Rhyming characteristics of Poetry: Indian Languages

Major Project – 2 Report

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science and Engineering

under the guidance of

Dr. Kamlesh Dutta

by

Aman Garg (185036), Rohit Kaushal (185037), Prashant Kumar (185045),

Atul Thakur (185050), Nikhil Sharma (185068), Azhan Ali (185096)

Department of Computer Science and Engineering
National Institute of Technology, Hamirpur
May 2022

Contents

ABSTRACT 1
Introduction 1
What is Rhyme? 1
Types of Rhyme 2
Purpose of Rhyme Detection 2
Related Work 2
Dataset 2
Characteristics of Devanagari Language 3
Data Collection 3
Dataset Analysis 3
Data Pre-processing 4
Rhyming Pairs Dataset 4
Text Similarity 4
Cosine Similarity 5
Euclidian Distance 5
Evaluation Measures 5
Character Based Encoding Schemes 6
Phonemes 6
TF-IDF 6
Deep Learning Techniques 7
LSTM's 7
Siamese Recurrent Networks 8
Implementation of Rhyme Detection Techniques 9
Encoding Words into Vectors 9
Cosine Similarity 9
Using Phonemes Representation 9
Using TF-IDF Representation 10
Euclidian Distance 11
Deep Learning Based 12
Siamese Recurrent Networks 12
Results 13
Classical Text Similarity based algorithms 13

Deep Learning based Algorithm 13
Web App Integration 15
Conclusions 18
Accomplishments 19
Future Work 19
Acknowledgements 19
Individual Contributions 20
References 20

ABSTRACT

Rhyme detection is a very important part of several text processing pipelines. A little amount of research has been done in this field. This paper proposes a procedure for automatic detection of rhyming pairs in poems for poetry analysis task, consisting of 10 Devanagari based languages of India i.e., Angika, Awadhi, Braj, Bhojpuri, Chhattisgarhi, Garhwali, Haryanvi, Hindi, Magahi, and Maithili. We collated a corpus of poems of varying length. We studied various lexical, syntactic, and semantic features of the poems. We created a corpora of rhyming pairs in Hindi language. Finally, various classification machine learning and deep learning techniques are applied and evaluated for the purpose of rhyme detection.

Introduction

Rhyme has been used as a style device for poetry, music and hip-hop. The previous research in the filed of Rhyme detection has been based on small, handcrafted datasets. Two words are said to be rhyming if they produce the same sound. A reliable system for the detection of rhyme between two words would allow large scale analyses, opening several directions for research. There has been no previous research done in rhyme detection for Indian languages. The task of rhyme detection becomes fairly easy when we are given the pronunciations and a definition of rhyme. However, for domain specific or historical data, obtaining precise pronunciation information is a challenge (Katz, 2015). Also, a narrow definition of perfect rhyme disregards frequently used and accepted deviations, as in imperfect rhyme (Primus, 2002) (Berg, 1990) or the related sonic devices assonance, consonance and alliteration. For the purpose of rhyme detection, information about phonological properties of a language can be used.

Previous research on the detection of rhyme is scarce. Reddy and Knight (2011) employed Expectation Maximization (EM) to predict (generate) the most probable scheme (e.g. 'abba') of a stanza. We used classical machine learning algorithms and a supervised deep learning based algorithm for studying the characteristics of rhyming words in Hindi. An accurate similarity measure between words would benefit an EM. We employed Phonemes and tf-idf features to study the relationship between two rhyming words using classical distance measures. Siamese Recurrent Networks can be of great advantage for the purpose of rhyme detection as they learn similarity between two inputs training two different neural network models.

The goal of our work is to try out various supervised machine learning and deep learning techniques to measure rhyming similarity between two pairs of words in Hindi language. We have created a dataset of poetry in Hindi and from those poems, we created a dataset of rhyming pairs. Before building the machine learning system, we computed the lexical, syntactic, and semantic analysis of the poems, to get the idea of the dataset. After that, we worked supervised machine learning algorithms for finding out the similarity between the rhyming pairs.

What is Rhyme?

According to Wikipedia, “A **rhyme** is a repetition of similar syllables (usually, exactly the same number of syllables) in the final stressed syllables and any following syllables of two or more words. Most often, this kind of perfect rhyming is consciously used for a musical or aesthetic effect in the final position of lines within poems or songs.” In simple terms, rhyme means the phonological similarity between two words in a specific language.

Types of Rhyme

Rhymes are of two types:

Internal Rhyme: The words that rhyme and are in the same sentence. They are also referred to as middle rhymes.

Full Rhyme: The words that rhyme and are at the end of two different lines. They are the rhymes in which all vowels and consonants after a particular stressed vowel are the same.

Purpose of Rhyme Detection

Rhyme is mainly used in poems, poetry, songs, hip-hop etc. for the purpose of adding some style. Use of rhyme makes a poem or a song more attractive and adds a rhythm to them. Only some sparse research has been done in the study of rhyming characteristics. The rhyme analysis can provide various insights about the phonological characteristics of a language, and it can prove as a metric for the rhythmic analysis of a language.

Related Work

There has been some decent amount of research done in rhyme detection in poetry and hip-hop for English, French and German. There has also been some research done for poems in Hindi language too. The goal of that research was oriented towards lexical analysis and language identification applications.

(Reddy and Knight, 2011) employed Expectation Maximization (EM) to predict (generate) the most probable scheme (e.g. ’abba’) of a stanza.

(Priyankit et al., 2012) studied the lexical characteristics of poetry in Devanagari based languages and developed a supervised machine learning algorithm for language identification of Devanagari poems.

(McCurdy et al., 2015) examined the phonological similarity between two rhyming words based on the analysis of sonic patterns.

(List et al., 2017) used the information about phonological similarity to reconstruct the historical pronunciation.

(T. Haider and Jonas Kuhn, 2018) used Siamese Recurrent Networks to develop a supervised deep learning algorithm for detection of rhyming pairs in English, French and German.

Dataset

Characteristics of Devanagari Language

Devanagari is a well-developed language originated from *Sanskrit. Devanagari* is an abugida script used to write several Indian languages, including Sanskrit, Hindi, Marathi, Sindhi, Bihari, Bhili, Marwari, Konkani, Bhojpuri, Pahari (Garhwali and Kumaoni), Santhali Language; languages from Nepal like Nepali, Nepal Bhasa, Tharu and sometimes Kashmiri and Romani. It is written and read from left to right. The alphabets in Devanagari can be classified into Vowels and Consonants. The vowels can again be divided into two classes i.e., Dependent Vowels (*matras*) and Independent Vowels.

The classification of various alphabets of Devanagari are:

Consonants	'क','ख','ग','घ','ङ','च','छ','ज','झ','ञ', 'ट','ठ','ड','ढ','ण', 'त','थ','द','ध','न', 'प','फ','ब','भ','म', 'य','र','ल','व','श', 'ष','स','ह','क्ष','त्र','श्र','ज्ञ'
Dependent Vowels	'ा','ि','ी','ु','ू',' ृ','ॄ','ॅ','े','ै','ॉ','ॊ','ो','ौ'
Independent Vowels	'अ','आ','इ','ई','उ','ऊ','ऋ','ए','ऐ','ओ','औ','अं','अः'

Data Collection

In order to collect the dataset of poetry in Devanagari based languages, we found that there is a website *https://www.kavitakosh.org* that contains thousands of poems in Devanagari. We wrote a script in Python language in order to scrap the poems from the website. Our script uses *Beautiful Soup*, a Python library that can parse HTML documents. We used this library to parse 4589 poems from *Kavitokosh* and download them into a CSV file along with some relevant information like, poet name and language. After that, we pre-processed the scrapped dataset by ignoring the rows that do not contain any poem i.e., the poem cell is NaN. We also tokenized the poems by removing punctuation.

Dataset Analysis

The dataset we scrapped contains 2979 poems in 10 Devanagari based languages of India i.e., *Angika, Awadhi, Braj, Bhojpuri, Chhattisgarhi, Garhwali, Haryanvi, Hindi, Magahi, and Maithili.* The metadata about the dataset is as follows:

Average number of words/ poems	501.9108040201005
Most Common Words	है: 8369 में: 8106 के: 8103 की: 7102 से: 5795 का: 4323 और: 4113 नहीं: 3666 को: 3544 पर: 3475

Most Common Poets	प्रमोद कुमार शर्मा: 74 अज्ञेय: 29 केदारनाथ अग्रवाल: 29 रवीन्द्रनाथ ठाकुर: 27 कविता भट्ट: 25 कन्हैया लाल सेठिया: 23 हरिवंशराय बच्चन: 23 सांवर दइया: 22 पुष्पिता: 21 सुमन पोखरेल: 21
Average number of sentences/ poems	25.779120510238336
Average number of stanzas/ poems	4.226250419603894
Average number of pronouns/ poems	2.97583081570997

Data Pre-processing

Data pre-processing simply means, cleaning the data. Since every real-world data is redundant, hence data pre-processing is very important for any machine learning model to give better results. In our case, we are pre-processing the poems by tokenization and by removal of punctuations. We removed the rows that do not have a poem as the web scrapping script that we wrote contained such rows. In order to process the CSV (Comma Separated Values) file, we used *Pandas*, a Python library for processing the data frames.

Language	Poem Name	Poet	Poem	words	sentences	stanzas	vowels	consonants	pronouns
hindi	सुभाष मुखोपाध्याय	अग्निकोण / रणजीत साहा / सुभाष मुखोपाध्याय	अग्निकोण के अंचलों में हलचल मचाती ज़ोरदार आँधी...	494	86	11	659	1047	1
hindi	सुभाष मुखोपाध्याय	अग्निगर्भ (कविता) / रणजीत साहा / सुभाष मुखोपाध...	अभी धड़ी भर पहले जो आदमी गले में फनदा डाले\r\n...	149	20	6	168	278	4
hindi	सुरेन्द्र डी सोनी	अग्निगर्भा ! / सुरेन्द्र डी सोनी	अग्निगर्भा ! \r\nधारण करो मुझे –\r\nइस जन्म मे...	22	6	1	22	36	2
hindi	पुष्पिता	अग्निगर्भी शक्ति / पुष्पिता	धूप में \r\nबढ़ाती हूँ \r\nअपनी आत्मा की अग्नि...	76	17	3	86	147	0
hindi	रामेश्वर खंडेलवाल 'तरुण'	अग्निचक्र: अग्नि-चित्र / तरुण	बालपने में\r\nअपने मटियाये-धूमधुमैले-से गाँव क...	118	19	5	173	253	5

Figure 1: Poetry dataset along with metadata

Rhyming Pairs Dataset

From the poems that were scrapped, another dataset of rhyming pairs was manually created. By rhyming pairs, we mean a pair of two words in Hindi that are rhyming. We manually found out the rhyming pairs in the poems and created a dataset out of that. There are 544 Rhyming Pairs. For training a supervised machine learning model, we randomly shuffled the rhyming pairs to create a dataset of non-rhyming pairs. There are 249 non-rhyming pairs in the dataset.

Text Similarity

To calculate the similarity between two words, there are various techniques that takes two vectors and find out the similarity between them. These techniques calculate the commonality existing between the two texts. For checking whether two words are rhyming, we can employ the text similarity measures and can obtain a threshold. The purpose of the threshold is to determine two words are rhyming only if the similarity between them is greater than the threshold. There are many text similarity measures that are listed as followed:

Cosine Similarity

This similarity measure calculates the cosine between two vectors by computing the dot product of the vectors. It measures the similarity between two vectors of an inner product space. It is often used to measure similarity between documents in text analysis.

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Figure 2: Cosine Similarity between two vectors A & B

Euclidian Distance

Euclidian distance is computed by calculating the straight-line distance between two points in Euclidian space. Text similarity between two documents can be calculated via Euclidian distance by representing the two documents as N-dimensional vectors. This distance gives an idea about how far or near the two documents are and can be used as a measure of similarity between the two documents. For two objects that are not points, Euclidian distance between them is calculated by smallest distance among pairs of points from the two objects.

$$\textbf{Euclidean Distance} = |X - Y| = \sqrt{\sum_{i=1}^{i=n} (x_i - y_i)^2}$$

Figure 3: Euclidian distance between two vectors X & Y

Evaluation Measures

Since we are training a binary classification machine learning model, F1 score, precision, recall, and accuracy can be used for determining the accuracy of the model. These metrics are used to fine tune the model. For computing the confusion matrix, following parameters are used:

True Positives (TP): Accurately classified Positive samples

True Negatives (TN): Accurately classified Negative samples

False Negatives (FN): Inaccurately classified Positive samples

False Positives (FP): Inaccurately classified Negative samples

Based on these terms the following evaluation measure are quantified as follows:

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

Accuracy = (TP + TN) / (TP + TN + FP + FN)

F1 score = 2* Precision*Recall / (Precision + Recall)

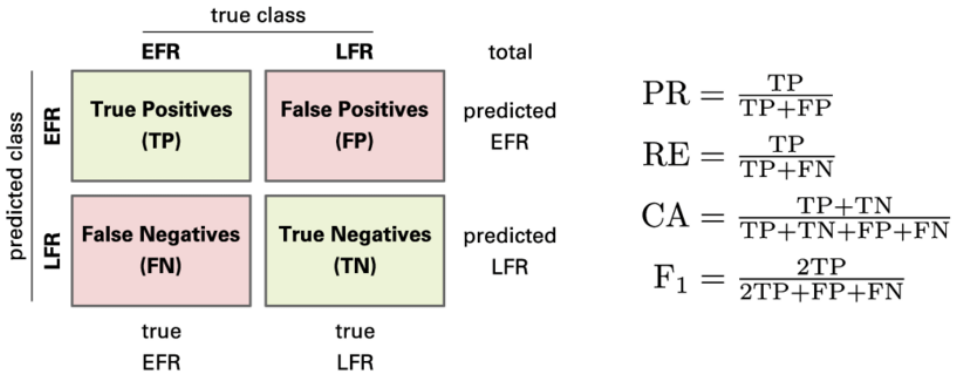

Figure 4: Confusion Matrix, Precision, Recall, Accuracy and F1 Score

Character Based Encoding Schemes

In order to train a supervised machine learning algorithm, we need to encode the words making use of some encoding scheme. Using that, we can represent the word a sequence and finally, our model can be trained on that sequence. We can find the similarity between two words based on that sequence. In order to keep the inputs from 0 to 1, we can use the concept of One Hot Encoding. The One Hot Encoded vectors are then passed to the machine learning model. We used the following two schemes for representing the words:

Phonemes

Phoneme is a unit of sound that differentiates one word from another in a specific language. There has been work done which uses phonemes for NLP tasks (Chourasia et al, 2007). Poets mostly care about the phonemes and look how they can manipulate the phonemes into musical arrangements. Devanagari has same phonemes as its letters. The order of letters in Devanagari takes care of phonetic principals. Vowels are of two types in Devanagari: dependent and independent. The dependent vowels are called *mantras.* They are used to indicate that there is a consonant that is attached to the vowel. The independent vowels are used when the vowel occurs alone, at the beginning of a word, or after another vowel. In our models, we used Phonemes (vowels + consonants), Independent Vowels, Only Consonants, Matras (Dependent Vowels) and Phonemes + Matras.

TF-IDF

TF-IDF stands for Term Frequency – Inverse Document Frequency. TF-IDF vectorization of a corpus assigns a numerical value to each word. This numerical value is proportional to the word's frequency in the document and inversely proportional to the number of documents in which it occurs. Thus, it basically states the importance of a word in a document. In our models, we used Unigram TF-IDF vectorization for representing the words and calculated the similarity based on that.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Figure 5: TF-IDF representation of a word, $w_{i,j}$

Deep Learning Techniques

LSTM's

LSTM stands for Long Short-Term Memory Sequences. When we deal with short-term dependencies then Recurrent Neural Networks work just fine. But RNN's are not able to understand the context behind the sentence. When making prediction in present, something that was there in long past can't be taken into account by the RNN's. This is due to the problem of Vanishing Gradients. RNN are only able to remember things for just small durations of time, i.e., if we need the information after a small time, it may be reproducible, but once a lot of words are fed in, this information gets lost somewhere. We can resolve this issue of Vanishing Gradients by modifying the structure of RNN's into an LSTM – the Long Short-Term Memory Network. LSTMs are way better than RNN's and Convolutional Feed Forward Networks. They have a property of remembering the things from long past.

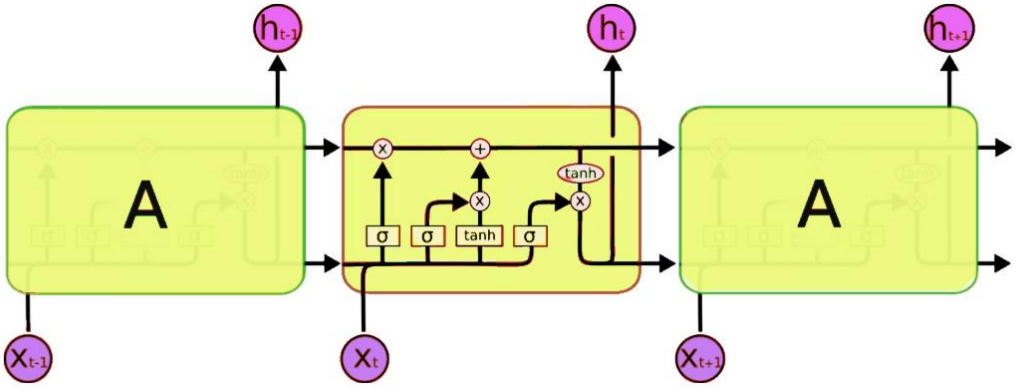

Figure 5: LSTM CELL

An LSTM network is typically comprised of different memory blocks called cells. Two states are transferred to next cell, they are: the cell state and the hidden state. The memory blocks are remember things and manipulations to the memory blocks is done through three major mechanisms, called **gates.**

Mathematical formulas behind LSTM:

$$
\begin{aligned}
f_t &= \sigma_g(W_f * x_t + U_f * h_{t-1} + V_f \circ c_{t-1} + b_f) \\
i_t &= \sigma_g(W_i * x_t + U_i * h_{t-1} + V_i \circ c_{t-1} + b_i) \\
o_t &= \sigma_g(W_o * x_t + U_o * h_{t-1} + V_o \circ c_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c * x_t + U_c * h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Figure 6: Mathematical formulas for LSTM

LSTMs perform exceptionally well in sequence and time series related problems. But, the only disadvantage about them is that they are difficult to train. For training a simple model, lots of resources and time are taken.

Siamese Recurrent Networks

Siamese Recurrent Networks (SRN) can be used to find the similarity between texts, both on the level of characters and words. (Neculoiu et al., 2016) used SRN's for job title normalization. (Mueller and Thyagarajan, 2016) used SRN's for sentence similarity. (Das et al., 2016) used SRN's for Quora question pair retrieval. An SRN contains two identical RNN's that learn a vector representation. The input pairs are represented as N-dimensional vectors. These Sub-RNN's each receive a rhyme word as character embedding vector and encode it through several layers of bidirectional Long Short Term Memory (biLSTM) Networks. The activations at each timestep of the final biLSTM layer are averaged to produce a fixed-dimensional output. This output is then sent to a single dense fully connected feed forward layer. The dense layers are finally connected to an energy function. The energy function calculates the similarity between the embeddings of two words, Word 1 and Word 2. We can use similarity measures described earlier as energy function. Some possible energy functions are cosine similarity, Euclidian distance etc.

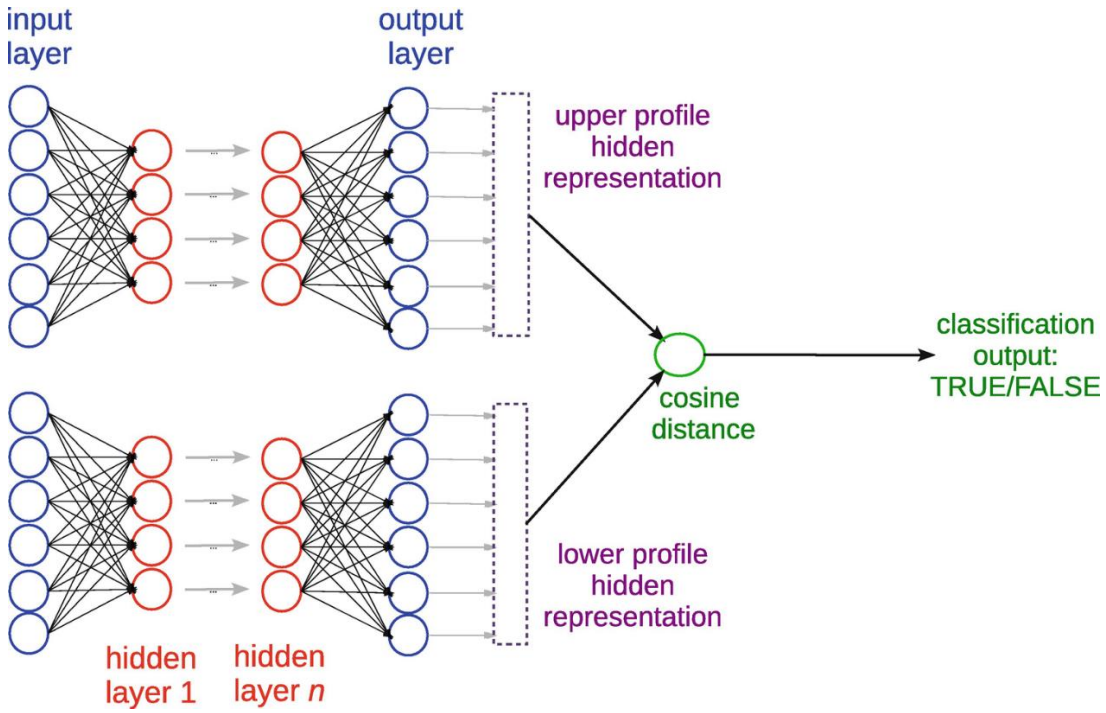

Figure 7: Structure of Siamese Network

Implementation of Rhyme Detection Techniques

We used Python programming language to implement the classifiers for classifying whether two words are rhyming are not. We then integrated the classifier onto a Web app using JavaScript, React and Flask framework for finding out all the rhyming pairs in a poem written in Hindi.

For training purposes, we used Google Collaboratory. The various Python libraries that we used are: TensorFlow, Keras, Matplotlib, NumPy, Pandas & Beautiful Soup.

Encoding Words into Vectors

```
ivowelList=['अ','आ','इ','ई','उ','ऊ','ऋ','ए','ऐ','ओ','औ','अं','अः']
dvowelList=['ा','ि','ी','ु','ू','ृ','ॅ','े','ै','ॉ','ो','ौ']
consonantList=['क','ख','ग','घ','ङ',
               'च','छ','ज','झ','ञ',
               'ट','ठ','ड','ढ','ण',
               'त','थ','द','ध','न',
               'प','फ','ब','भ','म',
               'य','र','ल','व','श',
               'ष','स','ह','क्ष','त्र','श्र','ज्ञ']
allCharactersList = ['अ','आ','इ','ई','उ','ऊ','ऋ','ए','ऐ','ओ','औ','अं','अः','ा','ि','ी','ु','ू','ृ','ॅ','े','ै','ॉ','ो',
               'च','छ','ज','झ','ञ',
               'ट','ठ','ड','ढ','ण',
               'त','थ','द','ध','न',
               'प','फ','ब','भ','म',
               'य','र','ल','व','श',
               'ष','स','ह','क्ष','त्र','श्र','ज्ञ']
characterToIndex = {}
for i in range(len(allCharactersList)):
    characterToIndex[allCharactersList[i]] = i
```

```
def getWordVector(word):
    vector = [0 for _ in range(65)]
    for character in word:
        if(character in characterToIndex.keys()):
            vector[characterToIndex[character]] += 1
    return vector
```

Figure 8: Code for encoding words

Cosine Similarity

Using Phonemes Representation

Cosine Similarity

```
def getWordVector(word):
    vector = [0 for _ in range(65)]
    for character in word:
        if(character in characterToIndex.keys()):
            vector[characterToIndex[character]] += 1
    return vector
```

```
def cosineSimilarity(word1, word2):
    A = getWordVector(word1)
    B = getWordVector(word2)
    return np.dot(A,B)/((norm(A)*norm(B))+0.0001)
```

```
cosine_similarity = []
for index, row in rhyming_pairs.iterrows():
    cosine_similarity.append(cosineSimilarity(row['word1'], row['word2']))

cosine_similarity_non_rhyming = []
for index, row in non_rhyming_pairs.iterrows():
    cosine_similarity_non_rhyming.append(cosineSimilarity(row['word1'], row['word2']))
```

```
mean_cosine_similarity = np.mean(cosine_similarity)
mean_cosine_similarity
```

```
0.6017182600548051
```

```
mean_cosine_similarity_non_rhyming = np.mean(cosine_similarity_non_rhyming)
mean_cosine_similarity_non_rhyming
```

```
0.19885919525110732
```

Figure 9: Code for calculating cosine similarity between two pairs

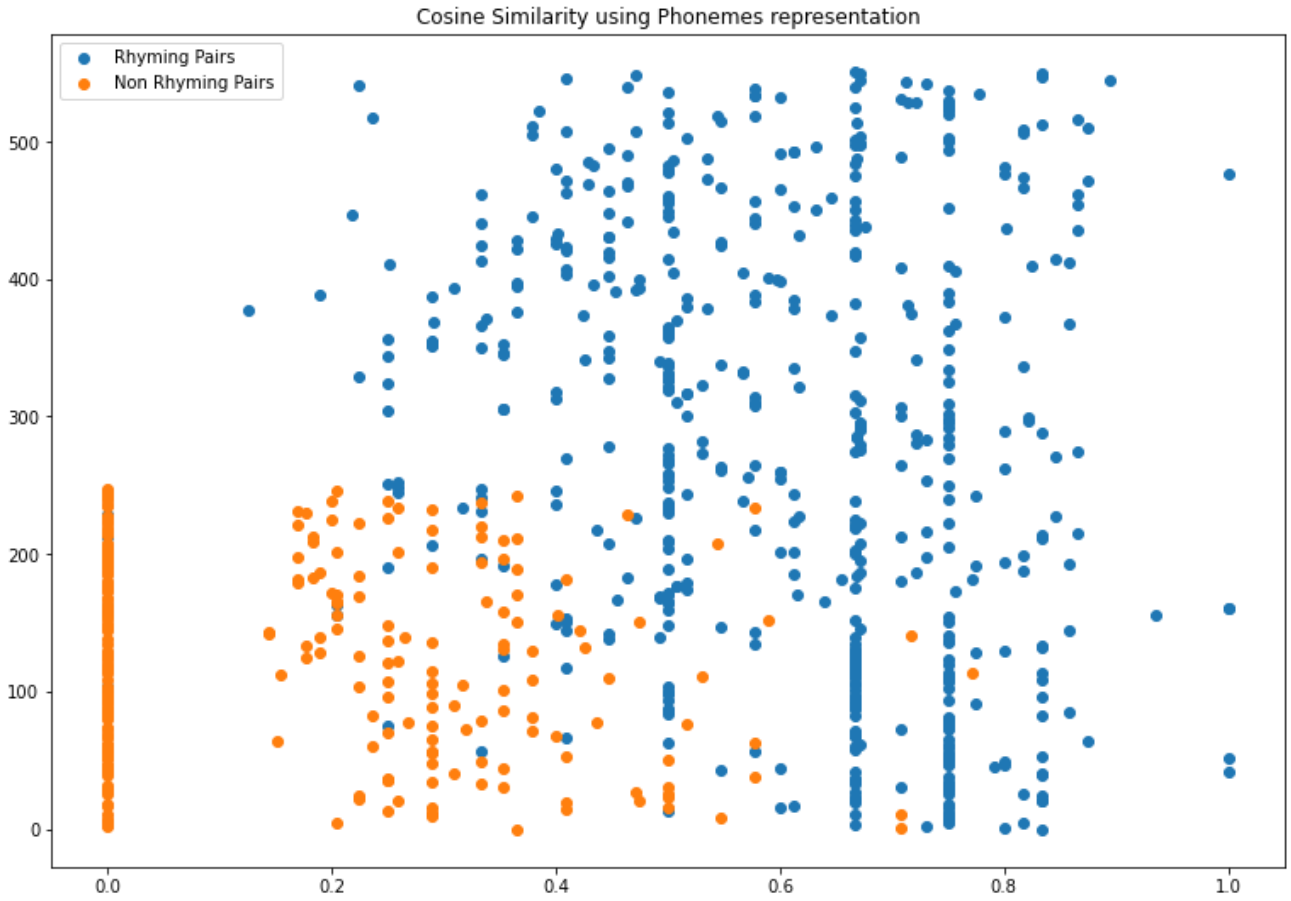

Figure 9: Plot of Cosine Similarities between Rhyming and Non-Rhyming pairs

Using TF-IDF Representation

```
import math
# Calculating IDF
idf = [0 for _ in range(65)]
term_frequency = [0 for _ in range(65)]
N = len(rhyming_pairs['word1']) - 1
for index, row in rhyming_pairs.iterrows():
    for i in range(len(allCharactersList)):
        if(allCharactersList[i] in row['word1']):
            term_frequency[i] += 1
        if(allCharactersList[i] in row['word2']):
            term_frequency[i] += 1
for i in range(len(allCharactersList)):
    idf[i] = math.log(N/(term_frequency[i]+1))
```

```
def getWordVector(word):
    vector = [0 for _ in range(65)]
    for character in word:
        if(character in characterToIndex.keys()):
            vector[characterToIndex[character]] += 1
    for i in range(len(allCharactersList)):
        vector[i] *= idf[i]
    return vector
```

```
cosine_similarity = []
for index, row in rhyming_pairs.iterrows():
    cosine_similarity.append(cosineSimilarity(row['word1'], row['word2']))

cosine_similarity_non_rhyming = []
for index, row in non_rhyming_pairs.iterrows():
    cosine_similarity_non_rhyming.append(cosineSimilarity(row['word1'], row['word2']))
```

Figure 10: Calculation of cosine similarity using TF-IDF representation

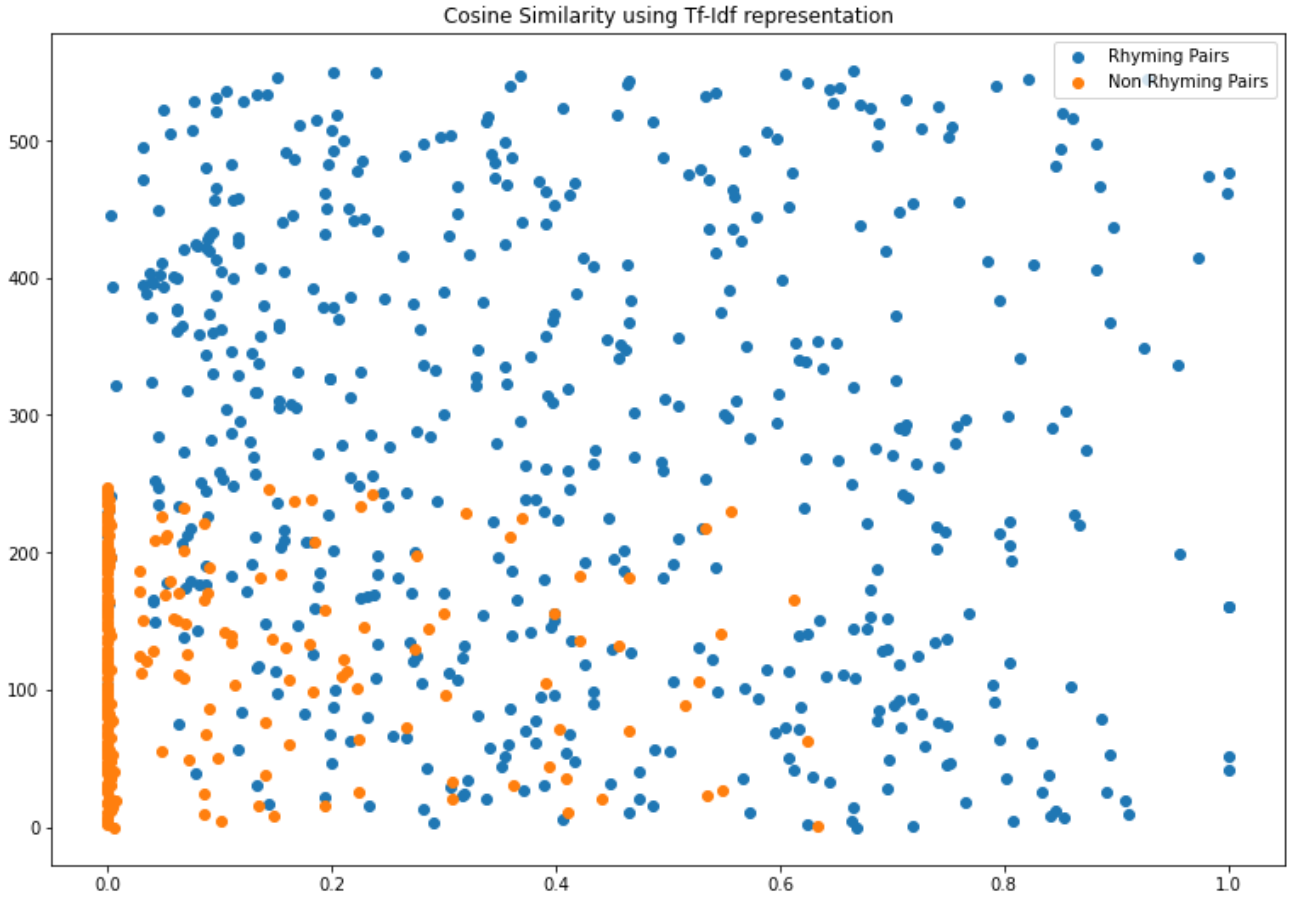

Figure 11: Plot of Cosine Similarities between Rhyming and Non-Rhyming pairs using TF-IDF

Euclidian Distance

```
def eucledianDistance(word1, word2):
    A = np.array(getWordVector(word1))
    B = np.array(getWordVector(word2))
    return np.linalg.norm(A - B)
```

```
eucledian_distance = []
for index, row in rhyming_pairs.iterrows():
    eucledian_distance.append(eucledianDistance(row['word1'], row['word2']))

eucledian_distance_non_rhyming = []
for index, row in non_rhyming_pairs.iterrows():
    eucledian_distance_non_rhyming.append(eucledianDistance(row['word1'], row['word2']))
```

```
mean_eucledian_distance = np.mean(eucledian_distance)
mean_eucledian_distance
```

```
1.8313671275158279
```

```
mean_eucledian_distance_non_rhyming = np.mean(eucledian_distance_non_rhyming)
mean_eucledian_distance_non_rhyming
```

```
2.6302945595690392
```

Figure 12: Computation of Euclidian Distance between all rhyming and non-rhyming pairs

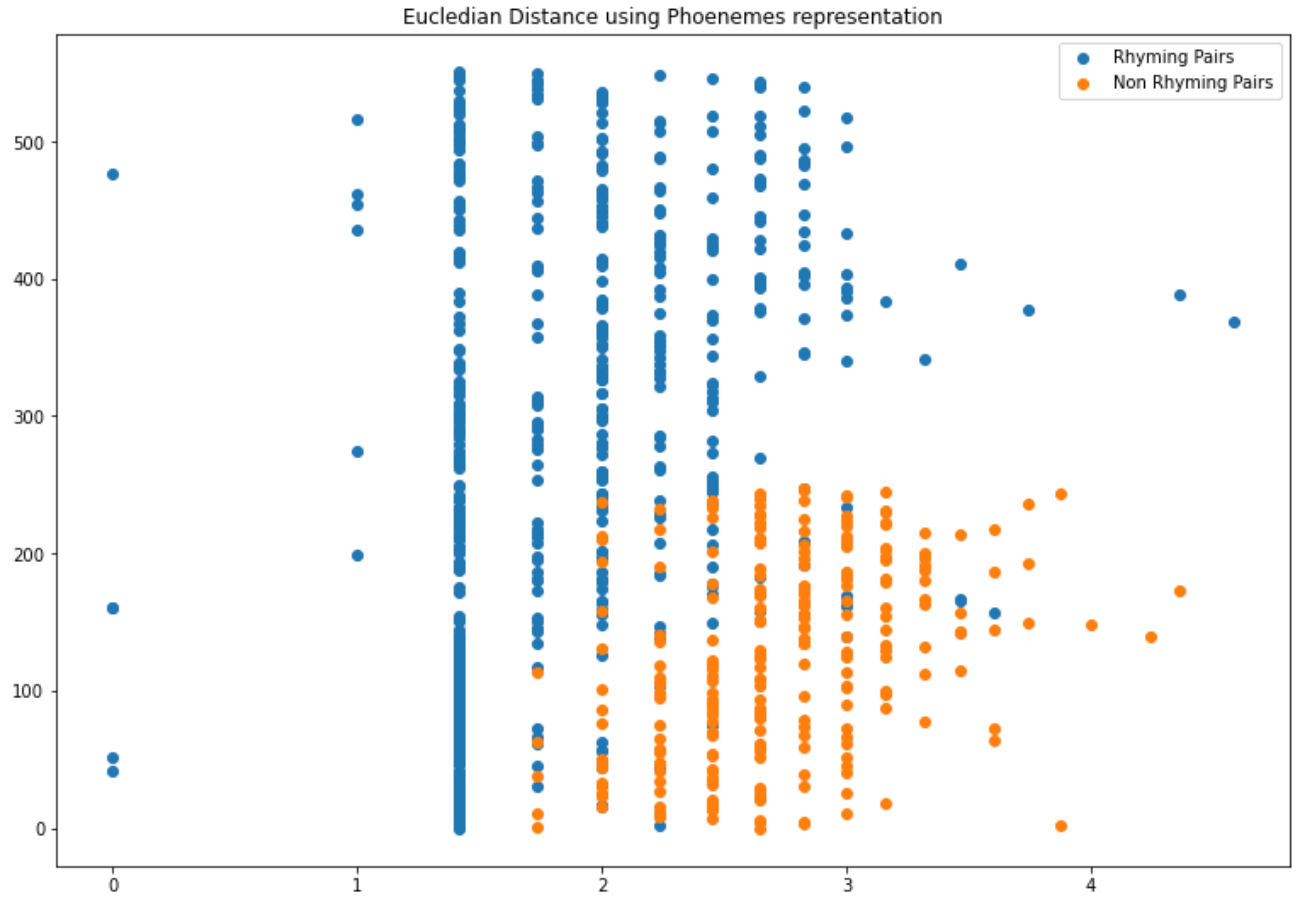

Figure 13: Plot for Euclidian Distance between Rhyming & Non-Rhyming pairs

Deep Learning Based

Siamese Recurrent Networks

For training the SRN, we encoded our input rhyming and non-rhyming pairs into a NumPy array of dimensions *(max_len, char_dims),* where *max_len* is the length of longest word in corpus and *char_dims* is the number of characters (both consonants and vowels) in the language.

For our case, *char_dims* were 66 and *max_len* was 14.

We used accuracy, precision and recall as the metrics to compare the accuracy of various models and for fine tuning the performance of our model.

The SRN we created, uses two 50 layers bidirectional LSTM's one for each word. Then the outputs of two LSTM's are combined and are Batch Normalized. After that the result is passed into a dropout layer. This dropout layer is connected to a Dense layer with 50 perceptron having *relu* activation function. The output of this layer is Batch Normalized and a dropout layer follows after that. Finally there is a Dense layer having sigmoid activation function.

The architecture of our SRN model is:

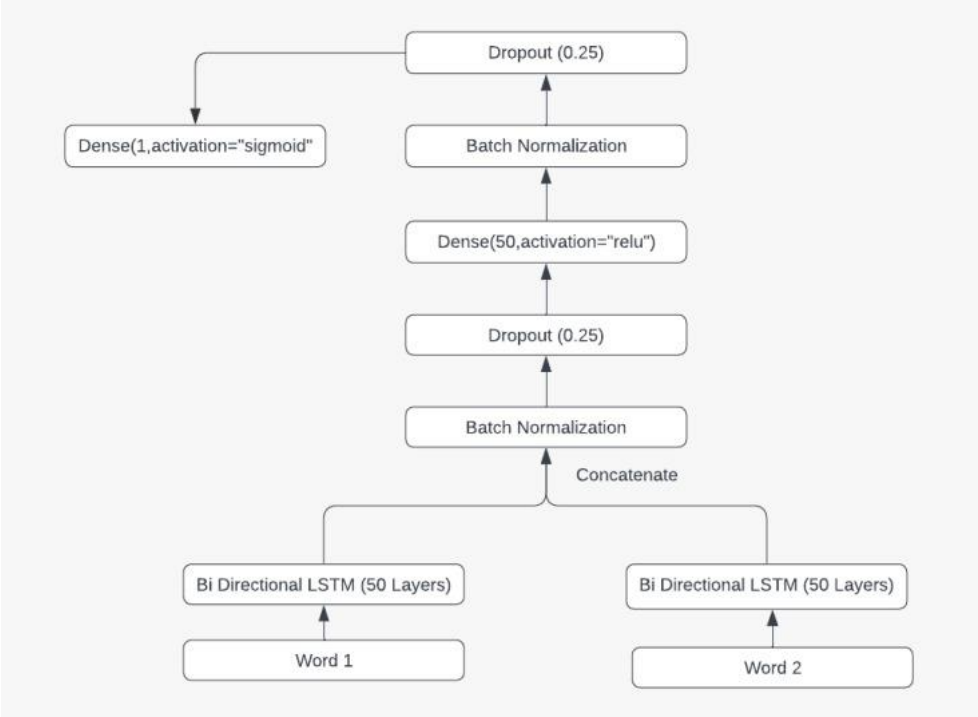

Figure 14: Architecture for SRN

Results

Classical Text Similarity based algorithms

Model	Accuracy	Recall	Precision	F1-Score
Cosine Similarity (using Phonemes)	69.038%	94.061%	88.788%	91.34%
Cosine Similarity (using TF-IDF)	74.984%	91.1917%	63.6528%	73.79%
Euclidian Distance	69.0387%	64.1618%	80.289%	71.32%

Deep Learning based Algorithm

We used a train-test split of 0.1 i.e., 9:1.

We used call back function of early stopping with a patience of 5. This help to stop training as soon as the accuracy of model doesn't increase anymore. The best accuracy was seen at epoch no. 27. The best dropout rate was found to be 0.25.

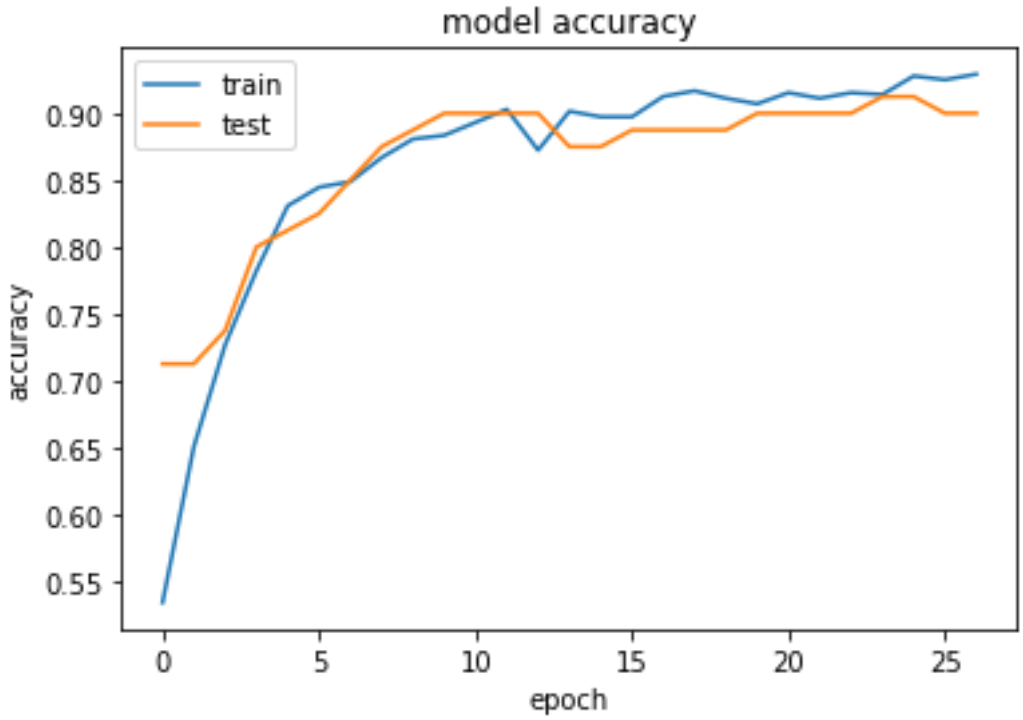

Figure 15: Plot of Accuracy/Epoch

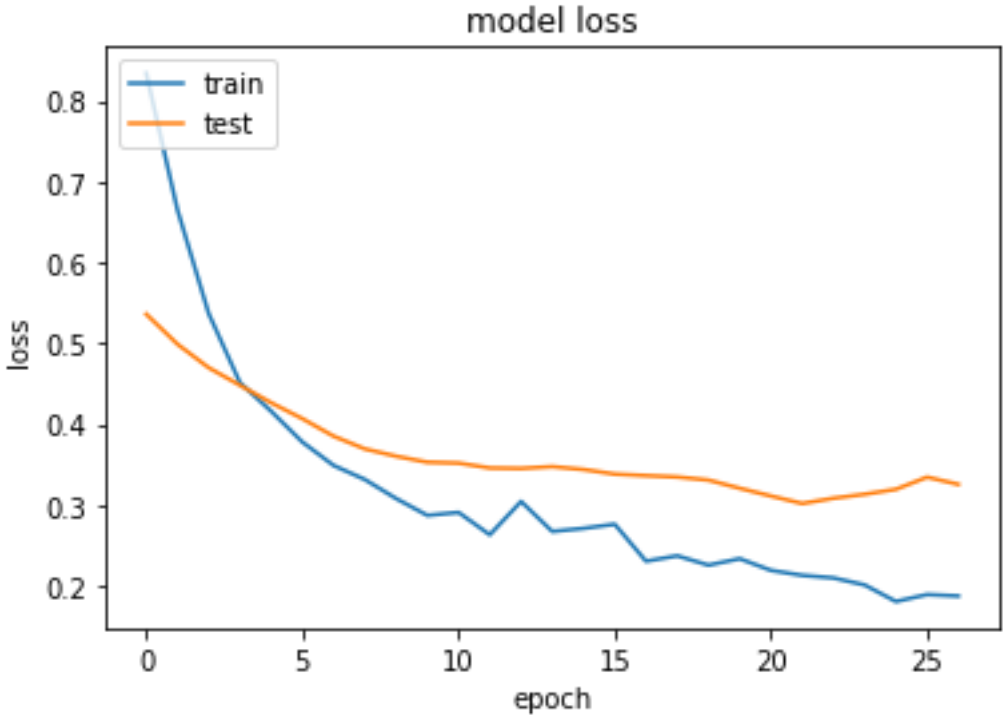

Figure 16: Plot of Model Loss/Epoch

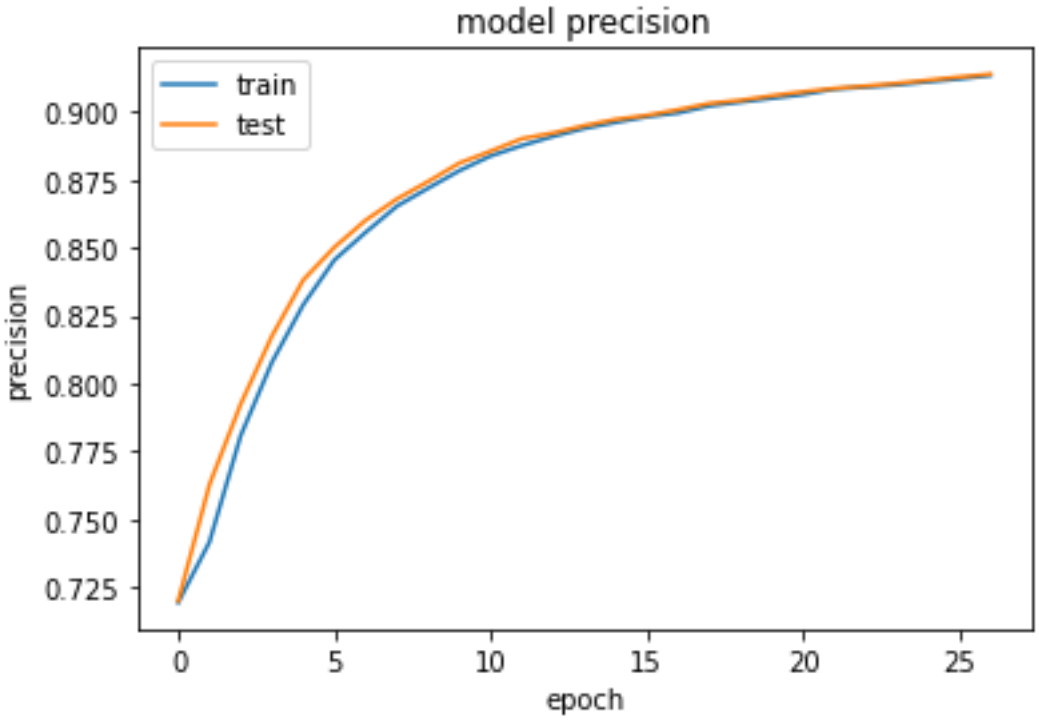

Figure 17: Plot of Precision/Epoch

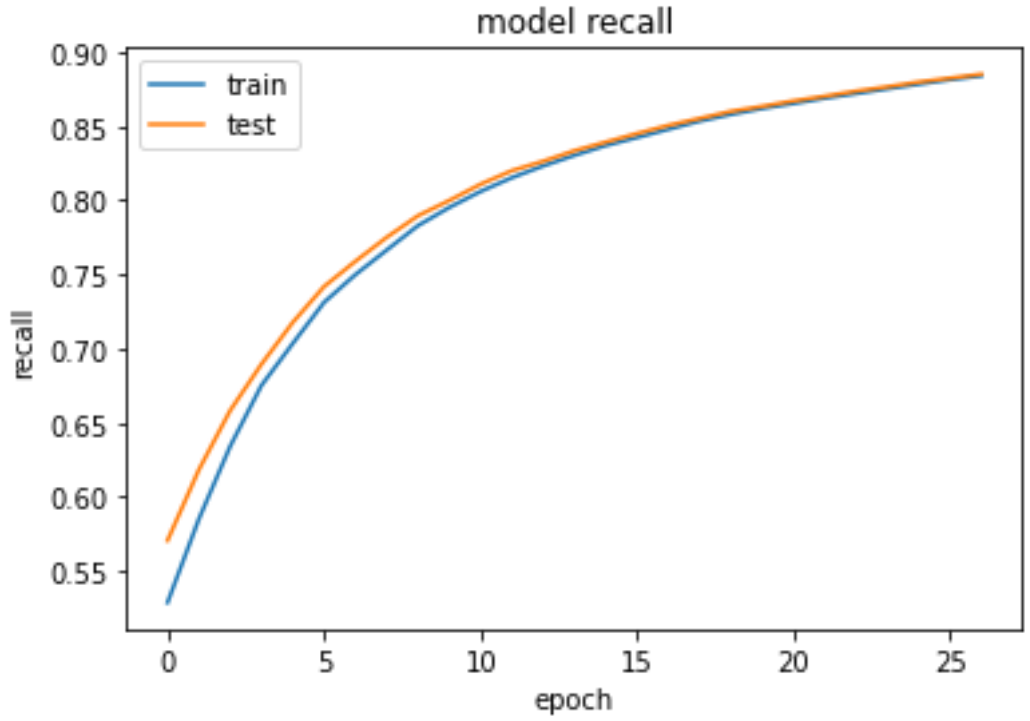

Figure 18: Plot of Recall/Epoch

Web App Integration

We created a web app that takes a poem as an input and detects all the rhyming pairs in that with the help of model trained using SRN. Web app uses HTML, CSS, JavaScript in frontend and Python - Flask framework to check whether a pair of words are rhyming or not.

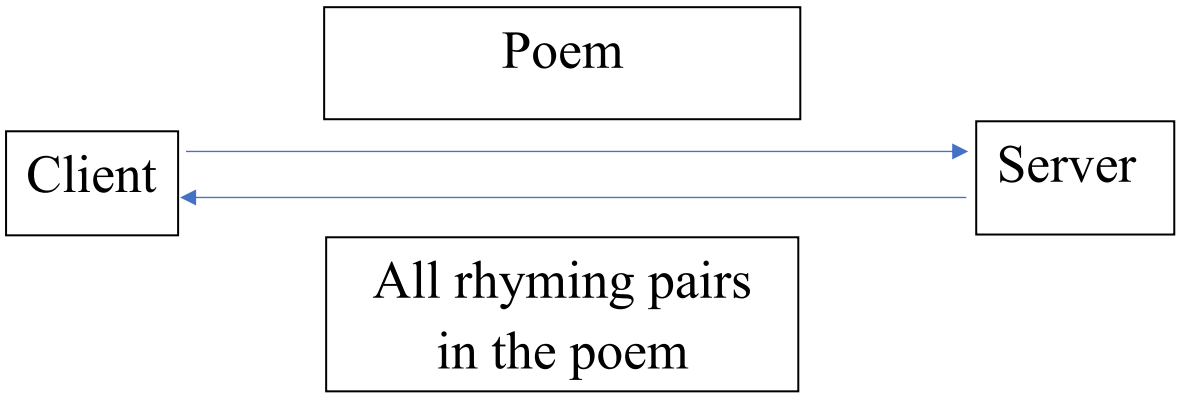

Figure 19: The client-server model for the web implementation

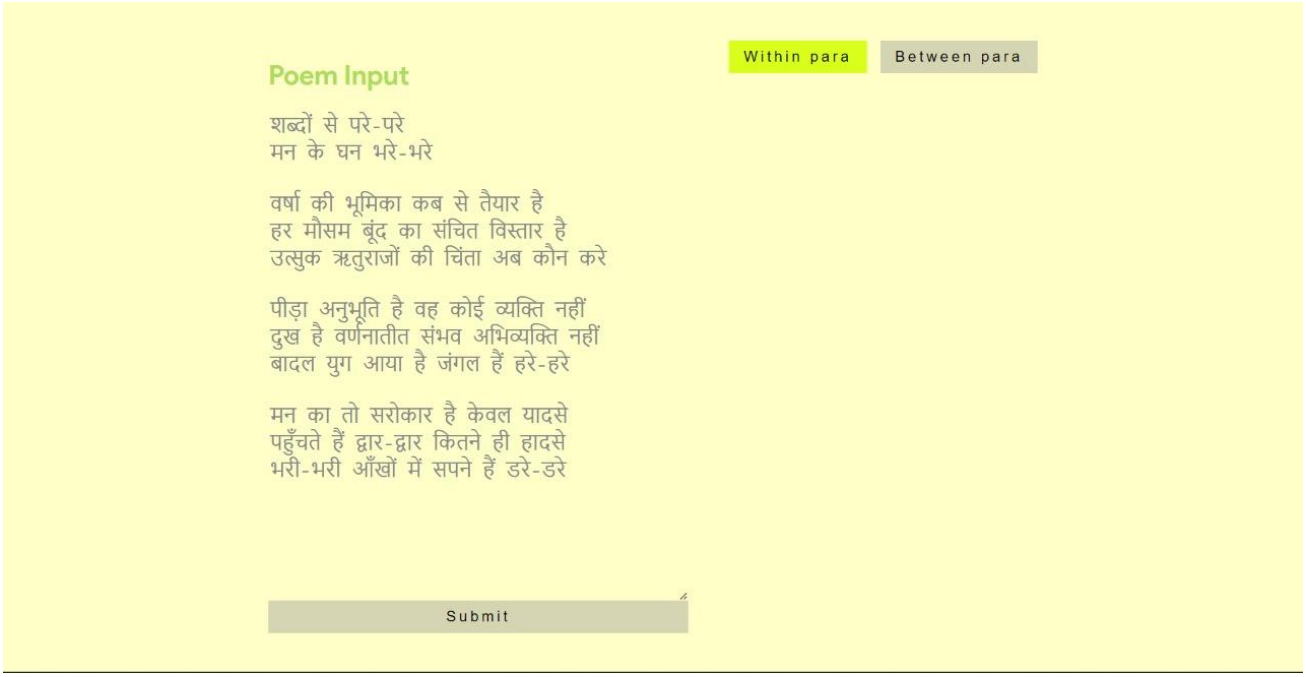

Figure 20: Screenshot of web application for taking a poem as an input

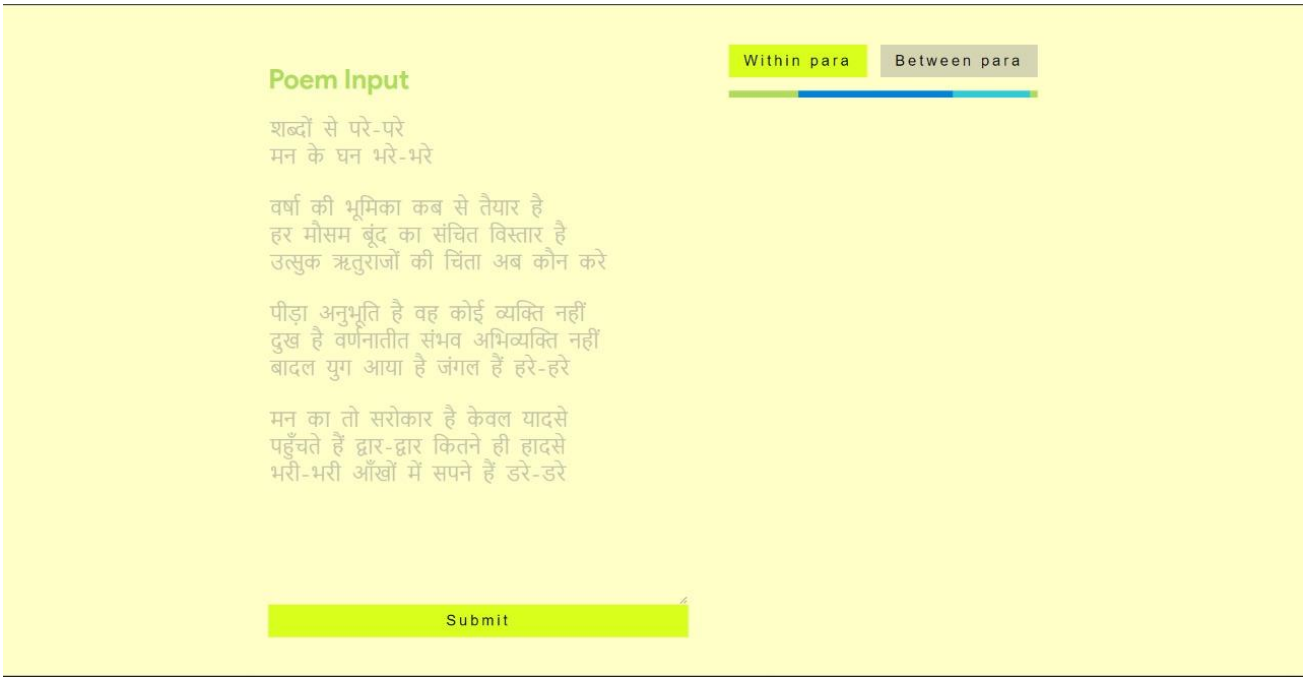

Figure 21: Screenshot of web application while submitting the poem for rhyme analysis

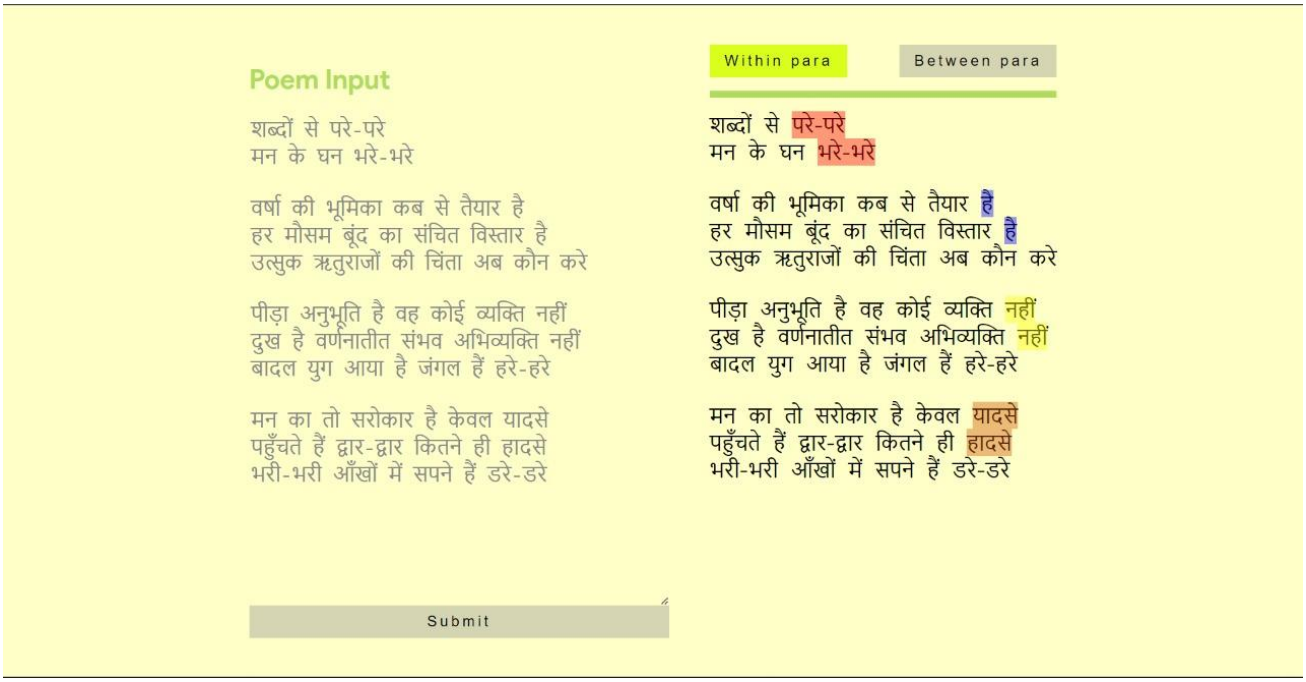

Figure 22: Screenshot of the web application for in-para rhyming pairs

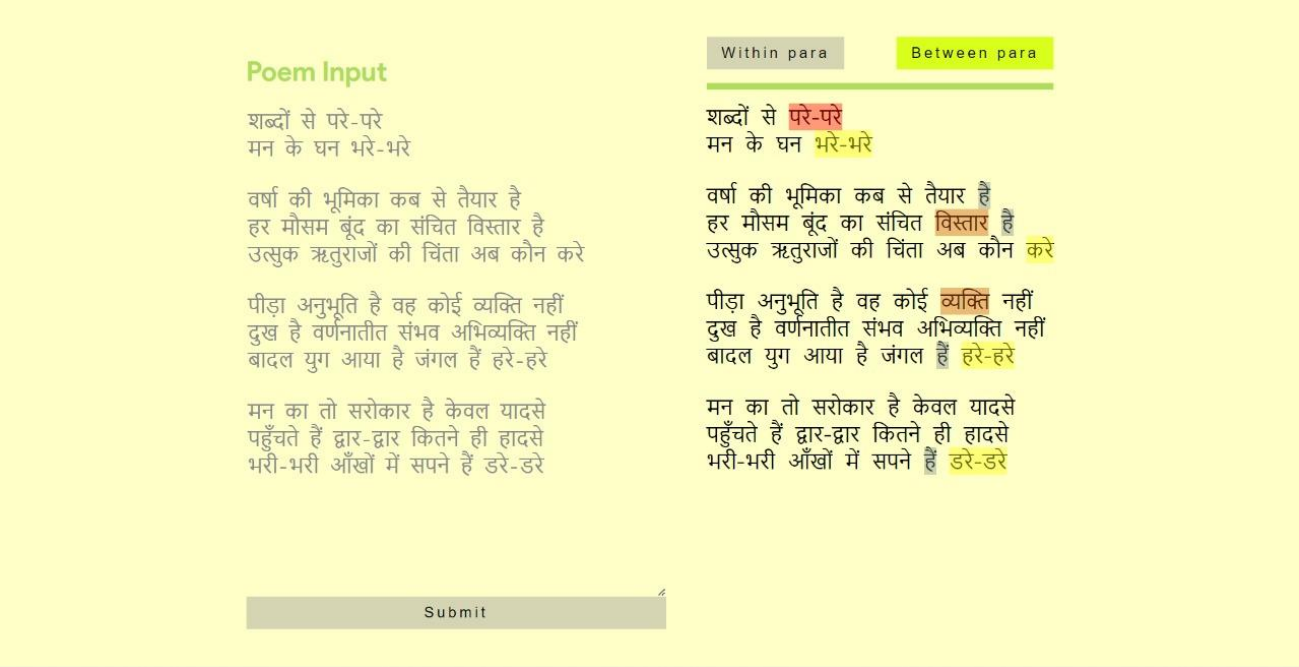

Figure 22: Screenshots of web application for Between para rhyming pairs

Conclusions

We used a total of 4 machine learning models. Out of them, three were based on classical similarity measures and one was trained via Siamese Recurrent Network. The three similarity measures that we used were, Cosine Similarity (using Phonetic representation), Euclidian Distance, and Cosine Similarity (using TF-IDF representation).

The accuracy obtained by various models was in order: SRN > Cosine Similarity (TF-IDF) > Euclidian Distance > Cosine Similarity (Phonetic Representation).

The reason for poor performance of classical techniques was, they only took the occurrence of a character in the word into account. They didn't take the sequence of occurrence of various characters into account. Also, two words containing same characters may not be rhyming. E.g., **राम-मरा** have same characters, but their sequence determines that they are not rhyming. However, the similarity measures will classify them as rhyming. This is the case of False Negative.

The TF-IDF based encodings gave better accuracy because they take the frequency of a character in other words too into consideration.

To take the sequence into account while training, the Recurrent Neural Networks into play. They learn dependencies and relationships among various different characters in a word and are trained in that way.

Since we were working on the text similarity measure at character-level, the use of Siamese Recurrent Network was the most optimal way to teach the model about rhyming relationship between two words. LSTM Recurrent Neural Network gave the best accuracy using the Phonetic representation of words and finally vectorizing them via one-hot vector encoding. The accuracy of model trained using SRN, was ~25% greater than the model trained via classical distance measures.

This implies that the various features have a sequential and context information that classical models were unable to capture.

The dimensions for LSTM units which gave the best accuracy were 50. The optimal dropout came out to be 0.25. We fine tuned the model based on metrics like accuracy, precision, recall, and F1-Score. The bidirectional LSTM worked better than the unidirectional LSTM units. This implies that the characters in future have backward rhyming relationship with characters in the past.

Using the results stated above we can state that:

The alphabets in Hindi are written in such a way that they follow the phonetic representation i.e., the dependent vowels are written together, and the consonants come after that. The poetry in Devanagari based language is an excellent source to study the rhythmic characteristics of Devanagari based languages. To develop a rhyme detection model for a particular language, we can use the rhyming pairs occurring in the poems written in that language for training purposes.

Accomplishments

We were able to complete the project under its deadlines. We delivered a dataset containing ~3000 poems written in various Devanagari based languages. We delivered a dataset containing 554 rhyming pairs in Hindi along with 249 non-rhyming pairs. We were able to train both classical machine learning models and deep-learning based models for the purpose of detection of rhyme between two words in Hindi. Finally, in order to demonstrate the capabilities of our deep-learning based model, we created a web application that takes a poem as an input and finally calculates all the inter-stanza and intra-stanza rhymes in that. The web application highlights the words that rhyme together in the same color. There was sparse research in the field of rhyme detection and no prior work had done in rhyme analysis for Indian languages. The dataset we created will help the future research on rhyming analysis in Hindi.

Future Work

As of now, the model is trained on the basis of sequence of the characters present in the words. We didn’t take the occurrence of more than one word together. The future work can be aligned on the analysis of rhyming characters based upon the n-gram model. Also, in SRN’s Convolutional Neural Networks can be tried. The further work includes the expansion of poems and rhyming pairs dataset. We have taken rhyming pairs from poetry only, but the pairs can also be taken for primary school poems. The rhymes in such poems are usually more informative. The other sources of rhyming pairs include songs, hip-hop etc. Such sources will help in the expansion of our dataset. This expansion will eventually lead to the increase in the accuracy of the model we trained and will help the future researches in a better way.

Acknowledgements

We would like to thank, Dr. Kamlesh Dutta, Associate Professor, Department of Computer Science and Engineering, NIT Hamirpur for providing us guidance throughout the project.

Individual Contributions

Name	Roll no.	Contributions
Aman Garg	185036	Trained and fine-tuned models: Cosine Similarity and SRN. Created the non-rhyming pairs dataset. Implemented the API for rhyme detection.
Rohit Kaushal	185037	Built the Web app that is integrated with the rhyme detection API. Wrote the web scrapping script to scrap the poetry dataset.
Prashant Kumar	185045	Analysed the available datasets. Created the rhyming pairs dataset.
Atul Thakur	185050	Trained and fine-tuned models: Euclidian similarity based and TF-IDF based.
Nikhil Sharma	185068	Verified the correctness of the dataset.
Azhan Ali	185096	Helped Prashant in creating the rhyming pairs dataset.

References

1. Thomas Haider and Jonas Kuhn, "Supervised Rhyme Detection with Siamese Recurrent Networks", *Proceedings of Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 81–86 Santa Fe, New Mexico, USA, August 25, 2018.

2. Priyanka Acharya, Aditya Ku. Pathak, Rakesh Ch. Balabantaray, Anil Ku. Singh, "Language Identification of Devanagari Poems", *Arxive.org* December, 2020

3. Luis-Gil Moreno-Jiménez, Juan-Manuel Torres-Moreno, Roseli S. Wedemann, "A Preliminary Study for Literary Rhyme Generation based on Neuronal Representation", Semantics and Shallow Parsing, *STIL 2021* - Symposium in Information and Human Language Technology.

4. Thomas Haider, Metrical Tagging in the Wild, "Building and Annotating Poetry Corpora with Rhythmic Features", Proceedings of the 16th *Conference of the European Chapter of the Association for Computational Linguistics*, pages 3715–3725 April 19 - 23, 2021. ©2021 Association for Computational Linguistics

5. Chourasia, V., Samudravijaya, K., Ingle, M., & Chandwani, M. (2007), "Statistical analysis of phonetic richness of Hindi text corpora."

6. Arpita Das, Harish Yenala, Manoj Chinnakotla, and Manish Shrivastava. 2016, “Together we stand: Siamese networks for similar question retrieval”. *In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 378–387.

7. Jonas Mueller and Aditya Thyagarajan. 2016, “Siamese recurrent architectures for learning sentence similarity”. In *AAAI*, pages 2786–2792.

8. Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016, “Learning text similarity with siamese recurrent networks”. *In Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157.

9. Sravana Reddy and Kevin Knight. 2011, “Unsupervised discovery of rhyme schemes”. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 77–82. Association for Computational Linguistics.

10. Morgan Sonderegger. 2011, “Applications of graph theory to an english rhyming corpus”. *Computer Speech & Language*, 25(3):655–678.