Mining text data

1. Overview
2. Preprocessing text data
3. Representations (bag-of-words)
4. Text clustering and its applications
5. Extra: Word embeddings

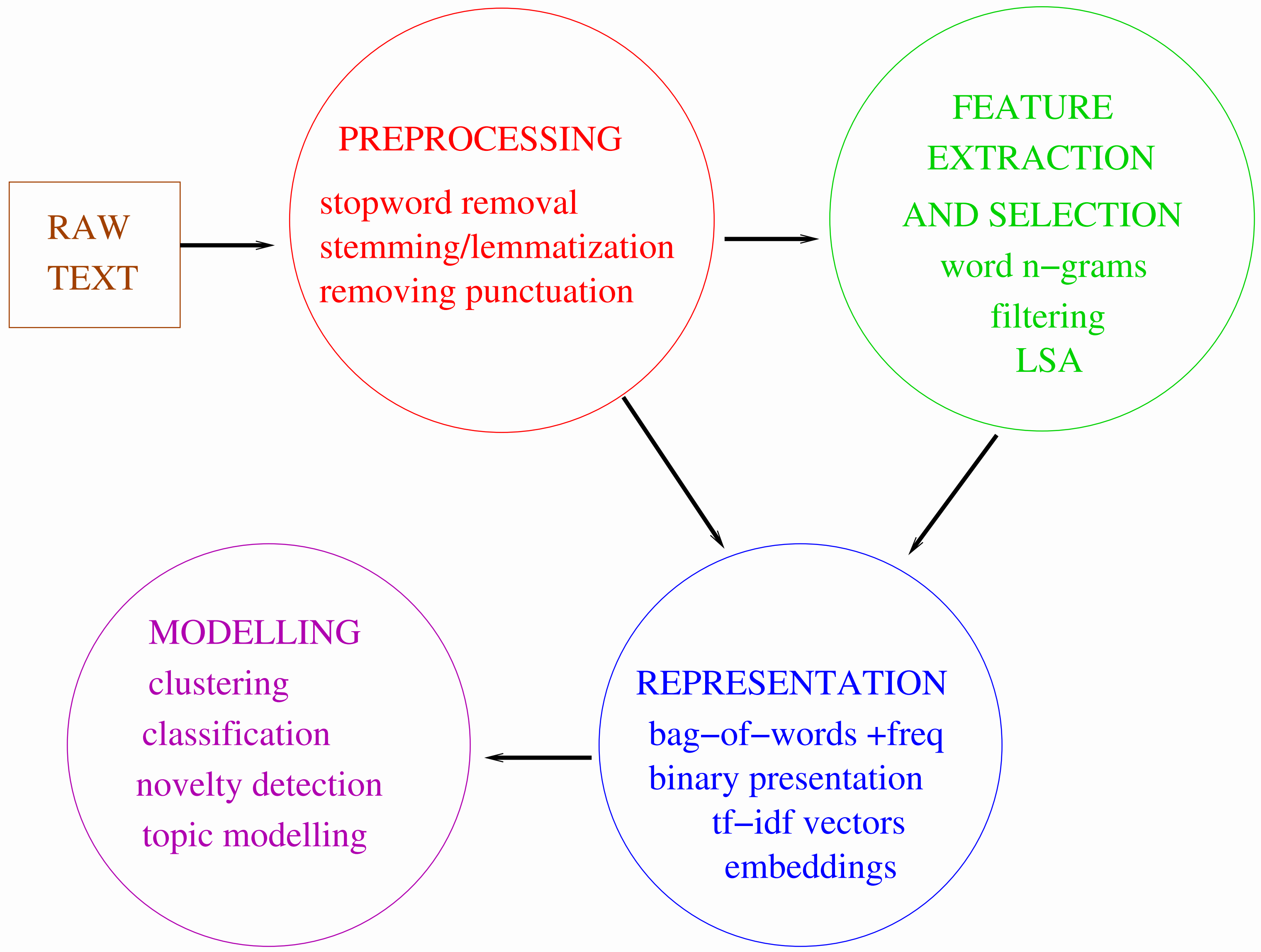
1. Overview: Text data

Corpus = collection of text documents $(\mathbf{d}_1, \dots, \mathbf{d}_n)$
Lexicon = set of words $(w_1, \dots, w_m)$

Document

- originally **ordered**: sequence of words (string), e.g., $\mathbf{d}_1 = (w_{11}, w_{12}, \dots, w_{1q})$, length q
- **bag-of-words model**: **unordered** set of words (+ frequencies), e.g., *Our cat likes the neighbour's cat.* → {*our: 1, cat: 2, likes: 1, the: 1, neighbour's: 1*}
- **vector space presentation** as a numerical (or binary) vector $\mathbf{x} = (x_1, \dots, x_m)$
  - very sparse! (many 0s)
  - occurrence of words more important than absence

Main steps and example tasks



2. Main tasks of preprocessing

- **Tokenization** (usually word=token)
- **Lower-casing**
- **Remove stopwords** (may be stemmed or not, may contain apostrophes (*it's, you'd*))
- Reduce inflected forms into a base form:
  - **Stemming**: cut suffixes (*running* →*run*)
  - **Lemmatization**: derive dictionary form (*was* → *be*, *mice* → *mouse*)
- **Remove punctuation**. Decide how to handle
  - digits (numbers sometimes informative)
  - dashes in compound words (*cat-food* → *catfood or cat food*?)

Stopwords

- frequently occurring words with little information about semantic content
- not discriminative
- articles, prepositions, pronouns, auxiliary verbs, ...
- multiple lists available
- Note: relevance of words depends on the context!
  $\rightarrow$ check & edit!

⋮

```
cannot
cant
co
computer
con
could
couldnt
cry
de
describe
detail
```

⋮

Warning:
may be
relevant

Stemming and lemmatization

Stemming cuts suffixes → root form

- Overstemming: word is over-truncated → false synonyms (e.g., *hopping* and *hope* → *hop*)
- Understemming: too little truncation → same root not detected (e.g., *alumnus* → *alumnu* vs. *alumni* → *alumni*)
- rule-based, possibly use look-up tables

Lemmatization derive dictionary form

- utilize context, part-of-speech tagging [a], look-up tables
- more difficult! → slower
- potentially better accuracy

[a] noun, verb, adjective, ...; singular/plural; verb tense; subject, object, ...

Example: simple preprocessing

lowercasing
punctuation and
stopword removal

d1: The kitty likes mice and mouse–flavoured cat food
d2: The beer had a strong hop flavour.
d3: I saw big hops in the garden.
d4: I hopped the zoo tour hoping to see big cats.

→

d1: kitty likes mice mouse flavoured cat food
d2: beer strong hop flavour
d3: saw big hops garden
d4: hopped zoo tour hoping see big cats

lemmatization

d1: kitty like mouse mouse flavour cat food
d2: beer strong hop flavour
d3: see big hop garden
d4: hop zoo tour hope see big cat

stemming

d1: kitty like mice mouse flavour cat food
d2: beer strong hop flavour
d3: saw big hop garden
d4: hop zoo tour hope see big cat

synonymy: kitty, cat

polysemy/homonymy: hop as a verb and a plant

Example: document-term matrix

d1: kitty like mouse mouse flavour cat food

d2: beer strong hop flavour

d3: see big hop garden

d4: hop zoo tour hope see big cat

Typically very sparse matrix! Present compactly!

	kitty	like	mouse	flavour	cat	food	beer	strong	hop	see
d1	1	1	2	1	1	1				
d2				1			1	1	1	
d3									1	1
d4					1				1	1

	big	garden	zoo	tour	hope
d1					
d2					
d3	1	1			
d4	1		1	1	1

Note: 0s often skipped in visual presentation (not stored in real structure)

3. Representation (VSM=vector space model)

1. **Frequency model**: $x_{ij} = fr(w_j|\mathbf{d}_i)$ + possibly normalize to $||\mathbf{x}|| = 1$ ($fr(w_j|\mathbf{d}_i)$=number of times w_j occurs in $\mathbf{d}_i$)
2. **Boolean (binary) model**: $x_{ij} = 1$, if w_j occurs in $\mathbf{d}_i$ and 0 otherwise
3. **Tf-idf representation**: $tf \cdot idf$, where tf=term frequency, idf=inverse document frequency
   - basic form $tfidf(w_j, \mathbf{d}_i) = fr(w_j|\mathbf{d}_i) \cdot \log \frac{n}{n_j}$
     - $df(w_j) = \frac{n_j}{n}$, $idf(w_j) = \frac{n}{n_j}$, n_j=number of documents containing w_j
   - When $tfidf$ is maximal?
   - many variants! $\rightarrow$ Check what your library calculates!

Some tf-idf variants

- *tf* can be normalized: $\frac{fr(w_j|\mathbf{d}_i)}{len(\mathbf{d}_i)}$
- frequency damping: $tf(w_j, \mathbf{d}_i) = \log(fr(w_j|\mathbf{d}_i))$ or $\sqrt{fr(w_j|\mathbf{d}_i)}$
- Note: **sklearn** offers **length-normalized** [a] versions of either $fr(w_j|\mathbf{d}_i) \cdot \left(\log\left(\frac{1+n}{1+n_j}\right) + 1\right)$ or $fr(w_j|\mathbf{d}_i) \cdot \left(\log\left(\frac{n}{n_j}\right) + 1\right)$
  - always $idf > 0$
  - $\frac{1+n_j}{1+n}$ smoothed estimate of relative document frequency

[a] either $\sum_j x_{ij}^2 = 1$ or $\sum_j |x_{ij}| = 1$ or none

Example: Basic tf-idf

$tf = fr(w_j|\mathbf{d}_i) = 1$ or 2 (*mouse*)
$n_j = df(w_j) \in \{1, 2, 3\} \Rightarrow idf(w_j) \in \{2, 1, 0.415\}$.

	kitty	like	mouse	flavour	cat	food	beer	strong	hop	see
$\mathbf{d}1$	2.0	2.0	4.0	1.0	1.0	2.0				
$\mathbf{d}2$				1.0			2.0	2.0	0.415	
$\mathbf{d}3$									0.415	1.0
$\mathbf{d}4$					1.0				0.415	1.0

	big	garden	zoo	tour	hope
$\mathbf{d}1$					
$\mathbf{d}2$					
$\mathbf{d}3$	1.0	2.0			
$\mathbf{d}4$	1.0		2.0	2.0	2.0

Are single words good features?

Basic features: (stemmed) words (excluding stopwords)

1. too frequent words don't separate documents
   - stopword removal and idf help
   - common word may be part of useful feature, e.g., *data mining, data management, text data* → word n-grams
2. unique or rare words don't reveal similarity between documents
   - words may still be synonyms (*kitty, cat*) → WordNet
   - or related → LSA and word embeddings can help
   - different spelling (neighbour, neighbor) or errors
3. polysemy problem: similar looking word (or stem) may have different meanings (*hop*)

Create features for word n-grams?

word n-gram = sequence of n words that often occur together (phrases, collocations)

- monogram (unigram, 1-gram): single word "*data*"
- bigram: two words: "*data analysis*"
- trigram: three words: "*Bayesian data analysis*"

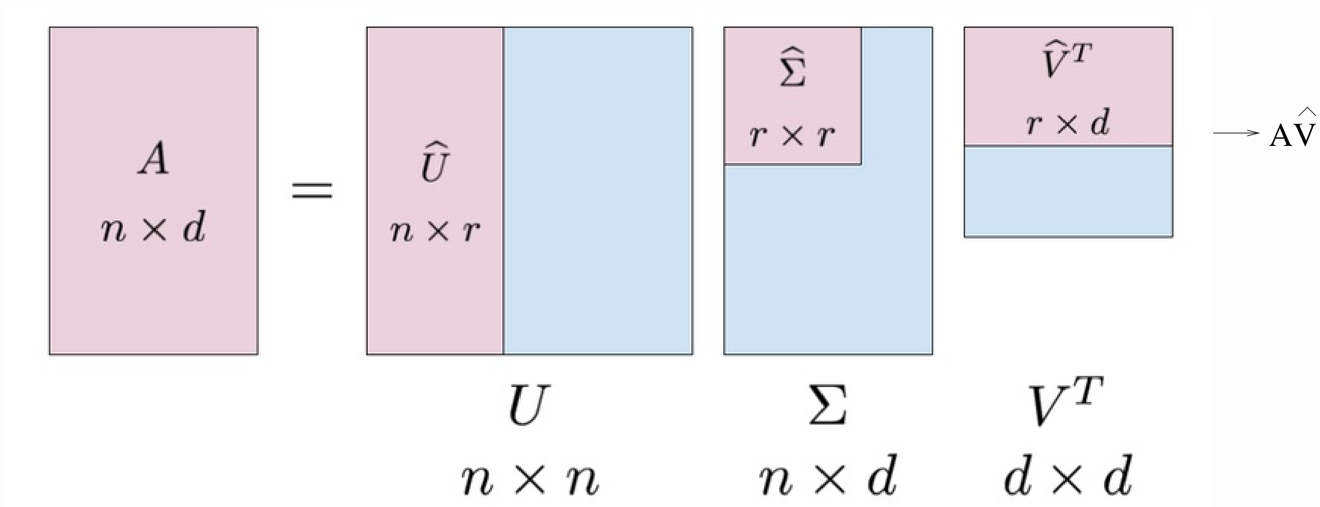
results a lot of features! $\Rightarrow$ filtering by

- frequency (keep if $fr(w_1 w_2)$ high)
- significance of association (e.g., $MI(w_1, w_2)$, $\chi^2(w_1, w_2)$)
- dimension reduction (e.g., LSA)

Note: character n-grams = n consecutive characters

Latent semantic analysis (LSA) = SVD applied to text data

- radical dimension reduction (e.g., 100 000 → 300)
- helps with synonymy and a bit with polysemy
- similar/related terms tend to be combined, e.g.,
  {car, truck, flower} → {1.3452 · car + 0.2828 · truck, flower}



Recall: If old features $F_1, \ldots, F_d$ and ith singular vector $(\mathbf{V}_{i1}, \ldots, \mathbf{V}_{id})^T$, ith new feature $\sum_{j=1}^{d} \mathbf{V}_{ij} F_j$. LSA: Recap Aggarwal 2.4.3.3. Image: Perunicic (2017).

4. Clustering text data

Given **vector-space representation** of text data:
any common clustering method

- K-representatives, spectral, hierarchical, probabilistic EM-algorithm, affinity propagation
- distance/similarity: prefer **cosine similarity**!
  remember: if data normalized to $||\mathbf{x}|| = 1$,
  $L_2^2(\mathbf{x}_1, \mathbf{x}_2) = 2(1 - cos(\mathbf{x}_1, \mathbf{x}_2)) \rightarrow L_2$ ok
- dimension reduction (PCA or LSA) can help

Suggestion: Use K-means always as a baseline to compare other methods!

Example: cos-sim vs. L_2 distance

$\mathbf{d}1$: kitty like mouse mouse flavour cat food
$\mathbf{d}2$: beer strong hop flavour
$\mathbf{d}3$: see big hop garden
$\mathbf{d}4$: hop zoo tour hope see big cat

$\cos_T$=cos-sim of tfidf vectors, $\cos_B$=cos-sim of binary vectors, L_2=Euclidean distance between tfidf vectors:

pair	$\cos_T$	$\cos_B$	L_2	note
$(\mathbf{d}1, \mathbf{d}2)$	0.0603	0.204	6.097	
$(\mathbf{d}1, \mathbf{d}3)$	0	0	6.014	No common words!
$(\mathbf{d}1, \mathbf{d}4)$	0.0469	0.154	6.571	
$(\mathbf{d}2, \mathbf{d}3)$	0.0229	0.250	**3.873**	
$(\mathbf{d}2, \mathbf{d}4)$	0.0146	0.189	4.899	
$(\mathbf{d}3, \mathbf{d}4)$	**0.2245**	**0.567**	4.123	

Special techniques for clustering text

1. Modifications of K-representatives
   - replace cluster centroids by **cluster digests** = most frequent words (e.g., 200–400)
   - better strategies to select seeds!
2. Co-clustering
   - Idea: cluster words and documents simultaneously
   - cluster $C_i = (R_i, V_i)$; V_i =most relevant words (cluster digest) in documents R_i

Note: “cluster digest” has slightly different meanings in different contexts (intention to describe the cluster)

Scatter/Gather: better K-means for text

1. Construct K **seeds** with agglomerative **hierarchical clustering**. Two alternatives:
   a) **Buckshot**: choose $\sqrt{Kn}$ random samples $\rightarrow K$ clusters $\rightarrow K$ seeds
   b) **Fractionation**: divide data into $\frac{n}{m}$ buckets $\rightarrow \nu m$ clusters per bucket ($\nu \in]0,1[$) $\rightarrow$ concatenate cluster documents $\rightarrow$ repeat until K clusters left
2. Apply K**-means**
   - centroid=concatenation of documents + remove infrequent words

Scatter/Gather: better K-means for text

3. (Optionally) **Refine clusters**:

   a) Split incoherent clusters

      - coherence score: $avg(sim(\mathbf{x}, \mathbf{c}))$ or $avg(sim(\mathbf{x}_i, \mathbf{x}_j) \mid \mathbf{x}_i, \mathbf{x}_j \in \mathbf{C})$
      - if too low $\rightarrow$ Buckshot with $K = 2$ + recluster rest

   b) Join similar clusters

      - topical words overlapping significantly

Cutting et al. (1992). Scatter/Gather: A cluster-based approach to browsing large document collections.

Co-clustering (biclustering)

Idea: rearrange rows and columns of doc-term matrix such that most non-zero entries form blocks.

	CHAMPION	ELECTRON	TROPHY	RELATIVITY	QUANTUM	TOURNAMENT
D_1	2	0	1	1	0	3
D_2	0	2	0	1	3	0
D_3	1	3	0	1	2	0
D_4	2	0	2	0	0	3
D_5	0	2	1	1	3	0
D_6	1	0	2	0	0	3

(a) Document-term matrix

SPORTS CO-CLUSTER →

	CHAMPION	TROPHY	TOURNAMENT	ELECTRON	RELATIVITY	QUANTUM
D_1	2	1	3	0	1	0
D_4	2	2	3	0	0	0
D_6	1	2	3	0	0	0
D_2	0	0	0	2	1	3
D_3	1	0	0	3	1	2
D_5	0	1	0	2	1	3

← PHYSICS CO-CLUSTER

(b) Re-arranged document-term matrix

Image source Aggarwal Fig. 13.1

=Bipartite graph partitioning problem!

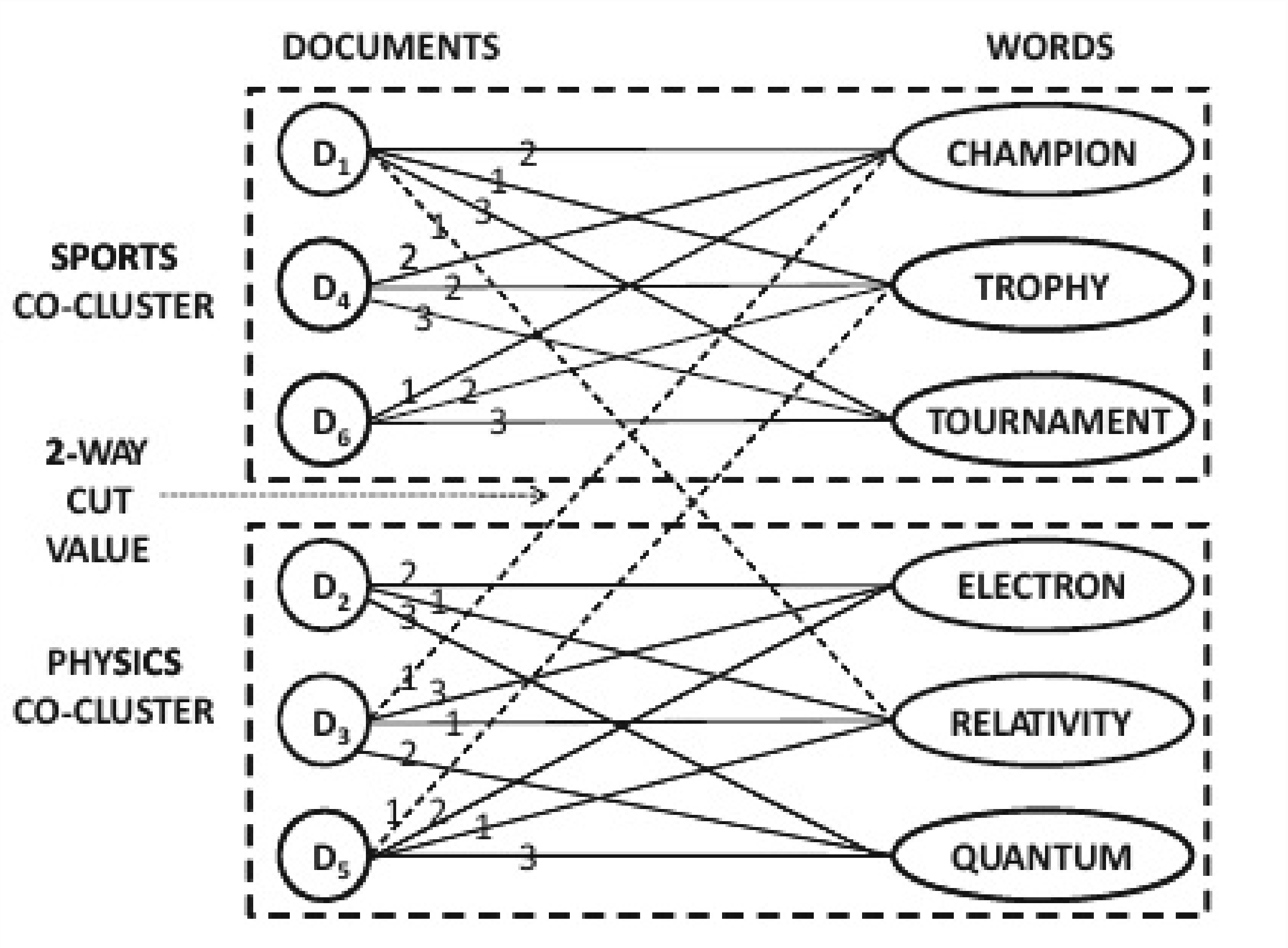


Image source Aggarwal Fig. 13.2

Graph partitioning

Construct graph $\mathbf{G} = (\mathbf{U} \cup \mathbf{V}, \mathbf{E}, \Gamma)$

- $\mathbf{U}$ = nodes for documents (for doc. $\mathbf{d}_i$ node u_i)
- $\mathbf{V}$ = nodes for words (for word w_j node v_j)
- $\mathbf{E}$ = edges
  $(u_i, v_j) \in \mathbf{E}$, if word w_j occurs in $\mathbf{d}_i$
- Γ = weights
  $\gamma_{ij} = fr(w_j|\mathbf{d}_i)$ or $\gamma_{ij} = tfidf(w_j, \mathbf{d}_i)$

Objective: Partition $\mathbf{U} \cup \mathbf{V}$ into groups such that edge-cut cost ($\sum \gamma_{ij}$ between groups) minimal (+ extra constraints)

Solution: Spectral clustering or other graph partitioning methods (vs. community detection)

Application 1: clustering for classification

Centroid-based classifier:

- Cluster documents of each class (size n_i) into $k_i \propto n_i$ clusters
- **cluster digest**=most common words of the centroid
- For $\mathbf{d}_{new}$:
  - determine K nearest digests (clusters)
  - report the dominant label

+ fast alternative to K-NN classifier
+ handles **synonymy** (similar words → same centroid) and **polysemy** (different meanings → different centroids)

Application 2: Novelty detection

Problem: Temporal stream of text documents, when a new topic appears? (e.g., news)

Simple solution:

- Maintain a sample of documents, $\mathcal{D}$
- For $\mathbf{d}_{new}$ calculate $sim(\mathbf{d}_{new}, \mathbf{d})$ for all $\mathbf{d} \in \mathcal{D}$
- If novelty score = $\frac{1}{\max_d sim(\mathbf{d}_{new}, \mathbf{d})}$ high, report $\mathbf{d}_{new}$

Problem: Pairwise similarity cannot handle synonymy or polysemy $\Rightarrow$ Utilize **micro-clustering**

Novelty detection with micro-clustering

Idea: Maintain K document clusters $C_1, \ldots, C_K$.

For C_i: $\mathbf{c}_i$ = centroid = **cluster digest**, $fr(w_j|C_i)$ word frequencies, t_i = time stamp when C_i updated

- given $\mathbf{d}_{new}$, determine nearest centroid $\mathbf{c}_i$
- if $(sim(\mathbf{d}_{new}, \mathbf{c}_i) \geq \theta)$
  - add $\mathbf{d}_{new}$ to C_i
  - update $fr(w_j|C_i)$, $\mathbf{c}_i$ (most frequent words) and t_i
- else
  - report $\mathbf{d}_{new}$ as novelty
  - create a new cluster $C = \{\mathbf{d}_{new}\}$ (with new time stamp)
  - remove an old cluster C_i with earliest t_i

5. Extra: Word embeddings

Idea: Present words as numerical vectors such that distances reflect semantic similarity

2D presentation of embeddings:

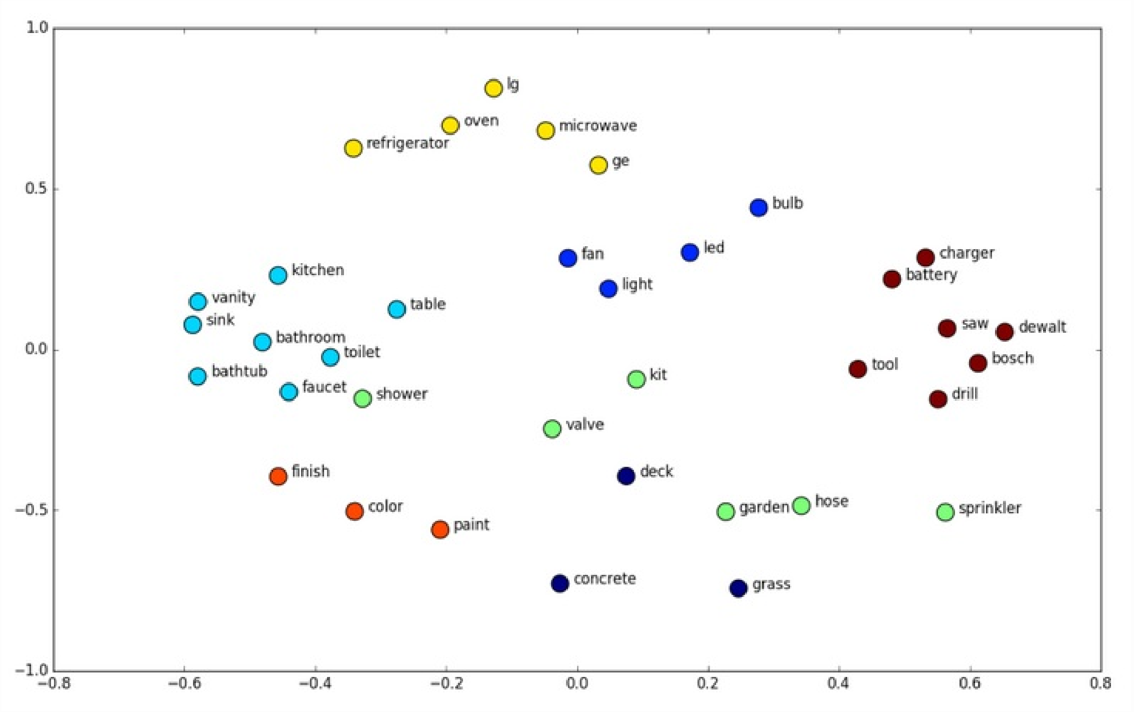
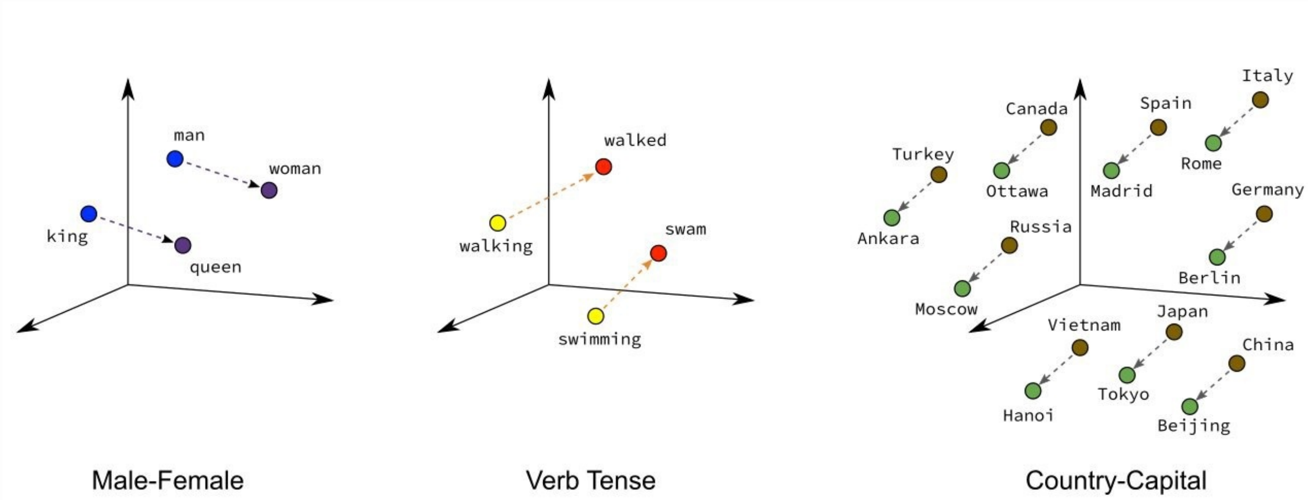


image source Barla (2021)

Word embedding examples



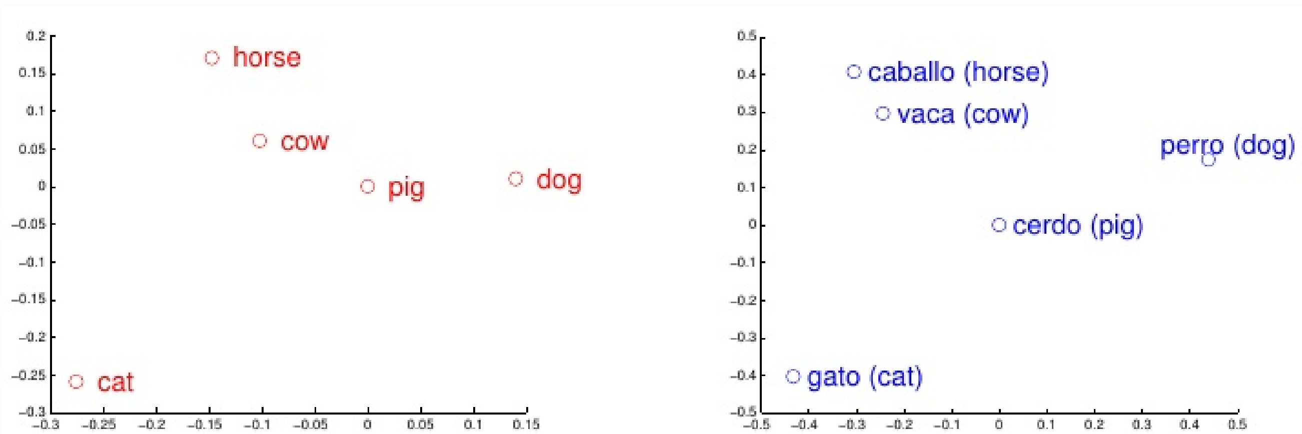


image sources Zhang (2020) and Mikolov et al. (2013)

Word embeddings: popular approaches

1. **Matrix factorization**: typically truncated SVD of word co-occurrence (or term-doc) matrix

2. **Word2vec models**: learn a shallow neural network and use its weights

   a) **CBOW**: predict target word given context words

   b) **Skip-gram**: predict context words given target word

   – good embeddings require large training sets
   but models can be transfered to other similar contexts

Mikolov et al. (2013)

Word embeddings: popular approaches

3. **GloVe**: learn embedding vectors $\mathbf{x}_i$, $\tilde{\mathbf{x}}_i$, such that $\mathbf{x}_i \cdot \tilde{\mathbf{x}}_j \propto \log(P(w_j|w_i))$

- $\tilde{\mathbf{x}}_i$ context embedding $\rightarrow$ later combine with $\mathbf{x}_i$
- \+ fast to learn
- \+ requires less data

Pennington et al. (2014)

Note: *Hype* and *best* are not synonyms. Test always simpler presentations (bag-of-words), too!

Summary

- text presented in vector space (as numerical vectors)
  - simplest: document=bag-of-words with binary occurence, frequencies, or *tfidf* of words
- preprocessing important $\rightarrow$ features. Be careful!
- Goal: try to capture important (mid-frequency) words and phrases
  - prune out stopwords, stemming/lemmatization, detect collocations/n-grams, maybe spell-checking
- dimension reduction often useful (+ LSA helps with synonymy/polysemy)
- clustering: K-representatives + modifications, spectral, hierarchical, co-clustering, ...

References & image sources

- Barla (2021): The ultimate guide to word embeddings `https://neptune.ai/blog/word-embeddings-guide`
- Cutting et al. (1992): Scatter/Gather: A cluster-based approach to browsing large document collections. ACM SIGIR Conference, pp. 318-329.
- Mikolov et al. (2013): Efficient estimation of word representations in vector space. ICLR and arXiv:1301.3781
- Mikolov et al. (2013): Exploiting similarities among languages for machine translation. arXiv:1309.4168

References & image sources

- Pennington et al. (2014): GloVe: Global vectors for word representation. Empirical Methods in Natural Language Processing (EMNLP).
- Perunicic (2017): How are principal component analysis and singular value decomposition related? https://intoli.com/blog/pca-and-svd/
- Zhang (2020): Legal applications of neural word embeddings https://towardsdatascience.com/legal-applications-of-neural-word-embeddings-556b7515012f