TEE

1) Word vectorisation techniques:

Word vectorization techniques convert words into numerical vectors, allowing machines to process and understand textual data. Two common word vectorization techniques are Bag of Words (BoW) and Word Embeddings. Let's briefly explain each:

1. Bag of Words (BoW):

Description:

- Bag of Words is a simple and widely used technique that represents a document as an unordered set of words, ignoring grammar and word order but considering word frequency.
- The basic idea is to create a vocabulary from all unique words in a corpus and represent each document as a vector, where each element corresponds to the frequency of a word in the document.
- The resulting vectors are high-dimensional and sparse, as each dimension corresponds to a unique word in the vocabulary.

Steps:

1. **Tokenization:**
    - Break the text into individual words or tokens.
2. **Vocabulary Building:**
    - Create a vocabulary by identifying all unique words in the corpus.
3. **Vectorization:**
    - Represent each document as a vector with dimensions equal to the size of the vocabulary. Each element in the vector corresponds to the frequency of a word in the document.

Pros:

- Simple and easy to implement.
- Captures term frequency information.

Cons:

- Ignores word order and semantic relationships.
- Results in high-dimensional, sparse vectors.

2. Word Embeddings:

Description:

- Word Embeddings, such as Word2Vec, GloVe, and FastText, represent words as dense vectors of fixed dimensions in a continuous vector space.
- These vectors are learned by training neural network models on large corpora, capturing semantic relationships and contextual information.
- Word embeddings have the property that words with similar meanings are located close to each other in the vector space.

Steps:

1. **Training Model:**
    - Train a neural network model (e.g., Word2Vec model) on a large corpus to predict the context of words.
2. **Vector Representation:**
    - Extract the learned vector representations for each word from the trained model.
3. **Vector Space:**
    - Words with similar meanings or contexts are closer in the vector space.

Pros:

- Captures semantic relationships and word context.
- Dense vectors with lower dimensionality compared to BoW.

Cons:

- Requires a large amount of training data.
- Computationally more intensive than simple BoW approaches.

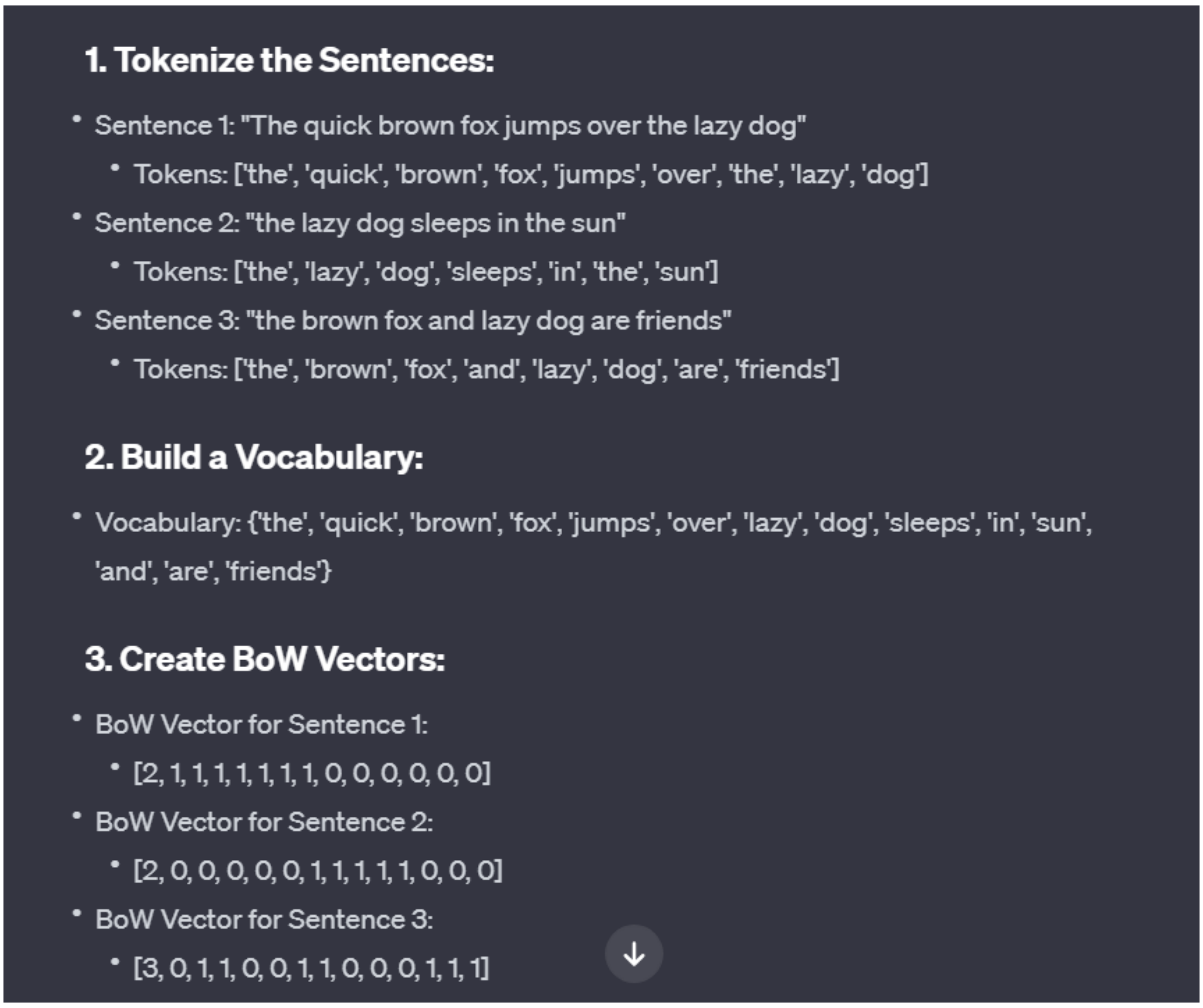
Example (Word2Vec):

pythonCopy code

from gensim.models import Word2Vec sentences = [['word', 'embeddings', 'example'], ['another', 'example', 'word']] model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4) vector = model.wv['word'] # Retrieve the vector representation for the word 'word'

In summary, Bag of Words is a simple and straightforward technique based on word frequency, while Word Embeddings capture semantic relationships and word context in dense vector representations, making them more sophisticated but computationally intensive. The choice between these techniques depends on the specific requirements of the task at hand.

numerical



1. Tokenize the Sentences:

- Sentence 1: "The quick brown fox jumps over the lazy dog"
  - Tokens: ['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
- Sentence 2: "the lazy dog sleeps in the sun"
  - Tokens: ['the', 'lazy', 'dog', 'sleeps', 'in', 'the', 'sun']
- Sentence 3: "the brown fox and lazy dog are friends"
  - Tokens: ['the', 'brown', 'fox', 'and', 'lazy', 'dog', 'are', 'friends']

2. Build a Vocabulary:

- Vocabulary: {'the', 'quick', 'brown', 'fox', 'jumps', 'over', 'lazy', 'dog', 'sleeps', 'in', 'sun', 'and', 'are', 'friends'}

3. Create BoW Vectors:

- BoW Vector for Sentence 1:
  - [2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
- BoW Vector for Sentence 2:
  - [2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0]
- BoW Vector for Sentence 3:
  - [3, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1]

2) sentiment analysis wala

Dealing with class imbalance in a sentiment analysis dataset (with three classes: positive, negative, and neutral) is crucial for developing a robust and accurate model. Here are several strategies to address class imbalance:

1. **Data Augmentation:**
    - Augment the minority classes by creating additional samples through techniques such as:
    - Synonym Replacement:
    - Description: Replace some words in the text with their synonyms while keeping the overall meaning of the sentence intact.
    - Implementation: Use a thesaurus or lexical database to find synonyms for words in the text and substitute them.
    - Back Translation:
    - Description: Translate the text to another language and then back to the original language, introducing variations in the text.
    - Implementation: Use machine translation services (e.g., Google Translate) to translate the text to a different language and then translate it back.
    - Text Paraphrasing:
    - Description: Reword or rephrase sentences while preserving their original meaning.
    - Implementation: Use paraphrasing tools or techniques like sentence embeddings to generate alternative versions of the input text.
    - Random Insertion:
    - Description: Randomly insert new words into the existing text to create variations.
    - Implementation: Select a position in the sentence and insert a randomly chosen word.
    - Random Deletion:

- Description: Randomly remove words from the text to create a shorter version.
- Implementation: With a certain probability, delete each word from the sentence.
- Random Swap:
- Description: Randomly swap the positions of two words in the sentence.
- Implementation: Select two positions in the sentence and swap the corresponding words.
- Contrastive Learning:
- Description: Train the model to distinguish between augmented and original samples.
- Implementation: Create augmented versions of the original data and train the model to correctly classify whether a sample is original or augmented.
-

2. **Resampling Techniques:**
    - **Oversampling:** Increase the number of instances in the minority classes.
    - **Undersampling:** Decrease the number of instances in the majority class.
    - Use techniques like SMOTE (Synthetic Minority Over-sampling Technique) for oversampling.
3. **Class-Weighted Loss:**
    - Adjust the loss function to penalize misclassifications of minority classes more than majority classes. Many machine learning libraries, including scikit-learn and TensorFlow, provide options to assign different weights to classes.
4. **Ensemble Methods:**

        - Utilize ensemble methods like bagging or boosting with techniques such as Random Forests or AdaBoost to combine predictions from multiple models.

5. **Evaluation Metrics:**
    - Choose appropriate evaluation metrics that account for class imbalance, such as F1 score, precision-recall curves, or area under the precision-recall curve.

Sentiment Analysis Libraries:

There are several libraries that can be used to generate sentiment scores for raw text. One popular library is the **Natural Language Toolkit (NLTK)** and **TextBlob**. TextBlob is built on top of NLTK and provides a simple API for common natural language processing (NLP) tasks, including sentiment analysis.

TextBlob for Sentiment Analysis:

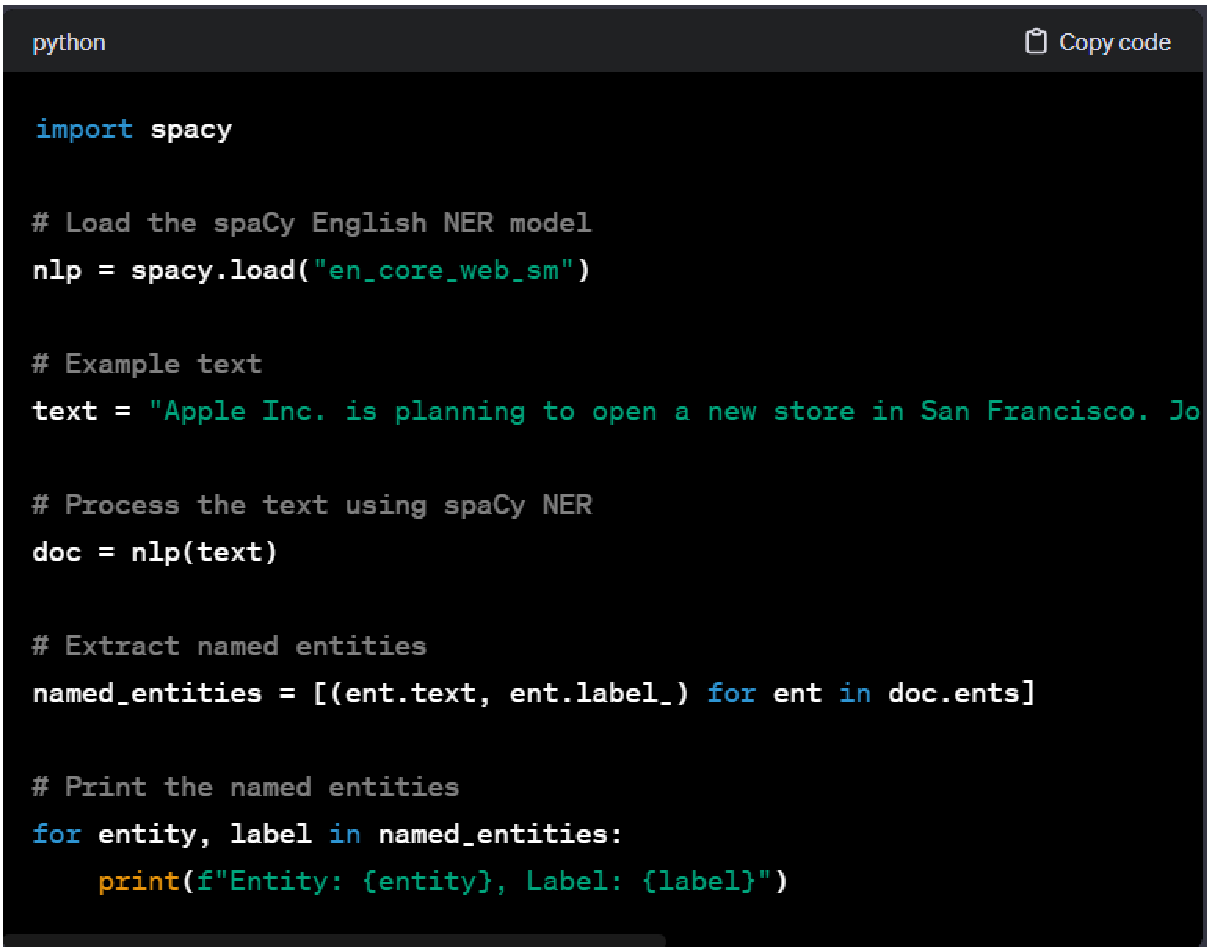
Pros:

- **Simplicity:** TextBlob is easy to use and provides a simple interface for sentiment analysis without requiring extensive setup.
- **Pre-trained Model:** TextBlob comes with a pre-trained sentiment analysis model.
- **Subjectivity Analysis:** TextBlob also provides subjectivity analysis, allowing you to assess the subjectivity of the text.

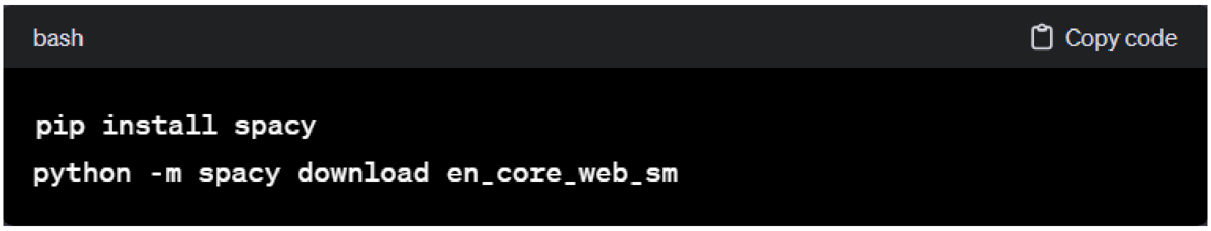
Cons:

- **Limited Customization:** While TextBlob is easy to use, it might be less suitable for highly specialized or domain-specific sentiment analysis tasks.

3) NER wala



```python
import spacy

# Load the spaCy English NER model
nlp = spacy.load("en_core_web_sm")

# Example text
text = "Apple Inc. is planning to open a new store in San Francisco. Jo

# Process the text using spaCy NER
doc = nlp(text)

# Extract named entities
named_entities = [(ent.text, ent.label_) for ent in doc.ents]

# Print the named entities
for entity, label in named_entities:
    print(f"Entity: {entity}, Label: {label}")
```



```bash
pip install spacy
python -m spacy download en_core_web_sm
```

4) E-comm wala

Keyword extraction algorithms are often unsupervised in nature because they don't require labeled training data to identify keywords in a given text. Unsupervised methods are based on inherent patterns, frequencies, or relationships within the text itself. In contrast, supervised methods would need training examples where each word or phrase is labeled as a keyword or non-keyword, which can be impractical or unavailable for many applications.

In the context of e-commerce product descriptions, keyword extraction techniques can be highly valuable for several purposes:

1. **SEO Optimization:**

- Identify and extract relevant keywords from product descriptions to enhance search engine optimization (SEO). Including relevant keywords in product descriptions can improve the visibility of products in search engine results.

2. **Content Summarization:**
    - Extracting keywords can contribute to content summarization by highlighting the most important terms in a product description. This can be useful for generating concise summaries or snippets for display on search engine result pages.
3. **Product Categorization:**
    - Analyzing keywords can aid in automatically categorizing products. Keywords often reflect the key attributes, features, or categories associated with products, helping in the organization and classification of items.
4. **Customer Insights:**
    - Analyzing frequently occurring keywords can provide insights into customer preferences and the language they use. This information can be valuable for understanding market trends, customer needs, and the effectiveness of marketing strategies.
5. **Competitor Analysis:**
    - Comparing keyword patterns across different product descriptions and brands can help in competitive analysis. It allows businesses to understand how competitors are describing similar products and adjust their strategies accordingly.

Techniques for Keyword Extraction in E-commerce Product Descriptions:

1. **TF-IDF (Term Frequency-Inverse Document Frequency):**
    - Calculate the TF-IDF scores for words in product descriptions. High TF-IDF scores indicate words that are important in a specific product description but not common across the entire dataset.
2. **Rapid Automatic Keyword Extraction (RAKE):**

- RAKE is an unsupervised algorithm that identifies keywords based on word frequency and co-occurrence. It can be applied to product descriptions to extract key terms.

3. **TextRank:**
    - TextRank is an unsupervised graph-based algorithm that assigns importance scores to words in a text based on their relationships with other words. It can be used to identify keywords by considering the structure of the text.
4. **Noun Phrase Extraction:**
    - Focus on extracting noun phrases, which often represent key entities and attributes in product descriptions. This can be achieved using part-of-speech tagging and grammatical analysis.
5. **Named Entity Recognition (NER):**
    - NER can identify specific entities such as product names, brands, and attributes within product descriptions, serving as a form of keyword extraction.
6. **Topic Modeling:**
    - Techniques like Latent Dirichlet Allocation (LDA) can be used to identify topics within product descriptions, and the most frequent terms within each topic can be considered as keywords.

Keyword extraction algorithms, including unsupervised methods like RAKE (Rapid Automatic Keyword Extraction) or TF-IDF (Term Frequency-Inverse Document Frequency), are typically unsupervised because they do not require labeled training data to identify keywords. Instead, these algorithms rely on statistical or frequency-based approaches to extract important terms or phrases from the text.

Applying Keyword Extraction for E-commerce Product Descriptions:

1. **Search Engine Optimization (SEO):**
    - **Benefit for Retailers:**

        - By extracting relevant keywords from product descriptions, retailers can optimize their product listings for search engines. This helps in improving the search engine ranking of the products when users search for specific terms related to those products.
    - **Example:**
        - If a retailer sells "wireless noise-canceling headphones," using keyword extraction, relevant terms like "wireless," "noise-canceling," and "headphones" can be identified. These keywords can then be strategically incorporated into the product title, description, and metadata to enhance SEO.

2. **Product Discoverability:**
    - **Benefit for Customers:**
        - Customers often use search engines on e-commerce websites to find products. Keyword extraction techniques can help improve the accuracy of search results by identifying and emphasizing the most relevant keywords in product descriptions.
    - **Example:**
        - Suppose a customer is looking for a "water-resistant backpack." By extracting keywords like "water-resistant" and "backpack" from product descriptions, the search engine can better match customer queries to relevant products. This enhances the discoverability of products meeting specific criteria.
3. **Content Summarization:**
    - **Benefit for Retailers and Customers:**
        - Keyword extraction can be used to generate concise summaries of product descriptions, making it easier for customers to quickly understand the key features of a product.
        - **Example:**

            - For a smartphone product description, keywords like "5G," "high-resolution camera," and "long battery life" can be extracted. These keywords can then be used to create a brief summary that highlights the most important aspects of the phone.

4. **Ad Copy Generation:**
    - **Benefit for Retailers:**
        - Extracted keywords can be used to generate effective ad copy for online advertising campaigns, improving the relevance of ads and attracting potential customers.
        - **Example:**
            - If the keywords extracted from a fashion product description include "trendy," "comfortable," and "affordable," these can be incorporated into ad copy to attract customers looking for fashionable and budget-friendly items.

7) coherence score and ___

When evaluating topic models, two commonly used metrics are perplexity score and coherence score. Let's briefly discuss each metric and then compare the results of topic models generated using Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA), and Non-Negative Matrix Factorization (NMF).

1. Perplexity Score:

- **Description:**
    - Perplexity is a measure of how well a probabilistic model predicts a sample.
    - Lower perplexity indicates a better model.
    - It is often used in the context of probabilistic topic models.
- **Application:**
    - In the case of topic modeling, lower perplexity suggests that the model is better at predicting the observed data.
- **Usage:**

    - The lower the perplexity, the better the model. However, it should be used cautiously as it may not always align with human judgment.

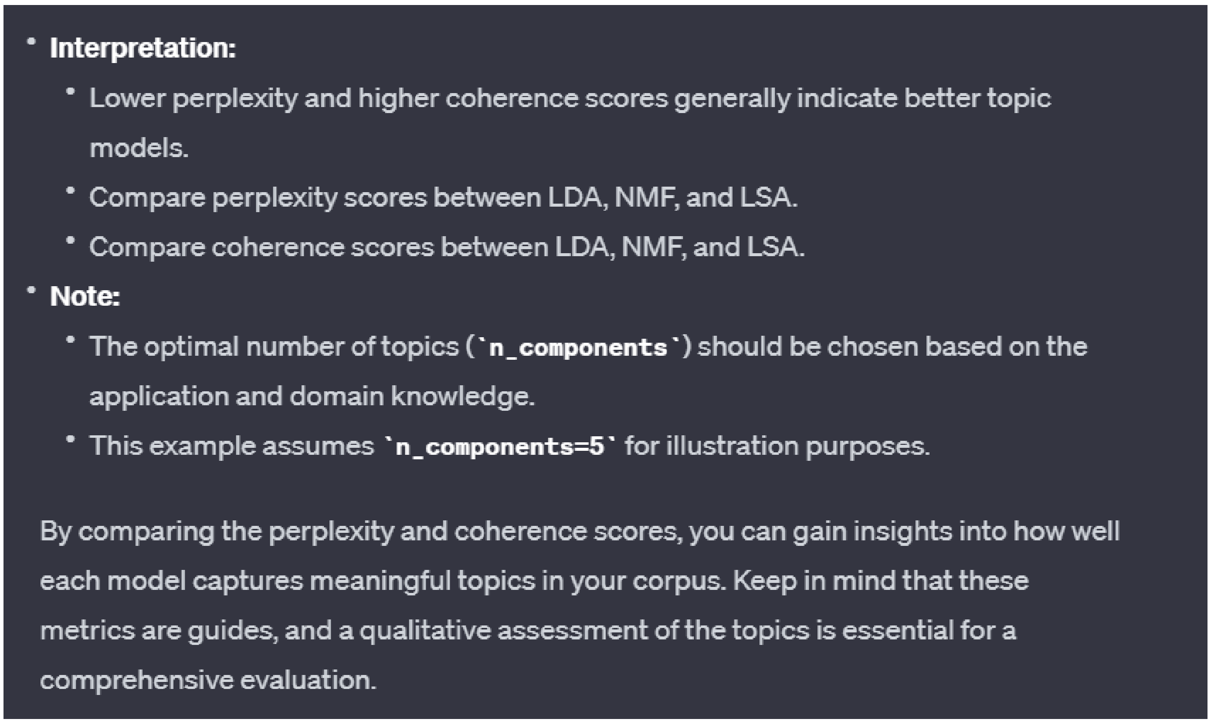
2. Coherence Score:

- **Description:**
    - Coherence measures the semantic similarity between high-scoring words within a topic.
    - Higher coherence scores indicate better-defined topics.
- **Application:**
    - It helps assess the interpretability and meaningfulness of topics.
- **Usage:**
    - A higher coherence score suggests that the topics generated by the model are more interpretable.

Code:

```
from sklearn.decomposition import LatentDirichletAllocation, NMF, TruncatedSVD

from gensim.models import CoherenceModel

from sklearn.feature_extraction.text import CountVectorizer


# Assuming 'documents' is a list of preprocessed documents


# Vectorize the text data

vectorizer = CountVectorizer()

X = vectorizer.fit_transform(documents)


# LDA

lda_model = LatentDirichletAllocation(n_components=5, random_state=42)

lda_topics = lda_model.fit_transform(X)
```

```
# NMF
nmf_model = NMF(n_components=5, random_state=42)
nmf_topics = nmf_model.fit_transform(X)

# LSA (Truncated SVD)
lsa_model = TruncatedSVD(n_components=5, random_state=42)
lsa_topics = lsa_model.fit_transform(X)

# Compute coherence scores
def compute_coherence(model, vectorizer, texts):
    topics = [[vectorizer.get_feature_names_out()[i] for i in topic.argsort()[:-11:-
1]] for topic in model.components_]
    coherence_model = CoherenceModel(topics=topics, texts=texts,
dictionary=vectorizer.get_feature_names_out(), coherence='c_v')
    coherence_score = coherence_model.get_coherence()
    return coherence_score

lda_coherence = compute_coherence(lda_model, vectorizer, documents)
nmf_coherence = compute_coherence(nmf_model, vectorizer, documents)
lsa_coherence = compute_coherence(lsa_model, vectorizer, documents)

# Print results
print("Perplexity Scores:")
print("LDA Perplexity:", lda_model.perplexity(X))
print("NMF Perplexity:", nmf_model.reconstruction_err_)
print("LSA Perplexity:", lsa_model.explained_variance_ratio_.sum())
print("\nCoherence Scores:")
print("LDA Coherence:", lda_coherence)
print("NMF Coherence:", nmf_coherence)
print("LSA Coherence:", lsa_coherence)
```



- **Interpretation:**
  - Lower perplexity and higher coherence scores generally indicate better topic models.
  - Compare perplexity scores between LDA, NMF, and LSA.
  - Compare coherence scores between LDA, NMF, and LSA.
- **Note:**
  - The optimal number of topics (`n_components`) should be chosen based on the application and domain knowledge.
  - This example assumes `n_components=5` for illustration purposes.

By comparing the perplexity and coherence scores, you can gain insights into how well each model captures meaningful topics in your corpus. Keep in mind that these metrics are guides, and a qualitative assessment of the topics is essential for a comprehensive evaluation.

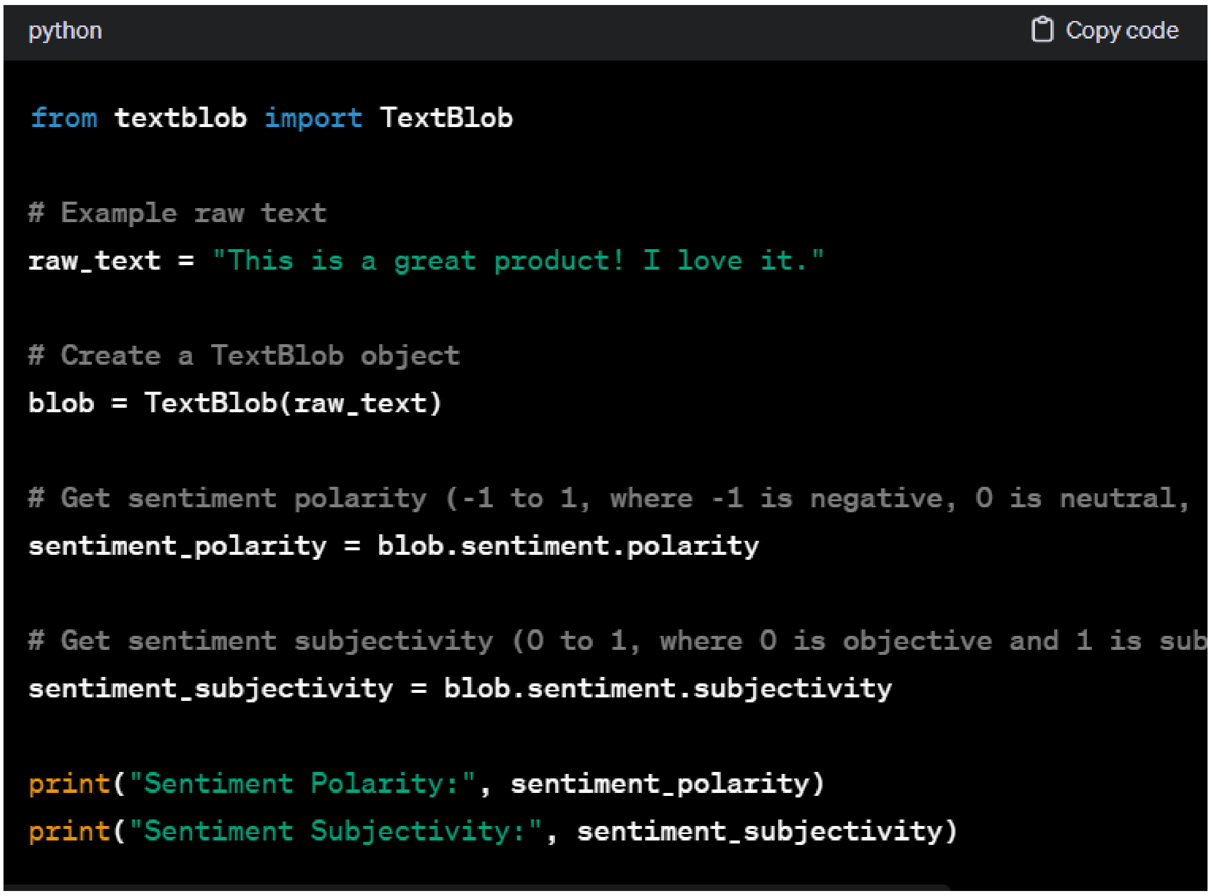
Evaluating topic models is a critical step in understanding how well they capture the latent themes within a corpus of text. Two commonly used metrics for assessing the quality of topic models are perplexity and coherence score.

Perplexity Score: Perplexity is a measure of how well a probabilistic model predicts a sample. In the context of topic modeling, perplexity gauges how accurately a model predicts the observed data, given its learned parameters. A lower perplexity score generally indicates a better model, as it suggests that the model is more effective at predicting the distribution of words in the corpus. However, it's important to note that perplexity should be interpreted with caution and is not always aligned with human judgment. It serves as a quantitative metric for comparing different models but may not reflect the interpretability of topics.

Coherence Score: Coherence measures the semantic similarity between high-scoring words within a topic. Higher coherence scores suggest that the topics generated by the model are more interpretable and semantically meaningful. Specifically, coherence examines whether words that frequently co-occur within a topic are also contextually related. A model with higher coherence scores is likely to produce more coherent and understandable topics. Unlike perplexity, coherence provides a more qualitative assessment of the semantic quality of topics, aligning closely with human interpretability.

In the context of e-commerce product descriptions, the application of these metrics becomes particularly relevant. Improving search engine optimization (SEO) and product discoverability on online retail websites relies on generating meaningful topics that accurately reflect the content of product descriptions. Retailers can use topic models to extract key terms and phrases, enhancing the relevance of product listings and improving the overall customer experience.

For instance, Latent Dirichlet Allocation (LDA), Non-Negative Matrix Factorization (NMF), and Latent Semantic Analysis (LSA) can be applied to identify topics within product descriptions. By comparing perplexity and coherence scores across these models, retailers can make informed decisions about which algorithm performs best for their specific dataset. The insights gained from topic modeling not only contribute to SEO strategies but also aid in creating more informative product listings, ultimately benefiting both retailers and customers by improving the accuracy of product searches and recommendations.


python

Copy code

```python
from textblob import TextBlob

# Example raw text
raw_text = "This is a great product! I love it."

# Create a TextBlob object
blob = TextBlob(raw_text)

# Get sentiment polarity (-1 to 1, where -1 is negative, 0 is neutral,
sentiment_polarity = blob.sentiment.polarity

# Get sentiment subjectivity (0 to 1, where 0 is objective and 1 is sub
sentiment_subjectivity = blob.sentiment.subjectivity

print("Sentiment Polarity:", sentiment_polarity)
print("Sentiment Subjectivity:", sentiment_subjectivity)
```