📝 Topic Modeling with BERTopic

Session: Text Embeddings

Michael Jantscher · TU Graz · Know Center Research GmbH

👋 Michael Jantscher

PhD Student - TU Graz
Researcher - Know Center Research GmbH

🧠 Focus Areas

- Natural Language Processing (NLP) in medical & clinical domains
- Causal reasoning in healthcare and (neuro)radiology
- Agentic AI systems for decision support and research workflows

📌 What is a Text Embedding Vector ?- Formal Definition

- Numerical representation of text (words, sentences or documents) in a multi-dimensional space
- Captures meaning and context
- Semantic similar words/sentences/documents -> vectors closer together

In [15]:

```
import pandas as pd
from sentence_transformers import SentenceTransformer
st_model_small = SentenceTransformer('all-minilm-l6-v2')

sample_string = "I really like this summer school!"

sample_string_embedding = st_model_small.encode(sample_string)
df = pd.DataFrame({
    "text": [sample_string],
    "embedding": [sample_string_embedding]
})
df
```

Out[15]:

	text	embedding
0	I really like this summer school!	[-0.057146158, -0.053543467, 0.045457065, 0.01...

💡 Uses Cases of Text Embedding Vectors

Use Case	Example
Semantic Search	Search "doctor" → find content on "physicians" or "healthcare providers" even without exact keywords
Recommendation Systems	Suggest similar research papers, movies, or products based on descriptions or reviews
Sentiment Analysis	Understand tone (positive/negative/neutral) beyond simple keywords in tweets or reviews
Clustering & Topic Modeling	Group thousands of news articles or support tickets by topic automatically
Chatbots & Virtual Assistants	Improve NLU so bots answer contextually, not just by keyword
Fraud Detection	Spot unusual or suspicious text patterns in financial or insurance claims

🦖 Pre-Embedding Area

📌 **Definition:** Early NLP approaches represented text as **sparse, high-dimensional vectors**. Each dimension corresponded to a **unique word or token**, with no sense of meaning or context.

Bag-of-Words (BoW)

- Represents documents as a **vector of word counts**
- Ignores grammar, order, and semantics
- Example: `"I like NLP"` → `[1, 1, 1, 0, 0, ...]`

TF-IDF (Term Frequency – Inverse Document Frequency)

- Adjusts raw counts to emphasize **rare, informative words** and downweight common words
- Example: "the" → low weight, "quantum" → high weight

TF-IDF Formula

$$TF\text{-}IDF(t,d) = TF(t,d) \times \log\left(\frac{N}{DF(t)}\right)$$

Where:

- (TF(t, d)): Frequency of term (t) in document (d)
- (DF(t)): Number of documents containing (t)
- (N): Total number of documents

Dense Text Embeddings

📌 **Definition:** Dense embeddings represent words, sentences, or documents as **low-dimensional, dense vectors** where similar meanings are **close together in vector space**.

🟢 Characteristics

- **Low-dimensional** (e.g., 100–1,536 dimensions, not vocab-sized)
- **Dense representation**: most values ≠ 0
- Captures **semantic meaning** and context
- Learned from data via **neural networks**

🟠 Brief History

- **Word2Vec (2013)** – First widely used dense word embeddings (Mikolov et al.)
- **GloVe (2014)** – Global Vectors for word representation
- **FastText (2016)** – Adds subword information for better handling of rare words
- **ELMo (2018)** – Contextual word embeddings
- **BERT (2018)** – Contextual embeddings for entire sentences
- **OpenAI / Modern Embeddings (2020s)** – High-quality sentence/document embeddings (e.g., `text-embedding-3-large`)

🔴 Why It's Better

- Reduces dimensionality dramatically
- Learns **semantic relationships**
- Powers modern **search, recommendation, and AI assistants**

Word2Vec: CBOW & Skip-Gram Overview

🟢 What is Word2Vec?

- A **shallow, two-layer neural network** that learns word embeddings from context.
- Maps words to **dense vectors** in a continuous space; similar words are **close together**.
- Introduced by Mikolov et al., **2013**.

🟠 Architectures

🔹 Continuous Bag-of-Words (CBOW)

- Predicts the **center word** given its context words.
- Fast to train; works well for **frequent words**.

🔹 Skip-Gram

- Predicts **context words** given a center word.
- Performs better for **rare words**, large datasets.

🔵 Why It Matters

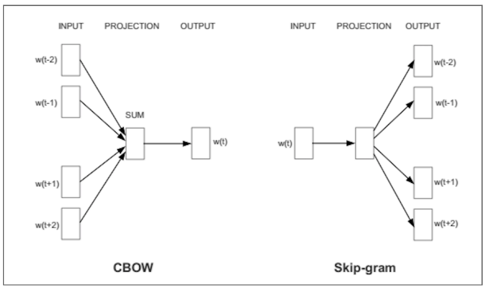
- Captures **semantic relationships**: `king - man + woman ≈ queen`
- Major leap from sparse (BoW/TF-IDF) to **dense, meaningful embeddings**.

Word2Vec: Training & Visual Intuition

🟢 Training Workflow

1. **One-hot encoding** for words.
2. **Hidden layer** = embedding lookup table.
3. **Output layer** predicts context words (softmax with negative sampling).
4. Final **hidden layer weights** = embeddings.

🟠 Visual Intuition



- Diagram shows the Skip-Gram model predicting context words.
- Only the **embedding layer weights** are retained.

Reference

- Israel G. (2017). *Word2Vec Explained.* **https://israelg99.github.io/2017-03-23-Word2Vec-Explained/**

BERT: Contextual Embeddings

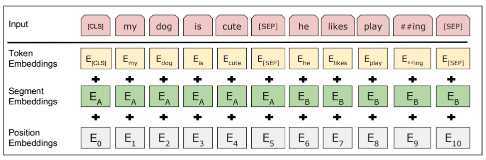
🟢 What is BERT?

- **Bidirectional Encoder Representations from Transformers** (2018, Google AI).
- Uses **Transformer architecture** to create **contextual word embeddings**:
    - Each word's vector depends on **all surrounding words** (left & right context).
- Trained on **masked language modeling** and **next sentence prediction** tasks.

🟠 Key Innovations

- **Bidirectional**: Unlike Word2Vec/Glove, captures context from both sides.
- **Transformer encoder layers** with self-attention results in rich, deep embeddings.
- **Contextualization**: Same word gets **different vectors** depending on context ("bank" in "river bank" vs. "bank account").

🔵 Visual Intuition



Reference

- Devlin et al. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* **https://arxiv.org/abs/1810.04805**

Sentence Transformers Recap & Training

🟢 Key Takeaways

- **Sparse vectors** (BoW, TF-IDF):
  - One dimension per word, mostly zeros
  - No deep semantic meaning
- **Dense embeddings** (Word2Vec, BERT, SBERT):
  - Low-dimensional, rich semantic context
  - Words/sentences cluster by meaning

🟠 How SBERT is Trained

- **Backbone:** Pretrained BERT or RoBERTa encoders
- **Siamese/Triplet Network Architecture:**
  - Encodes two or three sentences **independently** into embeddings
  - Trains to minimize distance for similar sentences and maximize for dissimilar ones
- **Training Objectives Loss Overview:**
  - **Contrastive loss** (distance-based similarity)
  - **Natural Language Inference** datasets (entailment, contradiction, neutral)
  - **MultipleNegativesRankingLoss** for retrieval tasks
- **Result:**
  - Embeddings suitable for **cosine similarity** → semantic search, clustering, recommendations

---

Reference

- Reimers & Gurevych (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* **https://arxiv.org/abs/1908.10084**
- SentenceTransformers Documentation **https://www.sbert.net/**

Distance & Similarity Measures (Brief Intro)

🟢 Why It Matters

- Compare embeddings → find **semantic similarity** between words, sentences, or documents.
- Core to **semantic search, clustering, and topic modeling**.

🟠 Common Measures

- 🔹 **Dot Product** Unnormalized similarity; sensitive to magnitude:

$$a \cdot b = \sum a_i b_i$$

- 🔹 **Cosine Similarity** Measures **angle** between vectors (ignores magnitude):

$$\text{cosine\_sim}(a, b) = \frac{a \cdot b}{||a|| \, ||b||}$$

- 🔹 **Euclidean Distance** Straight-line distance in vector space:

$$d(a, b) = \sqrt{\sum (a_i - b_i)^2}$$

- 🔹 **Manhattan (L1) Distance** Sum of absolute differences:

$$d_{\text{L1}}(a, b) = \sum |a_i - b_i|$$

Chunking Techniques for Embeddings

🟢 Why Chunking?

- Long documents exceed model token limits (e.g., BERT ~512 tokens).
- Splitting text into **manageable chunks** improves:
    - ✅ Embedding quality
    - ✅ Retrieval accuracy
    - ✅ Context management in downstream tasks
- 🎯 **Goal of Good Chunking:** Create chunks that are **small enough** to fit model limits but **large enough** to retain full semantic meaning.

🟠 Common Techniques

1. **Fixed-Length Chunking**
    - Split text into chunks of `N` tokens/words.
    - Simple, fast, but can cut off sentences mid-way.
2. **Sentence-Based Chunking**
    - Split by sentence boundaries (NLTK, spaCy).
    - Better for readability, semantic grouping.
3. **Paragraph-Based Chunking**
    - Keep natural paragraph structure.
    - Good for preserving context, but chunk sizes vary.
4. **Sliding Window / Overlapping Chunks**
    - Add overlap between chunks (e.g., 50 tokens).
    - Prevents loss of context between splits.
5. **Semantic Chunking**
    - Use topic segmentation or embeddings to find boundaries.
    - Most accurate, but computationally heavier.

💡 **Choose technique based on document length, model limits, and retrieval needs.**

Adding It All Together

🟢 The Journey So Far

- **Sparse Vectors (BoW, TF-IDF):** High-dimensional, simple counts, no semantics
- **Dense Word Embeddings (Word2Vec, GloVe):** Compact vectors capturing basic word meaning
- **Contextual Models (BERT):** Token embeddings adapt to context
- **Sentence Transformers (SBERT):** Sentence-level semantic embeddings for similarity & search
- **Chunking Strategies:** Break long texts into meaningful, model-friendly pieces

---

🟠 Why It Matters

- Transform raw text into **meaningful numerical representations**
- Enable **semantic search, clustering, Q&A, recommendations**
- Foundation for **modern NLP pipelines & AI assistants**

🔵 Key Takeaway

A well-designed embedding pipeline = **Chunking + Contextual Models + Smart Similarity Metrics** → Powerful, scalable text understanding!

🏋️ Exercises: Hands-On with Embeddings

🟠 Generate embeddings for the Parlamint (sub) dataset

- **Experiment with different embedding techniques:**
    - Compare different embedding techniques (dense vs sparse) regarding (i) vector dimensionality (ii) Semantic similarity (are similar texts actually closer together?)
    - Suggested algorithms: **BERTopic Models** or **HuggingFace (general)** or **Huggingface (sentence transformers)**
- **Consider different chunking techniques:**
    - Sentence based vs utterance level vs ??
- **Save them as pickle file(s):** `df_dataset.to_pickle("<path_and_filename>.pkl")`

🟠 Generate embeddings for the HSA (sub) dataset

- **Same tasks as for the Parlamint dataset**

🟠 Retrieval: Play around with embeddings and similarity retrieval

- **Write queries:**
    - Search for valid topics, write queries and manually evaluate the result
- **Consider different chunking techniques:**
    - Sentence based vs utterance level
    - Are topics semantically better captured on sentence level or utterance level=
- **Utilize different embedding models for retrieval:**
    - Sparse vs dense embeddings
    - Experiment with different similarity scores