UNIVERSITY OF GOTHENBURG

SPRÅKBANKENTEXT

Distributional Semantics

Ricardo Muñoz Sánchez

Based on Slides by Nikolai Ilinykh
and many others…

Where We're At

- We have seen how to represent the meaning of words and sentences using logic
- We now turn to vector spaces to represent meaning
- Today's topics:
  - Distributional semantics
  - Measuring semantic similarity based on the similarity of contexts
  - Using vector representations for downstream tasks

The main idea:

"A word is characterized by the company it keeps"

John Rupert Firth

The Distributional Hypothesis

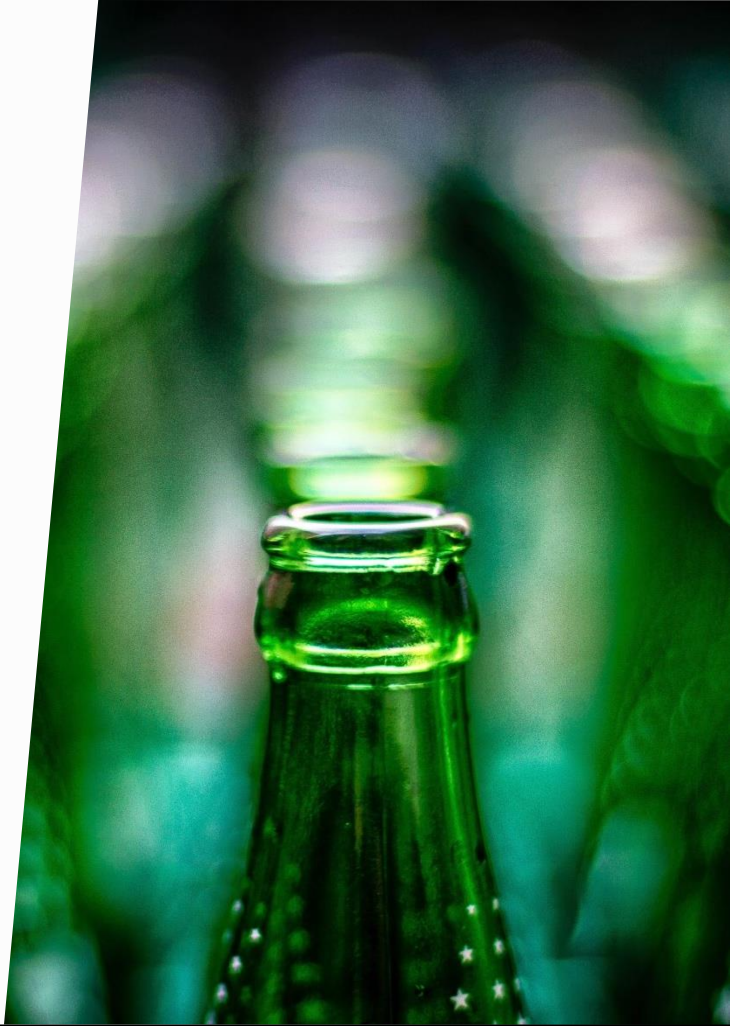
- We can infer the meaning of a word by its context

- Take an example: **tesgüino**
  - A bottle of **tesgüino** is on the table
  - Everybody likes **tesgüino**
  - **Tesgüino** makes you drunk
  - We make **tesgüino** out of corn

Example from Joos (1950), Harris (1954), and Firth (1957)

The Distributional Hypothesis

- We can infer the meaning of a word by its context
- Take an example: **tesgüino**
- It’s a corn beer made by the Rarámuri people!



Example from Joos (1950), Harris (1954), and Firth (1957)

The Main Idea

- We talk about similar things in similar contexts
- I will talk about pears and apples using similar words and expressions
- The words I use when talking about cars will (hopefully) be different ones

Learning Words from Context

- This is an example of *representation learning*

- Compare it with feature engineering
  - The computer learns in a *self-supervised* fashion
  - Less work from our side (yay!)
  - Might pick up things we do not want (yay?)

- More on the last point next time…

Some Things to Bear in Mind

LET’S TAKE A WORD **MOUSE**

WHAT DO WE MEAN BY IT?

Some Things to Bear in Mind

- Do we always refer to the same thing when we use the word “mouse”?
- What about similar words?
  - Say, rat and mouse
  - Both can refer to rodents
  - They also have other meanings
- And different but similar words
  - Cats and dogs are closer to each other than to chairs

Some Things to Bear in Mind

- Do we always refer to the same thing when we use the word “mouse”?

- What about similar words?
  – Say, rat and mouse
  – Both can refer to rodents
  – They also have other meanings

- And different but similar words
  – Cats and dogs are closer to each other than to chairs

Some Things to Bear in Mind

- What about synonyms?
  - Travel, trek, and journey are synonyms
  - However, they have different connotations

- Words can have other kinds of relations!
  - Superclass: fruit vs. apple
  - Eventiveness: food vs. chef
  - Semantic frames and roles: X bought Z from Y vs. Y sold Z to X

How Does the Computer Do This?

- Sparse vector representations
  - Check for word co-occurrence and make a matrix
  - Other ways to represent mutual information
  - We will focus on this for now
- Dense vector representations
  - Learnt through singular value decomposition
  - Can also be learnt through neural networks
  - We will work on this next time!

The Idea

- Get our corpus (websites, articles, books, etc.)

- Determine a context window
  - It can be the n words to the left and to the right
  - If our corpus is big enough, the context can be the whole document!

- We can also characterize documents by the words that appear within them

Occurrence Vector

The simplest possible solution

For each word, count the number of times it appears

Issues:

- No context for the words
- Stopwords are overrepresented

Occurrence Vector

```
def do_word_count(text):
    word_count = { }

    for word in preprocess(text):

        if word not in word_count:
            word_count[ word ] = 0

        word_count[ word ] += 1

    return word_count
```

Using Numerical Indices

Using the words as indices makes it easier for us humans to understand these concepts

However, it makes math harder!

The idea: create an index / dictionary based on the n most common words!

Creating an Index / Dictionary

```
def make_index( text , size ):

    word_count = do_word_count(text)

    def map_word_to_count(word): return word_count[ word ]

    keep_these_words = sorted(word_count.keys(), key = map_word_to_count)[:size]

    return keep_these_words
```

Co-Occurrence Matrix

We now take the context into account

Given a word and a context window, count all words within that context window

Issues

- Too sparse (i.e. lots of zeros in the matrix)
- Still overrepresents stopwords

Co-Occurrence Matrix

```
def do_co_ocurrence(text, context_window):

    co_ocurrence_matrix = { }

    preprocessed_text = preprocess(text)

    for i, current_word in enumerate(preprocessed_text):

        if current_word not in co_ocurrence_matrix:
            co_ocurrence_matrix[ current_word ] = {}

        min_context = max( 0 , i-context_window )
        max_context = min( len(preprocessed_text) , i+context_window)

        context = preprocess( min_context , max_context)

        for context_word in context:

            if context_word != current_word:

                if context_word not in co_ocurrence_matrix[ current_word ]:
                    co_ocurrence_matrix[ current_word ][ context_word ] = {}

                co_ocurrence_matrix[ current_word ][ context_word ] += 1

    return co_ocurrence_matrix
```

Doing Transformations

- We can now do more complex transformations!

- Here we will see:
  - Cosine Similarity
  - PMI (Pointwise Mutual Information)
  - SVD (Singular Value Decomposition)

- But there's others you can try!

Cosine Similarity

- Measures how close two vectors are to each other
- We do this by checking the cosine between them
- Let’s do funny math to justify this!

Cosine Similarity

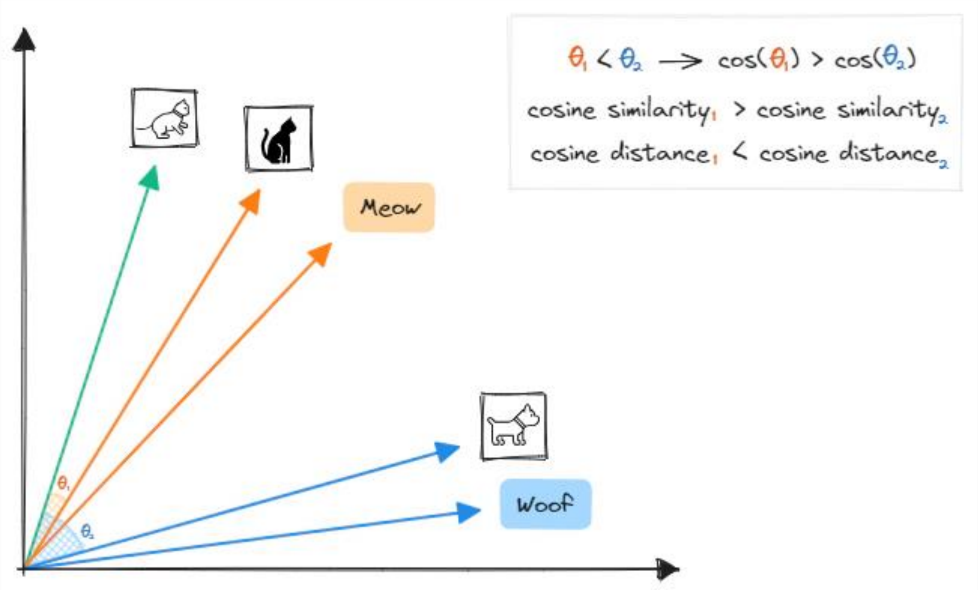


Image from Foteini Savvidou's blog

Cosine Similarity

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \|\mathbf{B}\| \cos \theta$$

Cosine Similarity

$$S_c(\mathbf{A}, \mathbf{B}) = \cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

Cosine Similarity

For ease of convenience, let’s define:

- $\text{norm} = \text{np.linalg.norm}$
- $X = \text{co\_ocurrence}[x]$
- $Y = \text{co\_ocurrence}[y]$

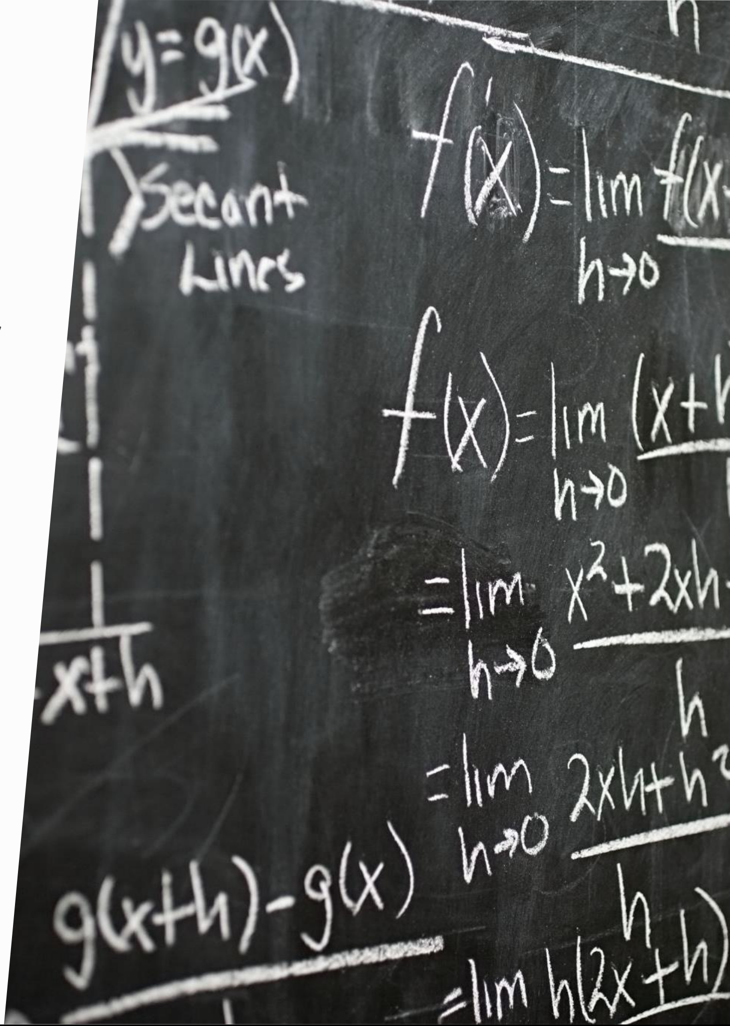
Cosine Similarity

$$S_c(x, y) = \frac{\mathrm{np.dot}(X, Y)}{\mathrm{norm}(X)\mathrm{norm}(Y)}$$

Pointwise Mutual Information (PMI)

- Compares the probability that two events occur together as opposed to if they were independent

- For us it means that we compare:
  - How often pairs of words co-occur
  - How often these words appear on their own



Pointwise Mutual Information (PMI)

$$pmi(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

Pointwise Mutual Information (PMI)

$$pmi(x, y) = \log_2 \frac{\text{co\_ocurrence}[x][y]}{\text{word\_count}[x] \ \cdot \text{word\_count}[y]}$$

Pointwise Mutual Information (PMI)

$$pmi(x, y) = \log_2 \frac{\text{co\_ocurrence}[x][y]}{\text{word\_count}[x] \cdot \text{word\_count}[y]}$$

What if we haven’t seen two words together??

Positive Pointwise Mutual Information (PPMI)

- PMI is useful when it is positive
  - It gives us a metric of how similar two words are
  - This can be somewhat easily interpreted

- When it is negative?
  - It is affected by how often rare words appear
  - Is unbound
  - This makes it hard to interpret

Positive Pointwise Mutual Information (PPMI)

$$ppmi(x, y) = \max(pmi(x, y), 0)$$

Singular Value Decomposition (SVD)

- Identifies the dimensions that contain the most information
- We can keep just the dimensions that are the most information-dense
- Less dimensions = less time making calculations

Singular Value Decomposition (SVD)

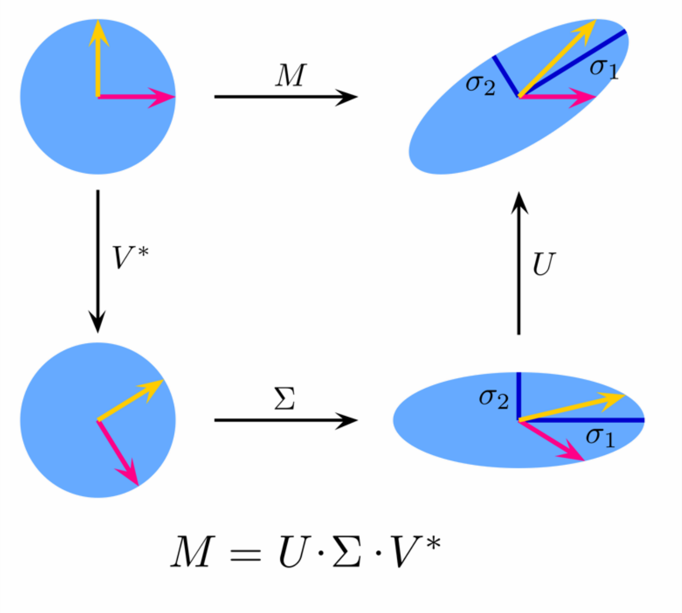


Image taken from the Wikipedia article on SVD

Singular Value Decomposition (SVD)

- We will skip the technicalities in this presentation

- NumPy has an implementation that is easy to use:

```
numpy.linalg.svd
```

- It can be used with the sparse matrices we have been using!

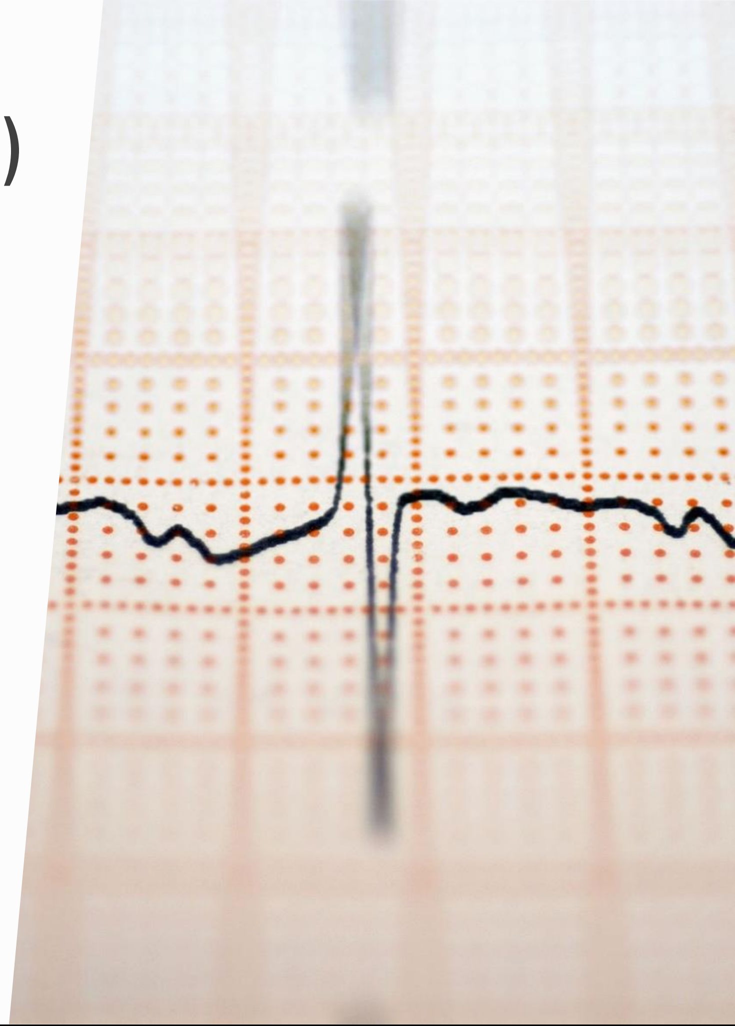
Singular Value Decomposition (SVD)

```
def do_co_ocurrence(text, context_window):

    co_ocurrence_matrix = { }

    preprocessed_text = preprocess(text)

    for i, current_word in enumerate(preprocessed_text):

        if current_word not in co_ocurrence_matrix:
            co_ocurrence_matrix[ current_word ] = {}

        min_context = max( 0 , i-context_window )
        max_context = min( len(preprocessed_text) , i+context_window)

        context = preprocess( min_context , max_context)

        for context_word in context:

            if context_word != current_word:

                if context_word not in co_ocurrence_matrix[ current_word ]:
                    co_ocurrence_matrix[ current_word ][ context_word ] = {}

                co_ocurrence_matrix[ current_word ][ context_word ] += 1

    return co_ocurrence_matrix
```

Singular Value Decomposition (SVD)

- When using SVD, we get three matrices:
  - U represents the rotations we did
  - Σ represents how we stretched the space
  - V is the information we originally had

- We want to keep the first few dimensions of U and of Σ



Singular Value Decomposition (SVD)

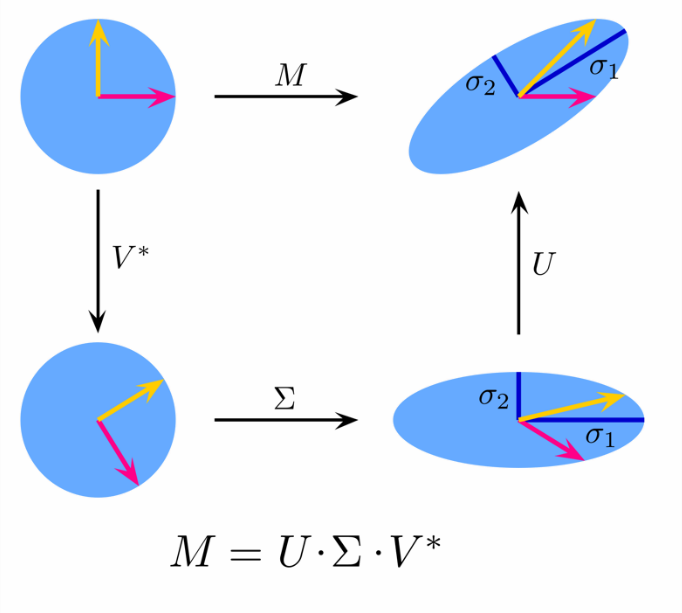


Image taken from the Wikipedia article on SVD

How Do We Use These for Downstream Tasks?

- In the end they create matrices/layers that:
  - Take the ID of a word as an input
  - Give a vector representation as an output

- We can then slot these as features for our favourite ML algorithms
  - We call these “embedding layers”

Questions?

- Yes, this is a lot of information
- You will get a chance to play with these concepts in the notebook
- Still, if you have any questions, don't hesitate to ask

GÖTEBORGS
UNIVERSITET

SPRÅKBANKENTEXT

Ricardo Muñoz Sánchez
ricardo.munoz.sanchez@svenska.gu.se
rimusa.github.io