Information Storage and Retrieval

CSCE 670
Texas A&M University
Department of Computer Science & Engineering
Instructor: Prof. James Caverlee

Word Embeddings
13 April 2017

How can we more robustly match a user's search intent?

We want to **understand** the query, not just do String equals()

- If user searches for [Dell notebook battery size], we would like to match documents discussing "Dell laptop battery capacity"
- If user searches for [Seattle motel], we would like to match documents containing "Seattle hotel"

A naïve information retrieval system does nothing to help

Simple facilities that we have already discussed do a bit to help

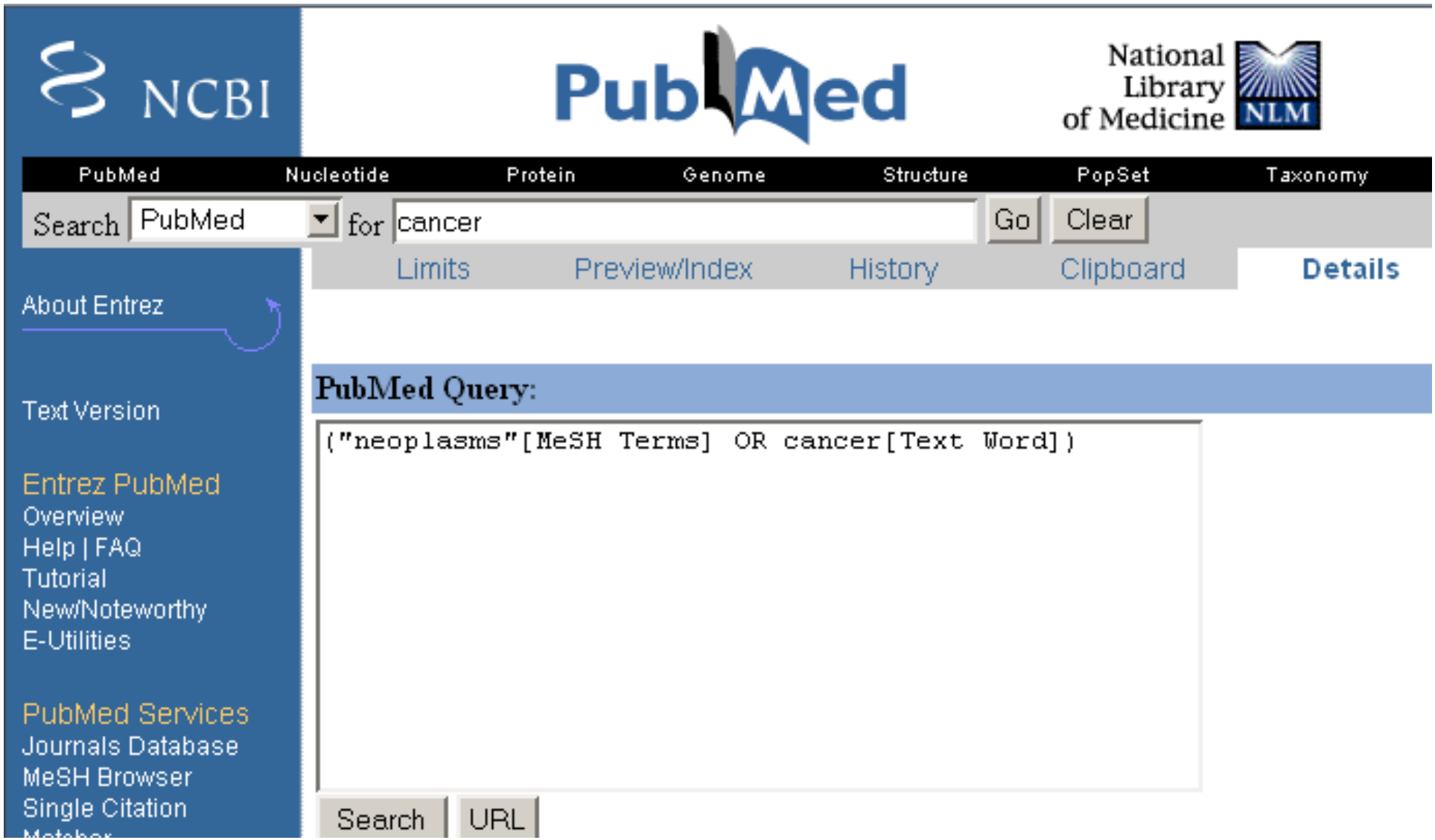
- Spelling correction
- Stemming / case folding

But we'd like to better **understand** when query/document match

How can we more robustly match a user's search intent?

- Use of **anchor text** may solve this by providing human authored synonyms, but not for new or less popular web pages, or non-hyperlinked collections
- **Relevance feedback** could allow us to capture this if we get near enough to matching documents with these words
- We can also fix this with information on **word similarities**:
  - A manual **thesaurus** of synonyms
  - A **measure of word similarity**
    - Calculated from a big document collection
    - Calculated by query log mining (common on the web)

Example of manual thesaurus



Thesaurus-based query expansion

- For each term *t* in a query, expand the query with synonyms and related words of *t* from the thesaurus
  - feline → feline cat
- May weight added terms less than original query terms.
- Generally increases recall
- Widely used in many science/engineering fields
- May significantly decrease precision, particularly with ambiguous terms.
  - "interest rate" → "interest rate fascinate evaluate"
- There is a high cost of manually producing a thesaurus
  - And for updating it for scientific changes

Search log query expansion

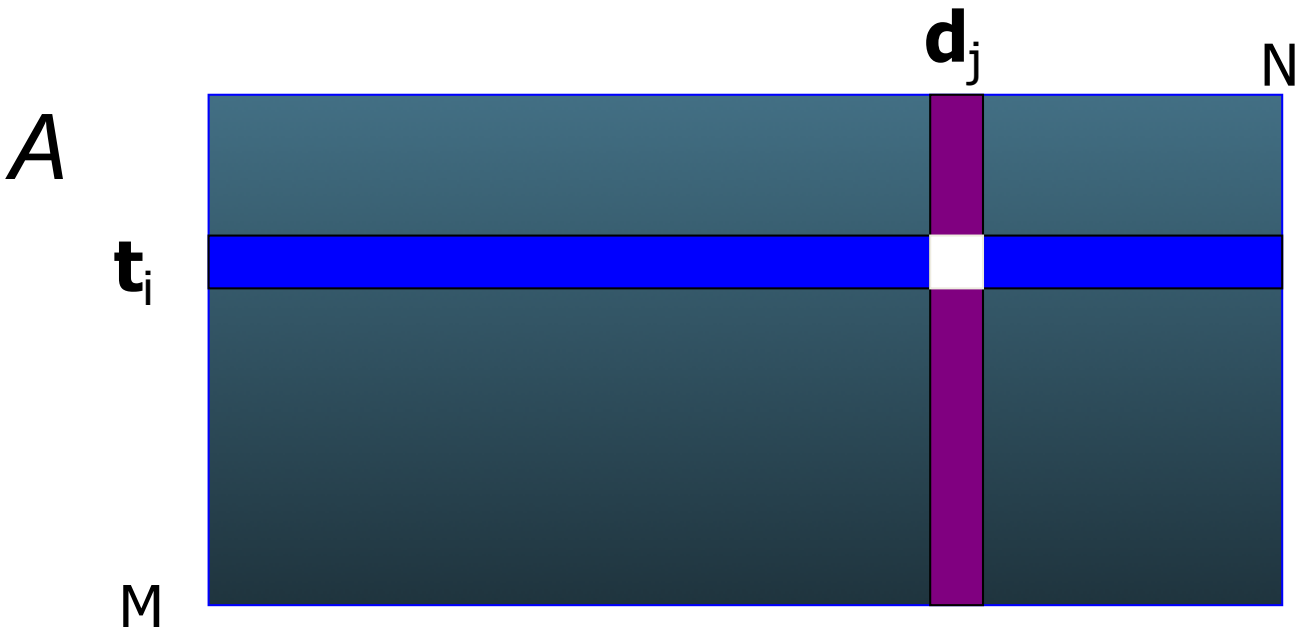
- Context-free query expansion ends up problematic
  - [light hair] ≈ [fair hair] At least in U.K./Australia? ≈ blonde
    - So expand [light] ⇒ [light fair]
  - But [bed light price] ≠ [bed fair price]
- You can learn query context-specific rewritings from search logs by attempting to identify the same user making a second attempt at the same user need
  - [Hinton word vector]
  - [Hinton word embedding]
- In this context, [vector] ≈ [embedding]
  - But not when talking about a *disease vector* or C++!

Automatic Thesaurus Generation

- Attempt to generate a thesaurus automatically by analyzing a collection of documents
- Fundamental notion: similarity between two words
- Definition 1: Two words are similar if they co-occur with similar words.
- Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
- You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Co-occurrence based is more robust, grammatical relations are more accurate.

Co-occurrence Thesaurus

- Simplest way to compute one is based on term-term similarities in $C = AA^T$ where A is term-document matrix.
- $w_{i,j}$ = (normalized) weight for $(t_i, \boldsymbol{d}_j)$



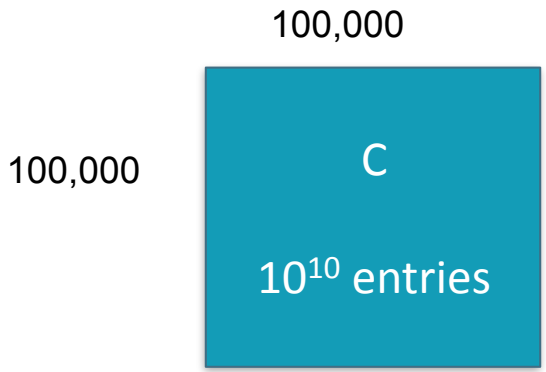
What does C contain if A is a term-doc incidence (0/1) matrix?

Automatic thesaurus generation example

Word	Nearest neighbors
absolutely	absurd, whatsoever, totally, exactly, nothing
bottomed	dip, copper, drops, topped, slide, trimmed
captivating	shimmer, stunningly, superbly, plucky, witty
doghouse	dog, porch, crawling, beside, downstairs
makeup	repellent, lotion, glossy, sunscreen, skin, gel
mediating	reconciliation, negotiate, cease, conciliation
keeping	hoping, bring, wiping, could, some, would
lithographs	drawings, Picasso, Dali, sculptures, Gauguin
pathogens	toxins, bacteria, organisms, bacterial, parasites
senses	grasp, psyche, truly, clumsy, naïve, innate

Automatic Thesaurus Generation Issues

- Quality of associations is usually a problem
- Sparsity



- Term ambiguity may introduce irrelevant statistically correlated terms.
  - "planet earth facts" → "planet earth soil ground facts"
- Since terms are highly correlated anyway, expansion may not retrieve many additional documents.

One idea: use search engine to help!

A Web-based Kernel Function for Measuring the Similarity of Short Text Snippets

http://robotics.stanford.edu/users/sahami/papers-dir/www2006.pdf

1. Issue x as a query to a search engine S.

2. Let $R(x)$ be the set of (at most) n retrieved documents $d_1, d_2, \ldots, d_n$

3. Compute the TFIDF term vector v_i for each document $d_i \in R(x)$

4. Truncate each vector v_i to include its m highest weighted terms

5. Let $C(x)$ be the centroid of the L_2 normalized vectors v_i:

$$C(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{v_i}{\|v_i\|_2}$$

6. Let $QE(x)$ be the L_2 normalization of the centroid $C(x)$:

$$QE(x) = \frac{C(x)}{\|C(x)\|_2}$$

Finally, given that we have a means for computing the query expansion for a short text, it is a simple matter to define the semantic kernel function K as the inner product of the query expansions for two text snippets. More formally, given two short text snippets x and y, we define the semantic similarity kernel between them as:

$$K(x, y) = QE(x) \cdot QE(y).$$

Text 1	Text 2	Kernel	Cosine	Set Overlap
Acronyms				
support vector machine	SVM	0.812	0.0	0.110
portable document format	PDF	0.732	0.0	0.060
artificial intelligence	AI	0.831	0.0	0.255
artificial insemination	AI	0.391	0.0	0.000
term frequency inverse document frequency	tf idf	0.831	0.0	0.125
term frequency inverse document frequency	tfidf	0.507	0.0	0.060
Individuals and their positions				
UN Secretary-General	Kofi Annan	0.825	0.0	0.065
UN Secretary-General	George W. Bush	0.110	0.0	0.000
US President	George W. Bush	0.688	0.0	0.045
Microsoft CEO	Steve Ballmer	0.838	0.0	0.090
Microsoft CEO	Bill Gates	0.317	0.0	0.000
Microsoft Founder	Bill Gates	0.677	0.0	0.010
Google CEO	Eric Schmidt	0.845	0.0	0.105
Google CEO	Larry Page	0.450	0.0	0.040
Google Founder	Larry Page	0.770	0.0	0.050
Microsoft Founder	Larry Page	0.189	0.0	0.000
Google Founder	Bill Gates	0.096	0.0	0.000
web page	Larry Page	0.123	0.5	0.000

Multi-faceted terms				
space exploration	NASA	0.691	0.0	0.070
space exploration	space travel	0.592	0.5	0.005
vacation travel	space travel	0.321	0.5	0.000
machine learning	ICML	0.586	0.0	0.065
machine learning	machine tooling	0.197	0.5	0.000
graphical UI	graphical models	0.275	0.5	0.000
graphical UI	graphical interface	0.643	0.5	0.000
java island	Indonesia	0.454	0.0	0.000
java programming	Indonesia	0.020	0.0	0.000
java programming	applet development	0.563	0.0	0.010
java island	java programming	0.280	0.5	0.000

Can you directly learn term relations?

- Basic IR is scoring on $q^T d$
- No treatment of synonyms; no machine learning
- Can we learn parameters W to rank via $q^T W d$
- Problem is again sparsity – W is huge > 10^{10}

How can we represent term relations?

- With the standard symbolic encoding of terms, each term is a dimension
- Different terms have no inherent similarity
- motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]^T
  hotel [0 0 0 0 0 0 0 3 0 0 0 0 0 0 0] = 0
- If query on *hotel* and document has *motel*, then our query and document vectors are **orthogonal**

Is there a better way?

- Idea:
  - Can we learn a low dimensional representation of a word in $\mathbb{R}^d$ such that dot products $u^T v$ express word similarity?
  - We could still if we want to include a "translation" matrix between vocabularies (e.g., cross-language): $u^T W v$
    - But now W is small!
  - Supervised Semantic Indexing (Bai et al. *Journal of Information Retrieval* 2009) shows successful use of learning W for information retrieval
- But we'll develop direct similarity in this class

Distributional similarity based representations

- You can get a lot of value by representing a word by means of its neighbors
- "You shall know a word by the company it keeps"
- (J. R. Firth 1957: 11)
- One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in
saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Solution: Low dimensional vectors

- The number of topics that people talk about is small (in some sense)
  - Clothes, movies, politics, …
- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25 – 1000 dimensions

- How to reduce the dimensionality?
  - Go from big, sparse co-occurrence count vector to low dimensional "word embedding"

Traditional Way: Latent Semantic Indexing/Analysis

- Use Singular Value Decomposition (SVD) – kind of like Principal Components Analysis (PCA) for an arbitrary rectangular matrix – or just random projection to find a low-dimensional basis
- Theory is that similarity is preserved as much as possible
- Weakly, you can actually gain in IR by doing LSA as "noise" of term variation gets replaced by semantic "concepts"
- Popular in the 1990s [Deerwester et al. 1990, etc.]
  - Results were always somewhat iffy (… it worked sometimes)
  - Harder to implement efficiently in an IR system (dense vectors!)

Recall SVD on user-item matrix, but consider now term-document matrix

SVD - Definition

$$A_{[m \times n]} = U_{[m \times r]} \Sigma_{[r \times r]} (V_{[n \times r]})^T$$

- **A**: **Input data matrix**
  - *m* x *n* matrix (e.g., *m* documents, *n* terms)
- **U**: **Left singular vectors**
  - *m* x *r* matrix (*m* documents, *r* concepts)
- Σ: **Singular values**
  - *r* x *r* diagonal matrix (strength of each 'concept') (*r* : rank of the matrix **A**)
- **V**: **Right singular vectors**
  - *n* x *r* matrix (*n* terms, *r* concepts)

SVD - Properties

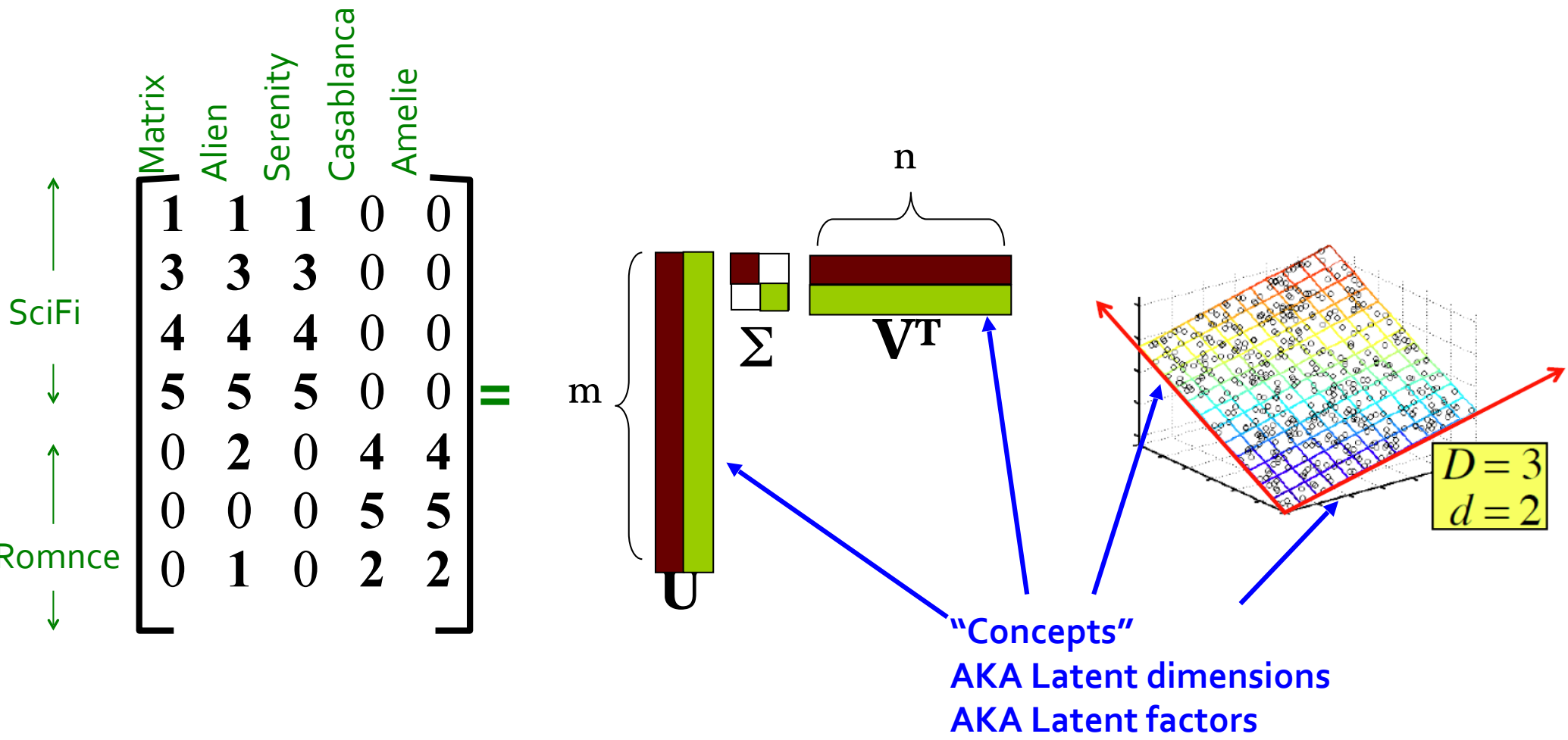
It is **always** possible to decompose a real matrix A into $A = U \Sigma V^T$, where

- U, Σ, V: unique
- U, V: column orthonormal
  - $U^T U = I; V^T V = I$ (I: identity matrix)
  - (Columns are orthogonal unit vectors)
- Σ: diagonal
  - Entries (**singular values**) are positive, and sorted in decreasing order ($\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$)

Nice proof of uniqueness: http://www.mpi-inf.mpg.de/~bast/ir-seminar-ws04/lecture2.pdf

SVD – Example: Users-to-Movies

- **$A = U \Sigma V^T$ - example: Users to Movies**



SVD – Example: Users-to-Movies

- **A = U Σ V^T - example: Users to Movies**

	Matrix	Alien	Serenity	Casablanca	Amelie
SciFi	1	1	1	0	0
	3	3	3	0	0
	4	4	4	0	0
	5	5	5	0	0
Romnce	0	2	0	4	4
	0	0	0	5	5
	0	1	0	2	2

=

0.13	0.02	-0.01
0.41	0.07	-0.03
0.55	0.09	-0.04
0.68	0.11	-0.05
0.15	**-0.59**	**0.65**
0.07	**-0.73**	**-0.67**
0.07	**-0.29**	**0.32**

x

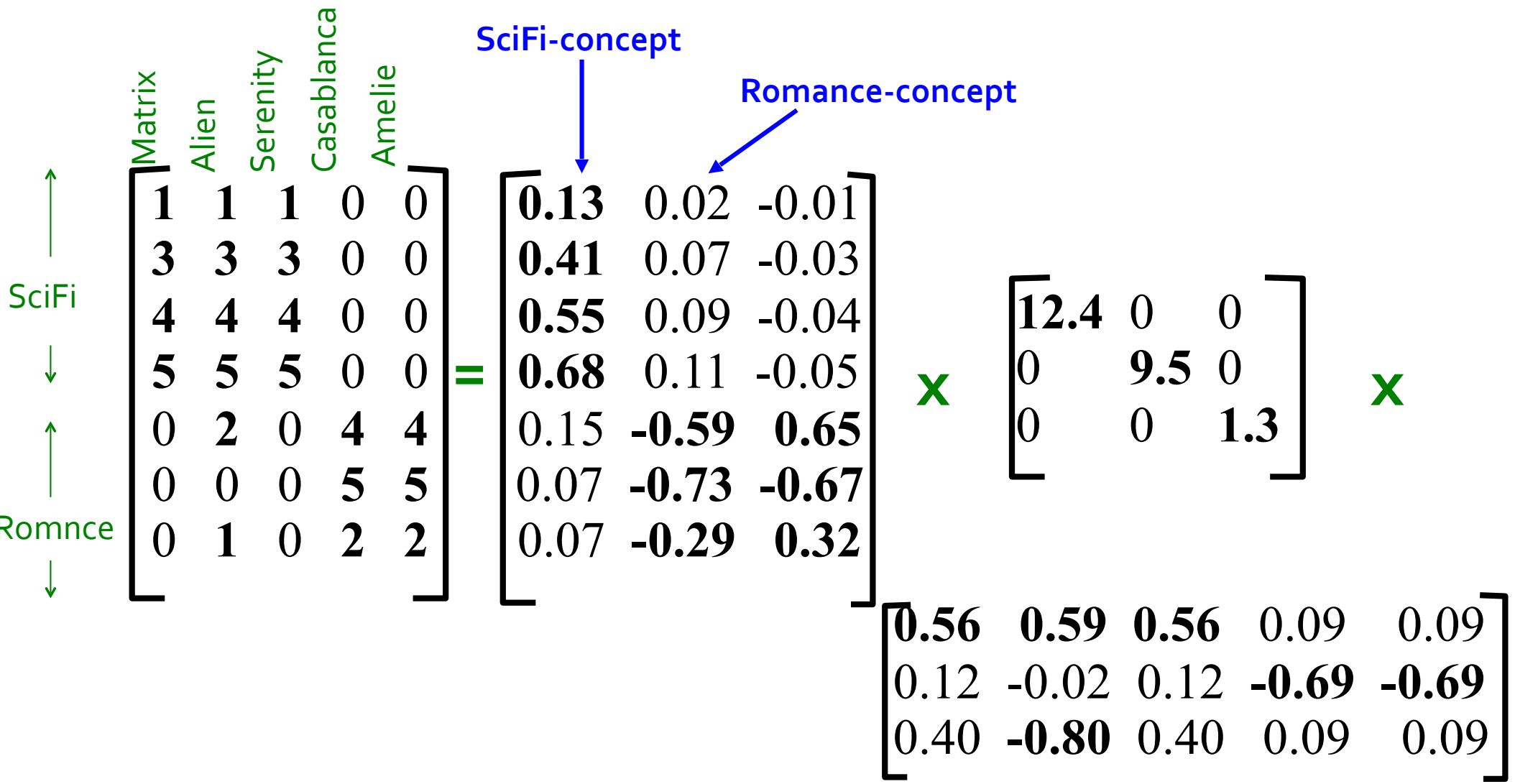
12.4	0	0
0	**9.5**	0
0	0	**1.3**

x

0.56	**0.59**	**0.56**	0.09	0.09
0.12	-0.02	0.12	**-0.69**	**-0.69**
0.40	**-0.80**	0.40	0.09	0.09

SVD – Example: Users-to-Movies

- **A = U Σ V^T - example: Users to Movies**



Today's hotness: word2vec

See: Manning and Socher: Natural Language Processing with Deep Learning, Stanford CS224N/Ling284

Word meaning is defined in terms of vectors

We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

… those other words also being represented by vectors … it all gets a bit recursive

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

2. Main idea of word2vec

Predict between every word and its context words!

Two algorithms

1. **Skip-grams (SG)**

   Predict context words given target (position independent)

2. Continuous Bag of Words (CBOW)

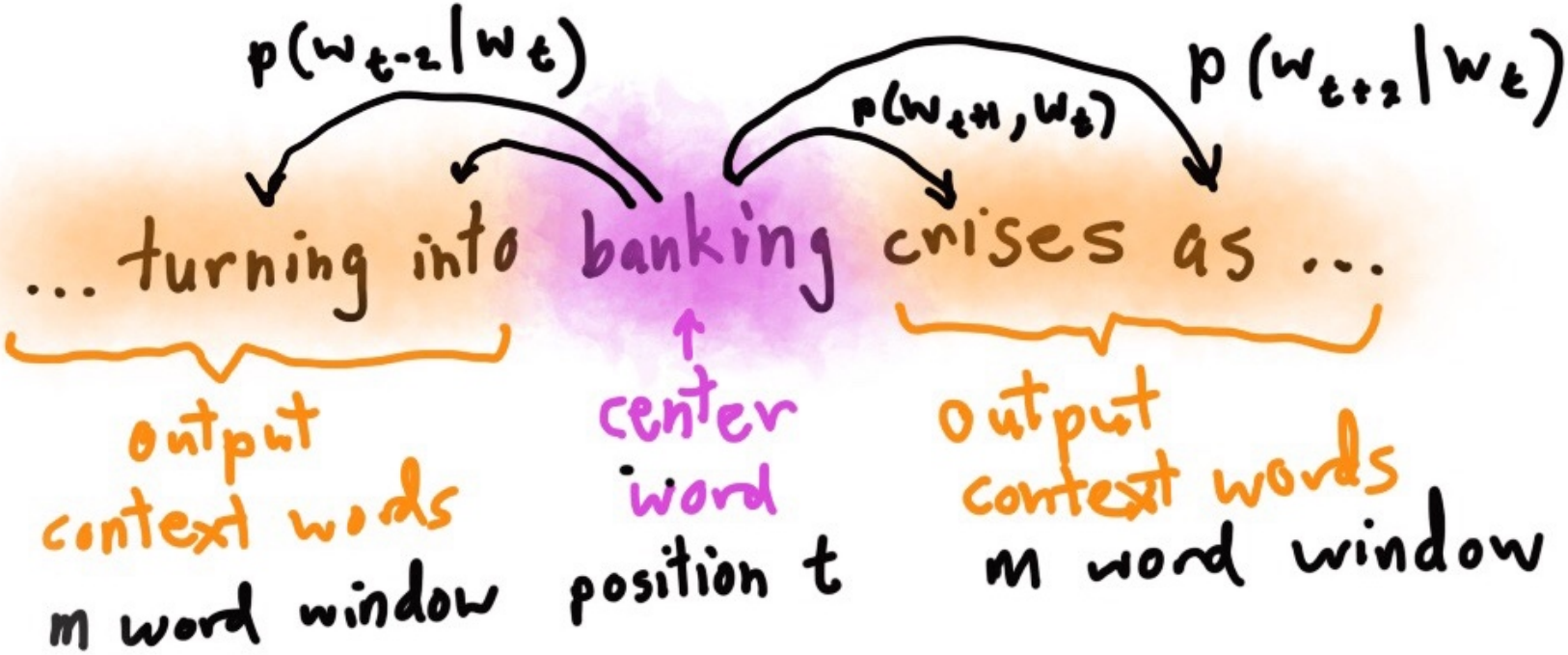
   Predict target word from bag-of-words context

Two (moderately efficient) training methods

1. Hierarchical softmax
2. Negative sampling

Naïve softmax

Skip-gram prediction



Details of word2vec

For each word $t = 1 \ldots T$, predict surrounding words in a window of "radius" m of every word.

Objective function: Maximize the probability of any context word given the current center word:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w_{t+j} | w_t ; \theta)$$

Negative Log Likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log p(w_{t+j} | w_t)$$

Where θ represents all variables we will optimize

The objective function – details

- Terminology: Loss function = cost function = objective function
- Usual loss for probability distribution: Cross-entropy loss
- With one-hot w_{t+j} target, the only term left is the negative log probability of the true class
- More on this later…

Details of Word2Vec

Predict surrounding words in a window of radius *m* of every word

For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

where *o* is the outside (or output) word index, *c* is the center word index, v_c and u_o are "center" and "outside" vectors of indices *c* and *o*

Softmax using word *c* to obtain probability of word *o*

Notation for Skip-Gram Model:

- w_i: Word i from vocabulary V
- $\mathcal{V} \in \mathbb{R}^{n \times |V|}$: Input word matrix
- v_i: i-th column of $\mathcal{V}$, the input vector representation of word w_i
- $\mathcal{U} \in \mathbb{R}^{n \times |V|}$: Output word matrix
- u_i: i-th row of $\mathcal{U}$, the output vector representation of word w_i

We breakdown the way this model works in these 6 steps:

1. We generate our one hot input vector $x \in \mathbb{R}^{|V|}$ of the center word.
2. We get our embedded word vector for the center word $v_c = \mathcal{V}x \in \mathbb{R}^n$
3. Generate a score vector $z = \mathcal{U}v_c$.
4. Turn the score vector into probabilities, $\hat{y} = \text{softmax}(z)$. Note that $\hat{y}_{c-m}, \ldots, \hat{y}_{c-1}, \hat{y}_{c+1}, \ldots, \hat{y}_{c+m}$ are the probabilities of observing each context word.
5. We desire our probability vector generated to match the true probabilities which is $y^{(c-m)}, \ldots, y^{(c-1)}, y^{(c+1)}, \ldots, y^{(c+m)}$, the one hot vectors of the actual output.