Task : Implement a method to identify the language a document is written in .

Dataset Used : http://www.statmt.org/europarl/

I cleaned the data, generated a csv file from each language corpus and then merged these csv files to create a single (multi-label) dataset, so that we can use it in the supervised training.

Drive Link (Data, Predictions and Models) : https://drive.google.com/drive/folders/1UWe1KH3Hyppc1U52b13k_v7P1uRwt16e?usp=sharing

Models Implemented

I have implemented 2 models :

1) LSTM Model : I have used a 2 layer LSTM model in which the 1st layer ia a Bi-directional LSTM and the 2nd layer is a normal LSTM layer.

Using the following model I obtain the following results on the **Validation** set :

Precison : 99.8%

Recall : 99.6%

F1 Score : 99.7%

epoch	train_loss	valid_loss	accuracy	time
0	3.174304	3.174028	0.991781	35:04
1	3.174232	3.174012	0.993948	35:08
2	3.174252	3.173991	0.996563	35:07
3	3.174207	3.173995	0.995737	35:06
4	3.174287	3.174070	0.968439	35:21
5	3.174207	3.173987	0.995023	35:21
6	3.174199	3.173990	0.993714	35:26
7	3.174184	3.173987	0.994759	35:24
8	3.174258	3.173986	0.996451	35:14
9	3.174238	3.173986	0.997149	35:01

```
----Ending Training--------
 Starting Validating Model:
                   recall precision  f1-score   support

<OOV>              0.9980    1.0000    0.999010372344.0000
label              0.0000    0.0000    0.0000    0.0000
en                 0.9985    0.9735    0.985820828.0000
nl                 0.3570    0.0488    0.0858 1000.0000
da                 0.0490    0.0469    0.0479 1000.0000
sv                 0.0280    0.0504    0.0360 1000.0000
pt                 0.0000    0.0000    0.0000 1000.0000
es                 0.0460    0.0510    0.0484 1000.0000
it                 0.3300    0.0448    0.0788 1000.0000
fr                 0.0000    0.0000    0.0000 1000.0000
de                 0.0760    0.0560    0.0645 1000.0000
el                 0.0100    0.0391    0.0159 1000.0000
bg                 0.0958    0.0522    0.0676  992.0000
fi                 0.0000    0.0000    0.0000 1000.0000
cs                 0.0020    0.0153    0.0035 1000.0000
sl                 0.0000    0.0000    0.0000 1000.0000
lt                 0.0000    0.0000    0.0000 1000.0000
et                 0.0000    0.0000    0.0000 1000.0000
lv                 0.0000    0.0000    0.0000 1000.0000
ro                 0.0000    0.0000    0.0000  979.0000
pl                 0.0000    0.0000    0.0000  928.0000
sk                 0.0000    0.0000    0.0000 1000.0000
hu                 0.0000    0.0000    0.0000  929.0000

avg / total        0.9961    0.9980    0.997010414000.0000

Accuracy: 0.9960656808142885
```

Due to resource constraints, I was able to train the model for 10 epochs only. If we train the model for more number of epochs and tune the hyperparameters properly, we'll definitely obtain better results.

Moreover, due to RAM constraints, I limited the maximum sentence length, on which the model is trained, to 500 (the maximum sentence length for the dataset is 15062). Nevertheless, the model was able to learn the structural composition of the words and phrases as well the dependencies between the words in a sentence and gave good results.

For more details and to see the working of the model, please refer to :
Language_detection_bi-lstm.ipynb (The downloaded ipynb file has also been provided by me.)

2) FastText Classifier Model :

FastText is another word embedding method that is an extension of the word2vec model. But, instead of learning vectors for words directly, fastText represents each word as an n-gram of characters. This helps capture the meaning of shorter words and allows the embeddings to understand suffixes and prefixes. Once the word has been represented using character n-grams, a skip-gram model is trained to learn the embeddings. This model is considered to be a bag of words model with a sliding window over a word because no internal structure of the word is taken into account. As long as the characters are within this window, the order of the n-grams doesn’t matter.

FastText also works well with rare words. So even if a word wasn't seen during training, it can be broken down into n-grams to get its embeddings.

Reference : https://fasttext.cc/docs/en/supervised-tutorial.html

FastText will generate two files during training:

1) a bin file: this is the learned model which contains the optimized parameters for predicting the language label from a given text.
2) a vec file : a text file that contains the learned vocabulary (around 1.8million) and their embeddings.

Using FastText, I obtain the following results on the **Validation** set :

- **Precision** : 99%
- **Recall** : 99%
- **F1 Score** : 99%

and the following results on the **Test** set :

- **Precision** : 99.3%
- **Recall** : 99.3%
- **F1 Score** : 99.3%

For more details and to see the working of the model, please refer to :
Language_detection_fasttext.ipynb (The downloaded ipynb file has also been provided by me.)