Recommendation System Documentation

AIM:

To develop a recommendation system that suggests similar documents based on cosine similarity and topic modeling with LDA.

Data Set:

The dataset used is the 20 Newsgroups dataset, fetched via the `sklearn.datasets` module.

WORKFLOW:

Workflow Explanation:

The workflow for this recommendation system includes:
1. Dataset: Fetch the 20 Newsgroups dataset, containing text documents from various topics.
2. Preprocessing: Clean, tokenize, and filter documents to retain only nouns for better topic extraction.
3. TF-IDF Vectorization: Convert preprocessed text to TF-IDF vectors for cosine similarity calculation.
4. Topic Modeling: Apply LDA with custom Gibbs sampling to identify topics in each document.
5. Recommendation: Provide two types of recommendations – Cosine Similarity-based and Topic-based.

CODE:

The complete code implementation is as follows:

```
import numpy as np
import re
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.datasets import fetch_20newsgroups

# Download NLTK stopwords if needed
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
nltk.download('punkt')

# Set up stop words
stop_words = set(nltk.corpus.stopwords.words('english'))
```

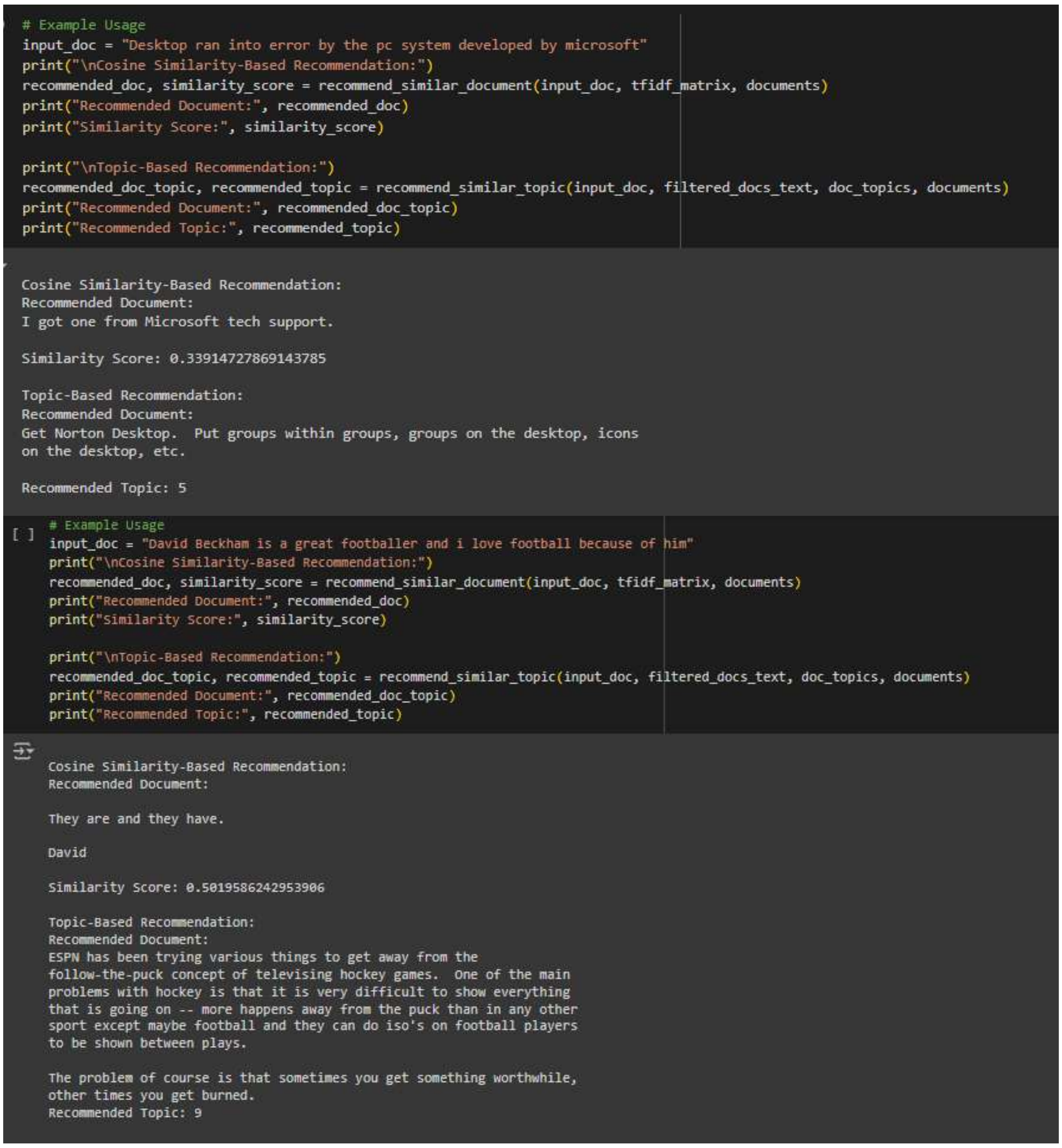
```
custom_stopwords = {
    "im", "also", "first", "get", "however", "last", "many", "may", "much", "on",
    "would", "make", "new", "even", "could", "think", "know", "see", "say",
    "people", "time", "since", "well", "want", "still", "take", "back"
}
stop_words.update(custom_stopwords)

# Load dataset
newsgroups_data = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))
documents = newsgroups_data.data

# Step 1: Text preprocessing
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)  # Keep only letters and space
    words = nltk.word_tokenize(text)
    return [word for word in words if word not in stop_words and len(word) > 1]

# Function to filter nouns only
def filter_words(words):
    filtered_words = []
    pos_tags = nltk.pos_tag(words)
    for word, tag in pos_tags:
        if tag in ['NN', 'NNP']:
            filtered_words.append(word)
    return filtered_words

# Preprocess and filter documents
processed_docs = [preprocess_text(doc) for doc in documents]
filtered_docs = [filter_words(doc) for doc in processed_docs]

# Convert processed documents back to full text
filtered_docs_text = [' '.join(doc) for doc in filtered_docs]

# Step 2: TF-IDF vectorization
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(filtered_docs_text)

# Step 3: Topic Modeling using LDA (custom Gibbs Sampling)
# Initialize LDA parameters
num_topics = 10
num_iterations = 5
alpha = 0.1
beta = 0.01

# Create vocabulary and word mappings
vocab = set(word for doc in filtered_docs for word in doc)
vocab_size = len(vocab)
word_to_id = {word: i for i, word in enumerate(vocab)}

# Initialize topic-related matrices
doc_topic_count = np.zeros((len(processed_docs), num_topics))
topic_word_count = np.zeros((num_topics, vocab_size))
topic_count = np.zeros(num_topics)
```

```
topic_assignment = []

# Random topic assignment
for doc_idx, doc in enumerate(filtered_docs):
    topic_assignment.append([])
    for word in doc:
        topic = np.random.randint(num_topics)
        word_id = word_to_id[word]
        topic_assignment[doc_idx].append(topic)
        doc_topic_count[doc_idx, topic] += 1
        topic_word_count[topic, word_id] += 1
        topic_count[topic] += 1

# Gibbs Sampling for LDA
for it in range(num_iterations):
    for doc_idx, doc in enumerate(filtered_docs):
        current_topics = topic_assignment[doc_idx]
        for word_idx, word in enumerate(doc):
            current_topic = current_topics[word_idx]
            word_id = word_to_id[word]
            doc_topic_count[doc_idx, current_topic] -= 1
            topic_word_count[current_topic, word_id] -= 1
            topic_count[current_topic] -= 1

            # Calculate topic probabilities
            topic_probs = (doc_topic_count[doc_idx] + alpha) * \
                       (topic_word_count[:, word_id] + beta) / \
                       (topic_count + beta * vocab_size)
            topic_probs /= topic_probs.sum()

            # Sample new topic
            new_topic = np.random.choice(num_topics, p=topic_probs)
            topic_assignment[doc_idx][word_idx] = new_topic
            doc_topic_count[doc_idx, new_topic] += 1
            topic_word_count[new_topic, word_id] += 1
            topic_count[new_topic] += 1

# Get dominant topic for each document
doc_topics = np.argmax(doc_topic_count, axis=1)

# Step 4: Document Recommendation Functions

# Cosine Similarity-Based Recommendation
def recommend_similar_document(input_doc, tfidf_matrix, documents):
    input_vector = vectorizer.transform([input_doc])
    similarity_scores = cosine_similarity(input_vector, tfidf_matrix).flatten()
    most_similar_idx = np.argmax(similarity_scores)
    return documents[most_similar_idx], similarity_scores[most_similar_idx]

# Topic-Based Recommendation
def recommend_similar_topic(input_doc, filtered_docs_text, doc_topics, documents):
    input_vector = vectorizer.transform([input_doc])
    input_topic_dist = np.zeros(num_topics)
```

```
  # Sample topic distribution for input
  for word in input_doc.split():
    if word in word_to_id:
      word_id = word_to_id[word]
      word_topic_probs = (topic_word_count[:, word_id] + beta) / \
                (topic_count + beta * vocab_size)
      input_topic_dist += word_topic_probs
  input_topic_dist /= input_topic_dist.sum()
  input_topic = np.argmax(input_topic_dist)

  # Find document with the closest matching topic
  candidate_docs = [idx for idx, topic in enumerate(doc_topics) if topic == input_topic]
  if candidate_docs:
    most_similar_doc = max(candidate_docs, key=lambda idx: cosine_similarity(input_vector,
tfidf_matrix[idx]))
    return documents[most_similar_doc], input_topic
  else:
    return "No similar topic document found.", input_topic

# Example Usage
input_doc = "Desktop ran into error by the pc system developed by microsoft"
print("\nCosine Similarity-Based Recommendation:")
recommended_doc, similarity_score = recommend_similar_document(input_doc, tfidf_matrix,
documents)
print("Recommended Document:", recommended_doc)
print("Similarity Score:", similarity_score)

print("\nTopic-Based Recommendation:")
recommended_doc_topic, recommended_topic = recommend_similar_topic(input_doc,
filtered_docs_text, doc_topics, documents)
print("Recommended Document:", recommended_doc_topic)
print("Recommended Topic:", recommended_topic)

# Example Usage
input_doc = "David Beckham is a great footballer and i love football because of him"
print("\nCosine Similarity-Based Recommendation:")
recommended_doc, similarity_score = recommend_similar_document(input_doc, tfidf_matrix,
documents)
print("Recommended Document:", recommended_doc)
print("Similarity Score:", similarity_score)

print("\nTopic-Based Recommendation:")
recommended_doc_topic, recommended_topic = recommend_similar_topic(input_doc,
filtered_docs_text, doc_topics, documents)
print("Recommended Document:", recommended_doc_topic)
print("Recommended Topic:", recommended_topic)
```

RESULT:

Below is a sample output screenshot of the recommendations generated by the system.



```python
# Example Usage
input_doc = "Desktop ran into error by the pc system developed by microsoft"
print("\nCosine Similarity-Based Recommendation:")
recommended_doc, similarity_score = recommend_similar_document(input_doc, tfidf_matrix, documents)
print("Recommended Document:", recommended_doc)
print("Similarity Score:", similarity_score)

print("\nTopic-Based Recommendation:")
recommended_doc_topic, recommended_topic = recommend_similar_topic(input_doc, filtered_docs_text, doc_topics, documents)
print("Recommended Document:", recommended_doc_topic)
print("Recommended Topic:", recommended_topic)
```

```
Cosine Similarity-Based Recommendation:
Recommended Document:
I got one from Microsoft tech support.

Similarity Score: 0.33914727869143785

Topic-Based Recommendation:
Recommended Document:
Get Norton Desktop.  Put groups within groups, groups on the desktop, icons
on the desktop, etc.

Recommended Topic: 5
```

```python
# Example Usage
input_doc = "David Beckham is a great footballer and i love football because of him"
print("\nCosine Similarity-Based Recommendation:")
recommended_doc, similarity_score = recommend_similar_document(input_doc, tfidf_matrix, documents)
print("Recommended Document:", recommended_doc)
print("Similarity Score:", similarity_score)

print("\nTopic-Based Recommendation:")
recommended_doc_topic, recommended_topic = recommend_similar_topic(input_doc, filtered_docs_text, doc_topics, documents)
print("Recommended Document:", recommended_doc_topic)
print("Recommended Topic:", recommended_topic)
```

```
Cosine Similarity-Based Recommendation:
Recommended Document:

They are and they have.

David

Similarity Score: 0.5019586242953906

Topic-Based Recommendation:
Recommended Document:
ESPN has been trying various things to get away from the
follow-the-puck concept of televising hockey games.  One of the main
problems with hockey is that it is very difficult to show everything
that is going on -- more happens away from the puck than in any other
sport except maybe football and they can do iso's on football players
to be shown between plays.

The problem of course is that sometimes you get something worthwhile,
other times you get burned.
Recommended Topic: 9
```