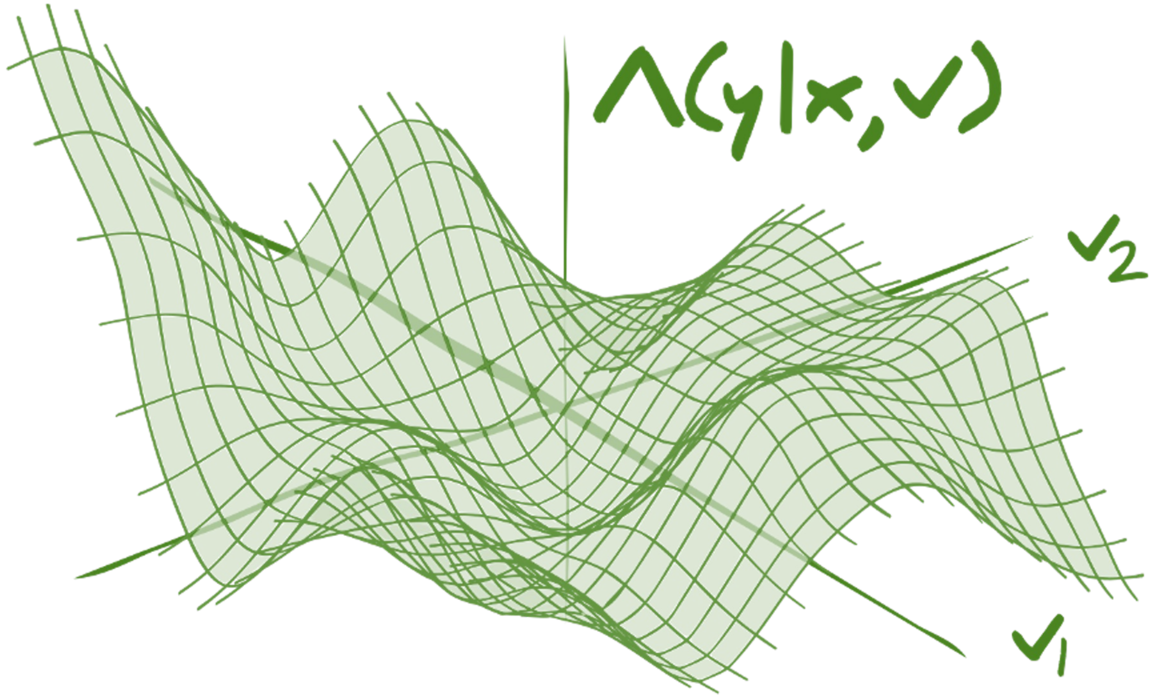
Meanings in Statistical and ML Models

Goals

1. Appreciating how traditional NLP models have used statistics to represent word meanings
2. Appreciating how modern NLP models have used statistics and ML to represent word meanings

Is there a statistical or Machine Leaning approach that might work in place of an annotated ontology or a pattern-matching method?



Two routes

- Traditional NLP
- Modern NLP

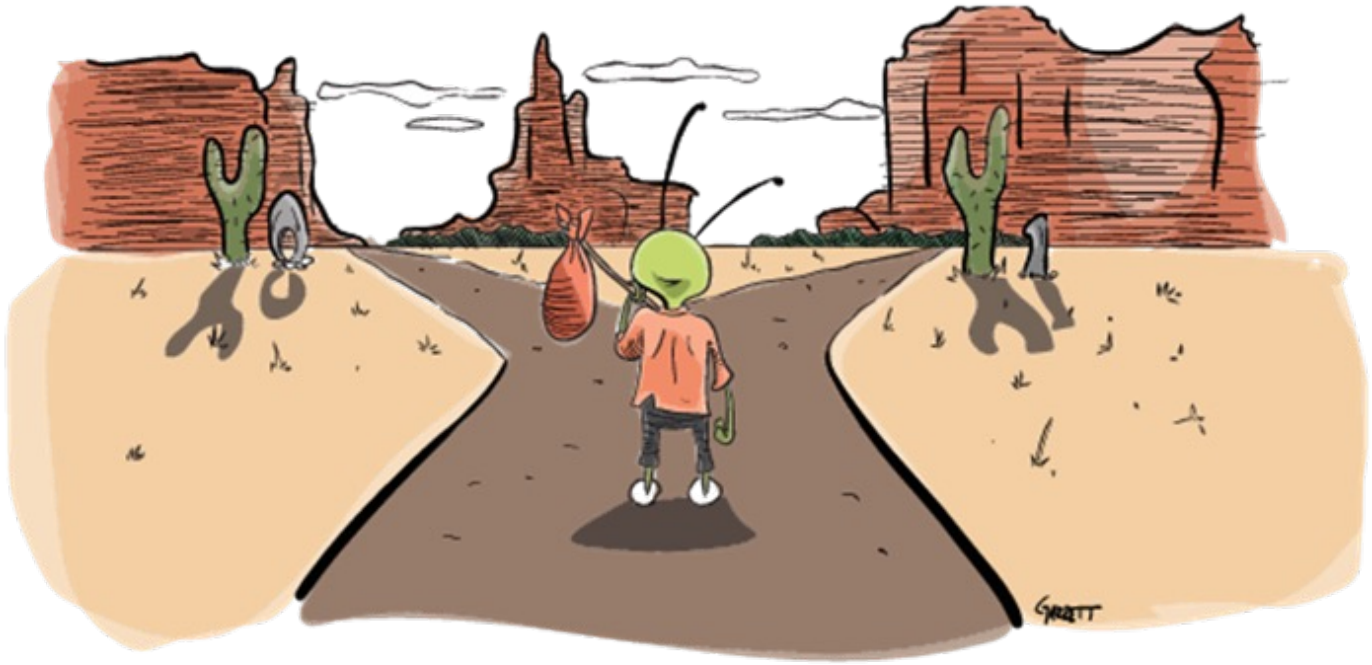


Image source: https://www.masteringmetrics.com

Traditional NLP

Bag-of-Words (BoW) approach

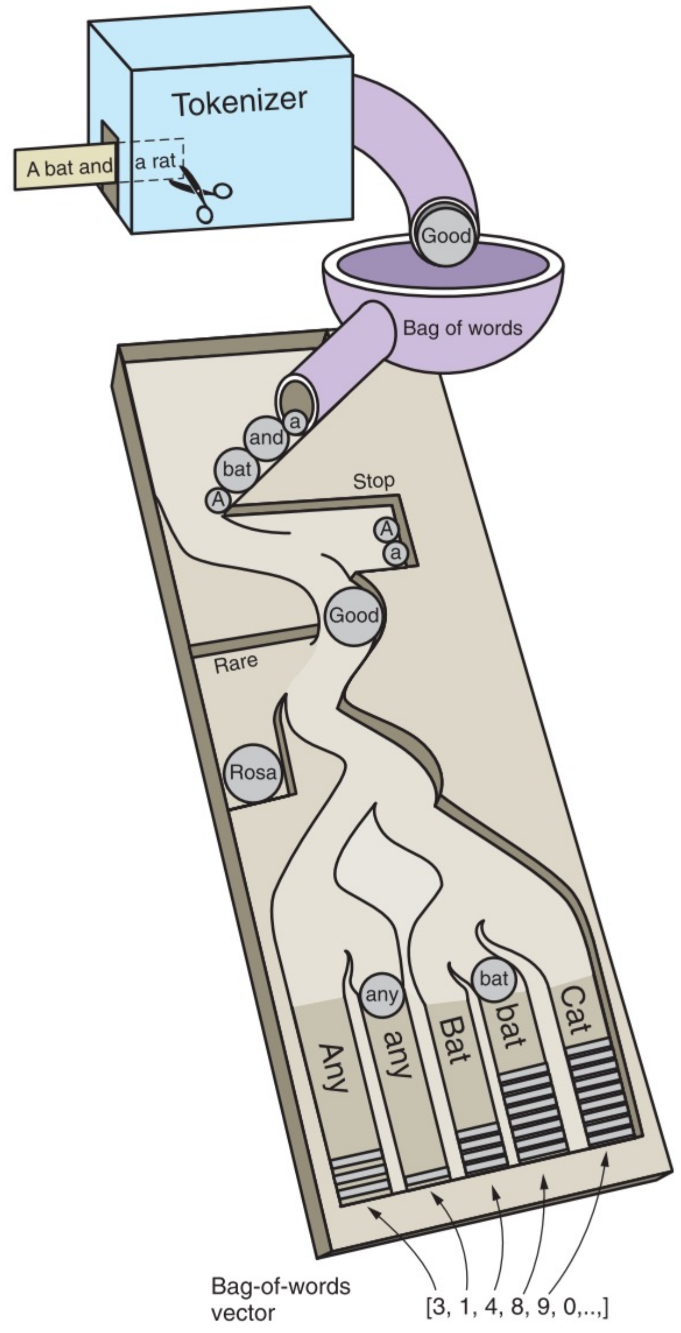
Given some text corpora **D**, a BoW work-flow implies the following steps:

1. ∀ **d** ∈ **D** (e.g., documents, statements, sentences, and even single words), get the token **t(d)**

2. ∀ **s** ∈ **S** (i.e., unique lexical items in **D**) and **d** ∈ **D**, get the cardinality **|s|**.

We call the possible vectors a machine might create this way a vector space.

Such a vector space allows us to use linear algebra (and libraries such as NumPy, Scipy, or Numba) to manipulate lexical items and compute things like distances and statistics involving natural language data.



Visual illustration of BoW

Source: Lane, Howard & Hapke, 2019

What can we do with Bow?

For example, we can address search queries such as:

- ‘What is the combination of words most likely to follow a particular bag of words?’
    - E.g., what is the expected content of the following review regarding our product?
- ‘What is the closest bag of words in our database to a bag-of-words vector provided by the user?’
    - E.g., which is the most comparable product review to the one customer ABC recently posted on platform XYZ?

BoW limitations (1/2)

One of the main limitations of the BoW approach is the proliferation of unique vectors to analyze. Likely as not, the documents in our dataset will have unique mixtures of lexical items. Consider for example the BoW $\mathbf{S_a}$ and $\mathbf{S_b}$ associated with documents **a** and **b:**

$\mathbf{S_a} = \{2, 4, 5, 0, 0\} \neq \mathbf{S_b} = \{3, 7, 1, 0, 0\}$

One-hot vectors mitigate the curse of dimensionality by considering whether a word is or is not present in a piece of text. The one-hot encoding of $\mathbf{S_a}$ and $\mathbf{S_a}$ would then be:

$\mathbf{S_a} = \mathbf{S_b} = \{1, 1, 1, 0, 0\}$

In other words, we can use the same vector to express the semantic properties of multiple documents.

BoW limitations (2/2)

Unfortunately, one-hot vectors have limitations.

Example 1: the vectors associated with the words `good’ and `fine’ are orthogonal:

good = [0, 0, 1, 0, 0, 0]

fine = [0, 0, 0, 0, 1, 0]

Example 2: 'greasy spoon' and 'British cafe' express the same category of eatery but the intersection of their one-hot vectors is empty:

greasy spoon = [[0, 1, 0, 0], [0, 0, 1, 0]]

British cafe = [[1, 0, 0, 0], [0, 0, 0, 1]]

Modern NLP

From DH to embeddings

Intuition

According to the Distributional Hypothesis, a focal word’s **ω** meaning is a function of the linguistic context — i.e., the lexical items in the neighborhood of the focal word

Then, considering (all) the many contexts of **ω** (e.g., regulation) helps to create an accurate vector representation of **ω**

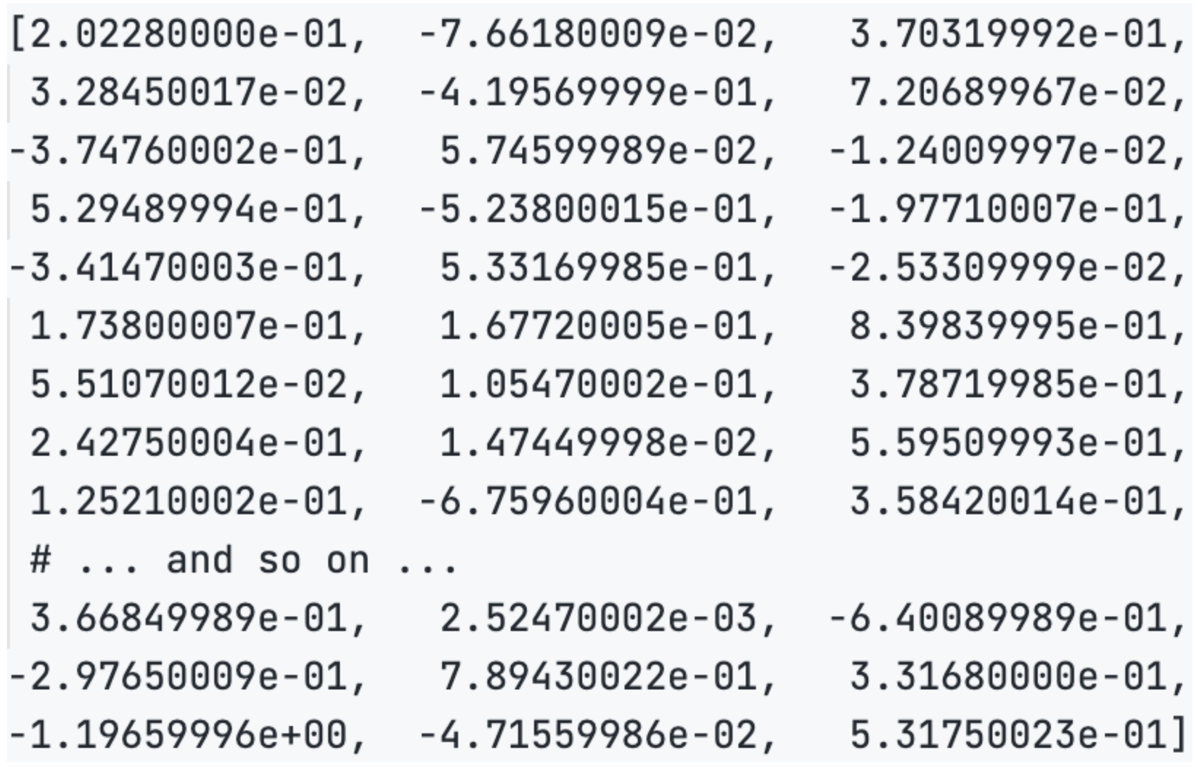
Example

- “... to encourage and implement the adoption of common **REGULATION**s for all forms of motor sports and series across the …”
- “countries should adhere to the cost-benefit paradigm of **REGULATION**, forcing bureaucrats to outline all the benefits of ...”
- “Agencies create **REGULATION**s (also known as "rules") under the authority of Congress to help ...”

Word vectors as dense, real valued vectors

Ultimately, by observing and analyzing a same word in multiple context, we aim at building a dense vector for each word, chosen so that it is like vectors of words that appear in similar contexts.

On the right is a portion of the vector associated with the word 'banana'.



```
[2.02280000e-01,  -7.66180009e-02,   3.70319992e-01,
 3.28450017e-02,  -4.19569999e-01,   7.20689967e-02,
-3.74760002e-01,   5.74599989e-02,  -1.24009997e-02,
 5.29489994e-01,  -5.23800015e-01,  -1.97710007e-01,
-3.41470003e-01,   5.33169985e-01,  -2.53309999e-02,
 1.73800007e-01,   1.67720005e-01,   8.39839995e-01,
 5.51070012e-02,   1.05470002e-01,   3.78719985e-01,
 2.42750004e-01,   1.47449998e-02,   5.59509993e-01,
 1.25210002e-01,  -6.75960004e-01,   3.58420014e-01,
 # ... and so on ...
 3.66849989e-01,   2.52470002e-03,  -6.40089989e-01,
-2.97650009e-01,   7.89430022e-01,   3.31680000e-01,
-1.19659996e+00,  -4.71559986e-02,   5.31750023e-01]
```

Overview of the word2vec

word2vec (Mikolov et al. 2013) is a framework for learning word vectors similar to the one presented in the previous slide.

Here is the concise description of the algorithm — given a corpus of text **D**:

- Associate a random vector with size **n** Each word **d** ∈ **D**
- Consider each position **k** in the text, which has a centre word **ω** and context words **η**
- Use the similarity of the word vectors for **ω** and **η** to calculate the probability **η** appears in a text span conditional on the presence of **ω** (or vice versa)
- Keep adjusting the word vectors to maximise this probability

Wrap-up

Main points

- BoW is the quintessential example of a traditional NLP model that represents meanings
- BoW is intuitive and easy to implement — however, it is computationally demanding and does not consider that different words may convey similar meanings (e.g., café and bar)
- Word2vec is the quintessential example of modern NLP model that relies on DH and ML to map the words’ meanings over dense, real valued vectors