Topic Modeling with Latent Dirichlet Allocation (LDA)

This section of the code performs topic modeling using a technique called Latent Dirichlet Allocation (LDA). Topic modeling is a statistical method used to discover abstract "topics" that occur in a collection of documents.

Code Explanation:

library\_add

content\_copy

# 🧠 4. Topic modeling (LDA)  
def perform\_topic\_modeling(texts, num\_topics=8):  
    tokenized\_docs = [clean\_text(text) for text in texts.values()]  
    dictionary = corpora.Dictionary(tokenized\_docs)  
    corpus = [dictionary.doc2bow(text) for text in tokenized\_docs]  
    lda\_model = models.LdaModel(corpus, num\_topics=num\_topics, id2word=dictionary, passes=10)  
    return lda\_model.print\_topics(), lda\_model, corpus, dictionary  
  
topics, lda\_model, corpus, dictionary = perform\_topic\_modeling(texts)  
for i, topic in topics:  
    print(f"Topic {i}: {topic}")

[Körültekintően használja a kódot](https://g.co/legal/generative-code)

1. **def perform\_topic\_modeling(texts, num\_topics=8):**: This line defines a function called perform\_topic\_modeling that takes the text data (texts) and the desired number of topics (num\_topics, defaulting to 8) as input.
2. **tokenized\_docs = [clean\_text(text) for text in texts.values()]**: This line uses list comprehension to apply the previously defined clean\_text function to each document in the texts dictionary. It effectively cleans and tokenizes each document, creating a list of tokenized documents.
3. **dictionary = corpora.Dictionary(tokenized\_docs)**: This line creates a gensim Dictionary object. The dictionary maps each unique word in the corpus to a unique integer ID.
4. **corpus = [dictionary.doc2bow(text) for text in tokenized\_docs]**: This line creates a "bag-of-words" (BoW) representation of the corpus. Each document is converted into a list of tuples, where each tuple contains a word ID and its frequency in the document.
5. **lda\_model = models.LdaModel(corpus, num\_topics=num\_topics, id2word=dictionary, passes=10)**: This line trains the LDA model using the gensim library. The model parameters include the corpus, the desired number of topics, the dictionary mapping, and the number of passes (iterations) for training.
6. **return lda\_model.print\_topics(), lda\_model, corpus, dictionary**: The function returns the topics discovered by the LDA model, the trained LDA model itself, the corpus, and the dictionary.
7. **topics, lda\_model, corpus, dictionary = perform\_topic\_modeling(texts)**: This line calls the perform\_topic\_modeling function to train the model on the input text data. It stores the returned values in the respective variables.
8. **for i, topic in topics:**: This line starts a loop that iterates through the discovered topics.
9. **print(f"Topic {i}: {topic}")**: This line prints each topic and its associated words (and their probabilities) to the console.

**In essence, this code takes a collection of documents, processes them, trains an LDA model to identify underlying topics, and then prints the discovered topics along with the words that characterize each topic.**

A válasz értékelése

thumb\_up

thumb\_down

i want to compare this to the random model in the end of the docuemnt

spark

Okay, let's compare the LDA topic model to a random baseline model using the code provided at the end of the document.

Comparing LDA Model to Random Baseline

The code snippet you're referring to calculates and compares the coherence scores of the trained LDA model and a random baseline model:

library\_add

content\_copy

import numpy as np  
from gensim.models import CoherenceModel  
  
# ... (Assuming you have lda\_model, corpus, dictionary, texts) ...  
  
# 1. Random Topic Assignment  
# ...  
  
# 2. Calculate Coherence for LDA Model  
# ...  
  
# 3. Calculate Coherence for Random Baseline  
# ...  
  
# 4. Compare Coherence Scores  
print(f"LDA Model Coherence: {lda\_coherence}")  
print(f"Random Baseline Coherence: {random\_coherence}")  
  
# ... (Visualization code) ...

[Körültekintően használja a kódot](https://g.co/legal/generative-code)

Explanation:

1. **Random Baseline:**
   * It creates a random topic assignment for each document in the corpus.
   * This random assignment is used as a baseline to compare against the LDA model.
2. **Coherence Score:**
   * Coherence is a metric that measures the quality of topics in a topic model. Higher coherence scores indicate that the topics are more semantically meaningful and interpretable.
   * The code calculates the coherence score for both the LDA model and the random baseline using the CoherenceModel from gensim.
3. **Comparison:**
   * The code prints the coherence scores for both models.
   * It also provides a visualization (bar chart) to compare the scores.

Why Compare to a Random Baseline?

Comparing the LDA model to a random baseline is crucial to assess the model's effectiveness. If the LDA model has a significantly higher coherence score than the random baseline, it indicates that the model is actually learning meaningful topics from the data rather than just randomly assigning words to topics. This comparison helps validate the quality and usefulness of the LDA model.

**In summary, the code snippet you highlighted aims to evaluate the performance of the LDA topic model by comparing its coherence score to a random baseline. This comparison helps determine whether the discovered topics are truly meaningful or just random groupings of words.**

A válasz értékelése

thumb\_up

thumb\_down

the model i want to compare it to is here : import numpy as np from gensim.models import CoherenceModel

Assuming you have already:

- 'lda\_model': Your trained LDA model

- 'corpus': Your document-term matrix

- 'dictionary': Your vocabulary dictionary

- 'texts': Your original document texts (dict)

1. Random Topic Assignment

num\_topics = lda\_model.num\_topics num\_documents = len(corpus) random\_topics = np.random.randint(0, num\_topics, num\_documents)

2. Calculate Coherence for LDA Model

lda\_coherence\_model = CoherenceModel(model=lda\_model, texts=list(texts.values()), dictionary=dictionary, coherence='c\_v') lda\_coherence = lda\_coherence\_model.get\_coherence()

3. Calculate Coherence for Random Baseline

Create a dummy model with random topic assignments and a get\_topics method

class RandomTopicModel: def **init**(self, num\_topics, random\_topics): self.num\_topics = num\_topics self.random\_topics = random\_topics

def get\_document\_topics(self, bow, minimum\_probability=None):  
    doc\_id = bow[0][0] if isinstance(bow, list) and bow else 0 # Handle different bow formats  
  
    topic\_id = self.random\_topics[doc\_id % len(self.random\_topics)] # Ensure doc\_id is within bounds  
    return [(topic\_id, 1.0)]  # Assign probability 1.0 to the random topic  
  
def get\_topics(self):  
    # Return a list of topics, each represented as a list of (word\_id, probability) tuples.  
    # Since this is a random model, we'll just return empty topics.  
    return [[] for \_ in range(self.num\_topics)]

random\_topic\_model = RandomTopicModel(num\_topics, random\_topics) random\_coherence\_model = CoherenceModel(model=random\_topic\_model, texts=list(texts.values()), dictionary=dictionary, coherence='c\_v') random\_coherence = random\_coherence\_model.get\_coherence()

4. Compare Coherence Scores

print(f"LDA Model Coherence: {lda\_coherence}") print(f"Random Baseline Coherence: {random\_coherence}")

You can further visualize the comparis

spark

Okay, let's break down the code for comparing the LDA model to a random baseline using coherence scores:

library\_add

content\_copy

import numpy as np  
from gensim.models import CoherenceModel  
  
# ... (Assuming lda\_model, corpus, dictionary, texts are available) ...  
  
# 1. Random Topic Assignment  
num\_topics = lda\_model.num\_topics  
num\_documents = len(corpus)  
random\_topics = np.random.randint(0, num\_topics, num\_documents)  
  
# 2. Calculate Coherence for LDA Model  
lda\_coherence\_model = CoherenceModel(model=lda\_model, texts=list(texts.values()), dictionary=dictionary, coherence='c\_v')  
lda\_coherence = lda\_coherence\_model.get\_coherence()  
  
# 3. Calculate Coherence for Random Baseline  
# ... (RandomTopicModel class definition) ...  
random\_topic\_model = RandomTopicModel(num\_topics, random\_topics)  
random\_coherence\_model = CoherenceModel(model=random\_topic\_model, texts=list(texts.values()), dictionary=dictionary, coherence='c\_v')  
random\_coherence = random\_coherence\_model.get\_coherence()  
  
# 4. Compare Coherence Scores  
print(f"LDA Model Coherence: {lda\_coherence}")  
print(f"Random Baseline Coherence: {random\_coherence}")  
  
# ... (Visualization code) ...

[Körültekintően használja a kódot](https://g.co/legal/generative-code)

Step-by-Step Explanation:

1. **Random Topic Assignment:**
   * num\_topics = lda\_model.num\_topics: Gets the number of topics from the trained LDA model.
   * num\_documents = len(corpus): Gets the number of documents in the corpus.
   * random\_topics = np.random.randint(0, num\_topics, num\_documents): This is the core of creating the random baseline. It generates a NumPy array of random integers between 0 (inclusive) and num\_topics (exclusive). Each integer represents a randomly assigned topic for a document in the corpus.
2. **Calculate Coherence for LDA Model:**
   * lda\_coherence\_model = CoherenceModel(...): Creates a CoherenceModel object for the LDA model, using the actual topic assignments from the LDA model.
   * lda\_coherence = lda\_coherence\_model.get\_coherence(): Calculates the coherence score for the LDA model using the 'c\_v' coherence measure (a common choice).
3. **Calculate Coherence for Random Baseline:**
   * **class RandomTopicModel:**: This class defines a dummy topic model that simply assigns random topics to documents.
     + \_\_init\_\_: Initializes the model with the number of topics and the random topic assignments.
     + get\_document\_topics: This method simulates retrieving topic assignments for a document. It uses the pre-generated random\_topics array to assign a random topic to the document.
     + get\_topics: Returns empty topics (not needed for coherence calculation in this case).
   * random\_topic\_model = RandomTopicModel(num\_topics, random\_topics): Creates an instance of the random topic model.
   * random\_coherence\_model = CoherenceModel(...): Creates a CoherenceModel object for the random baseline, using the random topic assignments.
   * random\_coherence = random\_coherence\_model.get\_coherence(): Calculates the coherence score for the random baseline.
4. **Compare Coherence Scores:**
   * print(...): Prints the coherence scores for both the LDA model and the random baseline to the console for comparison.
5. **Visualization (Optional):**
   * The code likely includes a section to create a bar chart visualizing the coherence scores, making the comparison easier to interpret.

**In essence, this code creates a random topic model for comparison with the real LDA model. It then uses coherence scores as a metric to evaluate how much better (if at all) the LDA model is at discovering meaningful topics compared to random topic assignments.** If the LDA model has a significantly higher coherence score, it suggests that the model is learning meaningful relationships between words and documents.

A válasz értékelése

2025.05.06