Assignment Report

Natural Language Processing

Document Classification Using Term Frequency

	Name
Prepared By	Muhammad Usman Siddiqui (27216)
Submitted To	Dr. Sarim

Table of Contents

1. INTRODUCTION:

This assignment aims to apply text classification techniques based on term frequency analysis.

2. BACKGROUND:

The frequency of a certain term in a document, also known as Term Frequency (TF), is an integral measurement for classifying documents. Machine learning algorithms can use TF to determine the significance of words and their impact on the content of a document. This insight helps classifiers distinguish between different types of documents by identifying unique patterns in key terms, resulting in precise and efficient categorization. Essentially, TF acts as an essential element supporting text analysis; it assists algorithms with comprehending textual data so that they can accurately classify them accordingly.

3. METHODOLOGY:

3.1 PDF to Text Conversion:

If the document is in PDF format, convert it to text using the `pdf_to_txt_converter` function. This function reads the PDF file, extracts the text, and returns it as a string.

3.2 Vectorization:

Convert the text data into numerical vectors using the `TfidfVectorizer` from the `sklearn` library. This creates a term frequency-inverse document frequency (TF-IDF) matrix, where each row represents a document and each column represents a term.

3.3 Logarithmic Scaling:

Apply logarithmic scaling to the term frequency matrix to normalize the data and reduce the impact of very frequent terms. This is done using the `numpy` library's `log1p` function, which calculates `log(1 + x)` for each element `x` in the matrix.

3.4 Cosine Similarity Calculation:

For a new document, calculate the cosine similarity between the new document and each existing document using the `compute_cosine_similarity` function. This function transforms the new document into a vector, applies logarithmic scaling, and then computes the cosine similarity.

3.5 Classification:

Based on the cosine similarities, classify the new document. This could be done by assigning the label of the most similar existing document, or by taking the average similarity for each label and assigning the label with the highest average.

This methodology allows you to classify a new document based on its similarity to a set of existing documents. The cosine similarity is a measure of the cosine of the angle between two vectors, and it ranges from -1 (completely dissimilar) to 1 (completely similar).

4. DATASET EXPLANATION:

The pdfs used for this assignment were downloaded from the following resources

1. Natural Language Processing (NLP) downloaded from arxiv.org (search keyword 'Natural Language Processing'
2. Medical Research downloaded from nih.gov (search keywords, heart disease, lung disease)
3. Deep Learning downloaded from arxiv.org (search keywords, CNN, LSTM)
4. Machine Learning downloaded from arxiv.org (search keywords Decision Trees, SVM)
5. Astrophysics downloaded from arxiv.org (no specific keyword)

10 of each pdfs were downloaded and placed inside folders of the same name

1. NLP
2. Med
3. DL
4. ML
5. Asp

5. CODE WORKING

5.1 PDF Extraction

This Python script is designed to extract text from all PDF files located in a specified directory and its subdirectories. It uses the `os` and `pdfminer.high_level` libraries to navigate the directory structure and extract text from the PDF files, respectively. For each PDF file, the script extracts the text and writes it to a new .txt file in the same directory. The .txt file is named according to the order in which the PDF files are processed. The script is initiated by calling the `extract_text_from_pdfs` function with the path to the root directory as an argument

```python
import os
from pdfminer.high_level import extract_text

def extract_text_from_pdfs(root_folder):
    # Iterate over all subdirectories in the root folder
    for subdir, dirs, files in os.walk(root_folder):
        pdf_files = [f for f in files if f.endswith('.pdf')]
        # Iterate over all PDF files in the subdirectory
        for i, pdf_file in enumerate(pdf_files, start=1):
            pdf_path = os.path.join(subdir, pdf_file)

            # Extract text from the PDF file
            text = extract_text(pdf_path)

            # Save the text to a new .txt file in the same subdirectory
            txt_file = os.path.join(subdir, f'{i}.txt')
            with open(txt_file, 'w') as f:
                f.write(text)

# Call the function with the path to your root folder
extract_text_from_pdfs('C:\\Users\\Dell\\Desktop\\NLP Assignment 3\\')
```

5.2 Term- Document Matrix Generation:

This Python script is designed to create a document-term matrix from all text files in a specified directory and its subdirectories. It uses the `os` and `pandas` libraries for file handling and data manipulation, and `CountVectorizer` from `sklearn.feature_extraction.text` for text vectorization. For each text file, the script appends the full file path and the name of the subdirectory (used as the label) to separate lists. It then creates a document-term matrix from the text files, converts the matrix to a pandas DataFrame, and adds the labels as a new column. The function is called with the path to the root directory as an argument, and the resulting DataFrame is stored in `df`. Finally, the script applies the `log1p` function from the `numpy` library to each element in the DataFrame (excluding the 'label' column), which calculates `log(1 + x)` for each element `x`. This is a way of applying logarithmic scaling to the term frequencies. The result is stored in `df_log`.

```
import os
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from pdfminer.high_level import extract_text
vectorizer = CountVectorizer(input='filename')

def create_document_term_matrix_with_labels(root_folder):

    # Collect all .txt files in all subdirectories
    txt_files = []
    labels = []
    for subdir, dirs, files in os.walk(root_folder):
        for f in files:

            if f.endswith('.txt'):
                txt_files.append(os.path.join(subdir, f))
                labels.append(os.path.basename(subdir))  # Use the subfolder name as the label

    # Create a document-term matrix
    dtm = vectorizer.fit_transform(txt_files)

    # Convert the matrix to a pandas DataFrame
    df = pd.DataFrame(dtm.toarray(), columns=vectorizer.get_feature_names_out(), index=['Doc'+str(i+1) for i in range(dtm.shape[0])])

    # Add the labels to the DataFrame
    df['label'] = labels

    return df
df = create_document_term_matrix_with_labels('C:\\Users\\Dell\\Desktop\\NLP Assignment 3\\')
df_log = np.log1p(df.drop('label', axis=1))
```

after execution we can see the size of our df_log dataframe.

(50, 27983)

This show each of the 50 document (1 document per row) and 27983 terms or the vocabulary of the whole corpus.

5.3 __Document Classification:__

This Python script contains two functions: `pdf_to_txt_converter` and `compute_cosine_similarity`. The `pdf_to_txt_converter` function takes a path to a PDF document, extracts the text from the document using the `extract_text` function from the `pdfminer.high_level` library, and writes the extracted text to a new .txt file in the same directory. The `compute_cosine_similarity` function takes a path to a new document (either a .txt or .pdf file). If the document is a PDF, it converts it to a .txt file using the `pdf_to_txt_converter` function. It then transforms the document into a vector using a pre-existing `CountVectorizer` instance, applies logarithmic scaling to the vector, and computes the cosine similarity between this vector and each document in a pre-existing document-term matrix (`df_log`). The function returns a list of cosine similarities. The script ends by calling `compute_cosine_similarity` with a path to a PDF document and storing the returned list of cosine similarities in `x`.

```
def pdf_to_txt_converter(doc_path):
    text = extract_text(doc_path)
    txt_path = os.path.splitext(doc_path)[0] + '.txt'
    # Save the text to a new .txt file in the same subdirectory
    with open(txt_path, 'w') as f:
        f.write(text)
    return None

def compute_cosine_similarity(new_doc):
    if new_doc.endswith('.pdf'):
        pdf_to_txt_converter(new_doc)
        new_doc = os.path.splitext(new_doc)[0] + '.txt'
    from sklearn.metrics.pairwise import cosine_similarity
    new_doc_vector = vectorizer.transform([new_doc])
    new_doc_row = pd.Series(new_doc_vector.toarray().flatten(), 
index=vectorizer.get_feature_names_out())
    new_doc_list = new_doc_row.drop('label').tolist()
    new_doc_list = np.log1p(new_doc_list)
    similarities = []
    for i in range(df_log.shape[0]):
        row = df_log.loc['Doc'+str(i+1)].tolist()
        sim = cosine_similarity(np.array(new_doc_list).reshape(1, -1), 
np.array(row).reshape(1, -1))
        similarities.append(sim[0][0])
    return similarities
#pdf_to_txt_converter('C:\\Users\\Dell\\Desktop\\NLP Assignment 3\\2312.16724.pdf')
x = compute_cosine_similarity(('C:\\Users\\Dell\\Desktop\\NLP Assignment 
3\\2312.16724.pdf'))
```

we get ‘x’ a list if size 50, with each element as the cosine similarity of the given document with the document in the df_log.

5.4 **Classification using Cosine Similarity:**

This Python script performs a series of operations on a list `x`. First, it divides `x` into sublists of length 10 using list comprehension, with any remaining elements at the end forming a sublist of length less than 10. It then computes the mean of each sublist and stores these averages in a new list. The script defines a list of labels, presumably corresponding to the sublists. It then finds the index of the maximum average and uses this index to select the corresponding label from the list of labels. Finally, it prints this label. The purpose of this script could be to categorize a larger dataset (represented by `x`) into smaller groups and label each group based on its average value.

```
sublists = [x[i:i+10] for i in range(0, len(x), 10)]
averages = [np.mean(sublist) for sublist in sublists]
labels = ['Asp','DL', 'Med', 'ML', 'NLP']
label = labels[np.argmax(averages)]
```

```
print(label)
```

We can see tat the given document is actually based on CNN’s which was included in the DL class while searching for the DL class. Which shows that the classification works correctly.

6. CONCLUSION

The document classifier is implemented as per prequirement.