AAMMANGO: A HINDI PRONUNCIATION TRAINING WEB APPLICATION

Sumedha R. Pramod
aammango.pramod.ninja

Department of Computer Science, University of Michigan, Ann Arbor
EECS 499 Fall 2014 – Intelligent Interactive Systems Independent Study

1. INTRODUCTION

There is surprisingly a very limited range of options in language learning tools, which are available to iOS users in the present day. Of the tools available in the market, few provide pronunciation feedback when dealing with complex languages, such as Hindi. Hindi speech contains many small nuances which non-native speakers find extremely difficult to properly reproduce. Many times, a small pronunciation variance can result in a completely different meaning. Few tools exist in the market today, such as DuoLingo [4] and Rosetta Stone [13].

AamMango is aimed at providing users with a simple and intuitive tool to learn basic Hindi vocabulary. The application will serve as a vocabulary and pronunciation-learning guide for people trying to learn Hindi. The application will aim to teach users how to pronounce a small set of vocabulary, focused on Indian foods.

One of the most difficult challenges in learning is the ability to correctly pronounce certain words and be able to reproduce the unique sounds that make up the language. In addition, there are many different words, which differ solely, based on aspirating certain syllables. AamMango is aimed at helping people learn those subtleties while learning to speak some basic Hindi vocabulary.

The complete Hindi consonant set with their phonetic property is given in table I [1]. Many of the nasal tones, such as in the word language in Hindi, **भाषा** (Bhāṣā), which has a cerebral nasal tone, are extremely difficult to reproduce

This application is not aimed at teaching the user sentences and a breadth of vocabulary. Rather it focuses on learning to pronounce the breadth of sounds in the Hindi vocabulary. Therefore, contrary to having an extremely intensive dictionary and testing the user against them, the application will have a robust, but compact set of vocabulary flashcard decks which will allow them to become familiar with the different types of pronunciation that are available in the Hindi language.

TABLE I. HINDI CONSONANT SET [1]

Phonetic Property	Primary Consonants (Unvoiced		Secondary Consonants (voiced)		Nasal
Category	Un-aspirated	aspirated	Un-aspirated	aspirated	
Gutturals (कवर्ग)	क	ख	ग	घ	ङ
Patatals (चवर्ग)	च	छ	ज	झ	ञ
Cerebrals (टवर्ग)	ट	ठ	ड	ढ	ण
Dental (तवर्ग)	त	थ	द	ध	न
Labials (पवर्ग)	प	फ	ब	भ	म
Sewivowels	य, र, ल, व				
Sibilants	श, ष, स				
Aspirate	ह				

2. DATA

2.1. Initial Data Collection

Since this application is being tested against live user data, the testing data will be recorded after the user presses the speak button on each flashcard. This testing data will then be run in the goodness of pronunciation algorithm to gauge the accuracy of the user's speech.

A phonetic dictionary from an open source Hindi Acoustic Speech Recognition (ASR) developed under Sarai FLOSS Fellowship in 2007 was used to create a custom Hindi acoustic model [5]. A language model was built using the phonetic dictionary provided with the Hindi ASR model in combination with CMU Sphinx tools for the words in each of the flashcards as a model for expected input. The language model provided in the Hindi ASR is trained using audio samples that were recorded by native Hindi speakers 16KHz, 16 bit, Microphone Based Acoustic Models and created from Clean Read Speech which results in a highly accurate Hindi language model. The training data is most accurate for native Hindi speakers from Delhi, Madhya Pradesh, Uttar Pradesh, and some parts of Rajasthan and Punjab, This enables users to learn to speak like people in

Northern India, which primarily consists of prominent Hindi speaking regions. The data was collected from forty speakers saying fifty different sentences each.

In order to store users and flashcard data, the Parse framework was integrated. Parse is a simple and light cloud database, which allows for easy login management and social media integration. In addition, to translate each English word into Hindi script, the freely provided Bing Translator API was used. Each of these APIs was called using asynchronous HTTP calls followed by parsing the returned JSON data. For speech synthesis, Apple's AVSpeechSynthesizer was used in conjunction with the "hi-in" voice, which takes in Devanagiri script text (Hindi) as input and returns a pronunciation of the inputted text.

2.2. Acoustic Model and Language Model

A custom phonetic dictionary was created and based an open source Hindi ASR developed under Sarai FLOSS Fellowship in 2007. This was used to create a custom Hindi acoustic model [5]. The dictionary is attached in Section 7.1.

A language model was built using the newly created phonetic dictionary in combination with CMU Sphinx tools for the words in each of the flashcards as a model for expected input. The language model was trained using audio samples that were recorded by five native Hindi speaking people. The data was collected from five speakers saying each word five times, totaling to 420 samples per speaker.

For the purposes of this paper, speakers are defined as follows. native speakers are defined as speakers whose primary language is Hindi. Non-native speakers are defined as speakers who speak another language as their mother tongue and have learned Hindi as a second language. Non-speakers are defined as speakers who do not speak Hindi in any capacity.

3. METHODS

When implementing a speech recognition system, there are a few available methods, but the most common tools used are CMU Sphinx and HTK. OpenEars is a recently developed free iOS framework, which ports a lightweight version of the CMU Sphinx controllers, or pocketSphinx, to the iOS operating system for simple integration into Apple devices.

3.1. Initial Implementation (OpenEars + CMU Sphinx)

In order to integrate the provided acoustic model, the OpenEars framework was integrated into the application to port the CMU pocketSphinx controllers to iOS. CMU Sphinx was also essential to create a basic language model to describe the grammar needed by the application.

However, OpenEars requires extra customizations to be used with languages other than English and Spanish.

In order to integrate a custom acoustic model into the OpenEars framework, a compatible bundle file must be created for the acoustic model. The bundle must contain the following files: *feat.params, mdef, means, mixture_weights, noisedict, transition_matrices, variances*, in addition to a language model file and dictionary stored directly within the application itself. All of these files were available in the provided Hindi acoustic model. However, the compiled bundle file, when integrated into the application as a framework, was not able to recognize the correct words. Majority of the hypotheses received from pocketSphinx were null. On occasion, the application received a non-null response, however it rarely corresponded to the correct word spoken. After testing various combinations of the files and feature parameters from the Hindi acoustic model, a new approach was necessary. This approach is described in section 3.2.

3.2. Implementation with English Acoustic Model

Due to lack of compatibility between the custom acoustic models and OpenEars, an attempt was made to add the Hindi words to the English phonetic dictionary while replacing the unusual phonemes with their appropriate English phoneme. The phonemes were mapped as follows:

TABLE II. HINDI TO ENGLISH PHONEME MAPPING [3]

Hindi	English	Hindi	English	Hindi	English
kh	k	b	B	n	n
ai	ey	e	Ih	p	p
ch	t	d	V	s	s
au	ow ey	g	L	r	d
gh	l	i	Iy	u	ah
th	l	h	L	t	t
ph	f	k	K	v	l
bh	lr	j	D	y	hh
jh	d	m	M		
a	ah	l	L		
c	t	o	Ah		

This mapping allowed the application to run by only adding the needed words to the English phonetic dictionary. The application was then able to recognize most of the Hindi words in the numbers flashcard deck, shown in screenshot 4 of section 3.8 below. However, due to the acoustic model remaining an English acoustic model, the confidence score of the words, which were correctly identified, was extremely poor. As AamMango is intended on being an application, which helps users fine-tune their pronunciation in Hindi using basic vocabulary sets, it wasn't appropriate to use the English acoustic model to recognize Hindi words.

3.3. HTK Implementation

At this point, a different approach was taken. A custom acoustic model was created using the data collected from four native Hindi speakers and one non-native speaker. All speakers were asked to speak the numbers one through ten in Hindi at random variation. In addition, they were also asked to recite randomly generated combinations of words compiled from the dictionary provided in the previous Hindi acoustic model. Each of these audio samples was transcribed in a prompts file. Additional test samples were recorded from each of the native speakers to test the accuracy of the generated acoustic model.

The following methodology was described in the "HTK Book" and was used to generate a simple Hindi acoustic model from the collected training data [14]. First, the prompts of the training data were used to create the associated word net and word list. This mapped each of the words used in the transcript for use in the model later. Then, a dictionary was created using the words from the word list created earlier. In addition, two files were created containing all the monophones from the words in the dictionary, one with silence specified as a monophone, and one without. This information is then used to create the Hidden Markov Models (HMM) files, while loading the generated sets of HMMs as they are generated. After creating a set of label files for each of the audio segments being processed in HTK, a first pass of Viterbi decoding is run in order to align the audio segments with the labels. Then, the remaining HMM sets are loaded which is then followed by another pass of Viterbi decoding. This generated a 94.29% correctness in the following when using only native speakers for test data.

---------- Overall Results ----------
SENT: %Correct=0.00 [H=0, S=70, N=70]
WORD: %Corr=94.29, Acc=-105.71 [H=66, D=0, S=4, I=140, N=70]

In running the experiment, data was collected from two native Hindi speakers and one non-Hindi speaker. The test prompts only included variations of Hindi numbers. The test data was the processed using the HVite command to forcefully align using the Viterbi Decoding algorithm. This command created a file, which provided the confidence score relating to the accuracy of the test samples in comparison to the training data for each of the test files that were run in the command. Using this score, it was possible to implement the algorithm for goodness of pronunciation scoring and assessment for interactive language learning.

3.4. Generating the Acoustic Model & Language Model

In order to generate an acoustic, audio samples were taken from five native speakers each saying each word five times, totaling to 420 samples per speaker. This methodology was also previously mentioned in section 2.2. A transcription of each of the audio samples (aammango.transcription) and a list of each audio file (aammango.fileids) were then compiled using python scripts.

Once the data was prepared, the sphinxtrain setup script was run on the audio samples and transcriptions, which created the default configuration file. After modifying the configuration file with the custom paths, audio formats, and model parameters, sphintrain was run on the audio database. This resulted in the appropriate acoustic model files needed for recognition: mdef, feat.params, mixture_weights, means, noisedict, transition_matrices, variances [1].

Using a transcript of the audio samples, a vocabulary file was created, which would then be fed into Sphinx to generate a language model. The vocab file was given as a parameter to generate an idngram file using the following command: " text2idngram -vocab aammango.vocab -idngram aammango.idngram < aammango.closed.txt". This idngram file was then converted into an ARPA language model using the following command: "idngram2lm -vocab_type 0 -idngram aammango.idngram -vocab. aammango.vocab -arpa aammango.arpa". This file was then converted into the CMU binary language models filetype (DMP) using the following command: "sphinx_lm_convert –i aammango.arpa –o aammango.lm.DMP" [16].

3.5. OpenEars Second Implementation

As I was finally able to create a sphinx acoustic model and language model, I used the files as input into an iOS framework bundle to be used with OpenEars in the iOS application. The application was then able to recognize which word was being spoken; however, OpenEars does not allow the developer to access the probabilities involved in calculating the Goodness of Pronunciation. As this platform did not allow for the key aim of the application, another method was needed, described below in section 3.6.

3.6. Pocketsphinx.js Implementation using CMU Sphinx

Pocketsphinx is a lightweight version of CMU Sphinx which is intended for mobile and application use over Sphinx which generally runs from terminal commands. However, as Pocketsphinx is also written in C, it is unusuable as a web application. Sylvain Chevalier has created a javascript library on Github, which ports Pocketsphinx for use in the web browser [2]. This makes use of an LLVM-to-Javascript compiler, Emscripten which converts C/C++ files to Javascript to be run in websites.

Before compiling the pocketSphinx.js file, which included the custom created acoustic and language models, the configuration file was modified such that it included the models. Then the following command was run "cmake -DEMSCRIPTEN=1 -

DCMAKE_TOOLCHAIN_FILE=path_to_emscripten/cmake/Modules/Platform/Emscripten.cmake," which created a Cmake.txt file. This file contained commands to merge and compile pocketSphinx, Sphinxbase, and the acoustic and language models using "make". The resulting javascript file was then usable in the web application.

In order to properly make use the javascript version of pocketSphinx, a custom recognizer needed to be written. Using the recognizer.js file and the scripts in live.html in the pocketpshinx.js Github repository as a base, the web application was set up [2]. The audio recorder provided in the example application was also used to capture user audio input in audioRecorder.js. This recorder, however, only supports Chrome and Firefox browsers and does not support mobile devices for the time being. Future work to integrate a different, more universal audio recorder is necessary.

On load, the site adds all words and grammar to the application and prompts the user for permission to capture audio input from their browser. Upon acceptance, a script completes setup of pocketSphinx in the backend and enables the front end for interaction.

The source code for this project is available on Github http://github.com/pramodsum/AamMango.

3.7. Goodness of Pronunciation

The basic goodness of pronunciation (GOP) algorithm is shown below, where the numerator in the log is the probability that the speaker uttered phone p, Q is the set of all phone models, and NF is the number of frames in the observation [13].

$$\mathrm{GOP}_1(p) = \left| \log \left(\frac{p(O^{(p)}|p)}{\max_{q \in Q} p(O^{(p)}|q)} \right) \right| \Big/ \mathrm{NF}(p).$$

Since the application has a restricted vocabulary, it allows the speech recognition portion of the application more focused on accuracy and comparing those values rather than trying to predict the most probable sentences. In addition, since the user only selects one word from their specified category, there is no need to be concerned with grammar and the progression of words in the recorded data.

Using the pocketSphinx.js library, I was able to pull the required probabilities and phone loop probabilities needed to perform the goodness of pronunciation algorithm. The yielded result allowed for a standard by which to grade the user's pronunciation. Different messages are then displayed to the user depending on their pronunciation. The application also maintains a count of the number of times a user correctly pronounces and incorrectly pronounces words, so that the users can have a sense of progress.

3.8. Screenshots

3.8.1 Initial iOS Application Screenshots

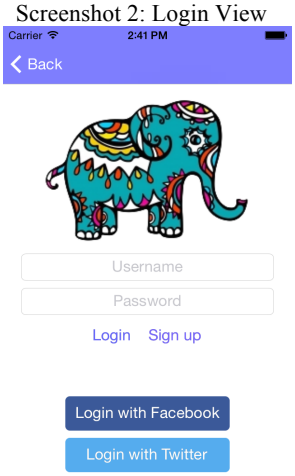
The iOS application will look as shown in the following screenshots. In addition, the flow of the application will be as described in the following sections.

3.8.1.1. Login

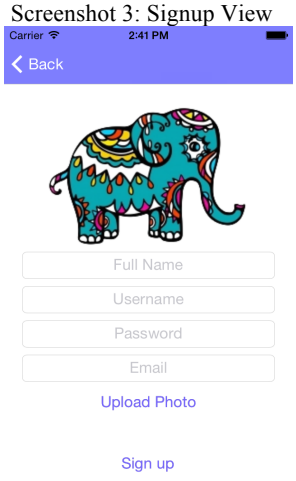
The user is asked to register using their email or sign in to the application via an existing account, Facebook, or Twitter as shown in screenshots 1, 2, and 3. Screenshot 1 depicts the root view, which is the initial view when the user opens the application. Screenshot 2 allows the user to login to the application using an existing account, a Facebook account, or a Twitter account. Screenshot 3 allows the user to create an account manually, which is then saved to the Parse User database.

Screenshot 1: Root View

Screenshot 2: Login View



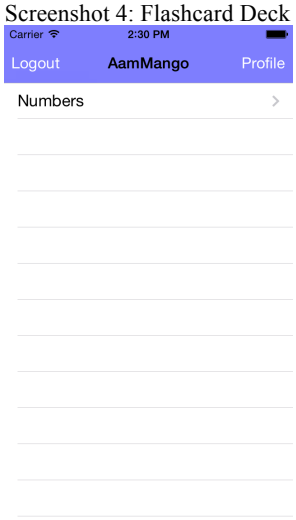
Screenshot 3: Signup View



3.8.1.2. Flashcard Set Selection

Once the user is logged in, they are allowed to select a set of flashcards as shown in screenshot 4. Initially, the application will only contain one set of flashcards, which can be expanded upon once a stable acoustic model can be created to use in conjunction with the speech recognition for the application.

Screenshot 4: Flashcard Deck Selection View



3.8.1.3. Flashcard View

When the user selects a deck of flashcards, they are brought to the first of the flashcards in the deck. Each card is displayed as shown below in screenshot 5. Each card has an image associated with it, the English pronunciation, the word in Hindi (written in Devanagari script), and two buttons.

The left button labeled “Listen” plays a pronunciation of the word in Hindi, provided by Bing Translator and iOS AVSpeechSynthesizer.

The right button labeled “Speak” will wake up the pocketSphinx controller in the backend and listen to the user’s speech. Upon recognition, the application will process the audio segment and calculate the goodness of pronunciation of the audio. If the audio is shown to have a high enough goodness of pronunciation, the user is given points for successfully pronouncing the word.

If the user wishes to try a different flashcard, they may swipe left or right to load a different card. Each card follows a similar format.

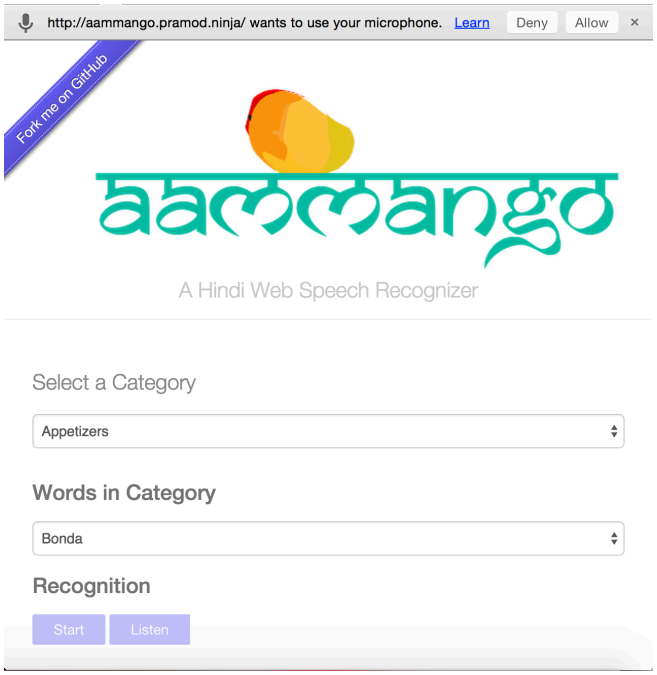
The user may also exit the flashcard deck and return to the flashcard set selection table view simply by pressing the “Home” button in the top left corner.

Screenshot 5: Flashcard View

3.8.2 Web Application Screenshots

The web application hosted at aammango.pramod.ninja looks as shown in the following screenshots. The application is a single page website as shown in screenshot 6. In addition, the flow of the web application is as described in the following sections.

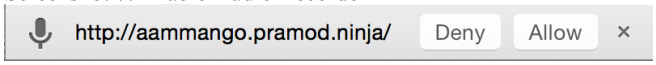
Screenshot 6: Main View



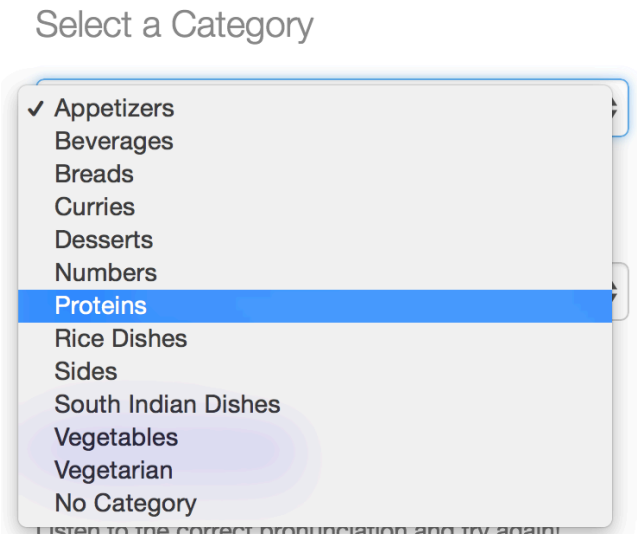
3.8.2.1. Recognition Setup

Upon loading the page, the user must grant the browser permission to record audio, as shown in screenshot 7. Once enabled, the user can select a category, as shown in screenshot 8. This narrows down the word choices available in that category, as shown in screenshot 9. Once a word is selected, the user can begin recording an audio sample for recognition.

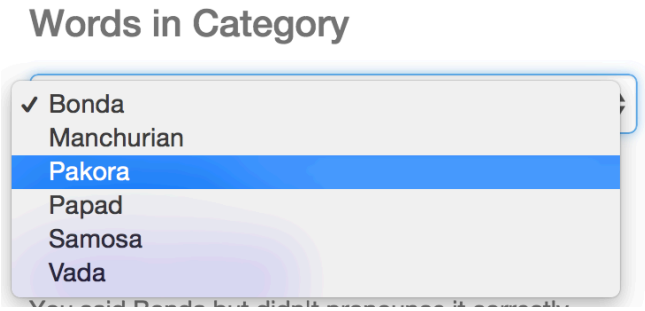
Screenshot 7: Enable Audio Recorder



Screenshot 8: Category Selection



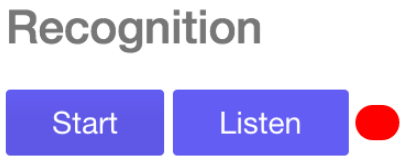
Screenshot 9: Word Selection



3.8.2.2. Recognition

There are two buttons available for the user to interact with: "Start" and "Listen," as shown in screenshot 10. If the user is unsure about the pronunciation of a particular word, the "Listen" button will provide them with an accurate pronunciation of the selected word. To begin recognition, the user may press "Start," which will then display a red circle to indicate that recording has begun.

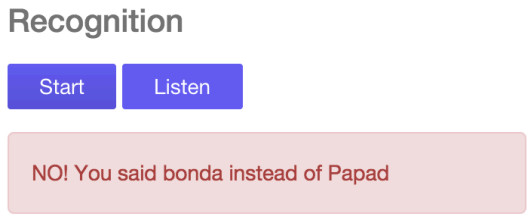
Screenshot 10: Recording Indicator



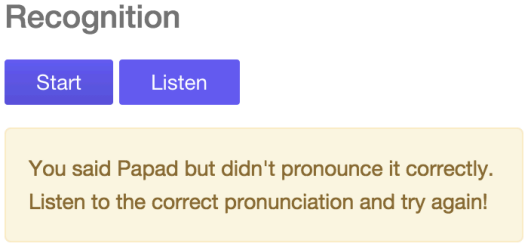
3.8.2.3. Recognition Example

If the user says the wrong word, the application will display a warning as is shown in screenshot 11. If the user says the correct word with an incorrect pronunciation, the application will inform the user accordingly, as shown in screenshot 12, and automatically plays a pronunciation of the word for comparison. If the user correctly pronounces the correct word, the application informs him or her of such as well, as is shown in screenshot 13. User statistics are also provided in order to give the user an idea of their current progress, as is shown in screenshot 14.

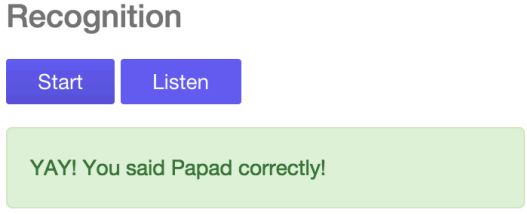
Screenshot 11: Incorrect Word Example



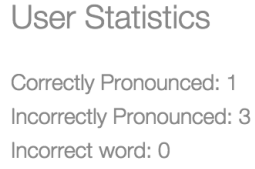
Screenshot 12: Incorrect Pronunciation Example



Screenshot 13: Correct Pronunciation Example



Screenshot 14: User Statistics

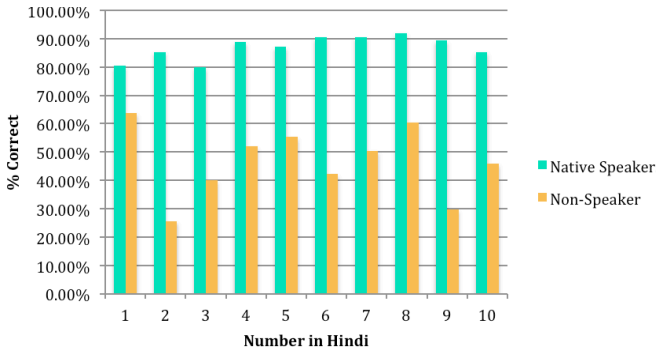


4. RESULTS

4.1 Initial HTK Results

The custom Hindi acoustic model and language model that was generated by using HTK and the recorded audio samples was able to generate an accurate model for Hindi. When the sample test audio was run using the model the data was able to display the differences in pronunciation in a visual manner. As shown in table III, the native speaker had much higher pronunciations of each of the words than the non-Hindi speaker. The difference is especially stark for words with unique pronunciations, such as 2 (दो), which in Hindi is pronounced "dhoo" Many non-Hindi speakers pronounce it as "doe" which is incorrect. Hindi contains a soft 'd' sound, which sounds similar to the 'th' in "the." Other words, such as 1, pronounced similar to the English word "ache" with a harder 'k' sound at the end, had very high GOP scores for both native and non-Hindi speakers due to the similarity of the sounds to the English language. The difference between speaker types was previously defined in section 2.2.

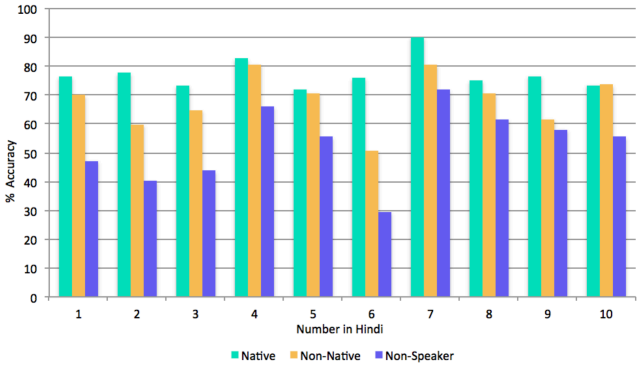
TABLE III. NATIVE SPEAKER VS. NON-NATIVE SPEAKER PRONUNCIATION ACCURACY



4.2. Final Sphinx Webapp Results

The custom Hindi acoustic model and language model that was generated by using HTK and the recorded audio samples was able to generate an accurate model for Hindi. In order to test the accuracy and results of the acoustic, language, and pronunciation modeling, I tested the application against native and non-native speakers. Each speaker was asked to use the application to pronounce the numbers one through ten in Hindi, whilst the application logged their scores for each word. As shown in table IV, the native speaker had much higher pronunciations of each of the words than the non-Hindi speaker. A larger table including data values is included in section 7.2. These results indicated similar conclusions as from the initial HTK implementation of this application. However, due to the trivial method of recording and filtering audio samples for recognition, the scores for both speaker types were much closer than expected. The difference between speaker types was previously defined in section 2.2.

TABLE IV. NATIVE SPEAKER VS. NON-NATIVE SPEAKER VS. NON-SPEAKER PRONUNCIATION ACCURACY



5. CONCLUSION

5.0.1. Initial Conclusions

Due to compatibility issues relating to porting the custom acoustic model with the speech recognition frameworks available with the iOS SDK, it was not possible to create a working prototype within the given time frame.

5.0.2. Final Conclusions

An initial prototype of the web application was posted at http://aammango.pramod.ninja. The application requires the use of a Chrome or Firefox browser. In order to minimize background noise, the use of a microphone and headphones is highly recommended. Using pocketSphinx.js, it was possible to implement a working prototype as the initial recognizer script and the pocketSphinx library is written in C.

5.1. Obstacles

5.1.1. Initial Obstacles

The provided Hindi ASR model is supposed to be a highly accurate acoustic model with accuracy relating to various North Indian native speakers. However, due to the lack of compatibility with the OpenEars, the model was not used in the initial application. In addition, there is very little documentation provided with the model and almost no support provided, therefore it became extremely difficult to debug and solve issues relating to the model. Future work looks to contact the authors of the model to potentially make

the model compatible with OpenEars or another similar iOS speech recognition platform.

5.1.2. Final Obstacles

As the number of speakers used to create the acoustic model was limited, the model is still in a rudimentary state. The collection of more data would allow for a more accurate model.

The OpenEars framework failed to provide the proper resources to conduct a goodness of pronunciation calculation. As the framework is not open source, it was not possible to modify the code to fit the use case for this project.

In addition, the current audio recorder being used for the web application picks up a lot of background noise. This caused many of the recognition results to be affected.

5.2. Future Work

In the future, a more robust acoustic model will be generated using a larger variety of data, including additional speakers, both native and non-native. Another web audio recording library, hark.js, is available which would allow for more accurate recordings and hopefully more accurate pronunciation scoring as well. The library also provides frame-by-frame data of the user's speech, which could be used to provide the user with a visualization of their speech in comparison to what is expected [10].

6. REFERENCES

[1] "Building Language Model." CMUSphinx. Carnegie Melon University, 22 Oct. 2014. Web. 18 Dec. 2014. <http://cmusphinx.sourceforge.net/wiki/tutoriallm#building_a_statistical_language_model_using_cmuclmtk>.

[2] Chevalier, Sylvain. "Syl22-00/pocketsphinx.js." Pocketsphinx.js. N.p., n.d. Web. 18 Dec. 2014. <https://github.com/syl22-00/pocketsphinx.js>.

[3] Choudhary, Annu, Mr RS Chauhan, and Mr Gautam Gupta. "Automatic Speech Recognition System For Isolated & Connected Words Of Hindi Language By Using Hidden Markov Model Toolkit (HTK)."

[2] "Duolingo." *Wikipedia*. Wikimedia Foundation, 22 Apr. 2014. 4eb. 22 Apr. 2014.

[5] FUNG, Anik DEY1 Ying Li1 Pascale. "Using English Acoustic Models for Hindi Automatic Speech Recognition." *24th International Conference on Computational Linguistics*. 2012.

[6] Irtza, Saad, and Sarmad Hussain. "Minimally balanced corpus for speech recognition." *Communications, Signal Processing, and their Applications (ICCSPA), 2013 1st International Conference on*. IEEE, 2013.

[7] Joshi, Sachin, and Venkatesh Keri. "Hindi ASR." *SourceForge.net*. N.p., n.d. Web. 22 Apr. 2014.

[8] Kumar, Kuldeep, and R. K. Aggarwal. "Hindi speech recognition system using HTK." *International Journal of Computing and Business Research* 2.2 (2011): 2229-6166.

[9] Mishra, A. N., et al. "Robust Features for Connected Hindi Digits Recognition."*International Journal of Signal Processing, Image Processing & Pattern Recognition* 4.2 (2011).

[10] "Otalk/hark." Hark. N.p., n.d. Web. 18 Dec. 2014. <https://github.com/otalk/hark>.

[11] Pammi, Satish C., and Venkatesh Keri. "HTKTrain: A Package for Automatic Segmentation." (n.d.): n. pag. Web. 22 Apr. 2014. <http://www.dfki.de/~chandra/marticles/pammi_HTKTrain.pdf>.

[12] Rabiner, Lawrence. "A tutorial on hidden Markov models and selected applications in speech recognition." *Proceedings of the IEEE* 77.2 (1989): 257-286.

[13] "Rosetta Stone (software)." *Wikipedia*. Wikimedia Foundation, 20 Apr. 2014. Web. 20 Apr. 2014.

[14] Sarfraz, Huda, et al. "Large vocabulary continuous speech recognition for Urdu." *Proceedings of the 8th International Conference on Frontiers of Information Technology*. ACM, 2010.

[15] Sharmila, Neeta Awasthy, and R. K. Singh. "Performance of Hindi Speech Isolated Digits In HTK Environment." *IOSR Journal of Engineering* 2.5 (2012): 1020-023. Web. 22 Apr. 2014. <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CCgQFjAA&url=http%3A%2F%2Fwww.iosrjen.org%2FPapers%2Fvol2_issue5%2FO02510201023.pdf&ei=jrNWU_bXIaLMygOK3YHYDw&usg=AFQjCNE9BU-r4j5Qna4ski-KX9qGpsM-oA&bvm=bv.65177938,d.bGQ>.

[16] "Training Acoustic Model For CMUSphinx." CMUSphinx Wiki. Carnegie Melon University, 10 Apr. 2014. Web. 18 Dec. 2014. <http://cmusphinx.sourceforge.net/wiki/tutorialam>.

[17] Witt, Silke M., and Steve J. Young. "Phone-level pronunciation scoring and assessment for interactive language learning." *Speech communication* 30.2 (2000): 95-108.

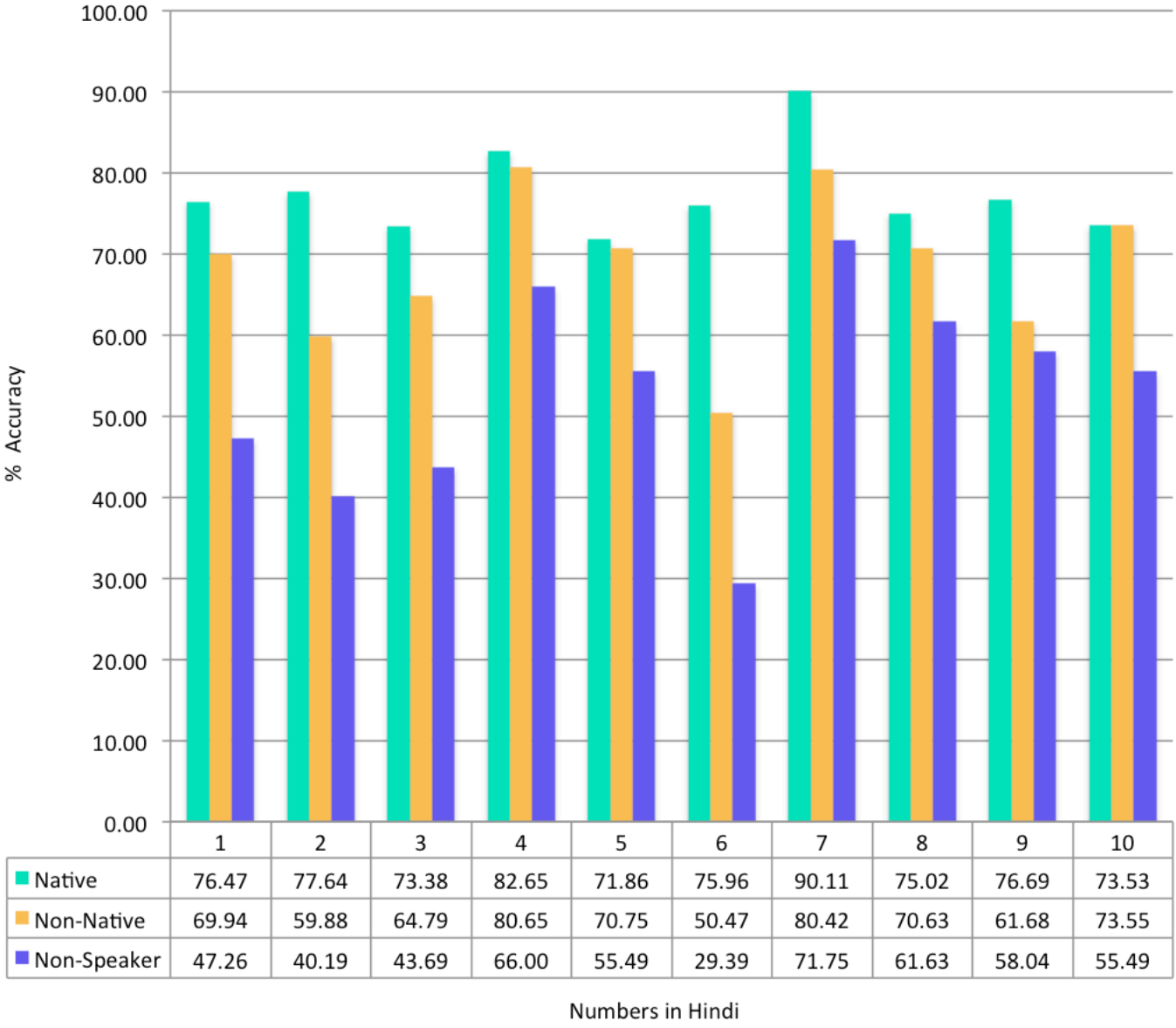
[18] Young, Steve. *The HTK Book*. Cambridge: Entropic Cambridge Research Laboratory, 1997. Print.

7. TABLES

7.1. AamMango Phoneme Dictionary

WORD	PHONEMES	WORD	PHONEMES	WORD	PHONEMES
bonda	b o nd. aa	vindaloo	w in: d aa l uu	baingan	b ae n:g a n
manchurian	m a nch uu r i y a n	ek	e k	bhindi	b:h i n:d. ii
pakora	p a k o r. aa	do	d o	channa	ch a n aa
papad	p aa p a r.	theen	th ii n	gajar	g aa j a r
samosa	s a m o s aa	chaar	ch aa r	gobi	g o b ii
vada	v a d. aa	paanch	p aa n:ch	matar	m a t a r
lassi	l as s ii	che	ch h	palak	p aa l a k
batura	b:h a t uu r aa	saath	s aa th	saag	s aa g
kulcha	k ul a ch aa	aat	aa t:h	chole	c:h o l e
naan	n aa n	nau	n au	dal	d aa l
paratha	p a r aa nt:h aa	dus	d a s	idli	I d. a l ii
poori	p uu r ii	badam	b aa d aam	jal frezi	j a l ph r:e z ii
roti	r o t ii	murg	m u r:g	rasam	r a s a m
bagara	b a g aa r aa	kaaju	k aa j uu	sambar	s a m:b aa r
bartha	b:h a r a th aa	biryani	b i r a y aa n ii	kadai	k a r. aa ii
chettinand	ch et ti n a n:d	bisibele baath	b i s i b e l e b aa th	keema	k ii m aa
gosht	g o shth	pulav	p u l aa w	lachcha	l ach ch: aa
makhani	m a k:h a n ii	upma	u p a m aa	shahi	sh aa h ii
navratan	n a v a r a th a n	vangibaath	v a n:g ii b aa th	kurma	k oo rm aa
pasanda	p a s a nd aa	venpongal	v e n:p o n:g a l	malai	m a l aa ii
tadka	th a r. a k aa	achaar	a ch aa r	kofta	k o phth aa
tikka	t ik k aa	dahi	d a h ii	dopyaz	d o py aa z
gulab jamun	g uu l aaa b j aa m uu n	raita	r aa y a th aa	pudina	p u d ii n aa
halwa	h a l a w aa	adai	a d. ae	rava	r a v aa
kheer	k:h ii r	avial	a v ii y a l	tandoor	t a n:d oo r
payasam	p aa y a s a m	dosa	d. o s aa	seekh	s ii k:h
kulfi	k ul ph ii	uthappam	uth th ap p a m	boti	b o t ii
rasmalai	r a s a m a l a ii	aloo	aa l uu	tangri	t a n:g r. ii

7.2. AamMango Web Application Pronunciation Comparisons



	1	2	3	4	5	6	7	8	9	10
Native	76.47	77.64	73.38	82.65	71.86	75.96	90.11	75.02	76.69	73.53
Non-Native	69.94	59.88	64.79	80.65	70.75	50.47	80.42	70.63	61.68	73.55
Non-Speaker	47.26	40.19	43.69	66.00	55.49	29.39	71.75	61.63	58.04	55.49