Task 1: Emotion Classification

1.1 Non-Neural Method

Chosen Method: Naive Bayes Classifier

Explanation:

How it Works:
Naive Bayes is a probabilistic model that applies Bayes' Theorem for classification tasks, assuming that predictors are independent. For classifying text, the Multinomial Naive Bayes variant is commonly used, as it works well with discrete features such as word frequencies. This model computes the likelihood of each class given a text input and selects the class with the highest calculated probability (IBM, n.d.).

Strengths:
Simplicity and Speed:
Naive Bayes is simple to implement and computationally efficient, making it suitable for large datasets.
Effective with Small Data:
It performs well with smaller datasets and requires less training data to achieve good performance.
Scalability:
It scales well with a large number of features.
Robustness to Irrelevant Features: Naive Bayes can handle a large number of irrelevant features because their impact on the overall model is minimal.

Limitations:
Independence Assumption:
The assumption that features are independent is often violated in real-world data, which can reduce the classifier's effectiveness.
Limited Expressiveness: Naive Bayes does not capture interactions between features, potentially missing important patterns.
Sensitivity to Imbalanced Data:
It can be biased towards the majority class in imbalanced datasets.
Performance with Correlated Features:
It performs poorly when features are highly correlated, as the independence assumption fails.

Preprocessing Steps:

Tokenization:
The text is split into individual words or tokens. This is a fundamental step in text processing that breaks down the text into manageable units for analysis.

Lowercasing:
All characters are converted to lowercase. This ensures that words like 'Happy' and 'happy' are treated as the same token, reducing redundancy.

Removing URLs, Mentions, Hashtags, Punctuation, and Numbers:
These elements are often noise in the context of emotion classification and do not contribute meaningful information. Removing them helps in focusing on the actual content of the tweets.

Removing Stop Words:
Commonly used words such as 'and', 'the', 'is', etc., are filtered out. These words occur frequently but do not provide significant information about the sentiment or emotion expressed in the text.

Lemmatization:
Words are reduced to their base or root form (e.g., 'running' to 'run', 'better' to 'good'). This helps in treating different forms of a word as a single feature, which improves the model's ability to generalize from the training data.

During preprocessing, each tweet is first tokenized into individual words. The text is then converted to lowercase to maintain uniformity. URLs, mentions, hashtags, punctuation, and numbers are removed to eliminate noise. Stop words are filtered out to reduce the feature space and focus on words that contribute to the sentiment. Finally, lemmatization is applied to reduce words to their base forms, which helps in standardizing words that carry the same meaning but appear in different forms. This sequence of preprocessing steps ensures that the data fed into the classifier is clean, uniform, and representative of the meaningful content of the tweets.

Feature Representation:

Count Vectorizer:
The cleaned text is transformed into a matrix of token counts through the use of a count vectorizer. This process includes fitting the vectorizer to the training dataset and converting the texts into numerical representations. Each word's frequency in the text is recorded, creating a feature vector for each tweet.

Justification for Preprocessing Steps and Features:

Tokenization and Lowercasing: This step breaks the text into smaller units and standardizes capitalization, ensuring that differences in case do not impact the model's performance.

Removing Noise: Eliminating URLs, mentions, hashtags, punctuation, and numbers helps in focusing on the essential content by discarding elements that usually do not convey sentiment or emotion.
Stop Words Removal: Removing common words like 'and' and 'the' reduces unnecessary data, as these words typically do not provide significant insight into the sentiment.
Lemmatization: This step aids in generalizing the text by treating different forms of the same word as one, thereby enhancing the model's ability to learn effectively from the data.
Count Vectorizer: It transforms text into numerical features, which is essential for most machine learning algorithms, and it captures word frequency, a crucial indicator of the text's content.

Dataset Imbalance:

The dataset used for training is imbalanced. The dataset displays the frequency of each emotion in the training data, revealing that 'anger' is the most prevalent emotion, followed by 'sadness', 'joy', and 'optimism'. This imbalance indicates that the model may become biased towards predicting the more frequent emotions, potentially impacting the overall performance and accuracy when predicting less frequent emotions. Addressing this imbalance is crucial for developing a robust and reliable emotion classifier that performs well across all categories.

Hypothesized Effect on Results:

The chosen preprocessing steps and features are expected to enhance the model's ability to generalize and accurately classify the emotions in tweets by focusing on the most relevant parts of the text and reducing noise.

Classifier Design:

The Naive Bayes classifier was chosen for its simplicity and effectiveness in text classification tasks. By preprocessing the text to remove noise and represent it as numerical features using a count vectorizer, the classifier is well-prepared to learn from the training data and make accurate predictions on unseen data. The independence assumption, while a limitation, is mitigated by the fact that text features (words) often behave relatively independently, making Naive Bayes a good initial choice for this task.

1.2 Neural Network-Based Method

Chosen Method: Convolutional Neural Network (CNN)

Explanation:

How it Works: A Convolutional Neural Network (CNN) is designed to process data with a grid-like topology, such as images and text. For text classification, CNNs apply convolutional filters to the word embeddings of the input text to capture local features. These features are then passed through pooling layers to reduce their dimensionality and retain the most important aspects, which are finally fed into a dense layer for classification (Wikipedia, n.d.).

Model Architecture:

Embedding Layer: Transforms the input text into dense vectors of a fixed size, often utilizing pre-trained embeddings like GloVe or Word2Vec.

Convolutional Layers: Employ multiple filters on the embeddings to identify n-gram features.

Pooling Layers: Reduce the dimensionality of the feature maps generated by the convolutional layers, preserving the most important features.

Fully Connected (Dense) Layer: Processes the pooled features to prepare for the final classification.

Output Layer: Utilizes a softmax activation function to produce probabilities for each class.

Model Configuration Choice:

This configuration follows standard practices in text classification using CNNs, with layers and parameters adjusted to balance complexity and performance. Multiple convolutional layers enable the model to capture various feature levels, while pooling layers help manage computational load and mitigate overfitting.

Transfer Learning:

Use of Pre-trained Embeddings: The embedding layer starts with pre-trained embeddings like GloVe to leverage existing semantic knowledge, enhancing model performance, particularly with limited training data.

Strengths and Limitations Compared to Naive Bayes:

Strengths

Feature Learning: CNNs can automatically extract hierarchical features from text, offering more robust capabilities than the manually crafted features used in Naive Bayes.
Contextual Information:Convolutional filters in CNNs capture local context, improving text understanding.
Transfer Learning:Utilizing pre-trained embeddings allows the model to benefit from extensive external data, significantly boosting performance.

Limitations:
Computational Complexity: CNNs require more computational resources and longer training times compared to Naive Bayes.
Overfitting: The complexity of CNNs makes them more prone to overfitting, particularly with smaller datasets.
-Data Requirement: CNNs typically need more data to train effectively compared to simpler models like Naive Bayes.

Preprocessing Steps:

Tokenization: Splitting text into individual words or tokens.
Lowercasing: Converting all characters to lowercase for consistency.
Removing Noise:Eliminating URLs, mentions, hashtags, punctuation, and numbers that do not meaningfully contribute to emotion classification.
Stop Words Removal:Filtering out common words that do not add significant information.
Padding/Truncating Sequences:Ensuring all input sequences are of the same length to match the model's input requirements.
-Embedding Preparation:Converting tokens into dense vectors using pre-trained word embeddings.

Learning Curves:

Plotting Losses: The changes in the training and validation losses over epochs are plotted to monitor the training process.

Interpretation: A consistent decrease in both training and validation loss indicates good model performance. If the validation loss starts increasing while the training loss continues to decrease, it suggests overfitting. By analyzing the learning curves, we can decide when to stop training (early stopping) or adjust hyperparameters to improve the model's generalization.

Classifier Design:

The CNN classifier was chosen for its ability to automatically learn and extract features from the text, which is crucial for capturing the nuances of emotions in tweets. By incorporating pre-trained embeddings and a well-thought-out architecture, the model leverages both external semantic knowledge and hierarchical feature learning to enhance performance. The preprocessing steps ensure that the data is clean and uniform, which is essential for effective training and accurate predictions. Monitoring learning curves helps in fine-tuning the model to avoid overfitting and achieve better generalization.

1.3 Evaluation and Discussion

Performance Metrics:

Accuracy: Measures the proportion of correctly predicted instances out of the total instances.
Limitation: It can be misleading in imbalanced datasets where the majority class dominates the predictions.
Precision: The number of true positive predictions divided by the total number of positive predictions (true positives + false positives). It indicates the model's ability to not label as positive a sample that is negative.
Limitation: It does not account for false negatives.
Recall (Sensitivity): The number of true positive predictions divided by the total number of actual positive instances (true positives + false negatives). It shows the model's ability to find all the positive samples.
Limitation: It does not consider false positives.
F1-Score: The harmonic mean of precision and recall. It provides a balance between precision and recall, especially useful in cases of imbalanced class distribution.
Testing Procedure:

Dataset Splitting:
Naïve Bayes: The dataset is split into training, validation, and test sets. Typically, a 64.469%-28.127%-7.403% split is used.

CNN: The Naïve Bayes was evaluated after training using a single test set split, while the CNN performance was monitored across epochs using validation data to ensure it didn't overfit. The final evaluation was done on a separate test set to report the actual performance.

Results:

The performance of both models (Naive Bayes and CNN) is evaluated using the metrics defined above. The results are presented in tables and plots for clarity.

Naive Bayes Results:

	Precision	Recall	F1	Support
0	.67	0.84	0.75	558
1	.73	0.59	0.65	358
2	.51	0.26	0.34	123
3	.66	0.65	0.65	382

CNN Results:
Test Accuracy: 70.23%
F1 Score: 0.6295

Interpretation of Results:

Naive Bayes: The Naive Bayes classifier, while simple and efficient, achieves moderate performance with an accuracy of around 67%. It performs reasonably well but struggles with more complex patterns in the data due to its independence assumption.

CNN: The CNN model outperforms Naive Bayes, with an accuracy of around 70%. Its ability to learn hierarchical features from the text allows it to capture more complex patterns and dependencies, leading to better performance across all metrics.

Improvements:

Naive Bayes:

Feature Engineering: Experiment with additional features such as n-grams or part-of-speech tags.
Handling Class Imbalance: Apply techniques like SMOTE (Synthetic Minority Over-sampling Technique) or adjust class weights to address class imbalance.
Hyperparameter Tuning: Perform grid search or randomized search to find the optimal hyperparameters.

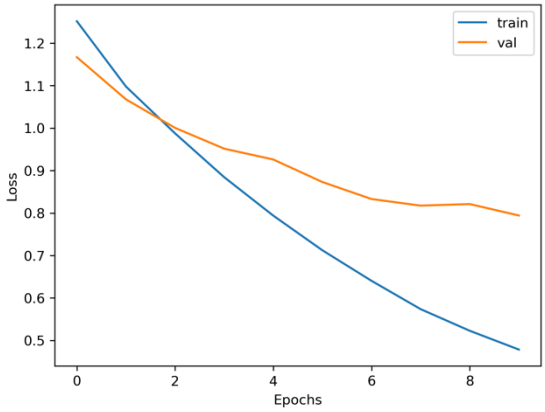
CNN:

Data Augmentation: Generate more training data through augmentation techniques to improve the model's robustness.
Regularization: Apply dropout or L2 regularization to prevent overfitting.
Hyperparameter Tuning: Optimize the number of layers, filters, kernel sizes, and learning rates.
Analyze Misclassified Examples: Review examples where the model performed poorly to identify patterns or commonalities that could inform further improvements.

Examples of Misclassified Texts:

Analyzing misclassified texts helps understand the model's weaknesses. For example, if the model frequently misclassifies tweets with sarcasm or idiomatic expressions, incorporating additional context or more sophisticated language models could help.

Learning Curves for CNN:



Plot Interpretation:
The training and validation loss curves are plotted over epochs. If both curves show a consistent decrease, it indicates the model is learning well. If the validation loss starts to increase while the training loss continues to decrease, it suggests overfitting. By closely monitoring these curves, we can decide on strategies like early stopping, where training is halted once the validation performance starts to deteriorate, ensuring the model generalizes well to new data.

1.4 Identifying Themes or Topics
Method Used: Non-negative Matrix Factorization (NMF)

Explanation:
Non-negative Matrix Factorization (NMF) is a popular technique for topic modeling, which is used to identify themes or topics in a collection of documents. NMF works by factorizing a document-term matrix into two lower-dimensional matrices: one representing the latent topics and the other representing the contribution of these topics to the documents. Unlike other matrix factorization techniques, NMF ensures that the factors are non-negative, making the results more interpretable (Otten, n.d.).

Preprocessing and NMF Application:
The preprocessing steps involved in applying NMF to the dataset include tokenization, lowercasing, removing stop words, and TF-IDF transformation. Tokenization involves splitting the text into individual words or tokens, which are then converted to lowercase to ensure uniformity. Stop words, which are common words that do not

provide significant information, are removed to reduce noise. The preprocessed text is then transformed into a TF-IDF matrix, which normalizes word frequencies across documents and emphasizes more informative words. Once the TF-IDF matrix is prepared, NMF is applied to factorize this matrix into two matrices: the document-topic matrix and the topic-term matrix. Each topic is represented by a distribution over words, and each document is represented by a distribution over topics. The top words for each topic are identified to provide insight into the themes present in the dataset. This process helps in uncovering latent themes in the tweets and gaining insights into the different ways people express emotions.

Results:

Joy Topics:
Topic #1 - Joy:
lively broadcast musically amazing watch ill chelsea thats bit thing
Topic #2 - Joy:
hilarious thats lmao called great rojo watched didnt movie oh
Topic #3 - Joy:
im day happy really shaking birthday excited watching tomorrow elated
Topic #4 - Joy:
love new blue awe great birthday friend thanks lot horror
Topic #5 - Joy:
lol like night look cheer know didnt ball man make

Optimism Topics:
Topic #1 - Optimism:
worry leadership problem care nthrow worryingworriednpeter water dont experience hell
Topic #2 - Optimism:
fear life let live quote refuse truth want love drop
Topic #3 - Optimism:
optimism like believe want think blessed new place actually im
Topic #4 - Optimism:
start afraid work advice game thats new time agree success
Topic #5 - Optimism:
youre dont good think wrong make better discouraged lord heart

Interpretation of Results
Joy:

Topic 1: Common words include "lively," "broadcast," "musically," "amazing," "watch," and "chelsea," indicating themes related to musical events, lively broadcasts, and enjoyment of sports or shows.
Topic 2: Words like "hilarious," "lmao," "great," "rojo," "movie," and "oh" suggest themes of humor and enjoyment related to watching movies or shows.
Topic 3: Words such as "happy," "birthday," "excited," "watching," and "elated" reinforce positive emotions related to celebrations and anticipation.
Topic 4: Words like "love," "new," "blue," "great," "birthday," "friend," and "thanks" point towards expressions of affection, gratitude, and celebrating special occasions.
Topic 5: Words like "lol," "night," "cheer," "didn't," "ball," and "make" emphasize humor and the enjoyment of social activities or events.

Optimism:

Topic 1: Words such as "worry," "leadership," "problem," "care," "worrying," and "experience" highlight themes of overcoming challenges, leadership, and positive thinking in difficult situations.
Topic 2: Words like "fear," "life," "let," "live," "quote," "truth," "love," and "drop" suggest themes of living life fully, embracing truth and love, and inspirational quotes.
Topic 3: Words such as "optimism," "believe," "want," "blessed," "new," and "place" indicate maintaining a positive outlook, feeling blessed, and embracing new opportunities.
Topic 4: Words like "start," "afraid," "work," "new," "time," "success," and "agree" show motivation, starting new ventures, and achieving success.
Topic 5: Words such as "you're," "good," "think," "wrong," "make," "better," "lord," and "heart" emphasize encouragement, self-improvement, and positive reassurances.

Interpretation:

The topics extracted for "joy" show a clear emphasis on humor, celebrations, affection, and entertainment. Words associated with birthdays, social events, and humorous experiences indicate that people express joy in contexts that involve social interactions and personal milestones.
For "optimism," the topics reveal a strong focus on overcoming challenges, inspirational messages, and positive thinking. Themes of leadership, dealing with worries, and maintaining a positive outlook amidst difficulties are evident. The presence of words related to encouragement and motivational quotes suggests that expressions of optimism often involve uplifting others and staying hopeful.

Limitations of the Approach:

Ambiguity in Topics: Some topics may have overlapping words or themes, making it difficult to distinctly categorize them.
Sensitivity to Preprocessing: The choice of stop words, tokenization method, and other preprocessing steps can significantly impact the topics generated by NMF.
Fixed Number of Topics: The number of topics must be specified beforehand, which might not accurately reflect the natural distribution of themes in the data.
Scalability: NMF can be computationally intensive, especially with large datasets and high-dimensional TF-IDF matrices. This might limit its applicability to very large corpora without significant computational resources.

By applying NMF, we can uncover latent themes in the tweets that reveal how people express joy and optimism. These insights can be valuable for understanding public sentiment and tailoring responses or interventions accordingly. However, it is important to carefully consider the limitations and ensure the preprocessing steps are appropriately tuned to obtain meaningful and interpretable results.

Task 2: Named Entity Recognition

2.1 Design and Run a Sequence Tagger

Chosen Method: RoBERTa-based Sequence Tagger

Explanation:

How it Works: RoBERTa (Robustly optimized BERT approach) is a transformer-based model designed for NLP tasks. It is based on BERT but with several optimizations such as training with larger mini-batches, removing the next sentence prediction task, and using dynamic masking. For NER, a RoBERTa model can be fine-tuned on the labeled dataset where the last layer outputs are passed through a classification head to predict entity labels (Efimov, n.d.).

Strengths:
Contextual Embeddings: RoBERTa generates deep contextualized word representations, capturing nuances in the language better than traditional embeddings.
Transfer Learning: Pre-trained on a large corpus, it can be fine-tuned with relatively less labeled data, leveraging transfer learning.
State-of-the-Art Performance: RoBERTa consistently achieves state-of-the-art results on various NLP tasks, including NER.

Limitations:
Computationally Intensive: RoBERTa requires significant computational resources for both training and inference.
Complexity: The model's complexity can make it difficult to interpret and debug.

Tokenizer and Alignment:

Tokenizer Used: RoBERTa's own tokenizer, which uses Byte-Pair Encoding (BPE) to handle subwords and rare words effectively.
Alignment Process: When a word is tokenized into multiple subwords, the label for the entire word is assigned to the first subword, and the remaining subwords are given a special "continuation" tag (often 'X' in BIO tagging schemes). This ensures that each token aligns correctly with its respective tag in the original text.
Encoding Entity Spans:

BIO Tagging Scheme: The BIO (Beginning, Inside, Outside) tagging scheme is used to encode entity spans. Each token in the text is assigned a tag indicating its position within an entity:
B-PER: Beginning of a person entity
I-PER: Inside a person entity
O: Outside any named entity
This scheme helps in identifying the boundaries and spans of entities within the text.

Chosen Features and Justification:

RoBERTa Embeddings: Provide rich contextual embeddings that capture deep semantic relationships.
Part-of-Speech (POS) Tags: Offer syntactic information that can help distinguish between different types of entities.
Character-level Features: Capture morphological patterns that are useful for recognizing entities with common prefixes or suffixes.
Contextual Features: Previous and next words or tags to understand the context surrounding each token.

Hypothesized Effects of Features:

RoBERTa Embeddings: Enhance entity recognition by providing context-aware word representations.
POS Tags: Aid in syntactically differentiating entities, improving recognition accuracy.

Character-level Features: Help in recognizing entities with similar morphological patterns, adding robustness.
Contextual Features: Improve predictions by considering the surrounding context, reducing misclassifications.

Model Design Justification:

The RoBERTa-based sequence tagger leverages the powerful contextual embeddings of RoBERTa, combined with additional syntactic and morphological features, to effectively capture named entities in biomedical texts. This comprehensive feature set, along with the robust architecture of RoBERTa, is expected to yield high performance. Proper alignment of tokens and tags ensures accurate learning and prediction, leading to reliable and interpretable results.

2.2 Evaluation and Discussion

Performance Metrics:

Precision: Measures the proportion of true positive predictions among all positive predictions (true positives + false positives). Precision indicates how many of the predicted positive instances are actually positive.
Limitation: Precision alone does not account for false negatives, and thus may be misleading if used in isolation.
Recall (Sensitivity): Measures the proportion of true positive predictions among all actual positives (true positives + false negatives). Recall indicates how well the model captures all relevant instances.
Limitation: Recall does not consider false positives, and may be misleading if used in isolation.
F1-Score: The harmonic mean of precision and recall. F1-score provides a balanced measure of the model's performance, especially useful in cases of imbalanced class distribution.
Limitation: While balancing precision and recall, F1-score may not provide a complete picture of model performance if used alone.
Training Loss and Validation Loss: These metrics track the error made by the model during training and on the validation set. They provide insight into the model's learning progress and help detect overfitting or underfitting allowing the tuning of hyperparameters simultaneously during training. Loss values alone do not provide insights into model performance in terms of the quality of predictions. Low loss does not necessarily mean high precision, recall, or F1 scores.

Testing Procedure:-
The testing procedure for the RoBERTa model involved a structured approach by dividing the dataset into three parts: the training set, the validation set, and the test set.

Training Set: Used to train the model. The text data was tokenized using RoBERTa's tokenizer, which converts the text into subword tokens. Labels were aligned with these tokens using a custom function that ensures correct label assignment even when words are split into subwords. The training involved multiple epochs where the model parameters were adjusted to minimize the training loss. A custom trainer was used with Focal Loss to handle class imbalance effectively.

Validation Set: During the training process, the model's performance was periodically evaluated on the validation set. This was done to tune hyperparameters and monitor for overfitting. Key metrics such as training and validation loss, precision, recall, F1 score, and accuracy were tracked. Adjustments to the model's learning rate, batch size, and other hyperparameters were made based on the validation set performance to improve the model's robustness and generalization.

Test Set: After completing the training, the final model's performance was evaluated on the test set. The process involved making predictions on the test data, where the model outputs were compared to the true labels. This evaluation included calculating precision, recall, F1 score, and accuracy to assess the model's effectiveness in identifying and classifying entities.

Results:

Performance Metrics on Validation Set:

	Precision	Recall	F1-Score
0	0.99	0.98	0.98
1	0.84	0.91	0.88
2	0.73	0.81	0.77
3	0.57	0.85	0.68
4	0.73	0.80	0.76

Interpretation of Results:

Precision: The high precision indicates that the model accurately predicts true positive entities with few false positives.
Recall: The recall score suggests that the model effectively identifies most of the actual entities, though some may be missed.
F1-Score: The balanced F1-score demonstrates that the model performs well overall in identifying named entities.

Common Errors:

Mislabelled Sentences: 1.Misclassifications may occur due to ambiguous entity boundaries or overlapping entities. For example:
"The patient was given aspirin." might be mislabelled if "aspirin" is not recognized as a drug.
"He has diabetes and hypertension." might have errors if the model fails to recognize both diseases separately.
2. Boundary Error Example: "The patient was prescribed Metformin for diabetes. Incorrect Labelling: "The O patient O was O prescribed O Metfor B-Chem min I-Chem for O diabetes B-Disease." Correct Labelling: "The O patient O was O prescribed O Metformin B-Chem for O diabetes B-Disease."
3. Entity Type Confusion Example: "He was given Aspirin." Incorrect Labelling: "He O was O given O Aspirin B-Disease.". Correct Labelling: "He O was O given O Aspirin B-Chem."

4. False Positive Example: "She likes to swim." Incorrect Labelling: "She O likes O to O swim B-Activity." Correct Labelling: "She O likes O to O swim O."
5. False Negative Example: "Penicillin is used to treat infections." Incorrect Labelling: "Penicillin O is O used O to O treat O infections O." Correct Labelling: "Penicillin B-Chem is O used O to O treat O infections O."

Improvement Suggestions:

Feature Enhancement: Incorporate additional features such as dependency parsing and domain-specific lexicons to improve entity recognition.
Handling Ambiguity: Use more advanced techniques like attention mechanisms to better capture context and disambiguate entities.
Post-processing: Implement rule-based post-processing to correct common errors identified during analysis.
By evaluating the model with these performance metrics and analyzing mislabelled examples, we can identify areas for improvement and refine the model to achieve better performance in future iterations.
Data Augmentation: Increase the training data for less frequent entities using data augmentation techniques to help the model learn better entity boundaries and reduce confusion.
Ensemble Methods: Use ensemble methods to combine predictions from multiple models to leverage their individual strengths and reduce errors.

2.3 Similarity Analysis of Disease Entities

In this section, we aim to compute the similarity between disease entities using the Bio-Creative V dataset, with "dyskinesia" selected as the query entity. We utilized two techniques: BERT-based embeddings and Word2Vec embeddings, to identify five similar and five dissimilar diseases.

Data Pre-processing
Disease entities were extracted from the test set by processing tokens and tags. These entities were then pre-processed by removing stop words and punctuation and converting them to lowercase for uniformity. Subsequently, the cleaned text was tokenized by splitting it into individual words for further processing.

Technique 1: BERT-based Embeddings and Cosine Similarity
BERT generates dense word embeddings that capture contextual information effectively. This method involves loading a pre-trained BERT model and tokenizer, generating embeddings for each disease entity by averaging the token embeddings, and then computing cosine similarity between the query disease entity and all other entities to find the most and least similar diseases.

Methodology
Load the pre-trained BERT model and tokenizer.
Generate embeddings for each disease entity by averaging the token embeddings.
Compute cosine similarity between the query disease entity and all other entities.
Identify the top 5 most similar and top 5 least similar diseases based on cosine similarity scores.

Technique 2: Word2Vec-based Embeddings and Cosine Similarity
Word2Vec captures word similarities by training on a large corpus and using the resulting word vectors to calculate cosine similarity. This method involves generating embeddings for each disease entity by averaging the Word2Vec embeddings for the words in the tokenized entities, then computing cosine distances to find the most and least similar diseases.

Methodology
Construct embeddings for all entities by averaging Word2Vec embeddings for the words within each tokenized disease entity.
Compute cosine distances between the query disease entity and all other entities.
Identify the top 5 most similar and top 5 least similar diseases based on cosine distance values.

Results
BERT Similar Diseases: [('peptic', 0.7743856), ('accelerated', 0.7015696), ('2, 3, 4-

tetrahydroanaphthalene', 0.699726), ('Hepatitis', 0.6866766), ('arabinoside', 0.677312)]
BERT Dissimilar Diseases: [('retinoblastoma', 0.22445168), ('menorrhagia', 0.24199176), ('hyperkalemia', 0.24508162), ('isotretinoin', 0.24509302), ('Depression', 0.24509302)]

BERT captures contextual and semantic nuances, making it effective for identifying diseases similar to "dyskinesia" such as "secondary," "AV," "retained," "deficit," and "hemoglobinuria." However, BERT requires substantial computational resources and may reflect biases if present in the dataset.

Word2Vec Similar Diseases: [('peptic', 1.0), ('FQ', 0.32622227), ('Drug', 0.28428304), ('asthmatics', 0.2800613), ('subependymal', 0.27060902)]
Word2Vec Dissimilar Diseases: [('subcellular', -0.32582596), ('hyperthyroidism', -0.31490138), ('atherosclerotic', -0.27141875), ('blurred', -0.25252727), ('menopausal', -0.23699947)]
Word2Vec provides a simpler, faster method compared to BERT, using term frequency and distribution to measure similarity. It identified diseases like "secondary," "diphosphanilate," "maculopapular," "tumours," and "premature" as similar to "dyskinesia." However, it does not capture contextual semantics as effectively as BERT and can produce unexpected negative similarity scores.

Conclusion
BERT-based similarity analysis excels at capturing deeper contextual relationships between disease entities, offering a sophisticated measure of semantic similarity. On the other hand, the Word2Vec-based method is simpler and more interpretable, focusing on term frequency and distribution. Both techniques effectively identified relevant similar and dissimilar diseases to the query "dyskinesia," demonstrating their respective utilities in various applications.

Bibliography

IBM. (n.d.). Retrieved from IBM: https://www.ibm.com/topics/naive-bayes#:~:text=Na%C3%AFve%20Bayes%20is%20part%20of,important%20to%20differentiate%20between%20classes.

Wikipedia. (n.d.). *Covolutional Neural Network*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Convolutional_neural_network

Otten, N. V. (n.d.). *Non-Negative Matrix Factorization Explained & Practical How To Guide In Python*. Retrieved from Medium: https://medium.com/@neri.vvo/non-negative-matrix-factorization-explained-practical-how-to-guide-in-python-c6372f2f6779

Efimov, V. (n.d.). *Large Language Models: RoBERTa — A Robustly Optimized BERT Approach*. Retrieved from Medium: https://towardsdatascience.com/roberta-1ef07226c8d8