EDAP30
Advanced Applied Machine Learning
Lecture 2: Language Models and Embeddings

Pierre Nugues

Pierre.Nugues@cs.lth.se

March 22, 2023

Natural Language Processing

Applications:

- Spelling and grammatical checkers: *MS Word*
- Text indexing and information retrieval on the Internet: *Google*, *Microsoft Bing*, *Yahoo*
- Telephone information that understands some spoken questions
- Speech dictation of letters or reports: *Windows*, *macOS*

Higher-level applications:

- Spoken interaction: Apple Siri, Google Assistant, Amazon Echo
- Translation: *Google Translate*, *Bing Translator*
- Question answering: *IBM Watson* and *Jeopardy!*
- Dialogue: GPT and the likes

Techniques

The inner engines of these applications tend to be powered by neural networks.

Big changes from the

- 1980's: rules,
- 1990's: Bayes,
- 2000's: SVM and logistic regression (logistic regression is an essential part of neural networks. It is still usable alone for many tasks).
- 2010's: (Deep) neural networks

Each generation tends to replace the previous one.

Neural nets expansion started in 2010 with images, in 2018-2019 with transformers. What in 2030?

Scope of the NLP Lectures

Three lectures

Techniques we will consider:

1. Language models and embeddings
2. Sequential networks
3. Transformers

Applications we will consider:

1. Information retrieval
2. Text categorization
3. Word or segment categorization
4. Translation: *Google Translate*, *DeepL*, *Bing translator*, etc.

Corpora

A corpus is a collection of texts (written or spoken) or speech
Corpora are balanced from different sources: news, novels, etc.

	English	**French**	**German**
Most frequent words in a collection of contemporary running texts	*the*	*de*	*der*
	of	*le* (article)	*die*
	to	*la* (article)	*und*
	in	*et*	*in*
	and	*les*	*des*
Most frequent words in Genesis	*and*	*et*	*und*
	the	*de*	*die*
	of	*la*	*der*
	his	*à*	*da*
	he	*il*	*er*

Characteristics of Current Corpora

Big:	starting with the Bank of English (Collins and U Birmingham) had more than 500 million words
Multilingual:	Wikipedia
Easy to collect:	The web is the largest corpus ever built and within the reach of a mouse click
Exhaustive:	Common crawl: `https://commoncrawl.org/`
Parallel:	same text in two languages: English/French (Canadian Hansards), European parliament (23 languages)
Annotated:	Part-of-speech or manually parsed (treebanks): *Characteristics/NOUN of/PREP Current/ADJ Corpora/NOUN*

Corpora as Knowledge Sources

1. Traditional use:
    - Describe usage more accurately
2. Machine learning
    - Learn statistical/machine-learning models for speech and text processing
    - Assess models and tools
    - Derive automatically knowledge from annotated or unannotated corpora
3. Applications:
    - Text categorization
    - Information extraction
    - Question answering from textual sources
    - Translation

The Reuters Corpus

The Reuters corpus contains 800,00 economic newswires

```
<title>USA: Tylan stock jumps; weighs sale of company.</title>
<headline>Tylan stock jumps; weighs sale of company.</headline>
<dateline>SAN DIEGO</dateline>
<text>
  <p>The stock of Tylan General Inc. jumped Tuesday after the maker of process-management equipment said it
  is exploring the sale of the company and added that it has already received some  inquiries from potential
  buyers.</p>
  <p>Tylan was up $2.50 to $12.75 in early trading on the Nasdaq market.</p>
  <p>The company said it has set up a committee of directors to oversee the sale and that Goldman, Sachs
  &amp; Co. has been retained as its financial adviser.</p>
</text>
<metadata>
  <codes class="bip:topics:1.0">
    <code code="C15"/>
    <code code="C152"/>
    <code code="C18"/>
    <code code="C181"/>
    <code code="CCAT"/>
  </codes>
  ...
url: http://trec.nist.gov/data/reuters/reuters.html
```

Newswires are manually annotated with pre-defined topics:

C15	Performance	C152	Comment/Forecasts
C18	Ownership changes	C181	Mergers/Acquisitions
CCAT	Corporate/Industrial	...	

Information Retrieval

Astronomic number of available documents
Search engines – Google, Yahoo – are examples of tools to retrieve information on the web
Usually, we have:

- A document collection
- A query
- A result consisting of a set of documents

The simplest technique is to use a Boolean formula of conjunctions and disjunctions that will return the documents satisfying it.
Another is to vectorize the documents and define a similarity measure

Preprocessing Text

Before we can apply machine learning algorithm, we need to preprocess the texts: Format it so that it can use it as input to our programs
It includes (but it is not limited to):

1. Format parsing (HTML, XML, etc.) and text extraction
2. Tokenization
3. Sentence segmentation
4. Encoding
5. Cleaning.

Encoding Words

Machine-learning models prefer numbers
We need then to encode the words or the characters with numbers.
Using ordinal numbers {a:1, b:2, c:3, d:4, etc.} is impossible.
Is a closer to b, than c?
One-hot encoding: We encode the words of a corpus with unit vectors

Example

A very small corpus of two documents:

D1: Chrysler plans new investments in Latin America.

D2: Chrysler plans major investments in Mexico.

One-hot encoding:

- Collect all the unique words, possibly in lower case, and sort them: (america, chrysler, in, investments, latin, major, mexico, new, plans)
- Assign them a unique index: {america: 1, chrysler: 2, in: 3, investments: 4, latin: 5, ...}

Associate a word with a unit vector:

america:	(1, 0, 0, 0, 0, 0, 0, 0, 0)
chrysler:	(0, 1, 0, 0, 0, 0, 0, 0, 0)
in:	(0, 0, 1, 0, 0, 0, 0, 0, 0)
investments:	(0, 0, 0, 1, 0, 0, 0, 0, 0)
...	

Vectorizing Documents

Following this idea, we represent documents by the words they contain:

$$\text{word} \in \text{document} \rightarrow \text{True (1)} \textit{ or } \text{False (0)}$$

With the corpus:

D1: Chrysler plans new investments in Latin America.

D2: Chrysler plans major investments in Mexico.

We have:

D#\ Words	america	chrysler	in	investments	latin	major	mexico	new	plans
1	1	1	1	1	1	0	0	1	1
2	0	1	1	1	0	1	1	0	1

And the vectorization:

D1: (1, 1, 1, 1, 1, 0, 0, 1, 1)

D2: (0, 1, 1, 1, 0, 1, 1, 0, 1)

This technique is often called *bag-of-word* (also multihot encoding) as the word order plays no role.

The Vector Space Model

We represented a document as a vector in a space of words, where the coordinates are the indices of the unit vectors
More generally, the vector space model represents a document in word space:

Documents \Words	w_1	w_2	w_3	**...**	w_m
D_1	$C(w_1, D_1)$	$C(w_2, D_1)$	$C(w_3, D_1)$	...	$C(w_m, D_1)$
D_2	$C(w_1, D_2)$	$C(w_2, D_2)$	$C(w_3, D_2)$	...	$C(w_m, D_2)$
...					
D_n	$C(w_1, D_n)$	$C(w_2, D_n)$	$C(w_3, D_n)$	...	$C(w_m, D_n)$

The most simple coordinates are Boolean values

Giving a Weight

Word clouds give visual weights to words

projekt
design
enklare
konstruktion
system
algoritmer
modellering
implementering
klasser
testning
implementation
programmering
objektorienterade
objektorienterad
användning
programkonstruktioner
studenterna
databaser
tekniker
coaching
sortering
registrering
analys
optimering
program
vektorer
samt
polymorfism
funktionella
funktioner
datastrukturer
konfigurationshantering
typer
kh
egna
andra
praktisk
exempel
består
får
relationsmodellen
operativsystem
elementära
sökning
verifiering
abstrakta
inkapsling
parsning
processer
felsökning
begrepp
färdiga
krav
metoder
följande
teorin
distribuerade
föreläsningar
minneshantering
träning

Image: Courtesy of Jonas Wisbrant

$TF \times IDF$

The presence of a word in a document does not give much information on its importance

The term frequency $tf_{i,j}$: frequency of term j in document i tells more on the document topic

Very frequent terms across the corpus are not informative. The inverted document frequency downplays their importance.

The inverted document frequency is defined as: $idf_j = \log(\frac{N}{n_j})$, where n_j is the number of documents in the corpus containing term j and N, the number of documents.

TFIDF is a baseline vectorization technique, where the document coordinates (D_i, w_j) are the products:

$$d_{i,j} = tf_{i,j} \times idf_j,$$

term frequency by inverted document frequency

Document Similarity

With vectorized documents, we can:

1. Measure the similarity between two documents $(\mathbf{d}_1, \mathbf{d}_2)$
2. Rank documents $(\mathbf{d_1}, \mathbf{d}_2, ..., \mathbf{d}_N)$ depending on a query $\mathbf{q}$.

The vector coordinates could be the the tfidf values or the count of each word: $\mathbf{d} = (C(w_1), C(w_2), C(w_3), ..., C(w_n))$
Queries $\mathbf{q}$ are represented similarly
The cosine of two documents $\mathbf{d}$ and $\mathbf{q}$:

$$\cos(\mathbf{q}, \mathbf{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}}.$$

defines their similarity.

Text Categorization

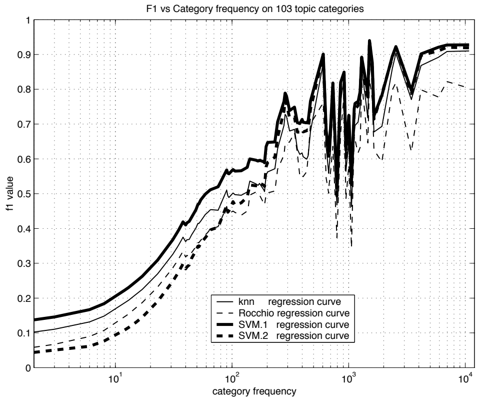
The objective is to determine the type of a text with a set of predefined categories, for instance: {spam, no spam}
Reuters uses 103 topic categories:

C11	STRATEGY/PLANS	C15	PERFORMANCE
C12	LEGAL/JUDICIAL	C151	ACCOUNTS/EARNINGS
C13	REGULATION/POLICY	C1511	ANNUAL RESULTS
C14	SHARE LISTINGS	C152	COMMENT/FORECASTS
C15	PERFORMANCE	C16	INSOLVENCY/LIQUIDITY
C151	ACCOUNTS/EARNINGS	C17	FUNDING/CAPITAL
C1511	ANNUAL RESULTS	C171	SHARE CAPITAL
C152	COMMENT/FORECASTS	C172	BONDS/DEBT ISSUES
C16	INSOLVENCY/LIQUIDITY	C173	LOANS/CREDITS
C17	FUNDING/CAPITAL	C174	CREDIT RATINGS
C171	SHARE CAPITAL	C18	OWNERSHIP CHANGES
C172	BONDS/DEBT ISSUES	C181	MERGERS/ACQUISITIONS
C173	LOANS/CREDITS	C182	ASSET TRANSFERS
C174	CREDIT RATINGS	C183	PRIVATISATIONS
C11	STRATEGY/PLANS	C21	PRODUCTION/SERVICES
C12	LEGAL/JUDICIAL	C22	NEW PRODUCTS/SERVICES
C13	REGULATION/POLICY	C23	RESEARCH/DEVELOPMENT
C14	SHARE LISTINGS	...	

A baseline technique is to represent the documents with in tfidf vector space and use logistic regression as classifier

Algorithms for Text Categorization

The performance depends on the number of samples



David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li, RCV1: A New Benchmark Collection for Text Categorization Research, *Journal of Machine Learning Research* 5 (2004) 361-397.

Word Contexts

Words and characters have specific contexts of use.
Pairs of words like *strong* and *tea* or *powerful* and *computer* are not random associations.
Psychological linguistics tells us that it is difficult to make a difference between *writer* and *rider* without context
A listener will discard the improbable *rider of books* and prefer *writer of books*
A **language model** is the statistical estimate of a word sequence:

- Originally developed for speech recognition
- The language model component predicts the next most likely words given a sequence of previous words

Language Models

$$\begin{aligned} P(S) &= P(w_1, ..., w_n), \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)...P(w_n|w_1, ..., w_{n-1}), \\ &= \prod_{i=1}^{n} P(w_i|w_1, ..., w_{i-1}). \end{aligned}$$

For the sentence (from *Nineteen Eighty-Four*):

It was a bright cold day in April

We have:

$$\begin{aligned} P(S) &= P(It\ was\ a\ bright\ cold\ day\ in\ April) \\ &= P(It) \times P(was|It) \times P(a|It, was) \times P(bright|It, was, a) \times ... \\ &\quad \times P(April|It, was, a, bright, ..., in). \end{aligned}$$

N-grams

In natural language processing, sequences of *n* words are called *n*-grams:

- Unigrams are single words;
- Bigrams are sequences of two words;
- Trigrams are sequences of three words;
- etc.

In the sentence:

Chrysler plans new investments in Latin America

We have:

The bigrams: (Chrysler, plans), (plans, new), (new, investments), (investments, in), (in, Latin), (Latin, America)

The trigrams: (Chrysler, plans, new), (plans, new, investments), (new, investments, in), (investments, in, Latin), (in, Latin, America)

Approximations

Bigrams:

$$P(w_i|w_1, w_2, ..., w_{i-1}) \approx P(w_i|w_{i-1}),$$

Trigrams:

$$P(w_i|w_1, w_2, ..., w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1}).$$

Using a trigram language model:

$$\begin{aligned} P(S) &= P(It\ was\ a\ bright\ cold\ day\ in\ April) \\ &= P(It) \times P(was|It) \times P(a|It, was) \times P(bright|It, was, a) \times ... \\ &\quad \times P(April|It, was, a, bright, ..., in). \end{aligned}$$

is approximated as:

$$\begin{aligned} P(S) \approx\ & P(It) \times P(was|It) \times P(a|It, was) \times P(bright|was, a) \times ... \\ & \times P(April|day, in). \end{aligned}$$

Predicting with Trigrams

Word	Rank	More likely alternatives
We	9	*The This One Two A Three Please In*
need	7	*are will the would also do*
to	1	
resolve	85	*have know do...*
all	9	*the this these problems...*
of	2	*the*
the	1	
important	657	*document question first...*
issues	14	*thing point to...*
within	74	*to of and in that...*
the	1	
next	2	*company*
two	5	*page exhibit meeting day*
days	5	*weeks years pages months*

Language Models and Generation

Using a n-gram language model, we can generate a sequence of words.
Starting from a first word, w_1, we extract the conditional probabilities: $P(w_2|w_1)$.
We could take the highest value, but it would always generate the same sequence.
Instead, we will draw our words from a multinomial distribution using `np.random.multinomial()`.
Given a probability distribution, this function draws a sample that complies the distribution.
Having $P(want|I) = 0.5$, $P(wish|I) = 0.3$, $P(will|I) = 0.2$, the function will draw wish 30% of the time.

Code Example

Generating sequences with Bayesian probabilities

Jupyter Notebooks: https://github.com/pnugues/ilppp/blob/master/programs/ch05/python/lm_generation.ipynb

Generating Word Sequences with a Temperature

In addition, we can use a “temperature” function to reweight the probability distribution: sharpen or damp it, as for instance in Chollet (2021): $\exp(\frac{\log(x)}{temp}) = x^{\frac{1}{temp}}$

```
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

with the input `[0.2, 0.5, 0.3]`, we obtain:

- Temperature = 2, `[0.26275107 0.41544591 0.32180302]`
- Temperature = 1, `[0.2 0.5 0.3]`
- Temperature = 0.5 `[0.10526316 0.65789474 0.23684211]`
- Temperature = 0.2 `[0.00941176 0.91911765 0.07147059]`

Code Example

Jupyter Notebooks: https://github.com/pnugues/ilppp/blob/master/programs/ch05/python/lm_generation.ipynb

Dimension Reduction and Embeddings

One-hot or TFIDF encoding can produce very long sparse vectors.
Imagine a vocabulary of one million words per language with 100 languages.
A solution is to use **dense vectors** in a vector space of small dimension
The new vectors are called **embeddings** with dimensions ranging from 10 to 1000
Input embeddings can be trained from scratch or initialized with pretrained vectors

Pretrained Embeddings

Many ways to create such embeddings:

- word2vec and CBOW
- GloVe

Comparable to techniques such as:

- Factor analysis
- Singular value decomposition or a principal component analysis

Note that these techniques are not used in practice in deep learning

GloVe Vectors

An excerpt of GloVe 50: 400,000 words represented by vectors of 50 dimensions.

```
...
then 0.19565 -0.32773 0.061642 -0.61557 0.55709 0.1319 -0.6402
company 0.62583 -0.57703 0.41163 0.86812 -0.083097 0.26555 -1.
group 0.74048 -0.1201 0.039916 0.77326 0.80822 0.32251 -0.7721
any 0.51292 0.09032 0.023552 0.21438 0.70226 0.5623 0.11839 0.
through 0.64925 0.068384 0.031703 -0.51479 -0.52809 -0.068008
china -0.22427 0.27427 0.054742 1.4692 0.061821 -0.51894 0.450
four 0.33375 0.44809 0.54759 0.21497 0.32637 0.70326 -0.95872
being 0.59049 -0.66076 -0.02551 -0.66217 0.3834 0.31517 -0.826
down -0.1981 -0.70847 0.85857 -0.48108 0.51562 -0.28924 -0.643
war 0.36544 -0.15746 -0.23966 -1.0307 -0.070691 0.21397 -0.041
...
```

Source: `https://nlp.stanford.edu/projects/glove/`

Cooccurrence Matrix

GloVe needs a cooccurrence matrix of the counts $C(w_i, w_j)$ for all the pairs of words.
For this, we use contexts of $2K$ words centered on the focus word w_i:

$$w_{i-K}, w_{i-K+1}, \ldots, w_{i-1}, \mathbf{w}_i, w_{i+1}, \ldots, w_{i+K-1}, w_{i+K},$$

K is set to 10 and the contribution of a word in the context to the count is $1/d$, d being the distance to the focus word.

	w_1	w_2	w_3	...	w_n
w_1	$C(w_1, w_1)$	$C(w_1, w_2)$	$C(w_1, w_3)$	...	$C(w_1, w_n)$
w_2	$C(w_2, w_1)$	$C(w_2, w_2)$	$C(w_2, w_3)$	...	$C(w_2, w_n)$
w_3	$C(w_3, w_1)$	$C(w_3, w_2)$	$C(w_3, w_3)$	...	$C(w_3, w_n)$
...	...	...	...	...	...
w_n	$C(w_n, w_1)$	$C(w_n, w_2)$	$C(w_n, w_3)$	...	$C(w_m, w_n)$

Each matrix element measures the association strength between two words

GloVe

We saw that the cosine of two vectors is a measure of their similarity
GloVe defines the embedding vectors of two words, $\mathbf{emb}(w_i)$ and $\mathbf{emb}(w_j)$, so that their dot product is equal to the logarithm of their cooccurrence count:

$$\mathbf{emb}(w_i) \cdot \mathbf{emb}(w_j) = \log C(w_i, w_j)$$

The corresponds to an optimization problem
As loss function, GloVe uses the sum of the squared errors:

$$(\mathbf{emb}(w_i) \cdot \mathbf{emb}(w_j) - \log C(w_i, w_j))^2$$

and fits the embeddings with a gradient descent.

Word2vec Embeddings

word2vec have two forms: CBOW and skipgrams.
CBOW's aim is to predict a word given its surrounding context:

I went to —— *a friend*

The setup is similar to fill-the-missing-word questionnaires.
The missing word is called the focus word, here *visit*
CBOW's architecture is simply a multinomial logistic regression
The word inputs are trainable dense vectors, where each word is associated with a vector
The word vectors, the embeddings, are trained on a corpus

Word2vec Example

Using contexts of five words and training sentences such as:

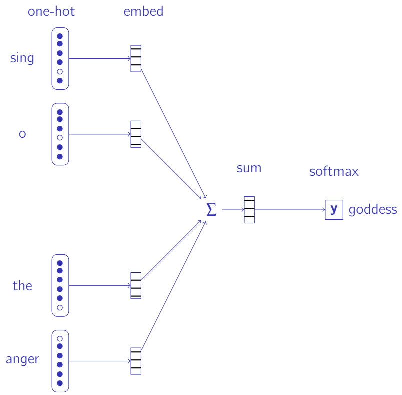
$\underline{\textit{Sing, O goddess, the anger}}$ *of Achilles son of Peleus,*

we generate a training set of contexts deprived of their focus word (X) and the focus word to predict ($\mathbf{y}$):

$$X = \begin{bmatrix} \text{sing} & \text{o} & \text{the} & \text{anger} \\ \text{o} & \text{goddess} & \text{anger} & \text{of} \\ \text{goddess} & \text{the} & \text{of} & \text{achilles} \\ \text{the} & \text{anger} & \text{achilles} & \text{son} \\ \text{anger} & \text{of} & \text{son} & \text{of} \\ \text{of} & \text{achilles} & \text{of} & \text{peleus} \end{bmatrix}; \mathbf{y} = \begin{bmatrix} \text{goddess} \\ \text{the} \\ \text{anger} \\ \text{of} \\ \text{achilles} \\ \text{son} \end{bmatrix}$$

CBOW Architecture

We train a neural network to get the CBOW embeddings: N dimension of the embeddings, V size of the vocabulary. First matrix (V, N), second (N, V).



Code Example

https://github.com/pnugues/ilppp/blob/master/programs/ch05/python/cbow_book_fit.ipynb
https://github.com/pnugues/ilppp/blob/master/programs/ch05/python/skipgram_book_gen.ipynb

Principal Component Analysis

We will use a small dataset to explain principal component analysis: The characters in *Salammbô*

Ch.	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'j'	'k'	'l'	'm'	'n'	'o'	'p'	'q'	'r'	's'	't'	'u'	'v'	'w
01_fr	2503	365	857	1151	4312	264	349	295	1945	65	4	1946	726	1896	1372	789	248	1948	2996	1938	1792	414	
02_fr	2992	391	1006	1388	4993	319	360	350	2345	81	6	2128	823	2308	1560	977	281	2376	3454	2411	2069	499	
03_fr	1042	152	326	489	1785	136	122	126	784	41	7	816	397	778	612	315	102	792	1174	856	707	147	
04_fr	2487	303	864	1137	4158	314	331	287	2028	57	3	1796	722	1958	1318	773	274	2000	2792	2031	1734	422	
05_fr	2014	268	645	949	3394	223	215	242	1617	67	3	1513	651	1547	1053	672	166	1601	2192	1736	1396	315	
06_fr	2805	368	910	1266	4535	332	384	378	2219	97	3	1900	841	2179	1569	868	285	2205	3065	2293	1895	453	
07_fr	5062	706	1770	2398	8512	623	622	620	4018	126	19	3726	1596	3851	2823	1532	468	4015	5634	4116	3518	844	
08_fr	2643	325	869	1085	4229	307	317	359	2102	85	4	1857	811	2041	1367	833	239	2132	2814	2134	1788	437	
09_fr	2126	289	771	920	3599	278	289	279	1805	52	6	1499	619	1711	1130	651	187	1719	2404	1763	1448	348	
10_fr	1784	249	546	805	3002	179	202	215	1319	60	5	1462	598	1246	922	557	172	1242	1769	1423	1191	270	
11_fr	2641	381	817	1078	4306	263	277	330	1985	114	0	1886	900	1966	1356	763	230	1912	2564	2218	1737	425	
12_fr	2766	373	935	1237	4618	329	350	349	2273	65	2	1955	812	2285	1419	865	272	2276	3131	2274	1923	455	
13_fr	5047	725	1730	2273	8678	648	566	642	3940	140	22	3746	1597	3984	2736	1550	425	4081	5599	4387	3480	767	
14_fr	5312	689	1754	2149	8870	628	630	673	4278	143	2	3780	1610	4255	2713	1599	512	4271	5770	4467	3697	914	
15_fr	1215	173	402	582	2195	150	134	148	969	27	6	950	387	906	697	417	103	985	1395	1037	893	206	
01_en	2217	451	729	1316	3967	596	662	2060	1823	22	200	1204	656	1851	1897	525	19	1764	1942	2547	704	258	65
02_en	2761	551	777	1548	4543	685	769	2530	2163	13	284	1319	829	2218	2237	606	21	2019	2411	3083	861	295	76
03_en	990	183	271	557	1570	279	253	875	783	4	82	520	333	816	828	194	13	711	864	1048	298	94	25
04_en	2274	454	736	1315	3814	595	559	1978	1835	22	198	1073	690	1771	1865	514	33	1726	1918	2704	745	245	66
05_en	1865	400	553	1135	3210	515	525	1693	1482	7	153	949	571	1468	1586	517	17	1357	1646	2178	663	194	56
06_en	2606	518	797	1509	4237	687	669	2254	2097	26	216	1239	763	2174	2231	613	25	1931	2192	2955	899	277	73
07_en	4805	913	1521	2681	7834	1366	1163	4379	3838	42	416	2434	1461	3816	4091	1040	39	3674	4060	5369	1552	465	133
08_en	2396	431	702	1416	4014	621	624	2171	2011	24	216	1152	748	2085	1947	527	33	1915	1966	2765	789	266	69
09_en	1993	408	653	1096	3373	575	517	1766	1648	16	146	861	629	1728	1698	442	20	1561	1626	2442	683	208	56
10_en	1627	359	451	933	2690	477	409	1475	1196	7	131	789	506	1266	1369	325	23	1211	1344	1759	502	181	41
11_en	2375	437	643	1364	3790	610	644	2217	1830	16	217	1122	799	1833	1948	486	23	1720	1945	2424	767	246	63
12_en	2560	489	757	1566	4331	677	650	2348	2033	28	234	1102	746	2125	2105	581	32	1939	2152	3046	750	278	72
13_en	4597	987	1462	2689	7963	1254	1201	4278	3634	39	432	2281	1493	3774	3911	1099	49	3577	3894	5540	1379	437	137
14_en	4871	948	1439	2799	8179	1335	1140	4534	3829	36	427	2218	1534	4053	3989	1019	36	3689	3946	5858	1490	539	137
15_en	1119	229	335	683	1994	323	281	1108	912	9	112	579	351	924	1004	305	9	863	997	1330	310	108	33

Table: Character counts per chapter, where the fr and en suffixes designate the language, either French or English

Each chapter is modeled by a vector of characters.

Character Counts

	French														
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
a	2503	2992	1042	2487	2014	2805	5062	2643	2126	1784	2641	2766	5047	5312	1215
b	365	391	152	303	268	368	706	325	289	249	381	373	725	689	173
c	857	1006	326	864	645	910	1770	869	771	546	817	935	1730	1754	402
d	1151	1388	489	1137	949	1266	2398	1085	920	805	1078	1237	2273	2149	582
e	4312	4993	1785	4158	3394	4535	8512	4229	3599	3002	4306	4618	8678	8870	2195
f	264	319	136	314	223	332	623	307	278	179	263	329	648	628	150
g	349	360	122	331	215	384	622	317	289	202	277	350	566	630	134
h	295	350	126	287	242	378	620	359	279	215	330	349	642	673	148
i	1945	2345	784	2028	1617	2219	4018	2102	1805	1319	1985	2273	3940	4278	969
j	65	81	41	57	67	97	126	85	52	60	114	65	140	143	27
k	4	6	7	3	3	3	19	4	6	5	0	2	22	2	6
l	1946	2128	816	1796	1513	1900	3726	1857	1499	1462	1886	1955	3746	3780	950
m	726	823	397	722	651	841	1596	811	619	598	900	812	1597	1610	387
n	1896	2308	778	1958	1547	2179	3851	2041	1711	1246	1966	2285	3984	4255	906
o	1372	1560	612	1318	1053	1569	2823	1367	1130	922	1356	1419	2736	2713	697
p	789	977	315	773	672	868	1532	833	651	557	763	865	1550	1599	417
q	248	281	102	274	166	285	468	239	187	172	230	272	425	512	103
r	1948	2376	792	2000	1601	2205	4015	2132	1719	1242	1912	2276	4081	4271	985
s	2996	3454	1174	2792	2192	3065	5634	2814	2404	1769	2564	3131	5599	5770	1395
t	1938	2411	856	2031	1736	2293	4116	2134	1763	1423	2218	2274	4387	4467	1037
u	1792	2069	707	1734	1396	1895	3518	1788	1448	1191	1737	1923	3480	3697	893
v	414	499	147	422	315	453	844	437	348	270	425	455	767	914	206
w	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x	129	175	42	138	83	151	272	135	119	65	114	149	288	283	63
y	94	89	31	81	67	80	148	64	58	61	61	98	119	145	36
z	20	23	7	27	18	39	71	30	20	11	25	37	41	41	3
à	128	136	39	110	90	131	246	130	90	73	101	129	209	224	48
â	36	50	9	43	67	42	50	43	24	18	40	33	55	75	20
æ	0	1	0	0	0	0	1	0	2	0	0	0	3	0	2
ç	35	28	10	22	24	30	46	34	16	16	34	23	61	56	17
è	102	147	49	138	112	122	232	119	99	68	108	151	237	260	58
é	423	513	194	424	367	548	966	502	370	304	438	480	940	1019	221
ê	43	68	24	36	44	57	96	54	43	53	68	60	126	94	32
ë	1	0	0	0	1	0	2	0	1	0	1	0	0	0	0
î	17	20	12	15	11	15	42	11	8	15	26	13	32	28	12
ï	2	0	0	2	8	12	9	1	2	5	15	3	5	2	0
ô	20	20	27	15	23	15	41	14	13	38	50	15	37	45	24
ù	14	9	4	6	18	14	30	6	5	3	7	11	24	21	7
û	7	9	7	4	15	15	38	8	15	10	9	14	30	21	11
œ	5	5	2	8	7	9	9	5	3	5	7	0	13	12	6

	English											
	01	02	03	04	05	06	07	08	09	10	11	12
a	2217	2761	990	2274	1865	2606	4805	2396	1993	1627	2375	2560
b	451	551	183	454	400	518	913	431	408	359	437	489
c	729	777	271	736	553	797	1521	702	653	451	643	757
d	1316	1548	557	1315	1135	1509	2681	1416	1096	933	1364	1566
e	3967	4543	1570	3814	3210	4237	7834	4014	3373	2690	3790	4331
f	596	685	279	595	515	687	1366	621	575	477	610	677
g	662	769	253	559	525	669	1163	624	517	409	644	650
h	2060	2530	875	1978	1693	2254	4379	2171	1766	1475	2217	2348
i	1823	2163	783	1835	1482	2097	3838	2011	1648	1196	1830	2033
j	22	13	4	22	7	26	42	24	16	7	16	28
k	200	284	82	198	153	216	416	216	146	131	217	234
l	1204	1319	520	1073	949	1239	2434	1152	861	789	1122	1102
m	656	829	333	690	571	763	1461	748	629	506	799	746
n	1851	2218	816	1771	1468	2174	3816	2085	1728	1266	1833	2125
o	1897	2237	828	1865	1586	2231	4091	1947	1698	1369	1948	2105
p	525	606	194	514	517	613	1040	527	442	325	486	581
q	19	21	13	33	17	25	39	33	20	23	23	32
r	1764	2019	711	1726	1357	1931	3674	1915	1561	1211	1720	1939
s	1942	2411	864	1918	1646	2192	4060	1966	1626	1344	1945	2152
t	2547	3083	1048	2704	2178	2955	5369	2765	2442	1759	2424	3046
u	704	861	298	745	663	899	1552	789	683	502	767	750
v	258	295	94	245	194	277	465	266	208	181	246	278
w	653	769	254	663	568	733	1332	695	560	410	632	721
x	29	37	8	60	26	49	74	65	25	31	20	35
y	401	475	145	467	330	464	843	379	328	255	457	418
z	18	31	15	19	33	37	52	24	18	20	39	40
à	0	0	0	0	0	0	0	0	0	0	0	0
â	0	0	0	0	0	0	0	0	0	0	0	0
æ	0	0	0	0	0	0	0	0	0	0	0	0
ç	0	0	0	0	0	0	0	0	0	0	0	0
è	0	0	0	0	0	0	0	0	0	0	0	0
é	0	0	0	0	0	0	0	0	0	0	0	0
ê	0	0	0	0	0	0	0	0	0	0	0	0
ë	0	0	0	0	0	0	0	0	0	0	0	0
î	0	0	0	0	0	0	0	0	0	0	0	0
ï	0	0	0	0	0	0	0	0	0	0	0	0
ô	0	0	0	0	0	0	0	0	0	0	0	0
ù	0	0	0	0	0	0	0	0	0	0	0	0
û	0	0	0	0	0	0	0	0	0	0	0	0
œ	0	0	0	0	0	0	0	0	0	0	0	0

Table: Character counts per chapter in French, left part, and English, right part

Each characters is modeled by a vector of chapters.

Singular Value Decomposition

There are as many as 40 characters: the 26 unaccented letters from a to z and the 14 French accented letters

Singular value decomposition (SVD) reduces these dimensions, while keeping the resulting vectors semantically close

X is the $m \times n$ matrix of the letter counts per chapter, in our case, $m = 30$ and $n = 40$.

We can rewrite X as:

$$X = U \Sigma V^{\intercal},$$

where U is a matrix of dimensions $m \times m$, Σ, a diagonal matrix of dimensions $m \times n$, and V, a matrix of dimensions $n \times n$.

The diagonal terms of Σ are called the **singular values** and are traditionally arranged by decreasing value.

We keep the highest values and set the rest to zero.

Code Example

Jupyter Notebook: https://github.com/pnugues/ilppp/blob/master/programs/ch05/python/ch05-3.ipynb

Popular Word Embeddings

Embeddings from large corpora are obtained with iterative techniques
Some popular embedding algorithms with open source programs:

word2vec:	`https://github.com/tmikolov/word2vec`
GloVe:	Global Vectors for Word Representation `https://nlp.stanford.edu/projects/glove/`
ELMo:	`https://allennlp.org/elmo`
fastText:	`https://fasttext.cc/`

To derive word embeddings, you will have to apply these programs on a very large corpus
Embeddings for many languages are also publicly available. You just download them
gensim is a Python library to create word embeddings from a corpus.
`https://radimrehurek.com/gensim/index.html`

Semantic Similarity

Word embeddings mitigate the dimension problem relatively to one-hot encoding

In addition, similar words will have similar vectors

Demo: `http://bionlp-www.utu.fi/wv_demo/`

This enables to cope with words unseen in a training set