Most of the material has been taken from Jurafsky & J. H. Martin: **Speech and Language Processing**, Pearson International 3ed draft https://web.stanford.edu/~jurafsky/slp3/

Jay Alammar, Maarten Grootendorst: Hands-On Large Language Models Published by O'Reilly Media, Inc.

# Text Analysis -Transformers

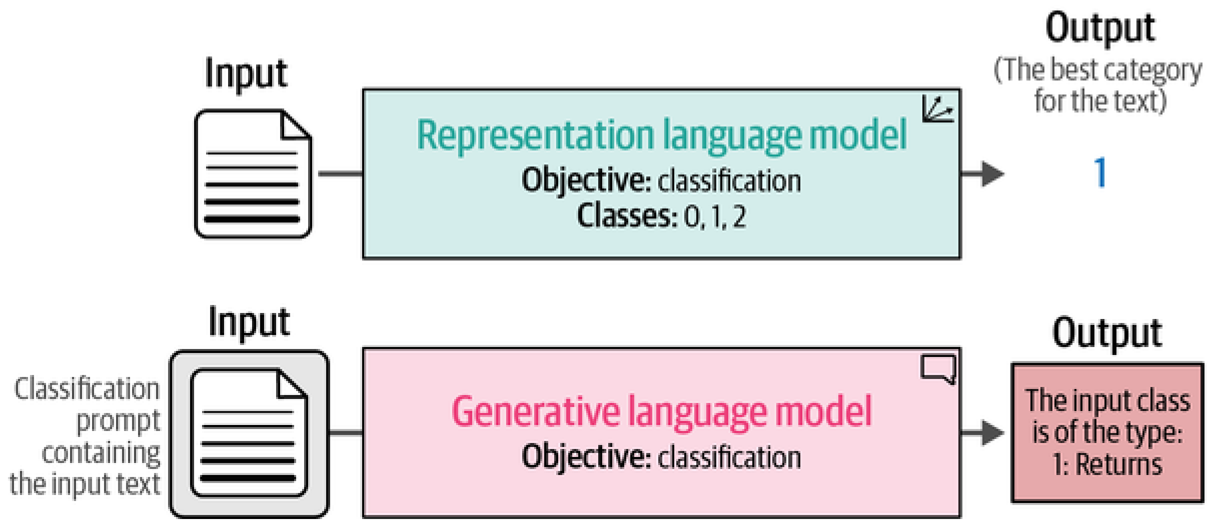
Francesco Guerra
francesco.guerra@unimore.it

Big Data Analysis

## Text Classification

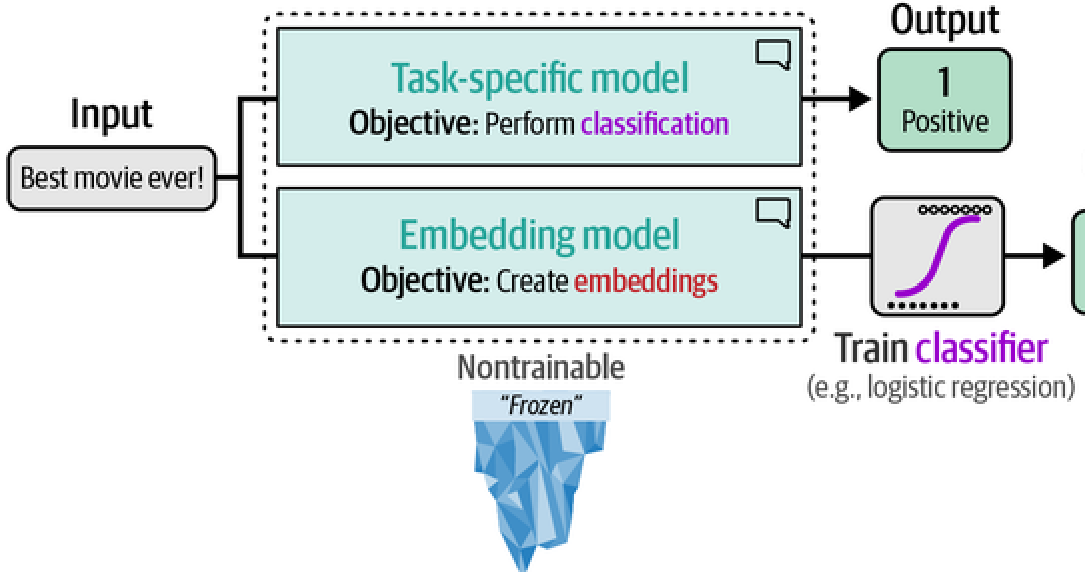
## Text Classification

- The goal of the task is to train a model to assign a label or class to some input text.
  - Classifying text is used across for sentiment analysis, intent detection, extracting entities, detecting language …
- Two base techniques:


Input
Representation language model
Objective: classification
Classes: 0, 1, 2
Output
(The best category for the text)
1
Input
Classification prompt containing the input text
Generative language model
Objective: classification
Output
The input class is of the type: 1: Returns

## Text Classification with Representation Models

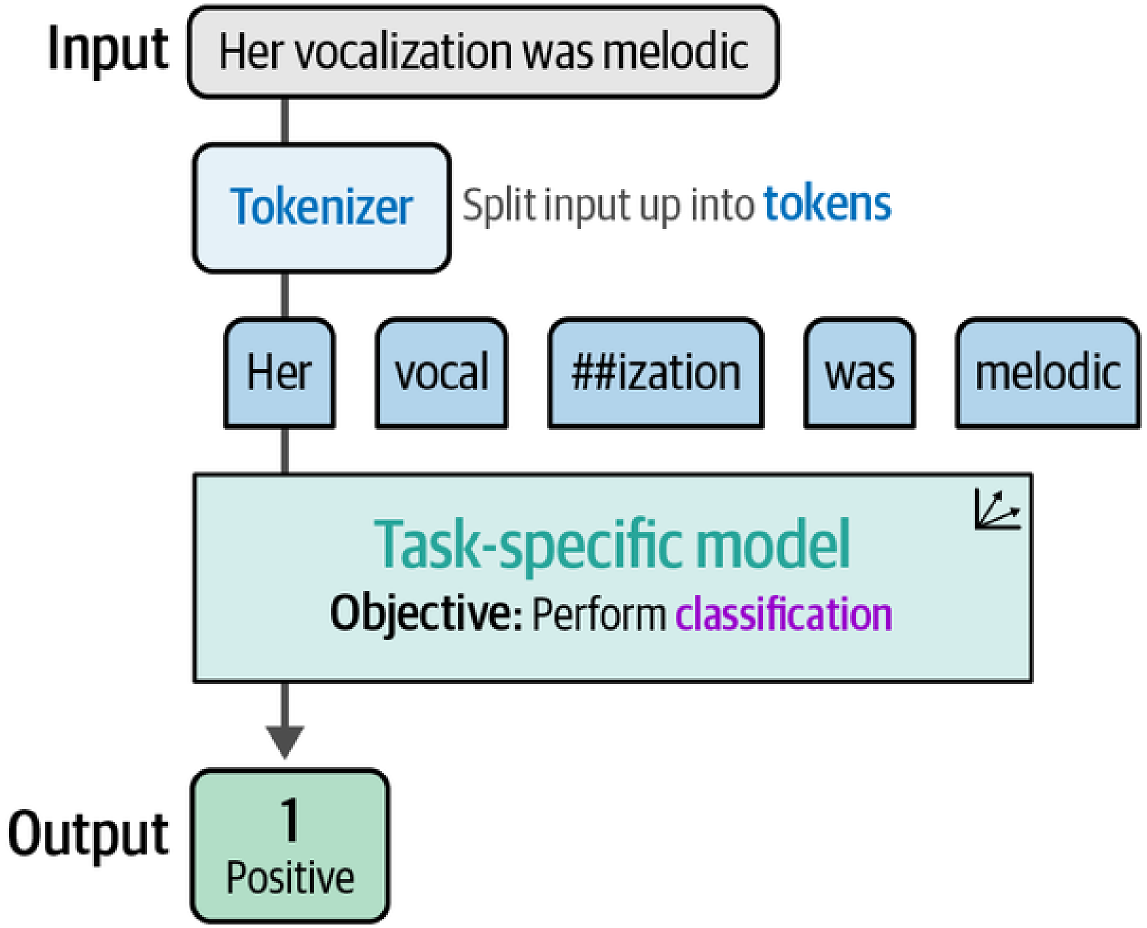
- Two techniques:
  - using a task-specific model or an embedding model



## Model Selection

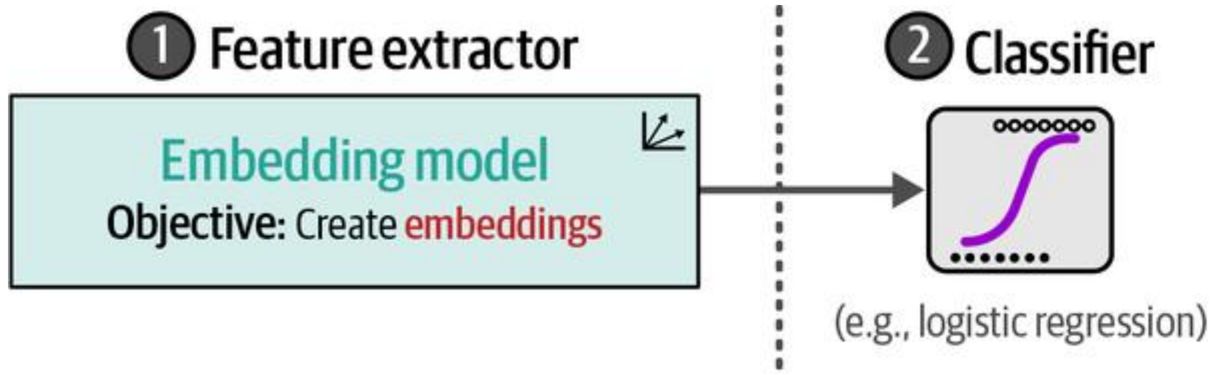
- Choosing the right models is not as straightforward
  - over 60,000 models on the Hugging Face Hub for text classification and more than 8,000 models that generate embeddings in 2025.
  - it's crucial to select a model that fits your use case and consider its language compatibility, the underlying architecture, size, and performance.
- BERT (or one of its variations) are a popular choice for creating task-specific and embedding models. https://huggingface.co/models
  - BERT base model (uncased)
  - RoBERTa base model
  - DistilBERT base model (uncased)
  - DeBERTa base model
  - bert-tiny
  - ALBERT base v2

## Using a Task-Specific Model


Input
Her vocalization was melodic
Tokenizer
Split input up into tokens
Her
vocal
##ization
was
melodic
Task-specific model
Objective: Perform classification
Output
1
Positive

## Classification Tasks That Leverage Embeddings

- Typically a two step approach is used:
  - an embedding model for generating features.
  - a usual classifier fed with the features.



  - Issue: generating embeddings of sentences

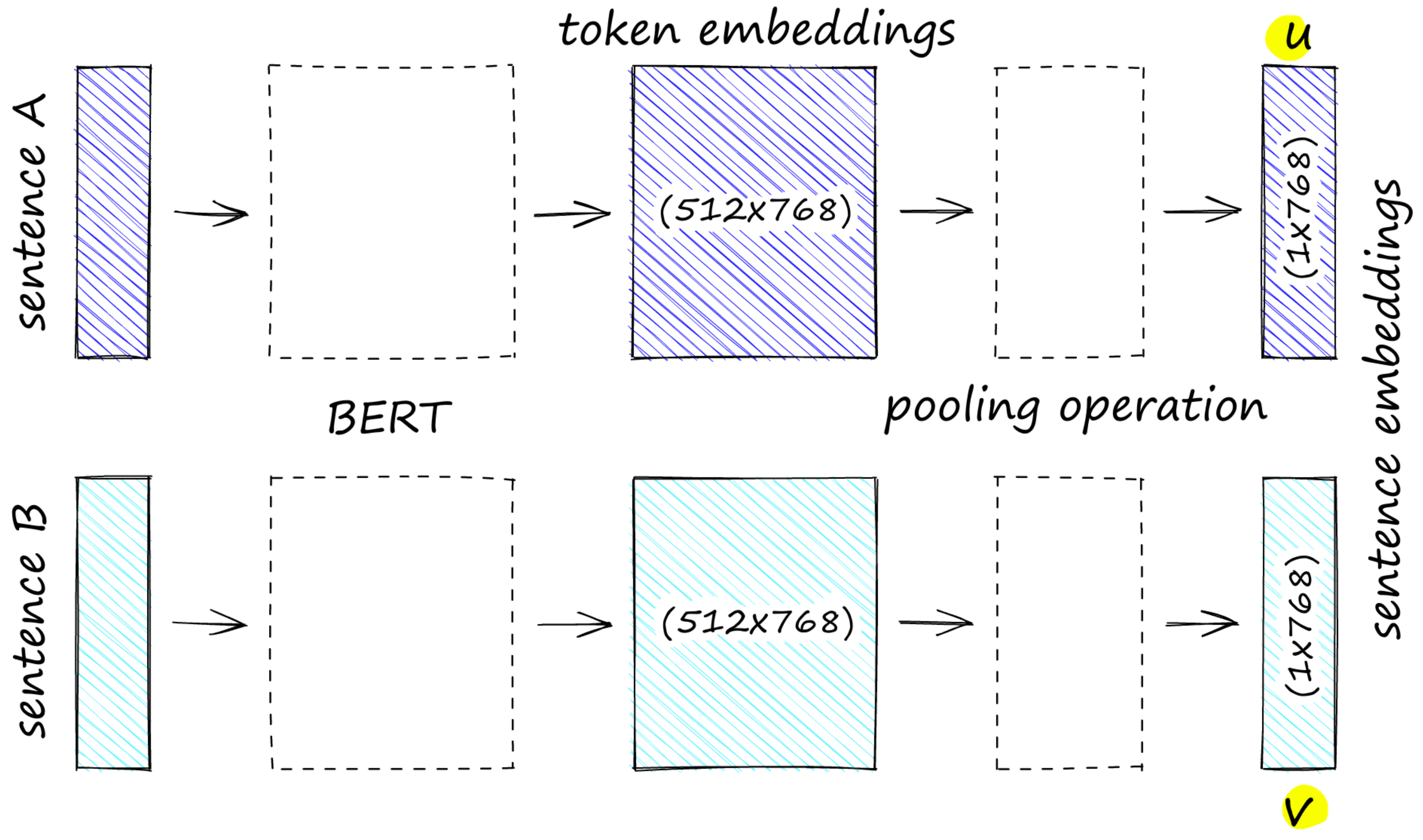
## But… How Do We Generate Sentence Embeddings?

- The most intuitive approach:
  - **Use BERT as an encoder to extract embeddings**
- Two common strategies:
  - **Use the [CLS] token**
  - **Average all token embeddings**
- This gives us a **vector per sentence** usable in the 2-step pipeline.

## Limits of BERT Embeddings

- BERT was **not designed for independent sentence embeddings.**
  - [CLS] and average pooling **do not capture sentence-level semantics**
  - They often perform **worse than simple GloVe averages**
- Embeddings are not positioned in vector space according to semantic similarity.
- ✓ Good for classification **when trained jointly**
- ❌ Poor when sentences are encoded **independently**.

# Sentence-BERT: Sentence embeddings using Siamese BERT-networks.(EMNLP 2019)

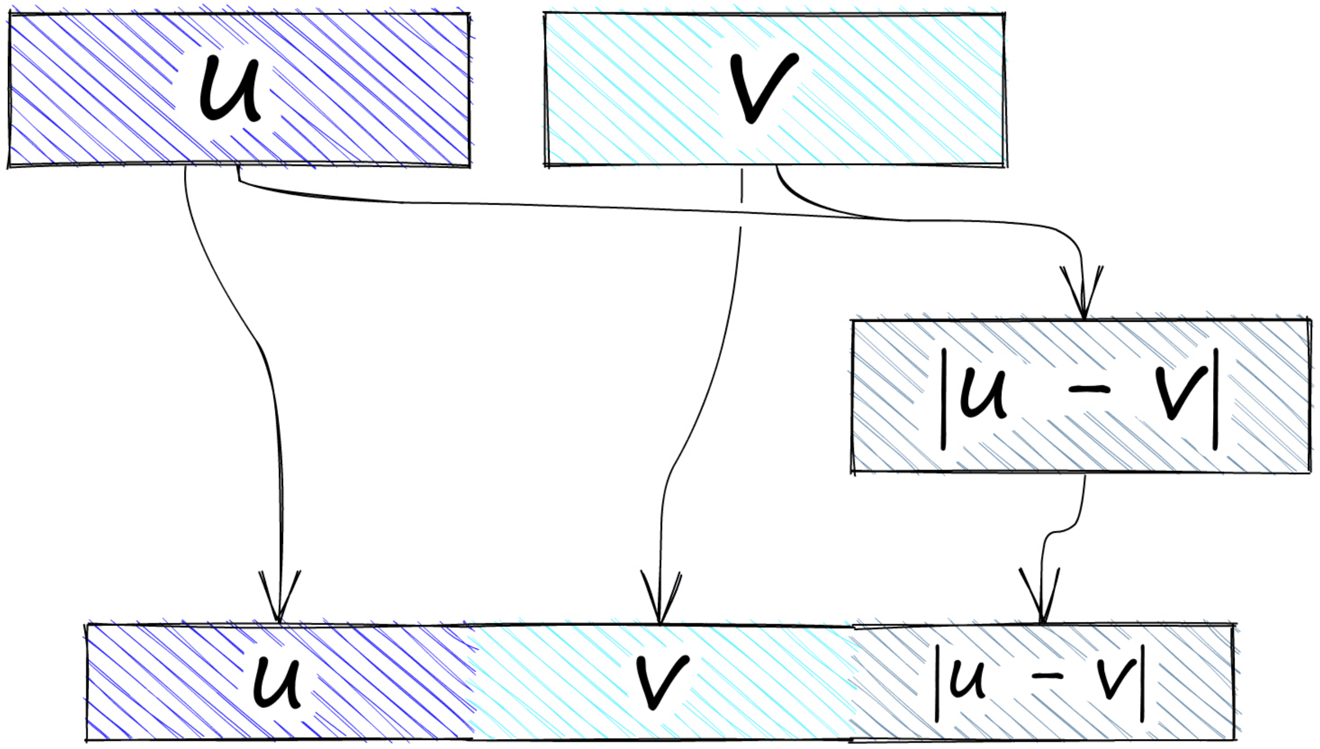


https://www.pinecone.io/learn/series/nlp/sentence-embeddings/#Sentence-Transformers

## Sentence-BERT: Pre-training

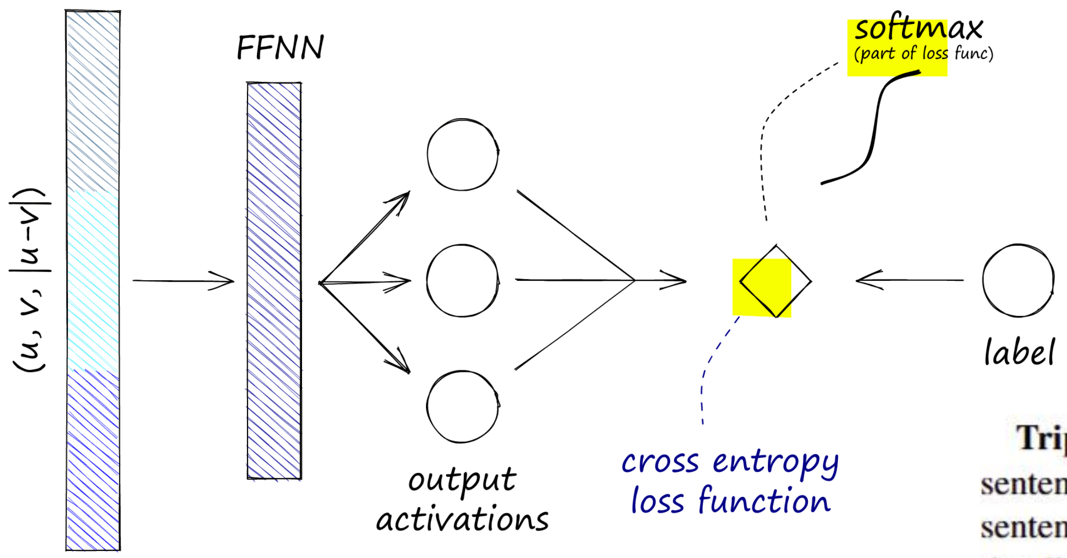
- **Objective:**
  - Fine-tune **Siamese BERT** using **softmax-loss.**
  - Datasets: **SNLI (570K pairs)** and **MNLI (430K pairs).**
- **Dataset labels:**
  - 0 — entailment: premise **implies** hypothesis.
  - 1 — neutral: premise and hypothesis **not necessarily related.**
  - 2 — contradiction: premise and hypothesis **contradict** each other.
- **Procedure:**
  - Feed **sentence A (premise)** into **BERT A** and **sentence B (hypothesis)** into **BERT B.**
  - Siamese BERT produces **pooled sentence embeddings.**
- **Pooling methods:**
  - **Mean pooling** (best performance for NLI and STS benchmark).
    - Max pooling
    - [CLS] pooling

## Sentence-BERT: Pre-training



There are now two sentence embeddings: embeddings A u and embeddings B v.
There are many concatenation approaches that we can test: highest performing is (u, v, |u-v|).

## Sentence-BERT: Pre-training



**Classification Objective Function.** We concatenate the sentence embeddings $u$ and $v$ with the element-wise difference $|u-v|$ and multiply it with the trainable weight $W_t \in \mathbb{R}^{3n \times k}$:

$$o = \text{softmax}(W_t(u, v, |u-v|))$$

where $n$ is the dimension of the sentence embeddings and $k$ the number of labels. We optimize cross-entropy loss. This structure is depicted in Figure 1.

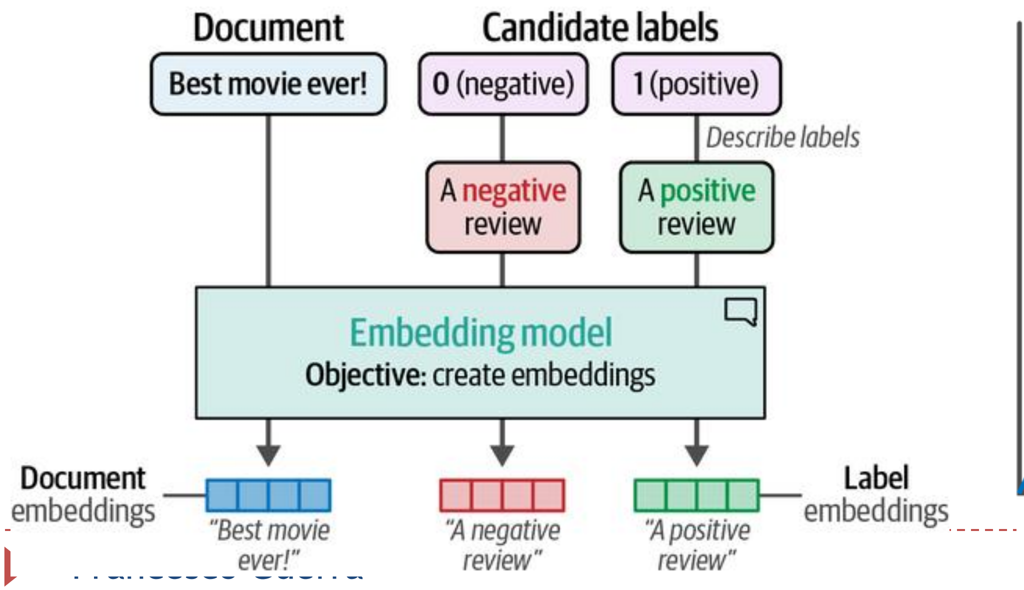
**Triplet Objective Function.** Given an anchor sentence $a$, a positive sentence $p$, and a negative sentence $n$, triplet loss tunes the network such that the distance between $a$ and $p$ is smaller than the distance between $a$ and $n$. Mathematically, we minimize the following loss function:

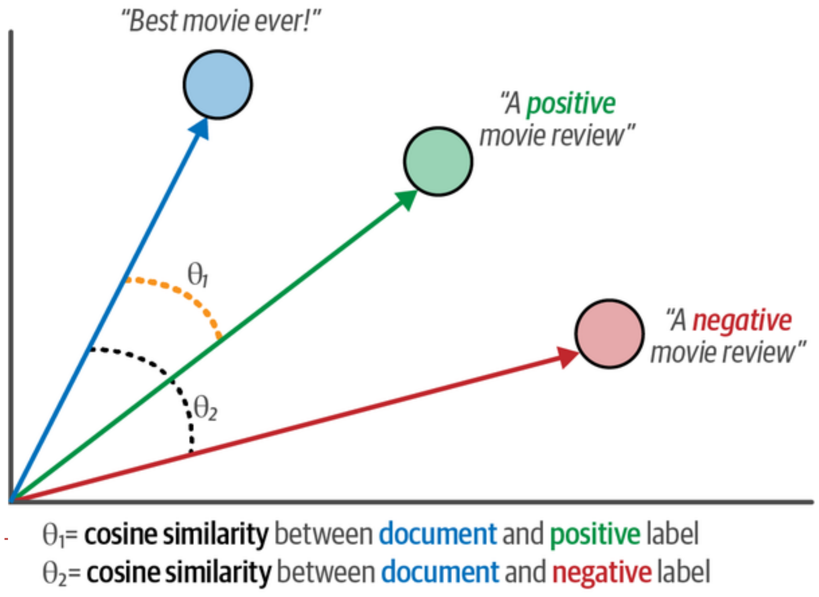
$$max(||s_a - s_p|| - ||s_a - s_n|| + \epsilon, 0)$$

with $s_x$ the sentence embedding for $a/n/p$, $||\cdot||$ a distance metric and margin $\epsilon$. Margin $\epsilon$ ensures that $s_p$ is at least $\epsilon$ closer to $s_a$ than $s_n$. As metric we use Euclidean distance and we set $\epsilon = 1$ in our experiments.

## Zero-shot classification

- Zero-shot classification attempts to predict the labels of input text even though it was not trained on them
  - we can describe our labels based on what they should represent.
    - a negative label can be described as “This is a negative movie review.”
  - To assign labels to documents, we can apply cosine similarity to the document label pairs.




"Best movie ever!"
"A positive movie review"
"A negative movie review"
θ1
θ2
θ1= cosine similarity between document and positive label
θ2= cosine similarity between document and negative label

## Text Classification with Generative Models

- Prompt engineering: iteratively improving your prompt to get your preferred output.
  - T5
  - GPT

## Text Clustering

## Text Clustering

- Text clustering aims to group similar texts based on their semantic content, meaning, and relationships, for
  - facilitating efficient categorization of large volumes of unstructured text
  - for quick exploratory data analysis.
- A possible pipeline:
  1. Convert the input documents to embeddings with an embedding model.
  2. Reduce the dimensionality of embeddings with a dimensionality reduction model.
  3. Find groups of semantically similar documents with a cluster model.

## Dimensionality Reduction

- Transforming high-dimensional features into a lower-dimensional space preserving the most representative information.
  - PCA (Principal Component Analysis): Linearly projects data onto axes of maximum variance; assumes global linear structure.
  - T-SNE (t-Distributed Stochastic Neighbor Embedding): A nonlinear technique that preserves local similarities.
  - UMAP (Uniform Manifold Approximation and Projection): A learning technique that preserves both local and some global structures.
  - Autoencoder: Learns a nonlinear compressed representation through a neural network.

## T-SNE key intuition

- Mapping high-dimensional data points into a two- or three-dimensional data, preserving the local relationships between points.
  - It achieves this by measuring the similarity between points in the high-dimensional space and representing this similarity as probabilities.
  - It constructs a similar probability distribution in the lower-dimensional space and minimizes the difference between distributions using gradient descent.
- Preserving local relationships between points
  - "Preserving the local relationships between points" refers to maintaining the relative distances and similarities between neighboring data points when they are mapped from a high-dimensional space to a lower-dimensional space.
  - Points that are **close in high-dimensional space** should remain **close in low-dimensional space**

## Why Not Work Directly with Distances?

- Compute the distance between points is not reliable in higher dimensions due to the “curse of dimensionality.”
  - In high-dimensional spaces, **all distances become similar**
  - Difference between nearest and farthest neighbors shrinks
  - Distance rankings become unstable and noisy
- **Example Intuition**
  - In 2D: distances vary a lot → meaningful neighborhoods
  - In 100D: distances concentrate around the mean
  - Raw distances lose discriminative power

## Step 1: Similarities in High Dimensions

- For each data point (e.g., $x_1$)
  - Compute distances to all other points
  - Convert distances into probabilities using a Gaussian distribution centered on the target point

$$P_{j|i} = \frac{\exp\left(-\frac{\left\|x_i - x_j\right\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\left\|x_i - x_k\right\|^2}{2\sigma_i^2}\right)}$$

  - Each point has its own $\sigma_i$
- This process is repeated for every point in the dataset, generating an n × n similarity matrix.
  - The resulting matrix is not symmetric, since the similarity from $x_1$ to $x_2$ may differ from $x_2$ to $x_1$.
  - t-SNE symmetrizes them using $P_{ij} = \frac{P_{j|i} + P_{i|j}}{2N}$

## Choosing Sigma ($\sigma_i$): Density Normalization via Entropy and Perplexity

- Data points may lie in regions with **very different densities**
- A fixed Gaussian width would:
  - oversmooth dense regions
  - undersmooth sparse regions
- Each point gets its own $\sigma_i$ to adapt to its local density
- Role of Entropy
  - Quantifies how **spread out** the neighborhood probabilities are
  - Low entropy → few dominant neighbors (very local view)
  - High entropy → many neighbors with similar weight (broader view)
- Role of Perplexity
  - $Perplexity(P_i) = 2^{H(P_i)}$
  - Measures the **effective number of neighbors** that $x_i$ considers close

## How $\sigma_i$ Is Computed (Binary Search)

- For each point $x_i$:
  - Start with an initial guess for $\sigma_i$
  - Compute conditional probabilities$P_{j|i}$
- Compute entropy $H(P_i)$ and perplexity
- Compare with the target (user-defined) number of neighbors to preserve
- Adjust $\sigma_i$:
  - increase $\sigma_i \rightarrow$ increase entropy / perplexity
  - decrease $\sigma_i \rightarrow$ decrease entropy / perplexity
- Repeat until perplexity matches the target

## Example

- Number of neighbors to preserve = 3
- Initial guess $\longrightarrow$ $\sigma_i = 0.2$

| Data point | Dist | Prob |
|---|---|---|
| A | 0.10 | 0.543 |
| B | 0.20 | 0.374 |
| C | 0.40 | 0.083 |
| D | 0.80 | ~0 |
| E | 1.60 | ~0 |

- Entropy = 1.31 $\longrightarrow$ Perplexity = $2^{1.31} \approx 2.48$
- The perlplexity suggests that less the 3 neighbors are taken into account…we have to adjust it
- $\sigma_i = 0.4$ $\longrightarrow$ Perplexity = $2^{1.58} \approx 3$

## Step 2: Similarities in Low Dimensions

- We now reduce the high-dimensional data into a lower-dimensional space, where data points are initially randomly placed along the x-axis.
- In this lower-dimensional space, we recalculate the similarity scores for each point relative to all others.
- This results in a second n × n similarity matrix, now based on low-dimensional distances.
- At this point, we have two matrices:
  - One representing similarity scores in the original high-dimensional space
  - One representing similarity scores in the lower-dimensional space

## Optimization Objective

- The goal is to make the low-dimensional similarity matrix resemble the high-dimensional one as closely as possible.
- To do this, we minimize the difference between the two matrices using a divergence measure, typically the Kullback-Leibler (KL) divergence.
  - $KL(P||Q) = \sum_{i,j} P_{ij} log \frac{P_{ij}}{Q_{ij}}$
- The algorithm iteratively adjusts the positions of points in the lower-dimensional space to reduce this divergence and better preserve the structure of the original data.
  - To minimize the KL divergence, we use gradient descent.

## T-SNE configuration

- Perplexity: Measures the effective number of neighbors for each point. (5-50)
  - Balances local vs. global structure: Low perplexity → focus on local details High perplexity → captures global structure
- Learning Rate: Controls the step size during optimization. (10-100)
  - Too high → algorithm may oscillate or miss global minimum.
  - Too low → slow convergence, may get stuck in local minima.
- Number of Iterations: Defines how many times the algorithm updates the embedding. (1000)
  - Too few → embedding may not converge.
  - Too many → unnecessary computational cost.

## UMAP in a nutshell

- Preserves Local & Global Structure: Captures both fine-grained details and overall data shape.
- Flexible Parameter Tuning: n_neighbors and min_dist control the locality-globality trade-off.
- Faster than t-SNE.
- High-level intuition
  - learns a graph in high-dimensional space
  - learns a similar graph in low-dimensional space
  - Goal: make the two graphs as similar as possible

## UMAP Parameters

- n_neighbors
  - Controls the size of the local neighborhood used to build the graph structure (typical: 5–50)
- min_dist
  - Sets the minimum distance between embedded points.
  - Low (0.0–0.1): tight clusters
  - Medium (0.2–0.4): balanced
  - High (0.5+): spread-out structure
- n_components
  - Dimensionality of the target space (e.g., 2D or 3D).
- Metric
  - Distance function for computing neighbor similarity (e.g. euclidean, cosine, manhattan).

## Step 1: Construct a k-Nearest Neighbor Graph

- For each point $x_i$:
  - find its k nearest neighbors (KNN)
- k is controlled by the parameter n_neighbors
- Interpretation:
  - Small k → focus on very local structure
  - Large k → incorporate more global structure

## From Distances to Edge Weights

- For each point $x_i$ and neighbor $x_j$:
  - Compute distance $d(x_i, x_j)$
  - Convert distances into weights $w_{ij} = \exp(-\frac{d(x_i, x_j) - \rho_i}{\sigma_i})$
- Exponential function
  - Enforces rapid decay of similarity with increasing distance
  - Ensures that only very close neighbors have strong influence
- $\rho_i$ = distance to the closest neighbor
  - zero distance to the nearest neighbor -> a **maximum weight of 1** for the nearest neighbor
  - Normalizes local neighborhoods across regions with different densities
- $\sigma_i$ = local connectivity scale
  - Similar to t-SNE, $\sigma_i$ adapts to local density
  - $\sum_{j \epsilon KNN(i)} \exp\left(-\frac{d(x_i, x_j) - \rho_i}{\sigma_i}\right) \approx log_2 k$

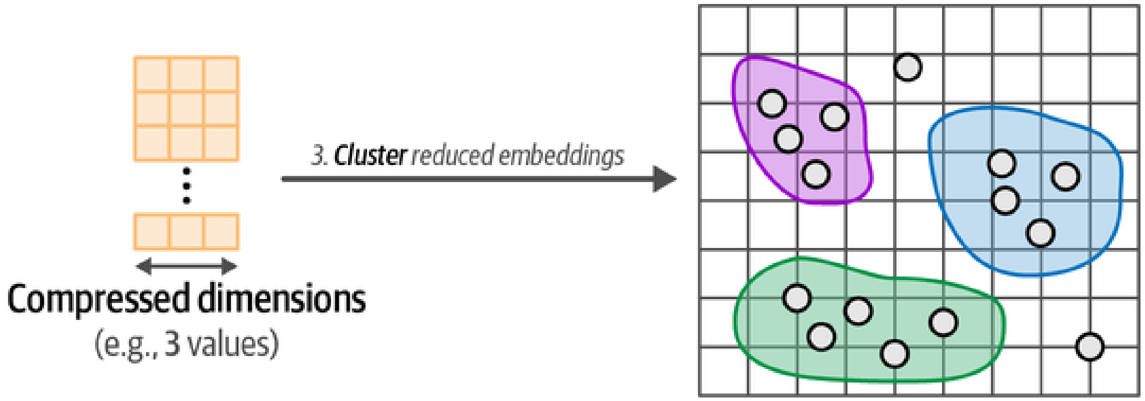
## Symmetrization of the Graph

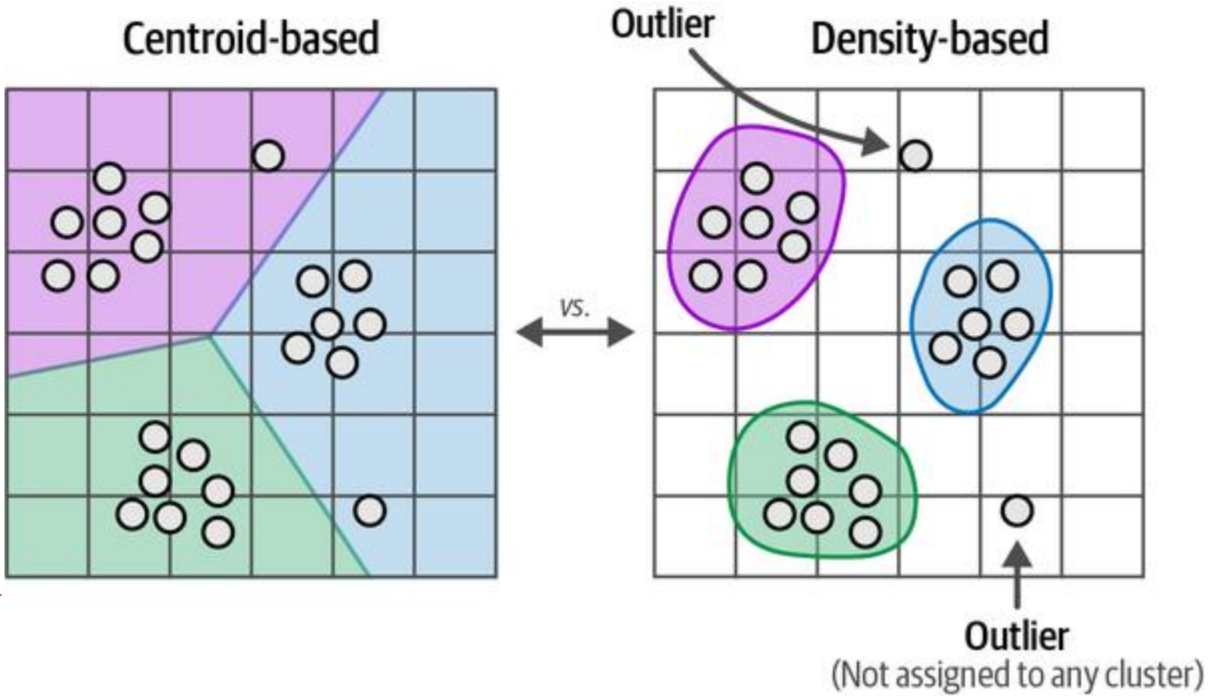
- Neighborhood relations are **asymmetric**
- UMAP symmetrizes edge weights using
  - $w_{ij}^{sym} = w_{ij} + w_{ji} - w_{ij} * w_{ji}$
- Preserves strong mutual connections
- Avoids shrinking weak one-sided links

## Optimization Objective

- Minimize **cross-entropy** between high-D and low-D graphs
  - $CE = \sum_{i,j} w_{ij} \log(q_{ij}) + (1 - w_{ij}) \log(1 - q_{ij}))$
- Encourages:
  - connected points to stay close
  - unconnected points to stay apart

## Cluster the Reduced Embeddings





## HDBSCAN in a nutshell

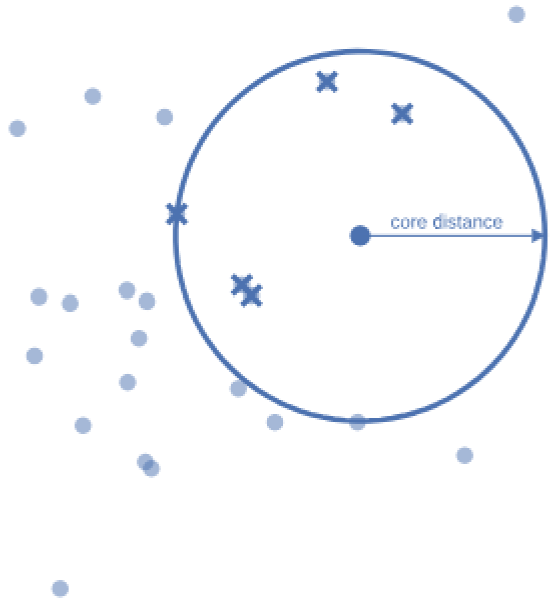
- HDBSCAN is a density-based clustering algorithm
  - Clusters are defined as regions with many nearby points
  - Sparse regions are considered separators or noise
- Intuition:
  - A cluster is a "crowded area"
  - Noise lives in "empty space"
- It automatically finds:
  - clusters of varying density
  - noise / outliers
- Widely used after UMAP or t-SNE embeddings
- No need to specify the number of clusters

## HDBSCAN pipeline

1. Estimate local density (core distance)
2. Modify distances (mutual reachability)
3. Build MST
4. Construct hierarchy
5. Condense using minimum cluster size
6. Select clusters via stability

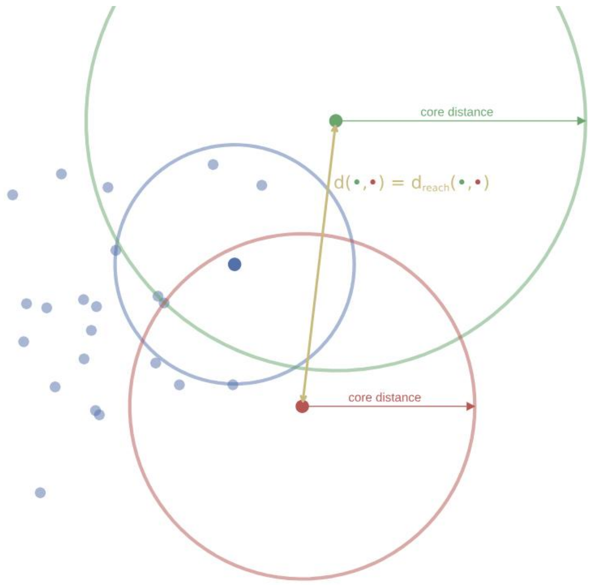
## Step 1: Measuring Local Density

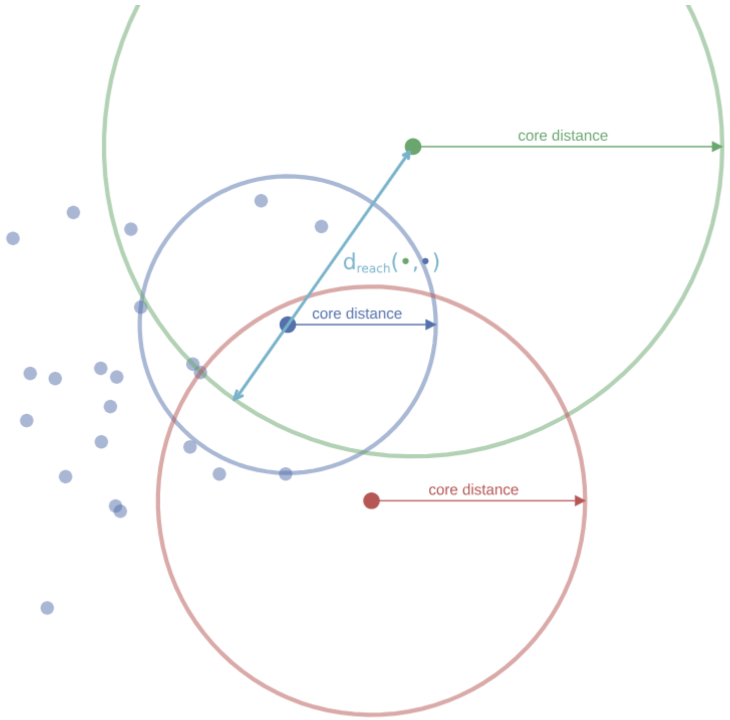
- For each point $x_i$:
  - find its k-th nearest neighbor (min_samples parameter)
- The distance to this neighbor defines the core distance:
- $\text{core\_dist}(x_i) = \text{distance to the k-th nearest neighbor}$
- Dense regions → small core distance
- Sparse regions → large core distance



## Mutual Reachability Distance

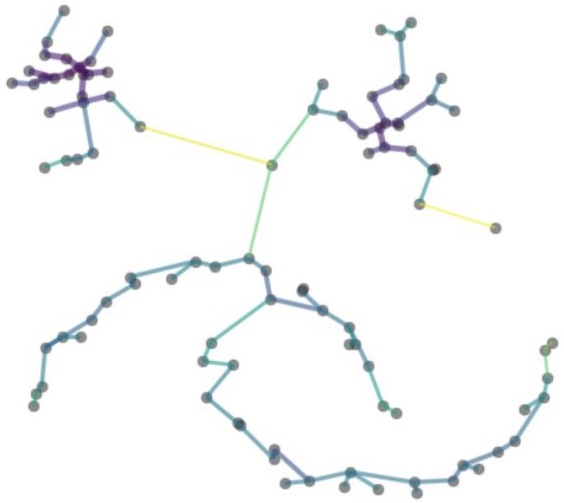
- To compare two points $x_i$ and $x_j$, HDBSCAN uses:
  - $\mathrm{mreach}(x_i, x_j) = \max(\mathrm{core\_dist}(x_i), \mathrm{core\_dist}(x_j), \mathrm{distance}(x_i, x_j))$
- Intuition:
  - Penalizes pairs involving sparse points
  - Makes distances density-aware





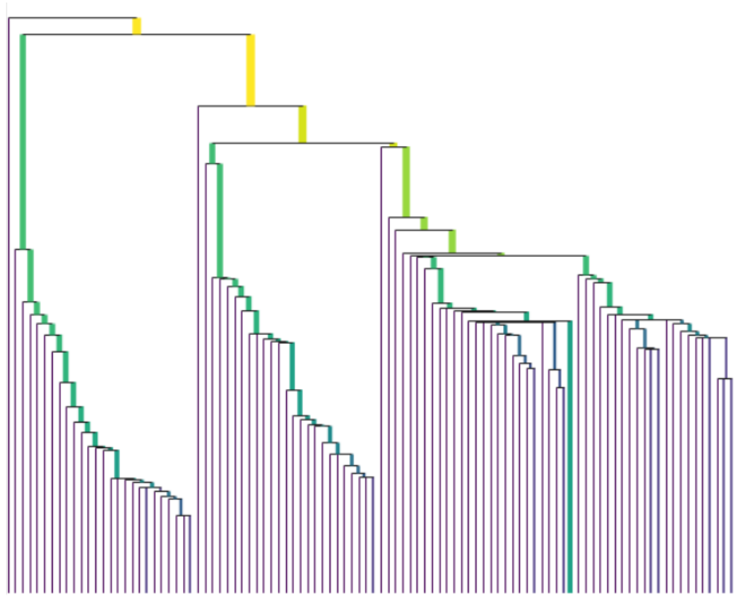
## Build a Density-Aware Graph

- Construct a graph where:
  - nodes = data points
  - edge weight = mutual reachability distance
- Iteratively cutting a fully connected graph at different weight thresholds would be very expensive.
- Compute a Minimum Spanning Tree (MST) of this graph
- MST captures the essential density structure



## From MST to Hierarchical Clustering

- Procedure:
  - Sort MST edges by increasing weight
  - Add edges one by one
  - Connected components merge over time
- This produces a **single-linkage dendrogram:**
  - Leaves = individual points
  - Merges happen at increasing distances



## Why Distance-Based Cuts Are Not Enough

- Suppose we cut the dendrogram at a fixed distance:
  - Dense cluster → preserved
  - Sparse cluster → destroyed
- Or:
  - Sparse cluster → preserved
  - Dense cluster → merged with others
- A **single distance threshold cannot handle variable densities**

## Reparameterization: From Distance to Density (λ)

- $\lambda = \frac{1}{distance}$
- Interpretation:
  - High λ → high density
  - Low λ → low density
- This does **not** change the tree structure, only how we *interpret* it.
- Now clusters live over **intervals of density**, not distance.

## Cluster birth and death

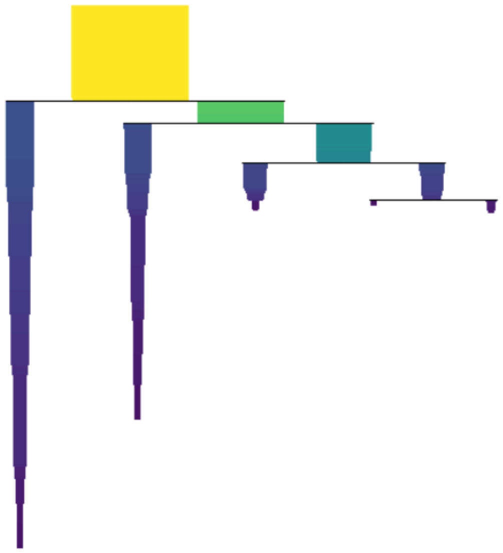
- A cluster is **born** when:
  - Its points first become connected
  - At a specific λ value
- A cluster **dies** when:
  - It merges with another cluster, or
  - It splits into valid subclusters
- Birth and death define the *lifetime* of a cluster.

## Step 5: Condensed Cluster Tree

- Observation:
  - Many splits separate only 1–2 points
  - These are usually noise, not real clusters
- Introduce the parameter *minimum_cluster_size*
  - If a split creates a cluster smaller than this size → treat it as points **falling out**
  - Otherwise → accept it as a real split
- Result:
  - A simplified hierarchy: the **condensed tree**

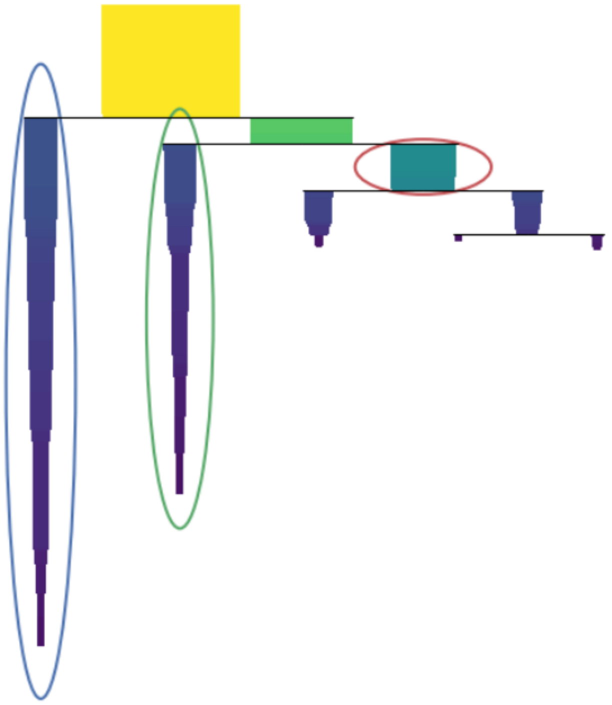
## Example of condensed tree

- In the condensed tree:
  - Each cluster is drawn as a **rectangle**
  - x-axis: number of points (cluster size)
  - y-axis: density λ
- As λ decreases:
  - Points may fall out → rectangle narrows
  - Cluster may split → new rectangles appear



## Cluster Stability

- Intuition:
  - A good cluster exists **for a long time** and contains **many points.**
- Stability measures:
  - How long a cluster persists
  - Weighted by how many points stay in it
- Formally:
  - stability = sum over points of (λ_p − λ_birth)
    - $\lambda$_birth: cluster birth
    - $\lambda\_p$: point leaves the cluster
- Geometrically:
  - **Stability = area of the rectangle**



## Step 6: Selecting the Final Clusters

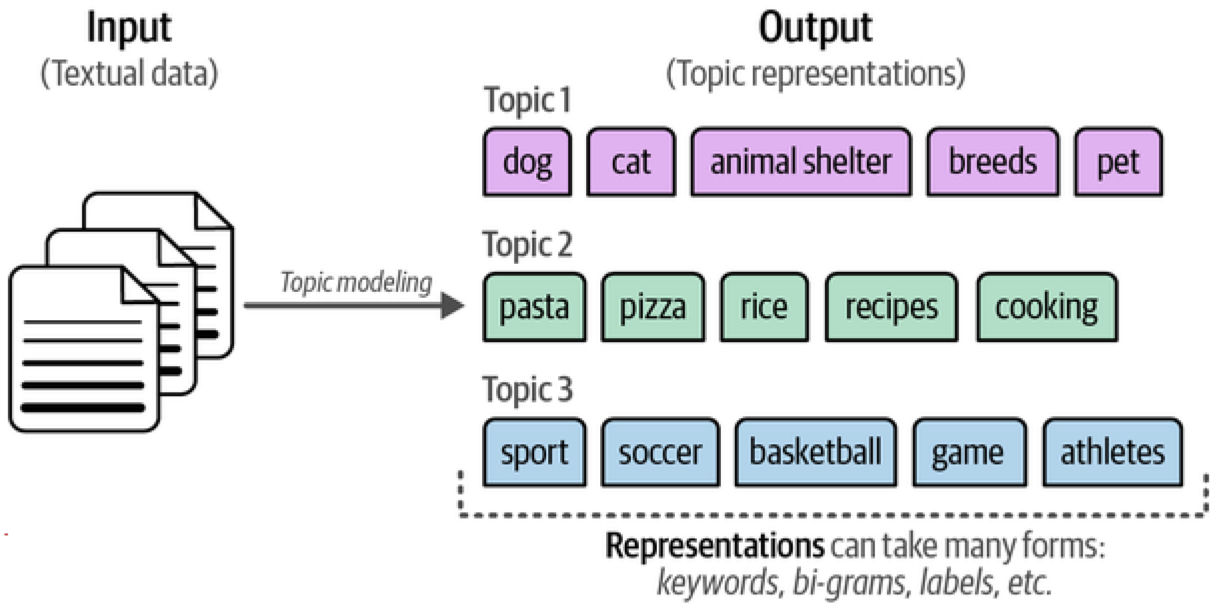
- We do **not** cut at a fixed λ.
- Instead:
  - Traverse the condensed tree bottom-up
  - For each cluster:
    - Compare its stability with the **sum of its children's stabilities**
- Rule:
  - If parent is more stable → select parent
  - Otherwise → select children
- This ensures:
  - No overlapping selections
  - Preference for robust structure

## Final Output of HDBSCAN

- HDBSCAN returns:
  - **Cluster labels**
    - Points belonging to stable clusters get a label
  - **Noise points** (label = −1)
    - Points not consistently assigned become **noise**
  - **Membership strength** for each point
- Membership strength:
  - Derived from normalized λ values
  - Indicates confidence of assignment

## Inspecting the Clusters / Topic modeling

- We can inspect each generated cluster manually and explore the assigned documents to get an understanding of its content.
- This idea of finding themes or latent topics in a collection of textual data is often referred to as *topic modeling*.
- Traditionally, it involves finding a set of keywords or phrases that best represent and capture the meaning of the topic



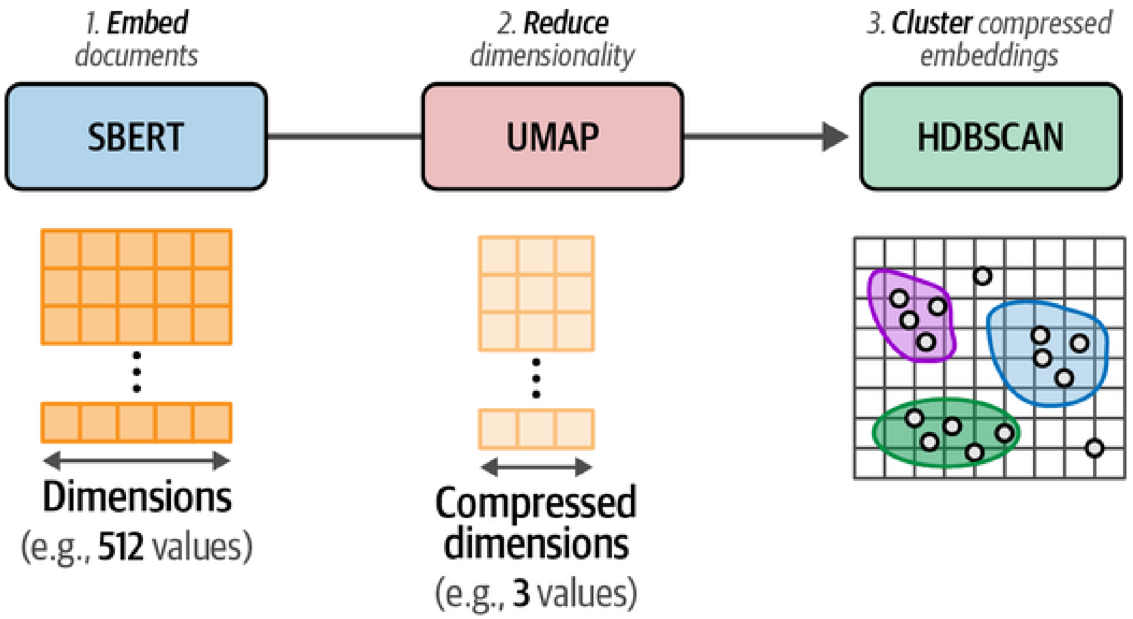
## Topic Modeling

- Unsupervised statistical method for discovering hidden themes in large text collections (typically in the form of keywords).
- Automatically identifies topics from word patterns—no manual labeling needed.
- Reveals how themes are connected and how they evolve over time.
- Enables scalable organization and summarization of large text archives.

David M. Blei: Probabilistic topic models. Commun. ACM 55(4): 77-84 (2012)

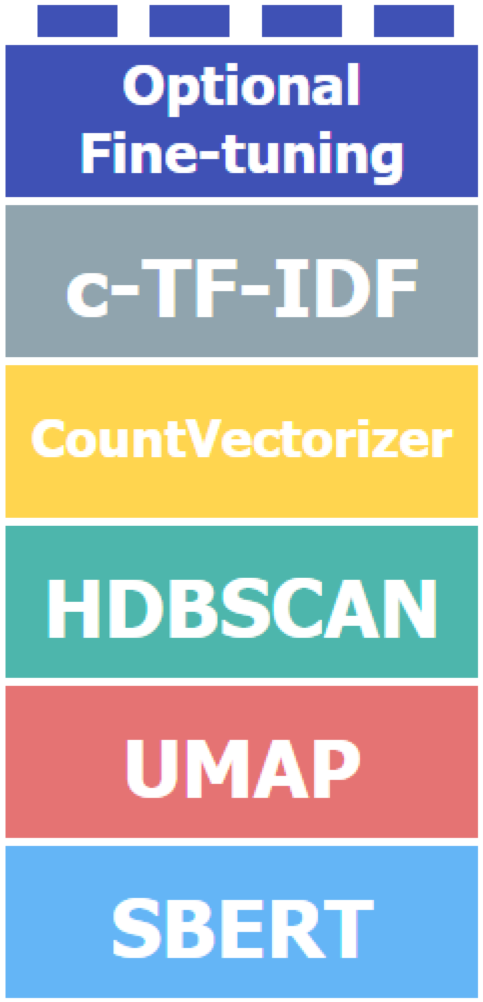
## BERTopic in a nutshell

- Combines:
  - Transformer embeddings (semantic meaning)
  - Dimensionality reduction (UMAP)
  - Density-based clustering (HDBSCAN)
  - Interpretable topic representations (c-TF-IDF)


1. Embed documents
2. Reduce dimensionality
3. Cluster compressed embeddings
SBERT
UMAP
HDBSCAN
Dimensions (e.g., 512 values)
Compressed dimensions (e.g., 3 values)

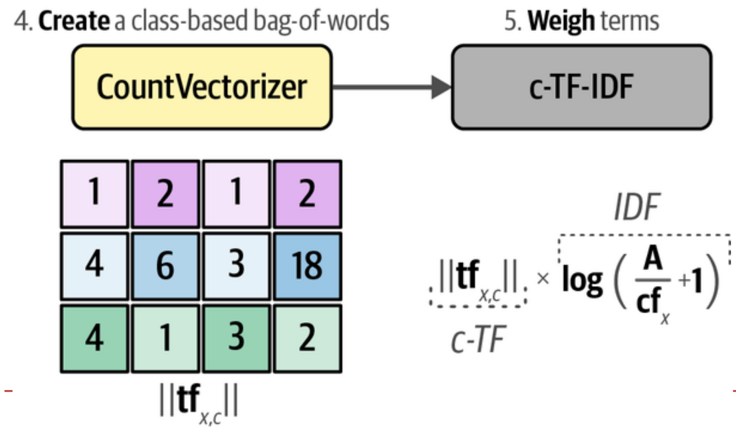
## High-level pipeline

1. Embed documents with BERT / SBERT
2. Reduce dimensionality with UMAP
3. Cluster documents with HDBSCAN
4. Extract topic words with c-TF-IDF
5. (Optional) Refine topics with re-ranking



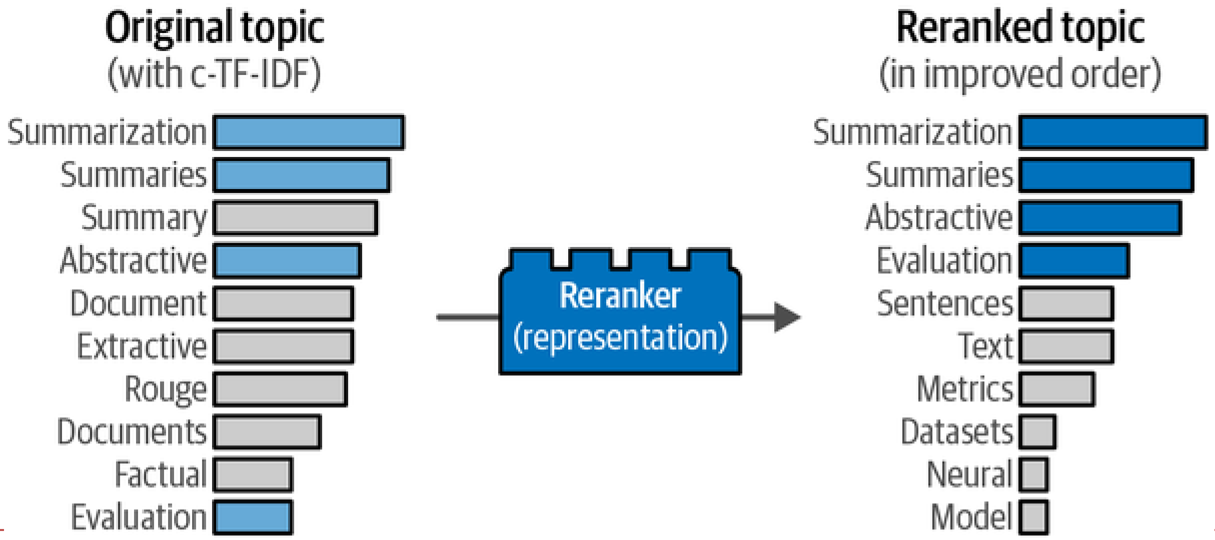
## Topic Representation with c-TF-IDF

- Clustering gives groups of documents
- c-TF-IDF summarizes each cluster with keywords
  - a class-based variant of TF-IDF to put more weight on words that are more meaningful to a cluster and less weight on words that are used across all clusters.
  - $tf_{x,c}$: frequency of term x in cluster c (raw count)
  - $cf_x$ : number of clusters where x appears
  - A: average number of words per cluster


4. Create a class-based bag-of-words
CountVectorizer
5. Weigh terms
c-TF-IDF
1 2 1 2
4 6 3 18
4 1 3 2
$||tf_{x,c}||$
IDF
$||tf_{x,c}|| \times \log\left(\frac{A}{cf_x}+1\right)$
c-TF

## Adding a Reranker

- The pipeline in BERTopic still represents a topic through a bag-of-words without taking into account semantic structures.
- A reranker leverages the strength of the bag-of-words representation to generate a meaningful representation.
  - We can use this first meaningful representation and tweak it using more powerful but slower techniques, like embedding models.



UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

## References

- D. Jurafsky & J. H. Martin: **Speech and Language Processing**, Pearson International 3ed draft https://web.stanford.edu/~jurafsky/slp3/
- D. Jurafsky, slides from NLP Courses http://web.stanford.edu/~jurafsky/
- Dan Klein, Slides from NLP Course http://www.cs.berkeley.edu/~klein/cs288/sp10/slides/SP10%20cs288%20lecture%203%20--%20language%20models%20II%20(6PP).pdf
- J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets Stanford University http://www.mmds.org
- Landauer, T. K., Foltz, P. W., & Laham, D. (1998). Introduction to Latent Semantic Analysis. *Discourse Processes*, **25**, 259-284.
- Bing Liu: Sentiment Analysis and Opinion Mining. Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers 2012
- Jay Alammar, Maarten Grootendorst: Hands-On Large Language Models Published by O'Reilly Media, Inc.

## References

- Zhijiang Guo, Michael Schlichtkrull, and Andreas Vlachos. 2022. A Survey on Automated Fact-Checking. *Transactions of the Association for Computational Linguistics*, 10:178–206.