Assignment Project Report

PLSA: Text Document Clustering

Name: Pavan Tiwari
Course: AI and ML
(Batch 4)

- **Problem Statement**

  Perform topic modelling using the 20 Newsgroup dataset (the dataset is also available in sklearn datasets sub-module). Perform the required data cleaning steps using NLP and then model the topics 1. Using Latent Dirichlet Allocation (LDA). 2. Using Probabilistic Latent Semantic Analysis (PLSA)

  Prerequisites

  - Software:
    - Python 3 (Use anaconda as your python distributor as well)

  - Tools:
    - Pandas
    - Numpy
    - Matplotlib
    - Seaborn
    - NLTK

  - Dataset Link: https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html

- **Method Used**

  PLSA or Probabilistic Latent Semantic Analysis is a technique used to model information under a probabilistic framework. It is a statistical technique for the analysis of two-mode and co-occurrence data. PLSA characterizes each word in a document as a sample from a mixture model, where mixture components are conditionally independent multinomial distributions. Its main goal is to model cooccurrence information under a probabilistic framework in order to discover the underlying semantic structure of the data.

- **Implementation:**

1. Load all required libraries and Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.decomposition import NMF
from sklearn.preprocessing import normalize
```

```
categories = ['alt.atheism', 'talk.religion.misc','comp.graphics', 'sci.space', 'sci.med','sci.space','soc.religion.christia
newsgroups_train = fetch_20newsgroups(subset='train',categories=categories)
```

2. Preprocessing data

```
import nltk
import re
```

```
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
ps = PorterStemmer()
```

```
corpus = []
lem = WordNetLemmatizer()
for i in range(0,len(newsgroups_train.filenames)):
    review = newsgroups_train.data[i].split()
    review = [lem.lemmatize(word) for word in review if word not in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

```
vectorizer = CountVectorizer(max_features=5000)
x_counts = vectorizer.fit_transform(corpus)
```

```
transformer = TfidfTransformer()
x_tfidf = transformer.fit_transform(x_counts)
```

3. Applying NMF model For topic Modeling

```
#number of topics
num_topics=7
#obtain a NMF model.
model = NMF(n_components=num_topics, init='nndsvd')
#fit the model
model.fit(xtfidf_norm)
```

```
NMF(init='nndsvd', n_components=7)
```

```
def get_nmf_topics(model, n_top_words):

    #the word ids obtained need to be reverse-mapped to the words so we can print the topic names.
    feat_names = vectorizer.get_feature_names()

    word_dict = {}
    for i in range(num_topics):

        #for each topic, obtain the largest values, and add the words they map to into the dictionary.
        words_ids = model.components_[i].argsort()[:-n_top_words - 1:-1]
        words = [feat_names[key] for key in words_ids]
        word_dict['Topic # ' + '{:02d}'.format(i+1)] = words

    return pd.DataFrame(word_dict)
```

1. Applying LDA model For topic Modeling

```
#number of topics
num_topics=7
#obtain a LDA model.
model = LatentDirichletAllocation(n_components=num_topics)
#fit the model
model.fit(xtfidf_norm)
```

```
LatentDirichletAllocation(n_components=7)
```

```
def get_lda_topics(model, n_top_words):

    #the word ids obtained need to be reverse-mapped to the words so we can print the topic names.
    feat_names = vectorizer.get_feature_names()

    word_dict = {}
    for i in range(num_topics):

        #for each topic, obtain the largest values, and add the words they map to into the dictionary.
        words_ids = model.components_[i].argsort()[:-n_top_words - 1:-1]
        words = [feat_names[key] for key in words_ids]
        word_dict['Topic # ' + '{:02d}'.format(i+1)] = words

    return pd.DataFrame(word_dict)
```