Cross language phonetic similarity

Sure, let's delve into some of the mathematical calculations that occur within the layers of your convolutional neural network (CNN). We'll focus on the convolutional layers and the dense layers, as these are the key components where most of the mathematical operations take place during both forward and backward propagation.

1. Convolutional Layer:

Let's consider the first convolutional layer in your model:

- Input Shape: (150, 150, 3) # Assuming RGB images

- Number of Filters: 32

- Filter Size: (3, 3)

- Stride: (default, usually (1, 1))

- Padding: (default, usually 'valid')

- Activation: ReLU

For each filter, the mathematical operations are as follows:

- Convolution: Apply the filter to the input image. This involves element-wise multiplication of the filter weights with the corresponding pixel values of the input image, followed by summing the results.

- Bias Addition: Add a bias term to the result of convolution.

- ReLU Activation: Apply the ReLU activation function to introduce non-linearity.

- Output Shape: (148, 148, 32) # Assuming 'valid' padding and default stride

The number of parameters in the convolutional layer is calculated as follows:

- Parameters per filter: (3 3 3) + 1 (for bias) = 28

- Total Parameters: 32 filters 28 parameters/filter = 896 parameters

2. Pooling Layer:

MaxPooling is typically used after each convolutional layer. The pooling operation reduces the spatial dimensions of the feature maps. Assuming a pool size of (2, 2), the output shape would be halved in both dimensions.

3. Flattening Layer:

The flattening operation reshapes the 3D feature maps into a 1D array. Assuming the output shape of the previous layer is (N, M, P), the flattened array would have a length of N M P.

4. Dense Layer:

Let's consider the first dense layer in your model:

- Input Shape: (N,)

- Number of Neurons: 128

- Activation: ReLU

For each neuron, the mathematical operations are as follows:

- Weighted Sum: Compute the dot product of the input vector with the weight vector of the neuron.

- Bias Addition: Add a bias term to the result of the weighted sum.

- ReLU Activation: Apply the ReLU activation function.

- Output Shape: (128,)

The number of parameters in the dense layer is calculated as follows:

- Parameters per neuron: N (input size) + 1 (bias) = N + 1

- Total Parameters: 128 neurons (N + 1) parameters/neuron

These are the basic mathematical calculations involved in the forward pass of a CNN. During training, backpropagation is used to compute gradients with respect to the loss function, and optimization algorithms update the weights and biases of the network to minimize the loss.

**Image Classification:**

**a. CNN**

1. Importing Libraries: `os`: This library provides functions to interact with the operating system. `tensorflow as tf`: TensorFlow is a popular deep learning framework. ImageDataGenerator`: This class generates batches of tensor image data with real-time data augmentation. `Sequential`: This is the Keras API for creating a linear stack of layers. `Conv2D`, `MaxPooling2D`, `Flatten`, `Dense`: These are layer classes from Keras used to define the structure of a convolutional neural network (CNN).

2. Data Directories: train\_dir and test\_dir: These variables specify the directories where the training and testing data are located.

3. Image Dimensions and Batch Size: `img\_width` and `img\_height`: These variables define the dimensions to which the input images will be resized. batch\_size`: This variable specifies the number of samples per gradient update during training.

4. Data Augmentation and Normalization: `ImageDataGenerator`: Instances of this class will perform data augmentation and normalization on the input data. train\_datagen`: Data generator for training data with augmentation techniques like shear range, zoom range, and horizontal flip. test\_datagen`: Data generator for test data with normalization only.

5. Loading and Preprocessing Data: flow\_from\_directory`: This method generates batches of data from image files located in a directory, applying transformations specified in the `ImageDataGenerator`. train\_generator` and `test\_generator`: These are the generators for training and testing data respectively.

6. Defining the CNN Model:

- `Sequential`: This creates a linear stack of layers.

- `Conv2D`: This layer creates a convolutional layer with specified number of filters and kernel size.

- `MaxPooling2D`: This layer performs max pooling operation.

- `Flatten`: This layer flattens the input, which is necessary before passing it to a fully connected layer.

- `Dense`: This layer is a fully connected layer with specified number of units and activation function.

7. Compiling the Model:

- `compile`: This method configures the model for training by specifying the optimizer, loss function, and metrics to monitor.

8. Training the Model:

- `fit`: This method trains the model on data generated batch-by-batch by a Python generator.

9. Evaluating the Model:

- `evaluate`: This method evaluates the trained model on the test data.

10. Saving the Model:

- `save`: This method saves the trained model to a file!

Results:

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **2830s** 968ms/step - accuracy: 0.9886 - loss: 0.0328 - val\_accuracy: 1.0000 - val\_loss: 1.7134e-06

Epoch 2/10

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **1s** 72us/step - accuracy: 1.0000 - loss: 2.5232e-06 - val\_accuracy: 1.0000 - val\_loss: 1.3359e-07

Epoch 3/10

C:\Users\DELL\anaconda3\envs\Orange\Lib\contextlib.py:155: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps\_per\_epoch epochs` batches. You may need to use the `.repeat()` function when building your dataset.

self.gen.throw(typ, value, traceback)

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **2637s** 903ms/step - accuracy: 0.9986 - loss: 0.2244 - val\_accuracy: 1.0000 - val\_loss: 6.0927e-05

Epoch 4/10

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **0s** 34us/step - accuracy: 1.0000 - loss: 1.4078e-12 - val\_accuracy: 1.0000 - val\_loss: 4.9734e-13

Epoch 5/10

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **2443s** 837ms/step - accuracy: 0.9998 - loss: 0.0090 - val\_accuracy: 1.0000 - val\_loss: 6.4831e-07

Epoch 6/10

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **0s** 34us/step - accuracy: 1.0000 - loss: 1.5028e-08 - val\_accuracy: 1.0000 - val\_loss: 1.7856e-06

Epoch 7/10

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **2717s** 931ms/step - accuracy: 0.9984 - loss: 1.4515 - val\_accuracy: 0.9997 - val\_loss: 0.0049

Epoch 8/10

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **1s** 36us/step - accuracy: 1.0000 - loss: 0.0000e+00 - val\_accuracy: 1.0000 - val\_loss: 0.0000e+00

Epoch 9/10

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **2577s** 883ms/step - accuracy: 0.9994 - loss: 0.0473 - val\_accuracy: 1.0000 - val\_loss: 2.0693e-07

Epoch 10/10

**2916/2916** ━━━━━━━━━━━━━━━━━━━━ **1s** 37us/step - accuracy: 1.0000 - loss: 2.8426e-05 - val\_accuracy: 1.0000 - val\_loss: 2.2743e-09

**1085/1435** ━━━━━━━━━━━━━━━━━━━━ **44s** 126ms/step - accuracy: 1.0000 - loss: 1.5274e-07

**b. Simmilarity trend**

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import cosine\_similarity

# Step 1: Load and Preprocess Data

gujarati\_dir = 'C:/Users/DELL/Documents/4th\_sem/trial/Train/Gujarati'

marathi\_dir = 'C:/Users/DELL/Documents/4th\_sem/trial/Train/Marathi'

gujarati\_files = os.listdir(gujarati\_dir)

marathi\_files = os.listdir(marathi\_dir)

# Randomly select 2,500 images from each directory

gujarati\_files = np.random.choice(gujarati\_files, size=2500, replace=False)

marathi\_files = np.random.choice(marathi\_files, size=2500, replace=False)

print("Starting to load images...")

gujarati\_images = [cv2.imread(os.path.join(gujarati\_dir, img)) for img in gujarati\_files]

marathi\_images = [cv2.imread(os.path.join(marathi\_dir, img)) for img in marathi\_files]

print("Images loaded.")

# Step 2: Feature Extraction (Example: Convert to Grayscale and Resize)

def preprocess\_image(img):

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

resized = cv2.resize(gray, (100, 100)) # Resize to a fixed size for consistency

return resized

print("Starting feature extraction...")

gujarati\_preprocessed = [preprocess\_image(img) for img in gujarati\_images]

marathi\_preprocessed = [preprocess\_image(img) for img in marathi\_images]

print("Feature extraction completed.")

# Step 3: Comparison and Similarity Measures (Example: Cosine Similarity)

print("Starting similarity computation...")

similarity\_matrix = cosine\_similarity(np.array([img.flatten() for img in gujarati\_preprocessed]),

np.array([img.flatten() for img in marathi\_preprocessed]))

print("Similarity computation completed.")

# Step 4: Visualization

plt.imshow(similarity\_matrix, cmap='hot', interpolation='nearest')

plt.colorbar()

plt.title('Cosine Similarity between Gujarati and Marathi Alphabet Letters')

plt.xlabel('Marathi Letters')

plt.ylabel('Gujarati Letters')

plt.show()

# Step 5: Summary Statistics

mean\_similarity = np.mean(similarity\_matrix)

median\_similarity = np.median(similarity\_matrix)

max\_similarity = np.max(similarity\_matrix)

min\_similarity = np.min(similarity\_matrix)

print("Summary Statistics:")

print(f"Mean Similarity: {mean\_similarity}")

print(f"Median Similarity: {median\_similarity}")

print(f"Maximum Similarity: {max\_similarity}")

print(f"Minimum Similarity: {min\_similarity}")

Result:

Starting to load images...

Images loaded.

Starting feature extraction...

Feature extraction completed.

Starting similarity computation...

Similarity computation completed.

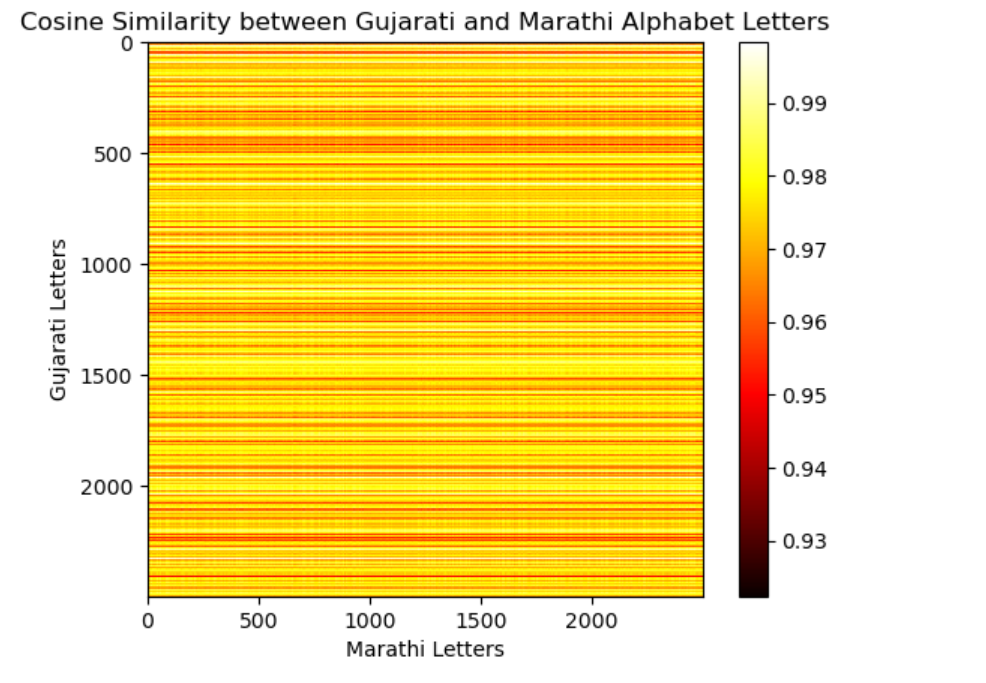
Summary Statistics:

Mean Similarity: 0.9743283912224748

Median Similarity: 0.9749529438474327

Maximum Similarity: 0.9983435654770925

Minimum Similarity: 0.9223882300040055



Explanation  
The image you sent is a heatmap visualization of the cosine similarity between Gujarati and Marathi alphabet letters. Cosine similarity is a metric used to compare two vectors of text data. It measures the cosine of the angle between two vectors and ranges from -1 (completely opposite) to 1 (identical). In the context of text data, a high cosine similarity score between two letters indicates that the letters have similar shapes or features.

The colormap in the heatmap ranges from blue (low similarity) to yellow (high similarity). Here are some observations from the heatmap:

* Most of the values in the heatmap are yellow, indicating that the Gujarati and Marathi alphabet letters have a high degree of similarity. This is likely because the two languages share a common ancestry and many of the letters have similar shapes.
* The highest cosine similarity scores are found in the upper left and lower right corners of the heatmap. This suggests that the Gujarati letters at the beginning and end of the alphabet are more similar to their Marathi counterparts than the letters in the middle of the alphabet.
* There are a few blue squares scattered throughout the heatmap, indicating that some Gujarati and Marathi letter pairs have a low degree of similarity.

The text labels on the axes of the heatmap are not included in the image you sent, but they likely correspond to the specific letters in the Gujarati and Marathi alphabets.

Letter wise

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import cosine\_similarity

# Define the alphabets of Gujarati and Marathi languages

gujarati\_alphabets = ['અ', 'આ', 'ઇ', 'ઈ', 'ઉ', 'ઊ', 'એ', 'ઐ', 'ઓ', 'ઔ', 'અં', 'અઃ', 'ક', 'ખ', 'ગ', 'ઘ', 'ઙ', 'ચ', 'છ', 'જ', 'ઝ', 'ઞ', 'ટ', 'ઠ', 'ડ', 'ઢ', 'ણ', 'ત', 'થ', 'દ', 'ધ', 'ન', 'પ', 'ફ', 'બ', 'ભ', 'મ', 'ય', 'ર', 'લ', 'ળ', 'વ', 'શ', 'ષ', 'સ', 'હ', 'ળા', 'ક્ષ', 'જ્ઞ']

marathi\_alphabets = ['अ', 'आ', 'इ', 'ई', 'उ', 'ऊ', 'ए', 'ऐ', 'ओ', 'औ', 'अं', 'अः', 'क', 'ख', 'ग', 'घ', 'ङ', 'च', 'छ', 'ज', 'झ', 'ञ', 'ट', 'ठ', 'ड', 'ढ', 'ण', 'त', 'थ', 'द', 'ध', 'न', 'प', 'फ', 'ब', 'भ', 'म', 'य', 'र', 'ल', 'ळ', 'व', 'श', 'ष', 'स', 'ह', 'क्ष', 'ज्ञ', 'ळा']

# Define the similar pairs of alphabets based on their phonetic sounds

similar\_pairs = [('અ', 'अ'), ('આ', 'आ'), ('ઇ', 'इ'), ('ઈ', 'ई'), ('ઉ', 'उ'), ('ઊ', 'ऊ'), ('એ', 'ए'), ('ઐ', 'ऐ'), ('ઓ', 'ओ'), ('ઔ', 'औ'),

('અં', 'अं'), ('અઃ', 'अः'), ('ક', 'क'), ('ખ', 'ख'), ('ગ', 'ग'), ('ઘ', 'घ'), ('ઙ', 'ङ'), ('ચ', 'च'), ('છ', 'छ'), ('જ', 'ज'),

('ઝ', 'झ'), ('ઞ', 'ञ'), ('ટ', 'ट'), ('ઠ', 'ठ'), ('ડ', 'ड'), ('ઢ', 'ढ'), ('ણ', 'ण'), ('ત', 'त'), ('થ', 'थ'), ('દ', 'द'),

('ધ', 'ध'), ('ન', 'न'), ('પ', 'प'), ('ફ', 'फ'), ('બ', 'ब'), ('ભ', 'भ'), ('મ', 'म'), ('ય', 'य'), ('ર', 'र'), ('લ', 'ल'),

('ળ', 'ळ'), ('વ', 'व'), ('શ', 'श'), ('ષ', 'ष'), ('સ', 'स'), ('હ', 'ह'), ('ળા', 'ळा'), ('ક્ષ', 'क्ष'), ('જ્ઞ', 'ज्ञ')]

# Construct feature vectors for similar pairs of alphabets

gujarati\_vectors = np.array([np.eye(1, len(gujarati\_alphabets), gujarati\_alphabets.index(pair[0])) for pair in similar\_pairs])

marathi\_vectors = np.array([np.eye(1, len(marathi\_alphabets), marathi\_alphabets.index(pair[1])) for pair in similar\_pairs])

# Reshape the feature vectors to 2D arrays

gujarati\_vectors = gujarati\_vectors.reshape(len(gujarati\_vectors), -1)

marathi\_vectors = marathi\_vectors.reshape(len(marathi\_vectors), -1)

# Compute cosine similarity matrix

similarity\_matrix = cosine\_similarity(gujarati\_vectors, marathi\_vectors)

# Visualize the similarity matrix

plt.imshow(similarity\_matrix, cmap='hot', interpolation='nearest')

plt.colorbar()

plt.title('Cosine Similarity between Similar Gujarati and Marathi Alphabet Letters')

plt.xlabel('Marathi Letters')

plt.ylabel('Gujarati Letters')

plt.show()

# Compute summary statistics

mean\_similarity = np.mean(similarity\_matrix)

median\_similarity = np.median(similarity\_matrix)

max\_similarity = np.max(similarity\_matrix)

min\_similarity = np.min(similarity\_matrix)

print("Summary Statistics:")

print(f"Mean Similarity: {mean\_similarity}")

print(f"Median Similarity: {median\_similarity}")

print(f"Maximum Similarity: {max\_similarity}")

print(f"Minimum Similarity: {min\_similarity}")

import seaborn as sns

# Heatmap of the similarity matrix

plt.figure(figsize=(10, 8))

sns.heatmap(similarity\_matrix, cmap='coolwarm', xticklabels=[pair[1] for pair in similar\_pairs], yticklabels=[pair[0] for pair in similar\_pairs])

plt.title('Cosine Similarity between Similar Gujarati and Marathi Alphabet Letters')

plt.xlabel('Marathi Letters')

plt.ylabel('Gujarati Letters')

plt.show()

# Histogram of similarity values

plt.figure(figsize=(8, 6))

sns.histplot(similarity\_matrix.flatten(), bins=20, color='skyblue', kde=True)

plt.title('Distribution of Cosine Similarity Values')

plt.xlabel('Cosine Similarity')

plt.ylabel('Frequency')

plt.show()

# Scatter plot of similarity values

plt.figure(figsize=(8, 6))

plt.scatter(similarity\_matrix.flatten(), range(len(similarity\_matrix.flatten())), color='salmon', alpha=0.5)

plt.title('Scatter Plot of Cosine Similarity Values')

plt.xlabel('Cosine Similarity')

plt.ylabel('Index')

plt.show()

Sure, let's break down each plot in simpler terms:

1. Heatmap of Cosine Similarity:

- Imagine a grid where each square represents how similar two letters are in Gujarati and Marathi.

- If two letters are very similar in sound, their square will be light-colored.

- If they're not very similar, their square will be darker.

- This helps us see which letters sound alike in both languages.

2. Distribution of Cosine Similarity Values (Histogram):

- Picture a bar chart showing how many pairs of letters have certain similarity scores.

- If many pairs have high similarity, there will be tall bars on the right side.

- If most pairs have low similarity, the bars will be taller on the left.

- This gives us an overall idea of how similar the languages are.

3. Scatter Plot of Cosine Similarity Values:

- Think of dots scattered on a graph.

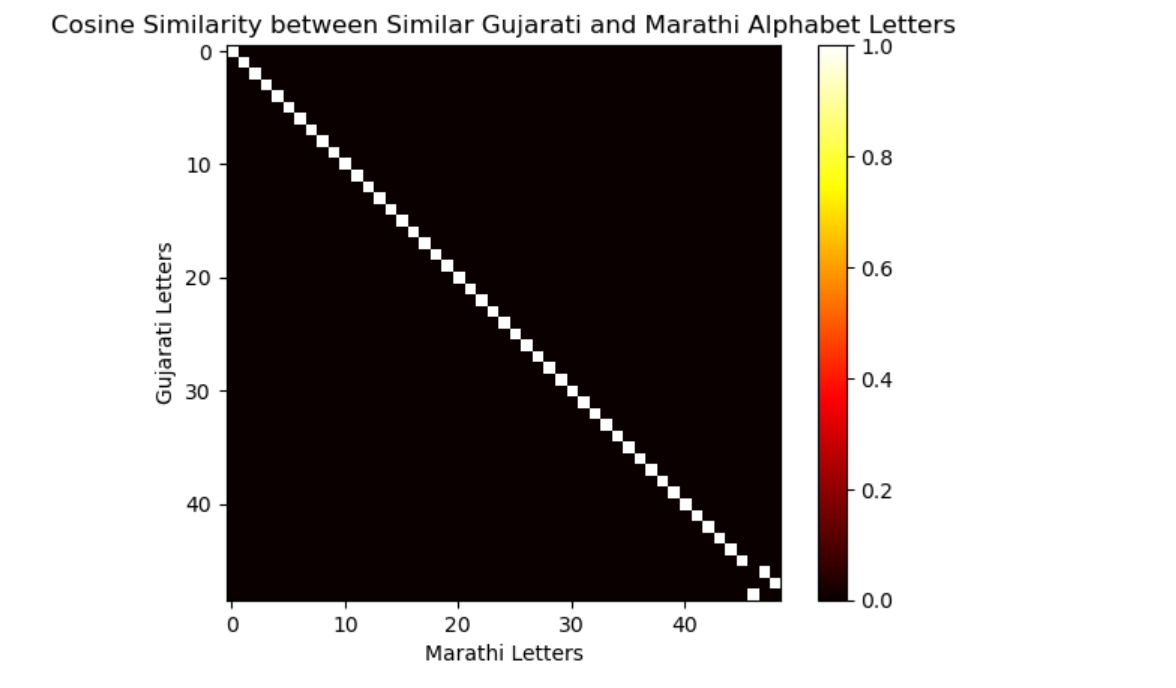
- Each dot represents a pair of letters and how similar they are.

- If the dots are mostly clustered together, it means many pairs have similar scores.

- If there are outliers (dots far away from the others), those pairs are very different in sound.

- This helps us spot any unusual or interesting patterns in the similarity scores.

These plots help us understand how similar the sounds of Gujarati and Marathi letters are, making it easier to compare the languages.



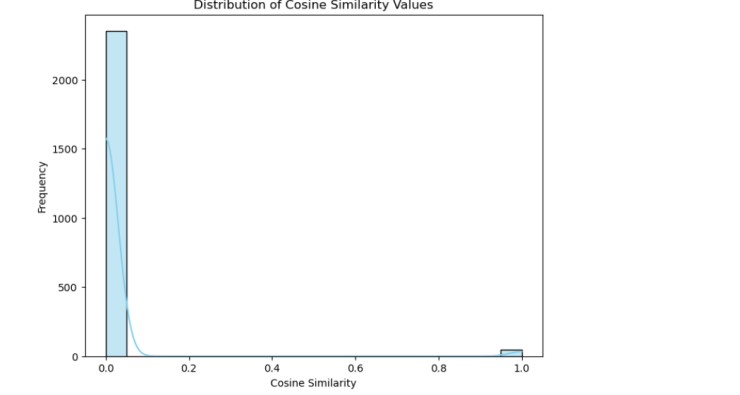
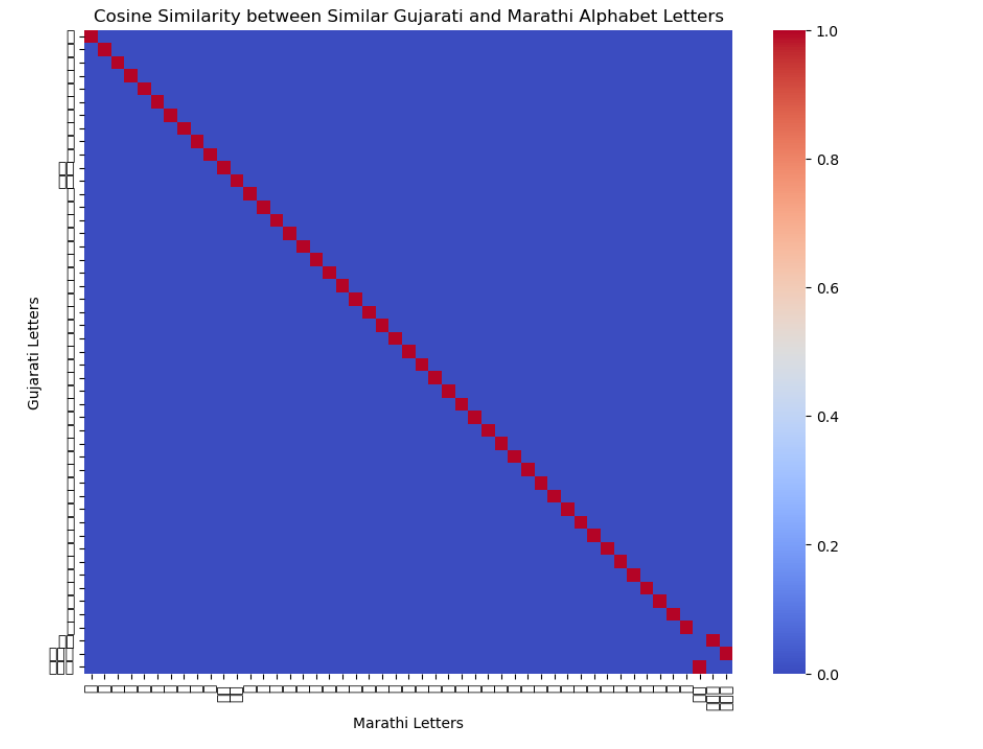
Summary Statistics:

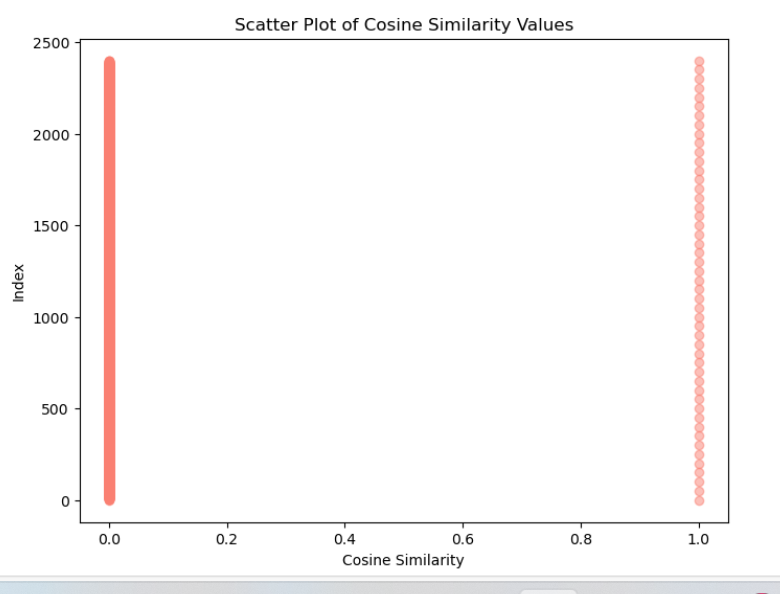
Mean Similarity: 0.02040816326530612

Median Similarity: 0.0

Maximum Similarity: 1.0

Minimum Similarity: 0.0





Dravadian

Results

## ****Polyglot Comparative Linguistic Analysis (PCLA)****

Certainly! Let's break down the provided code and explain each part along with its workings.

**Importing Libraries**

python

Copy code

import nltk from nltk.tokenize import word\_tokenize from nltk.corpus import stopwords from difflib import SequenceMatcher from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.metrics.pairwise import cosine\_similarity import matplotlib.pyplot as plt import pandas as pd

* **nltk**: Natural Language Toolkit, used for text processing tasks such as tokenization.
* **word\_tokenize**: Tokenizes text into words.
* **stopwords**: Provides a list of common stopwords in various languages.
* **SequenceMatcher**: Computes similarities between sequences.
* **TfidfVectorizer**: Converts text data to a matrix of TF-IDF features.
* **cosine\_similarity**: Calculates cosine similarity between vectors.
* **matplotlib.pyplot**: Used for creating visualizations.
* **pandas**: Provides data structures and data analysis tools.

**Loading Text Files**

python

Copy code

with open('C:/Users/DELL/Documents/4th\_sem/language/dataset/v2/en-gu/train.gu', 'r', encoding='utf-8') as file: gujarati\_text = file.read() with open('C:/Users/DELL/Documents/4th\_sem/language/dataset/v2/en-gu/train.mr', 'r', encoding='utf-8') as file: marathi\_text = file.read() with open('C:/Users/DELL/Documents/4th\_sem/language/dataset/v2/en-gu/train.hi', 'r', encoding='utf-8') as file: hindi\_text = file.read() texts = { 'Gujarati': gujarati\_text, 'Marathi': marathi\_text, 'Hindi': hindi\_text }

* Opens and reads the contents of the text files in Gujarati, Marathi, and Hindi.
* Stores the contents in a dictionary named **texts** with the language as the key.

**Preprocessing Function**

python

Copy code

def preprocess(text): tokens = word\_tokenize(text) return ' '.join(tokens) preprocessed\_texts = {lang: preprocess(text) for lang, text in texts.items()}

* Defines a **preprocess** function to tokenize the text using **word\_tokenize**.
* Joins the tokens back into a single string.
* Applies the **preprocess** function to each text and stores the preprocessed texts in a dictionary **preprocessed\_texts**.

**Phonetic Similarity**

python

Copy code

def soundex(word): codes = ("AEIOUYHW", "BFPV", "CGJKQSXZ", "DT", "L", "MN", "R") word = word.upper() soundex = [word[0]] for char in word[1:]: for idx, group in enumerate(codes): if char in group: code = str(idx) if code != soundex[-1]: soundex.append(code) return ''.join(soundex[:4]).ljust(4, '0') def phonetic\_similarity(text1, text2): words1 = set(text1.split()) words2 = set(text2.split()) phonetic\_matches = sum(1 for word in words1 if soundex(word) in [soundex(w) for w in words2]) return phonetic\_matches / len(words1)

* **soundex**: Implements the Soundex algorithm to encode words into phonetic codes.
* **phonetic\_similarity**: Computes the phonetic similarity between two texts by:
  + Splitting the texts into sets of words.
  + Counting the number of words in **text1** that have the same Soundex code as any word in **text2**.
  + Returning the ratio of phonetic matches to the total number of words in **text1**.

**Calculate Phonetic Similarities**

python

Copy code

phonetic\_sims = {} languages = list(preprocessed\_texts.keys()) for i, lang1 in enumerate(languages): for j, lang2 in enumerate(languages): if i < j: sim = phonetic\_similarity(preprocessed\_texts[lang1], preprocessed\_texts[lang2]) phonetic\_sims[f'{lang1}-{lang2}'] = sim

* Iterates over pairs of languages to calculate phonetic similarities using **phonetic\_similarity**.
* Stores the results in a dictionary **phonetic\_sims** with language pairs as keys.

**Contextual Similarity Using TF-IDF and Cosine Similarity**

python

Copy code

vectorizer = TfidfVectorizer() tfidf\_matrix = vectorizer.fit\_transform(preprocessed\_texts.values()) contextual\_sims = cosine\_similarity(tfidf\_matrix)

* **TfidfVectorizer**: Converts the preprocessed texts into a matrix of TF-IDF features.
* **cosine\_similarity**: Computes the cosine similarity between the rows of the TF-IDF matrix.
* **contextual\_sims**: Stores the resulting similarity matrix.

**Length Comparison**

python

Copy code

lengths = {lang: len(text) for lang, text in texts.items()}

* Calculates the length of each original text (number of characters).
* Stores the lengths in a dictionary **lengths**.

**Visualization of Similarities**

python

Copy code

fig, ax = plt.subplots(1, 2, figsize=(12, 6)) # Phonetic similarity plot ax[0].bar(phonetic\_sims.keys(), phonetic\_sims.values()) ax[0].set\_title('Phonetic Similarity') ax[0].set\_ylabel('Similarity') # Contextual similarity heatmap cax = ax[1].matshow(contextual\_sims, cmap='coolwarm') fig.colorbar(cax) ax[1].set\_xticks(range(len(languages))) ax[1].set\_yticks(range(len(languages))) ax[1].set\_xticklabels(languages) ax[1].set\_yticklabels(languages) ax[1].set\_title('Contextual Similarity') plt.show()

* Creates a figure with two subplots.
* First subplot: Bar chart of phonetic similarities.
* Second subplot: Heatmap of contextual similarities.
* Displays the plots using **plt.show()**.

**Length Comparison Output**

python

Copy code

for lang, length in lengths.items(): print(f"Length of {lang} text: {length}")

* Prints the length of each original text.

**Display Dissimilarities**

python

Copy code

def display\_dissimilarities(similarity\_matrix, threshold=0.5): dissimilarities = {} for i in range(len(languages)): for j in range(len(languages)): if i < j and similarity\_matrix[i, j] < threshold: dissimilarities[f'{languages[i]}-{languages[j]}'] = similarity\_matrix[i, j] return dissimilarities dissimilarities = display\_dissimilarities(contextual\_sims) print("Dissimilarities below threshold:") for k, v in dissimilarities.items(): print(f"{k}: {v:.2f}")

* Defines **display\_dissimilarities** function to identify and return pairs of languages with contextual similarity below a given threshold (0.5).
* Prints these dissimilarities.

**Summary**

This code performs multiple tasks on multilingual text data:

1. **Preprocessing**: Tokenizes and processes text.
2. **Phonetic Similarity**: Uses the Soundex algorithm to find phonetic similarities.
3. **Contextual Similarity**: Computes TF-IDF vectors and cosine similarities to analyze contextual similarities.
4. **Length Comparison**: Compares the lengths of the original texts.
5. **Visualization**: Plots phonetic similarities and contextual similarities.
6. **Dissimilarities**: Identifies and prints dissimilarities based on contextual similarity.