“Student Answer Evaluation Tool”: “Leveraging Cosine Similarity in NLP”

Introduction

Using cosine similarity, a mathematical technique commonly used in Natural Language Processing (NLP), this tool evaluates the similarity between a model answer and a student's answer. The evaluation includes a percentage similarity score and an assigned grade based on the similarity.

What is Natural Language Processing (NLP)?

Natural Language Processing (NLP) is a field of artificial intelligence focusing on the interaction between computers and human language. It enables machines to understand, interpret, and respond to text or spoken words in a meaningful and useful way.

Key applications of NLP include:

1. Machine Translation (e.g., Google Translate)
2. Sentiment Analysis
3. Chatbots and Virtual Assistants (e.g., Alexa, Siri)
4. Text Summarization
5. Information Retrieval (e.g., search engines)

What is Cosine Similarity?

Cosine similarity is a metric used to measure the similarity between two non-zero vectors. It calculates the cosine of the angle between the vectors in a multi-dimensional space, providing a value that quantifies their similarity. It is widely used in applications such as information retrieval, natural language processing, and recommendation systems.

Purpose

This program evaluates the similarity between a model answer and a student's answer using cosine similarity, a mathematical technique commonly used in Natural Language Processing (NLP). The evaluation includes a percentage similarity score and an assigned grade based on the similarity.

Cosine Similarity Example: Comparing Two Texts

In this example, we compare two texts to determine their similarity using the Cosine Similarity formula. The texts are preprocessed, vectorized, and mathematically analyzed step by step. The goal is to calculate the similarity percentage and understand the relationship between the two texts.

Step 1: Preprocessing Text

Before any computation, ensure the text is preprocessed to remove inconsistencies:

1. Convert text to lowercase: To standardize comparison.
2. Remove punctuation and special characters: To focus on meaningful content.
3. Tokenize text: Split text into individual words.
4. Remove stopwords: Exclude common words (e.g., "the", "is") that do not add value to similarity.

1. Representing Text as Vectors

Before using cosine similarity, textual data needs to be converted into numerical vectors. This is done using methods like TF-IDF (Term Frequency-Inverse Document Frequency):

- TF-IDF Vectorizer assigns weights to words based on how frequently they appear in a document compared to their overall frequency across all documents.
- Each document becomes a high-dimensional vector where each dimension corresponds to a term's importance.

Before computing cosine similarity, the texts are preprocessed to remove inconsistencies. This includes converting to lowercase, removing punctuation, and tokenizing. Stopwords are not removed here to preserve the original meaning.

Preprocessed Text 1: the quick brown fox jumps over the lazy dog

Preprocessed Text 2: a swift auburn fox leaps across the sleepy canine

Step 2: Vectorizing the Texts

Using TF-IDF (Term Frequency-Inverse Document Frequency), the terms in both texts are represented numerically. This assigns weights to words based on their importance in the text.

The TF-IDF weights for unique terms are as follows:

Term	TF-IDF (Text 1)	TF-IDF (Text 2)
quick	0.30	0.00
brown	0.30	0.00
fox	0.40	0.40
jumps	0.40	0.00
lazy	0.30	0.00
dog	0.40	0.00
swift	0.00	0.30
auburn	0.00	0.30
leaps	0.00	0.40

sleepy	0.00	0.30
canine	0.00	0.30
over	0.40	0.00
across	0.00	0.40

Step 3: Calculate the Dot Product

The dot product measures the overlap between the vectors of the two texts. It is calculated by summing the product of corresponding TF-IDF weights for each term.

Dot Product: (0.4 × 0.4) + (0.3 × 0.3) + ... = 0.34

Step 4: Calculate the Magnitudes

The magnitude of each vector is calculated as the square root of the sum of the squares of its components.

Magnitude of Text 1: √(0.4² + 0.3² + ...) = 0.76

Magnitude of Text 2: √(0.4² + 0.3² + ...) = 0.86

Step 5: Calculate Cosine Similarity

The formula for cosine similarity is:

Cosine Similarity = (A · B) / (||A|| × ||B||)

Where:

- A and B are two vectors representing the text.
- A · B is the dot product of the two vectors.
- ||A|| and ||B|| are the magnitudes (norms) of the vectors.

Cosine similarity is calculated as the dot product of the vectors divided by the product of their magnitudes.

Cosine Similarity: 0.34 / (0.76 × 0.86) ≈ 0.79

Result

The cosine similarity between the two texts is approximately 0.79, which means the texts are **79% similar**. This indicates a high level of similarity despite differences in word choice and structure.

2. Code Implementation

```
import os
import numpy as np
from tkinter import Tk, filedialog
from sklearn.feature_extraction.text import TfidfVectorizer
from colorama import Fore, Style, init
init(autoreset=True)
```

```
def get_terminal_width():
    try:
        return os.get_terminal_size().columns
    except OSError:
        return 80

def center_align(text):
    width = get_terminal_width()
    return text.center(width)

def cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)

def evaluate_grade(percentage):
    if percentage >= 90:
        return Fore.GREEN + "A+" + Style.RESET_ALL
    elif percentage >= 80:
        return Fore.GREEN + "A" + Style.RESET_ALL
    elif percentage >= 70:
        return Fore.BLUE + "B+" + Style.RESET_ALL
    elif percentage >= 60:
        return Fore.BLUE + "B" + Style.RESET_ALL
    elif percentage >= 50:
        return Fore.YELLOW + "Pass" + Style.RESET_ALL
    else:
        return Fore.RED + "Fail" + Style.RESET_ALL

def calculate_marks(model_answer, student_answer, total_marks=100):
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform([model_answer, student_answer])

    vec1 = tfidf_matrix[0].toarray()[0]
    vec2 = tfidf_matrix[1].toarray()[0]

    similarity = cosine_similarity(vec1, vec2)
    percentage = similarity * total_marks

    grade = evaluate_grade(percentage)

    return percentage, grade

def display_result(model_answer, student_answer, percentage, grade):
```

```
    print(Fore.BLUE + center_align("=" * 50))
    print(Fore.MAGENTA + center_align("COSINE SIMILARITY EVALUATION"))
    print(Fore.BLUE + center_align("=" * 50))

    print(Fore.LIGHTYELLOW_EX + "\n" + center_align("📘 MODEL ANSWER:"))
    print(center_align("-" * 50))
    for line in model_answer.splitlines():
        print(center_align(line))
    print(center_align("-" * 50))

    print(Fore.LIGHTCYAN_EX + "\n" + center_align("🖍️ STUDENT ANSWER:"))
    print(center_align("-" * 50))
    for line in student_answer.splitlines():
        print(center_align(line))
    print(center_align("-" * 50))

    print(Fore.LIGHTGREEN_EX + "\n" + center_align("📊 Evaluation Result:"))
    print(center_align("-" * 50))
    print(center_align(f"Result in Percentage: {percentage:.2f}%"))
    print(center_align(f"Final Result: - Grade: {grade}"))
    print(Fore.BLUE + center_align("=" * 50))

def upload_file(prompt):
    Tk().withdraw()
    file_path = filedialog.askopenfilename(title=prompt, filetypes=[("Text files", "*.txt"), ("All files",
"*.*")])
    if not file_path:
        raise FileNotFoundError("No file was selected!")
    return file_path

if __name__ == "__main__":
    print(Fore.CYAN + center_align("=" * 50))
    print(Fore.GREEN + center_align("COSINE SIMILARITY EVALUATION TOOL USING NATURAL
LANGUAGE PROCESSING (NLP)"))
    print(Fore.CYAN + center_align("=" * 50))

    print(center_align("Please upload the files for evaluation."))

    try:
        model_file = upload_file("Select the Model Answer File")
        student_file = upload_file("Select the Student Answer File")

        print(Fore.WHITE + "\n" + center_align(f"Model file selected: {model_file}"))
        print(Fore.WHITE + center_align(f"Student file selected: {student_file}"))
```

```
        with open(model_file, 'r') as mf:
            model_answer = mf.read()
        with open(student_file, 'r') as sf:
            student_answer = sf.read()

        percentage, grade = calculate_marks(model_answer, student_answer)

        display_result(model_answer, student_answer, percentage, grade)

    except FileNotFoundError as e:
        print(Fore.RED + "\n" + center_align(f"❌ Error: {e}"))
    except Exception as e:
        print(Fore.RED + "\n" + center_align(f"❌ Unexpected error: {e}"))
```