UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Project Final Paper 2023

Title: Transcribing isiXhosa SABC broadcast news using CMUSphinx

Author: Kristen Jodie Basson

Project Abbreviation: SABC2TXT

Supervisor(s): Prof. Hussein Suleman

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	0
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	20
System Development and Implementation	*0*	*20*	10
Results, Findings and Conclusions	*10*	*20*	20
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	0
Total marks		**80**	

Transcribing isiXhosa SABC broadcast news using CMUSphinx

Kristen Jodie Basson
Department of Computer Science
University of Cape Town
Cape Town, South Africa
bsskri003@myuct.ac.za

ABSTRACT

The lack of electronic linguistic resources in the southern Nguni languages, such as isiXhosa, negatively impacts computational and statistical systems that rely on these resources. Transcribing these languages would facilitate text-based research experiments and support multiple areas of natural language processing (NLP). This paper describes the methods used to create an isiXhosa automatic speech recognition (ASR) transcription system using the CMUSphinx speech recognition toolkit for South African Broadcasting Corporation (SABC) news. The accuracy of this system is evaluated by using metrics such as word error rate (WER), and the Levenshtein distance at the character and word level. Multiple experiments were conducted using a gold standard test corpus of approximately 40 minutes of SABC news and 3 hours of test audio provided by the South African Centre for Digital Language Resources (SADiLaR) website. The experiments done on the SABC news audio showed that the Levenshtein distances for the news anchor in the field were more accurate than the main news anchor and the distances for male speakers were more accurate than for females. Although Pocketsphinx performed better than Sphinx4 overall, both systems performed poorly on the SABC news audio, each with a WER of 100%. In comparison, the SADiLaR test audio had a WER of 39% for Pocketsphinx and 65% for Sphinx4.

KEYWORDS

isiXhosa Speech Recognition System, CMUSphinx, Acoustic model, Language model, Phonetic dictionary

1 INTRODUCTION

Nine out of the eleven official South African languages are drastically under-resourced, especially in terms of electronic resources or books [7]. This lack of electronic resources makes it difficult for these languages to be supported in search engines and other machine-learning tasks. It was not until recently that many of the South African languages had no data available to even develop an ASR system [6][9]. The first corpus developed to solve this problem was the AST corpus followed by the *Lwazi I* corpus [9]. Since then, the *Lwazi II* and *Lwazi III* corpora were developed as well as the National Centre for Human Language Technology (NCHLT) corpus [6]. The NCHLT corpus contains more than 50 hours of speech in each of the South African languages, available to support NLP.

Natural language processing (NLP) is a field in which computational techniques are used to automatically analyze text, to achieve human-like language processing for a variety of tasks [18]. A vital requirement for the continuation of research and development in the field is access to high-quality linguistic resources [8]. Therefore, the lack of high-quality text documentation in low-resource languages negatively impacts NLP [7]. Since ASR is a component of NLP where spoken words are transformed into text, also known as speech-to-text, it can be utilized to transcribe spoken speech [11]. This could potentially create valuable textual resources for low-resource languages to be further used in NLP and other research.

This project was divided into three sections where an1.) SABC news transcription system was developed for structured speech, 2.) live mobile speech transcription system was developed for unstructured speech and 3.) the gold standard corpus was developed to be used to test these transcription systems. This paper specifically investigates 1.) the accuracy of using CMUSphinx (Pocketsphinx and Sphinx4) in transcribing structured isiXhosa SABC news, where the metrics of WER and the Levenshtein word and character distances are used to measure the accuracy of transcription results. This paper comprises a brief background and relevant work section related to similar research, followed by details of the datasets, experimental methods and system design, the experimental results and discussion, and the conclusion drawn from the results.

2 BACKGROUND AND RELATED WORK

Multiple studies have been conducted on broadcast news transcription which initially focused on North American English but has since been expanded into several other languages [10]. The type of speech present in broadcast news ranges from noisy interviews to a structured, read from a manuscript, speech style. An example where a speech recognition transcription system was applied is the Turkish transcription system for broadcast news [5]. The data used to train the models used in that system was a large database of broadcast news collected from various radio and television channels. Those audio recordings were then transcribed and used as training data for the models required in the ASR system. Similarly, a Japanese transcription system was developed to provide real-time captions to their broadcasted news [12]. The models were trained in the same way as the Turkish transcription system by using the manuscripts accumulated over the years. This ensures that the system is trained on the same type of data that it will be transcribing, allowing fewer transcription errors to occur.

ASR transcription systems require a pronunciation dictionary, a language model and an acoustic model trained in the specific language. This is accomplished by using standard available speech recognition tools to train these models [1]. Some examples of toolkits available are the Hidden Markov Model toolkit (HTK), Kaldi toolkit, and the CMUSphinx toolkit [1]. Naidoo and Tsoeu [23] tested English and isiXhosa with HTK, CMUSphinx and Kaldi. The Kaldi toolkit had performed the best overall, yet the CMUSphinx toolkit performed the best for isiXhosa. Additionally,

there are other frameworks available that make use of deep neural networks (DNNs) to build an ASR system [1]. When a DNN was used in an ASR system where the language had limited resources, several studies noticed a significant increase in the performance of the system [14]. This is further motivated by Zhao and Zhang [22] who reviewed a hybrid DNN/HMM ASR system where the results confirmed that a DNN/HMM-based system had a better performance than an end-to-end model for under-resourced languages. Arisoy *et al.* [5] found that the phoneme error rates (PER) were also significantly less when using a DNN acoustic model. Although using DNNs for ASR systems for under-resourced languages could likely be beneficial, there are many complexities involved.

CMUSphinx is one of the standard toolkits available for building a speech recognition system and was developed in java by Carnegie Mellon University (CMU), Sun Microsystems Laboratories and Mitsubishi Research Laboratories. The CMUSphinx website also provides tutorials on using both Pocketsphinx and Sphinx4 in the transcription of speech [17]. It is worth noting that on the downloads page of the website, it states that the only tools currently being maintained are Pocketsphinx and Sphinxtrain [17]. To use either Pocketsphinx or Sphinx4 from CMUSphinx, a language and acoustic model as well as a phonetic dictionary are required. Although there are models provided by CMUSphinx, it is more commonly used with custom models that are built with the provided CMUSphinx tools. A study by Pantazoglou *et al.* [3] focused on the development of voice recognition tools for the CMUSphinx platform for the Greek language. The Greek speech recognition system was created using custom models. The language model was trained using the CMU language model toolkit (CMULMTK) while the acoustic model was trained using Sphinxtrain. Another study also made use of CMUSphinx toolkit for the speech-to-text conversion of the language of Kannada [16]. The metrics used were WER and sentence error rate (SER). These metrics were used to evaluate a context-dependent model, which refers to training and testing using one speaker, and a context-independent model, which is the training and testing of multiple speakers. A context-independent model required significantly more training data compared to the context-dependent model. The WER and SER of the context-independent model were also substantially higher. Segmentation could be used in this context before transcribing the audio where acoustic signals are further divided into homogenous segments [13][24]. The advantages of using segmentation include valuable information being extracted such as speaker turns and identities; speech discontinuity at speaker turns can be avoided and identified non-speech segments can be removed [2]. Segmentation can also significantly reduce computation time and simplify decoding.

CMUSphinx seems to be a widely used toolkit with a helpful website and tools to generate custom models. This is favourable since building a working transcription system can be challenging without the right available tools and documentation for guidance.

3 RESEARCH QUESTION AND PROBLEM DEFINITION

3.1 Problem definition

The lack of linguistic resources for the low-resource language of isiXhosa significantly impacts future research for NLP and the continued growth of the field. The importance of increasing the amount of text documentation available in the language is vital in facilitating text-based research experiments and other computational and statistical systems that require these linguistic resources. The possibility of automatically transcribing structured SABC broadcast speech for isiXhosa could potentially increase the number of textual resources available in the language.

3.2 Research question

How accurate is the use of CMUSphinx for transcribing the low-resource language of isiXhosa SABC news?

Accuracy in this context is determined by using the metrics of the WER and the Levenshtein word and character distances. This would give an accurate representation of how many words and characters were correctly transcribed by CMUSphinx.

4 DATASETS

SADiLaR hosts an array of freely available resources related to the eleven official languages of South Africa [19]. The datasets that were used were downloaded from the Language Resource Catalogue section on the site. The specific datasets that were obtained were the NCHLT isiXhosa speech corpus, the Lwazi isiXhosa TTS (text-to-speech) corpus, and the Lwazi II isiXhosa TTS corpus. These datasets contain multiple short audio recordings of a few seconds as well as a file containing each audio's transcription. This data is used to train the language and acoustic models for the ASR system. An additional dataset called the NCHLT-inlang pronunciation dictionaries was obtained and used in the creation of a phonetic dictionary. The ASR system's transcription accuracy is then tested using the 3 hours of test audio provided by the NCHLT corpus as well as the transcribed SABC news obtained from the developed gold standard corpus. This data consists of four, 10-minute audio files of SABC news with their transcriptions.

5 EXPERIMENTAL METHODS

For the ASR isiXhosa transcription system to work, an accurate language model and acoustic model of the language need to be built. The creation of these custom models is vital in facilitating the correct transcription results. Three iterations were completed before the final tests and experiments were done. The first iteration utilised only the NCHLT text-to-speech corpus for the training of the language and acoustic model. Therefore, to increase the amount of data used in the model training the second iteration combined the Lwazi I, Lwazi II and the NCHLT corpora. Finally, the third iteration was implemented to segment the audio into smaller sections to potentially increase the transcription results by passing in smaller audio segments before being transcribed by CMUSphinx (Pocketsphinx and Sphinx4).

5.1 Building a language model

A language model calculates the probability that a word will come next in a hypothetical sequence of words [3][16]. The creation of the isiXhosa language model was the first step in building an ASR transcription system. Datasets of audio files with their transcriptions were obtained from SADiLaR. The CMU language model toolkit (CMULMTK) was used to build the isiXhosa language model [15]. CMULMTK is available as a python library

therefore the commands can either be run in a script or separately in the terminal to generate a language model. By adhering to the instructions provided by the CMUSphinx website [17], and the CMU-Cambridge statistical language modelling toolkit website [15], the creation of a language model can be divided into data preparation and model training.

5.1.1 Data preparation

Transcriptions obtained from the datasets (Lwazi I, II and NCHLT) have training sets of data as well as one test set provided by the NCHLT dataset. The training sets are first combined into a single text file as one sentence per line. CMULMTK expects the text file to be normalized and each utterance to be delimited by <s> and </s> tags. Therefore, the text file is then run through a python script that automatically removes all punctuation marks, makes sure all the characters are lowercase and delimits utterances by the tags <s> and </s>. An example of the final correct text file format is shown in *Figure 1.*

<s> wokugqibela wengcambu yesenziwa </s>
<s> kuhlamba ngantelezi nasidlo </s>
<s> uneendawo ezifuna ukushiywa </s>

Figure 1: CMUCLMTK text file format

5.1.2 Language model training

Once this text file is in the correct format, either a python script can be used to run all the commands to train the language model or the commands can be run separately in the terminal. This process is shown in *Figure 2,* in a python script. The commands first generate a vocabulary file by using the text file shown in *Figure 1.* This vocabulary file is then used to generate an 'idngram' file, which is a list of every id n-gram, the information about the structure of words and phrases in a text dataset. This 'idngram' and vocabulary file are then used to generate the language model in an 'arpa' file format, a human-readable format that describes the statistical probabilities of the texts. However, this file needs to be in a "DMP" format, to be compatible with Sphinxtrain, to train an acoustic model, as well as in a binary format for CMUSphinx (Pocketsphinx and Sphinx4), to be used in transcription. Therefore, using a Linux computer or the Windows Subsystem for Linux (WSL) and having CMULMTK installed, a command, *"sphinx_lm_convert -i isixhosa.lm -o isixhosa.lm.bin",* can be run to convert the 'arpa' file format into the binary file format. Furthermore, *"sphinx_lm_convert -i isixhosa.lm -o isixhosa.lm.DMP",* can be used to convert the language model into the DMP file format. Thus, the language model has been built using CMULMTK and can be used in the training of the acoustic model.

```python
import cmuclmtk
import subprocess

#paths
text = "text_corpora.txt"
vocab = "isixhosa.vocab"
idngram = "isixhosa.idngram"
language_model = "isixhosa.lm"

# Read your text into a string variable
with open(text, 'r') as file:
    text_content = file.read()

# Create the vocabulary file using cmuclmtk function
cmuclmtk.text2vocab(text_content, vocab)

#TEXT2IDNGRAM
command = ['text2idngram', '-vocab', vocab, '-idngram', idngram, '-write_ascii']
subprocess.run(command, stdin=open(text, 'r'), check=True) #was sentences now text

#IDNGRAM2LM ARPA
command = ['idngram2lm', '-idngram', idngram, '-vocab', vocab, '-arpa', language_model,'-ascii_input']
subprocess.run(command)
```

Figure 2: Python script with CMULMTK commands to generate a language model.

5.2 Building an acoustic model

In ASR acoustic models are used to represent the relationship between an audio signal and the phonemes that constitute speech. The acoustic model for isiXhosa was built using Sphinxtrain developed by CMU [17]. Sphinxtrain acquires knowledge of the sound unit model parameters by analyzing a collection of example speech signals, within a specific file system structure. Sphinxtrain uses this structure to access all its key components needed during training. Additionally, for the training, WSL was used since the CMUSphinx website recommends that Sphinxtrain should be used on a Linux system for access to all its features. Some packages are further required to be installed including git, cmake (version 3.14 or higher), perl, python3, python3-numpy and python3-scipy. The process of training the acoustic model for isiXhosa can be broken down into steps.

5.2.1 Data preparation

5.2.1.1 File system structure

A specific file system structure is expected by Sphinxtrain. This structure represented in *Figure 3,* is used by Sphinxtrain to train the acoustic model. The Sphinxtrain and Pocketsphinx repositories are further required to be cloned from their GitHub site and then built into the file structure. The terminal commands to accomplish this are shown in the list below.

1.) git clone https://github.com/cmusphinx/sphinxtrain.git
2.) git clone https://github.com/cmusphinx/pocketsphinx.git
3.) cmake -S sphinxtrain -B sphinxtrain/build
4.) cmake --build sphinxtrain/build
5.) cmake" -S pocketsphinx -B pocketsphinx/build
6.) cmake --build pocketsphinx/build
7.) cp pocketsphinx/build/pocketsphinx_batch sphinxtrain/build

All the audio wav files for the datasets should be present under the 'wav' directory, partitioned by speaker. Furthermore, the language model in the DMP format should also be present in the 'etc' directory. The other files present are the training and testing audio files with a corresponding transcription file for each. Additionally, these testing and training audio files each need a 'fileid' file, containing the path to the audio recordings. Access to two dictionary files should be available: the phonetic and the filler dictionaries. The phonetic dictionary contains a list of the language's words linked to a sequence of sound units while the

filler dictionary associates non-speech sounds to non-speech or speech-like sound units. Lastly, there needs to be a phone file that contains the list of phones, the sound units of a language, used in the dictionaries.

```
├─ pocketsphinx
├─ sphinxtrain
├─ etc
│   ├─ isiXhosa.dic                    (Phonetic dictionary)
│   ├─ isiXhosa.phone                  (Phone file)
│   ├─ isiXhosa.lm.DMP                 (Language model)
│   ├─ isiXhosa.filler                 (List of fillers)
│   ├─ isiXhosa_train.fileids          (List of files for training)
│   ├─ isiXhosa_train.transcription    (Transcription for training)
│   ├─ isiXhosa_test.fileids           (List of files for testing)
│   └─ isiXhosa_test.transcription     (Transcription for testing)
└─ wav
    ├─ 001                             (Speaker)
    │   └─ file_1.wav                  (Recording of speech utterance)
    └─ 002
        └─ file_2.wav
```

Figure 3: isiXhosa file system structure for Sphinxtrain

5.2.1.2 Data pre-processing

Before adding all the files to the file system, the raw text-to-speech datasets were automatically pre-processed using python scripts. The training datasets, transcriptions are combined into a text file and then rewritten in the correct format expected by Sphinxtrain. Each utterance of the transcription file is delimited by <s> and </s> tags and the name of the audio file where the utterance is present is indicated after the </s> tag in parentheses. This format can be seen in *Figure 4,* with the utterance on the left between the tags and the transcription's audio file on the right in parentheses after the </s> tag. The testing datasets, transcriptions are similarly processed to reach the same format.

```
<s> ukuba sisiphi na </s> (nchlt_xho_001f_0009)
<s> yinto entle mpelele </s> (nchlt_xho_001f_0010)
<s> asekwe phezu kwaloo </s> (nchlt_xho_001f_0014)
<s> nalo ke eli </s> (nchlt_xho_001f_0016)
```

Figure 4: isiXhosa transcription file format for Sphinxtrain

Once the transcription files have been created the file ids are then determined. There are two 'fileid' files, one for the training data and another for the testing data. These files contain the path to each audio file relative to the 'wav' directory. The 'fileid' file format is represented in *Figure 5.*

```
001/nchlt_xho_001f_0001
001/nchlt_xho_001f_0002
001/nchlt_xho_001f_0003
001/nchlt_xho_001f_0004
```

Figure 5: isiXhosa fileid file format for Sphinxtrain

The two dictionaries present in the 'etc' directory are 'isiXhosa.dic' which is the phonetic dictionary and 'isiXhosa.filler' which is the filler dictionary. Following the CMUSphinx website, the isiXhosa filler dictionary was set to contain only silences as represented in *Figure 6.*

```
<s>     SIL
</s>    SIL
<sil>   SIL
```

Figure 6: isiXhosa filler dictionary file

A phonetic dictionary was available and utilized from the NCHLT dataset. However, this dictionary was incomplete and had to be checked to ensure that all the words in the transcription file were present in the dictionary accompanied by their phonetic transcription. Therefore, a grapheme-to-phoneme (G2P) model would need to be trained to ensure a complete dictionary. To accomplish this the following steps were taken: 1.) The training transcription file was compared to the current phonetic dictionary and any word present in the transcriptions but not in the dictionary was written to a text file. It was made sure that there were only unique words being written to the text file and no duplicates. 2.) The Sequitur G2P python library was installed and the command to train a G2P model was run four times to ensure an accurate model was created [20]. These commands made use of the current phonetic dictionary to train the model and were run on a Linux system. The commands to train the G2P model were run in the order below.

- g2p.py --train current.dic --devel 5% --write-model model-1
- g2p.py --model model-1 --ramp-up --train current.dic --devel 5% --write-model model-2
- g2p.py --model model-2 --ramp-up --train current.dic --devel 5% --write-model model-3
- g2p.py --model model-3 --ramp-up --train current.dic --devel 5% --write-model model-4

3.) The last command, *"g2p.py --model model-4 --apply absent_words.txt",* was run that uses the text file of absent words and the G2P model that was created to associate the words with their phonetic transcriptions. 4.) These newly associated words were then added to the current dictionary to create the completed phonetic dictionary for isiXhosa. The format of the dictionary is represented in *Figure 7,* indicating the word on the left and the phonetic transcription on the right.

```
ababini         a b_< a b_< i n i
ababodwa        a b_< a b_< O d w a
ababona         a b_< a b_< O n a
ababonakala     a b_< a b_< O n a k_> a l a
```

Figure 7: isiXhosa phonetic dictionary file format for Sphinxtrain

Finally, the 'isiXhosa.phone' file present in the 'etc' directory is the list of phones used in the phonetic dictionary. Every phone present in the phonetic dictionary must be present in the phone file as well as the silence phone present in the filler dictionary. This dictionary format is shown in *Figure 8,* with each phone transcription listed on a new line.

```
k_>
p_>
j
g
tK_>
SIL
```

Figure 8: isiXhosa phone file format for Sphinxtrain

Therefore, once pre-processing has been completed, all the files in the 'etc' directory should be present to be further used in training the acoustic model.

5.2.2 Acoustic model training

To start training the command *"python3 sphinxtrain/scripts/sphinxtrain -t isixhosa setup"* first needs to be run in the isiXhosa directory. This command copies all the required configuration files into the 'etc' directory of the isiXhosa file system and automatically sets the paths needed in the 'sphinx_train.cfg' configuration file. Inside this file, some configurations were edited to configure parallel jobs to speed up the training process. The following configurations were changed:

- $CFG_QUEUE_TYPE = "Queue::POSIX", specifies a multicore machine.
- $CFG_NPART = 10 and $CFG_DONE = 10, specifies the number of parallel processes to run. The recommended number is 10 for an 8-core machine.

Once this is done Sphinxtrain is run from the isiXhosa directory with *"sphinxtrain/scripts/sphinxtrain run"*. This starts the acoustic model training process which takes a few hours to complete depending on the computer's specifications such as CPU power and RAM. On completion, a WER of 30.9% was reached as well as a sentence error rate (SER) of 34.9% by using the test data provided by the NCHLT corpus.

5.3 Audio segmentation and file processing

Audio segmentation was implemented through a python script before decoding to potentially increase the accuracy of transcriptions. The libraries used in the audio segmentation were the numpy, and soundfile python libraries which segmented the audio into smaller sections by the energy level of the speech. This ensures that segmentation occurs when the energy level of the speech is low, thus, segmenting on breaks between speech. Although this somewhat worked the audio still manually needed to be segmented. Further segmentation was done manually by either dividing longer segments, that did not segment at clear pauses, or joining shorter segments that incorrectly split on a word without a clear pause between the speech. This improved the flow of the audio so that each segment was approximately the same length, and that each word was clear for transcribing purposes. Once the audio segmentation was complete, the source transcription files from the gold standard SABC news corpus were cleaned, by ensuring that all characters were lowercase, and no punctuation occurred. Its text file contained each audio segment's speech on a new line. This provided an easy way to later compare the source text file to the hypothesis text file line by line to calculate the metrics of how correct the hypothesis transcriptions were.

5.4 System development and implementation

The audio was transcribed using both Pocketsphinx and Sphinx4 speech recognition libraries. Therefore, the development and implementation of both systems will be discussed.

5.4.1 Pocketsphinx

Since Pocketsphinx has a python speech recognition library, python was chosen as the language to develop and implement the transcription system. Visual Studio Code was used as the integrated development environment (IDE) as it supports python and is easy to use. The Pocketsphinx python speech recognition library was therefore installed and the paths to the acoustic model, language model and phonetic dictionary were specified. As the four ten-minute SABC news audio files were segmented, the code had to ensure that each audio file segment was being transcribed in the correct order. The code takes an input folder that contains the segments of the audio file and transcribes each segment separately in order of their file name. These segment transcriptions are then written to a text file line by line to later accommodate metric comparison between the source and hypothesis transcription files. The code is shown in *Figure 9.*

```python
import os
from pocketsphinx import AudioFile, Pocketsphinx

def transcribe_audio(hmm, lm, dic, audio_file):
    config = {
        'verbose': False,
        'audio_file': audio_file,
        'buffer_size': 2048,
        'no_search': False,
        'full_utt': True,  # Transcribe the full utterance
        'hmm': hmm,
        'lm': lm,
        'dict': dic
    }

    speech = AudioFile(**config)

    transcription = ""
    for phrase in speech:
        print(phrase)
        transcription += str(phrase) + " "

    return transcription

def main():
    hmm = 'isixhosa.cd_cont_200'  # Acoustic model
    lm = 'isixhosa.lm.bin'  # Language model
    dic = 'isixhosa.dic'  # The phonetic dictionary
    input_folder = 'final_segments_6'  # Folder containing audio files
    output_file = 'transcriptPS_6.txt'  # The output transcript file


    # Get a sorted list of '.wav' files in the folder
    wav_files = sorted([file_name for file_name in os.listdir(input_folder) if file_name.endswith('.wav')])
            for file_name in wav_files:
                audio_file = os.path.join(input_folder, file_name)
                transcription = transcribe_audio(hmm, lm, dic, audio_file)
                file.write(transcription + "\n")

if __name__ == "__main__":
    main()
```

Figure 9: Pocketsphinx python application in Visual Studio Code

5.4.2 Sphinx4

On the contrary to Pocketsphinx, Sphinx4 is a pure java speech recognition library [17]. CMUSphinx's website has a basic tutorial showing how to code a basic transcription system for Sphinx4. Since Sphinx4 is a pure Java speech recognition library the website recommends using the Eclipse IDE as it supports Gradle. Sphinx4 then requires its three core jar files to be added to the classpath. These were downloaded from the Nexus repository manager where they were easily found by typing 'Sphinx4' into the search bar [25]. These jars included:

- sphinx4-core-5prealpha-20160628.232526-10.jar
- sphinx4-core-5prealpha-20160628.232526-10-javadoc.jar
- sphinx4-core-5prealpha-20160628.232526-10-sources.jar

To add these jar files to the classpath in Eclipse, the package explorer was opened on the left-hand side in Eclipse, the package 'isiXhosaTranscriber' was right-clicked and the properties option was selected, 'Java build path' was then selected and under 'Classpath' the external jar files were uploaded. After this was set the basic code from the website was used as a starting point where the paths to the acoustic model, language model and phonetic dictionary were specified. The code was then edited similarly to Pocketsphinx where a folder containing the audio file segments was used as input. The code then specifies that the segments need to be transcribed in order of their file names. Each segment's transcription is then written line by line to a text file for later metric comparison to the source transcriptions. In *Figure 10,* the Sphinx4 application is presented.

```
package isiXhosaTranscriber;

import java.io.*;
import java.util.Arrays;
import java.util.Comparator;

import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;

import edu.cmu.sphinx.api.Configuration;
import edu.cmu.sphinx.api.SpeechResult;
import edu.cmu.sphinx.api.StreamSpeechRecognizer;

public class Transcriber {

    public static void main(String[] args) throws Exception {

        Configuration configuration = new Configuration();

        configuration.setAcousticModelPath("/isixhosa.cd_cont_200");
        configuration.setDictionaryPath("isixhosa.dic");
        configuration.setLanguageModelPath("isixhosa.lm.bin");

        File inputFolder = new File("final_segments_6");
        File[] audioFiles = inputFolder.listFiles((dir, name) -> name.endsWith(".wav"));
        Arrays.sort(audioFiles, Comparator.comparing(File::getName));

        FileWriter writer = new FileWriter("transcriptS4_11.txt");

        for (File audioFile : audioFiles) {
            InputStream stream = new FileInputStream(audioFile);

            StreamSpeechRecognizer recognizer = new StreamSpeechRecognizer(configuration);
            recognizer.startRecognition(stream);

            StringBuilder transcriptionBuilder = new StringBuilder();

            SpeechResult result;
            while ((result = recognizer.getResult()) != null) {
                String transcription = result.getHypothesis();
                transcriptionBuilder.append(transcription).append(" ");
            }

            writer.write(transcriptionBuilder.toString().trim() + "\n"); // Write the transc

            recognizer.stopRecognition();
        }

        // Close the FileWriter
        writer.close();
    }
}
```

Figure 10: Sphinx4 java application in Eclipse

5.5 Evaluation metrics

Specific metrics are used to determine the accuracy of the Pocketsphinx and Sphinx4 speech transcription systems. The WER is calculated using *Equation 1* and measures the number of errors that occurred during the transcription of an audio signal divided by the total number of words in the source transcript [4]. These errors are either an incorrect, insertion, deletion, or substitution of a word in the transcription. The Levenshtein distance (LD) is calculated using *Equation 2* and measures the number of alterations (substitutions, deletions, or insertions) that need to take place in a target sequence of words so that it is identical to the source sequence [21]. The LD was calculated at the word and character level, therefore, *Equation 2.* can either be the number of alterations per word or character in a sequence.

$$WER = \frac{Insertions + Deletions + Substitutions}{total\ number\ of\ words\ in\ source\ transcript} \quad (1)$$

$$LD = Insertions + Deletions + Subtitutions \quad (2)$$

5.6 Experimental setup

For the experimental setup, both Sphinx4 and Pocketsphinx speech recognition tools were utilised to transcribe all four audios separately. Audio from the training data was transcribed several times to make sure the transcription outcome was the same each time, therefore an average over the transcriptions would not need to be taken and the audio can be transcribed once. The data that was experimented with was the 3 hours of test data provided by the NCHLT dataset and the gold standard SABC news dataset of 40 minutes that comprises four audios of ten minutes each. The four SABC news audios were experimented with in various ways. To accomplish these experiments categories of comparisons were created. These category comparisons included: *1.) Comparing Pocketsphinx and Sphinx4 over all the audios. 2.) Comparing male and female speakers. 3.) Comparing different types of speech.* These comparisons were done separately for each audio file over their respective segments. The different types of speech can be further expanded into: news anchor, being the main anchor conveying the broadcast news; field news, the secondary anchor in the field, although this audio might contain background noise the speaker is always clearly heard; interview, a journalist interviewing another person where there may be background noise and the interviewee may use words such as "uh" or "uhm"; weather speech, the anchor conveying the weather forecast. The segments with their associated source transcriptions of each audio file were divided into the specific category files (gender or type) to make the process of transcription and evaluation easier. These categories were then transcribed, and the transcriptions were stored in text files. A python script was developed to automatically calculate the WER and Levenshtein distances at the word and character level between the source and hypothesis transcriptions. Additionally using these metrics and python scripts, making use of matplotlib and numpy libraries, in Jupyter Notebook, the average (mean) WER and Levenshtein character and word distances with the standard deviation as an error bar were plotted.

6 RESULTS

Throughout the experimental results, a comparison between Sphinx4 and Pocketsphinx is given to show how each speech recognition system performs on the same audio data. The metrics of the test data from the NCHLT dataset are graphed to show how the systems perform on the same type of words and speech that the system was trained on. The metrics for the SABC news corpus then shows how the systems perform on unseen data. The average WER for all the experiments conducted on the SABC news corpus was 1, indicating a 100% WER, meaning the number of errors equalled the number of words.

6.1 NCHLT test data

In *Figure 11*, Pocketsphinx and Sphinx4 were tested on the 3 hours of NCHLT test data where Pocketsphinx resulted in a WER of 39% which is much better in comparison to Sphinx4 which had a WER of 65%. An explanation for Sphinx4's poor result could be due to

it no longer being maintained where Pocketsphinx is actively being maintained [17]. Another reason is that when Sphinx4 transcribed the audio, some lines in the text file were left blank indicating it did not transcribe some audio segments resulting in a 100% WER for those lines. The standard deviation of both indicates that the data is clustered around the mean, therefore there is little variance within the data values.

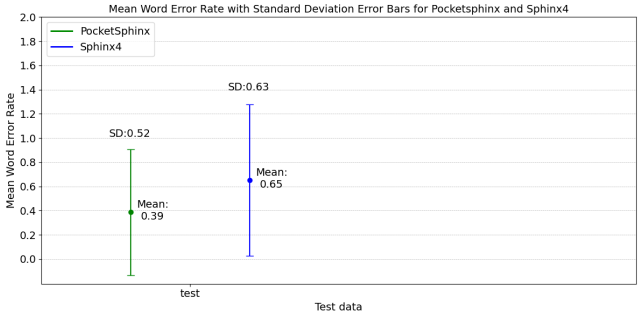

Figure 11: Mean Word Error Rate for Sphinx4 and Pocketphinx for the NCHLT test data

6.2 Gold standard SABC news data

Pocketsphinx and Sphinx4 were tested on all four ten-minute audios where the WER resulted in an average of 1 which is 100% across all the audios. The experiments done in this section are the comparisons between Sphinx4 and Pocketsphinx, male and female speakers and different types of speech.

6.2.1 Pocketsphinx and Sphinx4

Figure 14 and *Figure 15* shows the results for the Levenshtein distance at the character and word level for both Pocketsphinx and Sphinx4. In *Figure 14* and *Figure 15,* both systems performed, more or less, equally where the overall average Levenshtein character distance for Pocketsphinx was 103.75 and Sphinx4 was 103.58 and the overall average Levenshtein word distance for Pocketsphinx was 16.85 where Sphinx4 is 18.05.

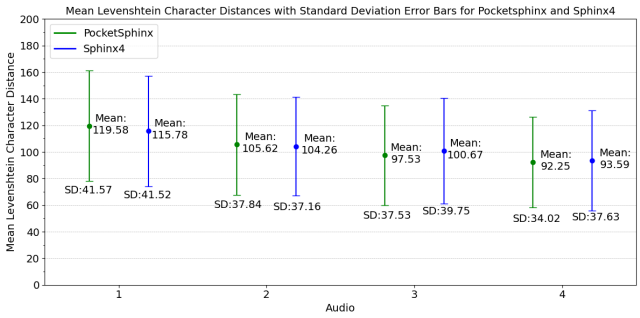

Figure 14: Mean Levenshtein character distances for Pocketsphinx and Sphinx 4

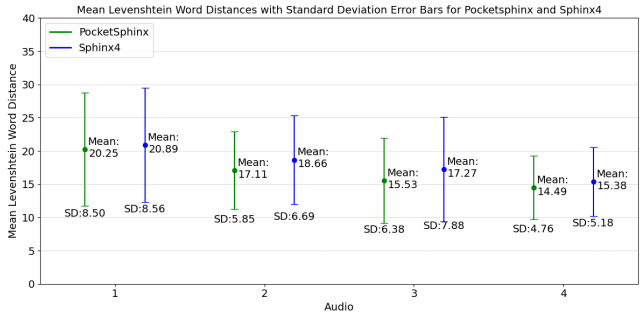

Figure 15: Mean Levenshtein word distances for Pocketsphinx and Sphinx4

6.2.2 Speaker gender

Each audio file contained both male and female speakers where the Levenshtein character distances for Pocketsphinx and Sphinx4 are represented in *Figure 16* and *Figure 17* respectively. In *Figure 16* the average character distance for the female speakers across all four audio's was 108.23 whereas for the male speakers it was 86.75. *Figure 17* had an average character distance for female speakers of 109.27 where male speakers was 86.85. Once again it can be seen that the performance of both systems is approximately the same. There is however a clear indication that speech recognition performs better on audio where the speaker present is male. This is observed for both system's character distances. *Figure 18* and *Figure 19* further represent the Levenshtein word distances for Pocketsphinx and Sphinx4. *Figure 18* had an average word distance of 17.62 for female speakers and 14.03 for male speakers. In *Figure 19*, the average overall word distance for female speakers was 19.12 and for male speakers was 15.11. Similarly, to the character distances results the word distances show that the speech recognition system performs much better on audio containing male speakers. The standard deviation between all these results shows a large variability between data indicating that the data is widely spread out and not clustered close to the mean.

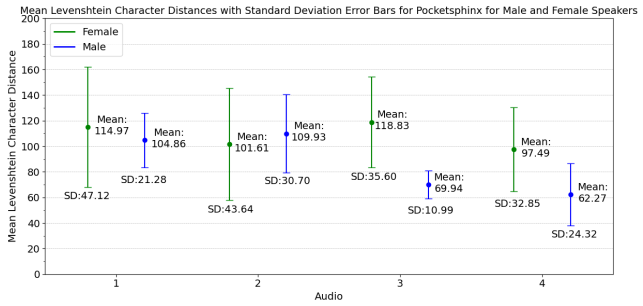

Figure 16: Mean Levenshtein character distances for Pocketsphinx for male and female speakers

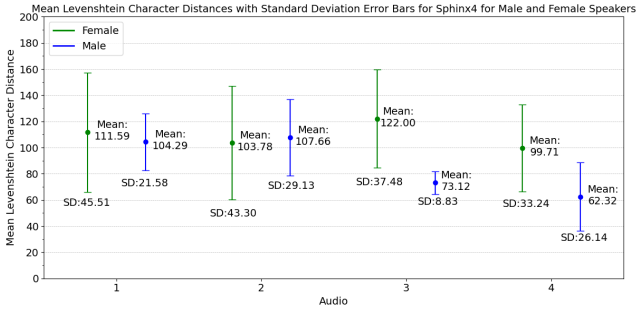

Figure 17: Mean Levenshtein character distances for Sphinx4 for male and female speakers

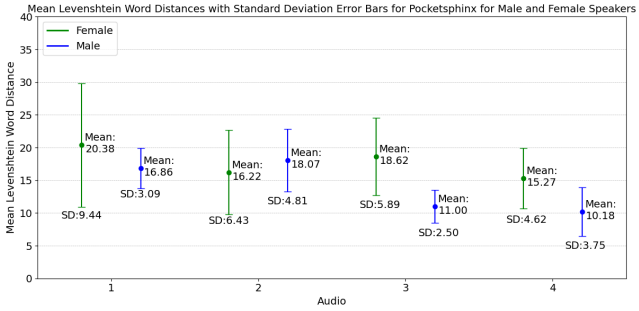

Figure 18: Mean Levenshtein word distances for Pocketsphinx for male and female speakers

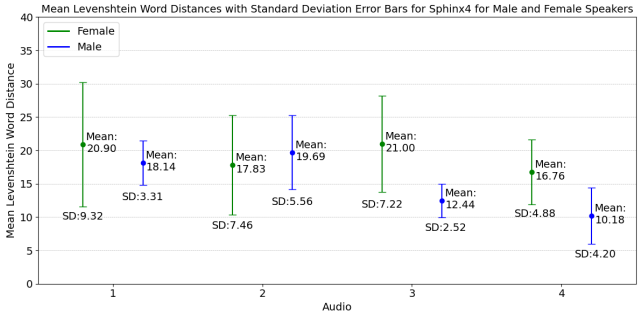

Figure 19: Mean Levenshtein word distances for Sphinx4 for male and female speakers

6.2.3 Type of speech

Comparisons between different types of speech include news anchor, which is the main news anchor conveying the broadcast news; field news, the secondary anchor in the field, although this audio may contain background noise the speaker is always clearly heard; interview, a journalist interviewing another person where there may be background noise and code-switching, using the words of another language; weather, the anchor conveying the weather forecast, only audio 1 contained a weather forecast and the speaker spoke much faster than in the other types of speech. *Figure 20* and *Figure 21* represent the Levenshetin character distances for Pocketsphinx and Sphinx4 while *Figure 22* and *Figure 23* show the word distances for Pocketsphinx and Sphinx4.

For all four figures, it is seen that the speech of the news anchor conveying the weather forecast had the worst character and word distance across the various types of speech. The type of speech that performed the best overall was the field news. This can be observed from the figures.

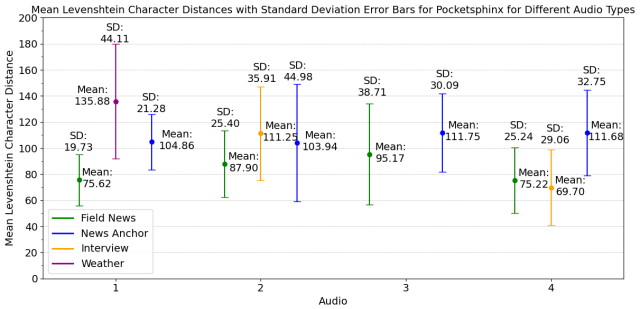

Figure 20: Mean Levenshtein character distances for Pocketsphinx for different types of speech

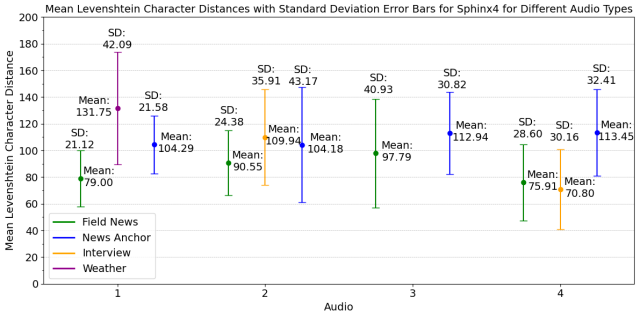

Figure 21: Mean Levenshtein character distances for Sphinx4 for different types of speech

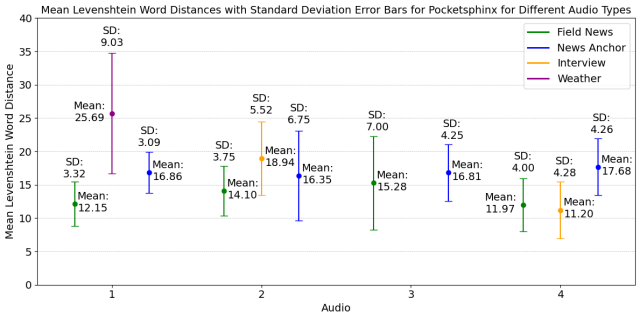

Figure 22: Mean Levenshtein word distances for Pocketsphinx for different types of speech

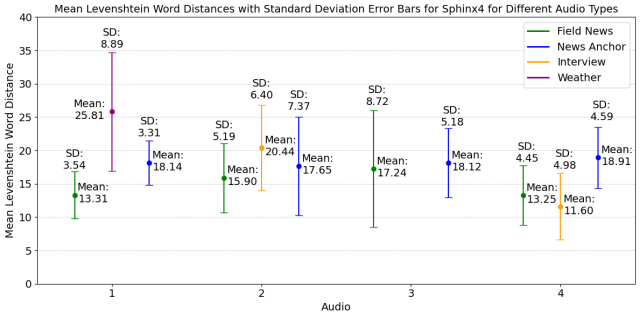

Figure 23: Mean Levenshtein word distances for Sphinx4 for different types of speech

7 DISCUSSION

From the results, it is observed that Pocketsphinx performs better than Sphinx4 on seen data otherwise both systems perform equally on unseen data. This could be due to Sphinx4 no longer being maintained by CMUSphinx. The only tools currently being maintained are Pocketsphinx and Sphinxtrain. Overall CMUSphinx performed poorly in transcribing isiXhosa SABC broadcast news using the SADiLaR datasets. Some of the reasons for this could be that: the datasets used for training the models were too small; the model was trained with too short segments of audio no longer than 30 seconds; the audio used for training, the speakers spoke slowly and clearly where the test audio the speakers spoke fast. In general, both systems did not adapt well to unseen data. Another reason for this was that no handling of out-of-vocabulary (OOV) words, words that are not present in the phonetic dictionary, was implemented into the system. When this isiXhosa speech recognition system runs into an OOV word it will either transcribe it to the most probable word it thinks it is or skip the word entirely. The study on the Japanese and Turkish broadcast news transcription systems both stated that a large amount of broadcast news audio and manuscripts were collected over several years that were used in the training of the speech recognition system [5][12]. Using similar training data to the data you would need to transcribe would improve the ASR system accuracy since the new data to be transcribed would not be unseen and most of the words would not be OOV words. Furthermore, the ASR system performs better on male speech segments than on female speech segments. This could be due to the pitch difference in the various voices. Interestingly, the field news speech performed better than the main news anchor speech sections, this is perhaps due to these sections' speakers speaking slower and more clearly. Creating an accurate speech recognition system for isiXhosa SABC news would need a large amount of audio and textual data related to SABC news for the training of the models for the ASR system. Other toolkits and deep neural network-based systems, although complex, should be explored and experimented with.

8 CONCLUSIONS

Although the experimental results showed that Pocketsphinx's WER for the NCHLT test data was 39% in comparison to Sphinx4 with 65%, both systems performed poorly on the SABC news, unseen data, with a WER of 100%. Furthermore, when comparing male and female speakers it was seen that both systems performed better on male speech than female. Moreover, the field news had the best results amongst the different types of speech, with a better result than the main news anchor sections. The standard deviation for all the results showed a large variability between the data indicating that the data was widely spread out and not clustered close to the mean. Overall, the results show that CMUSphinx (Pocketsphinx and Sphinx4) does not perform accurately for isiXhosa SABC news.

REFERENCES

[1] R. Jimerson, K. Simha, R. Ptucha, and E. Prud'hommeaux, "Improving ASR output for endangered language documentation," *The 6th intl. workshop on spoken language technologies for under-resourced languages,* 2018.

[2] J.-L Gauvain, L. Lamel, and G. Adda, "Transcribing broadcast news for audio and video indexing," *Communications of the ACM*, vol. 43, no. 2, pp. 64-70, 2000.

[3] F.K. Pantazoglou, N.K. Papadakis, G.P. Kladis, "Implementation of the generic Greek Model for CMU Sphinx speech recognition toolkit," *Proceedings of eRA-12,* pp. 1-11, 2017.

[4] C. Srun, V. Ny, C. Cheat, S. Ching, P. Ny, "Development of Speech Recognition System Based on CMUSphinx for Khmer Language," *International Journal of Innovative Science and Research Technology*, vol. 6, no. 8, pp. 770-775, 2021.

[5] E. Arisoy, D. Can, S. Parlak, H. Sak, and M. Saraçlar, "Turkish broadcast news transcription and retrieval," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 5, pp. 874-883, 2009.

[6] E. Barnard, M. Davel, C. van Heerden, F. De Wet, and J. Badenhorst, "The NCHLT speech corpus of the South African languages," *4th International Workshop on Spoken Language Technologies for Under-Resourced Languages,* pp. 194-200, 2014.

[7] L. Besacier, E. Barnard, A. Karpov, and T. Schultz, "Automatic speech recognition for under-resourced languages: A survey," *Speech communication*, vol. 56, pp. 85-100, 2014.

[8] R. Eiselen, and M. J. Puttkammer, "Developing Text Resources for Ten South African Languages," *LREC*, pp. 3698-3702, 2014.

[9] D. Henselmans, T. Niesler, and D. Van Leeuwen, "Baseline speech recognition of South African languages using Lwazi and AST," *in Proceedings of the 24th Annual Symposium of the Pattern Recognition Association of South Africa,* 2013.

[10] H. Kamper, F. De Wet, T. Hain, and T. Niesler, "Resource development and experiments in automatic SA broadcast news transcription," in *Workshop on Spoken Technologies for Under-Resourced Languages,* pp. 102-106, 2012.

[11] F.S. Al-Ansi and D. Abuzeina, "Performance evaluation of Sphinx and HTK speech recognizers for spoken Arabic language," *International Journal of Innovative Computing, Information and Control,* vol. 15, no. 3, pp. 1009-1021, 2019.

[12] A. Ando, T. Imai, A. Kobayashi, H. Isono, and K. Nakabayashi, "Real-time transcription system for simultaneous subtitling of Japanese broadcast news programs," in *IEEE Transactions on Broadcasting*, vol. 46, no. 3, pp. 189-196, 2000.

[13] P. Deléglise, Y. Estève, S. Meignier, and T. Merlin, "The LIUM speech transcription system: a CMUSphinx III-based system for French broadcast news," in *Interspeech,* 2005.

[14] M.J.F. Gales, K.M. Knill, A. Ragni, and S.P. Rath, "Speech recognition and keyword spotting for low-resource languages: Babel project research at cued," *International Speech Communication Association (ISCA),* 2014.

[15] P. Clarkson, 2023. The CMU-cambridge statistical language modeling toolkit V2, *Language Modeling Toolkit*. Retrieved August 23, 2023 from https://people.csail.mit.edu/joe/sctk1.2/src/slm_v2/doc/toolkit_documentation.html

[16] K.M. Shivakumar, K.G. Aravind, T.V. Anoop, "Kannada Speech to Text Conversion Using CMU Sphinx," *International Conference on Inventive Computation Technologies (ICICT),* 2016.

[17] Shmyrev, N. 2023. CMUSPHINX documentation, CMUSphinx Open Source Speech Recognition. Retrieved August 27, 2023 from https://cmusphinx.github.io/wiki/

[18] Elizabeth D. Liddy. 2001. Natural language processing. (2001).

[19] 2023. SADiLaR *Resource catalogue*. Retrieved August 27, 2023 from https://repo.sadilar.org/handle/20.500.12185/7

[20] 2023. Sequitur-g2p PyPI. Retrieved August 27, 2023 from https://pypi.org/project/sequitur-g2p/

[21] R. Hjulström, "Evaluation of a speech recognition system,".

[22] J. Zhao, and W.-Q. Zhang, "Improving Automatic Speech Recognition Performance for Low-Resource Language with Self-Supervised Models," *IEEE Journal of Selected Topics in Signal Processing*, vol 16, no 6, pp. 1227-1241, 2022.

[23] Naidoo, A. and Tsoeu, M., "Evaluating Open-source Toolkits for Automatic Speech Recognition of South African Languages," *IEEE SAUPEC/RobMech/PRASA Conference,* 2019.

[24] L. Lamel, J.-L. Gauvain, G. Adda, M. Adda-Decker, L. Canseco, L. Chen, O. Galibert, A. Messaoudi, and H. Schwenk, "Speech transcription in multiple languages," in *IEEE International Conference on Acoustics, Speech, and Signal Processing,* pp. 757-760, 2004.

[25] *Nexus Repository Manager*. Retrieved September 12, 2023 from https://oss.sonatype.org/