Language Detection using Neural Network

By CodeCube

Authors

Neel Desai

Nick Kornienko

Mansi Vekariya

Kelly Nguyen

Abstract

Language detection is a critical technology in today's globalized and digitally interconnected world. With over 7,000 languages spoken globally, the ability to accurately identify the language of speech is the first step in facilitating communication across language barriers. Our project aimed to develop a robust language detection model capable of distinguishing between English, French, and Spanish — the three most commonly used languages worldwide.

Adhering to the CRISP-DM methodology, our project began by setting clear success criteria: achieving an overall accuracy of 90% for the model. The data-gathering phase involved collecting over 50 hours of verified and recorded audio for each of the three languages. We then processed this data to extract 40 Mel Frequency Cepstral Coefficients (MFCCs) from each audio sample, providing a comprehensive set of features for training our neural network models.

Our approach involved experimenting with two types of neural networks: Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). RNNs are known for their effectiveness in handling sequential data, making them a logical choice for audio processing. CNNs, recognized for their ability to extract hierarchical features, were also considered due to their proficiency in dealing with pattern recognition tasks.

After extensive experimentation with various architectures of RNNs and CNNs, we discovered that a 5-layer 1D CNN model yielded the most promising results for our language detection task. This model architecture effectively leveraged the MFCC features to distinguish between the languages.

The culmination of our efforts resulted in the development of a highly accurate language detection model. Our final model surpassed our initial target, achieving an overall accuracy of 93%. It also demonstrated a remarkable precision rate of over 90% in identifying each of the three languages: English, French, and Spanish. Beyond the development of the model, we successfully deployed it on Azure ML, making it accessible to users through APIs. This deployment ensures wider applicability and ease of integration into various platforms and applications. The achieved level of performance not only met our project objectives but also underscored the efficacy of advanced neural networks in language detection tasks within a multilingual context.

Introduction

In an era of unprecedented globalization and digital connectivity, the ability to understand and communicate across languages is more crucial than ever. With over 7,000 languages spoken worldwide, the challenge of language detection — accurately identifying the spoken language — stands as a significant hurdle in bridging communication gaps. This project focuses on tackling this challenge by developing an advanced language detection model capable of running on inexpensive computational hardware. Our goal was to create a model capable of distinguishing between three of the most commonly used languages in the world: English, French, and Spanish. The importance of this endeavor lies not only in its immediate applicability in various fields such as translation services, content localization, and international communication but also in its potential to foster greater understanding and collaboration across linguistic boundaries.

The project was underpinned by the CRISP-DM methodology, a structured approach that guided us from understanding the business problem to deploying a fully functional model. Recognizing the complexity of spoken language and the nuances distinguishing different languages, we set an ambitious target for our model: to achieve an overall accuracy of 90%. To this end, we collected and processed a substantial dataset comprising over 50 hours of verified audio recordings for each language. The dataset was enriched with 40 Mel Frequency Cepstral Coefficients (MFCCs) for each audio sample, providing a robust foundation for our model training.

Our exploration led us to experiment with two types of neural networks known for their proficiency in handling audio data: Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). After rigorous testing and optimization of various network architectures, we found that a 5-layer 1D CNN model delivered the best performance. The success of our model was marked not just by its ability to meet our initial accuracy goal but also by exceeding it. We achieved an impressive overall accuracy of 93%, with each language being identified with over 90% precision. This result not only validates the effectiveness of our approach but also underscores the potential of neural network-based models in the field of language detection and beyond.

Related Work

Spoken Language Recognition using CNN[1]

This study explores the use of Convolutional Neural Networks (CNNs) for spoken language recognition, focusing on German, English, and Spanish. The research utilizes a balanced dataset from LibriVox recordings, emphasizing language features over specific voices and enhancing data with pitch, speed, and noise variations. The CNN model, trained with carefully pre-processed audio, shows high performance in recognizing languages, attributed to the diverse and well-distributed training data and the exclusion of training set speakers from the validation set.

Just like this paper we also intend on using CNN for language recognition, however, the set of languages that we are focusing on is different. Instead of using LibriVox recordings, we are using Mozilla Common Voices corpus. They have done a number of preprocessing steps and generated filter banks and performed the modelling on the banks, in contrast, we only rely on MFCCs for modelling and inference, their weighted average is 92% and ours is 93%.

Spoken Language Identification System Using Convolutional Recurrent Neural Network[2]

The article presents a spoken language identification system that utilizes a hybrid Convolutional Recurrent Neural Network (CRNN) architecture. This system is designed to identify seven languages, including Arabic, using the Mozilla Common Voice (MCV) corpus. It emphasizes feature extraction using both Gammatone Cepstral Coefficients (GTCC) and Mel Frequency Cepstral Coefficients (MFCC), individually and in combination. The system inputs speech signals as frames into the CRNN, with results showing that the combined GTCC and MFCC features yield higher performance than using either feature set alone. The system achieves an average accuracy of 92.81% in its best-performing experiment. In contrast, our model is designed to handle just 3 languages, we have used the same data source, and we are only using MFCC as feature and our average accuracy is 93%.

Data

For building our model, we needed validated and correctly labelled recorded audio data for English, French and Spanish. For this reason, we used different audio sets present in the Mozilla Common Voice datasets. The Mozilla Common Voice dataset is a large, open-source and multi-lingual dataset. We used specific subsets of this dataset, for gathering data for each of 3 languages.

Details regarding the data

	English	French	Spanish
Data Version	Common Voice Delta Segment 14.0	Common Voice Corpus 1	Common Voice Delta Segment 12.0
Compressed Dataset Size (Gb)	1.43	2.09	1.01
Number of audio files	44045	75022	36655
Audio file format	mp3	mp3	mp3
Number of Voices	1212	1697	373
Total Recorded Duration (hours)	70	80	52
Average audio duration (seconds)	5.7	3.7	5
Sampling Rate	22050	22050	22050

When you download and extract the data it will be in the following format

../{lang}/{all_audio_files}

However, keeping all audio files for a particular language in a single folder makes working with a subset of data challenging. At the same time, Google Collab has a limitation where if a folder has more than 1000 files, it causes the drive to dismount. Cause of this a script which would create subfolders inside each language folder, and each subfolder will have a max of 100 files. This helped us address the limitations mentioned above.

Method

In the beginning, we had 2 different methods which would have provided us with a structured approach to solve our problem CRISP-DM and SEMMA. These were the considerations that we took in before deciding on the method CRISP-DM would allow us to focus on the goal of satisfying our business goal and would provide us with an iterative approach to correct any mistakes. For SEMMA since our dataset was huge the main advantage it provided was that we would start with sampling, trying out new techniques would be faster. We decided to follow the CRISP-DM methodology to solve the problem, as it would provide us with more flexibility while having a clear understanding of the problem we are trying to solve, instead of just focusing on building a model. As per the methodology, we split the process into the following parts:

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modelling
5. Evaluation
6. Deployment

Business Understanding

- In this phase first formally describe the problem we intend to solve i.e. building a model which is capable of distinguishing between audio spoken in English, French and Spanish and which is capable of providing a real-time inference when running on a standard,low powered non GPU compute resource
- We define what the success parameters for the project look like
    - Achieve overall accuracy of over 90%
    - Achieve a precision of over 90% for each of the individual classes
    - It should be able to provide a real-time inference when deployed

Data Understanding

- With business understanding achieved we move to data understanding, in this phase get a good understanding of what the data is ,if there are any missing data or outliers or class imbalances that have to be addressed
- In this phase, we also perform the research to find out which are the possible features that we can extract from the data that can be used for training our model.
- Based on our research list of possible features which we could extract for language detection includes Filter Banks, Fourier Transform, Power spectrum, GTCC and MFCC

Data Processing

- As our data is very huge we cannot generate all the required features together, as it is a very time and memory-intensive process. So here is where we make our first assumption, of using only MFCC values as features and if after the evaluation phase we do not meet our goals we will come back to this step and try to use other features of a combination of a few
- For all the audio files, we generate their MFCC values (40 MFCC per file) and save them in a data frame along with their labels.
- We will also perform some analysis to check if we have any redundant values and if there exists sufficient contrast in the data for us to be able to use the data for training purposes

Modelling

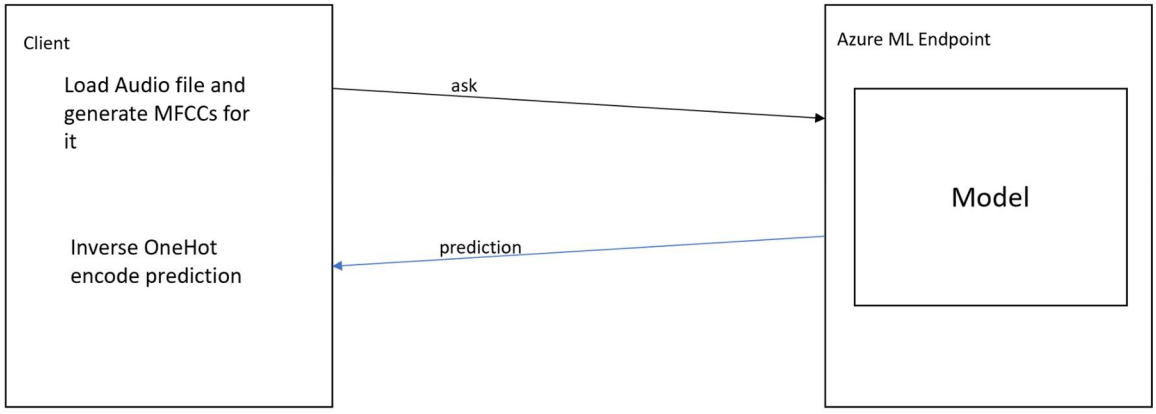
- In this step we will use the output of data processing as input and build many different models
- Our intuition was for our tasks RNN and 1D CNN would perform better than other models, to test it out we build 4 models a 3 layer RNN, 5 layer RNN a 3 layer CNN and a 5 layer CNN. The 5-layer CNN worked the best for us.

Evaluation

- In our implementation of CRISP-DM we have used Modelling in evaluation on conjunctions, that is you would train a model, evaluate it based on the primary performance metrics for us it was accuracy, if the accuracy is satisfactory you'll go ahead with further evaluation, or else you go back to try a new model or a new feature
- Once primary evaluation is satisfied we look at all the other performance metrics such as confusion matrix, precision, recall, AUC and ROC to gain confidence in our model
- After we have achieved all the required evaluation metrics along with the time taken for inference we can proceed to deployment, else we would go back to try a new model or different features

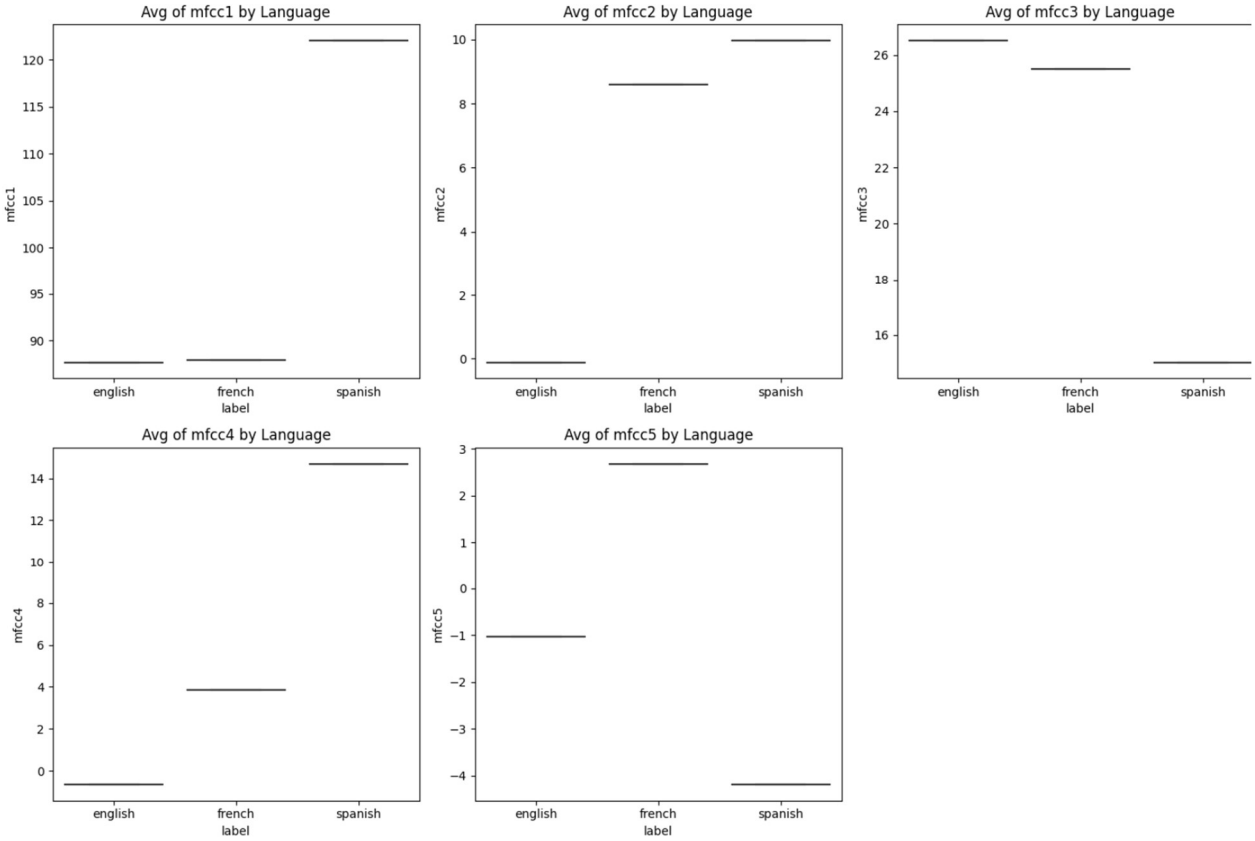
Deployment

- In the deployment phase we make the plan on how we'll have the model deployed, as in how the inference pipeline will work, how it can be used etc
- For our problem we decided to deploy our model as an API on Azure ML Endpoint
- Where the endpoint will be a low powered one as it was one of our goals, which will just handle the prediction part of inference rest everything else will be taken care by the client that is someone or something that is invoking the endpoint



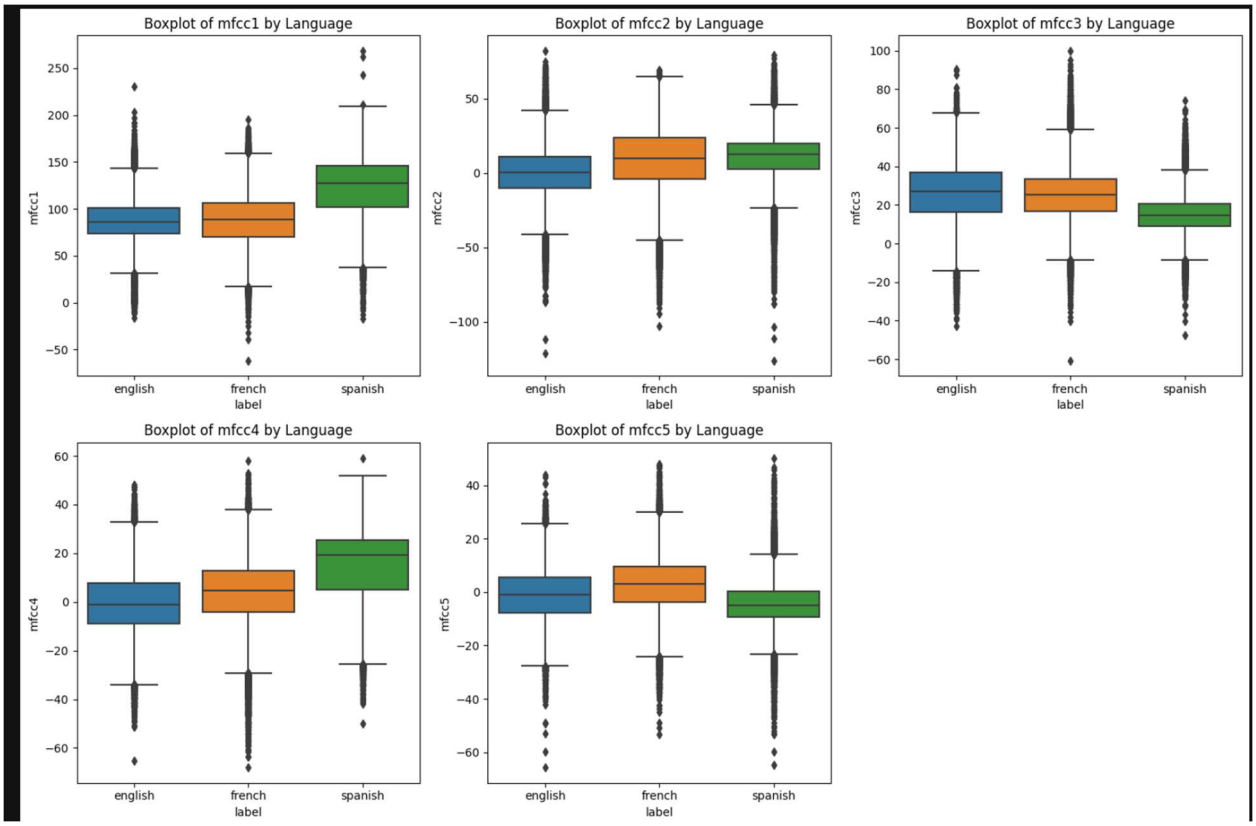
Experiments and Results

In our approach the first option that we opted for was during feature selection where we decided to used MFCC values for each audio files by it self. As MFCC are supposed to not only provide us features which we can use but they are also supposed to act as a dimensionality reduction technique, where they convert an audio file representation from sampling_rate*audio_length number to just n_mfcc which we specify in our case we specified n_mfcc=40, that means we' ll use 40 MFCC value to represent each audio files. This method might cause problems when the audio length are varying by a great factor however in our dataset the avg audio length for all 3 languages is close to 5 seconds, so it wouldn't cause any problems. Now that we generated all the MFCC values, we need to prove our hypothesis that they can be used for training purpose , before we actually commit to using them for training purposes. To test this out as a number of MFCC values are used for representing each file and they are some what correlated to each other a direct method of findinging the correlations with the target cannot be used . So here we use the data contrast approach, as long as there exists a data contrast between the features belonging to different classes we can use them for training purposes. To find out the contract we first find the average MFCC values for all 40 coefficients belonging to a particular class, after than we compare these average values with each other to see if there is any contrast.



The above visualization shows the difference between the 5 MFCCs values for each of the laguage. Based on the graph you can clearly see that there exists a good amount of of construct

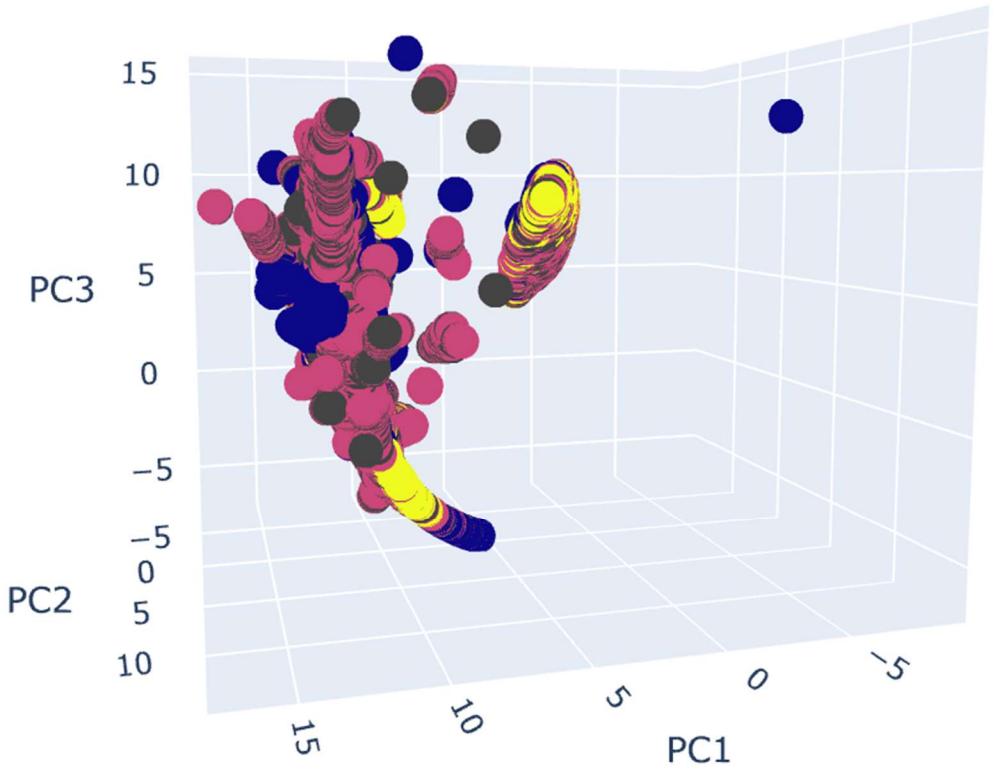
between the avg MFCC values of different class. Now just to double check out approach we will text for all MFCCs values for some some coefficients belonging to a certain class, and if we can see a contrast there, it implies that we are on the right track and go ahead with training.



We can clearly see that there is a good amount of contract between MFCC values, now not the contract in the box plot might not appear to be large however as we have 40 MFCC coefficients for each audio file, the small contract will be aggregated when the entire data will be used.

After this we addressed the class imbalance by dropping the about 40% of french records. After which we have an almost uniform class distribution.

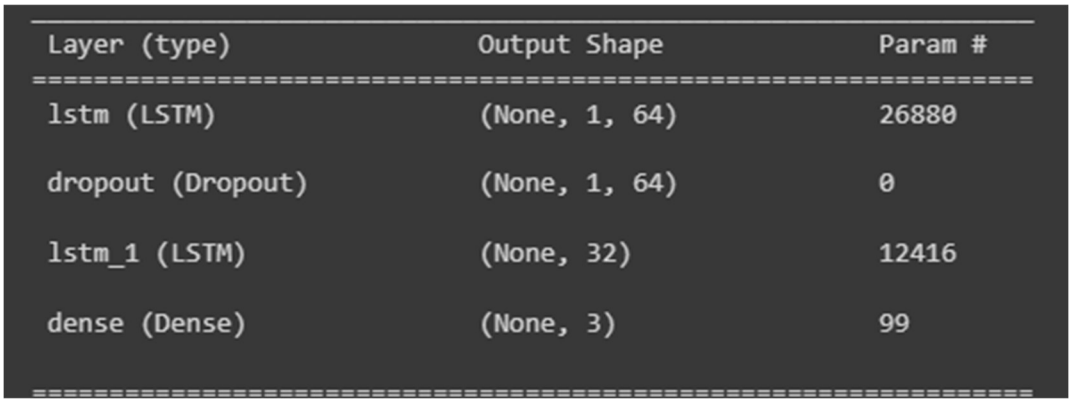
As we wanted the to keep our model as simple as possible and easy to run, we though ou using UMap to further reduce the number of features from 40 to 3. Here our idea was that if after applying UMap if we were to get 3 clear clusters then we can use the reduced dimensions for training the model.



After applying the UMap, we weren't able to get 3 clearly separate cluster, impling that this approach most likely wont give us a good results. We'll discard the approach and move ahead with modelling using the MFCC values.

Based on previous experience we thought to try out Recurrent Neural Networks along with Longterm Short Term Memory as they are generally good at handling time based series. Before we start with training the neural network, we need to first figure out certain hyperparameters such as loss function and optimizer that we will be using, for the problem. One thing to keep in mind is as this is a multiclass classification problem our output layer will be a softmax layer, where if the classification problem has 'n' classes the output layer will have 'n' nodes where each node represents the probability of the the data belonging to one of the class. So the loss function and the optimizer which we choose should work well with a softmax output layer. After doing some research we decided to use a Categorical Cross Entropy loss function along with Adam optimizer. Reason being Adam adjusts the learning rate during training for each parameter individually. It computes adaptive learning rates based on the estimates of first and second moments of the gradients.

For the first model we build a 3 layer RNN, along with a dropout layer in between to prevent overfitting.

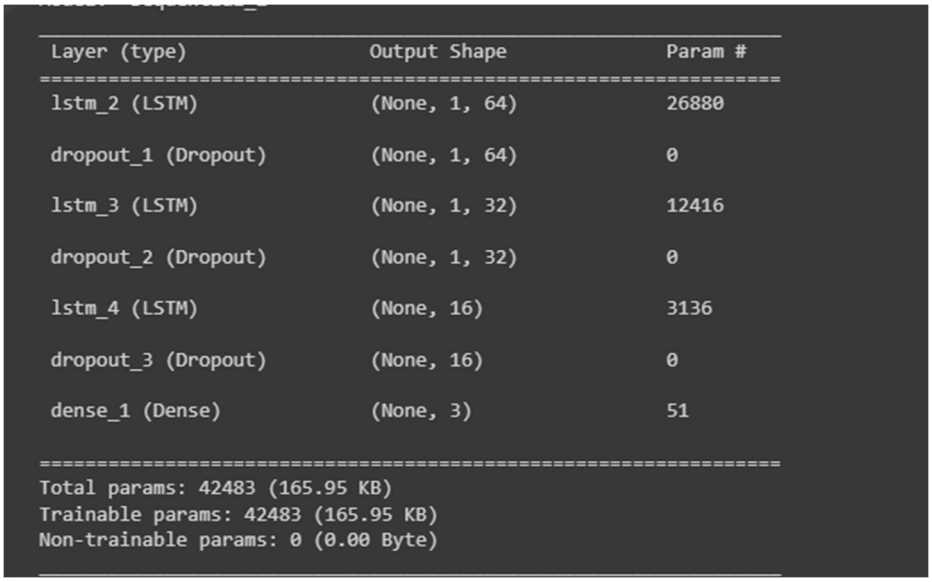


Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 1, 64)	26880
dropout (Dropout)	(None, 1, 64)	0
lstm_1 (LSTM)	(None, 32)	12416
dense (Dense)	(None, 3)	99

And we decided to run the model for 10 epochs, after 10 epochs this model gave us an accuracy 86% which is good but is not what was our goal. Now if you remember our methodology we would would only calculate secondary performance metrics if the primary metric is satisfied. In this case our accuracy doesn't meet our condition of being over 90% so we wont further evaluate this model, instead we will try a different model.

Based on our previous experiment we estimate that RNN is able to capture a large variance in the data but not all. One way which we though to improving it was adding more layers to the RNN,

So the next model which tried as a 5 layer RNN, along with 3 dropout layers at different points to prevent overfitting.



Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 1, 64)	26880
dropout_1 (Dropout)	(None, 1, 64)	0
lstm_3 (LSTM)	(None, 1, 32)	12416
dropout_2 (Dropout)	(None, 1, 32)	0
lstm_4 (LSTM)	(None, 16)	3136
dropout_3 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 3)	51

Total params: 42483 (165.95 KB)
Trainable params: 42483 (165.95 KB)
Non-trainable params: 0 (0.00 Byte)

After 10 epochs, the accuracy of this model was 85%, which is a little worse than the previous model. What;s worse is along with validation even the training accuracy decreased.

The inference from this is that maybe RNN is not the way to go and we might have to try some other methods.

Based on our research we came across academic papers , where they demonstrated the use of CNN for language learning. So we decide to try 1D CNN out.

First we tried a 3 layer 1D CNN, with Pooling layer and Dropout layer to prevent overfitting and reducing the number of parameters. We ReLU activation function for all but the output layer.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 38, 32)            128

 max_pooling1d (MaxPooling1  (None, 19, 32)            0
 D)

 conv1d_1 (Conv1D)           (None, 17, 64)            6208

 max_pooling1d_1 (MaxPoolin  (None, 8, 64)             0
 g1D)

 conv1d_2 (Conv1D)           (None, 6, 128)            24704

 max_pooling1d_2 (MaxPoolin  (None, 3, 128)            0
 g1D)

 flatten (Flatten)           (None, 384)               0

 dense_2 (Dense)             (None, 32)                12320

 dropout_4 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 3)                 99

=================================================================
Total params: 43459 (169.76 KB)
Trainable params: 43459 (169.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

After 10 epochs, this model gave us an accuracy of over 89%, which is better than all the previous model tried. This is very close to goal ov more than 90% and would require some tweeking to get there.

So the same logic which we used for improving the RNN, we ‘ll use with CNN, instead of a 3layer we will try a 5 layer CNN, with rest of the hyper parameters remaining the same.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_3 (Conv1D)           (None, 38, 32)            128

 activation (Activation)     (None, 38, 32)            0

 max_pooling1d_3 (MaxPoolin  (None, 19, 32)            0
 g1D)

 conv1d_4 (Conv1D)           (None, 17, 64)            6208

 activation_1 (Activation)   (None, 17, 64)            0

 conv1d_5 (Conv1D)           (None, 15, 128)           24704

 activation_2 (Activation)   (None, 15, 128)           0

 conv1d_6 (Conv1D)           (None, 13, 256)           98560

 activation_3 (Activation)   (None, 13, 256)           0

 conv1d_7 (Conv1D)           (None, 11, 512)           393728

 activation_4 (Activation)   (None, 11, 512)           0

 max_pooling1d_4 (MaxPoolin  (None, 5, 512)            0
 g1D)

 flatten_1 (Flatten)         (None, 2560)              0

 dense_4 (Dense)             (None, 512)               1311232

 dropout_5 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 3)                 1539

=================================================================
Total params: 1836099 (7.00 MB)
Trainable params: 1836099 (7.00 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

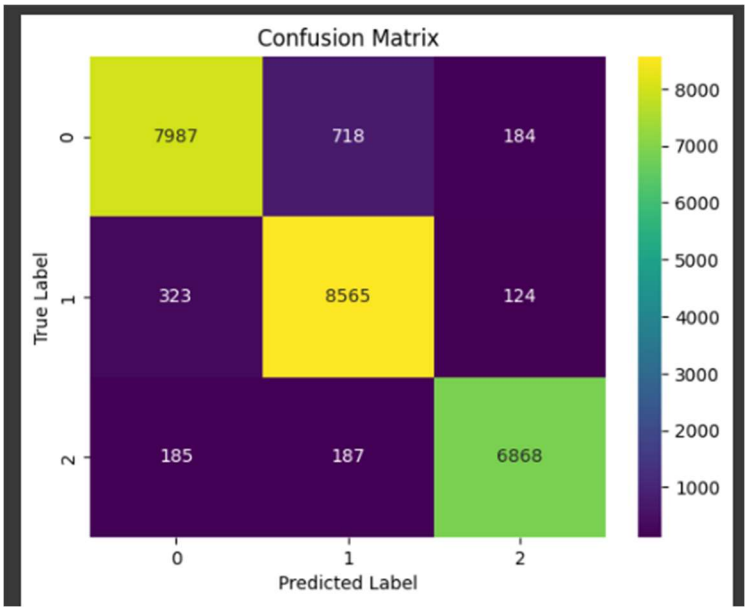
This model gave us an accuracy of 93% on the test set. This satisfies our requirement for primary metric, so we take this model further into the evaluation phase to see if the other requirements are satisfied.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.90      0.92      8889
           1       0.90      0.95      0.93      9012
           2       0.96      0.95      0.95      7240

    accuracy                           0.93     25141
   macro avg       0.93      0.93      0.93     25141
weighted avg       0.93      0.93      0.93     25141
```

The classification report for this model shows that it satisfied both our requirements of accuracy as well as individual class precisions.

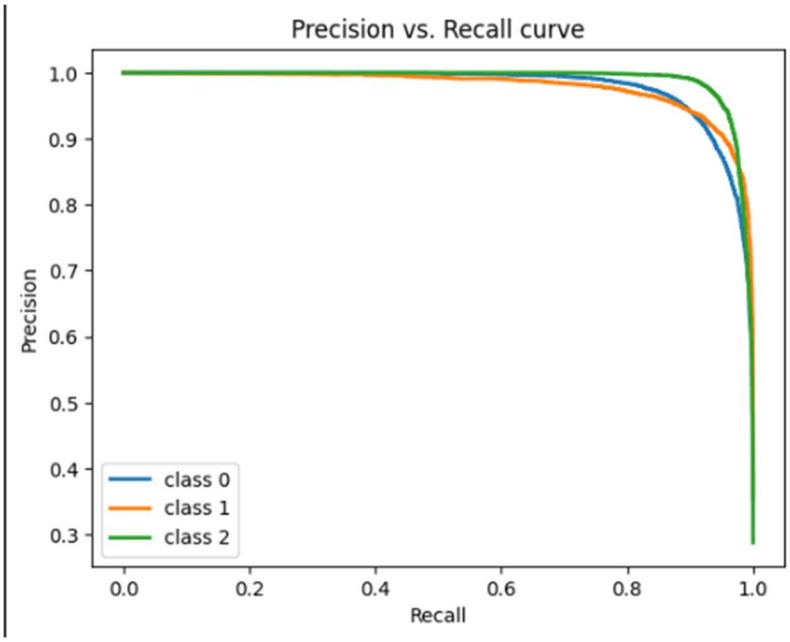
We further evaluate this model by checking its confusion matrix, precision- recall trade off and AUC.

Confusion matrix



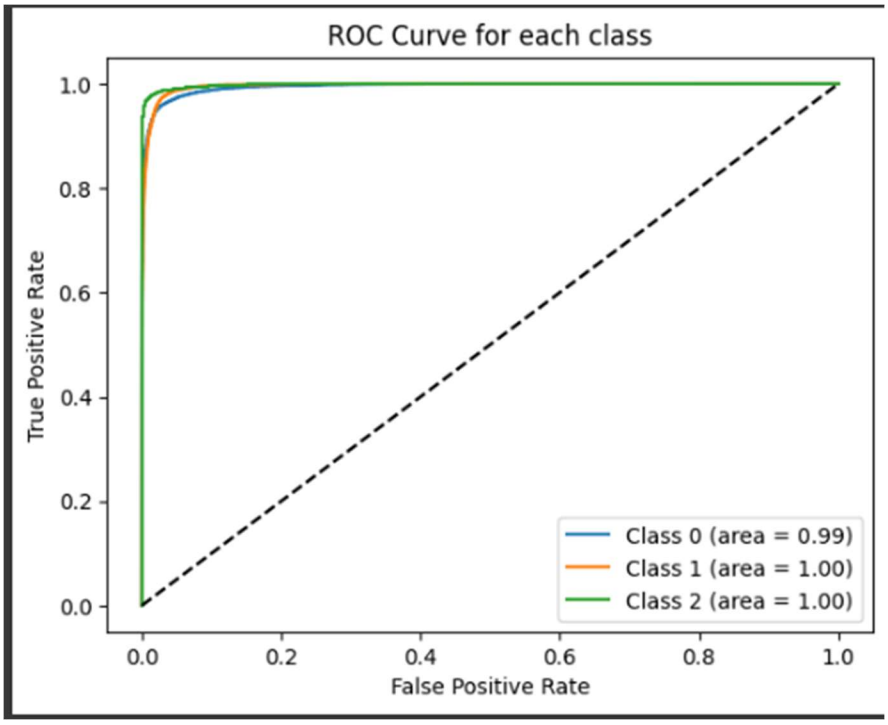
From the confusion matrix we can see that for the most part, we are able to predict the language correctly without having any issues, there are some incorrect predictions however based on the scale of the data they aren't that major. One thing that stands out is that most of the incorrect predictions have occurred when objects incorrectly labelled English as French or French as English. This is something we will keep in mind if further evaluation fails and we have to retrain the model. However for now as both the goal criteria are met we can continue with the evaluation and then deployment.

Precision-recall Curve



As for the precision-recall curve we can see that for all lases the curve is between 1.0 and 0.9, which is considered to be a high score, implying we have good classifiers in hand.

AUC



From the AUC curve we can see that the classification of each class is really good as the AUC for all 3 class are very close to 1.

This concludes our experimentation and evaluation phase, now we can deploy this mode so that it can be used by everyone.

For deployment purposes, we will deploy this model as an API on Azure ML Endpoints, which can then be later used by any client that is connected to the internet.

Now of our success criteria was to deploy it on a low-powered compute instance. Our final model is only 20MB in size, so it shouldn't have any problem running on any compute instance. The number of parameters are close to 1.5 million. Which for modern computers are very easy to handle especially when it comes to inference. However to give our model the best chance and to reduce the overhead on the Azure service, we created a custom environment for the model to run in, which only contains the most essential packages required for running the model.

The result of the above efforts the endpoint which runs the model got an average response time of 50 milliseconds (note this API response time, and not the time the model took for the prediction that is close to 15 milliseconds). Which is good and this response time permits us to use the model for real applications.

Conclusion

Through this project we were successful in able to build a language detection 1D CNN model following CRISP-DM methodology, with an overall accuracy of over 93%. To achieve this is learned about various types of features that can be extracted from audio data and the information that they represent. As our data size was huge we learned what are the ways in which could work with large data, generate features for them and save them in such a way that the data quality is preserved. When in the deployment phase we learned and successfully deployed our model as an endpoint on Azure, such that anyone can access it. The future scope of this project is to include features other than MFCC to make the model even more accurate, adding the capability to distinguish between even more languages, along with the deployment include additional MLOps capabilities for A/B testing and CI/CD with models.

References

[1] https://ieeexplore.ieee.org/document/9031923

[2] https://www.mdpi.com/2076-3417/12/18/9181