Software Engineering Department

ORT Braude College

Capstone Project Phase B – 61999

The Sentimental of Shakespeare

A BERT-Based Analysis Using Transfer Learning

23-2-R-9

Dorin Hila Bachar
Dorin.hila.bachar@e.braude.ac.il

Moshe Moalem

Moshe.Moalem@e.braude.ac.il

Supervisors:
Prof. Zeev Volkovich
Dr. Renata Avros

Table Of Contents

Abstract

This research introduces a novel approach to Shakespeare's authorship question by leveraging sentiment analysis to examine small, tweet-like segments of text. Sentiment analysis, a critical aspect of Natural Language Processing (NLP), has seen significant advancements since the introduction of deep learning models. Utilizing the Bidirectional Encoder Representations from Transformers (BERT), this study explores how advanced deep learning techniques can enhance our understanding of literary texts. The methodology involves iterative fine-tuning of a pre-trained BERT model, tailored specifically to grasp the nuances of Shakespearean English, and classify segments effectively. The results, derived from a series of numerical experiments, confirm the model's capability to accurately interpret these texts, demonstrating its potential to shed new light on authorship and stylistic attribution in literary studies.

Keywords: Natural Language Processing (NLP); Deep Learning; Sentiment Analysis; Transformers; Bidirectional Encoder Representations from Transformers (BERT); Shakespeare Authorship.

1. Introduction

Sentiment analysis, a subfield in the realm of Natural Language Processing (NLP), has been the focus of numerous studies in the past years. This process, which seeks to understand emotions through the body of text, has been approached through a variety of machine learning and deep learning methodologies. The goal of sentiment analysis is to comprehend an individual's sentiment or emotions, a task that can be accomplished by extracting hidden features from a text, constructing a model to classify these features, and subsequently assessing the model's performance. This technique is used worldwide in a multitude of domains, including the classification of movie reviews, online product reviews, and social media posts.

The scope of sentiment analysis research is not limited to a single language or domain but in fact, it spans various languages and fields, with models capable of distinguishing between positive and negative sentiments in restaurant reviews, amazon product reviews, and even conducting sentiment analysis on Twitter data in different languages.

The advent of pre-trained language models has revolutionized sentiment analysis. Models such as Elmo, GPT, and BERT have been used on large quantities of unlabeled data to get a deeper understanding of language. Among these models, BERT received significant attention due to its unique bidirectional attention mechanism.

This practical part aims to contribute to the ongoing discourse on sentiment analysis by investigating the effectiveness of fine-tuning the BERT model for sentiment analysis tasks. We propose the Shakespeare-BERT model, which extends BERT's capabilities by incorporating novel techniques for feature extraction and anomaly detection. By leveraging the strengths of BERT's pre-trained representations and integrating additional layers for sentiment analysis, our model aims to achieve superior performance in identifying sentiment polarity and detecting anomalies in textual data.

2. Related Work

In the realm of textual data analysis, sentiment analysis and anomaly detection are interconnected disciplines that have evolved significantly with advances in machine learning technologies. Sentiment analysis, which seeks to decipher the emotional context embedded within text, serves as a foundational layer for enhancing anomaly detection.

Initially, sentiment analysis methodologies focused on extracting pertinent features from text, such as word frequency and sentiment scores, to construct classification models. These models were essential for interpreting the emotional undertones in various contexts like movie reviews, product feedback, and social media content. The effectiveness of these models in accurately identifying sentiments laid the groundwork for more sophisticated applications in anomaly detection.

As sentiment analysis techniques became more refined, leveraging machine learning and later, deep learning approaches, the accuracy and depth of sentiment interpretation improved. This progress directly influenced the capabilities in anomaly detection. Traditionally, anomaly detection in textual data utilized statistical metrics like term frequency and document clustering to identify outliers. However, with enhanced sentiment analysis, anomalies could be detected not just based on the presence of unusual words, but also on subtle shifts in sentiment that deviate from the norm, offering a more nuanced approach to identifying irregularities.

The adoption of deep learning, particularly through the development of autoencoders as introduced by Schlegl et al. [4], revolutionized both fields. In sentiment analysis, deep learning models could capture complex emotional nuances across large datasets, improving the granularity of sentiment detection. When applied to anomaly detection, these models facilitated a deeper understanding of text by examining deviations in reconstruction errors, thus identifying anomalies based on both content and emotional incongruence.

The introduction of transformer-based models such as BERT, and its variant ALBERT that was proposed by Lan et al [2], further demonstrated this synergy. These models have demonstrated exceptional effectiveness in a wide spectrum of NLP tasks due to their deep contextual understanding [1]. Their ability to capture intricate linguistic nuances and contextual dependencies has significantly advanced various natural language processing applications. For anomaly detection, this means an enhanced ability to detect not only overt anomalies but also those subtle in nature, such as unusual sentiment expressions in texts where deviations might indicate potential fraud, deception, or emergent negative sentiment trends.

3. Background

This section is dedicated to describing the theory, the mathematical foundations, and the history of our model.

3.1 Neural Network

A neural network is a type of machine learning model that simulates the human brain's structure and attempts to teach it in the same way. Neural networks consist of multiple layers of connected nodes which are called neurons that process the input to generate output. The data is received from the Dendrite, the data processed in the Nucleus, and after that through the Axon to other neurons. In the neural network, each node is a neuron, and each edge is a connection between 2 neurons. The goal of a neural network is to learn from data and generalize the learning to make predictions, accurate decisions, problem-solving, and classification. By iteratively adjusting the weights and biases of the network during the training process, the neural network aims to optimize its performance and improve its ability to solve the task.

3.1.1 Input layer

The neural network receives input data, which is usually a set of numerical values that represent some features of the data.

3.1.2 Weights

Each edge in the network has a weight, which determines how strongly it responds to the input data.

3.1.3 Vanishing Gradient

In deep neural networks especially with recurrent connections or many layers, there is a problem called the vanishing gradient that challenges the training process of the model. The problem is that in the process of training the neural network using a gradient-based optimization algorithm such as gradient descent, the gradients of the loss function with respect to the parameters are computed and used to update the weights. The gradients are multiplied as they propagate back through the many layers so that each layer presents its own set of weights. When gradients are multiplied layer by layer, they can become exponentially small. As a result, the early layers receive very small gradients, leading to slow or no learning in those layers.

This can be seen in the graph of the sigmoid function and its derivative. For very large values for the sigmoid function, the derivative takes a very low value.

Several techniques have been developed to solve the vanishing gradient problem: activation functions, batch normalization, weight initialization, gradient clipping, and improving models by changing architecture.

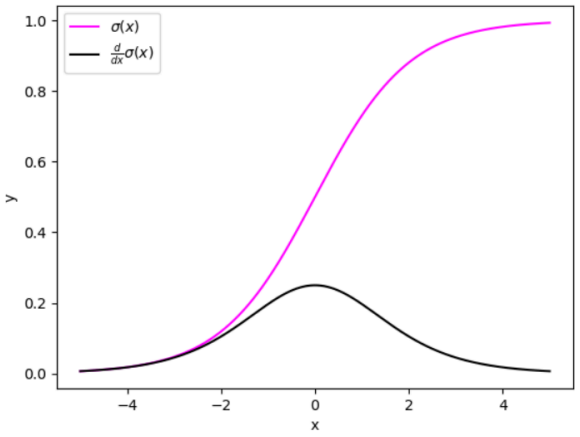


Figure 1, The Sigmoid activation function and its derivative. [source]

3.1.3.1 Activation Function

In a neural network, each neuron receives incoming data which is multiplied by the respective weights of its connections. These products are then aggregated, resulting in what is known as the 'weighted sum'. This sum is processed through an activation function which introduces non-linearity and determines the neuron's output. If the output of the activation function crosses a certain threshold, the data is transmitted to the following neuron.

The activation function plays a crucial role in addressing the vanishing gradient problem, which is a common issue in training deep neural networks. This problem arises when gradients approach zero, causing them to 'vanish' as they propagate backwards through the network during training. To mitigate this issue, certain activation functions are used that help maintain a non-zero gradient for positive inputs, thereby alleviating the impact of the vanishing gradient problem.

Sigmoid

The sigmoid is also known as a logistic function that maps an input value to a value between 0 and 1. The function has an S-shaped curve and it's symmetric around midpoint x=0. When the input value is very large the function will approach the value 1 and for a very low value, the function will approach the value 0. For input values near zero, the output of the sigmoid function is close to 0.5.

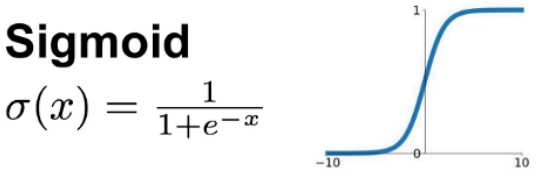


Figure 2, The Sigmoid function. [source]

tanh

A tanh function, also called the hyperbolic tangent function, is a mathematical function that maps a value between -1 and 1 and it's symmetric around the origin x=0. When the input value is very large the function will approach the value 1 and for a very low value, the

function will approach the value -1. For input values near zero, the output of the tanh function is close to 0.

There are situations where the tanh function offers an advantage over the sigmoid function. This is largely because tanh is zero-centered, implying its average output gravitates around the value of 0.

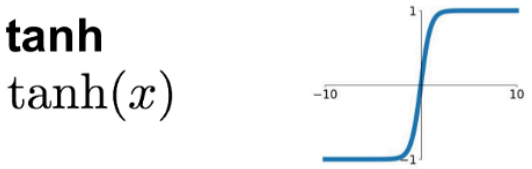


Figure 3, The Hyperbolic Tangent function. [source]

ReLU

The Rectified Linear Unit (ReLU) function returns a maximum between zero and the input value. If the input is positive, it outputs the value directly, else for negative values the output will be 0.

One of the main advantages of the ReLU function is that the function allows the model to learn more quickly and effectively by allowing the gradient to flow easier for positive inputs, as the derivative of ReLU is 1 for positive inputs. This can lead to a faster and more stable convergence during training. Yet, the function has a problem called "ReLU dying". ReLU dying is the condition where neurons become permanently inactive and output zero, due to a large negative bias or a bad initialization that no longer contributes to the learning process. This issue can be addressed by using Leaky ReLU or ELU.

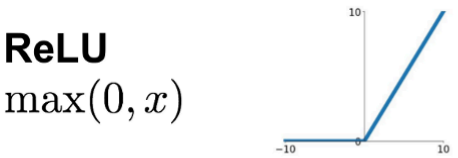


Figure 4, The Rel-U function. [source]

Leaky ReLU

Leaky ReLU, a variant of the ReLU activation function, offers a solution to the 'dying ReLU' problem. Like its parent function, Leaky ReLU outputs the input directly for positive values. However, for negative inputs, it returns the product of the input and a small constant 'a', which is typically set to a minor value such as 0.01 or 0.2. This adjustment ensures a non-zero output for negative inputs, thereby helping to prevent the ReLU neurons from 'dying'.

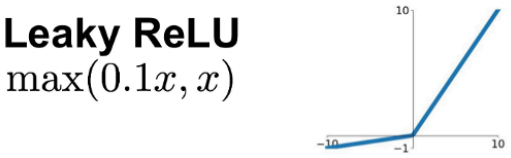


Figure 5, The Leaky Rel-U function. [source]

ELU

ELU (Exponential Linear Unit) is another variant of ReLU activation that addresses the limitations of both ReLU and Leaky ReLU. In addition to the non-zero slope for negative inputs, ELU also incorporates an exponential component, so that there is a smooth transition from negative to positive inputs.

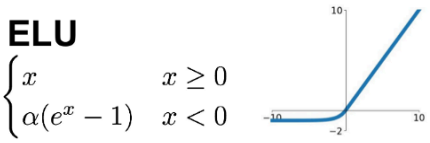


Figure 6, The ELU function. [source]

3.1.3.2 Batch Normalization

This technique stabilizes the distribution of input data fed into each layer of a neural network by normalizing it. The normalization process involves subtracting the mean of the input data and dividing it by its standard deviation. This leads to a reduced internal covariate shift, ensuring the inputs remain stable across various layers.

3.1.3.3 Weight Initialization

A controlled initialization of the weights in the network can alleviate the vanishing gradient problem. Initialization the weights so that gradients will be neither too small nor too large.

3.1.3.4 Gradient Clipping

Clipping the gradients to a predefined threshold prevents the gradients during training from becoming too large or too small. As a result, the network learns and updates the weights more efficiently.

3.1.4 Output Layer

The output layer will produce the prediction or output of the network's problem solution after the weights and activation function is applied. The weights and activation function act as a filter, allowing only data that meets certain criteria to pass through. The output layer then produces the desired output of the neural network.

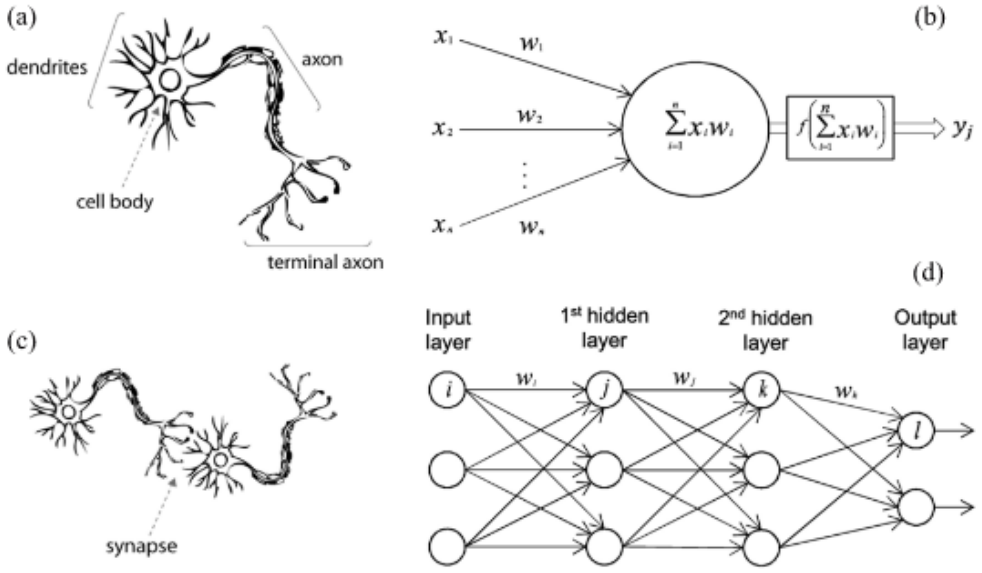


Figure 7, Neural network models simulate the structure of neurons in the human brain. [source]

3.2 CNN- Convolutional Neural Network

CNN is a type of neural network that is particularly effective in data processing such as: extracting features from text or images, object detection, and classifying. CNN uses particular layers: convolutional, max pooling, and fully connected to learn and extract relevant features from the input data.

3.2.1 Convolutional Layers

CNNs use convolutional layers to scan the input data for patterns or features. These layers apply a set of filters also called kernels to the input data, which identify specific features.

The kernel is a small matrix of weights that slides over the input and performs element multiplication with the corresponding elements of the input and sums the results. There are often multiple kernels in a CNN's convolutional layer. Each kernel focuses on detecting a different feature and each kernel output forms a separate feature map. During the CNN training process, the kernel weights are learned.

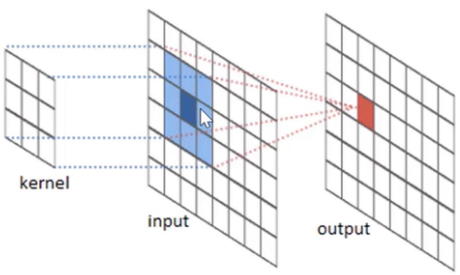


Figure 8, The operation of the convolution layer. [source]

3.2.2 Pooling Layers

Max pooling works by taking the maximum value from each of the sections of the feature map, while average pooling takes the average of the values in each of the sections. This helps to reduce the computational complexity of the model, while still preserving the most important information from input.

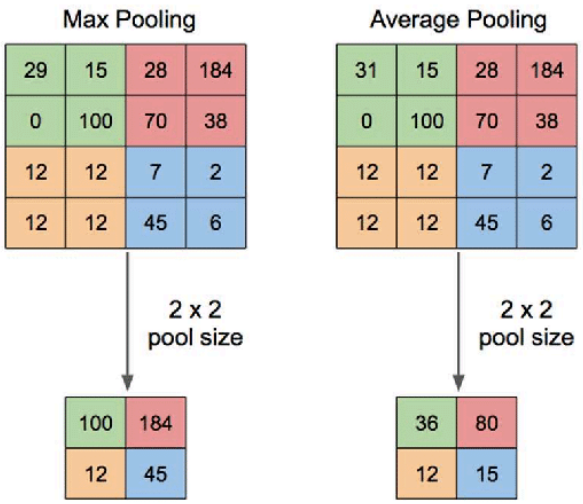


Figure 9, The operation of the polling layers. [source]

3.2.3 Fully Connected Layers

A final classification or regression layer is typically added to the network at the end of it to perform the final task, taking the high-level features extracted from the previous layers and mapping them to the desired output format, such as class probabilities. The purpose of the training is to find the optimal values of the kernel weights that allow the network to learn and detect meaningful features in the input data that are relevant to the specific task.

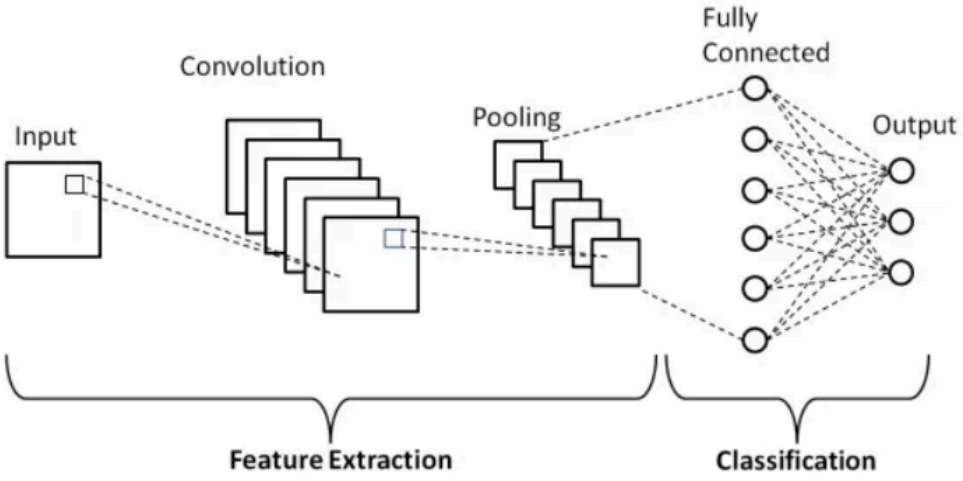


Figure 10, The architecture of CNN. [source]

3.2.4 Training of CNN

Training CNN involves the process of optimizing the network's parameters such as the weights for the purposes of learning from the training data and improving its performance on a specific task. Here are the steps in CNN training.

Initialization network

This step provides an initial starting point for the network to begin learning. Random values initialize the weights and biases of the CNN.

Forward Propagation

This step performs forward propagation through the network. In this process, the input data is fed through layers of the network, and then convolutions, pooling, activation functions and other operations are applied to produce predicted outputs.

Loss Function Calculation

This step compares the predicted output with the actual output from the data using a suitable loss function. Loss functions estimate the error between predicted and actual outputs.

Accuracy Calculation and Output

After each iteration of the training process, the accuracy of the model is calculated using a separate validation dataset or, in some cases, the training dataset itself. This accuracy metric evaluates how well the model performs on the given task by comparing its predictions to the actual labels or targets.

Validation Dataset

A subset of the dataset that is not used for training, known as the validation dataset, is employed to evaluate the model's performance during training. This dataset helps to assess how well the model generalizes to unseen data and prevents overfitting.

Backpropagation

The purpose of the backpropagation process is to calculate the gradient of the loss function in relation to the weights and biases of the network and then update the weights in the opposite direction of the gradient to minimize the loss.

Iteration

Repeat the training steps with multiple iterations until you reach the desired level of accuracy. Each iteration helps the network adjust its parameters based on the gradients and get closer to the optimal values that minimize the loss. The goal is to simultaneously maximize accuracy and minimize the loss function.

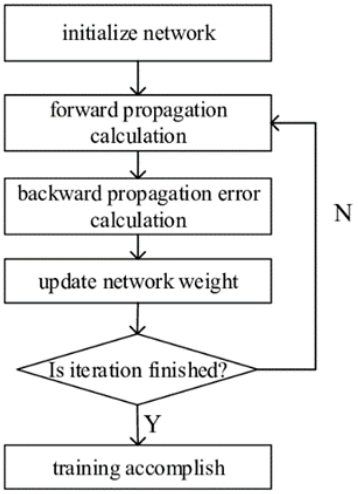


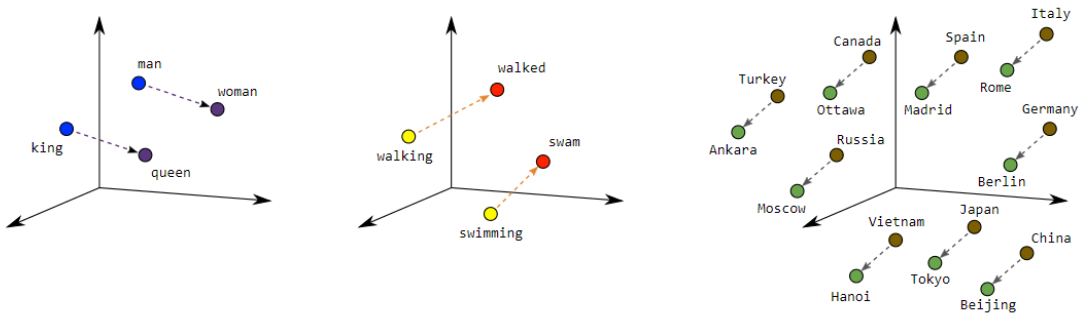
Figure 11, The stages of the training process of the CNN. [source]

3.3 Word Embedding

A crucial aspect of Natural Language Processing (NLP) that involves the representation of words or phrases in a numerical format mostly as a high-dimensional vector space. This technique allows algorithms to capture the semantic and syntactic relationships between words, and so enables more effective processing of text data.

Traditional methods of text representation such as Bag-of-Words (BoW) [5] represent words as discrete symbols, while those methods are simple and effective for certain tasks, they are failing to capture the semantic relationships between words.

After the words are represented as a vector there is a way to find similarities in the words, similar words will be positioned close to each other in space, while unrelated words are positioned far apart, this is what allows algorithms to understand the semantic similarity between words. There are several popular methods for generating word embeddings, including Word2Vec, Global Vectors for Word Representation (GloVe), and FastText. These methods use different algorithms to learn the vector representations of words based on their context in the text.



Male-Female Verb Tense Country-Capital

Figure 12, “ The king is to the queen as man is to woman encoded in the vector space as well as the verb Tense and Country, and their capitals are encoded in low dimensional space preserving the semantic relationships.”[7]. [source]

3.4 Transformers

The Transformer model, introduced in the paper "Attention is All You Need" by Vaswani et al, is a type of deep learning model that has revolutionized the field of Natural Language Processing (NLP). Unlike traditional recurrent neural network (RNN) models, which process sequence data in a linear manner, the Transformer model processes the entire sequence simultaneously, making it highly parallelizable and efficient for large-scale NLP tasks.

3.4.1 Transformers Architecture

Positional Encoding is important in the transformer architecture, as mentioned the transformer work is parallel, to keep the order and the connection between the input words the model uses positional encoding, which injects information about the relative or absolute position of the words in the sequence into the model, the positional encodings are added to the input embeddings before they are fed into the encoder or decoder.

Encoder is the First major component of the Transformer model, it is responsible for processing the input sequence and producing a sequence of continuous representations that capture the contextual information of each word in the sequence. The encoder consists of a stack of identical layers, each containing two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The input for the first layer is the input sequence after the positional encoding, the rest layers input is the output of the previous layer. Each sub-layer also includes a residual connection and a layer normalization operation.

Self-Attention also known as scaled dot-product attention, is the core component of the transformer model, It allows the model to weigh the relevance of each word in the sequence for the computation of the representation of a given word, in other words, it allows the model to focus on the most relevant parts of the input sequence when generating the output sequence. The self-attention mechanism computes a weighted sum of input values (values) based on their relevance to each query. The relevance is determined by computing the dot product of the query with all keys, followed by a softmax operation. The queries, keys, and values are all derived from the input sequence by applying linear transformations.

Multi-Head Self-Attention Mechanism applied multiple times in parallel (that is why it is called "multi-head") with different learned linear transformations of the input sequence. This is what allows the model to capture different types of relationships between words.

Position-Wise Fully Connected Feed-Forward Network applies a linear transformation followed by a non-linear activation function (ReLU) and another linear transformation to each word in the sequence independently, this allows the model to capture complex patterns in the data.

Decoder is the second major component of the transformer model, it is responsible for generating the output sequence based on the representations produced by the encoder. The first sublayer in each decoder layer is a **masked multi-head self-attention** mechanism, which is a similar concept to the self-attention mechanism in the encoder, but with an added "mask" that prevents it from attending to future words in the sequence.

Later, the decoder is just like the encoder, it consists of a stack of identical layers, however, each layer in the decoder contains an additional sub-layer that performs multi-head attention over the output of the encoder stack.

This ensures that the prediction for each word only depends on the previous words. This allows the decoder to focus on the relevant parts of the input sequence when generating each word in the output sequence.

Applications of transformers are diverse and include machine translation, text summarization, sentiment analysis, and question answering. Additionally, transformers are used as the foundation for many state-of-the-art NLP models, including BERT (Bidirectional Encoder Representations from Transformers), and GPT (Generative Pretrained Transformer).

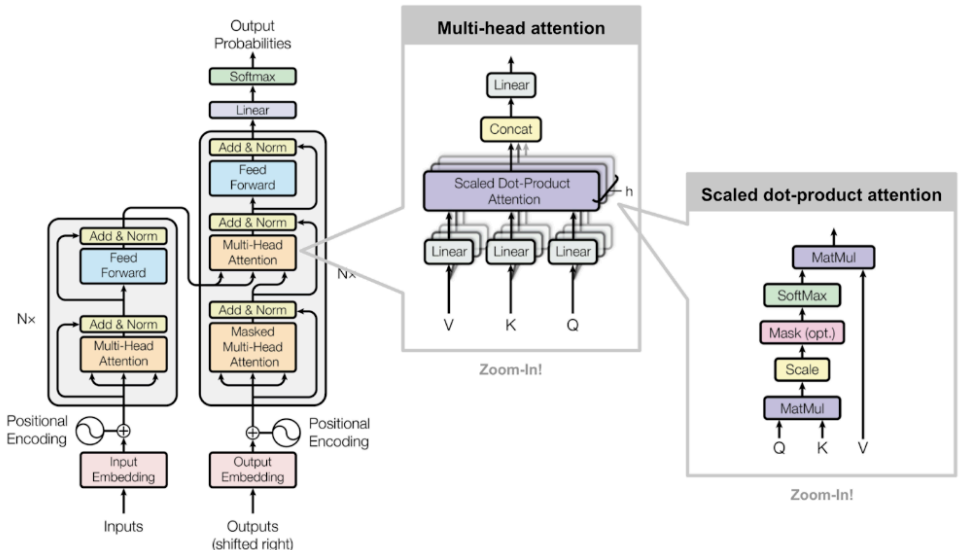


Figure 13, The Transformer - model architecture. [source]

3.5 BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a groundbreaking model in the field of Natural Language Processing (NLP) that has been introduced by researchers at Google in 2018. Ever since then, BERT has significantly improved the state-of-the-art performance on a wide range of NLP tasks. BERT is based solely on the encoder part in the Transformer model. The uniqueness of BERT is its bidirectional training approach, while traditional language models are either trained left-to-right or right-to-left, BERT breaks from this convention and is trained to read in both directions. This bidirectional understanding allows BERT to grasp the context of a word based on all of its surroundings (left and right of the word) and therefore have a better understanding of the logic and connection between the words in the text.

3.5.1 Training of BERT

BERT's training process is divided into two stages: pre-training and fine-tuning.

3.5.1.1 Pre-Training

In the pre-training stage BERT is trained on an unsupervised task of predicting words in a sentence, a process known as Masked Language Model (MLM), and a task of predicting whether a sentence follows another sentence, known as Next Sentence Prediction (NSP). This pre-training is performed on a large corpus of text, allowing BERT to learn a rich understanding of language, and providing a good initialization, enabling the model to learn the task with a relatively small amount of data.

3.5.1.2 Fine-Tuning

In the fine-tuning stage, BERT is adapted to a specific NLP task which involves continuing the training process on task-specific data. This fine-tuning process allows BERT to be adapted to a wide range of tasks, such as text classification, named entity recognition, and question answering

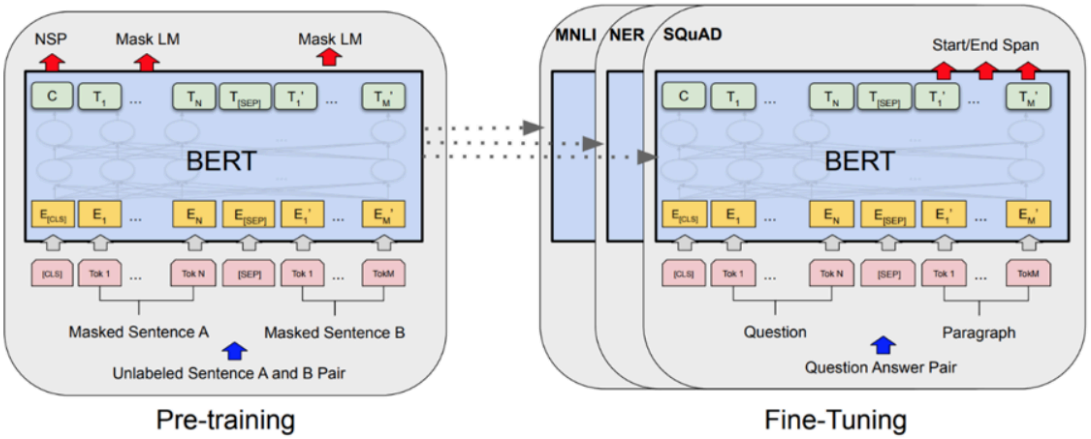


Figure 14, During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers). [source]

3.6 ALBERT

ALBERT, short for "A Lite BERT," is a model that was created to address the scalability and training efficiency problems of its predecessor, BERT (Bidirectional Encoder Representations from Transformers). The improvements brought about by ALBERT have made it into an appealing option for various Natural Language Processing (NLP) tasks. It has proven effective in a wide array of scenarios, from sentiment analysis to more intricate challenges such as question answering and language inference. Despite the model's smaller size, ALBERT manages to maintain competitive performance metrics, highlighting the effectiveness of its architectural innovations.

The model introduces two main architectural refinements that differentiate it from BERT: **factorized embedding parameterization** and **cross-layer parameter sharing**. These innovations are aimed at reducing the model's memory footprint and enhancing its training efficiency.

Factorized Embedding Parameterization reduces the number of parameters by separating the embedding matrix into two smaller matrices. This design decouples the size of the hidden layers from the size of vocabulary embeddings, addressing the parameter redundancy that plagues BERT. In BERT, the embedding size must match the hidden layer size, leading to a massive increase in parameters as the vocabulary scales. ALBERT's approach minimizes this issue by allowing for a more compact and efficient representation of vocabulary embeddings.

Cross-layer Parameter Sharing is implemented to reduce complexity and prevent overfitting. Unlike BERT, where each transformer layer has separate parameters, ALBERT shares parameters across all layers. This not only reduces the number of parameters significantly but also helps in generalizing the learning across different layers of the model. By sharing parameters, ALBERT mitigates the risk of overfitting and accelerates the training process as fewer unique parameters need to be learned.

3.7 Distance Matrix

Distance matrices are crucial tools in computational linguistics and text analysis, serving as quantitative frameworks to measure and visualize the disparities between multiple text samples. Conceptually, a distance matrix is a square matrix where each entry, d_{ij} , quantifies the distance between pairs of objects using specific metrics focused on aspects such as semantic content or stylistic features.
Each matrix element, d_{ij} , is not merely a numeric value but a quantification of the dissimilarity between text *i* and text *j*. This matrix is symmetric, and its diagonal elements are typically zeros, reflecting the axiom that a text is identically zero distance from itself. Commonly employed metrics include Euclidean distance, which measures the absolute disparity in vector components, and cosine similarity, which assesses the orientation similarity between text vectors in a high-dimensional space, irrespective of their magnitude.

In natural language processing (NLP), distance matrices are indispensable for implementing clustering algorithms, such as hierarchical clustering. These algorithms do not arbitrarily classify texts; instead, they organize them into clusters based on their proximity, as defined by the distance metrics. This clustering facilitates various analytical tasks, including document classification, thematic analysis, and explorations of authorship variations. Additionally, distance matrices are pivotal in anomaly detection, identifying texts that significantly deviate from typical cluster patterns as outliers.

Average Distance Matrix

In scenarios involving multiple text sets under varied conditions or models, the distance matrix emerges as a vital analytical tool. This matrix is constructed by averaging the distance matrices from each individual analysis, providing a consolidated view of inter-textual relationships across all considered scenarios.

The averaging process involves normalizing the individual matrices to ensure comparability, especially important when employing different distance metrics. Such normalization addresses potential scale discrepancies, preventing any single metric from disproportionately influencing the aggregated result.

This method of averaging matrices facilitates a more consistent and reproducible analysis, crucial for fields such as stylometric analysis, where it is desirable to integrate multiple textual features (lexical, syntactic, etc.) to form a holistic view of an author's style.

By providing a generalized depiction of textual relationships, the distance matrix not only enhances the robustness of the analyses but also supports more nuanced interpretations of complex literary or linguistic data, thus enriching scholarly discussions and advancing understanding in the humanities.

M	a	b	c	d	e
a	0	11	10	9	15
b	11	0	3	12	18
c	10	3	0	11	17
d	9	12	11	0	8
e	15	18	17	8	0

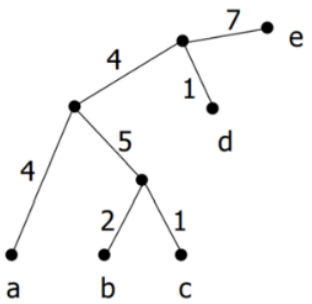


Figure 17, Distance Matrix [source].

3.8 Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW) is a powerful algorithm initially crafted for signal processing and speech recognition, now adeptly for the purpose of textual analysis. This method is particularly valuable for its ability to measure the similarity between two temporal sequences which may vary in speed or timing, making it essential for literary studies. DTW is employed to compare text sequences that unfold syntactically or semantically over time, a capability that proves crucial for tasks such as authorship attribution and comparative literary analysis.

DTW assesses the similarity between two data sequences, which could be of different lengths, by non-linearly aligning them in time. This is accomplished by stretching or compressing the time intervals of the sequences to minimize their overall distance, a technique that proves invaluable when analyzing text influences across different authors or works. This process involves constructing a cost matrix where each element represents the cost of aligning elements of two sequences. Researchers can then navigate this matrix to identify the path with the minimal total cost, representing the optimal alignment between the sequences.

Sequence Representation: Initially, texts are converted into numerical formats, typically using word embeddings to capture the semantic and syntactic nuances of the language. Each text segment or document is transformed into a sequence of vectors, preparing it for further DTW processing.

Cost Matrix Computation: Given two sequences, $s \in R^{1xN}$ and $t \in R^{1xM}$, the algorithm calculates the cost matrix $D \in R^{R(N+1)\times(M+1)}$ with specific rules that set the first row $D[1,0]$ through $D[N,0]$, and the first column $D[0,1]$ through $D[0,M]$ to infinity to provide boundaries for the computation.
for each element i,j in the matrix the cost $D_{i,j}$ is calculated as:

$$D_{i,j} = d(s_i, t_j) + min(D_{i-1,j}, D_{i,j-1}, D_{i-1,j-1})$$

where $d(s_i, t_j)$ is the distance between the i^{th} element of s and the j^{th} element of **t**, with $D_{i-1,j}$, $D_{i,j-1}$, and $D_{i-1,j-1}$ representing the costs of an insertion, deletion, and match, respectively.

Normalization: To ensure comparability between different sequences, DTW normalizes the path distance by the number of steps in the path. This normalization adjusts the distance measure to reflect the true similarity between sequences, independent of their lengths.

3.9 Isolation Forest

The isolation forest algorithm, developed by Fei Tony Liu in 2008 [3], represents a significant innovation in anomaly detection. Unlike traditional methods that focus on modeling the normal instances within a dataset, the isolation forest uniquely identifies and isolates anomalies. This approach is grounded in the premise that anomalies are inherently "few and different," which simplifies their isolation from normal points. The isolation forest's efficiency and effectiveness in anomaly detection stem from its fundamental design to target deviations directly rather than the norm.

In the field of NLP, isolation forests have proven particularly valuable. They are adept at identifying documents or texts that significantly deviate from linguistic norms. This capability extends to stylometric applications where the algorithm can detect texts that diverge from an author's typical stylistic patterns, potentially indicating different authorship or periods of literary experimentation. Furthermore, isolation forests enhance data clustering

processes by identifying and removing outliers, thereby ensuring that clusters are more homogeneous and truly representative of the underlying data structures.

The core mechanism of the isolation forest involves a collection of random decision trees that work collectively to isolate anomalies. Each tree in the forest contributes to a broader effort to segment the dataset by randomly selecting features and split values. This random partitioning is carried out recursively until each point is isolated into its own leaf node, effectively segregating anomalies from normal data based on their distinct characteristics.

Random Partitioning: The process begins with the random selection of a feature and a corresponding split value, initiating the partitioning of the data. This step is performed recursively, enhancing the algorithm's capacity to efficiently isolate individual data points.

Recursive Subdivision: As the partitioning progresses, the data is further subdivided at each tree node by continuously selecting random features and split values. This recursive subdivision strategy helps in isolating instances into increasingly smaller subsets, effectively distinguishing anomalies from normal instances.

Anomaly Score Calculation: Each instance is assigned an anomaly score based on the path length required to isolate it. Typically, shorter paths indicate anomalies due to their rarity and distinctness, making this measure a key indicator of anomalous behavior.

Thresholding: The final step involves establishing an anomaly score threshold that differentiates normal instances from outliers. Instances with scores below this threshold are classified as outliers, underscoring their substantial deviation from the typical patterns observed within the dataset.

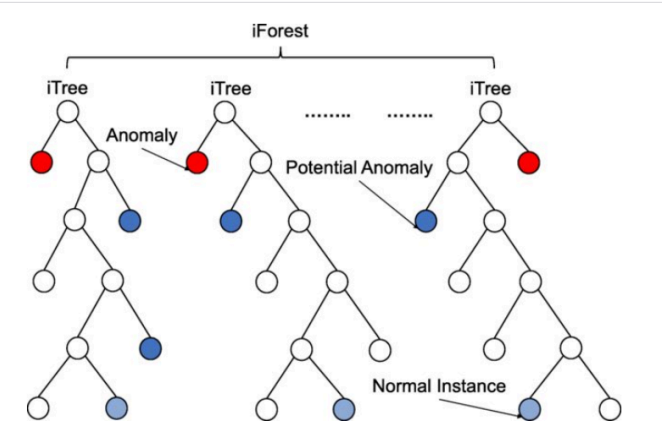


Figure 18, Isolation Forest [source].

4. Authorship Sentiment Analysis Process

The Shakespeare-BERT Model employs advanced machine learning techniques to analyze and distinguish the stylistic features of William Shakespeare's writings from those of known imposters. This section provides an in-depth, step-by-step explanation of the entire pipeline.

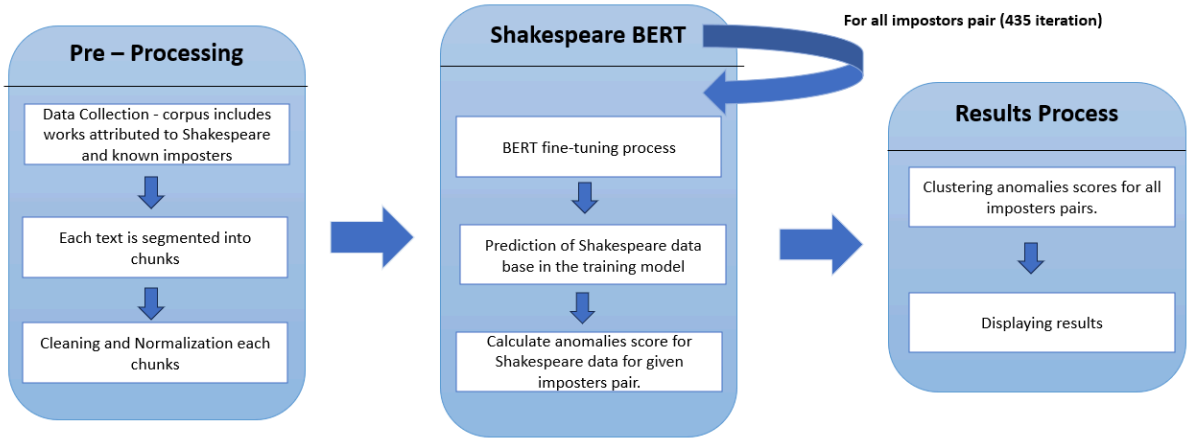


Figure 19, Model Flow-Chart.

4.1. Data Acquisition and Preprocessing

Data Collection: The corpus includes works attributed to Shakespeare and known imposters, the selected texts are historically significant and vary in style and content, providing a robust dataset for analysis.

Segmentation and Chunking: Each text is segmented into chunks of approximately 400 tokens. This size is chosen to balance detail with processing efficiency, allowing the model to capture enough linguistic information without being overwhelmed. This chunking helps in processing large texts in manageable pieces.

Cleaning and Normalization: Special characters are removed, and all text is converted to lowercase. This standardization is crucial to avoid discrepancies that could arise from simple differences like capitalization or punctuation, which are irrelevant to understanding the text's stylistic and semantic content.

4.2 Repeated Models Fine-Tuning

From the imposter dataset the unique pairs for different imposters is created, each imposter pairs texts are used to fine-tune a separate instance of the BERT model. This adaptation is aimed at teaching the model to recognize and replicate the specific linguistic style and nuances of the imposters, which would help in the next stages in comparing these learned styles against Shakespeare’s writing.

Label Creation: Labels are created and assigned to each chunk for the purposes of identifying which imposters wrote the text. labels are assigned randomly at first between the writers, and their values are either 0 or 1.

Data Split: Pairs text data is splitted into 3 categories, train, test and validation so that model could improve the learning from one epoch to another and for the purposes of evaluation the training results.

Data Feeding: Chunked labeled data is being fed into the BERT transformer, however the data is in the form of words which the model can't yet process, therefore the input data is tokenized and positional encoding is being added.

Learning Process: In the fine-tuning process the model adjusts the weights to minimize errors in predicting the next word in a sentence. By training on these texts, the model learns to identify and generate patterns that are characteristic of each imposter's writing style, creating a nuanced baseline for comparison with Shakespeare's style.

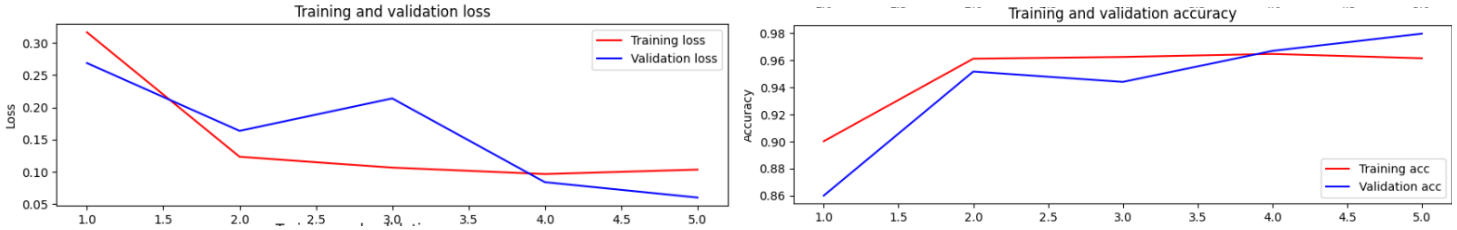


Figure 20, Example loss, accuracy graphs and results for Benjamin Disraeli vs Lewis Carroll.

During the training process for two imposters to achieve a well-trained model, our objective is to minimize loss and maximize accuracy. The graphs above display the model's training and validation results after 5 epochs (X-axis) and the corresponding results for each epoch (Y-axis).

Shakespeare Prediction: After the fine-tuning process, the model is being fed with Shakespeare's texts. These texts are chunked and transformed into embeddings suitable for the model. The model processes these embeddings to generate predictions that capture the linguistic and stylistic nuances of Shakespeare's writing. This step is critical as it produces the data necessary for the subsequent analysis of stylistic deviations between Shakespeare and the imposters.

Distance Matrix Calculation: The distance between embeddings of Shakespeare's text chunks is computed using the Dynamic Time Warping (DTW) algorithm. This measurement quantifies how similar or different two chunks are, in terms of their stylistic and linguistic features.

Isolation Forest: Previoused calculated distance matrix is fed into the isolation forest algorithm which isolates each chunk by randomly selecting features and attempting to separate the chunk from the rest. Chunks that are easier to isolate are considered more

anomalous, and they are given higher anomaly scores. These scores help identify which chunks of Shakespeare's texts are stylistically unique, potentially shedding light on his distinctive authorial signature.

4.3 Results Analysis

The anomaly detection model outputs provide a nuanced view of how each text aligns with or diverges from typical Shakespearean features, based on training against the styles of two imposters. By analyzing the mean anomaly scores across all models, we can categorize the texts into those that are clearly Shakespearean, those with ambiguous stylistic features, and those that have been misclassified, either as non-Shakespearean or incorrectly identified as Shakespearean.

High Positive Scores: These scores indicate that the text strongly deviates from what the model has learned as the style of the imposters. High positive scores would suggest that the text is likely to be authentically Shakespearean, exhibiting stylistic features that are distinctly different from those of the imposters.

Anomaly Scores Close to Zero: Scores that are close to zero suggest that the text exhibits neither strongly typical nor strongly atypical features relative to the training data (in this case, the imposter texts). Essentially, these scores indicate that the model perceives the text as stylistically neutral or ambiguous with respect to the characteristics it has learned to identify as Shakespearean or non-Shakespearean.

High Negative Scores: Conversely, high negative scores would suggest that the text closely aligns with the imposter styles and thus is likely not written by Shakespeare. It might indicate that the text shares more in common with the imposter's linguistic and stylistic patterns than with those typically attributed to Shakespeare.

Following the anomaly detection, the data is analyzed through clustering to better understand the stylistic similarities between texts. This involves normalizing the data to ensure equal contribution of features to the analysis, and incrementally including more features to identify the optimal set for clear clustering.

Clustering and Feature Analysis: Following the anomaly detection, we conducted a clustering analysis to further categorize texts based on their stylistic features. This involved normalizing the data to ensure equal contribution of each feature and performing clustering incrementally with an increasing number of features. This method helped identify the optimal number of features that provided coherent clustering results.

Silhouette Analysis: For each clustering configuration, silhouette scores were calculated to assess the clarity of separation between clusters. Higher scores indicated well-defined clustering, suggesting that texts within each cluster share substantial stylistic similarities. This metric was instrumental in determining the best feature set for analyzing the texts.

5. Research Process

Our project splits into two phases: research and implementation. We adopted the waterfall methodology, starting with the research and documentation phase before transitioning to system implementation.

5.1 The Process and Challenge

Phase A

Our project is divided into two main parts: Part A focuses on conducting thorough research on the assigned model, while Part B involves implementing and fine-tuning the model to accomplish the desired task of author identification.

During Part A, we commenced by immersing ourselves in an in-depth study of the provided paper [10]. Additionally, we summarized the topic at hand and extensively explored supplementary resources. To comprehend the workings of our BERT model, we delved into areas such as transforms, CNN, LSTM, word embedding, and prior models to better understand the advantages of our chosen module. Subsequently, we engaged in productive discussions with our mentors, presenting them with a concise summary and engaging in comprehensive conversations about the project's stages and our desired goals. This collaboration allowed us to outline a well-structured work plan to guide our book-writing process.

With the work plan in place, we embarked on the process of organizing and executing our writing tasks. Throughout the semester, we regularly shared sections of the book with our mentors, seeking their valuable feedback on each stage of our progress. We also posed targeted questions to gain a deeper understanding of the model and ensure our implementation aligned with our objectives.

To prepare for Part B of the project, we diligently considered the implementation of the algorithm required to solve the author identification task, drawing from our acquired knowledge of the model. This involved creating diagrams to guide our code design and developing a graphical user interface (GUI) to visualize the anticipated final product.

Upon completing the book, we shifted our focus to preparing a comprehensive presentation that encompassed our profound understanding of the model and the goals we aimed to achieve.

Phase B

As we transitioned into the second phase of our project, thorough research and consultations with our supervisors led to a refinement of our project goals. After extensive consideration, testing, and numerous trials, we strategically decided to transition from the Bengla-BERT model to the more tailored Shakespeare-BERT model. This decision was motivated by our dedication to enhancing project efficiency and aligning with the evolving needs of our research objectives. It was not taken lightly, but rather based on a comprehensive analysis of the project's specific requirements and challenges to ensure the best possible outcomes.

We encountered several challenges during the research phase. Initially, we began by learning to train the BERT model, but soon realized the limitations of training the entire BERT large model due to long training times. After consulting with lecturers, we discovered the effectiveness of fine-tuning pre-trained models on large corpora instead of training from scratch, leading us to focus on fine-tuning base BERT models. However, we faced difficulties with convergence and accuracy during training, prompting us to experiment with various parameters such as chunk sizes, batch sizes, and additional layers like Dropout, BatchNormalization, and activation functions (ReLU). After thorough experimentation, we found that the ALBERT[2] model provided the better results and fastest convergence, making it the ideal choice for our project, or so we thought. After deep analyzing ALBERT model results it was clear that the results may be good, however not consistent, therefore we tried an even larger and more robust BERT model, BERT-L12, which has been the final model in our project.

After training the models and achieving satisfactory results, we progressed to the next stage of the project, which involved computing the distance between embeddings of Shakespeare's text chunks. Initially, we utilized Euclidean distance but later switched to the Dynamic Time Warping (DTW) algorithm for better accuracy. To identify anomalies in the text, we employed the isolation forest algorithm, which we found to be innovative and suitable for our project's requirements.

With the project plan in place, we organized and executed our writing tasks, seeking regular feedback from mentors to ensure alignment with our objectives. Once everything worked as intended, we designed and implemented the graphical user interface (GUI), creating diagrams and images to illustrate the final product. Finally, we prepared a comprehensive presentation that showcased our deep understanding of the model and the goals we aimed to achieve.

5.2 Recommendations for Model Improvement

Exploring potential enhancements to the Shakespeare-BERT model could significantly elevate its robustness and effectiveness. Here are some ideas for future research, each of which may contribute to refining the model's capabilities in handling the complexities of Shakespearean English.

Incorporation of Larger Language Models: While the Shakespeare-BERT model provides substantial capabilities, leveraging larger and more powerful models such as GPT-3 or even the newer GPT-4 could introduce improvements in understanding and generating Shakespearean text. Adopting such models could enhance the depth of textual analysis and the quality of generated content, making the system more adept at handling the intricacies of Shakespearean English.

Adjustment of Training Epochs and Learning Rates: Fine-tuning the number of training epochs and adjusting the learning rate may impact the model's ability to learn and adapt to

Shakespearean text. Increasing the number of epochs might allow for a more thorough learning process, potentially uncovering deeper patterns within the training data. Conversely, experimenting with a dynamic or reduced learning rate could help in achieving a more stable and effective convergence, reducing the risk of overfitting while preserving the model's responsiveness to nuanced linguistic features.

Optimization of Isolation Forest Parameters: For the anomaly detection component using the isolation forest algorithm, tweaking parameters such as the number of trees, sub-sampling size, and contamination factor can lead to better performance in identifying outliers and anomalies in the text. Increasing the number of trees could improve the robustness of the anomaly detection, while adjusting the sub-sampling size may help in managing the balance between detection sensitivity and computational efficiency.

Incorporation of Syntactic and Semantic Enhancements: Introducing additional layers that focus on syntactic and semantic analysis within the transformer architecture could provide deeper insights into the text. Techniques such as syntactic parsing and semantic role labeling could enrich the model's understanding of complex sentence structures and thematic elements, which are prominent in Shakespearean works.

Each of these recommendations is worth considering making the Shakespeare-BERT model's performance better. They serve as starting points for further research and experimentation aimed at addressing the unique challenges of Shakespearean literature. However, it is important to recognize that these enhancements require substantial time and resources. Implementing such improvements is not only costly but also time-consuming, and real-life constraints like budget and computational resources must be carefully considered. Balancing ambition with practical limitations is essential as we advance the field of Shakespearean text analysis.

5.3 Use Case Diagram

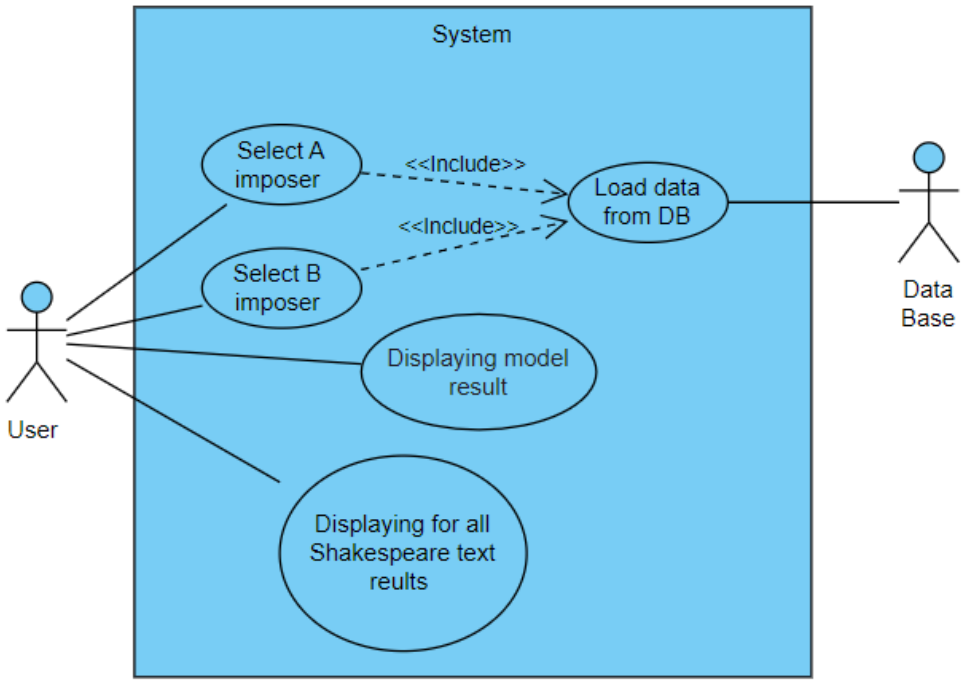


Figure 21, Use case diagram.

5.4 Activity Diagram

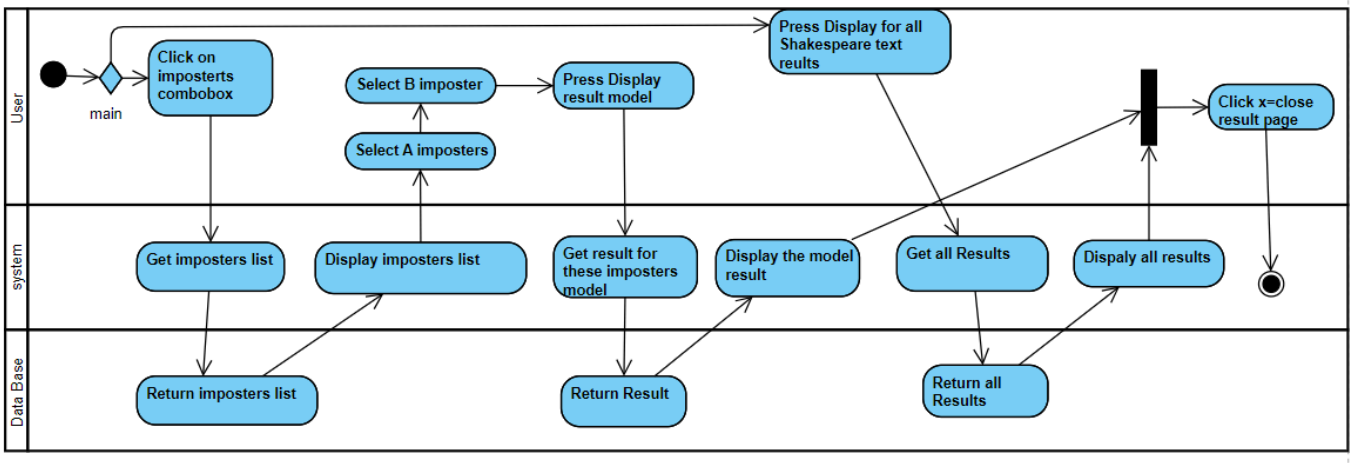


Figure 22, Activity Diagram.

5.5 User Interface (GUI Window)

The system's main page allows selection of two imposters and displays results for the selected imposters' model. Additionally, it shows results for all Shakespearean texts.

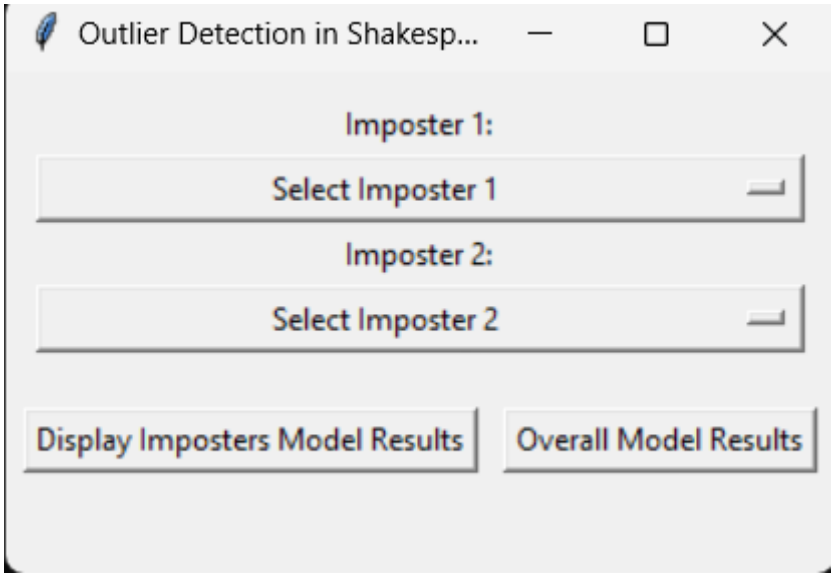


Figure 23, system's main page.

When the user doesn't select two imposters, open an error window.
It's impossible to select Impostor B before selecting Impostor A.

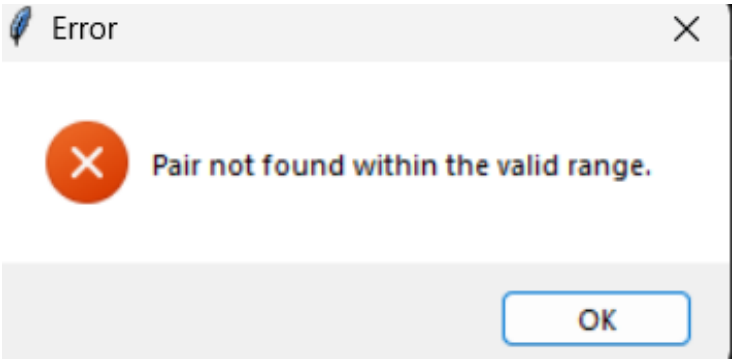


Figure 24, error window.

When the user clicks on the combobox, the imposter list opens:

Figure 25,imposters list .

Example: Benjamin Johnson and Mark Twain.
After pressing a button, the model's results are displayed:

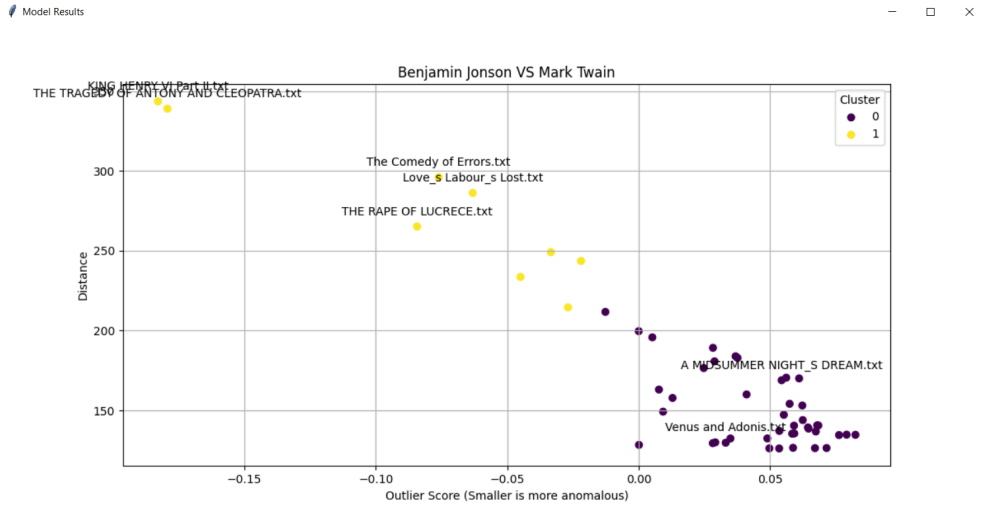


Figure 26, results for imposter model

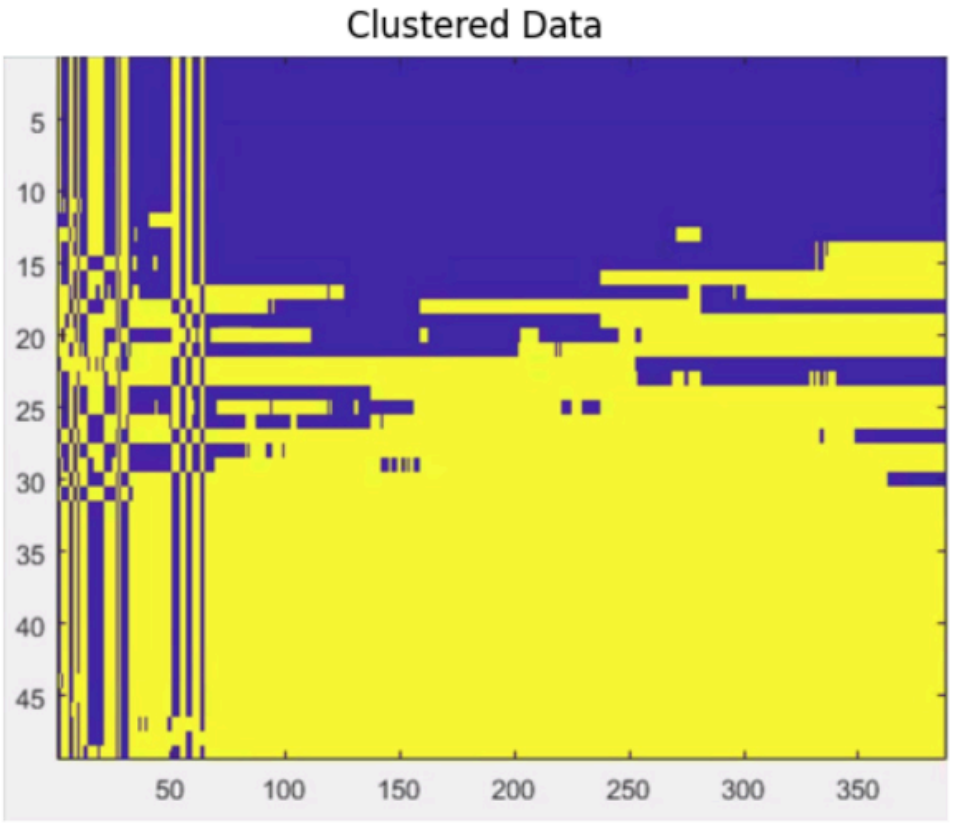


Figure 27,Shakespeare's texts clustering along different models.
Yellow indicates suspicious Shakespearean texts, while blue indicates anomalies, suggesting texts that may not be attributed to Shakespeare. Initially, the results appeared inconsistent and unclear, but as iterations progressed, they converged more closely to our expectations.

6. Results

The final normalized scores for the analyzed texts provide insightful distinctions between authentic Shakespearean texts and anomalies. Texts with scores below 1.5 are suspected to be non-Shakespearean, indicating a significant deviation from Shakespeare's recognized style.

Clear Identification of Shakespearean Texts
Texts with high positive mean anomaly scores clearly stand out as Shakespearean, reflecting their strong deviation from the imposter styles:

"Cymbeline" (Score: 1.938)
This play's high score indicates strong Shakespearean stylistic features, such as intricate plot development and character complexity, which are significantly different from the imposter texts used in model training.

"Measure for Measure" (Score: 1.9381)
The score reflects the unique thematic and linguistic elements typical of Shakespeare, like the blend of moral dilemmas and complex character interactions, clearly distinguishing it from non-Shakespearean works.

"The Tragedy Romeo and Juliet" (Score: 1.9882)
One of Shakespeare's famous plays that indeed received a high score.

Misclassification Issues
These are instances where Shakespearean texts received low scores or non-Shakespearean texts were scored highly, indicating significant areas for model improvement:

"The Tempest" (Score: 1.451)
Surprisingly low for one of Shakespeare's most imaginative works, suggesting the model may not adequately capture the unique thematic and stylistic elements of Shakespeare's late romances.

"Venus and Adonis" (Score: 1.0721)
This work, a narrative poem by Shakespeare, receiving low scores indicates a critical gap in the model's training on Shakespearean poetry versus dramatic texts.

"Henry VIII" (Score: 1.061)
As a historical play attributed to Shakespeare, its low score raises questions about the model's ability to detect historical and political themes typical of Shakespeare's later works.

Correct Identification of Non-Shakespearean Texts
This category includes texts that are not authored by Shakespeare, and our model has correctly identified them as such, indicating its effectiveness in distinguishing Shakespearean from non-Shakespearean styles:

“Fair Em by Shakespeare” (Score: 1.06)
Historically, this play has uncertain authorship and is often not included in the **Shakespearean** canon. The model's low score supports the argument that it may not have been written by Shakespeare.

“King Henry VI part I” (Score: 1.06)
The play's low score might reflect the scholarly consensus that attributes substantial parts of this historical drama to authors other than Shakespeare, such as Thomas Nashe or others involved in its creation.

“Locrine Mucedorus by Shakespeare.txt” (Score: 1.06)

This play, often excluded from the traditional Shakespearean corpus, shows a score that aligns with views questioning its Shakespearean authorship, supporting theories of different or multiple authors.

'King Henry IV, Part 1.txt'	1.832474
'THE TRAGEDY OF CORIOLANUS.txt'	1.837629
'THE TWO GENTLEMEN OF VERONA.txt'	1.863402
KING EDWARD III by Shakespeare.txt'	1.871134
'THE TRAGEDY OF OTHELLO MOOR OF VENIC	1.935567
'CYMBELINE.txt'	1.938144
'KING HENRY V.txt'	1.938144
'KING HENRY VI Part II.txt'	1.938144
'KING JOHN.txt'	1.938144
'KING RICHARD II.txt'	1.938144
'MEASURE FOR MEASURE.txt'	1.938144
'MUCH ADO ABOUT NOTHING.txt'	1.938144
'The Life and Death of the Lord Cromwell by Shak	1.938144
The Puritaine Widdow by Shakespeare.txt'	1.938144
'THE TAMING OF THE SHREW.txt'	1.938144
'THE TRAGEDY OF KING LEAR.txt'	1.938144
'THE TRAGEDY OF ROMEO AND JULIET.txt'	1.938144
'PERICLES PRINCE OF TYRE.txt'	1.940722
'The Comedy of Errors.txt'	1.940722
THE MERRY WIVES OF WINDSOR.txt'	1.940722
'KING HENRY VI Part III.txt'	1.945876
The Two Noble Kinsmen by Shakespeare.txt'	1.951031
'THE TRAGEDY OF TITUS ANDRONICUS.txt'	1.953608
'THE TRAGEDY OF ROMEO AND JULIET'	1.988235
'THE TRAGEDY OF OTHELLO MOOR OF VENIC	1.990588
'THE TAMING OF THE SHREW'	1.992941
'Taming of the Shrew'	1.995294
'The Life and Death of the Lord Cromwell by Shak	1.995294
'Sir Thomas More by Shakespeare'	2

'A MIDSUMMER NIGHT_S DREAM.txt'	1.061856
'Fair Em by Shakespeare.txt'	1.061856
'KING HENRY VI part I.txt'	1.061856
'KING HENRY VIII.txt'	1.061856
'Locrine Mucedorus by Shakespeare.txt'	1.061856
'Love_s Labour_s Lost.txt'	1.061856
'THE HISTORY OF TROILUS AND CRESSIDA.txt	1.061856
'THE RAPE OF LUCRECE.txt'	1.061856
'A LOVERS COMPLAINT.txt'	1.064433
'The Merry Devill of Edmonton by Shakespea	1.064433
'Venus and Adonis.txt'	1.072165
'Arden of Feversham.txt'	1.087629
'KING RICHARD III.txt'	1.103093
'THE WINTERS TALE.txt'	1.201031
'THE TRAGEDY OF HAMLET PRINCE OF DENM	1.216495
'THE TEMPEST.txt'	1.451031
'THE TRAGEDY OF MACBETH.txt'	1.481959
'Sir John Oldcastle by Shakespeare.txt'	1.497423
'THE TRAGEDY OF JULIUS CAESAR.txt'	1.512887
'THE LIFE OF TIMON OF ATHENS.txt'	1.582474
'THE TRAGEDY OF ANTONY AND CLEOPATRA	1.603093
'A Yorkshire Tragedy by Shakespeare.txt'	1.608247
'Sir Thomas More by Shakespeare.txt'	1.667526
'The London Prodigal.txt'	1.708763
'Merchant of Venice.txt'	1.760309
'AS YOU LIKE IT.txt'	1.770619

Figure 28,Over all models Results.

Key Findings:

- Texts in Question: Texts highlighted in red represent works historically not attributed to Shakespeare. Those in bold red suggest texts with questionable authorship.

- Model Accuracy and Limitations: The model demonstrates robust accuracy in several instances but also exhibits some limitations.

7. Verification Plan

Test Number	Test Subject	Expected Result	Pass/Failed
1	Open the GUI.	The GUI loads successfully.	Pass
2	Select the first imposter name from the combo box.	The name of the first imposter appears in the first combobox.	Pass
3	Select the second imposter name from the combo box.	The name of the second imposter appears in the first combobox.	Pass
4	Select two imposters and click the Display model result button.	Open the modal result window.	Pass
5	Click the display model result button without filling in all the fields.	Error message – Pair not found within the valid range.	Pass
6	Select imposter A and not imposer B and click the Display model result button.	Error message – Pair not found within the valid range.	Pass
7	Select imposter B and not imposer A	It is not possible to select impersonator B before impersonator A	Pass
8	Click the all for the Shakespeare texts results.	open the graph results window.	Pass

8. User's Guide Operating Instructions

8.1 Display APP - Exe File – With Gui:

Please download the ShakespeareGUI.exe in
https://drive.google.com/drive/folders/1JTcUdT6POX4jOEUuhg605AtTRXuNNak-
to your computer and click on ShakespeareGUI.

8.2 Training Code - Google Colab – Without Gui

- **Colab:** Used Colab to execute shell commands.

Running Instruction

Follow these additional steps:
1. Upload the code directory folder to your drive.
2. Open the Final_Continues_Shakespeare_Notbook.ipynb file on Google Colab from your drive.
3. Press Runtime -> Run all
4. See the results: training of all models and outlier detection for each model.

8.3 . GitHub Repository

https://github.com/DorinBachar/finalProject-/tree/main

9. Conclusions

This project takes on a difficult challenge: to further the understanding and analysis of Shakespeare's work with the help of an innovative Shakespeare-BERT model. The efforts that we have put into this project are now testaments of how adapting the BERT architecture for clearer interpretations of Shakespeare's unique language is purposefully linked to a meaningful use in the realms of machine learning applied to literary studies.

In this effort, we try to get deep into the subtleties of Shakespearean English. The proposed model, Shakespeare-BERT, reveals stylistic, thematic, and emotional layers within the text with the latest advancement in machine learning techniques. This has indeed allowed the bringing to light of possible variations in authorship and sentiment that more traditional approaches may elude, showing the great potential artificial intelligence has in literary analysis.

Looking forward, the following exciting directions should be elaborated on for further improvement of the model. This is going to involve integration with more advanced language models such as GPT-3 or GPT-4, improvement in training parameters, and extension of the abnormality detection. We expect with these advancements, the precision of the model will be developed, and its capability to perform complicated analyses will be enhanced. But this area of development does not come without significant barriers: the cost is high, the time required is great, and it requires intense computational power. These considerations will critically shape how the future of our research can advance—its speed and direction.

The development of the Shakespeare-BERT model offers new possibilities in exploration and provides a fresh set of tools into the hands of a digital humanities scholar. This project is launched with a hope that it will provide a basis for further development that may start a new tradition in successful integration of AI in literary studies. The insights we have mined down in this path will definitely make us able to enrich our approaches to NLP in the humanities, the appreciation of literary texts, and understanding in ways that at this juncture we are only able to imagine.

10. References

[1]Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[2]Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

[3]Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008, December). Isolation forest. In *2008 eighth ieee international conference on data mining* (pp. 413-422). IEEE.

[4]Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., & Langs, G. (2017, May). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging* (pp. 146-157). Cham: Springer International Publishing.

[5] Zhang, Y., Jin, R., & Zhou, Z. H. (2010). Understanding bag-of-words model: a statistical framework. International journal of machine learning and cybernetics, 1, 43-52.

[6]https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092

[7] https://towardsdatascience.com/word-embeddings-for-nlp-5b72991e01d4