LANGUAGE DETECTION

PR PROJECT REPORT

* ABSTRACT:

Language detection investigates machine learning techniques for precise language detection, crucial in addressing multilingual content moderation challenges on social media platforms. By analyzing supervised and unsupervised algorithms like SVM, Naive Bayes, and Neural Networks alongside feature engineering methods, such as n-grams and character-level representations, the research aims to enhance language identification accuracy. Real-world social media datasets are utilized to evaluate these approaches, aiming to contribute solutions for more effective multilingual content filtering systems."

* INTRODUCTION:

Language detection is a crucial component in natural language processing (NLP) systems, aiming to identify the language of a given text. This task is fundamental in various applications such as text translation, sentiment analysis, and information retrieval. Our project focuses on developing a language detection system to accurately determine the language of diverse textual content.

* Block diagram:

**REAL-TIME LANGUAGE DETECTION**

**MODEL EVALUATION**

**MODEL TRAINING**

**TRAINING DATA SPLIT**

**FEATURE EXTRACTION**

**DATA PREPROCESSING**

**DATA COLLECTION**

**LANGUAGE IDENTIFICATION**

**OUTPUT**

**MONITORING AND UPDATING**

* methadology:

1.Data Loading and Exploration: Here we are loading the data from the dataset and analysing data by displaying it.

import numpy as np

import pandas as pd

import re

* Load the dataset and display basic information

df = pd.read\_csv('LanguageDetection.csv')

* Display the first few rows

df.head()

* Display information about the dataset

df.info()

2. Data Preprocessing: Here we are removing the extra spaces, duplicating words everything which are not needed.

* Extract features and labels, encode labels, and clean text data

X = df['Text']

y = df['Language']

* Encode labels using LabelEncoder

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

y = le.fit\_transform(y)

* Clean and preprocess text data

df\_list = []

for text in X:

text = re.sub(r'[!@#$(),n"%^\*?:;~`0-9]', '', text)

text = re.sub(r'[[]]', '', text)

text = text.lower()

df\_list.append(text)

3. Feature Extraction:

* Count Vectorizer:

CountVectorizer is one of the feature extraction techniques. It is used to convert a collection of text documents to a matrix of token counts. It transforms a list of text documents into a matrix of token counts, where each row represents a document, and each column represents a token in the list.

\*\*\* For the full process we have done using counter vectorizer feature extraction. But below we have tried different feature extraction techniques.

* Convert text data into numerical features using CountVectorizer

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer()

X = cv.fit\_transform(df\_list).toarray()

* Display the shape of the feature matrix

print(X.shape)

Here the accuracy is coming about 95 percent.

* Word Frequency:

Word frequency refers to the count of each unique word in a given text. It provides a simple representation of the importance or prevalence of words in a document, serving as a basic feature for text analysis.

* + Word Frequency

word\_vectorizer = CountVectorizer()

X\_train\_word = word\_vectorizer.fit\_transform(X\_train)

X\_test\_word = word\_vectorizer.transform(X\_test)

Here the accuracy is coming about 95 percent.

* Character N-grams:
* Character N-grams refer to contiguous sequences of N characters within a given text.
* For example, in the word "language," 2-grams (bigrams) would be "la," "an," "ng," "gu," "ua," and "ag." Similarly, 3-grams (trigrams) would include "lan," "ang," "ngu," "gua," and "uag."
* These character N-grams capture local patterns of characters in a text and are often used as features in natural language processing tasks like text classification or language detection.
* Character N-grams

char\_ngram\_vectorizer = TfidfVectorizer(analyzer='char', ngram\_range=(2, 5), max\_features=5000) (Adjust max\_features based on requirement)

X\_train\_char\_ngram = char\_ngram\_vectorizer.fit\_transform(X\_train)

X\_test\_char\_ngram = char\_ngram\_vectorizer.transform(X\_test)

Here the accuracy is coming about 98 percent.

\*\*\* Statistical feature extraction gives 0.01 accuracy which is why we do not prefer to use that feature extraction technique for language detection.

4. Train-Test Split: Here we will split the dataset into training and testing sets.

* Split the dataset into training and testing sets

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X, y, test\_size=0.2, random\_state=41)

5. Model Training and Evaluation: Here we will train the Train a logistic regression model and calculate its accuracy

Code:

The provided code uses logistic regression to classify languages based on statistical features extracted from text data. It calculates features such as mean, standard deviation, minimum, and maximum word lengths, creating a logistic regression model for language detection and evaluating its accuracy using the F1 score.

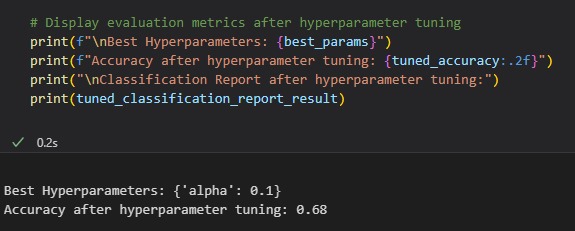
* Train a logistic regression model and evaluate its accuracy

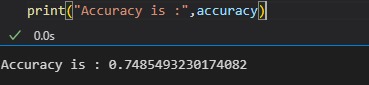
ada = LogisticRegression()

y\_pred = ada.fit(X\_train, y\_train).predict(X\_test)

ac = f1\_score(y\_test, y\_pred, average='macro')

print(f'Accuracy={ac:.2f}')





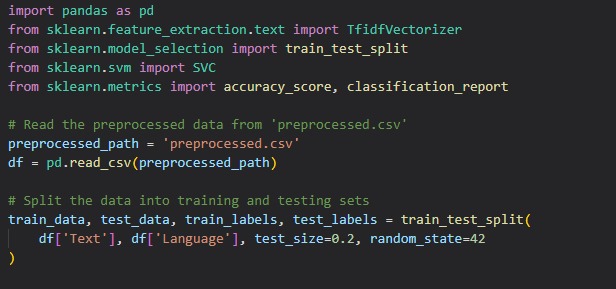
\*\*\*Here we are using a logistic regression classification model. Below are some of the classifications with which we can also do.

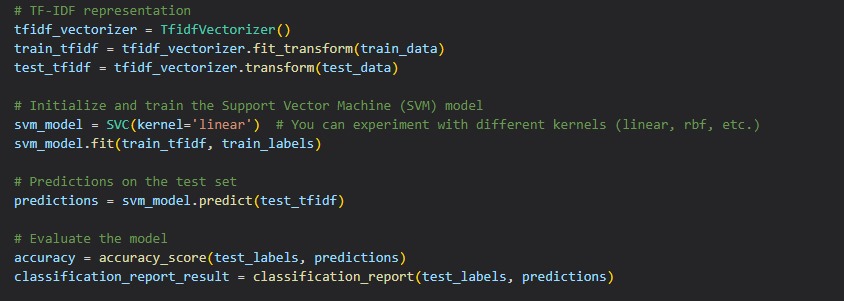
* SVM Classification model:

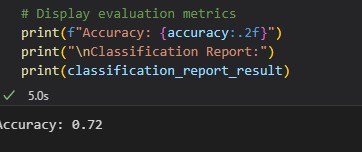
Support Vector Machine (SVM) is utilized in language detection to create a decision boundary that separates different languages in a feature space. It works by finding the hyperplane that maximally separates the language classes, making it effective for distinguishing between various linguistic patterns.

Code:

1. Import the SVM classifier (SVC), accuracy metrics (accuracy\_score), and data splitting functions (train\_test\_split).
2. Create an SVM model, train it on your feature set (X\_train), predict language labels for the test set (X\_test), and evaluate the accuracy of predictions using accuracy\_score. We can adjust kernel types and parameters based on your specific use case.





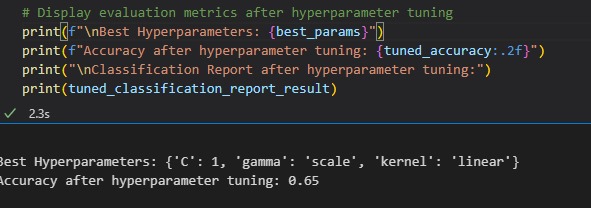


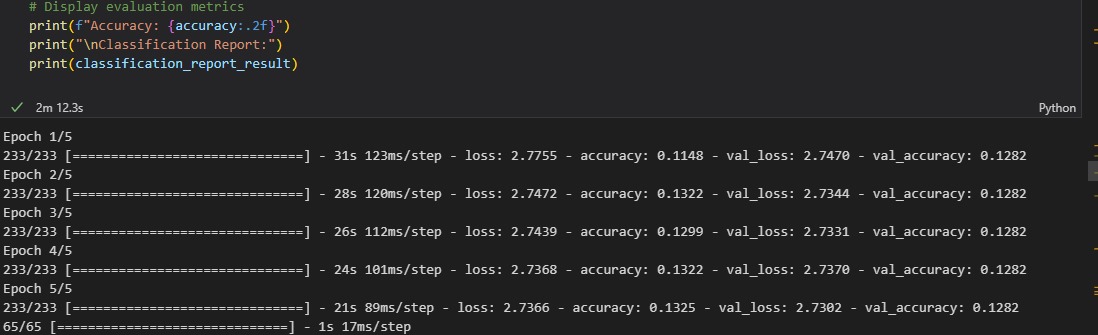
* Neural Networks:

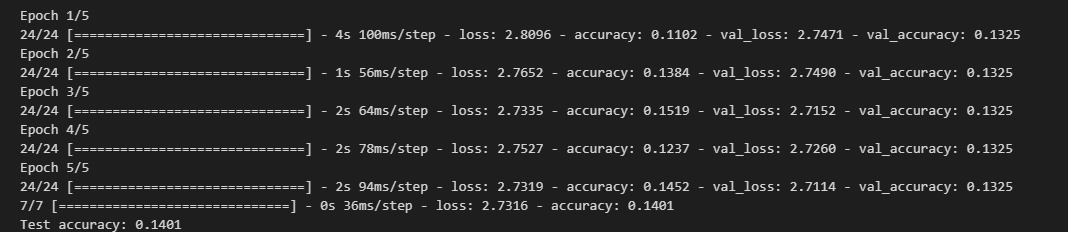
A neural network is a computational model inspired by the human brain's structure, composed of interconnected nodes (neurons) organized in layers. In the context of language detection, a neural network can be employed as a powerful tool for learning complex patterns in text data.

Code:

* Define a sequential neural network model using a deep learning library like TensorFlow and Keras.
* Specify the input layer based on the feature representation (e.g., TF-IDF vectors, word embeddings) and configure hidden layers with activation functions.
* Compile the model by specifying the optimizer, loss function, and evaluation metric. Train the model using the preprocessed data, adjusting the number of epochs and batch size. During training, the model learns to map input features to language labels. Evaluate the trained model on a separate test set to assess its performance. Monitor key metrics such as accuracy, precision, recall, or F1-score to gauge how well the model generalizes to new, unseen data.



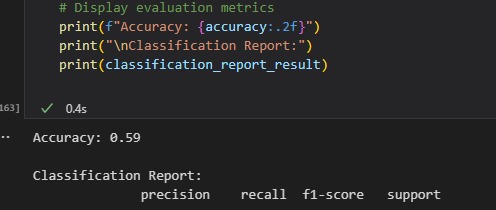




* Random Forest:
* Random Forest is an ensemble learning algorithm that builds multiple decision trees during training and merges their predictions to improve overall accuracy and control overfitting.
* A Random Forest model for language detection involves importing the necessary libraries, preprocessing text data, splitting it into training and testing sets, creating and training the Random Forest classifier, and evaluating its performance on the test set.
* The ensemble nature of Random Forest enhances classification accuracy compared to individual decision trees.

Code:

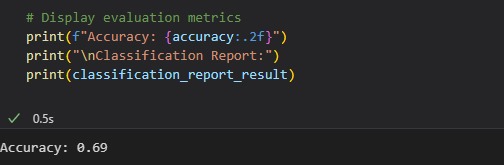
* Begin by importing essential libraries such as pandas for data manipulation, scikit-learn for machine learning tasks, and RandomForestClassifier for implementing the Random Forest algorithm.
* Load the dataset, preprocess the text data (e.g., using TF-IDF vectorization), and split it into training and testing sets. Create a RandomForestClassifier, fit it to the training data, and use it to predict the languages in the test set.
* Assess the model's performance by evaluating metrics like accuracy, precision, recall, or F1 score. This provides insights into how well the Random Forest model can classify languages based on the features derived from the text data



* Naive Bayes:
* Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem.
* In language detection, Naive Bayes calculates the probability of a given text belonging to each language class, and the class with the highest probability is assigned as the detected language.
* The Naive Bayes code typically involves vectorizing text data, training the model on labelled datasets, and then making predictions based on calculated probabilities.
* It is simple, computationally efficient, and often performs well in text classification tasks.

Code:

* The text data is preprocessed by removing unwanted characters, converting to lowercase, and using techniques like TF-IDF vectorization to transform the text into numerical features suitable for model training.
* A Multinomial Naive Bayes classification model is chosen for language detection. The model is trained on a labeled dataset, where each text is associated with a specific language label.
* After training, the model is used to predict the language of new texts. The predicted language is determined based on the class with the highest calculated probability, making Naive Bayes a probabilistic classifier.

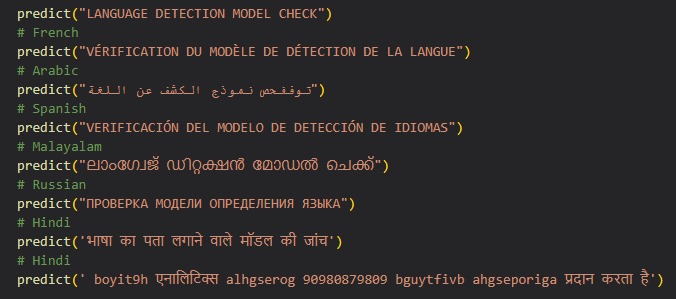


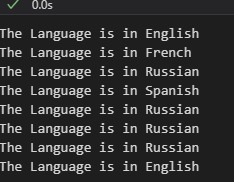
6.OUTPUT:

def predict(text):

    lang = model.predict([text])

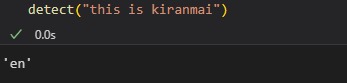
    print('The Language is in',lang[0])





\*\*This is the output if we use inbuilt function

(pip install langdetect

 from langdetect import detect) the output will be as follows

CASE-1: If we have to detects languages which are not present in our training dataset.

Solution:

* We will typically need a model that is capable of handling unseen classes.
* For language detection, you might want to consider using a more sophisticated model like a neural network-based approach or a pre-trained language model.
* In this response, we'll provide an example using the langid library, which is a simple language identification tool based on n-gram models.

Code:

This code uses the langid library to detect the language of each text in your dataset. The detected languages are then added to the dataframe in a new column called 'DetectedLanguage'.

pip install langid

import pandas as pd

import langid

# Load the preprocessed data (or replace with your data loading logic)

preprocessed\_path = 'preprocessed.csv'

df = pd.read\_csv(preprocessed\_path)

# Assume 'Text' is the column containing the text data

texts = df['Text'].tolist()

# Detect languages using langid

langid\_results = [langid.classify(text) for text in texts]

# Extract detected languages

detected\_languages = [lang for lang, \_ in langid\_results]

# Add detected languages to the dataframe

df['DetectedLanguage'] = detected\_languages

# Display the dataframe with detected languages

print(df[['Text', 'DetectedLanguage']])

import langid

# Example text to detect language

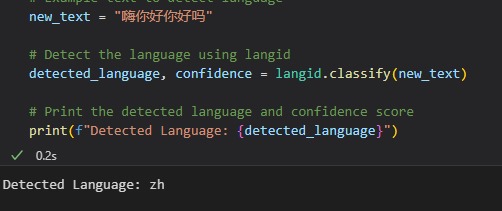
new\_text = "彼女は寝ています"

# Detect the language using langid

detected\_language, confidence = langid.classify(new\_text)

# Print the detected language and confidence score

print(f"Detected Language: {detected\_language}, Confidence: {confidence:.2f}")



“zh” represents the ISO 639-1 language code for Chinese

