B.E. PROJECT ON

**Spoken Language Identification using Neural Network**

SUBMITTED IN PARTIAL FULFILLMENT OF REQUIREMENTS OF AWARD OF

B.E. (COMPUTER ENGINEERING)

DEGREE OF UNIVERSITY OF DELHI

SUBMITTED BY:

**Aditya Jain 210/CO/13**

**Anmol Pandey 233/CO/13**

**Anmol Varshney 234/CO/13**

GUIDED BY:

**Dr. Shampa Chakraverty**



**COMPUTER ENGINEERING (COE)**

**NETAJI SUBHAS INSTITUTE OF TECHNOLOGY**

**UNIVERSITY OF DELHI**

**2016-17**



**CERTIFICATE**

The project titled “**Spoken Language Identification using Neural Networks**” by **Aditya Jain (210/CO/13), Anmol Pandey (233/CO/13) and Anmol Varshney (234/CO/13)** is a record of bona fide work carried out by us, in the Division of Computer Engineering, Netaji Subhas Institute of Technology, New Delhi, under the supervision and guidance of **Dr. Shampa Chakraverty** in partial fulfilment of requirement for the award of the degree of Bachelor of Engineering in Computer Engineering, University of Delhi in the academic year 2016 - 2017.

**Dr. Shampa Chakraverty**

Prof. and Head of Department

Computer Engineering Department

Netaji Subhas Institute of Technology

New Delhi

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Candidates’ Declaration**

This is to certify that the work which is being hereby presented by us in this project titled “Spoken Language Identification using Neural Networks” in partial fulfilment of the award of the Bachelor of Engineering submitted at the Department of Computer Engineering , Netaji Subhas Institute of Technology Delhi, is a genuine account of our work carried out during the period from January 2017 to May 2017 under the guidance of Dr. Shampa Chakraverty, Head of Department(COE), Netaji Subhas Institute of Technology, Delhi. The matter embodied in the project report to the best of our knowledge has not been submitted for the award of any other degree elsewhere.

Aditya Jain Anmol Pandey Anmol Varshney

Date: \_\_\_\_\_\_\_\_\_\_

This is to certify that the above declaration by the students is true to the best of my knowledge.

Dr. Shampa Chakraverty

Date: \_\_\_\_\_\_\_\_\_\_

**Acknowledgement**

Any Project indisputably plays one of the most important roles in an engineering student’s life to make him a successful engineer. It provides the students with an opportunity to gain valuable experience on the practical application of their technical knowledge and also brings out and hones their technical creativity. Thus the need for one is indispensable.

We would like to express our deep gratitude towards our mentor **Dr. Shampa Chakraverty,** Head of Department, Computer Engineering Department, Netaji Subhas Institute of Technology, New Delhiunder whose supervision we completed our work. Her invaluable suggestions, enlightening comments and constructive criticism always kept our spirits up during our work. She was always there to help whenever we faced any problems.

Our experience in working together has been wonderful. We hope that the knowledge, practical and theoretical, that we have gained through this term B.E. Project will help us in our future endeavors in the field.

We are grateful to all our friends for providing critical feedback and support whenever required.

We regret any inadvertent omissions.

Aditya Jain (210/CO/13)

Anmol Pandey (233/CO/13)

Anmol Varshney (234/CO/13)

**Abstract**

In this study we attempt to identify the spoken language of audio samples of speech using Deep Neural Networks (DNNs). Language Processing has been a subject of wide interest and a lot of work has been done towards that end. The first step in a language processing pipeline is always language identification (LID). Language identification facilitates transferring the control to an appropriate next stage of the processing system. In this dissertation, we adapt DNNs to the problem of language identification using the mel-frequency cepstral coefficients of audio signals as primary features. The audio signals are divided into several frames and the short term acoustic features extracted from each of these frames. Mean and Variance of these short-term acoustic features across several frames are calculated to capture the fluctuations of speech in order obtain a reasonable accuracy. These average features are then stacked together taking measures from both before and after the frame under consideration forming context windows to further capture the time dependent behavior of the signal. These windows result in a superior performance than many of the existing systems. These features are feeded into a Neural Network which is used to identify the top two prospective candidates for the given speech signal. Binary classification is then applied among these candidates to determine the final output. At both of these stages feature selection is applied to include only the most promising features and reduce the dimensionality of the inputs. Feature Selection for the binary classification stage results in a confusion matrix, which contains the selected features specifically aimed to differentiate between any two specified languages. Our findings suggest that DNNs can be used for LID tasks with reasonable accuracy. Although our model is tested using only 3 languages it can easily be extended to any number of languages.   
  
  
  
  
Keywords: Language Identification, Deep Neural Networks, mel-frequency coefficients, Context Window, Feature Selection

**List of Contents**

1. Certificate 2
2. Candidate Declaration 3
3. Acknowledgements 4
4. Abstract 5
5. List of Contents 6
6. List of Figures 8
7. List of Tables 9
8. Chapter 1: Introduction 10
9. Chapter 2: Literature Survey 12
   1. Phonotactic Language Identification Systems
      1. GMM Based
      2. Phone Recognition Followed by Language Modelling (PRLM)
      3. Parallel PRLM
      4. Parallel Phone Recognition
   2. Language Identification based on GMM and Ceptral Features
      1. Kshirod Sarmah and Utpal Bhattacharjee
      2. Pedro A. Torres-Carrasquillo
   3. Language Identification based on DNN and Ceptral Features
      1. Ignacio Lopez-Moreno
      2. Julien De Mori
      3. Najim Dehak
10. Chapter 3: Materials and Methods
    1. Machine Learning
    2. Supervised Learning
    3. pyAudioAnalysis
    4. LibROSA
    5. Chi Square Feature Selection
    6. Deep Learning
    7. Theano
    8. Keras
    9. scikit-learn
    10. pyAudio (Real Time Analysis)
    11. NumPy
    12. Matplotlib
11. Experimental Framework
    1. Feature Extraction
    2. Training the Initial Neural network
    3. Training the Binary Neural Network
12. Results
13. Conclusion and Future Work
14. Appendix – Tools and Platforms used

**List of Figures**

**List of Tables**

**Chapter 1: Introduction**

The problem of automatic language identification (LID) can be defined as the process of automatically identifying the language of a given spoken utterance. The first step in a language processing pipeline is always LID. LID is used either as a standalone task or as a preprocessing step, capturing the first seconds of the recording and processing it in order to transfer the control to the appropriate next stage; e.g. speech recognition systems, multilingual translation systems or call-centers (e.g., emergency calls) routing, where the response time of a native operator might be critical. LID is also a topic of great importance in areas of intelligence and security, where the language identities of recorded messages and archived materials need to be established before any information can be extracted.

Our project aims at spoken language identification in speech audio samples using Deep Learning Models (DNNs). Speech audio recognition features well-defined, clear voices and shows very little background noise. This is in contrast to song audio which combines background music, instruments and the singer’s voice into a complex mix. Even though several high level approaches based on phonotactic and prosody are used as meaningful complementary sources of information, nowadays, many state-of-the-art LID systems still include or rely on acoustic modelling. While previous works on neural networks applied to LID report results using shallow architectures or convolutional neural networks, in this study, we propose the use of DNNs as a new method to perform LID at the acoustic level. Recent breakthroughs in signal processing and acoustic modelling have shown the promise of deep learning models.

Motivated by those results and also by the discriminative nature of DNNs, we adapt DNNs to work at the acoustic frame level to perform LID. Particularly, in this work, we extract the acoustic features from an audio signal, mfcc and their first and second order differentials to be exact which has been divided into frames and compute their mean and variance across several frames in order to account for the fluctuation in the speech signal. This results in stacking of average features across several consecutive frames known as context windows which also result in much superior performance. The mel-frequency cepstrum is a representation of an audio signal on the mel scale, a nonlinear mapping of frequencies that down-samples higher frequencies to imitate the human ear’s ability to process sound. Delta features and the Delta Delta Features are the first and second time derivatives of the cepstral coefficients, capturing the change of the cepstral features over time. Feature Selection using Chi Square Statistics is also applied to obtain the best possible results as well as reduce dimensionality of the input. We compare the obtained results with a baseline Support Vector Machine (SVM) based system trained from exactly the same acoustic features.

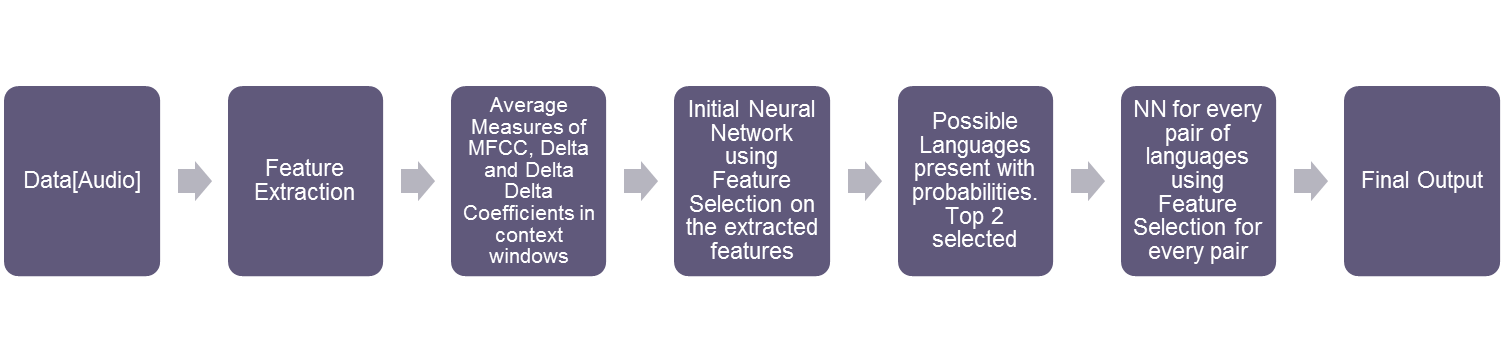
The remaining thesis is organized as follows. In chapter 2, previous work done and the current state of the art in this area are detailed, along with the limitations and the roadblocks encountered in this field. Chapter 3 provides and overview of the technologies used and the theoretical concepts behind using the frameworks or tools. Chapter 4 discusses about our contribution to this field of research. A detailed experimental framework and the course of action is elaborated. Chapter 5 provides an experimental evaluation of our proposed approach, and the findings of the experiment are reported and analyzed. Chapter 6 summarizes the thesis with conclusions and avenues of future work. 

Fig **{number}**Workflow of the project

**Chapter 3: Materials and Methods**

Machine Learning

Machine learning, a branch of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases. Machine Learning is a scientific discipline that addresses the following question: ‘How can we program systems to automatically learn and to improve with experience?’ Learning in this context is not learning by heart but recognizing complex patterns and make intelligent decisions based on data. The difficulty lies in the fact that the set of all possible decisions given all possible inputs is too complex to describe. To tackle this problem, the field of Machine Learning develops algorithms that discover knowledge from specific data and experience, based on sound statistical and computational principles.

The field of Machine Learning integrates many distinct approaches such as probability theory, logic, combinatorial optimization, search, statistics, reinforcement learning and control theory. The developed methods are at the basis of many applications, ranging from vision to language processing, forecasting, pattern recognition, games, data mining, expert systems and robotics.

A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as examples that illustrate relations between observed variables. A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples (training data). Hence the learner must generalize from the given examples, so as to be able to produce a useful output in new case. We have followed the same principle, where we have designed algorithms to generalize from the given examples and then produce a useful output. Since the amount of data available for the different languages was limited, we used KFold Cross Validation which makes efficient use of data available and avoids overfitting, which occurs when a model begins to "memorize" training data rather than "learning" to generalize from trend. Machine learning algorithms are described as either 'supervised' or 'unsupervised'.

Supervised Learning

Supervised learning is the machine learning task of inferring a function from supervised (labeled) training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way. It is called a classifier (if the output is discrete) or a regression function (if the output is continuous). Supervised learning is when the data you feed your algorithm is "tagged" to help your logic make decisions.

Our classifying approach is a supervised learning method that uses labelled dataset of audio speech samples, using the features extracted from the frames as the basis of the classifier. Using some of the audio speech samples as the training dataset, classifier is applied on testing dataset to predict the spoken language. The testing method used is KFold Cross Validation which makes efficient use of the available data and avoids overfitting.

pyAudioAnalysis

pyAudioAnalysis is an open Python library that provides a wide range of audio-related functionalities focusing on feature extraction, classification, segmentation and visualization issues. The purpose of the pyAudioAnalysis library is to provide a wide range of audio analysis functionalities through an easy-to-use and comprehensive programming design.

pyAudioAnalysis implements the following functionalities:

* Feature extraction: several audio features both from the time and frequency domain are implemented in the library.
* Classification: supervised knowledge (i.e. annotated recordings) is used to train classifiers. A cross-validation procedure is also implemented in order to estimate the optimal classifier parameter. The output of this functionality is a classifier model which can be stored in a file.
* Regression: models that map audio features to real-valued variables can also be trained in a supervised context. Again, cross validation is implemented to estimate the best parameters of the regression models.
* Segmentation: the following supervised or unsupervised segmentation tasks are implemented in the library: fix-sized segmentation and classification, silence removal, speaker diarization and audio thumbnailing.
* Visualization: given a collection of audio recordings pyAudioAnalysis can be used to extract visualizations of content relationships between these recordings.

One of the best things of using pyAudioAnalysis was the ease of its use. It was very easy to extract the time and frequency domain features from an audio signal. However, since the time domain and the frequency domain features apart from the mfcc features did not provide any improvement in the results, the library was used only to extract the mfcc features. Inspite of providing very accurate classifiers it did not contain any DNN Classifiers, which was the reason it wasn’t used for classifying spoken audio samples. A drawback in this library is the large processing time it takes to extract all the features from an audio. It extracts all the features regardless of how many maybe required. It is advised to tweak the code for the library to extract only the required features. It was used both in the test and training phases to extract features from the audio speech samples.

LibROSA

LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems. It is built to ease the transition of music information retrieval (MIR) researchers into Python (and modern software development practices), and also to make core MIR techniques readily available to the broader community of scientists and Python programmers. It has a relatively flat package layout, and following scipy rely upon numpy data types and functions, rather than abstract class hierarchies. Librosa’s functions expose all relevant parameters to the caller. While this provides a great deal of flexibility and a consistent interface to process audio files by defining a set of general conventions and standardized default parameter values shared across many functions. LibROSA supports the following basic features among others:

* Compute mel spectrogram, MFCC, delta features, chroma features
* Locate beat events
* Compute beat-synchronous features
* Display features
* Save beat tracker output

We used LibROSA to compute the delta and delta delta coefficients from the obtained mfcc coefficients. The best thing about using librosa was the ease of its use. Both delta and delta delta functions could be calculated in a straightforward manner. Althouh librosa does provide functionality to extract mfcc coefficients from an audio signal, it lacks the versatility that pyAudioAnalysis provides and hence wasn’t used for feature extraction.

Chi Square Feature Selection

Feature selection is the machine learning task of selecting subset of features which are more relevant for use in model construction. It is applied because:

* Data contains many features that are redundant or irrelevant.
* Less resource (computational time and memory) is required.
* Shorter training as well as prediction time.
* Avoids curse of dimensionality.
* Prevents overfitting of data and generalizes the model.

There are many ways to implement selection of features. One way is by scoring the features in their usefulness. When we have set of categorical data, we can apply chi-squared testing (χ2). It is a measure of goodness of fit and this test allows us to compare a collection of categorical data with some theoretical expected distribution. Chi square test can be thought of as a test of independence and tests if null hypothesis is true or not with null hypothesis being two categorical variables are independent. It is evaluated using the formula:

where

χ2 = chi – squared stat of null hypothesis

Oi = observed value of feature for class i

Ei = expected value of feature for class i

n = number of classes

The chi-squared value can be directly mapped to its corresponding p-value by subtracting cumulative distribution function from 1. Low p-value indicates greater statistical significance. A p-value of 0.05 is often used as a cut off between significant and non-significant results.

Deep Learning

**Deep learning** (also known as **deep structured learning**, **hierarchical learning** or **deep machine learning**) is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple non-linear transformations.Deep learning is part of a broader family of machine learning methods based on learning representations of data. An observation (e.g., an image) can be represented in many ways such as a vector of intensity values per pixel, or in a more abstract way as a set of edges, regions of particular shape, etc. Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence. It is one of the fastest growing fields of Machine Learning - it’s true bleeding edge.

For supervised learning tasks, deep learning methods obviate feature engineering, by translating the data into compact intermediate representations akin to principal components, and derive layered structures which remove redundancy in representation. Deep learning is nothing but a neural network with a lot of hidden layers of nonlinear processing and supervised or unsupervised learning of feature representations in each layer, such that the layers form a hierarchy from low level to high level features. Deep learning is called ‘deep’ learning, as its algorithms transform their inputs through more layers than shallow learning algorithms. At each layer, the signal is transformed by a processing unit, like an artificial neuron, whose parameters are learned through training.

We have used DNNs because they have the ability to learn complicated feature representations and classifiers jointly. They learn much better models of data that lie on or near a non-linear manifold and their performance does not saturate with increase in training data.

Theano

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

* Tight integration with NumPy.
* Transparent use of a GPU
* Efficient symbolic differentiation
* Speed and stability optimizations dynamic C code generation
* Extensive unit-testing and self-verification

Theano defines a language to represent mathematical expressions and manipulate them, a compiler to create functions that can compute values for these expressions, and a library which will execute these functions when evaluated on numeric values.

The Keras software used in our project builds on the strengths of Theano, by providing a higher level user interface. Keras makes it easier to express the architecture of deep learning models, and training algorithms, as mathematical expressions to be evaluated by Theano.

Keras

**Keras** is an open source neural network library written in Python. It is capable of running on top of Deeplearning4j, Tensorflow or Theano. Designed to enable fast experimentation with deep neural networks, it focuses on being minimal, modular and extensible. *Being able to go from idea to result with the least possible delay is key to doing good research.* It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer.

Features of Keras:

* Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
* Supports both convolutional networks and recurrent networks, as well as combinations of the two.
* Runs seamlessly on CPU and GPU.

We used Keras to design and implement the DNNs at both the initial stage as well as the binary classification stage. The main advantage of using Keras was how easy it was to ignore the difficult mathematical details of the underlying neural network and focus only on optimizing the performance of the net. Keras provided a smooth interface to change every parameter of the neural network including the number of layers, type of connections, activation functions, error function, number of neurons in each layer, weight initialization, dropout, regularization and performance metrics among others. It even has an option of loading and saving a model which enables fast learning. Keras is strongly advised for anyone who wishes to analyse the performance of DNN on a problem in a short amount of time.

scikit-learn

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance.

scikit-learn provides the following functionalities:

* Classification: Identifying to which category an object belongs to.

Algorithms: SVM, nearest neighbors, random forest

* Regression: Predicting a continuous-valued attribute associated with an object.

Algorithms: SVR, ridge regression, Lasso

* Clustering: Automatic grouping of similar objects into sets.

Algorithms: k-Means, spectral clustering, mean-shift

* Dimensionality reduction: Reducing the number of random variables to consider.

Algorithms: PCA, feature selection, non-negative matrix factorization

* Model selection: Comparing, validating and choosing parameters and models.

Modules: grid search, cross validation, metrics

* Preprocessing: Feature extraction and normalization.

Modules: preprocessing, feature extraction.

We used scikit-learn to build the SGD Classifier which was used as a baseline for comparison with our proposed Model. The library was used to perform KFold Cross Validation which makes efficient use of the limited data available and avoids overfitting. It was also used to preprocess the inputs before feeding it to the neural network to convert the output labels into one hot vectors and to shuffle the input before training.

pyAudio (Real time analysis)

PyAudio provides Python bindings for PortAudio, the cross-platform audio I/O library. With PyAudio, you can easily use Python to play and record audio on a variety of platforms, such as GNU/Linux, Microsoft Windows, and Apple Mac OS X / macOS. PyAudio is inspired by:

* pyPortAudio/fastaudio: Python bindings for PortAudio **v18** API.
* tkSnack: cross-platform sound toolkit for Tcl/Tk and Python.

It can be used in two modes:

* Blocking mode audio I/O
* Callback mode audio I/O

We have used it for doing real time analysis of audio. For this we need callback mode I/O. In callback mode python program’s main thread listens to audio being input from specified source and stores the audio in its buffer. When buffer is filled to a specified capacity, PyAudio will call a specified callback function with the audio data in it’s buffer. This process creates a new thread on which we input the data to our prediction model and plot the result resulting in real time processing of data.

PyQt

PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. PyQt is free software developed by the British Firm Riverbank Computing. PyQt is available in two editions: PyQt4 which will build against Qt 4.x and 5.x and PyQt5 which will only build against 5.x. Both editions can be built for Python 2 and 3. PyQt supports Microsoft Windows as well as various flavours of Unix, including Linux and macOS.

PyQt implements around 440 classes and over 6,000 functions and methods including:

* Substantial set of GUI widgets
* Classes for accessing SQL databases (ODBC, MySQL, PostgreSQL, Oracle, SQLite)
* QScintilla, Scintilla-based rich text editor widget
* Data aware widgets that are automatically populated from a database
* XML parser
* SVG support
* Classes for embedding ActiveX controls on Windows
* To automatically generate these bindings, Phil Thompson developed the tool SIP, which is also used in other projects.

The main advantage of using PyQt is the strong object oriented behavior which makes using different modules in the program very easy. We used PyQt to develop the GUI which provides a visual tool for our model’s working. Object oriented behavior led to easy integration of a graph window with the main window containing the option for selecting a file. PyQt is a cross platform GUI/XML/SQL C++ framework which makes it very efficient as well as makes it possible to run on any platform available.

NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy address the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring (re)writing some code, mostly inner loops using NumPy.

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* Powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* Tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

The main attraction of using NumPy is the fast and efficient processing of the NumPy arrays as compared to native Python lists. Performance in terms of processing time would have been much worse if NumPy wasn’t used. NumPy was used extensively in the project ranging from the extraction of features to the plotting of features against a suitable measure for visual representation. Features were extracted as NumPy arrays. They were processed as NumPy arrays using fast mathematical functions provided by NumPy. NumPy array functions were used to compute mean and variance of the features extracted. Both the baseline and proposed models accepted NumPy arrays as their input. And finally the results were obtained using NumPy arrays and the matplotlib graphs used NumPy array values as the values for plotting.

matplotlib

matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. Itis a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code.

The main advantage of matplotlib is the ease of usability and vast functions available for plotting. We used matplotlib for the visual representation of real time spoken language identification.  
It displays the predicted language at each second interval using the model built and running the prediction in a multiple threads. The predicted language was shown using a graph. Matplotlib provided easy methods to plot the graph that varies with time.

**Chapter 4: Experimental Framework**

The research carried out is a four phase project. The first phase involves extraction of the features from the speech audio samples. This also involves making context windows out of the frames of the audio speech sample and calculating the mean and variance of several consecutive context frames to further capture the fluctuations in the signal. The features are also normalized to ensure every sample contributes equally. The second phase involves feature selection among the extracted features for collecting the best contributing features and inputting these features to the Initial Neural network. The Initial Neural network is then trained with the selected features. The third phase involves applying feature selection among all possible pairs of languages under consideration so as to extract the best contributing features for every pair. These features are then inputted to the Binary Neural Network and the Binary Neural Network is trained. The fourth and the final phase involves testing the samples against the Hybrid Neural Network. The third and the fourth phase are intertwined in the sense that the training and testing data are split according to  
K Fold Cross Validation which makes efficient use of data. The samples are divided for training and testing purposes. Features are extracted from the test samples. This also involves making context windows out of the frames and calculating the mean and variance of several consecutive context frames. The features are then normalized and the same features during the training/second phase are selected and then inputted to the Initial Neural Network. This network generates the top two best candidates for the given sample. These candidates are provided to the appropriate Binary Neural Network which selects the same features during the second training/third phase. These features are focused on classification between the two given languages. These features are then inputted to the Binary Neural Network which provides the final output.

**1) Feature Extraction**

**Dataset Description**

The data set used for the purpose of this project was extracted from <https://www.audio-lingua.eu/>**[cite]**. A number of different sources were tried but the most promising results were obtained by using the data from the above mentioned website. The site contains large number of 16kHz frequency, mono audio speech samples for both male and female voices in a variety of languages. The clips do not have a fixed length and vary in their duration greatly. We created a script which automatically opened up the browser and loaded a specified page. It then copied the link to download each .mp3 provided on the site till a certain specified number. These links were collected into a file and were used to download the files into appropriate folders. These .mp3 files were converted into .wav format which was suitable for feature extraction. We scraped the recordings of 3 languages: Chinese, French and German. A total of 200 samples were downloaded for each language to be used for both training and testing purposes.

**Extraction of Features**

We experimented with using a number of features for classification but the most promising results were obtained by using the mfcc features along with delta mfcc and delta delta mfcc features. This resulted in 39 features for each frame. The mel-frequency cepstrum is a representation of an audio signal on the mel scale, a nonlinear mapping of frequencies that down-samples higher frequencies to imitate the human ear’s ability to process sound. In our implementations, we used the first 13 cepstral coefficients as our primary features, as is common in similar applications. Delta features and the Delta Delta Features are the first and second time derivatives of the cepstral coefficients, capturing the change of the cepstral features over time, which are useful in classifying language, since pace is an important factor in language recognition by humans. We calculated these features as the central finite difference approximation of these derivatives. The audio was divided into frames and the features were extracted using 400 frames in a single window with 100 frames overlap between the windows.

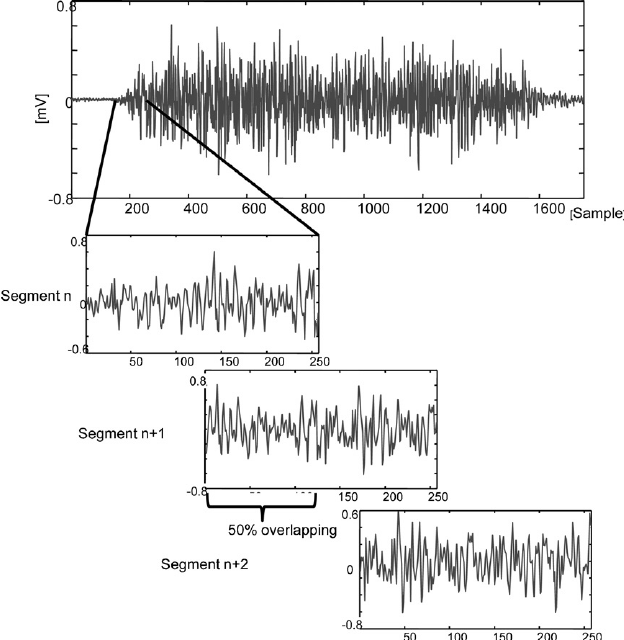
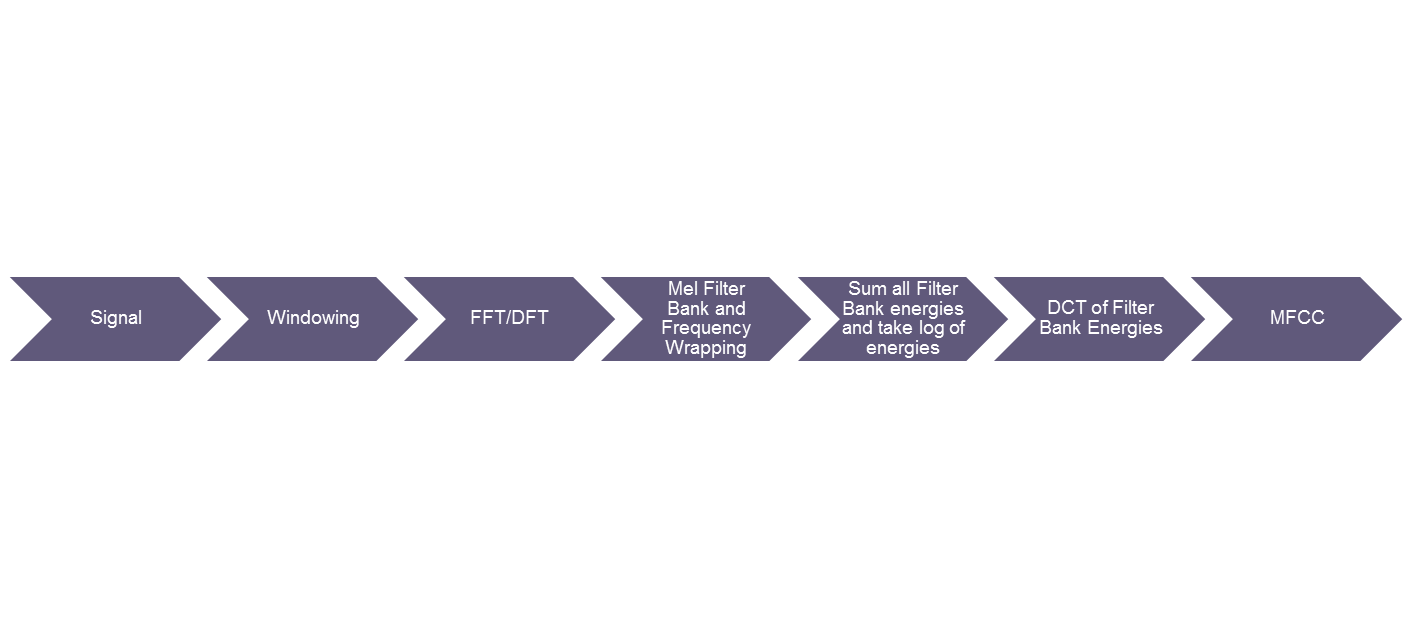


Fig **{number}** Sliding Window

The 13 mfcc features were extracted using pyAudioAnalysis**[cite]**. We tweaked the code for the library to extract only the required 13 mfcc features rather than extracting all the frequency and time domain features to reduce the processing time. Delta and Delta Delta features were calculated using LibROSA**[cite]** which used 3 frames to calculate the estimation of Delta and Delta Delta. We used a context window of size 5 to capture the fluctuations in the speech sample. These were made by stacking the 5 frames after the current frame to make a single feature vector. Average Windows were made after this by specifying the average frames per sample and breaking the audio into equal windows of this size. Mean and Variance for all the features in each feature vector was calculated by taking all the average windows into consideration. This resulted in a feature vector of length 78. This feature vector was then normalized using standard normalization using mu and sigma calculated on the entire test data.

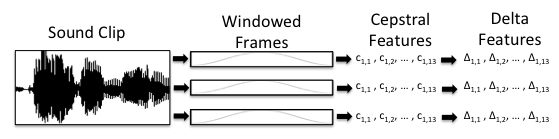


Figure **{number}** Feature Extraction Process

**Preprocessing of Data**

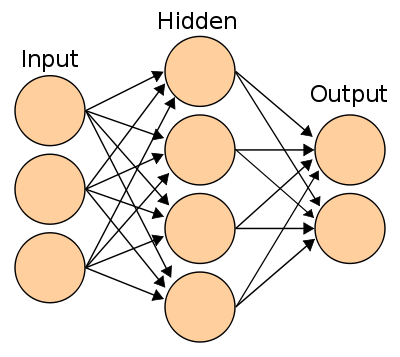
Since the recordings were of variables size we transformed the data by splitting each file such that we obtain a large number of equal duration files for consistency during the training as well the testing phase and also equalize the contribution from each recording. We obtained 1341 samples which were used for both training and testing. Since learning was supervised the labels for each language were to be specified. This was done by organizing the data into directories and while splitting each file the label was specified according to the directory.

**2) Training the Initial Neural Network**

Feature Selection was applied to the extracted features to extract a total of 180 features from each context frame. The features were ranked on the basis of chi squared statistics and the features with the k highest scores were selected, k being the number of features to be selected. The selected features are provided to the Baseline SGD Classifier as well as the Initial Neural Network. The SGD Classifier was implemented using the scikit-learn**[cite]** library and required only calling the fit method to train the model. Since the training was required to be in batches, warm start parameter which remembers the model trained previously was set to True. The Initial Neural network was implemented using Keras**[cite]** which provide a higher level of abstraction on top of the theano**[cite]** backend. The underlying neural network consisted of 3 fully connected layers with the first layer called the input layer containing neurons equal to the selected number of features i.e 180. The second layer called hidden layer contained 12 neurons which were determined by extensive tuning of the model. The third layer called the output layer contained neurons equal to the number of languages to be identified which are 3 in this project. The associated activation function for each of the neurons in the input and hidden layer is relu which stands for rectifier liner unit. The activation function used for the output layer is softmax which produces probability density as its output. The weights were initialized using a uniform initialization scheme and adadelta optimizer was used to control the learning rate. This optimizer makes the learning independent of the initial learning rate. Activity regularization L1 and L2 was added at each layer to avoid over fitting. The model aims to minimize the error function which the and is the actual process through which the model learns. Categorical crossentropy was used which is ideal for multiclass classification. The metric that we sought to maximize was accuracy which we found through extensive research was the only ideal metric for Speech related applications. All other parameters were left at default specified by the keras library. The model was trained for 30 epochs using a batch size of 30 samples. The data provided to the Neural Network was labeled data aimed at supervised learning. However, to adhere to the architecture of the Neural Network. The single output label was transformed into a 1 hot vector with the 1 corresponding to the actual spoken language. The training and testing was done using K Fold Cross Validation with K = 10. Therefore, the training data consisted of 9 folds of the total data available. The data samples included in training were further shuffled using sklearn library before fitting the Initial Neural Network to the data. The model changes the weights and bias associated with each neuron in order to minimize the error function. The model generates the best two candidates for further classification by producing probabilities for each language. The two languages with the highest probabilities are specified as the candidates.

**3) Training the Binary Neural Network**

This phase involves feature selection for all possible pairs of languages (excluding the same language pair). This involves applying chi squared statistics on all pairs and selecting k best scoring features. The number of selected features for each pair were found out through tuning of the models and was finally settled at 380. The Binary Neural network was implemented using Keras**[cite]** which provide a higher level of abstraction on top of the theano**[cite]** backend. The underlying neural network consisted of 3 fully connected layers with the first layer called the input layer containing neurons equal to the selected number of features i.e 380. The second layer called hidden layer contained 22 neurons which were determined by extensive tuning of the model. The third layer called the output layer contained neurons equal to the number of languages to be identified which are 2 for the Binary case. The associated activation function for each of the neurons in the input and hidden layer is relu which stands for rectifier liner unit. The activation function used for the output layer is softmax which produces probability density as its output. The weights were initialized using a uniform initialization scheme and adadelta optimizer was used to control the learning rate. Activity regularization L1 and L2 was added at each layer to avoid over fitting. The model aims to minimize the error function which the and is the actual process through which the model learns. Binary crossentropy was used which is ideal for binary classification. The metric that we sought to maximize was accuracy which we found through extensive research was the only ideal metric for Speech related applications. All other parameters were left at default specified by the keras library. The model was trained for 30 epochs using a batch size of 30 samples. The data provided to the Neural Network was labeled data aimed at supervised learning. However, to adhere to the architecture of the Neural Network. The single output label was transformed into a 1 hot vector with the 1 corresponding to the actual spoken language. The labels specified here were consistent with the Initial Neural network. The data samples included in training were further shuffled using sklearn library before fitting the Binary Neural Network to the data. The model changes the weights and bias associated with each neuron in order to minimize the error function. The model generates the final predicted language according to the trained model.

 Fig **{number}** Representative Neural Network Architecture

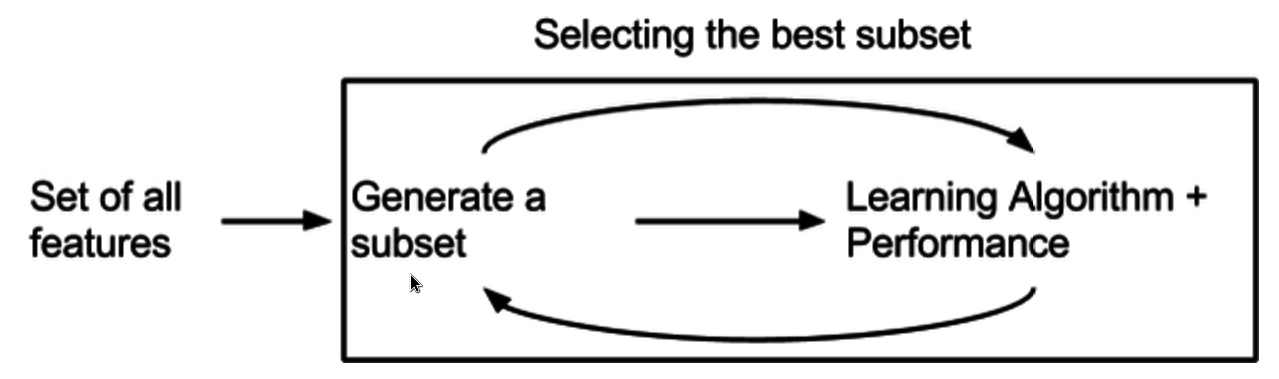


Fig **{number}** Feature Selection Process

**Appendix**

**Tool Installation**

**1) pyAudioAnalysis**

Install dependencies:

pip install numpy matplotlib scipy sklearn hmmlearn simplejson eyed3 pydub

Clone the source of this library:

git clone https://github.com/tyiannak/pyAudioAnalysis.git

**2) LibROSA**

The latest stable release is available on PyPI, and you can install it by saying

pip install librosa

librosa is also available on Anaconda. You can install it by saying

conda install -c conda-forge librosa

**3) NumPy**

Windows does not have any package manager analogous to that in Linux, so installing one of the scientific Python distributions mentioned above is preferred. However, if that is not an option, Christoph Gohlke provides pre-built Windows installers for many Python packages, including all of the core SciPy stack, which work extremely well.

**4) Theano**

Installation of the requirements supported only through conda.

* Python == 2.7\* or (>= 3.3 and < 3.6)
* NumPy >= 1.9.1 <= 1.12
* SciPy >= 0.14 < 0.17.1
* BLAS installation (with Level 3 functionality)

Assuming all the dependencies are already installed, theano can be installed as:  
**With conda**

If you use conda, you can directly install both theano and pygpu. Libgpuarray will be automatically installed as a dependency.

conda install theano pygpu

**With pip**

If you use pip, you have to install Theano and libgpuarray separately.

**theano**

Install the latest stable version of Theano with:

<sudo> pip install <--user> Theano[test, doc]

**libgpuarray**

For the stable version of Theano you need a specific version of libgpuarray, that has been tagged v0.6.2. Download it with:

git clone https:**//**github**.**com**/**Theano**/**libgpuarray**.**git

cd libgpuarray

git checkout tags**/**v0**.**6.2 **-**b v0**.**6.2

**Bleeding-Edge Installation (recommended)**

Install the latest, bleeding-edge, development version of Theano with:

<sudo> pip install <--user> <--no-deps> git+https://github.com/Theano/Theano.git#egg=Theano

**5) Keras**

Keras uses the following dependencies:

* numpy, scipy
* yaml
* HDF5 and h5py (optional, required if you use model saving/loading functions)
* Theano

It is assumed that all these dependencies are installed

To install Keras, cd to the Keras folder and run the install command:

sudo python setup.py install

You can also install Keras from PyPI:

sudo pip install keras

**6) scikit-learn**

Scikit-learn requires:

* Python (>= 2.6 or >= 3.3),
* NumPy (>= 1.6.1),
* SciPy (>= 0.9).

Assuming these dependencies are satisfied,

The easiest way to install scikit-learn is using pip

pip install -U scikit-learn

or conda:

conda install scikit-learn

**7) pyAudio (Real time analysis)**

Install using [pip](http://pip.readthedocs.org/en/stable/):

python -m pip install pyaudio

**8) PyQt**

Building and Installing from Source:

**Downloading SIP**

SIP must be installed before building and using PyQt5. You can get the latest release of the SIP source code from <https://www.riverbankcomputing.com/software/sip/download>.

**Downloading PyQt5**

You can get the latest release of the GPL version of the PyQt5 source code from <https://www.riverbankcomputing.com/software/pyqt/download5>.

**Configuring PyQt5**

In order to configure the build of PyQt5 you need to run the configure.py script as follows:

python3 configure.py

This assumes that the Python interpreter is on your path. Something like the following may be appropriate on Windows:

c:\Python36\python configure.py

If you have multiple versions of Python installed, then make sure you use the interpreter for which you wish to build PyQt5 for.

**Building PyQt5**

The next step is to build PyQt5 by running your platform’s make command. For example:

make

The final step is to install PyQt5 by running the following command:

make install

**9) matplotlib**

For standard Python installations, install matplotlib using pip:

python -m pip install -U pip setuptools

python -m pip install matplotlib