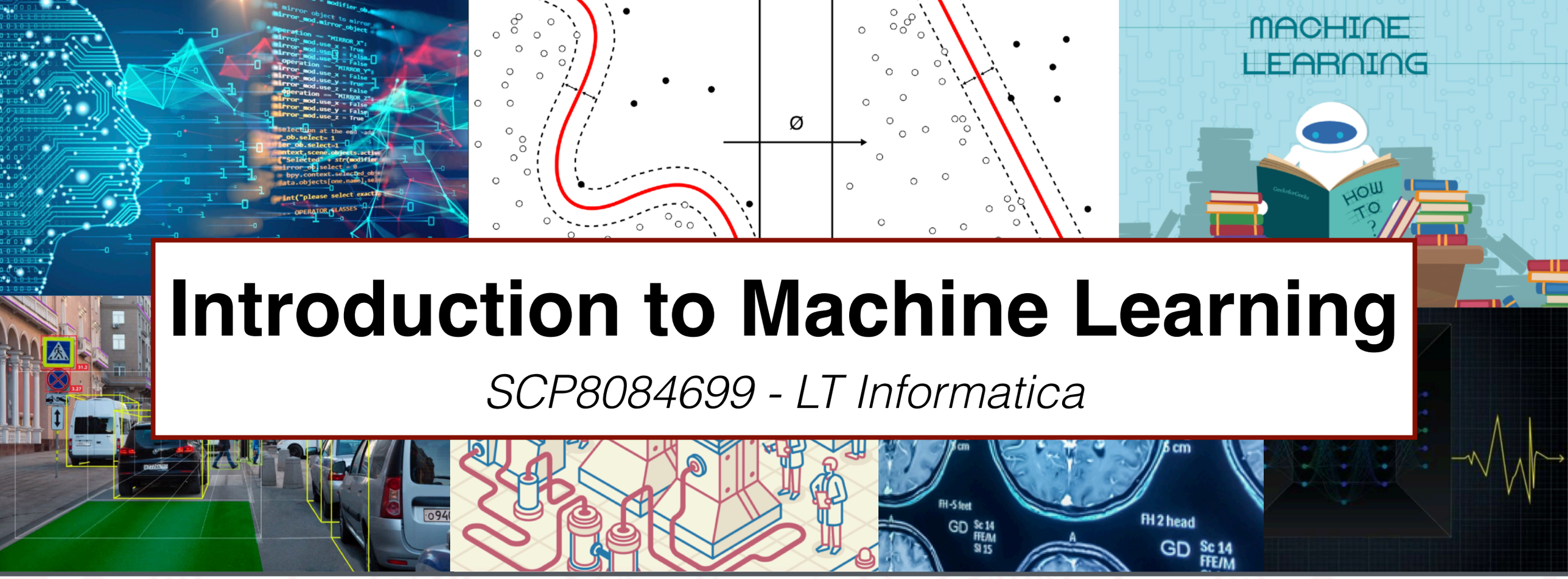


Introduction to Machine Learning

SCP8084699 - LT Informatica

Probabilistic models & NLP (part 2)

Prof. Lamberto Ballan

University of Padova, A.Y. 2021/22 May 19, 2022

DIPARTIMENTO MATEMATICA
Department of Mathematics “Tullio Levi-Civita”

Course calendar: update

- Next lectures:

L12	W12	Tuesday, 17 May 2022	Probabilistic models, NLP & information retrieval
L13	W12	Thursday, 19 May 2022	Probabilistic models, NLP & information retrieval
S1	W13	Tuesday, 24 May 2022	*Panel: Mathematical challenges in an AI driven world*
	W13	Thursday, 26 May 2022	
L14	W14	Tuesday, 31 May 2022	ML & CV: Teaching machines to see
	W14	Thursday, 2 June 2022	

- Important note: the panel (on May 24th) will be at 16:00; you can attend in either in person (room 2AB40) or virtually (live-streamed on Youtube)

Recap: *n*-grams and chain rule

- Goal: compute the probability of a sentence (words sequence): $P(\mathrm{W}) = P(w_1, w_2, w_3, \ldots, w_n)$

- A **language model** is a model that computes either of these: $P(\mathrm{W})$ or $P(w_n \mid w_1, w_2, w_3, \ldots, w_{n-1})$

- The chain rule applied to compute joint probability of words: $P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$

  - An example:

$$P(\text{“its water is so transparent that”}) =$$

$$= P(\text{“its”})\, P(\text{“water} \mid \text{its”})\, P(\text{“is} \mid \text{its water”})\, P(\text{“so} \mid \text{its water is”})$$
$$P(\text{“transparent} \mid \text{its water is so”})\, P(\text{“that} \mid \text{its water is so transparent”})$$

Recap: *n*-grams and chain rule

- How to estimate these probabilities? Could we just count?

$$P(\text{“the | its water is so transparent that”}) = \frac{Count(\text{“its water is so transparent that the”})}{Count(\text{“its water is so transparent that”})}$$

  - No! Too many sentences… we’ll never see enough data to estimate these probabilities

Recap: *n*-grams and chain rule

- Simplifying assumption (Markov assumption):

$$P(\text{“the | its water is so transparent that”}) \approx P(\text{“the | that”})$$

- Or maybe:

$$P(\text{“the | its water is so transparent that”}) \approx P(\text{“the | transparent that”})$$

- In other words, these are two examples of n-gram models
  - The first example is a bigram, i.e. conditioned on the previous word
  - The second example is a trigram, which is defined as:

$$P(w_1 w_2 \dots w_N) = \prod_{i=1}^{N} P(w_i | w_{1:i-1}) = \prod_{i=1}^{N} P(w_i | w_{i-2:i-1})$$

Estimating *n*-gram probabilities

- The Maximum Likelihood Estimate

$$P(w_i|w_{i-1}) = \frac{Count(w_i, w_{i-1})}{Count(w_{i-1})}$$

- Let's see a simple example on estimating bigram probabilities:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P(\texttt{I}\,|\,\texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam}\,|\,\texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am}\,|\,\texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>}\,|\,\texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam}\,|\,\texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do}\,|\,\texttt{I}) = \tfrac{1}{3} = .33$$

Estimating *n*-gram probabilities

- More examples: Berkeley Restaurant Project (BeRP) sentences
  - "tell me about chez panisse"
  - "mid priced thai food is what i'm looking for"
  - "i'm looking for a good place to eat breakfast"
  - "can you tell me about any good cantonese restaurants close by"
  - "when is caffe venezia open during the day"
  - … (approx. 9,200 sentences)

D. Jurafsky, C. Wooters, G. Tajchman, J. Segal, A. Stolcke, E. Fosler, N. Morgan, "The Berkeley Restaurant Project", *ICSLP 1994*

Estimating *n*-gram probabilities

- Raw bigram counts (out of ~9,200 sentences):

	i	want	to	eat	chinese	food	lunch	Spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Estimating *n*-gram probabilities

- Raw **bigram probabilities**: normalize by unigrams
  - Raw unigram counts:

i	want	to	eat	chinese	food	lunch	spend
2553	927	2417	746	158	1093	341	278

  - Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.0008
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.0008	0	0.0017	0.28	0.0008	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.0009	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Estimating *n*-gram probabilities

- Bigram estimates of sentence probabilities
  - An example:

$$P(\text{<s> I want English food </s>}) =$$

$$P(\text{I} \mid \text{<s>})$$

$$\cdot\ P(\text{want} \mid \text{I})$$

$$\cdot\ P(\text{English} \mid \text{want})$$

$$\cdot\ P(\text{food} \mid \text{English})$$

$$\cdot\ P(\text{</s>} \mid \text{food})$$

$$= 0.000031$$

Estimating *n*-gram probabilities

- What kinds of knowledge?

$P(\text{I} \mid \langle s \rangle) = 0.25$

$P(\text{English} \mid \text{want}) = 0.0011$

$P(\text{Chinese} \mid \text{want}) = 0.0065$

→ *Chinese food is more popular*

$P(\text{to} \mid \text{want}) = 0.66$

$P(\text{eat} \mid \text{to}) = 0.28$

→ *Grammar (infinitive)*

$P(\text{food} \mid \text{to}) = 0$

→ *Contingency: unobserved in the corpus*

$P(\text{want} \mid \text{spend}) = 0$

→ *Grammatical mistake*

Text classification

- **Text classification**: given a text/document, decide which of a predefined set of classes it belongs to
  - Language identification, genre classification, …
  - Spam detection (a binary classifier: spam *vs* "ham"), e.g.:

Spam: Wholesale Fashion Watches -57% today. Designer watches for cheap ...
Spam: You can buy ViagraFr$1.85 All Medications at unbeatable prices! ...
Spam: WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US ...
Spam: Sta.rt earn*ing the salary yo,u d-eserve by o'btaining the prope,r crede'ntials!

Ham: The practical significance of hypertree width in identifying more ...
Ham: Abstract: We will motivate the problem of social identity clustering: ...
Ham: Good to see you my friend. Hey Peter, It was good to hear from you. ...
Ham: PDS implies convexity of the resulting optimization problem (Kernel Ridge ...

n-grams such as "for cheap" and "you can buy" seem to be indicators of spam (although their ham prob would be nonzero)

Text classification

- We can have two different ways of tackling text classification:
  - Using hand-coded classification **rules**
  - Using **language modeling** and **machine learning**

- In the **rule-based** approach:
  - We define rules based on a combination of words (or other features)
    - Spam: black-list-addresses OR (“dollars” AND “have been selected”)
  - If rules are well refined by expert, accuracy can be high
  - Building such rules is expensive and not always possible

Text classification

- In the **language modeling** approach we define:
  - A *n*-gram model on the spam folder: $\mathbf{P}(Message \mid spam)$
  - A *n*-gram model on the inbox folder: $\mathbf{P}(Message \mid ham)$
  - Then we classify a new message using the Bayes' rule:

$$\underset{c \in \{spam, ham\}}{\operatorname{argmax}} P(c \mid message) = \underset{c \in \{spam, ham\}}{\operatorname{argmax}} P(message \mid c)\, P(c)$$

where $P(c)$ is estimated by counting the total number of spam/ham

- **Naive Bayes**: a simple classification approach based on just the Bayes rule

Naive Bayes classifier

- Bayes rule applied to a document d and a class c:

$$P(c|d) = \frac{P(d|c)P(c)}{p(d)}$$

- Let's derive the intuition for the **Naive Bayes classifier**:

$$c_{MAP} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{p(d)} = \arg\max_{c \in C} P(d|c)P(c)$$

MAP is maximum a posteriori, i.e. most likely class

Bayes rule

Dropping the denominator

Naive Bayes classifier

- Let's say that d can be represented using n features x_i
- The previous equation can be rewritten as:

$$c_{MAP} = \arg\max_{c \in C} P(d|c)P(c) = \arg\max_{c \in C} P(x_1, x_2, \ldots, x_n|c)P(c)$$

Likelihood *Prior*

- How do I compute these probabilities?

$$c_{MAP} = \arg\max_{c \in C} P(x_1, x_2, \ldots, x_n|c)P(c)$$

This is hard: could be estimated only if a very large number of training example is available.

Q: how often does this class occur?

A: we can just count the relative frequencies in a corpus.

Naive Bayes classifier

- We make some simplifying assumptions in order to compute the probabilities $P(x_1, x_2, \ldots, x_n|c)$
- **Conditional Independence**: assume the feature probabilities $P(x_i \mid c)$ are independent given the class c

$$P(x_1, x_2, \ldots, x_n|c) = P(x_1|c)P(x_2|c)P(x_3|c)\ldots P(x_n|c)$$

- This gives us the so-called Multinomial Naive Bayes Classifier:

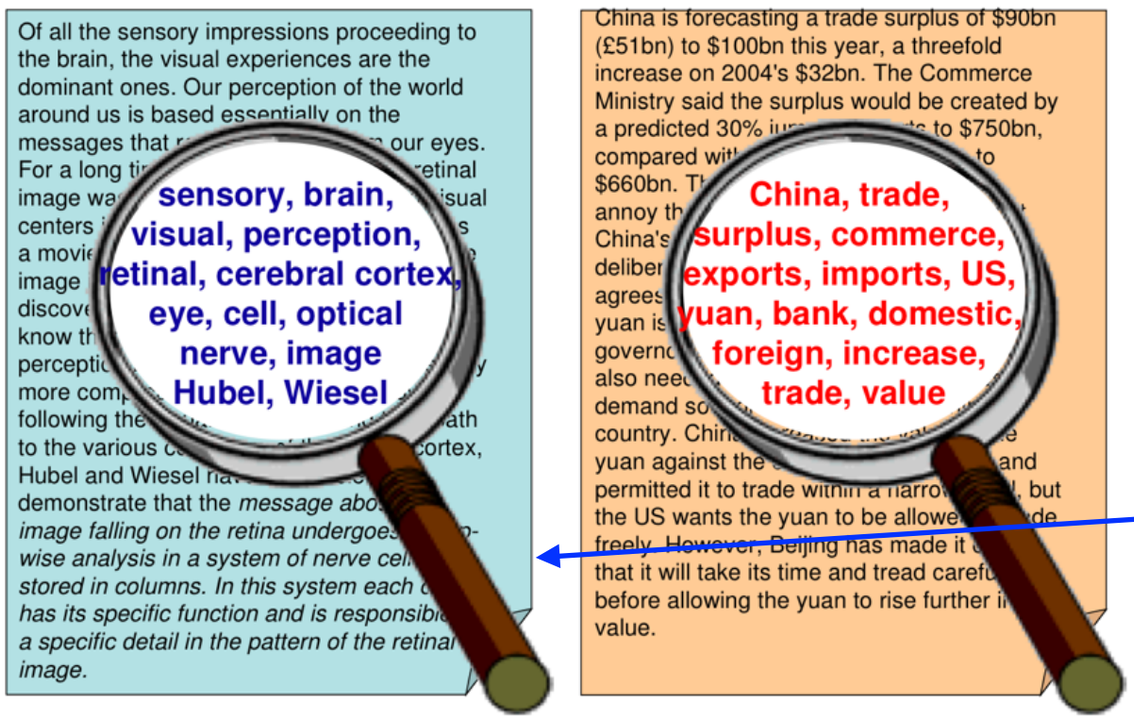
$$c_{MAP} = \arg\max_{c \in C} P(x_1, x_2, \ldots, x_n|c)P(c)$$

$$c_{NB} = \arg\max_{c \in C} P(c) \prod_{x \in X} P(x|c)$$

Text classification and features

- In the **machine learning** approach we represent the message as a feature set (x, y) and use a classifier
  - Language models and Naive Bayes
  - But also others: k-NN, SVM, Logistic Regression, …

- A simple alternative: the **Bag-of-Words** model



Bag-of-words

- The **Bag-of-Words** (or *vector model*) represents the frequency of occurrence of each word
  - A simple example:

D_1 = "John likes to watch movies. Mary likes movies too."
D_2 = "John also likes to watch football."

BoW_1 = {John:1,likes:2,to:1,watch:1,movies:2,Mary:1,too:1}
BoW_2 = {John:1,also:1,likes:1,to:1,watch:1,footbal:1}

  - The feature vectors are large and sparse
  - The notion of order of the words is lost: BoWs & unigrams give the same probability to any permutation of a text
    - Higher order n-grams maintain a local notion of word order
    - But using bigrams the number of features is squared, with trigrams is cubed, etc.

Words as discrete symbols

- In “traditional” NLP we regard words as **discrete symbols**
  - We have already introduced the BoW representation:

D_1 = “John likes to watch movies. Mary likes movies too.”

BoW_1 = {John:1,likes:2,to:1,watch:1,movies:2,Mary:1,too:1}

Assuming a 10-dim vocabulary, this could be represented as a vector such as:

BoW_1 = [0,1,2,0,1,1,0,2,1,0]

  - An even simpler representation is the one-hot vector:

HoT_1 = [0,1,1,0,1,1,0,1,1,0]

  - Vector dimension = number of words in vocabulary (in practice this is super-sparse, e.g. 500,000 words)

Words as discrete symbols

- Representing words as discrete symbols - such as one-hot vectors - has a major problem
  - "San Francisco Hotel": `hotel = [0000000010000…00]`
  - "San Francisco Motel": `motel = [0001000000000…00]`
  - These two vectors are orthogonal, *i.e.* there is no natural notion of **similarity** for one-hot vectors

- Possible solutions:
  - The "knowledge representation way": you could rely on list of synonyms or taxonomies (e.g. WordNet) to get similarity
  - The "machine learning way": learn to encode similarity in the vectors themselves

Words as discrete symbols

- Commonest linguistic way of thinking of **meaning**:

  Signifier (symbol) $\Longleftrightarrow$ Signified (concept or thing) = denotation

- **Knowledge representation**: use taxonomies (e.g. WordNet) to obtain more meaningful word vectors

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
for synset in wn.synsets("good"):
    print "(%s)" % synset.pos(),
    print ", ".join([l.name() for l in synset.lemmas()])
```

```
(adj) full, good
(adj) estimable, good, honorable, respectable
(adj) beneficial, good
(adj) good, just, upright
(adj) adept, expert, good, practiced,
proficient, skillful
(adj) dear, good, near
(adj) good, right, ripe
…
(adv) well, good
(adv) thoroughly, soundly, good
(n) good, goodness
(n) commodity, trade good, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Words as discrete symbols

- Problems with resources like WordNet
  - Great as a resource but missing nuance
    - *E.g. "proficient" is listed as a synonym for "good", but this is only correct in some contexts*
  - Missing new meanings of words
    - *E.g. wicked, badass, nifty, wizard, genius, ninja, …*
    - *Impossible to keep up-to-date!*
  - Subjective
  - Requires human labor to create and adapt
  - It is hard to compute accurate word similarity

Representing words by their context

- Core idea: a word's **meaning** is given by the words that frequently appear close-by
  - When a word w appears in a text, its context is the set of words that appear nearby, within a fixed-size window
  - Use the many contexts of w to build up a representation of w

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent banking

Word vectors

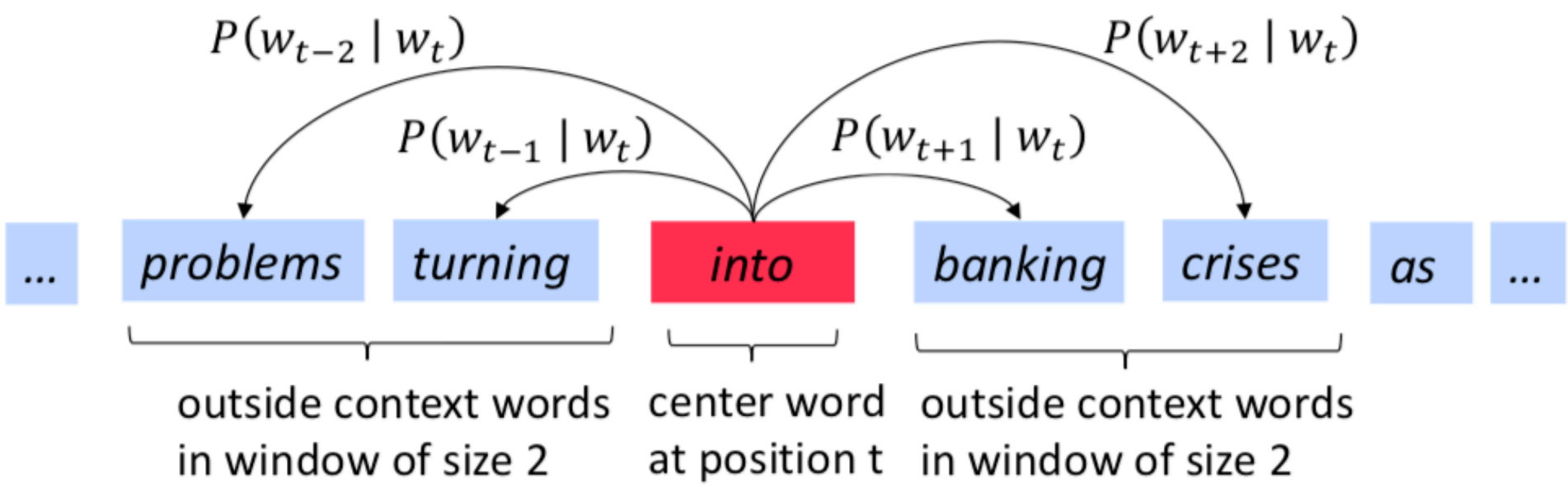
- We build a dense vector for each word, so that it's similar to vectors of words that appear in similar contexts

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

Note: word vectors are often called "word embeddings" (or word representations)

Word2Vec

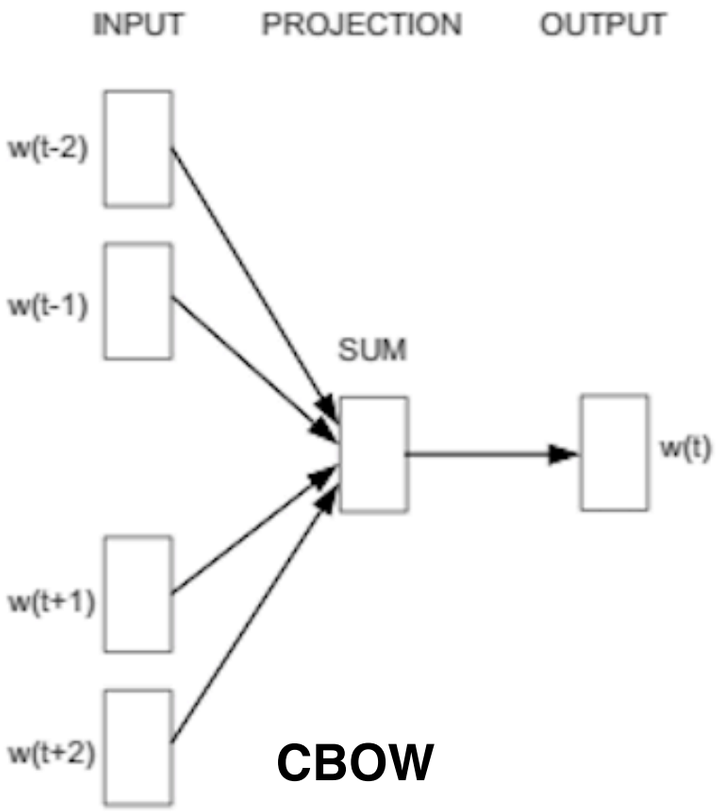
- **Word2Vec** is a framework for learning word vectors
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position t in the text, which has a center word c and context ("outside") words o
  - Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
  - Keep adjusting the word vectors to maximize this probability



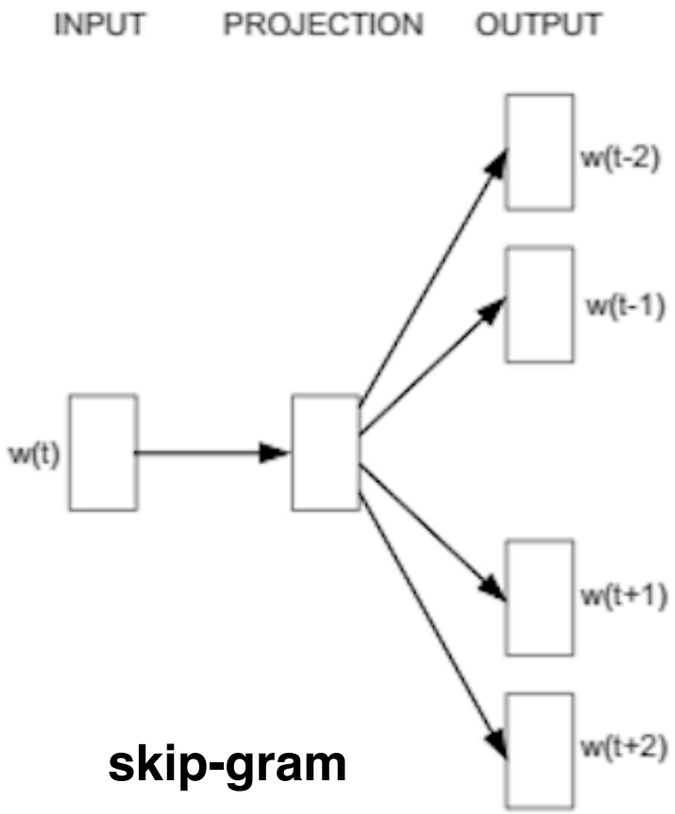
T. Mikolov et al., NIPS, 2013

Word2Vec

- **Word2Vec** is a neural network (an *autoencoder*) which implements this idea to learn word vectors
  - It does so using context to predict a target word (CBOW)
  - Or using a word to predict a target context (skip-gram)



CBOW

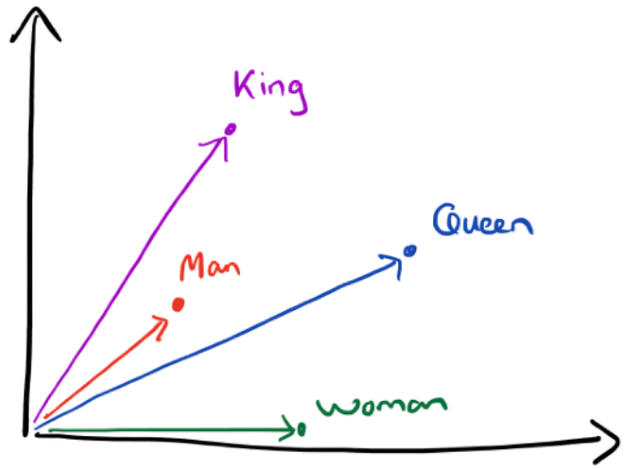


skip-gram

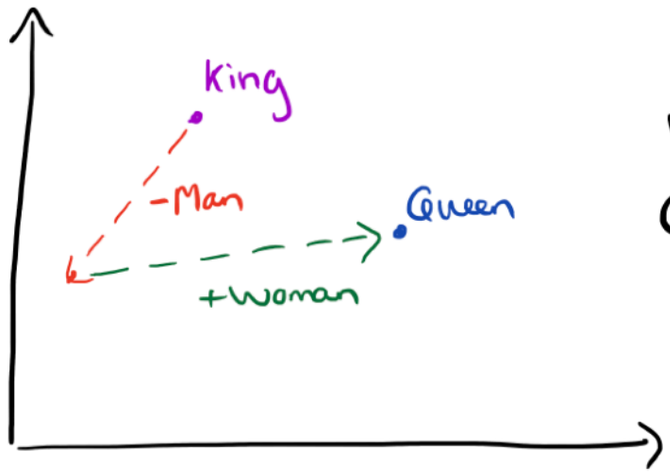
T. Mikolov et al., NIPS, 2013

Word2Vec: vector composition

- The learned word vectors capture meaningful syntactic and semantic regularities in a very simple way

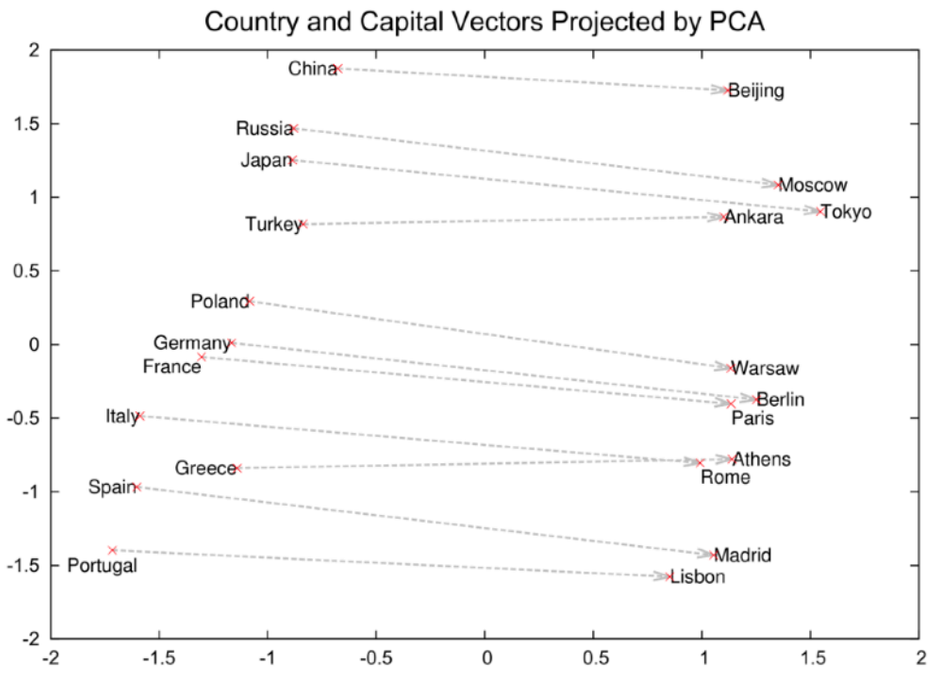


Word Vectors



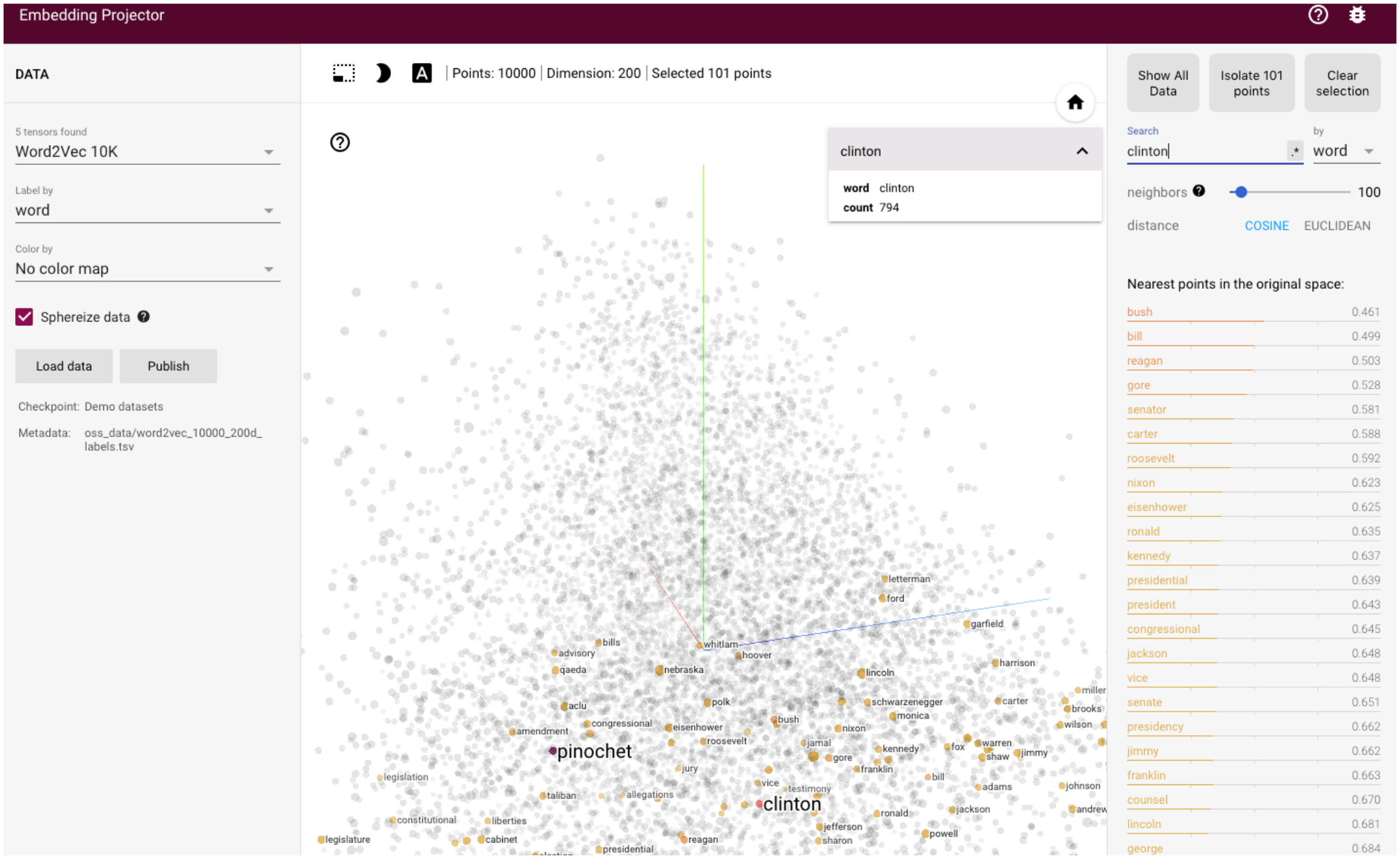
Vector Composition

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza



Word2Vec: demo

- Check the website: https://projector.tensorflow.org/



Words and feature spaces

- We have just seen **word vectors** as an alternative approach to learn text representations from data

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

  - A word like “banking” is represented by a dense vector:

```
"banking" = [0.29,0.79,-0.18,-0.11,0.11,-0.54,0.35,0.27]
```

  - How to represent sentences/documents?
    - Naive approach: average (or max or sum) of word vectors
    - More advanced fusion schemes

Vector space model

- **Vector space model** (or term vector model) is the model in which a text is represented as a vector of terms
  - Basically it's what we have already introduced (i.e. BoW)
  - But the emphasis here is on its geometrical interpretation
  - Documents are represented as:

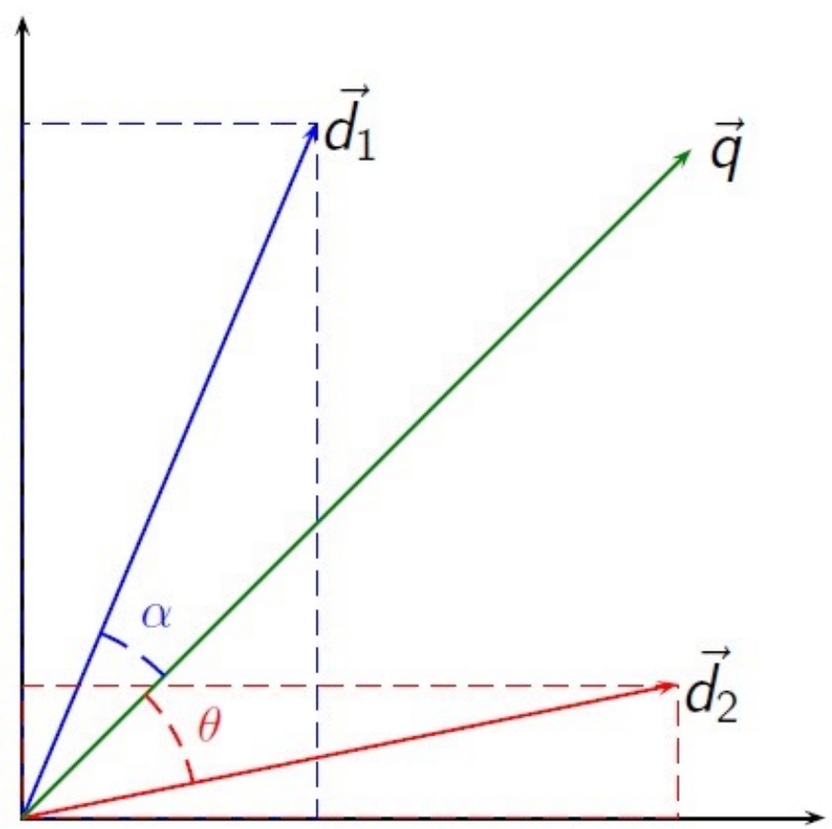
$$\mathbf{d}_j = (w_{1,j}, w_{2,j}, \ldots, w_{v,j})$$

  - Now we have a new document:

$$\mathbf{q} = (q_{1,j}, q_{2,j}, \ldots, q_{v,j})$$

  - We compute the similarity between $\mathbf{q}$ and $\mathbf{d}_2$ as:

$$cos(\theta) = \frac{\mathbf{d}_2 \cdot \mathbf{q}}{\|\mathbf{d}_2\| \cdot \|\mathbf{q}\|}$$



cosine similarity

Vector space model

- **Length normalization**: a vector can be normalised by dividing each of its components by its length
  - We can use L2 norm (it maps vectors onto the unit sphere)
  - As a result, longer documents and shorter documents have weights of the same order of magnitude
    - For normalised vectors it is equivalent to the dot product:

$$cos(\theta) = \frac{\mathbf{d}_2 \cdot \mathbf{q}}{\cancel{\|\mathbf{d}_2\|}_{1} \cdot \cancel{\|\mathbf{q}\|}_{1}}$$

$$\|\mathbf{d}_2\| = \sqrt{\sum_i w_{i,2}^2} = 1$$

Information Retrieval

- **Information Retrieval**: the task of finding documents of an unstructured nature (usually text) that satisfies an information need from within large collections

- An IR system can be characterized by:
  - A corpus of documents
  - Queries posed in a query language (e.g. using Boolean operators)
  - A result set, i.e. a subsets of document judged relevant to the query
  - A presentation of the result set (usually a ranked list)

Information Retrieval

- A **scoring function** takes a document, a query and returns a numeric score
  - Key principle: the most relevant has the highest score
  - E.g. the BM25 scoring function (defined by three factors)
    - The frequency with which a query term appears in a document (also known *TF*: Term Frequency)
    - The Inverse Document Frequency (IDF) of the term
    - The length of the document

Okapi BM25 scoring function

- We assume we have created an index of the N documents
  - We can compute $TF(q_i, d_j)$, i.e. the count of the number of times word q_i appears in document d_j
  - We assume a table of document frequency counts $DF(q_i)$

$$BM25(d_j, q_{1:N}) = \sum_{i=1}^{N} IDF(q_i) \cdot \frac{TF(q_i, d_j) \cdot (k+1)}{TF(q_i, d_j) + k \cdot (1 - b + b \cdot \frac{|d_j|}{L})}$$

where $IDF(q_i) = \log \frac{N - DF(q_i) + 0.5}{DF(q_i) + 0.5}$

TF-IDF weighting

- TF-IDF can be used as a weighting (smoothing) scheme in the classic vector space model
  - In the vector space model documents are represented as:

$$\mathbf{d}_j = (w_{1,j}, w_{2,j}, \ldots, w_{v,j})$$

  - Instead of simple term weights, now we have:

$$w_{t,j} = tf_{t,j} \cdot \log \frac{|D|}{\{d' \in D | t \in d'\}}$$

where $tf_{t,j}$ is term frequency of t in document j (local parameter)

and $\log \frac{|D|}{\{d' \in D | t \in d'\}}$ is inverse doc. frequency (global parameter)

TF-IDF weighting: example

- We have a corpus of 2 documents:

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

  - The TF for the term “this” is computed as follows:

$$tf(\text{“this”}, d_1) = 1/5 = 0.2 \qquad tf(\text{“this”}, d_2) = 1/7 \approx 0.14$$

  - IDF accounts for the ratio of documents that include “this”:

$$idf(\text{“this”}, D) = \log(2/2) = 0$$

  - So TF-IDF is zero for the word “this” (i.e. not informative):

$$tf\text{-}idf(\text{“this”}, d_1, D) = 0.2 * 0 = 0$$
$$tf\text{-}idf(\text{“this”}, d_2, D) = 0.14 * 0 = 0$$

TF-IDF weighting: example

- We have a corpus of 2 documents:

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

  - The TF for the term “this” is computed as follows:

$$tf(\text{“example”}, d_1) = 0/5 = 0 \qquad tf(\text{“example”}, d_2) = 3/7 \approx 0.429$$

  - IDF accounts for the ratio of docs that include “example”:

$$idf(\text{“example”}, D) = \log(2/1) = 0.301$$

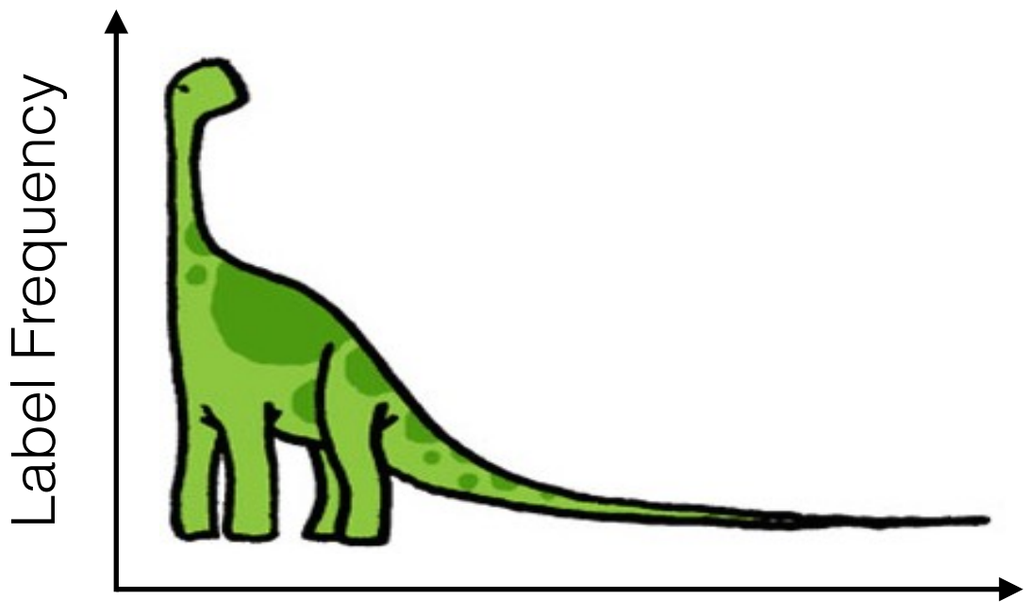
  - So TF-IDF is more interesting in this case:

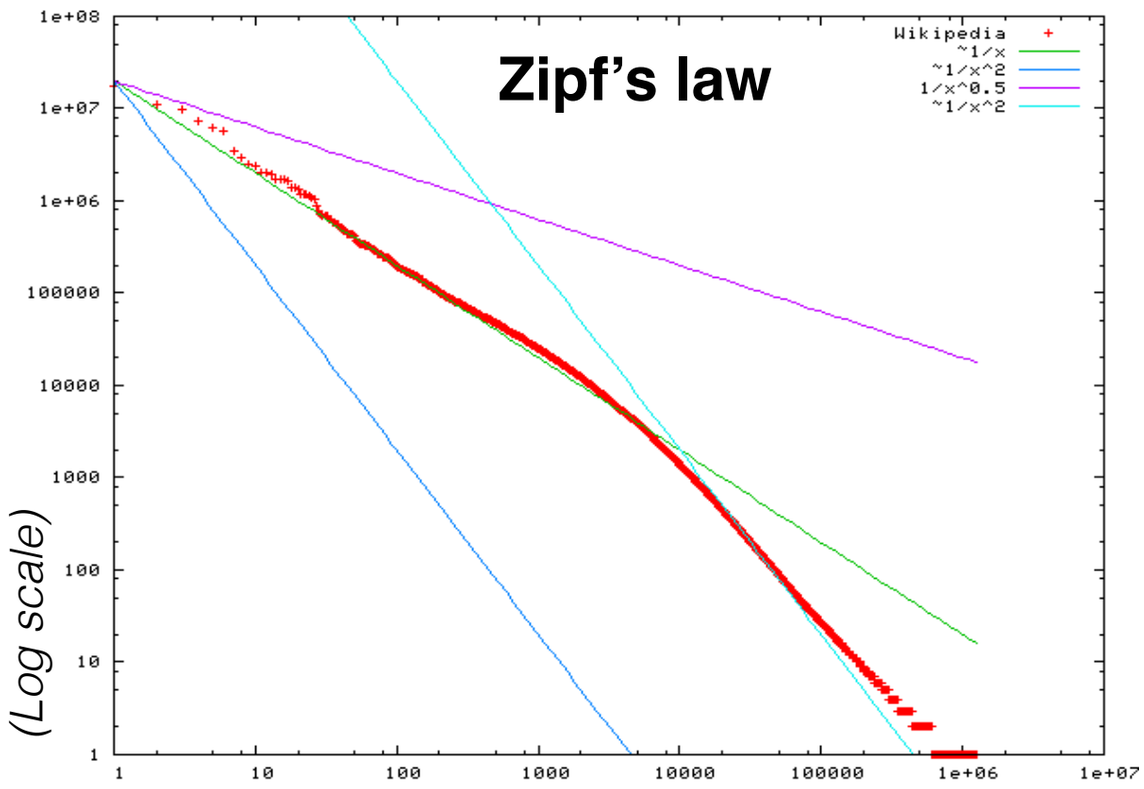
$$tf\text{-}idf(\text{“example”}, d_1, D) = 0 * 0.301 = 0$$
$$tf\text{-}idf(\text{“example”}, d_2, D) = 0.429 * 0.301 \approx 0.129$$

Frequency in document *vs* corpus

- How many frequent vs infrequent terms should we expect in a document collection (corpus)?
  - There are a few very frequent terms and very many rare terms
  - **Zipf's law**: given a corpus, the frequency of any word is inversely proportional to its rank in the frequency table

Long tail distribution





Frequency in document *vs* corpus

- Zipf's law: a few examples from five languages

	English		German		Spanish		Italian		Dutch	
1	the	61,847	der	7,377,879	que	32,894	non	25,757	de	4,770
2	of	29,391	die	7,036,092	de	32,116	di	22,868	en	2,709
3	and	26,817	und	4,813,169	no	29,897	che	22,738	het/'t	2,469
4	a	21,626	in	3,768,565	a	22,313	è	18,624	van	2,259
5	in	18,214	den	2,717,150	la	21,127	e	17,600	ik	1,999
6	to	16,284	von	2,250,642	el	18,112	la	16,404	te	1,935
7	it	10,875	zu	1,992,268	es	16,620	il	14,765	dat	1,875
8	is	9,982	das	1,983,589	y	15,743	un	14,460	die	1,807
9	to	9,343	mit	1,878,243	en	15,303	a	13,915	in	1,639
10	was	9,236	sich	1,680,106	lo	14,010	per	10,501	een	1,637

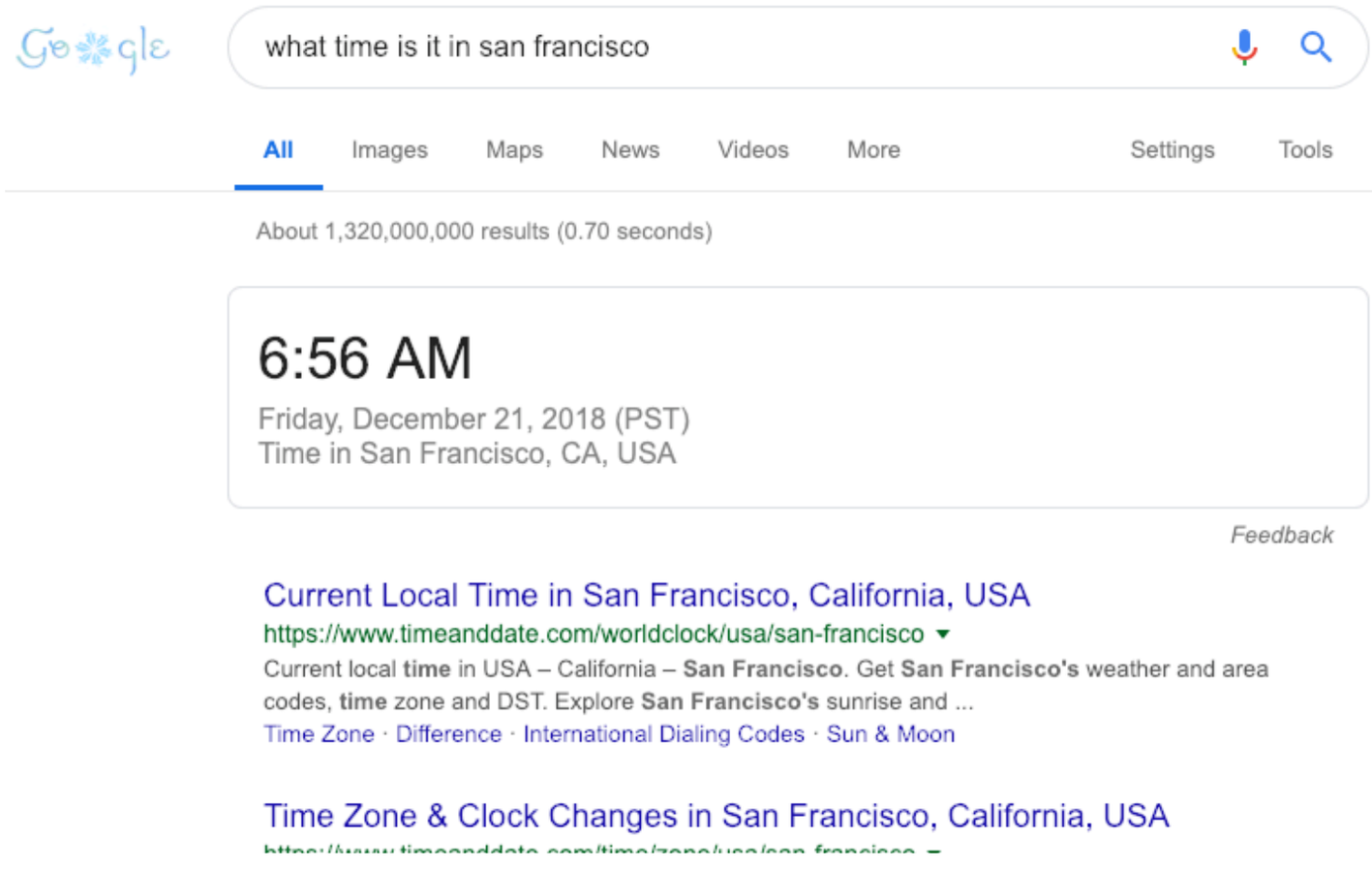
(Top-10 most frequent terms in a large language sample)

Summary: retrieval in the vector space

- Represent the query as a (weighted tf-idf) vector
- Represent each document as a (weighted tf-idf) vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents by similarity with respect to the query
- Return the top *K* results to the user (e.g. *K*=10)

Question Answering

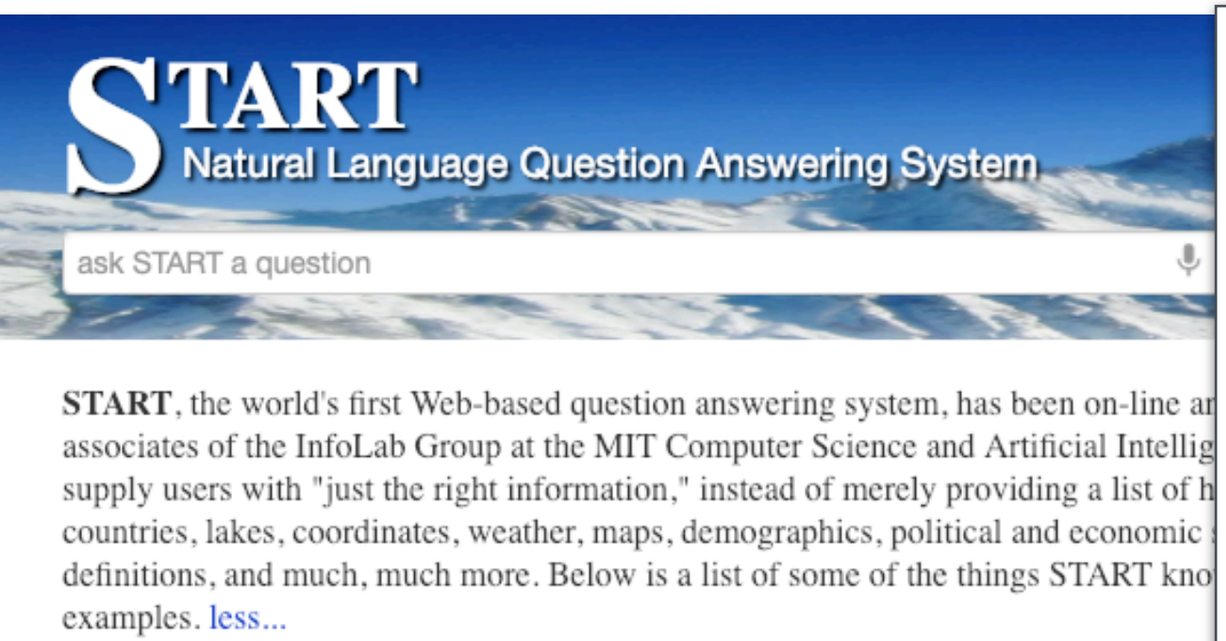
- Information Retrieval is the task of finding documents that are relevant to a query (e.g a keyword, a topic, …)
- **Question Answering** (QA) can be seen as just an application of IR where the query really is a question



i.e., “ask Google!”

Question Answering

- An example: the Start NLP QA System

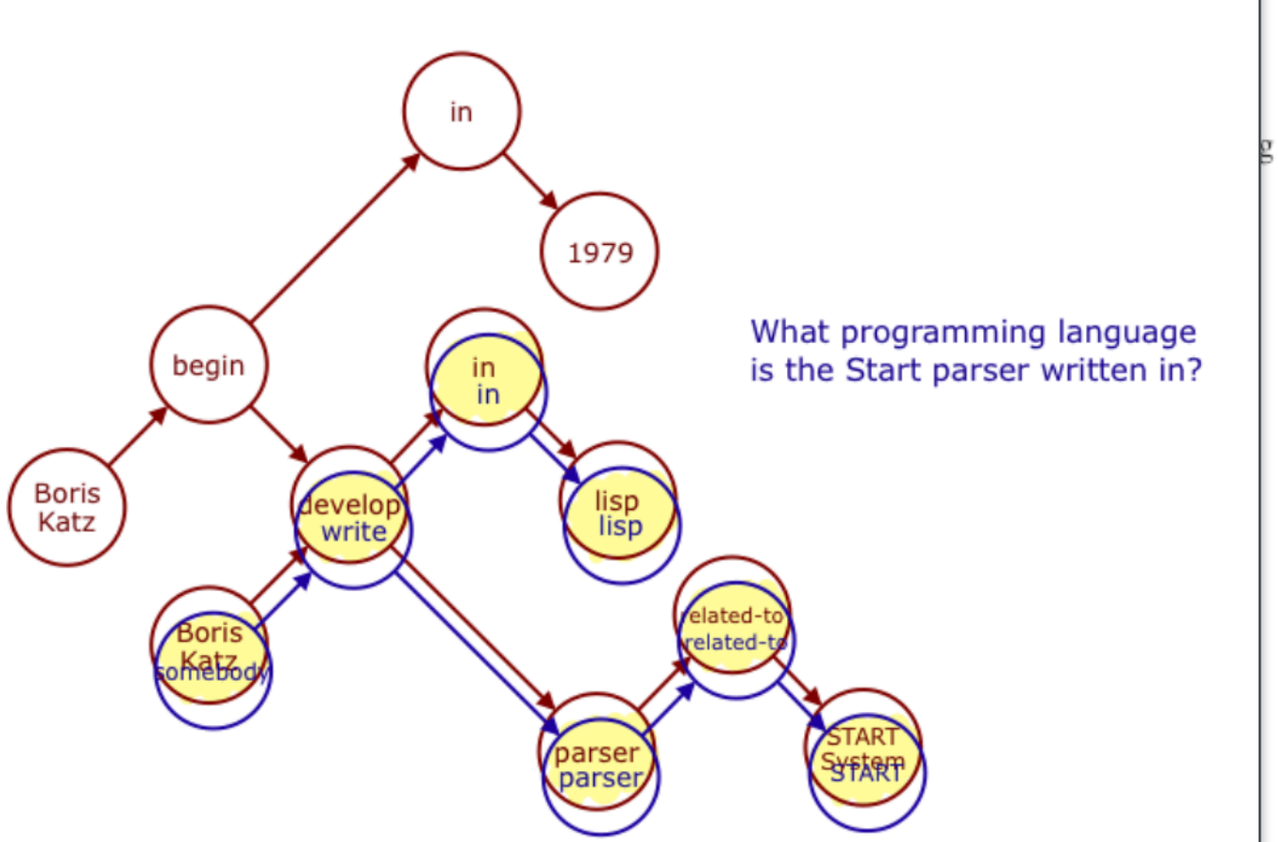


START, the world's first Web-based question answering system, has been on-line ar
associates of the InfoLab Group at the MIT Computer Science and Artificial Intellig
supply users with "just the right information," instead of merely providing a list of h
countries, lakes, coordinates, weather, maps, demographics, political and economic
definitions, and much, much more. Below is a list of some of the things START kno
examples. less...

Geography

- What South-American country has the largest population?
- What's the largest city in Florida?
- Give me the states that border Colorado.
- What cities are within 250 miles of the capital of Italy?
- How many people live in Israel?
- Show me a map of Denmark.
- How far is Mount Kilimanjaro from Mount Everest?
- List some large cities in Argentina.
- Show the capital of the 2nd largest country in Asia.
- How much does it cost to study at MIT?
- More examples...

The START Natural Language System understands and generates language and answers questions that are posed to it in natural language. As a question-answering system, START parses incoming questions, matches the queries created from the parse trees against its knowledge base and presents the appropriate information segments to the user, providing "just the right information".



http://start.csail.mit.edu/index.php

IR-based dialog agents

- **Main idea**: mine (a lot of) conversations of human chats or human-machine chats
  - E.g. starting from social media like Twitter (or proprietary data)

- A simple IR-based chatbot architecture:
  - Take user's $\mathbf{q}$ and find a TF-IDF similar $\mathbf{t}$ in the corpus C

    $\mathbf{q}$ = *"Do you like Doctor Who"*

    $\mathbf{t}$ = *"Do you like Doctor Strangelove"*

  - Return the response for the most similar turn t:

$$\mathbf{r} = response\left(\arg\max_{\mathbf{t}\in C} \frac{\mathbf{q}\cdot\mathbf{t}}{\|\mathbf{q}\|\cdot\|\mathbf{t}\|}\right)$$

IR-based dialog agents

- Pros:
  - Very easy to implement
  - They are good for narrow, scriptable applications
  - Big data usually help

- Cons:
  - They don’t really understand
  - IR-based chatbots can only mirror training data

Question Answering, chatbots

- A QA system requires a real understanding of the natural language questions
- It's also expected to give answers in natural language

Only by reputation

Alexa, do you know Siri?

A

Alexa, who is better, you or Siri?

Q

I like all AIs!

Alexa relies on machine learning to work. These bots transcribe human speech and then respond to that input with an educated guess based on what they have observed before. Alexa "learns" from new interactions, gradually improving over time.

Contact

- **Office:** Torre Archimede 6CD, room 622
- **Office hours** (ricevimento)**:** Monday 11:00-13:00

✉ lamberto.ballan@unipd.it

🏠 http://www.lambertoballan.net

🏠 http://vimp.math.unipd.it

@ twitter.com/lambertoballan