantically-related: pet stores, aquariums, scuba diving
    - Not topically-related: fishing, fish restaurants, mercury poison

Spam, Spam, Spam

- Classification--widely used to detect various types of spam
- There are many types of spam
  - Link spam (to increase link-based scores of web pages)
    - Adding links to message boards (link counts)
    - Link exchange networks
    - Link farming (using a large number of domains & sites)
  - Term spam (to modify the textual representation of a doc)
    - URL term spam (matching anchor text and URL)
    - Dumping (filling documents with unrelated words)
    - Phrase stitching (combining words/sentences from diff. sources)
    - Weaving (adding spam terms into a valid source)

Spam Example

Website:

BETTING NFL FOOTBALL PRO FOOTBALL SPORTSBOOKS NFL FOOTBALL LINE ONLINE NFL SPORTSBOOKS NFL

Players Super Book

When It Comes To Secure NFL Betting And Finding The Best Football Lines Players Super Book Is The Best Option! Sign Up And Ask For 30 % In Bonuses.

MVP Sportsbook

Football Betting Has Never been so easy and secure! MVP Sportsbook has all the NFL odds you are looking for. Sign Up Now and ask for up to

30 % in Cash bonuses.

Term spam:

pro football sportsbooks nfl football line online nfl sportsbooks nfl football gambling odds online pro nfl betting pro nfl gambling online nfl football spreads offshore football gambling online nfl gamblibg spreads online football gambling line online nfl betting nfl sportsbook online online nfl betting spreads betting nfl football online online football wagering online gambling online gambling football online nfl football betting odds offshore football sportsbook online nfl football gambling ...

Repetition of important words

Link spam:

MVP Sportsbook Football Gambling Beverly Hills Football Sportsbook
Players SB Football Wagering Popular Poker Football Odds
Virtual Bookmaker Football Lines V Wager Football Spreads
Bogarts Casino Football Point Spreads Gecko Casino Online Football Betting
Jackpot Hour Online Football Gambling MVP Casino Online Football Wagering
Toucan Casino NFL Betting Popular Poker NFL Gambling
All Tracks NFL Wagering Bet Jockey NFL Odds
Live Horse Betting NFL Lines MVP Racebook NFL Point Spreads
Popular Poker NFL Spreads Bogarts Poker NFL Sportsbook ...

Mentioning Related websites

Spam Detection

- Useful features
  - Unigrams (number of words in a page/title/anchor text)
  - Formatting (invisible text, flashing, etc.)
  - Fraction of terms drawn from popular words
  - Misspellings
  - IP address
- Different features are useful for diff. spam detection tasks
- Email/Web page spam are by far the most widely studied, well understood, and easily detected types of spam

Example Spam Assassin Output

To: ...
From: ...
Subject: non profit debt
X-Spam-Checked: This message probably not SPAM
X-Spam-Score: 3.853, Required: 5
X-Spam-Level: *** (3.853)
X-Spam-Tests: BAYES_50,DATE_IN_FUTURE_06_12,URIBL_BLACK
X-Spam-Report-rig: ---- Start SpamAssassin (v2.6xx-cscf) results

2.0 URIBL_BLACK Contains an URL listed in the URIBL blacklist
[URIs: bad-debtyh.net.cn]
1.9 DATE_IN_FUTURE_06_12 Date: is 6 to 12 hours after Received: date
0.0 BAYES_50 BODY: Bayesian spam probability is 40 to 60%
[score: 0.4857]

Say good bye to debt
Acceptable Unsecured Debt includes All Major Credit Cards, No-collateral
Bank Loans, Personal Loans,
Medical Bills etc.
http://www.bad-debtyh.net.cn

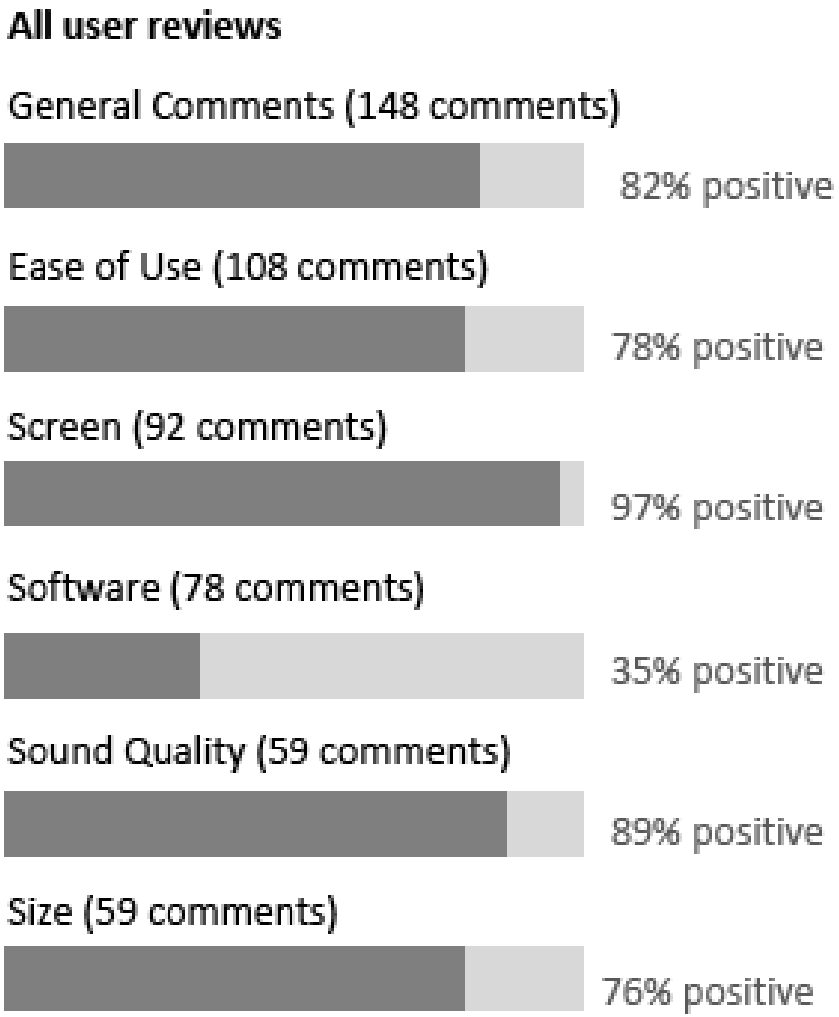
Sentiment

- Blogs, online reviews, & forum posts are often opinionated
- Sentiment classification attempts to automatically identify the polarity of the opinion
  - Negative opinion
  - Neutral opinion
  - Positive opinion
- Sometimes the strength of the opinion is also important
  - “Two stars” vs. “four stars”
  - Weakly negative vs. strongly negative

Classifying Sentiment

- Useful features (opinionated text)
  - Unigrams
  - Bigrams
  - Part of speech tags
  - Adjectives and nouns
- SVMs with unigram features have been shown to outperform hand built rules

Sentiment Classification Example



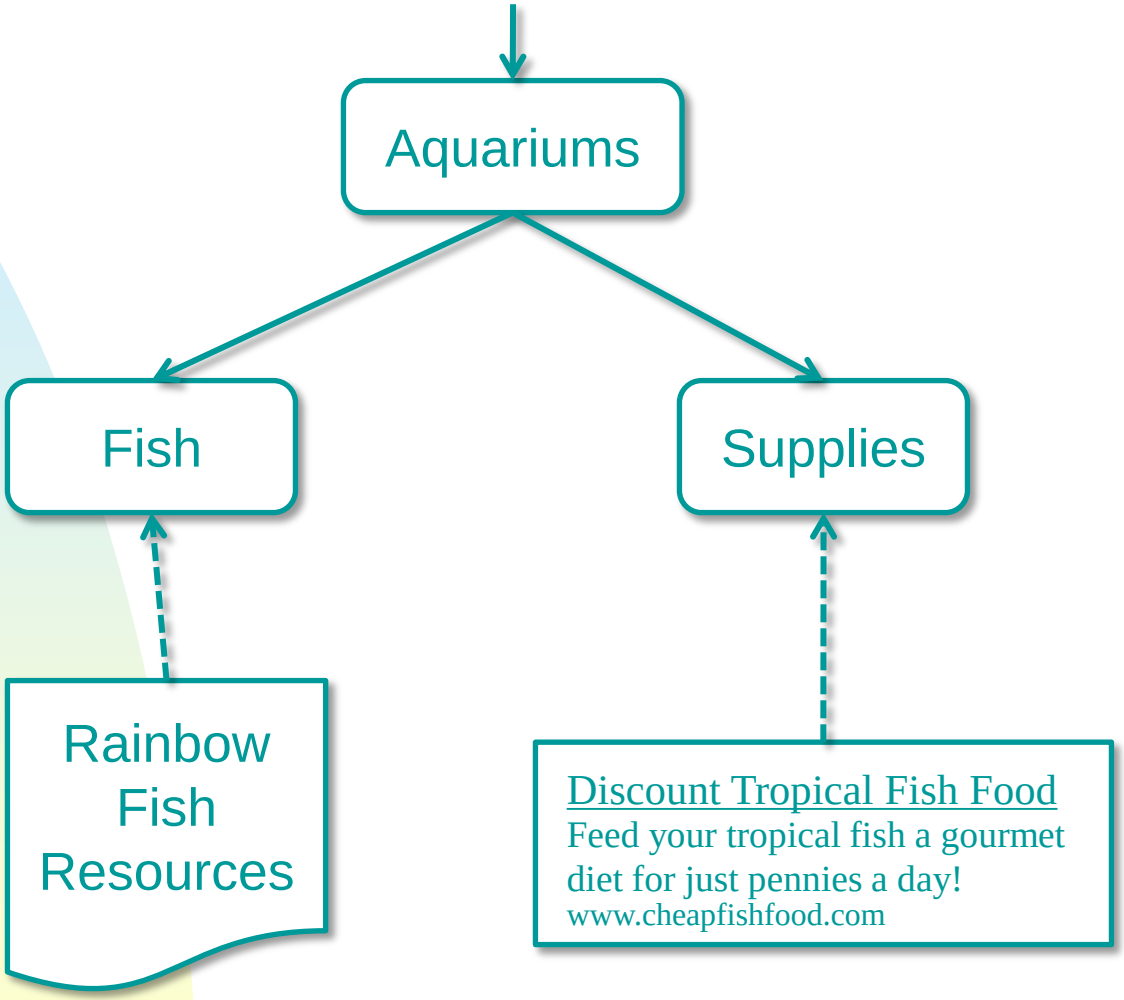
Classifying Online Ads

- Unlike traditional search, online advertising goes beyond "topical relevance"
- A user searching for 'tropical fish' may also be interested in pet stores, local aquariums, or even scuba diving lessons
- These are semantically related, not topically relevant!
- We can bridge the semantic gap by classifying ads & queries according to a semantic hierarchy

Semantic Classification

- Semantic hierarchy ontology, e.g., Pets/Aquariums/Supplies
- Training data
  - Large number of queries & ads are manually classified into the hierarchy
- *Nearest neighbor classification* has been shown to be effective for this task
- Manually constructed *hierarchical structure* of classes can be used to improve classification accuracy

Semantic Classification



Web Page

Ad

Clustering

- A set of unsupervised algorithms that attempt to find *latent structure* in a set of items
- Goal is to identify groups (clusters) of *similar items*, given a set of unlabeled instances
- Suppose I gave you the *shape*, *color*, *vitamin C* content and *price* of various fruits and asked you to cluster them
  - What criteria would you use?
  - How would you define similarity?
- Clustering is very sensitive to (i) how items are *represented* and (ii) how *similarity* is defined

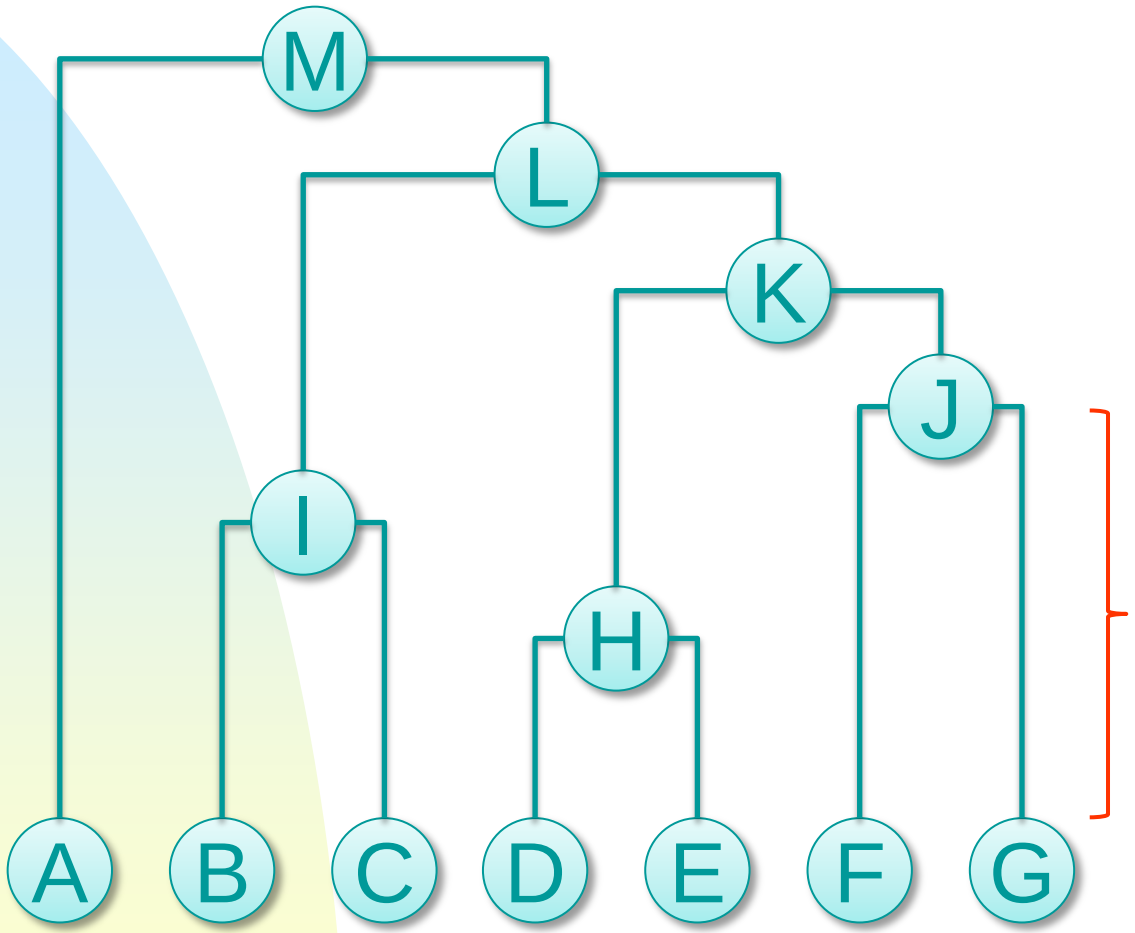
Clustering

- General outline of clustering algorithms
  1. Decide how items will be *represented* (e.g., feature vectors)
  2. Define *similarity measure* between pairs or groups of items (e.g., cosine similarity)
  3. Determine what makes a "*good*" clustering (e.g., using intra- & inter-cluster similarity measures
  4. *Iteratively construct* clusters that are increasingly "good"
  5. Stop after a *local/global optimum* clustering is found
- Steps 3 and 4 differ the most across algorithms

Hierarchical Clustering

- Constructs a *hierarchy* of *clusters*
  - Starting with some initial clustering of data & iteratively trying to improve the "quality" of clusters
  - The *top* level of the hierarchy consists of a *single* cluster with *all* items in it
  - The *bottom* level of the hierarchy consists of *N* (number of items) *singleton* clusters
- Different objectives lead to different types of clusters
- Two types of hierarchical clustering
  - Divisive ("*top down*")
  - Agglomerative ("*bottom up*")
- Hierarchy can be visualized as a *dendogram*

Example Dendrogram

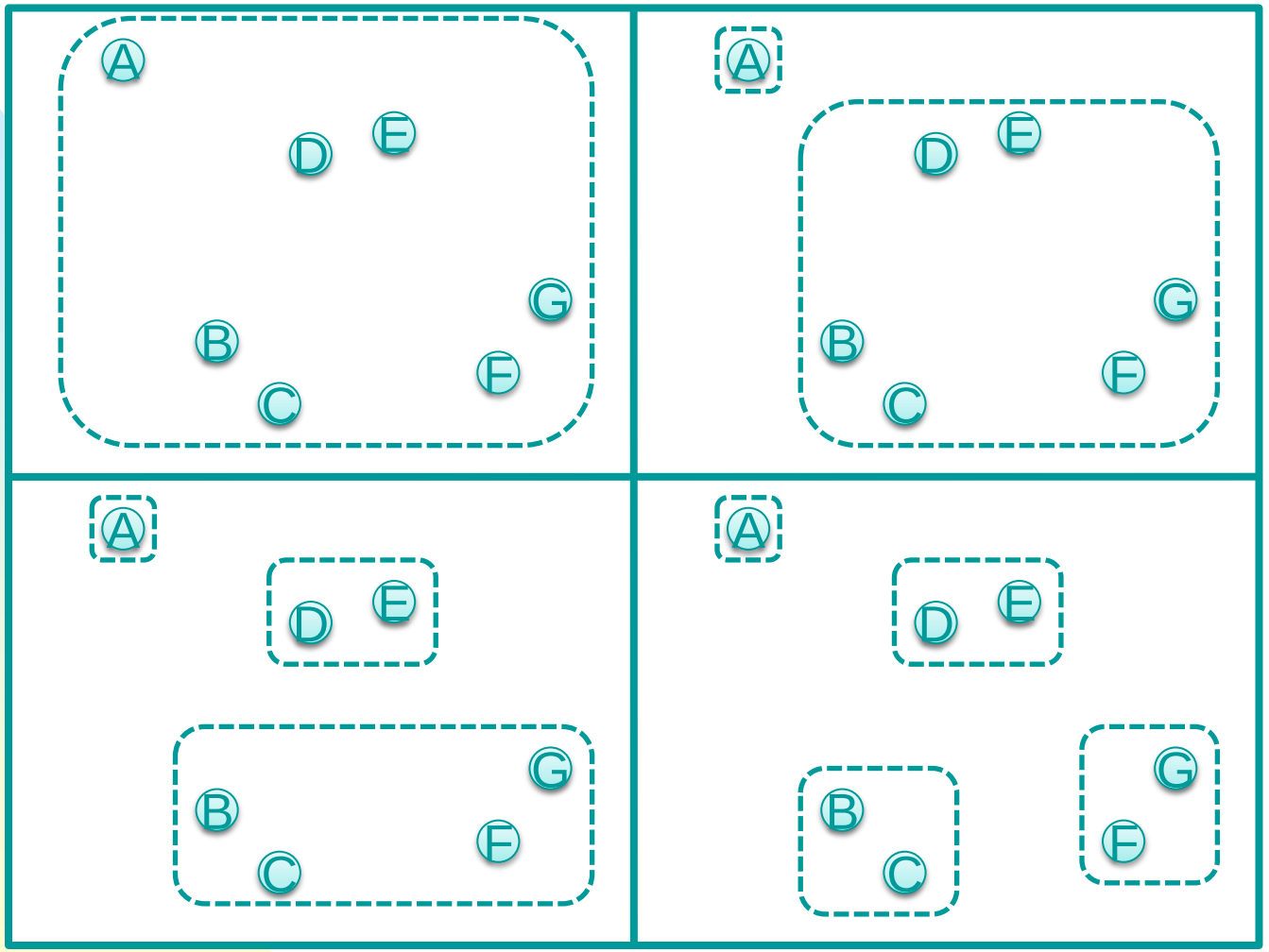


Height indicates the similarity of the clusters involved

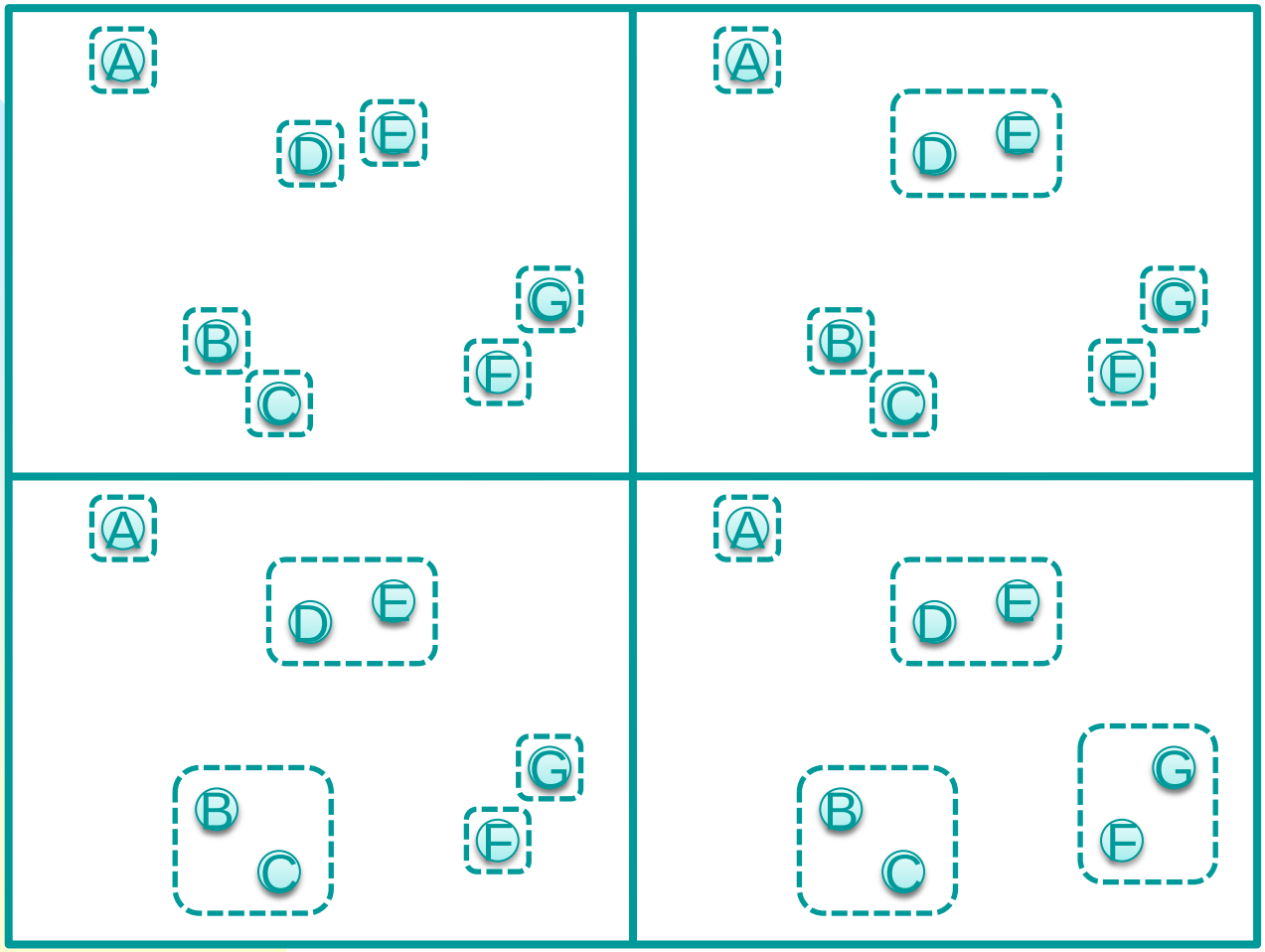
Divisive & Agglomerative Hierarchical Clustering

- Divisive
  - Start with a *single* cluster consisting of all of the items
  - Until only *singleton clusters* exist …
    - *Divide* an existing cluster into two (or more) new clusters
- Agglomerative
  - Start with *N* (number of items) singleton clusters
  - Until a single cluster exists …
    - *Combine* two (or more) existing cluster into a new cluster
- How do we know how to divide or combine clusters?
  - Define a *division* or *combination cost*
  - Perform the division or combination with the *lowest cost*

Divisive Hierarchical Clustering



Agglomerative Hierarchical Clustering



Clustering Costs

- Cost: a measure of how *expensive* to merge 2 clusters
- Single linkage

$$COST(C_i, C_j) = \min\{dist(X_i, X_j) | X_i \in C_i, X_j \in C_j\}$$

- Complete linkage

$$COST(C_i, C_j) = \max\{dist(X_i, X_j) | X_i \in C_i, X_j \in C_j\}$$

- Average linkage

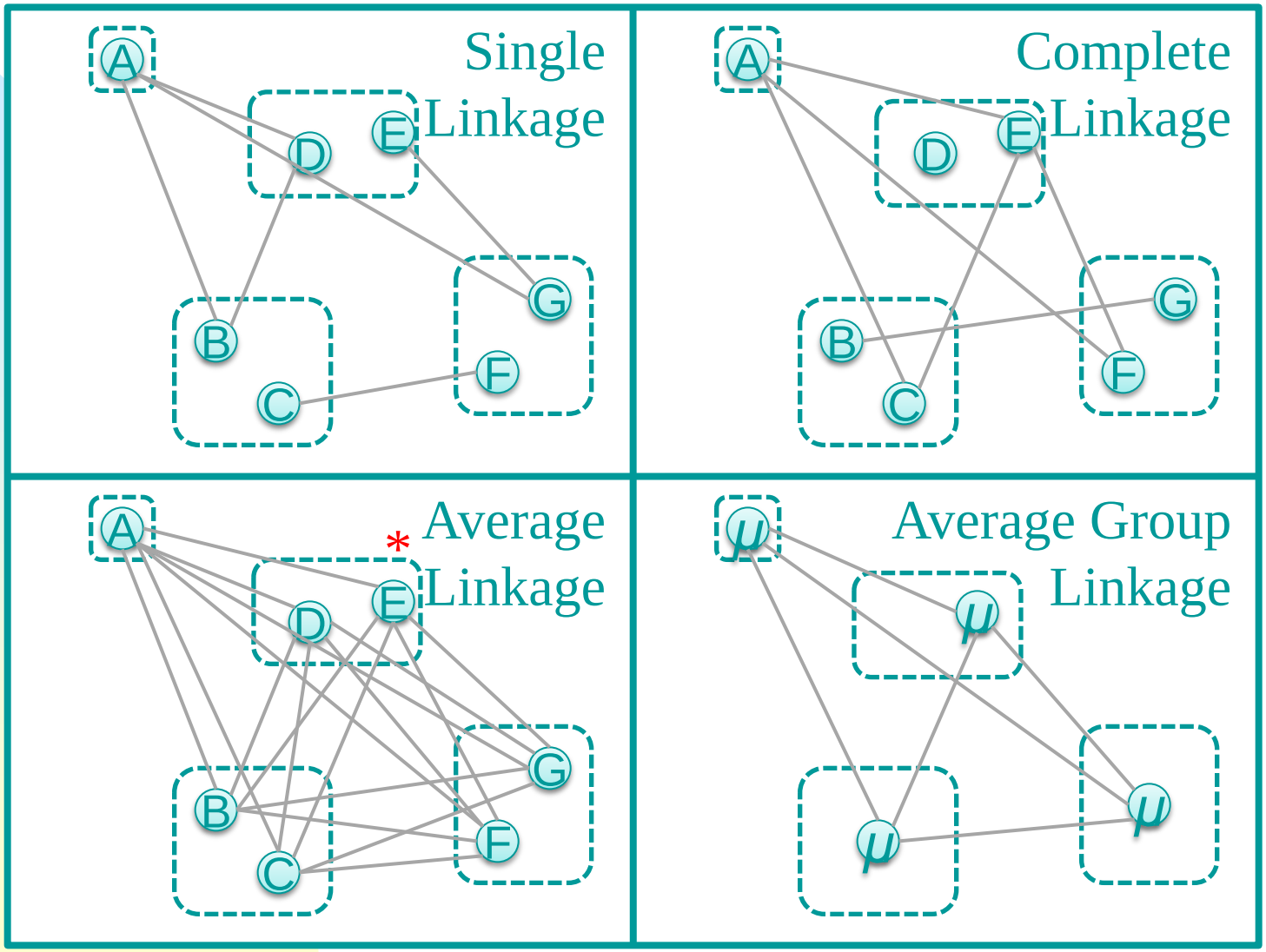
$$COST(C_i, C_j) = \frac{\sum_{X_i \in C_i, X_j \in C_j} dist(X_i, X_j)}{|C_i||C_j|}$$

- Average group linkage

$$COST(C_i, C_j) = dist(\mu_{C_i}, \mu_{C_j})$$

where $U_C = (\sum_{X \in C} X) / |C|$ is the centroid of cluster C

Clustering Strategies



* Generally, *Average-Link* Clustering yields the best effectiveness

Clustering Costs

- The choice of the *best* clustering technique/strategy requires experiments & evaluation
- Single linkage
  - Could result in " very long" or "spread-out" clusters
- Complete linkage
  - Clusters are *more compact* than Single Linkage
- Average linkage
  - A *compromise* between Single & Complete Linkage
- Average group linkage
  - *Closely related* to the Average Linkage approach

K-Means Clustering

- *Hierarchical clustering* constructs a hierarchy of clusters
- K-means always maintains exactly *K* clusters
  - Clusters are represented by their centroids ("centers of mass")
- Basic algorithm:
  - Step 0: Choose *K cluster centroids*
  - Step 1: Assign points to *closet centroid*
  - Step 2: Re-compute *cluster centroids*
  - Step 3: Goto Step 1
- Tends to *converge quickly*
- Can be *sensitive* to choice of *initial centroids*
- Must choose *K* to begin with!

K-Means Clustering

- Goal: find the cluster assignments (for the assignment vectors $A[1], \ldots, A[N]$) that minimize the cost function:

$$\text{COST}(A[1], \ldots, A[N]) = \sum_{k=1}^{K} \sum_{i:A[i]=k} \text{dist}(X_i, C_k)$$

where $\text{dist}(X_i, C_k) = \| X_i - \mu_{C_k} \|^2$, where μ_{C_k} is the *centroid* of C_k

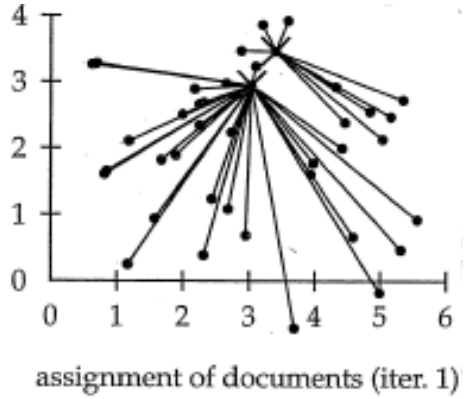
$= (X_i - \mu_{C_k}) \cdot (X_i - \mu_{C_k})$, the *Euclidean Distance*

- Strategy:
  1. Randomly select K initial cluster centers (instances) as *seeds*
  2. Move the cluster centers around to *minimize* the cost function
     i. *Re-assign* instances to the cluster with the *closest* centroid
     ii. *Re-compute* the cost value of each centroid based on the current members of its cluster

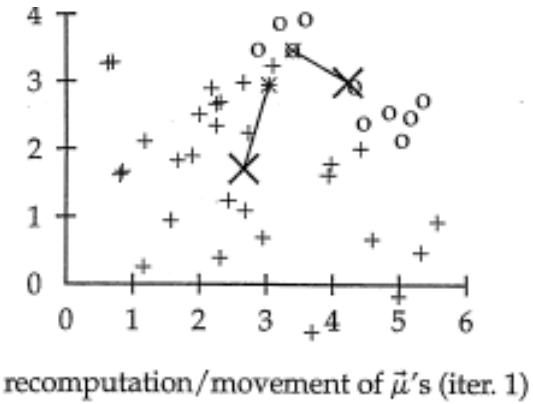
K-Means Clustering

- Example.

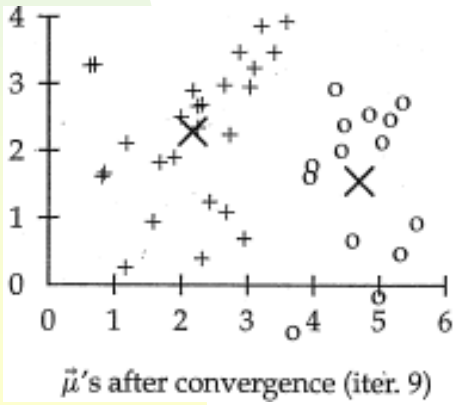
(a)



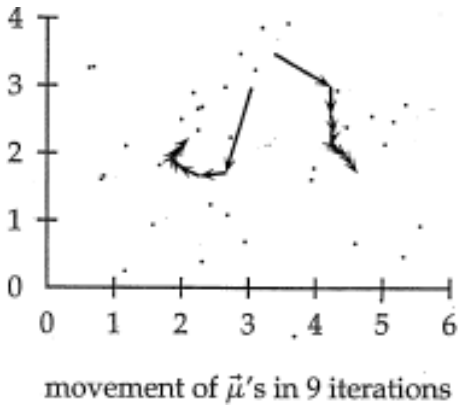
(b)



(c)



(d)



(e)

K-Means Clustering

- The *K*-means optimization problem:
  - A naïve approach is to try every possible combination of cluster assignments, which is *infeasible* for large data sets
  - The *K*-means algorithm should find an *approximate*, *heuristic solution* that iteratively tries to minimize the cost
  - Anticipated results:
    1. the solution is not guaranteed to be *globally optimal*
    2. despite the heuristic nature, the *K*-means algorithm tends to work very well in practice
- In practice, *K*-means clustering tends to converge quickly
  - Compared to hierarchical clustering *H*, *K*-means is more *efficient* and produces clusters of similar quality to *H*
  - Implementing *K*-means requires O(*KN*), rather than O(N^2) for *H*

K-Means Clustering Algorithm

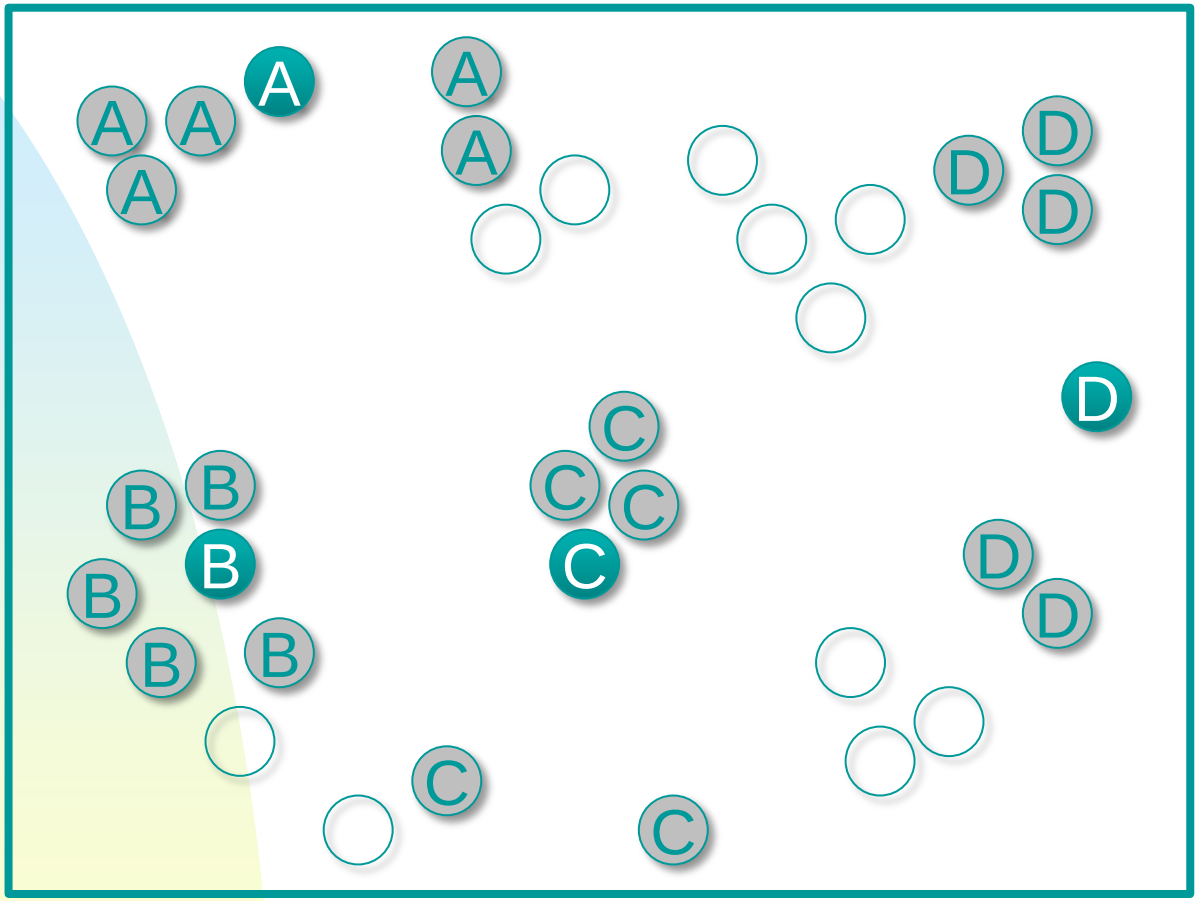
Algorithm 1 K-Means Clustering

```
1: procedure KMeansCluster(X_1, ..., X_N, K)
2:     A[1], ..., A[N] ← initial cluster assignment (* Either randomly or using
3:     repeat                                          some knowledge of the data *)
4:         change ← false
5:         for i = 1 to N do
6:             k̂ ← arg min_k dist(X_i, C_k)     (* Each instance is assigned to the
7:             if A[i] is not equal k̂ then        closest cluster *)
8:                 A[i] ← k̂
9:                 change ← true                 (* The cluster of an instance changes;
10:            end if                              proceeds *)
11:        end for
12:    until change is equal to false return A[1], ..., A[N]
13: end procedure
```

K Nearest Neighbor Clustering

- *Hierarchical* and *K-Means* clustering partition items into clusters
  - Every item is in exactly one cluster
- *K* Nearest neighbor clustering forms one cluster per item
  - The cluster for item *j* consists of *j* and the *K nearest neighbors* of *j*
  - Clusters now overlap

5 Nearest Neighbor Clustering



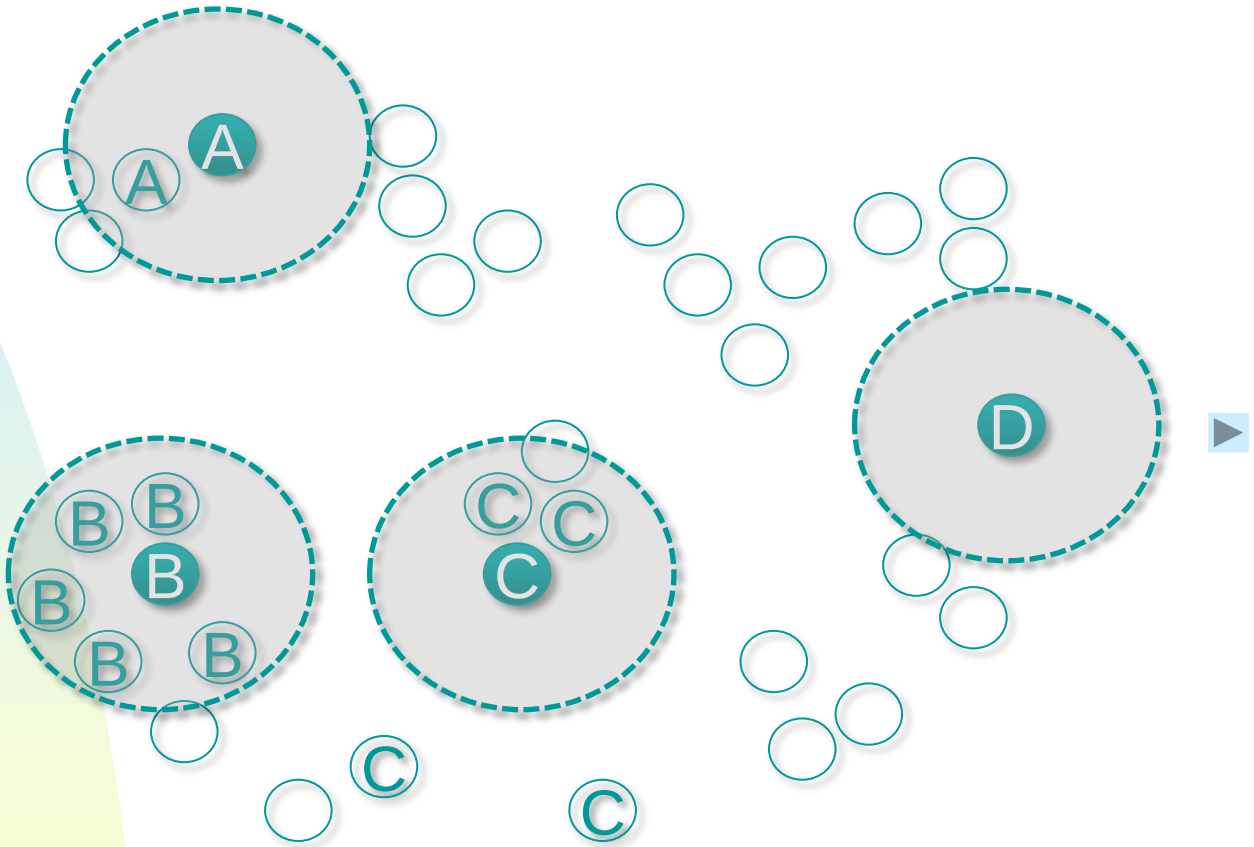
K Nearest Neighbor Clustering

- *Drawbacks* of the *K* Nearest Neighbor Clustering method:
  - Often fails to find meaningful clusters
    - In *spare areas* of the input space, the instances assigned to a cluster are father far away (e.g., *D* in the 5-NN example) ▶
    - In *dense areas*, some related instances may be missed if *K* is not large enough (e.g., *B* in the 5-NN example) ▶
  - *Computational expensive* (compared with *K*-means), since it computes distances between *each pair* of instances
- *Applications* of the *K* Nearest Neighbor Clustering method
  - Emphasize finding a small number (rather than all) of closely related instances, i.e., precision over recall

How to Choose K?

- *K*-means and *K* nearest neighbor clustering require us to choose *K*, the *number of clusters*
- No theoretically appealing way of choosing *K*
- Depends on the *application* and *data*; often chosen experimentally to evaluate the *quality* of the resulting clusters for various values of *K*
- Can use *hierarchical clustering* and choose the best level
- Can use *adaptive K* for *K-nearest neighbor* clustering
  - Larger (Smaller) *K* for *dense* (spare) areas
  - Challenge: choosing the *boundary* size
- Difficult problem with no clear solution

Adaptive Nearest Neighbor Clustering



Evaluating Clustering

- Evaluating clustering is challenging, since it is an unsupervised learning task
- If labels exist, can use standard IR metrics, e.g., *precision/recall*
- If not, then can use measures such as "*cluster precision*", which is defined as:

$$ClusterPrecision = \frac{\sum_{i=1}^{K} |\text{MaxClass}(C_i)|}{N}$$

where K (= $|C|$) is the total number of resultant clusters

$|\text{MaxClass}(C_i)|$ is the number of instances in cluster C_i with the *most* instances of (human-assigned) class label C_i

N is the total number of instances

- Another option is to evaluate clustering as part of an end-to-end system, e.g., *clusters* to improve web *ranking*