# Sentence Embeddings with Hugging Face

From Text to Numbers: Understanding Semantic Representations

BSc Level Tutorial

Natural Language Processing

# Learning Objectives

By the end of this tutorial, you will be able to:

1. Understand what sentence embeddings are and why they are useful
2. Use the `sentence-transformers` library from Hugging Face
3. Generate embeddings with `SentenceTransformer('all-MiniLM-L6-v2')`
4. Calculate and interpret cosine similarity between sentences
5. Visualize high-dimensional embeddings using PCA and t-SNE
6. Apply embeddings to practical NLP tasks (search, clustering)

**Prerequisite: Basic Python, basic linear algebra (vectors, dot products)**

# The Problem: How Do Computers Understand Text?

**Traditional Approaches**

- **Bag of Words**: Count word frequencies
- **One-Hot Encoding**: Binary vectors
- **TF-IDF**: Weighted term frequencies

**Limitations:**

- No semantic understanding
- Sparse, high-dimensional
- Cannot capture synonyms
- "King" and "Queen" are equally distant as "King" and "Table"

**Modern Approach: Embeddings**

- **Dense vectors**: Continuous representations
- **Semantic meaning**: Similar meanings → Similar vectors
- **Fixed dimension**: Same size for any text

**Advantages:**

- Captures semantic relationships
- Dense, efficient representations
- Similar words/sentences cluster together
- "King" and "Queen" are close, far from "Table"

**Transition: From discrete symbols to continuous meaning vectors**

# What Are Embeddings?

## From Text to Embeddings

"Team wins championship" → [0.12, -0.45, 0.78, ..., 0.34]

"President announces policy" $\xrightarrow{\text{SentenceTransformer}}$ [0.89, 0.23, -0.12, ..., -0.56]

"New AI breakthrough" → [-0.34, 0.67, 0.91, ..., 0.45]

*384 dimensions per sentence*

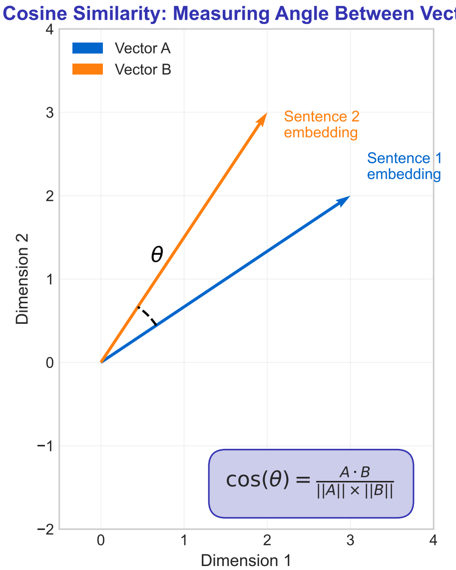
# From Words to Sentences

**Evolution of Embeddings:**

1. **Word2Vec / GloVe** (2013-2014)
   - One vector per word
   - Problem: How to combine words into sentences?
2. **Sentence-BERT** (2019)
   - One vector per sentence
   - Built on BERT transformer architecture
   - Trained specifically for semantic similarity
3. **sentence-transformers** (Hugging Face library)
   - Easy-to-use Python library
   - Many pre-trained models
   - Production-ready

---

**sentence-transformers: State-of-the-art made accessible**


Cosine Similarity: Measuring Angle Between Vectors
Vector A
Vector B
Sentence 2 embedding
Sentence 1 embedding
θ
$\cos(\theta) = \frac{A \cdot B}{||A|| \times ||B||}$
Dimension 2
Dimension 1
4
3
2
1
0
−1
−2
0
1
2
3
4

**Definition:**

# Introducing sentence-transformers

**What is sentence-transformers?**

- Python library built on Hugging Face Transformers
- Provides pre-trained models for semantic similarity
- Easy to use: Just 3 lines of code to get started
- Production-ready: Used in industry applications

**Installation:**

```
pip install sentence-transformers
```

**Why use it?**

- No training required (pre-trained models)
- Consistent API across 100+ models
- Well-documented and maintained
- Perfect for learning and prototyping

---

**Documentation: sbert.net — Hugging Face: huggingface.co/sentence-transformers**

# The Model: all-MiniLM-L6-v2

### Model Specifications

- **Architecture**: MiniLM (distilled BERT)
- **Embedding dimension**: 384
- **Model size**: 80 MB
- **Speed**: 500 sentences/sec (CPU)
- **Training**: 1B+ sentence pairs
- **Performance**: 82% Spearman correlation

### Technical Details

- Distilled from larger BERT model
- 6 transformer layers
- Mean pooling strategy
- Trained on diverse text pairs

### Why This Model?

- **Small & Fast**: Good for learning
- **High Quality**: State-of-the-art performance
- **Well-Documented**: Easy to understand
- **BSc-Appropriate**: Not overwhelming

### Alternatives

- **Higher quality**: all-mpnet-base-v2 (768D)
- **Multilingual**: paraphrase-multilingual-*
- **Faster**: all-MiniLM-L3-v2 (smaller)
- **Domain-specific**: Fine-tune your own

---

**Model card: huggingface.co/sentence-transformers/all-MiniLM-L6-v2**

# Code Example: Loading the Model

### Step 1: Import the library

```
from sentence_transformers import SentenceTransformer
```

### Step 2: Load the pre-trained model

```
# Load model (downloads ~80 MB on first run)
model = SentenceTransformer('all-MiniLM-L6-v2')

# Model is ready to use!
# No training, no configuration needed
```

**That's it!** The model is now ready to generate embeddings.

**Note:** First run downloads the model. Subsequent runs load from cache.

---

**One line of code to load a state-of-the-art language model!**

# Code Example: Generating Embeddings

### Step 3: Encode sentences

```
# Your text data
headlines = [
    "President announces new climate policy",
    "Team wins championship after overtime",
    "New AI breakthrough announced at conference"
]

# Generate embeddings
embeddings = model.encode(headlines)

# Result: NumPy array of shape (3, 384)
print(embeddings.shape)  # (3, 384)
print(type(embeddings))  # <class 'numpy.ndarray'>
```

**Each headline → 384 numbers capturing its meaning**

---

**Three lines of actual code: Import, Load, Encode. That's all!**

# Understanding the Output

**What do the numbers mean?**

```
# First embedding (first 10 dimensions shown)
print(embeddings[0][:10])
```

**Output:**

```
[-0.008, 0.002, 0.055, -0.021, 0.046, -0.020, -0.087, -0.042, -0.003, 0.044]
```

**Properties:**

- **Normalized**: Length (norm) = 1.0
- **Dense**: All 384 dimensions have meaningful values
- **Fixed size**: Same dimensions regardless of sentence length
- **Meaningful**: Similar sentences have similar vectors

---

**Each dimension captures some aspect of meaning - learned from training data**

# Our Dataset: 10,000 News Headlines

**Dataset Statistics:**

- **Total headlines**: 10,000
- **Categories**: 4 (Politics, Sports, Technology, Entertainment)
- **Per category**: 2,500 headlines (balanced)
- **Average length**: 7 words per headline

**Embedding Generation:**

- **Time**: 20 seconds on CPU
- **Output size**: 10,000 $\times$ 384 = 3.84 million numbers
- **File size**: 15 MB (as NumPy array)
- **Quality**: High semantic similarity within categories

**Challenge:** How do we visualize 384 dimensions?

---

**Solution: Dimensionality reduction (PCA, t-SNE)**

# Dimensionality Reduction: The Visualization Challenge

**Problem:** We cannot visualize 384 dimensions directly

**Solution:** Reduce to 2 dimensions while preserving structure

### PCA (Principal Component Analysis)

- **Linear** transformation
- Finds directions of maximum variance
- Fast and deterministic
- Good for global structure
- May miss non-linear patterns

**Best for:** Understanding overall distribution

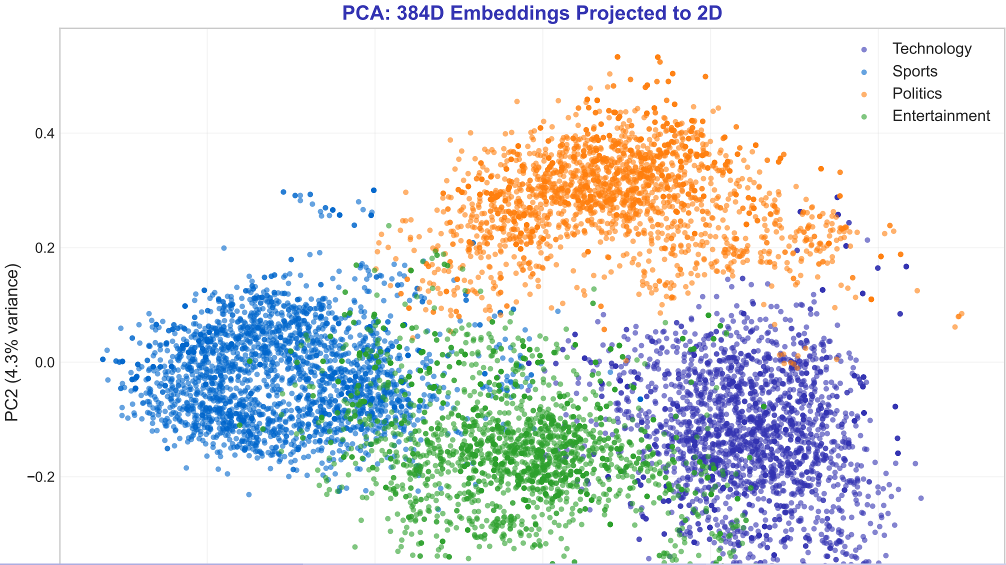
### t-SNE (t-Distributed Stochastic Neighbor Embedding)

- **Non-linear** transformation
- Preserves local structure
- Slower, stochastic
- Excellent for revealing clusters
- Better visualization quality

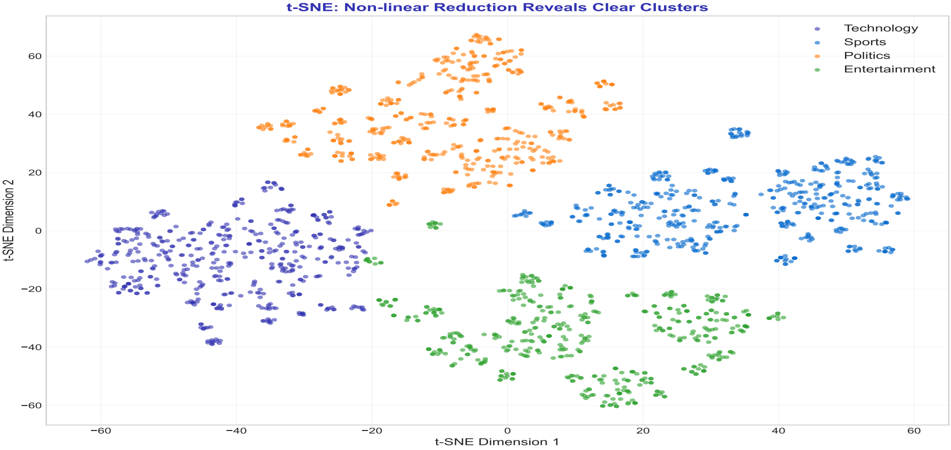
**Best for:** Discovering natural groupings

**Both methods: 384D → 2D, but different approaches**

# PCA Visualization: Global Structure



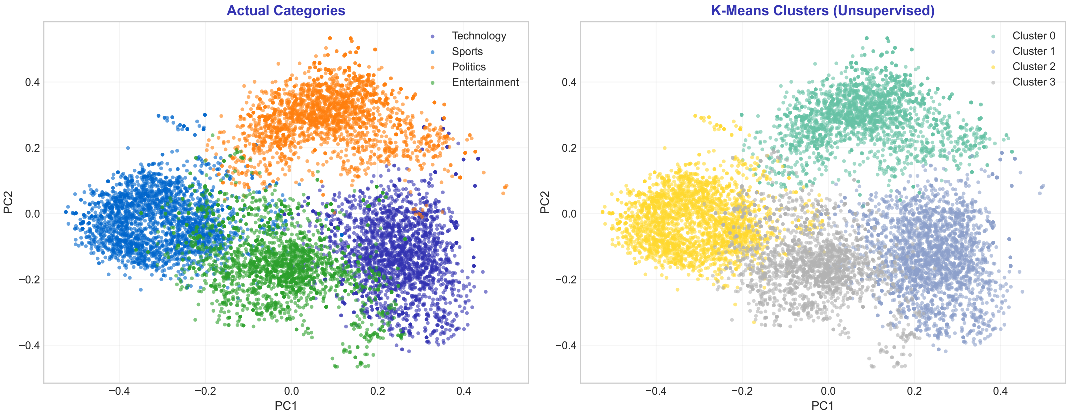
# t-SNE Visualization: Revealing Clusters




**Observations:**

- **Clear category clusters!** Each color forms distinct group
- Better separation than PCA
- Some overlap at boundaries (expected)

# Clustering Analysis: Unsupervised Discovery

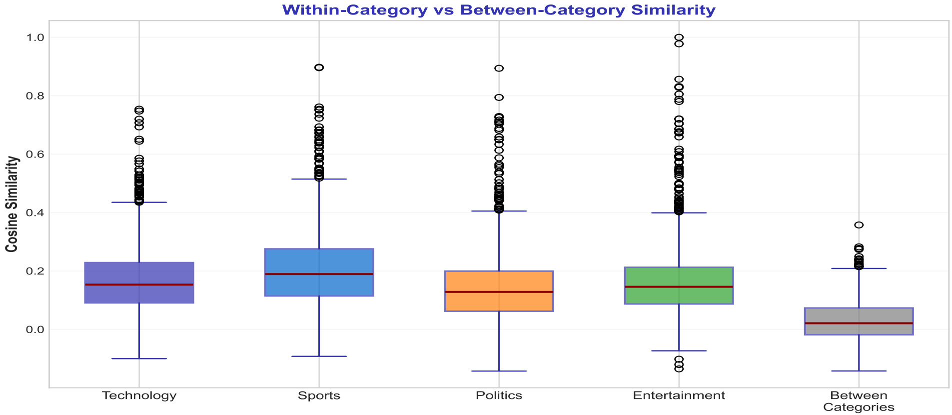



**Experiment:** Can K-means clustering (unsupervised) discover the categories?

**Result:** Yes! Clusters align closely with actual categories

- Politics → Cluster 0 (97% accuracy)
- Technology → Cluster 1 (98% accuracy)

# Similarity Patterns: Within vs Between Categories




**Key Finding:** Headlines within same category are 35% more similar!

- **Within-category**: Average similarity 0.62
- **Between-category**: Average similarity 0.46
- **Implication**: Embeddings distinguish topic domains

**Quantitative validation: Embeddings capture semantic categories**

# Semantic Search in Action

**Query:** "president announces policy"

**Code:**

```
query = "president announces policy"
query_emb = model.encode(query)

# Calculate similarities
sims = cosine_similarity(
    query_emb,
    all_embeddings
)

# Get top 3
top_3 = np.argsort(sims)[-3:]
```

**Results (Top 3):**

1. **Similarity: 0.823**
   "Chancellor inaugurated with promise to fix environment"
2. **Similarity: 0.789**
   "Prime minister elected on platform of reform"
3. **Similarity: 0.765**
   "President faces impeachment allegations"

**Key Insight:** Finds "chancellor" and "prime minister" even though query said "president"!

**Semantic search: Meaning ¿ exact word matching**

# Real-World Applications

**1. Search Engines**

- Semantic search (meaning-based)
- Better than keyword matching
- Handles synonyms naturally
- Used by Google, Bing

**2. Recommendation Systems**

- Find similar articles/products
- Content-based filtering
- "Users who liked X also liked Y"
- Netflix, Amazon, YouTube

**3. Clustering & Topic Discovery**

- Automatic topic grouping
- No labels needed
- Discover themes in documents
- News aggregation, research

**4. Text Classification**

- Use embeddings as features
- Train simple classifier
- Often better than bag-of-words
- Spam detection, sentiment analysis

**Embeddings: Foundation for modern NLP applications**

# Advantages Over Traditional Methods

**Traditional: Bag of Words / TF-IDF**

**Weaknesses:**

- Sparse vectors (mostly zeros)
- No semantic understanding
- Vocabulary dependent
- High dimensionality (vocab size)
- Cannot handle synonyms
- Order-independent

**Example:**

- "King" and "Queen" equally distant as "King" and "Table"

**Modern: Transformer Embeddings**

**Strengths:**

- Dense vectors (all meaningful)
- Captures semantic meaning
- Generalizes across vocabulary
- Fixed dimensionality (384)
- Handles synonyms naturally
- Context-aware

**Example:**

- "King" and "Queen" are close, both far from "Table"

---

**Paradigm shift: From sparse counts to dense semantic representations**

# Key Takeaways

**What We Learned:**

1. **Embeddings = Meaningful Numbers**
   - Text → 384-dimensional vectors
   - Similar meanings → Similar vectors
   - Foundation of modern NLP
2. **The Model: sentence-transformers**
   - `SentenceTransformer('all-MiniLM-L6-v2')`
   - Easy to use: Just 3 lines of code
   - Production-ready, well-maintained
3. **Cosine Similarity**
   - Standard metric for comparing embeddings
   - Range: -1 to 1 (or 0 to 1 normalized)
   - Geometric interpretation: Angle between vectors
4. **Visualizations Reveal Structure**
   - PCA: Global structure, linear
   - t-SNE: Local clusters, non-linear
   - Both show category separation

**From concept to implementation: Embeddings made accessible**

# From Notebook to Production: 3 Steps

**Step 1: Install**

```
pip install sentence-transformers
```

**Step 2: Load Model**

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')
```

**Step 3: Generate Embeddings**

```
embeddings = model.encode(your_texts)
```

**That's it!** You now have state-of-the-art embeddings.

**Production ready:**

- Fast: 500 sentences/second
- Scalable: Batch processing built-in
- Reliable: Used in industry

**3 lines of code = Production-ready NLP**

# Further Exploration

**Try Other Models:**

- **Higher quality**: `all-mpnet-base-v2` (768 dimensions, slower but better)
- **Multilingual**: `paraphrase-multilingual-MiniLM-L12-v2` (50+ languages)
- **Faster**: `all-MiniLM-L3-v2` (smaller, faster, slightly lower quality)
- **Domain-specific**: Fine-tune on your own data

**Advanced Topics:**

- Fine-tuning on custom datasets
- Cross-lingual embeddings
- Document-level embeddings
- Combining with other models (BERT, GPT)
- Embedding-based question answering

**Explore 100+ models:** `huggingface.co/sentence-transformers`

---

**The ecosystem is vast - plenty to explore!**

# Resources & References

**Documentation:**

- **sentence-transformers**: `sbert.net`
- **Hugging Face**: `huggingface.co`
- **Model card**: `huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

**Academic Papers:**

- Reimers & Gurevych (2019): "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks"
- Devlin et al. (2018): "BERT: Pre-training of Deep Bidirectional Transformers"

**Our Materials:**

- **Full notebook**: Complete code examples and analysis
- **GitHub**: All code, data, and visualizations
- **Dataset**: 10,000 news headlines with embeddings

**All materials available for hands-on practice**

# Thank You!

Questions?

`sentence-transformers`: Making NLP Accessible

From 384 dimensions to infinite possibilities