

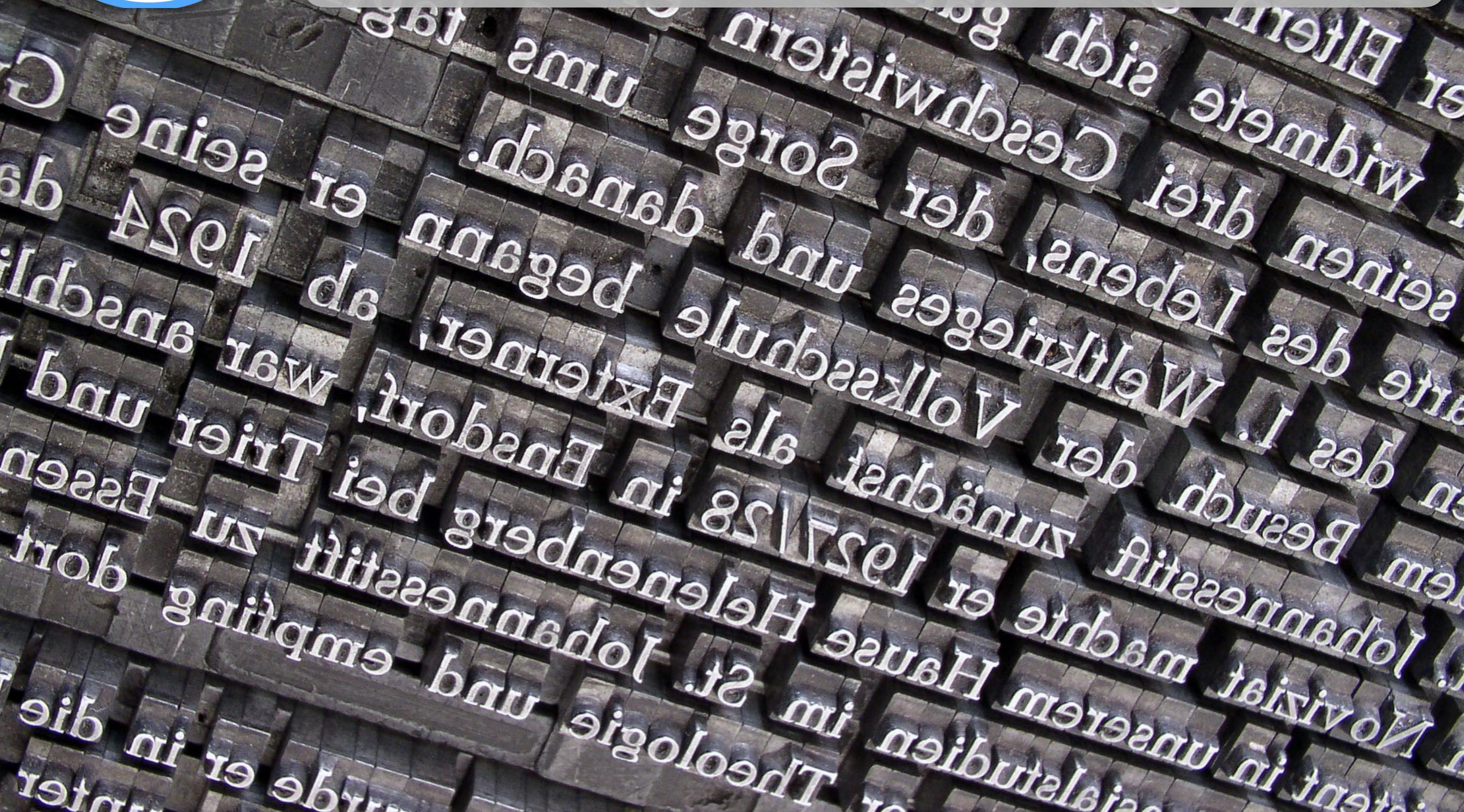


# Natural Language Processing for Everyone

Bruno Gonçalves

[www.data4sci.com/newsletter](http://www.data4sci.com/newsletter)

<https://github.com/DataForScience/NLP>





## Lesson 1: Text Representations



Lesson 1.1:  
Represent words and  
Numbers

# Words and Numbers

- How can computers **represent**, **analyze** and **understand** a piece of text?
- Computers are really good at **crunching numbers** but not so much when it comes to words.
- Perhaps we can substitute words with numbers?
  - Unfortunately, computers assume that numbers are sequential.
- **Vectors** work much better!
  - Each word corresponds to a unique **dimension**.

1	a
2	about
3	above
4	after
5	again
6	against
7	all
8	am
9	an
10	and
11	any
12	are
13	aren't
14	as
...	...



## Lesson 1.2: Use One-hot Encoding

# One-hot Encoding

$$v_{after} = (0, 0, 0, 1, 0, 0, \dots)^T$$
$$v_{above} = (0, 0, 1, 0, 0, 0, \dots)^T$$

One-hot  
encoding

- What about **full texts** instead of single words?
- The vector representation of a text is simply the **vector sum** of all the words it contains:

Mary had a little lamb, little lamb,  
little lamb, Mary had a little lamb  
whose fleece was white as snow.  
And everywhere that Mary went  
Mary went, Mary went, everywhere  
that Mary went  
The lamb was sure to go.

$$v_{text} = (2, 4, 1, 2, 2, 1, 1, 2, 6, 1, 5, 1, 2, 1, 1, 1, 1, 4, 1)^T$$

0	had	10	lamb
1	went	11	as
2	and	12	that
3	a	13	sure
4	was	14	whos
5	to	15	go
6	snow	16	the
7	everywhe	17	little
8	mary	18	white
9	fleece		

# One-hot Encoding

$$v_{after} = (0, 0, 0, 1, 0, 0, \dots)^T$$
$$v_{above} = (0, 0, 1, 0, 0, 0, \dots)^T$$

One-hot  
encoding

- What about **full texts** instead of single words?
- The vector representation of a text is simply the **vector sum** of all the words it contains:

Mary had a little lamb, little lamb,  
little lamb, Mary had a little lamb  
whose fleece was white as snow.  
And everywhere that Mary went  
Mary went, Mary went, everywhere  
that Mary went  
The lamb was sure to go.

$$v_{text} = (2, 4, 1, 2, 2, 1, 1, 2, 6, 1, 5, 1, 2, 1, 1, 1, 1, 4, 1)^T$$

0	had	10	<b>lamb</b>
1	went	11	as
2	and	12	that
3	a	13	sure
4	was	14	whos
5	to	15	go
6	snow	16	the
7	everywhe	17	little
8	<b>mary</b>	18	white
9	fleece		



## Lesson 1.3: Implement Bag of Words

# Bag of Words

- In practice it's much more convenient to use a dictionary instead of an actual vector
- This is known as a bag-of-words, and word order is discarded.

Mary had a little lamb, little lamb,  
little lamb, Mary had a little lamb  
whose fleece was white as snow.  
And everywhere that Mary went  
Mary went, Mary went, everywhere  
that Mary went  
The lamb was sure to go.

$$v_{text} = (2, 4, 1, 2, 2, 1, 1, 2, \textcolor{red}{6}, 1, \textcolor{blue}{5}, 1, 2, 1, 1, 1, 1, 4, 1)^T$$

had	2	<b>lamb</b>	<b>5</b>
went	4	as	1
and	1	that	2
a	2	sure	1
was	2	whos	1
to	1	go	1
snow	1	the	1
everywhe	2	little	4
<b>mary</b>	<b>6</b>	white	1
fleece	1		

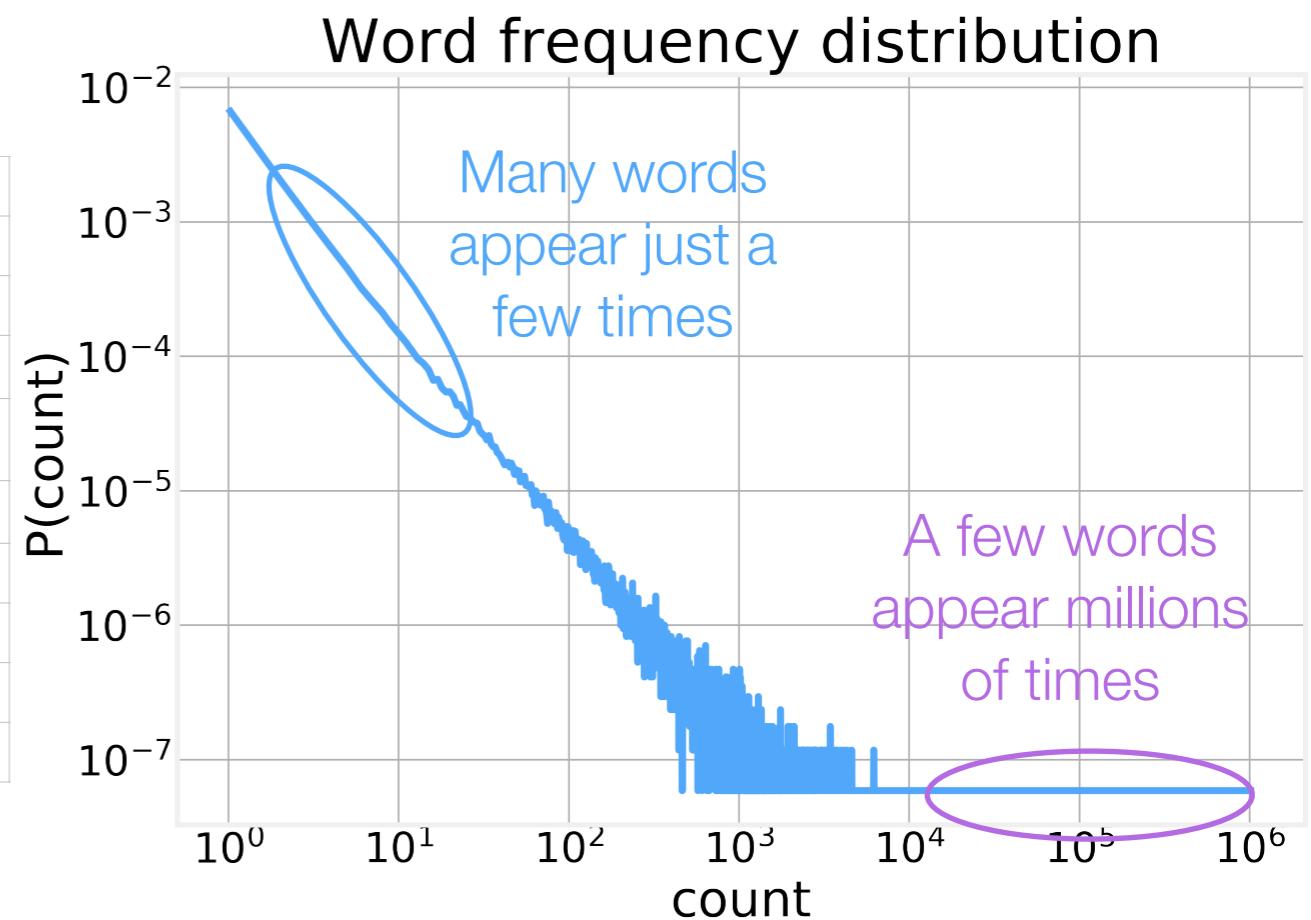


## Lesson 1.4: Apply Stopwords

# Stopwords

- Some words are **much more common** than others.
- While most words are **very rare**.
- The most common words in a corpus of **17M** words:
- These are known as **"stopwords"**, words that carry little meaning and **can be discarded**.

the	1061396
of	593677
and	416629
one	411764
in	372201
a	325873
to	316376
zero	264975
nine	250430
two	192644



# Stopwords

---

- After removing the most common words we go from **17M** words to just **9M**, without significantly losing any information!
- In practice, stopwords aren't simply the most common words but rather curated lists of common and non-informative words.
- Computational linguists have published lists of stop words that can easily be found online, and that were curated for different languages and purposes.
- Stopwords in **40** languages: <https://www.ranks.nl/stopwords>

# Stopwords

---

- **NLTK** also includes stopwords from the **14** languages listed here:

<http://anoncvs.postgresql.org/cvsweb.cgi/pgsql/src/backend/snowball/stopwords/>

plus **Romanian** (<http://arlc.ro/resources/>) and **Kasakh**.

- And perhaps there is a better way to quantify how much information is carried by a word, other than just the number of times it is used in a single document?
- How can we compare different documents?



## Lesson 1.5: Understand TF/IDF

# Term Frequency - Inverse Document Frequency

- We already saw that some words are much more common than others
- The number of times that a word appears in a document is known as the “**term frequency**” (TF)
- After the removal of stopwords, the term frequency is a good indicator of what words are most important. A book on **Python programming** will likely have words like “**code**”, “**script**”, “**print**”, “**error**”, etc much more frequently than a book on **football**.

the	1061396
of	593677
and	416629
one	411764
in	372201
a	325873
to	316376
zero	264975
nine	250430
two	192644

# Term Frequency - Inverse Document Frequency

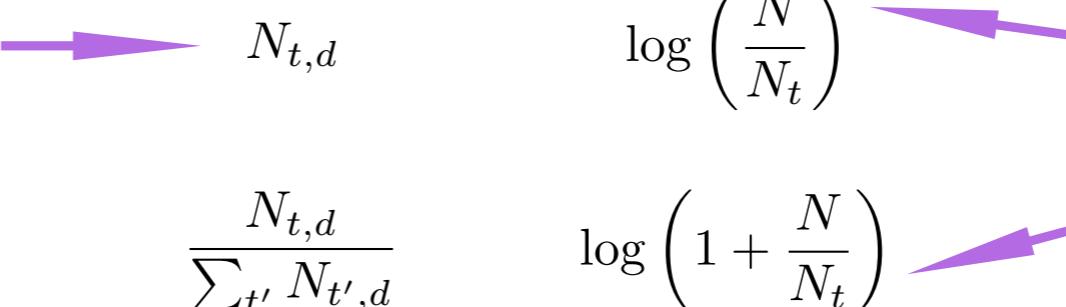
- We already saw that some words are much more common than others
- The number of times that a word appears in a document is known as the “**term frequency**” (TF)
- After the removal of stopwords, the term frequency is a good indicator of what words are most important. A book on **Python programming** will likely have words like “**code**”, “**script**”, “**print**”, “**error**”, etc much more frequently than a book on **football**.
- TF gives us an idea of how popular a specific term is within a document, but how can we **compare across documents** within a corpus?
- The **inverse document frequency** (IDF) tells us how **unusual** it is for a document to include that word. The idea is that words that appear in more documents are **less meaningful**.

the	1061396
of	593677
and	416629
one	411764
in	372201
a	325873
to	316376
zero	264975
nine	250430
two	192644

# TF-IDF

- Mathematically there are several possible definitions for both TF and IDF

	TF	IDF
Number of times term $t$ occurs in document $d$	$N_{t,d}$	$\log\left(\frac{N}{N_t}\right)$
	$\frac{N_{t,d}}{\sum_{t'} N_{t',d}}$	$\log\left(1 + \frac{N}{N_t}\right)$
	$1 + \log(N_{t,d})$	



- TF-IDF is the product of these two quantities and is useful for finding terms that are important for the specific document (high TF) and uncommon in the corpus as a whole (large IDF/small DF).

# TF-IDF

---

- In particular, a term that occurs in **every document is meaningless** when it comes to distinguishing between documents.
- Stopwords, are **naturally weighed down** due to appearing in all documents.



## Lesson 1.6: Understand Stemming

# Word Variants

- So far we have seen multiple techniques to represent words and to reduce the number of words we have to deal with.
- But what about word variants? Verb conjugations, plurals, nouns and adverbs?

love	
loved	
loves	
loving	
lovingly	
...	

- In some cases they should just be represented by the same stem or root.

# Word Variants

- So far we have seen multiple techniques to represent words and to reduce the number of words we have to deal with
- But what about word variants? Verb conjugations, plurals, nouns and adverbs?

love	
loved	
loves	love
loving	
lovingly	
...	

- In some cases they should just be represented by the same **stem** or root.

# Stemming

---

- Stemming is a series of techniques to automatically identify the stem or root of a word
- Naturally, the rules that must be applied depend on the **grammar** of the specific language being used
- For English, the most common algorithm is called **Porter stemmer**

# Porter Stemmer Algorithm

---

- Let  $V$  be a set of one or more **vowels** (a, e, i, o, u, y) and  $C$  be a set of one or more **consonants** (b, c, d, f, ...).
- Any word in the English language is of the form:

$$[C](VC)^m[V]$$

where the contents of the  $[]$  are optional and  $m \geq 0$  is known as the measure, the number of times that the sequence  $VC$  repeats.

# Porter Stemmer Algorithm

---

- The Porter stemmer algorithm develops over the course of several steps, by the successive application of hand-crafted rules of the form:

*(condition) old\_suffix → new\_suffix*

- where **condition** is a boolean expression evaluated on the stem. Each rule can be read as "if **condition** is True, then replace **old\_suffix** by **new\_suffix**.
- Rules are grouped together and applied in a greedy fashion, such that the longest matching **old\_suffix** takes precedence.
- **new\_suffix** can be an empty string and **condition** can be null.

# Porter Stemmer Algorithm

---

- For example, the rule:

$$(m > 0) \ eed \rightarrow ed$$

- would transform

$$\begin{aligned} agreed &\rightarrow agree \\ feed &\rightarrow feed \end{aligned}$$

- since

$$\begin{aligned} measure('agr') &= 1 \\ measure('f') &= 0 \end{aligned}$$

- Of course, not all *conditions* rely only on the value of *measure*

# Porter Stemmer Algorithm

---

- Other expressions commonly used in ***condition*** are:
  - \* **S** - The stem ends with the letter **S** (or any other specified)
  - \* **v** \* - The stem contains a vowel
  - \* **d** - The stem ends in a double consonant (-tt, -ss, etc)
  - \* **o** - The stem ends in cvc where the second c is not W, X, or Y
- Expressions can be combined using the usual boolean operators ***and***, ***or***, ***not***, grouped using parenthesis, etc.
- For example

$$(*d \text{ and not}(*L \text{ or } *S \text{ or } *Z))$$

represents a stem ending with a double consonant other than L, S or Z.



Code - Text Representation  
<https://github.com/DataForScience/NLP>



## Lesson 2: Topic Modeling



## Lesson 2.1:

### Find Topics in Documents

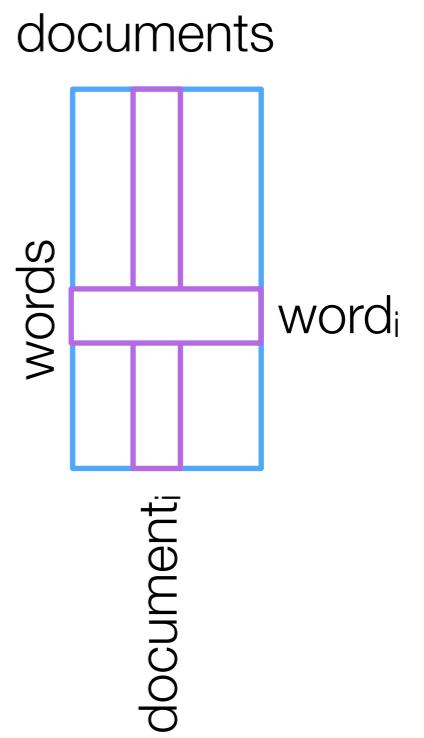
# Topics

---

- A common application of NLP is to Search and Information Extraction.
- Can we automatically define the subject of a document?
- We saw in the previous lesson that some words are more meaningful than others.
- Based on the importance of each word for a specific document, it should be possible to characterize its topic.

# Term-Document Matrix

- We already know how to represent documents in terms of the TFIDF weights of each of their words.
- If we similarly use a vector representation where each element corresponds to a specific word, we can represent a corpus of documents as a matrix where each column is a document.
- Conversely, each word can be thought of as being represented by a row vector defined by its importance over all documents.



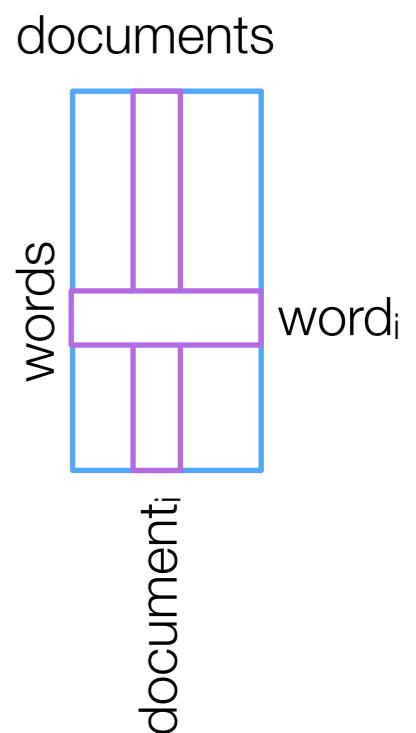


Lesson 2.2:  
Perform Explicit Semantic  
Analysis

# Explicit Semantic Analysis

- In the **term document matrix**, each word is defined, explicitly, by its contribution for each document.
- We can infer the meaning of each word by the concepts (documents) it contributes to.
- Typically, the **English Wikipedia** is used as the knowledge base (corpus).

- Using the TD matrix we can represent any (other) document by the sum (or average) over all the words it contains. This is similar to what we did with one-hot encodings to define bag-of-words.



# Explicit Semantic Analysis

- The similarity of words or new documents can be measured using the [cosine similarity](#):

$$\text{sim}(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|}$$

- [Explicit semantic analysis](#), despite its simplicity, has been shown to improve the performance of different kinds of systems for search.
- The main disadvantage of ESA is that it requires the use of a large knowledge base corpus (the entire English Wikipedia!), resulting in high dimensional representations of words and documents.



Lesson 2.3:  
Understand Document  
clustering

# Document Clustering

---

- Using the cosine similarity:

$$\text{sim}(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|}$$

- we can easily measure the similarity between every pair of documents in our corpus to generate a **square similarity matrix**.
- There is a large literature on clustering algorithms for matrices.
- The simplest approach is to impose a **minimum similarity** cutoff and consider any pair of documents with higher similarity to be part of the same cluster.

# Hierarchical Document Clustering

---

- Algorithm
  - Assign each document to its own cluster.
  - Calculate the similarity matrix between all clusters.
  - Identify the two most similar clusters and merge them.
  - Recompute the similarity matrix for this reduced set of clusters.
  - Repeat until we reach the desired number of clusters.
- Library implementations will usually merge all the clusters until there is just one left so that any cutoff can be imposed a posteriori.



## Lesson 2.4: Implement Latent Semantic Analysis

# Singular Value Decomposition (SVD)

- Any matrix  $\mathbf{M}$ , such as the TD matrix seen above, can be decomposed into three matrices of the form:

$$\boxed{\begin{matrix} \mathbf{M} \\ m \times n \end{matrix}} = \boxed{\begin{matrix} \mathbf{U} \\ m \times m \end{matrix}} \boxed{\begin{matrix} \Sigma \\ m \times n \end{matrix}} \boxed{\begin{matrix} \mathbf{V}^\dagger \\ n \times n \end{matrix}}$$

- where
  - $\mathbf{U}$  is a unitary matrix ( $\mathbf{U}\mathbf{U}^\dagger = \mathbf{I}$ )
  - $\Sigma$  is diagonal with non-negative elements
  - $\mathbf{V}^\dagger$  is a unitary matrix

# Singular Value Decomposition (SVD)

- The diagonal elements of  $\Sigma$ ,  $\sigma_i$  are called the **singular values** of the matrix  $M$  and are conceptually similar to **eigenvalues** in the case of square matrices.
- $\sigma_i$  are sorted from largest to smallest.
- The original matrix  $M$  can be **approximated** by keeping only the **top  $k$  dimensions** of the SVD decomposition.

$$M_k = U_k \begin{matrix} \Sigma_k \\ \vdots \\ \vdots \\ \vdots \end{matrix} V_k^\dagger$$

- Naturally, the larger the value of  $k$  the better the approximation.

# Latent Semantic Analysis (LSA)

---

- While ESA explicit defines each word based on the documents it contributes to in the knowledge base, LSA tries to implicitly determine a "latent" representation for each word and document.
- LSA defines the **term-document** matrix for the corpus under consideration (as opposed to an external knowledge base).
- The DT matrix is approximated using a  $k$  dimensional **singular value decomposition**
- Each of the  $k$  latent dimensions used represents a latent dimension (topic) of the underlying dataset.

# Latent Semantic Analysis (LSA)

---

- The  $\mathbf{U}_k$  represents the distribution of each topic among the words in our vocabulary, while the  $\mathbf{V}_k$  matrix describes the distribution of each document across topics.
- Documents and words can be more effectively clustered in the singular space.
- New documents (or queries) can be mapped into the singular space using:

$$\hat{\mathbf{v}} = \Sigma_k^{-1} \mathbf{U}_k^\dagger \mathbf{v}$$

where  $\mathbf{v}$  is the column vector representing the original query and  $\hat{\mathbf{v}}$  the transformed document vector.



## Lesson 2.5:

### Implement Non-negative Matrix Factorization

# Non-Negative Matrix Factorization

- Matrix factorization methods, like SVD, have a long history of application to **natural language processing**.
- Another common factorization method used to identify a **latent structure** to a dataset is **non-negative matrix factorization** (NMF).

$$\boxed{\begin{matrix} \mathbf{M} \\ m \times n \end{matrix}} = \boxed{\begin{matrix} \mathbf{W} \\ m \times k \end{matrix}} \boxed{\begin{matrix} \mathbf{H} \\ k \times n \end{matrix}}$$

- NMF directly approximates the matrix  $\mathbf{M}$  for a given value of  $k$ .

# Non-Negative Matrix Factorization

- In this formulation, it has the advantage of being easily interpretable:
  - Columns of  $\mathbf{W}$  are the underlying basis vectors for each topic.
  - Columns of  $\mathbf{H}$  are the contribution of each topic to a specific document.
- The value of  $\mathbf{W}$  and  $\mathbf{H}$  are found through an optimization procedure where we minimize the error of the approximation:
- One simple way to do this is given by:

$$h_{ij} \leftarrow h_{ij} \frac{(W^\dagger V)_{ij}}{(W^\dagger W H)_{ij}}$$
$$w_{ij} \leftarrow w_{ij} \frac{(V H^\dagger)_{ij}}{(V H H^\dagger)_{ij}}$$

- And we initiate the process by assigning random non-negative values to  $\mathbf{W}$  and  $\mathbf{H}$ .
- We stop updating the values of  $h_{ij}$  and  $w_{ij}$  when the cost function converges.



Code - Topic Modeling  
<https://github.com/DataForScience/NLP>



## Lesson 3: Sentiment Analysis



## Lesson 3.1:

### Quantify words and feelings

# Positive and Negative Words

---

- We often use words to describe how we are **feeling** or how we feel about something
- People associate **specific** connotations and meanings to words
- Some are obvious:
  - Love, yes, friendship, good, ... transmit **positive** feelings
  - Hate, no, animosity, bad, ... transmit **negative** feelings
- Others less so:
  - Blue, maybe, indifference, ...

# Positive and Negative Texts

---

- We can use the words in a text to determine the sentiment behind the text
- The simplest approach:
  - count **positive**  $P$  and **negative**  $N$  words
  - define sentiment as:
- This effectively weighs **positive** words as **+1** and **negative** words as **-1** and defines sentiment as how dominant one sentiment is over another
- Despite its simplicity, this approach is surprisingly powerful

$$\text{sentiment} = \frac{P - N}{P + N}$$

# Word Valence

---

- Many lexicons of **positive** and **negative** words are available online:
  - Opinion Lexicon: <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>
  - Opinion Finder: <https://mpqa.cs.pitt.edu/opinionfinder/>
- Even commercial products use similar approaches:
  - Linguistic Inquiry and Word Count (LIWC) <http://liwc.wpengine.com/>
  - Valence Aware Dictionary and sEntiment Reasoner (VADER) <https://github.com/cjhutto/vaderSentiment>



## Lesson 3.2:

### Use negations and modifiers

# Negations and Modifiers

---

- So far, our approach to sentiment analysis has relied on considering just **individual words**
- However it's clear that context matters.
- "not pretty" is very different from "pretty"
- **n-grams** are a natural extension, but increase significantly the memory requirements
- An intermediate approach is to consider modifier words like "not", "much", "little", "very", etc.
- By keeping track of the modifiers we can generate n-grams on the fly

# Modifiers

---

- While sentiment words contribute **additively** to the sentiment score, modifiers have a **multiplicative** effect
- If we define the weights of each modifier

not	-1
very	1.5
somewhat	1.2
pretty	1.5
...	...

- We just have to check the previous word whenever we encounter one of the sentiment words in our lexicon
- Special care must be taken whenever a modifier word can also be a sentiment word (such as "pretty").

# Modifiers

---

- We must keep two separate dictionaries of words: **modifiers** and **valence** words.
- For each word we encounter, we first check whether it is a **modifier** and only then whether it is **valence** word.
- Words that are in neither list act as a signal to end the current n-gram
- With this simple approach we can handle even long sequences of modifiers, double negatives, etc.



## Lesson 3.3: Understanding corpus-based approaches

# Corpus-based Approaches

---

- Sentiment analyses often rely on dictionaries of words and valences
- In many cases, these lists are curated manually with varying degrees of care
- Can we automatically generate them?
- With the advent of the Web, large corpora of product reviews became available
- Each review associates a piece of text with a numerical evaluation (typically 1-5 stars)

# Corpus-based Approaches

---

- Corpus based approaches to sentiment analysis rely on these datasets to generate the lexicons used
- These approaches leverage sophisticated **supervised machine learning** techniques to automatically determine the weight that should be assigned to each word
- Modifiers can be identified using **Part-of-Speech (POS)** tagging
- The details of these techniques are beyond the scope of this introductory course, but it's important to understand their advantages and disadvantages

# Corpus-based Approaches

---

- Hand curated lexicons are naturally **subjective** and potentially **incomplete**
- Lexicons generated automatically from large corpora can cover a **wider range of languages** and subjects
- Hand curation is more powerful when only **smaller datasets** are available
- Automatic generation relies on **large datasets** and their inherent biases. They are typically only applicable to specific domains and may **appear** to be more **objective**.
- Your results will only be as good as your lexicon. Make sure to understand its **biases** and **limitations**.



Code - Sentiment Analysis  
<https://github.com/DataForScience/NLP>



## Lesson 4: Applications



Lesson 4.1:  
Understand word2vec word  
embeddings

# Word Embeddings

- We already saw various way in which to represent word in a **vectorial form**, but never in a way that was semantically meaningful.
- The distributional hypothesis in linguistics states that words with **similar meanings** should occur in **similar contexts**.
- In other words, from a word we can get some idea about the context where it might appear.



$$\max p(C|w)$$

- And from the context we have some idea about possible words.

The red \_\_\_\_\_ is beautiful.  
The blue \_\_\_\_\_ is old.

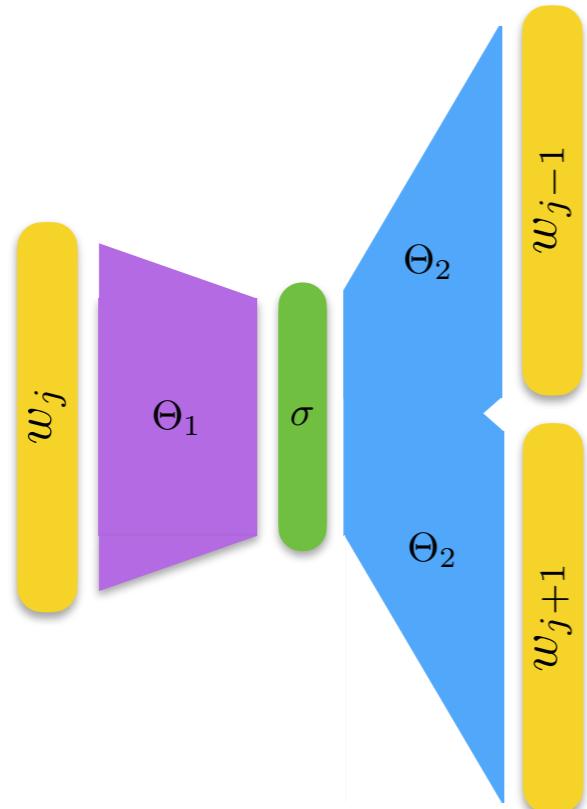
$$\max p(w|C)$$

# word2vec

Mikolov 2013

## Skipgram

$$\max p(C|w)$$

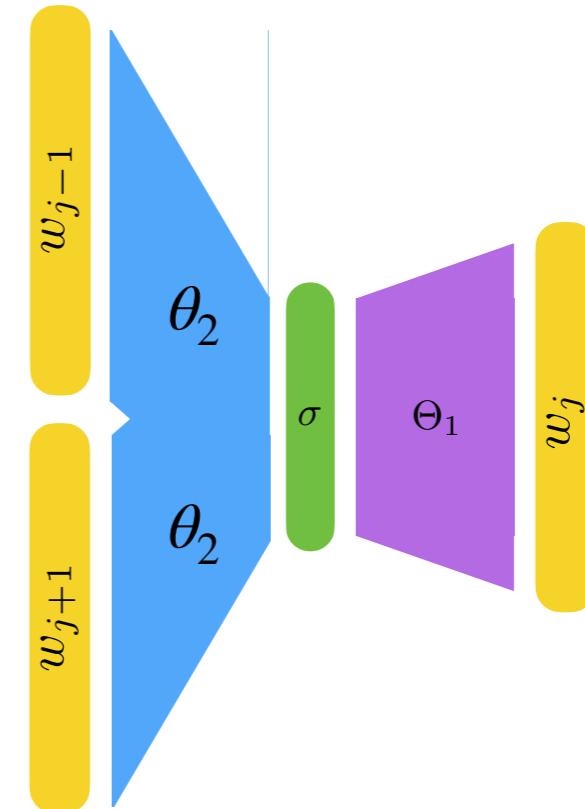


Word

Context

## Continuous Bag of Words

$$\max p(w|C)$$



Context

Word

# Variations

---

- Hierarchical Softmax:
  - Approximate the **softmax** using a binary tree
  - Reduces the number of calculations per training example from  $V$  to  $\log_2 V$  and increases performance by orders of magnitude.
- Negative Sampling:
  - Under sample the most frequent words by removing them from the text **before** generating the contexts
  - Similar idea to removing **stop-words** — very frequent words are less informative.
  - Effectively makes the window larger, increasing the amount of information available for context

# word2vec details

---

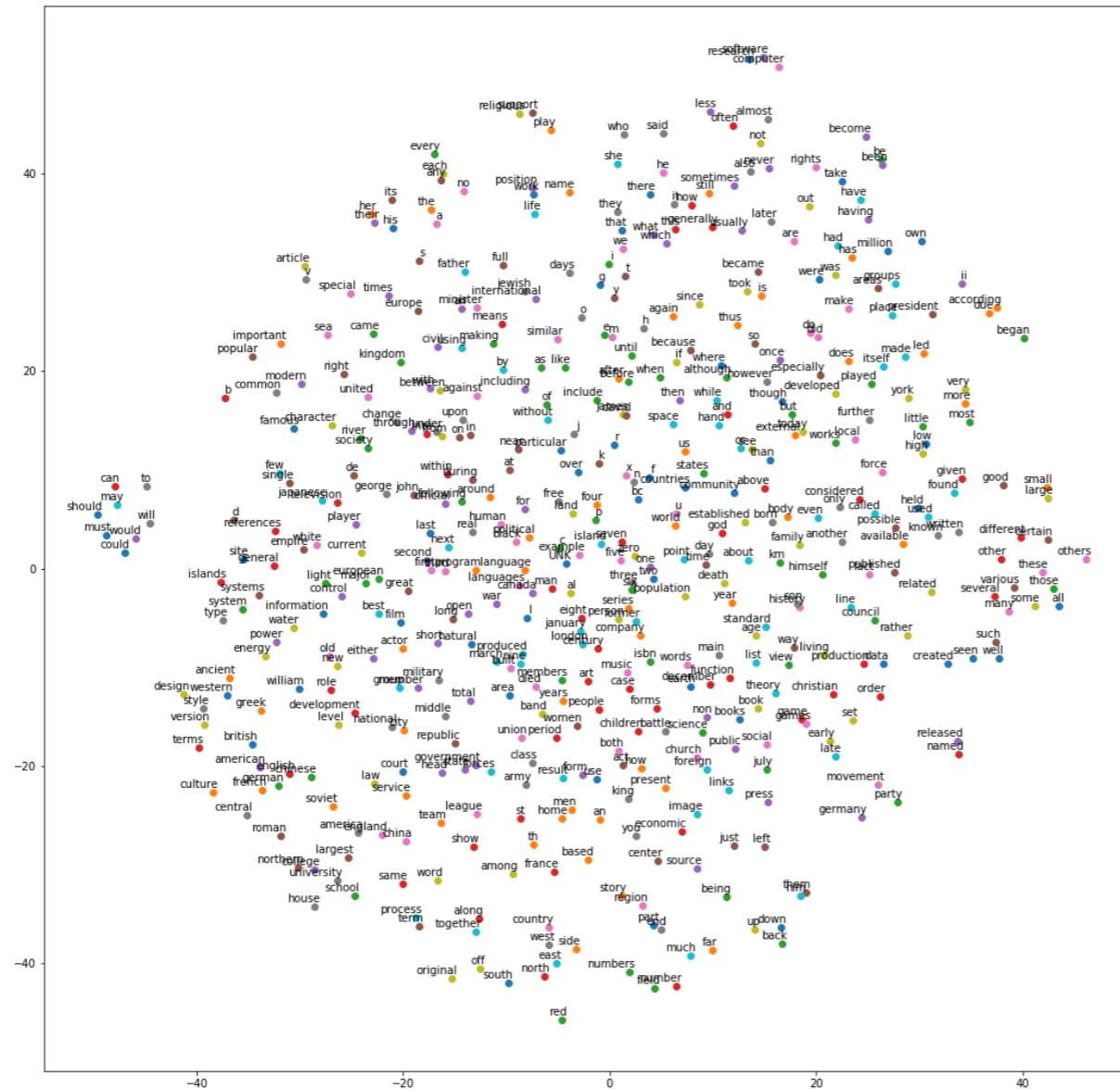
- The output of this neural network is deterministic:
  - If two words appear in the same context ("blue" vs "red", for e.g.), they will have similar internal representations in  $\theta_1$  and  $\theta_2$
  - $\theta_1$  and  $\theta_2$  are vector embeddings of the input words and the context words respectively
- Words that are too rare are also removed.
- The original implementation had a dynamic window size:
  - for each word in the corpus a window size  $k'$  is sampled uniformly between 1 and  $k$

# Online resources

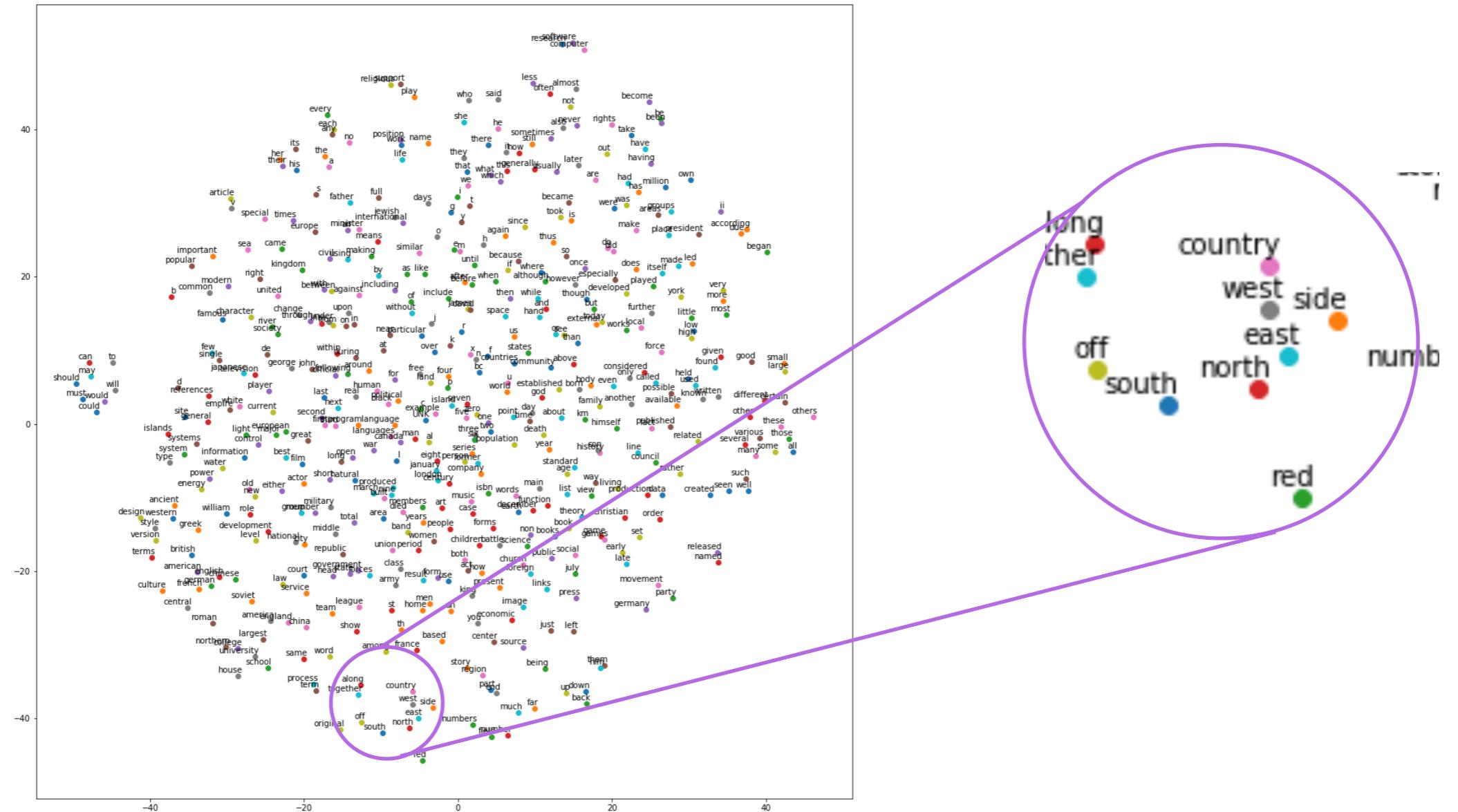
---

- C - <https://code.google.com/archive/p/word2vec/> (the original one)
- Python/tensorflow - <https://www.tensorflow.org/tutorials/word2vec>
- Python/gensim - <https://radimrehurek.com/gensim/models/word2vec.html>
- Pretrained embeddings:
  - 30+ languages, <https://github.com/Kyubyong/wordvectors>
  - 100+ languages trained using wikipedia: <https://sites.google.com/site/rmyeid/projects/polyglot>

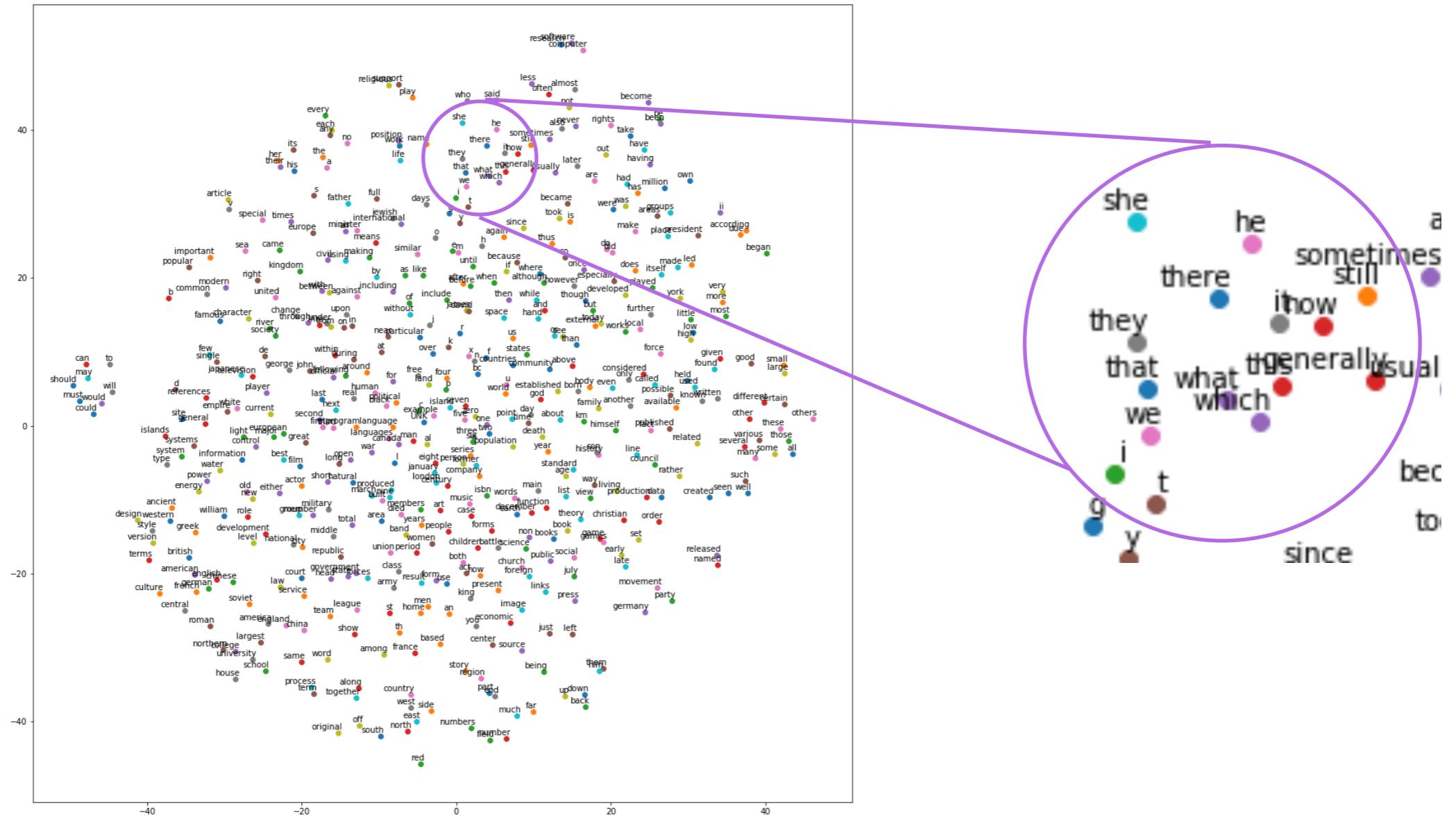
# Visualization



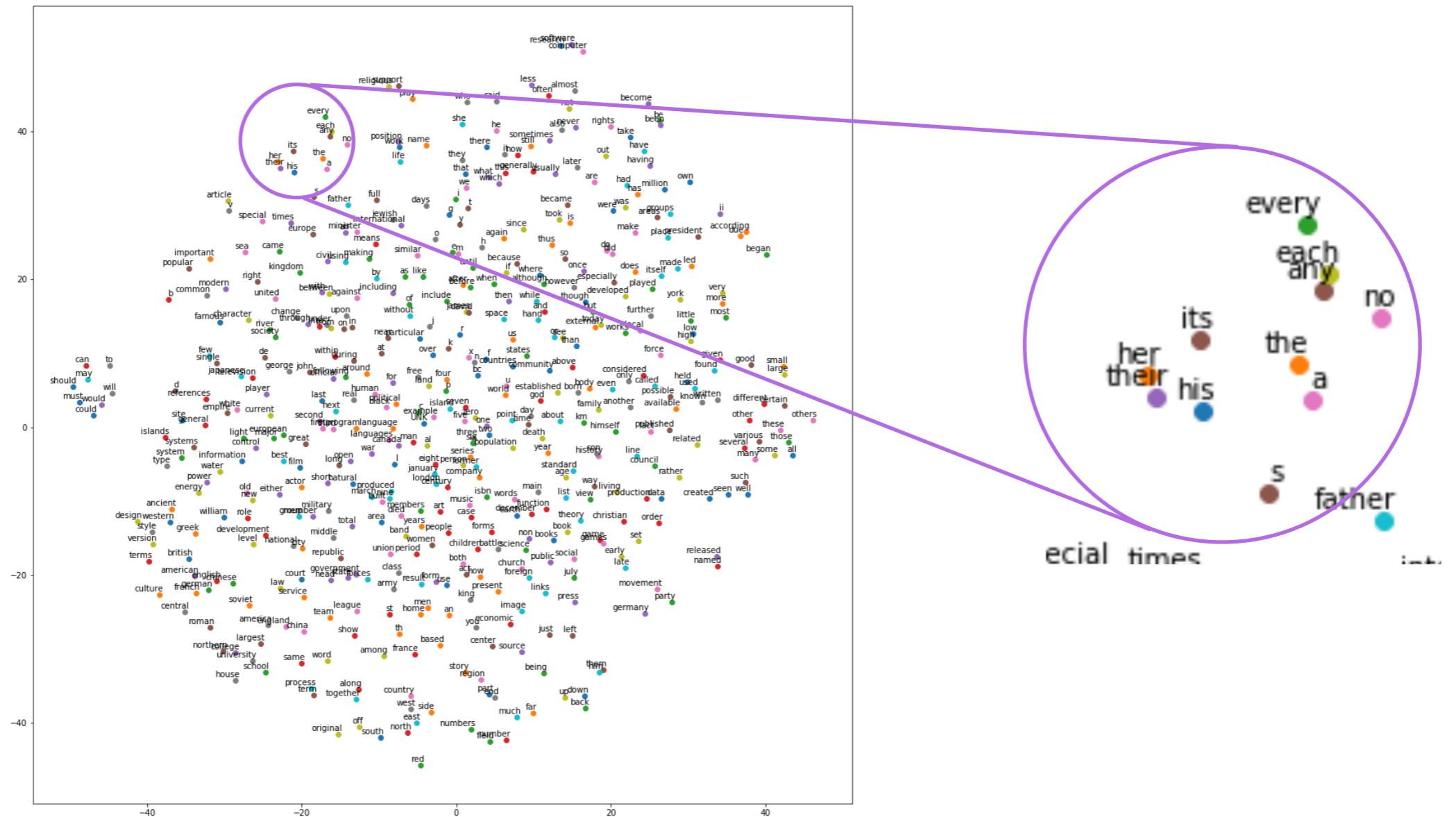
# Visualization



# Visualization



# Visualization



# Visualization



# Analogies

---

- The embedding of each word is a function of the context it appears in:

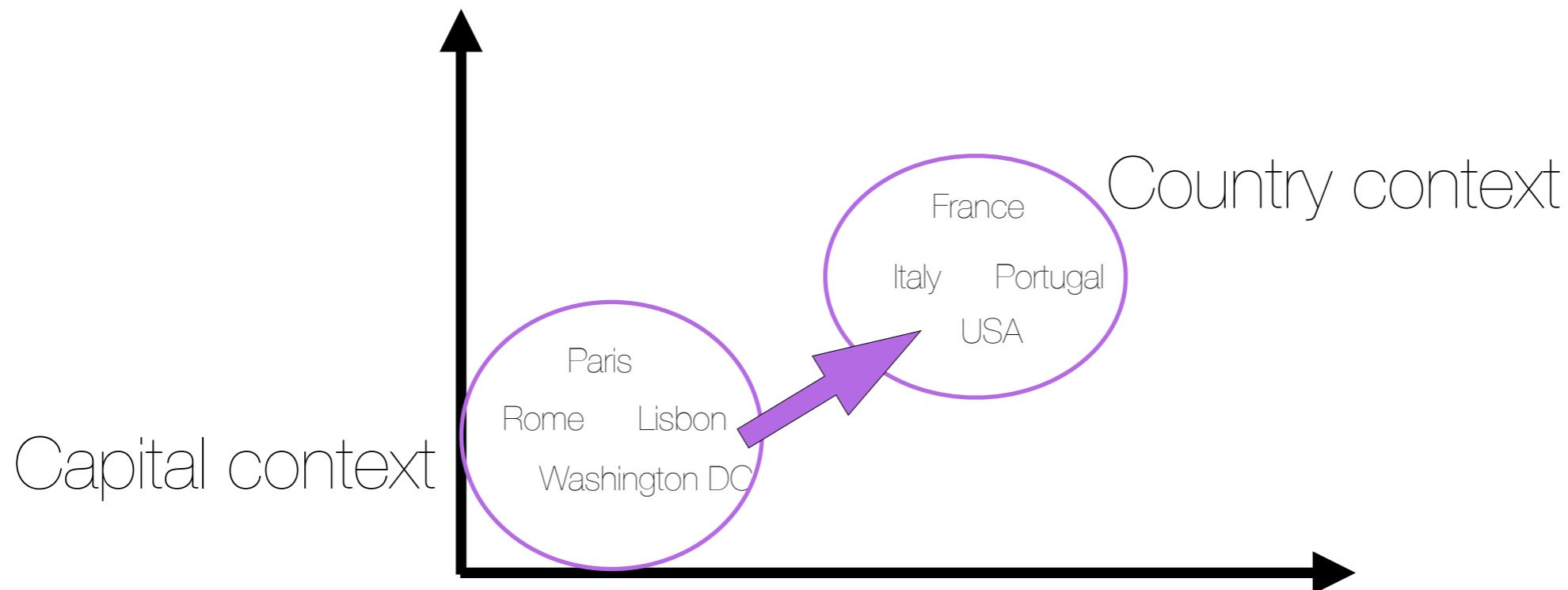
$$\sigma(red) = f(\text{context}(red))$$

- words that appear in similar contexts will have similar embeddings:

$$\text{context}(red) \approx \text{context}(blue) \implies \sigma(red) \approx \sigma(blue)$$

- “**Distributional hypothesis**” in linguistics

# Analogies



$$\sigma(France) - \sigma(Paris) = \sigma(Italy) - \sigma(Rome)$$

$$\sigma(France) - \sigma(Paris) + \sigma(Rome) = \sigma(Italy)$$

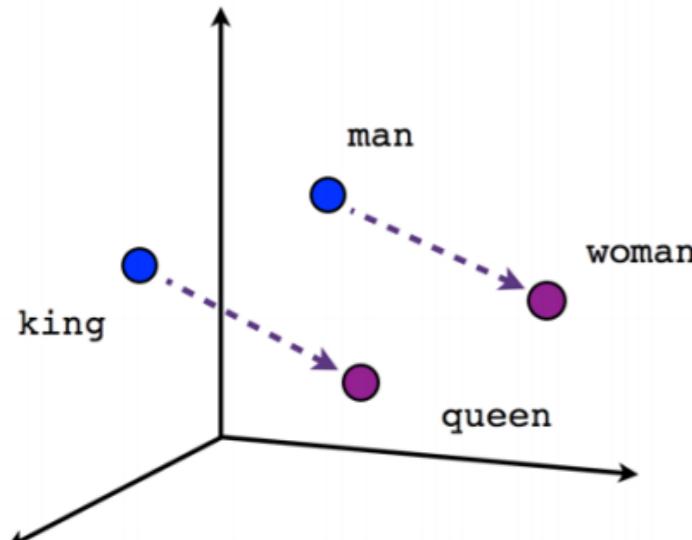
$$\vec{b} - \vec{a} + \vec{c} = \vec{d}$$

# Analogies

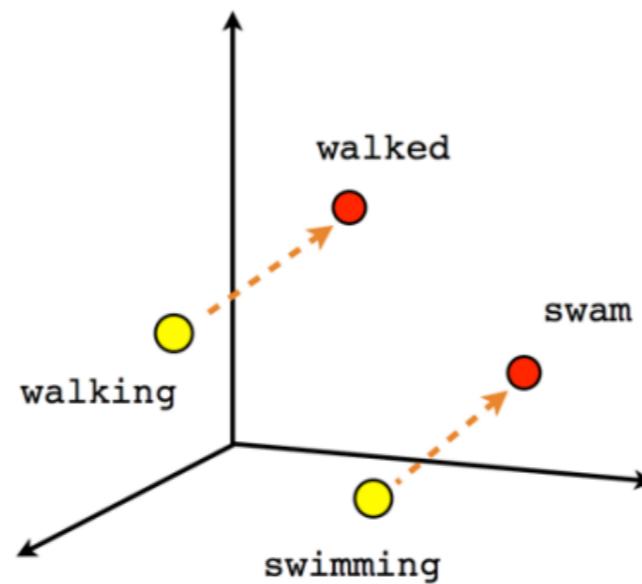
$$\vec{b} - \vec{a} + \vec{c} = \vec{d}$$

$$d^\dagger = \operatorname{argmax}_x \frac{\left( \vec{b} - \vec{a} + \vec{c} \right)^T}{\| \vec{b} - \vec{a} + \vec{c} \|} \vec{x}$$

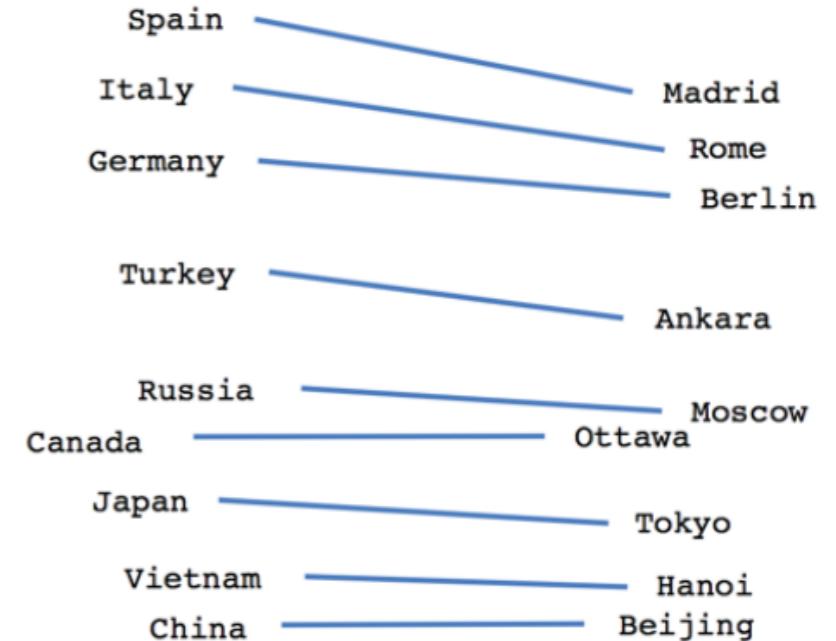
$$d^\dagger \sim \operatorname{argmax}_x \left( \vec{b}^T \vec{x} - \vec{a}^T \vec{x} + \vec{c}^T \vec{x} \right)$$



Male-Female



Verb tense



Country-Capital



## Lesson 4.2: Define GloVe

# Global Vectors (GloVe)

- An alternative to [word2vec](#) developed by the Stanford NLP Group in 2014
- Explicitly models word **cooccurrences** by a cooccurrence matrix  $\mathbf{X}$  where rows correspond to input words and columns to context words
- $X_{ij}$  is the number of times word  $i$  occurred with context word  $j$
- Contexts are defined through a sliding window
- Embedding vectors for word  $i$  is defined as:  $w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$
- Embedding vectors are found through an optimization procedure with a cost function of the form:

$$J = \sum_{i,j=1}^V f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij}) \right)^2$$
$$f(X_{ij}) = \begin{cases} \left( \frac{X_{ij}}{X_{max}} \right) & \text{if } X_{ij} < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

# Global Vectors (GloVe)

---

- $f(X_{ij})$  prevents common word pairs from skewing our cost function
- Explicitly **considers the context** in which each word appears
- Word-context matrix is **large, but sparse**
- Relatively fast to train
- Highlights the importance of relative frequency of word
- Impractical to use as a language model (to explicitly calculate the value of  **$\max p(w|C)$** )
- Resources:
  - The original implementation: <https://nlp.stanford.edu/projects/glove/>
  - Python package: <https://pypi.org/project/glove/>
  - Pre-trained vectors: <https://github.com/stanfordnlp/GloVe>



## Lesson 4.3: Apply Language detection

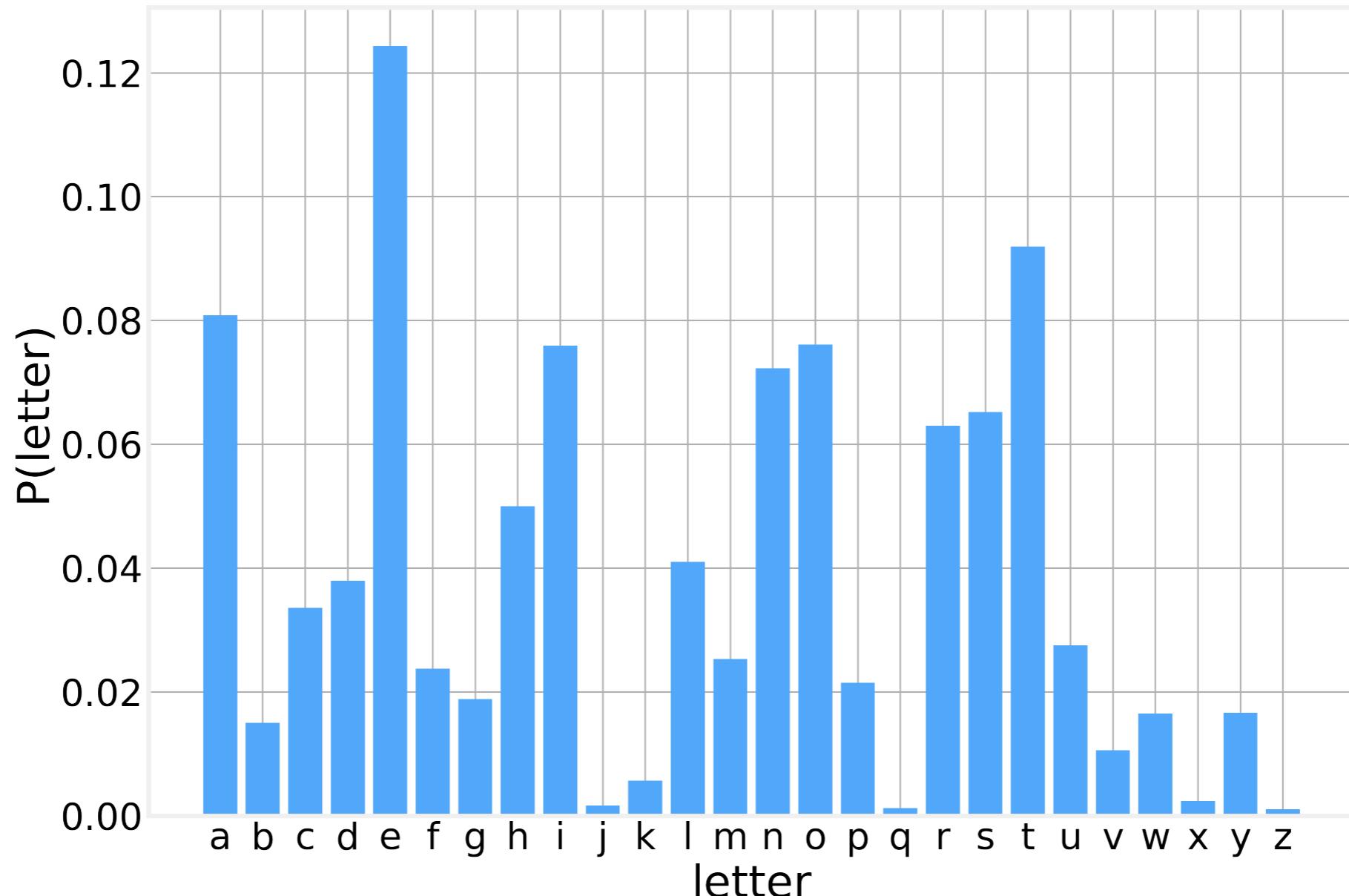
# Language Detection

---

- Sometimes a given corpus will contain documents written in **different languages** (comment boxes in international websites, for example)
- We intrinsically assumed that each **individual documents** is comparable with all others.
- When multiple languages are present within the same corpus (on Twitter, in the comment boxes of an international website, etc) this is no longer true.
- One way to handle this would be to **cluster documents**, as documents in the same language will naturally cluster together. However:
  - there might be multiple clusters using the same language
  - We might not be interested in doing all this work for languages other than, say, English
- **Language detection** allows us to preprocess our corpus so that we can focus on specific languages, sort documents by language (to use different **stopword** lists), etc.
- Languages can be characterized by their **character** (letter) **distribution**.

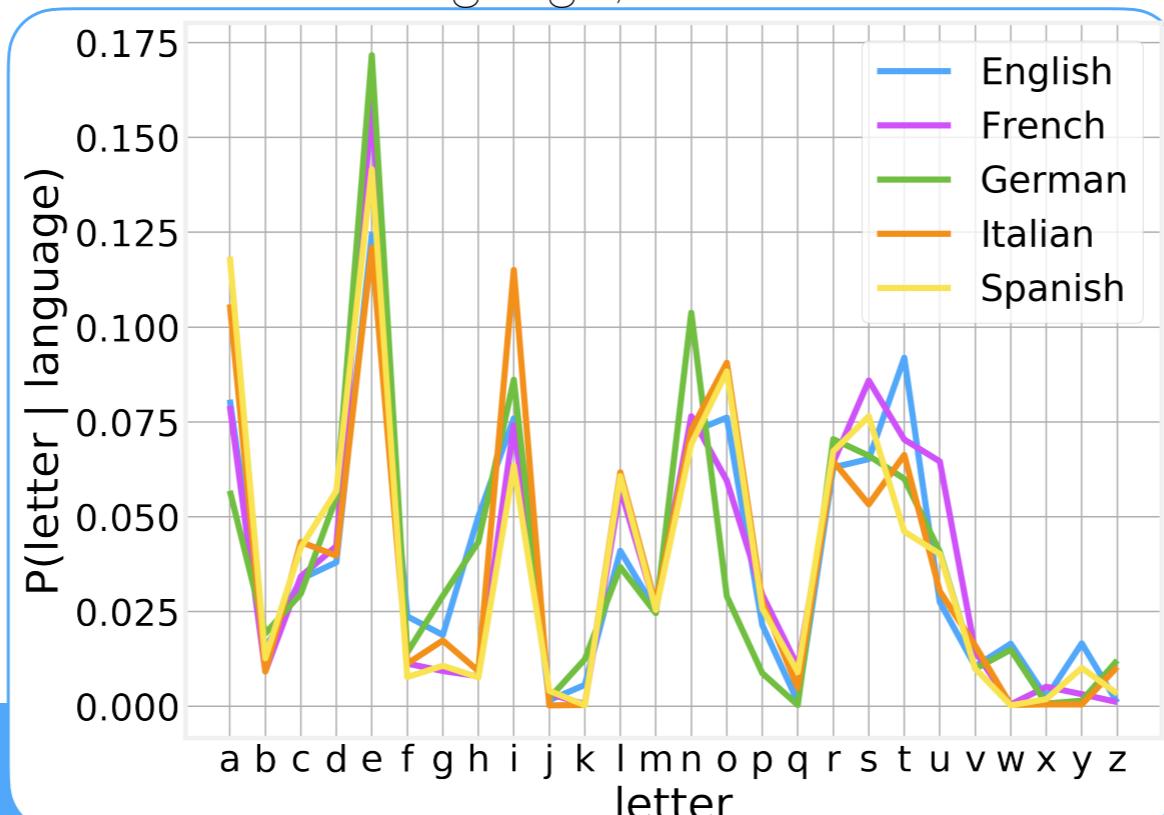
# Character Probabilities

- The character level distribution for English, obtained using Google Books [1-gram dataset](#) is:



# Language Detection

- We measured the probability distribution of letters in the english language. In effect, we calculated:  $P(\text{letter} | \text{english})$
- The probability of seeing a specific letter given that the text is in English. If we do this for a few other languages we can have a table of the form:  $P(\text{letter} | \text{language})$
- Google Books covers several different languages, among which we find 5 different European languages: English, French, German, Italian and Spanish.
- Character distributions for different languages look different in at least a few of the characters due to the idiosyncrasies of each language, even in the case of closely related languages.



# Conditional Probabilities

- Using these conditional probabilities, and Bayes Theorem, we can easily build a language detector. For that we just need to calculate:

$$P(\text{language} | \text{text})$$

- Which we can rewrite as:

$$P(\text{language} | \text{letter}_1, \text{letter}_2, \dots, \text{letter}_n)$$

- If we treat each letter independently, we obtain:

$$P(\text{language} | \text{letter}_1, \text{letter}_2, \dots, \text{letter}_n) = \prod_i P(\text{language} | \text{letter}_i)$$

- This is known as the **Naive Bayes Approach** and is an obvious oversimplification: It completely ignores correlations present in the sequence of letters.
- All we have to do now is apply Bayes Theorem to our original table:

$$P(\text{language} | \text{letter}) = \frac{P(\text{letter} | \text{language}) P(\text{language})}{P(\text{letter})}$$

# Naive Bayes

- And if we assume that all languages are equally probable (**non-informative prior**):

$$P(\text{language}) = \frac{1}{N_{langs}}$$

- Naive Bayes approaches (and many others) use terms of the form:

$$\prod_i P(A | B_i)$$

- which implies multiplying many **small** numbers. To avoid numerical complications, it is best to use, instead:

$$\sum_i \log P(A | B_i)$$

- Which is commonly referred to as the "**Log-Likelihood**". Our expression then becomes:

$$\mathcal{L}(\text{language} | \text{letter}_1, \text{letter}_2, \dots, \text{letter}_n) = \sum_i \log \left[ \frac{P(\text{letter}_i | \text{language}) P(\text{language})}{P(\text{letter}_i)} \right]$$

# Naive Bayes

---

- Or more simply:

$$\mathcal{L}(\text{language} | \text{text}) = \sum_i \log \left[ \frac{P(\text{letter}_i | \text{language}) P(\text{language})}{P(\text{letter}_i)} \right]$$

- And finally:

$$\mathcal{L}(\text{language} | \text{text}) = \sum_i \mathcal{L}(\text{language} | \text{letter}_i)$$

- Providing us with a quick and easy way to determine which language is more likely to be the correct one.

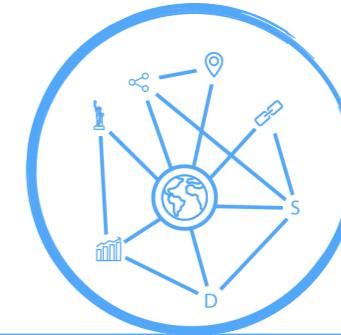


Code - Applications  
<https://github.com/DataForScience/NLP>

# Events



<http://paypal.me/data4sci>



[www.data4sci.com/newsletter](http://www.data4sci.com/newsletter)

## NLP with Deep Learning for Everyone

Jan 27, 2021 - 5am-9am (PST)

## Applied Probability Theory for Everyone

Feb 11, 2021 - 5am-9am (PST)

## Transforming Excel Analysis into Python and Pandas

Feb 25, 2021 - 5am-9am (PST)

## Graphs and Network Algorithms for Everyone

<https://graphs4sci.substack.com/> - Blog

Mar 9, 2021 - 5am-9am (PST)

## Why and What If – Causal Analysis for Everyone

Mar 25, 2021 - 5am-9am (PST)

## Natural Language Processing (NLP) from Scratch

<http://bit.ly/LiveLessonNLP> - On Demand