

Natural Language Processing (NLP)

Bag of Words, TF-IDF

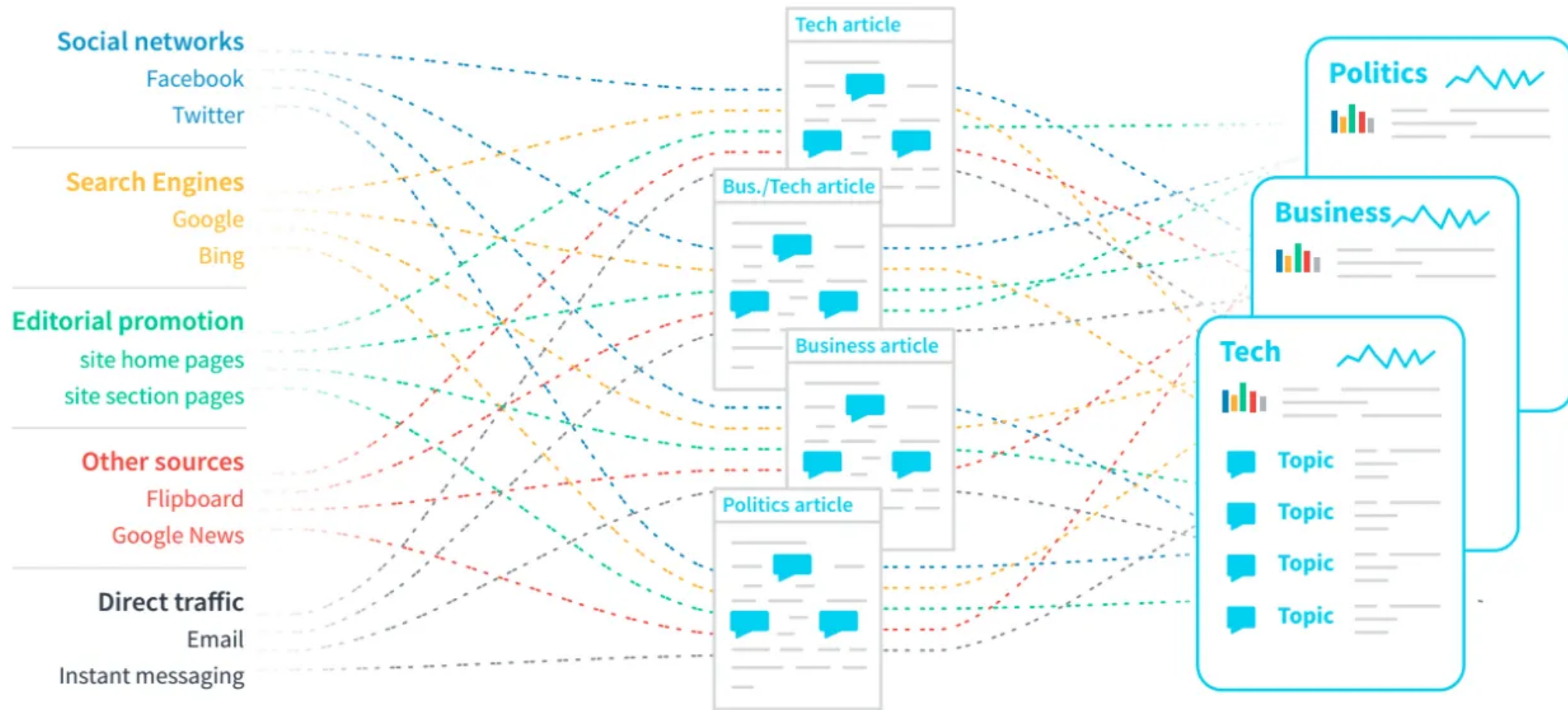
Language

- Language is our medium for communication and knowledge transfer.
- There are many languages with their own structures, alphabets and grammars.
- The text of a language is constructed for the human brain, not for computers.
- The majority of texts are unstructured.
- Less than 20% have the structure of even Tweets, Facebook or email.

Text Mining

- Preprocessing, feature engineering and analytics for textual data.
- Transforming unstructured data into normalized structured data for analysis or machine learning.
- The process of analyzing large collections of text to discover new information or answer specific questions.

Text Mining



Traffic sources

Online content

Categories & topics

Natural Language Processing

- A component of text mining, NLP helps computers understand, analyze, manipulate and potentially generate human language.
 - Siri, Alexa and OK Google use NLP to receive and respond to user requests.
 - NLP applications have also been developed in medical research, customer services, fraud detection in insurance industry, contextual advertising, financial analysis, detect spam emails, autocorrect, etc
 - Real-time massive news sentiment analysis and fake news detection
 - Topic Analysis and Clustering
 - Spam Filtering: Detect unsolicited and unwanted email/messages.
 - Language Translation: Translate a sentence from one language to another.


Email Suggestion:

Sandy Writtenhouse
Fri 6/11/2021 9:01 AM
To: You; Sandy Writtenhouse

Do you have the file from our discussion on Monday?


Should we set up a meeting for next Tuesday?

Thanks



 Schedule a meeting Yes, that would be great. **Yes, we do.**







groovyPost.com

Do you have the file?

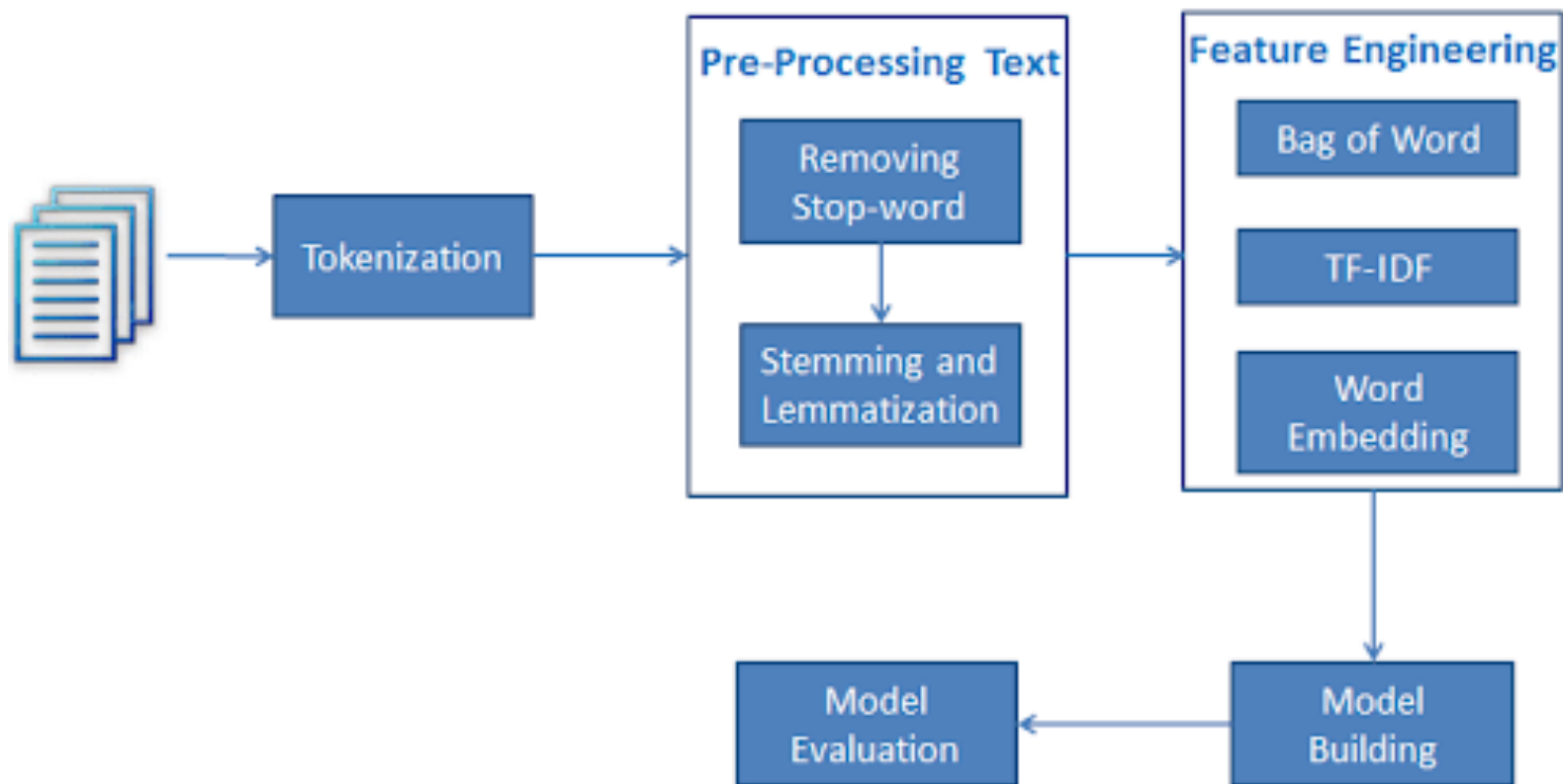
 To: Sandy Writtenhouse;

Yes, we do.

Send |  **Discard**      ...

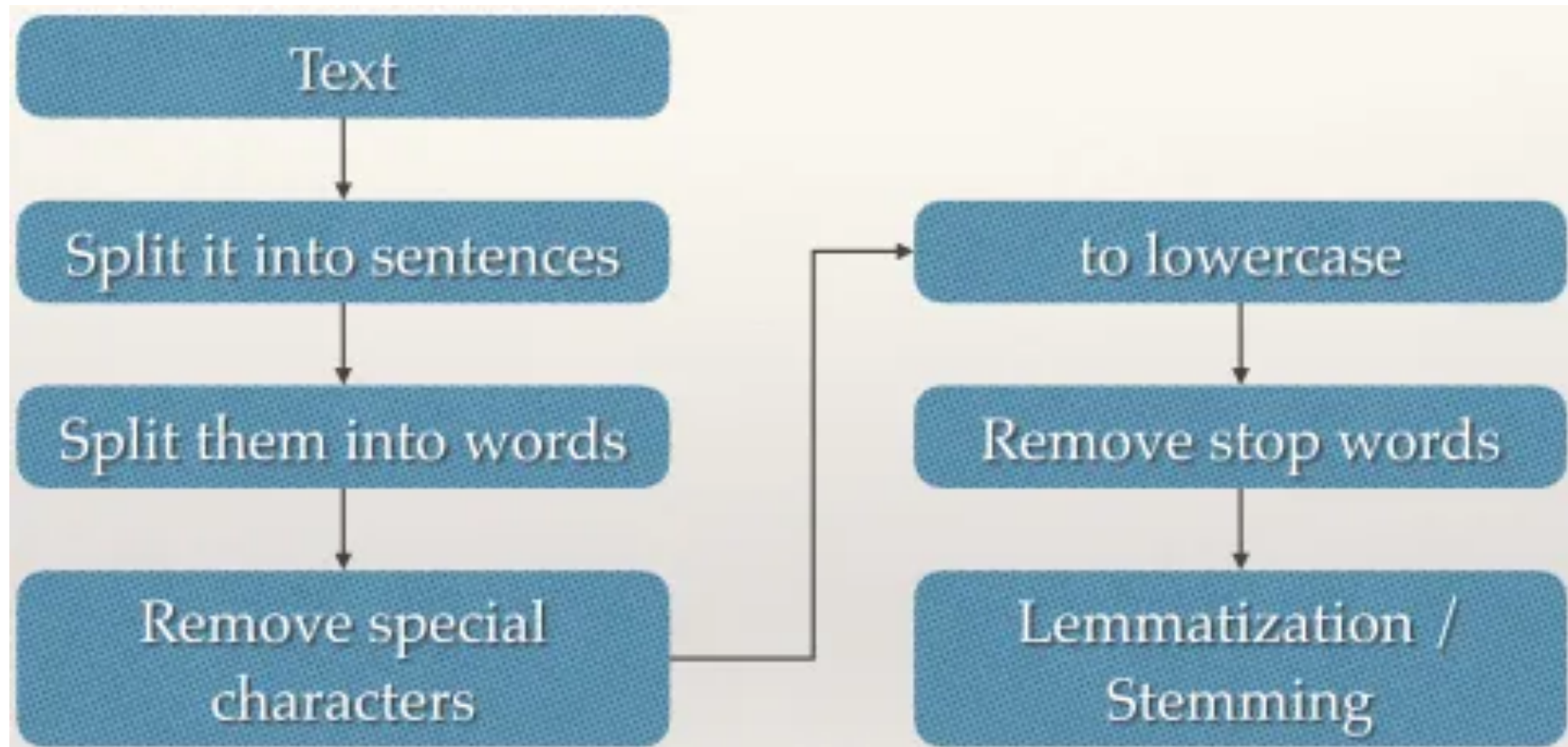
NLP Steps



Natural Language Toolkit (NLTK) and other tools

- open-source package in Python.
- provides all common NLP Tasks from tokenization, stemming, tagging, parsing, and beyond.
- Other useful tools
 - *BeautifulSoup*: Library for extracting data from HTML and XML documents
 - *Stanford Core Text*
 - *OpenNLP*

The general approach to preprocessing text data with NLTK



Preprocessing Data 1: Cleaning up and structuring

Remove punctuation

- Punctuation can provide grammatical context to a sentence which supports our understanding.
 - Classic Chinese doesn't have punctuation. It indicates that it is helpful but not absolutely necessary.
- But for our vectorizer which analyzes words with limited context, it does not add value.
- So we remove all special characters.

Preprocessing Data 1: Cleaning up and structuring

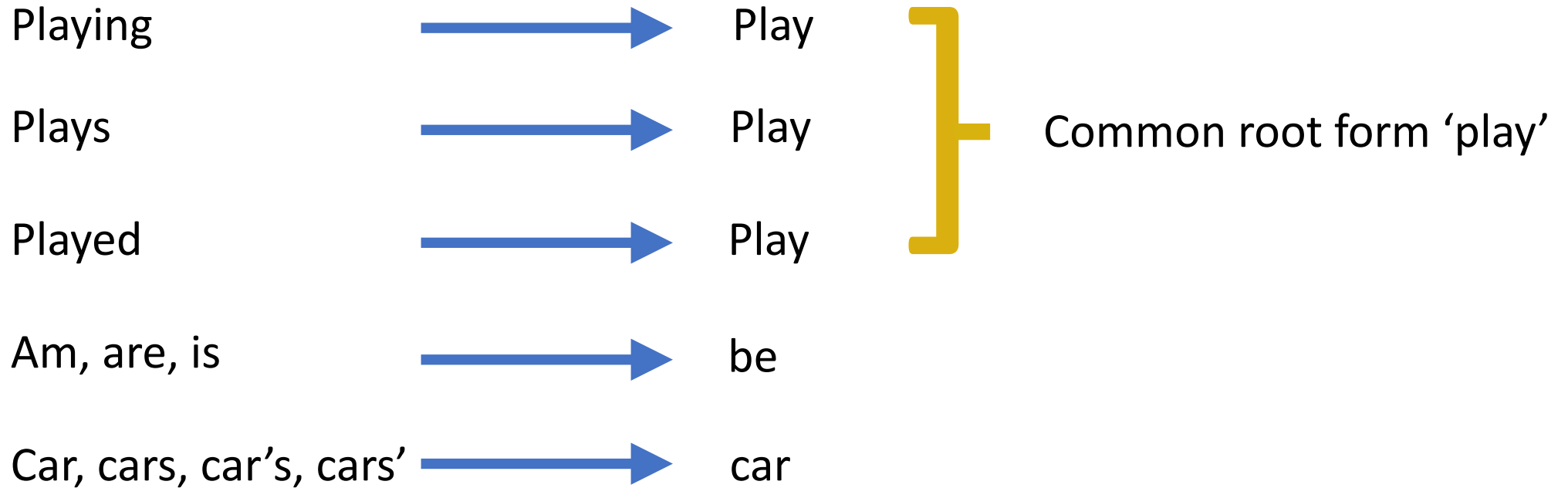
Tokenization

- Tokenizing separates text into units such as sentences or words.
- It generates unstructured text as a list.
- It is also referred to as text segmentation or lexical analysis.
- The general approach works well for any language that uses space as delimiter.

Remove stopwords

- Stopwords are common words that will likely appear in any text.
- They usually don't have much useful information of our data so we remove them

Preprocessing Data 2: Stemming and Lemmatization



Using the above mapping, a sentence could be normalized as:

The boy's cars are different colors → the boy car be differ color

Preprocessing Data 2: Stemming

- A simple rule-based approach
 - It reduces the corpus of words but often the actual words get neglected. e.g.:
Entitling, Entitled -> Entitl
- Stemming helps reduce a word to its stem form.
- It often makes sense to treat related words in the same way.
- It removes suffices, like “ing”, “ly”, “s”, etc
- NLTK provides several stemmer interfaces like
 - Porter stemmer
 - Lancaster Stemmer
 - Snowball Stemmer

Preprocessing Data 2: Stemming

PorterStemmer

- Most commonly used stemmer
- PorterStemmer uses Suffix Stripping to produce stems
- Notice how the PorterStemmer is giving the root (stem) of the word "algorithms" by simply removing the 's' after "algorithm"
 - This is the reason why PorterStemmer does not often generate stems that are actual English words.
- It does not keep a lookup table for actual stems of the word but applies algorithmic rules to generate stems.
- PorterStemmer is known for its simplicity and speed but not very precise.

Preprocessing Data 2: Stemming

LancasterStemmer

- The LancasterStemmer (Paice-Husk stemmer) is an iterative algorithm with rules saved externally.
- LancasterStemmer is significantly more aggressive than the porter stemmer
- *Algorithm:*
 - One table containing about 120 rules indexed by the last letter of a suffix.
 - On each iteration, it tries to find an applicable rule by the last character of the word.
 - Each rule specifies either a deletion or replacement of an ending.
 - If there is no such rule, it terminates.
 - It also terminates if a word starts with a vowel and there are only two letters left or if a word starts with a consonant and there are only three characters left.
 - Otherwise, the rule is applied, and the process repeats.

Preprocessing Data 2: Stemming

Snowball Stemmer

- The actual name of this stemmer is English Stemmer or Porter2 Stemmer
- There were some improvements done on Porter Stemmer which made it more precise over large data-sets
- It is an improvement over Porter Stemmer.
- Slightly faster computation time than porter, with a fairly large community around it.

Preprocessing Data 2: Stemming

Word	PorterStemmer	LancasterStemmer	SnowballStemmer
text	text	text	text
mining	mine	min	mine
analytics	analyt	analys	analyt
transforms	transform	transform	transform
unstructured	unstructur	unstruct	unstructur
normalized	normal	norm	normal
suitable	suitabl	suit	suitabl
analysis	analysi	analys	analysi

Preprocessing Data 2: Lemmatization

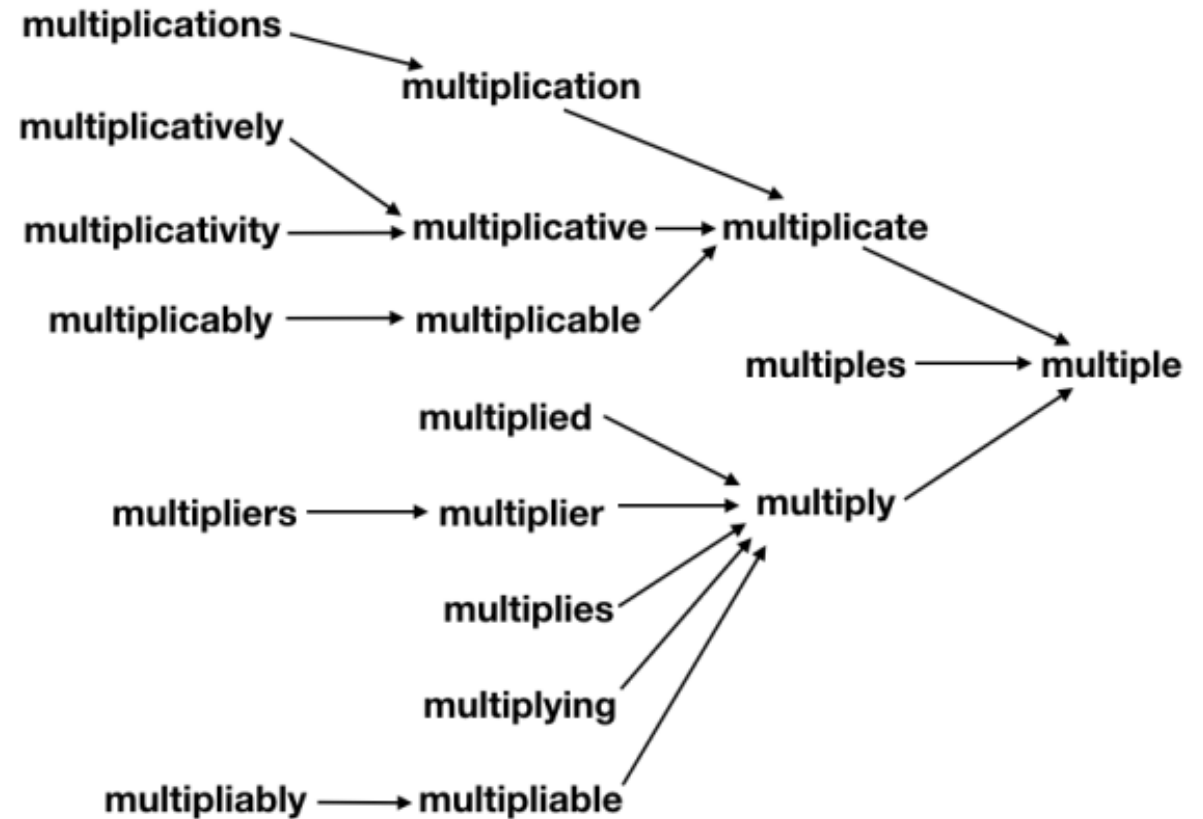
- Lemmatizing derives the canonical form ('lemma') of a word. i.e the root form.
- It is better than stemming as it uses a dictionary-based approach i.e a morphological analysis to the root word

Lemmatization vs Stemming

- Stemmers (usually) simply chop off endings in an attempt to eliminate relatively meaningless part
- While useful for their speed and simplicity, they suffer from an overly simplistic approach to manipulating words that results in many disconnected similar words

Preprocessing Data 2: Lemmatization

- Seek to capture the “lemma” or core/root of a word by looking through relationships in dictionaries.
- With lemmatizers, relationships like that between better and good are accurately captured, so they’re very useful when accuracy is needed.
- link together related words of different parts of speech, first detecting the part of speech (with error) and then traversing the web of derivationally related forms (and synsets and pertainyms, etc.) until one decides on a “root” form



Preprocessing Data 2: Lemmatization

- In Short, Stemming is typically faster as it simply chops off the end of the word, without understanding the context of the word.
- Lemmatizing is slower and more accurate as it takes an informed analysis with the context of the word in mind.

Stemming vs Lemmatization

change
changing
changes
changed
changer

chang

change
changing
changes
changed
changer

change

Feature Engineering: Vectorizing Data

- words of the text represent discrete, categorical features.
- Algorithms are usually not good for those features.
- Vectorizing is the mapping from textual data to real valued vectors
- Encode text as integers i.e. numeric form to create feature vectors so that machine learning algorithms can understand our data.
- Several options
 - Bag-Of-Words (BOW)
 - N-Grams
 - TF-IDF

Feature Engineering: Vectorizing Data

Why A Vector Space Model:

- Each document ➡ vector
- Each query ➡ vector
- Search for documents ➡ find 'similar' vectors
- Cluster documents ➡ find 'similar' vectors

Feature Engineering: Simple Bag of Words (BOW):

- We make the list of unique words in the text corpus called vocabulary.
- Then we can represent each sentence or document as a vector with each word represented as 1 for present and 0 for absent from the vocabulary.
- Another representation can be a count of the number of times each word appears in a document.

Bag of words model:

Represent each **document** as a **bag of words**, ignoring words' ordering. Why? For **simplicity**.

Unstructured text becomes **a vector of numbers**

e.g., docs: "I like visualization", "I like data".

1 : "I"

2 : "like"

3 : "data"

4 : "visualization"

"I like visualization" \Rightarrow [1, 1, 0, 1]

"I like data" \Rightarrow [1, 1, 1, 0]

Bag of Words:

- Hey mate, have you read about Hinton's capsule networks? → No
- Hi Kirill, are you coming to dinner tonight? → Yes
- Did you like that recipe I sent you last week? → Yes
- Dear Kirill, would you like to service your car with us again? → No
- Are you coming to Australia in December? → Yes

....

.....

↓ **Corresponding Bag of Words (vectors):**

[1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, ..., 2]	→ No
[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 1, 0, 0, ..., 0]	→ Yes
[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, ..., 1]	→ Yes
[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, ..., 1]	→ No
[1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, ..., 1]	→ Yes

...

Bag of Words:

- Hey mate, have you read about Hinton's capsule networks? → No
- Hi Kirill, are you coming to dinner tonight? → Yes
- Did you like that recipe I sent you last week? → Yes
- Dear Kirill, would you like to service your car with us again? → No
- Are you coming to Australia in December? → Yes

....

.....

↓ Corresponding Bag of Words (vectors):

```
[1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, ..., 2]
[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 1, 0, 0, ..., 0]
[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, ..., 1]
[1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, ..., 1]
[1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, ..., 1]
```

...

→ No
→ Yes
→ Yes
→ No
→ Yes

→ Yes/No?

Train the data, such as using logistic regression for Classification, or Neural Network (Deep Learning), etc:

Feature Engineering: N-Grams

- N-grams are simply all combinations of adjacent words or letters of length n that we can find in our source text.
- Ngrams with $n=1$ are called unigrams.
- Similarly, bigrams ($n=2$), trigrams ($n=3$) and so on can also be used.
- The basic principle behind n-grams is that they capture the letter or word is likely to follow the given word.
- The longer the n-gram (higher n), the more **context** you have to work with.

Feature Engineering: TF-IDF (term frequency-inverse document frequency)

- Relative frequency that a word appears in a document compared to its frequency across all documents.
- More useful than term frequency for identifying important words in each document
 - search engine scoring
 - text summarization
 - document clustering.

Feature Engineering: TF-IDF (term frequency-inverse document frequency)

- **Term Frequency (TF)** = (Number of times term t appears in a document) / (Number of terms in the document)
- **Inverse Document Frequency (IDF)** = $\log(N/n)$, where, N is the number of documents and n is the number of documents a term t has appeared in. The IDF of a rare word is high, whereas the IDF of a frequent word is likely to be low. Thus having the effect of highlighting words that are distinct.
- Then we calculate TF-IDF value of a term as = **TF * IDF**

Feature Engineering: TF-IDF (term frequency-inverse document frequency)

Document 1		Document 2	
Term	Count	Term	Count
This	1	This	1
is	1	is	1
a	1	a	1
beautiful	2	beautiful	1
day	5	night	2

$TF('beautiful', Document1) = 2/10$, $IDF('beautiful') = \log(2/2) = 0$

$TF('day', Document1) = 5/10$, $IDF('day') = \log(2/1) = 0.30$

$TF-IDF('beautiful', Document1) = (2/10) * 0 = 0$

$TF-IDF('day', Document1) = (5/10) * 0.30 = 0.15$

Question: Why does 'day' have a higher score than 'beautiful'?

Example1: TF-IDF

- Term Frequency (TF) matrix given in the table shows the frequency of terms per document.
- Calculate the Term Frequency – Inverse Document Frequency (TF-IDF) value.

Document/Term	T1	T2	T3	T4	T5	T6
D1	5	9	4	0	5	6
D2	0	8	5	3	10	8
D3	3	5	6	6	5	0
D4	4	6	7	8	4	4

Example1: TF-IDF

Document/Term	T1	T2	T3	T4	T5	T6
D1	5	9	4	0	5	6
D2	0	8	5	3	10	8
D3	3	5	6	6	5	0
D4	4	6	7	8	4	4

Formula to use:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j

df_i = number of documents containing i

N = total number of documents

Solution:

$$TF - IDF(T1 \text{ in } D1) = \frac{5}{29} * \log\left(\frac{4}{3}\right) =$$

$$TF - IDF(T2 \text{ in } D1) = \frac{9}{29} * \log\left(\frac{4}{4}\right) =$$

$$TF - IDF(T3 \text{ in } D1) = \frac{4}{29} * \log\left(\frac{4}{4}\right) =$$

$$TF - IDF(T4 \text{ in } D1) = 0 * \log\left(\frac{4}{3}\right) =$$

$$TF - IDF(T5 \text{ in } D1) = \frac{5}{29} * \log\left(\frac{4}{4}\right) =$$

$$TF - IDF(T6 \text{ in } D1) = \frac{6}{29} * \log\left(\frac{4}{3}\right) =$$

...And the same calculation will be applied for
The remaining documents(D2, D3, and D4)

Classwork: Apply TF-IDF

Ex: sentence 1: He is a good boy
sentence 2: she is a good girl
sentence 3: Boy and girl are good

Removing stopwords(collection of words such as "is, of, the, etc")
sentence 1: good boy
sentence 2: good girl
sentence 3: boy girl good

	Good	Boy	Girl
sentence1			
sentence2			
sentence3			