Explain to me the concepts and represntatin and transformers used in this convokit example, and how the output of this use case of the Expected Context Framework + tfidf can be evaluated for performance (what metrics of accuracy)

https://github.com/CornellNLP/ConvoKit/blob/master/convokit/expected_context_framework/demos/parliament_demo.ipynb

The ConvoKit Expected Context Framework (ECF) analyzes conversational data by modeling expected vs. observed interactions. In the parliamentary debates demo[1], it likely applies these components:

Core Concepts

1. **Expected Context Framework**:
   Models conversational norms by:
   - Comparing actual speaker behavior to statistically expected patterns
   - Identifying deviations (e.g., unusual rhetorical strategies in parliamentary debates)
   - Using context-aware features like speaker role, debate topic, and temporal dynamics
2. **TF-IDF Representation**:
   Transforms text using:
   - **Term Frequency (TF)**: Local word importance within a speech
     $ TF(t,d) = \frac{Count of t in d}{Total terms in d} $
   - **Inverse Document Frequency (IDF)**: Global word rarity across speeches
     $ IDF(t) = \log\left(\frac{Total documents}{Documents containing t}\right) $
   - **TF-IDF Score**: $ TF(t,d) \times IDF(t) $ [2] [3]

Key Components

Transformers/Pipeline:

1. **TF-IDF Vectorizer**
   - Converts speeches to sparse matrices (word importance scores)
   - Handles n-grams and stopwords (common parliamentary phrases like "honorable member")
2. **ECF Model**

- **Input**: TF-IDF features + metadata (speaker party, debate stage)
- **Output**:
  - Expected vs. observed interaction patterns
  - Anomaly detection (e.g., uncharacteristic aggression in a speech)

Evaluation Metrics

For performance assessment:

Metric	Use Case	Example
Accuracy	Overall prediction correctness	86% accuracy in debate role classification [4] [5]
F1-Score	Balance precision/recall for imbalanced data	Detecting rare procedural violations
Silhouette Score	Cluster quality (if using unsupervised ECF)	Grouping similar rhetorical strategies
Confusion Matrix	Error analysis per class	False positives in opposition party detection

Implementation Considerations

1. **Normalization**:
   TF-IDF vectors are typically L2-normalized to avoid document length bias [6].
2. **Feature Sparsity**:
   Parliamentary vocabularies create high-dimensional matrices – dimensionality reduction (e.g., Truncated SVD) may improve performance [3].
3. **Baseline Comparison**:
   The demo likely contrasts TF-IDF against:
   - Bag-of-Words (accuracy ~86.6% vs. TF-IDF's 89% in similar tasks [7])
   - Context-agnostic models

This framework enables quantitative analysis of political discourse strategies while accounting for institutional norms [1] [3].

⁂

What is the output data of this example

The output data of the ConvoKit Expected Context Framework (ECF) example in the parliament_demo.ipynb consists of several key results derived from the analysis of parliamentary debates. Here's a breakdown of the outputs:

Output Data

1. **Utterance Representations**:
    - Each utterance in the corpus is transformed into a vector representation using TF-IDF or other feature extraction techniques.
    - These representations capture the linguistic and contextual features of each utterance.
2. **Expected Contexts**:
    - The framework computes "expected contexts" for each utterance, based on its conversational surroundings (e.g., preceding or following utterances).
    - These are modeled using latent semantic analysis (LSA) to derive low-dimensional embeddings.
3. **Observed vs. Expected Deviation**:
    - For each utterance, the framework calculates how much its actual content deviates from its expected context.
    - This is represented as a "range" or score that quantifies the deviation.
4. **Cluster Assignments**:
    - Utterances are grouped into clusters based on their semantic or contextual similarities.
    - Cluster assignments are stored as attributes for each utterance, enabling further analysis of rhetorical patterns or discourse strategies.
5. **Term-Level Statistics**:
    - The framework computes term-level statistics such as "orientation" and "shift," which describe how terms are used differently across clusters or contexts.
6. **Visualization-Ready Data**:
    - Outputs include data structures (e.g., DataFrames) summarizing cluster assignments, ranges, and deviations, ready for visualization or further exploration.

Evaluation Metrics

To evaluate the performance of this use case, you can use the following metrics:

1. **Clustering Metrics**:
    - **Silhouette Score**: Measures how well utterances fit within their assigned clusters compared to other clusters.
    - **Adjusted Rand Index (ARI)**: Compares the clustering result to a ground truth (if available).
2. **Deviation Analysis**:
    - Evaluate whether deviations correlate with known anomalies or rhetorical strategies in debates.
    - Use precision/recall if deviations are treated as binary anomalies.

3. **Classification Accuracy (if applicable)**:
   - If the task involves classifying utterances into predefined categories (e.g., question vs. answer), accuracy and F1-score can be used.
4. **Qualitative Evaluation**:
   - Inspect clusters and deviations to assess whether they align with meaningful patterns in parliamentary discourse.

These outputs enable both quantitative and qualitative analysis of conversational dynamics in parliamentary settings, making it possible to uncover rhetorical strategies and deviations from norms.

⁂

What is this Expect context, is it a number?

The **Expected Context** in ConvoKit's Expected Context Framework (ECF) is **not a single number** but rather **a multi-dimensional representation** derived through computational analysis. Here's a breakdown:

What "Expected Context" Represents

1. **Vector Representation**
   The framework uses **Latent Semantic Analysis (LSA)** to model utterances and terms as vectors in a reduced-dimensionality space [8] [9]. These vectors capture semantic relationships between terms/utterances and their conversational contexts.
2. **Range Metric**
   For individual utterances, the framework computes a **"range"** (a numeric score) that quantifies how much an utterance deviates from its expected conversational context [8] [10]. This is the closest to a single-number output, but it exists alongside other metrics.
3. **Cluster Assignments**
   Utterances/terms are grouped into clusters based on their semantic similarity to expected contexts. These clusters are categorical labels rather than numerical values [8] [11].

Key Components

Output Type	Description	Example
Vector Embeddings	Low-dimensional numerical representations of terms/utterances	`[0.23, -0.45, 0.12]` (25 dimensions) [8]
Range Scores	Deviation magnitude between observed and expected context (single float value)	`1.87` (higher = greater deviation) [8]
Cluster Assignments	Categorical labels grouping similar utterances/terms	`Cluster 3: Procedural Debates` [11]

Output Type	Description	Example
Orientation/Shift	Term-level statistics describing usage patterns across contexts	`"motion" orientation: +0.92`[8]

How It's Used

The framework compares:

- **Observed Context**: Actual conversational patterns (e.g., replies to a parliamentary speech)
- **Expected Context**: Statistically normative patterns derived from training data[8] [10]

This comparison enables analysis of rhetorical strategies, anomaly detection, and discourse pattern identification in corpora like parliamentary debates[11] [10]. The outputs are primarily used for qualitative interpretation and downstream machine learning tasks rather than as standalone numerical metrics.

⁂

specifically, how is the expected context used for cluster assignment, and into what kind of clusters (i.e. how are they defined empirically)

The Expected Context Framework (ECF) in ConvoKit uses **vector representations derived from latent semantic analysis (LSA)** to assign utterances or terms to clusters based on their conversational context. Here's how it works:

Cluster Assignment Process

1. **Vector Space Construction**:
    - Utterances and context-utterances (e.g., replies or predecessors) are represented as TF-IDF vectors.
    - LSA reduces these vectors to a lower-dimensional space (e.g., 25 dimensions) to capture semantic relationships[12].
2. **Clustering Mechanism**:
    - **K-Means clustering** groups utterances/terms into `n_clusters` (default: 8) based on their vector similarity[12].
    - Two clustering modes:
        - **Utterance-based**: Default mode clusters training-data utterances first, then assigns terms/contexts to clusters.
        - **Term-based**: Alternative for noisy/long utterances, clustering terms first[12].
3. **Deviation Metric**:

- The "range" score quantifies how much an utterance deviates from its expected context, but clustering primarily uses the **semantic similarity** of vector representations[12] [13].

Empirical Cluster Definitions

Clusters represent **semantic or functional patterns** in conversations. For parliamentary debates (as in the demo), clusters might include:

Cluster Type	Characteristics	Example Terms/Utterances
Procedural Debates	Focus on parliamentary rules and processes	"point of order", "move to adjourn"
Policy Discussions	Substantive debates on legislation	"amendment", "public health implications"
Personal Rhetoric	Ad hominem arguments or appeals to authority	"honorable member fails to understand..."
Audience Engagement	Direct addresses to the chamber or public	"I yield to the speaker", "as this house knows"

Interpretation & Validation

- The `print_clusters()` method displays:
  - Top representative terms
  - Example utterances from each cluster
  - Context-utterance patterns[12].
- Clusters are validated through:
  - **Silhouette scores** (intra-cluster cohesion)
  - Manual inspection for rhetorical coherence[13].

This approach enables systematic analysis of discourse strategies by grouping utterances that share similar contextual roles or semantic content.

⁂

why is TF_idf representted as a matrix, gimme specific example of what this might look like

TF-IDF is represented as a matrix because it systematically captures **term importance across documents** in a structured numerical format. Below is a detailed explanation with a concrete example:

Why a Matrix?

The TF-IDF matrix (or document-term matrix) organizes data as:

- **Rows**: Documents in the corpus
- **Columns**: Unique terms (words) in the corpus
- **Cells**: TF-IDF scores for each term-document pair

This structure enables efficient computation for tasks like similarity analysis, clustering, and machine learning.

Example TF-IDF Matrix

Consider a corpus of **3 documents**:

1. `Doc1`: "The cat sat on the mat"
2. `Doc2`: "The dog chased the cat"
3. `Doc3`: "The cat and the dog played"

Step 1: Term Frequency (TF)

Term	Doc1	Doc2	Doc3
the	2/6	2/5	2/5
cat	1/6	1/5	1/5
dog	0	1/5	1/5
sat	1/6	0	0
mat	1/6	0	0
chased	0	1/5	0
played	0	0	1/5

Step 2: Inverse Document Frequency (IDF)

Term	IDF (log(N/df))
the	log(3/3) = 0
cat	log(3/2) ≈ 0.58
dog	log(3/2) ≈ 0.58
sat	log(3/1) ≈ 1.10
mat	log(3/1) ≈ 1.10
chased	log(3/1) ≈ 1.10
played	log(3/1) ≈ 1.10

Step 3: TF-IDF Scores

Term	Doc1	Doc2	Doc3
the	0	0	0
cat	0.58/6 ≈ 0.10	0.58/5 ≈ 0.12	0.58/5 ≈ 0.12
dog	0	0.58/5 ≈ 0.12	0.58/5 ≈ 0.12
sat	1.10/6 ≈ 0.18	0	0
mat	1.10/6 ≈ 0.18	0	0
chased	0	1.10/5 ≈ 0.22	0
played	0	0	1.10/5 ≈ 0.22

Matrix Representation

In practice, this is stored as a **sparse matrix** (most cells = 0):

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
    "The cat sat on the mat",
    "The dog chased the cat",
    "The cat and the dog played"
]
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)

# Resulting matrix (rows=documents, columns=terms)
print(X.toarray())
```

Output:

```
[[0.   0.1 0.   0.18 0.18 0.    0.   ]
 [0.   0.12 0.12 0.    0.    0.22 0.   ]
 [0.   0.12 0.12 0.    0.    0.    0.22]]
```

Key Observations

1. **Sparsity**: Common words like "the" get a score of 0 (IDF=0).
2. **High Scores**: Rare terms (e.g., "chased") have higher TF-IDF values.
3. **Dimensionality**: Columns = vocabulary size (7 terms here), rows = 3 documents.

This matrix directly feeds into NLP tasks like document classification or semantic search[14] [15].

⁂

gimme some specific examples on how tfidf matrix might look and how it's reduced in dimensions

Here's a breakdown of TF-IDF matrix examples and dimensionality reduction techniques, with concrete illustrations:

TF-IDF Matrix Example

Corpus

1. **Doc1**: "Jupiter is the largest planet"
2. **Doc2**: "Mars is the fourth planet from the sun"

TF-IDF Matrix

Term	TF (Doc1)	TF (Doc2)	IDF	TF-IDF (Doc1)	TF-IDF (Doc2)
jupiter	1/5	0	log(2/1) ≈ 0.69	0.69 × 1/5 = **0.138**	0
mars	0	1/8	log(2/1) ≈ 0.69	0	0.69 × 1/8 = **0.086**
is	1/5	1/8	log(2/2) = 0	0	0
the	1/5	2/8	log(2/2) = 0	0	0
largest	1/5	0	log(2/1) ≈ 0.69	0.69 × 1/5 = **0.138**	0
planet	1/5	1/8	log(2/2) = 0	0	0
fourth	0	1/8	log(2/1) ≈ 0.69	0	0.69 × 1/8 = **0.086**
from	0	1/8	log(2/1) ≈ 0.69	0	0.69 × 1/8 = **0.086**
sun	0	1/8	log(2/1) ≈ 0.69	0	0.69 × 1/8 = **0.086**

Resulting Sparse Matrix (rows = documents, columns = terms):

```
[[0.138, 0,    0, 0, 0.138, 0, 0,    0,    0   ]
 [0,     0.086,0, 0, 0,     0, 0.086,0.086,0.086]]
```

(Note: Zeros dominate due to sparsity)

Dimensionality Reduction

1. Singular Value Decomposition (SVD)

SVD decomposes the TF-IDF matrix into:

$$H = U\Sigma V^T$$

- **U**: Document-topic matrix (reduced dimensions)
- **Σ**: Diagonal matrix of singular values
- **V**: Term-topic matrix (reduced dimensions)

Example:
Original matrix (2 docs × 9 terms):

```
[[0.138, 0, 0, 0, 0.138, 0, 0, 0, 0]
 [0, 0.086, 0, 0, 0, 0, 0.086, 0.086, 0.086]]
```

After SVD (keeping top 2 singular values):

```
U (2×2): [[-0.707, -0.707],
          [-0.707,  0.707]]

Σ (2×2): [[0.25, 0],
          [0,   0.15]]

V (9×2): [[-0.5, 0],
          [0,   0.5],
          ...]
```

Reduced Matrix: $ U \Sigma $ (2×2) captures latent topics (e.g., "celestial bodies" vs. "positional terms") [16] [17].

2. Truncated SVD (LSA)

A variant of SVD for explicit dimensionality reduction. For a target of **2 dimensions**:

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=2)
reduced_matrix = svd.fit_transform(tfidf_matrix)
```

Output (approximate):

```
[[-0.12, 0.10]
 [ 0.08, 0.15]]
```

(Each row = document in 2D semantic space) [16] [18].

3. Non-Negative Matrix Factorization (NMF)

Factorizes the matrix into non-negative components:

```
from sklearn.decomposition import NMF

nmf = NMF(n_components=2)
W = nmf.fit_transform(tfidf_matrix)  # Document-topic matrix
H = nmf.components_                  # Topic-term matrix
```

Interpretation:

- **W**: Doc1 ≈ [0.7, 0.3] (mostly Topic 1)
- **H**: Topic 1 = ["jupiter", "planet"], Topic 2 = ["mars", "sun"] [17] [19].

Reduced Matrix Comparison

Method	Original Dims	Reduced Dims	Use Case
SVD	2×9	2×2	Semantic topic modeling
NMF	2×9	2×2	Interpretable topic labels
PCA	2×9	2×2	General-purpose reduction

Reduction preserves relationships while filtering noise (e.g., common words like "the" are minimized) [16] [17] [19].

⁂

How exactly does SVD capture Term-topic, and Document-topic realationships

Singular Value Decomposition (SVD) captures **term-topic** and **document-topic relationships** by decomposing a document-term matrix (DTM) into latent semantic components. Here's how it works:

Core Mechanism of SVD

SVD factorizes a matrix $ M $ (e.g., TF-IDF scores) into three components:

$$M = U\Sigma V^T$$

- **$ U $**: Document-topic matrix (rows = documents, columns = topics)
- **$ \Sigma $**: Diagonal matrix of singular values (topic importance)
- **$ V^T $**: Term-topic matrix (rows = topics, columns = terms)

How SVD Captures Relationships

1. Term-Topic Relationships

- **$ V^T $ (Right Singular Vectors)**:
  Each row in $ V^T $ represents a **topic** as a weighted combination of terms.
  - **Example**:
    A topic vector $ T_1 = [0.6 , (climate), , 0.5 , (warming), , 0.1 , (policy)] $ indicates "climate" and "warming" are key terms for that topic.
  - **Interpretation**:
    Terms with higher weights in a topic vector are semantically aligned with that topic.

2. Document-Topic Relationships

- **$ U $ (Left Singular Vectors)**:
  Each row in $ U $ represents a **document** as a weighted combination of topics.
  - **Example**:
    A document vector $ D_1 = [0.8 , (Topic 1), , 0.2 , (Topic 2)] $ indicates the document is mostly about Topic 1.
  - **Interpretation**:
    Documents with similar topic weights are semantically related.

**3. Singular Values ($ \Sigma $)

- **Role**:
  Singular values quantify the **importance** of each topic in explaining variance across documents.
  - **Example**:
    $ \sigma_1 = 14.4 $, $ \sigma_2 = 9.5 $, $ \sigma_3 = 8.4 $ implies Topic 1 explains the most variance in the corpus[20] [21].

Practical Workflow

1. **Input**: TF-IDF matrix (documents × terms)
2. **Decomposition**:
   - $ U $: Maps documents to topics (e.g., "Doc1 is 70% Topic A")
   - $ V^T $: Maps terms to topics (e.g., "Term 'bank' aligns with Topic B")
3. **Dimensionality Reduction**:
   Retain top-$ k $ singular values to filter noise and focus on dominant topics[20] [21].

Example

For a corpus about **climate change** and **economic policy**:

Component	Topic 1 (Climate)	Topic 2 (Economy)
$ V^T $	`climate: 0.6, warming: 0.5`	`tax: 0.7, growth: 0.6`
$ U $	`Doc1: [0.9, 0.1]`	`Doc2: [0.2, 0.8]`

- **Doc1** strongly relates to **Topic 1** (climate terms dominate).
- **Term "tax"** is a key indicator of **Topic 2** (economy).

Limitations and Alternatives

- **SVD Limitations**:
  - Assumes linear relationships between terms/documents[22] [23].
  - Topics are orthogonal, which may not reflect real-world overlaps (e.g., "climate policy").
- **Beyond SVD**:
  Probabilistic models like **LDA** address these by allowing mixed-topic documents[22] [21].

SVD provides a mathematically rigorous way to uncover latent semantic structures but works best when combined with domain-specific validation (e.g., inspecting top terms per topic)[20] [21].

⁂

I mean, how does the process of taking the svd capture asbtract concepts like topics

The **Singular Value Decomposition (SVD)** captures abstract concepts like "topics" by mathematically modeling **latent semantic relationships** between terms and documents. Here's how this works in practice:

Core Mechanism

SVD decomposes a **term-document matrix** (e.g., TF-IDF) into three components:

$$M = U\Sigma V^T$$

- **$ U $**: Document-topic matrix (rows = documents, columns = latent topics)
- **$ \Sigma $**: Singular values (strength of each topic)
- **$ V^T $**: Term-topic matrix (rows = topics, columns = terms)

How SVD Captures Abstract Concepts

1. Latent Semantic Space

- **Term-Topic Relationships ($ V^T $)**:
  Each row in $ V^T $ represents a **topic** as a weighted combination of terms.
  - **Example**:
    A topic vector might assign high weights to terms like `climate`, `warming`, and `emissions`, defining an abstract "climate change" concept[24] [25].
  - **Catchwords**:
    The paper in[24] emphasizes that SVD can identify **Catchwords**—terms that co-occur frequently in a specific topic but rarely in others (e.g., "tax" and "growth" for an "economy" topic).

2. Document-Topic Relationships ($ U $)

- Each row in $ U $ maps a document to its **dominant topics**.
  - **Example**:
    A document vector `[0.8, 0.1]` indicates the document is 80% about Topic 1 and 10% about Topic 2.
  - **Dominant Admixtures**:
    The work in[24] assumes documents have a **dominant topic** (e.g., "climate" dominates over "economy" in a climate policy paper), which SVD isolates by focusing on high singular values[24] [25].

3. Dimensionality Reduction ($ \Sigma $)

- **Singular Values**:
  The diagonal entries in $ \Sigma $ rank topics by their importance in explaining variance.
  - **Example**:
    A singular value of `14.4` for Topic 1 vs. `9.5` for Topic 2 implies Topic 1 is more central to the corpus.
  - **Noise Filtering**:
    Truncating smaller singular values removes noise (e.g., common stopwords like "the") and retains semantically meaningful patterns[25] [26].

Practical Workflow for Topic Modeling

1. **Preprocessing**:
   - **Thresholding**: Filter rare terms (e.g., frequency < 5) to focus on meaningful words[24].
   - **Normalization**: Adjust for word frequency heterogeneity (e.g., log-scaling TF-IDF)[25].
2. **SVD Application**:
   - Compute $ U \Sigma V^T $ on the term-document matrix.

- Retain top-k singular values (e.g., $k=10$ for 10 topics).

3. **Post-Processing**:
   - **Cluster Documents**: Use U to group documents by dominant topics (e.g., k-means clustering)[24] [27].
   - **Label Topics**: Extract high-weight terms from V^T (e.g., "jupiter", "planet" for an astronomy topic)[27] [26].

Example: Identifying "Climate Change" vs. "Economy"

Component	Topic 1 (Climate)	Topic 2 (Economy)
V^T	`climate: 0.6, warming: 0.5`	`tax: 0.7, growth: 0.6`
U	`Doc1: [0.9, 0.1]`	`Doc2: [0.2, 0.8]`

- **Doc1** aligns with **Topic 1** (climate terms dominate).
- **Doc2** aligns with **Topic 2** (economic terms dominate).

Why This Works

- **Co-Occurrence Patterns**: SVD detects terms that frequently appear together (e.g., "tax" and "growth"), defining abstract themes[24] [27].
- **Dominant Topics**: By focusing on high singular values, SVD emphasizes topics that explain the most variance, filtering out noise[25] [26].
- **Interpretability**: The V^T matrix provides human-readable labels for topics via high-weight terms (e.g., "jupiter" for astronomy)[27] [26].

Limitations

- **Linear Assumptions**: SVD cannot model non-linear term relationships (e.g., synonyms like "car" and "automobile")[26].
- **Orthogonality**: Topics are forced to be orthogonal, which may not reflect real-world overlaps (e.g., "climate policy" bridging two topics)[24].

SVD's strength lies in its simplicity and ability to uncover latent structures, but it often requires preprocessing (thresholding/normalization) and post-processing (clustering) to yield meaningful topics[24] [25].

⁂

Linear vs. Nonlinear Term Relationships

Linear and nonlinear term relationships describe how two variables (terms) are mathematically related, often in the context of statistical modeling or machine learning. Here's a detailed explanation:

Linear Term Relationships

Definition:

A linear relationship between terms means that the change in one term is directly proportional to the change in another. Graphically, this relationship forms a straight line.

Mathematical Representation:

The equation for a linear relationship is:

$$y = mx + b$$

Where:

- y: Dependent variable
- x: Independent variable
- m: Slope (rate of change)
- b: Intercept (value of y when $x = 0$) [28] [29].

Example:

- **Scenario**: Minutes played in a basketball game vs. points scored.
- **Interpretation**: Players who play more minutes tend to score more points, with a constant rate of increase.

Graphical Representation:

A straight line where each increase in x corresponds to a predictable increase (or decrease) in y.

Nonlinear Term Relationships

Definition:

A nonlinear relationship occurs when the change in one term does not correspond proportionally to the change in another. This relationship forms curves or other complex shapes on a graph.

Mathematical Representation:

Nonlinear relationships can take many forms, such as:

1. **Quadratic**: $y = ax^2 + bx + c$
2. **Exponential**: $y = ae^{bx}$
3. **Logarithmic**: $y = a\ln(x) + b$
4. **Cosine/Sinusoidal**: $y = a\cos(bx) + c$ [30] [31] [32].

Examples:

1. **Quadratic Relationship**:
   - **Scenario**: Happiness vs. working hours.
   - **Interpretation**: Happiness increases with working hours up to a certain point but decreases beyond that threshold (U-shaped curve).
2. **Exponential Relationship**:
   - **Scenario**: Bamboo growth vs. time.
   - **Interpretation**: Bamboo grows slowly initially but rapidly after reaching maturity.
3. **Logarithmic Relationship**:
   - **Scenario**: Efficiency of smart-home technology over time.
   - **Interpretation**: Efficiency increases rapidly at first but plateaus later.
4. **Cosine Relationship**:
   - **Scenario**: Sound wave frequency vs. time.
   - **Interpretation**: The relationship oscillates in a wave-like pattern[30] [32].

Key Differences

Feature	Linear Relationship	Nonlinear Relationship
Graph Shape	Straight line	Curves, waves, or complex shapes
Predictability	Constant rate of change	Variable rate of change
Examples	Points scored vs. minutes played	Happiness vs. working hours

Applications

1. Linear relationships are simpler and often used for basic regression models.
2. Nonlinear relationships are crucial for more complex phenomena, such as population growth, biological processes, or oscillatory systems.

Understanding these relationships enables better modeling and prediction across diverse fields like economics, biology, and physics[28] [30] [32].

⁂

So am I correct to assume that LSA (svd) essentially models realtionships by examining the wther the frequency of a terms iis linearly correlated with the frequency of apperance of another term?

No, LSA (via SVD) does not explicitly model linear correlations between term frequencies. Instead, it captures **latent semantic relationships** by analyzing **co-occurrence patterns** across documents and terms in a reduced-dimensional space. Here's a detailed breakdown:

How LSA/SVD Works

1. Term-Document Matrix Construction

LSA starts with a **term-document matrix** (e.g., TF-IDF-weighted), where:

- **Rows**: Terms (words)
- **Columns**: Documents
- **Cells**: Weighted frequency (e.g., TF-IDF) of a term in a document

This matrix is **sparse** and high-dimensional, with no explicit modeling of term-term correlations.

2. Singular Value Decomposition (SVD)

SVD decomposes the term-document matrix into three components:

$$A = U\Sigma V^T$$

- **$ U $**: Term-topic matrix (terms × latent topics)
- **$ \Sigma $**: Singular values (topic importance)
- **$ V^T $**: Document-topic matrix (topics × documents)

3. Rank Reduction

By retaining only the largest singular values in $ \Sigma $, LSA reduces dimensionality to:

- Filter noise (e.g., rare or overly common terms)
- Merge terms/documents into **latent topics** (e.g., "climate" and "warming" grouped into a "climate change" topic)

Why It’s Not About Linear Term Correlations

Key Differences

Aspect	Linear Correlation	LSA/SVD
Focus	Direct pairwise relationships (term A ↔ term B)	Global co-occurrence patterns across documents
Mechanism	Pearson correlation, covariance	Latent semantic spaces (linear algebra)
Dimensionality	Explicit term-term relationships	Implicit topics derived from term-document interactions

Example

- **Linear Correlation**: Measures if "car" and "truck" appear together more often than random.
- **LSA/SVD**: Groups "car" and "truck" into a latent "vehicle" topic because they co-occur in similar contexts (e.g., in documents about transportation), even if they never appear in the same document.

What LSA/SVD Captures

1. **Synonymy**: Terms with similar meanings (e.g., "movie" and "film") are mapped to the same latent topic.
2. **Polysemy**: Words with multiple meanings (e.g., "bank" as a riverbank vs. financial institution) are partially disambiguated by their dominant topic associations.
3. **Document Similarity**: Documents sharing overlapping latent topics are deemed semantically related.

Limitations

- **Linear Assumptions**: Relationships are modeled linearly (via matrix factorization), which may miss nonlinear interactions.
- **No Explicit Term-Term Modeling**: LSA does not directly compute pairwise term correlations but infers them indirectly through shared topic membership.

Conclusion

LSA/SVD **does not model linear correlations between term frequencies** but instead **uncovers latent topics** by analyzing how terms co-occur across documents. This approach captures broader semantic relationships (synonymy, polysemy) and reduces noise, making it more powerful than simple term-frequency correlations for tasks like document clustering or search.

⁂

how can I achieve something similar to this but with modern transformer technology like SBERT, modellig topics, topic importance, and term topic realtion like LSA SVD does

To achieve something similar to **LSA (SVD)** but with modern transformer technology like **SBERT**, you can leverage **semantic-driven topic modeling** techniques that use contextual embeddings, clustering, and dimensionality reduction. Here's a step-by-step explanation of how these methods work and how they model topics, topic importance, and term-topic relationships:

Steps to Model Topics Using SBERT and Transformers

1. Document Embedding

- Use **SBERT** (Sentence-BERT) to generate dense vector embeddings for sentences or documents. SBERT produces fixed-length embeddings that capture the semantic meaning of text by leveraging the contextual information encoded in transformers like BERT.
- These embeddings represent documents or terms in a high-dimensional semantic space, where proximity reflects similarity.

Example:
For documents about "climate change" and "economy," SBERT might embed them as:

- Document 1: `[0.23, 0.45, -0.12, ...]` (semantic vector for "climate change")
- Document 2: `[0.15, -0.34, 0.67, ...]` (semantic vector for "economy").

2. Dimensionality Reduction

- High-dimensional embeddings can be challenging for clustering algorithms due to the "curse of dimensionality." Apply dimensionality reduction techniques like:
  - **UMAP** (Uniform Manifold Approximation and Projection): Preserves both global and local structures while reducing dimensions.
  - **PCA** (Principal Component Analysis): Focuses on variance across dimensions.
- Dimensionality reduction ensures better clustering results by simplifying the representation while retaining semantic relationships.

Example:
After UMAP reduction, Document 1 might be reduced to `[0.5, -0.3]` and Document 2 to `[0.2, 0.7]`.

3. Clustering

- Use clustering algorithms like **HDBSCAN** or **K-Means** to group documents or sentences into clusters based on their reduced embeddings.
- Each cluster represents a latent topic inferred from the semantic similarity of its members.

Example:
Documents about "renewable energy" and "carbon emissions" might be grouped into a single cluster representing "climate change."

4. Topic Extraction

- For each cluster:
  - Extract representative words by analyzing the embeddings of terms within the cluster.
  - Compute the average semantic similarity between candidate topic words and all sentences/documents in the cluster using cosine similarity or other measures.
  - Rank words based on their representativeness to form a vocabulary for each topic.

Example:
For a "climate change" cluster:

- Representative words: `["climate", "emissions", "renewable"]`.

5. Topic Importance

- Measure topic importance using metrics like:
  - **Cluster Size**: Larger clusters indicate more prevalent topics.
  - **Semantic Coherence**: Evaluate how semantically similar the documents within a cluster are (e.g., using average cosine similarity).
  - **Singular Values (if using PCA)**: Higher singular values indicate more variance explained by a topic.

How This Mirrors LSA/SVD

Feature	LSA/SVD	Transformer-Based Approach (SBERT)
Term-Topic Relationships	Derived from matrix factorization	Derived from contextual embeddings and clustering
Document-Topic Relationships	Documents mapped to latent topics via U	Documents clustered into topics based on embedding proximity
Dimensionality Reduction	SVD reduces matrix rank	UMAP/PCA reduces embedding dimensions
Topic Importance	Singular values indicate importance	Cluster size/coherence measures relevance

Advantages of Transformer-Based Topic Modeling

1. **Contextual Understanding**:
   Transformers like SBERT encode rich semantic relationships between terms and documents, capturing nuances such as synonymy and polysemy better than traditional methods like LSA or LDA.

2. **Flexibility in Representation**:
   Unlike LSA/SVD, which relies on linear algebraic decomposition of term-document matrices, transformers directly model complex nonlinear relationships using self-attention mechanisms.

3. **Improved Coherence**:
   Transformer-based approaches often produce more coherent topics because embeddings account for word context rather than just frequency.

Implementation Example

Here’s how you could implement this pipeline:

```
from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans
import umap

# Step 1: Generate embeddings with SBERT
model = SentenceTransformer('all-MiniLM-L6-v2')
documents = ["Climate change is accelerating.",
             "Carbon emissions are rising.",
             "Economic growth depends on renewable energy."]
embeddings = model.encode(documents)

# Step 2: Dimensionality reduction with UMAP
reducer = umap.UMAP(n_components=2)
reduced_embeddings = reducer.fit_transform(embeddings)

# Step 3: Clustering with K-Means
kmeans = KMeans(n_clusters=2)
clusters = kmeans.fit_predict(reduced_embeddings)

# Step 4: Extract representative terms (manual or automated ranking)
print("Cluster assignments:", clusters)
```

Conclusion

Using modern transformer-based approaches like SBERT combined with dimensionality reduction (UMAP) and clustering algorithms (HDBSCAN/K-Means), you can achieve topic modeling that mirrors the functionality of LSA/SVD while benefiting from richer contextual embeddings and non-linear semantic relationships. This method provides more coherent topics, better captures term-topic relationships, and adapts well to complex data distributions.

⁂

Here are my goals - produce similar term-topic, term-document, and topic extrapolation ability using SBERT whilst maintaiing similar performance metrics as LSA

To achieve **term-topic relationships**, **document-topic assignments**, and **topic extrapolation** using SBERT while maintaining performance metrics comparable to LSA, follow this structured approach:

Solution: SBERT-Based Topic Modeling with BERTopic

Key Components

1. **SBERT Embeddings**: Leverage pre-trained transformer models (e.g., `all-MiniLM-L6-v2`) to generate dense, context-aware document embeddings.
2. **Dimensionality Reduction**: Use UMAP to reduce embeddings to a lower-dimensional space while preserving semantic relationships.
3. **Clustering**: Apply HDBSCAN to group documents into topics based on embedding proximity.
4. **c-TF-IDF**: Extract term importance per topic using class-based TF-IDF for interpretable term-topic relationships.

Workflow

1. **Document Embedding**
   Use SBERT to convert documents into fixed-length vectors.
   ```
   from sentence_transformers import SentenceTransformer
   model = SentenceTransformer('all-MiniLM-L6-v2')
   embeddings = model.encode(documents)
   ```
2. **Dimensionality Reduction (UMAP)**
   Reduce embedding dimensions to improve clustering efficiency.
   ```
   from umap import UMAP
   umap_model = UMAP(n_components=5, random_state=42)
   reduced_embeddings = umap_model.fit_transform(embeddings)
   ```
3. **Clustering (HDBSCAN)**
   Group documents into topics while handling noise.

```
from hdbscan import HDBSCAN
cluster_model = HDBSCAN(min_cluster_size=10)
topics = cluster_model.fit_predict(reduced_embeddings)
```

4. **Term-Topic Relationships (c-TF-IDF)**
   Compute term importance for each topic using class-based TF-IDF.

```
from sklearn.feature_extraction.text import CountVectorizer
from bertopic.vectorizers import ClassTfidfTransformer

# Create document-term matrix
vectorizer = CountVectorizer()
doc_term_matrix = vectorizer.fit_transform(documents)

# Apply c-TF-IDF
ctfidf_model = ClassTfidfTransformer()
topic_term_matrix = ctfidf_model.fit_transform(doc_term_matrix, topics)
```

5. **Topic Representation**
   Extract top terms per topic and calculate topic importance.

```
# Get top 10 terms per topic
terms = vectorizer.get_feature_names_out()
top_terms_per_topic = {
    topic_idx: [terms[i] for i in topic_term_matrix[topic_idx].argsort()[-10:]]
    for topic_idx in set(topics)
}

# Topic importance (cluster size)
topic_sizes = {topic: (topics == topic).sum() for topic in set(topics)}
```

Performance Metrics & Validation

1. **Topic Coherence**
   Use metrics like `NPMI` or `CV` to evaluate semantic consistency of topics.

```
from gensim.models.coherencemodel import CoherenceModel
coherence_model = CoherenceModel(topics=top_terms_per_topic, texts=documents, diction
coherence_score = coherence_model.get_coherence()
```

2. **Document Clustering Quality**
   - **Silhouette Score**: Measures intra-cluster similarity vs. inter-cluster dissimilarity.
   - **Calinski-Harabasz Index**: Evaluates cluster separation.
3. **Extrapolation to New Documents**
   Assign topics to unseen documents by:
   - Generating SBERT embeddings for the new document.
   - Reducing dimensions with the pre-trained UMAP model.

- Finding the nearest cluster centroid in HDBSCAN.

Comparison with LSA

Feature	LSA	SBERT + BERTopic
Term-Topic Relationships	Linear (SVD on TF-IDF)	Nonlinear (c-TF-IDF on clusters)
Document-Topic Mapping	SVD-derived document-topic matrix	Clustering on reduced embeddings
Context Awareness	Limited (bag-of-words)	High (contextual embeddings)
Scalability	Better for long documents	Limited by transformer context
Interpretability	Direct via V^T matrix	Indirect via c-TF-IDF and clusters

Implementation Example

```
from bertopic import BERTopic

# Initialize and fit model
topic_model = BERTopic(embedding_model="all-MiniLM-L6-v2", umap_model=umap_model, hdbscar
topics, probs = topic_model.fit_transform(documents)

# Extract term-topic relationships
topic_terms = topic_model.get_topics()
```

Advantages Over LSA

1. **Semantic Richness**: SBERT captures polysemy and synonymy better than LSA's bag-of-words approach.
2. **Noise Robustness**: HDBSCAN handles outliers, while LSA requires manual noise filtering.
3. **Interpretability**: c-TF-IDF provides clear term importance scores per topic.

Limitations

- **Long Documents**: SBERT has a 512-token limit (use sliding windows or extractive summarization).
- **Computational Cost**: Transformers are slower than SVD for large corpora.

By combining SBERT embeddings with BERTopic's clustering and c-TF-IDF, you achieve **comparable performance to LSA** while leveraging modern transformer capabilities for richer semantic modeling.

⁂

what would the UMAP lower dimnension representation of the relationship look like

The **UMAP lower-dimensional representation** of term/document relationships typically appears as **clusters of points in 2D or 3D space**, where proximity reflects semantic similarity. Below is a detailed breakdown of what this visualization looks like and how to interpret it:

UMAP Output Structure

Key Characteristics

- **Axes**: Abstract dimensions (no direct interpretability like PCA's variance-based axes).
- **Points**: Represent terms or documents, depending on the input matrix.
- **Clusters**: Groups of points with similar semantic meaning (e.g., "climate change" terms cluster together).
- **Outliers**: Isolated points representing rare terms or documents.

Example Visualization

Input Data

- **Documents**:
  1. *"Climate change accelerates global warming."*
  2. *"Renewable energy reduces carbon emissions."*
  3. *"Economic growth depends on fiscal policy."*
- **Terms**: `["climate", "warming", "renewable", "carbon", "economic", "growth"]`

UMAP Projection

A 2D UMAP plot might look like this:
UMAP Example
(Hypothetical coordinates for illustration)

Point	X (UMAP1)	Y (UMAP2)	Label
1	0.5	0.8	climate
2	0.6	0.7	warming
3	-0.3	0.4	renewable
4	-0.2	0.5	carbon
5	-0.7	-0.6	economic
6	-0.8	-0.5	growth

Interpreting the Visualization

Cluster Analysis

1. **Semantic Clusters**:
   - **Top-right**: Terms like `climate` and `warming` cluster tightly, reflecting their contextual similarity.
   - **Mid-left**: `renewable` and `carbon` form a smaller cluster linked to energy/environment topics.
   - **Bottom-left**: `economic` and `growth` group together, representing fiscal themes.
2. **Global vs. Local Structure**:
   - **Local**: Tight clusters preserve fine-grained relationships (e.g., synonyms like "climate" and "warming").
   - **Global**: Distances between clusters reflect broader topic relationships (e.g., environment vs. economy).
3. **Outliers**:
   - A term like `policy` might appear isolated if it lacks strong co-occurrence patterns with other terms.

Comparison with Other Methods

Feature	UMAP	PCA	t-SNE
Structure	Balances local/global	Global linear trends	Focuses on local
Axes Interpretability	Abstract (no clear meaning)	Variance-based	Abstract
Scalability	Efficient for large datasets	Highly scalable	Slower for large data

Practical Implementation

```
import umap
import matplotlib.pyplot as plt

# Example: Reduce 384D SBERT embeddings to 2D
reducer = umap.UMAP(n_components=2, random_state=42)
embeddings_2d = reducer.fit_transform(sbert_embeddings)

# Plot
plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1], c=labels)
plt.title("UMAP Projection of Document Embeddings")
plt.show()
```

Key Takeaways

1. **Clusters = Topics**: Dense groupings indicate semantically related terms/documents.
2. **Distance = Similarity**: Closer points share stronger contextual relationships.
3. **Nonlinearity**: UMAP captures complex relationships that linear methods (e.g., PCA) miss.

For reproducibility, adjust UMAP hyperparameters like `n_neighbors` (local vs. global balance) and `min_dist` (cluster tightness) [33] [34] .

⁂

for the LSA method in convokit, where does the the context (forward context and backward context fit in to the process) are the terms from the forward and backward context computed the same way in the same matrix or what? What are the assumptions for number of documents, because the tf-idf represents terms in terms of the whole corpus of documents, but the context seems to imply we only look at a few

In ConvoKit's **Expected Context Framework** using **LSA**, the **forward context** and **backward context** are handled as **separate semantic spaces** derived from distinct document-term matrices. Here's how the process works and how contexts fit into LSA:

Context Handling in LSA

1. Context Definitions

- **Forward Context**: Utterances that follow a given utterance (e.g., replies).
- **Backward Context**: Utterances that precede a given utterance (e.g., predecessors).

2. Separate Term-Context Matrices

- **Term-Forward Context Matrix**:
  Each row represents a **term**, and columns represent **forward context utterances**.
  - **Cell values**: TF-IDF scores of terms in forward contexts.
- **Term-Backward Context Matrix**:
  Each row represents a **term**, and columns represent **backward context utterances**.
  - **Cell values**: TF-IDF scores of terms in backward contexts.

These matrices are **computed independently** and represent distinct semantic spaces.

Assumptions and Process

Key Assumptions

1. **Context as Documents**:
   Forward/backward contexts are treated as pseudo-"documents" for TF-IDF computation.
   - Example: If analyzing a conversation thread, the forward context of an utterance might include its immediate replies.
2. **Term Representation**:
   Terms are represented in two ways:
   - **Global Corpus TF-IDF**: Standard term importance across the entire corpus.
   - **Context-Specific TF-IDF**: Term importance within forward/backward contexts.
3. **Dimensionality**:
   LSA (via SVD) reduces the dimensionality of both matrices to capture **latent semantic relationships** specific to each context type.

Workflow Summary

1. **Context Extraction**:
   - For each utterance, extract its forward/backward context utterances.
2. **Matrix Construction**:
   - Build separate term-context matrices for forward and backward contexts.
3. **LSA Application**:
   - Apply SVD to each matrix to derive:
     - **Term-topic relationships** (via V^T) for forward/backward contexts.
     - **Context-topic relationships** (via U).
4. **Comparison**:
   - Compare term representations across forward/backward contexts to analyze how terms shift in meaning based on conversational flow.

Document Count Considerations

- **Global Corpus**:
  The standard TF-IDF matrix considers the **entire corpus** as the document set, ensuring term importance is weighted by global frequency.
- **Local Contexts**:
  Forward/backward contexts are subsets of the corpus. Their matrices focus on **local co-occurrence patterns** within specific conversational sequences.

- **Scalability**:
  The method assumes sufficient context utterances exist to form meaningful matrices. Sparse contexts (e.g., short conversations) may require smoothing or alternative representations.

Example

For a conversation:

```
A: "Climate change is accelerating."
B: "Yes, carbon emissions are rising."
C: "Economic growth complicates this."
```

- **Backward Context for C**: Utterances A and B.
- **Forward Context for A**: Utterance B.

LSA would:

1. Compute a term-backward context matrix for C using A and B as "documents."
2. Compute a term-forward context matrix for A using B as a "document."
3. Derive separate topic models for each context type.

Key Takeaways

- **Separate Treatment**: Forward/backward contexts are modeled in distinct matrices and semantic spaces.
- **Local vs. Global**: Context matrices focus on local sequences, while global TF-IDF captures corpus-wide term importance.
- **LSA Application**: SVD is applied independently to each context matrix to derive latent topics.

This approach allows ConvoKit to model how terms behave differently depending on their conversational position (e.g., preceding vs. following utterances).

⁂

but clustering goes over teh entire corpus no?

Yes, you're correct that **clustering** in the Expected Context Framework (ECF) of ConvoKit **operates over the entire corpus**, but the **context representation** (forward/backward) is computed **locally** for each utterance. Here's how these two aspects interact and where the assumptions about document count come into play:

Clustering Over the Entire Corpus

1. **Global Topic Discovery**:
   Clustering (e.g., K-Means or HDBSCAN) groups **all utterances** in the corpus into topics based on their **latent semantic representations** (e.g., LSA-reduced vectors). This is a **global process** that identifies overarching themes in the entire dataset.
2. **Term-Topic Relationships**:
   Terms are mapped to topics using the **global term-topic matrix** (from SVD), which captures how terms contribute to topics across the entire corpus.

Context Representation: Local Computation

While clustering is global, the **forward/backward context** for each utterance is computed **locally**:

1. **Forward Context**:
   For a given utterance, its forward context includes utterances that **follow it** (e.g., replies in a conversation thread).
2. **Backward Context**:
   For a given utterance, its backward context includes utterances that **precede it** (e.g., prior messages in a thread).

How Context Fits Into LSA

- **Separate Term-Context Matrices**:
  ConvoKit constructs **two distinct term-context matrices**:
  - **Term-Forward Context Matrix**: Terms × forward context utterances.
  - **Term-Backward Context Matrix**: Terms × backward context utterances.
    These matrices are **local** to each utterance's context but use **global IDF weights** (computed over the entire corpus).
- **LSA Application**:
  SVD is applied **separately** to each term-context matrix to derive **latent semantic spaces** for forward and backward contexts. These spaces are then used to compute deviations between expected and observed contexts.

Assumptions About Document Count

1. **Global Corpus**:
   - **TF-IDF**: The IDF component is computed over the **entire corpus**, assuming sufficient documents (utterances) exist to reliably estimate term rarity.
   - **Clustering**: Requires enough utterances to form meaningful clusters (no strict minimum, but sparse corpora may yield noisy topics).
2. **Local Context**:

- Each utterance’s forward/backward context is a **subset of the corpus** (e.g., replies to a specific message).
- **No Minimum Context Size**: The framework can handle cases where an utterance has few or no context utterances (e.g., the first/last message in a thread).

Example Workflow

For a corpus of parliamentary debates:

1. **Global Clustering**:
   - All speeches are clustered into topics like "climate policy" or "economic reform" using LSA.
2. **Local Context**:
   - For a speech about "carbon taxes," its **backward context** might include prior speeches on "energy subsidies," and its **forward context** might include replies debating "economic impact."
3. **Deviation Analysis**:
   - Compute how much the speech deviates from the expected semantic patterns of its forward/backward contexts (using the term-context matrices).

Key Differences

Aspect	Global Clustering	Local Context
Scope	Entire corpus	Subset of utterances (per unit)
Matrix	Term-document (corpus)	Term-context (forward/backward)
Purpose	Topic discovery	Contextual deviation analysis

Why This Works

- **Global IDF**: Ensures term importance is weighted by corpus-wide rarity, even when computing local contexts.
- **Local SVD**: Captures how terms behave in specific conversational positions (e.g., replies vs. initial messages).
- **Clustering**: Groups utterances into topics based on **global semantic patterns**, while context analysis explains **local deviations**.

Limitations

- **Sparse Contexts**: If most utterances lack sufficient forward/backward context (e.g., short conversations), the term-context matrices may be too sparse for meaningful analysis.
- **Global-Local Tradeoff**: Clustering optimizes for global patterns, which might overlook niche but important local dynamics.

In summary, clustering operates globally, while context analysis is local, with both relying on the same underlying corpus-wide IDF weights. This hybrid approach allows ConvoKit to model both broad themes and fine-grained contextual dynamics.

1. https://github.com/CornellNLP/ConvoKit/blob/master/convokit/expected_context_framework/demos/parliament_demo.ipynb
2. https://builtin.com/articles/tf-idf
3. https://letsdatascience.com/tf-idf/
4. https://datagraphi.com/blog/post/2021/9/24/comparing-performance-of-a-modern-nlp-framework-bert-vs-a-classical-approach-tf-idf-for-document-classification-with-simple-and-easy-to-understand-code
5. https://stackoverflow.com/questions/62268459/accuracy-with-tf-idf-and-non-tf-idf-features
6. https://stackoverflow.com/questions/75763642/tf-idf-value-is-not-matching-the-output-of-tfidfvectorizer
7. https://pmc.ncbi.nlm.nih.gov/articles/PMC9985492/
8. https://convokit.cornell.edu/documentation/expected_context_model.html
9. https://aclanthology.org/2020.sigdial-1.8/
10. https://tisjune.github.io/papers/phd-thesis.pdf
11. https://convokit.cornell.edu/documentation/examples.html
12. https://convokit.cornell.edu/documentation/expected_context_model.html
13. https://www.cs.cornell.edu/~cristian/ConvoKit_Demo_Paper_files/convokit-demo-paper.pdf
14. https://www.kdnuggets.com/2022/09/convert-text-documents-tfidf-matrix-tfidfvectorizer.html
15. https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/
16. https://conferences.tiu.edu.iq/iec/wp-content/uploads/2018/02/IEC2017_A16_Improving-TF-IDF-with-Singular-Value-Decomposition-SVD-for-Feature-Extraction-on-Twitter.pdf
17. https://stackoverflow.com/questions/61274499/reduce-dimension-of-word-vectors-from-tfidfvectorizer-countvectorizer
18. https://www.reddit.com/r/MachineLearning/comments/30xo25/how_to_reduce_dimension_for_tfidf_bow_vector/
19. https://www.linkedin.com/advice/0/what-some-techniques-reduce-dimensionality-text-data
20. https://www.casact.org/sites/default/files/2022-07/M2_Creating_Text_Topics_for_Modeling.pdf
21. https://www.datacamp.com/tutorial/what-is-topic-modeling
22. https://www.ias.edu/sites/default/files/math/csdm/11-12/amoitra_learning_topic_models_going_beyond_svd.pdf
23. https://math.stackexchange.com/questions/3942807/how-is-the-svd-of-an-item-document-matrix-as-component-equal-to-the-eigenve

24. https://trapitbansal.com/papers/nips_final.pdf
25. https://arxiv.org/pdf/1704.07016.pdf
26. https://peterbloem.nl/blog/pca-4
27. http://www.cs.put.poznan.pl/jstefanowski/pub/lingo.pdf
28. https://www.investopedia.com/terms/l/linearrelationship.asp
29. https://sites.utexas.edu/sos/guided/inferential/numeric/bivariate/cor/
30. https://www.statology.org/nonlinear-relationship-examples/
31. https://www3.nd.edu/~rwilliam/stats2/l61.pdf
32. https://multivariada.netlify.app/docs/lecturas/Darlington&Hayes_13nonlinear.pdf
33. https://umap-learn.readthedocs.io
34. https://www.scikit-yb.org/en/latest/api/text/umap_vis.html