THE DEVELOPMENT OF AN AUTOMATIC PRONUNCIATION ASSISTANT

by

TSHEPHISHO JOSEPH SEFARA

DISSERTATION

Submitted in fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

in the

FACULTY OF SCIENCE AND AGRICULTURE

(School of Mathematical and Computer Sciences)

at the

UNIVERSITY OF LIMPOPO

SUPERVISOR: Mr MJD Manamela

CO-SUPERVISOR: Dr TI Modipa

2019

DEDICATION

This dissertation is dedicated to my family who sacrificed the time we should have spent together.

DECLARATION OF AUTHORSHIP

I, Tshephisho Joseph Sefara, declare that the dissertation entitled “THE DEVELOPMENT OF AN AUTOMATIC PRONUNCIATION ASSISTANT”, is my own work and has been generated by me as the result of my own original research proposal. I confirm that where collaboration with other people has taken place, or material from other researchers is included, the parties or material are appropriately indicated in the acknowledgements or references. I further confirm that this work has not been submitted to any other university for any other degree or examination.

_____

Sefara, T.J.

05/04/2019

Date

ACKNOWLEDGEMENTS

As I finish this academic work, I would like to thank almighty God for giving the strength, wisdom and guidance throughout the course of the research study. It is a great pleasure to express my thanks to the following people for contributing towards the success of this research study:

- Mr MJD Manamela and Dr TI Modipa, my supervisors, for their supervision and guidance without which this work would not have been a reality;
- Credit has to be given to ARMSCOR and CSIR for the financial support they provided for this research study;
- Special thanks must go to all the hard-working people that met with me every day and gave their one hundred and ten percent effort during evaluation of the system;
- Special thanks must go to my friends and colleagues, for their moral support.
- Finally, I would like to thank my amazing family for their love and support.

ABSTRACT

The pronunciation of words and phrases in any language involves careful manipulation of linguistic features. Factors such as age, motivation, accent, phonetics, stress and intonation sometimes cause a problem of inappropriate or incorrect pronunciation of words from non-native languages. Pronunciation of words using different phonological rules has a tendency of changing the meaning of those words. This study presents the development of an automatic pronunciation assistant system for under-resourced languages of Limpopo Province, namely, Sepedi, Xitsonga, Tshivenda and isiNdebele.

The aim of the proposed system is to help non-native speakers to learn appropriate and correct pronunciation of words/phrases in these under-resourced languages. The system is composed of a language identification module on the front-end side and a speech synthesis module on the back-end side. A support vector machine was compared to the baseline multinomial naive Bayes to build the language identification module. The language identification phase performs supervised multiclass text classification to predict a person's first language based on input text before the speech synthesis phase continues with pronunciation issues using the identified language. The speech synthesis on the back-end phase is composed of four baseline text-to-speech synthesis systems in selected target languages. These text-to-speech synthesis systems were based on the hidden Markov model method of development. Subjective listening tests were conducted to evaluate the performance of the quality of the synthesised speech using a mean opinion score test. The mean opinion score test obtained good performance results on all targeted languages for naturalness, pronunciation, pleasantness, understandability, intelligibility, overall quality of the system and user acceptance. The developed system has been implemented on a "real-live" production web-server for performance evaluation and stability testing using live data.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

LIST OF ABBREVIATIONS

API	Application Programming Interface
ARFF	Attribute-Relation File Format
CSIR	Council for Scientific and Industrial Research
CSV	Comma-Separated Values
DNN	Deep Neural Network
DRT	Diagnostic Rhyme Test
DSP	Digital Signal Processing
EMOTIONML	Emotion Markup Language
F0	Fundamental Frequency
FST	Finite State Transducer
G2P	Grapheme-to-Phoneme
GUI	Graphical User Interface
HLTs	Human Language Technologies
HMM	Hidden Markov Model
HTS	HMM-based Speech Synthesis System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
JSM	Joint Sequence Model
LIBSVM	Library for Support Vector Machine
LID	Language Identification
LTS	Letter-To-Sound

MAG	Fourier Magnitude
MARY	Modular Architecture for Research on speech sYnthesis
MFCCs	Mel-frequency Cepstral coefficients
MGC	Mel-Generalised Cepstral
MNB	Multinomial Naive Bayes
MOS	Mean Opinion Score
MRT	Modified Rhyme Test
NCHLT	National Centre for Human Language Technology
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
PHP	Hypertext Pre-Processor
POS	Part-of-Speech
RBF	Radial Basis Function
RMA	Resource Management Agency
SADE	South African Directory Enquiry
SAMPA	Speech Assessment Methods Phonetic Alphabet
SER	Sentence Error Rate
SOX	Sound Exchange
SPEECT	Speech Synthesis with Extensible Architecture
SPSS	Statistical Parametric Speech Synthesis
SPTK	Speech Signal Processing Toolkit
SSML	Speech Synthesis Markup Language
SUS	Semantically Unpredictable Sentence

SVC	Support Vector Classification
SVM	Support Vector Machine
TTS	Text-to-Speech
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VPS	Virtual Private Server
WEKA	Waikato Environment for Knowledge Analysis
WER	Word Error Rate
XML	Extensible Markup Language

1 CHAPTER 1: INTRODUCTION

1.1 Preamble

Within the realm of human-computer interaction systems, there are human language technologies (HLTs) that simplify communication between humans and computational systems. These technologies are available for languages such as English, Spanish, French and other well-known languages. HLT comprises text and speech processing technologies. The speech technologies make it possible for people to use computational devices such as mobile phones to access information, use email systems or even do voice dialling in their first language. The text technologies make it possible for people to manipulate historic textual data to make future predictions (e.g. weather predictions, financial markets, and others). Furthermore, text technologies are used to make spell checkers, and do text normalisation, text analytics, and text classification tasks.

Text classification is a challenging task in computational, library and information sciences. The main task involves the ability to classify or assign a text unit or document to a pre-defined class (Botha *et al.*, 2007). Text classification is used as language identification (LID) in language-specific systems. An automatic LID is a growing application of the speech processing technology that has many practical uses. It can be used as a front-end system to a telecommunication company - routing a caller to an appropriate human emergency operator - depending on the correct identification of the caller's language. It can be used in multilingual speech recognition (Rao & Nandi, 2015), speech translation (Heck *et al.*, 2012), and document processing systems (Shukla *et al.*, 2016).

LID is a problem of discovering the identity of the natural language of a given spoken or textual content (Lamabam & Chakma, 2016). In text and speech processing systems, LID of text units or documents is an important prerequisite for systems such as an automatic machine translation, information extraction, email spam filtering systems, topic identification systems, document summarisation systems, pronunciation prediction and text-to-speech (TTS) synthesis systems (Giwa & Davel, 2015).

A TTS synthesis system is a computational system that generates synthetic speech from a given input text in a specific language (Zen *et al.*, 2009). Figure 1.1 shows an example of a typical TTS synthesis system generating synthetic speech from text. There are different kinds of speech synthesis methods that are used when building synthetic voices. These methods include rule-driven synthesis and data-driven synthesis. Data-driven synthesis includes hidden Markov model (HMM)-based synthesis and concatenation synthesis. The HMM speech synthesis system (or HTS – 'H Triple S') uses the statistical parametric model that extracts speech parameters from the speech corpus to produce equivalent sound of an input text (Zen *et al.*, 2009). The TTS synthesis systems for well-resourced languages are becoming more readily commercially available in the market as the quality of these systems continues to improve rapidly. Mainly, commercial systems apply the unit-selection based concatenation method to generate high quality synthetic speech waveform. However, the unit-selection approach demands a very large database to store the pre-recorded training speech data. Avoiding such a problem, the HTS approach has successfully evolved with appreciable improvements to the concatenative method in terms of processing speed and utilisation time.

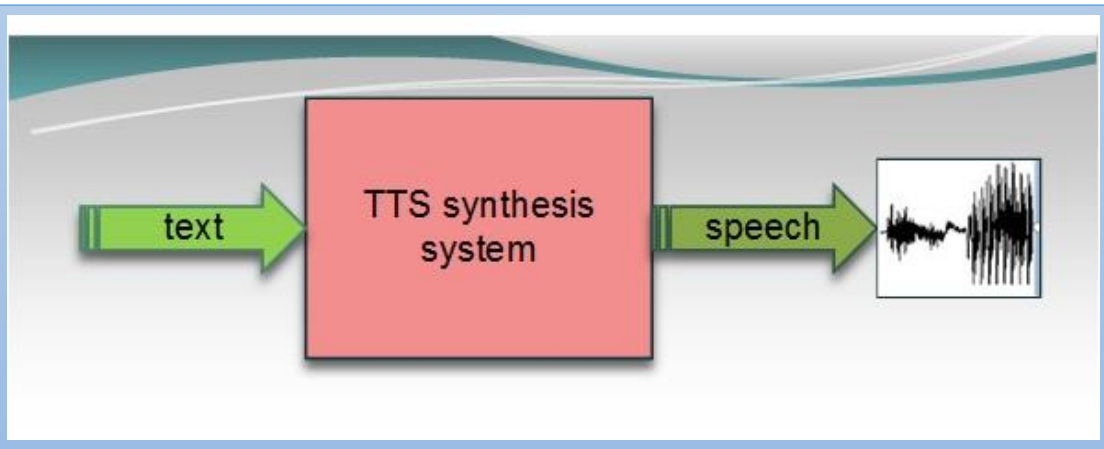


Figure 1.1: General TTS system showing text as input and speech waveform as output (Baloyi, 2012).

1.2 Motivation

The area of speech processing research has gradually advanced over recent decades with many systems developed with ability to produce natural sounding synthetic speech (Zen *et al.*, 2009). Research in the subfield of speech synthesis has been powered by the increase of new and robust software applications that have been developed (Pammi *et al.*, 2010). These include, amongst others, the reading out of manuscripts for collation, public announcements at public places and information retrieval services over the telephone such as customer care services or banking services (Violano & van Collie, 1992). In today's electronic digital age, the use of TTS synthesis technologies has increased exponentially in mobile smartphones, computers, internet-based services, and dialogue systems**.**

A TTS synthesis system can be embedded in special equipment as a voice-enabling tool for vocally challenged people[1]. FingerReader is one of the initiated applications of TTS synthesis that assists visually impaired people to read documents (Shilkrot *et al.*, 2014). Most of the developed software applications like word processors have capability to read words, phrases and sentences aloud. These systems provide a way for the visually impaired people to read text that would not be available to them. The current modern TTS synthesis systems such as the Watson [2] TTS and Microsoft Speak [3] function can synthesise documents such as Portable Document Format (Adobe Acrobat), emails, Microsoft Word files and text files. For most South African indigenous official languages, these enabling linguistic developments have not been attained by the speech and language processing research community. The research community in speech and language processing is rapidly improving its utility systems that can read images using optical character recognition systems (Li *et al.*, 2012). In the 1970s, Texas Instruments' "Speak and Spell"[4] was the first educational toy that applied speech synthesis in the field of computer-assisted education. The

[1] Available at: http://www.hawking.org.uk/the-computer.html
[2] Available at: http://www.ibm.com/watson/developercloud/text-to-speech/api/v1/
[3] Available at: https://support.office.com/en-us/article/use-the-speak-text-to-speech-feature-to-read-text-aloud-459e7704-a76d-4fe2-ab48-189d6b83333c
[4] Available at: http://www.ti.com/corp/docs/company/history/timeline/eps/1970/docs/78-speak-spell_introduced.htm

TTS synthesis systems are used in education institutions to improve learning the pronunciation of new languages (Badenhorst *et al.*, 2006).

1.3 Problem Statement

The pronunciation of words and phrases in any language involves careful manipulation of stress, intonation, and articulation, frequently with reference to some standard of correctness (Gilakjani & Ahmadi, 2011). Most of the time, the pronunciation of words (particularly proper names from new or unfamiliar languages) is difficult for non-native speakers of those languages. In our approach, it is important and preferable to attempt identifying the first language of a person using something like that person's proper name details in order to facilitate selection of appropriate tools to generate its correct pronunciation (Llitjos & Black, 2001). Knowledge of or ability to predict the first language associated with a word or a proper name may reduce the pronunciation difficulties experienced by non-native speakers by appropriately applying phonetic rules from that first language.

One of the language processing phenomena required in a multilingual environment is mostly established using LID. In a multilingual environment like South Africa, the problem of incorrect pronunciation of words or phrases from other languages is sometimes caused by the following (Kenworthy, 1987):

- **Phonetics** – pronunciation of a word using different phonological rules.
- **Age** – production of speech is affected at different age groups.
- **Motivation** – lack of motivation to learn how to speak other languages.
- **Accent** – naturally, people use their first language accent to speak other languages; the use of strong accent changes or distorts the meaning of the words.
- **Stress and intonation** – randomly applying strong stress or intonation on some words changes the meaning of those words.

The use of an LID system on the front-end in speech and language processing systems helps to determine the language of an input audio or text before a TTS synthesis system can produce corresponding synthetic speech. This has an effect of reducing computational complexity and searching time (Rao & Nandi, 2015). Current estimates indicate that there are more than 7000 natural languages in the world, but the TTS synthesis systems are not readily available for thousands of under-resourced languages (Lewis *et al.*, 2016), including South African official languages. The unavailability of TTS synthesis systems worldwide often leads to difficulties in correct pronunciation of words and phrases from most of the under-resourced languages. One of the possible solutions to improve quality of pronunciation in resource-scarce languages is to build TTS synthesis systems for the benefit of assisting users in the learning of correct pronunciation of other languages. To this end, the integration of a text-based LID system and a TTS synthesis system may increase accuracy with automatic pronunciation of proper names for under-resourced languages of Limpopo Province.

1.3.1 Aim

The aim of the study is to build a prototype software system that uses a trained classifier to enhance pronunciation of words and phrases, particularly proper names for Sepedi, Tshivenda, Xitsonga and isiNdebele.

1.3.2 Objectives

The objectives of this study are to:

a) Collect training text data consisting of proper names including surnames or maiden names for the front-end LID system.
b) Acquire pre-recorded speech data to create synthetic voices in Sepedi, Xitsonga, Tshivenda and isiNdebele.

c) Use appropriate machine-learning algorithms to train text-based LID front-end system for classification of surnames into respective first languages.
d) Use a trained LID system to predict the first language of a person given that person's proper name.
e) Activate the TTS synthesis system of that first language to continue with pronunciation guidance and training, using the predicted first language automatically.
f) Evaluate the performance of integrated LID-TTS systems.

1.3.3 Research Questions

The main research questions of the study are as follows:

- Can a computational system use a person's surname to predict the identity the first language of that person?
- Can a computational system produce an appropriate pronunciation of indigenous proper names?

1.4 Research Methods

The speech training data were acquired from the Lwazi (a word which means "knowledge" in isiZulu) project in collaboration with the Council for Scientific and Industrial Research (CSIR), South African Department of Arts and Culture, and Department of Science and Technology (Language Resource Management Agency, 2016). An appropriate text dataset with surnames was acquired from the student database of University of Limpopo for the targeted under-resourced languages. Appropriate classification methods were tried and tested before the best method was selected to implement the proposed system. The acquired text data was used to train machine-learning algorithms such as multinomial naive Bayes (MNB) and support vector machines (SVMs) (Chang & Lin, 2011).

The Eclipse software and Waikato Environment for Knowledge Analysis (WEKA) Java application programming interface (API) (Hall *et al.*, 2009) were used to build the LID model. A machine-learning classifier was trained and used as the

core LID module classifying any given input text. The accuracy of the trained machine-learning classifier was evaluated on a dataset which was not used during the training of the classifier. The Modular Architecture for Research on speech sYnthesis (MARY) TTS synthesis system (Pammi *et al.*, 2010), which supports multilingual languages, was used to build new TTS synthesis voices in Sepedi, Tshivenda, Xitsonga and isiNdebele. The system integration for a user-friendly human-computer interface was built using Java programming language.

The evaluation of the performance of the developed system was conducted in a form of a questionnaire survey, using first language speakers from different professions. The data gathered from the survey was analysed using the mean opinion score (MOS) method for naturalness, pronunciation, pleasantness, understandability, intelligibility, overall quality of the system and user acceptance. The MOS provides a numerical measure of the quality of human speech using a Likert scale. The system was deployed to the production servers as a website for performance evaluation on a "real-world" platform including test usability and acceptability issues. An Android application was also developed. This application acts as a client to the deployed system on the internet. The prototype TTS synthesis system can be accessed on the website http://www.speechtech.co.za.

1.5 Scientific Contribution

This research project intends to deliver an intelligible and natural sounding TTS synthesis prototype system that embeds a first language identification predictor on the front-end, based on the surname of user. The integration of LID as a front-end module has a greater chance of increasing the pronunciation quality of the TTS synthesis system.

This study contributes to the provision of new attribute-driven LID systems to facilitate selection of an appropriate first language speech synthesiser that will assist with pronunciation of words/phrases/sentences in a specific under-resourced language to non-native speakers of that language. This system can be

used as a learning tool in educational institutions to help and facilitate learners at different levels to learn additional South African official languages. For a multilingual region such as Limpopo Province with more than five official languages and wherein the majority of people speaking three of those languages, the outputs of this study may be helpful to people learning a second language.

This study is the first research study to create and deploy an automatic pronunciation assistance system for targeted under-resourced South African languages. The availability of diverse TTS synthesis systems for under-resourced languages of South Africa is found to be low. There are four official languages of South Africa (isiZulu, Sesotho, Afrikaans and isiXhosa) included in Google translate[1] excluding well-known global lingua franca, English. Although research in TTS synthesis systems in South Africa is relatively young, many TTS synthesis systems efforts occurring in South Africa have acquired international awareness and exposure in terms of the quality and impact of the research work (Louw, 2008). This shows that more research work is still required on TTS synthesis systems for all official South African languages, including their varied dialects (Langa *et al.*, 2012). The development of pronunciation assistance systems for these languages tries to bridge the digital divide by striving to create user-friendly interfaces. Speech synthesis systems have an important role of lessening the impact of the historical linguistic discrimination and domination imposed onto marginalised and under-resourced indigenous South African languages by colonial powers. This study tries to enhance and elevate the recognition and use of indigenous South African official languages in the broader information and communication technology (ICT) sector.

This system can help people wishing to learn pronunciation of surnames and praise names in Sepedi, Xitsonga, Tshivenda, and isiNdebele; an important aspect within a typical greeting episode amongst people from differing speaker population groups. A variety of TTS synthesis systems applications are available in the following fields:

[1] Available at: https://translate.google.com

a) *Education* – interactive language learning software may optimise educational opportunities especially for disabled leaners (Singh & Kaur, 2015).
b) *Economics* – speech-based systems may improve customer services and provide information about services and products.
c) *Financial services* – speech-driven automatic teller machines (Violano & van Collie, 1992).
d) *Telecommunications* – in medical consultations (Kourkouta & Papathanasiou, 2014).
e) (e) *Information management* – improved access to documents through search engines.

1.6 Ethical Considerations

The term *ethics* is defined as a set of moral principles and rules aimed to protect the interest of the participants when conducting research (Julnes & Bustelo, 2014). The following ethical issues were considered during the course of the study:

1.6.1 Informed Consent

Informed consent was gained from the subjects by means of a written and verbal agreement. The researcher informed subjects about the study, its goals, the rights of the subjects, and information confidentiality.

1.6.2 Voluntary participation

The participants were informed that their participation in the study was voluntary and they could withdraw at any time. Subjects were not forced to take part in the study.

1.6.3 Privacy and Confidentiality

The subjects were assured that all the information will be treated in strict confidence, their answers will be kept confidential, and no one will have access to them. Data coding was used to link data to the study participants, and this code was kept in privacy, so no names will be enclosed in analysis and report writing.

1.6.4 Physical or Psychological Harm

The researcher neither subjected the participants to any physical harm and nor forced the participants to provide answers to questions they did not want to answer.

1.7 Structure of Dissertation

The rest of the dissertation is organised as follows:

- Chapter 2 provides previous studies on LID and speech synthesis.
- Chapter 3 presents a detailed description of the design and implementation of the integrated system.
- Chapter 4 presents the research findings of the study. The evaluation metrics for LID front-end system and subjective listening test are discussed. The optimum SVM parameters are outlined and the evaluation procedure is presented.
- Chapter 5 presents the analysis of the performance results of LID system, speech synthesis, system usability, and summary of the findings.
- Chapter 6 presents research limitation, provides contribution of this work, recommends potential directions of the future and provides the conclusion of the research study.

2 CHAPTER 2: BACKGROUND

In this chapter, we review the text classification technologies literature used in machine-learning fields before giving an overview of the speech synthesis methods and the components composing the state-of-the-art in TTS synthesis systems. We also have a brief discussion of the application areas and current development tools of the technology.

2.1 Introduction

The human language technologies (HLTs) make it simple for humans to communicate with machines. These can furthermore assist corporate industries and government departments to make e-services and information accessible to the society at large in different languages. Most smart computational systems such as Siri[1] can use HLTs to facilitate man-machine communication. The HLTs have an important task by participating in adjusting the historical linguistic discrimination imposed onto under-resourced indigenous South African languages by levelling the language playing field. The speech and language technologies are the vital core part of HLTs. Their main purpose is to make machines “speak”, “listen” and “understand” natural or human languages. This chapter details the broader aspects of the background to text and speech technology.

This chapter is organised as follows:

- Section 2.2 explains the importance of proper names in communication episodes.
- Section 2.3 explains the importance of correct and appropriate pronunciation.
- Section 2.4 overviews supervised machine-learning algorithms.
- Section 2.5 discusses applied LID studies.

[1] Available at: http://www.apple.com/ios/siri/

- Section 2.6 discusses toolkits used for training and implementation of classifiers.
- Section 2.7 briefly outlines basic components of TTS synthesis systems including different types of speech synthesis methods such as formant synthesis, articulatory synthesis, concatenative synthesis, and statistical parametric speech synthesis (SPSS) using HMMs and deep neural networks (DNNs).
- Section 2.8 details evaluation methods and factors or units in TTS synthesis systems.
- Section 2.9 reviews application areas of TTS synthesis systems.
- Section 2.10 discusses toolkits used for development of TTS synthesis systems.

2.2 Proper Names

In many cultures and traditions worldwide, people are given names from family members and relatives; some may be named after a saint, family member, weather, or a positive personality characteristic. These names may recall an event or describe the position of the star at birth, or state a future ambition (Guma, 2001). Since people are unique, their names are also unique because these names are attached to their cultural identity and they would not wish to have their names mispronounced in conversations. Names, specifically surnames in the culture of the Basotho of Southern Africa, carry important information about ones' history such as physical original geographic location, identity, clan name (totem), first language, culture and heritage, and ethnic group (Guma, 2001).

As the world becomes increasingly connected, cross-cultural communication increases, and when mispronouncing one's name, it is often deemed to be or associated with a misinterpretation of that person's identity. For instance, continued mispronouncing a student's names may contribute to lessening or belittling the identity of that student and this can lead to unexpected anxiety and

resentment which, in turn, can retard the student's academic progress. *My name my identity*[1] is an American campaign that brings recognition of appreciating one's name and identity in schools. One of the main goals of this campaign is to create a humble and caring culture in academic institutions that values diversity as measured by name stories posted on social media.

2.3 Pronunciation

Most people agree that for someone to fluently pronounce a second language like a native speaker, they most likely must have learned it at their childhood stage. Conversely, if a learner does not begin to learn a second language until adulthood, they would not have a native-like accent. These beliefs or observations seem to be supported by cases of adults who learn to speak second languages fluently but still maintain their first language accent (Gilakjani & Ahmadi, 2011). There are factors that result in systematic pronunciation differences between speakers including age, first language, accent (a manner of pronunciation peculiar to a particular individual), speaker's geographical location, cultural groups and socio-economic status (Kenworthy, 1987).

In language education, Gilakjani (2012) defines pronunciation as a vital part of foreign language learning since it directly affects the performance and communicative competence of the learners. The important key and *sine qua non* for language proficiency is the exposure to the target language, attitude, motivation, and instruction of learning or teaching. There are pronunciation features that may apply to any spoken language. Gilakjani (2012) identified the most important features in English pronunciation to include segmental features (phonemes) and prosody features (linking, stress and intonation). A prosodic feature relates to small units of a sound within a word. This sound can be a combination of consonants and vowels (Jurafsky & Martin, 2014).

[1] Available at: https://www.mynamemyidentity.org/campaign/about

It is generally agreed that incorrect pronunciation of phonemes changes the meaning of the word (Suortti & Lipponen, 2014). The sound of a typical vowel has the following distinct and characteristic features: length, height, roundness and frontness, and the consonant sound has the following distinct and characteristic features: type, place of articulation, and voicing. These features constitute the initial requirement during the development process of a new TTS voice (Stavropoulou *et al.*, 2014).

The knowledge of their probable first language can generally improve pronunciation of proper names (Llitjos & Black, 2001). Machine-learning technologies can be used to ascertain and predict the first language given a person's proper name. These technologies may use linguistic rules for a specific language (rule-driven machine learning) or require learning proper names data from a (un)labelled database (data-driven machine learning).

2.4 Supervised Learning Techniques

An automatic arrangement of documents is an important data mining research matter since the advent of online text information. There are two types of machine-learning approaches for classification of text documents, namely supervised and unsupervised learning (Rana *et al.*, 2016). In unsupervised learning, the dataset is not labelled; instead the data can be clustered into different classes, whereas in supervised learning, a classifier is developed containing predefined class labels that are assigned to documents/text based on the probability recommended by a training dataset of labelled documents/text.

Machine-learning algorithms can be used to automatically build LID classifiers (Rana *et al.*, 2016). The procedure of building a classifier can be seen as a problem of supervised learning whereby an algorithm obtains a labelled dataset to build the classifier. Several machine-learning techniques in text classification have been applied, including *n*-gram rank ordering (Cavnar & Trenkle, 1994), decision trees (Farid *et al.*, 2014), logistic regression (Yuan *et al.*, 2012), the naive

Bayes classifiers (Fourie *et al.*, 2014), neural networks (Nicolaou *et al.*, 2016), k-nearest neighbour classifiers (Al-Badarenah *et al.*, 2016), and support vector machines (SVMs) (Fourie *et al.*, 2014; Mabokela & Manamela, 2013).

2.4.1 Multinomial Naive Bayes Classifier

The naive Bayes classifier is a probabilistic model that uses the joint probabilities of terms and classes to estimate the probabilities of classes given a test document (Mitchell, 1997). There are two event models commonly used: multivariate Bernoulli event model and multinomial event model, commonly called MNB (Kibriya *et al.*, 2004). MNB classifies text given a set of classes $C = \{c_1, c_2, \dots, c_k\}$, and a set of unique words $W = \{w_1, w_2, \dots, w_n\}$, and $N = \{1,2, \dots, n\}$ defines the size of the vocabulary where $k > 1$ and $n \geq i \geq 1$. Then MNB assigns a test document d_i to a class that has a highest probability $p(c|d_i)$, as given by the Bayes' rule in Equation (2.1):

$$p(c|d_i) = \frac{p(c)p(d_i|c)}{p(d_i)}, c \in C \tag{2.1}$$

From Equation (2.1), $p(c|d_i)$ is the posterior probability of a class, and $p(c)$, the prior probability of a class can be predicted by dividing the number of documents in a class c by the total number of documents; $p(d_i|c)$ is the probability of a document d_i given its class c and is calculated as:

$$p(d_i|c) = (\sum_n f_{ni})! \prod_n \frac{P(w_n|c)^{f_{ni}}}{f_{ni}!}, \tag{2.2}$$

where f_{ni} denotes a count of word n for test document d_i and $p(w_n|c)$ denotes the probability of word n given class c and is estimated as:

$$\hat{p}(w_n|c) = \frac{1+F_{nc}}{N+\sum_{x=1}^{N} F_{xc}}, \tag{2.3}$$

where F_{xc} is the number of word x from all training documents belonging to class c, and the Laplace estimator is used to prime each word's count with one to avoid the zero-frequency problem. The normalization factor $p(d_i)$ in Equation (2.1) can be determined using:

$$p(d_i) = \sum_{k=1}^{|c|} p(k)p(d_i|k) \tag{2.4}$$

From equation (2.2), the computational expensive terms $(\sum_n f_{ni})!$ and $\prod_n f_{ni}!$ do not depend on class c and can be removed. Therefore, Equation (2.2) can be rewritten as:

$$p(d_i|c) = \beta \prod_n p(w_n|c)^{f_{ni}}, \tag{2.5}$$

where β is a constant and can also be removed. Equation (2.5) can be expanded by substituting Equation (2.3) to form Equation (2.6):

$$p(d_i|c) = \prod_n \left(\frac{1+F_{nc}}{N+\sum_{x=1}^{N} F_{xc}}\right)^{f_{ni}}. \tag{2.6}$$

Given the estimates of these parameters calculated from the training documents, classification can be performed on test documents by calculating the posterior probability in Equation (2.1). The right hand side of Equation (2.1) can be expanded by substituting with Equation (2.4) and (2.6). Naive Bayes learning is frequently used to solve text classification problems. This approach is known to perform best on textual data to become the common baseline classifier in text classifications (Fourie *et al.*, 2014).

2.4.2 Support Vector Machines

A SVM is a technique based on the structural risk minimisation principle (Burges, 1998) that is defined as an inductive principle for model selection used for learning from finite training data. A SVM is a popular machine-learning method for regression, classification and other learning tasks (Suthaharan, 2016). The SVMs were designed to solve problems with two classes initially, but can now be used for more than two classes (also known as multiclass). Multiclass classification is the problem of classifying instances into one of more than two classes, and binary classification is a task of classifying instances into one of the two classes. In a binary classification, the task is to find the decision surface that separates the positive and negative training samples of a class with a maximum margin (see Figure 2.1). Samples closest to the decision surface are called support vectors. A margin is the distance from the decision hyperplane to the support vectors. Figure 2.1 shows an example of a linearly separable data. Non-separable data requires sophisticated classification algorithms. SVMs solve multiclass classification using the *kernel trick* and slack variables (van Heerden, 2012). Chang and Lin (2011) developed a library for the support vector machine (LibSVM) that supports various SVM formulations for regression, classification and distribution estimation. Moreover, Chang and Lin (2011) discussed issues such as solving SVM optimisation problems, probability estimates, multiclas classification, theoretical convergence, and parameter selection. Some SVMs contain parameters that need to be optimised for better performance. Some of the kernels include, among others, linear, polynomial, radial basis function (RBF) and sigmoid kernel. The formulations or equations of these kernels are further discussed by Chang and Lin (2011). Even though SVMs do not support string attributes, there are several data processing tools that can be utilised to convert text into feature representation (Hall *et al.*, 2009). This enables the use of SVMs in text or document classification.

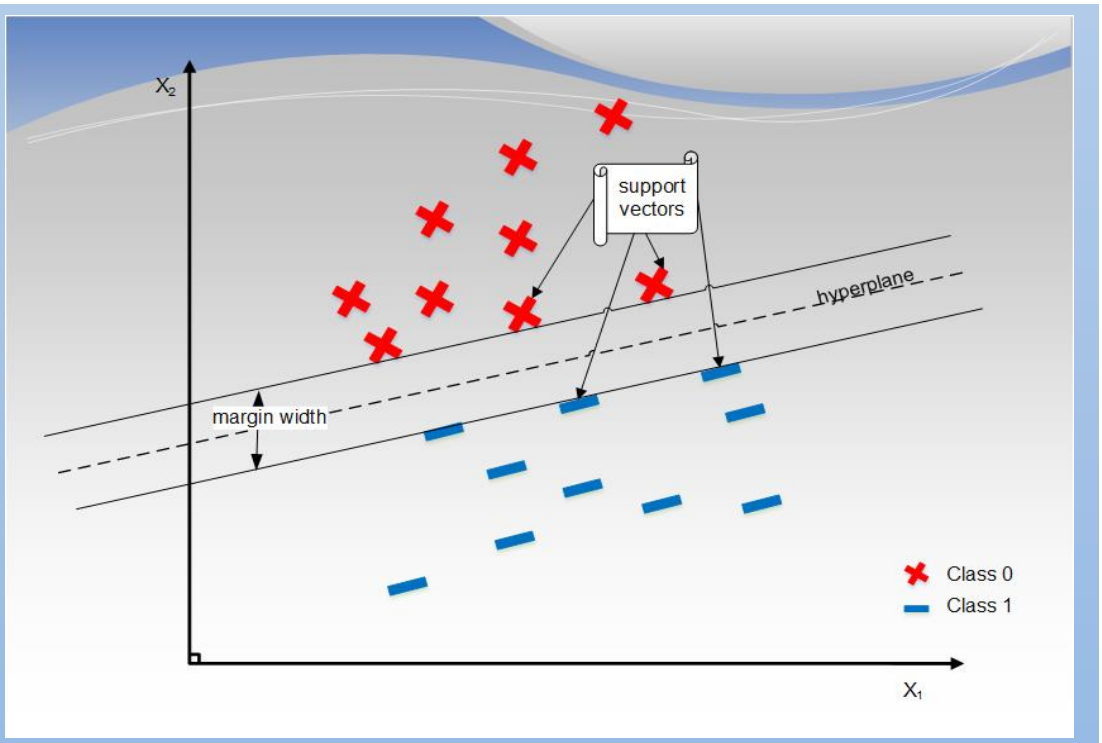


Figure 2.1: Example of a binary classification. Adapted from Chang and Lin (2011).

2.5 Language Identification

The goal of LID in speech and language processing is to classify a document or text based on the language of the document or text. An automatic LID is an essential requirement to any language-based processing. For instance, LID has many practical uses, including pre-selecting a speech synthesiser depending on the language of the input text in automatic machine translation systems. In text and speech processing systems, identification of an input text is important for applications such as an automatic machine translation, information extraction, pronunciation prediction and speech synthesis systems. The task of identifying a language from text or document is solved by applying text classification method. This text classification (also known as text categorisation) is a task of automatically associating a given text with one or more predefined classes (Agarwal & Mittal, 2012). The classes can be nominal and hold types of weather, gender, set of languages, or any other type of data.

Rule-based algorithms are used to solve the LID problem with prior linguistic knowledge of the target language. Indhuja *et al.* (2014) used *n*-grams to investigate the performance of statistical measures on a Devanagari script for LID on text. They extracted features such as unigram, bigram and trigram at word and character level. Bigrams are pair of consecutive words, syllables, or letters. For example, a word “champion” has character bigrams *ch, ha, am, mp* etc. The character level trigram reached an accuracy of 82%, while word level unigram reached the highest accuracy of 88% when testing with all five languages. This results shows that at word level the accuracy is high, meaning that words from these five languages are written differently and do not overlap. Moreover, low accuracy of trigrams can mean more features were identified as belonging to more than one language. Similarly, Hannan and Sarma (2015) applied rule-based analysis to design and implement the text-based LID system for Indian languages (Assamese and Bodo) following an Assamese-Bengali script. For the best LID, Unicode range, suffix and frequent word comparison were main features. The Unicode range checks the Unicode of each character in a word. In their case Unicodes between 0980 and 09FF are from an Assamese-Bengali script, and Unicodes between 0900 and 097F are from a Devanagari script. The suffix module accepts a word and verifies if the suffix of that word is available in the generated suffix list, while the frequent word module compares the word with the frequent words list. However, the algorithm reached 100% accuracy for a small sized text file of approximately ten words. However, as the word size increases, the accuracy of the Bodo language decreases to 77.35% while the accuracy of the Assamese language remains above 97.75%. Devanagari script is a writing system used in India for writing languages like Nepal, Marathi, Hindi, Sanskrit, Bhojpuri and other dialects. However, Indhuja *et al.* (2014) and Hannan and Sarma (2015) did not use recent well-performing statistical machine-learning algorithms such as SVMs and decision trees, and can still be experimented with on these languages using the same feature set.

As textual data is becoming more and more available online, the LID of web pages is an initial requirement when processing multilingual web documents, performing web page translation, and when retrieving data using web crawlers.

Kordestanchi and Naderi (2013) noted that language identification tools can perform differently on the same dataset. They compared the tools used in LID of web pages in Farsi (Persian), Arabic, and Urdu languages using *n*-grams as features. The Java tools used were Java Text Categorization Library (JTCL)[1], Language Detection[2], Tika[3], and JLangDetect[4]. The Web pages were classified according to a "clean web" data set that contains web pages that are similar to text that is present in non-webpage noise-free documents, and an "ordinary web" data set which contains web pages that are likely to have spelling or grammatical errors and possibly having limited content. Kordestanchi and Naderi (2013) found that JTCL and Language Detection outperformed other tools with respect to accuracy measures including recall, precision, negative recall, and F-measure. This study can further be extended to examine and minimise the root mean squared errors that can help in reducing errors to achieve higher overall accuracy.

Persian texts are difficult to classify in text classification because there are no explicit whitespaces included between words. This means proper word segmentation is required before further processing. Farhoodi *et al.* (2011) examined text classification using word level n-grams of different units from newspaper corpus on Persian text. They observed that trigram language models perform better, with or without linguistic pre-processing. They also examined the influence of smoothing methods on the trigram language model, and a back-off smoothing method obtained better accuracy outperforming add-one and absolute discounting smoothing methods.

2.5.1 Language Identification for South African Languages

Although research in LID systems for South African official languages is relatively young, a few LID efforts occurring in South Africa have acquired international awareness and exposure in terms of the quality and impact of the research work (Giwa & Davel, 2015; 2013; Botha & Barnard, 2012; Botha *et al.*, 2007). Several

[1] Available at: http://textcat.sourceforge.net/
[2] Available at: https://github.com/shuyo/language-detection
[3] Available at: https://tika.apache.org/1.1/detection.html
[4] Available at: https://github.com/melix/jlangdetect

research projects in text processing technology has been done on LID of South African indigenous languages. Botha *et al*. (2007) conducted the LID research to address the problem of under-resourced languages of South Africa using SVMs, naive Bayes and difference-in-frequency classifiers to investigate the accuracy achievable for all eleven official languages of South Africa, with data obtained from sources such as newspapers, the Bible, books, government documents and periodicals. The SVMs performed better for an *n*-gram size of three units but when increasing *n*-gram size to six units, the likelihood-based classifier outperformed other classifiers. Giwa and Davel (2015) adapted a text-based LID from their previous study (Giwa & Davel, 2014) to perform multilingual word classification from two corpora: the South African Directory Enquiry (SADE) corpus composed of Afrikaans, English, isiZulu and Sesotho, and the National Centre for Human Language Technologies (NCHLT) 40k models using a joint sequence model (JSM). The NCHLT 40k models showed better results than SADE, with 81.79% to 79.99% F-measure difference respectively. Hence, the adapted version of JSM provided good classification accuracy on a challenging task.

Fourie *et al*. (2014) compared the SVM and MNB classifiers for named entity classification of English and Afrikaans. They used WEKA toolkit to conduct the experiments using MNB and SVM algorithms. Implementation of the baseline SVM classifier was through Platt's Sequential Minimal Optimization algorithm. The data was converted with a string-to-word vector filter whereby words in the data were defined as classes and strings were converted to decimal arrays. The goal of named entity recognition and classification is to classify and recognise textual units (commonly referred to as named entities). The classification of proper names may be difficult in a situation where these names have idiosyncratic spelling and some proper names are considered multilingual (in other words, belonging to two or more languages). Their experiment showed that using 10-fold cross-validation SVMs performed better than MNB models across all granularity levels and both languages. Botha and Barnard (2012) also discussed various factors that affect text-based LID accuracy. These factors included *n*-gram size, text input size, training data, and machine learning algorithm employed, as well

as language similarities. Giwa and Davel (2013) discussed factors that influence LID accuracy of individual words for official languages of South Africa. Their experiment resulted with RBF SVM outperforming naive Bayes classifiers using Witten-Bell smoothing technique.

2.6 Toolkits for Classifier Implementation

2.6.1 WEKA

Most studies in machine learning use WEKA toolkit (Hall *et al.*, 2009). The WEKA workbench is a collection of state-of-the-art data processing tools and machine learning algorithms implemented in Java. WEKA provides extensive support for the whole process of experimental data mining. It has the option of statistically evaluating various learning methods, and visualising input data and learning results. We use machine learning algorithms implemented in WEKA.

2.6.2 Scikit-learn

A Python-based scikit-learn is one popular machine-learning library written in Python programming language (Pedregosa *et al.*, 2011). It offers methods for data mining and data analysis, including classification, regression, data preparation, and many others.

2.6.3 NLTK

Natural language toolkit (NLTK) is a Python-based library that offers methods to work with human language data (Bird, 2006). NLTK offers access to textual corpora and lexical resources such as WordNet along with text analysis libraries for regression, tokenisation, classification, tagging, and many others.

2.7 Components of a Typical TTS Synthesis System

This section details the basic architecture of a TTS synthesis system. The main components of TTS synthesis system are natural language processing (NLP) and digital signal processing (DSP). The modules of the NLP component are reviewed. The types of speech synthesis are also reviewed.

The NLP component is capable of producing a phonetic transcription of the text read, together with the desired intonation and rhythm (Huang *et al*., 2001), while DSP is the process of modifying and analysing speech signals to improve its performance based on linguistic features from NLP. DSP uses appropriate speech synthesis methods for proper speech generation.

2.7.1 Natural Language Processing

The NLP modules shown in Figure 2.2 consist of the text, phonetic and prosodic analysis that consists of the following sub-modules:

Document structure detection module in text analysis provides a context for all other modules. Some elements of document structure including paragraph, sentence, and word segmentation may have direct consequences for prosody.

Text normalisation module in text analysis converts acronyms, numbers, emails, websites, dates, times, currencies, mathematical expressions, percentages, measures, years, abbreviations, and other non-standard orthographic entities of text into word format when needed.

Linguistic analysis module in text analysis recovers the syntactic constituency and semantic features of words, phrases, clauses, and sentences.

Homograph disambiguation in phonetic analysis removes ambiguity from homographs. Homographs are words that share same spelling but have different meaning or pronunciation, for example, *desert* (*/dI"z@:t/*) as a verb or as a noun (*/"dEz@t/*) in Speech Assessment Methods Phonetic Alphabet (SAMPA) notation.

Morphological analysis module in phonetic analysis analyses component morphemes to provide indication or hint for pronunciation of similar words.

Letter-to-sound (LTS) conversion module is the last step of the phonetic analysis. Once all non-standard words are expanded and looked up in a pronunciation dictionary, then unknown words need to be pronounced by converting series of letters into a series of phones. This process is called grapheme-to-phoneme (G2P) conversion (Vasek *et al.*, 2016).

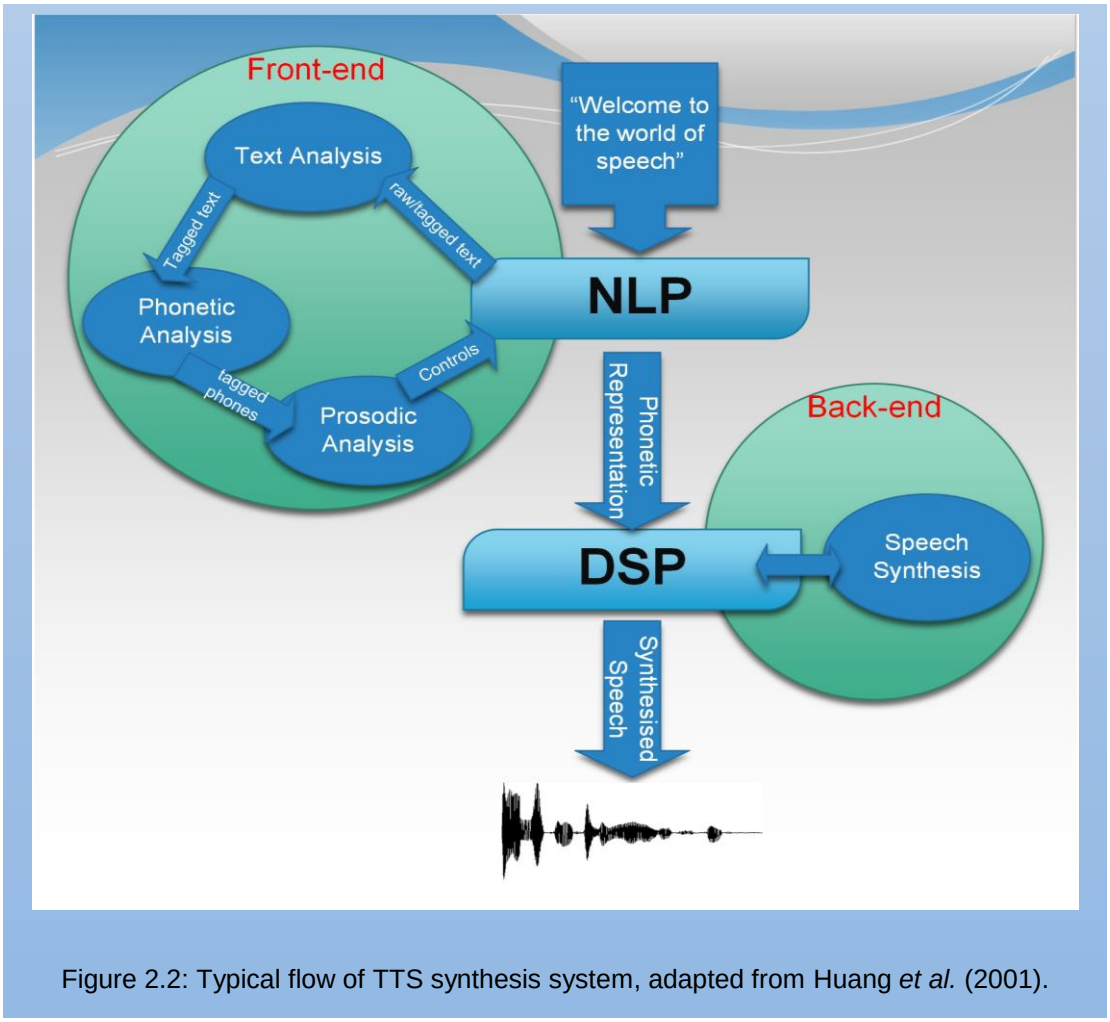


Figure 2.2: Typical flow of TTS synthesis system, adapted from Huang *et al.* (2001).

Prosodic analysis is the study of the intonational (includes prominence and phrasing) and rhythmic aspects of language contextual analysis. Prosody can be affected by emotion, mental state and speaker attitude (Taylor, 2009). From the listener's point of view, prosody consists of recovery of a speaker's intentions and systematic perception based on *pauses*, *pitch*, *duration* and *loudness*.

2.7.2 Digital Signal Processing

Speech synthesis takes place in the DSP component. Several speech synthesis methods exist, including the rule-based and data-driven methods. Rule-based method is sometimes called *synthesis-by-rule* and refers to a collection of rules defining how to adjust the formant frequencies, duration, pitch, and other parameters from one sound to another, while preserving continuity present in physical systems like the human production system (Huang *et al.* 2001). Formant and articulatory synthesis are good examples of rule-based synthesis. Data-driven method includes the concatenation synthesis method that takes advantage of the rich and large amount of speech corpus. The term concatenative synthesis refers to the use of segments of pre-recorded speech data to assemble the resulting speech waveform (Tiomkin *et al*. 2011).

2.7.2.1 Formant speech synthesis

Figure 2.3 describes a formant synthesiser receiving phonetic representation and generating waveform from a set of parameters. Pitch and formants are shown as the parameters of the synthesiser, but there are more than 40 parameters. Huang *et al*. (2001) detailed the architecture of the formant speech synthesis. Formant synthesisers can produce intelligible speech although the produced speech is far from natural.

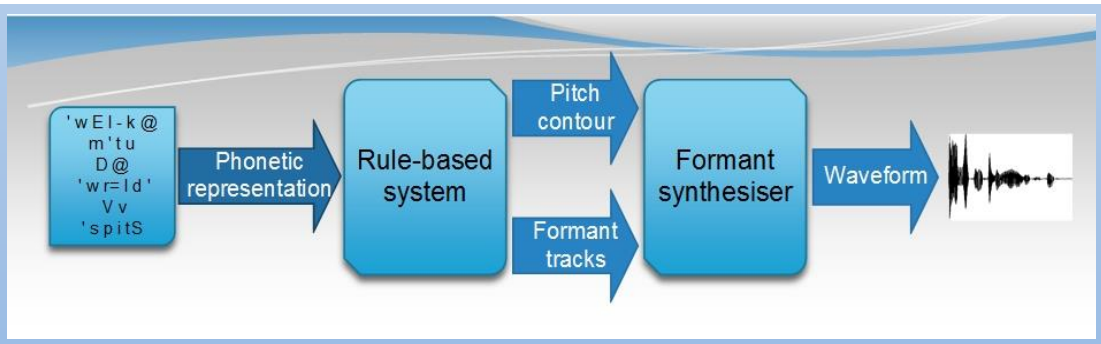


Figure 2.3: Diagram of a rule-based formant synthesiser system adapted from Huang *et al.*

2.7.2.2 Articulatory speech synthesis

One way to synthesise speech is to try a direct simulation of human speech production, and this procedure is called articulatory synthesis. Articulatory synthesis is another rule-based synthesis that uses parameters that model the mechanical motions of the articulators and the resulting distributions of volume velocity and sound pressure in the lungs, larynx, and vocal and nasal tracts (Flanagan *et al.*, 1975). There are two challenges in articulatory synthesis. The first challenge is data acquisition for the articulatory model and this data is normally obtained from X-ray images that do not describe the degree of freedom of the articulators. The second challenge is to discover stability across an accurate model and a model that is simple to control and design (Klatt, 1987). In general, this method was found to be the best approach to synthesise speech, although the state-of-the-art in articulatory synthesis does not produce speech with high quality as compared to that of formant or concatenative systems.

2.7.2.3 Concatenative speech synthesis

While the rule-based synthesis is quite intelligible, it is nonetheless sounding unnatural because it is very difficult to store all the sounds of natural speech in a minor set of manually derived rules. In concatenative synthesis, a speech segment is synthesised by concatenating together several speech fragments from the speech corpus (see Figure 2.4). Concatenative synthesis is a good example of a corpus-based speech synthesis – sometimes called an example-based approach. The advantage of this method is that each segment is completely natural, and a high quality of speech output is expected. When designing a concatenative speech synthesis, we can use diphones, syllables, phonemes, words, and phrases. Diphone synthesis is one of concatenative speech synthesis that produces flexible synthesised speech, although it lacks naturalness, pleasantness and understandability (Lemmetty, 1999). A diphone is an adjacent pair of phones. Diphones can be extracted from a set of nonsense words or natural words. The nonsense word technique has the advantage of

being simpler to ensure diphone coverage of all possible sounds in the given language.

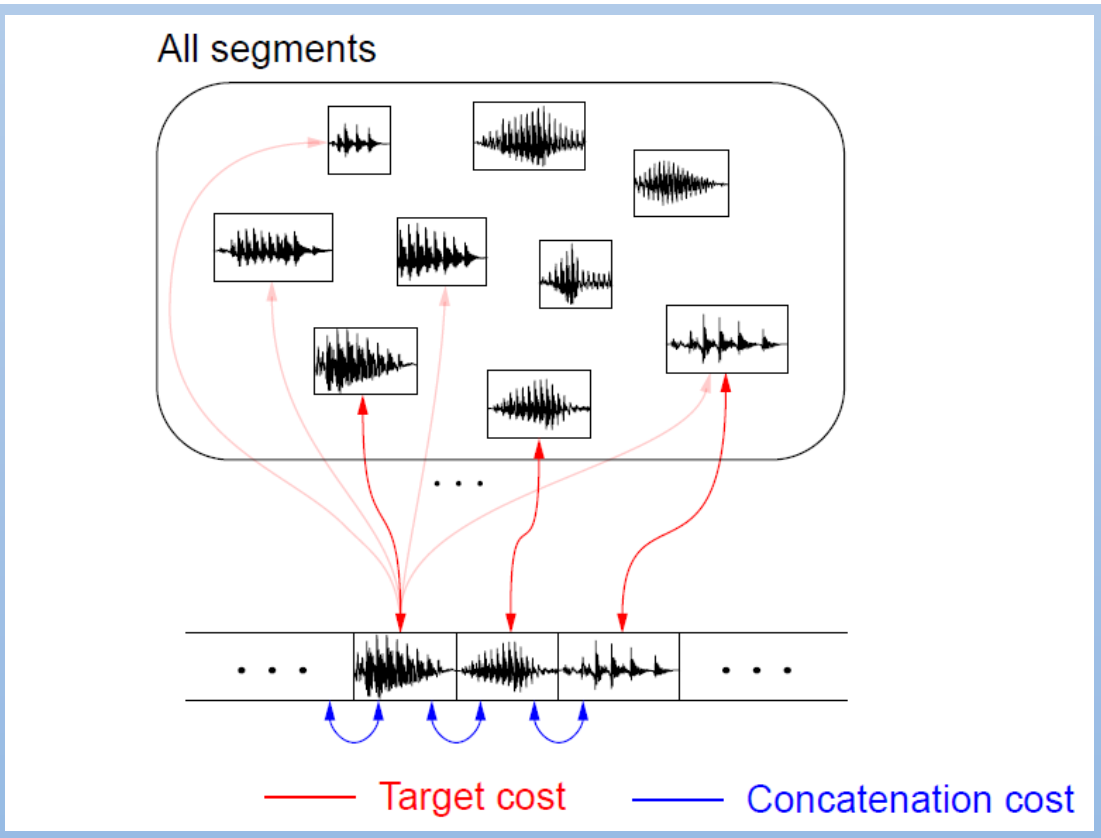


Figure 2.4: An overview of a general unit-selection scheme. Adapted from Zen *et al.* (2009).

Rousseau and Mashao (2005) developed a hybrid TTS synthesis system for Afrikaans using a combination of diphone unit selection synthesis and diphone concatenative synthesis. Their system was evaluated using subjective evaluation obtaining good results on pleasantness, understandability, naturalness, and overall impression. Tiomkin *et al*. (2011) developed a hybrid TTS synthesis system that is optimal, natural, and has smooth transitions between adjacent segments, by combining statistical and concatenative synthesis units. Kiflu and Beshah (2012) developed a concatenative unit selection synthesis system for Ethiopian language (Tigrinya). This is the first unit selection speech synthesis system ever developed for Tigrinya. Uddin *et al*. (2015) developed a phoneme-based TTS synthesis system for Bangla using a small corpus although distortion

was the issue for longer words. Authors concluded that their system is more natural, even though hearing tests were not conducted. Hearing tests so far are the most accurate measure of synthetic speech quality. Sharma *et al*. (2015) developed a bilingual TTS synthesis system for Assamese language using both unit selection and HMM-based synthesis. The system was evaluated using the subjective and objective methods. Sharma *et al*. (2015) used manual segmentation to improve the quality of a database that resulted in decreasing distortion of synthesised speech. Anil and Shirbahadurkar (2014) developed an expressive TTS synthesis system based on pitch modification and prosodic word detection. Authors reported that neutral speech can be converted into emotional speech by modifying pitch frequency. Emotional speech is a speech that contains emotional features such as excitement, sadness, happiness, and others. While neutral speech is the speech that does not contains emotions. Aoga *et al.* (2016) developed a unit selection speech synthesis for Yoruba language. The system was implemented using the MARY TTS system. The subjective evaluation tests resulted with MOS of 2.9 out of 5, which shows the system was acceptable, although objective tests were not conducted. Louw *et al.* (2005) developed the isiZulu speech synthesiser using 'Multisyn' unit selection synthesis.

Authors discussed the challenges encountered when developing the synthesiser and their solutions. The problems included selection of appropriate phone units, generation of reliable pronunciation, and developing a cost function that selects and joins appropriate phone units. Mhlana (2011) developed the isiXhosa TTS synthesis system to support e-services in marginalised rural areas of Eastern Cape Province of South Africa. The system was tested and obtained acceptable level of usability. Mohasi (2006) used a hybrid TTS synthesis system developed by Rousseau and Mashao (2005) as a baseline to create another advanced Sesotho language hybrid TTS synthesis system by applying intonation modelling techniques, duration, and fundamental frequency on the unit selection hybrid system. Mohasi (2006) conducted listening tests to assess speech quality and this resulted in improved naturalness and fluency.

2.7.2.4 Statistical parametric speech synthesis using HMMs

In concatenative synthesis, synthetic speech is generated by concatenating pre-recorded speech segments from the speech corpus. Its disadvantage is the requirement of large amounts of speech training corpora and the effort needed to calculate the concatenation cost. An alternative is to apply a statistical parametric synthesis method. The SPSS is a model-based approach and conforms to corpus-based synthesis. The SPSS might be described as generating the average of some sets of similarly sounding speech segments (Black *et al.*, 2007), (Zen *et al.*, 2009). This approach differs directly from unit selection synthesis to preserve natural speech. However, SPSS has dominated the speech synthesis research area over the last decade because they offer more benefits, including less memory requirement to store model parameters, flexibility to change voice characteristics (Yamagishi & Kobayashi , 2007), robustness (Yamagishi *et al.*, 2009), small footprint (Zen *et al.*, 2009), and multilingual support (Gibson *et al.*, 2010).

In a typical SPSS system, parametric representations of speech including spectral and excitation parameters from the speech corpus are extracted, then train them using a set of generative models (e.g. HMMs). A maximum likelihood criterion is normally used to predict the model parameters as

$$\hat{\lambda} = arg\ \max_{\lambda}\ \{p(y|x,\lambda)\}, \tag{2.7}$$

where λ is a set of acoustic models, y is a set of acoustic features, and x is a set of linguistic features corresponding to y. The pictorial representation of Equation (2.7) is shown in Figure 2.5.

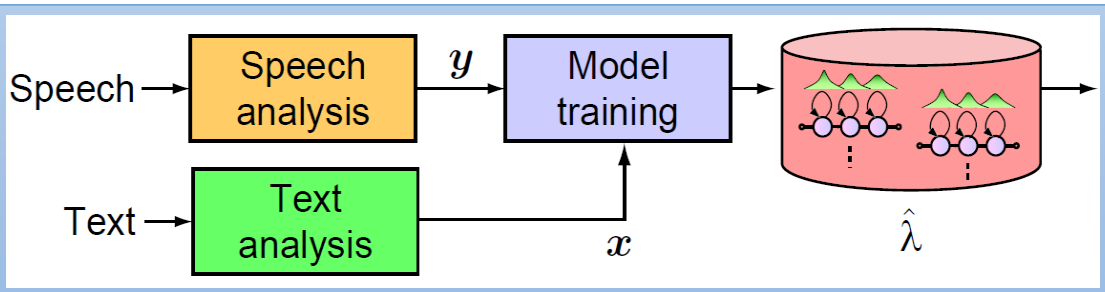


Figure 2.5: HTS training phase adapted from Zen *et al.* (2009).

The synthesis phase is shown in Figure 2.6. This phase extracts linguistic features x from text to be synthesised, generates most probable acoustic features y from set of predicted acoustic models $\hat{\lambda}$, to maximise their output probabilities as

$$\hat{y} = arg\ \max_{y} \{p(y|x,\hat{\lambda})\}. \tag{2.8}$$

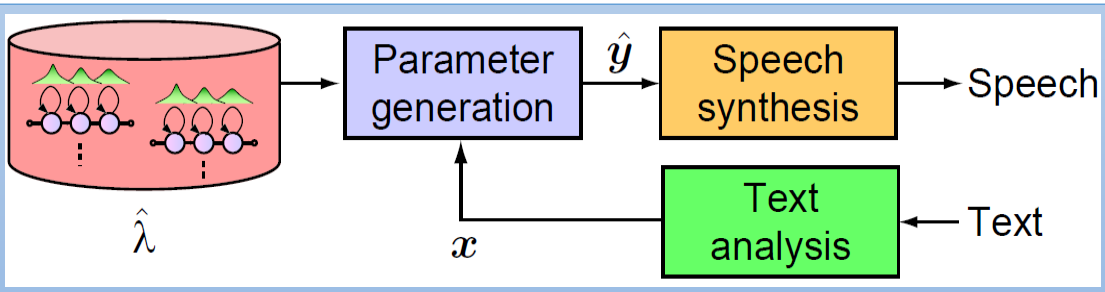


Figure 2.6: HTS speech generation adapted from Zen *et al.* (2009)

Finally, a speech output is built from parametric representation of speech. SPSS that uses HMMs is referred to as the HTS method (Yoshimura *et al.*, 1999). The HTS engine is a toolkit commonly used to develop HMM-based voices (Zen *et al.*, 2007).

Baloyi (2012) developed a TTS synthesis system using HMMs for Xitsonga. This is the first TTS synthesis system ever developed for Xitsonga at the University of Limpopo. The system was evaluated using listening tests and good results were

observed. Adiga and Prasanna (2014) developed a hybrid TTS synthesis by combining HTS and unit selection speech synthesis. The speech sound units were classified into *vowel-like regions* (e.g. vowel, semivowel, diphthong, nasal sounds) and *non-vowel regions* (e.g. stop consonants, fricatives, affricatives). The *vowel-like regions* were modelled from HMM framework and waveform units were chosen from HTS. The *non-vowel-like regions* which were not properly modelled by HMMs were picked from unit selection speech synthesis. The verification of *vowel and non-vowel like regions* were obtained by manual and automatic segmentation of speech signal. The hybrid system was compared to HTS and unit selection speech synthesis using both objective and subjective evaluations. The hybrid TTS synthesis system with manual segmentation outperformed HTS system but unit selection speech synthesis performed better than other systems. Mullah *et al*. (2015) developed a TTS synthesis system for Indian English using the HTS. Authors used MOS to test naturalness and intelligibility of the system. The system resulted with MOS of above 3 for both naturalness and intelligibility. A TTS synthesis system for isiXhosa was incorporated in development of a mobile platform for e-learning (Roux *et al.*, 2010). Stan *et al*. (2011) developed the HMM-based speech synthesis system for Romanian language using a different sampling rate (16, 32 and 48 kHz) to test the effectiveness of sampling rate on similarity, naturalness and intelligibility. Stan *et al*. (2011) observed that down sampling speech data to 16 kHz degrades similarity of speaker, however high sampling rate did not improve either naturalness or intelligibility of synthetic speech. More advanced NLP modules can try to increase performance of the TTS synthesis system.

The HMM-based synthesis systems suffer from speech quality caused by the inadequacy of acoustic modelling (e.g. trajectory HMM), limitations of the vocoder (e.g. speech transformation and representation using adaptive interpolation of weighted spectrum), and over-smoothing of parameter generation (e.g. global variance). The HMMs have been widely used in the development of speech synthesis systems. An alternative generative model can be applied to overcome these problems.

2.7.2.5 Statistical parametric speech synthesis using deep neural networks (DNN)

Recent advanced deep machine-learning algorithms can be used to address limitations of HMMs (Bengio, 2009). The decision trees in HTS that perform mapping from linguistic contexts extracted from text to probability densities of speech parameters are replaced by a DNN. Deep neural networks with multiple hidden layers can perform better than having only one hidden layer although learning such networks require high computational costs and becomes impractical. However, with recent development of improved hardware (e.g. a graphics processing unit) and software enabled us to train a DNN from large training data. The deep neural networks have achieved good results in machine-learning fields including pattern recognition, and automatic speech recognition (Hinton *et al.*, 2012). A DNN-based SPSS can be used to address the conventional approach (Zen *et al.*, 2013). A DNN was used to model the relationship between input text and their acoustic realization. The DNN-based approach showed potential to address the limitations in the conventional decision tree-clustered context-dependent HMM-based approach. Zen *et al*. (2013) applied subjective and objective evaluations to compare the performance of DNN-based synthesis with HMM-based synthesis. Objective evaluations showed the DNN-based approach obtaining better prediction of spectral and excitation parameters than the HMM-based approach. Moreover, the DNN-based approach achieved better preference over the HMM-based synthesis in the subjective listening test. In a recent study, Van den Oord *et al*. (2016) applied a DNN approach to generate raw audio from text. This approach outperformed the HMM-based unit selection concatenative speech synthesiser (Gonzalvo *et al.*, 2016) and the long short-term memory recurrent neural network-based statistical parametric speech synthesiser (Zen *et al.*, 2016) in naturalness. However, the HMM-based approach has reduced computational costs as an advantage over the DNN-based approach.

2.8 Evaluation of TTS Synthesis System

The quality of synthesised speech is evaluated using objective and subjective listening tests. Objective speech quality measures include, among others, linear predictive coding-based measure, time-domain and frequency-weighted signal-to-noise ratio measures, and composite measures (Hu & Loizou, 2008). Composite measures are obtained by combining other objective measures to create a new measure. More detailed objective measures are discussed by Hu and Loizou (2008). Zen *et al*. (2013) used 5-th Mel-Cepstral coefficients while Sharma *et al*. (2015) used first 13 Cepstral coefficients of cepstrum to conduct objective evaluation. Beněk (2014) developed a TTS synthesis system for the Czech language. Beněk only performed subjective evaluation and stated that there are no objective tests (Beněk, 2014, p. 30). This is impossible, since objective tests have been used before 2014 (Zen *et al.*, 2013), (Hu & Loizou, 2008). Theoretically, the quality of speech is best measured via a listening test where qualified evaluators listen to the synthesised speech and give quality opinions using a Likert scale. This method is also known as an MOS test. The quality of synthesised speech can be measured according to the following properties: naturalness, intelligibility, listening effort[1], flexibility, pleasantness, similarity, pronunciation, and other recent sophisticated measures. Naturalness is the degree to which the synthesised speech sounds close to natural speech. Intelligibility focuses on the ability for people to understand the synthesised speech. The listening effort is sometimes used as a factor of intelligibility. Flexibility focuses on how well the system handles out-of-vocabulary words and other non-standard words. Pleasantness focuses on the pleasure that one associates with listening to the synthesised voice. Similarity deals with how close the synthesised speech is compared to that of the original speaker. Pronunciation focuses on how well the synthesised speech pronounces words. In addition, prosody is the main factor of pronunciation.

There are several methods used to test the quality of synthetic speech such as:

[1] Popularly known as *understandability*

- **Preference test** (e.g. AB test) is used to compare two or more synthesised speeches. It is a common practice to compare natural speech with a synthesised speech.
- **Analysis of variance** can be derived to test factors of certain features between synthesised speeches.
- **MOS** is used to rate synthesised speech on a Likert scale. MOS is commonly used to measure naturalness, listening effort, pleasantness, similarity, pronunciation, and flexibility (Viswanathan & Viswanathan, 2005).
- **Diagnostic rhyme test (DRT)** tests intelligibility of initial consonants based on 96 pairs of confusable rhyming words (e.g. pond/bond or tense/dense) (Greenspan *et al.*, 1998). Evaluators listen to one word and choose the correct one from the pair. The percentage of correct identifications is used an intelligibility measure.
- **Modified rhyme test (MRT)** is commonly used to test intelligibility of synthesised speech (House *et al.*, 1965). This method focuses on either initial or final consonants (e.g. went, dent, rent, bent, sent, tent). A list of 300 words contains 50 sets of 6 words. Evaluators must identify a single word from a closed list of six words. The percentage of correct identifications is used as an intelligibility measure.
- **Semantically unpredictable sentences (SUS)** can be used to test speech quality at sentence level (Benoît *et al.*, 1996).
- **Word error rate (WER) and sentence error rate (SER)** are commonly used in speech recognition; however, these methods are currently employed to measure intelligibility of synthesised speech.

2.9 TTS Synthesis Application Areas

Speech synthesis system makes it possible for people to use computational devices such as smartphones to access information, use email systems or even do voice dialling in their first language. In today's electronic digital age, the use

of TTS synthesis technologies has increased exponentially in mobile smartphones, computers, internet-based services, banking, and dialogue systems[1]. Currently, access to computer-based information is an important demand for social and technological evolution, and the use of human language technology has increasingly become an essential technological evolution for linguistic resources. In order to ensure that people have enough access to information or linguistic resources given in their first language, these resources should be electronically digitalised. For example, dictionaries are available in digitalised audio format for different languages. Google voice search, Siri[2] and other voice-enabled speech applications are good examples of TTS synthesis systems in mobile devices. Modern navigation systems use back-end TTS synthesis speech navigation for faster guidance (Jeon *et al.*, 2015). These applications are embedded in vehicles, aeroplanes, and mobile devices (Ramani *et al.*, 2013).

These technologies enable humans to interact and communicate with machines, and deliver valuable and useful e-services ranging from sciences, health, economics, and education. Speech technology applications play an important role in teaching and language learning. An increase in the development of such systems enhances learning using computer-assisted language learning and computer-assisted pronunciation training (CAPT). Developing CAPT systems for pronunciation learning and teaching requires extensive linguistic resources and experts. Chen and Li (2016) reviewed approaches and challenges used in CAPT development. Eskenazi (1999) discusses a good analysis of using ASR for training students to learn new languages. Yu and Wang (2016) proposed a pronunciation visualisation instruction system based on an articulatory mesh model. Their system was tested on students learning Chinese in second language and achieved accuracy of 97.6% (after learning) from 68.4% (before learning). Speech synthesis applications simplify language and pronunciation

[1] Available at: http://www.acapela-group.com/voices/demo/
[2] Available at: http://www.apple.com/ios/siri/

learning; such applications can be applied not only in language learning but extended to subject-specific domains of learning.

2.10 Speech Synthesis Systems Toolkits

2.10.1 Festival TTS

The Festival TTS synthesis system is one of the well-known popular multilingual TTS synthesis systems used for creating new synthetic voices in a limited domain or an open vocabulary domain (Taylor *et al.*, 1998). The Festival TTS system is open source software allowing personal and commercial usage. The system uses FestVox for building synthetic voices (Black & Lenzo, 2014). Festival TTS system supports waveform generation using diphone-based unit selection and HTS approach. This system has a large memory footprint and is relatively slow. Hence, a small, fast runtime TTS engine called Flite was developed, aiming to provide improvements with regards to speech, code size, data size, thread safety, and portability of maintenance (Black & Lenzo, 2001).

2.10.2 Speect TTS

Speech synthesis with extensible architecture (Speect)[1] is a multilingual TTS synthesis system that offers various APIs (Louw, 2008). Speect has a capability of creating new TTS synthesis voices. It offers python bindings for customisation and implementation of advanced ideas. This program is under active development at the CSIR[2].

2.10.3 IBM Watson TTS

The IBM Watson[3] TTS service is a cloud-based service that provides API to synthesise text into speech in a variety of *languages*, *accents*, and *voices*. The IBM Watson TTS supports speech synthesis markup language (SSML) for translation of text into International Phonetic Alphabet or IBM Symbolic Phonetic

[1] Available at: http://speect.sourceforge.net
[2] Available at: http://www.csir.co.za/meraka/
[3] Available at: http://www.ibm.com/watson/developercloud/text-to-speech/api/v1/

Representation. The IBM Watson TTS API consists of (a) synthetic voices, (b) methods to synthesise text or SSML over internet, (c) pronunciation (a method that shows pronunciation of a specified word), (d) custom model (that provides methods for creation of custom voice models), and (e) custom words (that provide methods that allow clients to manage word or translation pairs in a custom voice model). Furthermore, IBM Watson consists of the following features:

- Interact – allow creation of dialog systems.
- Learn – use machine-learning to grow the subject matter expertise in applications.
- Reason – provides customised recommendations by understanding client's emotion, tone, and personality.
- Understand – interpret data including unstructured texts, videos, images and audios.

2.10.4 Merlin

Merlin is the latest open source toolkit to offer DNN-based speech synthesis (Wu *et al.*, 2016). Merlin toolkit provides the acoustic modelling functions, including acoustic and linguistic feature normalisation, linguistic feature vectorisation, neural network acoustic model training, and generation. Merlin is written in Python and is not a complete TTS. Hence, it requires an external front-end system such as Festival TTS. Merlin has been used in recent research work (Valentini-Botinhao *et al.*, 2015) (Wu *et al.*, 2015) (Watts *et al.*, 2016).

2.10.5 MARY TTS

The MARY TTS synthesis system is a tool for research development and teaching in the domain of TTS synthesis (Schröder & Trouvain, 2003). The MARY TTS system is an open source voice builder software designed to run on Java (Schröder *et al.*, 2011). New languages and synthetic voices can be created and added to MARY TTS system (Pammi *et al.*, 2010). Synthetic voices can be created using unit selection synthesis or the HMM-based speech synthesis approach. The program uses SAMPA format for transcription of a new language

module (Wells, 2005). The MARY TTS supports generation of phonemes from text, generation of TEXTGRID file, generation of emotional speech via emotion markup language (EmotionML), and prosody generation from MARY XML format (Schröder & Breuer, 2004). EmotionML is commonly used for (a) manual annotations of data, (b) generation of emotion-related system behaviour, (c) and automatic recognition of emotion-related states from user behaviour. The program documentation can be accessed from the MARY TTS GitHub web page[1]. For this research work, the MARY TTS is used to develop the synthetic voices.

2.11 Summary

The study of text-based LID is a well-known topic and many approaches have been presented. Text-based LID can be approached from a pattern recognition viewpoint by examining statistical attributes in text as feature measures. MNB is the simple and effective method using *n*-gram statistics as features. Statistical measures can depend on keywords, frequent words, or special letters found in a document. The accuracy can be improved by increasing the feature dimensionality. Various LID techniques have been discussed, including the popular baseline MNB and robust SVMs. MNB can perform classification on string data; however, SVMs require data transformation before classification. We have discussed LID studies for South African languages and realised that research for these languages is still open. Some of the toolkits used for text classification have been discussed.

The NLP and DSP components of the TTS synthesis system have been discussed. Various speech synthesis methods have been discussed and we realised that the HTS approach is better than the DNN approach on computation cost. The DNN approach requires massive amounts of memory when training, with many hidden layers. The disadvantage of the unit selection concatenative synthesis is the requirement of a large database and this negatively affects the under-resourced languages. The advantage of the HTS approach is the ability for

[1] Available at: https://github.com/marytts/marytts/wiki

development of new synthetic voices on a limited domain under normal computational cost. We discussed evaluation methods of a TTS synthesis system. The advantage of the SUS evaluation method is that there are no semantic contextual cues to the intelligibility of the individual words. Several measures of speech quality, including, among others, naturalness, and intelligibility, were discussed. The application areas of TTS systems were detailed. Several programs that are currently used to develop new advanced TTS voices were also discussed. The next chapter gives the design and implementation of the proposed system.

3 CHAPTER 3: DESIGN AND IMPLEMENTATION

In this chapter, we describe the datasets and the procedure for the implementation of the pronunciation assistant, which is a combination of a front-end LID and back-end TTS synthesis system. Two supervised machine-learning methods are implemented to perform a three-way multiclass classification, and to explore the LID accuracy that can be achieved for the following under-resourced official languages of South Africa, namely Sepedi, Xitsonga, isiNdebele and Tshivenda, using *n*-gram statistics as features. The development toolkits deployed to implement back-end and front-end are both Java based. Hence, WEKA toolkit has been used to build the LID model and MARY TTS synthesis system has been used to build new TTS voices using HMM method.

3.1 Introduction

South Africa is a multilingual country with 11 official languages. The pronunciation of words, mostly proper names, is difficult for non-native speakers, since most proper names are written in native languages. However, knowing the native language of the proper name may lead to the reduction of such pronunciation difficulties. In our experiment, we aim to predict the first language associated with an input surname for South African under-resourced official languages, namely, Sepedi, Xitsonga Tshivenda and isiNdebele. To achieve this aim, we acquired training textual data comprising of surnames usually associated with these languages. We compared the machine-learning algorithms to select the best one for building a text-based LID predictor for surnames classification. The LID front-end component predictor is used to classify an input surname by predicting the first language associated with that surname as shown in Figure 3.1. Once the language is predicted, the TTS phase continues with the pronunciation rendition of that surname using the predicted language. The LID systems discussed in Chapter 2 (Literature review) are not available for testing on our training text data; hence, no comparisons could be examined.

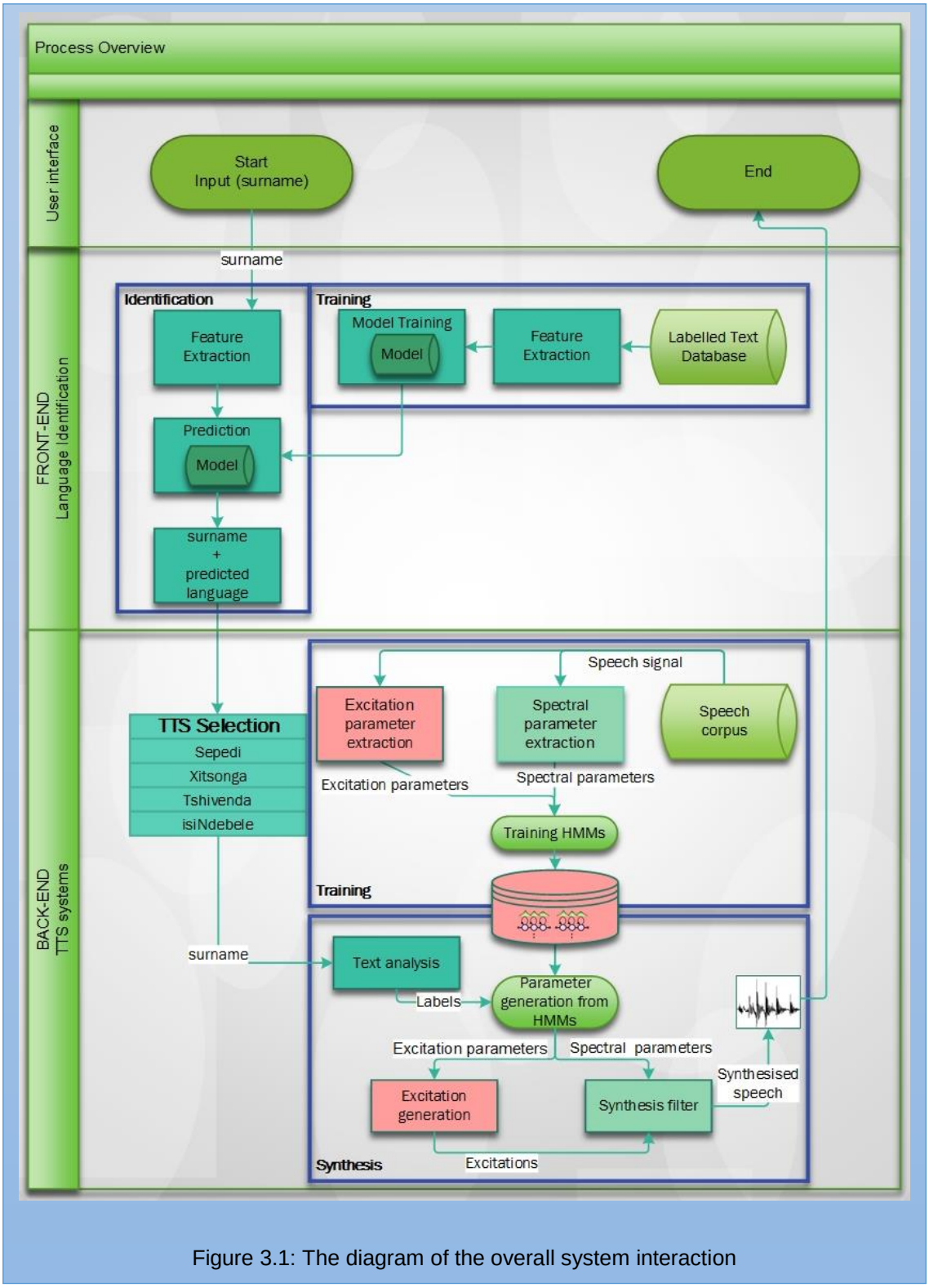


Figure 3.1: The diagram of the overall system interaction

The layout of this chapter is as follows:

- Section 3.2 discusses the LID dataset, features, deployed machine-learning algorithms, and implementation of the classifier.
- Section 3.3 discusses the speech dataset, toolkits, procedure followed for development and implementation of new languages and HMM voices.
- Section 3.4 explains the integration of both TTS and LID interfaces.
- Section 3.5 details the deployment of the system to the “real-life” production server.

3.2 Front-end Phase: Language Identification Module

This section details the process, tools and methods used in developing the LID for under-resourced languages. Rule-based and statistical methods are used for text-based LID implementation. The rule-based approach requires linguistic experts and a large amount of time is also required. Hence, we selected to use the statistical machine-leaning method due to its flexibility, accuracy, and robustness.

3.2.1 Data Acquisition Pre-processing

The training textual data for LID front-end has been acquired from the department of ICT at the University of Limpopo. The training data consists of both surnames and their corresponding first languages in Sepedi, Tshivenda, isiNdebele and Xitsonga. The Sepedi text data is the highest, with 1100 surnames, followed by Tshivenda with 1035 and Xitsonga with 908. IsiNdebele text data is less than 213, which is not enough because it will cause biased results and misperformance of the LID predictor; hence, it is excluded in the LID front-end development (See Figure 3.2). The complete dataset contains 3043 surnames excluding isiNdebele.

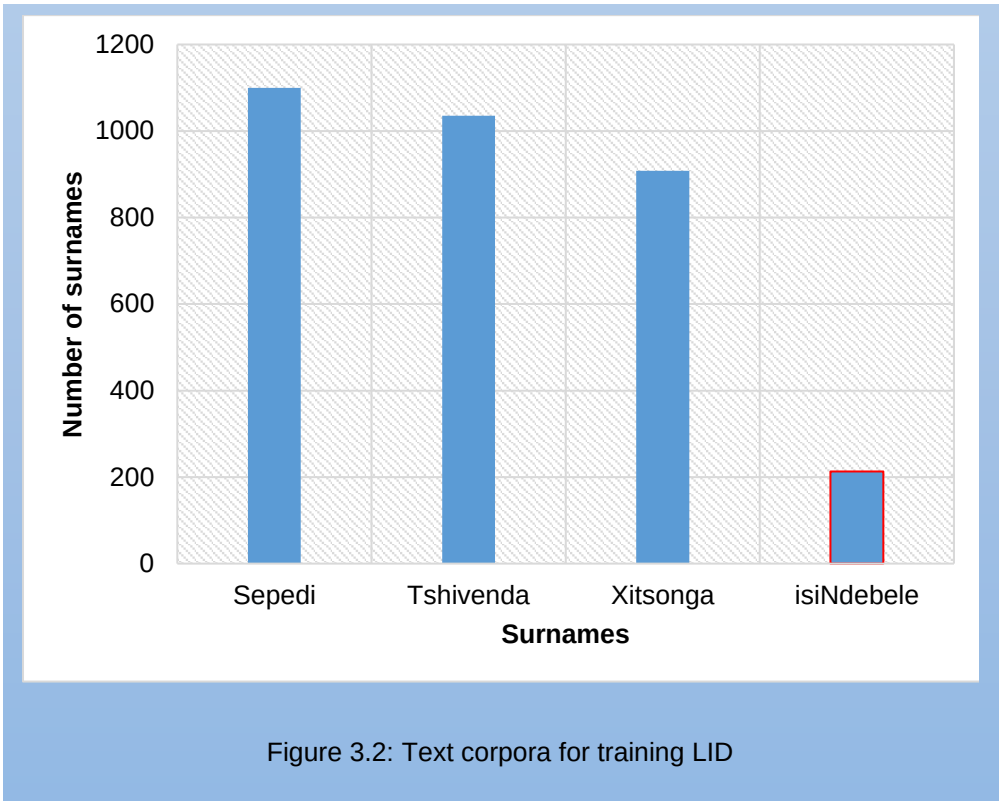


Figure 3.2: Text corpora for training LID

The acquired training data was in Microsoft excel worksheet format and needed to be converted to WEKA format. WEKA supports attribute-relation file format (ARFF) and comma-separated values (CSV). The data was converted into CSV using Microsoft excel, and then converted into attribute-relation file format (ARFF) using WEKA toolkit. The easy way to introduce the datasets in WEKA is by using the ARFF files. An example of an ARFF file is shown in Listing 3.1. The first section of the ARFF file is the header information that contains *relation* declaration and *attribute* declarations including the name of the relation, a list of the attributes (the columns in the data section), and their datatypes. The datatype can be –

- numeric (integer or real numbers),
- nominal,
- string,
- date,
- and relational (for multi-instance data).

```
@relation LanguageIdentification

@attribute surname string
@attribute class {nso, tso, ven}


@data
"MTHEBULE",tso
"MKHONTO",tso
"MAKARINGE",tso
"MASINA",tso
"SHIVAMBU",tso
"NEDZAMBA",ven
"NEVHUFUMBA",ven
"NELUVHALANI",ven
"TSHIVHANGANI",ven
"PHANDAVHUDZI",ven
"MOGANO",nso
"THABA",nso
"RASEEMELA",nso
"KGANYAGO",nso
"LETEBELE",nso
...
```

Listing 3.1: Extract of the WEKA ARFF used for creating the LID module

The *string* attribute allows for the creation of datasets containing any arbitrary textual values. This is important in data mining systems, as *string* attributes datasets can be created and then used with filters in WEKA to manipulate and convert *string* datasets to *numeric* or *nominal* sets or any desired target sets. *Nominal* attributes are defined by specifying a list of possible values they can take. The second section contains the data declaration line and the actual instance lines. The *@data* declaration is a single line denoting the start of the data segment in the file. Each instance is represented on a single line, with carriage returns denoting the end of the instance and attribute values are separated by commas. Attribute values must appear in the order in which they are declared in the header section. An unknown or missing value is represented by a question mark and this feature is used for the prediction of a surname (or class) in the online demo (see project website (Sefara, 2017)).

Listing 3.1 shows an extract of the dataset file used in training the LID module. This file consists of the *relation* (dataset), two attributes *surname* and *class*, and

instance *data values*. *Surname* is a *string* attribute that stores the actual instances values (surnames), while c*lass* is a *nominal* attribute that stores the list of languages. All the surnames are labelled or tagged with their first languages in the data section. This approach is called supervised learning wherein labelled data is used in training a classifier model. The languages are represented by their language codes (or locales). Table 3.1 shows language codes for representing Sepedi (nso), Tshivenda (ven), Xitsonga (tso) and isiNdebele (nbl) under the International Organization for Standardisation ISO 639-2:2008[1].

Table 3.1: Language Locales

Code	Name
nso	Sepedi
nbl	isiNdebele
ven	Tshivenda
tso	Xitsonga

3.2.2 N-gram Feature set

Text-based LID problem can be tackled from a linguistic or statistical approach. The linguistic approach would be the favourable choice where high classification accuracies are expected although a large amount of linguistic expertise and resources are required to code a language. However, for under-resourced languages, such resources are not readily available therefore a statistical approach becomes a viable alternative. Statistical language models can be built from sequence of letters, words or *n*-grams. The character *n*-gram based models are suitable for identification of individual and unique words. This is also the most popular choice in the literature reviewed and we have restricted our feature sets

[1] Available at: https://www.loc.gov/standards/iso639-2/php/code_changes.php

to character *n*-gram based features (Cavnar & Trenkle, 1994). The size of *n* can increase the accuracy of the classifier; however, the accuracy decreases beyond a certain level of *n*. Furthermore, much computation and memory usage is needed for higher value of *n.* In most experiments, trigrams yield satisfactory results (Fourie *et al.*, 2014). Hence, we have restricted our concentration to the cases of unigrams up to five grams.

Most classifiers, including SVMs, cannot handle s*tring* attributes, hence the acquired training data needed to be processed using appropriate filters. The WEKA toolkit contains supervised and unsupervised filters. The *string-to-word-vector* filter is an unsupervised filter that converts string attributes into a set of attributes representing word occurrence information from the text contained in the strings. This filter supports word, word *n*-gram and character n-gram tokenisation. Since our training data consists of single words, character n-gram is favourably used to tokenise all the surnames by generating character *n*-grams of size one to size five and then converting them to feature vectors.

3.2.3 Machine-learning Algorithms

Machine-learning algorithms can deliver optimal performance for prediction of a class or category when applied on a high-performing computer. The project is set up on a desktop computer with 2Gig of RAM and 2.94 GHz Intel (R) Core™ 2 Duo CPU. This research project used a text classification method as LID based on *n*-gram features. There are many machine-learning algorithms applied in pattern recognition, big data analytics, statistical analysis and text mining (Botha & Barnard, 2008). However, SVM and multinomial naive Bayes (MNB) have shown better performance on text identification (Fourie *et al.*, 2014). The supervised machine-learning models use associated learning algorithms that recognise patterns and analyse data for regression analysis and classification. Our experiments employed the MNB as our baseline classifier and the SVM libraries for multiclass classification under different SVM kernels for K-fold cross-validation in WEKA.

3.2.3.1 K-fold cross validation

Cross-validation is a technique to evaluate predictive models by partitioning the original data into testing and training set. The cross-validation method consists of K-fold cross-validation, random subsampling, and leave-one-out validation. The K-fold cross-validation is used due to its properties of being easy, simple, and using all data for training and testing. Moreover, K-fold cross-validation assists in selecting the best model and its parameters to create the final model. We performed K-fold cross-validation on our training set where K was set to 10 to (a) balance the reliable estimates and computational costs and to (b) avoid biased results.

As illustrated in Figure 3.3, the 10-fold cross-validation approach partitions the corpus into 10 equal parts and performs training on 9 parts leaving one part for testing. The cross-validation approach is repeated 10 times so that each partition is used for training.

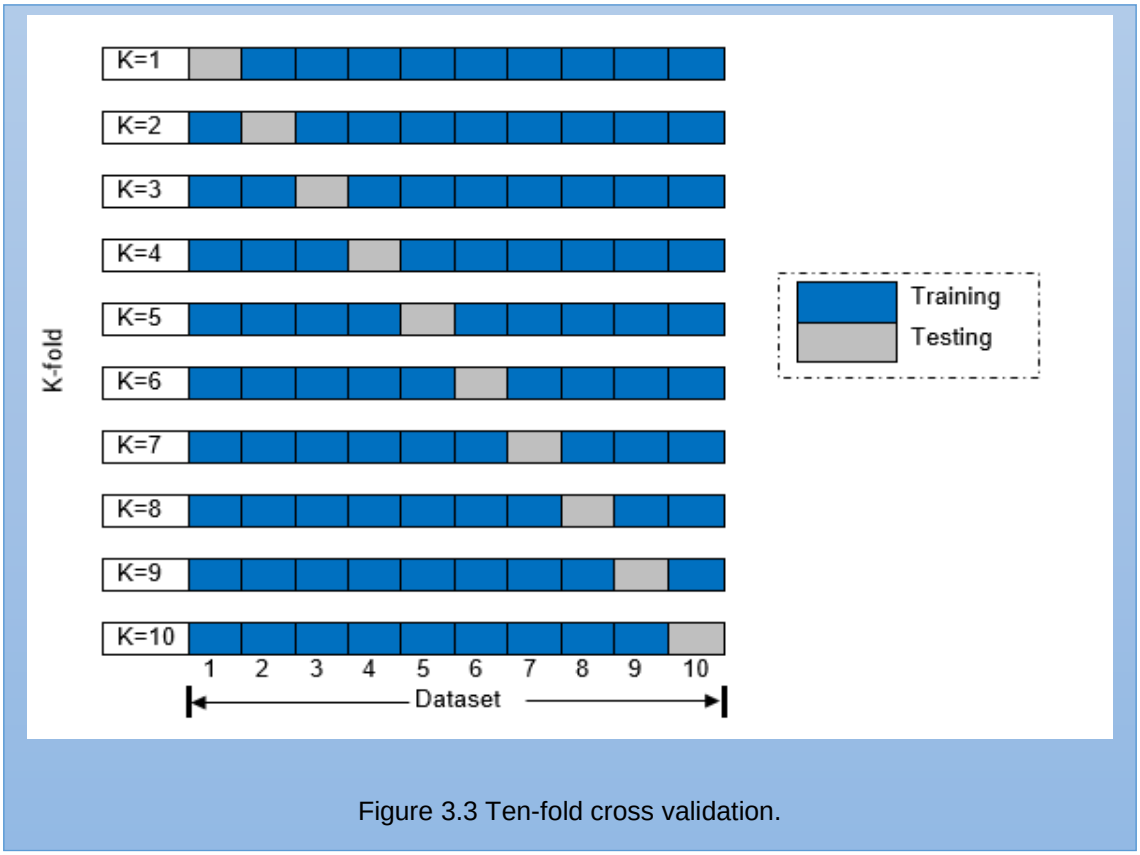


Figure 3.3 Ten-fold cross validation.

3.2.3.2 Multinomial naive Bayes

Naive Bayes classifier is a simple probability classifier based on Bayes theorem (see Equation (3.1)) with independence assumptions. There are several variations of naive Bayes, viz. MNB, Bernoulli naive Bayes and Binarised MNB. The MNB is one of the classifiers that are often selected as baseline classifiers in language detection tasks since they are less computationally intensive in both memory and processor consumption (Giwa & Davel, 2013). Moreover, it performs well under small or limited data conditions and takes shorter training time compared to SVMs. During classification, the MNB learning is employed using frequency estimate that determines parameters by computing the appropriate frequencies from the data. A classifier based on MNB is implemented using an open-source WEKA toolkit that contains many classification algorithms. More details in the experimental setup are found in the next sections.

3.2.3.3 Support vector machines

Several classifiers based on SVMs are implemented using a freely customised downloadable library for SVM called LibSVM on WEKA toolkit. The LibSVM is a function on WEKA that allows the user to create SVM-based classifiers under different kernels. The SVM uses kernels to allow non-separable data in the higher dimensional feature space. The goal of an SVM classifier is to find an optimal separating hyperplane, which maximises the margin of the training data. There are various methods that can be used to solve SVM classification problem with more than three classes (Hsu & Lin, 2002):

- **Directed acyclic graph SVM** is a novel algorithm for multi-class classification.
- **One-against-one** or **pairwise classification**, where one binary SVM is created for each pair of classes to separate vectors of one class from vectors of the other class.
- **One-against-all classification**, where there is one binary SVM for each class to separate vectors of one class from vectors of other classes.

LibSVM uses one-against-one classification method where a total of three classifiers are constructed and classification is performed using a voting strategy. The WEKA toolkit contains classifiers called filtered classifiers that can run an arbitrary classifier on data that has been passed through a filter. In this experimentation, filtered classifiers are used to run a classifier on filtered data. This approach is shown in Figure 3.4.

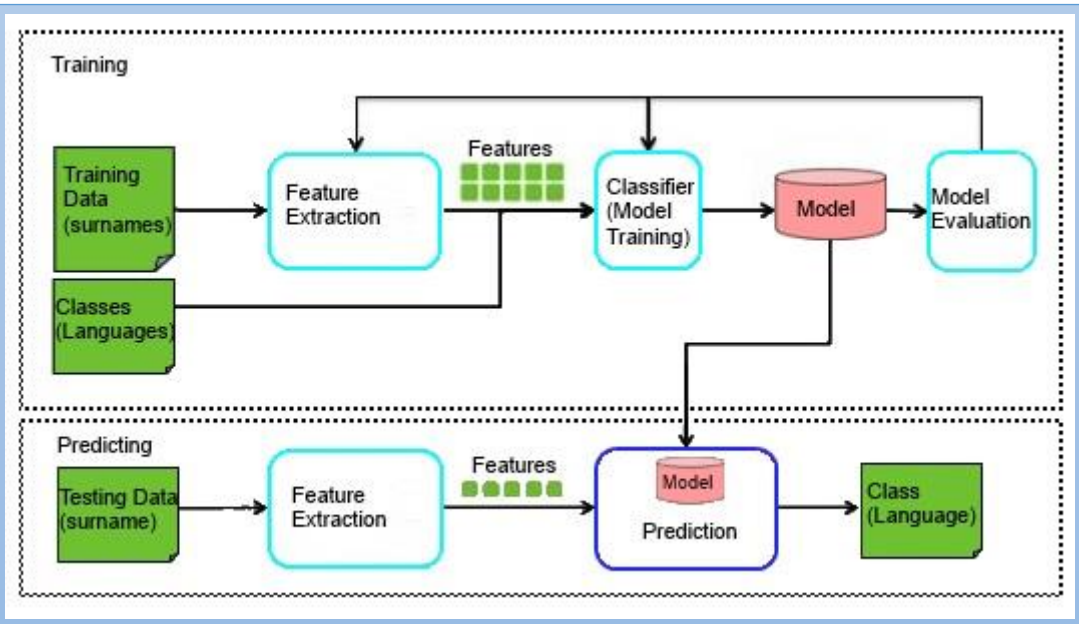


Figure 3.4: Proposed supervised learning workflow

3.2.3.4 Experiment setup

The experiments are focused on the SVM and MNB machine-learning methods. The LibSVM contains various SVM kernels hence same dataset is applied for the various kernels to examine their performance. In total, five experimental setups are conducted.

a) **Experiment 1**: This experiment employed the baseline MNB classifier on the original dataset. The MNB classifies text given a set of classes $C = \{c_1, c_{2\ldots}c_k\}$ and a set of unique words $W = \{w_1, w_{2\ldots}w_n\}$ and $N = \{1,2, \ldots, n\}$ defines the size of the vocabulary where $n \geq i \geq 1$ and $k > 1$. Then MNB assigns a test

document d_i to a class that has the highest probability $\mathrm{P}(c|d_i)$, which is given by the Bayes' rule:

$$\mathrm{P}(c|d_i) = \frac{\mathrm{P}(c)\mathrm{P}(d_i|c)}{\mathrm{P}(d_i)} \quad (3.1)$$

The following receipt is used to train the model.

```
weka.classifiers.bayes.NaiveBayesMultinomialText -P 0 -M 2.0 -
norm 1.0 -lnorm 2.0 -stopwords-handler weka.core.stopwords.Null
-tokenizer "weka.core.tokenizers.CharacterNGramTokenizer -max 3
-min 1" -stemmer weka.core.stemmers.NullStemmer
```

where *min* and *max* are ngram ranges.

b) **Experiment 2**: This experiment employed a LibSVM classifier using RBF kernel on the original dataset. The RBF kernel is given by the formula,

$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma > 0, \quad (3.2)$$

where γ is a kernel parameter.

The following receipt is used to train the model.

```
weka.classifiers.meta.FilteredClassifier -F
"weka.filters.unsupervised.attribute.StringToWordVector -R 1 -W
1000 -prune-rate -1.0 -N 0 -stemmer
weka.core.stemmers.NullStemmer -stopwords-handler
weka.core.stopwords.Null -M 1 -tokenizer
\"weka.core.tokenizers.CharacterNGramTokenizer -max 5 -min 4\""
-W weka.classifiers.functions.LibSVM -- -S 1 -K 2 -D 3 -G 0.0 -R
0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -Z -W "1.0 1.0 1.0"
```

where *min* and *max* are ngram ranges and K is the kernel.

c) **Experiment 3**: This experiment employed a LibSVM classifier using sigmoid kernel on the original dataset. The sigmoid kernel is given by the formula,

$$K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r), \quad (3.3)$$

where γ and r are kernel parameters.

The following receipt is used to train the model.

```
weka.classifiers.meta.FilteredClassifier -F
"weka.filters.unsupervised.attribute.StringToWordVector -R 1 -W
1000 -prune-rate -1.0 -N 0 -stemmer
weka.core.stemmers.NullStemmer -stopwords-handler
weka.core.stopwords.Null -M 1 -tokenizer
\"weka.core.tokenizers.CharacterNGramTokenizer -max 5 -min 4\""
-W weka.classifiers.functions.LibSVM -- -S 1 -K 3 -D 3 -G 0.0 -R
-0.95 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -Z -W "1.0 1.0 1.0"
```

where *min* and *max* are ngram ranges and K is the kernel.

d) **Experiment 4**: This experiment employed a LibSVM classifier using polynomial kernel on the original dataset. The polynomial kernel is given by the formula,

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0, \quad (3.4)$$

where $\gamma, r,$ and d are kernel parameters.

The following receipt is used to train the model.

```
weka.classifiers.meta.FilteredClassifier -F
"weka.filters.unsupervised.attribute.StringToWordVector -R
first-last -W 1000 -prune-rate -1.0 -N 0 -L -stemmer
weka.core.stemmers.NullStemmer -stopwords-handler
weka.core.stopwords.Null -M 1 -tokenizer
```

```
\"weka.core.tokenizers.CharacterNGramTokenizer -max 5 -min 4\""
-W weka.classifiers.functions.LibSVM -- -S 1 -K 1 -D 1 -G 0.0 -R
0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1
```

where *min* and *max* are ngram ranges and K is the kernel.

e) **Experiment 5**: This experiment employed a LibSVM classifier using linear kernel on original dataset. The linear kernel is given by the formula:

$$K(x_i, x_j) = x_i^T x_j. \tag{3.5}$$

The following receipt is used to train the model.

```
weka.classifiers.meta.FilteredClassifier -F
"weka.filters.unsupervised.attribute.StringToWordVector -R 1 -W
1000 -prune-rate -1.0 -N 0 -L -stemmer
weka.core.stemmers.NullStemmer -stopwords-handler
weka.core.stopwords.Null -M 1 -tokenizer
\"weka.core.tokenizers.CharacterNGramTokenizer -max 5 -min 5\""
-W weka.classifiers.functions.LibSVM -- -S 1 -K 0 -D 3 -G 0.0 -R
0.0 -N 0.6 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -Z -W "1.0 1.0 1.0"
```

where *min* and *max* are ngram ranges and K is the kernel.

All experiments are conducted using machine learning algorithms (MNB and SVM) obtained from WEKA toolkit. Five models are built and saved to a local file for future predictions. Each model is evaluated on 10-fold cross-validation. The presentation and analyses of the output from these experiments are discussed in the next chapter.

3.3 Back-end Phase: Speech Synthesis Module

There are different methods for producing synthetic speech from any given input text that includes concatenative synthesis, articulatory synthesis, formant synthesis, and HMM-based synthesis. This study focuses on synthetic voices created from the preferred HMM-based synthesis system because of its following advantages (Zen *et al.*, 2009):

- Language independent architecture
- Small footprint
- Rapid prototyping of new voices
- Flexible synthesis parameters
- Fast and portable

This section covers the process undertaken to develop TTS synthesis system for the targeted Sepedi, isiNdebele, Tshivenda and Xitsonga languages using HTS. It details the acquired datasets used and explains the files and data used for training the system. It also explains the required software development and implementation phase.

3.3.1 Datasets

Two important requirements for building a new voice are waveform files and corresponding transcription text files. The acquired training speech data shown in Figure 3.5 is acquired from the Lwazi project. Four recruited and volunteering mother tongue or first language speakers were engaged for the recording of training data. About 1318 sentences in Sepedi are used to train Sepedi TTS voice and 1000 sentences in Tshivenda are used to develop Tshivenda TTS voice, while 994 sentences are used to train isiNdebele TTS voice and 910 sentences are used to train Xitsonga TTS voice (see Figure 3.5). Both Sepedi and Tshivenda audio waveform files are in female voice while both isiNdebele and Xitsonga audio waveform files are in male voice. The recording durations are also highlighted in Figure 3.6 with Xitsonga having a shorter duration of 1.28 hours and Sepedi having the longer duration of 2.23 hours.

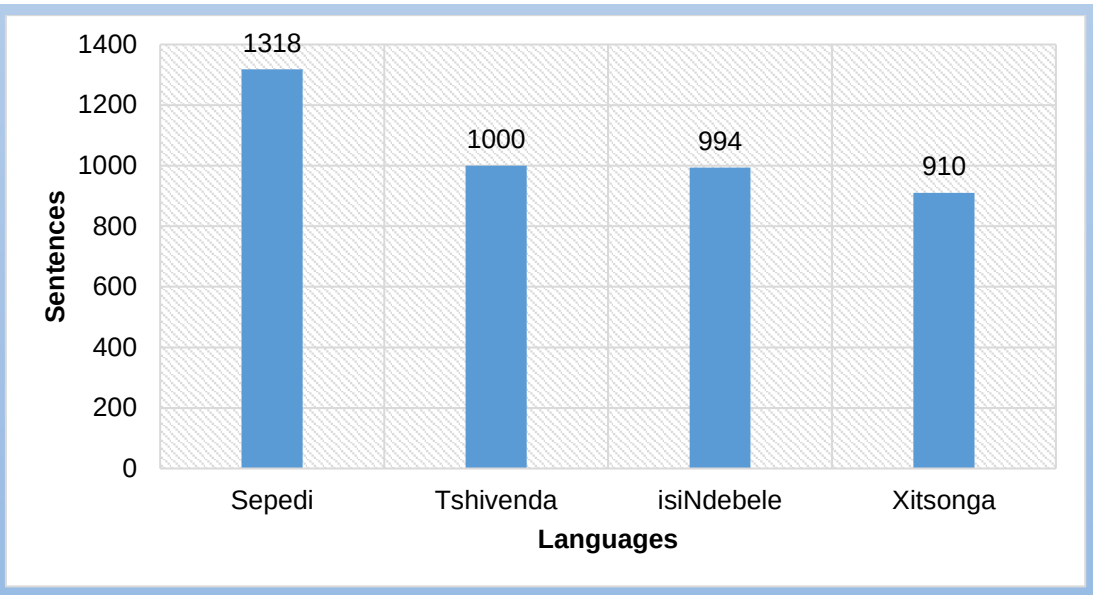


Figure 3.5: Number of sentences

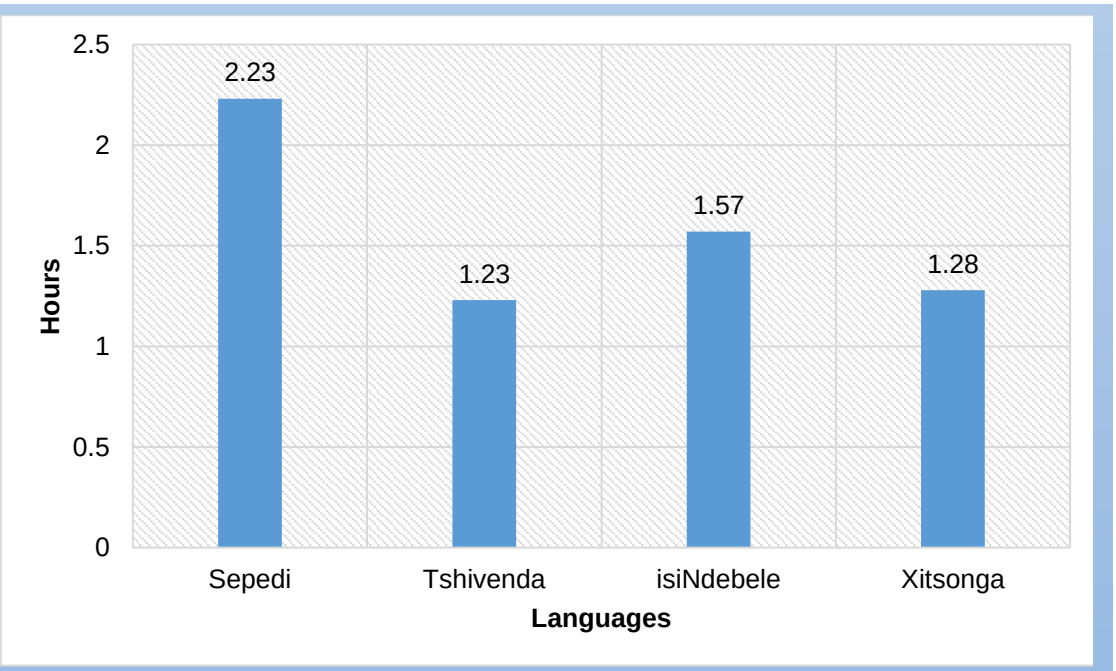


Figure 3.6: Speech corpora duration comparison

The corpus size for Sepedi is the highest compared to other languages with 274.6 MB. The larger size of the corpus helps with naturalness of the synthetic speech; however, the training duration is much longer (Zen *et al.*, 2009). Second highest

corpus size is isiNdebele with 224.4 MB, followed by Xitsonga with 169.2MB. Tshivenda had the smallest corpus size of 160.5 MB. The isiNdebele language has the smallest number of sentences compared to the Tshivenda language, but its corpus duration is higher than that of the Tshivenda language. This is attributable to the fact that sentences in the isiNdebele corpus are longer (see Figure 3.7).

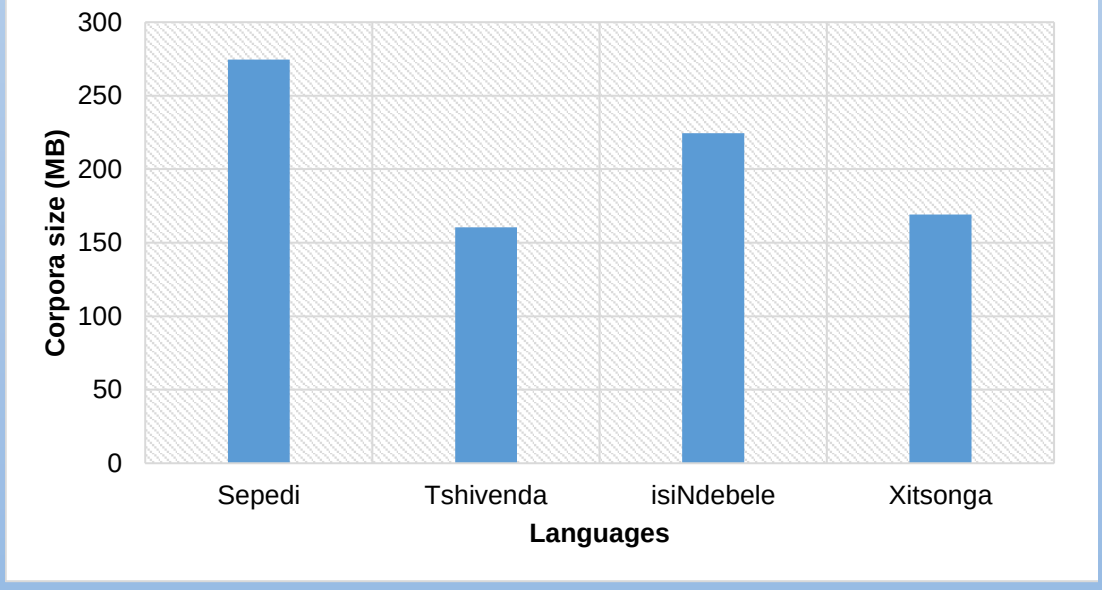


Figure 3.7: Corpora size

3.3.2 Compiling MARY TTS Builder Tools

Additionally, some software packages had to be installed to create a working environment before experiments are conducted. The following items comprise a list of required software to execute MARY TTS synthesis system on a 32-bits Ubuntu 14.04 Long-term support: build-essentials, git, mc, libc6-dev, libx11-dev, libncurses5-dev, Sox, tcl-snack, g++, and python3-dev.

The Git tool is installed to clone MARY TTS SNAPSHOT 5.2 from GitHub[1] to the present research project environment. The MARY TTS software requires installation of additional speech software packages to the current working

[1] Available at: https://github.com/marytts/marytts

environment. The following software packages are installed onto the Ubuntu Linux platform before the MARY TTS builder is compiled:

- **Open Java development kit 8**: is used to run MARY TTS.
- **Apache-maven-3.3.9**: is used to compile MARY TTS.
- **HTK-3.4.1.tar.gz** and **HDecode-3.4.1.tar.gz** (HTK website, 2009): is a toolkit for research in automatic speech recognition (ASR). Speech generation depends on ASR, hence ASR tools are required.
- **HTS-2.2_for_HTK-3.4.1.patch** (Tokuda *et al.*, 2016): used to modify HTK-3.4.1 to form HMM-based Speech Synthesis System (HTS) that is used to train HMM voices.
- **Hts_engine_API-1.05** (Tokuda *et al.*, 2011): is used to synthesize speech waveform from HMMs trained by the HTS.
- **Edinburgh_Speech_tools-2.4-release** (King *et al.*, 2003): is a library that is written is C++ programming language and provides a range of tools for common tasks found in speech processing. This library is used to extract Mel-frequency Cepstral coefficients (MFCCs).
- **Festvox-2.7.0-release** (Black & Lenzo, 2014): helps to building synthetic voices for limited domains.
- **Festival-2.4-release** (The Festival Speech Synthesis System , 2014): is a multilingual TTS synthesis system that provides a platform for creating new TTS synthesis systems.
- **SPTK-3.6** (SPTK Working Group, 2012): contains speech signal processing tools.
- **Praat** (Boersma & Weenink, 2013): is used for pitch marking and for extracting speech signal features.
- **EHMM_labeller** (Black & Lenzo, 2014): is used for automatic labelling and is included with latest version of MARY TTS synthesis system.

A script in Appendix A is used to install necessary packages and to set global path variables. Additional installation instructions are mostly found in the file named INSTALL inside each software package. These software packages are freely downloadable and one should agree to and obey their licences.

The MARY TTS synthesis system was developed as a collaborative project of Language Technology Lab at the German Research Centre for Artificial Intelligence and the Institute of Phonetics at Saarland University (Pammi *et al.*, 2010). This system supports creation of new HMM-based and unit selection synthetic voices from well-resourced to under-resourced languages. The adapted workflow of the MARY TTS synthesis system for voice creation is illustrated in Figure 3.8. This system is developed in Java and provides an easy to use graphical user interface (GUI). The steps followed under transcription GUI and basic NLP components are detailed in Section 3.3.3. The speech synthesis system components are described in Section 3.3.4. The MARY TTS builder uses a database from Wikipedia[1] to create new language support. Some steps are removed from the original workflow of Schröder *et al*. (2011) because this research study is mainly focused on selected under-resourced languages not available from the Wikipedia database.

3.3.3 Natural Language Processing Modules

The transcription GUI component used is a part of MARY TTS synthesis system that is used to create new under-resourced language modules before speech synthesis voices are created. The finite state transducer (FST) training procedure requires a phone set file (*allophone.xy.xml*) and pronunciation dictionary (*xy.txt*) in the target language. From this point, letters *xy* are used as a locale to refer to all under-resourced language locales.

[1] Available at: https://en.wikipedia.org/wiki/Wikipedia:Database_download

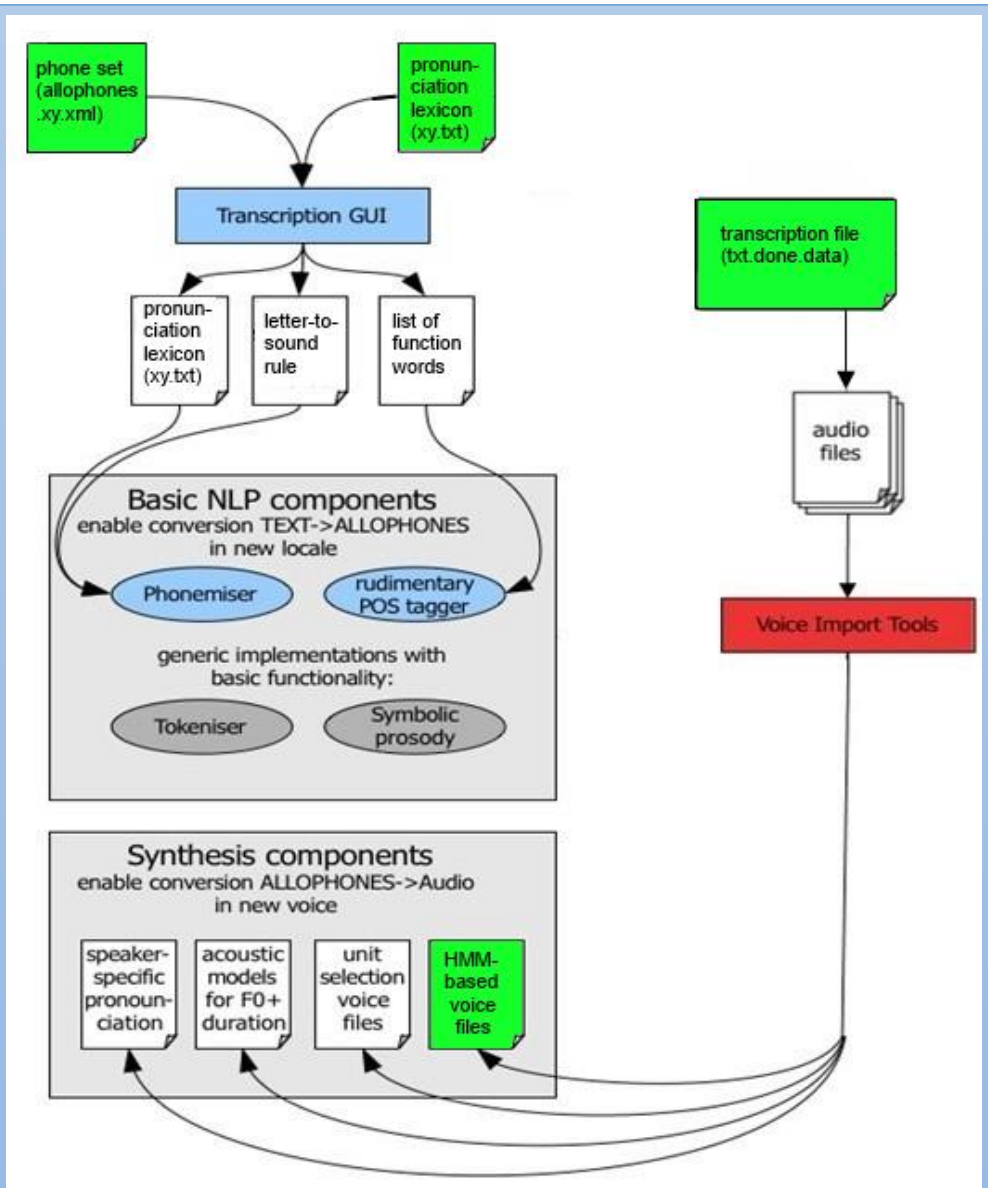


Figure 3.8: Workflow for multilingual voice creation in MARY TTS builder. Adapted from Schröder *et al.* (2011).

The Lwazi[1] phone set is adapted and used to create the new phone set files that are used by MARY TTS to recognise all phones involved in a target language. The four languages phone set files are created for Sepedi, Tshivenda, isiNdebele and Xitsonga as shown in Appendix B. The phone set file follows the SAMPA

[1] Available at: http://hlt.mirror.ac.za/Phoneset/Lwazi.Phoneset.1.2.pdf

representation format. Table 3.2 shows all consonants and vowels used in this research project. The transcription GUI tool uses WEKA toolkit to create NLP components in a form of a FST trained by J45 tree classifier. We have encountered two difficulties when creating HMM voices (for Xitsonga and isiNdebele) and language modules (for Xitsonga):

Challenge 1: The MARY transcription aligner uses a pipe character to align phones. Since our phone sets contain pipe character, then the transcription tool produced an error while compiling a new voice.

Solution 1: We used an alternative notation to phones that contain a pipe character (see Appendix B for isiNdebele and Xitsonga phone set).

Table 3.2: Phone Set

Language	Consonants	Vowels
isiNdebele	b, bh, c, ch, d, dl, dz, f, g, gc, gh, gq, h, hl, j, k, kgh, kh, l, m, n, ng, nk, ny, p, ph, q, r, rh, s, t, th, tj, tjh, tl, tlh, ts, tsh, v, w, y, z	a, e, i, o, u
Sepedi	b, bj, d, f, fs, fš, g, h, hl, j, k, kg, kh, l, m, my, n, ng, ny, p, ph, ps, psh, pšh, r, s, t, th, tl, tlh, ts, tsh, tš, tšh, w, y, š	a, e, i, o, u
Tshivenda	b, d, dy, dz, dzh, f, fh, g, h, h, k, kh, l, m, n, n', ny, p, ph, r, s, sh, sw, t, ts, tsh, v, w, x, y, yh, z, zh, zw	a, e, i, o, u
Xitsonga	b, by, c, ch, d, dh, dl, dlh, dy, dz, dzh, f, g, gh, h, hl, j, k, kh, l, m, mh, n, n', ng, nh, njh, ny, p, ph, phy, py, q, r, rh, s, sw, t, th, thy, tl, tlh, ty, v, vh, w	a, e, i, o, u

Problem 2: Furthermore, the Xitsonga has related phones (e.g. /d/, /dh/, /dl/, /dlh/, /dy/, /dz/, /dzh/ and the transcription tool tries to classify or map these phones to one letter “d”) compared to other languages. This caused the transcription tool to produce the error (saying *cannot handle multi-valued nominal class*). This error is mostly encountered where there are many similar phones that are not related and none of them explicitly defines certain alphabets (from English alphabets). The transcription tool is unable to define SAMPA phones for

letters “**c”** and “**j”** automatically; hence, this causes WEKA to produce an error while creating the NLP components shown in Figure 3.8.

Solution 2: Hence, we solved the issue by mapping SAMPA phones to an alternative notation (see Appendix B for Xitsonga phone set).

Table 3.3 shows the features and their possible values. These features carry important parts of a language including intonation, prosody, stress, and others. In addition, this can be used for generating expressive or emotional speech. A feature can carry only one value at a time. Feature values are described as follows:

- ***ph*** is assigned to a SAMPA phonetic symbol of a vowel or consonant.
- ***vlng*** is assigned to 0 = n/a, s = short, l = long, d = diphthong, and a = schwa
- ***vheight*** is assigned to 0 = n/a, 1 = high, 2 = mid-high, 3 = mid-low, and 4 = low
- ***vfront*** is assigned to 0 = n/a, 1 = front, 2 = mid, and 3 = back
- ***vrnd*** is assigned to 0 = n/a, + = on, and – = off
- ***ctype*** is assigned to s = stop, f = fricative, a = affricative, n = nasal, l = liquid, and r = approximant
- ***cplace*** is assigned to l = bilabial, a = alveolar, p = palatal, b = labio-dental, d = dental, and v = velar
- ***cvox*** is assigned to 0 = n/a, + = on, and – = off
- ***casp*** is assigned to 0 = n/a, + = on, and – = off
- ***cpal*** is assigned to 0 = n/a, + = on, and – = off

Table 3.3: Phone Set Features		
Feature description	**Feature name**	**Possible values**
Phone	ph	Vowel or consonant
Vowel length	vlng	0, s, l, d, and a
Vowel height	vheight	0,1, 2, 3, and 4
Vowel frontness	vfront	0, 1, 2, and 3
Vowel lip rounding	vrnd	0, +, and -
Consonant type	ctype	s, f, a, n, l, and r
Consonant place	cplace	l, a, p, b, d, and v
Consonant voicing	cvox	0, +, and -
Aspirated consonant	casp	0, +, and -
Palatal consonant	cpal	0, +, and -

In addition, the transcription GUI tool assigns a feature to zero when that feature is not set or declared for a particular phone. By default, all phones must be included in the phone set for a particular target language. The phone set (*allophone.xy.xml*) file contains the following line declaring the name, language and most importantly the features used in the file.

```
<allophones name="sampa" xml:lang="xy"
       features="vlng vheight vfront vrnd ctype cplace cvox casp cpal">
```

3.3.3.1 Pronunciation Dictionary

The phoneme-based pronunciation dictionaries are obtained from the Lwazi project. They are available at the language Resource Management Agency (RMA) (Language Resource Management Agency, 2016) for Sepedi, Tshivenda, isiNdebele and Xitsonga. The dictionaries used are the original versions of the

Lwazi pronunciation dictionaries by Davel and Martirosian (2009). All the words in the pronunciation dictionaries are manually checked and adapted to be compatible with the transcription GUI tool. The details of the pronunciation dictionaries are outlined in Figure 3.9.

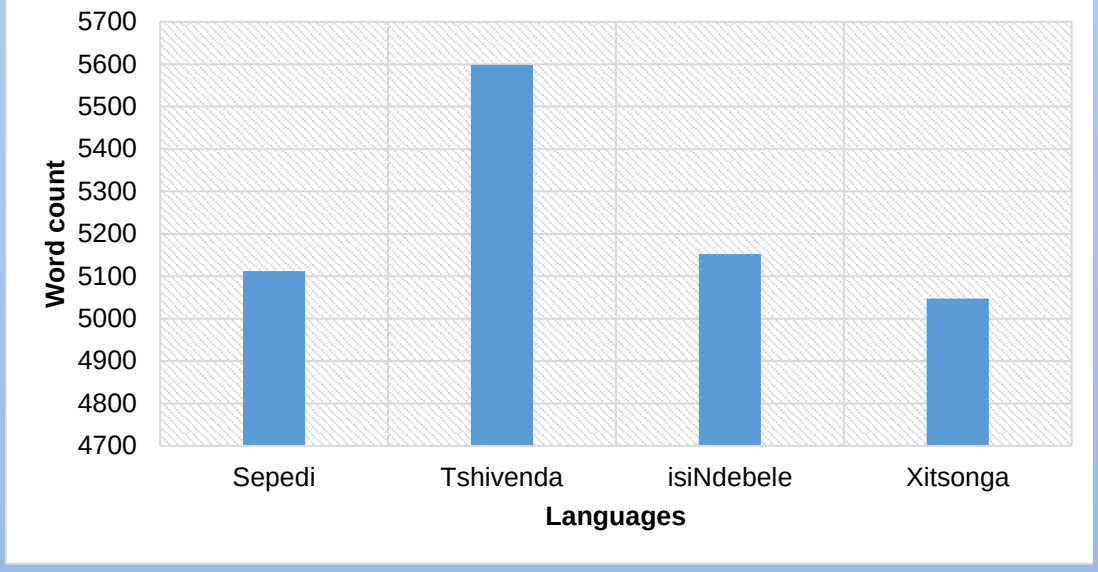


Figure 3.9: Pronunciation dictionary setup

The python script generating the dictionary is given in Appendix C. This script outputs a phoneme-based pronunciation dictionary that contains words together with their SAMPA phonetic description. This script also marks all the words functional. The transformation format is as follows:

Original Transcription			**Transformed transcription**
tshepišo	ts_h E p_> I S O	1	0 → tshepišo ts_hEp_>iSO functional
magetla	m a G E tl_> a	1	0 → magetla maGEtl_>a functional

The procedure used to adapt or transform the pronunciation dictionary to MARY TTS is as follows:

i. Open input pronunciation dictionary file,
ii. Read the next line from the input file,

iii. Remove all single blank spaces,
iv. Split the line according to tabs
v. Create new line of string by joining an item at index 1, index 2 and appending string “*functional*”; all items are separated by a blank space,
vi. Write a line to an output file
vii. Repeat step (ii) to (vi) until all lines are read from input file.
viii. Close output file.

3.3.3.2 Lexicon, Letter-to-Sound Rule and Part-of-Speech Tagger

The transcription tool is a user graphical interface that supports semi-automatic procedure for transcribing new language database and automatic training of a G2P rule file and a LTS rule file. The transcription tool is launched on the terminal by the following command:

```
$ ./voice/sources/marytts/target/marytts-builder-5.2-
SNAPSHOT/bin/transcription.sh
```

On the transcription GUI tool, function words can be selected or checked. Furthermore, the LTS predictor is trained and is used to predict the phonetic description of words that are not transcribed. All functional words are saved in order to build a primitive part-of-speech (POS) tagger that works on simple context-free string matching. The transcription GUI tool generated the following important files:

- *xy*.*lts* – LTS for transcribing unknown words,
- *xy_lexicon.dict* – phoneme-based pronunciation dictionary file,
- *xy_lexicon.fst* – grapheme-to-phoneme file,
- *xy_pos*.*fst* – POS tagger classifies and tags parts of a sentence according to their classes or categories (classes can be prepositions, articles, verbs, adjectives, and others). Functional words and content words are the only two categories in the POS tagger creation,

where *xy* represents a locale of a target language as previously explained in Table 3.1.

3.3.3.3 Implementation of New Language Modules

A minimal NLP module is built using the files generated by transcription GUI tool. The simple way to build the NLP module for the new under-resourced language is to modify the pre-existing language project with simple NLP components. One of the projects in the MARY TTS synthesis system that contains simple NLP components is adapted for the selected four target languages. The four under-resourced language projects are renamed to *marytts-lang-xy* and saved to directory at path */voice/sources/marytts/marytts-languages/.*

The under-resourced language projects are created by adopting one of the pre-existing language projects. The procedure to implement the minimal NLP module is adopted from the MARY TTS GitHub page[1].

The language projects are added to master pom (*marytts/marytts-languages/pom.xml*) as a new subproject under modules tag (see Listing 3.2). These language projects are also added as a dependency in the assembly-runtime plugin (*marytts/marytts-assembly/assembly-runtime/pom.xml*) and assembly-builder plugin (*marytts/marytts-assembly/assembly-builder/pom.xml*) files using the code given in Listing 3.3.

```
...
<modules>
      <module>marytts-lang-nso</module>
      <module>marytts-lang-ven</module>
      <module>marytts-lang-nbl</module>
      <module>marytts-lang-tso</module>
</modules>
...
```

Listing 3.2: Languge modules included in *marytts-languages* project

[1] Available at: https://github.com/marytts/marytts/wiki/New-Language-Support

```
...
<dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>marytts-lang-nso</artifactId>
    <version>${project.version}</version>
  </dependency>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>marytts-lang-ven</artifactId>
    <version>${project.version}</version>
  </dependency>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>marytts-lang-nbl</artifactId>
    <version>${project.version}</version>
  </dependency>
  <dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>marytts-lang-tso</artifactId>
    <version>${project.version}</version>
  </dependency>
...
```

Listing 3.3: Languge modules included in *assembly-builder* module

The final step is to build the full project. A new Maven directory is created inside each language project called *target* and it contains the following executable Java files (.jar) of the newly built language projects;

- `marytts/marytts-languages/marytts-lang-nso/target/marytts-lang-nso-5.2-SNAPSHOT.jar`
- `marytts/marytts-languages/marytts-lang-nbl/target/marytts-lang-nbl-5.2-SNAPSHOT.jar`
- `marytts/marytts-languages/marytts-lang-ven/target/marytts-lang-ven-5.2-SNAPSHOT.jar`
- `marytts/marytts-languages/marytts-lang-tso/target/marytts-lang-tso-5.2-SNAPSHOT.jar`

These Java files are saved to the MARY TTS builder and runtime at the following directories respectively:

```
marytts/target/marytts-builder-5.2-SNAPSHOT/lib
marytts/target/marytts-5.2-SNAPSHOT/lib
```

The following commands are used to start the MARY TTS server on Ubuntu terminal or Windows command prompt respectively:

```
$ ./voice/sources/marytts/target/marytts-5.2-
SNAPSHOT/bin/marytts-server

> voice\sources\marytts\target\marytts-5.2-SNAPSHOT\bin\marytts-
server.bat
```

These new languages can be tested by accessing this link http://localhost:59125/locales on an internet browser. The link retrieves installed locales or languages. The output contains all the locales installed in the MARY TTS synthesis system as shown in Figure 3.10. Therefore, we have successfully added a support for the four new under-resourced languages (*nbl*, *nso, ven* and *tso*).

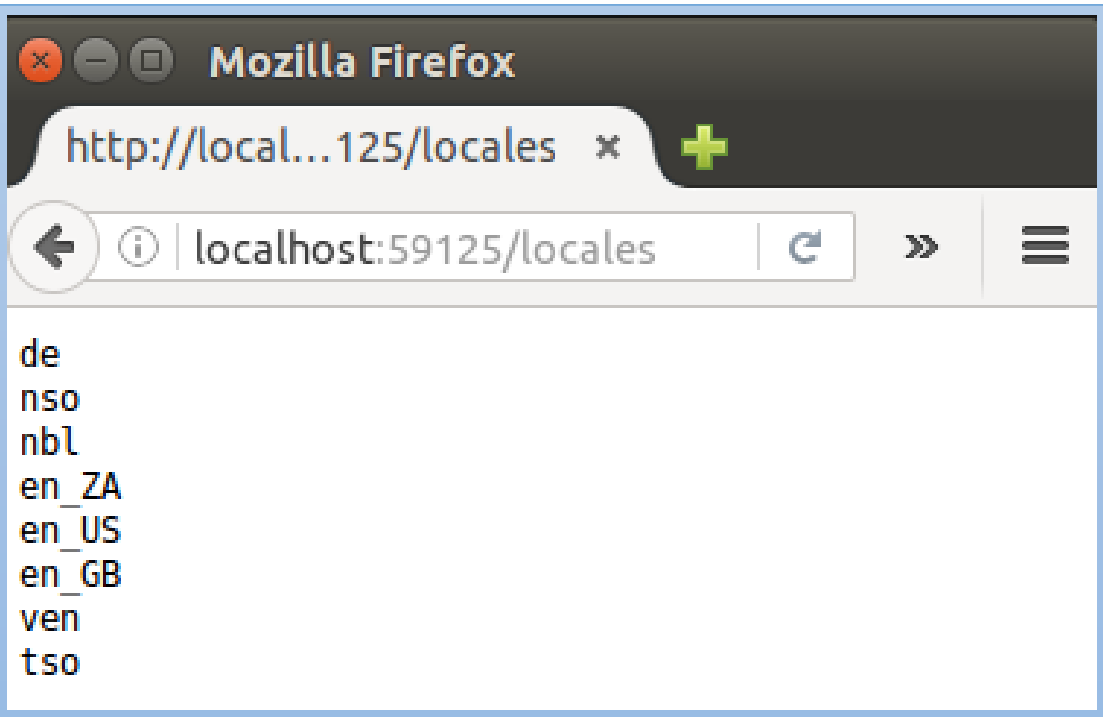


Figure 3.10: The output of current language locales installed in MARY TTS synthesis system

3.3.4 TTS Synthesis Modules

This section details the steps followed to create new synthetic voices. There are different kinds of speech synthesis methods that can be used when creating synthetic voices. The HTS method is adopted to create the synthetic voices. This method is selected since it requires a small training dataset, and is fast and efficient.

3.3.4.1 Preparation

The acquired speech corpus dataset is of type waveform with sampling frequency of 16 KHz. Moreover, the audio format is on mono channel with bitrate of 16 bit per sample.

The acquired speech audio files are used to train HMM models for speech synthesis. The corpus consists of speech utterances together with their corresponding text annotation which helps to transcribe the speech sample and to properly train the HMM models. The sentences are carefully numbered and saved to a FestVox file called *txt.done.data* that is used when creating MARY prompt or transcription files in which each sentence is saved to its text file. The root directory for voice building is set to a directory at */voice/xy* for each language. The speech corpus is saved to the directory at */voice/xy/wav/* and the transcription files are saved to the file at */voice/xy/txt.done.data*.

The working directories need to be created before launching the voice import tool. The four directories are created for each language. The settings in Table 3.4 are configured according to the type of target voice by launching voice import tools from the following command at root level on the terminal:

```
$ ./voice/sources/marytts/target/marytts-builder-5.2-
SNAPSHOT/bin/voiceimport.sh
```

On the new pop-up window, the working directory is selected and used during the entire process of training a new voice. The *db.estDir* property is the path to speech tools while db.gender is the gender of voice. The *db.locale* is the locale of the language, *db.marybase* is the path to MARY TTS, and *db.rootDir* is the working root directory. The *db.samplingrate* is the sampling rate of the recordings, *db.voicename* is the name of the new voice, and *db.wavDir* is the path to audio wave files. Other settings are filled automatically depending on the voice-building path. After saving the general configuration settings, then the main window appears showing the methods in Figure 3.11.

Table 3.4: General Configuration Settings

Property	Value
db.estDir	/voice/sources/speech_tools/
db.gender	female (or male)
db.locale	xy
db.marybase	/voice/sources/marytts/
db.rootDir	/voice/xy
db.samplingrate	16000
db.voicename	new_language_voice_name
db.wavDir	/voice/xy/wav/

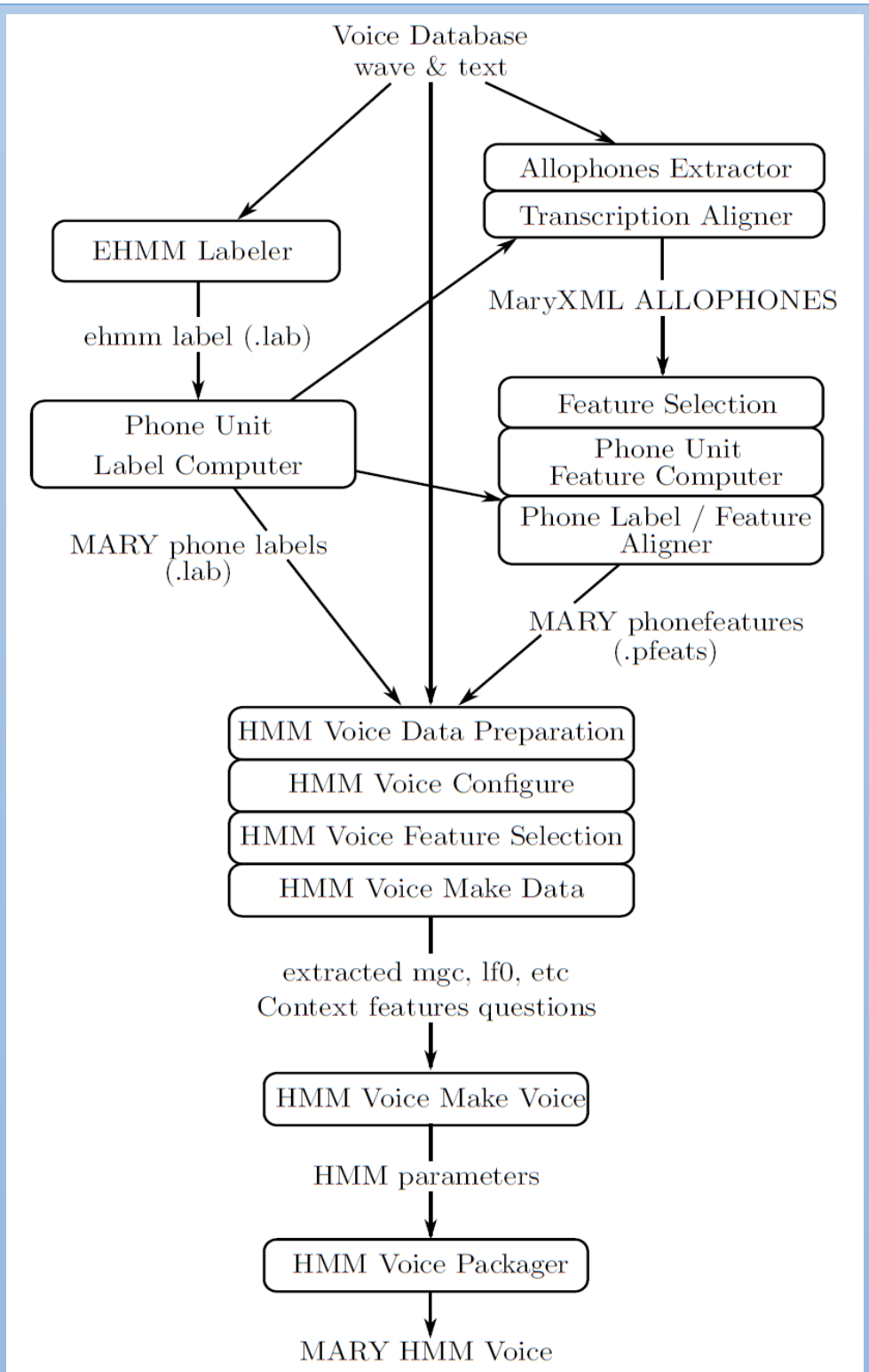


Figure 3.11: HMM-based voice training in MARY TTS. Adapted from Würgler (2011)

3.3.4.2 HMM-based Voice Training

The voice training process take several hours to complete building each TTS voice depending on training corpus size and performance of the workstation or computer used for training. This section details the procedure taken to develop HMM-based voices by assuming that the installation instructions and declaration of path variables given in Appendix A have been done.

Once the language support is implemented as discussed in Section 3.3.3, a new voice is now ready to be created. We used the *voice import tool* for creating new voices (Pammi *et al.,* 2010). The tool is periodically updated on the MARY TTS page on GitHub[1]. The training of the new voice uses the main processing components given in Figure 3.11. A component is executed by selecting or checking the appropriate checkbox and then clicking on the "run" button. Parameters and configurations of each module are accessed by clicking the settings editor button next to the checkbox. The main modules shown in Figure 3.11 are explained below, according to their execution order (Pammi *et al.*, 2010):

The *PraatPitchmarker* verifies the frequency range for male and female voices. Settings for male voice have maximum pitch of 300 and minimum pitch of 75 while female voices have maximum pitch of 500 and minimum pitch of 100 (Pammi *et al.*, 2010). The raw acoustics are saved to the directory at */voice/xy/pm/**.

The *MCEPMaker* uses Edinburgh speech tools to extract MFCCs from all audio wave files. The raw acoustics are saved to the directory at */voice/sources/mcep/*.mcep*.

The *Festvox2MaryTranscriptions* use the FestVox transcription file to create MARY transcription files (separate text files). The results are saved to the directory at */voice/xy/text/*.txt*.

[1] Available at: https://github.com/marytts/marytts/wiki/VoiceImportToolsTutorial

The *Allophones Extractor* processes the transcription file (txt.done.data) of the speech audio files to generate a MARY extensible markup language (XML) allophone files (initial allophones). This component requires a MARY TTS server. The results are saved to the directory at */voice/xy/promp_allophones/*.xml*.

The *EHMM Labeller* is an external component called by MARY TTS to label the audio wave files using the corresponding transcriptions. This step may take several hours to complete, depending on the amount of data used. The results are saved to the directory at */voice/xy/ehmm/**. The following variable is set in the component settings editor:

```
EHMMLabeler.ehmmDir = /voice/sources/marytts/lib/external/ehmm
```

The *Label Pause Deleter* removes pauses from label files. The results are saved to the directory at */voice/xy/lab/*.lab*. The following variable is set in the component settings editor:

```
LabelPauseDeleter.pauseDurationThreshold = 10
```

The *Phone Unit Label Computer* converts the label format from EHMM to MARY. This component inputs the lab directory and outputs the phone lab directory at */voice/xy/phonelab/*.lab*.

The *Transcription Aligner* verifies that labels and allophone files are correctly aligned. The results are saved to the allophone directory (final allophones) at */voice/xy/allophones/*.xml*.

The *Feature Selection* confirms a list of all the features to be considered in the next steps and saves the features to a file at */voice/xy/mary/features.txt*. This component extracts contextual and linguistic features such as vowel height, vowel frontness, consonant place, consonant type, consonant roundness, accent, stress, and others.

The *Phone Unit Feature Computer* extracts context feature vectors from the text data. The component creates a phone features directory at

/voice/xy/phonefeatures/.pfeats*. This component requires the MARY TTS server to be running.

The *Phone Label Feature Aligner* verifies that there is proper alignment between phone features and phone labels. The results are displayed on the console, saying, “0 problem”.

The next steps illustrate HMM voice creation showed in Figure 3.12.

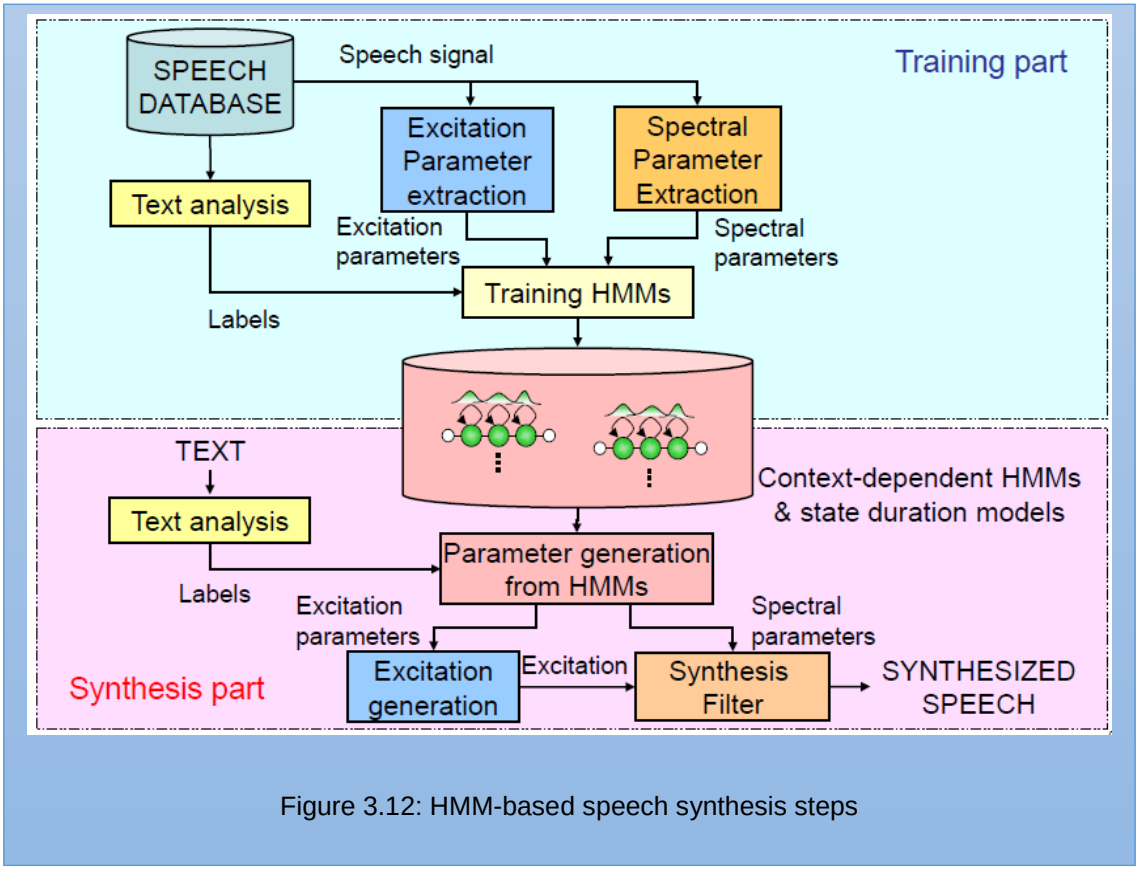


Figure 3.12: HMM-based speech synthesis steps

The *HMM Voice Data Preparation* sets up the environment to create a HMM voice and uses the following configuration file to check availability of the required external programs, text and audio wave files and their paths: */voice/sources/marytts/lib/external/externalBinaries.config*. It converts all audio wave files to raw files by using the Sound eXchange (SoX) tool.

The *HMM Voice Configure* configures voice properties. Both male and female voices are available in the Lwazi corpus at 16 kHz and 16 bitrates. Table 3.5

shows different setting values used for configuring voices, including fast Fourier transform length, frame length, frame shift, frequency wrapping, lower fundamental frequency (F0), upper F0, and other settings remain as default. These values depend on the sampling rate and gender of the voice.

The *HMM Voice Feature Selection* reads features created by the *Feature Selection* component and generates a new list of features used to train the HMM models. The results are saved to a file at */voice/xy/mary/hmmFeatures.txt*.

The *HMM Voice Make Data* component uses the HTK tool patched with the code provided by HTS to train HMMs. This step uses speech signal processing toolkit (SPTK) and *Snack* to extract speech signal parameters including voicing strengths for mixed excitations (STR), log F0 (LF0), Mel-generalised Cepstral (MGC) coefficients and Fourier magnitudes (MAG) to form an acoustic parameter vector (MGC+LF0+STR+MAG). This component executes in the */voice/xy/hts/* directory and generates question sets saved to the directory at */voice/xy/hts/data/questions/**.hed.

Table 3.5: Some HMM Voice Configuration	
Property	**Value**
HMMVoiceConfigure.fftLen	512
HMMVoiceConfigure.frameLen	400
HMMVoiceConfigure.frameShift	80
HMMVoiceConfigure.freqWarp	0.42
HMMVoiceConfigure.lowerF0	Male=40, Female=80
HMMVoiceConfigure.mgcBandWidth	24 (for cepstral form)
HMMVoiceConfigure.mgcOrder	24 (for cepstral form)
HMMVoiceConfigure.sampfreq	16000
HMMVoiceConfigure.speaker	Speaker_name
HMMVoiceConfigure.upperF0	Male=280, Female=350

The *HMM Voice Make Voice* uses a version of the speaker dependent training scripts provided by HTS that was adapted to the MARY TTS platform by adding sections for the acoustic parameter vector. As shown in Figure 3.13, this step takes several hours to finish to execute the following script:

```
perl scripts/Training.pl scripts/Config.pm > logfile &
```

The *HMM Voice Compiler* compiles the voice project using Maven. The path to Maven is set to */voice/soft/maven/bin*. The new HHM-based voice components are ready to be installed. The next section explains the procedure taken to install the new HMM-based voices onto the MARY TTS synthesis system.

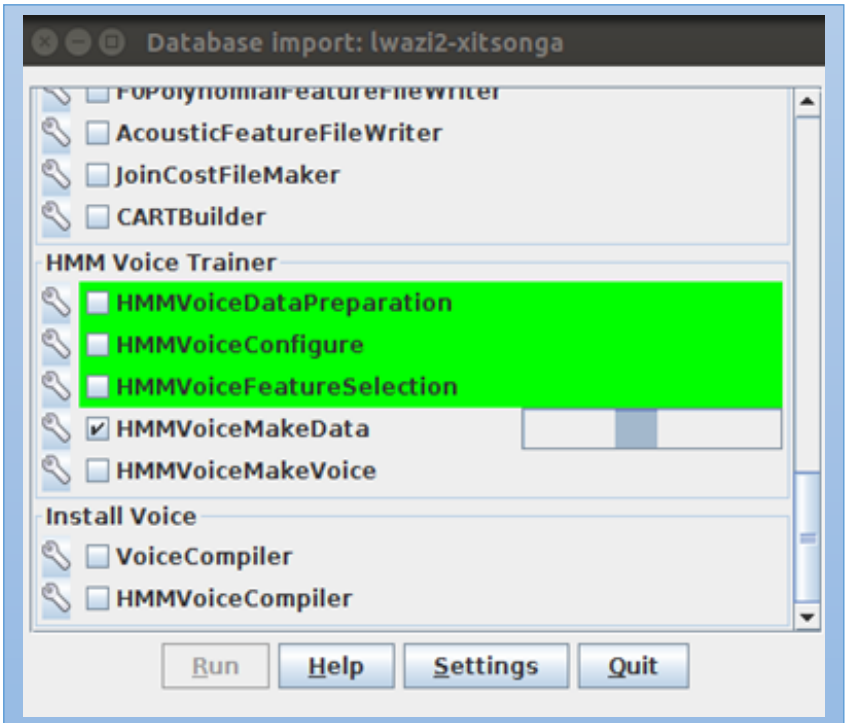


Figure 3.13: HMM-based voice creation in process

3.3.4.3 Implementation of New TTS Voice Components

The previous section explained the training procedures of new voices. This section highlights the procedure of integrating new voices to the MARY TTS synthesis system.

Table 3.6 shows the size of each developed voice. The HMM-based voices have a small size compared to unit-selection voices (Zen et al., 2009). This training process took less time to train each voice compared to when training unit-selection TTS voices, which may take days to train. The HMM-based TTS synthesis systems are robust, fast and efficient, and require less computation costs. This enables the use of such systems in small storage devices like mobile handsets, navigation devices, and many similar others.

Table 3.6: Developed TTS Voice Sizes	
Voice	**Size**
Sepedi	1.9 MB
Tshivenda	1.4 MB
isiNdebele	1.8 MB
Xitsonga	1.5 MB

After training the new under-resourced HMM-based voices, the results are four installable zip files:

```
/voice/nso/mary/voice-sepedi-lwazi2-hsmm/target/voice-lwazi2-sepedi-hsmm-5.2-SNAPSHOT.zip
/voice/nbl/mary/voice-lwazi2-ndebele-hsmm/target/voice-lwazi2-ndebele-hsmm-5.2-SNAPSHOT.zip
/voice/ven/mary/voice-lwazi2-tshivenda-hsmm/target/voice-lwazi2-tshivenda-hsmm-5.2-SNAPSHOT.zip
/voice/tso/mary/voice-lwazi2-xitsonga-hsmm/target/voice-lwazi2-xitsonga-hsmm-5.2-SNAPSHOT.zip
```

These four installable zip files contain the HMM-based voices that are compatible to the MARY TTS synthesis system. These files are saved to the MARY TTS runtime download directory at:

```
/voice/sources/marytts/target/marytts-5.2-SNAPSHOT/download/
```

The MARY TTS component installer in Figure 3.13 is a GUI that is used to install new languages and voices to the MARY TTS synthesis system. The following command is used to launch the GUI component installer.

```
/voice/sources/marytts/target/marytts-5.2-SNAPSHOT/bin/marytts-component-installer
```

The new voices are installed under their respective languages (locales). As shown in Figure 3.14, the voice (e.g. lwazi2sepedi) is found under its language (e.g. nso).

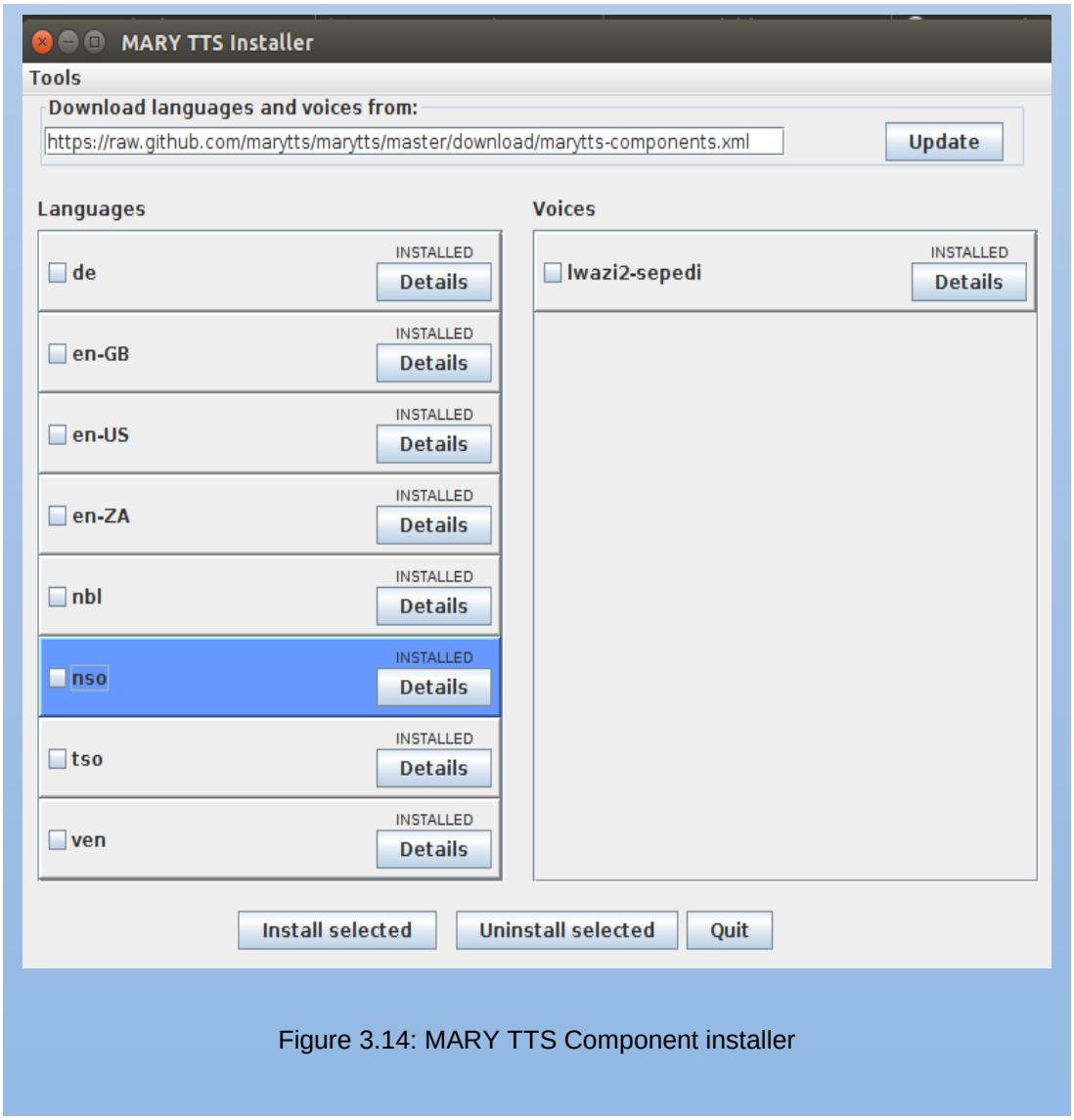


Figure 3.14: MARY TTS Component installer

The MARY TTS GUI client can be accessed through the web address http://localhost:59125 and through the GUI component on the Ubuntu Linux platform at */voice/sources/marytts/target/marytts-5.2-SNAPSHOT/bin/marytts-client* or on the Microsoft windows operating system at */voice/sources/marytts/target/marytts-5.2-SNAPSHOT/bin/marytts-client.bat* (see Figure 3.15).

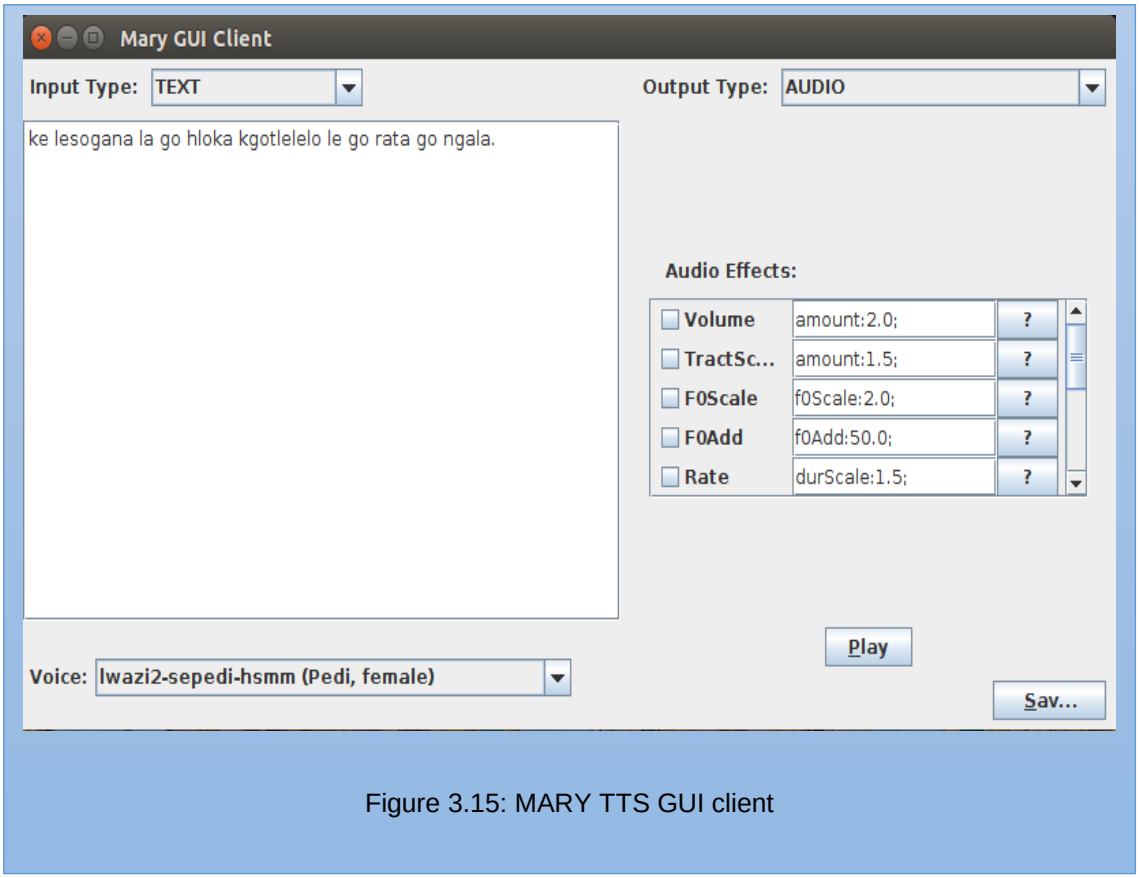


Figure 3.15: MARY TTS GUI client

3.4 Integration of the LID and TTS Synthesis

This section details the implementation of the LID and TTS synthesis system module as one system function (or patching the MARY TTS synthesis system to support LID). The LID is integrated into MARY TTS synthesis system with modifications and additions of certain Java files. Maven is used to install MARY TTS synthesis system from source. The following sub-projects (or modules) are included in the MARY TTS synthesis system during installation:

- marytts-languages
- marytts-builder
- **marytts-runtime**
- marytts-assembly
- marytts-client

- marytts-common
- marytts-redstart
- marytts-signalproc
- marytts-transcription
- voice-cmu-slt-hsmm

The MARY TTS synthesis system uses mainly the runtime module to generate new synthetic speech. As a result, this module is upgraded in order to support machine-learning functions that implement classification functionality. The WEKA software is added as a dependency to the runtime module by including the WEKA repository in the maven project file at path */voice/sources/marytts/marytts-runtime/pom.xml* (see Listing 3.4).

```
...
<dependency>
      <groupId> nz.ac.waikato.cms.weka </groupId>
      <artifactId> weka-dev </artifactId>
</dependency>
...
```

Listing 3.4: The WEKA Maven repository included in *marytts-runtime* module

The MARY TTS synthesis system server is upgraded to support incoming text to be classified. The "*classify"* extension or pattern (in Listing 3.5) is added to the *HttpRequestHandlerRegistry*. The Java file used for handlers is found at path */voice/sources/marytts/marytts-runtime/src/main/java/marytts/server/http/ MaryHttpServer.java*.

HttpRequestHandlerRegistry maintains a map of hypertext transfer protocol (HTTP) request handlers keyed by a request uniform resource identifier (URI) pattern and *InfoRequestHandler* looks up a handler matching the given request. The class *InfoRequestHandler* and *SynthesisRequestHandler* are called when the pattern */classify* and */process* are matched respectively.

```
...
// Set up request handlers
HttpRequestHandlerRegistry registry = new HttpRequestHandlerRegistry();
registry.register("/process", new SynthesisRequestHandler());
InfoRequestHandler infoRH = new InfoRequestHandler();
registry.register("/classify", infoRH);
...
```

Listing 3.5: Upgraded *MaryHttpServer* java file by including handler for pattern /classify

MARY TTS system supports both sockets and HTTP protocols. It listens on HTTP port 59125 with the following patterns:

- **/classify** – used for classification. It receives *text* and *model* from hypertext markup language (HTML) form. Text represents a surname and model represents the name of the model or classifier (where there may be multiply classifiers). Finally, it returns the locale.
- **/process** – used for voice generation. It receives *text* and *locale* from HTML form. Text represents the input surname and locale represents the classified or chosen language. Finally, it returns the raw audio data.

The Java file *InfoRequestHandler* is upgraded to recognise the pattern */classify* (see extract of Listing 3.6). The LID usage uniform resource locator (URL) pattern is as follows:

http://localhost:59125/classify?text=surname&model=namesModel

As shown in extract of Listing 3.6, the *classifyLang string variable* stores the value of text (surname) and *model string variable* stores the LID model name. If these two variables are set, then they are passed as arguments to the constructor method in class *NamesPredictor.* The contents of *NamesPredictor.java* file are given in Appendix D. This Java file belongs to the package *weka.classifiers*. The *NamesPredictor* class receives two parameters where the first parameter sets the text and the second parameter sets the target classifier model. The method *makeInstance()* is called from the constructor to prepare an instance to be classified. This instance follows the WEKA ARFF file format and must match the

data that is used to train the target classifier model. The *loadModel()* method is called by the constructor to use WEKA *SerializationHelper* to read or load the model from local file. The class *InfoRequestHandler* calls the classify method from *NamesPredictor* class to perform classification and returns the predicted language locale.

The MARY TTS system is reinstalled using the following command from terminal:

```
voice/sources/marytts$ mvn install
```

The WEKA toolkit (weka.jar) is saved to the *lib* directory with other Java files of MARY TTS synthesis system at

```
/voice/sources/marytts/target/marytts-5.2-SNAPSHOT/lib/*
```

The LID model (classifier) is saved to the same *bin* directory containing *marytts-server* and *marytts-client* scripts at

```
/voice/sources/marytts/target/marytts-5.2-SNAPSHOT/bin/*
```

```
...
if (request.equals("classify")) {
   //Text classification starts here ...
   if (queryItems != null) {
      String classifyLang = queryItems.get("text");
      String model = queryItems.get("model");
      if (classifyLang != null) {
         if (model.equals("namesModel")) {
            NamesPredictor pred = new NamesPredictor(classifyLang,model);
            return pred.classify();
         }
      }
   }
   MaryHttpServerUtils.errorMissingQueryParameter(response, "'effect'");
   return null;
}
...
```

Listing 3.6: Upgraded *InfoRequestHandler* Java file by including conditional statement for pattern */classify*

3.5 Live Demonstration of the System

The TTS synthesis systems for well-resourced languages are becoming more readily commercially available on the internet platform. Some of the available multi-language commercial TTS synthesis systems are CereProc[1], NeoSpeech[2], Cepstral[3], IBM Watson[4], and Acapela[5] TTS demo. These TTS synthesis systems do not reflect inclusion of indigenous South African official languages. Hence, developing and deploying a prototype TTS synthesis system that covers some of the under-resourced indigenous languages promotes technological footprint and awareness of these languages and also helps encourage initiatives towards retention and endorsement of the cultural identity of the languages.

This section details the implementation of the complete system. It explains the commands used to set up the cloud-based server. It details the procedures used to deploy the developed system. It also details the interaction of the developed system with a client-based application. The development of the Android application is explained.

3.5.1 Server

The MARY TTS synthesis system is deployed to a virtual private server (VPS) running Ubuntu server 16.04 long-term support operating system installed with Java. The VPS contains the following specifications that are technologically sufficient to host the proposed system.

- 1 core processor
- 1 Gigabyte of random access memory
- 50 Gigabytes of solid-state drive space
- 64 bit operating system architecture

[1] Available at: https://www.cereproc.com/
[2] Available at: http://www.neospeech.com/
[3] Available at: http://www.cepstral.com/en/demos/
[4] Available at: http://text-to-speech-demo.mybluemix.net/
[5] Available at: http://www.acapela-group.com/voices/demo/

The server is installed with *nginx* reverse proxy on port 80 to serve MARY TTS synthesis system. A special service account is created named *mary* to run the service and to manage installation files. The newly created account is given ownership of the installation files that are saved to path */local/mary/*.

A configuration file is added to */etc/nginx/sites-available/default* with contents in Listing 3.7. Then the *nginx* service is restarted.

```
...
server {
    location / {
                proxy_pass http://127.0.0.1:59125;
}
...
```

Listing 3.7: Configuration file added to nginx

The MARY TTS sysnthesis system server is started by running *marytts-server* script from bin directory using the new *mary* account and the script was allowed to run in the background. A *cron job* is set in order to restart the MARY TTS server in case of any system crash or memory leakage or system reboot. A new bash file is created with contents given in Listing 3.8. This bash file is made executable.

```
#!/bin/bash

# Version to adapt to your system
VERSION=5.2

#Check if our service is currently running
ps auxw | grep marytts-server | grep -v grep

# if the service is not running it returns a non zero to the environment
viriable,
# in that case we start the service, else we ignore.
if [ $? != 0 ]
then
    bash /local/mary/marytts/bin/marytts-server –Xmx150m
fi
```

Listing 3.8: Mary.sh file to restart the server

This file is saved to location */usr/local/bin/mary.sh.* The cron job is set to 5 minutes interval using the following command:

```
sudo crontab –e
*/5 * * * * /usr/local/bin/mary.sh
```

For every 5 minutes interval this cron job checks if the MARY TTS server is still running, and it launches the *mary.sh* script that restarts the MARY TTS server if is not running. The firewall is set up to allow incoming traffic on port 80 (HTTP). The traffic received on port 80 is proxied to the MARY TTS synthesis system server. The system is successfully deployed on the VPS as the server. Application clients can access the system via internet connection (see Figure 3.16).

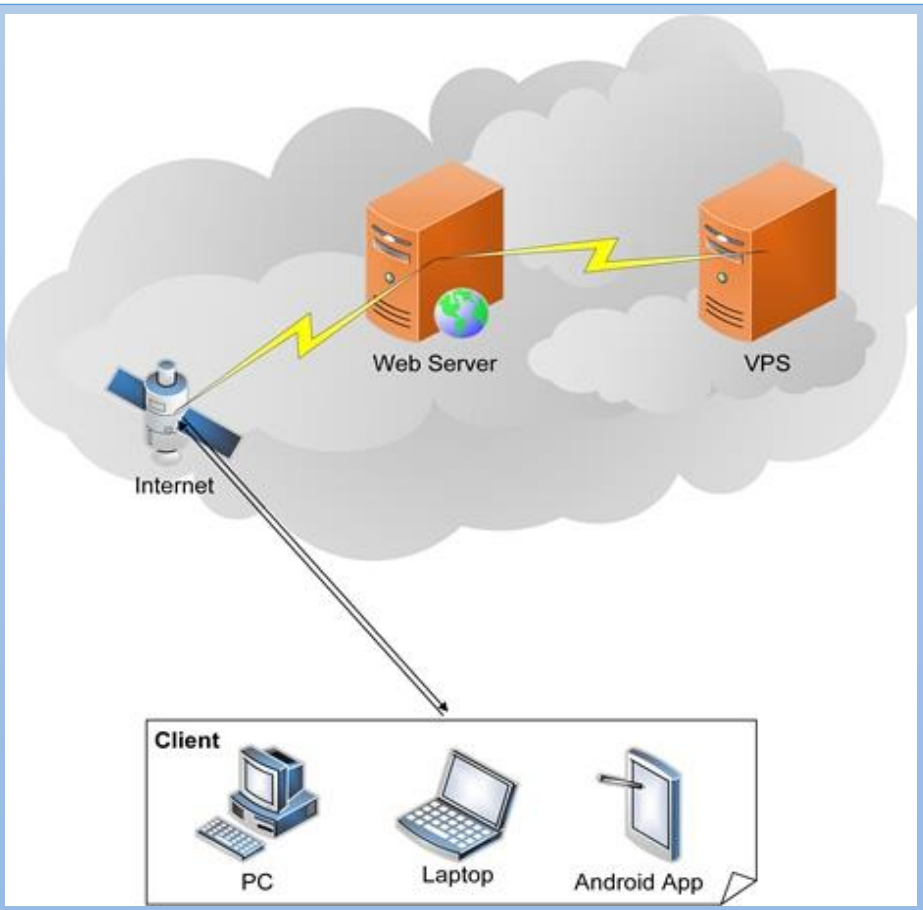


Figure 3.16: Design of the application client and server connection via wireless connection

3.5.2 Client – Internet Browser

Client is an internet browser accessing the website on a web server. The website is deployed to hide the VPS. It contains HTML and hypertext pre-processor (PHP) scripts that point to the VPS for both voice generation and LID prediction of a language. The system has API that can be found on the API tab on the website. It explains how to fetch audio using HTML form. The live demonstration of the system is available on the project website and can be accessed using the internet connection on https://www.speechtech.co.za.

3.5.3 Client – Android Application

The application is developed for Android and tested on API version 23. As shown in Figure 3.16, the application serves as a client that connects to the system on the server-side. The surname and locale inputs are sent to the server and the server acknowledges the connection and returns the audio data back to the client.

The application has the following four classes which are given in Appendix E:

MainActivity – this class sets the main layout to front and the layout contains text areas and buttons.

LanguageIdentification – this class handles LID of surnames by creating HTTP connection to the server. It receives surname and return locale.

Methods – this class contains methods needed by the main activity, including network connection, verification of text, and text encoding.

PlayAudioManager – this class plays audio using media player instance. It receives URL containing audio location.

The application GUI in Figure 3.17 contains (a) menu option with list of languages, (b) *Text Area* and (c) three *buttons* namely *Clear*, *Download* and *Speak*. These buttons are clickable and call certain methods defined with *setOnClickListener*. The *Clear* button resets the *Text Area* when clicked, the

download button creates HTTP requests to download the audio, and the *Speak button* when clicked applies the following procedure:

Step 1: Calls *urlBuilder(model)* method by passing model name as a parameter.

Step 2: If there is no surname parameter then a message is displayed to the user and returns false, or else then encodes the surname by calling the method *encode_text(string)* from class *Method*.

Step 3: Checks if the language is set by calling a method *isempty(string)* from class *Method*, or else displays the message to the user to select the language.

Step 4: If the language is set to "**detect**", then check if there is internet connection by calling method *isInternet* from class *Method*. If internet is available, the URL is built and passed to the LID class.

Step 4.1: The LID class creates HTTP connection to the server in the background using an asynchronous task to classify the input surname parameter. If the surname parameter is classified, then a message is displayed to the user stating a classified language and the language is used to build the final URL (LID URL), and returns true.

Step 5: If the user sets the language then a final URL is built (user URL) and returns true.

Step 6: If step 4 or 5 returns true then it means the URL is built and the method *player(URL)* is called with URL as parameter. This method calls class *PlayAudioManager* to create *MediaPlayer* instance that plays the audio from a given URL. The android application unified modelling language (UML) diagram of all the classes is given in Appendix F. The UML diagram details the relationship between the classes and all the member functions (methods) and variables.

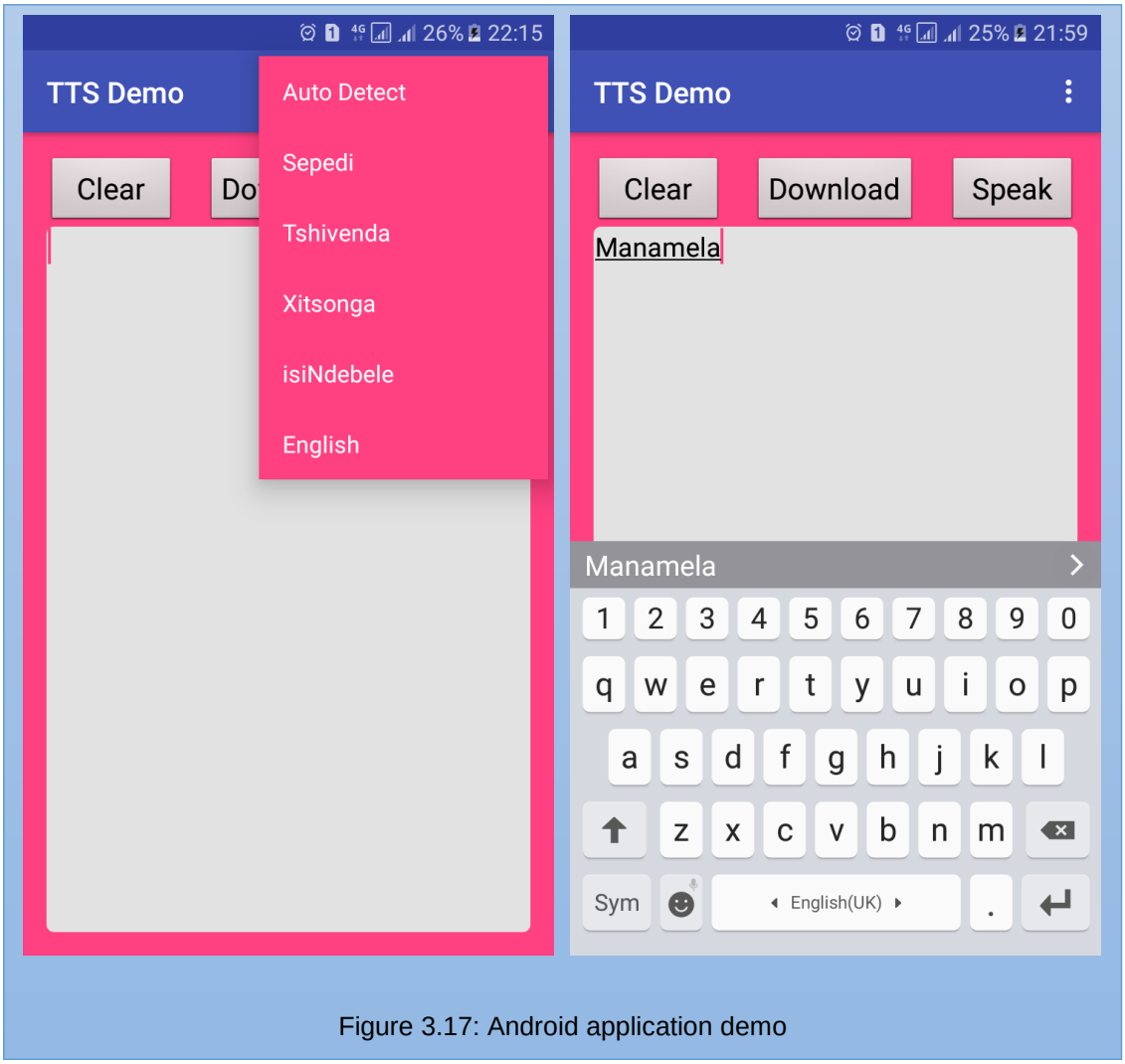


Figure 3.17: Android application demo

3.6 Summary

In this chapter, we have discussed the most important steps of the implementation phase in detail. The text and speech data used for training and evaluation in all experiments were discussed. The speech technology and data mining resources and toolkits to set up LID classifier and TTS experiments were outlined. The feature set for creating LID is outlined. We have discussed the supervised machine-learning algorithms. The pronunciation dictionaries used were also discussed. We have discussed the steps followed when creating LID classifier and TTS in Sepedi, Xitsonga, Tshivenda and isiNdebele. The implementation phase of the proposed system was outlined. The developed

system is deployed to the production cloud server stability testing on real data. The Android application and project website were also explained.

As captured in the first chapter, the aim of the research study was to develop a TTS synthesis system that uses a trained classifier to enhance correct pronunciation of words and phrases, particularly proper names for four under-resourced languages of the Limpopo province, South Africa, namely Sepedi, Tshivenda, Xitsonga and IsiNdebele. To achieve this goal, the following objectives have been accomplished:

a) The acquisition of textual data in the form of a corpus containing surnames was acquired from the University of Limpopo student database.
b) The speech training data was acquired from the Lwazi TTS corpus for four languages.
c) We have used machine-learning classifiers to train LID model for classification of surnames into respective first languages.
d) The LID model was used as a language predictor to identify a first language given surname. This model forms a front-end phase of the system. We built the baseline TTS synthesis modules in Sepedi, Xitsonga, Tshivenda and isiNdebele. These modules serve as a back-end of the system.
e) Once the identity of a language has been predicted from a given surname, then an appropriate TTS synthesis system is activated to continue with pronunciation guidance using the predicted first language.

The last objective is discussed in the next chapter, including evaluation metrics and the results obtained. The results of the evaluation try to answer research questions outlined in the first chapter.

4 CHAPTER 4: EVALUATION RESULTS

This chapter illustrates the methods used for evaluating the developed LID and TTS synthesis system. It details the analysis of the results in an attempt to answer the research questions stated:

- Can a computational system use a person's surname to predict the identity of the first-language of that person?
- Can a computational system produce an appropriate pronunciation of indigenous proper names?

4.1 Introduction

The automatic pronunciation assistant is designed and implemented to be a complete HMM-based TTS synthesis system implementing machine-learning technologies. These learning technologies were applied during LID training and the same classifier models were evaluated or tested on the same data using a n-fold cross-validation. The prototype system is currently available on the internet platform for further testing on real-world data.

In this chapter, the perceptual evaluations are described, illustrating the state of our work. The following evaluation metrics were selected for distinct purposes:

- The **test for intelligibility**, the ability of a human listener to understand and interpret the words and meaning of the synthesised utterances with ease. Subjects are asked to listen and write down the synthesised utterances.
- The **test for quality**, an abstract measure of pleasantness of voice, naturalness of voice, and correct pronunciation of the utterance**.** Subjects are asked to rate the speech quality on a 5-point Likert scale.

The layout is as follows:

- Section 4.2 discusses performance metrics used to measure the performance of the LID component and speech synthesis phases.

- Section 4.3 discusses evaluation results of the LID component.
- Section 4.4 discusses MOS test results of the TTS synthesis phase.
- Section 4.5 discusses evaluation results of the entire system usability.

4.2 Performance Measures of the Proposed System

The user acceptance of the state-of-the-art speech technology systems depends on the appropriate appraisal of the system performance. We evaluated the performance of the proposed system by measuring the front-end LID system, back-end TTS system and overall system usability separately.

4.2.1 Evaluation Metrics for LID

This section details the testing results obtained during LID evaluation on a 10-fold cross-validation. The experimentation setup was defined from the previous chapter (see Section 3.2 of Chapter 3). We have evaluated the performance of each classifier model based on certain criteria to assess the prediction accuracy of the classifier model. The performance of a trained model is determined by how good the predictions reflect the actual classes. The evaluation measurements are given in the following classification terms:

- FP = False positive means observations where the actual class is negative and the predicted class is positive.
- TN = True negative means observations where the actual and predicted class is negative.
- TP = True positive means observations where the actual and predicted class is positive.
- FN = False negative means observations where the actual class is positive and the predicted class is negative.

These classification terms are used when evaluating a supervised model wherein a labelled observation is compared to a predicted observation such that the confusion matrices can be generated from each experiment. Table 4.1 shows the design of a confusion matrix from a binary classification.

Table 4.1: Confusion Matrix

Model Class		**Actual class**	
		Positive	Negative
Predicted class	Positive	**True Positive**	False Positive
	Negative	False Negative	**True Negative**

The true positives and true negatives shown in blue font define surnames that were correctly predicted. The following evaluation metrics were used to evaluate the performance of the LID classifier models:

Accuracy is measured as the proportion of correctly identified surnames in the test set, compared to all the surnames in the same test set. It is defined by the equation:

$$Accuracy = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \times 100\% \quad (4.1)$$

Where i represent number of occurrences.

Precision is defined as the percentage of applicable surnames identified out of all identified surnames. The equation for precision is given by:

$$Precision = \frac{TP_i}{TP_i + FP_i} \times 100\% \quad (4.2)$$

Where i represent number of occurrences.

Recall (or sensitivity) is defined as the percentage of applicable surnames correctly predicted out of all applicable surnames in the collection and it is defined by the equation:

$$Recall = \frac{TP_i}{TP_i + FN_i} \times 100\% \tag{4.3}$$

Where i represent number of occurances.

F_1 score is the weighted average of precision and recall. It takes both false positives and false negatives into account. Intuitively, F_1 score is more useful where there is uneven class distribution. The F_1 score formula is given by

$$F_1 = \frac{2*(Recall*Precision)}{Recall+Precision} \tag{4.4}$$

Root mean-squared error (RMSE) is arguably the most essential criterion used to evaluate the performance of a predictor. Its equation is represented by the square root of the average of the squares of the differences between actual and predicted values.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.5}$$

where n is the total number of observations, y_i is the actual values and $\hat{y}_i$ is the predicted values $\forall i \in \mathbb{Z}^+$ and $0 \notin \mathbb{Z}^+$.

4.2.2 Subjective Evaluation Metrics for TTS

Subjective listening tests were used to measure the intelligibility and quality of the developed TTS voices. Thirty-two native speakers were recruited via a “word of mouth” campaign to participate in the evaluation tests. No remuneration or gifts were provided to participants in exchange for information. The participants were University of Limpopo under-graduate and post-graduate students. They were divided into four groups of mixed gender according to their native language. Each group contained 8 participants to evaluate their first-language TTS voice. The participants’ age ranged from 18 to 35 and the subjects had no prior experience working with TTS synthesis systems. The participants agreed to sign a consent form and answered the questionnaire given in Appendices G and H respectively. The questionnaire is used to gather all the data required for evaluation of the synthesised speech generated by the developed system against the natural speech. Evaluations were conducted in a reasonably sound-controlled room with no noise disturbances.

The intelligibility and quality of synthesised speech is measured using the following example sentence, more sentences are given in Appendix I.

Buka e sepetše godimo ga lefase le le botse.	The book walked through the attractive floor.

The sentences given in Appendix J were used to measure quality of natural speech using MOS test. The sentences to measure natural speech were in 16-bit linear pulse-code modulation (PCM) and were extracted from the speech corpus. The TTS intelligibility tests are done using different methods including DRT, MRT, SUS, and WER. The SUS method has an advantage of providing no semantic contextual cues to the intelligibility of the individual words and hence it is chosen for testing TTS intelligibility at sentence level and word level. The WER is also used to measure intelligibility at word level. The quality of generated TTS voices is measured using MOS test against the following factors: *understandability, pleasantness, pronunciation,* and *naturalness*.

4.2.2.1 Semantically Unpredictable Sentences

The SUS test measures speech intelligibility in synthesised sentences that are syntactically correct but semantically meaningless (Benoît *et al.*, 1996). Five sentences of different syntactic structure were constructed as recommended by Benoît *et al*. (1996) and synthesised audio is played to the subject in a form of mixed part-of-speech template like DET ADJ NOUN VERB ADJ (see Appendix I.1). A good example of a SUS is “The model modal successful and the guide”. To avoid learning effects, subjects listened to the sentences only once. An additional three sentences were constructed and used in training session so that participants were well aware of acoustic content of the audio. The participants listened to the sentences synthesised by the developed synthesiser during the testing session where they transcribed the synthetic utterances of the audio. The participants wrote the sentences on a given questionnaire and those who have unclear handwriting were asked to type on a computer. The percentage of correct identifications is used as an intelligibility metric. The metric at word level is given by:

$$SUS_w = \frac{1}{S}\sum_{i=1}^{S}\frac{\hat{w}}{W} \times 100\% \tag{4.6}$$

The metric at sentence level is given by:

$$SUS_s = \frac{1}{S}\sum_{i=1}^{S}\hat{s} \times 100\% \tag{4.7}$$

where $\hat{w}, W, \hat{s}$ and S have the following meanings:

- $\hat{w}$ is the number of correctly identified words.
- W is the total number of words in a sentence.
- $\hat{s}$ is the number of correctly identified sentences.
- S is the total number of sentences.

4.2.2.2 Word Error Rate

A WER is a common metric used to measure the performance of an ASR or machine translation system on word-level (Jurafsky & Martin, 2014). The WER is used in subjective evaluation of intelligibility of TTS synthesis systems. The WER is based on the minimum number of insertions, deletions and substitutions that have to be performed to convert the generated text (or hypothesis) into the reference text. The first step in calculating word error is to find the minimum edit distance in words between the hypothesised and reference strings (Jurafsky & Martin, 2014). The results of the calculations will be the minimum number of word substitutions, word insertions, and word deletions necessary to match between the correct and hypothesized strings. We have applied WER on five sentences constructed by SUS test in the previous section. The intelligibility measure is typically captured by the WER metric formulated as:

$$WER = \frac{S+I+D}{N} \times 100\% \tag{4.8}$$

Alternatively, SER is formulated as:

$$SER = \frac{S_S}{N_S} \times 100\% \tag{4.9}$$

where S, I, D and N have the following meanings:

- S is the number of word substitution errors.
- I is the number of word insertion errors.
- D is the number of word deletion errors.
- N is the total number of words.
- S_s is number of sentences with at least one word error.

- N_s total number of sentences.

The implementation of SER and WER is given as a python script in Appendix K. The script receives a text file (as a parameter) that contains reference and hypothesis sentences respectively. These sentences are separated by a carriage return or line feed. The SER and WER results are printed on the terminal in percentage notation.

4.2.2.3 Mean Opinion Score

The quality of speech is measured based on the categories; naturalness, pleasantness, pronunciation, intelligibility, listening effort, flexibility, and similarity. Evaluators rated the produced speech signal based on a five-point Likert scale where 1 means “horrible” to 5 meaning “best”. A five-point Likert scale is chosen to allow evaluators to give neutral answers. The MOS score is described below.

1. Bad means no meaning understood
2. Poor means effort required
3. Fair means moderate effort required
4. Good means no appreciable effort required
5. Excellent means no effort required

The mean of the responses is calculated to compute the MOS results. The MOS is a performance metric applied to measure the quality of speech from subjective evaluations and the metric is given by:

$$MOS = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x \tag{4.10}$$

where x and n have the following meanings:

- x is the score of the evaluator.
- n is the total number evaluators.

The value of n is formulated as follows:

Eight subjects evaluate each synthetic voice and the highest possible score is five. The default value of n is the same as number of evaluators, hence $n = 8$.

4.3 Evaluation Results and Analysis of the Developed Front-end LID

The data collected during the testing of the developed front-end LID was analysed using descriptive statistics on a Microsoft Excel spreadsheet. The LID accuracy can be affected by several factors including features, type of the employed algorithm, target language, and the size of the evaluation and training data (Botha & Barnard, 2012). The classifier models were tested on different *n*-gram features to find the best features that can later be used for implementation of the system. The MNB and SVM classifiers were used to find the best accuracy for those features. The classifier models were built using a multiclass approach where each class represents its language. The classifier models were trained and tested using a 10-fold cross-validation on the same dataset of 3043 surnames of different length. The main aim of the LID is to answer the research question: "Can a computational system use a person's surname to predict the identity of the first language of that person?"

4.3.1 Kernel Parameter Selection

In order to calculate the optimum parameter selections, a parameter search process must be used to select the optimal kernel parameter values, so that the classifier can accurately discriminate unseen data (Chang & Lin, 2011). The *Cross Validation (CV) Parameter Selection* is one of the classifiers in the WEKA toolkit that searches the best model parameters on a given search interval for a given cross validation (Kohavi, 1995). The SVMs can perform better with correct

values for the kernel parameters. The MNB and linear SVM do not require special kernel parameters. The sigmoid SVM contains a *gamma* ($\boldsymbol{\gamma}$) and *coefficient* ($\boldsymbol{r}$) parameters that needed to be set. Hence, the *CV Parameter Selection* was performed on positive *gamma* values of [0 to 20], and $\boldsymbol{\gamma = 0}$ was selected as the best value for *gamma*. The value of *coefficient* parameter was selected as $\boldsymbol{r = -0.95}$ from the search interval of [-100 to 100]. We used the following recipe for sigmoid SVM.

```
weka.classifiers.meta.CVParameterSelection -P "R -100.0 100.0
1.0" -P "G 0.0 20.0 1.0" -X 6 -S 1 -W
weka.classifiers.functions.LibSVM -- -S 1 -K 3 -D 3 -G 0.0 -R -
0.95 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model
C:\Users\User\Documents\weka-3-8-0 -seed 1
```

The RBF SVM contains the *gamma* kernel parameter. The *CV Parameter Selection* for RBF SVM was performed to find the value of *gamma* on the interval [0 to 20], and $\boldsymbol{\gamma = 0}$ was selected as the best value. We used the following recipe for RBF SVM.

```
weka.classifiers.meta.CVParameterSelection -P "G 0.0 20.0 1.0" -
X 6 -S 1 -W weka.classifiers.functions.LibSVM -- -S 1 -K 2 -D 3
-G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model
C:\Users\User\Documents\weka-3-8-0 -seed 1
```

The polynomial SVM contains three kernel parameters, namely *gamma*, *coefficient* and *degree* ($\boldsymbol{d}$)**.** *Gamma* was searched on the interval of [0 to 20] and $\boldsymbol{\gamma = 0}$ was selected as the best value. The *Coefficient* was searched on the interval [-100 to 100] and $\boldsymbol{r = 0}$ was selected as the best value. The value for *degree* parameter was searched on the interval [0 to 20] and $\boldsymbol{d = 3}$ was selected as the best value. We used the following recipe for polynomial SVM.

```
weka.classifiers.meta.CVParameterSelection -P "R -100.0 100.0
1.0" -P "G 0.0 20.0 1.0" -D "G 0.0 20.0 1.0"  -X 6 -S 1 -W
weka.classifiers.functions.LibSVM -- -S 1 -K 1 -D 3 -G 0.0 -R
```

```
0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model
C:\Users\User\Documents\weka-3-8-0 -seed 1
```

The parameter selection was performed on a 10-fold cross-validation. All the support vector classification (SVC) experiments were performed using *nu-SVC* because the results were better than that of *C-SVC*. The parameter selection for larger numbers and *nu* parameter were not calculated because WEKA required high computational costs and out of memory error was encountered on the low computational power implementation platform used.

4.3.2 Multinomial Naive Bayes

The MNB model was built and evaluated on different character *n*-gram features. Figure 4.1 shows the classification accuracy obtained using character *n*-grams of size one up to five. The character unigrams (1-grams) achieved the lowest accuracy of 56.23% that is very low because the unigram classification set contained only the 26 letter English alphabet. The bigrams (2-grams) achieved second highest accuracy of 68.62% while trigrams outperformed other *n*-gram classification features with 69.34%. As shown in Figure 4.1, the trigrams (3-grams) performed better than all other *n*-gram sizes. The accuracy starts to decrease when *n*-gram size is further increased, and this is observed in most studies in the literature, that the higher the *n*-gram is, the lower the accuracy.

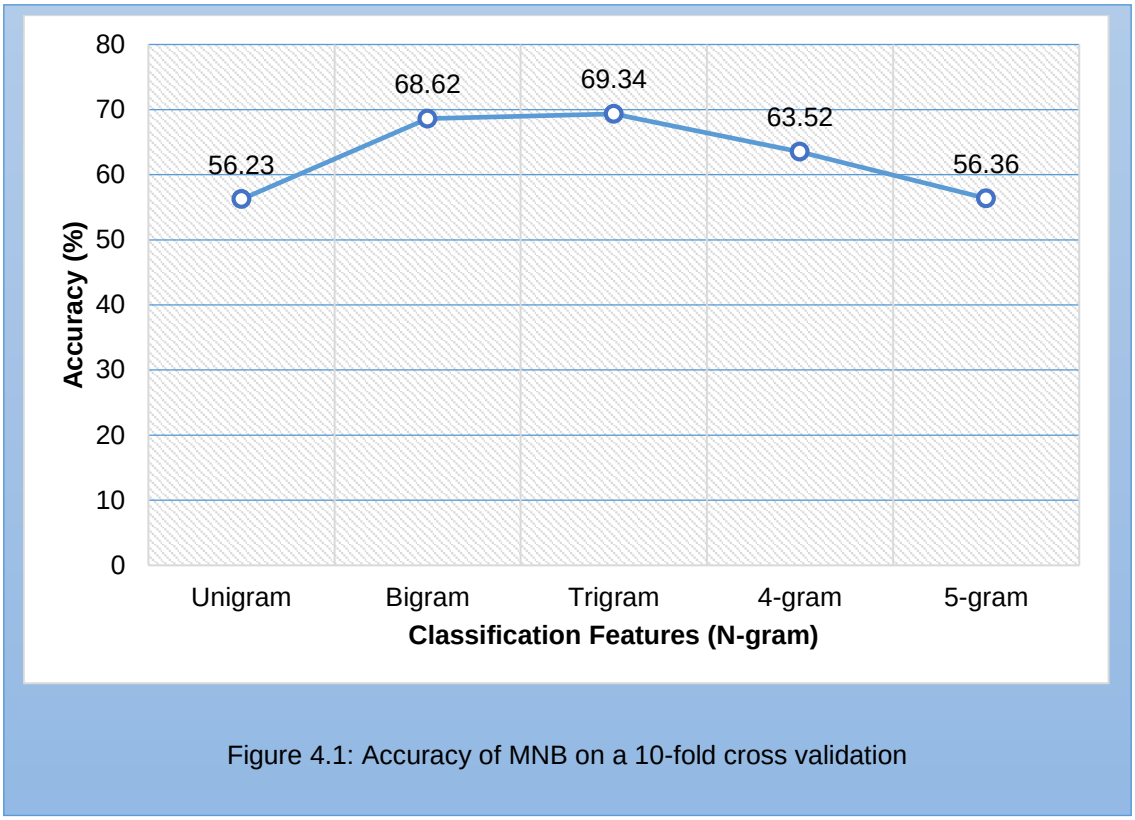


Figure 4.1: Accuracy of MNB on a 10-fold cross validation

Further tests were investigated by combining *n*-gram classification features. The results on *n*-gram combinations for MNB classifier are shown in Figure 4.2 with 4-gram and 5-gram features resulting in the lowest accuracy of 63.42%. This is because 4-gram and 5-gram features contain longer characters and cannot be used to discriminate the whole dataset. The 1-gram to 3-gram features resulted in accuracy of 69.64%; when adding 4-gram features the accuracy decreases to 69.54% and the accuracy further decreases to 69.24% when adding 5-grams. The MNB achieved highest accuracy of 69.80% on 2-gram to 4-gram features.

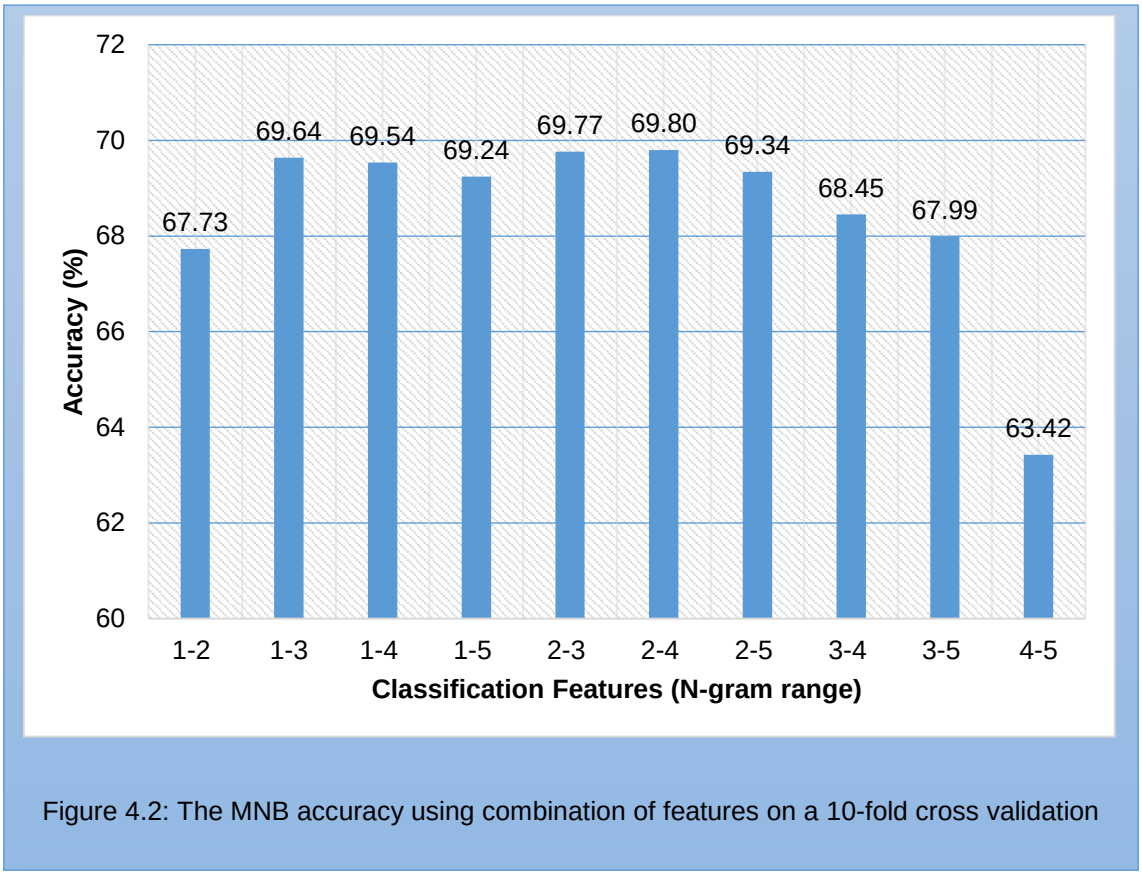


Figure 4.2: The MNB accuracy using combination of features on a 10-fold cross validation

4.3.3 SVM with Linear Kernel

The SVM performs best on a large amount of data. The robust SVM wrapper was used to experiment with the feature sets on a linear kernel. The linear SVM produced the lowest accuracy on unigrams and the accuracy further increased on bigrams, reaching highest accuracy of 68.32%; then it decreased by 0.62% on trigrams and further decreased on 4-gram and 5-gram achieving 55.73% (see Figure 4.3). The decrease in accuracy was encountered when the features were smaller in size and did not cover the whole feature space to make predictions on new or unseen dataset. The bigrams and 5-grams were nearly the same, which means the set of single characters and a 5-gram which was close to word level performed very poorly on the dataset. As illustrated in Figure 4.4, the classification features were mixed together and the accuracy reached above 70.00% on 2-3, 2-4, and 2-5 *n*-gram sets with 2-5 *n*-gram set having the higher accuracy.

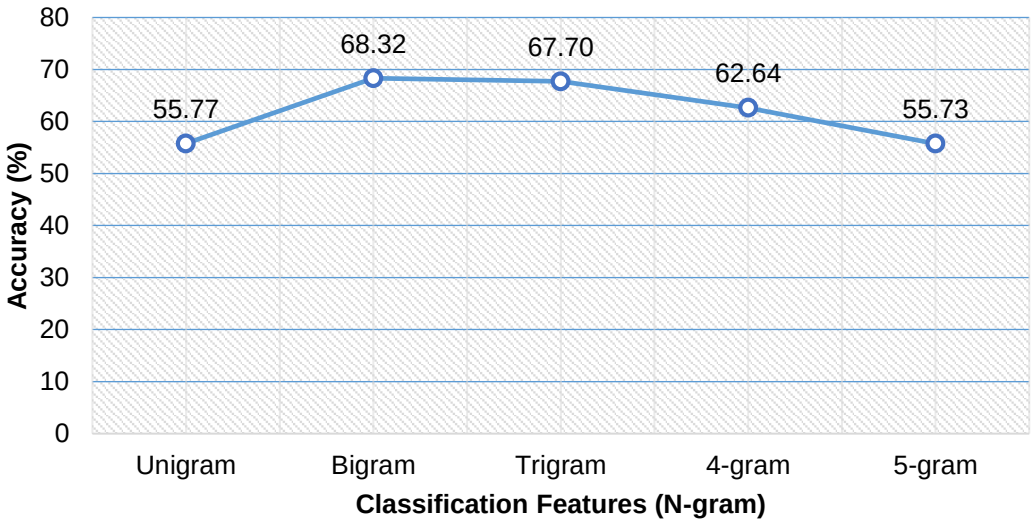


Figure 4.3: Accuracy of linear SVM on a 10-fold cross validation

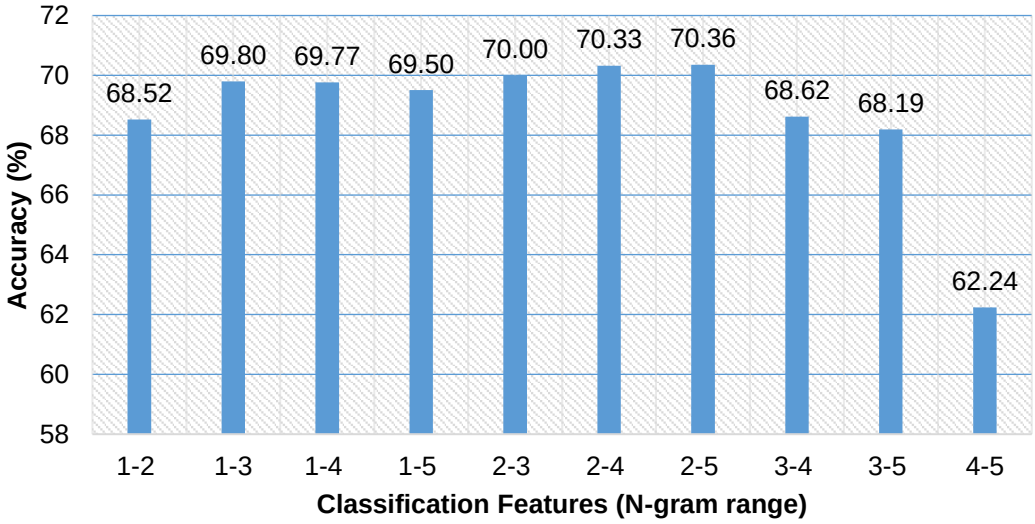


Figure 4.4: The linear SVM accuracy using combination of features on a 10-fold cross validation

4.3.4 SVM with RBF Kernel

RBF kernel is most commonly used to classify non-linearly separable data at a higher dimensional feature space. The RBF SVM produced accuracy of 61.12% on unigrams and reached a higher peak on bigrams with accuracy of 68.81%

(see Figure 4.5). When the number of *n*-grams increases, this results in a slight decrease in accuracy. In most studies, trigrams perform well compared to other *n*-grams (Fourie *et al.*, 2014). In this research work, larger character *n*-gram size has similar results with word unigram, since our data consists of single words. Thus, this results in low accuracy.

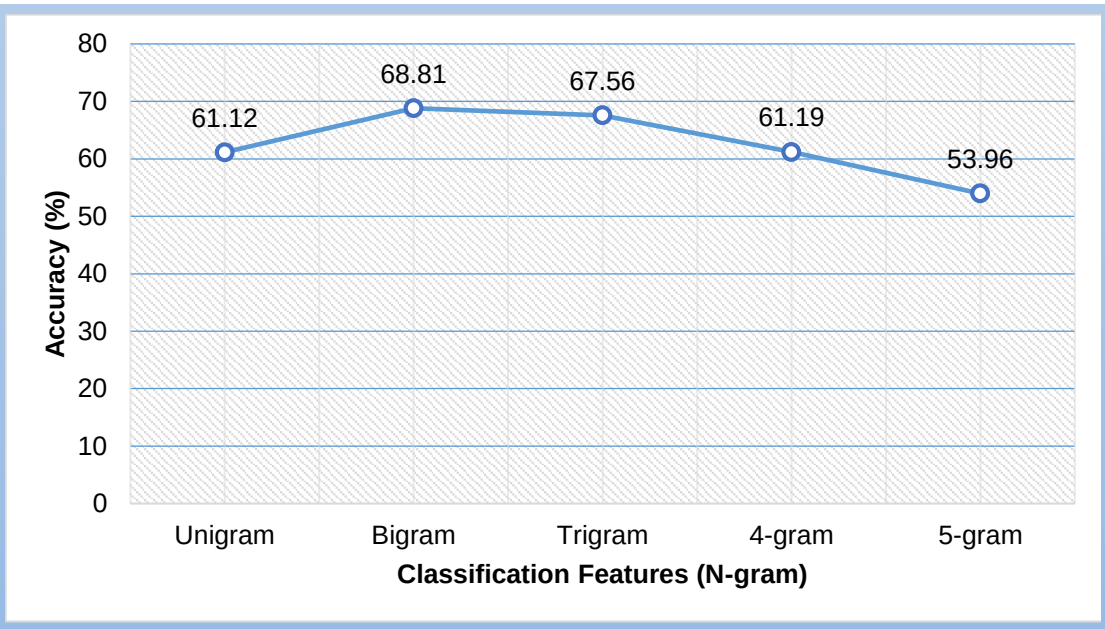


Figure 4.5: Accuracy of RBF SVM on a 10-fold cross validation

The RBF SVM achieved the accuracy of 70.52% on 2-5 n-gram sets (see Figure 4.6). This shows that the use of *kernel trick* does increase the accuracy at a certain feature level. The accuracy is above 69.00% at 1-2, 1-3, 1-4, 1-5, 2-3, 2-4, and 2-5 n-gram sets; this can be further improved by using correct kernel parameter $\boldsymbol{\gamma}$ (given in Equation 3.2).

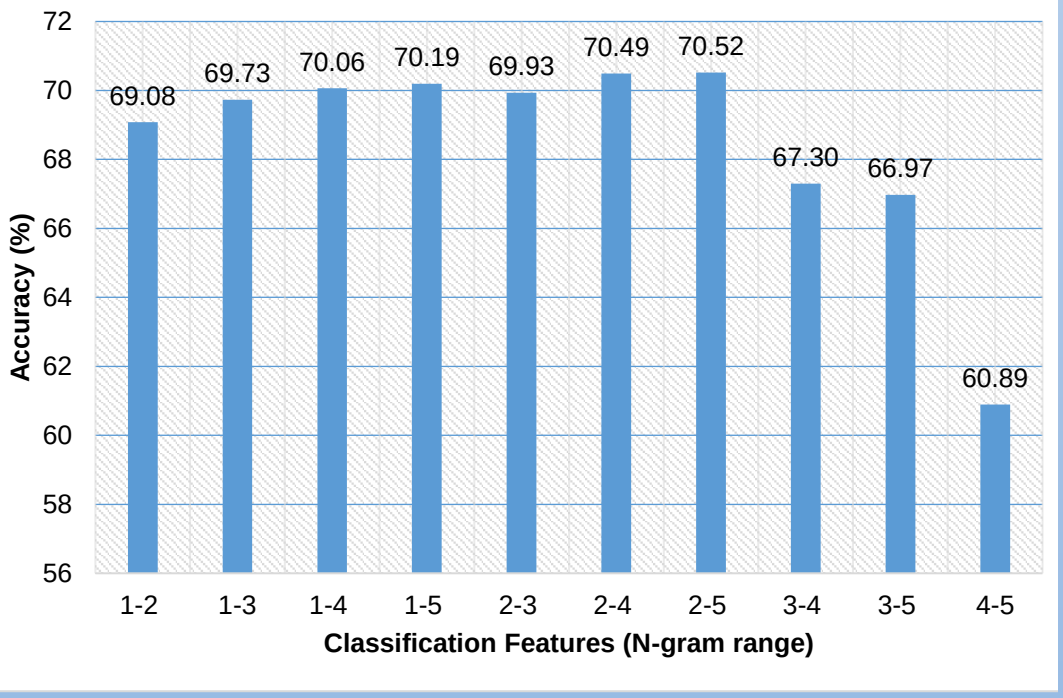


Figure 4.6: The RBF SVM accuracy using combination of features on a 10-fold cross validation

4.3.5 SVM with Sigmoid Kernel

The sigmoid SVM was trained according to Experiment 3 and achieved the lowest accuracy of 45.15% on 5-grams (see Figure 4.7). This shows that 5-grams were unable to correctly discriminate the classes or languages. The bigrams outperformed other *n*-grams with an accuracy of 68.75%. The feature set was increased to evaluate the sigmoid SVM on *n*-gram sets. As shown in Figure 4.8, the *n*-gram sets 1-2, 1-3, 1-4, 1-5, 2-3, 2-4, and 2-5 performed well by reaching above 69.00%, but the *n*-gram set 2-4 was the highest in accuracy.

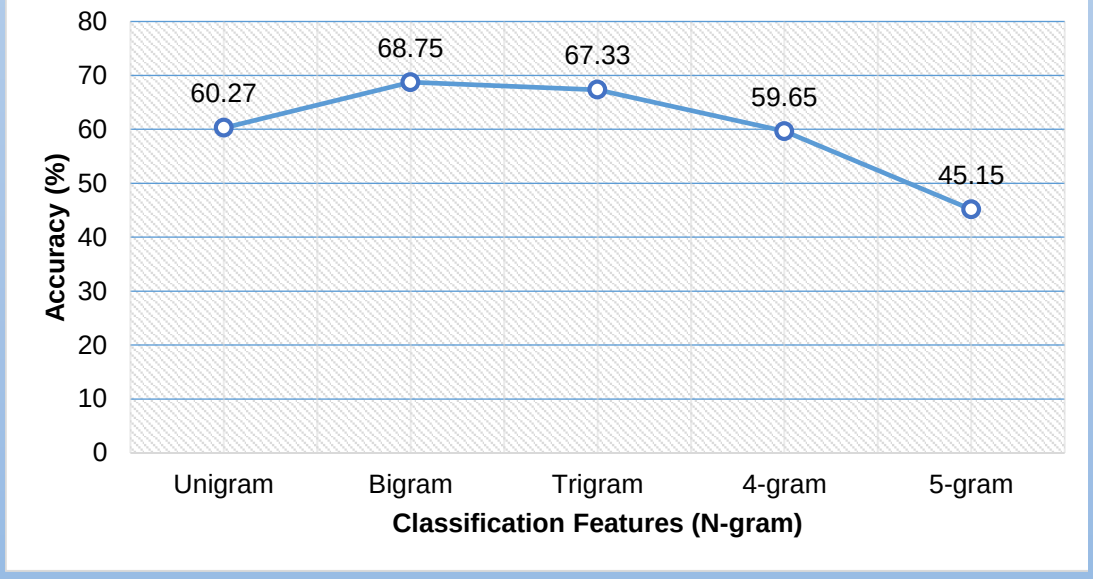


Figure 4.7: Accuracy of sigmoid SVM on a 10-fold cross validation

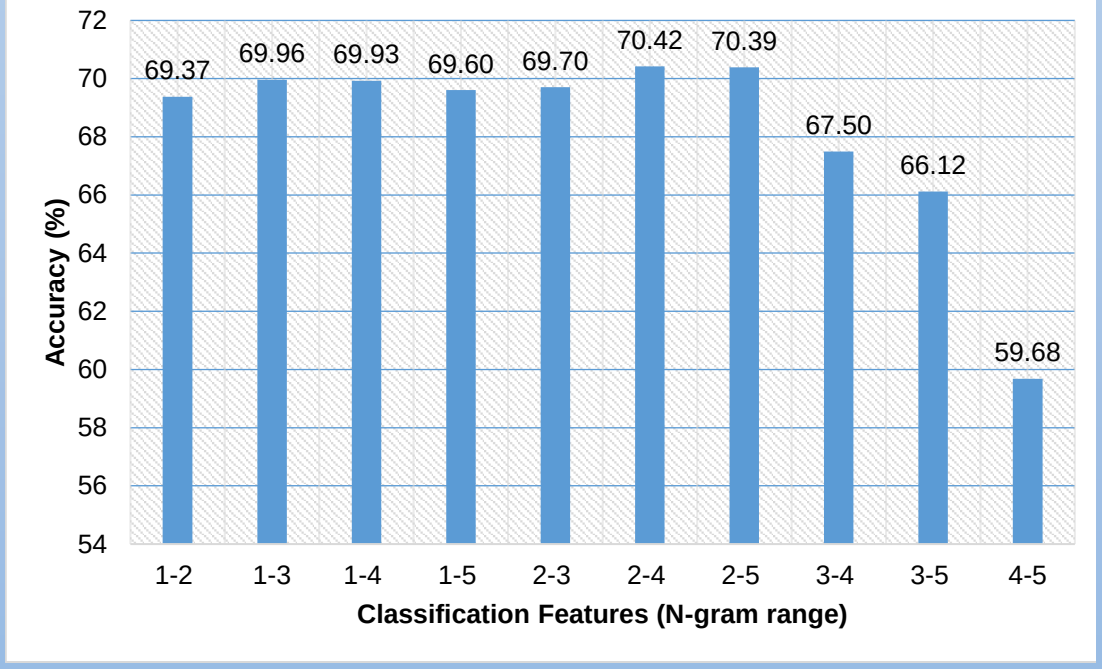


Figure 4.8: The sigmoid SVM accuracy using combination of features on a 10-fold cross validation

4.3.6 SVM with Polynomial Kernel

The polynomial SVM was evaluated and the accuracy was 51.36% on unigrams, and 68.22% on bigrams. The bigrams outperformed other *n*-grams with a difference of 0.39% for trigrams, 7.65% for 4-grams, and 20.01% for 5-grams as shown in Figure 4.9. The polynomial SVM was further evaluated on combination of *n*-gram. Most sets reached accuracy of above 68%, but the *n*-gram set 2-5 was the highest with accuracy of 70.72% so far (see Figure 4.10).

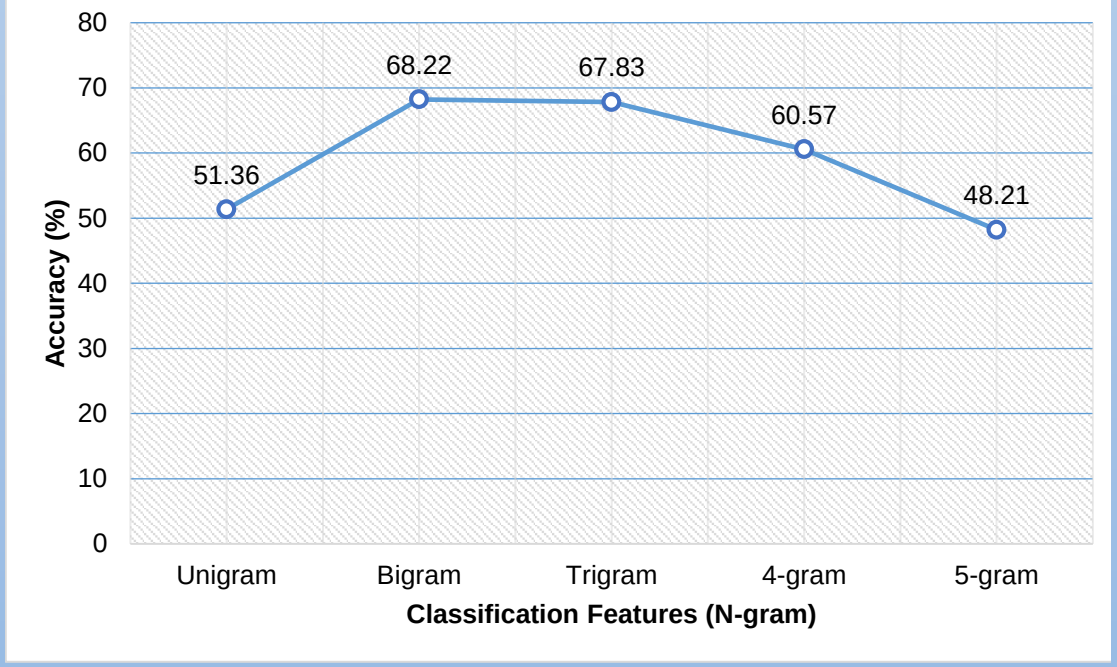


Figure 4.9: Accuracy of polynomial SVM on a 10-fold cross validation

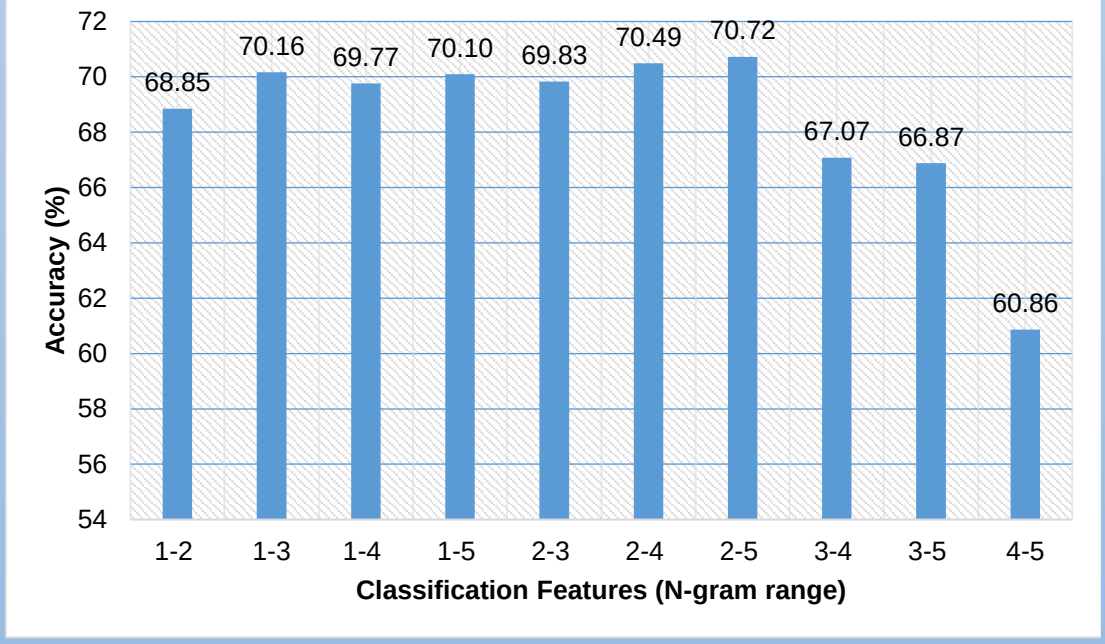


Figure 4.10: The polynomial SVM accuracy using combination of features on a 10-fold cross validation

4.3.7 Final Model

The cross-validation resampling method assists in selecting the best model on unseen data. The polynomial SVM outperformed other algorithms on unseen data. Thus, we further built the final model using the polynomial SVM on the complete dataset. Figure 4.13 shows the results of a polynomial SVM on a 10-fold cross-validation in terms of precision, recall and F-score measurements. Figure 4.14 shows the results of a polynomial SVM in terms of precision, recall and F-score measurements for the final model. The results of the final model were higher than that with cross-validation because the model is built on known data. On average, the precision of the final model increased from 71.30% to

81.20%. Recall increased from 70.70% to 80.80% while F-score increased from 70.70% to 80.80%.

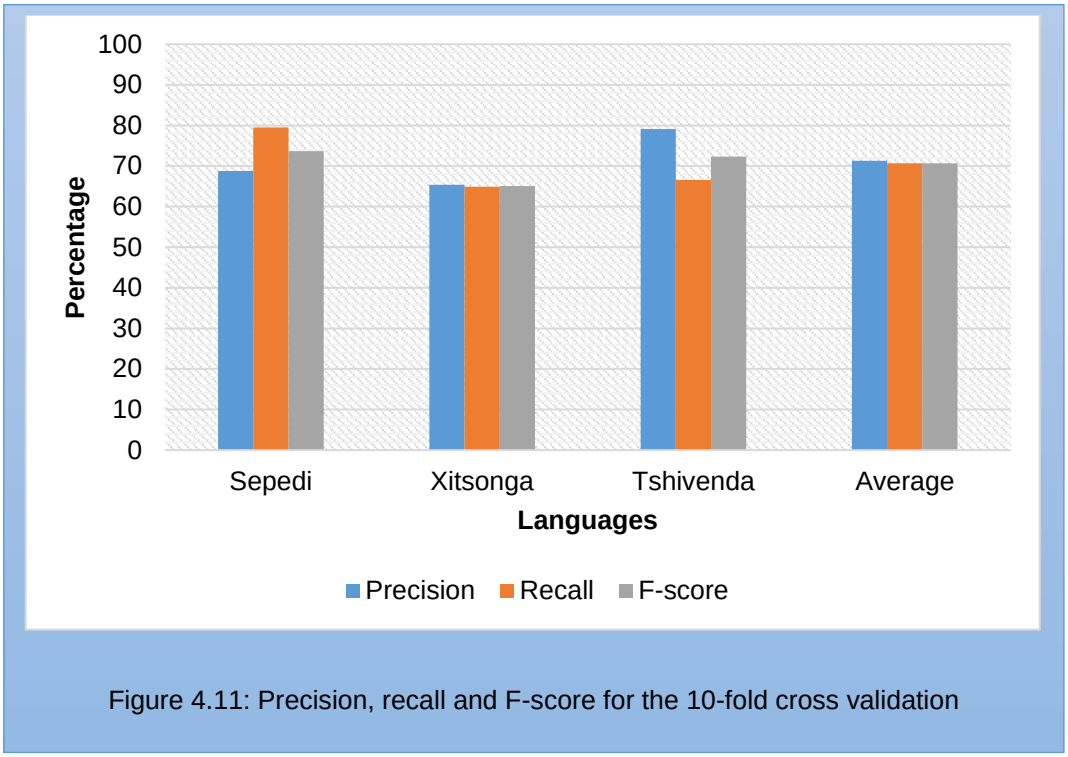


Figure 4.11: Precision, recall and F-score for the 10-fold cross validation

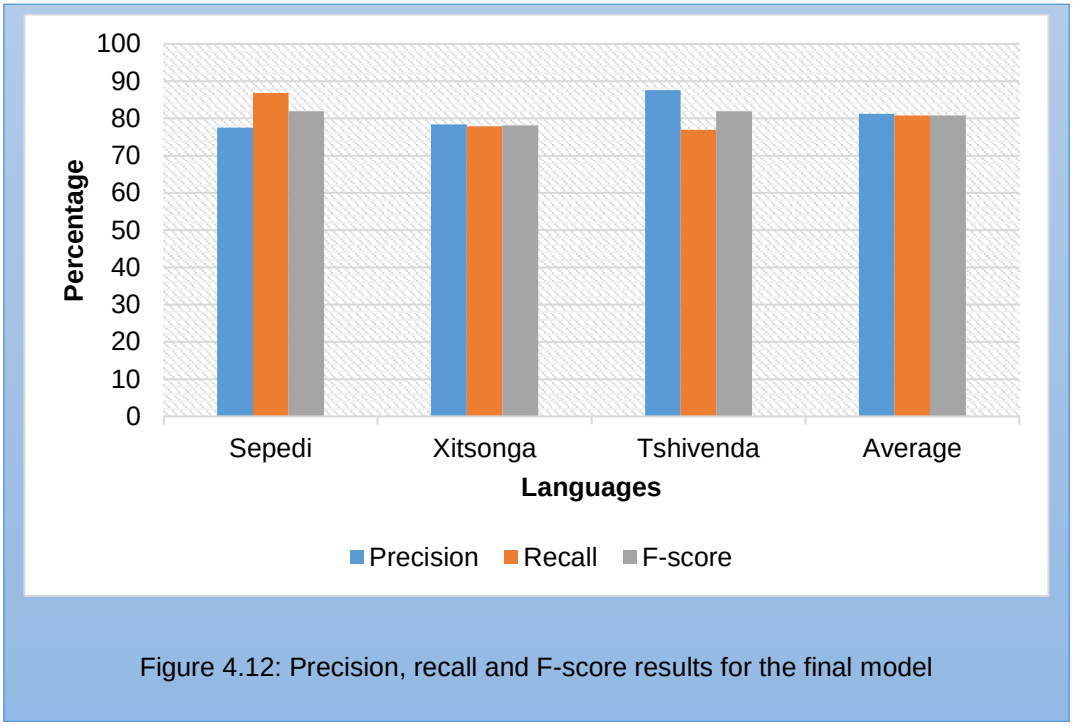


Figure 4.12: Precision, recall and F-score results for the final model

The final model resulted in an increase of accuracy from 70.72% to 80.77%. The errors of the final model decreased by 8.38% (see Figure 4.15). This accuracy was found to be reasonably adequate to make predictions of unknown dataset. As a result, this model was implemented on the prototype system for automatic LID feature.

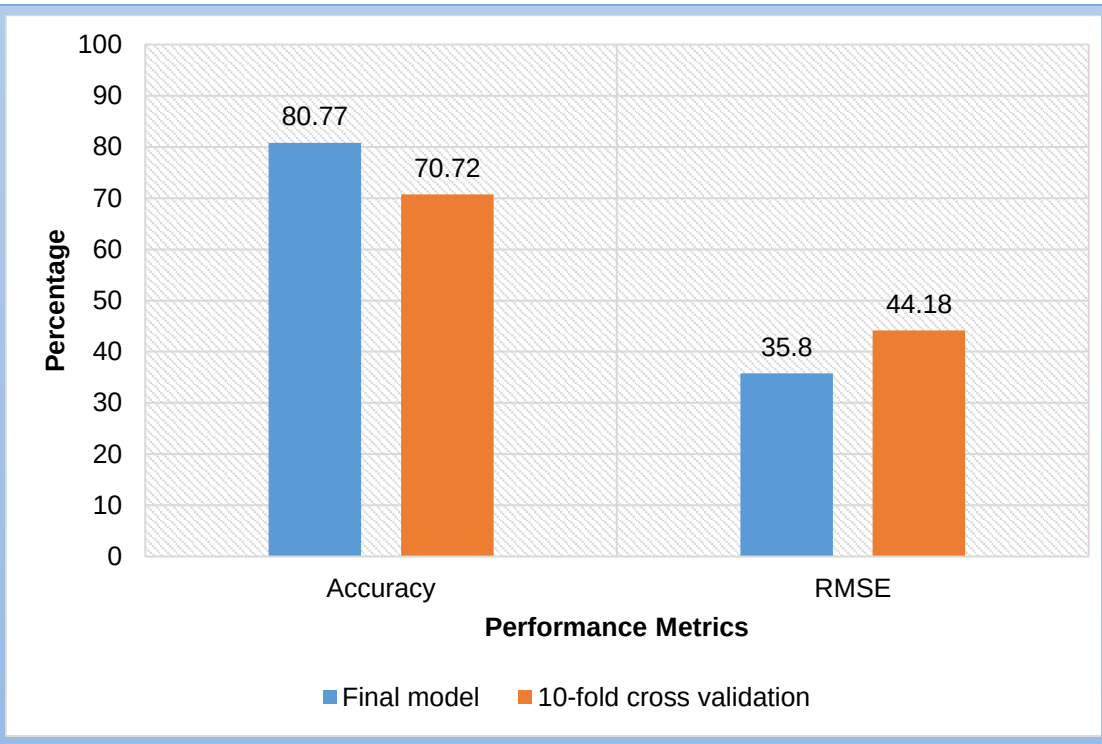


Figure 4.13: Accuracy and RMSE using polynomial SVM for final model and 10-fold cross validation

4.4 Evaluation Results and Analysis of the Developed TTS

The data collected during the evaluation of the system was analysed using descriptive statistics on a Microsoft Excel spreadsheet. The main aim of the TTS synthesis system is to answer the research question: Can a computational system produce an appropriate pronunciation of indigenous proper names?

4.4.1 Test for Intelligibility

Five sentences were constructed and eight evaluators performed the evaluation for intelligibility of each item of synthetic speech. A total of $5 \times 8 = 40$ sentences per language were used during the evaluation. The total number of words differed per language. The total number of words for Sepedi, Xitsonga, Tshivenda, and isiNdebele were 312, 240, 248, and 176 respectively. Figure 4.16 shows the results of SUS method at sentence level and word level. The SUS accuracy at word level is higher than those at sentence level. The SUS accuracy at word level is above 90% for all the languages, with isiNdebele outperforming other languages at an accuracy of 95%. These results are quite adequate for under-resourced languages. Hence, from Figure 4.16 we see that the developed system is quite intelligible.

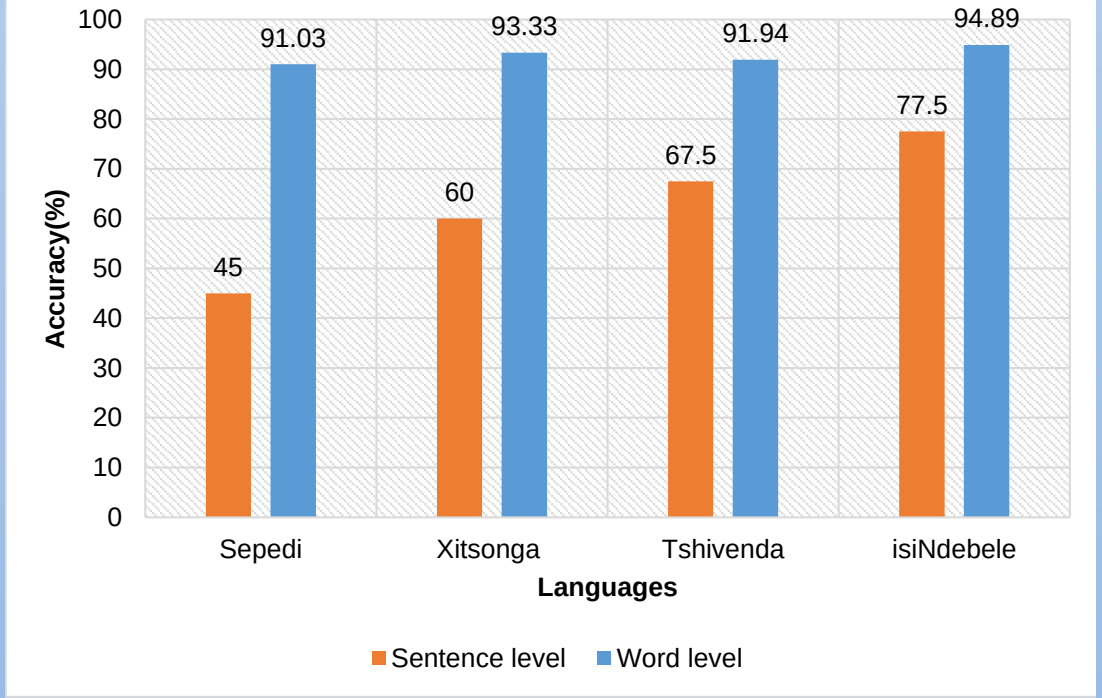


Figure 4.14: The SUS accuracy at sentence and word level for intelligibility of the developed system.

The script in Appendix K was used to calculate both sentence and word error rate. The SER is computed by counting the sentences which contain WER of above zero. We have conducted error rates on SUS. These error tests were conducted on SUS synthesised by the developed system and results are given in Figure 4.17. We see that error rates significantly decrease from sentence to word level. As such, WER shows errors found on each tested word. Sepedi contained a total of 312 words while Xitsonga, Tshivenda, and isiNdebele contained a total of 240, 248, and 176 words respectively. Sepedi obtained higher WER of 14.82% compared to other languages. This may be caused by the familiarity of the speaker with the language since the speaker's geographical location is populated with Setswana speaking people. IsiNdebele obtained good results on both SER and WER. From these results, we see that all the built synthetic voices are intelligible.

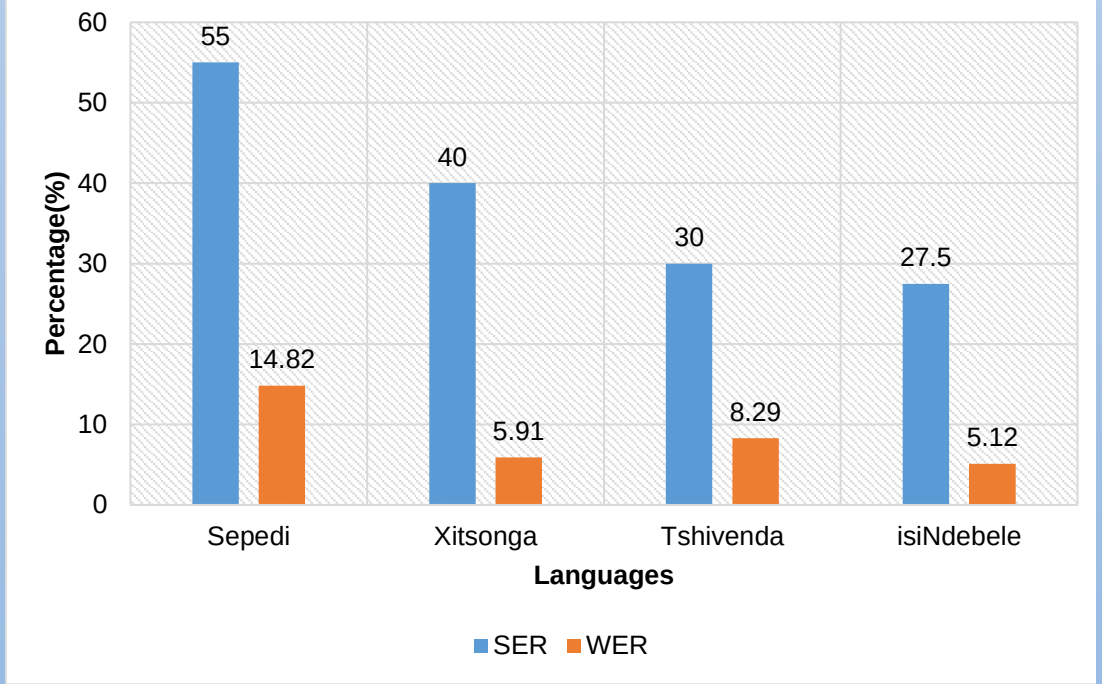


Figure 4.15: The SER and WER for intelligibility of the developed system.

4.4.2 Test for Naturalness

The system is evaluated for naturalness using the MOS test discussed in Section 4.2.2.3. The results in Figure 4.18 shows the synthesised speech compared to the natural speech. Therein, the isiNdebele language synthesised speech achieved higher MOS than other synthesised voices. The difference in MOS from the isiNdebele language synthesised speech to the isiNdebele natural speech is 0.1. This means that isiNdebele language synthesised speech is close to very natural sounding. The Sepedi language synthesised speech obtained the MOS difference of 1.0 from natural voice and that means the Sepedi synthesised voice was found to be more natural sounding. The Tshivenda language synthesised speech achieved a gap of 0.5 from natural speech and that means Tshivenda synthesised speech is found to be natural. The Xitsonga language synthesised speech obtained MOS score of 4.6 which is 0.4 away from natural speech. This means the Xitsonga synthesised speech is found to be natural. These results are found to be acceptable for these training datasets and test sentences.

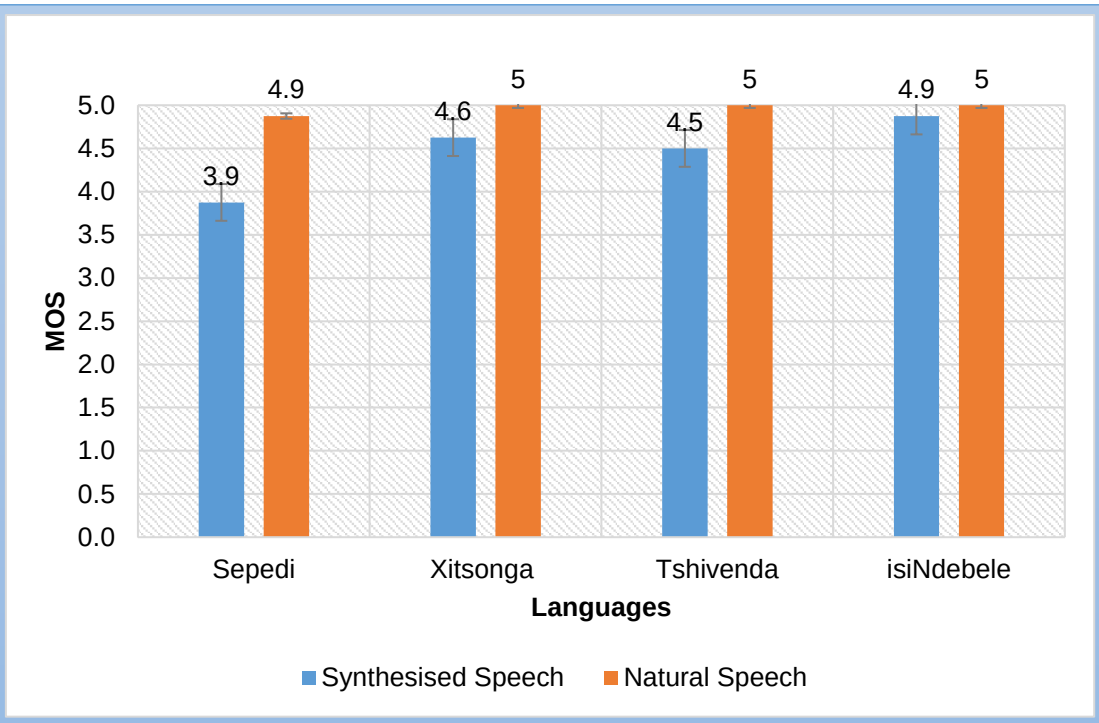


Figure 4.16: Results of test for naturalness of speech samples.

4.4.3 Test for Correct Pronunciation

The system was evaluated for correct pronunciation using the MOS test. Figure 4.19 shows the evaluation results of the synthesised speech and natural speech. The isiNdebele language synthesised speech obtained MOS score of 4.8 which is 0.2 away from natural voice. This means isiNdebele language synthesised speech pronounced words correctly. The Sepedi language synthesised speech obtained MOS score of 3.6 which is lower than the others. This means that some of the words were not pronounced the way evaluators expected. The Xitsonga and Tshivenda languages synthesised speech obtained at least MOS score of 4.3, which means that their pronunciation was found to be excellent.

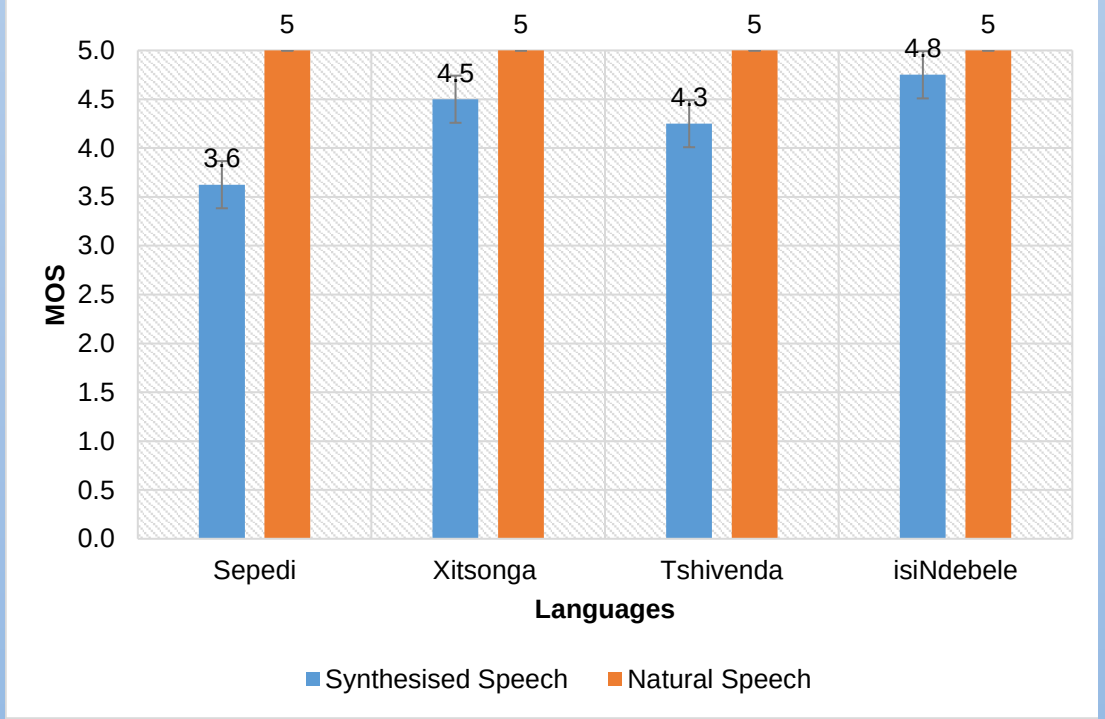


Figure 4.17: Results of test for pronunciation of speech samples.

4.4.4 Test for Pleasantness

The MOS test was used to evaluate pleasantness of the system. Figure 4.20 illustrates the comparison of the synthesised speech and natural speech. The isiNdebele language synthesised speech obtained MOS score of 4.9 which is 0.1 away from natural speech. This means the isiNdebele language synthesised speech was very pleasant. The Sepedi language synthesised speech obtained MOS score of 3.9 which means the speech was found to be pleasant to listen to. The Tshivenda and Xitsonga languages synthesised speech obtained an MOS score of above 4.3 which means the synthesised speech was found to be pleasant.

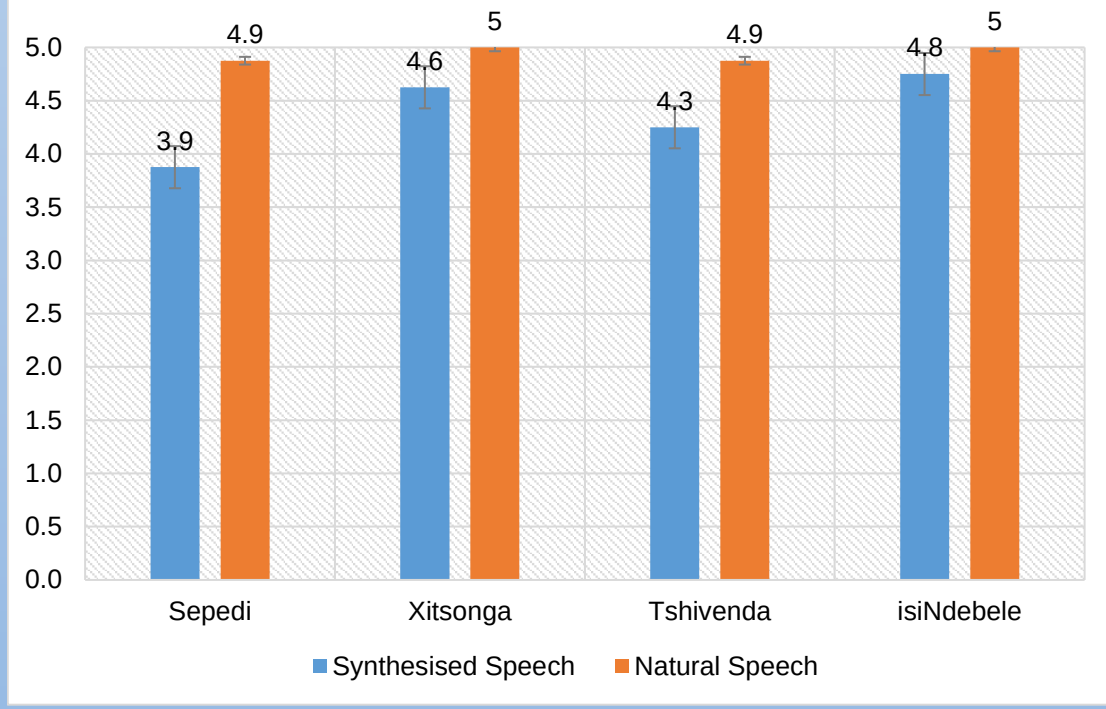


Figure 4.18: Results of test for pleasantness of speech samples.

4.4.5 Test for Understandability or Listening Effort

The system was evaluated for required listening effort needed to understand the synthesised speech (see Figure 4.21). The MOS test was used to test understandability with isiNdebele language obtaining MOS score of 4.9 which means that no effort was required to understand the synthesised speech. Sepedi, Tshivenda, and Xitsonga obtained MOS scores of 4.1, 4.3 and 4.4 respectively; this means that no appreciable effort was required to understand the synthesised speech.

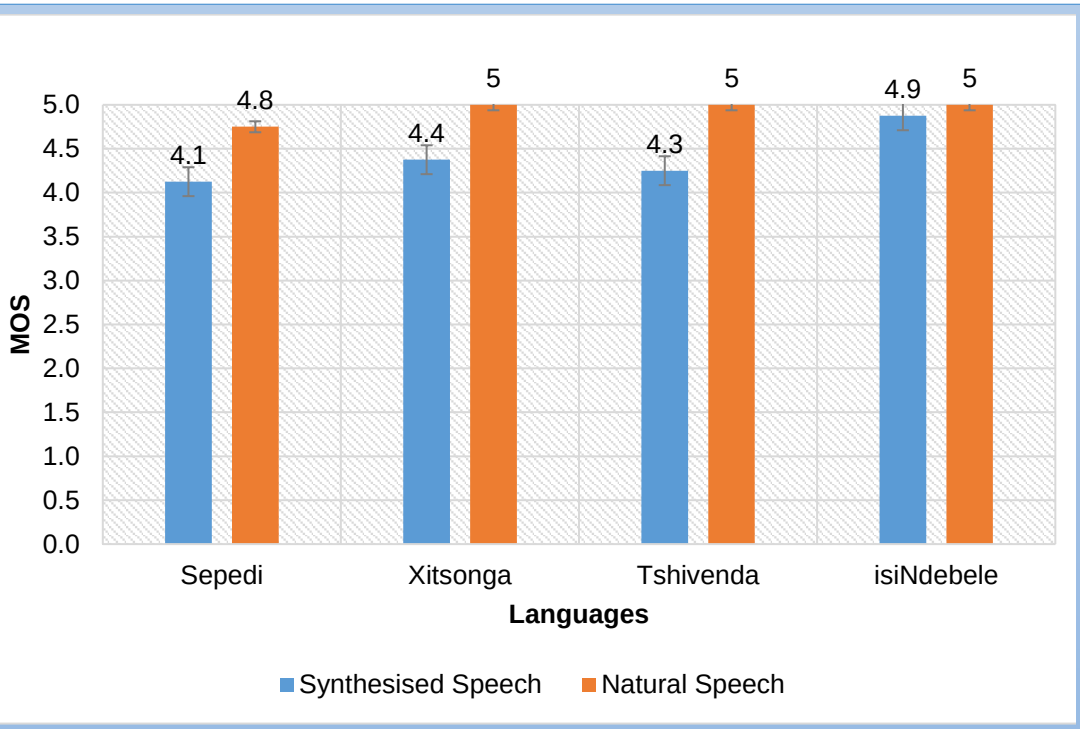


Figure 4.19: Results of test for understandability of speech samples.

4.4.6 Test for the Overall Quality

The quality for the overall system was evaluated using MOS test. Figure 4.22 shows the results of the overall quality between synthesised and natural speech. The Sepedi and Tshivenda synthesised speeches achieved MOS score of 4.1

and 4.3 respectively which means the speech quality was found to be acceptable. Tshivenda and isiNdebele synthesised speeches obtained MOS score of 4.6 and 4.8 respectively which means the speech quality was found to be excellent.

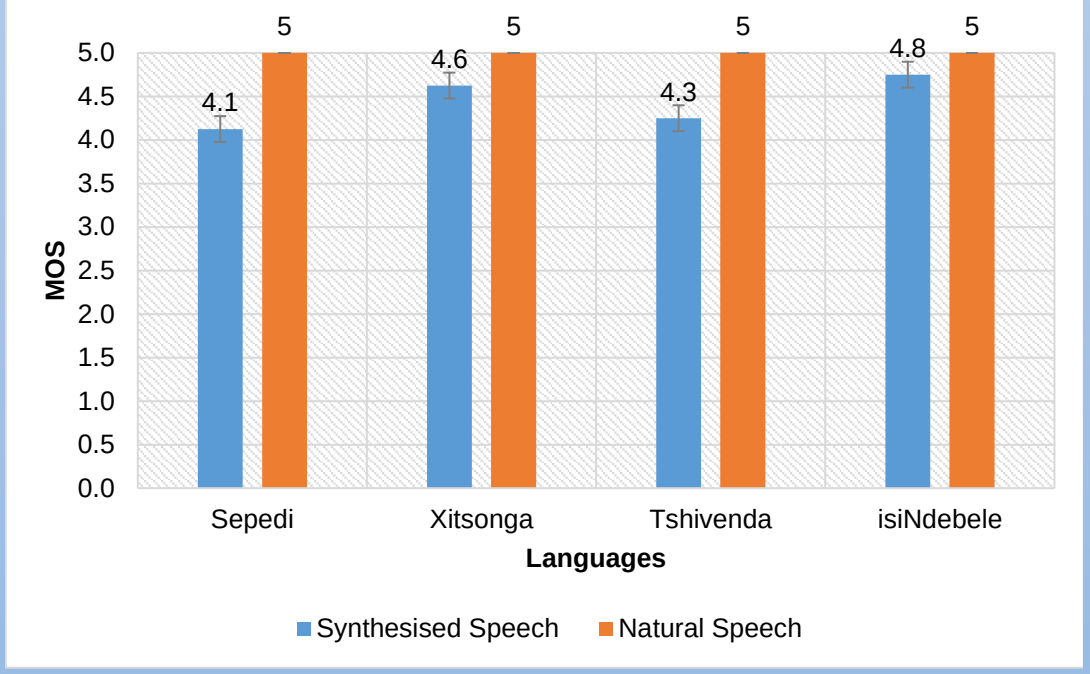


Figure 4.20: Results of test for overall quality of speech samples.

4.4.7 Comparison with other Studies

The Xitsonga TTS synthesis system developed by Baloyi (2012) was the first TTS developed for Xitsonga in South Africa. Figure 4.23 shows the evaluation results of Xitsonga TTS developed by Baloyi (2012) and our Xitsonga TTS results. We compared these systems because they both use HMMs as speech synthesiser for the same language. The gap in the MOSs from the synthetic speech to the natural speech decreased from 1.1 to 0.4 (64%) in naturalness. The MOS score for pleasantness decreased from 1.4 to 0.4 (71%), while MOS score for

understandability decreased from 1.1 to 0.6 (46%). The developed system reduced the gap between natural speech and synthesised speech by more than 46%.

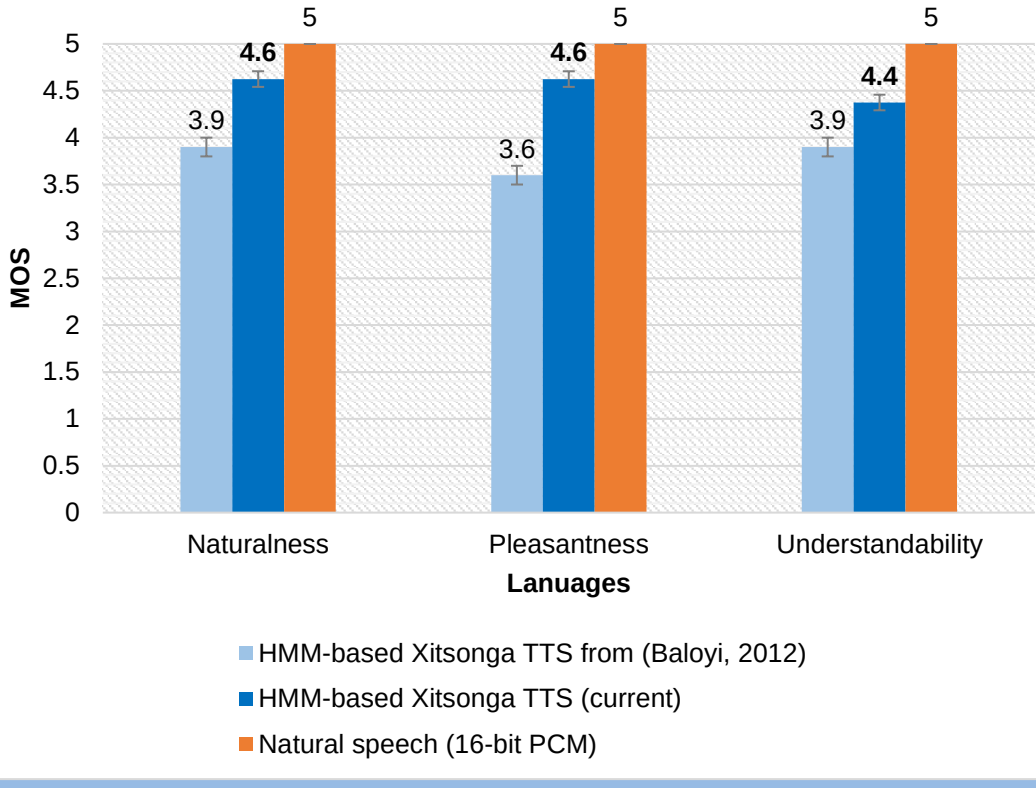


Figure 4.21: Subjective 5-scale MOS of Xitsonga TTS

4.5 Evaluation Results and Analysis of the Complete System Usability

The evaluators tested the functionality and usability of the developed system. The last section of the questionnaire in Appendix H contains the evaluation sheet in

terms of MOS test. After the evaluators familiarised themselves with the system on the website they rated the following review statements:

Statement 1: *The buttons are visible and easy to find.*

All the evaluators strongly agreed with this statement.

Statement 2: *Languages can be switched easily.*

All the evaluators strongly agreed with this statement.

Statement 3: *Text is visible and clear.*

All the evaluators strongly agreed with this statement.

Statement 4: *Layout and colours are displayed perfectly.*

All the evaluators strongly agreed with this statement.

Statement 5: *The application is difficult to understand*.

Most of the evaluators strongly disagreed with this statement, but 15% of them disagreed, with 33% (of 15%) coming from Xitsonga and Tshivenda speakers and 17% (of 15%) coming from isiNdebele and Sepedi speakers (see Figure 4.24).

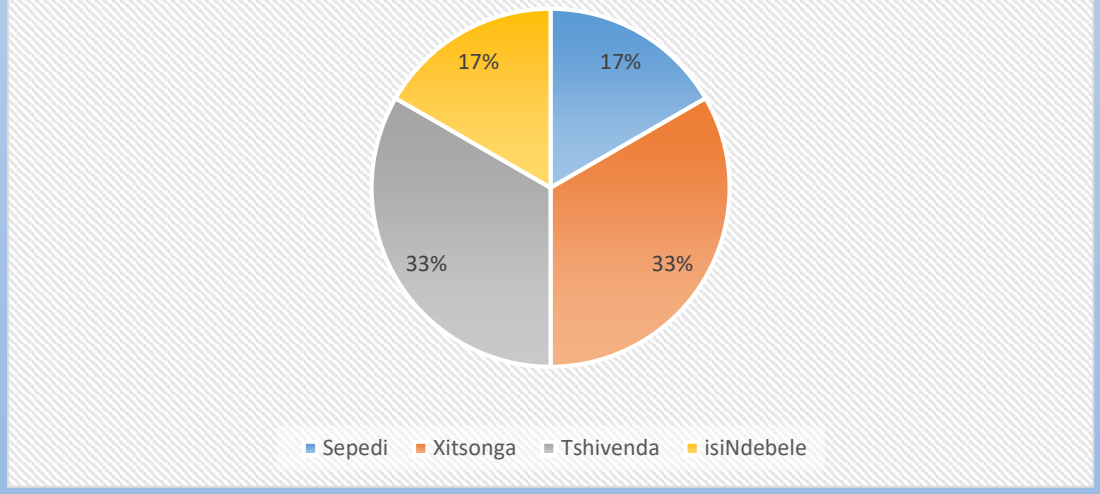


Figure 4.22: The fifteen percent of the evaluators disagreed that the application is difficult.

Statement 6: *I can use the application on my own.*

Most of the evaluators strongly agreed to use the application on their own but 15% of them agreed. Figure 4.25 shows the 15% of evaluators who agreed per language. Most speakers who agreed to use the application on their own were Tshivenda speakers.

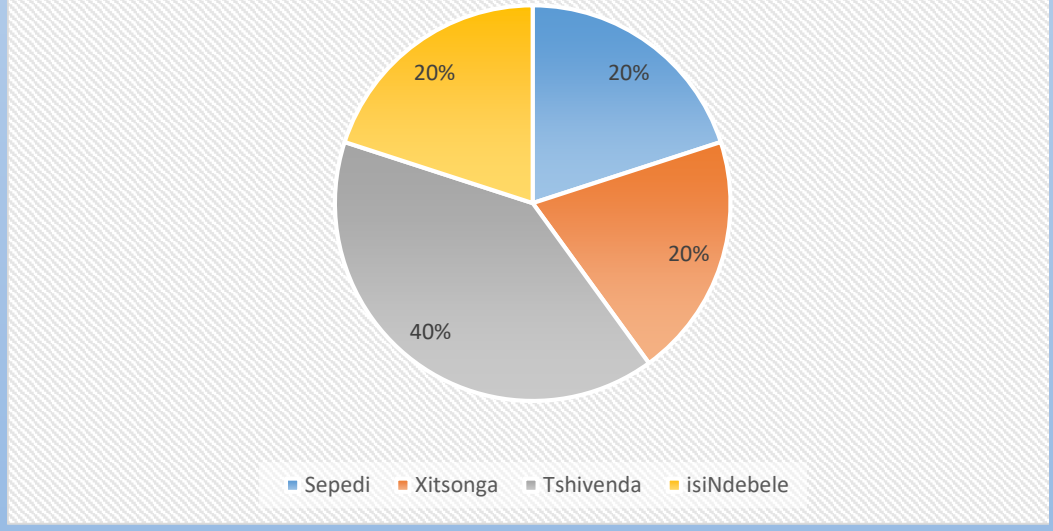


Figure 4.23: The fifteen percent of the evaluators agreed to use the system on their own

Statement 7: *I felt very confident using the application.*

All the evaluators strongly agreed with this statement.

Statement 8: *I would recommend this application to someone else.*

All the evaluators strongly agreed with this statement.

Statement 9: *I would frequently use this application.*

Most of the evaluators strongly agreed to use this application and only 9% of them gave neutral answers.

Statement 10: *This application can help me learn pronunciation of new languages.*

All the evaluators strongly agreed with this statement.

Statement 11: *Would you recommend these voices to be integrated in future devices?*

All the evaluators strongly agreed with this statement.

4.6 Summary

We have seen from the LID results that the higher the number of different languages or classes, the less the LID accuracy will be achievable. In other words, the number of classes is inversely proportional to classification accuracy. We have explained the LID results from various *n*-gram features. We observed that *n*-gram set of two to five obtained better results on polynomial SVM. The final model of the LID was built on the entire dataset. We have explained the evaluation metrics and procedure taken to evaluate the speech generation phase. Subjective perception listening tests were conducted using 32 students and obtained good results after applying MOS test. The usability of the system on the website was evaluated and good results were observed. The next chapter provides a conclusion, summary of our findings and recommendations for future work.

5 CHAPTER 5: DISCUSSIONS

This chapter explains the discussions of the results obtained in Chapter 4. The first section discusses the comparison of the classifiers. It compares the results of the classifiers on certain n-gram features. The second section discusses the speech synthesiser while the third section discusses system usability. And lastly, the last section discusses summary of the findings.

5.1 Classifier Model Comparison

We combine the results of the classifiers on 1-gram to 5-gram features in Figure 5.1 from Chapter 4. We observe the larger n-gram size decreases classification accuracy on both the MNB and SVM kernels. The unigrams obtained better accuracy of 61.12% on RBF SVM, compared to other algorithms. The bigrams obtained accuracy of 68.00% on average using all the classification algorithms. This shows that the bigrams were perfect to discriminate between the languages (classes) compared to the unigrams. The trigrams on MNB performed better than SVM with an accuracy of 69.34%. This shows that MNB classify well on trigrams compared to other n-gram sizes. The accuracy decreased on 4-gram and 5-gram for both MNB and SVM. This shows that trigrams were the turning point on our dataset

We combine the results of the classifiers in Figure 5.2 from Chapter 4. The combination of unigram and bigram resulted in accuracy of 69.37% on sigmoid SVM. We added trigrams, and polynomial SVM achieved accuracy of 69.96%. When 4-grams were added, the accuracy went up to 70.09% by RBF SVM. The 5-grams were added, but RBF SVM still obtained higher accuracy of 70.19%, and MNB performed low at 69.24%. The unigrams were removed and left with 2- to 5-grams. The MNB did not perform well. However, polynomial SVM produced the best results with 70.72% accuracy, which was found to be the highest result when compared to all other results.

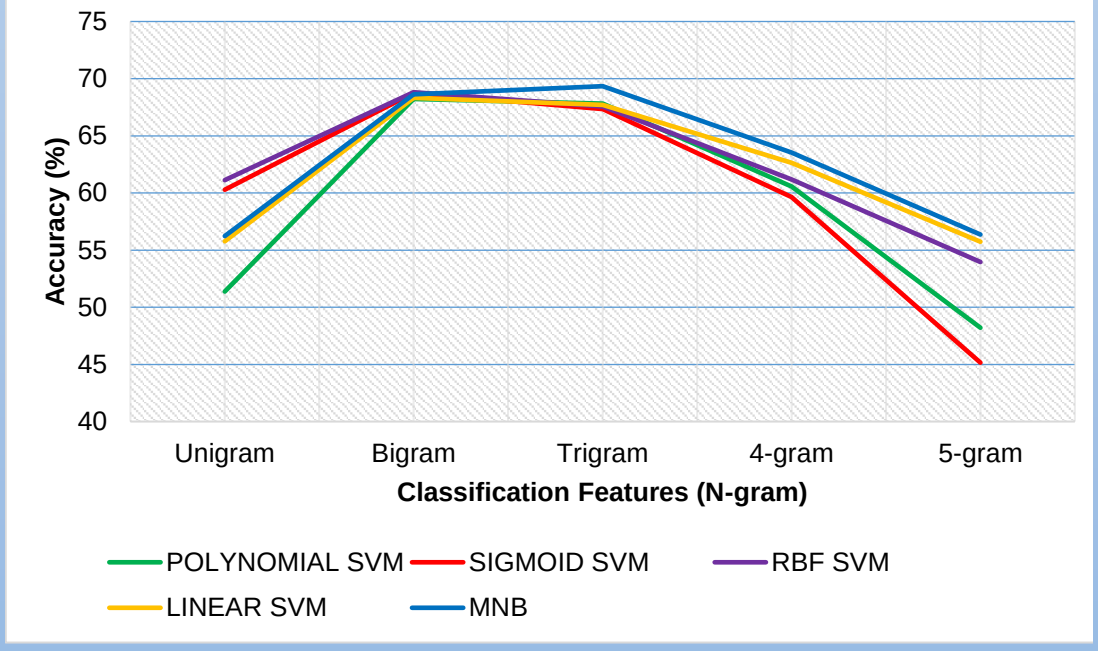


Figure 5.1: Accuracy comparison on a 10-fold cross validation

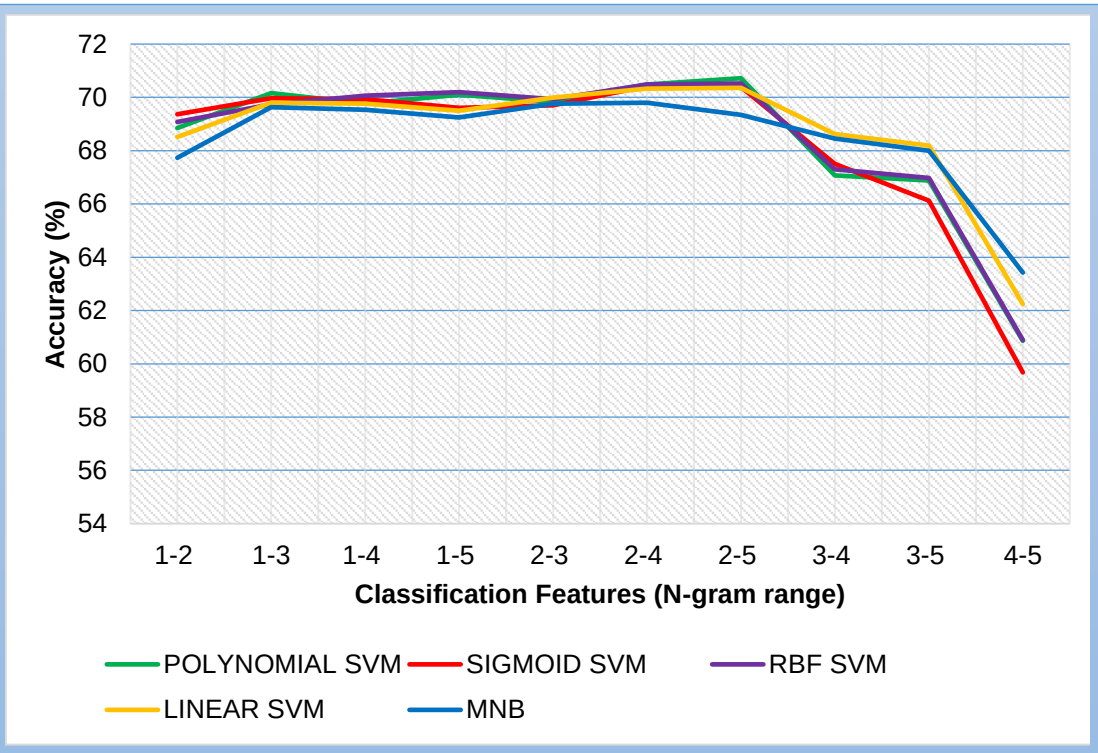


Figure 5.2: Accuracy comparison using combination of features on a 10-fold cross validation

5.2 Text-to-Speech Synthesiser

We can see from the results that the developed system obtained high accuracy from the SUS test at word level in Figure 4.16. This shows that the developed system is intelligible enough to synthesise new words without the context of the sentence. The WER rated below 10% for Xitsonga, Tshivenda and isiNdebele, but Sepedi was an outlier with 14.82% of errors in Figure 4.17. This outlier is caused by the accent of the Sepedi speaker with other language families of Sepedi. We nonetheless find that the results show that the system is intelligible enough to synthesise new words.

5.3 System Usability

We see from the results in Section 4.5 that most evaluators highly rated the user interface and functionality of the application moving from one language to another. All of the evaluators were confident after using the application and agreed that the application is not difficult. They were able to play a synthesised speech on their own without any help from someone. Some of the evaluators gave neutral answers regarding if they would frequently use the application. The evaluators agreed to that this application is helpful towards learning new languages. Therefore, this satisfies the last objective.

5.4 Summary of the Findings

As discussed in chapter three and four, it was found that all the research objectives were satisfied. We now try to answer the following research questions:

The first research question was formulated as follows: *Can a computational system use a person's surname to predict the identity the first language of that person?*

To answer this research question, we compared the performance of MNB to SVM kernels in the previous chapter. We showed that the features or *n*-grams are important factors that can affect classification accuracy of text-based LID. Four indigenous official languages of South Africa were employed for the study. These

languages belong to a number of closely related groups of languages. This complicated the classification between languages. Two machine-learning classifiers were contrasted in this study:

- The MNB, which obtained good results in previous studies of text classification.
- The SVMs, which are more robust and complex classifiers. These classifiers perform better when fitted with correct parameters.

Character *n*-grams were used as classification feature set, since the LID training dataset contains single words; hence, word *n*-grams were found unsuitable for this task. The *string-to-word* filter in WEKA was used to convert the dataset into feature vectors, since SVM do not support string attributes. The classifiers were deployed on unigrams, bigrams, trigrams, 4-grams and 5-grams. The trigrams yielded good results on both MNB and SVMs. We further expanded the feature set to contain a mixture of *n*-grams of different sizes and tested them on both classifiers. The MNB achieved an accuracy of 69.34% on *n*-gram sets of two to five grams, which acts as our baseline classifier. The polynomial SVM obtained higher classification results of 70.72% accuracy on *n*-gram sets of two to five grams. The final model was built on this feature set, using the polynomial SVM. The classification accuracy of the final model was above 80%. In addition, the classification errors decreased by 8.38%. This higher accuracy of 80% is enough for text-based LID of under-resourced languages of South Africa. Since polynomial SVM outperformed the baseline MNB (70.72% > 69.34%), we can significantly conclude with 70.72% accuracy that a computational system can reasonably use a person's surname to identify the first language of that person.

The second research question was formulated as follows: *Can a computational system produce an appropriate pronunciation of indigenous proper names?*

To answer this research question, we have developed the four baseline TTS synthesis systems which were evaluated on correct pronunciation, naturalness, pleasantness, understandability, intelligibility, and overall quality of synthesised speech. The subjective listening tests were conducted. The participants gave

good results based on their opinions. The MOS test was used to evaluate the system and good results were observed, which showed that the correct pronunciation was found to be excellent. With these high results, we can significantly conclude with a minimum MOS of 3.6 that a computational system can produce an appropriate pronunciation of indigenous proper names.

Therefore, the developed pronunciation assistant can help reduce the surname pronunciation problem for the indigenous official languages of South Africa as experienced by non-native speakers.

6 CHAPTER 6: CONCLUSIONS

6.1 Introduction

This chapter summarises the conducted experimental research including the limitations, scientific contribution, future work, and research questions. The layout is as follows:

- Section 6.2 discusses research limitations and challenges.
- Section 6.3 discusses some of the contributions of this research work to the scientific world.
- Section 6.4 provides recommendations and future work directions.

6.2 Limitations and Challenges

The study was conducted on a 32-bit Ubuntu desktop with two gigabytes of RAM. The deep neural network techniques were not used because they require massive computation resources. *The two gigabytes RAM and 32-bit processor was not enough to implement deep neural networks.* The search for SVM parameters was limited to 10 because the higher numbers resulted with a computer not responding.

We have encountered *challenges when creating the isiNdebele and Xitsonga synthetic voices*. The MARY TTS transcription aligner uses a pipe "|" character to align phones. Since our phone sets contained pipe character, the transcription tool produced an error while compiling a new voice. The solution was to use an alternative notation to phones that contain a pipe character (see Appendix B for isiNdebele and Xitsonga phone set).

The isiNdebele LID feature was not implemented due to inadequate data which could cause the results to be biased. However, isiNdebele TTS synthesis was implemented successfully.

We encountered challenges when recruiting people to participate and we used maximum number of 32 people who agreed to participate. Hence, we evaluated

each system with groups of 8 people (8x4 systems). Similarly, 10 people were used in the studies by Gahlawat *et al*. (2014) and Dagba and Boco (2014). Therefore, it is sufficient to obtain good results when evaluating a speech synthesiser with this number of participants.

6.3 Contributions of the Study

6.3.1 Language-specific Applications

- The developed LID can be used in any language-specific system to classify surnames.
- The developed TTS synthesis system can be embedded in any language-specific applications to provide synthesised speech.

6.3.2 Pre-processing Files

In the initial phase, when pre-processing was performed on the collected data, a few scripts were developed that can help other researchers. Examples are:

- A bash script to prepare and install required programs for the MARY TTS synthesis system (see Appendix A).
- A python script to generate a pronunciation dictionary compatible with MARY TTS is given in Appendix C.
- Java file for patching MARY TTS to support LID is given in Appendix D.
- The source code of the Android application is given in Appendix E.
- A python script to calculate WER and SER is given in Appendix K.

6.3.3 Importance of the Developed System

- The system was deployed on the internet for further evaluations on “real-world” data. Moreover, this system may be used by anyone ranging from visually impaired people to students in schools.
- The system may help people learning pronunciation of surnames in Sepedi, Xitsonga, Tshivenda, and isiNdebele.

- The system may provide a platform for supporting new research in the field of ICT for language learning and teaching of South African official languages.
- A detailed API of the developed system is given in the Methodology chapter. The API may be utilised by any researcher to use the functions of the developed system.
- A website of the system was deployed and may be used in any area of speech technology or educational technologies.

6.3.4 Speech Synthesis Results

The Xitsonga synthetic voice was compared to the HMM-based Xitsonga TTS synthesis system developed by Baloyi (2012). We have observed that our system improved naturalness, pleasantness, and understandability by 64%, 71% and 46%. These high results are found to be excellent for these under-resourced languages and their training data.

6.3.5 Common Dataset

Currently there are no common datasets for comparing text-based LID performance on under-resourced languages.

6.4 Future Work and Recommendations

6.4.1 Machine-learning Phase

The observed LID accuracies can be increased by deploying other sophisticated machine-learning algorithms such as DNN (van den Oord *et al.*, 2016). In addition, the training data for text-based LID can be increased to cover most surnames.

This study can be further expanded by:

- covering classification of hyphenated (or so-called double-barrelled) surnames (e.g. Sefara-Dzambukeri);
- including classification of multilingual surnames (e.g. Baloyi);
- including both first name and last name (e.g. Mathapelo Alice Sefara);
- including classification of words in general sentences.

The isiNdebele LID feature can be implemented by further obtaining enough textual corpus.

6.4.2 Speech Synthesis Phase

- The text analysis phase can be further enhanced to normalise other classes of input data elements such as numbers, currency, money and other non-standard words to solve normalisation problems. This phase was not fully implemented since the study is focused on surnames.
- The EHMM labeller was used to automatically label phones to their corresponding utterances. Hand-labelling can be used to manually verify labels even though it requires much linguistic expertise.
- Incorporating prosody into the system can be optimised to increase the quality of synthesised speech.
- The SPSS systems suffer from poor quality caused by the inadequacy of acoustic modelling (e.g. trajectory HMM), limitations of the vocoder (e.g. STRAIGHT), and over-smoothing of parameter generation (e.g. global variance). These can be enhanced by applying recent advanced deep learning algorithms to replace HMMs.
- Additional under-resourced languages can be included to expand the coverage.
- Sepedi synthetic voice performed low compared to other languages and this can be improved by using a professional speaker who is fluent in Sepedi.
- The evaluation of the speech synthesiser can further be evaluated with more number of people and more number of sentences.

6.5 Final Remarks

This study presented an automatic pronunciation assistant system that implemented machine-learning algorithm and speech synthesis in Sepedi, Tshivenda, Xitsonga and isiNdebele. The automatic pronunciation assistant synthesised speech that is rich in quality. A HMM-based method was used for implementation of the developed system. Although it is commonly known that this method does not produce high quality synthesised speech as compared to unit selection-based systems, this method is very flexible, efficient, and requires less training data. Furthermore, this method offers a room for adaptation and development of TTS voices from under-resourced languages.

LIST OF PUBLICATIONS

Sefara, T.J., Manamela, M.J. & Modipa, T.I., 2017. Web-based automatic pronunciation assistant. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*. Barcelona, pp. 112-117.

Mokgonyane, T.B., **Sefara, T.J.**, Manamela, P.J., Manamela, M.J. & Modipa, T.I., 2017. Development of a speech-enabled basic arithmetic m-learning application for foundation phase learners, In *2017 IEEE AFRICON*, Cape Town, pp. 794-799.

Malatji, P.T., Manamela, M.J. & **Sefara, T.J.,** 2017. Second language learning through accented synthetic voices. In *South Africa International Conference on Educational Technologies (SAICET)*. Pretoria, AARF, pp. 106-116.

Sefara, T.J., Manamela, M.J. & Malatji, P.T., 2016. Text-based language identification for some of the under-resourced languages of South Africa. In *3rd International Conference on Advances in Computing and Communication Engineering (ICACCE-2016)*. Durban, IEEE, pp. 303-307.

Sefara, T.J. & Manamela, J.M., 2016. The development of local synthetic voices for an automatic pronunciation assistant. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*. George, pp. 142-146.

Mokgonyane, T.B., **Sefara, T.J.,** & Manamela, M.J., 2016. Speech-enabled applications for foundation phase. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*. Work-in-progress, George, pp. 94-95.

Sefara, T.J., Manamela, M.J. & Malatji, P.T., 2016. Applying Speech Synthesis to Basic Mathematics as a Language. In *South Africa International Conference on Educational Technologies (SAICET)*. Pretoria, AARF, pp. 243-253.

Malatji, P.T., Manamela, M.J. & **Sefara, T.J.,** 2016. Creating Accented Text-To-Speech English Voices to Facilitate Second Language Learning. In *South Africa International Conference on Educational Technologies (SAICET)*. Pretoria, AARF, pp. 234-242.

Sefara, T.J., & Manamela, M.J., 2015. Towards Development of an Automatic Pronunciation Assistant. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*. Work-in-progress, Hermanus, pp. 15-16.

APPENDIX A: INSTALLATION GUIDE AND PATH VARIABLES – install.sh

The following script was used to install all packages and libraries on the terminal. More installation details are provided by software vendor and can also be found on the world wide web. Ubuntu 14.04 LTS 32-bit was used for this research project. All the software packages were downloaded and saved to the directory */voice/sources/*. The installations provided below are mainly guided by the instructions on MARY TTS voice import tutorial on GitHub[1].

```
#!/bin/bash
sudo apt-get update
sudo mkdir /voice
sudo chown -R $USER /voice/
sudo apt-get install build-essential git mc libc6-dev libx11-dev
libncurses5-dev sox tcl-snack g++ python3-dev openjdk-8-jdk
#Apache-maven
cd /voice/sources
tar xf apache-maven-3.3.9-bin.tar.gz -C /voice/soft/
mv /voice/soft/apache-maven-3.3.9 /voice/soft/maven
#HTK, HDecode and HTS
tar xf HTK-3.4.1.tar.gz
tar xf HDecode-3.4.1.tar.gz
mkdir hts
tar xf HTS-2.2_for_HTK-3.4.1.tar.bz2 -C hts
cd htk
patch -p1 -d . < ../hts/HTS-2.2_for_HTK-3.4.1.patch
./configure --prefix=/voice/soft/hts
make all hdecode
make install install-hdecode
cd ../
#HTS engine
tar xf hts_engine_API-1.05.tar.gz
cd hts_engine_API-1.05
./configure --prefix = /voice/soft/hts_engine
make
make install
cd ../
#Edinburgh Speech tools
tar xf speech_tools-2.4-release.tar.gz
cd speech_tools/
./configure
make
make test
cd ../
#Festvox
tar xf festvox-2.7.0-release.tar.gz
cd festvox/
./configure
make
cd ../
#Festival
tar xf festival-2.4-release.tar.gz
cd festival/
```

[1] Available at: https://github.com/marytts/marytts/wiki/VoiceImportToolsTutorial

```
./configure
make
cd ../
#SPTK
tar xf SPTK-3.6.tar.gz
cd SPTK-3.6/
./configure --prefix=/voice/soft/SPTK
make
make install
cd ../
#Praat
mkdir /voice/soft/praat
tar xf praat6014_linux32.tar.gz -C /voice/soft/praat/
# Path environment variables: alternatively these variables can be appended to
# the .bashrc file located at /home/$USER/.bashrc
export HTSDIR=/voice/soft/hts
export FESTVOXDIR=/voice/sources/festvox
export FESTIVALDIR=/voice/sources/festival
export ESTDIR=/voice/sources/speech_tools
export SPTKDIR=/voice/soft/SPTK
export HTSEngine=/voice/soft/hts_engine
export MAVEN=/voice/soft/maven
export PATH=$PATH:$ESTDIR/bin:$ESTDIR/include:$ESTDIR/lib:$FESTVOXDIR:$HTSDIR/bin
       :$HTSEngine/bin:$ESTDIR/main:$FESTIVALDIR/bin:$SPTKDIR/bin:$MAVEN/bin

#Mary TTS and EHMM labeller
git config global http.postBuffer 2M
git clone https://github.com/marytts/marytts.git
#Alternatively, if the above command clones different version of MARY TTS
#then the source code for MARY TTS version 5.2 can be manually
#downloaded and extracted from GitHub at web address:
#https://github.com/marytts/marytts/archive/v5.2.tar.gz
cd marytts/lib/external/ehmm
make
export EHMM=/voice/sources/marytts/lib/external/ehmm/bin
cd ../
./check_install_external_programs.sh check
#Results: should display OK.
cd ../../
mvn install
#Results: A new folder called target was created, it contains the MARY TTS client and
voice builder located at /voice/sources/marytts/target/.

export MARYTTS=/voice/sources/marytts/target/marytts-builder-5.2-SNAPSHOT/bin
```

APPENDIX B: PHONE SET FILES

B1: Sepedi phone set – allophone.nso.xml

```
<allophones name="sampa" xml:lang="nso"
    features="vlng vheight vfront vrnd ctype cplace cvox casp">

    <silence ph="_"/>
    <!-- Vowels  -->
    <vowel ph="i" vlng="s" vheight="1" vfront="1" vrnd="-"/> <!-- dira  -->
    <vowel ph="u" vlng="s" vheight="1" vfront="3" vrnd="+"/> <!-- Pula  -->
    <vowel ph="I" vlng="s" vheight="2" vfront="2" vrnd="-"/> <!-- tsela  -->
    <vowel ph="E" vlng="s" vheight="3" vfront="1" vrnd="-"/> <!-- rema  -->
    <vowel ph="O" vlng="s" vheight="3" vfront="3" vrnd="+"/> <!-- poo  -->
    <vowel ph="a" vlng="s" vheight="4" vfront="1" vrnd="-"/> <!-- rata  -->
    <vowel ph="U" vlng="s" vheight="2" vfront="2" vrnd="+"/> <!-- motho  -->

    <!-- Stop consonants -->
    <consonant ph="p_h" ctype="s" cplace="l" cvox="0" casp="+"/> <!-- phela  -->
    <consonant ph="p_>" ctype="s" cplace="l" cvox="0"/> <!-- pela  -->
    <consonant ph="t_h" ctype="s" cplace="a" cvox="0" casp="+"/> <!-- thapo  -->
    <consonant ph="t_>" ctype="s" cplace="a" cvox="0"/> <!-- topa  -->
    <consonant ph="k_h" ctype="s" cplace="v" cvox="-" casp="+"/> <!-- Khudu  -->
    <consonant ph="k_>" ctype="s" cplace="v" cvox="0"/> <!-- kudu  -->
    <consonant ph="tl_>" ctype="s" cplace="a" cvox="-"/> <!-- tla  -->
    <consonant ph="tl_h" ctype="s" cplace="a" cvox="-" casp="+"/><!--tlhabo -->

    <!-- Affricative consonants -->
    <consonant ph="tS_>" ctype="a" cplace="a" cvox="0"/> <!-- tšaka  -->
    <consonant ph="tS_h" ctype="a" cplace="a" cvox="0" casp="+"/><!--tšhaba-->
    <consonant ph="ts_>" ctype="a" cplace="a" cvox="0"/> <!-- tsela  -->
    <consonant ph="ts_h" ctype="a" cplace="a" cvox="0" casp="+"/><!--tshadi-->
    <consonant ph="kx" ctype="a" cplace="v" cvox="-"/> <!-- kgama  -->
    <consonant ph="ps_>" ctype="a" cplace="a" cvox="-"/> <!-- psila  -->
    <consonant ph="ps_h" ctype="a" cplace="a" cvox="0" casp="+"/><!--pshio-->
    <consonant ph="pS_h" ctype="a" cplace="p" cvox="0" casp="+"/> <!-- pšhatla-->
    <consonant ph="d_0Z" ctype="a" cplace="a" cvox="+"/> <!-- ja  -->

    <!-- Fricative consonants -->
    <consonant ph="f" ctype="f" cplace="b" cvox="-"/> <!-- fofa  -->
    <consonant ph="s" ctype="f" cplace="a" cvox="-"/> <!-- sesadi  -->
    <consonant ph="S" ctype="f" cplace="a" cvox="-"/> <!-- šala  -->
    <consonant ph="h" ctype="f" cplace="g" cvox="-"/> <!-- hema  -->
    <consonant ph="h\" ctype="f" cplace="g" cvox="+"/> <!-- lehodu  -->
    <consonant ph="K" ctype="f" cplace="a" cvox="-"/> <!-- hlaba  -->
    <consonant ph="G" ctype="f" cplace="v" cvox="+"/> <!-- goga  -->
    <consonant ph="B" ctype="f" cplace="l" cvox="+"/> <!-- baba  -->
    <consonant ph="p\s" ctype="f" cplace="l" cvox="0"/> <!-- lefsifsi  -->
    <consonant ph="p\S" ctype="f" cplace="l" cvox="-"/> <!-- Bofša  -->
    <consonant ph="BZ" ctype="f" cplace="l" cvox="+"/> <!-- bjala  -->

    <!-- Nasal consonants -->
    <consonant ph="m" ctype="n" cplace="l" cvox="0"/> <!-- ema  -->
    <consonant ph="n" ctype="n" cplace="a" cvox="0"/> <!-- nama  -->
    <consonant ph="J" ctype="n" cplace="p" cvox="0"/> <!-- nyaka  -->
    <consonant ph="N" ctype="n" cplace="v" cvox="0"/> <!-- ngwana  -->
    <consonant ph="m_j" ctype="n" cplace="b" cvox="0"/> <!-- myemyela  -->

    <!-- Approximant consonants (semivowels) -->
    <consonant ph="r" ctype="r" cplace="a" cvox="0"/> <!-- rata  -->
    <consonant ph="j" ctype="r" cplace="p" cvox="0"/> <!-- ya  -->
    <consonant ph="w" ctype="r" cplace="v" cvox="+"/> <!-- wa  -->
```

```
    <!-- Liquid consonants -->
    <consonant ph="l`" ctype="l" cplace="a" cvox="0"/> <!-- dira  -->
    <consonant ph="l" ctype="l" cplace="a" cvox="0"/> <!-- lala  -->
</allophones>
```

B2: Tshivenda phone set – allophone.ven.xml

```
<allophones name="sampa" xml:lang="ven"
    features="vlng vheight vfront vrnd ctype cplace cvox casp cpal">

    <silence ph="_"/>
    <!-- Vowels  -->
    <vowel ph="i" vlng="s" vheight="1" vfront="1" vrnd="-"/> <!-- lisa -->
    <vowel ph="u" vlng="s" vheight="1" vfront="3" vrnd="+"/> <!-- luma -->
    <vowel ph="E" vlng="s" vheight="3" vfront="1" vrnd="-"/> <!-- rema -->
    <vowel ph="O" vlng="s" vheight="3" vfront="3" vrnd="+"/> <!-- mboho -->
    <vowel ph="a" vlng="s" vheight="4" vfront="1" vrnd="-"/> <!-- rafha -->

    <!-- Stop consonants -->
    <consonant ph="p_h" ctype="s" cplace="l" cvox="0" casp="+"/><!--phala-->
    <consonant ph="p_>" ctype="s" cplace="l" cvox="0" cpal="+"/><!--panda-->
    <consonant ph="b" ctype="s" cplace="l" cvox="+"/> <!-- bako  -->
    <consonant ph="t_h" ctype="s" cplace="a" cvox="0" casp="+"/><!--hula-->
    <consonant ph="t_>" ctype="s" cplace="a" cvox="0"/> <!-- tafuna -->
    <consonant ph="d" ctype="s" cplace="a" cvox="+"/> <!-- daha -->
    <consonant ph="J\" ctype="s" cplace="p" cvox="+" casp="+"/><!--dyelo -->
    <consonant ph="k_h" ctype="s" cplace="v" cvox="-" casp="+"/> <!-- khuhu  -->
    <consonant ph="k_>" ctype="s" cplace="v" cvox="0"/> <!-- kokodza -->
    <consonant ph="g" ctype="s" cplace="v" cvox="+"/> <!-- goga  -->

    <!-- Affricative consonants -->
    <consonant ph="tS_h" ctype="a" cplace="a" cvox="0" casp="+"/><!--mutshila-->
    <consonant ph="ts_>" ctype="a" cplace="a" cvox="0"/> <!-- kutsimu -->
    <consonant ph="d_0Z" ctype="a" cplace="a" cvox="+"/> <!-- dzhena  -->
    <consonant ph="dz" ctype="a" cplace="a" cvox="+"/> <!-- dzembe-->

    <!-- Fricative consonants -->
    <consonant ph="f" ctype="f" cplace="b" cvox="-"/> <!-- fana  -->
    <consonant ph="v" ctype="f" cplace="b" cvox="+"/> <!-- vili  -->
    <consonant ph="s" ctype="f" cplace="a" cvox="-"/> <!-- sala  -->
    <consonant ph="z" ctype="f" cplace="a" cvox="+"/> <!-- zazamela  -->
    <consonant ph="S" ctype="f" cplace="a" cvox="-"/> <!-- shavha  -->
    <consonant ph="Z" ctype="f" cplace="a" cvox="+"/> <!-- zhaka  -->
    <consonant ph="x" ctype="f" cplace="v" cvox="-"/> <!-- xa  -->
    <consonant ph="h\" ctype="f" cplace="g" cvox="+"/> <!-- hana  -->
    <consonant ph="B" ctype="f" cplace="l" cvox="+"/> <!-- vhavha  -->
    <consonant ph="p\" ctype="f" cplace="l" cvox="-"/> <!-- fhala  -->
    <consonant ph="sw" ctype="f" cplace="a" cvox="-"/> <!-- swara  -->
    <consonant ph="zw" ctype="f" cplace="a" cvox="+"/> <!-- zwifha  -->

    <!-- Nasal consonants -->
    <consonant ph="m" ctype="n" cplace="l" cvox="0"/> <!-- ima  -->
    <consonant ph="n" ctype="n" cplace="a" cvox="0"/> <!-- nona  -->
    <consonant ph="J" ctype="n" cplace="p" cvox="0"/> <!-- nyamalala  -->
    <consonant ph="N" ctype="n" cplace="v" cvox="0"/> <!-- n'an'a  -->

    <!-- Approximant consonants (semivowels) and Trills and Flaps-->
    <consonant ph="r" ctype="r" cplace="a" cvox="0"/> <!-- ranga  -->
    <consonant ph="j" ctype="r" cplace="p" cvox="0"/> <!-- ya  -->
    <consonant ph="w" ctype="r" cplace="v" cvox="+"/> <!-- wa  -->

    <!-- Liquid consonants -->
```

```
    <consonant ph="l" ctype="l" cplace="a" cvox="0"/> <!-- lala  -->
</allophones>
```

B3: IsiNdebele phone set – allophone.nbl.xml

```
<allophones name="sampa" xml:lang="nbl"
    features="vlng vheight vfront vrnd ctype cplace cvox casp">

    <silence ph="_"/>
    <!-- Vowels  -->
    <vowel ph="i" vlng="s" vheight="1" vfront="1" vrnd="-"/> <!-- lima -->
    <vowel ph="u" vlng="s" vheight="1" vfront="3" vrnd="+"/> <!-- thunga -->
    <vowel ph="E" vlng="s" vheight="3" vfront="1" vrnd="-"/> <!-- sela -->
    <vowel ph="O" vlng="s" vheight="3" vfront="3" vrnd="+"/> <!-- bona -->
    <vowel ph="a" vlng="s" vheight="4" vfront="1" vrnd="-"/> <!-- lala -->

    <!-- Stop consonants -->
    <consonant ph="p_h" ctype="s" cplace="l" cvox="0" casp="+"/><!--iphaphu-->
    <consonant ph="p_>" ctype="s" cplace="l" cvox="0" casp="+"/> <!--iposo-->
    <consonant ph="b" ctype="s" cplace="l" cvox="+"/> <!-- khamba -->
    <consonant ph="b_&lt;" ctype="s" cplace="l" cvox="+"/> <!-- bona -->
    <consonant ph="t_h" ctype="s" cplace="a" cvox="0" casp="+"/> <!-- thuta  -->
    <consonant ph="t_>" ctype="s" cplace="a" cvox="0"/> <!-- itafula -->
    <consonant ph="d" ctype="s" cplace="a" cvox="+"/> <!-- idada -->
    <consonant ph="k" ctype="s" cplace="a" cvox="-"/> <!-- kela -->
    <consonant ph="k_h" ctype="s" cplace="v" cvox="-" casp="+"/> <!-- khamba -->
    <consonant ph="k_>" ctype="s" cplace="v" cvox="0"/> <!-- inkomo -->
    <consonant ph="g" ctype="s" cplace="v" cvox="+"/> <!-- igama -->

    <!-- Affricative consonants -->
    <consonant ph="tK_>" ctype="a" cplace="a" cvox="0"/> <!-- Itlawana -->
    <consonant ph="tK_h" ctype="a" cplace="a" cvox="-" casp="+"/> <!-- tlhaga -->
    <consonant ph="tS_>" ctype="a" cplace="a" cvox="0"/> <!-- utjani -->
    <consonant ph="tS_h" ctype="a" cplace="a" cvox="0" casp="+"/> <!—itjhatjha-->
    <consonant ph="ts_>" ctype="a" cplace="a" cvox="0"/> <!-- itsikiri -->
    <consonant ph="ts_h" ctype="a" cplace="a" cvox="0" casp="+"/> <!-- tshanya-->
    <consonant ph="kx" ctype="a" cplace="v" cvox="-"/> <!-- kghupula* -->
    <consonant ph="d_0Z" ctype="a" cplace="a" cvox="+"/> <!-- jabula -->
    <consonant ph="dz" ctype="a" cplace="a" cvox="+"/> <!-- idzila -->

    <!-- Fricative consonants -->
    <consonant ph="f" ctype="f" cplace="b" cvox="-"/> <!-- funa -->
    <consonant ph="v" ctype="f" cplace="b" cvox="+"/> <!-- vula  -->
    <consonant ph="s" ctype="f" cplace="a" cvox="-"/> <!-- susa -->
    <consonant ph="z" ctype="f" cplace="a" cvox="+"/> <!-- Zala -->
    <consonant ph="x" ctype="f" cplace="v" cvox="-"/> <!-- rhonona  -->
    <consonant ph="h" ctype="f" cplace="g" cvox="-"/> <!-- ihogo -->
    <consonant ph="K" ctype="f" cplace="a" cvox="-"/> <!-- hlala -->
    <consonant ph="K\" ctype="f" cplace="a" cvox="+"/> <!-- dlala -->

    <!-- Clicks consonants -->
    <consonant ph="1\" ctype="c" cplace="d" cvox="0"/> <!-- cima -->
    <consonant ph="1\g_0" ctype="c" cplace="d" cvox="+"/> <!-- gcina -->
    <consonant ph="1\h" ctype="c" cplace="d" cvox="0" casp="+"/><!--chacha-->
    <consonant ph="!\" ctype="c" cplace="p" cvox="0"/> <!-- qina -->
    <consonant ph="!\g_0" ctype="c" cplace="p" cvox="+"/> <!-- umgqomu -->
    <consonant ph="!_bh" ctype="c" cplace="p" cvox="0" casp="+"/><!--qhula-->

    <!-- Nasal consonants -->
    <consonant ph="m" ctype="n" cplace="l" cvox="0"/> <!-- mina -->
    <consonant ph="n" ctype="n" cplace="a" cvox="0"/> <!-- nina -->
    <consonant ph="J" ctype="n" cplace="p" cvox="0"/> <!-- inyama -->
    <consonant ph="N" ctype="n" cplace="v" cvox="0"/> <!-- nghala -->
```

```
    <!-- Approximant consonants (semivowels) and Trills and Flaps-->
    <consonant ph="r" ctype="r" cplace="a" cvox="0"/> <!-- rara -->
    <consonant ph="j" ctype="r" cplace="p" cvox="0"/> <!-- yena -->
    <consonant ph="w" ctype="r" cplace="v" cvox="+"/> <!-- wona -->

    <!-- Liquid consonants -->
    <consonant ph="l" ctype="l" cplace="a" cvox="0"/> <!-- lila -->
</allophones>
```

B4: Xitsonga phone set – allophone.tso.xml

```
<allophones name="sampa" xml:lang="tso"
              features="vlng vheight vfront vrnd ctype cplace cvox casp cpal">

    <silence ph="_"/>
    <!-- Vowels  -->
    <vowel ph="i" vlng="s" vheight="1" vfront="1" vrnd="-"/> <!-- ribye  -->
    <vowel ph="u" vlng="s" vheight="1" vfront="3" vrnd="+"/> <!-- huma  -->
    <vowel ph="E" vlng="s" vheight="3" vfront="1" vrnd="-"/> <!-- hela  -->
    <vowel ph="O" vlng="s" vheight="3" vfront="3" vrnd="+"/> <!-- songa  -->
    <vowel ph="a" vlng="s" vheight="4" vfront="1" vrnd="-"/> <!-- aka  -->

    <!-- Stop consonants -->
    <consonant ph="p" ctype="s" cplace="l" cvox="-"/> <!-- mpunga  -->
    <consonant ph="p_h" ctype="s" cplace="l" cvox="0" casp="+"/> <!-- phanga  -->
    <consonant ph="pj_e" ctype="s" cplace="l" cvox="-" cpal="+"/> <!-- pyopya -->
<consonant ph="pj_h" ctype="s" cplace="l" cvox="-" casp="+" cpal="+"/><!--phya-->
    <consonant ph="b" ctype="s" cplace="l" cvox="+"/> <!-- ba  -->
    <consonant ph="b_&amp;lt;" ctype="s" cplace="l" cvox="+"/> <!-- baba  -->
    <consonant ph="bj" ctype="s" cplace="l" cvox="+" cpal="+"/> <!-- byala  -->
    <consonant ph="t_h" ctype="s" cplace="a" cvox="0" casp="+"/> <!-- thula  -->
    <consonant ph="t_e" ctype="s" cplace="a" cvox="0"/> <!-- tatana  -->
    <consonant ph="tj" ctype="s" cplace="a" cvox="-" cpal="+"/> <!-- tyatyasa -->
    <consonant ph="d" ctype="s" cplace="a" cvox="+"/> <!-- dedeleka  -->
    <consonant ph="dj" ctype="s" cplace="a" cvox="+" cpal="+"/> <!-- dya  -->
    <consonant ph="dh_v" ctype="s" cplace="a" cvox="+" casp="+"/> <!-- ndhambi-->
    <consonant ph="k" ctype="s" cplace="a" cvox="-"/> <!-- kula  -->
    <consonant ph="k_h" ctype="s" cplace="v" cvox="-" casp="+"/> <!-- khoma  -->
    <consonant ph="g" ctype="s" cplace="v" cvox="+"/> <!-- gamba  -->
    <consonant ph="gh_v" ctype="s" cplace="v" cvox="+" casp="+"/> <!-- nghena -->
    <consonant ph="tl_e" ctype="s" cplace="a" cvox="-"/> <!-- tlanga  -->
    <consonant ph="tl_h" ctype="s" cplace="a" cvox="-" casp="+"/> <!-- tlhari -->

    <!-- Affricative consonants -->
    <consonant ph="dK" ctype="a" cplace="a" cvox="+"/> <!-- dlala  -->
    <consonant ph="dK_v" ctype="a" cplace="a" cvox="+" casp="+"/> <!-- ndlhazi-->
    <consonant ph="c" ctype="a" cplace="a" cvox="0"/> <!-- cina  -->
    <consonant ph="c_h" ctype="a" cplace="a" cvox="0" casp="+"/> <!-- chukucha-->
    <consonant ph="j_h" ctype="a" cplace="a" cvox="+"/> <!-- jaha  -->
    <consonant ph="dz`" ctype="a" cplace="a" cvox="+"/> <!-- dzwi, dzaha  -->
    <consonant ph="dz`h_v" ctype="a" cplace="a" cvox="+" casp="+"/><!--ndzhaku-->

    <!-- Fricative consonants -->
    <consonant ph="f" ctype="f" cplace="b" cvox="-"/> <!-- famba  -->
    <consonant ph="v" ctype="f" cplace="b" cvox="+"/> <!-- vhilwa  -->
    <consonant ph="s" ctype="f" cplace="a" cvox="-"/> <!-- sila  -->
    <consonant ph="z" ctype="f" cplace="a" cvox="+"/> <!-- muzumbi  -->
    <consonant ph="S" ctype="f" cplace="a" cvox="-"/> <!-- xava  -->
    <consonant ph="h_v" ctype="f" cplace="g" cvox="+"/> <!-- huma  -->
    <consonant ph="K" ctype="f" cplace="a" cvox="-"/> <!-- hlamula  -->
    <consonant ph="B" ctype="f" cplace="l" cvox="+"/> <!-- vona  -->
    <consonant ph="s`" ctype="f" cplace="a" cvox="-"/> <!-- sweka  -->
```

```
    <!-- Clicks consonants -->
    <consonant ph="1\" ctype="f" cplace="p" cvox="0"/> <!-- xiqoko  -->

    <!-- Nasal consonants -->
    <consonant ph="m" ctype="n" cplace="l" cvox="0"/> <!-- kuma  -->
    <consonant ph="n" ctype="n" cplace="a" cvox="0"/> <!-- manana  -->
    <consonant ph="J" ctype="n" cplace="p" cvox="0"/> <!-- nyoka  -->
    <consonant ph="N" ctype="n" cplace="v" cvox="0"/> <!-- ngulube  -->
    <consonant ph="n'" ctype="n" cplace="a" cvox="0"/> <!-- ndzhaku  -->
    <consonant ph="m_h" ctype="n" cplace="l" cvox="+" casp="+"/> <!-- mhaka  -->
    <consonant ph="n_h" ctype="n" cplace="a" cvox="+" casp="+"/> <!-- mhamu  -->

    <!-- Approximant consonants (semivowels) and Trills and Flaps-->
    <consonant ph="r" ctype="r" cplace="a" cvox="0"/> <!-- raha  -->
    <consonant ph="rh_v" ctype="r" cplace="a" cvox="0" casp="+"/> <!-- rhama  -->
    <consonant ph="j" ctype="r" cplace="p" cvox="0"/> <!-- yima  -->
    <consonant ph="w" ctype="r" cplace="v" cvox="+"/> <!-- wa  -->

    <!-- Liquid consonants -->
    <consonant ph="l" ctype="l" cplace="a" cvox="0"/> <!-- lomba  -->
</allophones>
```

APPENDIX C: CREATING DICTIONARY – gen_dictionary.py

```
#!/usr/bin/env python
# This script generates a dictionary (xy.txt) from a given pronunciation
# dictionary
# The new dictionary contains the actual word followed by its SAMPA pronunciation #
and is functional

import sys

if __name__ == '__main__':
    if len( sys.argv ) < 2:
        print( "Usage: " + sys.argv[0] +" inputFile" )
        sys.exit()

inFile = open( sys.argv[1], 'r' )
outFile = open( 'xy.txt', 'w' )

for line in inFile:
    line = line.replace( ' ', '' )
    line = line.split( '\t' )
    line = line[0] + " " + line[1] + " functional\n"
    outFile.write( line )

outFile.close()
inFile.close()
```

APPENDIX D: CLASSIFICATION FUNCTION – NamesPredictor.java

```
package weka.classifiers;
import weka.classifiers.Classifier;
import weka.core.Attribute;
import weka.core.DenseInstance;
import weka.core.Instances;
import java.util.ArrayList;
import java.util.List;
/**
 * Class for Language Identification of Names
 *
 * @author            Tshephisho Joseph Sefara
 * @institution       University of Limpopo
 * @citation          Master's Thesis
 */
public class NamesPredictor {
   /**
    * String that stores the text to guess its language.
    */
   static String text;
   /**
    * Object that stores the instance.
    */
   static Instances instances;
   /**
    *String that stores locale of the predicted language
    */
    static String locale;
   /**
    * Object that stores the classifier.
    */
    static Classifier cls;
   /**
    * Creates the constructor.
    * @throws Exception
    */
   public NamesPredictor(String strText, String model) throws Exception {
      // sets text to be classified
      text = strText;
      // this function prepares the new instance
      makeInstance();
      // this function loads the classifier model
      loadModel(model);
   }
   /**
    * This method reads the classifier object or model.
    * @throws Exception
    */
   public static void loadModel(String modelName) throws Exception {
         cls = (Classifier) weka.core.SerializationHelper.read(modelName);
   }

   /**
    * This method creates an instance to be predicted.
    */
   public static void makeInstance() {
      // Create the header
      List<Attribute> attributeList = new ArrayList<Attribute>(2);
      // Create first attribute, the class
      List<String> values = new ArrayList<String>(3);
      values.add("nso");
```

```
        values.add("tso");
        values.add("ven");
        Attribute attribute1 = new Attribute("class", values);
        attributeList.add(attribute1);
        // Create second attribute, the text
        Attribute attribute2 = new Attribute("surname",(List<String>) null);
        attributeList.add(attribute2);
        // Build instance set with just one instance
        instances = new Instances("Test relation", (java.util.ArrayList<Attribute>)
attributeList, 1);
        // Set class index
        instances.setClassIndex(0);
        // Create and add the instance
        DenseInstance instance = new DenseInstance(2);
        instance.setDataset(instances);
        instance.setValue(attribute2, text);
        instances.add(instance);
    }
    /**
     * This method performs the classification of the instance
     * and returns a locale (string).
     * @throws Exception
     */
    public String classify() throws Exception {
          // Predicts a language given an instance
        double pred = cls.classifyInstance(instances.instance(0));
        locale = instances.classAttribute().value((int) pred);
        return locale;
    }
}
```

APPENDIX E: ANDROID SOURCE CODE

E.1: MainActivity.java

```
package za.co.speechtech.ttsdemo;
import android.app.DownloadManager;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import java.io.IOException;
import java.util.concurrent.ExecutionException;
import static android.widget.Toast.makeText;
public class MainActivity extends AppCompatActivity {
    //private TextView mTextMessage;
    String locale=" "; //declare and initialise locale
    EditText area; //declare text area
    Button clear, speak, download; //declare buttons
    String text=" "; //declare and initialise text/surname
    String model = "namesModel"; //declare default predictor model
    String lidUrl; //address from language identification
    String userUrl; //address from user selection
    final static String [] locales = {"nso","tso","ven","nbl","en_US","detect"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        area = (EditText) findViewById(R.id.editText2);
        clear = (Button) findViewById(R.id.button1);
        //clear text area
        clear.setOnClickListener(new View.OnClickListener() {
            public void onClick (View view) {
                area.setText("");
            }
        });
        speak = (Button) findViewById(R.id.button2);
        //Plays audio file from remote site
        speak.setOnClickListener(new View.OnClickListener() {
            public void onClick (View view) {
                try {
                    if(urlBuilder(model)) {
                        player(locale.equals(locales[5]) ? lidUrl : userUrl);
                    }
                } catch (IOException | InterruptedException | ExecutionException e) {
                    e.printStackTrace();
                }
            }
        });
        download = (Button) findViewById(R.id.button);
        //Download audio file from remote site
        download.setOnClickListener(new View.OnClickListener() {
            public void onClick (View view) {
                try {
                    if (urlBuilder(model))
                    {
                        String uri;
```

```
                        if (!Methods.isempty(text)) {
                            uri = locale.equals(locales[5]) ? lidUrl : userUrl;
                            DownloadManager.Request req = new
DownloadManager.Request(Uri.parse(uri));

req.setDestinationInExternalPublicDir((Environment.DIRECTORY_DOWNLOADS),"audio.wav");

req.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE_NOTIFY_COMPL
ETED);
                            req.allowScanningByMediaScanner();
                            DownloadManager downloadManager = (DownloadManager)
getSystemService(DOWNLOAD_SERVICE);
                            downloadManager.enqueue(req);
                        }
                    }
                }catch(IOException | InterruptedException | ExecutionException e){
                        e.printStackTrace();
                }
            }
        });
    }
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
    //sets languages based on user selection
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        switch (id) {
            case R.id.item1:
                locale = locales[0];
                return true;
            case R.id.item2:
                locale = locales[2];
                return true;
            case R.id.item3:
                locale = locales[1];
                return true;
            case R.id.item4:
                locale = locales[3];
                return true;
            case R.id.item5:
                locale = locales[4];
                return true;
            case R.id.item6:
                locale = locales[5];
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
    //generate url based on user selection or automatic language identification
    public boolean urlBuilder(String model) throws IOException, ExecutionException,
InterruptedException {
        text = area.getText().toString().trim();
        if (Methods.isempty(text)){
            makeText(getBaseContext(), "Enter surname", Toast.LENGTH_SHORT).show();
            return false;
        }
        else {
            text = Methods.encode_text(text);
        }
        if (Methods.isempty(locale.trim())){
```

```
            makeText(getBaseContext(), "Select the language",
Toast.LENGTH_SHORT).show();
            return false;
        }
        if(locale.equals(locales[5])) {
            text = area.getText().toString().trim();
            text = Methods.encode_text(text);
            if (Methods.isInternet(getBaseContext())) {
                String urlAddress =
"http://www.speechtech.co.za/alltts/classify.php?text=" + text + "&model=" + model;
                LanguageIdentification LID = new LanguageIdentification();
                LID.execute(urlAddress);
                String language = LID.get();
                switch (language) {
                    case "nso":
                        makeText(getBaseContext(), "Language set to Sepedi",
Toast.LENGTH_LONG).show();
                        break;
                    case "tso":
                        makeText(getBaseContext(), "Language set to Xitsonga",
Toast.LENGTH_LONG).show();
                        break;
                    case "ven":
                        makeText(getBaseContext(), "Language set to Tshivenda",
Toast.LENGTH_LONG).show();
                        break;
                    case "nbl":
                        makeText(getBaseContext(), "Language set to IsiNdebele",
Toast.LENGTH_LONG).show();
                        break;
                    default:
                        makeText(getBaseContext(), "Language not detected",
Toast.LENGTH_LONG).show();
                        return false;
                }
                //create URL based on identified language
                lidUrl =
"http://www.speechtech.co.za/alltts/download.php?INPUT_TEXT=" + text + "&LOCALE=" +
language + "&act=download";
                return true;
            }
            makeText(getBaseContext(), "No Internet Availability",
Toast.LENGTH_LONG).show();
            return false;
        }
        //create URL based on user selection
        userUrl =
"http://www.speechtech.co.za/alltts/download.php?INPUT_TEXT="+text+"&LOCALE="+locale+
"&act=download";
        return true;
    }
    //plays the audio given a URL
    public void player(String url) {
        //check internet status
        if (Methods.isInternet(getBaseContext()))
            try {
                PlayAudioManager.playWave(getApplicationContext(), url);
            } catch (Exception e) {
                e.printStackTrace();
            }
        else {
            makeText(getBaseContext(), "No Internet Availability",
Toast.LENGTH_SHORT).show();
```

```
        }
    }
}
```

E.2: LanguageIdentification.java

```
package za.co.speechtech.ttsdemo;
import android.os.AsyncTask;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
/**
 * Created by Tshephisho Joseph Sefara on 2017/03/12.
 *
 */
class LanguageIdentification extends AsyncTask<String , Void, String> {
    HttpURLConnection urlConnection = null;
    @Override
    protected String doInBackground(String... params) {
        StringBuilder response  = new StringBuilder();
        try {
            URL url = new URL(params[0]);
            urlConnection = (HttpURLConnection) url.openConnection();
            InputStream in = new
BufferedInputStream(urlConnection.getInputStream());
            BufferedReader read = new BufferedReader(new InputStreamReader(in));
            String line;
            while ((line = read.readLine()) != null) {
                response.append(line);
            }
        } catch (IOException e){
            e.printStackTrace();
        }
        return response.toString();
    }
    public void onPostExecute (String results) {
        urlConnection.disconnect();
    }
}
```

E.3: Methods.java

```
package za.co.speechtech.ttsdemo;
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import android.content.Context;
import android.net.NetworkInfo;
import android.net.ConnectivityManager;
/**
 * Created by Tshephisho Joseph Sefara on 2017/03/12.
 * This class contains static methods
 */
class Methods {
    //Checks internet connectivity
    static boolean isInternet(Context c) {
        ConnectivityManager connect = (ConnectivityManager)
c.getSystemService(Context.CONNECTIVITY_SERVICE);
```

```
        NetworkInfo activeNet = connect.getActiveNetworkInfo();
        return activeNet != null && activeNet.getState() ==
NetworkInfo.State.CONNECTED;
    }
    //Check if the argument is empty
    static boolean isempty(String tempString) {
        return tempString.isEmpty() || tempString.length() == 0 ||
tempString.equals("");
    }
    //Use URLEncoder to encode the argument
    static String encode_text(String text) throws UnsupportedEncodingException {
        text = URLEncoder.encode(text, "UTF-8");
        return text;
    }
}
```

E.4: PlayAudioManager.java

```
package za.co.speechtech.ttsdemo;
import android.content.Context;
import android.media.MediaPlayer;
import android.net.Uri;
/**
 * Created by Tshephisho Joseph Sefara on 2017/03/12.
 *
 */
class PlayAudioManager {
    private static MediaPlayer mediaplayer;
    static void playWave (final Context context, final String urlPath) throws
Exception {
        if (mediaplayer == null) {
            mediaplayer = MediaPlayer.create(context, Uri.parse(urlPath));
        }
        mediaplayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer arg0) {
                try {
                    if (mediaplayer != null) {
                        mediaplayer.reset();
                        mediaplayer.release();
                        mediaplayer = null;
                    }
                }catch (Exception e ){
                    e.printStackTrace();
                }
            }
        });
        mediaplayer.start();
    }
}
```

E.5: Activity_main.xml

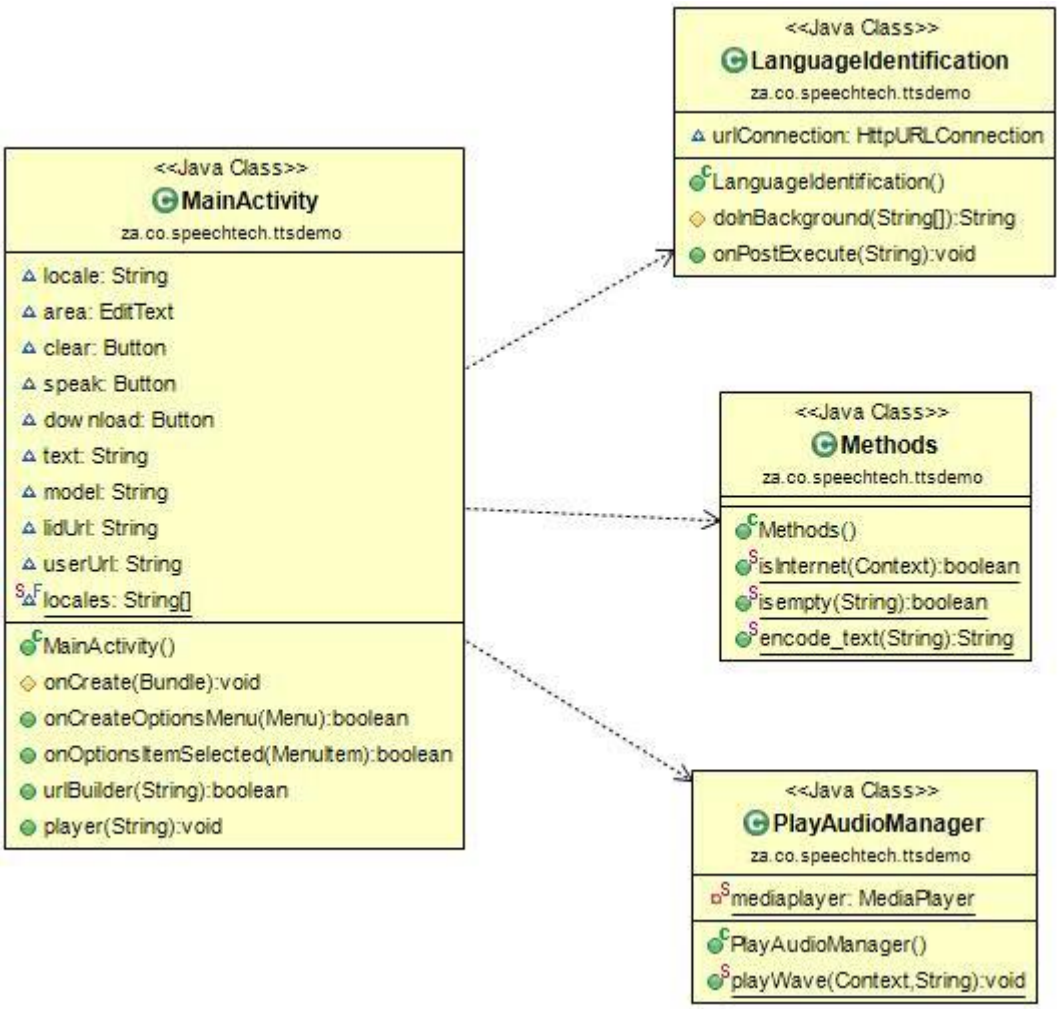
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:background="@color/colorAccent"
    tools:context="za.co.speechtech.ttsdemo.MainActivity"
    android:paddingTop="@dimen/activity_vertical_margin">
    <EditText
        android:id="@+id/editText2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:textColor="#000000"
        android:ems="10"
        android:background="@drawable/textareaborder"
        android:padding="1dp"
        android:gravity="top"
        android:inputType="textMultiLine"
        android:layout_below="@+id/button1">
        <requestFocus />
    </EditText>
    <Button
        android:id="@+id/button1"
        style="@android:style/Widget.Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/button2"
        android:layout_alignLeft="@+id/editText2"
        android:layout_alignParentTop="true"
        android:layout_alignStart="@+id/editText2"
        android:clickable="true"
        android:text="@string/button_clear"
        android:textSize="20sp"
        tools:text="@string/button_clear" />
    <Button
        android:id="@+id/button2"
        style="@android:style/Widget.Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignEnd="@+id/editText2"
        android:layout_alignParentTop="true"
        android:layout_alignRight="@+id/editText2"
        android:clickable="true"
        android:text="@string/button_speak"
        android:textSize="20sp"
        tools:text="@string/button_speak" />
    <Button
        android:id="@+id/button"
        style="@android:style/Widget.Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/editText2"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:text="@string/button_download"
        android:textSize="20sp"
        tools:text="@string/button_download" />
</RelativeLayout>
```

E.6: Menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1" android:title="Sepedi"
android:orderInCategory="2"></item>
    <item android:id="@+id/item2" android:title="Tshivenda"
android:orderInCategory="3"></item>
    <item android:id="@+id/item3" android:title="Xitsonga"
android:orderInCategory="4"></item>
    <item android:id="@+id/item4" android:title="isiNdebele"
android:orderInCategory="5"></item>
    <item android:id="@+id/item5" android:title="English"
android:orderInCategory="6"></item>
    <item android:id="@+id/item6" android:title="Auto Detect"
android:orderInCategory="1"></item>
</menu>
```

APPENDIX F: ANDROID APPLICATION UML DIAGRAM



APPENDIX G: CONSENT FORM

University of Limpopo
Telkom Centre of Excellence for Speech Technology
Department of Computer Science
Office 1015 Mathematical Sciences Building
Private Bag X1106, SOVENGA, 0727, South Africa
Tel: (015) 268 2751, Fax: (015) 268 3487, Email: sefaratj@gmail.com

THE DEVELOPMENT OF AN AUTOMATIC PRONUNCIATION ASSISTANT

Dear Participant

My name is Tshephisho Joseph Sefara and I am a postgraduate student enrolled in the Department of Computer Sciences at the University of Limpopo. I am conducting research on the development of an automatic pronunciation system that uses voice and machine-learning technologies. The purpose of the research is to build a voice-enabled system that uses a trained classifier to enhance pronunciation of words and phrases, particularly surnames of the Sepedi, Tshivenda, Xitsonga and isiNdebele mother tongue speakers. The information gathered here will be used purely for academic purposes, but the final document will be a public document in the form of a research report. I hereby invite you to participate in the research study to evaluate the performance of the developed system.

Your participation in this research is voluntary and you are free to withdraw anytime. There will be no remuneration or gifts in exchange for information provided. Your identity will remain anonymous and the information you provide will be confidential. There are no known risks to participation beyond those encountered in everyday life. You are entitled to withhold information that you feel is too personal or sensitive to you and you can choose not to answer any of the questions. The enclosed questionnaire has been designed to collect information on the quality and intelligibility of the developed system. If you are willing and available to participate in this research project, please sign below and answer the questions on the questionnaire as best you can:

Consent Form Code (filled by interviewer): _____

Signature: _____ Date: ____/_____/2017 Place: _____

It should take approximately 15 minutes to complete. Thank you for agreeing to participate in this research study.

APPENDIX H: QUESTIONNAIRE

This questionnaire was presented to the evaluators where they evaluate ttwo systems, namely, the developed system and natural speech recorded at 16-bit PCM. System A represents the **developed TTS synthesis system** and system B represents **natural speech**.

Evaluation Form

Introduction:

The aim of this questionnaire is to gather information to evaluate the performance of an automatic pronunciation system that is presented to the evaluators. This evaluation form is completed once the subjects have familiarised themselves with the system. The respondents MUST complete the consent form before answering this questionnaire.

Instructions:

- Always give personal honest opinions.
- Select an appropriate answer were multiple answers are given by means of a cross (x).
- Answer all the questions as completely as possible.

Section 1: General Questions

1. Home language:

Sepedi	Xitsonga	Tshivenda	isiNdebele

2. Level of study:

Postgraduate	Undergraduate

3. Gender:

Male	Female

4. Age range:

18-35	36 and above

5. Are you familiar with text-to-speech synthesis systems?

Yes	No

Section 2: Intelligibility

The researcher will play the audio files sequentially.

Write down each sentence before the researcher plays the next audio.

System A:

Audio 1:

Audio 2:

Audio 3:

Audio 4:

Audio 5:

Section 3: Speech Quality

This section is completed after five normal sentences are played to the respondents. Please answer each question on a scale of 1 to 5 where:

MOS	Quality	Listening Effort
1	Bad	No meaning understood
2	Poor	Effort required
3	Fair	Moderate effort required
4	Good	No appreciable effort required
5	Excellent	No effort required

Questions for System A	1	2	3	4	5
1. How was the pronunciation of words?					
2. How was the naturalness of the voice?					
3. How was the pleasantness of the voice?					
4. How much effort was needed to listen and understand the message?					
5. How was the overall quality of the audio on all aspects?					
Questions for System B	1	2	3	4	5
1. How was the pronunciation of words?					
2. How was the naturalness of the voice?					
3. How was the pleasantness of the voice?					
4. How much effort was needed to listen and understand the message?					
5. How was the overall quality of the audio on all aspects?					

Section 4: User Acceptance

This section is completed after the respondents have interacted with the system on the website. Please answer each question on a scale of 1 to 5 where:

MOS	Meaning
1	Strongly disagree
2	Disagree
3	Neither
4	Agree
5	Strongly agree

Questions	1	2	3	4	5
1. Buttons are visible and easy to find.					
2. Languages can be switched easily.					
3. Text is visible and clear.					
4. Layout and colours are displayed perfectly.					
5. The application was difficult to understand.					
6. I can use the application on my own.					
7. I felt very confident using the application.					
8. I would recommend this application to someone else.					
9. I would frequently use this application.					
10. This application can help me learn pronunciation of new languages.					
11. Would you recommend these voices to be integrated in future devices?					

APPENDIX I: SPEECH SYNTHESISER – TEST CORPUS SAMPLES

I.1: Test corpus samples of intelligibility test

The five sample sentences for intelligibility test of each language using syntactic rules explained by Benoît *et al.* (1996) are shown below:

	Sepedi Text	**English Meaning**
1.	Buka e sepetše godimo ga lefase le le botse.	The book walked through the attractive floor.
2.	Komiki ya go hloka maatla e rata letšatši.	The weak cup loves the day
3.	Ruta ntlo le leoto.	Teach the house and the leg.
4.	Go tla bjang gore bošego bo hloye letšatši le le ntsho?	How does the night hate the black sun?
5.	Meetse a bone mohlare wo o hwilego.	The water saw the tree that died.
	Tshivenda Text	
6.	Bugu yo tshimbila fhasi ho no tamisa.	The book walked through the attractive floor.
7.	Tshinwelo tshi sina nungo tshi funa Duvha.	The weak cup loves the day
8.	Funza nndu mulenzhe	Teach the house and the leg
9.	Vhusiku vhu vhenga hani Duvha litshwu	How does the night hate the black sun?
10.	Madi o vhona muri une wa khou fa	The water saw the tree that died.
	Xitsonga Text	
11.	Buku yi fambile hole ndhawini yo navetisa.	The book walked through the attractive floor.
12.	Xinwelo xi rhandza siku.	The cup loves the day
13.	Dyondzisa yindlu na nenge.	Teach the house and the leg
14.	Vusiku byi vanga hi ndlela yini vunyama bja fambi?	How does the night hate the black sun?
15.	Mati ya vonile nsinya lowu fake.	The water saw the tree that died.
	isiNdebele Text	
16.	Incwadi ikhambe ephasini isitubhe selihle	The book walked through the attractive floor.
17.	Ibhigiri elinganamandla lithanda ilanga	The weak cup loves the day
18.	Fundisa indlu nomlenze	Teach the house and the leg
19.	Kuzanjani ukuthi ubusuku buzonde ilanga elimnyama?	How does the night hate the black sun?
20.	Amanzi abone isihlahla esifileko.	The water saw the tree that died.

I.2: Test corpus samples of MOS test

The five sample sentences for MOS test of each language are given below:

	Sepedi Text	**English Meaning**
1.	Modipa o rata go etela mosadi wa gagwe ma felelong a beke.	Modipa likes to visit his wife on weekends.
2.	Mo ponong Modimo a mpontšha Moprista yo Mogolo, a eme pele ga morongwa wa Morena.	He showed me the high priest standing before the angel of the LORD.
3.	Thobela Manamela!	Hello Manamela!
4.	Mothapo o a bolela.	Mothapo is speaking.
5.	Ke wena Mokgalong?	Are you Mokgalong?
	Tshivenda Text	
6.	Mathoho utakalela uya hayani nga lavhutanu	Mathoho likes to visit home on Fridays.
7.	Aa vho Nemagovhani!	Hello Nemagovhani!
8.	Tshivhombela o thanya	Tshivhombela is wise.
9.	Ndi ini vho Netshivhulana vha mudivhale?	Are you the famous Netshivhulana?
10.	Ne ndo takala vhukuma	I am fine, thank you
	isiNdebele Text	
11.	uMothwa ungumhlanyeli o wazekako.	Mothwa is a famous farmer.
12.	Lotjha Sibiya!	Hello Sibiya!
13.	uSindane ulele.	Sindane is sleeping.
14.	Unjani Mahlangu?	How are you Mahlangu?
15.	Maucala ngendlela leyo uSindane enzangakhona, uba ngumrhubhululi omkhulu.	Looking at how Sindane is doing, he is becoming a great researcher.
	Xitsonga Text	
16.	Mayindi u rhandza ku khongela a ri wexe.	Mayindi likes to pray alone.
17.	Shikwambane hi yena ntsena mlungu exikolweni.	Shikwambane is only the only white person at school.
18.	Xewani Baloyi!	Hello Baloyi!
19.	Kunjhani Dzambukeri?	How are you Dzambukeri?
20.	Tsakane u catile ahari exikolweni.	Tsakane got married while she was still at school.

APPENDIX J: NATURAL SPEECH – TEST CORPUS SAMPLES

The HMM voices were built on secondary data and the following sentences were extracted from the speech corpus to be used for MOS test

	Sepedi Text	**English Meaning**
1.	Botagwa bo ka bolaya.	Drinking can kill you.
2.	Mešomo ya dikontraka e tla abelwa banna fela.	Only men will be allocated contractual jobs.
3.	Se bolele ka mosadi wa go se botege.	Do not talk about unfaithful woman.
4.	Tokomane ya boitsebišo.	Identity document
5.	Kopano e tla tšwela pele.	The meeting will continue.
	Tshivenda Text	
6.	Nahone hu na zwidini zwa hone.	There is something bothering me.
7.	Yakobo a ri u songo mpha tshithu.	Jacob said do not give me anything.
8.	Na zwifuiwa hezwi ndi zwanga.	These animals are mine.
9.	Ri khou lifhedzwa malofha awe zwino.	We are being punished for his blood.
10.	Ri vha rine nga mvelele heyi.	We are who we are because of this culture.
	isiNdebele Text	
11.	Ekuseni ngizokuza ngizokuhlola ilembe lami.	In the morning I will come to check my plough.
12.	Akhe ugijime ngikutjele bona kwenzekeni.	May you hurry so I can tell you what happenend.
13.	Ngiyiqunte izipho ikukhu yathi ngiyilise.	I have cut off the chickens' nails and it said I should stop.
14.	Yarhubha yarhubha kodwana yangalitholi.	It scratched and scratched but did not find it.
15.	Ikukhu yakhamba ihlengezela iya kibo.	The chicken went about casually home.
	Xitsonga Text	
16.	I ntiyiso Khanyisa.	It is the truth Khanyisa.
17.	Hayikhona, ndza sola mani!	No, I doubt!
18.	Timhuti ti sungula ku khana.	Everthing is quiet.
19.	Va rhukaniwa ro vuyavuyani.	They continually insulted them.
20.	Hlomani na yena a ri kona.	Even Hlomani was there.

APPENDIX K: SENTENCE AND WORD ERROR RATE – error_rates.py

```
#!/usr/bin/env python
# This script calculate sentence and word error rate (SER and WER) using
# Levenshtein distance. It receives a text file and test cases. The text file
# contains sequence of sentences where the first sentence is the reference and
# the second sentence is the hypothesis. Usage:
# Sentence reference1
# Sentence hypothesis1
# Sentence reference2
# Sentence hypothesis2 . . .
# The SER and WER are displayed in percentage notation.
# Usage: wer.py inputFile.txt #testCases

import sys
import numpy

if __name__ == '__main__':
    if len( sys.argv ) < 3:
        print( "Usage: " + sys.argv[0] +" inputFile #testCase" )
        sys.exit()

inFile, wer, ser = open( sys.argv[1], 'r' ), [], 0.0
for n in range( int( sys.argv[2] ) ):
    ref = inFile.readline().split()
    ref_len = len( ref ) + 1
    hyp = inFile.readline().split()
    hyp_len = len( hyp ) + 1
    distance = numpy.zeros( ref_len * hyp_len, dtype=numpy.uint8 )
    distance = distance.reshape( ref_len, hyp_len )
    #Build the matrix
    for x in range( ref_len ):
        for y in range( hyp_len ):
            if x == 0:
                distance[0][y] = y
            elif y == 0:
                distance[x][0] = x
    #Calculations
    for x in range( 1, ref_len ):
        for y in range( 1, hyp_len ):
            if ref[x-1] == hyp[y-1]:
                distance[x][y] = distance[x-1][y-1]
            else:
                substitution = distance[x-1][y-1] + 1
                insertion    = distance[x][y-1] + 1
                deletion     = distance[x-1][y] + 1
                distance[x][y] = min( substitution, insertion, deletion )
    #Calculate word error
    error = float( distance[len( ref )][len( hyp )]) / len( ref ) * 100
    #Calculate sentence errors if there is word error
    if error != 0: ser += 1
    #Prepare word error list
    wer.append( error )
average_ser = ser / int( sys.argv[2] ) * 100  #SER results
average_wer = sum(wer) / int( sys.argv[2] )   #WER results
print ( "Total SER = %.2f %%" % average_ser ) #Display SER results
print ( "Total WER = %.2f %%" % average_wer ) #Display WER results
inFile.close()
```

REFERENCES

Adiga, N. & Prasanna, S.R., 2014. A hybrid text-to-speech synthesis using vowel and non vowel like regions. In *2014 Annual IEEE India Conference (INDICON)*. Pune, India, IEEE, pp. 1-5.

Agarwal, B. & Mittal, N., 2012. Text classification using machine learning methods - a survey. In *Second International Conference on Soft Computing for Problem Solving (SocProS 2012)*. Jaipur, pp. 701-710.

Al-Badarenah, A. *et al.*, 2016. Classifying Arabic text using KNN classifier. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 7(6), pp. 259-268.

Anil, M.C. & Shirbahadurkar, S.D., 2014. Speech modification for prosody conversion in expressive Marathi text-to-speech synthesis. In *2014 International Conference on Signal Processing and Integrated Networks (SPIN)*. Noida, India, IEEE, pp. 56-58.

HTK website, 2009. [Online] Available at: http://htk.eng.cam.ac.uk/ [Accessed 15 March 2016].

The Festival Speech Synthesis System , 2014. [Online] (2.4) Available at: http://www.cstr.ed.ac.uk/projects/festival/ [Accessed 25 March 2016].

Language Resource Management Agency, 2016. [Online] Available at: http://rma.nwu.ac.za [Accessed 04 May 2016].

Aoga, J.O., Dagba, T.K. & Fanou, C.C., 2016. Integration of Yoruba language into MaryTTS. *International Journal of Speech Technology*, 19(1), pp. 151-158.

Badenhorst, J.A., Van Niekerk, D.R. & Barnard, E., 2006. Automatic systems for assistance in improving pronunciations. In *Proceedings of the Seventeenth*

Annual Symposium of the Pattern Recognition Association of South Africa (PRASA). Parys, pp. 23-29.

Baloyi, N., 2012. *A text-to-speech synthesis system for Xitsonga using hidden Markov models*. Master's thesis. Polokwane: University of Limpopo.

Beněk, T., 2014. *Implementing and improving a speech synthesis system*. Master's thesis. Brno, Czech Republic: Brno University of Technology.

Bengio, Y., 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1), pp. 1-127.

Benoît, C., Grice, M. & Hazan, V., 1996. The SUS test: A method for the assessment of text-to-speech synthesis intelligibility using Semantically Unpredictable Sentences. *Speech Communication*, 18(4), pp. 381-392.

Bird, S., 2006. NLTK: The natural language toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*. Sydney, Australia, Association for Computational Linguistics, pp. 69-72.

Black, A.W. & Lenzo, K.A., 2001. Flite: a small fast run time synthesis engine. In *SSW4-2001*. Perthshire, Scotland, pp. 157-162.

Black, A.W. & Lenzo, K.A., 2014. *Festvox: building synthetic voices*. [Document] (2.7) Available at: http://www.festvox.org/bsv/ [Accessed 3 November 2016].

Black, A.W., Zen, H. & Tokuda, T., 2007. Statistical parametric speech synthesis. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07 (Volume:4)*. Honolulu, HI, IEEE, pp. IV1229 - IV1232.

Boersma, P. & Weenink, D., 2013. *Praat: doing phonetics by computer*. [Online] Available at: http://www.fon.hum.uva.nl/praat/ [Accessed 20 March 2016].

Botha, G.R. & Barnard, E., 2008. *Text-Based Language Identification for the South African Languages*. Gauteng Province, Republic of South Africa: Master's thesis, University of Pretoria.

Botha, G.R. & Barnard, E., 2012. Factors that affect the accuracy of text-based lagnuage identification. *Computer Speech and Language*, 26(5), pp. 307-320.

Botha, G., Zimu, V. & Barnard, E., 2007. Text-based language identification for South African languages. *South African Institute of Electrical Engineers*, 98(4), pp. 141-146.

Burges, C.J., 1998. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2), pp. 121-167.

Cavnar, W.B. & Trenkle, J.M., 1994. N-gram based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*. Las Vegas, US, pp. 161-175.

Chang, C.-C. & Lin, C.-J., 2011. LIBSVM : A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), pp. 1-27.

Chen, N.F. & Li, H., 2016. Computer-assisted pronunciation training: From pronunciation scoring towards spoken language learning. In *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. Jeju, IEEE, pp. 1-7.

Dagba, T.K., & Boco, C., 2014. A text to speech system for Fon language using Multisyn algorithm. *Procedia Computer Science*, 35, pp. 447-455.

Davel, M. & Martirosian, O., 2009. Pronunciation dictionary development in resource-scarce environments. In *Proceedings of Interspeech*. Brighton, UK, pp. 2851-2854.

Eskenazi, M., 1999. Using automatic speech processing for foreign language pronunciation tutoring: Some issues and a prototype. *Language Learning & Technology*, 2(2), pp. 62-76.

Farhoodi, M., Yari, A. & Sayah, A., 2011. N-gram based text classification for Persian newspaper corpus. In *Digital Content, Multimedia Technology and its*

Applications (IDCTA), 2011 7th International Conference on. Busan, IEEE, pp. 55-59.

Farid, D.M. *et al.*, 2014. Hybrid decision tree and naïve Bayes classifiers for multi-class classification tasks. *Expert Systems with Applications*, 41(4), pp. 1937-1946.

Flanagan, J.L., Ishizaka, K. & Shipley, K.L., 1975. Synthesis of speech from a dynamic model of the vocal cords and vocal tract. *The Bell System Technical Journal*, 54(3), pp. 485-506.

Fourie, W., Du Toit, J.V. & Snyman, D.P., 2014. Comparing support vector machine and multinomial naïve Bayes for named entity classification of South African languages. In *Proceedings of the 2014 PRASA, RobMech and AfLaT International Joint Symposium*. Cape Town, pp. 183-188.

Gahlawat, M., Malik, A. & Bansal, P., 2014. Natural speech synthesiser for blind persons using hybrid approach. *Procedia Computer Science*, 41, pp. 83-88.

Gibson, M. *et al.*, 2010. Unsupervised cross-lingual speaker adaptation for HMM-based speech synthesis using two-pass decision tree construction. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. Dallas, TX, IEEE, pp. 4642-4645.

Gilakjani, A.P., 2012. A Study of Factors Affecting EFL Learners' English Pronunciation Learning and the Strategies for Instruction. *International Journal of Humanities and Social Science*, 2(3), pp. 119-128.

Gilakjani, A.P. & Ahmadi, M.R., 2011. Why is Pronunciation So Difficult to Learn? *English Language Teaching*, 4(3), pp. 74-83.

Giwa, O. & Davel, M.H., 2013. N-gram based language identification of individual words. In *The 24th Annual Symposium of the Pattern Recognition Association of South Africa (PRASA)*. Johannesburg, pp. 15-21.

Giwa, O. & Davel, M.H., 2014. Language identification of individual words with joint sequence models. In *Proceedings of 15th Annual Conference of the International Speech Communication Association, INTERSPEECH*. Singapore, pp. 1400-1404.

Giwa, O. & Davel, M.H., 2015. Text-based language identification of multilingual names. In *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*. Port Elizabeth, South Africa, IEEE, pp. 166-171.

Gonzalvo, X. *et al.*, 2016. Recent advances in Google real-time HMM-driven unit selection synthesizer. In *Proceedings of Interspeech 2016*. San Francisco, USA, pp. 2238-2242.

Greenspan, S.L., Bennett, R.W. & Syrdal, A.K., 1998. An evaluation of diagonastic rhyme test. *International Journal of Speech Technology*, 2(3), pp. 201-214.

Guma, M., 2001. The cultural meaning of names among Basotho of Southern Africa: A historical and linguistic analysis. *Nordic Journal of African Studies*, 10(3), pp. 265-279.

Hall, M. *et al.*, 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations Newsletter*, 11(1), pp. 10-18.

Hannan, A. & Sarma, S.K., 2015. Identification of Assamese and Bodo language from text- an approach. *International Journal of Engineering Research & Technology (IJERT)*, 4(12), pp. 67-70.

Heck, M., Stuker, S. & Waibel, A., 2012. A hybrid phonotactic language identification system with an SVM back-end for simultaneous lecture translation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Kyoto, IEEE, pp. 4857-4860.

Hinton, G. *et al.*, 2012. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6), pp. 82-97.

House, A.S., Williams, C.E., Hecker, M.H. & Kryter, K.D., 1965. Articulation-testing methods: Consonantal differentiation with a closed-response set. *Journal of the Acoustical Society of America*, 37(1), pp. 158-166.

Hsu, C.W. & Lin, C.J., 2002. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2), pp. 415-425.

Huang, X., Acero, A. & Hon, H.-W., 2001. *Spoken language processing: A guide to theory, algorithm, and system development*. 1st ed. NJ, USA: Prentice Hall PTR.

Hu, Y. & Loizou, P.C., 2008. Evaluation of objective quality measures for speech enhancement. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(1), pp. 229-238.

Indhuja, K. *et al.*, 2014. Text based language identification system for Indian languages following devanagiri script. *International Journal of Engineering Research & Technology (IJERT)*, 3(4), pp. 327-331.

Jeon, M. *et al.*, 2015. Menu Navigation With In-Vehicle Technologies: Auditory Menu Cues Improve Dual Task Performance, Preference, and Workload. *International Journal of Human-Computer Interaction*, 31(1), pp. 1-16.

Julnes, G. & Bustelo, M., 2014. Proffesional values and ethics in evaluation. *American Journal of Education*, 35(4), pp. 525-526.

Jurafsky, D. & Martin, J.H., 2014. *Speech and language processing*. Pearson Prentice Hall.

Kenworthy, J., 1987. *Teaching english pronunciation*. New York: Longman. pp. 1-11.

Kibriya, A.M., Frank, E., Pfahringer, B. & Holmes, G., 2004. Multinomial naive Bayes for text categorization revisited. In *Australian Joint Conference on Artificial Intelligence*. Berlin, Heidelberg, Springer, pp. 488-499.

Kiflu, A. & Beshah, T., 2012. Unit selection based text-to-speech synthesizer for Tigrinya language. *HiLCoE Journal of Computer Science and Technology*, 1(1), pp. 13-21.

King, S. *et al.*, 2003. *Edinburgh Speech Tools Library*. [Online] Available at: http://www.cstr.ed.ac.uk/projects/speech_tools/manual-1.2.0/ [Accessed 16 March 2016].

Klatt, D.H., 1987. Review of text-to-speech conversion for English. *Journal of the Acoustic Society of America*, 82(3), pp. 737-793.

Kohavi, R., 1995. *Wrappers for performance enhancement and oblivious decision graphs*. Doctoral dissertation. Stanford University.

Kordestanchi, H. & Naderi, H., 2013. Performance comparison study of language identification tools for identification of Farsi web pages. In *on Information and Knowledge Technology (IKT), 2013 5th Conference on*. Shiraz, pp. 489-494.

Kourkouta, L. & Papathanasiou, I.V., 2014. Communication in Nursing Practice. *Materia Socio-Medica*, 26(1), pp. 65-67.

Lamabam , P. & Chakma, K., 2016. A language identification system for code-mixed English-Manipuri Social Media text. In *2016 IEEE International Conference on Engineering and Technology (ICETECH)*. Coimbatore, India, IEEE, pp. 79-83.

Langa, R., Manamela, M.J. & Gasela, N., 2012. Synthesis of dialect speech for an under-resourced language. In *The Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*. George, pp. 160-161.

Laprie, Y. *et al.*, 2013. Articulatory copy synthesis from cine x-ray films. In *INTERSPEECH-2013*. Lyon, France, pp. 2024-2028.

Lemmetty, S., 1999. *Review of speech synthesis technology*. Master's thesis. Espoo, Finland: Helsinki University of Technology.

Lewis, M.P., Simons, G.F. & Fennig, C.D., 2016. *Ethnologue: Languages of the World*. [Online] Available at: http://www.ethnologue.com/ [Accessed 01 May 2016].

Li, Y.Q. *et al.*, 2012. A Visual-Audio Assisting System for Senior Citizen Reading. In *Computer Science and its Applications*. Springer Netherlands, pp. 667-675.

Llitjos, A.F. & Black, A.W., 2001. Knowledge of Language Origin Improves Pronunciation Accuracy of Proper Names. In *Proceedings of the EUROSPEECH*. Aalborg, pp. 1919-1922.

Louw, J.A., 2008. Speect: A multilingual text-to-speech system. In *Proceedings of the Nineteenth Annual Symposium of the Pattern Recognition Association of South Africa (PRASA)*. Cape Town, South Africa, pp. 165-168.

Louw, J.A., Davel, M. & Barnard, E., 2005. A general-purpose isiZulu speech synthesizer. *South African Journal of African Languages*, 25(2), pp. 92-100.

Mabokela, K.R. & Manamela, M.J., 2013. An integrated language identification for code-switched speech using decoded-phonemes and support vector machine. In *Speech Technology and Human - Computer Dialogue (SpeD), 2013 7th Conference on*. Cluj-Napoca, pp. 1-6.

Mhlana, S., 2011. *Development of isiXhosa text-to-speech modules to support e-services in marginalised rural areas*. Master's thesis. East London, South Africa: University of Fort Hare.

Mitchell, T.M., 1997. *Machine Learning*. International ed. McGraw Hill.

Mohasi, L., 2006. *Design of an advanced and fluent Sesotho text-to-speech system through intonation*. Master's thesis. Cape Town, South Africa: University of Cape Town.

Mullah, H.E., Pyrtuh, F. & Singh, L.J., 2015. Development of an HMM-based speech synthesis system for Indian English language. In *2015 International Symposium on Advanced Computing and Communication (ISACC)*. Silchar, India, IEEE, pp. 124-127.

Nicolaou, A., Bagdanov, A.D., Gomez-Bigorda, L. & Karatzas, D., 2016. Visual script and language identification. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*. Santorini, IEEE, pp. 393-398.

Pammi, S., Charfuelan, M. & Schröder, M., 2010. Multilingual Voice Creation Toolkit for the MARY TTS Platform. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*., pp. 19-21.

Pedregosa, F. *et al.*, 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* , 12, pp. 2825-2830.

Ramani, R. *et al.*, 2013. Vehicle Tracking and Locking System Based on. *International Journal of Intelligent Systems and Applications(IJISA)*, 5(9), pp. 86-93.

Rana, M.R., Akbar, M.A., Ahmad, T. & Gulfam, S., 2016. A supervised learning technique for language identification. *International Journal of Computing, Communication and Instrumentation Engineering (IJCCIE)*, 3(2), pp. 436-440.

Rao, K.S. & Nandi, D., 2015. *Language Identification Using Excitation Source Features*. Springer International Publishing.

Rousseau, F. & Mashao, D., 2005. A hybrid text-to-speech system for Afrikaans. In *Proceedings of South African Telecommunication Networks and Applications Conference (SATNAC)*. Central Drakensberg

Roux, J.C. *et al.*, 2010. Incorporating speech synthesis in the development of a mobile platform for e-learning. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation: LREC'10, 2010*. Valletta, Malta, pp. 1507-1510.

Schröder, M. & Breuer, S., 2004. XML reperentation languages as a way of interconnecting TTS modules. In *INTERSPEECH-2004*. Jeju Island, Korea, pp. 1889-1892.

Schröder, M., Charfuelan, M., Pammi, S. & Steiner, I., 2011. Open source voice creation toolkit for the MARY TTS platform. In *INTERSPEECH-2011*. Florence, Italy, pp. 3253-3256.

Schröder, M. & Trouvain, J., 2003. The German text-to-speech synthesis system MARY: A tool for research, development and teaching. *International Journal of Speech Technology*, 6(4), pp. 365-377.

Sefara, T.J., 2017. *Demo of The Automatic Pronunciation Assistant*. [Online] Available at: https://www.speechtech.co.za [Accessed 15 March 2017].

Sharma, B., Adiga, N. & Prasanna, S.R., 2015. Development of Assamese text-to-speech synthesis system. In *TENCON 2015 - 2015 IEEE Region 10 Conference*. Macao, China, IEEE, pp. 1-6.

Shilkrot, R. *et al.*, 2014. FingerReader: a wearable device to support text reading on the go. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems*. Ontario, Canada, ACM, pp. 2359-2364.

Shukla, M.K., Rana, A. & Banka, H., 2016. Classification of the Bangla script document using SVM. In *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*. Dhanbad, India, IEEE, pp. 182-185.

Singh, A. & Kaur, A., 2015. Case study of touch technology Used for teaching physically disabled students. In *MOOCs, Innovation and Technology in Education (MITE), 2015 IEEE 3rd International Conference on*. Amritsar, IEEE, pp. 392-395.

SPTK Working Group, 2012. *Speech Signal Processing Toolkit (SPTK) website*. [Online] Available at: http://sp-tk.sourceforge.net/ [Accessed 20 March 2016].

Stan, A., Yamagishi, J., King, S. & Aylett, M., 2011. The Romanian spech synthesis (RSS) corpus: Buiding a high quality HMM-based speech synthesis system using a high sampling rate. *Speech Communication*, 53(3), pp. 442-450.

Stavropoulou, P., Tsonos, D. & Kouroupetroglou, G., 2014. Language resources and evaluation for the support of the Greek language in the MARY text-to-speech. In *Text, Speech and Dialogue: 17th International Conference, TSD 2014, Brno, Czech Republic, September 8-12, 2014. Proceedings*. Springer International Publishing. pp. 523-528.

Suortti, O. & Lipponen, L., 2014. Phonological skills and ability to perceive auditorily the structure of a word at the level of a single phoneme at ages 2–6. *Journal of Early Childhood Literacy*, 14(4), pp. 510-533.

Suthaharan, S., 2016. Support Vector Machine. In *Machine Learning Models and Algorithms for Big Data Classification*. New York, US: Springer. pp. 207-235.

Taylor, P., 2009. *Text-to-Speech Synthesis*. New York, USA: Cambridge University Press.

Taylor, P., Black, A.W. & Caley, R., 1998. The architecture of the festival speech synthesis system. In *SSW3-1998*. NSW, Australia, pp. 147-152.

Tiomkin, S., Malah, D., Shechtman, S. & Kons, Z., 2011. A hybrid text-to-speech system that combines concatenative and statistical synthesis units. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(5), pp. 1278-1288.

Tokuda, K. *et al.*, 2016. *The HMM-based speech synthesis system (HTS)*. [Online] Available at: http://hts.sp.nitech.ac.jp [Accessed 19 March 2016].

Tokuda, K. *et al.*, 2011. *The HTS engine API*. [Online] (1.05) Available at: http://hts-engine.sourceforge.net/ [Accessed 10 March 2016].

Uddin, M.A. *et al.*, 2015. Phoneme based Bangla text to speech conversion. In *2015 18th International Conference on Computer and Information Technology (ICCIT)*. Dhaka, Bangladesh, IEEE, pp. 531-533.

Valentini-Botinhao, C., Wu, Z. & King, S., 2015. Towards minimum perceptual error training for DNN-based speech synthesis. In *INTERSPEECH-2015*. Dresden, Germany, pp. 869-873.

van den Oord, A. *et al.*, 2016. Wavenet: A generative model for raw audio. *Computing Research Repository (CoRR)*, abs/1609.03499.

van Heerden, C.J., 2012. *Efficient training of support vector machines and thier hyperparameters*. PhD thesis. Potchefstroom, South Africa: University of North-West.

Vasek, M., Rozinaj, G. & Rybárová, R., 2016. Letter-To-Sound conversion for speech synthesizer. In *2016 International Conference on Systems, Signals and Image Processing (IWSSIP)*. Bratislava, Slovakia, IEEE, pp. 1-4.

Violano, M. & van Collie, S.-C., 1992. *Retail Banking Technology: Strategies and Resources That Seize the Competitive Advantage*. New York: John Wiley & Sons.

Viswanathan, M. & Viswanathan, M., 2005. Measuring speech quality for text-to-speech systems: development and assessment of a modified mean opinion score (MOS) scale. *Computer Speech & Language*, 19(1), pp. 55-83.

Watts, O. *et al.*, 2016. From HMMs to DNNs: Where do the improvements come from? In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Shanghai, China, IEEE, pp. 5505-5509.

Wells, J.C., 2005. *SAMPA - computer readable phonetic alphabet*. [Online] Available at: http://www.phon.ucl.ac.uk/home/sampa/ [Accessed 6 April 2016].

Würgler, S., 2011. *Implementation and evaluation of an HMM-based speech generation component for the SVOX TTS system*. Master's thesis, Swiss Federal Intitute of Technology, Zurich.

Wu, Z., Valentini-Botinhao, C., Watts, O. & King, S., 2015. Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. South Brisbane, Australia, IEEE, pp. 4460-4464.

Wu, Z., Watts, O., King & King, S., 2016. Merlin: An open source neural network speech synthesis system. In *Proceedings of the 9th ISCA Speech Synthesis Workshop*. Sunnyvale, CA, pp. 202-207.

Yamagishi , J. & Kobayashi , T., 2007. Average-voice-based speech synthesis using HSMM-based speaker adaptation and adaptive training. *IEICE Transactions on Information and Systems*, 90(2), pp. 533-543.

Yamagishi, J. *et al.*, 2009. Analysis of Speaker Adaptation Algorithms for HMM-Based Speech Synthesis and a Constrained SMAPLR Adaptation Algorithm. *IEEE Transactions on Audio, Speech, and Language Processing*, pp. 66-83.

Yoshimura, T. *et al.*, 1999. Simultaneous modeling of spectrum, pitch and duration in HMM-based speech synthesis. In *Proc. Eurospeech*. Budapest, Hungary, pp. 2347-2350.

Yuan, G.-X., Ho, C.-H. & Lin, C.-J., 2012. An improved GLMNET for L1-regularized logistic regression. *Journal of Machine Learning Research*, 13, pp. 1999-2030.

Yu, J. & Wang, Z., 2016. A realistic and reliable 3D pronunciation visualization instruction system for computer-assisted language learning. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. Shenzhen, IEEE, pp. 786-789.

Zen, H. *et al.*, 2016. Fast, compact, and high quality LSTM-RNN based statistical parametric speech synthesizers for mobile devices. In *Proceedings of Interspeech 2016*. San Francisco, USA, pp. 2273-2277.

Zen, H. *et al.*, 2007. The HMM-based speech synthesis system (HTS) version 2.0. In *SSW6-2007*. Bonn, Germany, pp. 294-299.

Zen, H., Senior, A. & Schuster, M., 2013. Statistical parametric speech synthesis using deep neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Vancouver, Canada, IEEE, pp. 7962-7966.

Zen, H., Tokuda, K. & Black, A.W., 2009. Statistical parametric speech synthesis. *Speech Communication*, 51(11), pp. 1039-1064.