Automated essay scoring using Applied Machine Learning Techniques.

SOKRAT BASHIROV, ADA University, George Washington University, Azerbaijan, sbashirov16723@ada.edu.az

ELJAN MAHAMMADLI, ADA University, George Washington University, Azerbaijan, emahammadli16744@ada.edu.az

Additional Keywords and Phrases: AES System, Automated Essay System, Machine Learning, Deep Learning

1. INTRODUCTION

One of the most important ways to evaluate students' learning and performance is through essay writing. For instructors, however, the manual grading of essays takes a lot of time and resources. Automated Essay Scoring (AES) systems have become popular as a potentially effective way to support teachers' efforts by giving students timely and consistent feedback. With the use of these tools, grading processes can be greatly decreased, and essay assessments can be more widely included into educational environments. Many AES systems have limitations despite their potential, by expensive costs and the lack of large, diverse datasets that are required for training reliable models.

Previous research initiatives have often used inadequate data sets, which don't consider many different writing styles and situations of students. Besides, it is hard for today's technology-based instruments to be as accurate as real people while their algorithms are often prejudiced because some young people come from poorer areas whereas others live abroad. To solve these restrictions, the study uses a detailed publicly available set of examples showing classroom writing as it is. With examples to cover economic and geographic diversity, it is the data set of AES training models. This study aims to construct an open-source AES algorithm that increases improvement accuracy and justice using many machine learning methods like neural networks, support vector machines and linear regression (Mughal, 2017). The quadratic weighted kappa metric was employed for model evaluation. It is an evaluation metric, that measures the degree of agreement between the expected and actual scores given by human graders. This measure is especially well-suited for essay scoring, because of its ability to handle the average nature of essay scores and the ability to assess larger differences between expected and actual scores more severely.

Research is so crucial simply because it could open advanced AES tools to many students and different educational institutions, thus making it easy to reach. Should this project be successful, the aim will be to make sure that tutors provide continuous and high-quality feedback thereby improving students' general learning experience through a reliable and easy-to-get-to AES tool.

2. LITERATURE REVIEW

Research has focused on AES due to its ability to accelerate grading and provide live feedback thus becoming a leading area. This paper reviews different techniques and models that have been developed for AES with an interest in evolution.

<u>Neural Network Approaches to AES</u>

Taghipour and Ng (2016) investigated using Recurrent Neural Networks (RNNs) in AES where feature engineering is not required. It was found that LSTM networks outperformed classical systems by 5.6% on quadratic weighted kappa metric. This showed that neural network's ability to learn the relationship between an essay and its score without relying on predefined features. [1]

Cai (2019) also explored the use of RNNs, specifically combining feature scoring with neural network models. The study utilized the ASAP essay dataset and found that incorporating GloVe embeddings significantly improved the results. This highlights the importance of advanced embeddings in enhancing model performance for AES tasks. [2]

Nguyen and Dery (2016) also assessed different neural network architectures for AES. The objective of their study was to create a dependable automated essay grading system which could tackle the high costs involved in conventional assessments. The research established that the use of neural networks could offer a feasible answer to precise essay grading.[3]

<u>Transformer-Based Models</u>

Ludwig et al. (2021) introduced transformer models for AES. He compared their performances with traditional bag-of-words (BOW) approaches. Transformer's analysis demonstrated that their models outperformed logistic regression models based on BOW, particularly in tasks requiring the understanding of word order and context. [4]

In the study by Rodriguez and colleagues (2019), BERT and XLNet models for AES were compared with a conclusion that transformer architectures are superior to recurrent neural networks. According to their report, these models have the capability of performing better than humans in public AES data sets demonstrating the potential for transformers in this field.[5]

Deep Learning and Hybrid Models

Boulanger and Kumar (2018) explored deep learning application within AES by using deep neural network trained with an extensive collection of writing features. Consequently, it was evident from their findings that deep learning models have the potential to greatly improve the precision of AES; however, there is need for a much bigger dataset incorporating more hand-graded essays to fully exploit their capability. The study also underscored the advantages of ensemble methods in boosting model performance.[6]

In his write-up on AES so far, Lim et al. (2021) dissected extant systems into content similarity, machine learning and hybrid frameworks. They called for systems that marry content and style analytics for better grades prediction on essays. During the research, it was noted that creation of efficient AES calls for reliable evaluation metrics such as quadratic weighted kappa. [7]

Traditional and Emerging Techniques

Dıklı of 2006 examined the development of AES systems in his research. Moreover, he talked about some popular instruments such as Project Essay Grader (PEG), Intelligent Essay Assessor (IEA), E-rater, and IntelliMetric. High fidelity and validity of such systems was underscored in his work with regards to other attempts being made within this context aimed at enhancing them more extensively. [8]

Patil and Ali (2018) have taken a review of different ways through which automated scoring for short and long answers can be done: challenges faced by AI grading systems were discussed, as well as how these technologies are limited by current ones. They concluded that next generation Automated Essay Scoring models should be hybrid in nature and supported by feedback mechanisms to realize their potential as being effective evaluation tools within this domain (html element). [9]

Sanuvala and Fatima (2021) employed OCR and machine learning in designing a method used to evaluate the accuracy of handwritten exam papers produced during tests. This work has proved how possible it is to automatically mark scripts written by hand thus extending the capabilities of Automatic Essay Scoring systems. [10]

In general, there are substantial improvements realized in implementing Neural Network based systems as well as transformer models which are very effective when it comes to increasing grading accuracies with high levels of efficiency as shown by literature on AES The best way of developing powerful systems of AES is by combining the analysis of both content and style. But as much as there has been progress in such systems, there are still some challenges that have not been fully addressed; these include algorithmic bias as well as dataset size which are important before AES could be made totally practical within educational contexts.

3. DATA PROCESSING

First things first, let us begin on pre-processing steps which are cleansing, normalization and feature extraction in our dataset. In the exploration phase, we begin with explorative data analysis where we present visualizations about summary statistics and correlation matrices to understand the aspects of word count, sentence count and essay length. Consequently, these visualizations become windows through which to see through of feature extraction. The following figure shows how the relationship between feature level score, number of words, number of sentences and length of essay looks like using a correlation matrix heatmap. The distribution in the given graph explains how the given attributes are distributed across different documents.

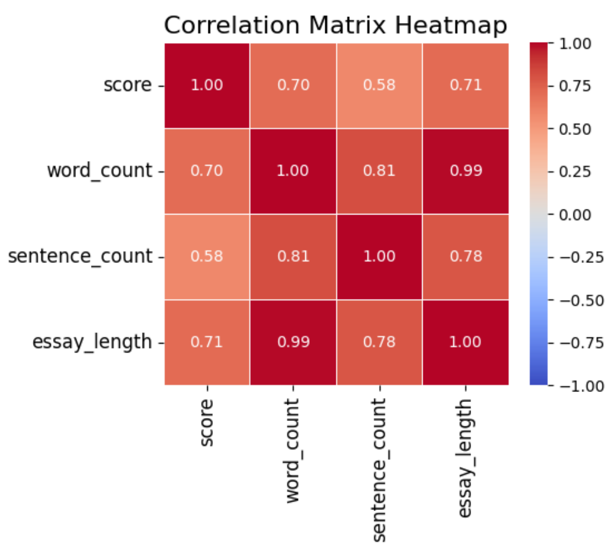

Figure 1: Correlation matrix heatmap.

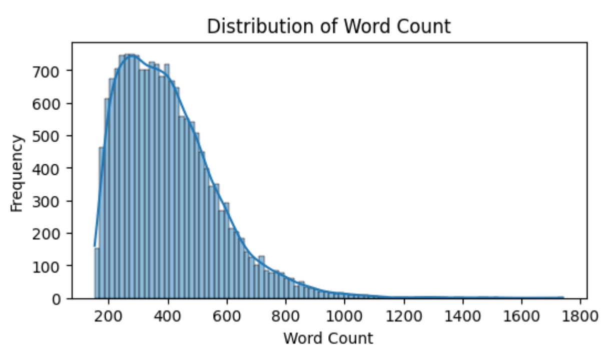

Figure 2: Distribution of Word Count.

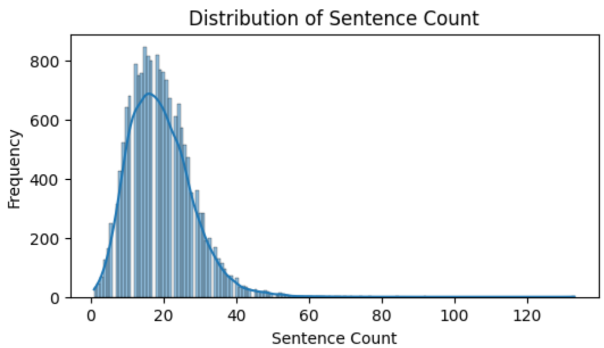

Figure 3: Distribution of Sentence Count.

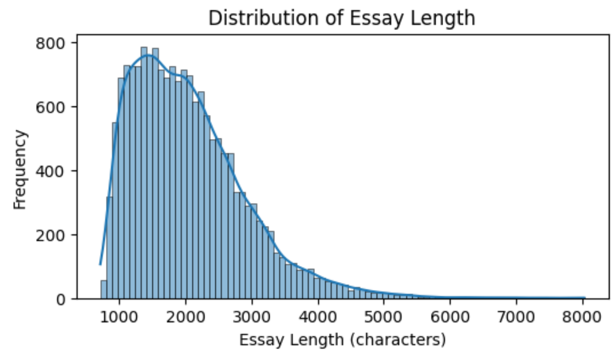

Figure 4: Distribution of Essay length.

One of the steps that can never be left out during data processing is preprocessing which includes conversion to lowercase, removal of URLs and HTMLs, expansion of contractions, removal of numbers and strings starting with @ such as twitter handles these are some steps required if one wants to build a model which will be precise in future.
The other one of the most important steps is feature engineering. In this stage of the process, features are extracted based on various characteristics of paragraphs, sentences, and words of essays. To show the different types of extracted features, these can be enlisted:

Basic Text Features provide information about the text.
- word_count, sentence_count, essay_length.

Readability Indices assess the complexity and readability of the text.
- Coleman-Liau Index, Gunning Fog Index, Flesch Reading Ease, Flesch-Kincaid Grade Level, SMOG Index, Dale-Chall Readability Score, Linsear Write Formula, Spache Readability etc.

Lexical Diversity and Language Features evaluate the diversity and complexity of the vocabulary and language used.
- Type-Token Ratio, Yule's K Measure, ngram diversity score, total distinct Ngrams, average ngram frequency, max ngram frequency, syntactic complexity, semantic diversity etc.

Sentence and Paragraph Engineering analysis text based on sentences and paragraphs.
- Sentence Level Features,
    - sentence length, sentence word count.
- Paragraph Level Features
    - paragraph length count for specific thresholds.

Text Processing Features derived from advanced text processing techniques.
- Bag of Words, TF-IDF Features, CountVectorizer Features

Error Analysis
- spelling errors, count and analysis of spelling mistakes in the text.

Custom Metrics Unique or specific metrics that may be useful for analyses.
- punctuation analysis, punctuation diversity, dialogue and grammar, prepositions frequency

All of these methods make for a rich set of features that exist within a dataset, providing a strong basis upon which machine learning algorithms can perform well.

4. METHODOLOGY

In order to train a highly accurate AES system using all the simple and complex features we extracted we have utilized ensemble learning techniques. In essence, we are utilizing LightGBM [11] which is based on tree-based algorithm and is known for superior speed and performance in competitive Machine Learning. To enrich our feature space, we integrated 5 OOF (out-of-fold) predictions generated by DeBERTa [12] model which is pre-trained masked language model based on BERT [13] architecture. DeBERTa is an improvement over BERT and RoBERTa that uses disentangled attention mechanism. This is useful in our case since this model encodes both the content and position using attention mechanism, allowing it to understand the complex context and relationship in the essay. Essentially, using this as a feature, we can extract hidden features such as the language use, coherence, style, and tone. The reason we are using out-of-fold prediction is that since the models that are evaluated on the training sets have not seen those specific points thus enabling us too unbiased to evaluate. We use OOF predictions with stacked models on ensemble setting and this shows better performance rather than using one string model. Finally, it helps to avoid overfitting as well.

4.1. Objective Function

Since each essay in the training set was scored on the scale of 1 to 6 using holistic rating system, we have various ways to set the objective. Although we can use metrics like precision, recall or F1, those metrics are not effective in our setting. The reason is because the classes are ordinal meaning, they have e natural order. That is why to better represent the objective and evaluation score we are using Quadratic Weighted Kappa (QWK) also known as Cohen's kappa. Using this metric, we make sure that the difference between the scores matters and it introduces weights to distinguish between the ratings. QWK penalizes disagreements between different classes depending on how far those ratings are. For example, the difference between scores 1 and 2 is not as severe as difference between the 1 and 6. All those features make it very suitable to evaluate educational AES systems making it standard choice. QWK is calculated as:

$$\mathrm{QWK} = \frac{1 - \sum_{i,j} w_{ij} o_{ij}}{1 - \sum_{i,j} w_{ij} e_{ij}}$$

Where i and j are rating categories, w_{ij} is the disagreement between the raters from items in category i and j, o_{ij} is the is the observed frequency count of items rated by the first rater in category i and by the second rater in category j and e_{ij} is the expected frequency count of items rated in categories i and j. Weight matrix w is calculated as:

$$w_{ij} = \frac{\left(i - j\right)^2}{(C - 1)^2}$$

This equation penalized the disagreement according to the squared distance from total agreement and is normalized by the maximum possible squared distance $(C - 1)^2$ to ensure it is between 0 and 1 where 0 means there is no agreement and 1 means a perfect agreement. Observed counts o_{ij} is calculated directly from the data. Expected counts e_{ij} is calculated under the assumption that there is no agreement better than a chance via:

$$e_{ij} = \frac{n_i \times n_j}{N}$$

where n_i is the total number of ratings by the first rater in category i, n_j is the total number of ratings by the second rater in category j, and N is the total number of items.

To optimize the gradient boosting models, we have adjusted QWK with a custom objective function. First, we introduce constants a and b where a equals to 2.9 and b equals to 1.0. The former is used to avoid potential cases where the scoring does not start from zero and the latter is used to add another level of regularization to help to optimization process. The prediction from the models is clipped between [0, 6] since those are the only accepted scores. During the optimization process the gradient and Hessian is calculated according to the objective function to train the gradient boosting machine-based models by finding the squared differences between adjusted predictions and labels with the objective of maximizing the QKW.

4.2. Feature Selection

Since we have engineered and extracted a lot of features ranging from simple text statistics, TF-IDF, Count-vectorizer, paragraph, sentence, word level features, DeBERTa embeddings, vocabulary richness, readability, grammatical complexness, and so on we have ended up with 22000 features. To reduce this number to only use the top important ones we have implemented a custom feature selection method using LightGBM with Stratified K-Fold cross-validation. In each fold out of 5 folds data is divided o train and validation splits. Within each fold the LightGBM is trained to maximize the Quadratic Weighted Kappa until the validation score does not increase for 75 iterations. The objective for the LightGBM is QWK, the learning rate is set to low number as 0.01, maximum number of depths is set to 5 which limits the maximum depth of each tree built during the training process. It helps to avoid overfitting. The maximum number of leaves in each tree chosen to be 10. The fraction of features to be used for each tree is 0.3 (colsample_bytree). Finally, L1 regularization (reg_alpha) is 0.7 and L1 regularization is 0.1 (reg_lambda) where the where the former is used to control over-fitting by penalizing large values and latter penalizes the square of the magnitude of the model parameters. Additionally, extra trees are enabled to randomize the thresholds for each split on each feature which makes the model more robust and maximum number of estimators is set to 700 which stands for boosting rounds or trees to build.

Table 1: Parameters table

Parameter	Value	Definition
Objective	QWK (Quadratic Weighted Kappa)	Specifies the custom objective function to be optimized during training.
learning_rate	0.01	Controls the step size at each iteration while moving toward a minimum of the loss function.
max_depth	5	Limits the maximum number of levels in each decision tree.
num_leaves	10	Sets the maximum number of leaves per tree.
colsample_bytree	0.3	Determines the fraction of features to be used for each tree, providing a subsampling of features.
reg_alpha	0.7	Applies L1 regularization on weights, penalizing large values to prevent overfitting.
reg_lambda	0.1	Applies L2 regularization on weights, also aimed at preventing overfitting by penalizing weight size.
n_estimators	700	Specifies the number of trees to build.
extra_trees	True	Enables the Extra Trees method, adding randomness to thresholds for each split to improve model robustness.
class_weight	Balanced	Adjusts weights inversely proportional to class frequencies to address class imbalance.

Algorithm 1 Feature Selection Using Wrapper Method

```
procedure FeatureSelectWrapper
    features ← feature_names
    Initialize skf ← StratifiedKFold(5, True, 0)
    fse ← pd.Series(0, index = features)
    Initialize empty lists: f1_scores, κ_scores, models, predictions
    callbacks ← [log_evaluation(25), early_stopping(75, True)]
    for train_index, test_index in skf.split(X, y_split) do
        X_train_fold, X_test_fold ← X[train_index], X[test_index]
        y_train_fold, y_test_fold, y_test_fold_int ← y[train_index], y[test_index], y_split[test_index]
        Initialize model with specified parameters
        predictor ← model.fit(X_train_fold, y_train_fold, . . .)
        Add predictor to models
        predictions_fold ← predictor.predict(X_test_fold) + a
        oof[test_index] ← predictions_fold
        predictions_fold ← clip and round(predictions_fold)
        Add predictions_fold to predictions
        f1_fold ← f1_score(y_test_fold_int, predictions_fold)
        Add f1_fold to f1_scores
        kappa_fold ← cohen_kappa_score(y_test_fold_int, predictions_fold)
        Add kappa_fold to κ_scores
        Display confusion matrix
        fse += pd.Series(predictor.feature_importances_, features)
    end for
    feature_select ← fse.sort_values(False).index[: 13000]
    return feature_select
end procedure
```

Algorithm 2 LightGBM: Gradient Boosting with Histogram-based Decision Trees

```
Input: Training data {(x_i, y_i)}_{i=1}^n, number of trees M, learning rate η, max depth d, min data in leaf min_data,
  number of bins b
Output: Ensemble of trees {f_m}_{m=1}^M
procedure LightGBM
    Initialize model F_0(x) = 0
    for m = 1 to M do
        Compute gradients g_i = ∂_{F(x_i)} ℓ(y_i, F(x_i))
        Compute hessians h_i = ∂^2_{F(x_i)} ℓ(y_i, F(x_i))
        Apply GOSS to select a subset A ⊂ {1, . . . , n}
        Build histogram for features using data in A
        tree = HistogramBasedTreeLearning(A, g, h)
        Update model F_m(x) = F_{m-1}(x) + η · tree(x)
    end for
    return {F_m}
end procedure
procedure HistogramBasedTreeLearning(A, g, h)
    Initialize tree tree = {}
    nodes = {(root node, A)}
    while nodes ≠ ∅ do
        Select node, split data into subsets based on best split using histograms
        Evaluate splits based on gain Gain = G^2/(H+λ) − (G_L^2/(H_L+λ) + G_R^2/(H_R+λ))
        If depth < d and data in leaf > min_data, add new nodes
        Otherwise, make current node a leaf with value γ = −G/(H+λ)
        Add children to nodes or set current node as leaf
    end while
    return tree
end procedure
```

After feature selection we ended up with 13000 rich and various features to use for the model training. Following a similar approach but this time using 15 folds for cross-validation we trained LightGBM with same parameters that we used for subset selection.

5. RESULTS

Initial baseline model was only using 5000 TF-IDF features and some simple statistics such as number of words, sentences, and characters. And the learning algorithm was Logistic Regression for simplicity. The Cohen Cappa score was around 73 and F1 score which is not significantly important metric, was around 45. As discussed in the feature engineering section we generated various features such as Basic Text Features, Readability Indices, Lexical Diversity and Language Features, Sentence and Paragraph Engineering, Text Processing Features (bag of words), Error Analysis which increased the score substantially. Adding LightGBM on top of those thousands of features increased the final score to 82 with almost 12% improvement over the previous baseline score. From the plot we see that the F1 score also improved to around 67 which is roughly 49% improvement. In the confusion matrix we can see that model performs decent in general and there are errors in consecutive scores such as between score 3 and 2 there are roughly 60 and 55 confusions. As discussed, that is why Cohen's Kappe is more useful here, as this error is acceptable. There are almost no errors between the extreme classes such as 1 and 6 or 2 and 5. The Cohen Kappe score ranges from 82 to 85 between 15 different folds and F1 scores ranges from 65 to 70.

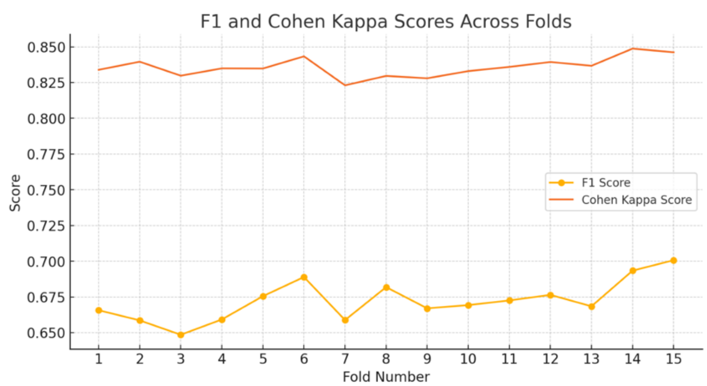

Figure 5. F1 and Cohen Kappa Scores Across Folds.

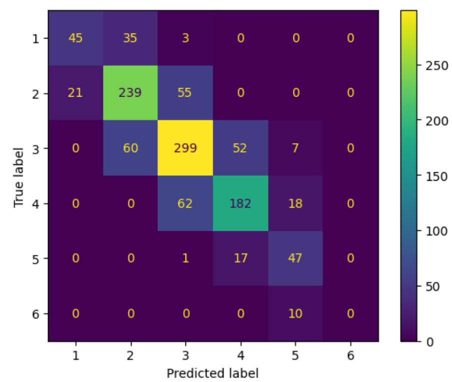

Figure 6. Confusion matrix

Figure 7. Training and Validation QWK Scores by Fold.

6. CONCLUSION

REFERENCES

[1] Taghipour, K., & Ng, H. T. (2016, November). A neural approach to automated essay scoring. In Proceedings of the 2016 conference on empirical methods in natural language processing (pp. 1882-1891).

[2] Cai, C. (2019, March). Automatic essay scoring with recurrent neural network. In Proceedings of the 3rd International Conference on High Performance Compilation, Computing and Communications (pp. 1-7).

[3] Nguyen, H., & Dery, L. (2016). Neural networks for automated essay grading. CS224d Stanford reports, 1-11.
[4] Ludwig, S., Mayer, C., Hansen, C., Eilers, K., & Brandt, S. (2021). Automated essay scoring using transformer models. Psych, 3(4), 897-915.
[5] Rodriguez, P. U., Jafari, A., & Ormerod, C. M. (2019). Language models and automated essay scoring. arXiv preprint arXiv:1909.09482.
[6] Boulanger, D., & Kumar, V. (2018). Deep learning in automated essay scoring. In Intelligent Tutoring Systems: 14th International Conference, ITS 2018, Montreal, QC, Canada, June 11–15, 2018, Proceedings 14 (pp. 294-299). Springer International Publishing.
[7] Lim, C. T., Bong, C. H., Wong, W. S., & Lee, N. K. (2021). A comprehensive review of automated essay scoring (AES) research and development. Pertanika Journal of Science & Technology, 29(3), 1875-1899.
[8] Dıklı, S. (2006). Automated essay scoring. Turkish Online Journal of Distance Education, 7(1), 49-62.
[9] Patil, R. G., & Ali, S. Z. (2018, December). Approaches for automation in assisting evaluator for grading of answer scripts: a survey. In 2018 4th International Conference on Computing Communication and Automation (ICCCA) (pp. 1-6). IEEE.
[10] Sanuvala, G., & Fatima, S. S. (2021, February). A study of automated evaluation of student's examination paper using machine learning techniques. In 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS) (pp. 1049-1054). IEEE.
[11] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), Advances in Neural Information Processing Systems (Vol. 30). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf
[12] He, P., Liu, X., Gao, J., & Chen, W. (2020). Deberta: Decoding-enhanced bert with disentangled attention. arXiv preprint arXiv:2006.03654
[13] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171-4186). Minneapolis, Minnesota: Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423

A APPENDICES

Appendices contains features extracted from essays and their description:

1. Hapax Legomena Ratio: Measures the proportion of words that appear only once in the text.
2. Moving Average Type-Token Ratio (TTR): Calculates the average type-token ratio over a moving window, which indicates lexical diversity.
3. Coleman-Liau Index: Estimates the readability of the text based on character, word, and sentence counts.
4. Gunning Fog Index: Measures the readability of English writing, indicating the number of years of formal education needed to understand the text on a first reading.
5. Automated Readability Index (ARI): Provides a readability score based on characters, words, and sentences.
6. Flesch Reading Ease: Rates text on a 100-point scale; higher scores indicate easier readability.
7. Flesch-Kincaid Grade Level: Translates the Flesch Reading Ease score to a U.S. school grade level.
8. SMOG Index: Estimates the years of education required to understand the text based on the number of polysyllabic words.
9. Dale-Chall Readability Score: Assesses text readability by considering familiar and unfamiliar words.
10. Difficult Words Count: Counts words that are not found on a predefined list of easy words.
11. Linsear Write Formula: Determines the readability of English text aimed at U.S. Air Force technical manuals.
12. Text Standard: An average readability score derived from multiple readability indices.
13. Spache Readability Formula: Measures readability for texts aimed at children up to fourth grade.
14. Polysyllabic Word Count: Counts words with three or more syllables.
15. Monosyllabic Word Count: Counts words with only one syllable.
16. Average Parse Tree Depth: Calculates the average depth of syntactic trees in the text.
17. Entity Density: Measures the density of named entities (e.g., persons, locations, organizations) in the text.
18. Total Distinct N-grams: Counts the distinct n-grams (subsequences of n items) in the text.
19. Average N-gram Frequency: Measures the average frequency of n-grams in the text.
20. Max N-gram Frequency: Identifies the maximum frequency of any n-gram in the text.
21. Max N-gram Frequency Ratio: Ratio of the highest n-gram frequency to the total number of n-grams.

22. N-gram Diversity Score: Evaluates the diversity of n-grams by comparing distinct n-grams to total n-grams.
23. Total Distinct N-grams Ratio: Ratio of distinct n-grams to total n-grams.
24. Syntactic Tree Depth: Measures the maximum depth of syntactic trees in sentences.
25. Type-Token Ratio (TTR): Calculates the ratio of unique words to the total number of words, indicating lexical diversity.
26. Yule's K: A statistical measure of lexical richness.
27. Average Sentence Length: Average number of words per sentence.
28. Average Word Length: Average number of characters per word.
29. Average Clauses per Sentence: Measures the average number of clauses per sentence.
30. Punctuation Diversity: Evaluates the variety of punctuation marks used in the text.
31. Punctuation Density: Measures the density of punctuation marks in the text.
32. Dialogue Marker Frequency: Counts the frequency of dialogue markers (e.g., quotation marks) in the text.
33. Determiners Frequency: Measures the frequency of determiners (e.g., the, a, an) in the text.
34. Prepositions Frequency: Measures the frequency of prepositions in the text.
35. Semantic Diversity: Measures the diversity of semantic content in the text using word vectors and clustering techniques.
36. Estimated Difficult Words Ratio: Ratio of words with three or more syllables to the total word count.
37. Estimated Slightly Difficult Words Ratio: Ratio of words with two or more syllables to the total word count.
38. Measure of Textual Lexical Diversity (MTLD): Evaluates lexical diversity by computing the mean length of sequences that maintain a given type-token ratio threshold.
39. D Measure: A lexical diversity measure based on the relationship between text length and vocabulary size.