

Language Identifier

PYATA SUHRUDH - SE22UARI194
SRINIVAS SRIRIGIRI - SE22UARI167
AKHIL P - SE22UARI196

Abstract Introduction

The code implements a language identification model using the XLM-RoBERTa architecture to predict the language of sentences from the WiLI-2018 dataset. It begins by importing necessary libraries and loading the training split of the dataset into a Pandas DataFrame. It initializes the XLM-RoBERTa tokenizer and model for sequence classification, setting it up to handle multiple unique labels from the dataset. A preprocessing function tokenizes the sentences, truncating and padding them to a uniform length of 256 tokens. The dataset is then divided into training and testing subsets, followed by setting up training arguments such as the learning rate and batch sizes. A Trainer object is instantiated to manage the training process and is executed with `trainer.train()`. The last function defines a prediction function for the model to classify new text inputs for real-time language predictions. An example is shown where the functionality is demonstrated using a Hindi sentence to depict how well the model classifies languages based on textual content.

Prior related work

We experimented with different models such as n-gram models and naives bayes models before turning to transformers. The n-gram models were sufficiently simple and fast, but were held back by their bad performance, mainly accuracy. The naives bayes model on the other hand showed significant improvements over its predecessor. While the improvement in performance was good, it wasnt enough. Which is why we finally turned to use a transformer model. Based on our project requirements, we chose the XLM-RoBERTa model. The reason we first experimented with the other models was because previous studies have demonstrated that while transformers excel in accuracy, traditional methods still perform reasonably well and are faster in training and deployment.

Dataset

To train our models, we chose datasets from

- Kaggle
- huggingface
- commoncrawl
- WiLI-2018 Dataset

WiLI-2018 dataset is a collection of sentences in 50+ languages, designed specifically for language identification tasks.

It contains both short and long sentences, with labels corresponding to their language.

This dataset is split into train, dev and test sets for model evaluation in machine learning.

DATA PREPROCESSING

The Preprocessing Function Looks like this :-

We will create a function called preprocess_function.

Handling tokenization: This operation takes input examples (sentences) and applies the following

Tokenization: It converts sentences into sequences of token IDs.

Truncation: Truncates sentences longer than 256 tokens.

Padding: adds zeros for short sentences to length of 256 tokens.

```
tokenizer = XLMRobertaTokenizer.from_pretrained("xlm-roberta-base")
model = XLMRobertaForSequenceClassification.from_pretrained("xlm-roberta-base", num_labels=len(df['label'].unique()))
```

```
def preprocess_function(examples):
    return tokenizer(
        examples['sentence'],
        truncation=True,
        padding='max_length',
        max_length=256
    )
```

Methodology

1)Library Import:-

Import necessary libraries pandas : for data manipulation torch, for deep learning components from datasets and transformers libraries to handle datasets and models.

2)Data Inloading:

Use the load_dataset function to load the WiLI-2018 dataset, containing sentences in multiple languages with their labels. Use the training split of this dataset as a Pandas DataFrame to expose it and potentially act on it.

Methodology

3)Start Model and Tokenizer:

Initialize the XLM-RoBERTa tokenizer which converts input text into token IDs interpretable by the model.

Initialize the model XLM-RoBERTa developed for sequence

4) Classification: configured to accept a number of possible language labels based on the dataset.

Methodology

5) Data Preprocessing

Define a preprocessing function

that will do the following: take sentences, truncate to 256 tokens in length, then pad to have uniformity across input sequences.

This preprocessing is then applied to the entire training dataset in batches, tokenizing all examples efficiently.

6) Formatting for PyTorch:

Format the tokenized dataset for PyTorch and specify what columns are used during training- input IDs, attention masks, and labels.

Methodology

7) Training-Testing Split

Divide the already tokenized dataset into the ratio 80:20 for training and testing subsets to evaluate the model after training.

Training Configuration:

Set up training arguments like output directory, evaluation strategy for example evaluating at every epoch, learning rate, batch size for training and eval, number of epochs, and weight decay.

8) Training Models:

Creates a Trainer object with the model, training arguments, and datasets.

Call the function train() on a Trainer object to run a training process whereby the model learns to classify the language in sentences.

Methodology

9) Training-Testing Split

Divide the already tokenized dataset into the ratio 80:20 for training and testing subsets to evaluate the model after training.

Training Configuration:

Set up training arguments like output directory, evaluation strategy for example evaluating at every epoch, learning rate, batch size for training and eval, number of epochs, and weight decay.

10) Training Models:

Creates a Trainer object with the model, training arguments, and datasets.

Call the function `train()` on a Trainer object to run a training process whereby the model learns to classify the language in sentences.

Experiments

Naïve Bayes :-

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string

nltk.download('stopwords')

stemmer = PorterStemmer()

def preprocess_text(text):

    text = text.lower()

    text = text.translate(str.maketrans('', '', string.punctuation))

    stop_words = set(stopwords.words('english'))
    words = [word for word in text.split() if word not in stop_words]

    stemmed_words = [stemmer.stem(word) for word in words]
    return ' '.join(stemmed_words)

df['cleaned_text'] = df['sentence'].apply(preprocess_text)

print(df[['sentence', 'cleaned_text', 'label']].head())
```

```
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\Srinivas\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

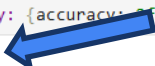
```
      sentence \
0  Klement Gottwaldi surnukeha palsameeriti ning ...
1  Sebes, Joseph; Pereira Thomas (1961) (på eng)....
2  भारतीय स्वातन्त्र्य आन्दोलन राष्ट्रीय एवम क्षे...
3  Après lo cort periòde d'establiment a Basilèa,...
4  ถนนเจริญกรุง (อักษรโรมัน: Thanon Charoen Krung...

      cleaned_text  label
0  klement gottwaldi surnukeha palsameer ning pai...  112
1  sebe joseph pereira thoma 1961 på eng jesuit s...  93
2  भारतीय स्वातन्त्र्य आन्दोलन राष्ट्रीय एवम क्षे...  83
3  après lo cort periòd destabli basilèa tornèt vi...  176
4  ถนนเจริญกรุง อักษรโรมัน thanon charoen krung &...  42
```


Experiments

```
testing data shape: (5000, 10000),  
[ ]: from sklearn.naive_bayes import MultinomialNB  
  
model = MultinomialNB()  
model.fit(X_train_vectorized, y_train)
```

```
•[5]: from sklearn.metrics import accuracy_score  
  
y_pred = model.predict(X_test_vectorized)  
  
accuracy = accuracy_score(y_test, y_pred)  
print(f'Accuracy: {accuracy:.2f}')  
  
Accuracy: 0.94
```



```
•[14]: def predict_language(text):  
        cleaned_text = preprocess_text(text)  
        vectorized_text = vectorizer.transform([cleaned_text])  
        prediction = model.predict(vectorized_text)  
        return prediction[0]  
  
new_texts = [  
    "Hello, how are you?",  
    "Bonjour, comment ça va?", # French (this may not be correctly identified without more training data)  
    "Hola, ¿cómo estás?", # Spanish (same as above)  
    "ब्रेकिंग न्यूज़, वीडियो, ऑडियो और फ़ीचर.?"  
]  
  
for text in new_texts:  
    lang_prediction = predict_language(text)  
    print(f'Text: "{text}" is predicted to be in language: {lang_prediction}')
```



```
Text: "Hello, how are you?" is predicted to be in language: 28  
Text: "Bonjour, comment ça va?" is predicted to be in language: 102  
Text: "Hola, ¿cómo estás?" is predicted to be in language: 72  
Text: "ब्रेकिंग न्यूज़, वीडियो, ऑडियो और फ़ीचर.?" is predicted to be in language: 60
```

Experiments

3-GRAM+4-GRAM Model:-

```
cleaned_text \
0 klement gottwalddi surnukeha palsameer ning pai...
1 sebe joseph pereira thoma 1961 på eng jesuit s...
2 भारतीय स्वातन्त्र्य आन्दोलन राष्ट्रीय एवम क्षे...
3 après lo cort periòd destabli basilèa tornèt vi...
4 ถอนแค้นญกรุง อักษรโรมัน thanon charoen krung l...
```

```
bigrams \
0 [(klement, gottwalddi), (gottwalddi, surnukeha),...
1 [(sebe, joseph), (joseph, pereira), (pereira, ...
2 [(भारतीय, स्वातन्त्र्य), (स्वातन्त्र्य, आन्दोलन...
3 [(après, lo), (lo, cort), (cort, periòd), (peri...
4 [(ถอนแค้นญกรุง, อักษรโรมัน), (อักษรโรมัน, than...
```

```
trigrams
0 [(klement, gottwalddi, surnukeha), (gottwalddi, ...
1 [(sebe, joseph, pereira), (joseph, pereira, th...
2 [(भारतीय, स्वातन्त्र्य, आन्दोलन), (स्वातन्त्र...
3 [(après, lo, cort), (lo, cort, periòd), (cort, ...
4 [(ถอนแค้นญกรุง, อักษรโรมัน, thanon), (อักษรโร...
```

Text: "ब्रेकिंग न्यूज़, वीडियो, ऑडियो और फ़ीचर" is predicted to be in language: Unknown Language

Can see that it is very Bad at Machine Learning?

Our Best :- BERT Model

```
[ ]: import pandas as pd
import torch
from datasets import load_dataset
from transformers import XLMRobertaTokenizer, XLMRobertaForSequenceClassification
from transformers import Trainer, TrainingArguments

data = load_dataset("MartinThoma/wili_2018")
df = pd.DataFrame(data['train'])

tokenizer = XLMRobertaTokenizer.from_pretrained("xlm-roberta-base")
model = XLMRobertaForSequenceClassification.from_pretrained("xlm-roberta-base", num_labels=len(df['label'].unique()))

def preprocess_function(examples):
    return tokenizer(
        examples['sentence'],
        truncation=True,
        padding='max_length',
        max_length=256
    )

tokenized_datasets = data['train'].map(preprocess_function, batched=True)

tokenized_datasets.set_format('torch', columns=['input_ids', 'attention_mask', 'label'])

train_test_split = tokenized_datasets.train_test_split(test_size=0.2)

training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy="epoch",
    learning_rate=3e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=1,
    weight_decay=0.01,
)
```

```

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_test_split['train'],
    eval_dataset=train_test_split['test'],
)

# Train the model
trainer.train()

# Example usage of the prediction function with new text input
def predict_language(text):
    inputs = tokenizer(
        text,
        return_tensors="pt",
        truncation=True,
        padding='max_length',
        max_length=256
    )
    with torch.no_grad():
        logits = model(**inputs).logits
        predicted_class_id = logits.argmax().item()
    return df['label'].unique()[predicted_class_id]

# Test with Hindi text input
new_texts = [
    "ब्रेकिंग न्यूज़, वीडियो, ऑडियो और फ़ीचर", # Hindi text example
]

for text in new_texts:
    lang_prediction = predict_language(text)
    print(f'Text: "{text}" is predicted to be in language: {lang_prediction}')

```

Output

After Some modification to the above given code

Without need of supercomputer we got this result.

Note:- This was output came on a very small dataset Limit. We set limit to 5000 whereas actually 230k are Present

Accuracy: 0.87

Text: "ब्रेकिंग न्यूज़, वीडियो, ऑडियो और फ़ीचर" is predicted to be in language: 60

Step	Training Loss
100	3.073300
200	2.551100
300	2.222700
400	2.050900
500	1.962600
600	1.889300
700	1.856300
800	1.818400

IMPORTANT INFO :-

- Each Language prediction output is based on a number
If 60==Hindi (60 represents Hindi)

THANKS