Literature Review

After looking through the various approaches done by many other teams on Kaggle, we can see a common approach that was seen to have some of the best accuracy results among other approaches. This approach involves the combination of two models, as well as extensive data preprocessing steps that allow these models to effectively work on the data that was given for the competition. One of the teams with a high accuracy that also published their approach was done by user Cody_Null, and you can see their work here in this link:
https://www.kaggle.com/code/cody11null/lgbm-x2-nn.

In this approach, we can see the utilization of two different models: one of the models is a gradient boosting algorithm, and the other model is a neural network. They used the library LightGBM to accomplish this and proceeded to have 5 runs with this model. The model parameters they used were a 10-fold cross validation technique, where the training of the gradient-boosted model is done with 9/10 of those folds, and the last 1/10 fold is used for prediction error. This error is called OOF error, or out-of-fold error. The parameters used for this gradient boosted decision trees model are reg_alpha: 0.007678095440286993, reg_lambda: 0.34230534302168353, 'colsample_bytree: 0.627061253588415, subsample: 0.854942238828458, learning_rate: 0.04, num_leaves: 22, max_depth: 37, min_child_samples: 18, n_jobs:4.

After running this model with 5 passes, they then use a neural net provided by the library called LightAutoML, which is a library that serves this neural network. They use this model for 3 runs, and this similarly also uses a 10-fold cross-validation method to obtain the OOF error.

Now, after using both of these models and obtaining the OOF error as well as a prediction associated with each of the models, they combine the results utilizing a specific weighted formula to come up with the final prediction. This weight that is used must also be derived from various other elements of the train and test data, as well as it has to be calculated keeping in mind the objective of minimizing the root mean squared error and the true labels for the test data. Once this weight is calculated, the two models' predictions are combined using this formula: (weight * GBM prediction + (1 -

weight) * NN prediction - score)^2. This final formula represents the final prediction that is then used for the test data.

However, easily the biggest part of this approach was the dedicated sections for feature engineering, which is what allowed for these models to be as effective as they are. Using a preexisting essay constructor function, they build multiple different feature engineering functions, such as splitting the essays into sentences, computing the sentence aggregations, splitting the essays into paragraphs, and getting the paragraph aggregations. Applying these changes to the raw data given by Kaggle allows for the prediction error to be minimized in the way that it has been.

Another approach I found was to use a SVM with a chi-square kernel which is linked here: https://www.kaggle.com/code/ianchute/simple-chi-kernel-svm. For this approach, they processed the data by adding indicator variables for the presence of punctuation such as a comma, period, or an enter, as well as converting the units for some of the keystroke measures. They also ensured that all the values were positive, as the SVM would not work with any negative values. After doing that, they used the SVR model from scikit-learn, with an additive chi-square kernel with epsilon and C parameters both being 0.1, and a 10-fold cross-validation.