LDS Model Documentation

In this documentation you will find the LD model used to test the data followed by the step-by-step explanation of the code.

## Language Detection System Code:

## 1. Imports

## ALL  
import re  
import nltk  
import joblib  
  
## AS  
import pandas as pd  
import numpy as np  
  
## FROM  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, classification\_report  
  
from datasets import load\_dataset

## 2. Data

### 2.1. Calling Data

# data = load\_dataset('masakhaner', 'yor')  
# data = load\_dataset('masakhane/masakhaner2', 'yor')

nigeria\_data\_file = "/Users/izzymohamed/Documents/WORK/AFRICA AGILE/Hackathon/Language Detection System/Data/Final/masakhane/all\_masakhane.csv"

data = pd.read\_csv(nigeria\_data\_file)

### 2.2. Preprocessing Data

df = pd.read\_csv(nigeria\_data\_file) #pd.DataFrame(data)  
  
# Preprocess the text data  
def preprocess\_text(text):  
 text = text.lower()  
 text = re.sub(r'[^\w\s]', '', text)  
 return text  
  
df['text'] = df['text'].apply(preprocess\_text)

### 2.3. Tokenization

# Tokenization (using NLTK tokenizer)  
nltk.download('punkt')  
df['text'] = df['text'].apply(nltk.word\_tokenize)

[nltk\_data] Downloading package punkt to  
[nltk\_data] /Users/izzymohamed/nltk\_data...  
[nltk\_data] Package punkt is already up-to-date!

# Convert list of tokens back to text  
df['text'] = df['text'].apply(lambda x: ' '.join(x))

### 2.4. Feature Extraction

# Feature Extraction using TF-IDF  
vectorizer = TfidfVectorizer()  
X = vectorizer.fit\_transform(df['text'])

### 2.5. Label Encoding

# Label Encoding  
label\_to\_id = {lang: i for i, lang in enumerate(df['language'].unique())}  
df['label'] = df['language'].map(label\_to\_id)  
y = df['label']

### 2.6. Splitting Data

# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

## 3. Training

# Train the Support Vector Machine (SVM) model  
model = SVC(kernel='linear')  
model.fit(X\_train, y\_train)

SVC(kernel='linear')

## 4. Predict

# Make predictions on the test set  
y\_pred = model.predict(X\_test)

# Evaluate the model  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Accuracy:", accuracy)  
  
print("Classification Report:")  
print(classification\_report(y\_test, y\_pred, target\_names=label\_to\_id.keys()))

Accuracy: 0.9950191570881226  
Classification Report:  
 precision recall f1-score support  
  
 yoruba 1.00 0.99 0.99 1691  
 hausa 1.00 0.99 1.00 1552  
 igbo 0.99 1.00 0.99 1977  
  
 accuracy 1.00 5220  
 macro avg 1.00 0.99 1.00 5220  
weighted avg 1.00 1.00 1.00 5220

## 5. Save Models

# Dir for output if we scrape directly to CSV  
# Make sure to create this folder  
  
directory = "/Users/izzymohamed/Documents/WORK/AFRICA AGILE/Hackathon/Language Detection System/" #TODO: CHANGE DIRECTORY

# Save the model to disk  
model\_filename = "language\_detection\_model.joblib"  
joblib.dump(model, model\_filename)

['language\_detection\_model.joblib']

## 6. Detect language of a new text

Now, to detect the language of a new text, you can use the trained model as follows:

1. Preprocess the new text using the same preprocessing steps applied to the training data.
2. Tokenize the preprocessed text using NLTK tokenizer.
3. Convert the list of tokens back to text.
4. Transform the text using the TF-IDF vectorizer.
5. Use the trained SVM model to predict the language label of the new text.

# Load the saved model from the disk  
model\_filename = "language\_detection\_model.joblib"  
model = joblib.load(model\_filename)

# New text to predict the language for  
new\_text = "Mo ti jade si ile"

# Preprocess the new text (similar to preprocessing in the training data)  
def preprocess\_text(text):  
 text = text.lower()  
 text = re.sub(r'[^\w\s]', '', text)  
 return text  
  
preprocessed\_text = preprocess\_text(new\_text)

# Tokenize the preprocessed text using NLTK tokenizer  
nltk.download('punkt')  
tokens = nltk.word\_tokenize(preprocessed\_text)

[nltk\_data] Downloading package punkt to  
[nltk\_data] /Users/izzymohamed/nltk\_data...  
[nltk\_data] Package punkt is already up-to-date!

# Convert the list of tokens back to text  
tokenized\_text = ' '.join(tokens)

# Transform the text using the TF-IDF vectorizer fitted on the training data  
X\_new = vectorizer.transform([tokenized\_text])

# Use the trained SVM model to predict the language label of the new text  
predicted\_label = model.predict(X\_new)[0]

# Convert the predicted label back to the original language  
label\_to\_id = {i: lang for lang, i in label\_to\_id.items()}  
predicted\_language = label\_to\_id[predicted\_label]

# Print the predicted language  
print("Predicted Language:", predicted\_language)

Predicted Language: yoruba

## Explanation Documentation for the provided code:

1. Imports:

The code begins by importing the required libraries and modules. These include standard Python libraries such as `re`, `nltk`, `joblib`, `pandas`, and `numpy`. Additionally, it imports specific components from the `sklearn` library, including `TfidfVectorizer` for feature extraction, `train\_test\_split` for splitting the dataset, `SVC` (Support Vector Classification) for training the SVM model, and `accuracy\_score` and `classification\_report` for evaluating the model. The code also imports the `load\_dataset` function from the `datasets` module, which is used for loading language datasets.

2. Data:

In this section, the code loads the data to be used for training the language detection model. There are two options for loading the data:

a. Option 2.1: Using the `load\_dataset` function from the `datasets` module to load a specific language dataset. However, this option is currently commented out using `#` symbols, meaning it is not being used.

b. Option 2.2: Loading the data from a CSV file located at the specified path in `nigeria\_data\_file`, The data is read into a pandas DataFrame named `df`,

3. Preprocessing Data:

The text data in the DataFrame is preprocessed in this section. The `preprocess\_text` function is defined to lowercase the text and remove any special characters using regular expressions (`re.sub`). The function is then applied to the 'text' column in the DataFrame using the `apply` method.

4. Tokenization:

In this step, the NLTK tokenizer is used to tokenize the preprocessed text. The NLTK `word\_tokenize` function is applied to each text in the 'text' column, resulting in tokenized sentences. The list of tokens for each sentence is then converted back to a space-separated string using the `lambda x: ' '.join(x)` function.

5. Feature Extraction:

The feature extraction step uses the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer from `sklearn`. The vectorizer converts the text data into numerical features suitable for machine learning algorithms. The `fit\_transform` method of the vectorizer is applied to the tokenized text in the 'text' column, and the resulting feature matrix `X` is obtained.

6. Label Encoding:

The 'language' column in the DataFrame contains language names ('Yoruba', 'Hausa', 'Igbo'). To convert these language names into numerical labels, a mapping of languages to integer labels is created using a dictionary comprehension. The 'label' column is then added to the DataFrame by mapping the language names to their respective integer labels.

7. Splitting Data:

The dataset is split into training and testing sets using the `train\_test\_split` function from `sklearn`. The `X\_train`, `X\_test`, `y\_train`, and `y\_test` variables contain the training and testing data for features and labels, respectively.

8. Training:

A Support Vector Machine (SVM) model with a linear kernel is used for training the language detection model. The `SVC` class is instantiated with the `kernel='linear'` parameter, and the model is trained on the training data using the `fit` method.

9. Predict:

The trained model is used to make predictions on the test set (`X\_test`). The predicted labels are stored in the `y\_pred` variable. The accuracy of the model is then calculated using the `accuracy\_score` function and printed.

10. Save Models:

The trained SVM model is saved to a disk file named "language\_detection\_model.joblib" using the `joblib.dump` function.

11. Detect Language of a New Text:

This section demonstrates how to use the saved model to predict the language of a new text. The process involves preprocessing the new text, tokenizing it, converting it to a numerical feature using the TF-IDF vectorizer fitted on the training data, and then using the SVM model to make predictions on the new feature. The predicted label is converted back to the original language using the mapping created earlier. Finally, the predicted language is printed.

In summary, the provided code performs the following tasks:

1. Loads language data from a CSV file or a language dataset (commented out).
2. Preprocesses the text data by converting it to lowercase and removing special characters.
3. Tokenizes the preprocessed text using NLTK tokenizer.
4. Extracts features from the text data using the TF-IDF vectorizer.
5. Encodes the language labels as numerical values.
6. Splits the dataset into training and testing sets.
7. Trains a Support Vector Machine (SVM) model on the training data.
8. Evaluates the model's accuracy and prints a classification report.
9. Saves the trained model to a file named "language\_detection\_model.joblib".
10. Provides a guide on how to use the saved model to predict the language of new text.