Task-specific Language Modeling for Oral Reading Assessment

Dual Degree Project Report

Submitted in partial fulfillment of the requirements for

Dual Degree (B.Tech + M.Tech)

by

Kanhaiya Kumar

(Roll No. 13D070046)

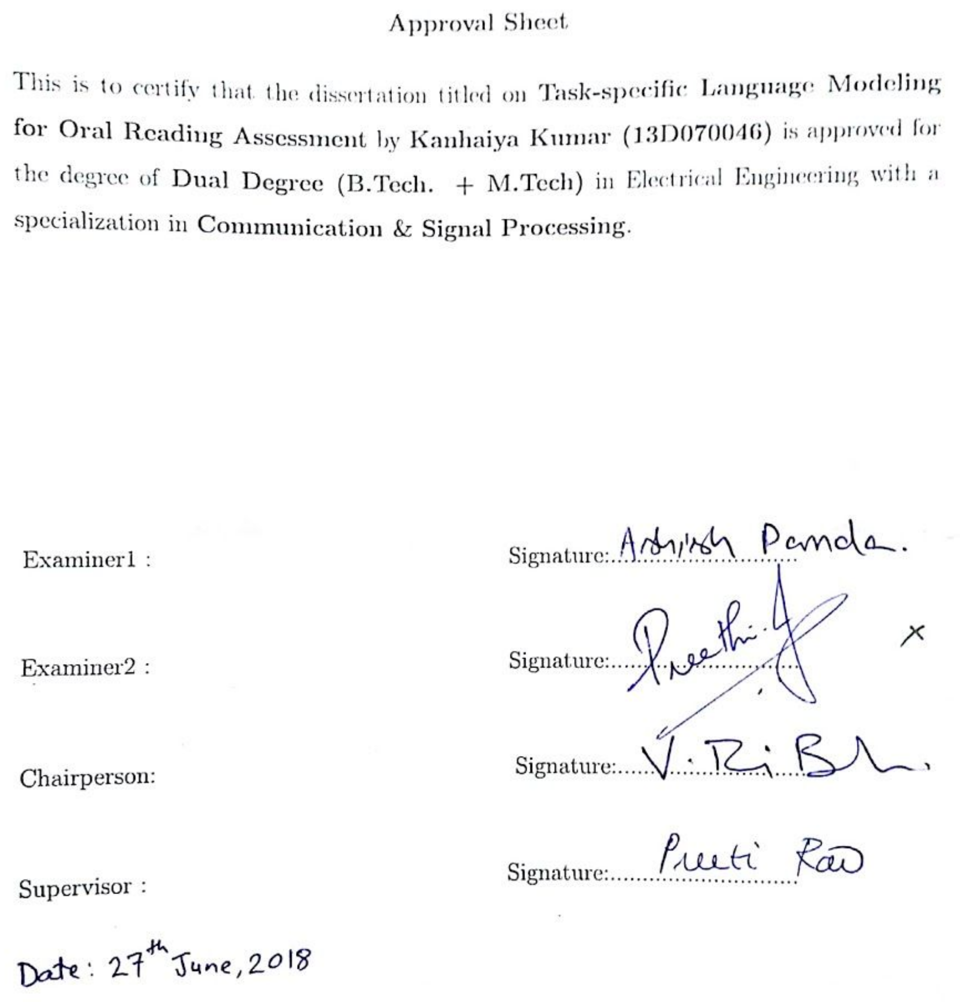
Under the guidance of

Prof. Preeti Rao

Department of Electrical Engineering

Indian Institute of Technology Bombay

May 2017 - June 2018

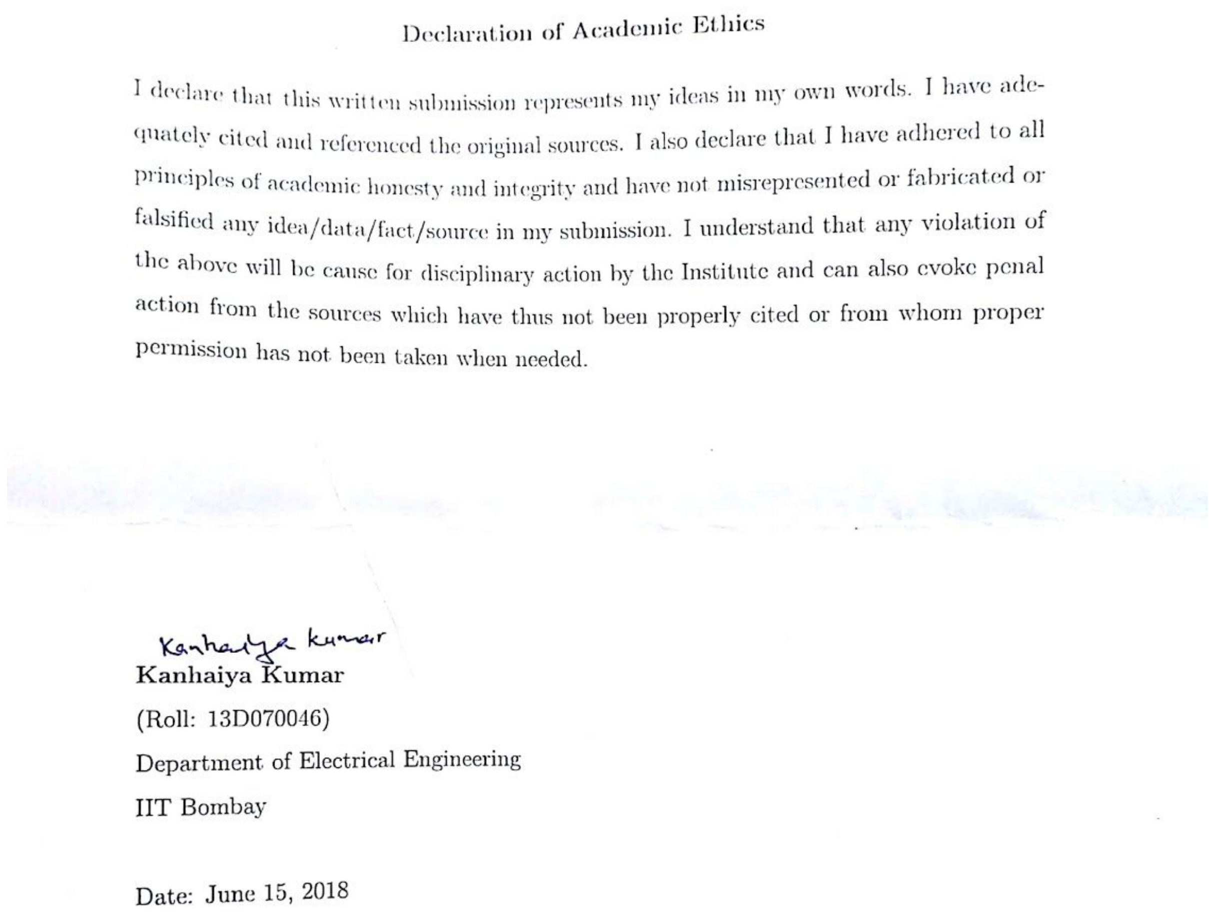


Approval Sheet

This is to certify that the dissertation titled on **Task-specific Language Modeling for Oral Reading Assessment** by **Kanhaiya Kumar (13D070046)** is approved for the degree of **Dual Degree (B.Tech. + M.Tech)** in Electrical Engineering with a specialization in **Communication & Signal Processing**.

Examiner1 :	Signature:......Ashish Panda.
Examiner2 :	Signature:......[illegible]
Chairperson:	Signature:......[illegible]
Supervisor :	Signature:......Preeti Rao

Date: 27th June, 2018



Declaration of Academic Ethics

I declare that this written submission represents my ideas in my own words. I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Kanhaiya kumar

Kanhaiya Kumar

(Roll: 13D070046)

Department of Electrical Engineering

IIT Bombay

Date: June 15, 2018

Acknowledgement

I express my deepest gratitude to my guide Prof. Preeti Rao for providing me the constant support and guidance throughout the project. I am thankful to Kamini Sabu for collaboration with the project, Hitesh Tulsiani and Prakhar Swarup for explaining the basic concepts of ASR systems.

Kanhaiya Kumar
Electrical Engineering
IIT Bombay

Abstract

The automatic assessment of oral reading skill can play an important role in enhancing the reading abilities of a child. The accuracy of the assessment is crucial in ensuring the constructive engagement of the child. One component in the assessment task is the word-level decoding of the utterance, typically achieved by an Automatic Speech Recognition (ASR) system. A task specific ASR system can utilize the known story text together with the understanding of expected reading mistakes or miscues such as missed words, incorrectly pronounced words and disfluencies. The performance of the system is measured by its accuracy in detecting the reading miscues rather than by ASR word error rate. Task-based adaptation methods can significantly improve both the Acoustic Model (AM) and the Language Model(LM) components of the ASR system. This thesis focuses on designing an efficient task specific language model for the children's oral reading task. The challenge is to determine an architecture that can combine domain-specific constraints from the story text with the flexibility needed to detect miscues including disfluencies. We obtain this using an N-gram model trained on the target story text combined in an FST framework with a garbage model containing parallel paths of a large number of general domain words with equal probability. The system gives 7.26% word error rate (WER) and 3.95% phone error rate (PER) on our field dataset of children's reading recordings. Moreover, at only 5% false alarm rate (FAR), we get around 74.03% miscue detection rate. As opposed to a unigram garbage model, where the words are constrained by the training data, here we can add any words(Hindi or English) of our choice in the garbage model. The garbage model weight can be learned on a small development set. Using the proposed LM model, we have built an assessment system which takes an audio with the corresponding canonical text as input and detects the miscues such as substitution, insertion and deletion and displays it via a graphical user interface (GUI).

Contents

List of Figures **III**

1 Introduction **1**

1.1 Focus of the present study . . . 2

1.2 Data description . . . 2

1.3 System block diagram . . . 3

1.4 Report outline . . . 4

2 Task Specific ASR System **5**

2.1 Data-set . . . 5

2.1.1 Training data-set for acoustic model . . . 5

2.1.2 Evaluation data-set . . . 5

2.1.3 Comparison of Training and Evaluation data-sets . . . 5

2.1.4 Training data-set for Language model . . . 6

2.2 Acoustic Model . . . 6

2.3 Task-Specific Language Modelling . . . 7

2.3.1 Our proposed LM architecture . . . 9

3 N-Gram Language Models **11**

3.1 Interpolation . . . 12

3.2 Back-off . . . 12

3.3 Improved Kneser-Ney Smoothing . . . 13

3.4 Witten-Bell Discount . . . 14

3.5 Evaluation of Language Models . . . 14

3.6 Implementation details . . . 15

4 Evaluation Metrics **17**

4.1 Edit distance . . . 17

4.2 Word Error Rate(WER) 18
4.3 Phone Error Rate(PER) 18
4.4 Miscue Detection Rate 19

5 Experiments and Observations **23**
5.1 Google Speech Engine 23
5.1.1 Challenges with Google's speech engine 24
5.2 Our LM architectures 26
5.3 Comparison based on different N-gram models 28
5.3.1 Why Witten-Bell performs better here 29
5.4 Comparison based on different LM architectures 30
5.5 Comparison based on different Acoustic Model weight 32
5.6 Showing Output on GUI 34
5.7 Performance of proposed LM on different categories of data 35
5.7.1 For substituted words 35
5.7.2 For mispronounced words 35
5.7.3 Effect of Noise 35
5.7.4 Murmurings or Sound-outs 35
5.7.5 Repetition and long pauses 35

6 Conclusion and Future Work **37**
6.1 Conclusion 37
6.2 Future Work 38

List of Figures

1.1 Overall System block diagram 3

2.1 LM for the sentence “The moon smiled” [1] 8
2.2 FST diagram of Trigram model along with garbage model for the sentence “Giants are huge” [2] 8
2.3 Weighted FST for each word [3] 9

4.1 Edit Graph [4] 21

5.1 Word Error Rate using Google Speech API 25
5.2 WER and PER curve with different weights of garbage model 26
5.3 ROC curve by changing weight of garbage model 27
5.4 WER by changing weight of garbage model 28
5.5 PER by changing weight of garbage model 28
5.6 ROC curve by changing weight of garbage model 29
5.7 Probability left after discounting from N-gram counts 30
5.8 ROC Curves for different models 31
5.9 WER and PER with different Acoustic weight 33
5.10 ROC curve using different Acoustic weight 33
5.11 System audio-visual feedback interface 34

Chapter 1

Introduction

Millions of children from rural areas of India complete primary school without even achieving the basic reading skills [5]. Reading aloud is conventionally an important instructional method in many countries [6]. It is practically difficult for a teacher to pay attention to each and every child in class, so having an automatic personalized feedback on readings will help students improve their reading skills. So, automatic speech assessment is a very important task in making learning more efficient and attractive. If implemented properly, it will not only make learning more efficient and interactive for the students but also help lift the burden off the teachers' shoulders so that they can concentrate more on planning and provide individualized help using the reliable feedback from the assessment system. Making this assessment accurate is very important for ensuring the constructive engagement of child with the reading. It will enhance the overall learning by enabling the child to self-learn from vast available text resources. The importance of imparting reading skills in early school cannot be over-stated. The goal of our present work is to consider scalable technology solutions that facilitate oral reading practice and assessment in limited availability of language teachers. We focus on the specific context of second language i.e. English (L2), since the medium of instruction in schools of rural India is primarily regional language. To further motivate the need for such an application, it has been observed that 10 to 12 year old students, who are participating in the on-field deployment of this project, have highly under-developed reading skills as compared to their urban counterparts studying in the same grades. Automatic assessment system will be a very significant and useful contribution in such regions. The task of automatic assessment of reading ability can be broadly classified into two categories [1]:

1. Word-level assessment: In this category, assessment is mainly concerned with detect-

ing word-level reading miscues (substitution, omission, dis-fluency). It can highlight the specific errors made by a child with reference to the given text. Here, it requires an ASR model to decode the uttered sentences.

2. Prosody-level assessment: It involves assessing the children's reading ability based on fluency, speech rate and prosody. The prosody of the student's reading can help in predicting comprehension which is valuable.

1.1 Focus of the present study

In our present work, we focus our attention to the word-level assessment of reading ability for English language by rural Indian children in the age group of 11-15 years (grade 6-9). The targeted students learn English as a second language in school where the medium of instruction is primarily the regional language. We focus on the language modeling (LM) aspect of speech recognition to detect reading miscues efficiently. This ASR system will also give us the phone or word level alignment, which is very important for prosodic analysis (or feedback). Prosodic analysis along with the knowledge of mispronunciations can be used to assign an overall score suggestive of the child's reading ability. Hence the current work can be thought of as a stepping stone to high-level assessment. A major focus of this work is also to model Out-of-Vocabulary (OOV) words to account for large number of OOV substitutions observed while analyzing recordings of read aloud stories. A task-specific LM that should also be flexible enough to accommodate OOV words or any dis-fluency is required.

1.2 Data description

Depending on the reading skill level of children, we can broadly classify them in three categories:

- Category 1: Here the children are very fluent in reading, they are good at decoding the words, but sometimes they just skip a word or substitute it with other valid English words or reverse the order of two consecutive words
- Category 2: The children in this category are slightly dis-fluent i.e. they are having some problem with decoding the word, so they often mispronounce the difficult words. These substituted words can also be meaningless.

- Category 3: These children can't even decode the simple words like 'head'. There are many hesitations(repetition of words partially), long pauses, sound-outs in their utterances.

We have discussed the performance of our system in Section 5.7 on these different cases of children's speech data. We have also shown the performance in case of noise which is very common in the data considering the acquisition in the school environment.

1.3 System block diagram

The block diagram of overall system is shown in Figure 1.1. We first acquire the audio of stories read by students using an Android application. For the task of miscue detection using ASR, each story's utterances are fed to an ASR, where the utterances are first converted to acoustic feature vectors. These acoustic feature vectors along with the trained acoustic and task-specific language model act as input to the decoder. The decoder decodes the audio using both the models and returns a word sequence corresponding to the input audio. It also gives the word alignment and phone alignment which is very crucial for the prosody-level evaluation task. The word-level miscue detection task is evaluated using detection rate and false alarm rate of mispronunciations by comparing the decoded text with the canonical and ground-truth text. We also report results in terms of word error rate and phone error rate which are indicative of the ASR performance.

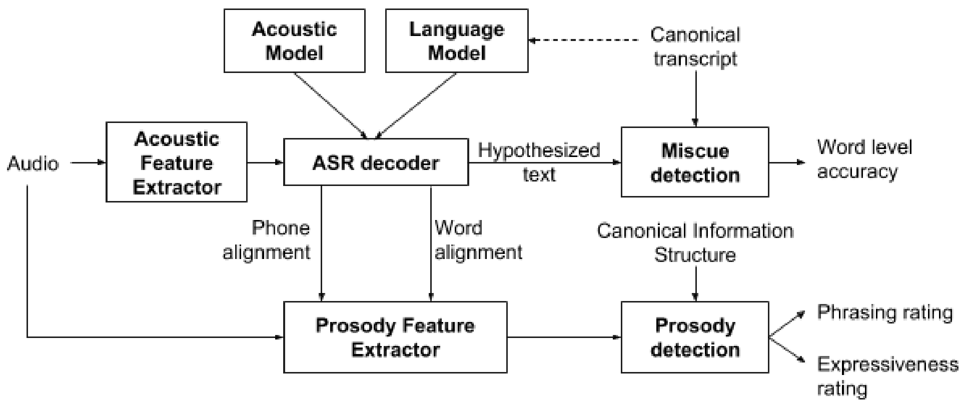


Figure 1.1: Overall System block diagram

1.4 Report outline

Chapter 2 shows different task specific LM architectures which have been used in the literature and also presents the proposed architecture, Chapter 3 covers the theoretical part of the N-gram Language model used in the generalized ASR system. Different task specific LM architectures have been designed and tested using these N-grams. Chapter 4 presents different evaluation metrics which have been used in our experiments for comparisons. We look at all the components to design and evaluate our LM. The experiments and observations with the designed LM have been presented in Chapter 5. Finally the discussion on conclusion and future works has been done in Chapter 6.

Chapter 2

Task Specific ASR System

2.1 Data-set

2.1.1 Training data-set for acoustic model

The training data is required for the training of acoustic phone models. [7] designed 200 phonetically balanced English and Hindi sentences from middle school level reading material. This training data consists of 300 utterances comprising 5.2 hours of audio. It is developed using 30 fluent English and 11 fluent Hindi child speakers between the age of 10 to 14 years. Also, the acoustic model has been adapted on a small set of data acquired with the help of students of a school of Dahanu district in Maharashtra. The acquisition and description of this data-set has been described in details in [8]

2.1.2 Evaluation data-set

The evaluation data for all the experiments involved in this chapter has been obtained using the experiments conducted with the help of our campus school children. It consists of 15 different English stories read by 3 dis-fluent (i.e. making mispronunciations, repetitions, hesitation) speakers of age group 10-14 years. It comprises of 0.5 hours of evaluation data across 30 utterances.

2.1.3 Comparison of Training and Evaluation data-sets

The evaluation and training data described above is very different in terms of deviation from the canonical text. We can actually measure these deviation(or mistakes) in the utterances by comparing their canonical text with the corresponding ground-truth text.

The Table 2.1 shows very high deviation in Evaluation data in terms of WER and PER.

	Training data	**Evaluation data**
WER	5.40%	12.12%
PER	4.19%	8.70%

Table 2.1: WER and PER between canonical and ground-truth

We can also compare the dis-fluency or quality of the utterances in both data-sets by decoding them with an ASR system and comparing the performance through WER. By using google speech engine as the ASR (which is independent of both the data-sets), we found that it gives 5.92% WER for training data while for evaluation data it gives 16.81% WER (also shown in Figure 5.1) at best settings. It concludes that, evaluation data is very dis-fluent and has many mistakes in the utterances than the training data which is very smooth and fluent.

2.1.4 Training data-set for Language model

This is a text data contains 57 English and 23 Hindi stories each comprising around 10-20 sentences. This text data is used to train the N-gram models in the LM architecture whenever required during the experiment.
For the case of All_trigram and All_bigram LM architecture (as described in Section 5.4), all of these stories have been used to train the N-gram model inside it. Major problem with these architectures is that it doesn't give a way for a new story. But our best model doesn't require the entire text data only the current story text. So, it is also good for handing the case when a new story comes.

2.2 Acoustic Model

The acoustic model required can be the conventional acoustic model, because the task-specific constraint will only be added in the language model. So, we use the deep neural network (DNN) based acoustic model. Because of their high discriminating nature, the DNNs show better performance compared to Gaussian Mixture Model in building a good phone classification model (or acoustic models) on many speech recognition tasks. This architecture also has the distinct advantage in case of model adaptation with task-specific data due to its GMM-HMM back-end. The training procedure for the DNN Hybrid SAT models is provided below.

1. A SAT GMM-HMM system was trained on adaptation data (described in 2.1.1) by estimating fMLLR transforms for each speaker in the training set using raw MFCC features.

2. Speaker-normalized features were generated by transforming each speaker's data through the estimated fMLLR matrix.

3. A DNN having 40 dimensional bottleneck(BN) layer was trained using these speaker-normalized features.

4. These features were used to train a SAT GMM-HMM system using the same procedure as in Step 1.

The GMM-HMM model used has 1000 context-dependent tied HMM states with 8000 Gaussians shared across them. A single global speaker-specific fMLLR transform is estimated for each speaker in the training set. The decoding process used is the standard two-pass unsupervised fMLLR decoding process which uses the first pass decoding hypothesis of the SAT GMM-HMM system as the transcription labels for speaker-specific transform estimation. During decoding, fMLLR transform at the speaker-story level is also estimated. The DNN architecture with Bottleneck layer consists of 6 hidden layers with 1024 neurons each, with the penultimate hidden layer replaced by a 40-dimensional layer to lower dimensionality. Other implementation details such as the DNN training procedure, raw feature configuration etc. can be found in the extensive work in [9]

2.3 Task-Specific Language Modelling

The Language model(LM) of an ASR should be specific to the story which the child is reading and could also easily accommodate the Out-of-vocabulary words or any disfluency or repetition of words. There are many studies in literature which have proposed efficient Language models architectures compatible with our assessment task.

[1], [10] build LM for each sentence in the story. Figure 2.1 shows the finite state transducer(FST) of LM for the sentence "The moon smiled". It is designed by concatenating the FST for each word but there are few problems with this LM design:

1. Expected substitution ("di" for "the", or "moan" for "moon") for each word has also been added in parallel with the actual words with some probability. These

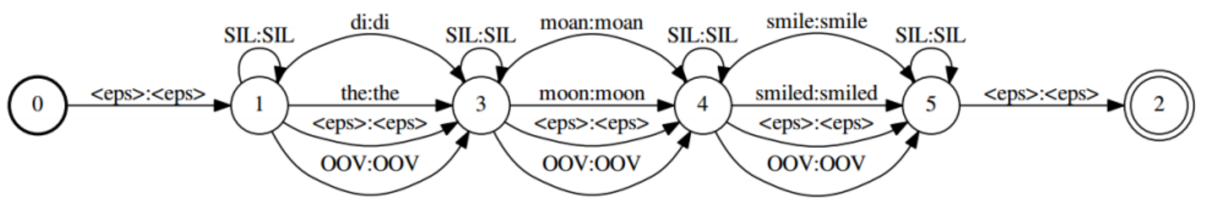


Figure 2.1: LM for the sentence “The moon smiled” [1]

could also be any meaningless words whose pronunciation is close to the target word; finding these expected substitution words can be an exhaustive task requiring annotated(ground-truth) data possibly.

2. If a word does not exactly match with the original or expected substitution path then it will follow an OOV path, which has also been put in parallel with each word. The OOV has been designed at phone level and also at fragment level. The decoded output is very much sensitive to the probability assigned to this OOV path because wrong value of probability can make it gobble up the neighbouring valid words.

3. No back loops are present to account for the repetition of words which is quite frequent in our children reading task. So, whenever there is a repetition of words, the first word may get recognized correctly but the successive words which may be correctly uttered will now be decoded as OOV substitution.

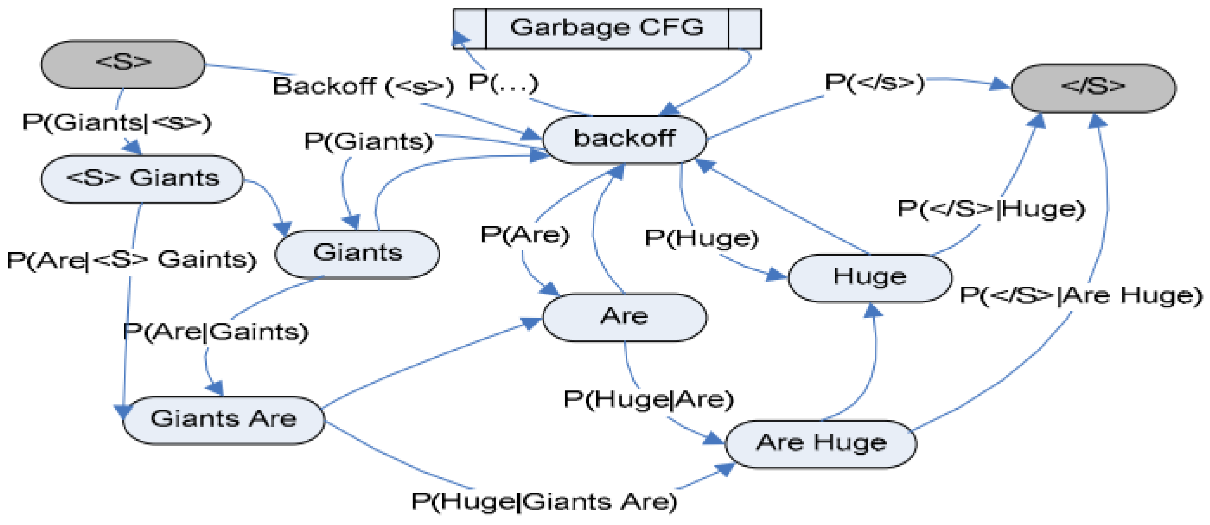


Figure 2.2: FST diagram of Trigram model along with garbage model for the sentence “Giants are huge” [2]

[2] proposes an efficient and robust LM which can be easily built on-the-fly with current reading sentences. With an additional parallel “garbage” model, the LM can also deal

effectively with a wide range of reading miscues. The architecture contains a target LM trained only using a limited training data (current story paragraph or sentences). Another LM "garbage" model trained on a general domain text is attached with the target model at the back-off state to improve robustness as shown in Figure 2.2. Due to the back-off state which is connected to all the states, it can easily deal with the repetition of words or dis-fluency. This garbage model act as an OOV-model which is built using a uni-gram model trained from the list of common words in general domain. There are total 1600 common words used in the garbage model. They have built both the target and garbage model using Context Free Grammar(CFG)(target CFG and garbage CFG). Because of its simple architecture, it is also highly scalable.

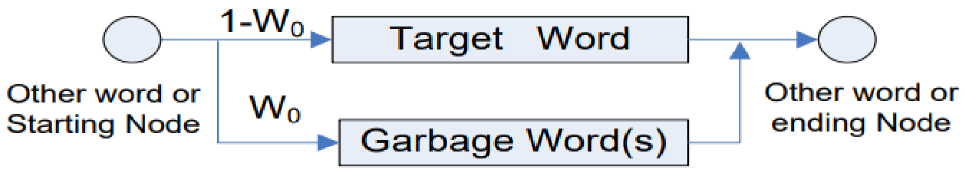


Figure 2.3: Weighted FST for each word [3]

Figure 2.3 shows the weighted Finite State Transducer (FST) for each word (or maybe pair of words). The weight W_0 represents the probability of going into the garbage model. For different weights the Word Error Rate(WER), Detection Rate and False Alarm have been compared in [3]. They claim their best LM contains tri-gram target model and uni-gram garbage model.

2.3.1 Our proposed LM architecture

Our language model is similar to that proposed in [2], [3] with slight modification. We have used a garbage model similar to uni-gram but with equal probability for all words. We call this model "zero-gram" . The motivation to introduce the "zero-gram" model is based on the fact that the mispronounced word may not necessarily be a frequent English word, and hence would not follow the normally observed probability given the correct pronunciation. As an example, consider the word "jumped". If a child pronounces it as "jump"+"aid", here "aid" is not as frequent as the word "the"(which has generally the highest probability in the uni-gram because of its high frequency of occurrence).
We have assigned every word as equally likely candidate for the mispronounced words. So, each word will have $1/N$ probability where N is the total number of words in the

dictionary. As shown in Figure 2.3, we multiplied each word in garbage model with W_0. Now the multiplication by $1 - W_0$ in the target model is similar to the division in garbage model, so each word will now have probability $\frac{W_0}{N(1-W_0)}$. Taking negative log(base 10) will give the cost= $log(1 - W_0) - log(W_0) + log(N)$. We can easily find the best W_0 using a small development set or we can set it according to the desired False Alarm rate(FAR).
The target model is an N-gram model trained on the all sentences of the current story text. The zero-gram garbage model however is built using a vocabulary of around 3000 English words which are accumulated from 57 English stories and other general words are added manually. We have built the finite state transducer(FST) corresponding to both the models using IRSTLM [11] and Openfst tools [12], the detailed implementation of system is described in Section 3.6.
We can use different order of N-gram in target as well as garbage model and the garbage model weight can be changed to different value to get a nice ROC curve. The garbage weight can be tuned accordingly for the desired False Alarm rate(FAR).

The performance of different variations of this architecture has been compared and presented in Chapter 5. The next Chapter discusses the N-gram LM in general and different smoothing algorithm used in the literature.

Chapter 3

N-Gram Language Models

Consider, we have been given an English word sequence, $W = w_1, w_2, w_3, ...w_n$.We need to find the probability of occurrence of this sequence P(W) using an LM model, we can decompose it using chain rule,

$$P(W) = p(w_1, w_2, w_3, ..., w_n) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)...p(w_n|w_1, w_2, ...w_{n-1}) \quad (3.1)$$

We proceed by making following Markov assumptions:

- Probability of current words only depends on its previous history
- Older words are less relevant (so only k previous words are included)

So, for a 2-gram (bigram) language model, here we consider only one previous word

$$p(w_1, w_2, w_3, ..., w_n) \approx p(w_1)p(w_2|w_1)p(w_3|w_2)...p(w_n|w_{n-1}) \quad (3.2)$$

Here,

$$p(w_2|w_1) = \frac{Count(w_1, w_2)}{Count(w_1)} \quad (3.3)$$

Similarly, if we consider N-1 previous words then it is called N-gram, in that case we need the count of word sequence of length N which may or may not have occurred in our(training) corpus. We can't assign the probability to zero just because it had never occurred in the training corpus, so there are many "smoothing" algorithms in the literature like "Add-One smoothing". It adds the count one to every possible n-gram. Since no.of unseen n-grams are much larger than the seen n-gram, too much probability mass gets shifted to fill the zeros. This highly affects the actual(seen) probability counts.

So, we need to be more logical in assigning the probabilities of the unseen N-grams. We need to use the count of things that we have seen to estimate the probability of the things that we have never seen. If we want to calculate the probability of an unseen trigram then we can estimate its probability by the corresponding bigram. Similarly we can estimate unseen bigram probability from its corresponding unigram probability. Generally, backoff and interpolation algorithms are used to estimate these probabilities.

3.1 Interpolation

Higher order n-grams have very sparse counts but are good for learning the context. On the other hand lower order n-grams have robust counts but have a limited context. We appropriately combine these in interpolation technique to get the best out of both.

So, we mix the probabilities of all the N-grams i.e. we take the weighted interpolation of tri-gram,bi-gram and uni-gram counts.

$$P_I(w_n, w_{n-1}, w_{n-2}) = \lambda_1 P(w_n, w_{n-1}, w_{n-2}) + \lambda_2 P(w_n, w_{n-1}) + \lambda_3 P(w_n) \tag{3.4}$$

where, $\lambda_i \in R^+, i = \{1, 2, 3\}$ are generally calculated on a held-out corpus (some part of training corpus is held-out to train these hyper parameters). In general, we can write the interpolation probability as,

$$P_n^I(w_i|w_{i-n+1}^{i-1}) = \tau(w_i|w_{i-n+1}^{i-1}) + \gamma(w_{i-n+1}^{i-1})P_{n-1}^I(w_i|w_{i-n+2}^{i-1}) \tag{3.5}$$

where τ is the modified probability of the seen sentence and γ is the scaling factor for including the probability of lower N-grams.

3.2 Back-off

In case of back-off, we "back-off" to the lower order N-gram only if we have zero counts of the current N-gram. The general formulation of the N-gram probability can be written as eq. 3.6

$$P_n^B(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \tau(w_i|w_{i-n+1}^{i-1}) & \text{if} \quad count(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1})P_{n-1}^B(w_i|w_{i-n+2}^{i-1}) & \text{if} \quad count(w_{i-n+1}^{i}) = 0 \end{cases} \tag{3.6}$$

Because of these extra probabilities assigned to the unseen n-gram, there will be a discounting factor associated with each seen n-gram to compensate for the overall probability sum. τ and γ have been defined differently by different algorithms which also accounts for these discounting factors. We have used and compared two such algorithms in our task of assessment, one is 'improved Kneser-Ney smoothing' and the other one is 'Witten-Bell discounting'.

3.3 Improved Kneser-Ney Smoothing

Initially bi-gram probability using Kneser-Ney algorithm is defined by

$$P_{KN}(w_i|w_{i-1}) = \frac{max(count(w_{i-1}, w_i) - d, 0)}{count(w_{i-1})} + \lambda(w_{i-1})P_{CONTINUATION}(w_i) \tag{3.7}$$

λ here is context dependent, $\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})}|w : c(w_{i-1}, w) > 0|$ and d is the discounting factor subtracted from each count. Using experiments, the value of d is generally found to be close to 0.75 for higher counts. The continuation probability in eq.3.7 is defined as:

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w_i) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|} \tag{3.8}$$

It is the measure of how many times the word "w" appears as a novel continuation [13]. Denominator represents the total number of word bi-gram types.

By adding this probability to the N-gram probability, the chance of the word to club with other words also gets enhanced.

However, [14] modified this to use interpolation technique. They have also introduced multiple discounting factors for different counts of N-gram. So the τ_n and γ_n in the above interpolation equation are defined as [15]

$$\begin{aligned} \tau_n(w_i|w_{i-n+1}^{i-1}) &= \frac{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i}))}{\sum_{w_i} c(w_{i-n+1}^{i})} \\ \gamma_n(w_{i-n+1}^{i-1}) &= \frac{\sum_{j=1}^{3} D_j N_j(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})} \end{aligned} \tag{3.9}$$

where, D_j, the discounting factor, depends on the count of the N-gram and

$$N_j(w_{i-n+1}^{i-1}\bullet) = |\{w_i|c(w_{i-n+1}^{i-1}) = j\}| \tag{3.10}$$

is the number of words that appear after the context w_{i-n+1}^{i-1} exactly j times.

3.4 Witten-Bell Discount

In ths method, instead of subtracting from the numerator, discounting is done by addition of a factor in the denominator:

$$N_{1+}(w_{i-n+1}^{i-1}\bullet) = |\{w_i | c(w_{i-n+1}^{i-1}, w_i) > 0\}| \tag{3.11}$$

which is the total number of words that appear after the context w_{i-n+1}^{i-1}. Therefore, the τ_n and γ_n here are defined as [15]

$$\begin{aligned} \tau_n(w_i|w_{i-n+1}^{i-1}) &= \frac{c(w_{i-n+1}^{i})}{N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i})} \\ \gamma_n(w_{i-n+1}^{i-1}) &= \frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i})} \end{aligned} \tag{3.12}$$

3.5 Evaluation of Language Models

Now that we have our language model, we can compare the performance of two LMs by using these LMs into other system and compare the outputs, this is called extrinsic evaluation. In this work, we have reported this comparison between the Improved Kneser-Ney and Witten-Bell algorithms. We can also compare the performance of the LM intrinsically with the help of perplexity, which is defined as:

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}} \tag{3.13}$$

Using the LM, we can easily find the perplexity of an unseen sentence. The best model will be the one which gives lowest perplexity i.e. higher probability for an unseen sentence or better predicts the sentences. In general, the perplexity decreases as we increase the order of N-gram.

Ismail in his paper [15], has compared both the discounting algorithms, and using perplexity as a intrinsic evaluation metric, found that Improved Kneser-Ney gives better perplexity value than compared to Witten-Bell on a small text data. We have also compared the performance of both the algorithms on our evaluation data. The observations and discussion regarding same has been described in Section 5.3.

3.6 Implementation details

We have used following scripts to build and execute our programs:

```
build-lm.sh -i input_training.txt -n 3 -s witten-bell -o lm.gz
```

We can give the training text for building the LM in input_training.txt. Every sentence should start and end with sentence starting $< s >$ and sentence ending $< /s >$ symbols. The order of N-gram to prepare can be specified using -n and smoothing algorithm to use in the LM can be specified by -s (witten-bell or improved-kneser-ney). The output will be stored in a gzipped file lm.gz.

```
compile-lm lm.gz -t=yes lm.arpa
```

This compiles the lm.gz to get lm.arpa file which contains the count of each N-gram. Further to make an fst using these counts we use openfst tool [12]. But making an fst in itself requires a text file containing the node connections of fst. So, we need to convert the lm.arpa file into a text file which contains the node connections in the prescribed format described in [12]. For this we use a python based script [16]

```
convert_lm_to_fst.py lm.arpa G.txt
```

Finally we use openfst tool to make fst

```
fstcompile --isymbols=words.txt --osymbols=words.txt G.txt >G.fst
```

This takes a words.txt file which contains the integer symbol corresponding to each words and builds an ".fst" file Now, we use kaldi [17] to finally create a fully expanded decoding graph that represents all the language-model, pronunciation dictionary, and HMM structure.

```
utils/mkgraph.sh <lang-dir> <model-dir> <graph-dir>
```

$< lang-dir >$ is the language directory where G.fst is present and $< model-dir >$ is the Acoustic model directory This script is used for making the complete graph (HCLG.fst) used for decoding in Kaldi. This script first composes L.fst (transduces phones to words) and G.fst (transduces word to word sequences) to form LG.fst. This is followed by composing with C.fst (transduces triphones to monophones) to form CLG.fst. Finally CLG.fst is composed with H.fst (transduces frame sequences into triphones) to give HCLG.fst. The script then puts the complete graph in $< graph - dir >$

```
steps/nnet/decode_modified.sh --beam 13 <graph-dir> <data-dir>
<decode-dir>
```

Here, $< data - dir >$ contains the data on which the decoding will be done. Before giving it to this script it gets modified by $MakeFeatures\_KaldiData.sh$ script which makes the fMLLR features as described in [9]. We have fixed the beam size for the decoding graph to be 13. Since the algorithmic complexity of the decoding is $O(beam\_size^2 * Time)$, a lower value of beam size will ensure faster decoding. The script puts the output after decoding in $< decode - dir >$.
To join an fst with another at a particular node

```
fstreplace A.fst -1 B.fst node_label out.fst
```

This will attach B.fst in A.fst at node with label "node_label". We are using this to combine target model and garbage model of our LM, this has been explained in Section 2.3.1.

The next Chapter presents different evaluation metrics which have been used in our experiments for comparisons.

Chapter 4

Evaluation Metrics

Evaluation metric explains (quantifies) "How good your model is?" by comparing the model output (predicted results) with the ground-truth result. A good evaluation metric should have the capability of discriminating the performance of different models. Whatever model we make, we get the feedback of its performance only from the evaluation metric. This feedback helps to decide how to improve the model. Having a bad evaluation metric could lead to a worse model. So, requirement of a good evaluation metric in any task is very important. Before we actually go to the evaluation metric, we will look at a tool which is good for comparing two sequences, "Editdistance".

4.1 Edit distance

It is a way to quantify the dissimilarity between two strings by calculating the minimum number of operations required to convert one string into the other. For exactly same strings this cost is zero. There are different variants for edit distance but here we will be using what is called as the "Levenshtein Distance" where insertion, deletion and substitution in one string is allowed for the transformation to other string. Using Levenshtein method, the edit distance between the sequences $a = a_1, a_2...a_n$ and $b = b_1, b_2...b_m$ is given by d_{mn} , where $D = \{d_{ij}\}$ is a matrix whose elements are filled by the following Dynamic Programming algorithm (algorithm 1).

Because of the nested for loop inside another for loop, this algorithm takes O(mn) time to fill in all the values of matrix D. Now, we will go for the evaluation metric used in this task and will see their pros and cons:

Algorithm 1 Edit distance

```
function EDITDISTANCE($a, b$)
    d[i,j] =0
    for $i \leftarrow 1$ to $len(a)$ do
        d[i,0]=i
    for $j \leftarrow 1$ to $len(b)$ do
        d[0,j]=j
    for $i \leftarrow 1$ to $len(a)$ do
        for $j \leftarrow 1$ to $len(b)$ do
            if a[i]=b[j] then
                cost=0
            else
                cost=1
            d[i,j]=min{d[i-1,j-1]+cost,d[i-1,j]+1,d[i,j-1]+1}
    return $d[len(a) - 1, len(b) - 1]$
```

4.2 Word Error Rate(WER)

For the ASR task, Word Error Rate(WER) is the conventional evaluation metric to compare the performance of different models. It is generally calculated by using the "editdistance" between the decoded and transcribed sequences of words.

$$WER = \frac{editdistance(transcribed, decoded)}{len(transcribed)} \tag{4.1}$$

There are few problems with this metric when:

- a word differs by only a single letter like "hunter" → "hunters", it will consider it as a substitution error.
- two words are clubbed together like "every time" → "everytime", it will be treated as substitution followed by deletion!

So, even if the decoder is predicting the results very close to the ground truth results, the penalty seems to be very high. This word level comparison of the two strings does not reflect the true performance of the decoder hypothesis. We need to compare the strings at the sub-word level (or phone level).

4.3 Phone Error Rate(PER)

It compares the strings at the phone level. For this, we convert all the words into their phone sequences using word to phone mapping dictionary. Now we have a long sequence of phones instead of words. We then calculate the editdistance between this long decoder phone sequence with its ground truth counterpart, which we call as Phone Error Rate (PER). This PER is a very good measurement metric which also tells us how close the decoded word is with respect to the ground-truth.

$$PER = \frac{editdistance(trans_{phone}, decoded_{phone})}{len(trans_{phone})} \tag{4.2}$$

e.g. :
Ground-truth:- He exercise for two hours every day
Decoded:- He exercises for 2 hours everyday
Editdistance = 4. So, WER = 57.14
Ground-truth:- hh/II ex/k/s/aa/r/s/ay/z f/OO/r T/UU AA/w/aa/r/s ex/w/r/II D/ee
Decoded:- hh/II ex/k/s/aa/r/s/ay/z/ee/z f/OO/r T/UU AA/w/aa/r/s ex/w/r/II/D/ee
Editdistance = 2. So, PER = 6.89

Both decoded and ground-truth sentences are almost similar but WER is too high whereas PER is very low as expected.

4.4 Miscue Detection Rate

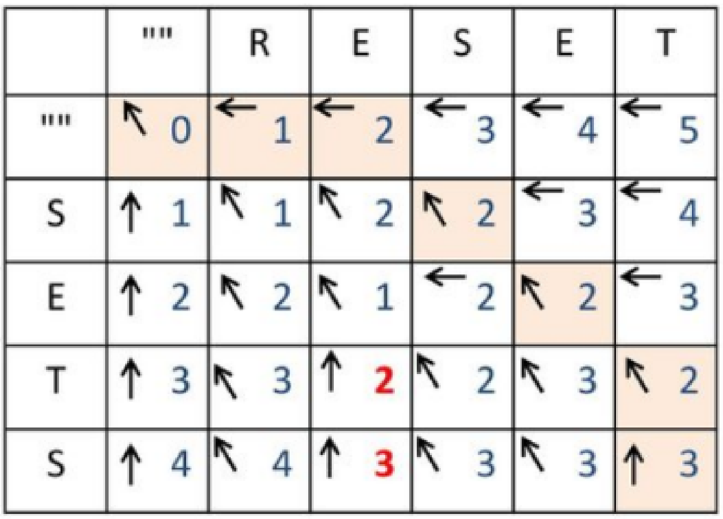
We are more interested in the miscue detection (“the event”) not on the actual “miscued” word which has been decoded or whether it is matching with the ground truth or not. We are just interested in finding the correct miscue, if there is a miscue reported by ground-truth than whether the decoder detects the miscue at the same “place” or not. For this we compare our ground-truth with the canonical text and using the backtracking path in the calculation of edit distance, we can easily classify whether the word is correctly spoken or substituted or inserted or deleted (“CSID”). The following algorithm is the modification to the editdistance algorithm to also get the optimal path using backtracking and returns the “CSID” sequence:

Fig. 4.1 shows the edit graph representation of the editdistance algorithmic flow as discussed in Section 4.1. The vertical and horizontal arrows represent deletion(“D”) and insertion(“I”) operation respectively whereas every diagonal arrow represents the correct (“C”) or substitution (“S”) depending on whether both symbols are same or different. This will give us a sequence of “CSID” symbols, where S,I,D represent the miscues.

We find this CSID sequence for ground-truth and decoded text by comparing them with

Algorithm 2 Edit distance with backtracking

```
function GET_CSID($a, b$)
    d[i,j] =0
    bt =                                                  ▷ initialize the dictionary
    for $i \leftarrow$ 1 to $len(a)$ do
        d[i,0]=i
        bt[(i,0)] =(i-1,0,'d')                            ▷ Vertical arrow showing deletion
    for $j \leftarrow$ 1 to $len(b)$ do
        d[0,j]=j
        bt[(0,j)] =(0,j-1,'i')                            ▷ Horizontal arrow showing insertion
    for $i \leftarrow$ 1 to $len(a)$ do
        for $j \leftarrow$ 1 to $len(b)$ do
            if a[i]=b[j] then cost=0
            else cost=1
            d[i,j]=min{d[i-1,j-1]+cost,d[i-1,j]+1,d[i,j-1]+1}
            if d[i,j] = d[i-1,j]+1 then
                bt[(i,j)] =(i-1,j,'d')
            else if d[i,j] = d[i,j-1]+1 then
                bt[(i,j)] =(i,j-1,'i')
            else if d[i,j] = d[i-1,j-1]+1 then
                bt[(i,j)] =(i-1,j-1,'s')
            else bt[(i,j)] =(i-1,j,'d')
                                                          ▷ Starting to backtrack
    i=len(a)-1
    j=len(b)-1
    path=[]
    while $(i, j) \neq (0, 0)$ do
        (i,j,p) $\leftarrow$ bt[(i,j)]
        path = [p]+path
    return path,d
```



	""	R	E	S	E	T
""	↖ 0	← 1	← 2	← 3	← 4	← 5
S	↑ 1	↖ 1	↖ 2	↖ 2	← 3	← 4
E	↑ 2	↖ 2	↖ 1	← 2	↖ 2	← 3
T	↑ 3	↖ 3	↑ 2	↖ 2	↖ 3	↖ 2
S	↑ 4	↖ 4	↑ 3	↖ 3	↖ 3	↑ 3

Figure 4.1: Edit Graph [4]

the canonical text. If the size of ground-truth and decoded text are different, then so as their CSID sequences. But for finding the detection rate we need one-to-one mapping of these CSID sequences. Since the difference of length is only because of the insertion operation, we remove all the "I" occurrences from the sequence and place a miscue sign (say "M") just before every first occurrence of "I". We also replace all "S" and "D" symbols by "M". Basically, we convert the CSID sequence into a miscue("M") and non-miscue("C") sequence because we can't actually calculate the individual detection rate for each miscue type (because of the non-uniqueness of the backtracking path in editdistance calculation). So, the string "CICCSIICD" gets converted into "MCCMCM".

Algorithm 3 CSID to binary miscue conversion

```
function CSID_TO_MISCUE(path)
    vector=[]
    if path[0]=='i' then
        vector.append(1.)                          ▷ '1' represents miscue
    else
        vector.append(0.)                          ▷ '0' represents correct
    for i in path do
        if i=='0' then
            vector.append(0.)
        else if  i in ('d','s') then
            vector.append(1.)
        else
            vector[-1] = 1. ▷ if there is insertion then make the previous one as miscue
    return vector
```

Now that we have two sequences of miscue, we can calculate the Detection Rate (DR) and False Alarm Rate (FAR).

$$\begin{aligned} DR &= \frac{TP}{TP+FN} \\ FAR &= \frac{FP}{FP+TN} \end{aligned} \tag{4.3}$$

In these equations,

True Positive (TP) is correctly detecting a miscue "M" in both

False Negative (FN) is incorrectly detecting no miscue → "M" in ground-truth and "C" in decoded,

False Positive (FP) is incorrectly detecting miscue →"C" in ground-truth and "M" in decoded and

True Negative (TN) is correctly detecting no miscue → "C" in both.

Consider the following example:

- Canonical text : We were very happy
- Ground truth text: We where a very happy happy
- Hypothesized text: We were aware happy happily

The CSID sequence for Ground-truth and Hypothesized text are "CSICCI" and "CCSCI" respectively. It will be then converted to "CMCM" and "CCMM" respectively. In this case, we have 1 TP since we have correctly identified the repetition of word in the last ("M" in the last matches in both), 1 TN since we have correctly recognized 'We' ("C" in the starting matches in both). Incorrectly recognizing 'where' as 'were' is FN ("M" in ground-truth and "C" in hypothesized) and incorrectly recognizing 'very' as 'aware' is FP ("C" in ground-truth and "M" in hypothesized). So, both the miscue detection and false alarm rate will be 50% in this case.

In the previous Chapters, we have looked at all the components to design and evaluate our LM. The experiments and observations with the designed LM have been presented in Chapter 5.

Chapter 5

Experiments and Observations

This chapter describes all the experiments performed for the assessment task. Corresponding observations and inferences are discussed which provide useful insight into our problem of designing better language models for automatic reading assessment. We use the Kaldi [17] speech recognition toolkit for all our experiments. Kaldi is a freely available toolkit licensed under the Apache License v2.0 and written in C++.

5.1 Google Speech Engine

This assessment task requires a powerful ASR system, so we started our experimentation with the google's ASR system which generally seems to be good for the recognition task. Google offers a speech API, the installation and authentication details are given in [18]. This is a chargeable service which costs $0.006 per 15 seconds, but there is provision for $300 wallet amount for the 1st year. The api gets called using python by

```
response = client.recognize(config, audio)
```

where, client is an initial variable defined by SpeechClient() function which is imported from google.cloud.speech library, config is a configuration variable defined using a class "types" inside google.cloud.speech library. It tells the recognizer about the audio encoding or sampling rate and also tells about the configuration of language model to use.

```
config = types.RecognitionConfig(
    encoding=enums.RecognitionConfig.AudioEncoding.LINEAR16
    ,sample_rate_hertz=16000
    ,language_code='en-IN'
    ,profanity_filter=True
```

```
        ,max_alternatives=5
        ,enable_word_time_offsets=True
      ,speech_contexts=[types.SpeechContext(phrases=phrase)]
            )
```

For different languages and accents, different language code has been provided in [19]. We have set language_code="en-IN" where 'en' means English language and 'IN' means Indian dialect. The profanity_filter can be set to hide the decoded abusive words. The maximum number of recognition results is determined by max_alternative and to get the word level alignment, enable_word_time_offsets can set to "True". This api also provides an option for giving the context of the audio to be decoded which is extremely important for our assessment task, it needs a "guided" ASR system. We can give this context as a variable phrase which is the list of sentences. So, the canonical text (which child was supposed to read ideally) have been given as context in different ways. We can give the list of all words in the canonical or the list of consecutive pairs or triplets of words can also be given. The output "response" of the google api gives a confidence with each recognition result, we select the one with the highest confidence and compare it with the ground-truth to get the word error rate (WER) as defined in Section 4.2.
In the case of long audios (> 1min), we need to put the audios in the google's cloud storage first, and then call a different API.

```
response=client.long_running_recognize(config,
  types.RecognitionAudio(uri=''gs://audio_path_in_cloud_storage"))
```

Figure 5.1 shows the WER with different orders of context when we run it on the evaluation data as described in subsection 2.1.2. It can be observed that the WER without any context(canonical story text) is really high(27.36%). After giving the uni-gram of context, it gets decreased slightly(to 26.6%) but a drastic decrease can be seen when bi-gram context is given, WER gets decreased almost by 10% (to 16.8%). After the 2nd order there is hardly any improvement in WER. This result shows that giving certain level of guidance (using canonical story text) to the ASR system enhances its performance compared to the general ASR system.

5.1.1 Challenges with Google's speech engine

1. We calculated the WER between ground-truth and canonical and found that it is 12.12% only. The decoded text from google at best setting is giving 16.81% wrt

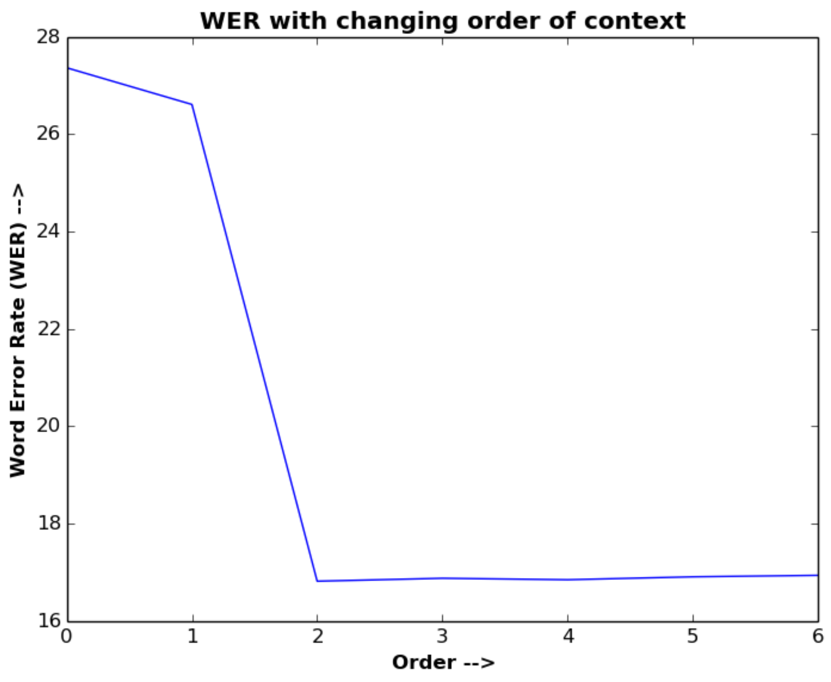


Figure 5.1: Word Error Rate using Google Speech API

ground-truth, so it is not wise to use a decoder like that when using the canonical text itself can give better results.

2. The word and phone level alignment data of decoded text is very crucial for our prosody-level assessment task. Google's speech engine only gives word level alignment (by setting the enable_word_time_offsets parameter in config variable), which itself is not entirely correct. It doesn't remove the silences between the words from the alignments, the silences are clubbed together with the adjacent words

3. When we calculated the Detection rate and False Alarm rate (as described in Section 4.4, we got only 43.4% detection rate(DR) at 4.1% false alarm rate(FAR), at different settings the detection rate increases by 17% but so does the FAR by 12%. For the assessment task, we need really low FAR to ensure child's constructive engagement with the feedback.

4. For using this system, we need to be always connected to internet which might not be available in the remote areas where our target children are.

5. Also it is a paid system, they billed it for each utterance as described in starting of this section.

Our proposed system (in Section 2.3.1) tackles all these problems. The experiments with the proposed architecture and it's comparison with the slight modification of itself is presented in the next section.

5.2 Our LM architectures

As described in Section 2.3.1, we are using a tri-gram target model and zero-gram garbage model. We want to see the performance of the system with different weights(W_0) of the (zero-gram) garbage model. Figure 5.2 shows WER and PER at different values of weights. It achieves minimum WER and PER as low as 7.26% and 3.95% respectively, which is much better than what we got with google's speech engine. This value has been achieved at $log(1 - weight) - log(weight) = 1.32$ which corresponds to weight=0.046, it means 0.046 probability is given to the garbage model to achieve the best WER. If we consider the resolution error of plotting the figure, then this probability could go from 0.02 to 0.10. The left side of the plot shows the higher probability and right shows lower probability

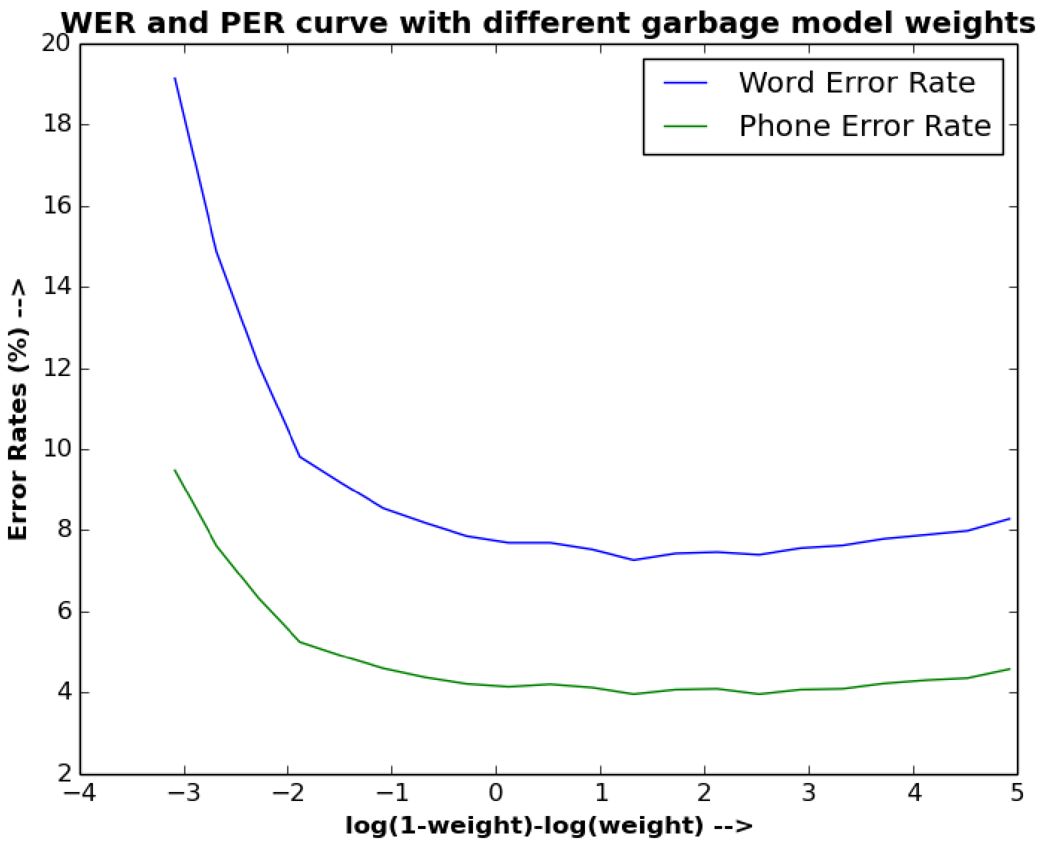


Figure 5.2: WER and PER curve with different weights of garbage model

given to the garbage model. That's why the right portion saturates to the case where no garbage model is used, only tri-gram model on target canonical text. And left side rapidly jumps to the case where only garbage model is used (means zero-gram model on all words).

We have also evaluated the proposed model in terms of Detection rate and False Alarm rate.

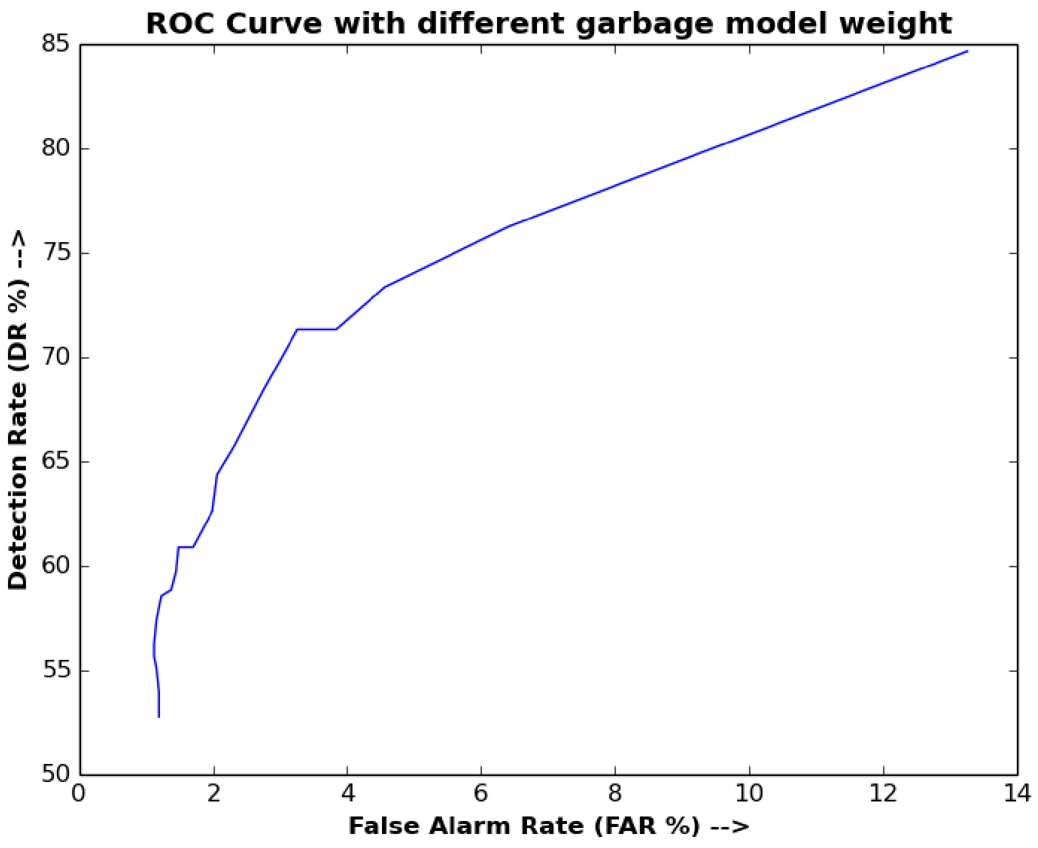


Figure 5.3: ROC curve by changing weight of garbage model

Figure 5.3 shows the Receiver Operating Characteristic (ROC) curve. We can see that, for FAR of around 4% we get miscue Detection rate of around 71.80% compared to the 43.4% with the google's speech engine. The ROC curve still increases steeply up-to 84.4% DR at 13.2% FAR. In our task we need very low FAR to ensure the constructive engagement of a child with the feedback. So, we compare the ROC curve in the lower FAR reason only.

5.3 Comparison based on different N-gram models

For the fixed garbage model, we have changed the smoothing algorithm used in the tri-gram target model. Witten-bell Discount and Improved Kneser-Ney Smoothing (as described in Section 3.4 and 3.3) have been compared in Figure 5.4 and 5.5. We can see that for both the metrics (WER & PER), Witten-bell performs much better as opposed to the claim of [15] with the Bahasa Indonesia data.

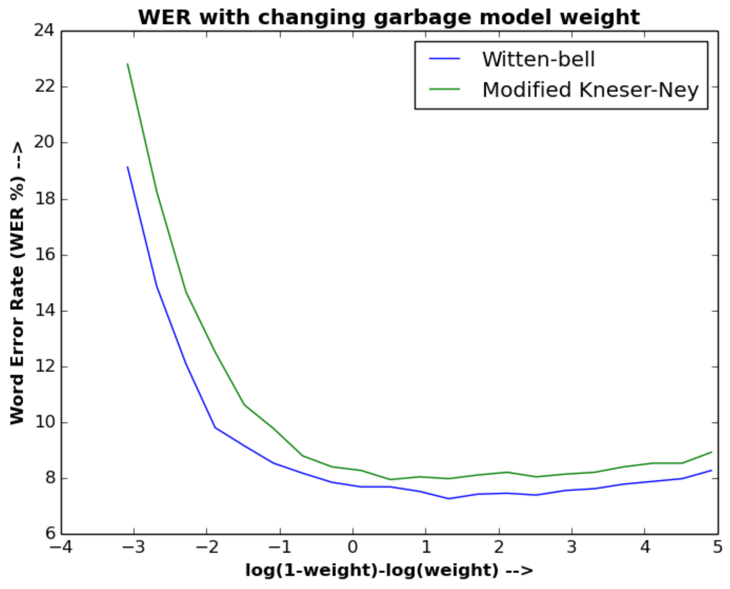


Figure 5.4: WER by changing weight of garbage model

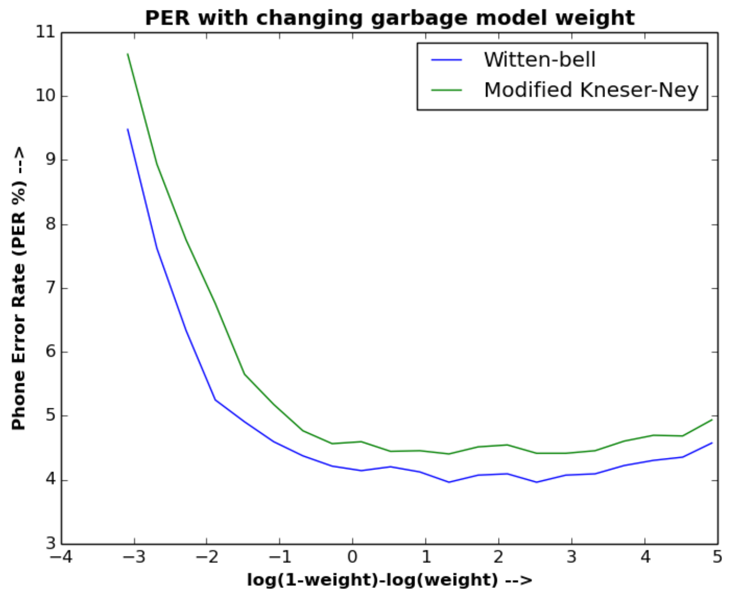


Figure 5.5: PER by changing weight of garbage model

However, ROC curve shown in Figure 5.6 shows almost equal performance of both the algorithms except in the lower FAR reason where still Witten-bell performs better.

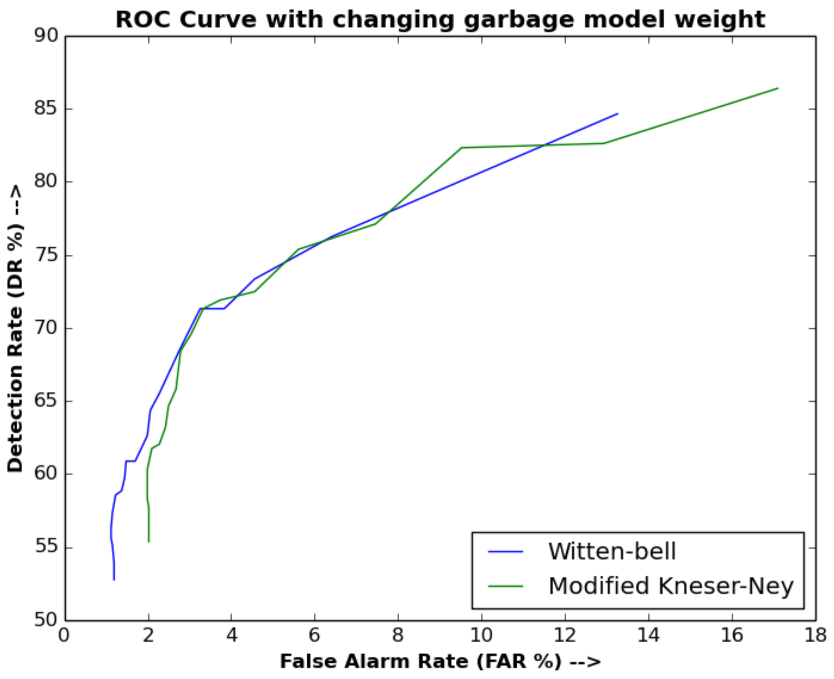


Figure 5.6: ROC curve by changing weight of garbage model

5.3.1 Why Witten-Bell performs better here

As described in Section 3.2, probabilities are given to the unseen N-gram also. To compensate for the overall probability sum, some probabilities are subtracted from the seen N-grams i.e. discounting is done in their counts. This discounting is done differently for different counts. Using the formulation of both the algorithms (Witten-bell and modified Kneser-Ney as described in Section 3.4 and 3.3 respectively), we draw a rough plot of the probability left after discounting from the N-gram counts as shown in Figure 5.7. For lower counts, it clearly shows that higher probability is assigned in Witten-Bell than compared to Improved Kneser-Ney. For our task of assessment, where there are many mistakes done by children, a higher probability for the lower count grams (which generally occupies most of the dictionary) is desirable. Witten-Bell algorithm is basically shifting some probability mass from higher counts to the lower counts.

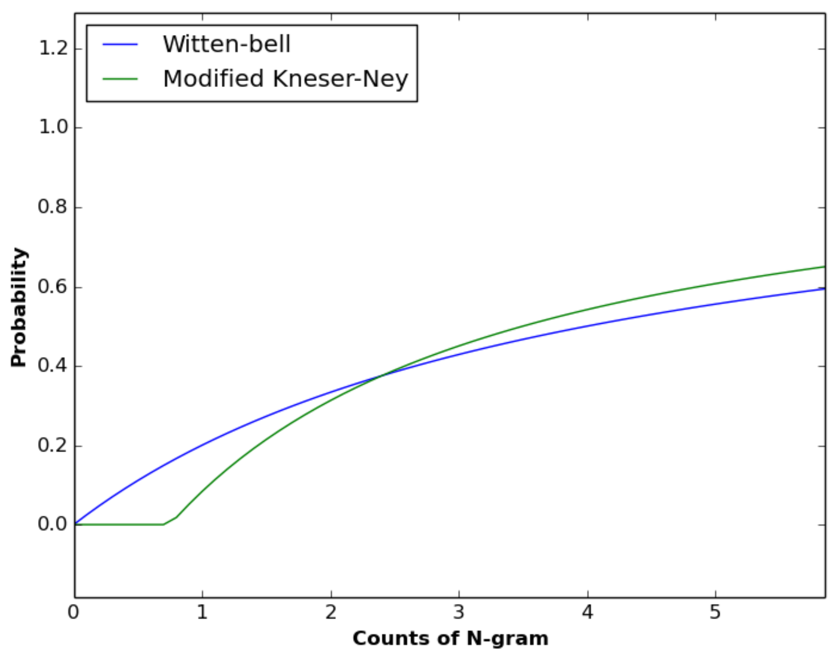


Figure 5.7: Probability left after discounting from N-gram counts

The dummy probability functions used to draw the plots in Figure 5.7 are:

$$y = \begin{cases} \frac{x}{N+x+C} + B & \text{for Witten-Bell} \\ \frac{max(x-d,0)}{x+C} + B' & \text{for Modified Kneser-Ney} \end{cases} \tag{5.1}$$

where, d is the discounting factor (taken as 0.75). N,C,B and B' are constant with respect to the N-gram count. The intersection point in the graph will change for different values of these constants.

5.4 Comparison based on different LM architectures

Using the flexibility we have in changing the order of N-gram models which we can use in target and garbage model, we can have following LM structures:

1. Trigram_zerogram - Target model is tri-gram and garbage model is zerogram
2. Bigram_zerogram - Target model is changed to bi-gram while garbage model is still zerogram

3. Trigram_unigram - Target model is tri-gram but the garbage model is uni-gram. This architecture is same as that in [2], [3] papers.

4. All_trigram - Here the language model is just a trigram model trained on 80 Hindi and English stories(No garbage model has been used)

5. All_bigram - Similar to All_trigram model, here we have used bigram model to train.

We have used Witten-Bell smoothing in all the above models. ROC curve in Figure 5.8 shows the performance of these models on the evaluation data by varying the garbage model weights. Clearly, Trigram_zerogram model performs well for almost all the regions, better than Trigram_unigram([2] [3]). It is also way better compared to the google's speech engine.

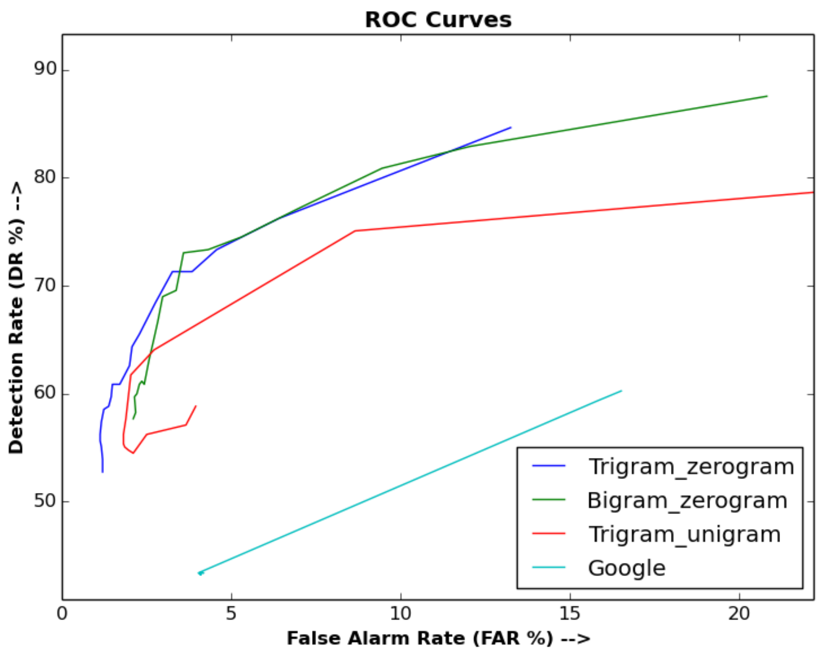


Figure 5.8: ROC Curves for different models

We have also compared them with respect to the WER, PER and the Detection rate at 5% FAR as shown in Table 5.1.

Sr. No.	Target N-gram	Garbage N-gram	Smoothing Algorithm	WER (%)	PER (%)	DR(%) at 5% FAR
1	Trigram	zerogram	Witten-bell	**<u>7.26</u>**	**<u>3.95</u>**	74.03
2	Trigram	zerogram	Modified Kneser-Ney	7.95	4.44	73.73
3	Trigram	unigram	Witten-bell	8.08	4.25	68.47
4	Bigram	zerogram	Witten-bell	8.08	4.42	**<u>74.15</u>**
5	All Trigram		Witten-bell	8.60	4.70	(63.18, 2.42)
6	All Bigram		Witten-bell	10.55	5.87	(71.59, 5.24)
7	Google's with bigram context			16.81	-	(43.4, 4.1)

Table 5.1: Table showing the performance of each models

Clearly, for all the cases Trigram_zerogram model(proposed model) performs better except the miscue detection rate at 5% where Bigram_zerogram is better. Both Detection and False Alarm rate in case of All_trigram, All_bigram and Google speech engine has been shown, because we can't change any parameter there to get 5% FAR. Google speech engine's performance has also been shown which is worst of all for our assessment task.

5.5 Comparison based on different Acoustic Model weight

We have fixed out LM model with the trigram target model and zerogram garbage model and compared the performance of the Assessment system by changing weight that has been given to the acoustic model compared to the LM model.

Figure 5.9 shows the WER and PER on Evaluation data with different acoustic weights. We get a minimum WER and PER of 7.23% and 3.95% respectively at 1/27 Acoustic weight. The values in Table 5.1 has been computed with 1/20 Acoustic weight.

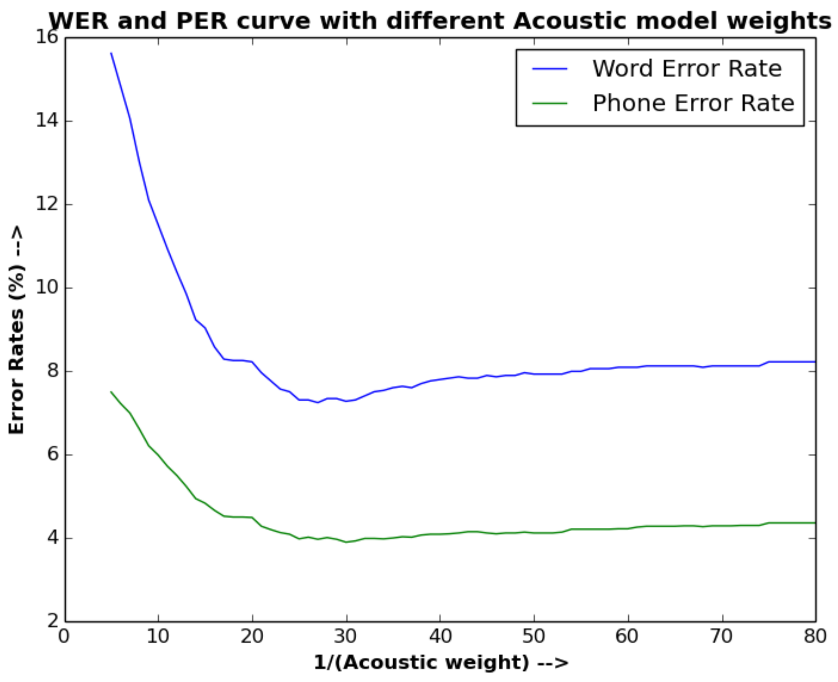


Figure 5.9: WER and PER with different Acoustic weight

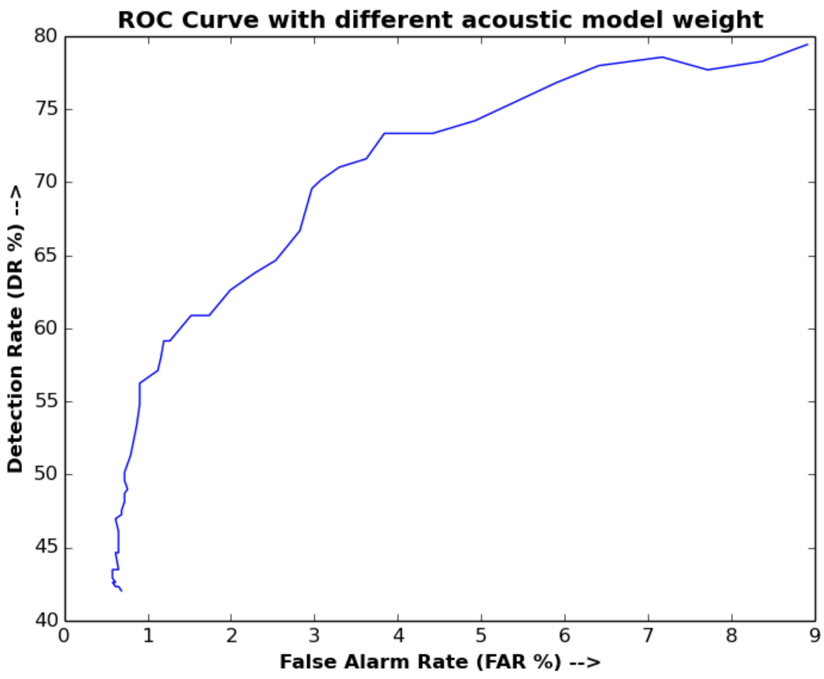


Figure 5.10: ROC curve using different Acoustic weight

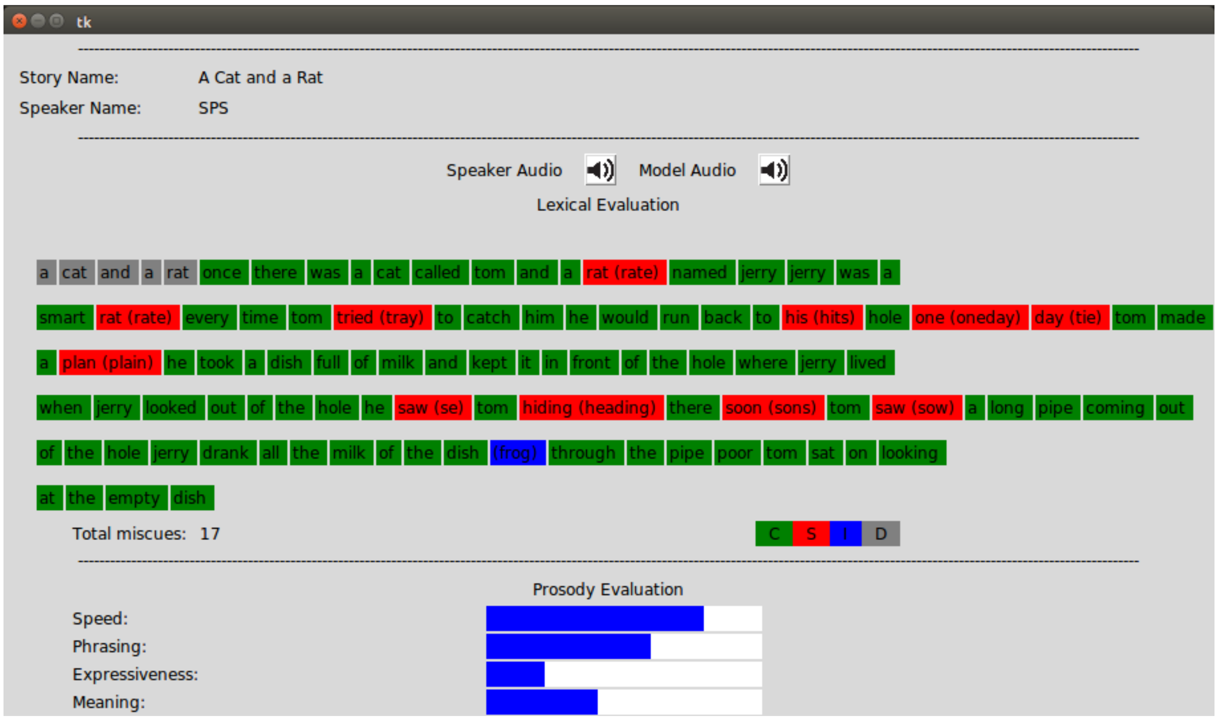
Figure 5.10 shows the ROC curve using different acoustic weights. At 5% FAR, we are getting 74.42% miscue detection rate. Both WER and miscue detection rate at 5% FAR got increased slightly at different acoustic weight. This is another parameter along with garbage model weight that we need to tune for different variations of children speech, described in subsection 5.7.

5.6 Showing Output on GUI

We have created a Graphical User Interface(GUI) in python that displays the lexical and prosodic evaluation carried out using our system. Audio feedback (comparison with an ideal recording of the same text) and visual feedback (showing all the miscues) are provided as depicted in Figure 5.11. The display is designed using Tkinter library [20] of python. The lexical miscues are marked on the displayed story text using colour codes.

- Green represents that the word has been pronounced correctly
- Red represents that the word is substituted, the detected substituted word is shown in brackets
- Blue represents an inserted word, also has been put in brackets
- Gray tells that the word has been missed

Total miscues has also been shown below it. The prosodic evaluation shown further is the dummy one, to be implemented as in [7].



Story Name: A Cat and a Rat
Speaker Name: SPS

Speaker Audio Model Audio
Lexical Evaluation

a cat and a rat once there was a cat called tom and a rat (rate) named jerry jerry was a
smart rat (rate) every time tom tried (tray) to catch him he would run back to his (hits) hole one (oneday) day (tie) tom made
a plan (plain) he took a dish full of milk and kept it in front of the hole where jerry lived
when jerry looked out of the hole he saw (se) tom hiding (heading) there soon (sons) tom saw (sow) a long pipe coming out
of the hole jerry drank all the milk of the dish (frog) through the pipe poor tom sat on looking
at the empty dish

Total miscues: 17 C S I D

Prosody Evaluation
Speed:
Phrasing:
Expressiveness:
Meaning:

Figure 5.11: System audio-visual feedback interface

5.7 Performance of proposed LM on different categories of data

5.7.1 For substituted words

For this case where substituted word is a valid English word, the system performs well at finding the miscues as well as at decoding the substituted words with high precision.

5.7.2 For mispronounced words

For this case where the substituted word can be any meaningless word like “jumped” can be pronounced as “jump”+“aid”. The system predicts the most similar sounding word or sequence of words. Hindi words can also be added in the garbage model to have more options to find the similar words.

5.7.3 Effect of Noise

In case of noise, it generally is biased towards the words with \f and \s phones because of their noisy acoustic characteristics. This could be solved if we add noisy data carefully to our acoustic model training.

5.7.4 Murmurings or Sound-outs

In this case it generally outputs similar sounding small words. They will be treated as insertion only, so will not affect much in the miscue detection rate, since all the consecutive insertions are taken as only one miscue. Here, if we add the phones as words in the garbage model then we can get the corresponding phone sequence also. The probability of these phones needs to be tuned.

5.7.5 Repetition and long pauses

It works quite well for this case, long pauses are handled easily because of the silence loop in the model. It easily detects the repeated words and also decodes it properly. There can be half words, currently it is giving the similar sounding word, but can be improved further by adding phones in the garbage model.

For the improvement, we could change some hyper-parameter involved in the system like acoustic weight (probability assigned to acoustic model) or the garbage model weight. While tuning these parameters following was observed:

- The system performance is more affected with changing Acoustic weight rather than changing garbage model weight
- System performs better when garbage model weight is in the range 3-60%, and do not vary much in this range
- Its better to fix the garbage model weight to 20% (for all cases) and change the acoustic weight accordingly
- For fluent speakers (even if substituted valid words), low AM weight is preferable (1/80 to 1/60)
- For dis-fluent speakers (mispronunciation, repetitions, Murmurings or Sound-outs), a high value of acoustic weights would be optimal(1/25 to 1/10)

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we propose a task-specific Language Model for the children reading assessment task.

- The proposed model gives 7.26% WER and 3.95% PER. Also at only 5% FAR, we get around 74.03% miscue detection rate.
- In our proposed model, we do not require any task-specific or story specific annotated data as opposed to the case described in [1].
- Since our training data is very different from the evaluation data in terms of disfluency, this model is generalization to any other children reading data.
- As opposed to the uni-gram garbage model, where the words are constrained by the training data, here we can add any words of our choice in the garbage model, irrespective of the language. The garbage weight can be learned on a small development set.
- We have compared two N-gram model smoothing algorithms and found that Witten-Bell Discounting works better for our assessment task compared to modified Kneser-Ney algorithm. We have also given the reasons (insights) for the better performance of Witten-Bell algorithm in Section 5.3.1 and concluded that Witten-Bell discounting is more suitable for the assessment task in general.

Using the proposed model, we have built an assessment system which will take an audio with the corresponding canonical text and will detect the miscues such as substitution, insertion and deletion into a GUI(Section 5.6)

6.2 Future Work

- We could add the individual phones in the garbage model similar to the words, and the weights could be trained on the development set.
- Uni-gram of training data, zero-gram of extra unique words and uni-gram of phones could be added in parallel and the corresponding parameters could be learned on development set.
- Currently, the acoustic model used, has been trained on clean utterances; we could train it for the noisy campus data to make the overall system robust to noise.
- There are utterances where the words follow the partial utterance of same words, so we could add the dis-fluency path at phone level also.
- Currently, we manually build the pronunciation dictionary, the pronunciation model could be built to automate this process.

List of Publications

1. K. Sabu, K. Kumar, and P. Rao, “Improving the Noise Robustness of Prominence Detection for Children’s Oral Reading Assessment”, Proc. of NCC, Feb 2018, Hyderabad, India.

2. K. Sabu, K. Kumar, and P. Rao, “Automatic detection of expressiveness in oral reading ”, Show & Tell demonstration, Interspeech, Hyderabad, India, 2018.

3. P. Rao, M. Pandya, K. Sabu, K. Kumar, and N. Bondale, “A Study of Lexical and Prosodic Cues to Segmentation in a Hindi-English Code-switched Discourse ”, Interspeech, Hyderabad, India, 2018.

References

[1] P. Swarup H. Tulsiani and P. Rao. "Acoustic and language modeling for children's read speech Assessment". *Proceedings of National Conference on Communications, Chennai, India*, 2017.

[2] Yun-Cheng Ju Xiaolong Li, Li Deng and Alex Acero. "Automatic Children's Reading Tutor on Hand-Held Devices". *ISCA*, 2008.

[3] Yun-Cheng Ju Xiaolong Li, Li Deng and Alex Acero. "Efficient and Robust Language Modeling for An Automatic Children's Reading Tutor System". *ICASSP, Honolulu, Hawaii*, 2007.

[4] Edit graph link. `http://slideplayer.com/slide/12622591/`. last accessed 14/6/2018.

[5] ASER: The Annual Status of Education Report (rural). `http://img.asercentre.org/docs/Publications/ASER%20Reports/ASER_2012/fullaser2012report.pdf`. ASER Centre 2012, last accessed 14/6/2018.

[6] S. Dowhower. "Repeated reading revisited: Research into practice". *Reading Writing Quarterly, vol. 10, no. 4*, page 343–358, 1994.

[7] H. Tulsiani K. Sabu, P. Swarup and P. Rao. "Automatic assessment of children's L2 reading for accuracy and fluency". *Proceedings of Speech and Language Technology in Education, Stockholm, Sweden*, 2017.

[8] A. Pasad. "Voice Activity Detection for Children's Read Speech Assessment in Noisy Conditions ". *Dual Degree dissertation, Department of Electrical Engineering, IIT Bombay*, 2017.

[9] P. Swarup. "Acoustic model training and adaptation for children's read speech recognition". *M.Tech dissertation, Department of Electrical Engineering, IIT Bombay*, 2017.

[10] J. Tepperman P. Black and S. Narayanan. "Automatic prediction of children's reading ability for high-level literacy assessment". *IEEE Transactions on Audio, Speech, and Language Processing, vol. 19, no. 4, pp. 1015–1028*, 2011.

[11] M. Cettolo M. Federico, N. Bertoldi. "IRSTLM: an Open Source Toolkit for Handling Large Scale Language Models". *Proceedings of Interspeech, Brisbane, Australia*, 2008.

[12] Openfst tool. `http://www.openfst.org/twiki/bin/view/FST/WebHome`. last accessed 14/6/2018.

[13] Daniel Jurafsky James H. Martin. "Chapter-4 Language Modeling with Ngrams, Speech and Language Processing". *August*, 2017.

[14] S. F. Chen and J. Goodman. "An empirical study of smoothing techniques for language modeling". *Proceedings of the 34th annual meeting on Association for Computational Linguistics. Association for Computational Linguistics, 1996, pp. 310–318*, 1998.

[15] Ismail. "Comparison of Modified Kneser-Ney and Witten-Bell Smoothing Techniques in Statistical Language Model of Bahasa Indonesia". *2nd International Conference on Information and Communication Technology (ICoICT), May*, 2014.

[16] Script to converts LM into FST. `https://github.com/hartmannw/BlogCode/blob/master/2014/FSMConversion/convert_lm_to_fst.py`. last accessed 14/6/2018.

[17] G. Boulianne L. Burget O. Glembek N. Goel M. Hannemann P. Motlicek Y. Qian P. Schwarz J. Silovsky G. Stemmer D. Povey, A. Ghoshal and K. Vesely. "The Kaldi speech recognition toolkit". *Proc. of IEEE Workshop on Automatic Speech Recognition and Understanding, Hawaii, USA*, 2011.

[18] Google Speech API. `https://cloud.google.com/speech-to-text/docs/reference/libraries?hl=es#client-libraries-install-python`. last accessed 14/6/2018.

[19] Google cloud speech-to-text api language support. `https://cloud.google.com/speech-to-text/docs/languages`. last accessed 14/6/2018.

[20] Tkinter- python interface. https://docs.python.org/2/library/tkinter.html. last accessed 14/6/2018.