# Teaching Machines to Write Like Dostoevsky and Recognize the Word of God: Language Processing with Neural Networks and Clustering Methods

Jonas Jørgensen Telle[1], Marius Torsheim[1], and Maria Klüwer Øvrebø[2]
[1]*Department of Physics, University of Oslo, N-0316 Oslo, Norway*
[2]*Department of Mathematics, University of Oslo, N-0316 Oslo, Norway*
(Dated: December 13, 2024)

**Abstract:** Language processing has long been a central part of AI research, and the ability to determine the authorship of a text is of particular interest recently with the increased use of large language models in educational contexts. We develop a Recurrent Neural Network-based text *generator* capable of emulating five selected authors, and a *classifier* which assigns text-chunks to one author among these five, employing either a neural net or clustering methods such as $k$-means and HDBSCAN. We seek cost efficient solutions, exploring dimension reduction techniques like PCA and UMAP and the use of semi-supervised learning to reduce labeling costs for classification. On 150 word chunks, our best classifier achieves a test accuracy of 96.5%, and generalizes successfully in an actual application on other works by the same authors, giving an accuracy of 97.9%. Our generator produces text chunks which are correctly classified 99.2% of the time.

## I. INTRODUCTION

A primary focus of contemporary machine learning applications is the analysis of language. We aim to create a classifier able to correctly assign text to its author, and a generator able to emulate specific writing styles. Such classification can be extended to facilitate studies of literary traditions, revealing previously unknown styles and characteristics. Furthermore, with the recent success of large language models, it is increasingly relevant to seek the development of reliable methods to determine the authorship of a text, particularly in educational contexts.

Our overarching goal is to develop a dual-system framework: a Recurrent Neural Network (RNN)-based text *generator* capable of emulating the literary styles of selected authors, and a *discriminator* employing either a Feedforward Neural Network (FFNN) or clustering methods such as $k$-means and HDBSCAN with dimension reduction using UMAP and PCA to classify the generated text [1–3]. This framework might offer a first stepping stone toward a generative adversarial network (GAN) that could refine style replication autonomously, a goal that lies beyond the scope of this work but remains a tantalizing prospect for future research.

Specifically, we aim to capture the styles of five prominent authors: Jane Austen, Fyodor Dostoevsky, Miguel de Cervantes Saavedra, Snorri Sturluson and the anonymous contributors of the King James Bible, whom we for brevity shall refer to as *God*. Our approach involves two steps: First, we develop our classifiers to accurately attribute authorship of a given chunk of text by training on selected representative works from each author and then testing on different works by these same authors. The success of our models on this task indicates that they are also qualified to assess the performance of our generator. We then train a generator on the work of each of the authors, and use the classifier to assert whether the generated text is in fact most similar to the author whom we attempt to emulate. We repeat this for each author with different generation temperatures both on several chunk sizes (50-150 words) and on grammatical sentences which do not have a fixed length, finding that low temperatures and large chunk sizes gives the most solid performance.

Given the high costs associated with classification tasks we explore several methods to reduce expenses. This include both dimension reduction methods and a study of the necessary labeled dataset size to reduce computational cost. UMAP allow for semi-supervised classification, and since the labeling of data is oftentimes the most expensive part of the training process, it is of great interest to study the model accuracy as a function of the amount of labeled data. Thus, one might hope that clustering methods could make use of large amounts of unlabeled data to compensate for reduced labeling, but our investigation indicates that the fully supervised FFNN outperforms clustering methods given the same amount of labeled data, even when the clustering methods are assisted by additional unlabeled data.

The report is organized as follows: It begins with a detailed explanation of the methodology, outlining the design and implementation of the classifier and generator algorithms. The next section is a presentation of the experimental results, describing the performance of the generator and classifier across various tasks. The final section offers a critical evaluation of the results, discussing their implications, identifying limitations, and suggesting potential directions for future research.

## II. THEORY

There are several key concepts used in this report which warrant an initial theoretical description. We use **text embeddings** to transform text into vector inputs for our classification algorithms and we use **clustering methods** supplemented by **dimension reduction algorithms** and **FFNNs** for classification. Our generator is implemented as an **RNN** using a `char2int`-dictionary.

### A. Text Embeddings

Text embeddings are a cornerstone of modern natural language processing (NLP), leveraging the **distributional hypothesis**, which posits that words appearing in similar contexts tend to have similar meanings [4]. Embedding models, such as `mxbai-embed-large-v1`[5] used in this project, leverage this simple hypothesis to represent words or larger text units in a high-dimensional vector space. Text embeddings have proven surprisingly successful in capturing semantic and syntactic relationships, making them an obvious choice to transform the text input for our discriminators.

### B. Dimension reduction

Text embeddings typically utilize large vector spaces (in our case, $\mathbb{R}^{1024}$), but typically, the relevant information for classification is contained in only some directions of the space. Dimension reduction algorithms seek to reduce the complexity of the classification problem by isolating these important directions. In this projec,t PCA and UMAP were used.

#### 1. PCA

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms data into a set of orthogonal components while preserving as much variance as possible. Mathematically, PCA involves finding the eigenvectors and eigenvalues of the covariance matrix of the data [6].

#### 2. UMAP

Uniform Manifold Approximation and Projection (UMAP) is a non-linear dimensionality reduction technique designed to preserve the global and local structure of high-dimensional data [2]. Unlike PCA, UMAP builds a graph representation of the data and optimizes a low-dimensional layout by minimizing a cross-entropy-like loss:

$$\mathcal{L} = \sum_{(i,j)\in \text{edges}} -w_{ij} \log(p_{ij}),$$

where $w_{ij}$ is the weight of the edge between points $i$ and $j$, and $p_{ij}$ is the probability of a connection in the low-dimensional space [7]. UMAP is particularly suited for clustering and visualizing text embeddings due to its ability to capture non-linear relationships.

Additionally, UMAP has the important advantage that one can use data labels to ensure that the reduction preserves information useful to distinguish the particular labels. Thus, UMAP implementations for dimensionality reduction can be partially supervised, such that our otherwise unsupervised clustering methods may be assisted by labels. This functionality is essential to our exploration of how the classification accuracy varies as a function of the amount of labeled data.

### C. Clustering

In this work, $k$-means and HDBSCAN are employed to cluster text embeddings derived from the generated and original texts, providing a complementary perspective to supervised classification and aiding the exploration of using only a subset of the labeled data to assert whether less labeling is possible without large performance loss.

#### 1. *k*-means

$k$-means clustering is an unsupervised machine learning algorithm used for partitioning data into $k$ clusters. The algorithm minimizes the within-cluster variance by iteratively updating the cluster centroids and assignments. The objective function is:

$$J = \sum_{i=1}^{K} \sum_{x\in C_i} \|\mathbf{x} - \mu_i\|^2,$$

where $C_i$ is the set of points assigned to cluster $i$, $\mu_i$ is the centroid of cluster $i$, and $\|\cdot\|^2$ denotes the squared Euclidean distance. $k$-means has been widely applied since its formalization [8]. In our case, we have a predetermined number of possible classifications - one of the five selected authors - making $k$-means the most natural choice.

#### 2. HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) is a density-based clustering method that builds a hierarchy of clusters and identifies stable clusters by looking at their persistence as the density threshold varies [3]. Unlike partition-based methods like $k$-means, HDBSCAN does not assign all points to clusters; some may be labeled as outliers, effectively handling complex data distributions and potentially revealing previously unseen structures.

However, HDBSCAN is also sensitive to high-dimensional data. Without effective dimensionality reduction, it may fail to identify meaningful clusters, often assigning the majority of points to noise [2]. Consequently, dimensionality reduction techniques such as UMAP or PCA are often employed beforehand to project the data into a space where density-based clustering is more feasible. Still, in most practical applications, a subset of the data often remains unlabeled, reflecting outliers or genuinely novel samples. While this can be seen as a limitation, it also highlights the algorithm's strength in distinguishing not only between known classes but also in signaling potentially novel categories.

#### *3. Adjusted rand score*

Performance of the clustering methods was measured using the adjusted rand score (ARS). This index, ranging from $-1$ to 1, measures the agreement between the clustering output and the ground truth labels, providing a quantitative basis for comparing the clustering methods [9]. The point is that, since the clustering algorithms do not specify which cluster belongs to which label, the naive evaluation where one assigns the labels to maximize the model performance gives an artificially high score. ARS corrects for this by dividing the difference between the raw rand index and the expected rand index (by pure chance) by the difference between the *maximal* rand index and the expected one.

### D. FFNN

FFNNs are among the simplest and most widely used neural network architectures for classification tasks. In a typical FFNN, information flows uni-directionally from input nodes through hidden layers to the output layer, with no cycles or feedback. Mathematically, the output of a layer is computed as:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias vector, and $f$ is a nonlinear activation function such as ReLU, sigmoid, or tanh.

Activation functions introduce non-linearity into the model, enabling neural networks to approximate complex functions that go beyond simple linear mappings. Commonly used activation functions in FFNNs include the sigmoid, tanh, and Rectified Linear Unit (ReLU) functions. The *sigmoid* function $\sigma(z) = \frac{1}{1+e^{-z}}$ is bounded between 0 and 1, making it a suitable choice for output layers dealing with probability estimates, as in our case [10].

FFNNs are trained using supervised learning, optimizing the model parameters by minimizing a loss function such as cross-entropy [10]:

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{j=1}^{C} t_{ij}\log(p_{ij}),$$

where $t_{ij}$ is the true label for class $j$ for sample $i$, and $p_{ij}$ is the predicted probability. Backpropagation and stochastic gradient descent are commonly used to update the parameters. In this project, FFNNs serve as the primary classifier for distinguishing between the stylistic features of generated and original text. For details concerning their theoretical background and gradient descent methods, the reader may consult our previous work at [11].

### E. Recurrent Neural Networks

RNNs are a class of neural networks designed for sequential data. Unlike feedforward neural networks, RNNs introduce a feedback loop, allowing information to persist across time steps. This makes RNNs particularly well-suited for tasks like text generation, where the order of words and contextual dependencies play a crucial role.

At each time step $t$, an RNN computes the hidden state $\mathbf{h}_t$ as:

$$\mathbf{h}_t = f(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t + \mathbf{b}),$$

where $\mathbf{W}_h$ and $\mathbf{W}_x$ are weight matrices for the previous hidden state and input vector $\mathbf{x}_t$, respectively, $\mathbf{b}$ is the bias vector, and $f$ is an activation function, typically tanh or ReLU [12].

The output at each time step can be computed as:

$$\mathbf{y}_t = g(\mathbf{W}_y\mathbf{h}_t + \mathbf{c}),$$

where $\mathbf{W}_y$ is the weight matrix for the output layer, $\mathbf{c}$ is the bias term, and $g$ is the output activation function. This architecture enables the network to generate sequences by predicting the next word based on prior outputs.

In text generation, Recurrent Neural Networks (RNNs) are trained to predict the next token in a sequence. Given a training sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$, the model minimizes the negative log-likelihood loss:

$$\mathcal{L} = -\sum_{t=1}^{T}\log P(\mathbf{x}_t|\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}),$$

where $P(\mathbf{x}_t|\mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$ is the probability of token $\mathbf{x}_t$ conditioned on the previous tokens.

While "vanilla" RNNs are effective for short sequences, they struggle with long-term dependencies due to issues like the vanishing gradient problem. To address this, we employ a Long Short-Term Memory (LSTM) architecture, which incorporates gating mechanisms that allow the model to better retain and utilize relevant information over longer spans of text. This is particularly important for generating stylistically consistent and contextually coherent text, as the model can "remember" information about the author's style and lexical choices over many time steps [13].

Another key aspect of our text generation setup is the representation of textual data at the character level. Instead of working at the word-level, we use a simple `char2int` dictionary mapping each unique character to an integer index. This approach simplifies the handling of out-of-vocabulary terms and makes the model more flexible in capturing nuances such as punctuation and capitalization. Similar character-level modeling approaches are discussed in detail by Raschka et al. [12], who outline best practices for implementing RNN-based language models, including preprocessing steps and hyperparameter considerations.

In summary, by leveraging LSTM's gating mechanisms to handle long sequences and employing a `char2int` representation for fine-grained control over the text input, our text generation model is able to produce outputs that mimic the stylistic characteristics of the original text data.

## III. METHOD

### A. Data collection and Preprocessing

The preprocessing pipeline for this project was designed to transform raw textual data into structured embeddings suitable for machine learning analysis. The following steps outline the key aspects of this process:

#### 1. *Data Collection*

The text data was sourced from Project Gutenberg, a comprehensive repository of public domain texts. Five texts were selected for their distinct linguistic styles:

1. **Pride and Prejudice**: The prime example of the English regency-era romance.
2. **Crime and Punishment**: A dark Russian classic renowned for its psychological depth.
3. **Genesis through Deuteronomy**: A representative of religious writing, with formal and archaic language.
4. **Don Quixote, Volume I**: A satire on the French chivalric literature, with a light, extravagant style.
5. **Heimskringla, Sagas 1-7**: A compilation of Norse sagas with a recounting style and poetic skaldic interludes.

The focus on stylistically divergent texts ensured that the machine learning models were exposed to a broad range of linguistic features as well as providing more distinct cases making the classification easier.

These texts were downloaded in plaintext format. Some text-specific preprocessing was made to remove licenses and forewords before chunking. It is common practice to use fixed-length chunks when doing chunk-wise text analysis with embeddings, since the chunk length influences the embedding. When using fixed-size chunking (with chunks of size 50, 100 and 150), the classification task turned out to be rather simple, and we wanted a more complex and natural problem. Sentence-wise embedding turned out to provide a more challenging testing ground allowing us to differentiate between models to find the best ones.

In the case of sentence embedding, the texts were divided into individual sentences using the `nltk` library [14]. Sentence tokenization ensured that the input units were linguistically coherent, preserving the semantic and syntactic structure of the texts. However, some shorter sentences may be ubiquitous and thus very hard to accurately classify, so we would expect imperfect classification in this case. To counteract this effect, a random choice of 2500 sentences that had a minimum character length of 100 were chosen for each author and made up the total sentence-dataset.

In the case of embedding chunks of text, the texts were divided into chunks of predefined sizes: 50, 100, or 150 words. This chunking strategy allowed for experimentation with context size, balancing the trade-offs between computational efficiency and the amount of stylistic information captured. Unsurprisingly, more context gives better performance in the range we tested, and we expect more complex models to perform better with large chunk sizes. Note also that the transformer time complexity is at least $O(n^2)$, so that longer chunk-sizes are generally slightly more computationally expensive.

#### 2. *Embedding*

Each text chunk or sentence was transformed into a high-dimensional vector representation using the `mixed-bread` embedding model from the `sentence-transformers` library [5]. The embeddings served as the numerical representation of the texts, enabling machine learning algorithms to operate on the data effectively. The embeddings were then labeled according to their author and split into training and test sets.

To get an intuition for the problem at hand, the generated embeddings were visualized in two dimensions using dimensionality reduction techniques such as PCA and UMAP. These initial visualizations, seen in Figure 6, confirmed that embeddings from the same text generally form distinguishable clusters at least in the case of the 150 word chunk embeddings. The UMAP visualizations particularly supported our intuition in the sense that the sentence problem would be more challenging than the chunk classification. For a more in-depth discussion of these visualizations, we refer to the results section.

### B. Classification

The classification process in this project involved both supervised methods with FFNNs, and unsupervised/semi-supervised learning techniques using clustering methods.

### 1. FFNN

The neural network classifier was implemented using `PyTorch`, leveraging its flexibility for defining and training custom architectures [15]. This network was then interfaced with `skorch` to make it compatible with the `scikit-learn` API to allow for usage of their gridsearch implementation [16].

Utilizing this, we used `GridSearchCV` from `scikit-learn` to gauge the accuracy of our model on a validation dataset as a function of different hyperparameters [1]. Due to the computational cost of this procedure, and the fact that all models were very good on the chunk-case, we optimized only for the sentence-case and used these parameters for all datasets. Other model parameters include 50 epochs and 5 folds for the $k$-fold used in cross-validation scoring.

- **Model Architecture**: The neural network consisted of an input layer, one fully connected hidden layer with sigmoid activation, and an output layer using a softmax activation function to produce class probabilities.

- **Hyperparameters**: Training was conducted with a batch size of 64 and a learning rate of 0.01, as determined by a gridsearch. The loss function was cross-entropy, appropriate for multi-class classification, and the optimizer was Adam [17].

- **Training Procedure**: The network was trained for 100 epochs, which was determined to be past a point of diminishing returns and chosen to limit the computation time.

- **Dataset size**: The effect of the size of the training dataset was studied through cross-validation scoring of the model when training only on a percentage of the total training dataset.

- **PCA**: PCA was applied to the embeddings to reduce their dimensionality while preserving key variance. PCA was implemented through the `PCA`-class from `scikit-learn` [1]. The impact of dimensionality reduction on classification accuracy was studied by varying the number of principal components retained, using `cross_val_score` to measure the performance on the training dataset for both embeddings of sentences and chunks of size 150 [1].

- **Application to Unseen Works**: The trained neural net was then used to classify chunks from other works (*Sense and Sensibility*, *The Brothers Karamazov*, *The New Testament of the King James Bible*, *Don Quixiote* Volume II and *Heimskringla* sagas 8-15) by the same authors, to ensure a satisfactory generalizability of the model.

### 2. Clustering

For the unsupervised analysis, clustering methods were applied to the embeddings to identify potential stylistic groupings. Both UMAP and PCA were tested as dimensionality reduction techniques in combination with both $k$-means and HDBSCAN. $k$-means and PCA were implemented using the `KMeans` and `PCA` classes from `scikit-learn`, UMAP from the `umap` library and HDBSCAN from the `hdbscan` library [1–3]. Recall that $k$-means partitions the data into a predefined number of clusters, while HDBSCAN allows for density-based clustering without requiring the number of clusters as an input.

To evaluate clustering performance, the Adjusted Rand Index, in the form of `adjusted_rand_score` from `scikit-learn`, was used [9]. When using PCA, performance was measured only as a function of the number of components of the projection. Confusion matrices were automatically rearranged to get the maximal elements along the diagonals.

UMAP allowed for semi-supervised classification, and since the labeling of data is oftentimes the most expensive part of the training process, it is of great interest to study the model accuracy as a function of the amount of labeled data. Thus, both the percentage of the dataset that were labeled as well as the number of components were tested with this method. To be clear; in the case of partially labeled data, a certain percentage of the training dataset was labeled and then used to transform the *entire* training dataset before clustering was performed.

## C. Generation

The generation of text were implemented using an RNN to model sequential dependencies in the data and produce outputs that mimic the style of the input text. This section describes the methodology for designing and training the text generation model, along with the key experiments conducted.

### 1. RNN

A character-level RNN was implemented using `PyTorch`, designed to predict the next character in a sequence given a context of preceding characters [15]. The RNN employed an LSTM architecture to capture long-range dependencies and mitigate issues such as vanishing gradients, commonly encountered in standard RNNs [18]. This part of the project was based on the second RNN project in Raschka et al. and follow this implementation closely [12]. Key elements of the implementation include:

- **Model Architecture**: The RNN consisted of an encoding layer, an LSTM layer, and a linear output layer. The encoding layer is in our case a simple

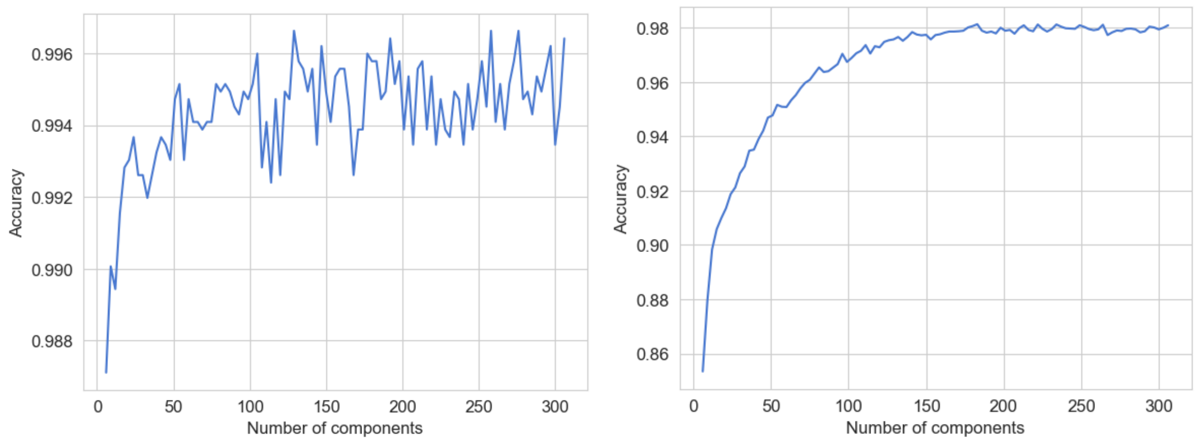

Figure 1. Cross-validation accuracy as a function of the number of components from PCA dimensionality reduction for a chunk size 150 (left) and sentences (right).

`char2int`-dictionary which transforms input characters into integers. These are then processed by the LSTM layer to maintain a hidden state over the sequence. The output layer applied a softmax function to produce a probability distribution over the next possible characters.

- **char2int**: The char2int-dictionary was generated based on the characters found in the given text. The main reason for using this instead of embeddings is that the char2int-dictionary is a bijective correspondence, so that a naive inverse transformation can be used to retrieve a text output from the model's number output. With traditional text embeddings, there is no naive transformation from the continuous vector space into the discrete space of words. Though reverse transformation algorithms do exist to find a "close match" as used in modern LLMs, but since we were not familiar with these, it would have increased computational expense and been rather complex to implement.
- **Hyperparameters**: The hidden state dimensions were tuned experimentally. Training was conducted using a cross-entropy loss function, with the Adam optimizer applied for parameter updates.
- **Training Data**: The training data consisted of character sequences from the selected work of the author the specific model was supposed to emulate. The text was preprocessed as described in the classification section and run through the `char2int`-dictionary for training.

#### 2. *Generation Process*

During text generation, the RNN predicted one character at a time, conditioned on the previously generated characters. The randomness of generation was controlled using a temperature parameter $\alpha$, which modulated the output probability distribution. Higher temperatures, corresponding to lower $\alpha$'s, increased the randomness of the predictions, leading to more creative but less coherent text, while lower temperatures favored more solid outputs. The generation process followed these steps:

1. The RNN was initialized with a random or predefined seed sequence; in practice, we used each of the 100 most common words in the English language to initialize a generation for each author on both sentence and chunk generation.
2. At each step, the model produced a probability distribution over the vocabulary of characters.
3. A character was sampled from this distribution, with the sampling influenced by the temperature parameter.
4. The sampled character was appended to the output sequence and fed back into the model as input for the next step.

#### 3. *Evaluation*

To evaluate the generated text, the neural network classifier was used to assess whether it retained the stylistic features of the source text. The accuracy was measured for both 150 word chunks and sentence embeddings for different temperatures.

## IV. RESULTS

### A. Discriminator

This section will first present the general trends of FFNN and clustering classification with relevant graphs, before comparing the performance of different models exemplified by a series of confusion matrices and visualizations.

### 1. *FFNN*

The neural network classifier's accuracy from cross-validation as a function of dataset size for chunk sizes of 50, 100, and 150 words is presented in Figure 2. For large chunk sizes, the performance were excellent even with little training data, though larger context size gives slightly better performance. With only 50 words, the model required about 15% of the allocated training data before the performance stabilized at 97.5%. These stable results show that the chunk classification task were relatively easy at these chunk sizes with our dataset and text embeddings, which encouraged us to explore the more challenging sentence case to evaluate the effect of dataset size on performance.

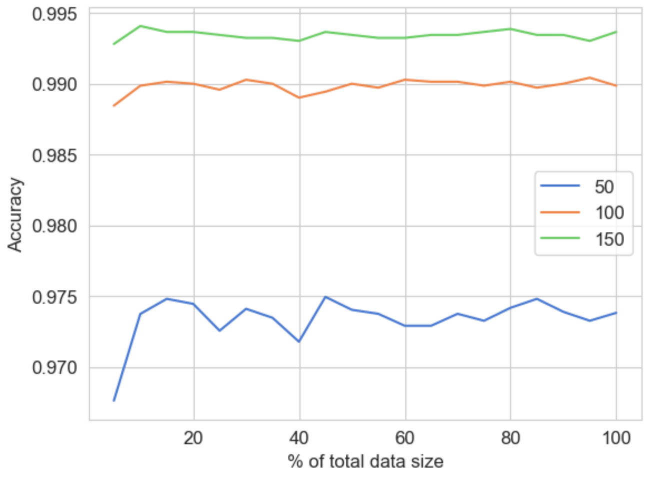

Figure 2. Accuracy as a function of the dataset size as a percentage of the total dataset for chunk size 50, 100, and 150.

Figure 3 shows the classifier's accuracy from cross-validation for the embeddings of sentences as a function of the dataset size. Here, the accuracy increases initially and stabilizes around 94%, indicating good performance that is slightly lower than the chunk-size embeddings. Still, we see that we have a lot of data and that similar performance could be achieved with only 10% of the total training dataset.

The impact of dimensionality reduction with PCA is presented in Figure 1. For 150-word chunks, accuracy increases sharply as more components are added, stabilizing around 99.6% after 50 components. Do note the scaling of the axes; the effect is rather minor. For sentence embeddings, the accuracy curve shows a more gradual increase from around 86% before stabilizing near 98% at about 100 components.

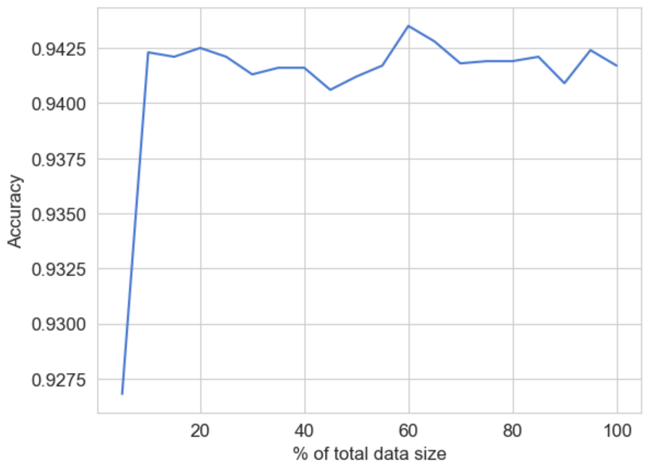

Figure 3. Accuracy as a function of the dataset size for sentence embeddings.

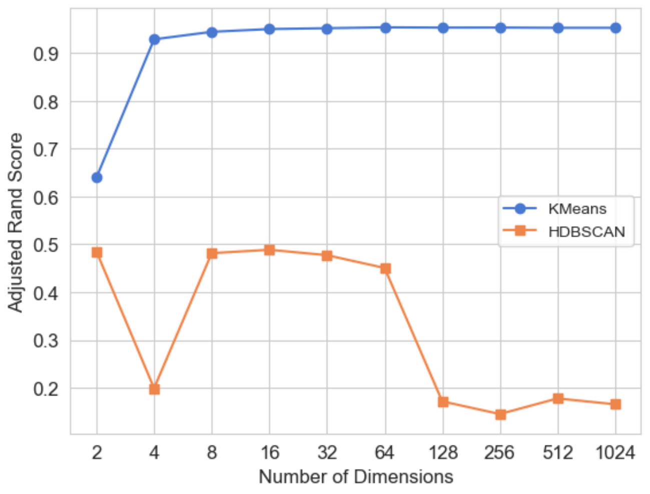

Figure 4. Adjusted rand score for the training set with chunk size 150 when clustering using HDBSCAN or KMeans as a function of the dimensionality of the projection.

### 2. *Clustering*

The adjusted rand score for clustering 150-word chunk embeddings using $k$-means and HDBSCAN is presented in Figure 4. $k$-means achieved a high adjusted rand score, stabilizing near 0.9 after reduction to 16 dimensions. In contrast, HDBSCAN exhibits significant fluctuations, performing only slightly better than random assignment. This is likely due to HDBSCAN leaving most datapoints unclassified in the high dimensional cases; its reliance on dimension reduction techniques with high dimensional data was touched on in the theory section and will be discussed further in the next section.

In Figure 5, we display heatmaps of the Adjusted Rand Score of $k$-means and HDBSCAN with UMAP for a range of dimension sizes and relative labeled dataset sizes on

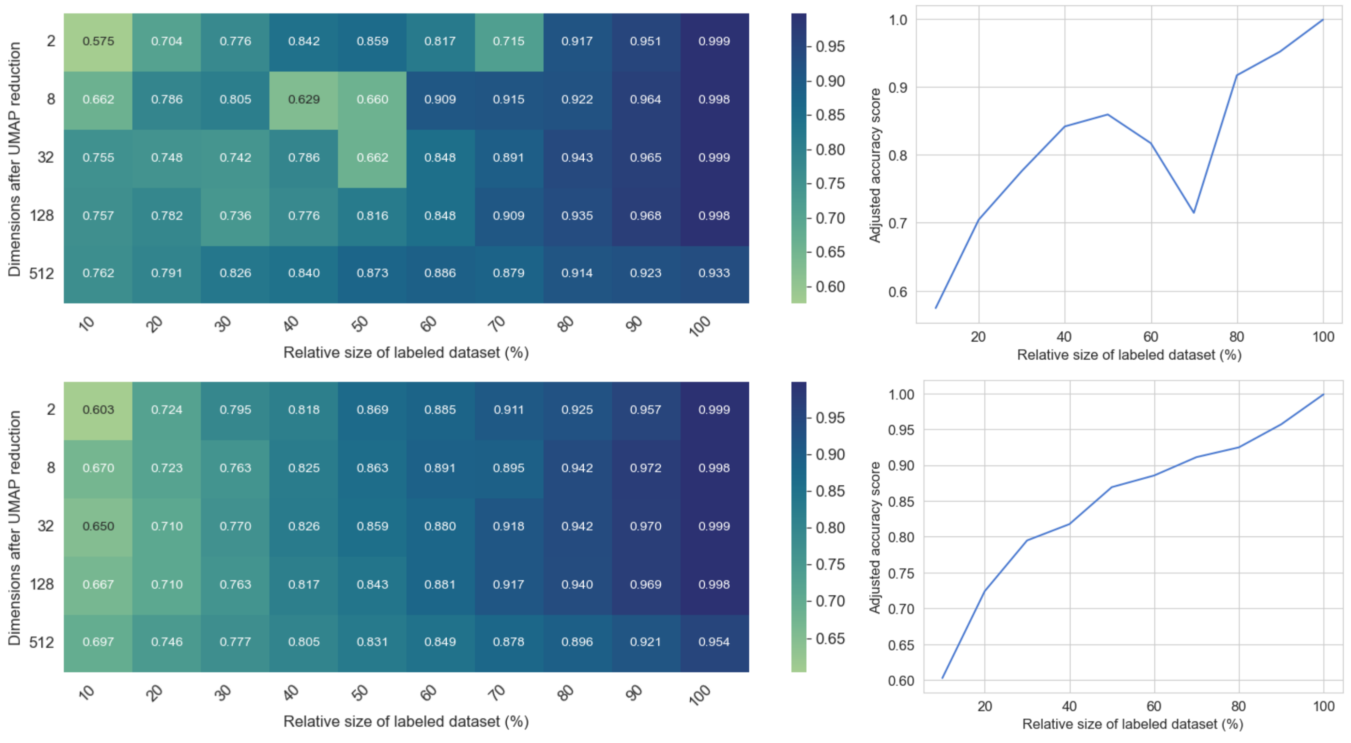

Figure 5. On the left, heatmaps of Adjusted Rand Score of $k$-means (above) and HDBSCAN (below) with UMAP dimension reduction as a function of dimensions and the portion of the dataset which is labeled to aid the reduction, on sentences. On the right, cross sectional plots of the dimension 2 case showing the evolution of performance as more labeled data is provided. $k$-means seems less stable than HDBSCAN when less labeled data is used in the UMAP reduction.

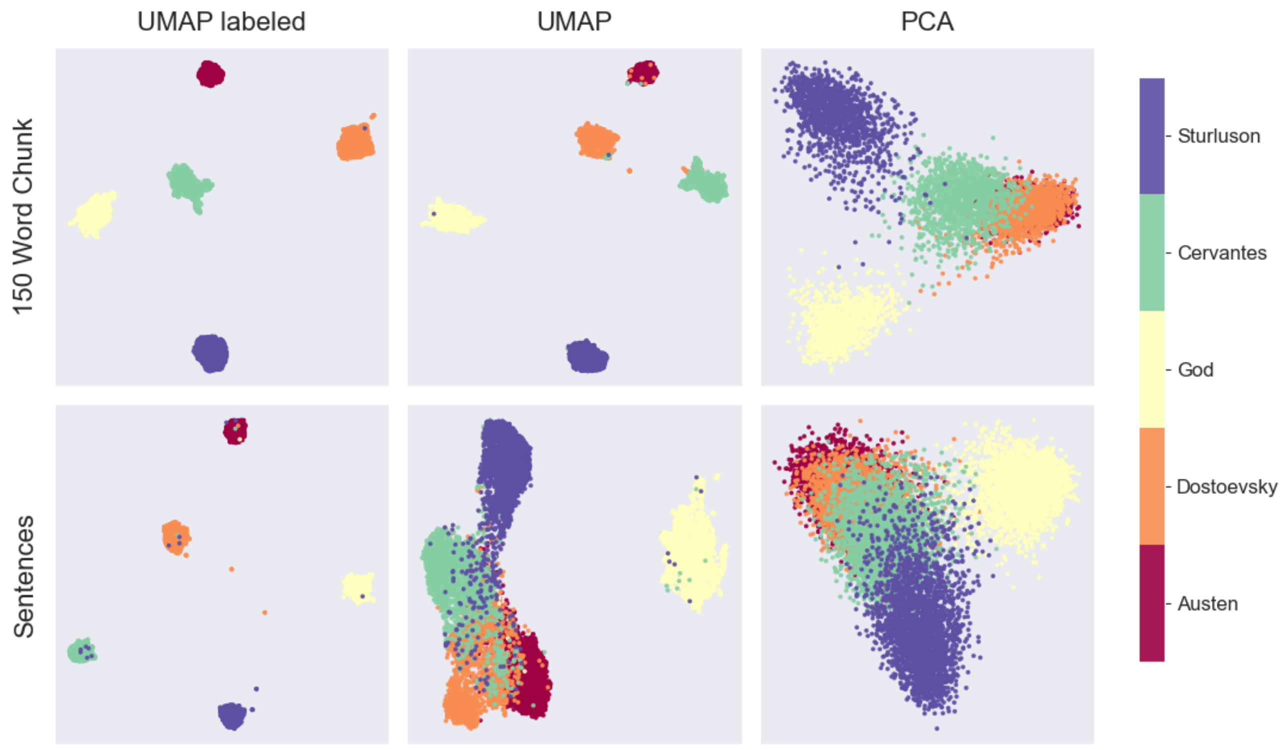

Figure 6. 2D projection of both 150 word chunk and sentence embeddings using UMAP with and without labeling of all data as well as PCA for the dimensionality reduction. Notice that God is very clearly separated out, while Dostoevsky seems especially intermingled with the others.

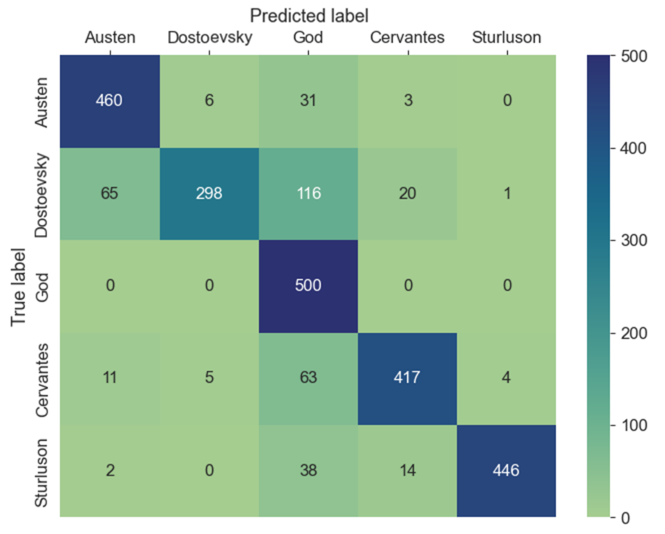

Figure 7. Confusion matrix for sentence classification on test dataset with K-Means with a UMAP reduction to 16 components using labels for 25% of the labeled training data.

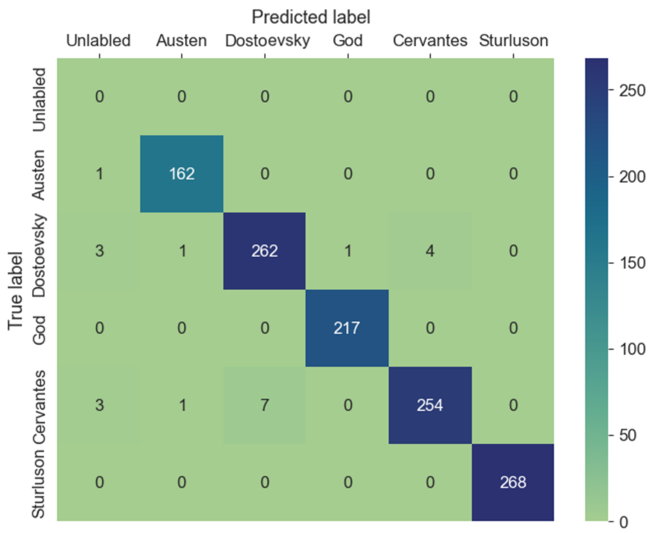

Figure 8. Confusion matrix for chunk classification with 150-chunks on test dataset with HDBSCAN with a UMAP reduction to 16 components using 10% of the labeled training data.

validation data for the sentence embedding case. Notice that HDBSCAN performs very well with UMAP, in stark contrast to the performance with PCA; this is not uncommon and is even noted in the UMAP documentation [2]. In fact, HDBSCAN now shows greater performance and stability than $k$-means, as is especially clear from the accommodating graphs displaying the 2-dimensional case for each method. Notice that the general trend seems nearly linear, indicating that UMAP is data-hungry and that there is no obvious point of diminishing returns before near-perfect performance is achieved.

Figure 6 displays the 2D projections of both 150-word chunk and sentence embeddings using UMAP (with and without labeling) and PCA. The embeddings for God are very clearly separated in all visualizations, while Dostoevsky seems especially intermingled with the other classes, particularly in the sentence embedding space. Interestingly, this will be reflected in the confusion matrices of low dimensional clustering classification.

### *3. Performance on the Test Dataset*

The confusion matrix for sentence classification on the test dataset with $k$-means clustering using UMAP reduction to 16 dimensions and only 25% labeled training data is presented in Figure 7, and shows similar patterns to those we observed in the 2D-visualization: Sentences from the work by God were *all* correctly classified, though at the cost of many false positives. Particularly, there is significant trouble with the text by Dostoevsky. In this low-dimensional case on sentence classification, overclassification of God and underclassification of Dostoevsky are thus the general trends. The accuracy of the model is about 84.8%, which is not great.

In Figure 8, we provide the confusion matrix for the test dataset with 150 word chunks using HDBSCAN and only 10% labeled training data. Even with little data, HDBSCAN achieved strong predictive performance on the chunk problem, with an accuracy of 98.2%. However, the figure illustrates that, as opposed to $k$-means, it did not necessarily classify all data points, allowing some to remain unlabeled.

The confusion matrix for sentence classification with HDBSCAN after UMAP reduction to 128 components using a fully-labeled training dataset is shown in Figure 9. Unlike in Figure 8, in this case there were sufficient data such that no points were left unlabeled by HDBSCAN. Increasing the dimensionality to 128 successfully reduces the overclassification of God, though the model continues to struggle with Dostoevsky. The accuracy is 90.8%, which is decent on the sentence case.

To compare the clustering methods to the FFNN, a comparison of Figure 9 and 10 can be made. Figure 10 shows the FFNN sentence classification results, achieving an accuracy of 93.0%. There are some misclassifications between Austen and Dostoevsky, but performance is high overall, outperforming the best clustering algorithms with a more even performance - note particularly that Dostoevsky is not significantly underclassified. Therefore, we use this model on different, unseen works by our chosen authors and finally, to evaluate our generator.

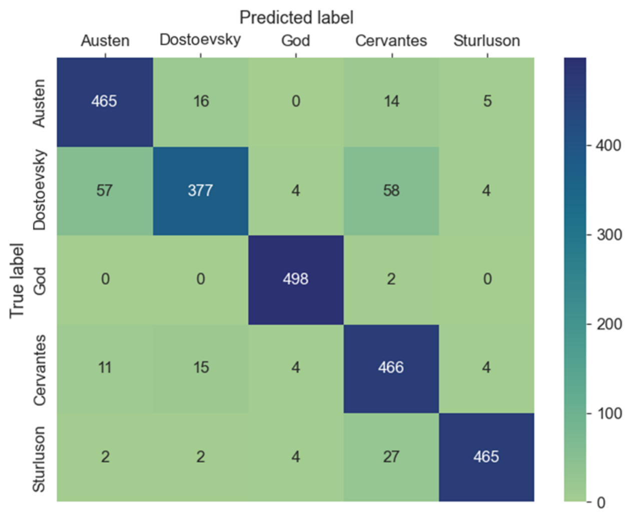

Figure 9. Confusion matrix for sentence classification on test dataset with HDBScan with a UMAP reduction to 128 components using the labels for the entire training data.

### 4. *Generalization to Other Works*

When applied to 150 word chunks from unseen texts by the same authors, the confusion matrix in Figure 11 reveals that the neural network maintains robust classification accuracy, at about 97.9%. For comparison, the performance on 150 word chunks on the test dataset was 96.5%, which is in fact lower; this seems to be because the biblical chunks are more easily classified than others, and a large portion of the unseen chunks happened to be biblical. Even on the other authors, misclassifications remain infrequent, showing the network's ability to generalize effectively to new datasets and particularily that our chosen authors, unsurprisingly, have styles which transcend their individual works.

## B. Generator

The text below shows two excerpts from low-temperature ($\alpha = 2$) generated text created by the RNN trained on the works of Austen and God, respectively. It is a disorienting read, because the grammar is excellent and the sentences seem coherent at a glance, but closer inspection reveals that the content is quite nonsensical. Nonetheless, it should be clear that the generated text indeed captures something of the respective authors writing styles.

God, prompt word "Lord":

> 10:18 And the LORD spake unto Moses, saying, 15:2 Speak unto the children of Israel, saying, If a man shall have an holy convocation; ye shall not see my wife; and thou shalt teach the manner of workmanship: and the priest shall burn them upon the altar

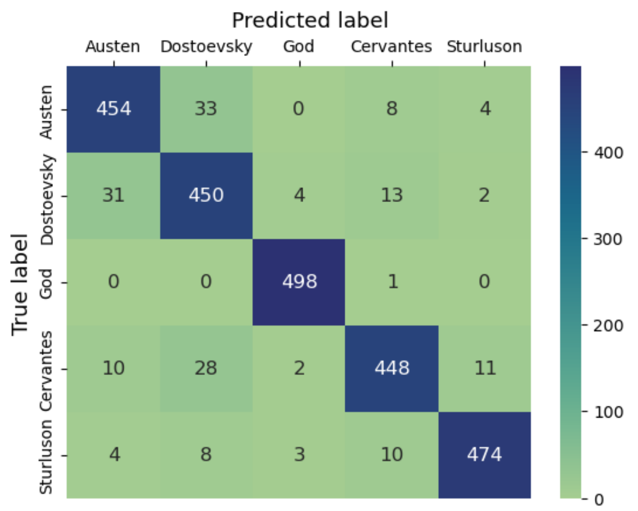

Figure 10. Confusion matrix for optimized neural net on the test dataset when classifying sentence embeddings. This is our final chosen classification model.

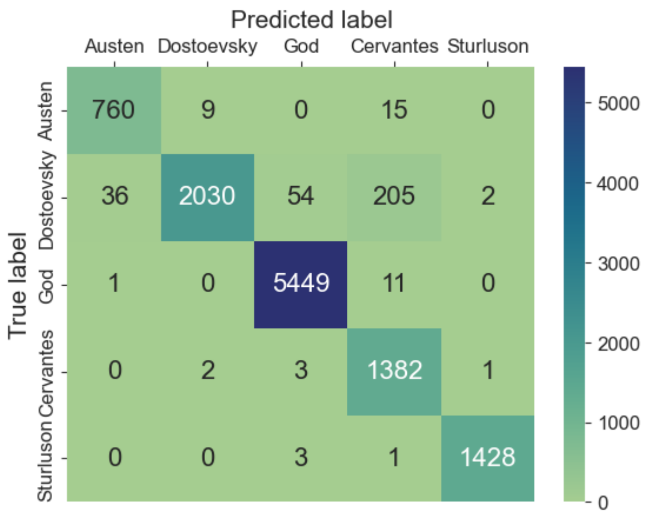

Figure 11. Confusion matrix when using the neural net to classify other works (*Sense and Sensibility*, *The Brothers Karamazov*, *The New Testament of the King James Bible*, *Don Quixiote* Volume II and *Heimskringla* sagas 8-15) by the same authors, with 150 chunks. Here Dostoevsky proves the most challenging, mirroring our experience from clustering.

Austen, prompt word "Darcy":

> letter, and the two gentlemen accompanied by the sisters was a little satisfaction of the party, and he walked away to consider it as the same time who had passed the change was anxious to decide on the coach. I was gone, I shall certainly design. They can be the carriage was only one consequence. But it was a liberal pleasure.

Table I shows the accuracy score for the classification of generated text for both embeddings of sentences and chunk-size 150 for different $\alpha$ values. The accuracy is consistently lower for the sentence embeddings as compared to the chunk embeddings, as could be expected from our previous results. The accuracy decreases as the temperature increases ($\alpha$ decreases), which results in more randomness and the RNN sampling less likely characters in the text generation. Indeed, we find that high-temperature generation leads to grammatical breakdown and hence unintelligible text.

Table I. Accuracy of generator according to the neural net classifier for different scaling values $\alpha$. Note that the scaling parameter is inversely proportional to the temperature.

| **Accuracy** | **Chunk** | **Sentences** |
|---:|---:|---:|
| $\alpha = 2$ | 0.992 | 0.804 |
| $\alpha = 1$ | 0.992 | 0.756 |
| $\alpha = 0.5$ | 0.946 | 0.61 |
| $\alpha = 0.25$ | 0.402 | 0.336 |

## V. DISCUSSION

### A. Classification

#### 1. *Dimension Reduction and Limited Labeling*

Using PCA for dimensionality reduction did not improve the FFNN's accuracy, but it allowed for comparable performance with reduced computational costs as seen in Figure 1. In the case of clustering, however, dimension reduction generally improves performance. Particularly, assisting the unsupervised algorithms with UMAP dimension reduction with some labeled data has a major impact, giving nearly linear improvement as the amount of labeled data increases. In this case, we find that HDBSCAN outperforms $k$-means in stability and slightly in performance, as illustrated in Figure 5.

However, with PCA, $k$-means achieves near-perfect performance and dramatically outperforms HDBSCAN. In fact, Figure 4 shows that HDBSCAN performs only slighly better than random assignment with PCA, particularly for high dimensionalities. This is due to HDBSCAN leaving most data unlabeled, which is a known problem for high dimensional datasets using PCA with HDBSCAN [2]. One may think of this as a consequence of the curse of dimensionality, since the high dimensional space is very sparse and hence makes obvious clusters unlikely. This underlines the importance of dimension reduction with clustering algorithms.

The linear improvement with more labeled data for clustering indicates that, at least in the specific case of sentence classification, there is no obvious point of diminishing returns when it comes to labeling, so that one should set an initial threshold of desired performance and do as much labeling as possible until this threshold is reached, if that proves feasible. Our hope that one would be able to use the unsupervised clustering algorithms to compensate for a lack of labeled data with a lot of unlabeled data therefore seems to be a false one. Since Figure 3 demonstrates that the performance of the FFNN remains high even with about 10% of the training data, it seems that for our sentence classification one is better off using a FFNN on the labeled data available than using clustering with the additional unlabeled data.

#### 2. *Text Embeddings*

Embedding-based approaches are powerful due to their ability to capture both local and global linguistic features. In our study, we observed that the embeddings of 150 word chunks are more easily classified than smaller context sizes, supporting the notion that they are able to encode sufficient information, which aligns with our expectations.

However, one should note that the embedding model we use is likely primarily trained on contemporary corpora, and the application to older or poetic styles like those found in the *Heimskringla* or the *King James Bible* could potentially give worse performance. We did not deem this a likely problem nor a performance bottleneck considering other limitations like computational resources and simple classification models.

#### 3. *Text Analysis*

When applying the final FFNN classifier to unseen works by the same authors, we get excellent results as seen in Figure 11. It is not trivial that this generalization should work so well, since an author may not display consistent style across works. However, with our chosen unseen works, it seemed very intuitively possible to make a successful classification, given that we have achieved a stable classifier and avoided excessive overfitting.

One key task-specific takeaway is the impact of context size in text analysis. The FFNN results clearly show that 150-word chunks outperform smaller chunk sizes, with accuracy steadily increasing as context size grows as seen in Figure 2. From a human perspective, this is reasonable: If given an entire chapter of one of these books, anyone could guess the author, but with only a single sentence, it would oftentimes be pretty hard. Though it seems likely that the embeddings and our simple classifiers will at some point be unable to make use of additional context due to their limited complexity, this is very far from a limiting factor in our case. Instead, the human intuition applies, as illustrated directly in Figure 2, but also in the overlap observed in the clustering visualizations of Figure 6 and the higher rate of misclassifications in confusion matrices on the sentence case than on chunks.

### B. Generation

The importance of context size may also apply to generation. While our RNN works well (given low temperature) on the word-level - that is, it has excellent grammar and spelling and uses almost exclusively real words with meaning - and oftentimes will link together a couple of words which fit together, it does not have sufficient short term memory to create entire sentences which make sense. It is probably possible to improve the LSTM-algorithm or use more computational resources to improve the context size in the generation, though the more obvious improvement is to generate token by token instead of character by character using text embeddings rather than our `char2int` solution.

A smaller potential drawback with our `char2int` dictionary implementation is that the dictionary is generated based on each text. This means that it would be impossible, even in principle, for the model trained to emulate Austen to generate a character which is not in *Pride and Prejudice*. This is unlikely to make the model worse in the eyes of any human, but it might make some room for 'cheating' - essentially overfitting - by the classifiers in evaluation, if certain characters are common in one text and not present in another. This could, in theory, slightly inflate our evaluation results. In a similar vain, the capitalized 'LORD' is a dead giveaway to any human that the text in question belongs to God. However, since our classifiers generalize well, it is very unlikely that they *rely* on such random features.

Overall, the classifier deems the generator very successful on chunk generation, with a 99.2% hit rate with low temperature, as seen in table I. This is better than the test accuracy for the classifiers, indicating that the generator and classifier picked up on similar trends so that the generated text is typical for each author. Notice that higher temperatures ($\alpha = 0.5$) also give decent results in the chunk case. On sentences, the results are much worse with a best accuracy of about 80%. Since our classifiers did not show optimal performance on the sentence case either, the results here are slightly dubious: Particularly, every sentence generated was capped at a set word count, so the information relating to sentence length, which could important since the sentences in *Don Quixote* by Cervantes are particularly long, was stripped away.

### C. Future Work

Future work may use text embeddings for the generation in place of our `char2int`-solution. This requires finding an efficient and accurate inverse algorithm from the continuous embedding space into the discrete space of word tokens. While seemingly difficult, there are surely pre-existing solutions which can be utilized. Another natural extension of our work is to attempt to link a discriminator and generator together to create a GAN.

## VI. CONCLUSION

We find that FFNNs achieve high accuracy when classifying the author of fixed-size text chunks from books. In particular, 150-word chunks led to excellent performance with a 96.5% accuracy on the test dataset, as well as 97.9% on previously unseen data in the form of other works by the same authors. The performance for the sentence embeddings was slightly poorer, scoring a 93% accuracy on the test dataset. The FFNN benefited from greater contextual information, and smaller chunks proved more difficult with sentence-level classification being the most challenging. Still, both FFNNs and clustering methods like $k$-means and HDBSCAN, enhanced by UMAP dimensionality reduction, achieved reasonable accuracy with sufficient labeled data.

Dimension reduction showed promise for reducing the computational cost, as the dimensionality of the input for the FFNN could be drastically reduced while still retaining satisfactory performance. In the case of the clustering methods, dimensionality reduction, especially UMAP, proved very beneficial.

Finally, our text generator, based on an LSTM-based RNN with a `char2int` dictionary, successfully produced stylistically plausible text for all five authors, both to humans and machine. Our neural net classifiers gave the generator an accuracy of 99.2% for chunk embeddings at low temperatures ($\alpha \geq 1$). Still, the produced text was nonsensical on the sentence level, indicating that our RNN did not accommodate sufficient context memory to connect ideas between sentences or across multiple words.

Substituting the `char2int` dictionary with text embeddings, such as were utilized for classification in the generation case is likely to improve context immediately, though it went beyond the scope of our work. A synergy between generation and classification methods suggests promise for future work, such as refining the generative model or integrating fully adversarial training frameworks to push stylistic imitation further. A simpler expansion of our work would be the investigation of more similar authors, expanding the range of literary styles or applying our classifiers to ChatGPT-generated text to assist teachers in determining what parts of a student text might be AI-generated.

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011).

[2] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," (2020), arXiv:1802.03426 [stat.ML].

[3] C. Malzer and M. Baum, in *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)* (IEEE, 2020) p. 223–228.

[4] Z. Harris, Word **10**, 146 (1954).

[5] H. Face, "mxbai-embed-large-v1," (2024), accessed: 2024-11-25.

[6] I. T. Jolliffe, *Principal Component Analysis* (Springer, 2002).

[7] L. McInnes, J. Healy, and J. Melville, arXiv preprint arXiv:1802.03426 (2018).

[8] S. Lloyd, IEEE Transactions on Information Theory **28**, 129 (1982).

[9] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, Journal of Intelligent Information Systems **17**, 107 (2001).

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).

[11] J. J. Telle, M. K. Øvrebø, and M. Torsheim, "Project 2 FYS-STK3155," .

[12] S. Raschka, Y. Liu, and V. Mirjalili, *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python* (Packt Publishing, 2022).

[13] S. Hochreiter and J. Schmidhuber, Neural Computation **9**, 1735 (1997).

[14] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit* (O'Reilly Media, Inc., 2009).

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," (2019), arXiv:1912.01703 [cs.LG].

[16] M. Tietz, T. J. Fan, D. Nouri, B. Bossan, and skorch Developers, *skorch: A scikit-learn compatible neural network library that wraps PyTorch* (2017).

[17] D. P. Kingma and J. Ba, arXiv preprint arXiv:1412.6980 (2014).

[18] S. Hochreiter and J. Schmidhuber, Neural computation **9**, 1735 (1997).