**Researcher and collector:**

Yunus Lak

**Subject:**

Robust Processing Theory for Artificial Intelligence-Assisted Text Processing: A Case Study on Topic Modeling using Latent Dirichlet Allocation

**Completion:**

2024

Robust Processing Theory for Artificial Intelligence-Assisted Text Processing: A Case Study on Topic Modeling using Latent Dirichlet Allocation

Abstract

This paper presents a robust processing theory for artificial intelligence-assisted text processing, focusing on topic modeling using Latent Dirichlet Allocation (LDA). We demonstrate a comprehensive framework for text preprocessing, feature extraction, artificial intelligence model creation, and post-processing. Our implementation utilizes scikit-learn's TfidfVectorizer and LatentDirichletAllocation to perform topic modeling on a list of text samples. We visualize the results using word clouds and a heatmap, providing insights into the topic weights and relationships between topics and words.

Introduction

Artificial intelligence-assisted text processing has become increasingly important in various applications, including information retrieval, sentiment analysis, and topic modeling. A robust processing theory is essential for developing accurate and efficient text processing systems. This paper presents a comprehensive framework for topic modeling using LDA, a widely used technique for extracting underlying topics from large text corpora.

Methodology

Our implementation consists of the following steps:

Text Preprocessing: We load a list of 10 text samples into a text\_data variable. We create a TF-IDF vectorizer using scikit-learn's TfidfVectorizer, removing English stop words and transforming the text data into numerical features.

LDA Model Creation: We create an LDA model with 10 topics using scikit-learn's LatentDirichletAllocation. We fit the model to the TF-IDF features using the fit method.

Topic Weight Extraction: We extract topic weights for each feature from the LDA model using the components\_ attribute.

Feature Names Extraction: We receive the feature names (words) from the vectorizer using the get\_feature\_names\_out method.

Topic Words Extraction: We create a topic\_words dictionary to store the top 10 words for each topic. We iterate over the topics using np.argsort and extract the top 10 words for each topic.

Word Cloud Generation: We create a word cloud for each topic using the WordCloud library, setting the width and height to 800 pixels, the random seed to 21, and the maximum font size to 110.

Topic Weight Visualization: We use the Seaborn clustermap function to visualize the topic weights as a heatmap, setting the colormap to viridis and the size of the shape to (10, 10).

Results

Our implementation outputs 10 word clouds, one for each topic, showing the top 10 words for each topic. We also output a heatmap showing topic weights and relationships between topics and words.

Conclusion

This paper presents a comprehensive framework for topic modeling using LDA, highlighting the importance of careful preprocessing, feature extraction, and model selection. Our implementation serves as a foundation for developing robust text processing systems, providing insights into the topic weights and relationships between topics and words. Future work will focus on extending this framework to other text processing tasks and exploring the application of our robust processing theory in various domains.

Code Breakdown

The provided code is an implementation of a topic modeling processor using Latent Dirichlet Allocation (LDA) and visualizes the results using word clouds and a heatmap. Here's a breakdown of the code:

Text Preprocessing

The code loads a list of 10 text samples into a text\_data variable.

It creates a TF-IDF vectorizer using scikit-learn's TfidfVectorizer.

The vectorizer is fitted to the text data and transformed into numerical features using the fit\_transform method.

The stop\_words='english' parameter specifies that English stop words (common words like "the", "and", etc.) should be removed from the text data.

LDA Model Creation

The code creates an LDA model with 10 topics using scikit-learn's LatentDirichletAllocation.

The model is fitted to the TF-IDF features using the fit method.

Topic Weight Extraction

The code extracts topic weights for each feature from the LDA model using the components\_ attribute.

Feature Names Extraction

The code receives the feature names (words) from the vectorizer using the get\_feature\_names\_out method.

Topic Words Extraction

The code creates a topic\_words dictionary to store the top 10 words for each topic.

It iterates over the topics using np.argsort and extracts the top 10 words for each topic.

It creates a string of top words for each topic and adds it to the topic\_words dictionary.

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.decomposition import LatentDirichletAllocation

from wordcloud import WordCloud

# Load your text data into a list

text\_data = ["The quick brown fox jumps over the lazy dog",

    "The sun is shining brightly in the clear blue sky",

    "The cat purrs contentedly on my lap",

    "The baby laughs and plays with the colorful toys",

    "The flowers are blooming beautifully in the garden",

    "The dog runs quickly through the green grass",

    "The music is playing softly in the background",

    "The kids are playing happily in the park",

    "The food is delicious and flavorful at the restaurant",

    "The book is interesting and informative to read"]

# Create a TF-IDF vectorizer

vectorizer = TfidfVectorizer(stop\_words='english')

# Fit the vectorizer to your text data and transform it into numerical features

tfidf = vectorizer.fit\_transform(text\_data)

# Create a Latent Dirichlet Allocation (LDA) model

lda = LatentDirichletAllocation(n\_components=10, max\_iter=5, learning\_method='online', learning\_offset=50.,random\_state=0).fit(tfidf)

# Get the topic weights for each feature

topic\_weights = lda.components\_

# Get the feature names (words)

feature\_names = vectorizer.get\_feature\_names\_out()

# Create a dictionary to store the topic words

topic\_words = {}

# Iterate over the topics

for topic\_idx, topic in enumerate(topic\_weights):

    # Get the top 10 words for this topic

    top\_words = np.argsort(topic)[::-1][:10]

    # Create a string of the top words

    topic\_string = ' '.join([feature\_names[i] for i in top\_words])

    # Add the topic string to the dictionary

    topic\_words['Topic {}'.format(topic\_idx+1)] = topic\_string

# Create a word cloud for each topic

for topic, words in topic\_words.items():

    wordcloud = WordCloud(width = 800, height = 800, random\_state=21, max\_font\_size = 110).generate(words)

    plt.figure(figsize=(10, 10))

    plt.imshow(wordcloud, interpolation="bilinear")

    plt.axis("off")

    plt.title(topic)

    plt.show()

# Use Seaborn to visualize the topic weights

sns.set()

sns.clustermap(topic\_weights, cmap='viridis', figsize=(10, 10))

plt.show()