Language Identification Challenge for TEDx Talks

December 1, 2021

1 Introduction

Language Identification (LID) systems from voice are classification models that predict the spoken language from a given audio recording. The LID systems can facilitate the process of any speech processing system such as speech recognition (ASR) or speech translation systems. In speech-based assistant systems, LID works as a first step by selecting the appropriate grammar from a list of available languages for further semantic analysis. Also, these models can be employed in call centers in order to redirect an international user to an operator who is fluent in that identified language.

The objective of this project is to use machine learning methods for constructing a LID model which can discriminate 4 languages; English, French, Arabic, Japanese. There are 3 expected phases. The first phase is the data collection. It is expected to prepare the dataset by recording audio signals. The objective is to get familiar with the challenges of data collection, and understand the trade-off between the cost of having more data and the improvement of performance. The second phase is constructing a classifier. It is expected to compare the performance of different models, optimize the hyperparameters, and practice of finding the best model. The third phase is the evaluation of the model in a simulated situation of real life deployment. The objective is to understand the challenge of generalization. It's also expected to analyze the result of the model's performance and make hypothesis about the weak and strong aspect of models. The competition among models' accuracy can provide a better understanding of performance.

2 Data collection

It is proposed to collect the data from TEDx talks YouTube for the Language Identification task from audio. The samples have to be a recorded audio of speaker speech from available TEDx talks videos. In order to have a standard sample's type, it is expected to respect the following convention.

- The length of recorded audio files should be around 5 seconds (5.00 - 5.99 seconds).
- The format of audio files should be **.wav*.
- The sample rate of recording files should be 16 kHz (in mono format).

You can find a playlist of TEDx talk in different language in below links:

- English
- French
- Arabic
- Japanese

For recording the audio samples, you can use phone with respect to the asked convention. It is suggested to record the speech file from a part of video that contain speech with the lowest background noise possible. Also, be careful to record your file in an as silent as possible environment. The following app allows recording, trim, and export audio file in desired format.

- Android : Voice Recorder

- iOS : Hokusai Audio Editor
- Windows : Audacity
- Linux : Audacity

In order to track the collected data it is expected to create a text file with 3 information of samples; sample's file name, TEDx talk video address, the starting time of recording, speaker's language. It would be suggested to at least record 5 samples per language. The recorded samples will be collected and a part of it would be shared with others.

You can find an example of data with 3 samples in each language. More samples will be provided after collecting samples from others.

The data should be recorded and uploaded in the UMTICE **before the second session**. Like the data example that is provided, You should upload a compressed folder contains your recording files (at least 5 files per languages) in asked format (*.wav, 16kHz, mono, 5-6 seconds) and a *.txt file same as info.txt in provided folder with 4 information for each recorded file (one file per line); The 1st column should be the name of *.wav file, the 2nd column should be the URL address of YouTube video, the 3rd column should be the starting time of recording from YouTube video, 4th column the label (language) of recorded speech.

3 Classifier

After collecting enough data (or even in mean time of extending training dataset), it is expected to write a script for feature extraction and training a classifier model.

3.1 Feature extractor

By using librosa (more information here) different time domain and frequency domain features can be extracted. librosa is a python package for audio and music analysis.

```
import librosa

# sr should be set to your recording sample rate (16k)
x,freq = librosa.load("[your_wav_files_directory]/FR_001.wav",sr=16000)
# The load function will return a time series value (x) and
#   the input sample rate (freq) which is 16000

print("The duration of FR_001.wav in seconds:",len(x)/freq)
```

It would be proposed to use Mel-Frequency Cepstral Coefficients (MFCC) which is a short term spectral feature They are commonly used to extract important information from a voice signal. MFCC can be extracted by *librosa.feature.mfcc()* as follows.

```
# This function will return n_mfcc number of MFCC per
#     a window of time in audio time series
x_mfcc=librosa.feature.mfcc(x,sr=freq, n_mfcc=40)
print(x_mfcc.shape)
# x_mfcc is an array with 40 values for a window of time
# The len(x_mfcc) is a proportion of wav file duration
```

You can find more information about MFCC feature and other types of feature such as Root Mean Squared Energy, Spectral Centroid, Zero Crossing Rate, etc. in this link that can be used in this case. It is also possible to extract several types of feature and concatenate them.

The length of extracted x_mfcc is a proportion of wav file duration. So it means the by having audio files with different duration (5.00 - 5.99 seconds) the length of extracted array would be varied. In the case of LID problem, several methods for feeding features to model can be suggested.

- Computing statistics from time series data : By computing mean, variance, median, etc. it is possible to summarize and convert a list of values with variable length to one (or certain number of) value. In this method, the information related to the time will be lost.

- Converting sample to a fixed length : The audio time series data can be cut to a fix length (such as minimum length of 5.00 seconds) which called *Sequence Truncation*. Another method to have a fix length sequence is padding to a maximum length (for example 6.00 seconds) by concatenating a constant value such as 0 to the end (or beginning) of the sequence. These two model will keep the information related to the time in the time series data.

Although there are some machine learning models that can be fed by a non-fixed length input such as HMM, RNN or LSTM models. But to start, it would be suggested to convert data to a fixed length and use a simple model for classification.

3.2 Classical Machine learning classifier with Sklearn

Based on your extracted features, you can design a classification model. Here is an example with mean, variance of 20 MFCC feature from samples (the function called *feature_extractor*) with random forest classifier in sklearn.

```
from sklearn.ensemble import RandomForestClassifier
import csv
import numpy as np
import random

def feature_extractor(audio_file_dir):

    #load the audio files
    x,freq = librosa.load(audio_file_dir,sr=16000)
    #extract 20 MFCCs
    mfcc=librosa.feature.mfcc(x,sr=freq,n_mfcc=20)
    #calculate the mean and variance of each MFFC
    mean_mfccs=np.mean(mfcc,axis=1)
    var_mfccs=np.var(mfcc,axis=1)
    #return mean and variance as the audio file feature
    return list(mean_mfccs)+list(var_mfccs)

#set data_dir to the directory of your data files
data_dir= "TP_data/"

# Read file info file to get the list of audio files and their labels
file_list=[]
label_list=[]
with open(data_dir+"0Info.txt", 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        # The first column contains the file name
        file_list.append(row[0])
        # The last column contains the lable (language)
        label_list.append(row[-1])

# create a dictionary for labels
lang_dic={'EN':0,'FR':1,'AR':2,'JP':3,}

# create a list of extracted feature (MFCC) for files
x_data=[]

for audio_file in file_list:
    file_feature = feature_extractor(data_dir+audio_file)
    #add extracted feature to dataset
    x_data.append(file_feature)

# create a list of labels for files
y_data=[]
for lang_label in label_list:
    #convert the label to a value in {0,1,2,3} as the class label
    y_data.append(lang_dic[lang_label])

# shuffle two lists
temp_list = list(zip(x_data, y_data))
random.shuffle(temp_list)
x_data, y_data = zip(*temp_list)
```

```
# create Random Forest model from sklearn to classify languages of audio files
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(x_data, y_data)
# the resulted accuracy is on a small set which is same for train and test
print("Accuracy",clf.score(x_data, y_data))
```

3.3 Neural Network Classifier with Pytorch

Here is another example when the Sequence Truncation is used on 20 MFCC feature to train a multi-layer perceptron classifier in PyTorch.

- By changing *feature_extractor* function, a fixed length sequence (5 seconds) of 20 MFCC features using sequence truncation can be obtained.

```
def feature_extractor(audio_file_dir):

    #load the audio files
    x,freq = librosa.load(audio_file_dir,sr=16000)
    # trim the first 5 seconds (Sequence Truncation)
    x_5sec=x[:length_of_5seconds]
    # extract 20 MFCCs
    mfccs_5sec=librosa.feature.mfcc(x_5sec,sr=freq,n_mfcc=20)
    # return mfcc of the first 5 sec as the audio file feature
    return mfccs_5sec
```

- Thanks to *torch.utils.data.DataLoader*, it is possible to create a dataloader to manage the feeding process of samples. In this code we defined only one dataloader which will be used for train and test. But by having enough samples and splitting them into sets, two separate dataloaders can be defined and employed for train set and test set.

```
from torch.utils.data import TensorDataset, DataLoader
import torch

# transform to torch tensor
tensor_x_data = torch.Tensor(x_data)
tensor_y_data = torch.Tensor(y_data)
# create your datset
dataset = TensorDataset(tensor_x_data,tensor_y_data)
# the batch size can be changed to a lrger value when you have more data
batch_size = 1
# create your dataloader
dataloader = DataLoader(dataset, batch_size=batch_size)
```

- By using the below code, a neural network model (MLP) would be designed. The model will get a Flatten format (convert 2d array to 1d array, more information here) of extracted features and passes through 4 fully connected layers. In the last lines, the loss function has been set to *Cross Entropy* and stochastic gradient descent (SGD) with learning rate of 0.001 has been set as optimizer.

```
from torch import nn

# Get cpu or gpu device for training.
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Using {} device".format(device))

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            # the size of input should be the number of features (20 MFCC) times
            #       length of sequence (157)
            nn.Linear(20*157, 512),
            nn.ReLU(),
```

```
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 4)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)

# to train a model, we need a loss function and an optimizer.
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

- By defining two functions as the train and test, the training and evaluation and displaying the log of process can be facilitated.

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y.type(torch.LongTensor))

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y.type(torch.LongTensor)).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss
                                                    :>8f} \n")
```

- Now the training and evaluation process can be done by few below lines for a given number of epochs.

```
number_of_epochs = 10
for t in range(number_of_epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(dataloader, model, loss_fn, optimizer)
    test(dataloader, model, loss_fn)
print("Done!")
```

3.4 Optimization

It is expected to create, evaluate and optimized classification models. In order to prepare a model, the below modification can be investigated to propose the best model.

- The impact of using different features types, different methods of fixing audio time series, using a feature selection process such as PCA.
- Comparing different type of machine learning models, changing the models' hyperparameters such as number of layers, optimizer, learning rate, batch size, number of epoch in the neural network model.
- The impact of input dataset size by preparing more data and compare the performance. The number of samples per classes can be change to have unbalance class deliberately and study changes on the confusion matrix.

4 Evaluation

Finally, the best model can be saved and load by using the below code.

```
# saving models' weights as a state dictionary (containing the model parameters)
torch.save(model.state_dict(), "model.pth")

# creating model's graph
model = NeuralNetwork()
# load weights from saved directory
model.load_state_dict(torch.load("model.pth"))
```

By collecting audio files, a reasonable number of audio samples would be available. A part of these samples which has been recorded from different video with different recorder would be shared to be added to your training data. But the majority of data will be used as evaluation set and comparing of each student's best model. The evaluation set will be provided to all without target labels, and models should predict the language of samples. This below code will help you to save the predicted target labels.

```
#set data_dir to the directory of your data files
data_dir= "TP_data/test/"

# Read file info file to get the list of audio files and their labels
file_list=[]
label_list=[]
with open(data_dir+"1Info.txt", 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        # The first column contains the file name
        file_list.append(row[0])

class2lang_dic={0:"EN",1:"FR",2:"AR",3:"JP"}

for test_sample in file_list:
    test_sample_feature=feature_extractor(test_sample)
    model.eval()
    with torch.no_grad():
        #test_sample_feature is the extracted feature of a given audio file
        pred = model(test_sample_feature)
        predicted_lang = class2lang_dic[pred[0].argmax(0)]
        print(f'Predicted class: "{predicted}"')
        # save the predicted output in Output_evaluation.txt
        with open(data_dir+"Output_evaluation.txt",'a+') as file:
            file.write(f"{test_sample},{predicted}\n")
```

The best models will get extra points.

A report and codes should be prepared for evaluation of this practical. It is highly recommended to work and prepare the report in a group of two persons.

5 More information

- Feature extraction using librosa
- Pytorch tutorials
- Paper with code Bartz, C., Herold, T., Yang, H., Meinel, C. (2017, November). Language identification using deep convolutional recurrent neural networks. In International conference on neural information processing (pp. 880-889). Springer.
- A project report for music classification
- An example code for classification of animal sound