Lecture 5

Embeddings

Lars A. Lochstoer

UCLA Anderson School of Management

Overview

Old school

- Word counts, bag-of-words, Document-term-matrix
- Very limited view of language

New school

- Embeddings
- Contextual awareness

Embedding models used in practice

Recall: Document term matrix (DTM)

Upload corpus

- Stem words, get rid of stop words
- Create list of all unique words in the corpus
  - Put these in column headers. N unique stemmed words, N columns
- For each document (row), count the number of times each of the words in the column header appears and enter that number in the corresponding column

	part	forward	look	statement	busi	section	\
AAPL_10_K_2000_clean.txt	86	40	14	158	80	24	
AAPL_10_K_2001_clean.txt	62	35	9	62	78	12	
AAPL_10_K_2002_clean.txt	76	32	8	76	96	13	
AAPL_10_K_2003_clean.txt	55	52	14	79	103	16	
AAPL_10_K_2004_clean.txt	51	47	16	72	105	18	
AAPL_10_K_2005_clean.txt	50	30	14	106	114	13	
AAPL_10_K_2006_clean.txt	79	18	6	203	154	14	
AAPL_10_K_2007_clean.txt	71	17	8	97	126	31	
AAPL_10_K_2008_clean.txt	54	17	8	110	133	9	
AAPL_10_K_2009_clean.txt	27	16	10	102	42	5	

Recall: Cosine similarity

Define a document D_j is j'th row of the DTM

Define a dictionary on the words in the corpus, D_i

- For instance, this could be a sentiment dictionary that has, say, 2 for each positive sentiment word, 0 for each neutral word, and -1 for each negative sentiment word.
- Alternatively, it could be a different document (e.g., row i in DTM)

Cosine similarity is then

$$C_{ij} = \frac{D_i' D_j}{\|D_i\| \|D_j\|}$$

Recall: Cosine similarity

- To see how this works, consider the following (short) documents

$$
\begin{aligned}
D_A &: \text{We expect demand to increase} \\
D_B &: \text{We expect worldwide demand to increase} \\
D_C &: \text{We expect weakness in sales}
\end{aligned}
$$

- Clearly, D_A and D_B are more similar than, say, D_A and D_C
- Let's calculate the cosine measure between A and B:

$$T(D_A, D_B) = [\text{we, expect, worldwide, demand, to, increase}]$$

Recall: Cosine similarity

- From last slide

$$T\left(D_A, D_B\right) = [\text{we, expect, worldwide, demand, to, increase}]$$

- Then

$$\begin{aligned} D_1^{TF} &= [1,1,0,1,1,1] \\ D_2^{TF} &= [1,1,1,1,1,1] \end{aligned}$$

- So cosine similarity is:

$$\frac{\left(D_1^{TF}\right)' D_2^{TF}}{\left\|D_1^{TF}\right\| \times \left\|D_2^{TF}\right\|}$$

$$\begin{aligned} &= \frac{1\times 1+1\times 1+0\times 1+1\times 1+1\times 1+1\times 1}{\sqrt{1^2+1^2+0^2+1^2+1^2+1^2}\times\sqrt{1^2+1^2+1^2+1^2+1^2+1^2}} \\ &= 0.91 \end{aligned}$$

DTM shortcomings

Multiple meanings of the same word

Same meaning of different words

Sparse, high-dimensional DTM matrix for large Corpus

Lack of context, overall meaning

Examples on next slides

Multiple meanings of same word

mouse (N)

- any of numerous small rodents...
- a hand-operated device that controls a cursor...

Here *mouse* is the lemma for mice, *sing* is lemma for sang, sung, etc.

- Even lemmas can have different meanings (word senses, polysemous)

We would like *word sense disambiguation*

- The task of determining which sense of a word is being used in a particular context

Same meaning of different words

A DTM treats synonyms as separate, whereas in usual language they are not

Examples:

- couch/sofa
- car/automobile
- water/H_2O

These mean the same so should not be treated as separate

- But probably not entirely identical
- E.g., if you use H_2O the setting is more likely science-related than, say, surfing-related

Sparse DTM

Since there are so many words one could use to describe very similar situations, a DTM can get cosine similarity "wrong" relative to what we would generally like:

$$D_A = \{\text{cold, water}\}$$
$$D_B = \{\text{icy, } H_2O\}$$

DTM

$$\begin{bmatrix} \text{cold} & \text{water} & \text{icy} & H_2O \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Cosine similarity

$$\frac{D_A' D_B}{\|D_A\| \, \|D_B\|} = 0$$

The DTM has too many zeros, misses lots of degrees of similarity, relatedness, and connotations

Word similarity

Many words are not exact synonyms but strongly related

Humans asked to rank similarity from 0 to 10:

vanish	disappear	9.80
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.30

This gives a measure of "word distance"

Word relatedness and connotation

Not similar, but definitely related

- coffee/cup
- surgeon/scalpel

Positive and negative connotations

- *wonderful* vs. *dreary*
- *fake, knockoff, forgery* vs. *copy, replica, reproduction*

Sentiment is a way to classify words based on connotations

Embeddings

Vector semantics

Osgood et al (1957): represents words in vector space. A more recent variant:

1. **valence**: the pleasantness of the stimulus
2. **arousal**: the intensity of emotion provided by the stimulus
3. **dominance**: the degree of control exerted by the stimulus

	Valence	Arousal	Dominance
courageous	8.0	5.5	7.4
music	7.7	5.6	6.5
heartbreak	2.5	5.7	3.6
cub	6.7	4.0	4.2

3-dimensional representation of all words!

- Revolutionary idea: embeddings

Vector semantics

"Close" words are close in the vector space:

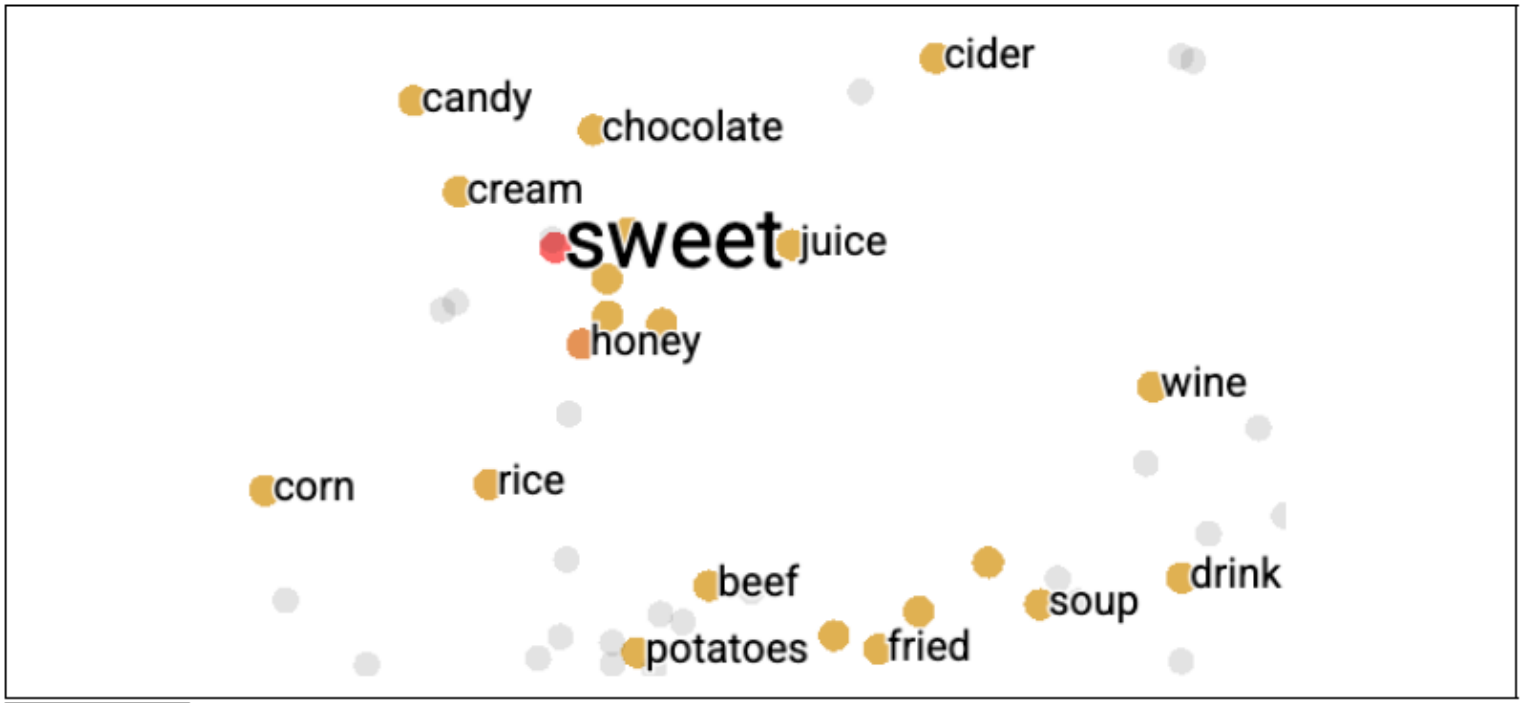


Figure 5.1 A two-dimensional (t-SNE) visualization of 200-dimensional word2vec embeddings for some words close to the word *sweet*, showing that words with similar meanings are nearby in space. Visualization created using the TensorBoard Embedding Projector https://projector.tensorflow.org/.

Sparse embeddings

the DTM is already a sparse embedding, with no reduction in dimensionality

A more sophisticated sparse embedding is the tf-idf embedding

To understand the general idea, we will consider a simpler setting:

- co-occurence matrix, word-context matrix
- simply, how often words appear close to each other
  - e.g., cherry $\leftrightarrow$ pie

Example: word-context, co-occurrence matrix

A subset of corpus, +/- 4 words from the word we are considering:

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

A row represents a word (showing only 3 context words for our 4 words)

	a	**computer**	**pie**
cherry	1	0	1
strawberry	0	0	2
digital	0	1	0
information	1	1	0

Figure 5.2 Co-occurrence vectors for four words with counts from the 4 windows above, showing just 3 of the potential context word dimensions. The vector for *cherry* is outlined in red. Note that a real vector would have vastly more dimensions and thus be even sparser.

- Thus, *cherry* is represented by $[1, 0, 1]$, *information* by $[1, 1, 0]$, etc.
- Vector representation implies that *cherry* is more similar to strawberry than to *digital* and *information*

(Dense) Embeddings

Relatively short vectors (e.g., 50-1000 elements)

- works much better in NLP tasks (some abstraction is good)

Word2vec: skip-gram with negative sampling (SGNS)

- *Static* embedding
- One vector for each word in vocabulary
- Does not have dynamic sense of *context*
  - E.g. *queen* rock band, *queen* of England

Word2vec

Choose window L, say $L = 2$

- How likely is it that a word will show up +/- two words next to word w

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2    w     c3       c4
```

- c_{1-4} are context words for w (apricot)

Train classifier: how likely is it that a candidate word c shows up next to w?

- Similarity $\approx \mathbf{c} \cdot \mathbf{w}$ (dot product)
- $\mathbf{c}$ is vector embedding for context word c and $\mathbf{w}$ is vector embedding for word w
- Find $\mathbf{c}$ and $\mathbf{w}$ for all words in vocabulary such that classifier has best fit to text used for training

Word2vec: Negative sampling

Redefines the problem from a multiclass classification task (predicting the correct context word from the entire vocabulary) to a series of binary classification tasks.

- For each training step:
  1. Select a positive pair: Take one target word (e.g., "fox") and one of its actual context words (e.g., "quick").
  2. Generate negative samples: Randomly select a small number of words from the vocabulary that are not in the context window. These are the "negative" samples (e.g., "table," "dog," "machine"). The number of negative samples, k, is a hyperparameter and is typically between 5 and 20 for smaller datasets.
  3. Adjust the objective: The model is no longer trained to predict a single context word. Instead:
     1. Maximize the probability that the positive pair is a "true" pair.
     2. Minimize the probability that the sampled negative pairs are "true" pairs.
  4. Update weights selectively: The weight updates are performed only for the positive word and the handful of negative words, leaving the rest of the vocabulary untouched. This dramatically reduces the computational load and speeds up training

Dense embedding properties

Cool feature: relational similarity

King - man + woman $\longrightarrow$ vector close to queen

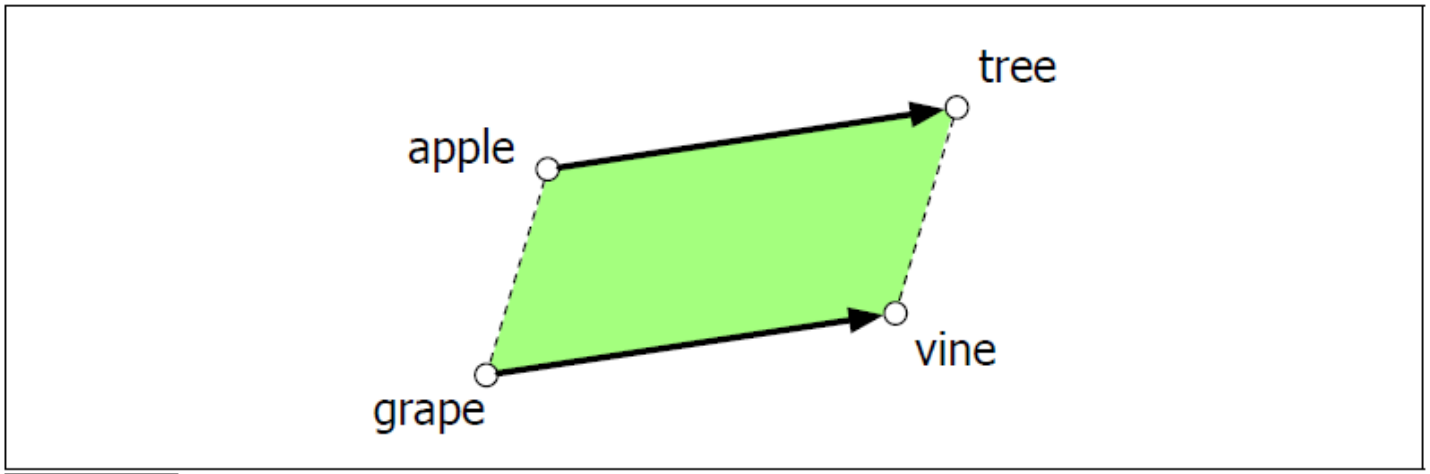


Figure 5.8 The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of $\overrightarrow{\text{vine}}$ can be found by subtracting $\overrightarrow{\text{apple}}$ from $\overrightarrow{\text{tree}}$ and adding $\overrightarrow{\text{grape}}$.

Embeddings as (minimal) Encoders

Note that the vector embeddings takes a high-dimensional vocabulary and creates a relatively low-dimensional representation

This is a minimal example of an *Encoder*

An Encoder-Decoder takes, for instance, text in one language and outputs text in another language

- The embeddings space could be the same and we just need the map from language A to embedding and language B to embedding to have a simple translator (we decode the embedding when outputting in language B)

Contextual Embeddings

BERT: Bidirectional Encoder Representations from Transformers

- More on transformers later
- Bidirectional refers to using language both before and after a token to optimally predict a masked word:

Instead of predicting next word in sentence, we mask a word in the sentence and ask model to predict it (probability distribution effectively over possible words) given its context

The water of Walden Pond is so beautifully ____

we're asked to predict a missing item given the rest of the sentence.

The ____ of Walden Pond is so beautifully ...

Contextual Embeddings: BERT

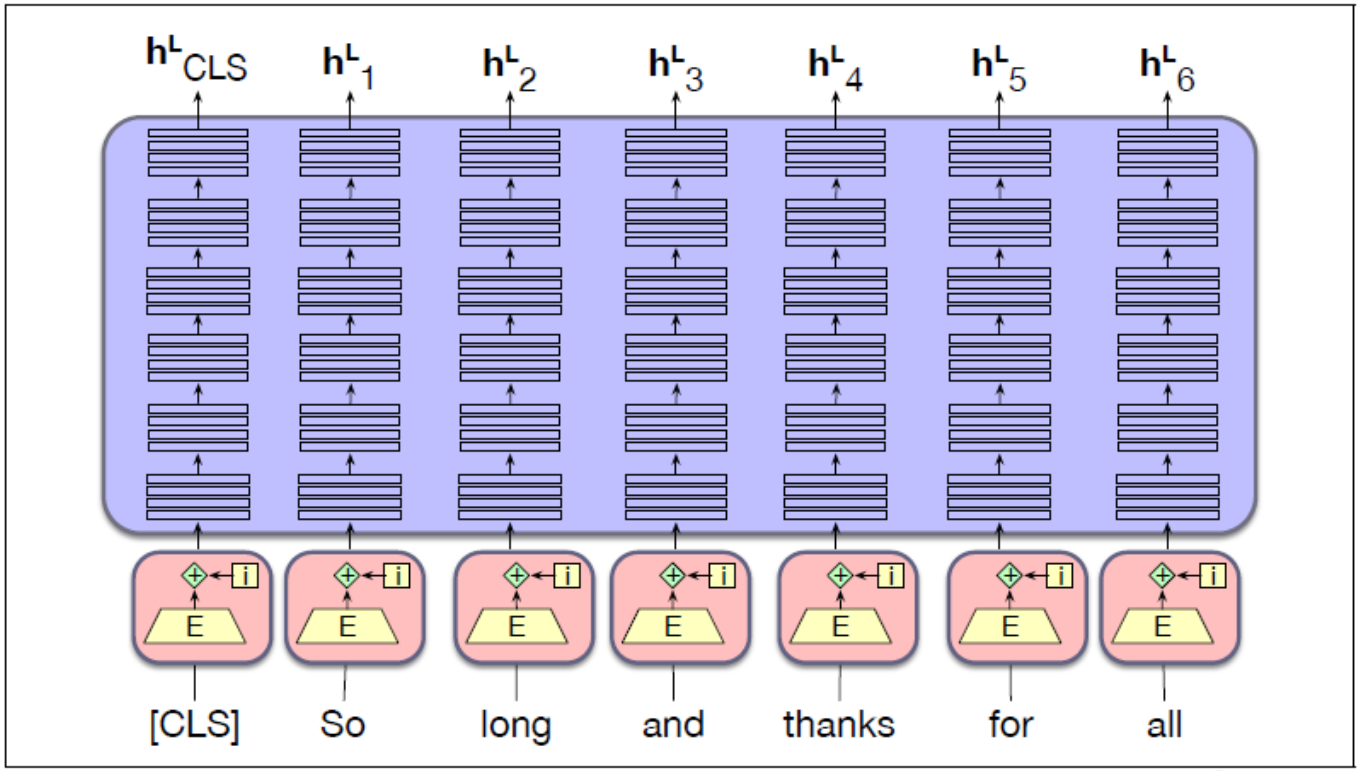


Figure 9.5 The output of a BERT-style model is a contextual embedding vector $\mathbf{h}_i^L$ for each input token x_i.

- The sentence contextualizes the token x_i
- The plot should have arrows between elements in each stack, as this is what BERT does and is critical for the contextualization
- E is the unconditional (without context) embedding matrix, and e_i is the element chosen by inputting the token x_i

Contextual Embeddings and Word Sense

Words have multiple meanings, the context disambiguates

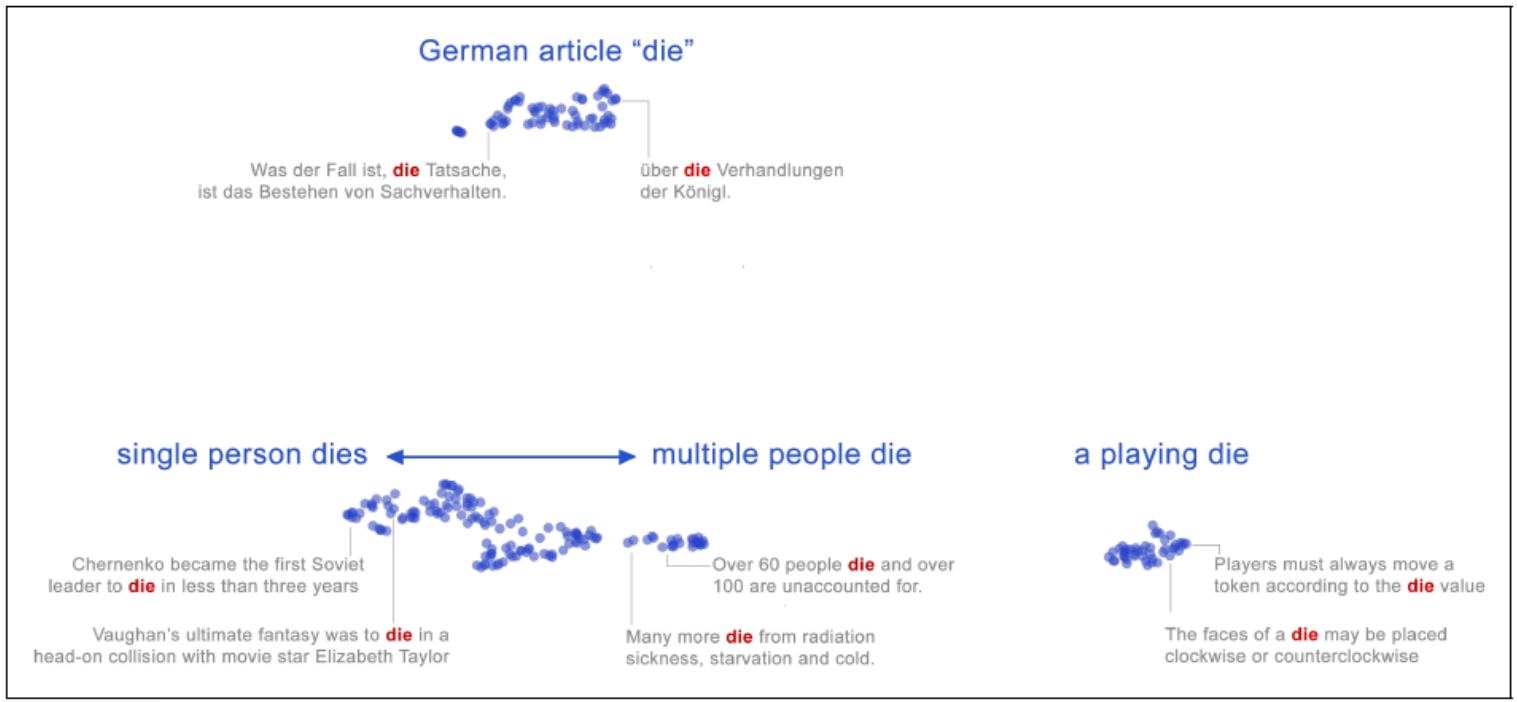


Figure 9.6 Each blue dot shows a BERT contextual embedding for the word *die* from different sentences in English and German, projected into two dimensions with the UMAP algorithm. The German and English meanings and the different English senses fall into different clusters. Some sample points are shown with the contextual sentence they came from. Figure from Coenen et al. (2019).

Embedding Models in Practice

EmbeddingGemma-300m

Recent, high-performing model

- Contextual embeddings
- 768-dimensional dense embeddings vector
- Context length: 2,048 tokens
- Trained on 100+ languages
- 308M parameters, derived from Gemma 3, on device use

It's purpose is not to generate text, like an LLM, but to create an accurate vector representation of text that can be used for search, indexing, classification, clustering, ++

- Download using SentenceTransformer (e.g., HuggingFace)

Examples of inputs and outputs

If you feed the model:

1. "*Hello world!*"
2. "*The zebras were grazing when the lions attacked.*"
3. "*First, we model the degree of present bias and agents' ability to forecast their future tastes as time-varying. Evidence from neuroscience and psychology shows that stress impairs executive functions (planning and self-control), amplifying present-biased behavior (Arnsten 2009; Sapolsky 2017). Consequently, shifts in economic or technological conditions, recessions, and traumatic events can exacerbate cognitive distortions, raising present bias.*"

The output is, in all three cases, a 768×1 numerical embedding vector

- This vector is a dense representation of the meaning of the context window you send the model
- 2,048 tokens (numerical representation of words, on average 3/4 words per token)
  - If you feed it more than 2,048 tokens, it only analyzes the first 2,048

Sentence-Transformer

EmbeddingGemma is not typically used in a standalone function call but is integrated with popular frameworks that wrap the core model logic. The specific functions you use will be part of the library you choose.

We will use HuggingFace's Sentence-Transformer library

- Provides the **encode()** function for use in Python

To do:

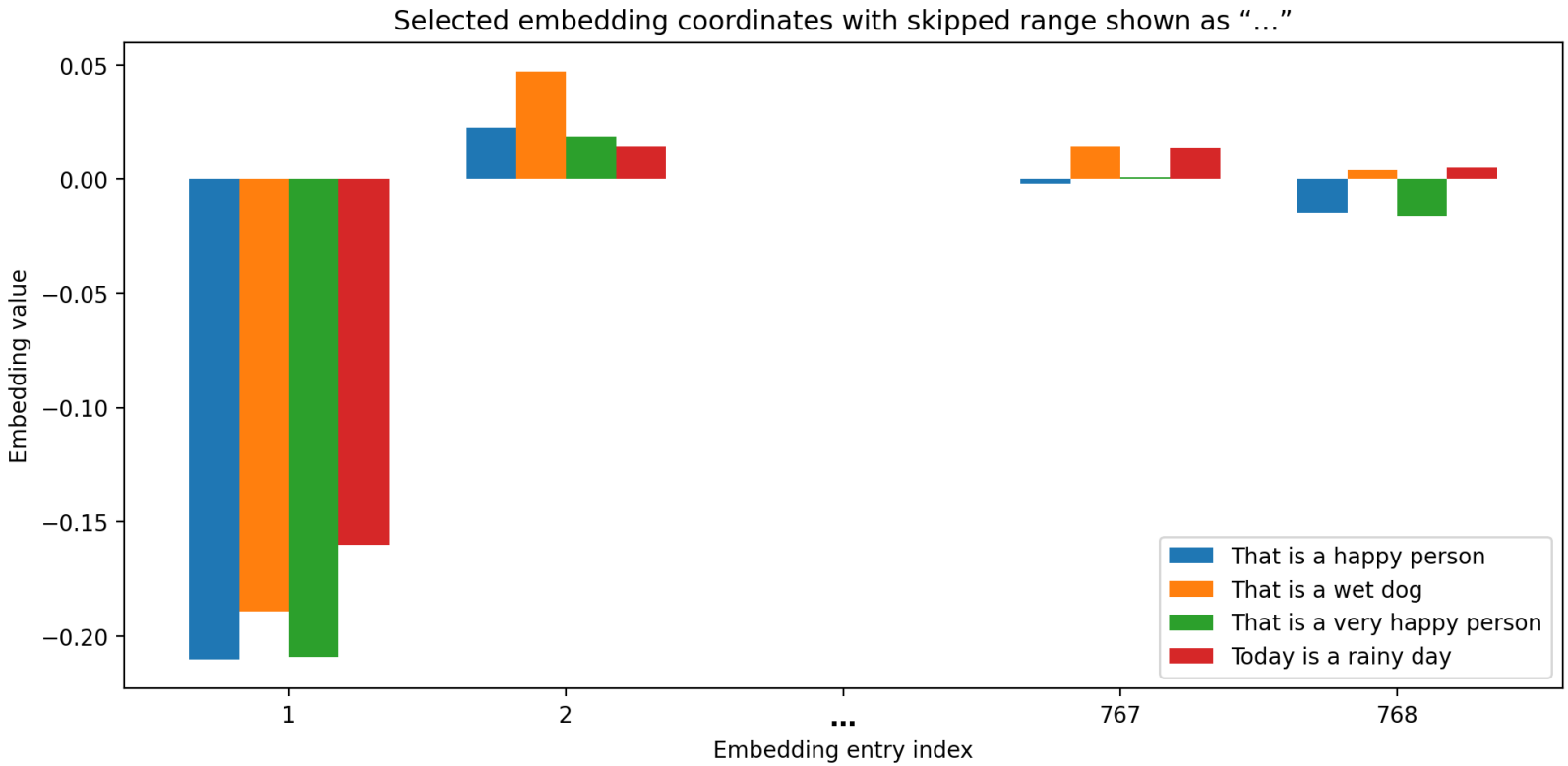
- Log in to HuggingFace (HF), search for EmbeddingGemma-300m and accept user agreement
- Generate your personal token for accessing the model through HF
  - Click on your profile icon, go to settings, click Access Tokens, Create new token

Simple start script

```
from huggingface_hub import login
your_token = 'YOUR HF TOKEN HERE '
login(token=your_token)
from sentence_transformers import SentenceTransformer
model = SentenceTransformer("google/embeddinggemma-300m")
sentences = [
    "That is a happy person",
    "That is a wet dog",
    "That is a very happy person",
    "Today is a rainy day"
]
embeddings = model.encode(sentences)
similarities = model.similarity(embeddings, embeddings)
print(similarities.shape)
# [4, 4]
```

Example of embedding values

From the embeddings on previous slide:



Adding prompts for different use cases

EmbeddingGemma has different embeddings depending on the use case

- Increase performance for that use

Available tasks:

query: "task: search result | query: "
document: "title: none | text: "
BitextMining: "task: search result | query: "
Clustering: "task: clustering | query: "
Classification: "task: classification | query: "
InstructionRetrieval: "task: code retrieval | query: "
MultilabelClassification: "task: classification | query: "
PairClassification: "task: sentence similarity | query: "
Reranking: "task: search result | query: "
Retrieval: "task: search result | query: "
Retrieval-query: "task: search result | query: "
Retrieval-document: "title: none | text: "
STS: "task: sentence similarity | query: "
Summarization: "task: summarization | query: "

Prompt example: search

Use *encode_query* function

```python
from sentence_transformers import SentenceTransformer

# Download from the 🤗 Hub
model = SentenceTransformer("google/embeddinggemma-300m")

# Run inference with queries and documents
query = "Which planet is known as the Red Planet?"
documents = [
    "Venus is often called Earth's twin because of its similar size and proximity."
    "Mars, known for its reddish appearance, is often referred to as the Red Planet
    "Jupiter, the largest planet in our solar system, has a prominent red spot.",
    "Saturn, famous for its rings, is sometimes mistaken for the Red Planet."
]
query_embeddings = model.encode_query(query)
document_embeddings = model.encode_document(documents)
print(query_embeddings.shape, document_embeddings.shape)
# (768,) (4, 768)

# Compute similarities to determine a ranking
similarities = model.similarity(query_embeddings, document_embeddings)
print(similarities)
# tensor([[0.3011, 0.6359, 0.4930, 0.4889]])
```

Prompt example: classification

Choose answer as label that is most similar to sentence

```
labels = ["Billing Issue", "Technical Support", "Sales Inquiry"]

sentence = [
  "Excuse me, the app freezes on the login screen. It won't work even when I try to reset my password.",
  "I would like to inquire about your enterprise plan pricing and features for a team of 50 people.",
]

# Calculate embeddings by calling model.encode()
label_embeddings = model.encode(labels, prompt_name="Classification")
embeddings = model.encode(sentence, prompt_name="Classification")

# Calculate the embedding similarities
similarities = model.similarity(embeddings, label_embeddings)
print(similarities)

idx = similarities.argmax(1)
print(idx)

for example in sentence:
  print("🙋", example, "-> 👹", labels[idx[sentence.index(example)]])
```

```
tensor([[0.4673, 0.5145, 0.3604],
        [0.4191, 0.5010, 0.5966]])
tensor([1, 2])
🙋 Excuse me, the app freezes on the login screen. It won't work even when I try to reset my password. -> 👹 Technical Su
🙋 I would like to inquire about your enterprise plan pricing and features for a team of 50 people. -> 👹 Sales Inquiry
```

Zero- and few-shot prompts

In the previous slide, we used a zero-shot prompt

- We only had the labels for classification, but no examples

Few-shot prompting

- Give explicit examples to help the classification task become more precise relative to what you want
- See next slide for example

Few-shot prompt example

```
# Label set
labels = ["Billing Issue", "Technical Support", "Sales Inquiry"]

# Few-shot supports per label
shots = {
    "Billing Issue": [
        "I was charged twice on my invoice.",
        "My bill amount is wrong this month."
    ],
    "Technical Support": [
        "The app crashes on login.",
        "Password reset keeps failing with an error."
    ],
    "Sales Inquiry": [
        "I need enterprise plan pricing.",
        "Do you offer discounts for a team of 50?"
    ],
}
```

Few-shot prompt example (cont'd)

```python
# Build label prototypes: mean of normalized embeddings for each label's shots
protos = []
for lab in labels:
    E = model.encode(shots[lab], prompt_name="classification", normalize_embeddings=True)  # (n_sh
    proto = E.mean(axis=0)
    proto /= np.linalg.norm(proto) + 1e-12
    protos.append(proto)
prototypes = np.vstack(protos)  # (n_labels, D)

# Examples to classify
sentences = [
    "Excuse me, the app freezes on the login screen. It won't work even when I try to reset my pas
    "I would like to inquire about your enterprise plan pricing and features for a team of 50 peop
]

# Encode with the same prompt; normalize for cosine
X = model.encode(sentences, prompt_name="classification", normalize_embeddings=True)  # (n_sent, D

# Cosine similarity to prototypes (SentenceTransformers helper)
S = model.similarity(X, prototypes)  # (n_sent, n_labels)
print(S)

# Argmax decision
idx = S.argmax(1)
print(idx)
```

Prompt: Clustering

Recall LDA from DAML

- Creating “topics” from text documents

Clustering prompt of EmbeddingGemma is optimized for a similar purpose

- Get document embeddings using prompt = "clustering"
- Use something like K-means clustering to define clusters
  - Articles in a cluster are related
  - Look at top articles (the ones that fit the cluster the best) to give cluster a name

Prompt: Clustering (example)

Example Corpus

- Documents (docs) are made very short just as an example

```
# ---------- Corpus ----------
docs = [
    # Revenue / demand
    "Revenue grew 12% year over year, driven by demand in North America.",
    "Topline beat guidance on strong holiday-season sell-through.",
    "We saw double-digit growth in APAC and solid wallet share gains.",
    # Guidance / outlook
    "We are lowering full-year guidance due to a slower enterprise pipeline.",
    "Q4 outlook implies flat unit volumes with pricing pressure.",
    "Management raised FY EPS guidance on operating leverage.",
    # Costs / margins
    "Gross margin expanded 80 bps from mix and cost controls.",
    "Freight costs normalized, while components remain a tailwind.",
    "We expect opex discipline to sustain margin improvement.",
    # Capital allocation / capex
    "We authorized a new $5B buyback and increased the dividend.",
    "Capex will step up for data center capacity and AI infrastructure.",
    "We plan to delever to 2.0x net debt over the next 12 months.",
    # Macro / FX / risk
    "FX headwinds reduced revenue by 200 bps in EMEA.",
    "Consumer demand remains uneven given macro uncertainty.",
    "Export restrictions create risk to our China business.",
    # Product / customers
    "New product adoption accelerated among Fortune 500 customers.",
    "Churn improved as customers consolidated onto our platform.",
    "We launched a low-latency tier for real-time analytics.",
```

Prompt: Clustering (example cont'd)

```python
# ---------- Embed with task=clustering ----------
model = SentenceTransformer("google/embeddinggemma-300m")
E = model.encode(docs, prompt_name="clustering", normalize_embeddings=True)  # (N, D)

# ---------- KMeans ----------
K = 5
labels = KMeans(n_clusters=K, n_init=10, random_state=42).fit_predict(E)
sil = silhouette_score(E, labels, metric="euclidean")
print(f"Silhouette: {sil:.3f}")

# ---------- Top-3 representatives per cluster ----------
for c in range(K):
    idx = np.where(labels == c)[0]
    if idx.size == 0:
        continue
    centroid = E[idx].mean(axis=0)
    centroid /= np.linalg.norm(centroid) + 1e-12
    sims = E[idx] @ centroid
    top = idx[np.argsort(-sims)[:min(3, idx.size)]]
    print(f"\nCluster {c} (n={idx.size})")
    for i in top:
```

Prompt: Summarization

A useful task is to summarize a set of documents

- Quicker and to the point reading of information
  - particularly useful if large corpus and long documents
- Makes files smaller for post-summary analysis
- Can be weighted to some use case
  - E.g., summarize the content that relates to the competitive situation of the company

EmbeddingGemma does **not** summarize

- Embeddings are for retrieval/selection
- But, embeddings using summarize prompt optimized for achieving summarization in post-embedding analysis
- Use LLM, like Gemma 3 etc, to go from embeddings to summary of texts

Prompt: RAG

RAG = Retrieval-Augmented Generation

1. You embed a user query
2. Retrieve the top-k relevant chunks from an indexed corpus of your choosing that helps put context to the query
3. Re-rank: second pass to find maybe top-3 from the top-k (e.g., k = 50) chunks
4. "Stuff" the best snippets into the prompt of an LLM so the generator writes an answer grounded in those sources
5. The core loop is: index documents $\longrightarrow$ kNN retrieve $\longrightarrow$ rerank $\longrightarrow$ construct a constrained prompt with citations and instructions $\longrightarrow$ generate $\longrightarrow$ (optionally) verify to fine tune the RAG

This cuts hallucinations, lets you use fresh or private data without fine-tuning, and keeps latency under control by tuning k and the context budget.

Homework 5

High-level embeddings models and incredibly powerful and useful!!!

In HW 5, you will be working with the EmbeddingGemma-300m embedding model

Sentiment analysis of earnings call transcripts