NLP1: Report Poetry

Promotion: SCIA 2026

Group 5

Matthias Laithier
Erwin Rodrigues
Aziz Zeghal
Robin De Bastos
Lucas Tilly

Repository: https://github.com/Aziz-Zeghal/NLP1-Poetry/tree/main

Contenu

1. Introduction

This report outlines the work conducted in the *NLP1* course, focusing on the *Poetree* dataset. The primary objective of this project is to classify the writing date of a poem based solely on its text.
Additionally, we will leverage the same dataset for poem generation, where the objective is to generate poems starting from an initial sentence.

Given the stylistic diversity, and limited labeled data, how can we design machine learning models that accurately classify the period in which a poem was written ?

For each task, we present the two best performing models as well as one of the least effective models, to highlight both the strengths and limitations of the approaches explored.

1.1 Dataset caracteristics

The dataset was sourced from the Python library *Poetree*, comprising poems written across different periods and in various languages. The initial step involved retrieving the dataset and storing each author's creations in a serialized pickle file for further analysis.

We began with the French dataset but, after careful consideration, switched to the German dataset. The French dataset's date distribution was highly imbalanced, and the sample size was smaller, making it less suitable for our analysis.

For our analysis, we have selected the German language, as it contains many poems, which are well-distributed across different time periods.
Each poem in the dataset contains the following attributes:

- **Title**: The title of the poem.
- **Author**: The poet's name.
- **Text**: The full text of the poem.
- **Date**: The date when the poem was written.

1.2 Exploratory Data Analysis

Our first step was to explore the dates our our poems, as it will determine the repartition of the classes.

General information

The dataset contains 71,570 poems with a total of 12.6 million words. The word count analysis reveals that poems tend to have an average of 176 words, with a wide spread in word count ranging from a minimum of 3 words to a maximum of 15,700 words.
The most frequent words in the German poems highlights common poetic themes. Notably, words such as *Gott* (God), *Herz* (heart), *Welt* (world), *Wer* (who), and *Himmel* (heaven) appear prominently, reflecting strong spiritual, emotional, and philosophical motifs.

Poems per century

When organized by century, a clear imbalance is evident with the 19th century being significantly overrepresented (*Figure 1*). The average word count for poems varies by century, with the 11th century having the lowest average word count of 37.68 words, and the 13th century having the highest average word count of 1,804.37 words. This shows that, while the number of poems varies across centuries, there are also variations in the length of poems.

Poems per German movement

When grouping poems by artistic movement, we observe a strong predominance of the Barock movement (*Figure 2*). This literary era, known for its ornate style and deep religious or philosophical content, is heavily represented in the dataset.
A key contributor to this imbalance is the high frequency of poems by prominent Baroque authors. Notably, Angelus Silesius stands out as the most prolific, followed by Rückert, Friedrich and Logau, Friedrich von (*Figure 3*).

Overall

The dataset is imbalanced both temporally with a strong bias toward the 19th century, and stylistically, with the Barock movement heavily represented. Depending on the model, we may address this by applying class weighting or generating synthetic samples to improve balance (with SMOTE for example).
We chose to classify poems by century rather than by exact date to simplify the multi-class classification task by reducing the number of target classes.
For further analysis, please refer to the *EDA.ipynb* file in the repository, which includes detailed markdown organizing the various observed trends.

1.3 Preprocessing

Each model will be encoded using both TF-IDF and Word2Vec representations. As a general preprocessing step, German stopwords are removed using the *NLTK* library, and text is stripped of leading and trailing whitespace.

Additional preprocessing steps are applied depending on the specific model requirements. For instance, label encoding is performed for neural network architectures, and lemmatization may be applied to improve text normalization.

2. Benchmark of Classification Models

2.1 TF-IDF Logistic Regression

This model is among the best-performing models. It is implemented within a pipeline to allow for streamlined preprocessing and hyperparameter tuning. The first stage of the pipeline applies TF-IDF vectorization, which offers both fast computation and an effective representation of term importance across the corpus.

To address class imbalance in the dataset, we integrate the SMOTE algorithm. This technique generates synthetic examples for underrepresented classes, helping to mitigate the impact of sample disparity during training.

The final stage of the pipeline applies logistic regression. Among the solvers tested, *sag* and *lbfgs* yielded almost identical performance. We selected *sag* due to its superior speed on large datasets.

As shown in the confusion matrix (*Figure 4*), the model exhibits strong performance with predictions concentrated along the diagonal, indicating a high level of accuracy. However, there is a noticeable tendency for the model to overpredict the 18th, 19th, and 20th centuries. This bias is likely due to the overrepresentation of poems from these centuries in the dataset, which skews the learned decision boundaries in favor of the more frequent classes. Despite this, the model demonstrates good generalization across other classes.

2.2 Word2Vec Feedforward Neural Network

This model combines Word2Vec embeddings with a two-layer neural network and ranks among our best-performing classifiers. The preprocessing step applies spaCy based lemmatization, which, while slower than NLTK, yielded better results.

The Word2Vec vectors have a dimensionality of 500 and feed into a perceptron with a hidden layer of 128 neurons using ReLU activation. We trained the model with a batch size of 64 using the Adam optimizer (learning rate = 0.001) and CrossEntropyLoss.

As shown in the confusion matrix (*Figure 5*), the model performs especially well on the 11th, 15th, 17th, and 18th centuries (classes 1, 4, 5, and 7), with classification accuracies reaching up to 93.6%. However, it struggles with the 12th and 21st centuries (classes 2 and 8), which suffer from data scarcity and lower F1-scores (0.38 and 0.45).

The model occasionally misclassifies 19th-century poems as 18th-century (22.89% confusion), reflecting stylistic continuity across eras. Overall, the model achieves 84% accuracy and a weighted F1-score of 0.84, matching the performance of TF-IDF Logistic Regression while benefiting from semantic awareness via embeddings.

2.3 TF-IDF Naive Bayes

This model uses a TF-IDF vectorizer in combination with a Multinomial Naive Bayes classifier. We decided to use a Scikit-learn pipeline that allows us to have a clean integration of preprocessing, feature selection, and hyperparameter tuning.

The TF-IDF stage converts raw poem texts into a weighted term-document matrix, which helps us highlight stylistic changes across centuries. As Naive Bayes does not natively support class weighting, we addressed class imbalance by resampling the training data. This ensured that each century was represented more evenly, preventing the model from being biased toward the overrepresented 18th, 19th, and 20th centuries. Additionally, we applied chi-squared feature selection to reduce dimensionality and remove irrelevant terms, which helped improve both training speed and generalization.

As shown in the confusion matrix (*Figure 6*), the model performs well on some centuries like the 14th, 16th and 17th, with an accuracy above 75%. However, there is significant confusion between adjacent centuries, particularly between the 13th and the 14th. This reflects the stylistic continuity in poetic language over those periods, which poses a challenge for bag-of-words approaches.
Despite its simplicity, the TF-IDF Naive Bayes model remains computationally efficient and interpretable, making it a good balance between simplicity and pertinence.

2.4 Overall

Across all classification models tested, **TF-IDF Logistic Regression** and **TF-IDF Feedforward Neural Network** emerged as top performers, each achieving above 85% accuracy and weighted F1-score. The former benefited from simple, fast computation and effective regularization, while the latter leveraged semantic embeddings for better contextual understanding.

TF-IDF Naive Bayes offered strong results given its simplicity and was particularly effective in earlier centuries, despite struggling with stylistic overlaps in adjacent classes.

We also experimented with methods like **Bagging** and **Stacking**. Although they did not surpass our best models, they improved robustness and broadened our understanding of model combination techniques.

Overall, these experiments provided valuable insights into trade-offs between accuracy, complexity, and interpretability. A comparative benchmark of all models is available in *Figure 7* and the project repository[1].

[1] `https://github.com/Aziz-Zeghal/NLP1-Poetry`

3. Benchmark of Text Generation Models

For all our generation models, we trained exclusively on the English-language poems from our dataset, containing approximately 42,000 items. We chose the ROUGE and BERTScore metrics to evaluate our models performance. In order to do so, we selected 20 diverse prompts and, for each, identified a reference poem from the dataset. These reference poems were chosen by comparing each prompt to all available poems, selecting the one with the closest semantic match. This comparison was facilitated by a pre-trained language model, which helped assess the semantic similarity between prompts and candidate poems. By doing so, we ensured that the generated poems could be evaluated fairly against meaningful and contextually relevant references.

3.1 N-gram Method

The first model evaluated is based on a simple statistical approach using N-grams. It generates text by relying on the frequency of word sequences in the training corpus, without modeling context or meaning.

ROUGE scores are low: ROUGE-1 = 0.0628, ROUGE-2 = 0.0000, ROUGE-L = 0.0628, and ROUGE-Lsum = 0.0628. This indicates minimal lexical similarity between the generated poems and the references.

BERTScore provides a more nuanced perspective, with a precision of 0.7174, recall of 0.7594, and F1 score of 0.7378. These results suggest that the model sometimes reflects thematic closeness, even if the outputs lack fluency and cohesion.

Overall, while the N-gram model fails to produce structured or stylistically rich poetry, it occasionally approaches the intended meaning. It offers a simple but useful point of comparison for evaluating more advanced systems.

3.2 Feedforward Neural Network

For this model, we trained a Word2Vec embedding on our full corpus and used the resulting vectors as input to a feedforward neural network. The goal was to generate poetry by combining distributional word semantics with a simple neural architecture.

In terms of ROUGE, we obtained ROUGE-1 = 0.1633, ROUGE-2 = 0.0111, ROUGE-L = 0.1089, and ROUGE-Lsum = 0.1559. These results show that the model performs better than the N-gram baseline in terms of lexical overlap, though it still lags behind the Transformer model in surface-level similarity.

Looking at BERTScore, the model achieved a precision of 0.7905, recall of 0.7643, and F1-score of 0.7771. These scores are quite solid and suggest that the generated poems are semantically meaningful even if the exact wording differs from the references.

Overall, the MLP model offers a good balance between statistical and neural approaches. It captures semantic content better than N-gram models and is more lightweight than Transformers. However, its main limitation is the lack of sequential modeling, which affects its ability to maintain coherence across longer text spans.

3.3 Transformer

For this approach, we used the GPT2 pre-trained model from Hugging Face's transformers package, then trained it with its trainer on our dataset which we tokenized using GPT2's pre-trained tokenizer available via the same library.

For ROUGE, the scores are a bit low (ROUGE-1 = 0.19, ROUGE-L = 0.12 and ROUGE-Lsum = 0.18). This tells us that our generated poems are different in the exact word choice and structure form from the references, but we believe it is not something to be worried about since poetry should be really free in terms of choices.

On the other hand, for BERT, we got a precision of 0.84 and an F1-score of 0.83. We believe that this is consistently strong, especially across the 20 different prompts we made. The model has a decent semantic alignement and it shows that it can produce meaningful and related poetry. It is not just grammatically coherent but also thematically and semantically grounded.

An example of a generated poem with the prompt "It never ends" can be found here: (*Figure 8*).

3.4 Overall

We notice overall pretty low ROUGE scores, which are explained by the fact that poetry is inherently creative, diverse, and often non-literal. A good generated poem might express the same idea as the reference but with entirely different wording, structure, or metaphors. Since ROUGE is based on exact word overlaps, it penalizes originality, even when the meaning is preserved.

On the other hand, our BERT-based metrics are overall way better, especially for our transformer approach. This suggests that our model is generating text that is semantically aligned with the reference poems. This means that while the surface form differs, the core message, themes, or emotions are preserved, and this is exactly what we wanted in poetry generation: freedom of form with preservation of meaning or aesthetic quality.

In conclusion, the combination of low ROUGE and high BERT-based scores supports the idea that we successfully managed to generate original yet semantically faithful poems. This outcome highlights the limitations of traditional lexical overlap metrics in evaluating creative text and reinforces the relevance of embedding-based evaluations for tasks like poetry generation.

4. In-depth Approaches

4.1 Model combination

To research more complex classification methods, we chose to implement the combination of multiple models and evaluate their performances. We decided to go with Bagging and Stacking, two well known methods in that domain.

The first one consists of having the same model multiple times and giving each of them a random sample of the train data. The result is then calculated from all of the models. This approach prevents over fitting.

The stacking method consists of training multiple different models, providing us with a way to have the advantages of multiple models at once. The result is then calculated from the aggregation of all trained models.

While in our case, we managed to have better results with more basic methods, this approach allowed us to learn other possibilities that may become essential later on.

Results are available in *Figure 9.1* and *Figure 9.2*.

4.2 Data Augmentation

For the classification task, we explored techniques for generating new poems from existing ones. Poems were grouped by century, and new synthetic poems were created by combining lines extracted from different poems within the same century. To further increase variability and reduce stylistic coherence, lines were shuffled to break any original sequential structure, resulting in novel, artificial compositions that retained temporal relevance.

However, this augmentation technique did not lead to improved performance. One likely reason is that shuffling and recombining lines disrupted the inherent poetic structure and stylistic coherence that models use as implicit features for temporal classification. Moreover, the resulting synthetic poems may have introduced noise rather than useful variation, especially if the model relies on more global textual cues or coherent linguistic patterns that were lost during recombination.

In addition, we explored other augmentation techniques using libraries such as *textattack* on poems translated into English. One of the methods we tested was synonym replacement to introduce lexical variation. *Germanet*, a semantic network for the German language, could have facilitated controlled word substitutions. Unfortunately, its access requires an institutional license, which was not available for this project.

4.3 Dataset merge

For the generation task, we expanded our dataset by translating German poems into English. Using a translation API without rate limits, we enriched the corpus with stylistically diverse texts, aiming to improve the model's generalization and creative capacity.

However, when we compared the evaluation metrics (ROUGE and BERTScore) for models trained on the original dataset and those trained on the expanded one, we found that adding translated German poems had little to no effect. BERTScore remained effectively unchanged for all of our different models, indicating that the semantic similarity between generated and reference poems did not improve. Moreover, the mostly unchanged ROUGE scores suggest that the additional data did not introduce meaningful new patterns, leading to no improvement in the model's ability to match the reference texts at the lexical level.

These results imply that the added data did not provide significant new learning signals, likely due to redundancy with the previous dataset or insufficient stylistic diversity. Overall, the augmentation did not meaningfully enhance the quality of the generated poems under our evaluation metrics.

5. Conclusion

In this project, we explored both classification and generation tasks using a large corpus of historical poetry.

For the classification task, our goal was to predict the century in which a poem was written, and we benchmarked several models which seemed pertinent. Despite the challenges of imbalanced data and stylistic overlap across time periods, both TF-IDF Logistic Regression and the Word2Vec Feedforward Neural Network achieved strong performance, reaching above 85% accuracy and good generalization.

For the generation task, we implemented and compared N-gram methods, a feedforward neural network, and a fine-tuned GPT-2 Transformer. Evaluation using ROUGE and BERTScore highlighted the limitations of traditional lexical metrics for creative text, but confirmed the semantic strength of our Transformer-based approach. The model demonstrated the ability to produce thematically and stylistically consistent poetry, despite significant lexical diversity.

We also experimented with advanced techniques such as model ensembling, data augmentation, and dataset merging. While some approaches yielded limited performance improvements, they provided valuable insights into the complexities of text modeling in both structured and creative contexts.

To answer the initial problematic, we demonstrated that machine learning techniques can effectively classify poems by century and reveal stylistic trends across different historical periods.

It would also be interesting to explore other forms of literary expression, such as prose or theater, to examine whether similar temporal patterns or themes emerge. Extending this research to a broader community of texts across multiple languages and cultural contexts could offer deeper insights into shared motifs and the evolution of literary style over time.

Annexes

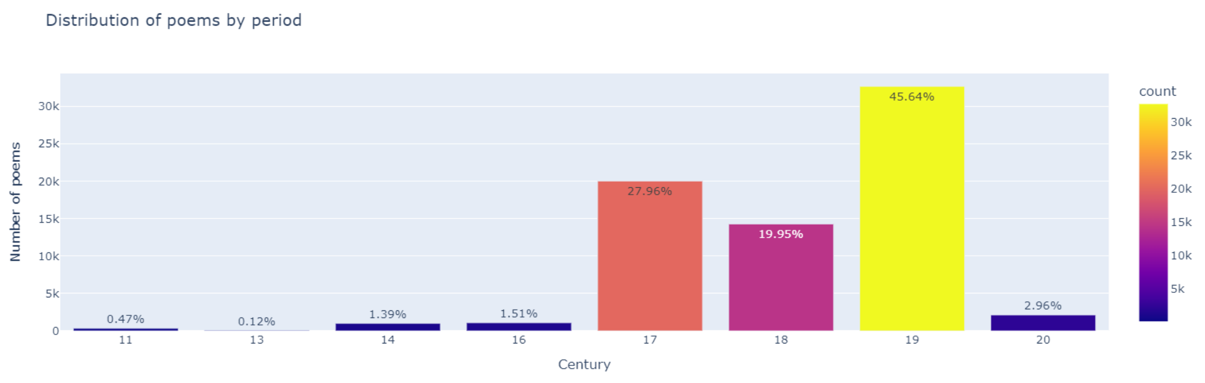


Figure 1: Distribution of poems by century.

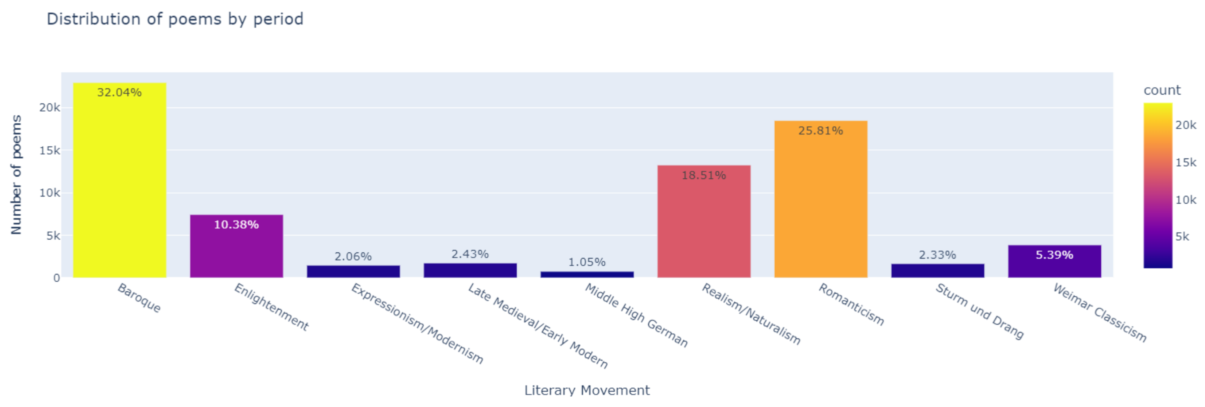


Figure 2: Distribution of poems by movement.

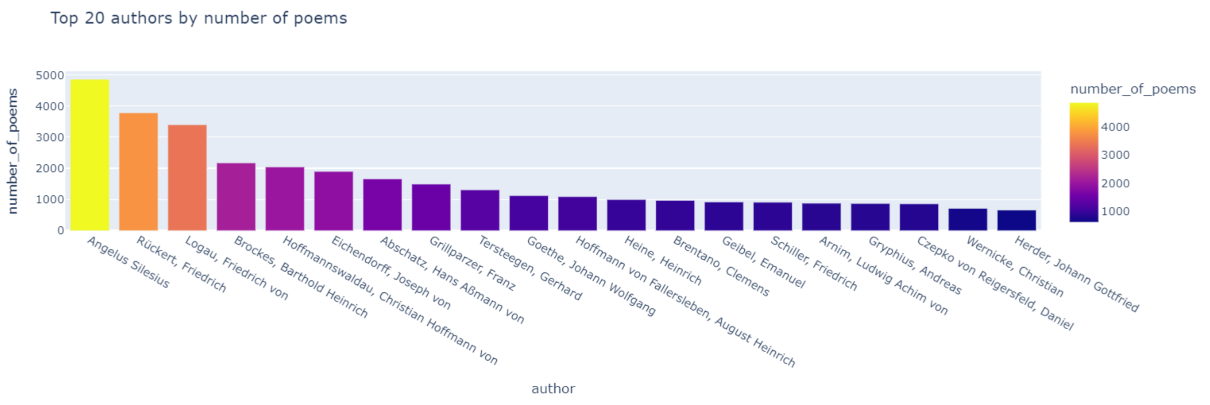


Figure 3: Top 20 authors.

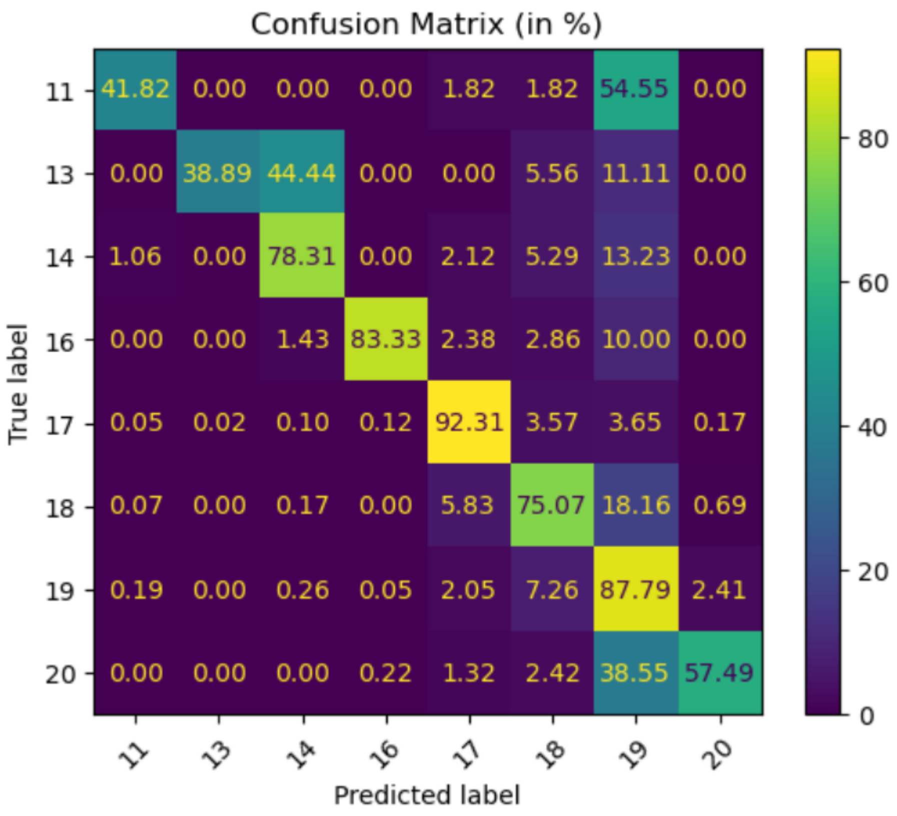


Figure 4: Confusion matrix of the TF-IDF Logistic regression model.

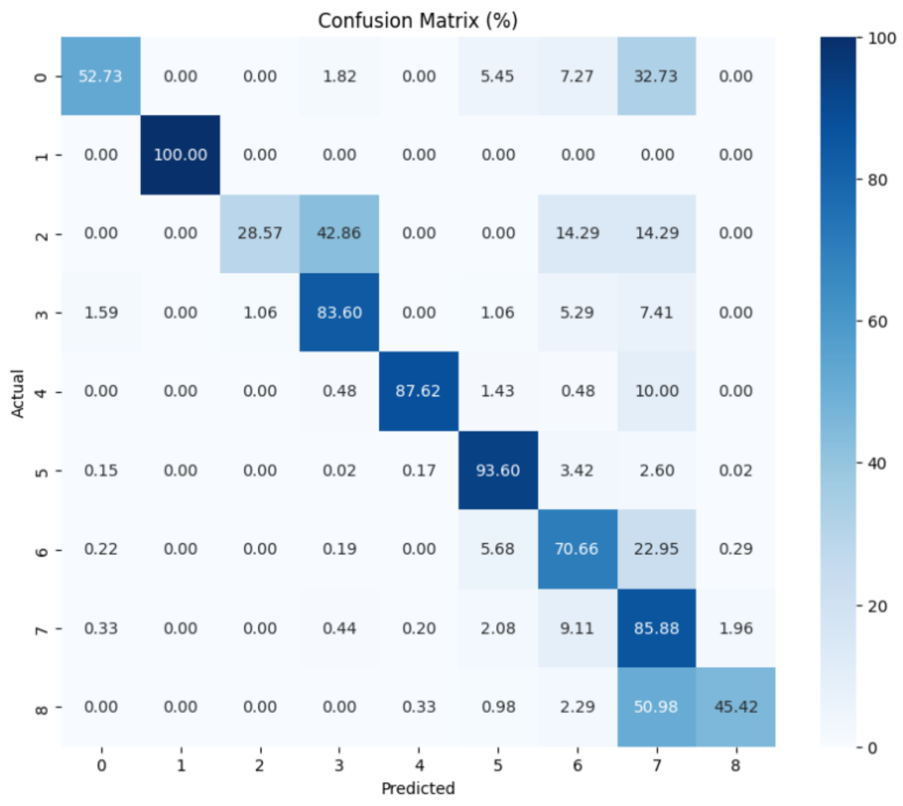


Figure 5: Confusion matrix of the Word2Vec Feedforward Neural Network model.

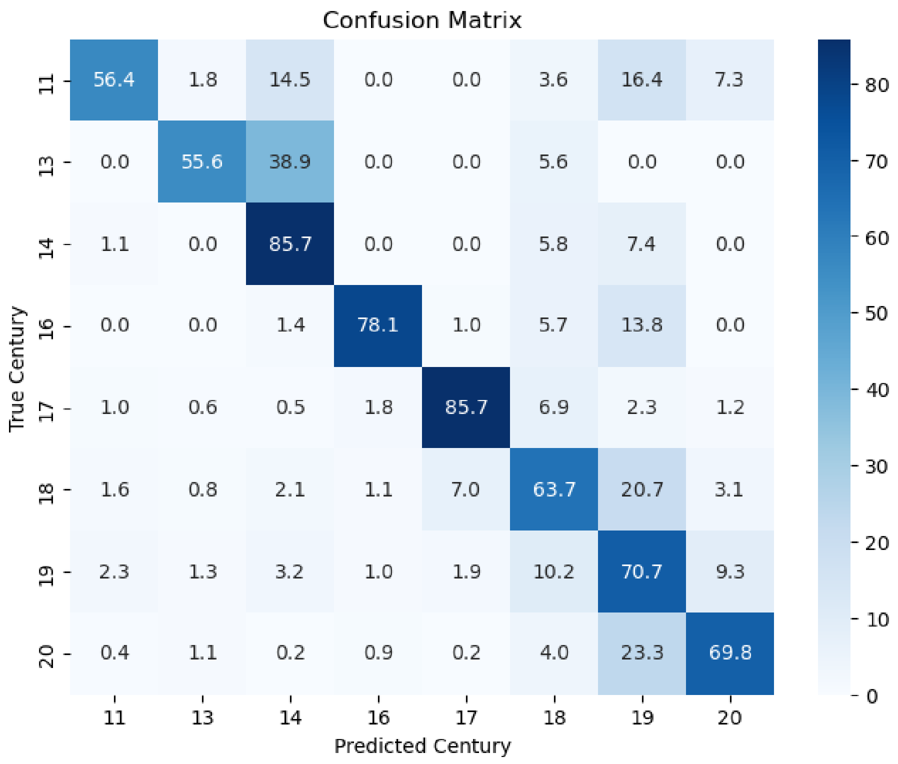


Figure 6: Confusion matrix of the TF-IDF Naive Bayes model.

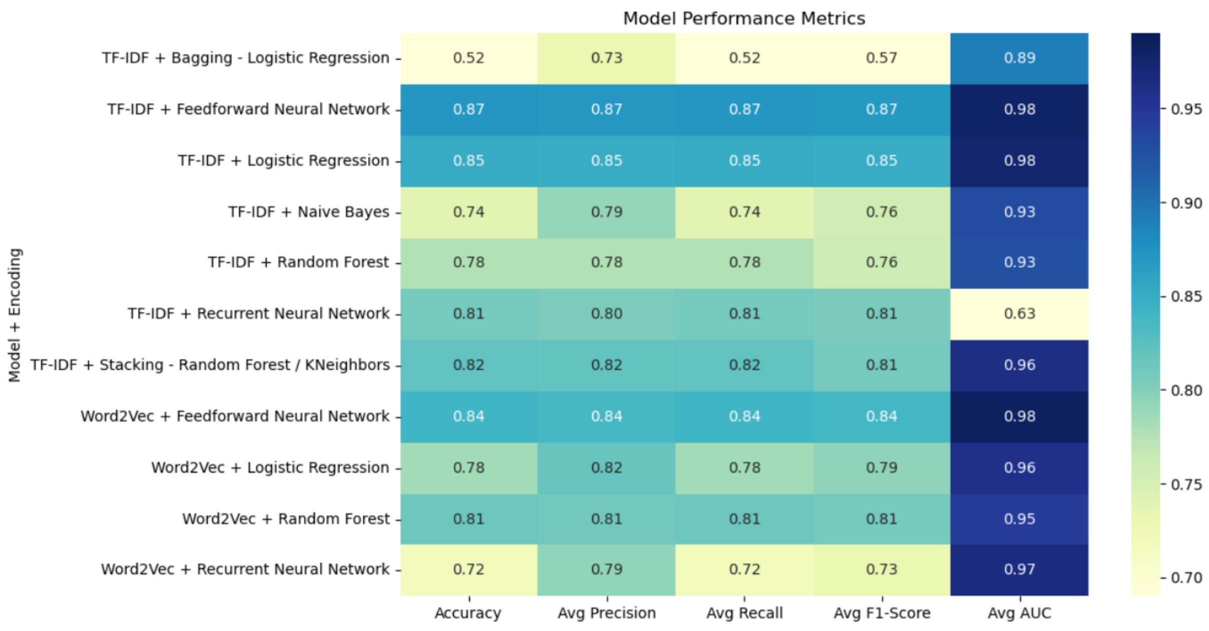
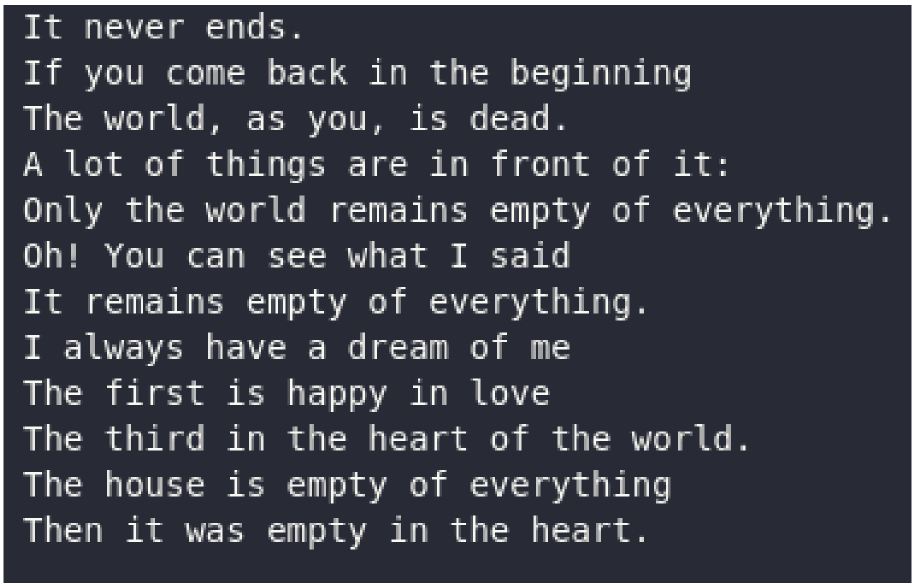


Figure 7: Metrics of all classification models.



```
It never ends.
If you come back in the beginning
The world, as you, is dead.
A lot of things are in front of it:
Only the world remains empty of everything.
Oh! You can see what I said
It remains empty of everything.
I always have a dream of me
The first is happy in love
The third in the heart of the world.
The house is empty of everything
Then it was empty in the heart.
```

Figure 8: Example of generated poem with the transformer method.

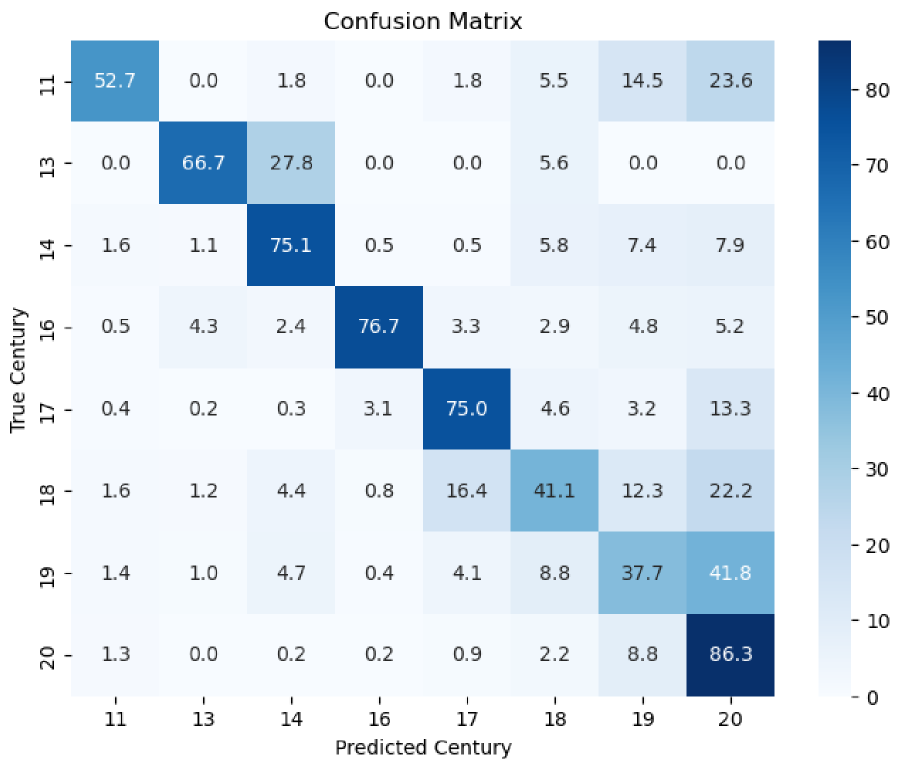


Figure 9.1: Confusion matrix of the bagging method.

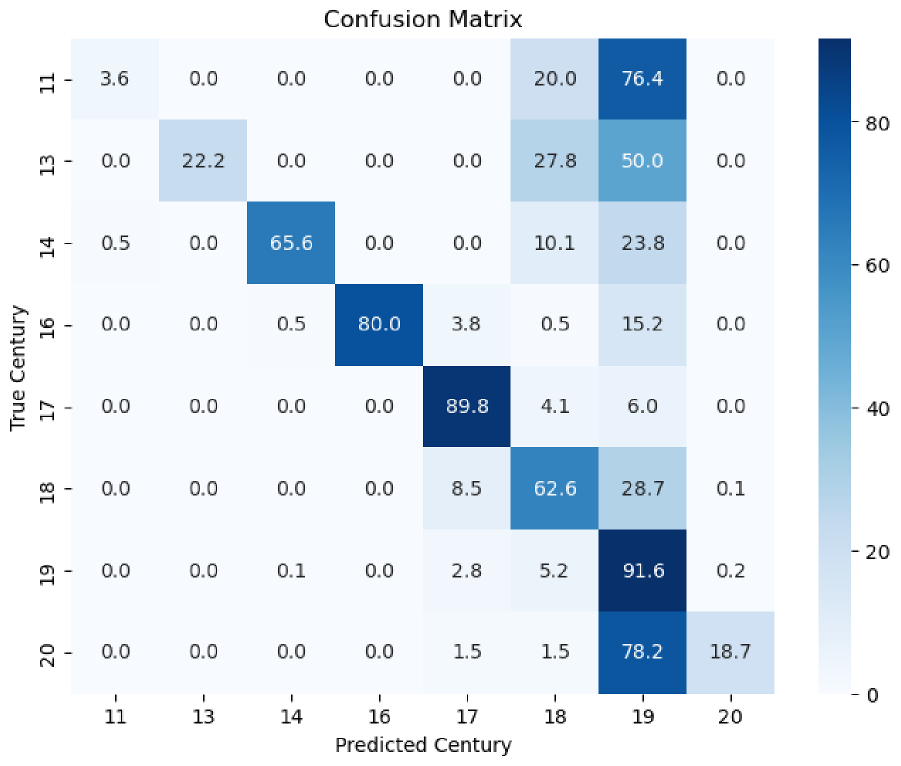


Figure 9.2: Confusion matrix of the stacking method.