Chatbot in isiXhosa for Remote Pre/post-Natal Care

Carla Wilby* Amit Kumar Mishra†

Abstract

In this work we present the development of an isiXhosa ChatBot, a ChatBot developed for pre and postnatal care in the Southern African language of isiXhosa. Multiple implementations of User Interfaces where developed - notably the Android Mobile Application Implementation and the WhatsApp Integration Implementation. We deem this as an enabler for providing first-hand health-care advisors in remote African communities.

Keywords: Chatbot, NLP, African Language, Sustainable Development, AI

1 Introduction

Availability of health-care professionals in remote African villages is still a major challenge. However, the penetration of mobile-phones have been pretty successful in the continent. This has created ripe grounds for the development of online health-care platforms which can be useful in taking care of the initial health-care related queries. ChatBot technology, in this context, is defined as software solutions for device-user communication. Chatbots have been popular in modern software after the introduction of a number of open-source chatbot development platforms. However, most of these have been developed for English speaking users. Most African language grammar differ substantially from English and hence the development of chatbots in African languages is particularly challenging.

In our current work we have developed a chatbot for pre and post-natal care in isiXhosa which is a major language spoken in Southern Africa. The technologies needed to develop a project of this nature include language tree specific Natural Language Processing (NLP) algorithms, Xhosa corpus datasets, and ChatBot implementation technologies to assess the potential for the development of an isiXhosa ChatBot.

*With Zelda Learning Technologies

†With University of Cape Town. Email: akmishra@ieee.org

Multiple implementations of User Interfaces where developed - notably the Android Mobile Application Implementation and the WhatsApp Integration Implementation.

One of the major challenge in this project was the fact that implementing an isiXhosa ChatBot is very difficult on a conceptual NLP level. However, available technologies prove that it is possible to develop a purpose-driven mobile application ChatBot.

The potential of this technology, with the FAQ nature, could be extended to legal advice, access to emergency healthcare assistance and career and personal guidance.

The rest of the paper is organized as follows. In Section 2 we describe the challenges for such a project and the pieces of existing open-source platforms which we have used in the project. Section 3 presents the development of the project. We show the results in Section 4 and conclude in Section 5. For a more detailed treatment of the work, the readers are requested to refer the final thesis which can be found at `https://tinyurl.com/y4xwxzp2`.

2 Motivation

The linguistic data needed to develop the *ontology* (practical language semantics and formation) and *corpus* (structured set of language texts) of these languages is heavily under-resourced and has fallen well behind that of other family trees. The availability and quantity of *lexicographic* (language specific word definitions) data-sets is a requirement in the development of language technologies such as NLP [1];.

Furthermore, due to growing multilingualism and the evolution of complex dialects, the implementation of language models is difficult South African context. Not only are South African NLP algorithms not feasible to implement using existing rule-based algorithms due to syntactic complexity, but linguistic resources are still not available in large enough quantities to enable big data probabilistic models.

The ChatBot will be trained for a specific range of data as a proof of concept, in this case Pre- and Postnatal Maternal Health FAQs will be used as the training dataset.

The purpose of the study is to determine the viability and efficacy of developing a purpose-driven mobile application ChatBot using existing technologies. This includes the research, design and development of a Natural Language Processing algorithm using existing technologies, for conversational isiXhosa, such that a ChatBot would be able to understand and respond in the same language.

[1] M. Keet. "An assessment of orthographic similarity measures for several African languages". In: *University of Cape Town* (2016)

3 Previous Work and Available Tools

The primary component of the NLP implementation is FeersumNLU, a local South African language processing solution. Feersum NLU is designed to be language-agnostic, which means it can be scaled to work with any language, even in markets where large bodies of labelled data do not exist. FeersumNLU is developed on open-source building blocks like NLTK, skilearn, PyTorch and Duckling. Prototype conversations are purposefully quick to set up and test, allowing for multiple design improvements and extensive user testing. The Python library `Scikit-learn` was used as it provides useful tools for implementing Text Feature Extraction. The Feersum API interfacing application component called `faq_matchers_api` was developed to train and query an NLP model. This allows the system to firstly build a model, before receive and request queries to the model through the Feersum API. Feersum NLU is particularly suitable for this implementation because of it's wide selection of language wrappers for the API, and a specific JSON-REST API library for FAQ labelling and matching[2];.

The the Python Natural Language Toolkit (NLTK) provides an open source platform for language processing implementation. It provides libraries for classification, POS tagging, tokenization, syntactic analysis.Tools for implementing core NLP components from first principles. In order to emulate the service at user-scale, the Google Cloud Platform (GCP) provides platforms and services - such as Cloud IAM, Cloud PubSub and Cloud Functions - all the tools that are necessary to deploy an Android App - based ChatBot. The mobile platform Firebase was selected as a suitable choice for User Authentication and Datastorage, in a system that is integrated with the Native Android Development.

Firebase supports Firestore - a realtime synchronous cloud storage database. Firestore datastructure is composed of a noSQL Collection-Document data model similar to that of MongoDB. This Collection-Document model makes Firestore ideal for real-time messaging applications[3].

Twilio is a service that provides APIs for programmatically sending and receiving SMS, MMS and voice messaging services - including WhatsApp.

A specific Twilio API is provided for WhatsApp through REST requests. At time of writing the API is still only available in early access, which means that developers must request access to but can prototype on WhatsApp immediately with test users who can join the sandbox environment.

[2]Praekelt.com. "Feersum Engine - User Documentation - Feersum Engine - User 0.13.18 documentation". In: (2017)

[3]Ryan Ackermann. "Firebase Tutorial: Real-time Chat". In: *raywenderlich.com* (2018)

4 Development of Project

4.1 System Overview

The Activity Phase Diagram in Figure 1 demonstrates the desired phases of functionality and component designations. The **Training Phase** is shown to operate separately to the other phases, as it does not need to be conducted as often. The Training Phase will involve the NLP Algorithm component receiving, interpreting and training on input phrases.

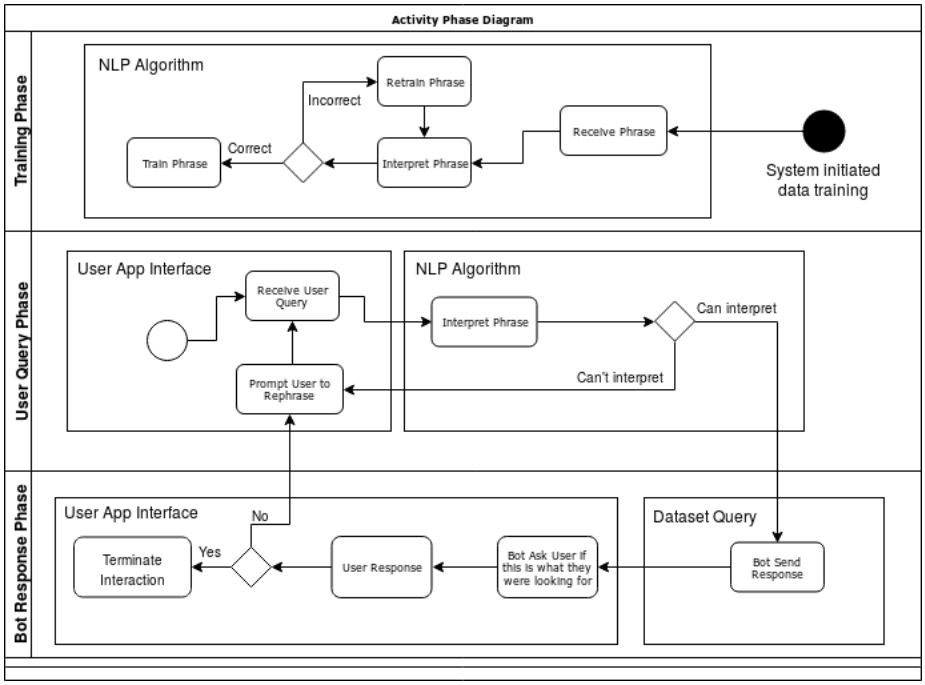

Figure 1: Activity Phase Diagram of High Level System

The **User Query Phase** contains the functionality that is initiated on the submission of a query from a user. The query is submitted through the User App Interface Component. The Phrase is sent to the NLP Algorithm Component - this component uses the existing NLP Model to attempt to interpret the query. If the message cannot be interpreted, the prompt to rephrase is returned to the user interface.

If the query is interpreted, the corresponding response is found in the dataset and returned by the ChatBot in the **Bot Response Phase**. The Bot prompts the user for further input before terminating the interaction.

4.2 High Level System Architecture

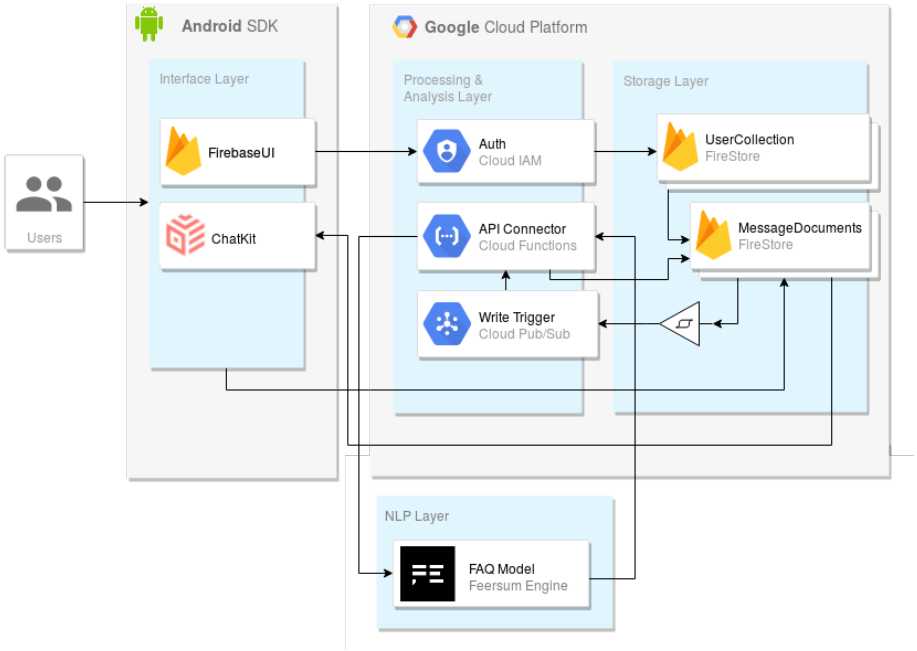

Figure 2: Mobile App Frontend System Architecture

4.3 Implementation 1: NLP Integration

Flow of operation: Initiating NLP API Model Creation with provided training data on command. Receiving the User's input from another component. Submitting the user's input to the NLP API in the required Query format. Listening for response from the API. Returning the response to the receiver component.

In order to train the NLP Model proposed in Feature 3 a dataset of query and response phrases must be provided. These phrases will involve a particular topic. This component will be responsible for creating and training the NLP Model using the NLP API. This component may be included in the NLP API Interaction component at a later stage as the functionality could easily be hosted on the same system. The input and output functionality of this component is primarily:

- Receive training dataset from the dataset text file
- Submit the Training Data to the NLP API Integration Component
- Receive model successfully/unsuccessfully created response

4.3.1 Implementation of TF-IDF Similarily Measure

In order to quantify the measurements, a metric of phrase variance is required. This is where the TF-IDF method is implemented. The Cosine Similarity of Vector Space Models is a useful implementation for providing a metric of text document similarity.

In order to create a responsive ChatBot, the Feersum NLP API needs to be called in order to create an NLP Model. This model is trained using the maternal health dataset. In order to assess the accuracy of the NLP Model, a mechanism must be developed to test phrase variance and response accuracy. This tool has been called **XhosaBot-PhraseVarianceGenerator** and a link to the source code repository can be found in Appendix A.

The "FAQ Matching" model must be created and trained using the provided dataset.
The **XhosaBot-PhraseVarianceGenerator** Testing Component provides a set of functions used to analyse the morphological differences between phrases, as well as query and compare the label match probability for these phrases attained by submitting test requests to the `feersum_nlu` model.

Phrase variance is used as a rudimentary method of morphological analysis, as a semantic - or meaning based - understanding of phrases for variation analysis is not yet viable. The function in code Listing 1 shows the process of applying a TF-IDF vectorizer to the phrases in a set, before finding the similarity between the original phrase and the other phrases in the sequence. This measure is then used to determine the *variance* of a phrase from the original phrase.

```
def find_variations(phrase_set):
    tfidf_vectorizer = TfidfVectorizer()
    tfidf_matrix = tfidf_vectorizer.fit_transform(phrase_set)
    # Variation comparison to the first element in the matrix - the
        original phrase
    sim = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix)
    for i in range(0, len(sim[0])):
        # Normalize cosine measure and round
        norm_cosine = round(100*(1 - sim[0][i]), 2)
        print("Phrase " + str(i) + " variation: " + str(norm_cosine) + "
            \%")
```

Listing 1: TF-IDF Vectorizer for Text Set

4.3.2 XhosaBot-APIConnect Functionality Overview

was developed to train and query an NLP model. This allows the system to firstly build a model, before receive and request queries to the model through the Feersum API.
The implementation of the Feersum API interfacing application component has been called **XhosaBot-APIConnect** and is built in Python. The system is

built to interface with the Feersum API using the `feersum_nlu` library. The **Training Phase** block contains the commands used to execute this component, and the calls initiated in this component. This component is executed separately from the other components. The **Operational Phase** block contains the command used to execute this component. As illustrated this command is initiated externally by a Cloud PubSub - this allows the component to initiate on upload of a new Message object in the Firestore database. This block also shows the representation of the database authorisation calls and query submission.

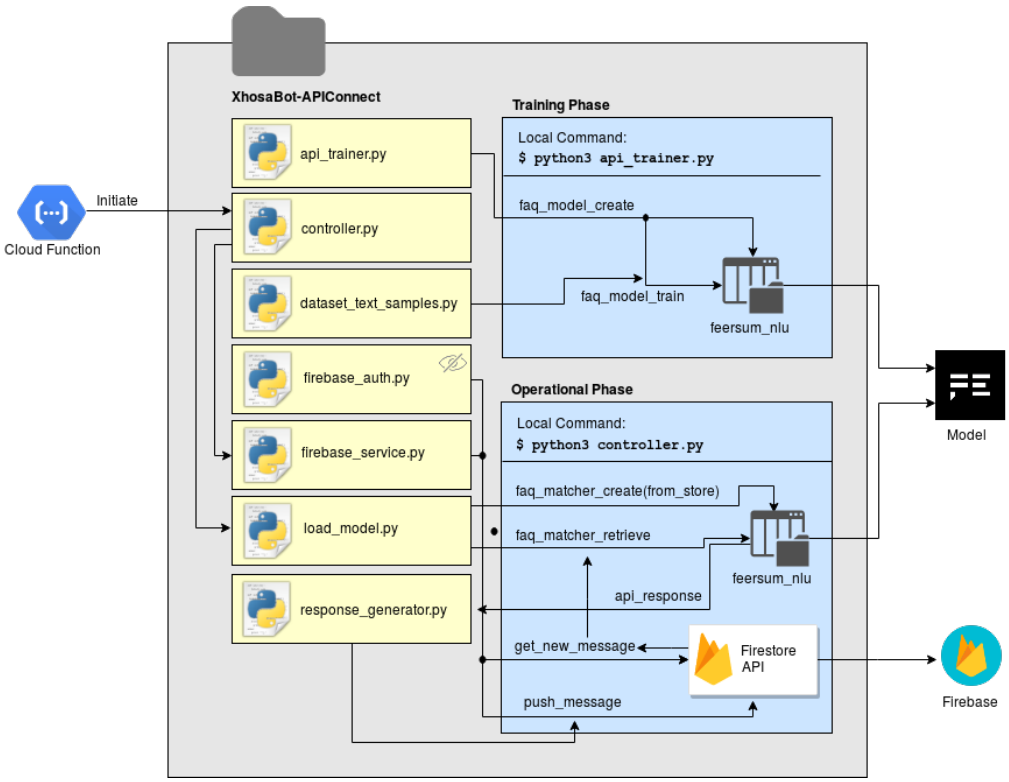

Figure 3: APIConnect Internal Component Interactions

5 Implementation 2: The WhatsApp Integration

The full code for these listings along with installation instructions can be found in the **XhosaBot-WhatsAppAPI** Github repository link found in the Appendix A. To check out this Implementation for yourself please follow these instructions:

"Invite your friends to your Sandbox. Ask them to send a WhatsApp message to `+14155238886` with code `join corn-oryx`."

5.1 High Level System Architecture

The Implementation of this System is in many ways less complex than the XhosaBot Mobile Application, but far less configurable.
This Implementation builds on the Components built in the Mobile version, and as such these Components will only be discussed should their implementation be different. The High Level System Architecture of the software components and layers involved in this implementation can be seen in Figure 4.

This is done using the early access WhatsApp Business API, available through the Twilio framework. This Component is written in Node.js.

5.1.1 XhosaBot-WhatsAppAPI Functionality Overview

Whatsapp API creates a user initiated *Session* - a 24 hour window after a user sends a request in which the API can communicate with the user. As such the Authentication and Datastorage is dealt with through WhatsApp.
The primary functionality of the **XhosaBot-WhatsAppAPI** Application is described as follows:

Configure and listen to an inbound message Webhook, create a Session on reception of an inbound message.Retrieve query from user REST formatted message Enable the Feersum NLU APIClient and create an asynchronous Promise request to `feersum_nlu_api` library to load the instance model. Upon Promise return, then create an asynchronous Promise request using the API instance to match the text input. Once the asynchronous instance receives a response, the most closely predicted label is found in the API response object. The label is matched against the labelled dataset. Return of the output text initiates the response callback.The message response is submitted in a REST formatted message to the user.

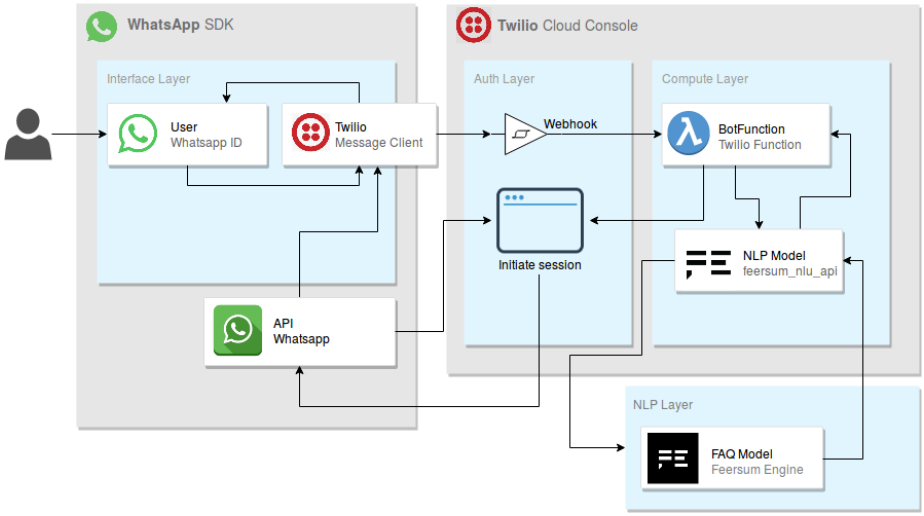

Figure 4: Whatsapp Implementation System Architecture

6 Results and Discussion

The following chapters outline the procedures used for analysing the implemented systems against the predetermined Acceptance Test Criteria. This will include testing NLP algorithm label accuracy and user interface accessibility and implementation.

6.1 FAQ Model Dataset Analysis

The Phrase Generator can be seen to produce very poor results for the first iteration of Variance and Match Probability testing. These poor results are sensible, as the Phrase Generator does not have a corpus and thus cannot compare synonyms, rather just providing a measure of semantic difference, as discussed prior.
The poor Label Match Probability is due to the small training set provided to the FAQ model. With only one phrase provided per label, it is reasonable that the model has been shown to struggle to differentiate sentences that have more than a 50% phrase variance.

The model was then retrained using these additional variations, as well as variations for the other 20 phrases. This amounted to an additional 77 phrases. The `feersum_nlu_api` provides a method for adding additional training samples to an existing model.

Figure 5: Small Screen App Flow

6.2 Acceptance Criteria AT3: ChatBot Understanding

Conduct NLP testing procedure to determine whether 90% of queries relating to the preset topic are successfully understood.

A query not in the dataset will be presented.

The `feersum_nlu` algorithm contains a test samples set. This means that the results from these tests will not effect the algorithm. The testing set is conducted by creating a test set of queries, the testing sets are described below:

Here the testing sets will be developed using Cortical.io to develop input phrases within measures of variance for testing.

The initial results show that the results performed poorly on phrases with a variance above 60%. These poor results are sensible, as the Phrase Generator

does not have a corpus and thus cannot compare synonyms, rather just providing a measure of semantic difference, as discussed prior.
The poor Label Match Probability is due to the small training set provided to the FAQ model. With only one phrase provided per label, it is reasonable that the model has been shown to struggle to differentiate sentences that have more than a 50% phrase variance.

In order to investigate the effect of a larger training set on the performance of the model, the model was retrained with 77 more phrases. New phrases were then generated in order to perform the variation vs match probability metrics that are shown in Figures **??** and 6.

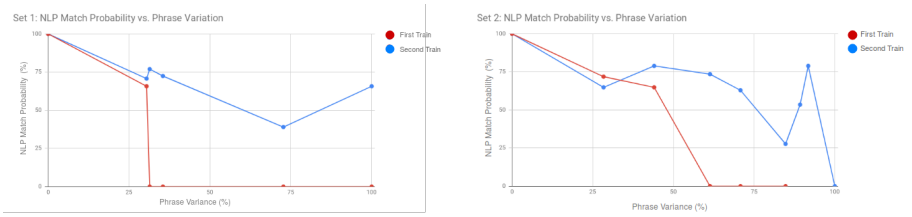

Figure 6: Match Probability vs. Variation For Phrase Set 1 2

6.3 Acceptance Criteria AT4: Realistic Conversation Flow

Timing tests conducted to determine the average time taken for the system to respond to a user's query. In order to determine the system response time, the metric is taken from as soon as the user enters a message on the Mobile platform, to the time at which the response appears on the device. This was done in order to encompass all the system functionality in between, from submitting and storing to Firestore, to triggering the Cloud Function to creating and querying the FAQ Model API, to the dataset lookup to the Firestore write, to the snapshotListener in the application.
The average time taken to complete the request and response is therefore **3.86s**. This is well below the Functional Requirement FR4 limit of 10 seconds. This has therefore met the **User Requirement UR4:** User must experience a realistic flow of conversation that responds in realtime. This testing cannot be implemented for WhatsApp as there is no way of conducting end to end testing.

7 Conclusions

This project has conducted the research, design and development of a NLP enabled ChatBot for conversational isiXhosa. User and Functional Requirements of the conceptualized system were analysed and relevant Acceptance Criteria determined.
This project has determined that implementing an isiXhosa ChatBot is very difficult on a conceptual NLP level; but available technologies prove that it is

possible to develop a purpose-driven mobile application ChatBot.
Our Natural Language Understanding models are developed locally, and can be modified to provide solutions for specialised industries like finance or health. Multiple implementations of User Interfaces where considered, with the aim of ease of user experience and accessibility in mind. A Zulu or Xhosa ChatBot has the potential to lower the effort of a user who may not be a first language English speaker who may not be wholly technology literate [4];. Implications of natural language ChatBots for access to information - both in the social impact and consumer space vary from medical and legal assistance, to personal banking. Due to limitations in access to information for South Africans, modern technology should be used to find novel solutions to bridging the *digital divide* and providing access to essential information to those who need it most, in their home language.

[4]Ocular Technologies. "The multilingual chatbot: Sawubona, molweni, goeie more, good morning, thobela, dumela, opportunities for South Africans". In: *iWeb* (2017)