GDAŃSK UNIVERSITY OF TECHNOLOGY

TASK Quarterly 27 (3) 2023

Neural approach for rhyming word recommendations

Aleksandra Borowska
s181307@student.pg.edu.pl
Gdańsk University of Technology
ul. Narutowicza 11/12, Gdańsk, Poland

Natalia Kosakowska
s180771@student.pg.edu.pl
Gdańsk University of Technology
ul. Narutowicza 11/12, Gdańsk, Poland

Samuel Szurman
s175702@student.pg.edu.pl
Gdańsk University of Technology
ul. Narutowicza 11/12, Gdańsk, Poland

https://doi.org/10.34808/0vcg-ps40

Abstract

This research paper deals with the problem of rhyme generation. The project concerns words in the Polish language. Two methods have been proposed to determine whether two words rhyme. A proprietary algorithm was created and three types of neural networks were trained. The efficiency of the methods and the way each of the discussed methods works was compared.

Keywords:

algorithm, neural network, poem, rhyme

1. Introduction

Rhymes play a significant role for the artist, not only in terms of lyric writing, but also for musicians, where rhyme unifies the text and gives it rhythm and regularity [1]. Rhymes have always been associated with the creativity and originality of the artist. In modern poetry, rhymes are less common, but in music they are still the basis of songwriting [2]. With advances in technology, lyric creation can become much easier with the use of programs that can generate or recommend a rhyme for a given word. In addition, when the algorithm is properly corrected, these rhymes can have a better quality or accuracy compared to the rhymes that a human will suggest [3].

It is not possible for an algorithm to be universal or adaptable depending on the language. The process of rhyming can be complicated not only by the number of words appearing in a given language, but also by the rhyming rules or the phonetics and rhythmicity of the words. Whether words rhyme or not often depends not on spelling, but on pronunciation. All the rules, definitions and types of rhyme for a specific language must be taken into account in order for this program to best serve its purpose [4].

In English, rhymes rely primarily on phonetics, with pronunciation, accent, and rhythm playing key roles. This language, however, has a complex relationship between spelling and pronunciation, which can make rhyme detection more challenging [5]. In languages with more consistent orthography and phonetic rules, such as Portuguese [6] or Spanish [7], rhymes are more closely linked to spelling, though phonetic factors like pronunciation and stress still need to be considered for accurate rhyme detection.

This paper proposes an algorithm for identifying and evaluating rhyming words in the Polish language, primarily based on the sound of word endings. Additionally, neural network models were trained to mimic this algorithm.

2. State of the art

Rhyme generation has already appeared in methods that are based on recurrent neural networks and language models. The essence of these solutions is not rhyme, because rhyme is only one component of the works. In most cases, this problem is solved from a broader perspective and the main goal is to generate poetry or song lyrics in such a way that they sound as realistic as possible and do not suggest the use of artificial intelligence. It is important to maintain the best possible quality of the generated texts and rhymes and to properly evaluate this process. In the paper “Deep-speare: A Joint Neural Model of Poetic Language, Meter and Rhyme” [8] a unidirectional LSTM is used to model rhymes. LSTM learns to separate rhyming and non-rhyming pairs in a quatrain. It does this by comparing the last word (t) of one line with the remaining words (x, y, z) also at the end of the line. The final result is one pair of rhyming words (t, x) and two pairs of non-rhyming words (t, y) and (t, z). The trained model learns the margin and determines the best rhyming pair. The second-best pair is used to quantify the remaining rhymes. Estimating the rhyme between two words is done by calculating the cosine of similarity during generation. By adding other non-rhyming words from the dictionary, the process can be easily increased.

The solution presented in "GPoeT: a Language Model Trained for Rhyme Generation on Synthetic Data" [9] utilises a language model (LM) to generate rhymes in English. It relies on a dictionary that provides each word’s pronunciation along with its corresponding phonemes, encoded in ASCII format. Based on this data, two separate dictionaries were constructed with different key structures.

The first dictionary maps phoneme sequences to words that form perfect rhymes, e.g., ’eyn’ → ’campaign’, ’overtrain’, ’plane’. The second dictionary focuses solely on assonant rhymes. The rhyme generation process is guided by specific metrics derived from selected NLTK functions, such as word_tokenize and get_close_matches. Words are then classified into one of three categories: perfect rhyme, assonant rhyme, or no rhyme.

The paper "Automatic Detection of Internal and Imperfect Rhymes in Rap Lyrics" [10] was based on a dictionary containing phonemes. These are compared and returned as log-odds, indicating the probability of a pair. They are stored in two matrices, one for vowels and the other for consonants, and the results of phoneme pairs are used to determine scores according to the formula (1).

$$vowelScore + stressScore + consonantScore \quad (1)$$

VowelScore is the score found in the vowel matrix for vowels. StressScore corresponds to the indication of the accent on the vowels. The more similar the accent, the higher the score. ConsonantScore corresponds to the entries in the matrix of consonants occurring after a vowel as in the case of the words: code and mold.

The paper "Creating and Evaluating a Lyrics Generator Specialized in Rap Lyrics with a High Rhyme Density" [11] is based on the work of Hirjee and Brown [10]. However, there is a difference - instead of grouping syllables by evenly distributing consonants to vowels, syllables are divided by using the CMU pronunciation dictionary. Thanks to this, words are better divided into syllables, and

thus the detection of multi-syllable rhymes is improved.

In "Rapformer: Conditional Rap Lyrics Generation with Denoising Autoencoders" [12], a rap corpus built on the BERT-base model is used. The system predicts 200 candidate words to end a line, selecting those with the highest rhyme score. This score is determined by the longest vowel overlap between the candidate word and the target word.

The paper "Supervised Rhyme Detection with Siamese Recurrent Networks" [13] describes the detection of rhymes in German. In this solution, Siamese Recurrent Networks (SRN) are trained to predict whether two words rhyme. The architecture is based on a three-layer model of bidirectional LSTMs followed by another dense layer that forwards. Sequences rhyme when the model score is 0.5. Using 5000 rhymes from the database, non-rhyming sequences are generated. Based on these, SRNs are trained for 100 epochs in a $2:3$ ratio and an accuracy of 96% is obtained. Additionally, the SRN was also trained on an English database of $10,000$ rhyme pairs, which also resulted in an accuracy of 96% and for $30,000$ rhyme pairs, it was 97% accurate.

In "Using Siamese Neural Networks to Create a Simple Rhyme Detection System" [14], a Siamese Recurrent Network (SRN) trained on rap lyrics from the Genius website was used to predict rhymes. The dataset consisted of $1,000,000$ rhyme pairs — $500,000$ positive and $500,000$ negative. The data was split into three subsets: 60% for training, 30% for validation, and 10% for testing. The model's architecture includes a Siamese LSTM layer with 64 hidden units, followed by three dense layers and an input layer. The system achieved a 95% accuracy on the test set.

Chinese poetry requires that the last characters of some lines rhyme according to precise tonal rules. Each character has a tone assigned to it. This causes a bit more problems because it takes into account not only the pronunciation but also the tonality, which consists of the tones *Ping* (horizontal), *Ze* (falling) and *Ping*|*Ze* [15].

In the "Rhyming Knowledge-Aware Deep Neural Network for Chinese Poetry Generation" [16] solution, the model is based on the Sequence-to-Sequence (Seq2Seq) architecture, which consists of an encoder and a decoder. They use the LSTM structure to analyse the relationships between characters within a line, as well as the influence of previous lines on the generation of subsequent lines. In this case, the encoder in the Recursive Encoder Layer transforms the input character sequence into a fixed-size context vector, and these vectors are additionally enriched with tonal information using a $one-hot$ vector, thus maintaining the requirements of Chinese poetry. Phonological tones replace the original cell states and enforce control governing the selection of characters. The Embedding Layer uses the *word2vec* method, where Chinese characters are represented as dense, low-dimensional vectors, which is important for creating poetry. In the Recurrent Decoder Layers, the context vector is taken as input and generates an output sequence of characters. Both the semantics and the tone of the poem are taken into account. In order to optimise the selection of suitable characters, the $softmax$ function is used, which calculates the probability distribution for the next character and assigns a higher score to characters that meet the tone and rhyme requirements.

In the article "Rhyme Detection of Hindi and Rajasthani Poems using Statistical-Based Methods" [17], classical static methods and analysis of similarity between phonetic features were used. Rhyme analysis was based on the repetition of final sounds "a" and "i", frequency of words with specific properties and characteristics of phonetic waves. For example, for the word *aatma* the phonetic wave is:

$$ASCII(a)+ASCII(a)+length(aatma)= \\ 97+97+5=199 \qquad (2)$$

where a higher wave value indicates a greater tonal influence of a given word in the structure of the poem. Each of the poems was represented as a feature vector, which allowed for its comparison with other texts in the feature space. Then, comparative methods were used to determine cosine similarity, Euclidean distance and Jaccard index. If at least two of the three methods assessed the text as similar, it was classified as a poem. Three algorithms were used to evaluate the effectiveness of the solution. The first one involved an SVM classifier that searched for the optimal decision boundary between classes (poems and non-lyrical texts) in the feature space. Thanks to its ability to work in high-dimensional spaces, SVM proved effective in modeling complex dependencies between features. Another method was the use of neural networks. These modeled nonlinear dependencies in the data by relying on weight layers and activation functions. Kernel methods were also used, where the algorithms transformed the input data into a new feature space, which allowed for better class differentiation. This method proved to be particularly effective in working with small data sets. For both Hindi and Rajasthani, the kernel methods proved to be the most effective, with accuracy of over 93%.

"Dynamically Scoring Rhymes with Phonetic Features and Sequence Alignment" [18] uses linguistic foundations with genetic optimisation methods to create a universal rhyme scoring function. Rhyme is defined as the phonetic similarity between the *stress tails* of two or more words. The *stress tail* includes: *nucleus* (the central vowel sound of a syllable), *coda* (consonant sounds after

the nucleus) and *onset* (ll subsequent syllables after the most stressed syllable).

This definition assumes that the most stressed part in a word determines its significance for rhyme scoring. For example, the words *station* and *creation* rhyme, despite having different numbers of syllables, because their stress tails are similar.

Phonemes, which are the basic sound units, are decomposed into phonetic features, which allows the rhyme function to be generalised to any language. Phoneme analysis is based on the International Phonetic Alphabet (IPA) standard. The calculation process is multi-stage:

- extraction of stress tails - for two words w_1 and w_2, the function extracts stress tails s_1 and s_2 according to syllables from the highest stress value
- matching syllables - syllables s_1 and s_2 are matched, and each pair of syllables is assessed according to the formula:

$$S(\sigma_1, \sigma_2) = \alpha_w * A(w_1, w_2) + \alpha_v * R_v(v_1, v_2) + \alpha_\kappa * A(\kappa_1, \kappa_2) \quad (3)$$

where:

- α_w, α_v, α_κ - weights assigned to onset, nucleus and coda,
- $A(x, y)$ - consonant matching in onset and coda using the Needleman-Wunsch algorithm,
- R_v - evaluation of vowels based on the phonetic similarity matrix

Matching of consonants and vowels is carried out dynamically in a similar way to the formula 3 and the final rhyme result for words R(w_1, w_2) is calculated as a weighted average of the obtained results.

In this solution, weight optimisation was performed using a genetic algorithm. The population was 100 individuals representing a set of weights. The matching function was to minimise the squared error between the rhyme function results and the reference data. The obtained results were satisfactory, as it was possible to obtain a mean square error (MSE) of 0.012, which means very good agreement of the model results with human assessments.

3. Data

An important stage of every project is the data collection phase. This project utilised the publicly available database from SJP (Dictionary of the Polish Language) [19]. The database is used both for creating the training and test datasets for the neural networks described in Section 5.1, as well as serving as the solution space for the algorithm described in Section 4, whose purpose is to recommend rhymes for a given word.

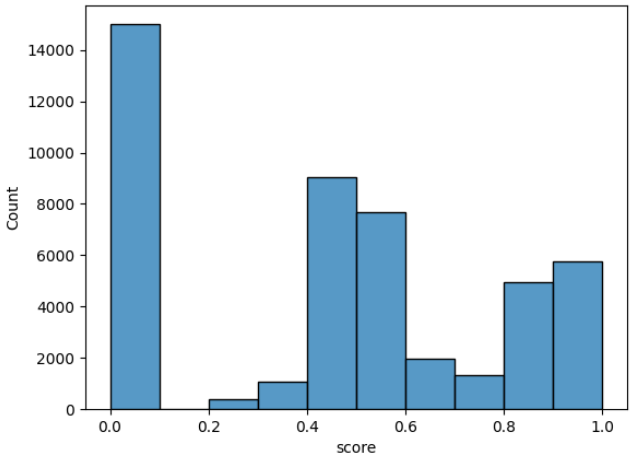


Figure 1: Distribution of the score parameter in the data set

The files used by the models have the following structure: *base word*, *rhyme candidate*, *score*. The *score* value is calculated using the aforementioned algorithm and ranges from 0 to 1 (the closer the value is to 1, the more the words rhyme). Figure 1 shows the distribution of the *score* parameter for word pairs included in the training dataset. For clarity, instances were grouped into 10 intervals based on the *score* value. The largest number of instances belongs to the group with a *score* range of 0.0 to 0.1, followed by the ranges of 0.4–0.5 and 0.5–0.6.

4. Our approach

4.1. Scoring

Our algorithm determines whether a pair of words rhyme or not, producing a final score on a scale of 0 to 1. The method assumes that each word contains at least one vowel and does not include capital letters or special characters.

The first step involves dividing each word in a pair into two parts, labeled Segment A and Segment B. Segment B consists of the last vowel (or group of vowels next to each other) and the last consonant (or group of consonants). All letters preceding Segment B constitute Segment A. The next step unifies Segment B based on pronunciation. In the Polish language, some letters represent the same sound, especially consonants at the end of a word. During this step, different written variants of the same sound are unified according to Table 1. If the unified

Segments B of both words being compared are different, the final score is set to 0, and the algorithm terminates. Otherwise, the base rhyming score is set to either 0.5.

Table 1: Unification of different versions of the same sounds

Written versions of the same sounds	Unified version
sz, ż, rz	sz
cz, dż	cz
p, b	p
k, g	k
ć, dź	ć
c, dz	c
f, w	f
t, d	t
s, z	s
h, ch	h
u, ó	u

The previous step essentially determines whether the two words rhyme or not. The subsequent step assesses the quality of the rhyme. In this phase, Segments B are compared to each other. The score is calculated using the formula (4). Here, $length_{sequence}$ represents the common sequence starting from the end of the Segment A. This value is then divided by the length of the maximum sequence, which is equal to the length of the Segment A of the first word minus 1 (excluding the first letter of Segment A).

$$score = score_{base} + (1 - score_{base}) \cdot \frac{length_{sequence}}{length_{max}} \quad (4)$$

The final step involves penalising differences in the number of syllables, as shown in formula (5). The 'max' function ensures that the score does not fall below 0.

$$score_{final} = max(score - 0.05 \cdot |\Delta_{syllables}|, 0) \quad (5)$$

4.2. Syllable counting

In our solution, we used pyphen to determine the number of syllables in words. Pyphen is a Python library that divides words into syllables and can be applied to many languages, including Polish. The divisions are mostly correct; however, pyphen encounters difficulties, especially with words starting with a vowel. To address this issue, we proposed our own algorithms for counting syllables. The algorithm counts all the vowels in a word, with two exceptions. The first exception occurs when 'i' precedes another vowel, as in this scenario the letter 'i' softens the preceding consonant(s) and does not form a separate syllable. The second exception is when the letter 'u' follows 'a' or 'e'. In the vast majority of cases, the letter 'u' sounds like the consonant 'ł' in this scenario, therefore it is not counted by the algorithm.

4.3. Rhyme searching

Our application searches for rhyming words for a given one entered by the user from a list sourced from the website `https://sjp.pl/sl/growy/`. This list currently contains 3,186,487 Polish words. Instead of comparing each word in the list with the user's input individually, we grouped the words by their properties. We created a new dictionary where the keys consist of two parts: the number of syllables and the unified Segment B of the words. Each key in the dictionary corresponds to a list of words from the original list. The largest list in the dictionary contains 19,358 words. Therefore, in the worst-case scenario, our algorithm checks around 164.6 times fewer words than beforehand. This approach is necessary due to limited computational power on the server.

5. Approach based on neural networks

5.1. Neural Network architecture

Three different neural network architectures were implemented, as illustrated in Fig. 2-3. The models consist of the following layers:

- **Embedding**: A layer responsible for representing words in a vector space, where similar words are located close to each other, capturing semantic and phonetic relationships between words.
- **Flatten**: A layer that transforms data from the Embedding layer into a single long sequence to be passed to the next layer.
- **Dense**: A deep layer with a ReLU activation function that models nonlinear relationships between data. A higher number of neurons in the layer allows for the capture of more complex patterns.
- **Dropout**: A layer introducing random neuron dropout during training to prevent overfitting.
- **LSTM (Long Short-Term Memory)**: A recurrent layer that facilitates the retention of dependencies between words in a pair. Its memory capabilities enable better pattern analysis.
- **Output**: A Dense layer with a sigmoid activation function returning a value in the range of 0–1, indicating the degree to which two words rhyme.

Table 2: Model 1 architecture

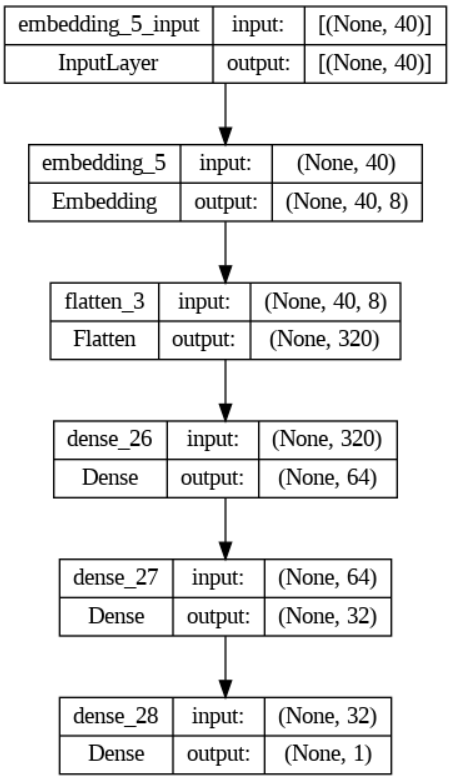


Table 3: Model 3 architecture

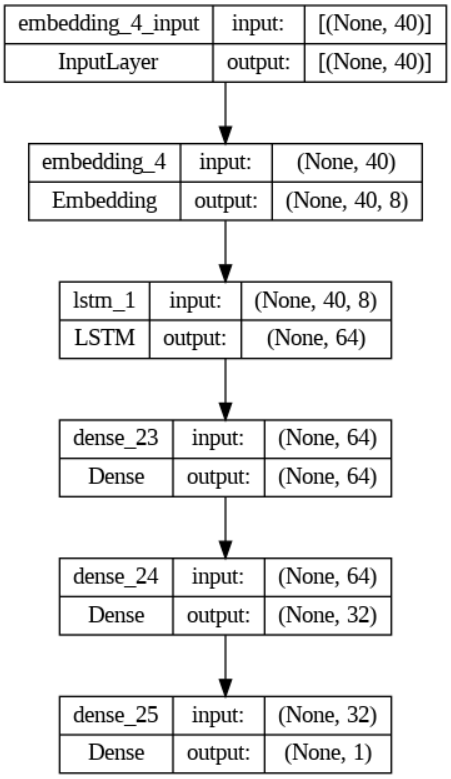
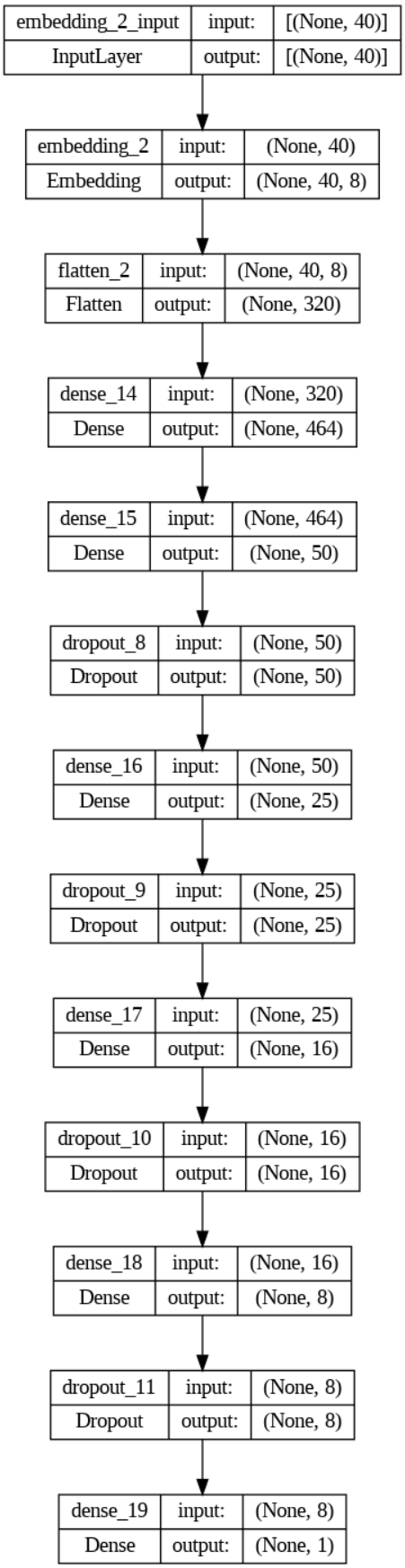


Table 4: Model 2 architecture



The first model architecture (Model 1) comprises only an Embedding layer, a Flatten layer, two Dense layers, and an Output layer. This is the simplest of all the proposed architectures, designed to assess the effectiveness of a very simple neural network for the given task.

The second model architecture (Model 2) extends the first by including a larger number of Dense layers (5 layers, allowing for capture of more complex patterns) and Dropout layers, aimed at preventing the network from learning patterns only for majority classes.

The third model architecture (Model 3) is the most advanced, incorporating an LSTM layer which captures long-term dependencies and sequential patterns in data.

5.2. Experiments

The neural networks discussed in Section 5.1 were trained using nested cross-validation, which allowed simultaneous hyperparameter tuning (optimiser type, number of epochs, batch size) and statistically significant evaluation of model performance.

For the outer loop, the value of $k = 10$ was defined, where k is the number of folds into which the dataset is divided. Out of these, $k - 1$ folds are used as the training set (passed to the inner loop), and 1 fold is used as the test set. For the inner loop (responsible for hyperparameter tuning), the value of $k = 3$ was applied.

During training, models were protected against overfitting by employing the *EarlyStopping* callback, which monitored changes in a specific metric during training and stopped the process when the metric showed insufficient improvement. Additionally, the *ModelCheckpoint* callback was used to save the model weights corresponding to the lowest loss value. The loss function used in the training process was the mean squared error (MSE). In Tables 5-7, the metrics MSE, MAE, and R^2 obtained for each model and each fold are presented.

Table 5: Performance metrics for Model 1

Fold	R^2	MSE	MAE
1	0.9617	0.0043	0.0352
2	0.9497	0.0056	0.0411
3	0.9634	0.0042	0.0345
4	0.9567	0.0048	0.0371
5	0.9557	0.0050	0.0378
6	0.9509	0.0056	0.0383
7	0.9596	0.0046	0.0355
8	0.9545	0.0051	0.0391
9	0.9528	0.0053	0.0392
10	0.9553	0.0050	0.0361
Average	0.9560	0.0049	0.0374

Table 6: Performance metrics for Model 2

Fold	R^2	MSE	MAE
1	0.9595	0.0046	0.0337
2	0.9420	0.0064	0.0445
3	0.9581	0.0048	0.0347
4	0.9432	0.0063	0.0399
5	0.9525	0.0054	0.0362
6	0.9474	0.0060	0.0382
7	0.9567	0.0049	0.0355
8	0.9571	0.0048	0.0358
9	0.9455	0.0061	0.0367
10	0.9525	0.0053	0.0377
Average	0.9514	0.0054	0.0373

Table 7: Performance metrics for Model 3

Fold	R^2	MSE	MAE
1	0.6957	0.0343	0.1179
2	0.8405	0.0177	0.0782
3	0.6998	0.0341	0.1201
4	0.8792	0.0133	0.0620
5	0.7828	0.0246	0.1014
6	0.7891	0.0241	0.0877
7	0.8265	0.0196	0.0788
8	0.8734	0.0142	0.0606
9	0.8808	0.0133	0.0599
10	0.8772	0.0137	0.0607
Average	0.8145	0.0209	0.0827

Based on the results presented in the tables above, it is evident that Model 1 and Model 2 achieve very similar metric values. Compared to Model 3, they demonstrate higher performance and lower variance in the results obtained across different folds.

6. Results

6.1. Results of experiments

To evaluate the performance of the trained models, they were tested on a dataset of nearly 5,000 word pairs. The evaluation involved calculating the MSE, MAE, and R^2 metrics by comparing the models' predictions with those of the reference algorithm. Based on the values presented in Table 8, Models 1 and 2 achieved the best results, with both models yielding similar performance across all metrics.

Table 8: Metrics obtained during model testing

Metrics	Model 1	Model 2	Model 3
MSE	0.0052	0.0057	0.0145
MAE	0.0367	0.0382	0.0608
R^2	0.9542	0.95	0.8729

Furthermore, to compare the performance of the three models, the rhyme quality results for 15 random word pairs presented in Table 9 were analysed. The results indicate that neural networks achieve very good performance for shorter words, but perform worse with longer ones. This insight could be valuable for refining the models to enhance their accuracy. Additionally, it can be observed that the LSTM model often produces results that are significantly different from those of other models, both better and worse. The discrepancies in results obtained by the LSTM model compared to models 1 and 2 may stem from the fact that the LSTM network treats consecutive word pairs as sequential data, where the order of pairs and words matters. However, in the considered problem, the order does not play a role. For LSTM, the sequence of input vectors is crucial, which likely explains the cases where the network generates significantly different results compared to the other two models.

Table 9: Calculated rhyming scores by compared algorithms

Pair of words	Ground truth	Model 1	Model 2	Model 3
białawą, sztucy	0	0	0	0.03
bądźcie, wióra	0	0	0	0.17
międzysezonem, bursztynnikiem	0	0.38	0.45	0.17
fotokopiach, anhydrynowych	0	0.37	0.47	0.13
chciałbyś, roztrwaniałabyś	0.47	0.53	0.66	0.44
widoku, rosołku	0.5	0.38	0.5	0.47
rysunki, tętniczki	0.5	0.41	0.5	0.46
bała, siodła	0.5	0.56	0.5	0.46
czarowi, korbowi	0.67	0.6	0.67	0.52
skrzyżowaniu, majstrowaniu	0.8	0.6	0.52	0.38
gotowość, wariantowość	0.82	0.4	0.48	0.49
kpi, wygłupi	0.9	0.9	0.96	0.89
pakowaniem, blikowaniem	0.92	0.56	0.61	0.43
smakowaniem, lokowaniem	0.92	0.57	0.6	0.34
drań, pasań	0.95	0.95	0.98	0.92

7. Discussion and future works

The project successfully developed an algorithm for evaluating rhymes in the Polish language and trained neural network models that effectively replicate its functionality. The algorithm primarily focuses on the phonetic structure of word endings. However, the words in the datasets were not subjected to morphological analysis, nor were they selected to cover the full range of possible cases. This likely contributes to the discrepancies observed in the R^2 metric. The neural networks predominantly learned to evaluate rhymes correctly for word pairs representing the most common morphological patterns. A key challenge moving forward would be to create a new dataset that accounts for a more balanced distribution of examples, incorporating a broader spectrum of morphological variations.

The presented approach has been implemented and is now accessible on the website `https://rhymesgenerator.eu.pythonanywhere.com/` in Polish. The back-end component was developed in Python using the Flask library, while the front-end was crafted in TypeScript utilising the Angular framework.

Further efforts could focus on devising algorithms for other languages. For languages like Spanish or Portuguese, which, similarly to Polish, have a spelling system that largely unambiguously reflects pronunciation, the algorithms should be similar, provided they account for the phonetic rules specific to each language. Conversely, a challenge would arise, for example, with English, where such ambiguity does not exist. It is important to remember, however, that spelling alone does not convey all the information about a word's pronunciation. In future work, it would be beneficial to rely on a purely phonetic transcription.

References

[1] iZotope, "Mastering rhyme schemes in lyric writing," 2018. Accessed: 2024-12-01.

[2] C. Obermeier, W. Menninghaus, M. von Koppenfels, T. Raettig, M. Schmidt-Kassow, and S. A. Kotz, "Aesthetic and emotional effects of meter and rhyme in poetry," *Frontiers in Psychology*, vol. 4, p. 10, 2013.

[3] O. Vechtomova, G. Sahu, and D. Kumar, "Researchers develop real-time lyric generation technology to inspire song writing," *AIhub*, 2021. Accessed: 2024-12-01.

[4] A. Popescu-Belis, Atrio, B. Bernath, E. Boisson, T. Ferrari, X. Theimer-Lienhard, and G. Vernikos, "Gpoet: a language model trained for rhyme generation on synthetic data," in *Proceedings of the 7th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pp. 10–20, Association for Computational Linguistics, 2023.

[5] E. Suresh Kumar and P. Sreehari, "Accent, rhythm and intonation," in *A Handbook for English Language Laboratories*, pp. 41–66, Foundation Books, 2009.

[6] H. Gonçalo Oliveira, A. Cardoso, and F. Pereira, "Tra-la-lyrics: An approach to generate text based on rhythm," pp. 47–55, 06 2007.

[7] P. Gervás, "An expert system for the composition of formal spanish poetry," *Knowledge-Based Systems*, vol. 14, pp. 181–188, 06 2001.

[8] J. Lau, T. Cohn, T. Baldwin, J. Brooke, and A. Hammond, "Deep-speare: A joint neural model of poetic language, meter and rhyme," 07 2018.

[9] A. Popescu-Belis, R. Atrio, B. Bernath, E. Boisson, T. Ferrari, X. Theimer-lienhard, and G. Vernikos, "Gpoet: a language model trained for rhyme generation on synthetic data," 01 2023.

[10] H. Hirjee and D. Brown, "Automatic detection of internal and imperfect rhymes in rap lyrics.," 01 2009.

[11] M. Wentink, "Creating and evaluating a lyrics generator specialized in rap lyrics with a high rhyme density," February 2023.

[12] N. I. Nikolov, E. Malmi, C. Northcutt, and L. Parisi, "Rapformer: Conditional rap lyrics generation with denoising autoencoders," in *Proceedings of the 13th International Conference on Natural Language Generation* (B. Davis, Y. Graham, J. Kelleher, and Y. Sripada, eds.), (Dublin, Ireland), pp. 360–373, Association for Computational Linguistics, Dec. 2020.

[13] T. Haider and J. Kuhn, "Supervised rhyme detection with Siamese recurrent networks," in *Proceedings of the Second Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature* (B. Alex, S. Degaetano-Ortlieb, A. Feldman, A. Kazantseva, N. Reiter, and S. Szpakowicz, eds.), (Santa Fe, New Mexico), pp. 81–86, Association for Computational Linguistics, Aug. 2018.

[14] P. Minogue, "Using siamese neural networks to create a simple rhyme detection system," 2021.

[15] G. B. Downer and A. C. Graham, "Tone patterns in chinese poetry," *Bulletin of the School of Oriental and African Studies*, vol. 26, no. 1, pp. 145–148, 1963.

[16] W.-C. Yeh, Y.-C. Chang, Y.-H. Li, and W.-C. Chang, "Rhyming knowledge-aware deep neural network for chinese poetry generation," in *2019 International Conference on Machine Learning and Cybernetics (ICMLC)*, pp. 1–6, 2019.

[17] I. Kumar and K. Dutta, "Rhyme detection of hindi and rajasthani poems using statistical-based methods," in *2023 IEEE International Conference on Contemporary Computing and Communications (InC4)*, vol. 1, pp. 1–5, 2023.

[18] B. Bay, P. Bodily, and D. Ventura, "Dynamically scoring rhymes with phonetic features and sequence alignment," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1581–1585, 2019.

[19] *Słownik języka polskiego*. Wydawnictwo Naukowe PWN, 2023.