Embeddings with MOWL

John Beverley

Assistant Professor, *University at Buffalo*
Co-Director, *National Center for Ontological Research*
Affiliate Faculty, *Institute of Artificial Intelligence and Data Science*

Outline

- The Gostak
- Live Demo
- Engineering Session

Outline

- The Gostak
- Live Demo
- Engineering Session

When you see a sentence, how do you know what it means?

The Gostak

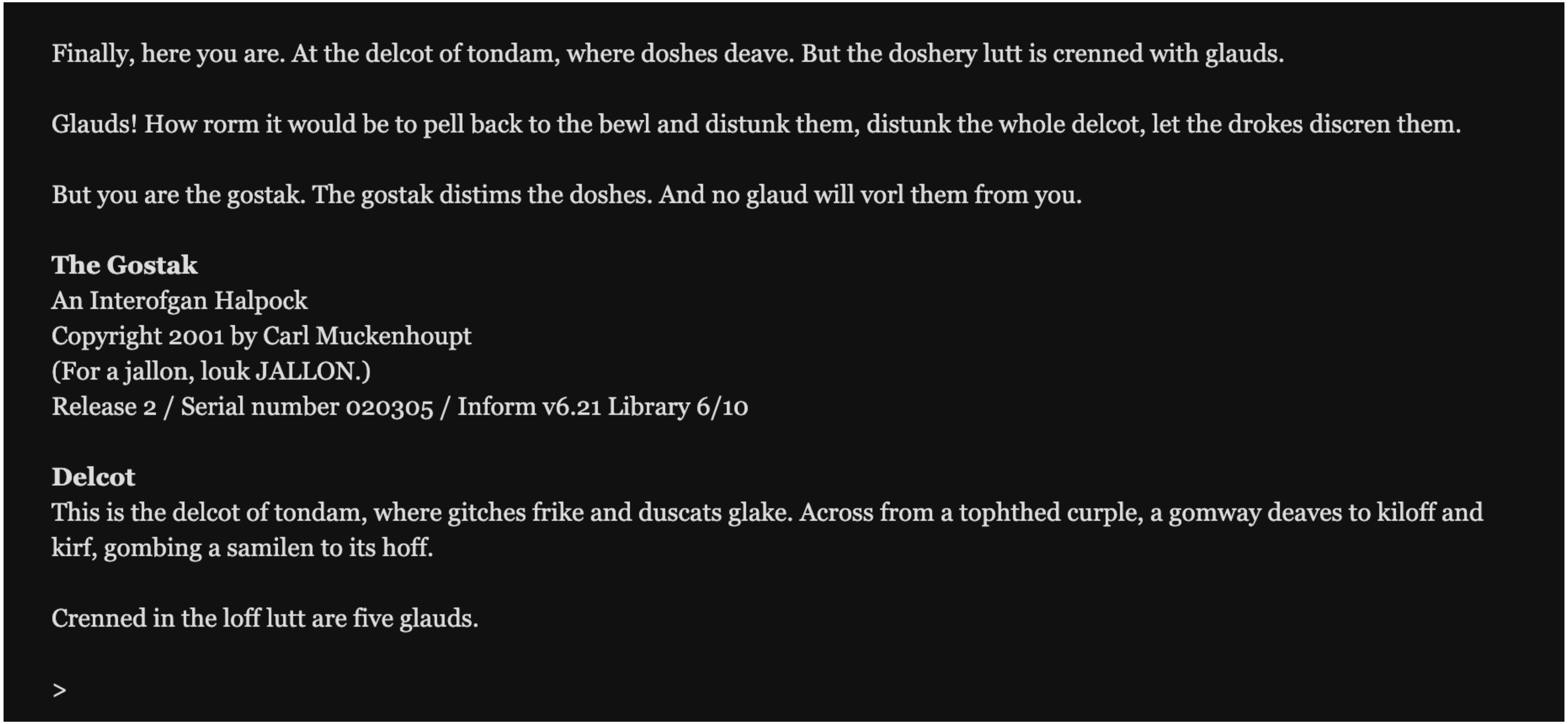
Can one extract **meaning** from the **shapes of terms** even when you don't know their meanings?

Group Exercise

- Can one extract **meaning** from the **shapes of terms** even when you don't know their meanings?

- Follow the link below to play **The Gostak**

http://iplayif.com/?story=http%3A%2F%2Fwww.ifarchive.org%2Fif-archive%2Fgames%2Fzcode%2Fgostak.z5



```
Finally, here you are. At the delcot of tondam, where doshes deave. But the doshery lutt is crenned with glauds.

Glauds! How rorm it would be to pell back to the bewl and distunk them, distunk the whole delcot, let the drokes discren them.

But you are the gostak. The gostak distims the doshes. And no glaud will vorl them from you.

The Gostak
An Interofgan Halpock
Copyright 2001 by Carl Muckenhoupt
(For a jallon, louk JALLON.)
Release 2 / Serial number 020305 / Inform v6.21 Library 6/10

Delcot
This is the delcot of tondam, where gitches frike and duscats glake. Across from a tophthed curple, a gomway deaves to kiloff and
kirf, gombing a samilen to its hoff.

Crenned in the loff lutt are five glauds.

>
```

Bewly dedges

This is a halpock. As in all halpocks, you doatch at it about what to do in a camling by louking murr English dedges, and the halpock louks back. Durly, you could rask a chuld (if you rebbed one) by louking:

```
>RASK THE CHULD
Rasked.
```

You could then tunk the chuld, or pob it at a tendo, or whatever:

```
>DURCH CHULD
Nothing heamy results.
```

```
>POB MY CHULD AT THE GHARMY TENDO
The tendo leils it.
```

You can use multiple objects with some dapes:

```
>RASK THE GELN, THE POTCHNER, AND THE FACK
>POB ALL EXCEPT DRAGUE
```

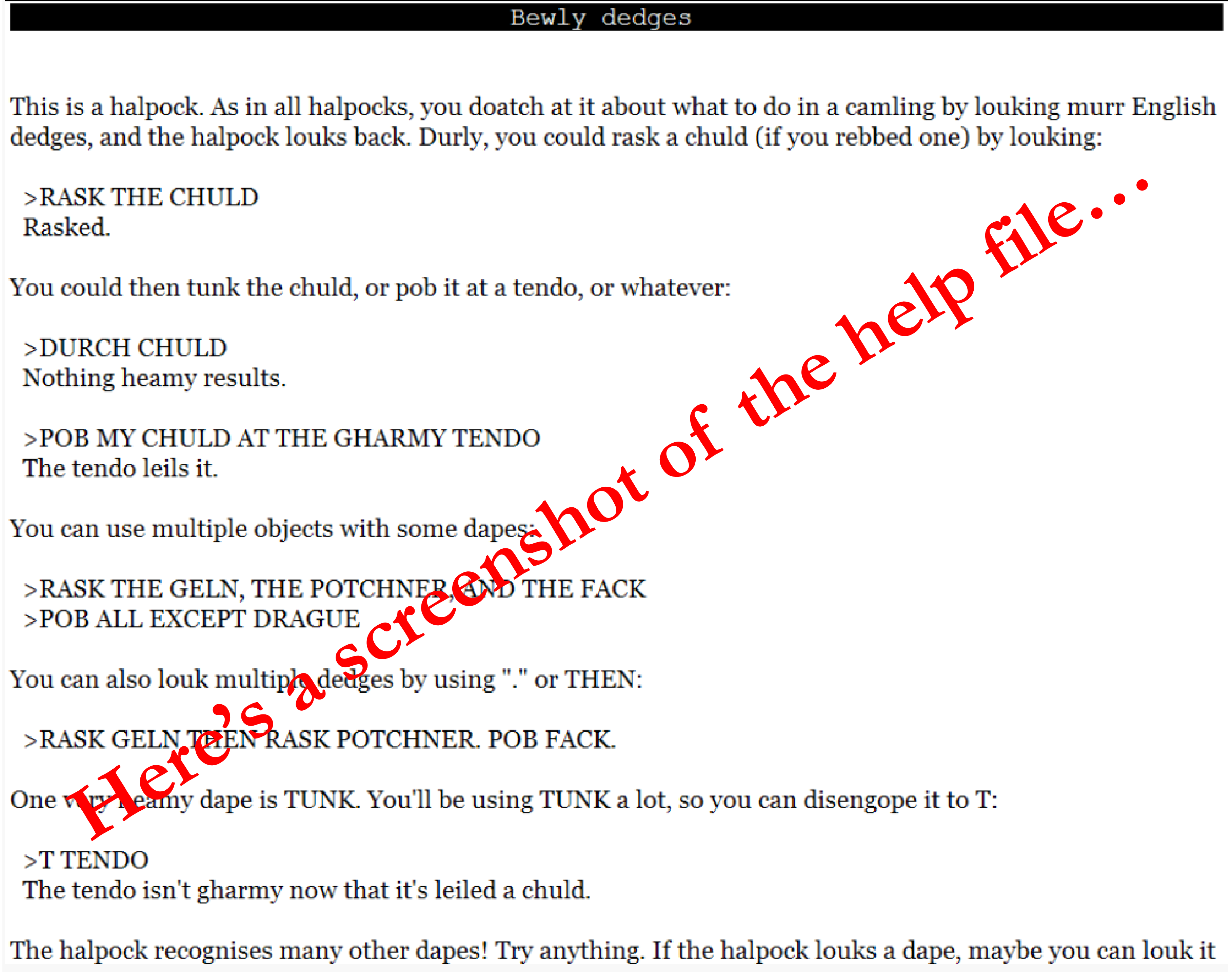
You can also louk multiple dedges by using "." or THEN:

```
>RASK GELN THEN RASK POTCHNER. POB FACK.
```

One very heamy dape is TUNK. You'll be using TUNK a lot, so you can disengope it to T:

```
>T TENDO
The tendo isn't gharmy now that it's leiled a chuld.
```

The halpock recognises many other dapes! Try anything. If the halpock louks a dape, maybe you can louk it



Bewly dedges

This is a halpock. As in all halpocks, you doatch at it about what to do in a camling by louking murr English dedges, and the halpock louks back. Durly, you could rask a chuld (if you rebbed one) by louking:

```
 >RASK THE CHULD
 Rasked.
```

You could then tunk the chuld, or pob it at a tendo, or whatever:

```
 >DURCH CHULD
 Nothing heamy results.

 >POB MY CHULD AT THE GHARMY TENDO
 The tendo leils it.
```

You can use multiple objects with some dapes:

```
 >RASK THE GELN, THE POTCHNER, AND THE FACK
 >POB ALL EXCEPT DRAGUE
```

You can also louk multiple dedges by using "." or THEN:

```
 >RASK GELN THEN RASK POTCHNER. POB FACK.
```

One very heamy dape is TUNK. You'll be using TUNK a lot, so you can disengope it to T:

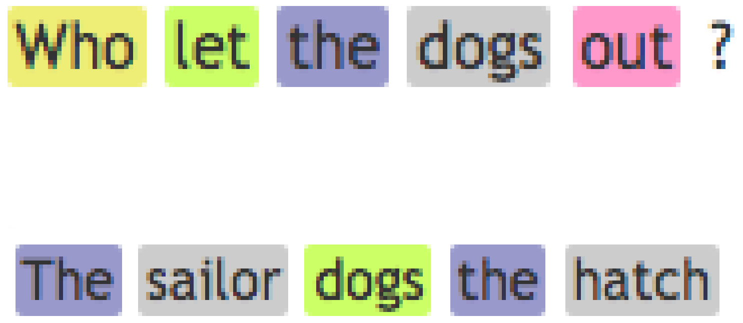
```
 >T TENDO
 The tendo isn't gharmy now that it's leiled a chuld.
```

The halpock recognises many other dapes! Try anything. If the halpock louks a dape, maybe you can louk it

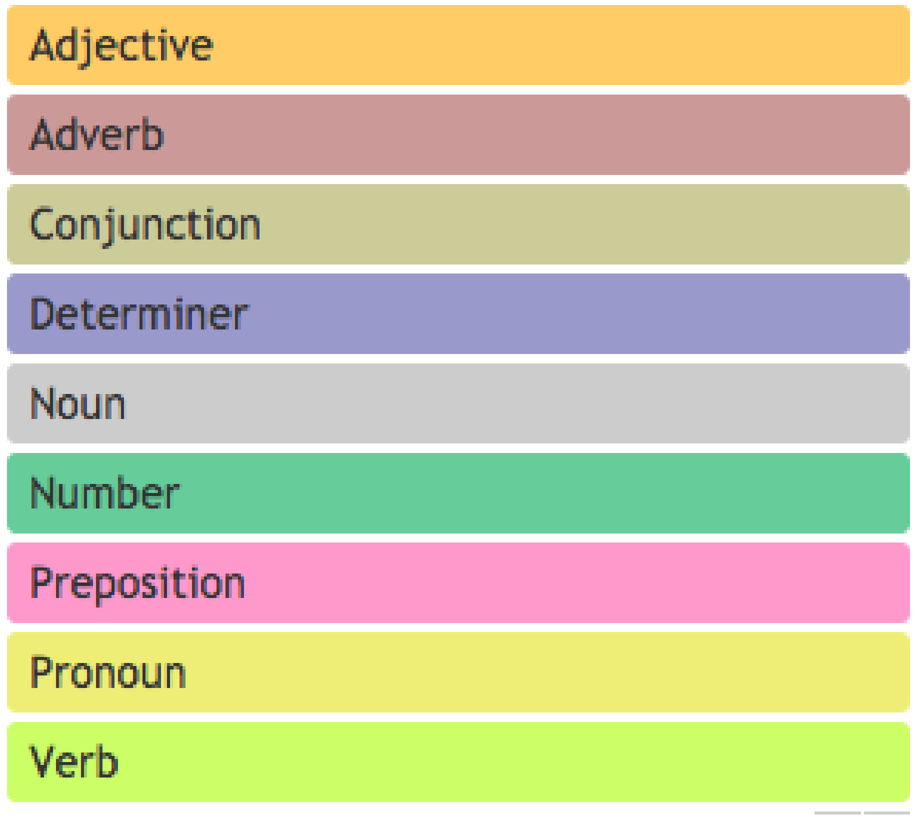
Here's a screenshot of the help file…

Parts of Speech Tagging

Annotating words in text based on the parts of speech


Who let the dogs out ?

The sailor dogs the hatch


Adjective

Adverb

Conjunction

Determiner

Noun

Number

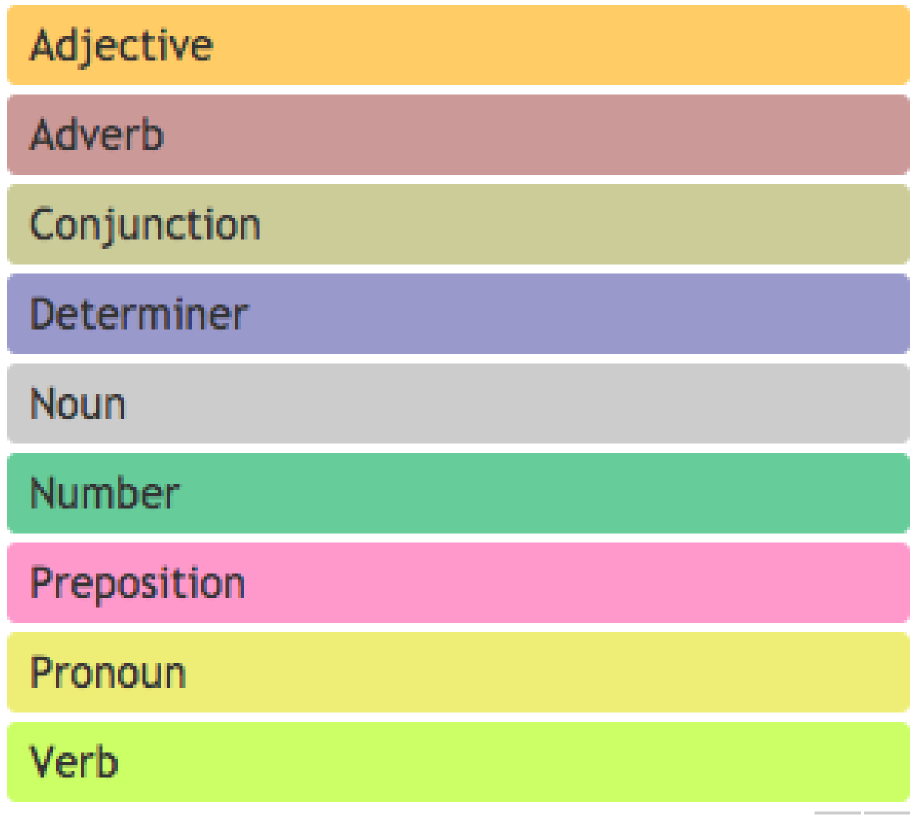
Preposition

Pronoun

Verb

Gostak

That is not a dape I recognize



Adjective
Adverb
Conjunction
Determiner
Noun
Number
Preposition
Pronoun
Verb

Embeddings

- An embedding is a mapping from one structure to another that preserves features of interest from the former in the latter

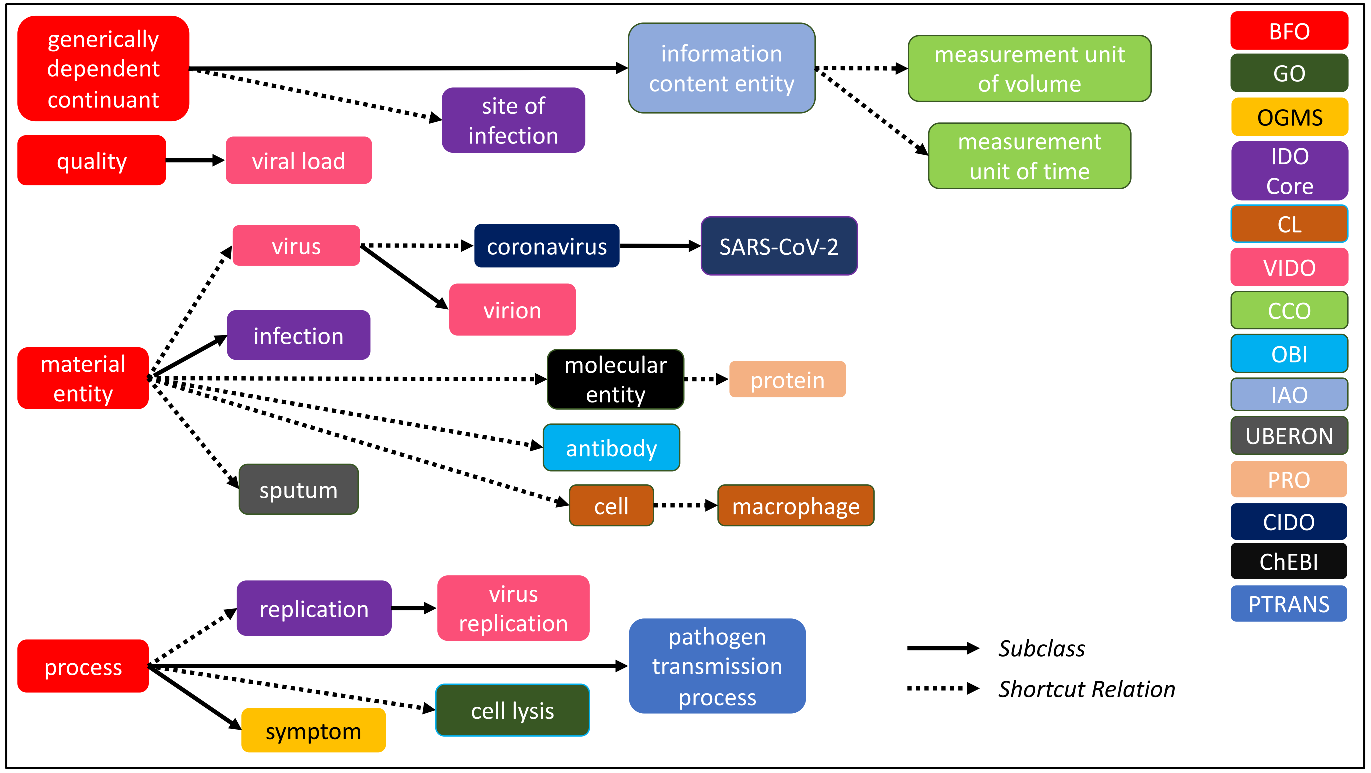
- We were, in a sense, manually creating an embedding from the Gostak sentence structures into more familiar structures

- Observing certain linguistic features were **preserved**

Embeddings

- An embedding is a mapping from one structure to another that preserves features of interest from the former in the latter
- We were, in a sense, manually creating an embedding from the Gostak sentence structures into more familiar structures
- Observing certain linguistic features were **preserved**

Who has the time to do that manually?



Following **replication**, **cell lysis** of **SARS-CoV-2 coronavirus virions** causes **host cells** to **release molecules** which **function** to warn nearby **cells**. When **recognized** by **epithelial cells**, **endothelial**, and **alveolar macrophages**, **proteins** such as **IL-6**, **IP-10**, and **MCPI**, are **released** which **attract T cells**, **macrophages**, and **monocytes to** the **site of infection**, **promoting inflammation**. In **disordered immune systems**, **immune cells accumulate in** the **lungs**, then **propagate to** and **damage** other **organs**. In **normal immune systems**, **inflammation attracts T cells** which **neutralize** the **virus at** the **site of infection**. **Antibodies circulate**, preventing **SARS-CoV-2 infection**, and **alveolar macrophages recognize SARS-CoV-2** and **eliminate virions** via **phagocytosis** [111-113].

Distribution Hypothesis

- "A word is characterized by the company it keeps." -Firth

- The **distribution hypothesis** underlies all modern embedding models

- The central idea being that words that occur in similar contexts tend to have similar meanings

"The doctor treated the patient." doctor ↔ hospital, nurse, patient

"The nurse assisted in surgery." nurse ↔ hospital, doctor, patient

Distribution Hypothesis

FIELDS HAVE THEIR OWN LEXICONS

- "A word is characterized by the company it keeps." -Firth
- The **distribution hypothesis** underlies all modern embedding models
- The central idea being that words that occur in similar contexts tend to have similar meanings

"The doctor treated the patient." doctor ↔ hospital, nurse, patient

"The nurse assisted in surgery." nurse ↔ hospital, doctor, patient

Distribution Hypothesis

FIELDS HAVE THEIR OWN ~~LEXICONS~~ LINGUISTIC SILOS

- "A word is characterized by the company it keeps." -Firth
- The **distribution hypothesis** underlies all modern embedding models
- The central idea being that words that occur in similar contexts tend to have similar meanings

"The doctor treated the patient." doctor ↔ hospital, nurse, patient

"The nurse assisted in surgery." nurse ↔ hospital, doctor, patient

Embedding Targets

- Units of text are assigned IDs to create a vocabulary
  - **Characters**
  - Words
  - N-grams
  - Sentences

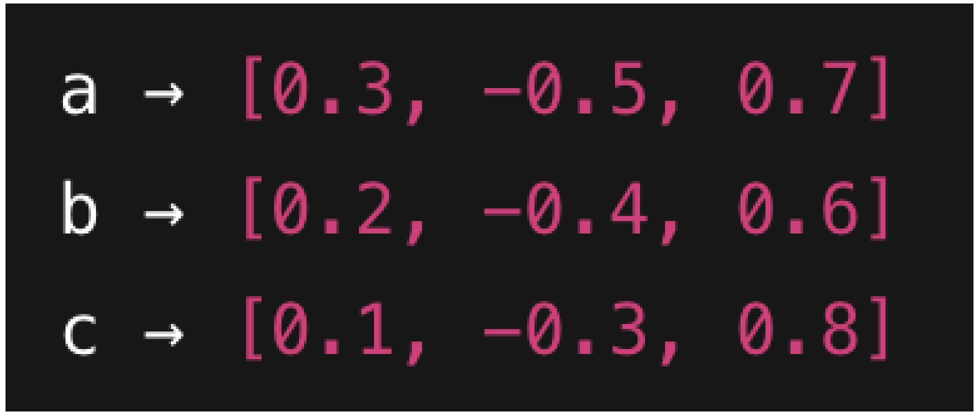
be bad → 250214

	0
a	1
b	2
c	3
d	4
e	5
f	6
g	7
h	8
i	9

Sumyyah Toonsi, Ontologies and text mining, MOWL Documentation. https://github.com/bio-ontology-research-group/mowl-tutorial/blob/main/slides/02_Ontologies%20and%20text%20mining.pdf

Character Embeddings

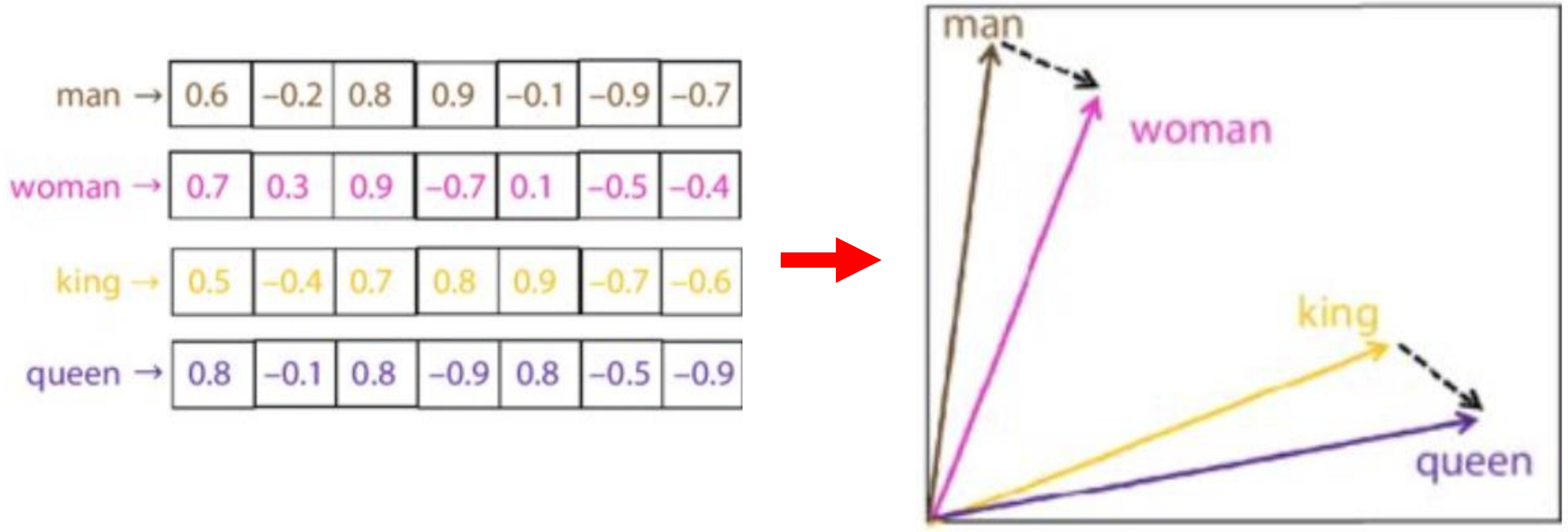
- The **simplest** embedding units
- Each character (a, b, c, …) is assigned a point in vector space
- Points capture morphological patterns (prefixes, suffixes, capitalization)



```
a → [0.3, -0.5, 0.7]
b → [0.2, -0.4, 0.6]
c → [0.1, -0.3, 0.8]
```

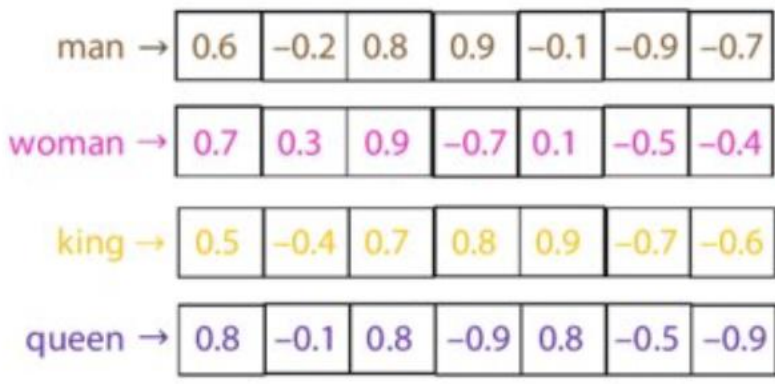
Embedding Targets

- Units of text are assigned IDs to create a vocabulary
  - Characters
  - **Words**
  - N-grams
  - Sentences

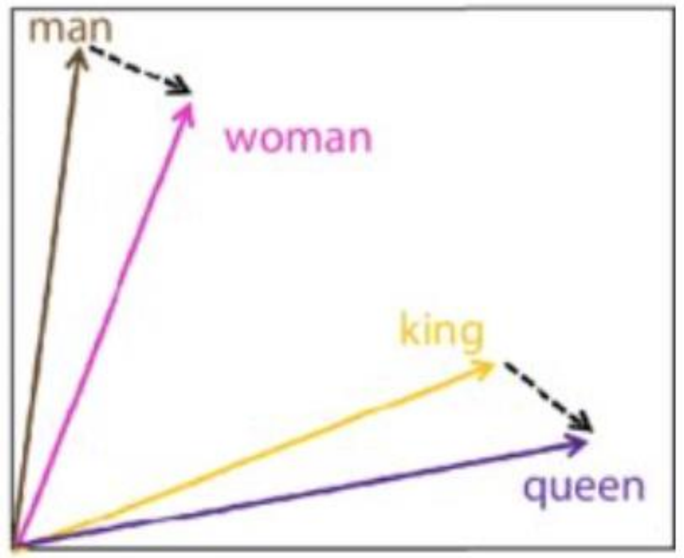


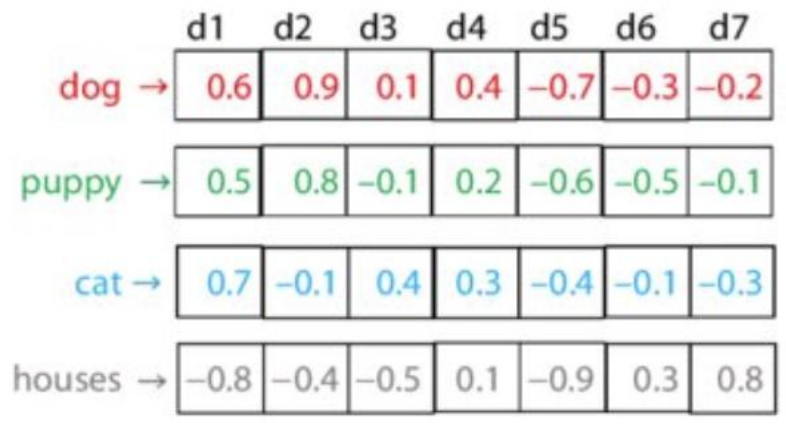
man →	0.6	–0.2	0.8	0.9	–0.1	–0.9	–0.7
woman →	0.7	0.3	0.9	–0.7	0.1	–0.5	–0.4
king →	0.5	–0.4	0.7	0.8	0.9	–0.7	–0.6
queen →	0.8	–0.1	0.8	–0.9	0.8	–0.5	–0.9

Khan, M. I. H., Sablani, S. S., Nayak, R., & Gu, Y. (2022). Machine learning-based modeling in food processing applications: State of the art. Comprehensive reviews in food science and food safety, 21(2), 1409–1438. https://doi.org/10.1111/1541-4337.12912

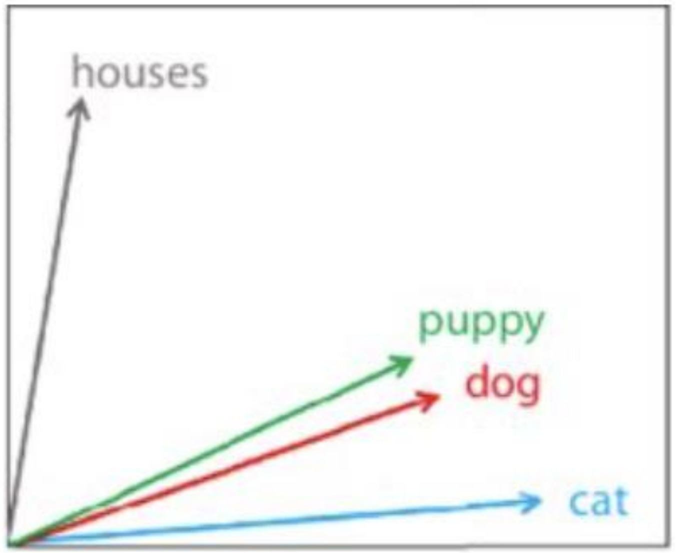


man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9





	d1	d2	d3	d4	d5	d6	d7
dog →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
puppy →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
cat →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

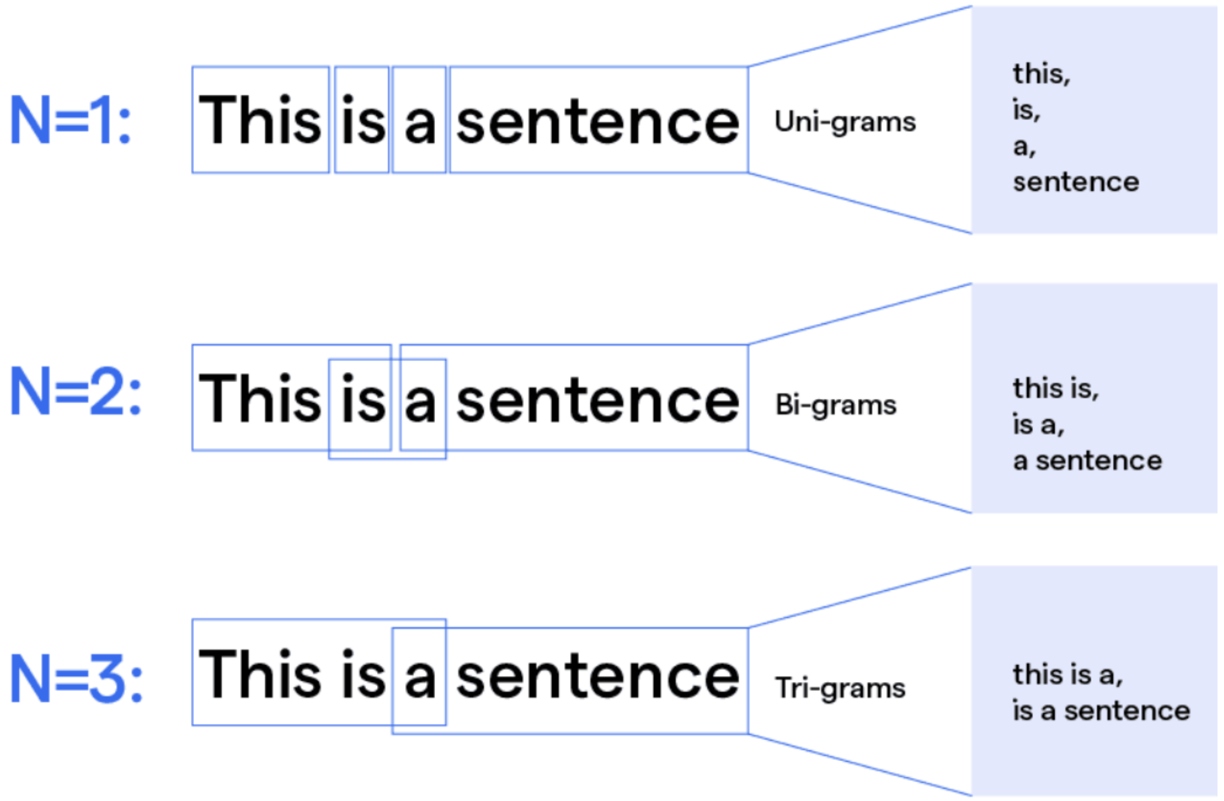


Reduction of words represented in a 7 dimensional space to a 2 dimensional space

Khan, M. I. H., Sablani, S. S., Nayak, R., & Gu, Y. (2022). Machine learning-based modeling in food processing applications: State of the art. Comprehensive reviews in food science and food safety, 21(2), 1409–1438. https://doi.org/10.1111/1541-4337.12912

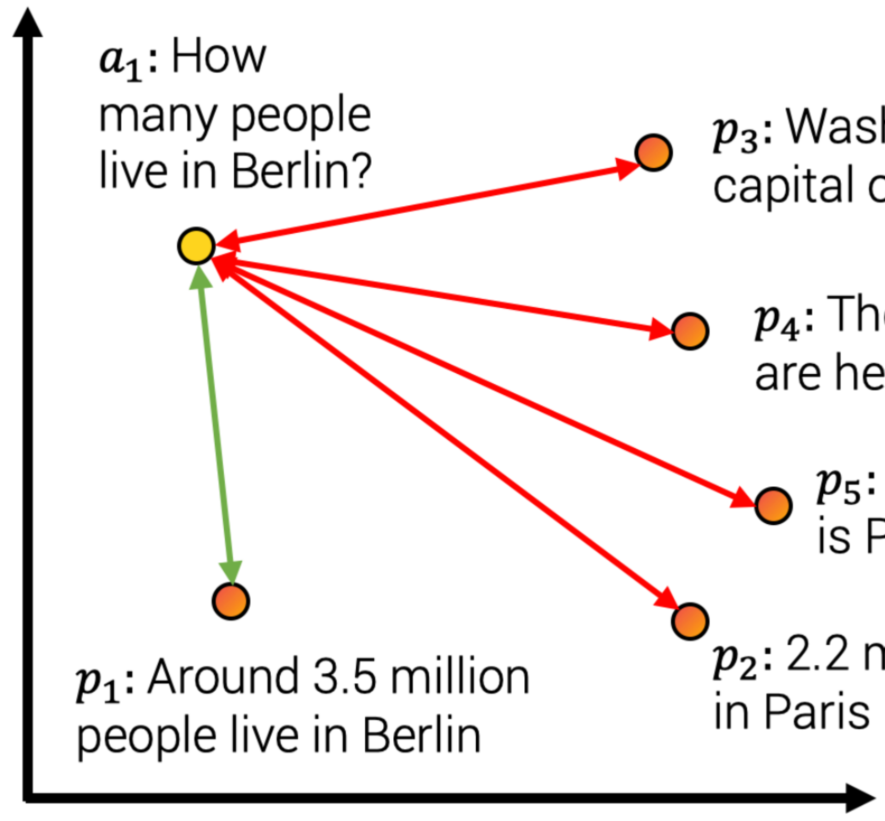
Embedding Targets

- Units of text are assigned IDs to create a vocabulary
  - Characters
  - Words
  - **N-grams**
  - Sentences



Embedding Targets

- Units of text are assigned IDs to create a vocabulary
  - Characters
  - Words
  - N-grams
  - **Sentences**


a_1: How many people live in Berlin?
p_3: Washington DC is the capital of the US
p_4: The 2021 Olympics are held in Japan
p_5: The capital of France is Paris
p_1: Around 3.5 million people live in Berlin
p_2: 2.2 million people live in Paris

Embedding Targets

- Units of text are assigned IDs to create a vocabulary
  - Characters
  - Words
  - N-grams
  - Sentences
  - **OWL**

mOWL

Gostak Analogy

Gostak	Word Embedding
Meaning from surrounding words	Meaning encoded in nearby vectors
Context defines concept roles	Context windows define embeddings
Players learn categories (noun, verb)	Models learn syntactic/semantic structure
No explicit labels	Unsupervised learning

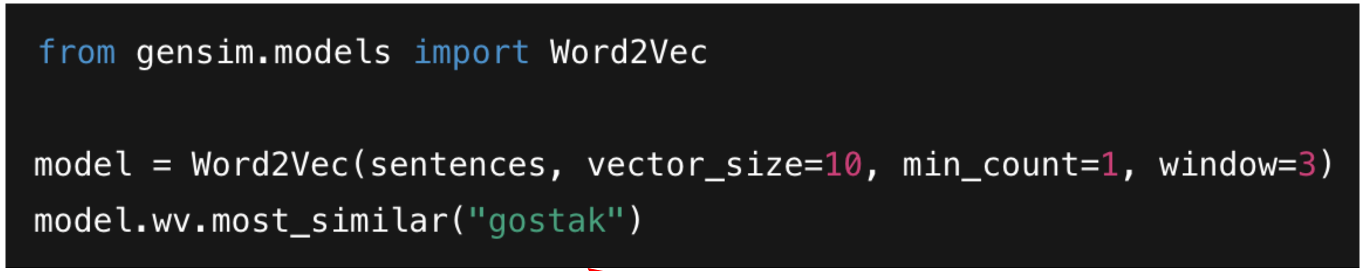
```python
from gensim.models import Word2Vec

model = Word2Vec(sentences, vector_size=10, min_count=1, window=3)
model.wv.most_similar("gostak")
```

```python
sentences = [
    ["the", "gostak", "distims", "the", "doshes"],
    ["the", "doshes", "are", "flimsy"],
    ["to", "distim", "is", "to", "make", "doshes"],
    ["a", "gostak", "must", "distim", "well"]
]
```

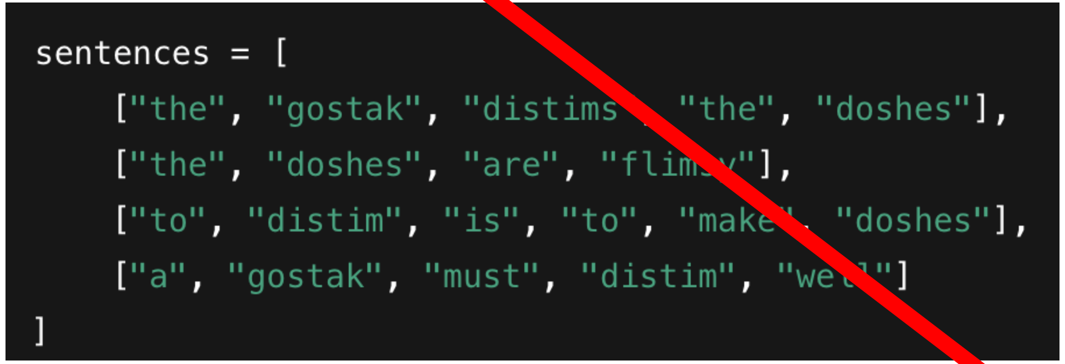
```
[('distims', 0.312),
 ('doshes', 0.278),
 ('make', 0.166),
 ('the', 0.094),
 ('well', 0.061),
 ('must', 0.043),
 ('flimsy', -0.010)]
```

Trained a distributional model on made-up data; model learned relational structures; meaning emerged through co-occurrence statistics

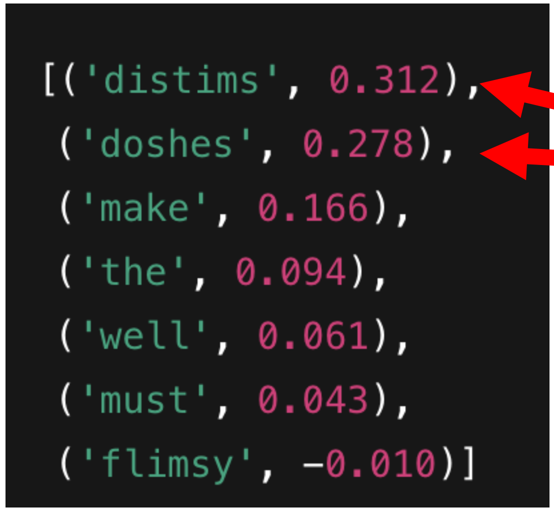
gostak appears most often near distims and doshes, so they end up close in embedding space.



```
from gensim.models import Word2Vec

model = Word2Vec(sentences, vector_size=10, min_count=1, window=3)
model.wv.most_similar("gostak")
```



```
sentences = [
    ["the", "gostak", "distims", "the", "doshes"],
    ["the", "doshes", "are", "flimsy"],
    ["to", "distim", "is", "to", "make", "doshes"],
    ["a", "gostak", "must", "distim", "well"]
]
```



```
[('distims', 0.312),
 ('doshes', 0.278),
 ('make', 0.166),
 ('the', 0.094),
 ('well', 0.061),
 ('must', 0.043),
 ('flimsy', -0.010)]
```

Trained a distributional model on made-up data; model learned relational structures; meaning emerged through co-occurrence statistics

gostak appears most often near distims and doshes, so they end up close in embedding space.

Go play it again, I know you want to

Outline

- The Gostak
- Live Demo
- Engineering Session

Preparation

- Navigate to the Artifact Ontology:

  https://www.commoncoreontologies.org/ArtifactOntology

- Open it in Protégé

- Identify **5 classes** that seem under-specified and write one hypothesis each about how each might be better specified, e.g.

  "Sensor should have an OWL axiom connecting it to a sensor function."

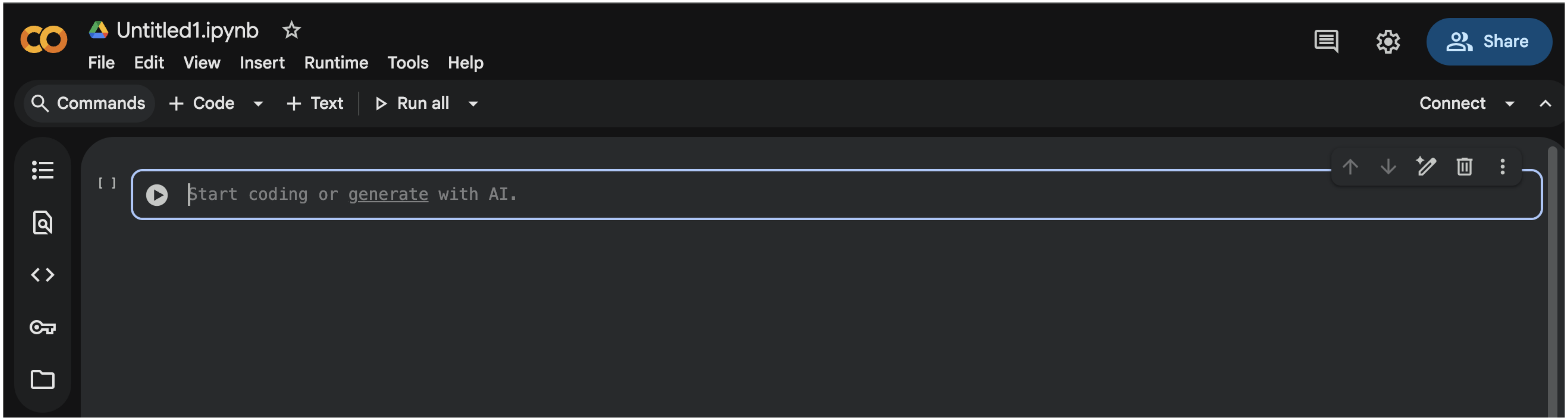
Live Demo

- I want to highlight the training, validation, and testing ideas
- I will do so using Word2Vec since it’s much easier to work with than MOWL, e.g. no need for Java, PyTorch, etc.
- That said, the script I’m using here **knows nothing about OWL semantics**; it treats the ontology as a plain graph

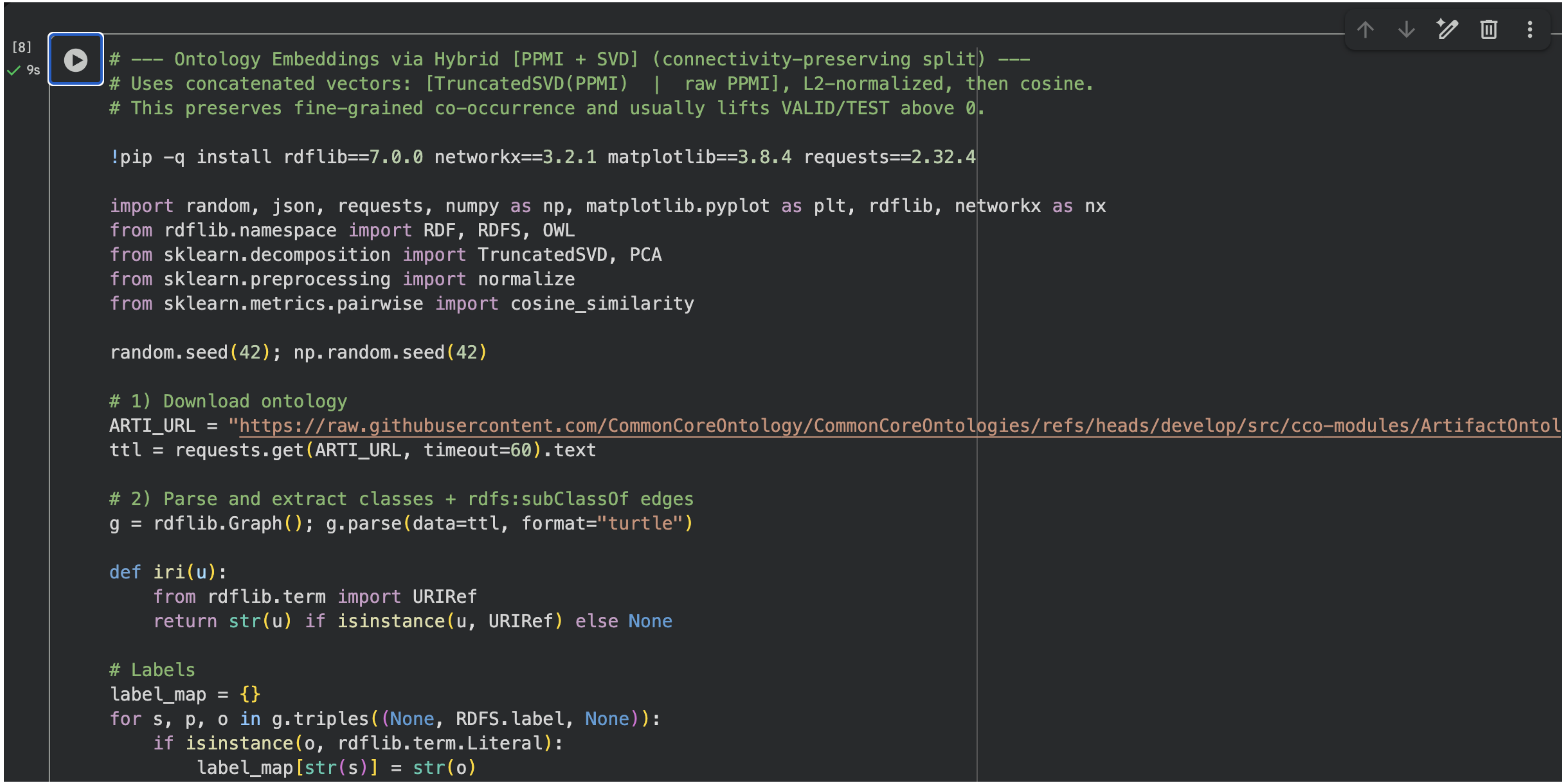
You’ll use MOWL for your project…

Google colab

https://colab.research.google.com/



https://colab.research.google.com/



```
[8]
9s
# --- Ontology Embeddings via Hybrid [PPMI + SVD] (connectivity-preserving split) ---
# Uses concatenated vectors: [TruncatedSVD(PPMI)  |  raw PPMI], L2-normalized, then cosine.
# This preserves fine-grained co-occurrence and usually lifts VALID/TEST above 0.

!pip -q install rdflib==7.0.0 networkx==3.2.1 matplotlib==3.8.4 requests==2.32.4

import random, json, requests, numpy as np, matplotlib.pyplot as plt, rdflib, networkx as nx
from rdflib.namespace import RDF, RDFS, OWL
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.preprocessing import normalize
from sklearn.metrics.pairwise import cosine_similarity

random.seed(42); np.random.seed(42)

# 1) Download ontology
ARTI_URL = "https://raw.githubusercontent.com/CommonCoreOntology/CommonCoreOntologies/refs/heads/develop/src/cco-modules/ArtifactOntol
ttl = requests.get(ARTI_URL, timeout=60).text

# 2) Parse and extract classes + rdfs:subClassOf edges
g = rdflib.Graph(); g.parse(data=ttl, format="turtle")

def iri(u):
    from rdflib.term import URIRef
    return str(u) if isinstance(u, URIRef) else None

# Labels
label_map = {}
for s, p, o in g.triples((None, RDFS.label, None)):
    if isinstance(o, rdflib.term.Literal):
        label_map[str(s)] = str(o)
```

https://github.com/Applied-Ontology-Education/Ontology-Tradecraft/blob/main/documentation/embedding_example.py

What the Code Does

- Extracts classes, labels, and rdfs:subClassOf from Artifact Ontology

Every time you see X rdfs:subClassOf Y, remember that as a connection from X to Y

- Builds two lists:
  - Classes = all the nodes
  - Edges = all the subclass links
- Now we have a directed graph where each node is a class

Splitting the Graph

- Before we start learning, we split the edges into:
  - **Training Edges** - What the computer can see
  - **Validation Edges** - To check learning quality
  - **Test Edges** - To see if it can guess unseen ones

- We do this carefully so we don't remove the only link a node has

- Otherwise, some classes would be floating in space, never visited, and the computer couldn't learn about them

Random Walk

- Imagine you start at a random class, and at each step you follow a subclass or superclass link for about 20 hops, resulting in walks like:

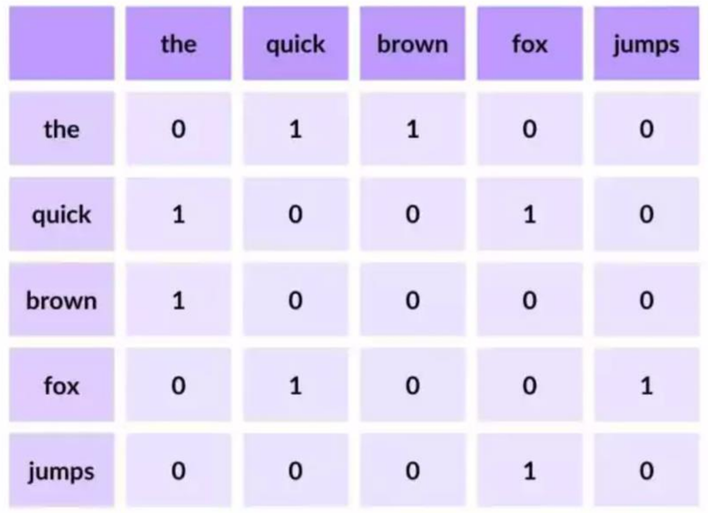
  Laser → Optical Instrument → Measurement Device → Artifact Function …

- Do this thousands of times and you get a corpus of walks

Counting

- From those walks, we count:
  - How often each pair of classes co-occur in the same walks

- We also give extra weight to:
  - Direct subclass edges
  - Grandparent/grandchild pairs (2-hops),
  - Neighbors within 3 hops



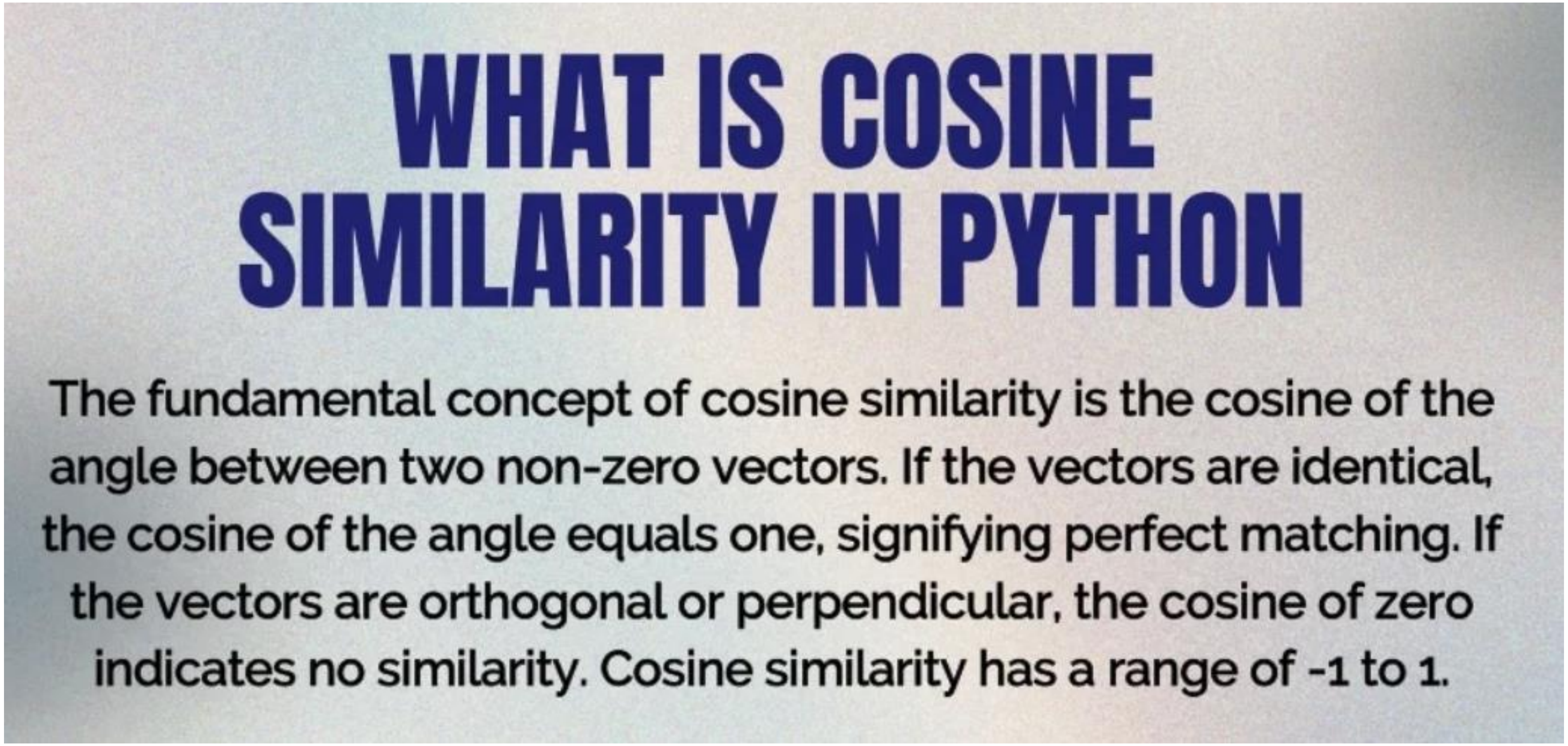
	the	quick	brown	fox	jumps
the	0	1	1	0	0
quick	1	0	0	1	0
brown	1	0	0	0	0
fox	0	1	0	0	1
jumps	0	0	0	1	0

- These counts go into a giant table called the **co-occurrence matrix**

Validation

- While classes such as class A and class B are in the training set, there are triples such as “A rdfs:subClassOf B” that are not
- Rather, they are **held back in the validation set**
- Given what the model knows about A and B, do they **look like** a subclass pair?

Cosine Similarity



WHAT IS COSINE SIMILARITY IN PYTHON

The fundamental concept of cosine similarity is the cosine of the angle between two non-zero vectors. If the vectors are identical, the cosine of the angle equals one, signifying perfect matching. If the vectors are orthogonal or perpendicular, the cosine of zero indicates no similarity. Cosine similarity has a range of -1 to 1.

Cosine Similarity

- Every class has been turned into a vector; cosine similarity measures the angle between vectors where:
  - 1.0 → identical direction (very similar)
  - 0.0 → unrelated
  - -1.0 → opposite meaning (very different)

- For example, if **cosine(Laser, Optical Instrument) = 0.95**, then the model thinks they live in almost the same direction, i.e. very similar

- If it's 0.0 or negative, they're far apart

Validation

- We collect those similarity scores into a list, then compute mean and percentiles
- We pick a threshold (τ), e.g. 60th percentile, such that if cosine $\geq \tau$, we consider it a “predicted subclass link”
- Percentiles help us see what counts as “typical” similarity for real subclass pairs

Threshold

- We need a rule to decide when the model is confident enough to "predict" a subclass relation

- If we pick a threshold τ of 60th percentile, this means:

If the cosine similarity between two classes is higher than what 60% of the real subclass pairs had, we'll count it as a predicted link.

- Suppose the 60th percentile = 0.70; then $\tau = 0.70$ means:
  - If cosine $\geq 0.70 \rightarrow$ model predicts "A is a subclass of B"
  - If cosine $< 0.70 \rightarrow$ model does not predict that

Not Arbitrary

- We need a data-driven way to decide what “close enough” means
- If we just picked $\tau = 0.9$ because it “feels right,” that would be arbitrary
- Instead, we look at the real subclass pairs to find a threshold that matches the data

Testing

- We trained our model on the **co-occurrence matrix** constructed from the Artifact Ontology

- In training, we determine whether we our model can recognize the subclass pairs that we kept separate in the training set

- In other words, we are checking whether the learned patterns generalize beyond what the model saw

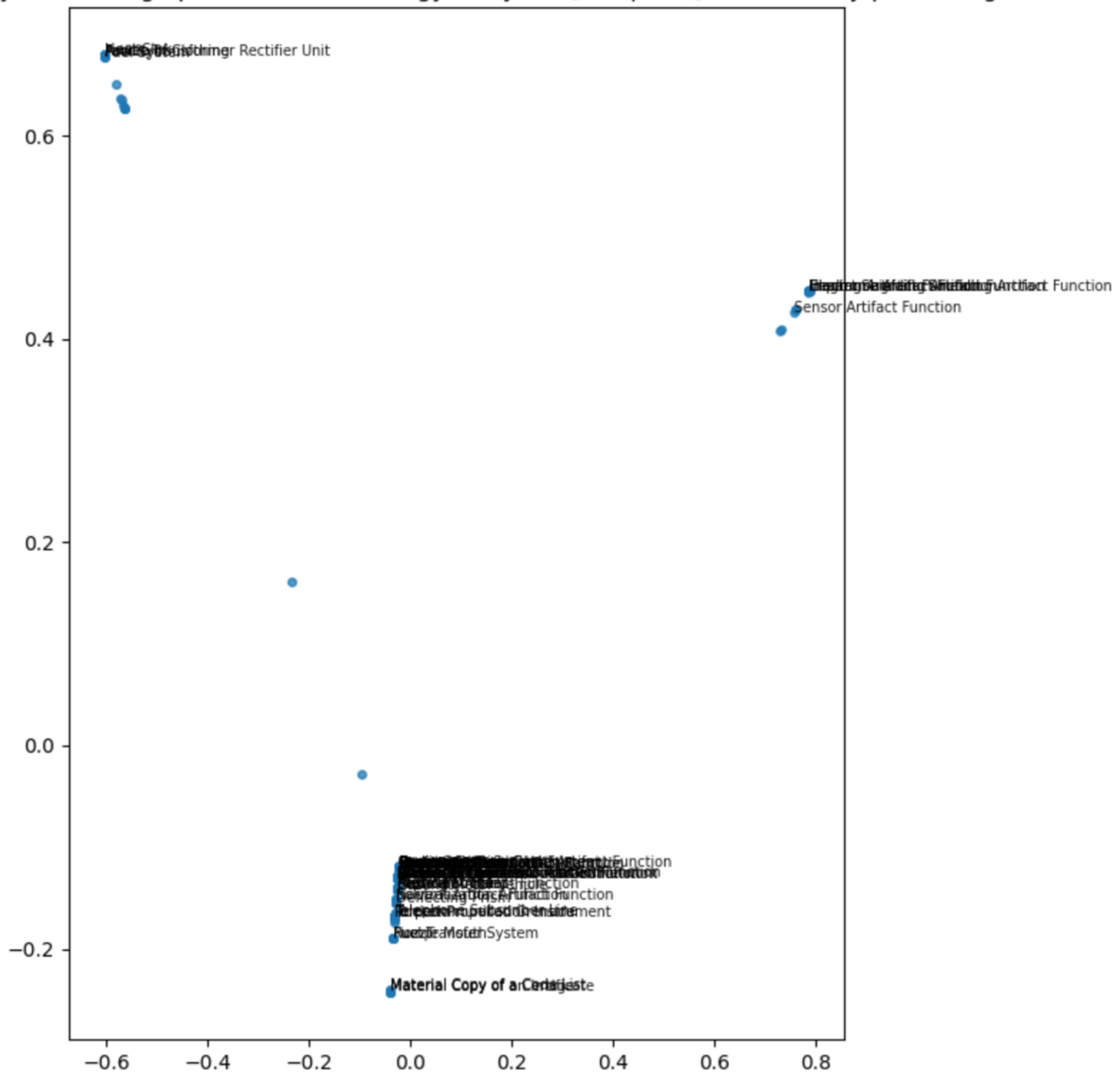
Visualization

- We look at all those high-dimensional points and asks:

If I had to draw a single line through this cloud of points that captures the biggest differences between them, which direction should that line go?

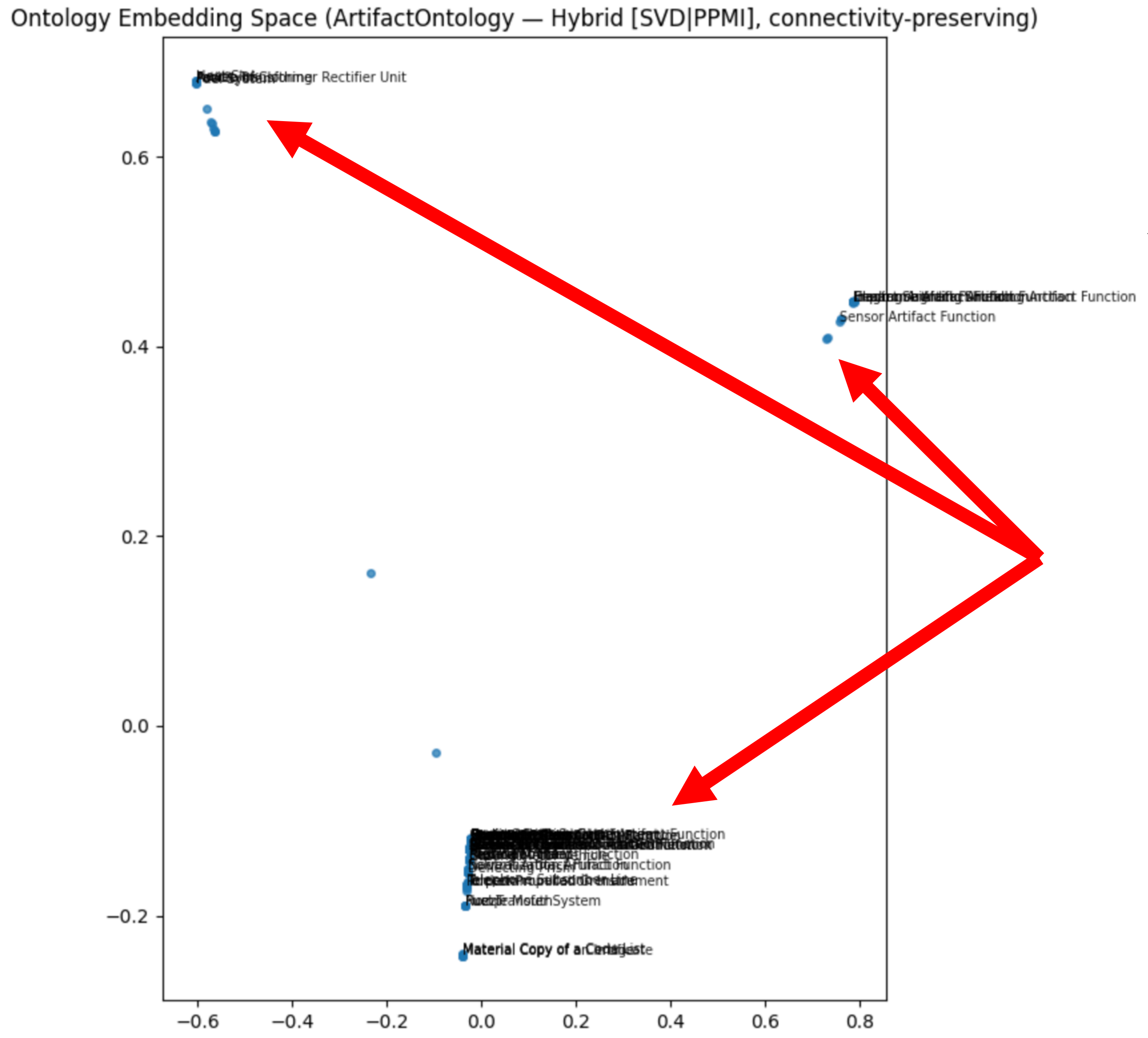
- That line becomes the X-Axis in the plot; second biggest differences become the Y-Axis

Ontology Embedding Space (ArtifactOntology — Hybrid [SVD|PPMI], connectivity-preserving)



ArtifactOntology.ttl has ~577 classes and ~564 subclass edges, roughly one edge per class

Most nodes sit in small trees under a **handful of root categories** e.g., Artifact Function, Weapon, Barcode, etc.



ArtifactOntology.ttl has ~577 classes and ~564 subclass edges, roughly one edge per class

Most nodes sit in small trees under a **handful of root categories** e.g., Artifact Function, Weapon, Barcode, etc.

Clustering

- Each point is a CCO class

- Because the Artifact Ontology is basically a few big hierarchies, the reduction strategy used in the script, collapses it into 3 clusters

- Within each cluster, nodes are locally similar because they share subclass chains and co-occurrence contexts

  - Cluster 1 is Artifact Function
  - Cluster 2 is Material Copy of Barcode
  - Cluster 3 is Informational Artifacts

Traditional Embeddings

- Traditional embeddings are good at grouping siblings and pattern-repeated subtrees
- Traditional embeddings are bad at retrieving held-out entailments when the training graph is sparse and an edge removal breaks the short paths needed for co-occurrence
- The Artifact Ontology is bushy in places but globally sparse

Put Another Way

- Our validation scores clustered around zero because held-out parent–child pairs (for testing) stopped co-occurring in training walks (**too sparse!!!)**

- Nearest neighbors still look intuitively right in 3 clusters, i.e. the distributional hypothesis on display

- Distributional embeddings see the clusters, but don’t reconstruct held-out subclass edges

Why the Distributional Approach Fails

- Relies on proximity, not entailment; random walks see “what is nearby”
- When we remove subclass edges for testing, many nodes become isolated in the graph, so embeddings have no path to learn those relationships
- Embeddings here only see structural co-occurrence, not implications
- Cosine similarity is great for “semantic closeness” but not for reasoning

Where MOWL Comes In

- MOWL (Machine Learning over OWL) bridges this gap
- Using ELEmbeddings, it interprets the ontology’s axioms as constraints in a geometric space
- Each class and relation is represented as a region; subclass and restriction axioms shape how those regions must overlap

Diagnostics

- The flat cosine results are **diagnostic**: distributional embeddings see surface structure, not meaningful entailment

- MOWL steps in to learn from logic by embedding the inference patterns themselves

- This shift from “who appears near whom” to “who entails whom” is what makes ontology-specific ML powerful

Outline

- The Gostak
- Live Demo
- Engineering Session

Project 5: Learning Objectives

- You will use machine learning to discover missing logical axioms in the Common Core Ontologies (CCO)

- While CCO's textual definitions contain rich semantics, most logical structure is still implicit

- Your task is to use MOWL semantic modeling to make that structure explicit with no manual intervention

Project 5: Learning Objectives

- Each student will select **exactly one** from the eleven CCO modules, train a MOWL model on its existing axioms, generate new candidate OWL 2 EL axioms from textual definitions, use MOWL to filter those candidates by learned semantic similarity, and then auto-merge the enriched ontology

- Students should coordinate so that no two students are using the same CCO module for this project.

Project 5: Learning Outcomes

- Understand how both symbolic and neural models can support ontology engineering
- Learn how to use MOWL to train embeddings over OWL 2 EL ontologies
- Combine MOWL semantic similarity for axiom filtering
- Validate ontology quality and consistency using ROBOT and ELK

Project Overview

- Each student will:
    - Choose a unique CCO module
    - Extract textual definitions into a CSV
    - Use an LLM to rewrite, normalize, and generate OWL 2 EL-compliant candidate axioms from the CSV
    - Train a MOWL model on the existing axioms in your module to learn a semantic embedding space
    - Filter candidates using both MOWL cosine similarity and LLM-based plausibility scores, keeping axioms above the learned threshold
    - Automatically merge accepted axioms back into the module
    - Automatically reason with ELK via ROBOT to check for logical consistency.

MOWL

- MOWL (Machine Learning with Ontologies) is a Python library for:
  - Converting OWL ontologies into graph data usable in deep learning
  - Training knowledge graph embedding models under logical constraints
  - Evaluating models for axiom prediction, link prediction, or ontology completion

mOWL

Python library for
Machine Learning with Ontologies

Practically Uses…

- Use design patterns to generate embeddings that can then be used for classification, clustering, etc.

- If missing axioms, train embeddings so the model suggests missing links

- Compute vector distances between entities to find ontologically "near" items for mapping

- Predict novel instances of classes based on ontological relationships, e.g. sensor instance → predict class/units with ontology embedding

Install and Setup

mOWL

Python library for
Machine Learning with Ontologies

https://mowl.readthedocs.io/en/latest/install/index.html

Remainder of Seminar

Make sure you are set up to work on Project 5

If you are, then make sure your classmates are set up appropriately

Otherwise, begin working on Project 5