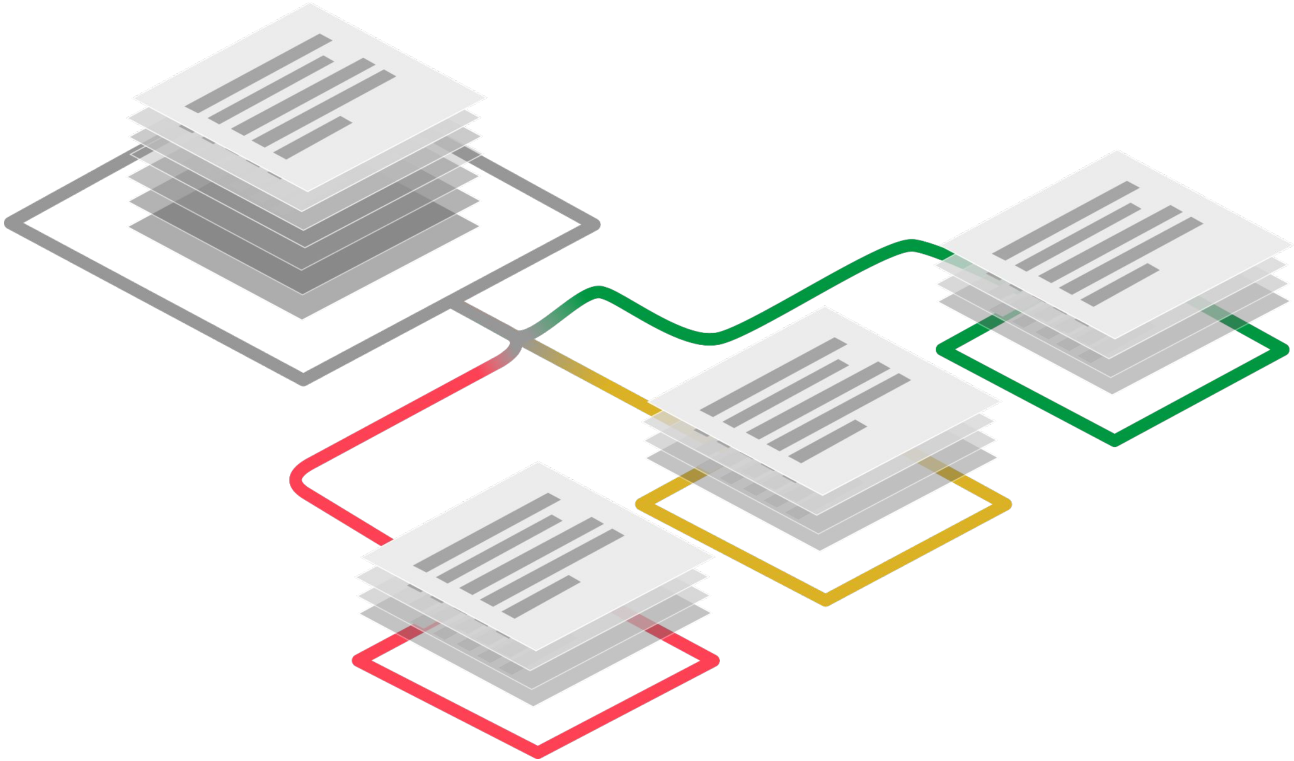
Text Classification with Bag-of-Words Models

MMAI 5400 – lecture 4
Fall 2024

Contents

What is text classification?



Text classification
- examples

Spam* detection

- Classify an email as spam or ham (non-spam).

Sentiment analysis

- Classify the sentiment, i.e. the positive or negative orientation expressed towards some object.
- E.g. reviews, political texts about candidates, market sentiment, …

Language detection

- Classifying the language of a text.
- Often an important first step in an NLP pipeline.

Topic classification

- Classifying texts based on topic, e.g., journal articles by research topic.
- **Automated routing of incident tickets to support teams.**

* See [Monty Python - Spam]

Text classification

- Sentiment analysis from simple to advanced

Count key-words

- Hard-coded key-words.
- Examples:
    - Positive: ***great***, ***fantastic***, ***amazing***, …
    - Negative: ***horrible***, ***boring***, ***lackluster***, …
- No training data not required.
- Sentiment words are listed in sentiment lexicons, e.g. the MPQA subjectivity lexicon.
- But, inflexible, requires much manual labour & not sensitive to syntax.

Bag-of-Words text classification

- Use ML to learn what is positive or negative.
- Quick & flexible.
- But, not sensitive to word order (syntax), & requires labeled data.

Text embeddings

- State-of-the-art results.
- Deep learning-based.
- Requires big datasets → not so quick.
- Sensitive to word order.
- But, requires labeled data.

BoW text representation

Two ways of representing text for ML

Bag-of-Words

- Based on word (token) counts.
- Documents are represented by their word frequencies (often transformed).
- Word order is ignored.
- Good for long documents.

Embeddings (word/sentence)

- Often for neural network models.
- Documents are represented as a sequence of tokens (word/sub-word/character).
- Tokens are represented as vectors.
- Word order is important.
- Works well with short documents (e.g. single sentences).

Bag-of-Words (BoW)

- A way of representing text

Binary count

- `True` indicate the presence of a word in a text & `False` its absence

Term frequency (TF)

- The frequency by which a word occurs in a text.

Term Frequency Inverse Document Frequency (TF-IDF)

- TF weighted by how unique a word is to a particular text.

Binary count

Document

“The strange but compelling Bacurau, plunging us into the hinterlands of Brazil, is the kind of modern-day Western that Sam Peckinpah might have made if he were a) Brazilian & b) alive.” *

Term	Bin-count
bacurau	`True`
brazil	`True`
compelling	`True`
flowers	`False`
hinterlands	`True`
love	`False`
of	`True`
plunging	`True`
strange	`True`
the	`True`
western	`True`

* Parts of a review of Bacurau, written by Brian Viner & published in the Daily Mail (UK)

Term Frequency (TF)

Document

*“The strange but compelling Bacurau, plunging us into the hinterlands of Brazil, is the kind of modern-day Western that Sam Peckinpah might have made if he were a) Brazilian & b) alive.” **

Term	count	TF
bacurau	1	1/30
brazil	1	1/30
compelling	1	1/30
flowers	0	0
hinterlands	1	1/30
love	0	0
of	2	1/15
plunging	1	1/30
strange	1	1/30
the	3	1/10
western	1	1/30

* Parts of a review of Bacurau, written by Brian Viner & published in the Daily Mail (UK)

Term Frequency (TF)

Question:
Why + 1?

Measure how frequent a word is in a document

- Raw count of term t in document d: $TF(t, d) = C(t, d)/(N_d + 1)$.
- Squash frequencies with log: $TF(t, d) = \log_{10}(C(t, d)/(N_d + 1))$.
- $C(t, d)$ is the count of t in d, & N_d is the total number of words in d.

Represent the relevance of a term to a document

- A word appears frequently in a document $\rightarrow$ important.
- Often it is good to remove stop words (common words such as `a`, `the`, `is`).

Raw term frequency

Issue

- Skewed & not very discriminative.
- Common words dominate the counts but are not very informative about document content.
  - E.g. the, it, or they
- High frequency words are important, but **too high** frequency words are unimportant

We want to measure both how ***frequent*** words are in a document, & how ***unique*** they are to a document

The **TF-IDF** is the product of two terms, each term capturing one of these two.

TF (count) example

documents

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

terms

* example from figure 6.2 in section 6.3.1 of *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019)

Detour

- Information retrieval & document comparison

Simple Information Retrieval (IR)

Document vectors
I.e. points in a $|V|$ dimensional space.

Information retrieval (IR)
- Finding the document d from the D documents in the corpus that best matches a query q.
- The query is also a vector of length $|V|$.
- I.e. we need to compare pairs of vectors to find the most similar.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

* example from figure 6.3 in section 6.3.1 of *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019)

Question: What does $|V|$ mean?

Blue rectangles: next slide

Documents as vectors

Document vectors
- points in a $|V|$ dimensional space.

Each document is represented by a vector of length $|V|$.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

* example from figure 6.3 in section 6.3.1 of *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019)

Documents as vectors

Here each document is represented by a vector of length 2.

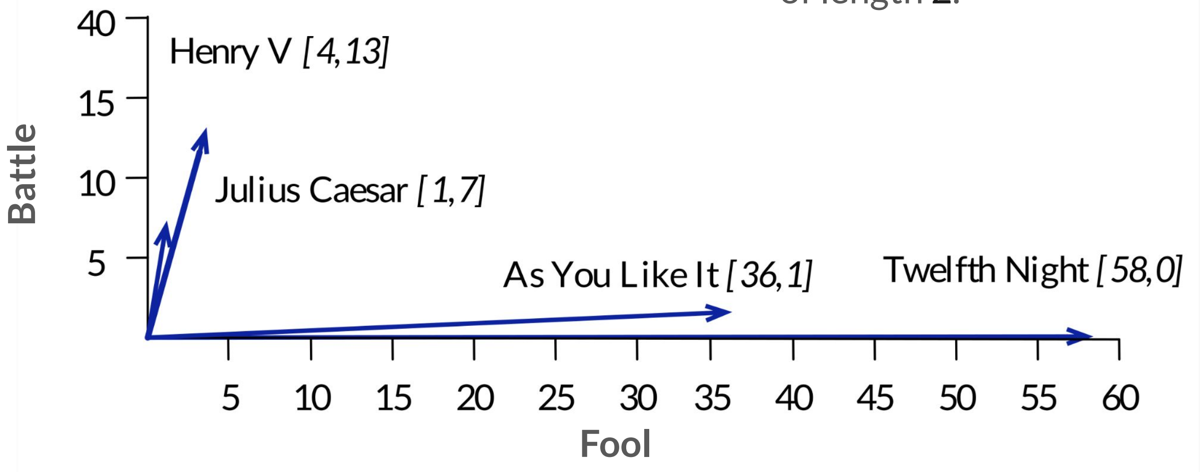


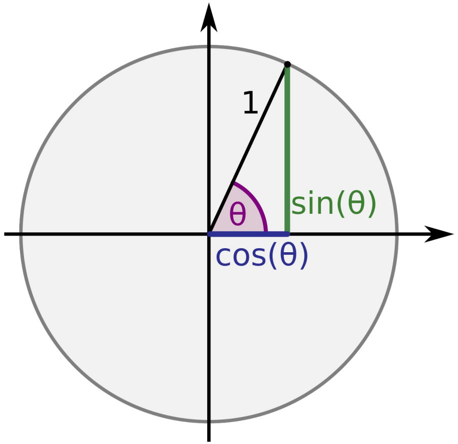
Image credit: Figure 6.4 in section 6.3.1 of *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019)

Question: What does $|\mathbf{v}|$ mean?

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} v_i w_i \qquad |\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

Comparing documents

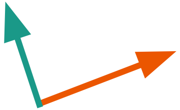
- Cosine similarity



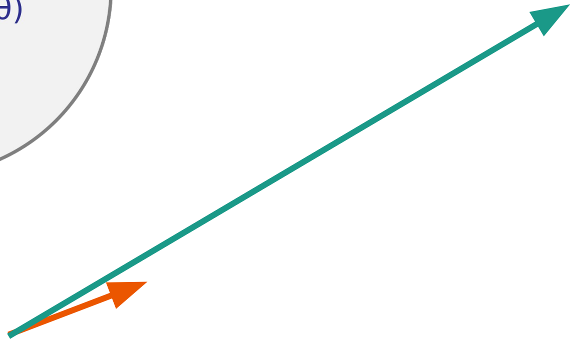
How do we measure the similarity between two document vectors?

Cosine similarity

$$\frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}|\,|\mathbf{w}|} = cos\ \theta$$



$$\theta \approx 90^\circ \rightarrow cosine(\mathbf{v}, \mathbf{w}) \approx 0$$



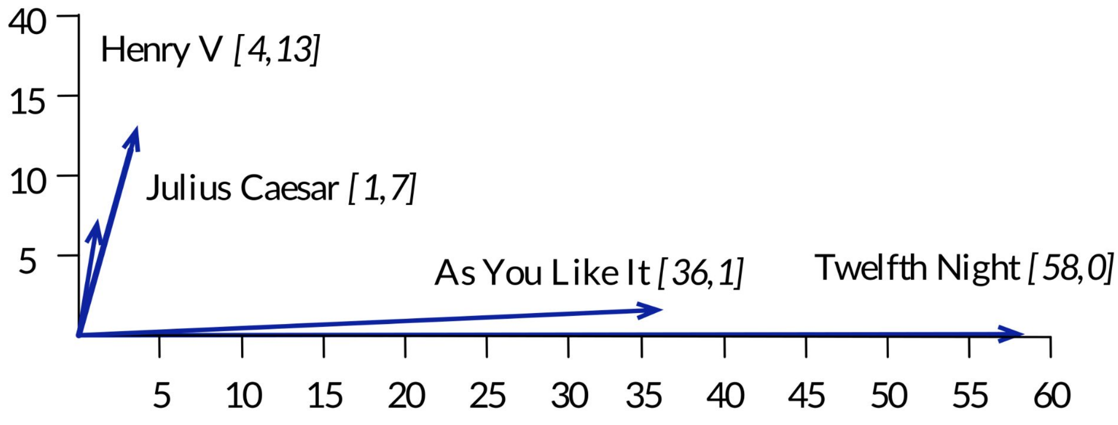
$$\theta \approx 0^\circ \rightarrow cosine(\mathbf{v}, \mathbf{w}) \approx 1$$

$$cosine(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}|\,|\mathbf{w}|}$$

Image credit: Stephan Kulla - CC0, https://commons.wikimedia.org/w/index.php?curid=57551646

Question: What is the literary reason for these similarities?

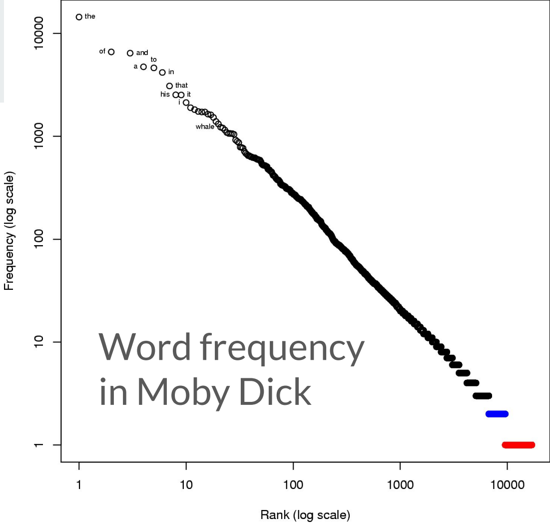
Simple Information Retrieval (IR)



Pair	Cosine similarity
Henry V - Julius Caesar	≈ 1
Henry V - As You Like It	≈ 0
Henry V - Twelfth Night	≈ 0
Julius Caesar - As You Like It	≈ 0
Julius Caesar - Twelfth Night	≈ 0
As You Like It - Twelfth Night	≈ 1

Image credit: Figure 6.4 in section 6.3.1 of *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019)

Sparse vectors & the curse of dimensionality



In the toy example above we used a vocabulary of size 4. However, for a real corpus can have millions of words & many of those words only occur only once in the corpus (aka *hapax legomenon* words). For example the Brown corpus has around a million unique words & 50 000 *hapax legomena*.

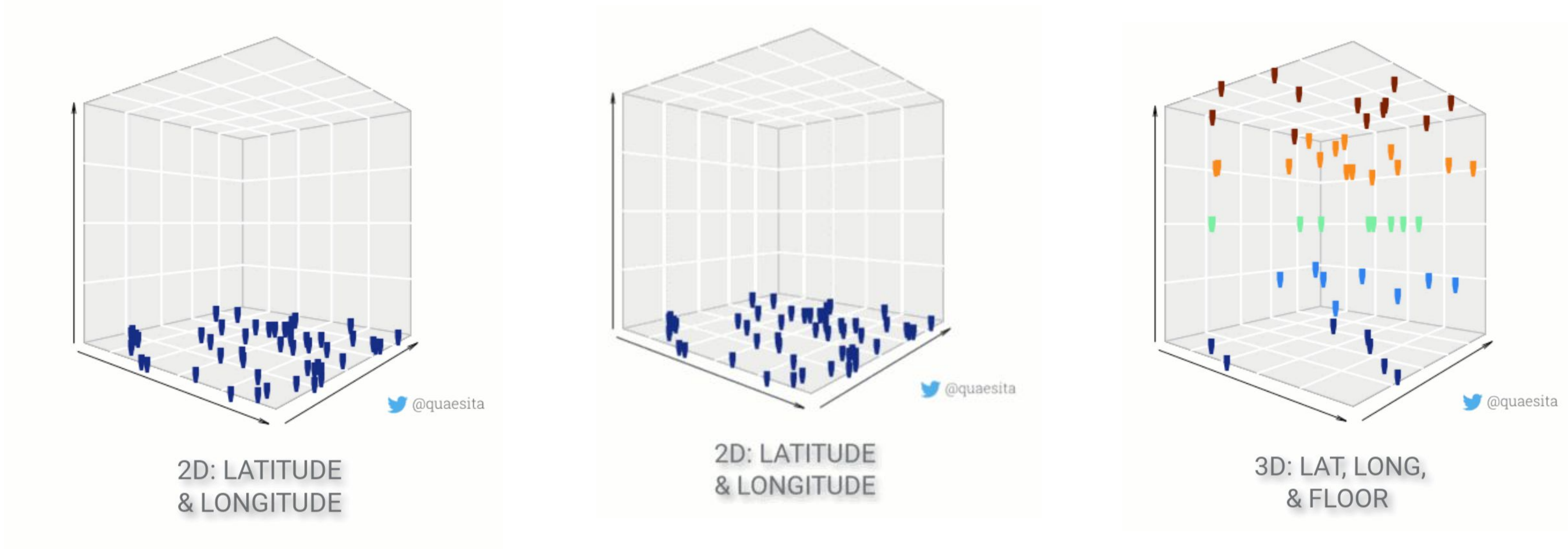
Thus, the dimensionality of the representational space will be huge & individual document vectors will be sparse, i.e. mainly populated by zeros.

- More CPU & memory to process the large vectors.
- ML becomes harder due to the *curse of dimensionality*.

Image source: Radagast3, Public Domain, https://commons.wikimedia.org/w/index.php?curid=10306247

The curse of dimensionality

Most ML algorithms need the space to be (relatively) densely populated to fit the parameters.
But, with increasing number of dimensions the amount of data required for the same density increases ***exponentially***.



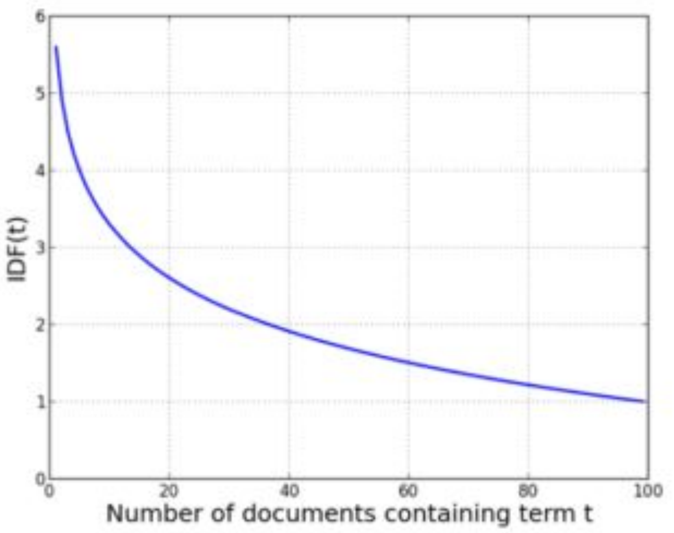
End of detour

Inverse Document Frequency (IDF)

Measures how specific a word is to a document

A word appearing in many document implies less importance since it is not a unique identifier.

$$IDF(t) = log_{10}\frac{N_{doc}}{C(t \in docs)}$$



IDF example

Document frequency among 37 Shakespeare plays.

Term	DF	IDF
battle	21	$\log_{10}(37/21) = 0.246$
good	37	$\log_{10}(37/37) = 0$
fool	36	$\log_{10}(37/36) = 0.012$
wit	34	$\log_{10}(37/34) = 0.037$

* example from section 6.5 in *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019)

Combined – TF-IDF

Measures the importance of words based on frequency & specificity

Frequent words that are specific to small number of documents → high score

$$TF\text{-}IDF(t, d) = TF(t, d) \times IDF(t)$$

$TF(t,d)$ – each word t in each document d

$IDF(t)$ – for word t in all documents

TF-IDF example

terms

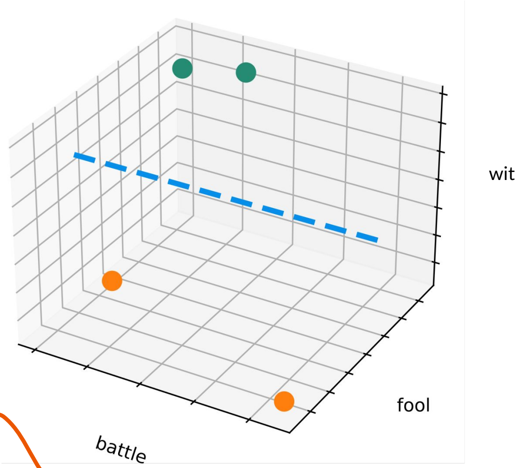
documents

	battle	good	fool	wit
As You Like It	0.074	0	0.019	0.049
Twelfth Night	0	0	0.021	0.044
Julius Caesar	0.22	0	0.0036	0.018
Henry V	0.28	0	0.0083	0.022

* example from figure 6.8 in section 6.5 of *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019)

Text classification continued

TF-IDF for text classification



Features

	battle	good	fool	wit	target
As You Like It	0.074	0	0.019	0.049	1
Twelfth Night	0	0	0.021	0.044	1
Julius Caesar	0.22	0	0.0036	0.018	0
Henry V	0.028	0	0.0083	0.022	0

Examples

The target can represent e.g. *comedy*.

* example from figure 6.8 in section 6.5 of *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019)

Question: It’s good to know of popular learners for BoW text classification, but how to you select a learner in practice?

Popular learners for BoW text classification

Traditional ML algorithms that do well with high-dimensional data & few examples.

- Naïve Bayes Classifier
- Logistic Regression
- Support Vector Machines (SVM)
- Tree-based classifiers

Limitations of Bag-of-Words

BoW is simple to understand, implement & can easily be customized to specific tasks.

However, it suffers from shortcomings:

- **Vocabulary**: The vocabulary has to be kept small for document vectors not to be too sparse.
- **Sparsity**: Sparse representations are harder to model due to computational complexity & because it's hard to model little information in a huge representational space.
- **Word meaning**: Synonyms like "*exhausted*" & "*fatigued*" should be closer in the representational space than antonyms like "*exhausted*" & "*replenished*". But, with BoW they are not.
- **Sentence meaning**: Discarding word order result in critical loss of information (e.g. "**this is interesting**" vs "**is this interesting**").

A word-order fix

BoW discards word order which means that important information is often lost.

For example, imagine sentiment classification on a sentence like "*BoW is not good.*" This sentence is quite likely to be classified as **positive** since the **unigrams** "*BoW*", "*is*" & "*not*" are **neutral** & "*good*" is **positive**. However, the **bigram** "*not good*" is **negative**.

Thus, one improvement to text classification where short sequences of word carry critical information is to add bigrams (& possibly trigrams).

Data storage for NLP

Data storage for NLP

Consider the following

- Convenience , i.e. easy inspection, read/write & programming language support.
- Meta/structured data, e.g. class labels, dates & other info associated with a document.
- The file formats handling of special characters, e.g. commas in CSV files.
- IO speed; for big datasets.
- Ability to read in batches during training; for data that doesn't fit in RAM or GPU memory.

Data storage

Normal size datasets

- plain text, i.e. `TXT` files
    - Convenient, no metadata, no issue with special characters
- `CSV/TSV`
    - Convenient, meta/structured data, issue with special characters (e.g. “,”, “\t”)
- `JSON/JSONL` (newline-delimited JSON)
    - Used for sending structured data over the web
    - Quite convenient, meta/structured data, no issue with special characters
    - Python: `df = pd.read_json(“fancy_nlp_data.json”)`

Big datasets

- Gzip of text files (can be read by Python)
    - Reduces storage
    - Doesn’t solve RAM issue

Huge dataset (too big for RAM/GPU-memory)

- Need to read batches during training
    - fast.ai: `TextDataBunch`
    - TensorFlow: `tf.data.TextLineDataset`

Terminology

Term frequency

Bag-of-Words

Inverse document frequency

TF-IDF

Topic classification

The curse of dimensionality

Information retrieval

Cosine similarity

Tutorial

Text classification

`MMAI5400_class04_textclass.ipynb`

or

`MMAI5400_class04_TicketClass.ipynb`