STAT432 Final Project: Linking Writing Processes to Writing Quality

Sarvagnya Vijay, Jason Vu, Khatija Syeda

December 8th 2023

For more information, visit our GitHub repository: STAT432 Final Project on GitHub.

Project Description and Summary

This project aims to leverage keystroke log data to predict overall writing quality. By analyzing process features such as typing speed, revision patterns, and text structure changes, we seek to uncover relationships between writing behaviors and writing performance. Unlike traditional assessment tools that focus solely on the final written product, our approach emphasizes the writing process, potentially enhancing learner autonomy and metacognitive awareness in writing.

Our methodology involved applying a variety of machine learning models to the keystroke log data, including Elastic Net Regression, K-Nearest Neighbors Regression, Random Forest Regression, and Support Vector Machines Classification. Unique predictors like *"avg_cursor_movement"*, *"avg_text_change_length"*, and *"final_word_count"* were pivotal in our analysis. We also utilized unsupervised learning techniques like UMAP and K-Means Clustering, followed by Principal Component Analysis for dimensionality reduction, to better understand the natural clustering of the data.

Apart from the standard models, we employed Elastic Net Regression for its dual ability to handle multicollinearity and perform variable selection, crucial for our dataset with highly correlated variables. The application of Principal Component Analysis for dimensionality reduction before applying K-Means Clustering is another specialized method. These approaches differ from standard methods by offering a more nuanced understanding of the data structure, crucial for handling our dataset's high dimensionality and intricate variable interactions.

Our findings revealed that Random Forest Regression yielded the most accurate predictions (lowest RMSE), highlighting the importance of variables like *"final_word_count*", "*total_events*", and "*max_cursor_movement*". However, clustering models showed limited efficacy due to the data's complexity, underscoring the challenges in modeling such intricate datasets. This project demonstrates the potential of using process features in writing assessment, paving the way for more nuanced and formative writing evaluation tools.

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.

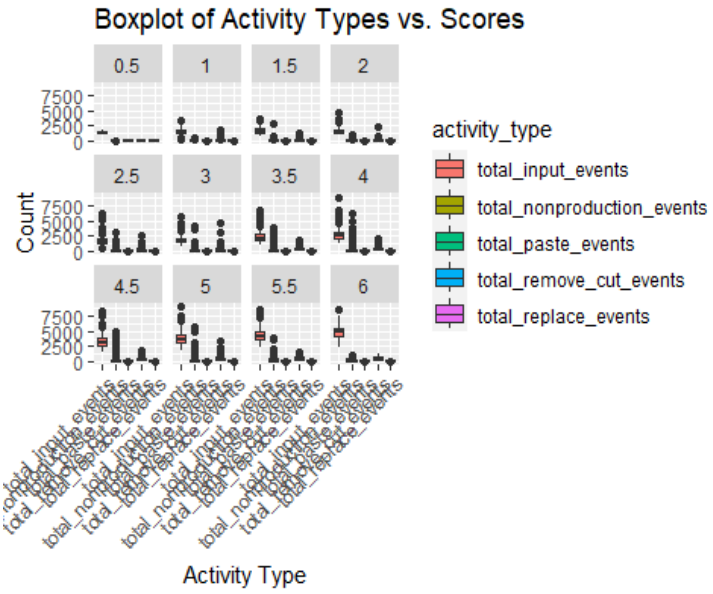
Now, after using both of these models and obtaining the OOF error as well as a prediction associated with each of the models, they combine the results utilizing a specific weighted formula to come up with the final prediction. This weight that is used must also be derived from various other elements 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.

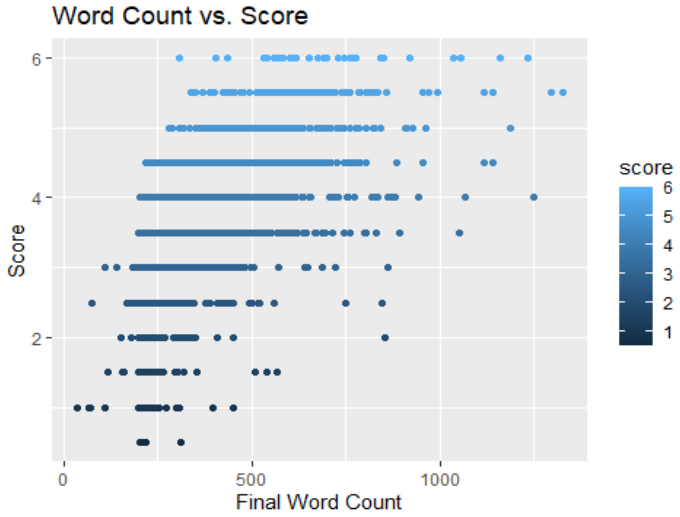
However, easily the biggest part of this approach was the dedicated sections for feature engineering, which is what allowed for these models to be as effective as they are. Using a preexisting essay constructor function, they build multiple different feature engineering functions, such as splitting the 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.

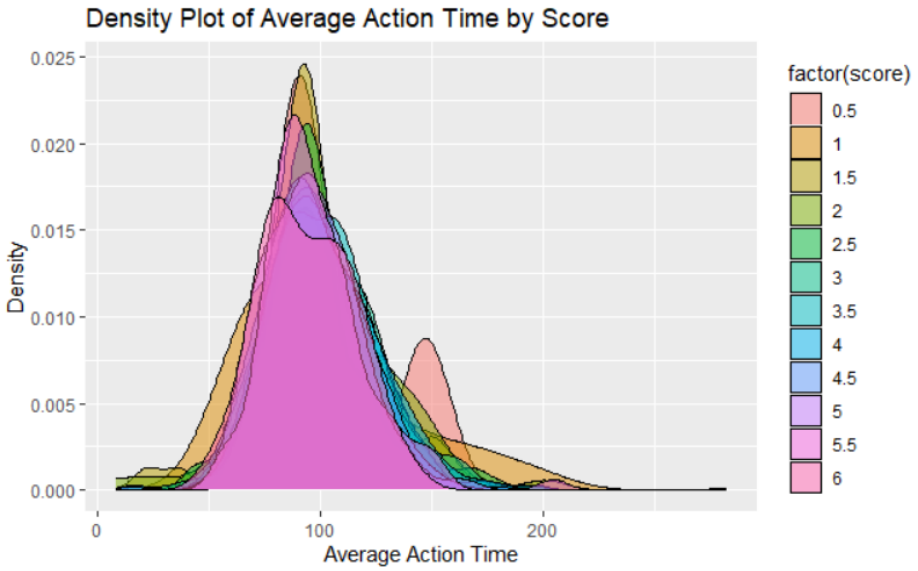
Data Processing:



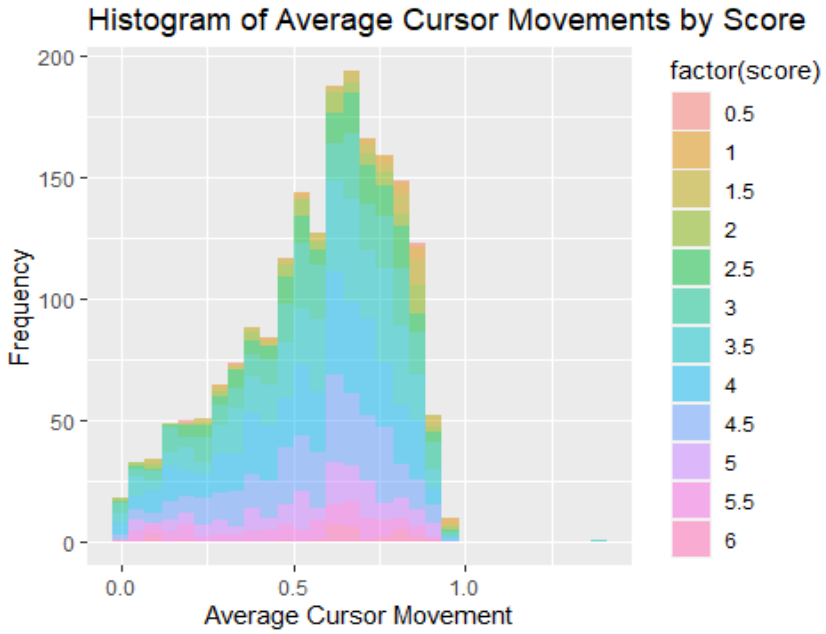
The median counts for most activities seem to increase with the scores up to a certain point, suggesting a correlation between the amount of activity and higher scores. Some activities, like “total_input_events” and “total_replace_events,” have wide variations at higher scores, indicating that these activities are more variable for essays with higher scores. “Total_remove_cut_events” and “total_nonproduction_events” appear to have a more stable count across different scores, suggesting these activities may not be as strongly correlated with the scores as others.



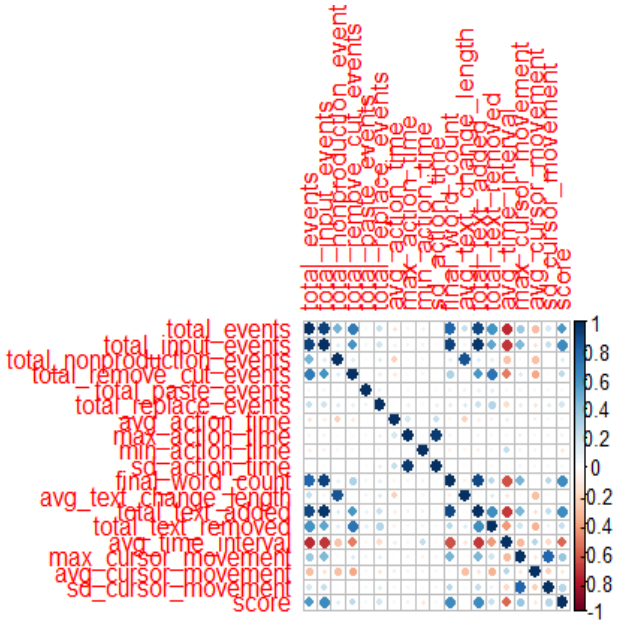
There is a potential correlation between longer essays and higher scores.



There seems to be subtle indication that the highest scoring categories tend to have slightly quicker average action times. Histogram of Cursor Movements: distribution of cursor movements for different scores.



The visual suggests that there may be an optimal range of average cursor movement that is associated with higher scores, while both very low and very high cursor movements are less frequent among higher-scoring activities.



There's a decent amount of collinearity. We will remove some variables that are kind of redundant.

After EDA, we finalized predictors. Each of these predictors can provide insights into the writing process of each essay.

total_events: This is the total number of input events for each essay. A higher number of events might indicate more effort, which could be associated with higher quality writing.

total_nonproduction_events: These events don't alter the text. A high count could imply extensive planning, which might be a trait of careful writing.

total_remove_cut_events: The number of events where text is removed. A higher number might suggest significant editing and refining of the essay, correlating with better content quality.

total_paste_events: Indicates the number of paste actions. This could reflect the use of external references.

total_replace_events: Shows how often text is replaced. Frequent replacements might indicate refinement and improvement of the essay's content and structure.

total_move_events: Counts events of moving text sections. This might be indicative of significant restructuring, suggesting an effort to improve the essay's flow and coherence.

avg_action_time, max_action_time, min_action_time, sd_action_time: These metrics give insights into the duration and variability of actions. Longer action times might indicate more thoughtful writing, while a higher standard deviation could suggest varying writing speeds, possibly reflecting different stages of the writing process.

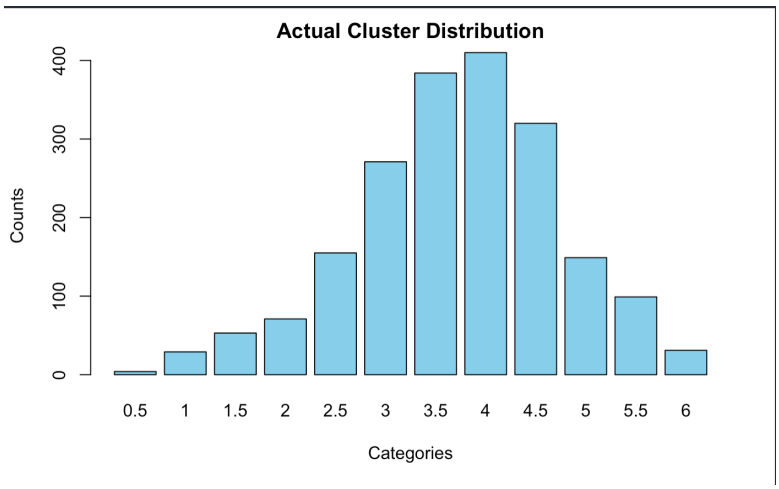
final_word_count: The word count of the essay after the last event. Generally, longer essays might cover more content. This metric helps capture the essay's scope.

avg_text_change_length, total_text_removed: These features relate to the extent of text changes. Large amounts of text change might indicate significant editing and refinement.

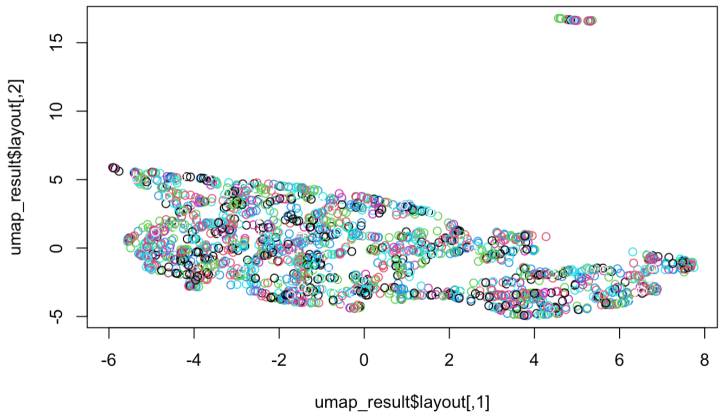
max_cursor_movement, avg_cursor_movement, sd_cursor_movement: These metrics measure how much the cursor position changes, reflecting the extent of navigation and editing. Significant movement might indicate active organization and editing of the essay.

For this data, as we have 17 predictors, and for the response, 12 unique labels: scores from 0.5 to 6 in increments of 12. We can already see a problem of high dimensionality which could affect our model. Before we jump to these conclusions, we can first fit several unsupervised learning models to see if we can glean any insights into the natural clustering of the data. The clustering algorithms we chose to use are the UMAP (Uniform Manifold Approximation and Projection), a K-Means clustering algorithm, and then we apply Principal Component Analysis to the data in order to reduce dimensionality, before fitting a second K-Means clustering algorithm.

We first want to know the actual cluster distribution for reference so we have a sense of how each model performs. I've plotted it on a bar chart to visualize how the scores are distributed.



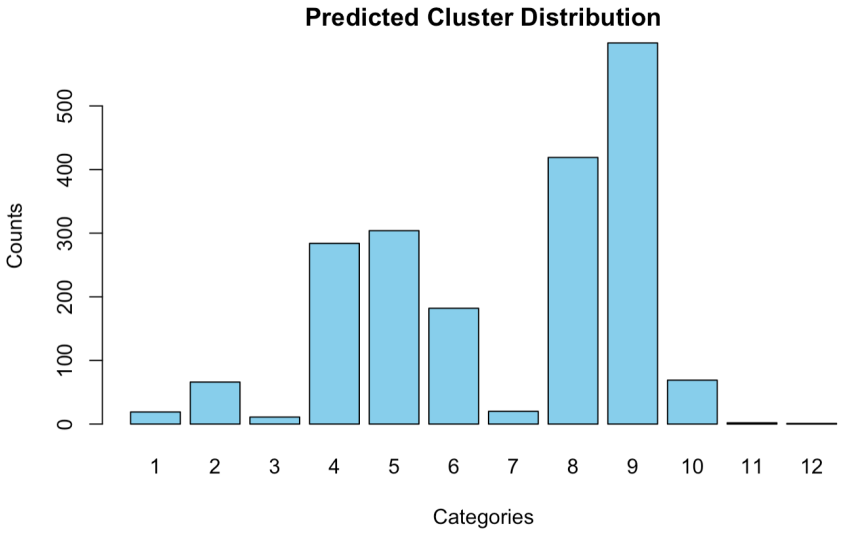
First, we fit a UMAP model to gain insights into the data.



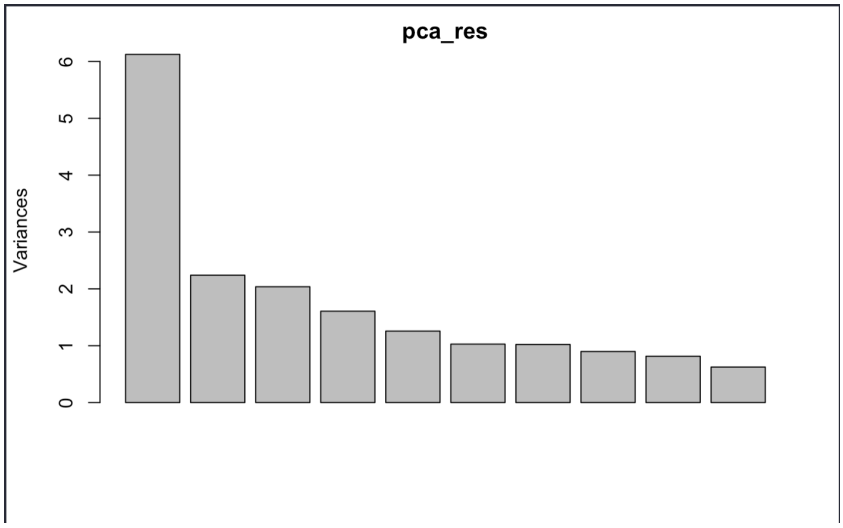
We can see that this data does not form nice clusters by itself, and it may lead to bad results without any form of dimension reduction. We can see this for ourselves by first fitting a preliminary K-Means clustering model with this data. We can try plotting the centroids by cluster label, but due to the high number of predictors, this is not too useful.

As we can see here, there is a very high dimensionality, which may possibly lead to issues with our model. Due to this aforementioned high dimensionality, we can take a look at the distributions of cluster labels to see if this model did a good job. If the distribution is similar, we can correctly assume the model performed adequately. To accomplish this, we use a bar chart that plots the counts of each of the labels.

From this bar chart, we can see that the general shape is followed, but the predicted

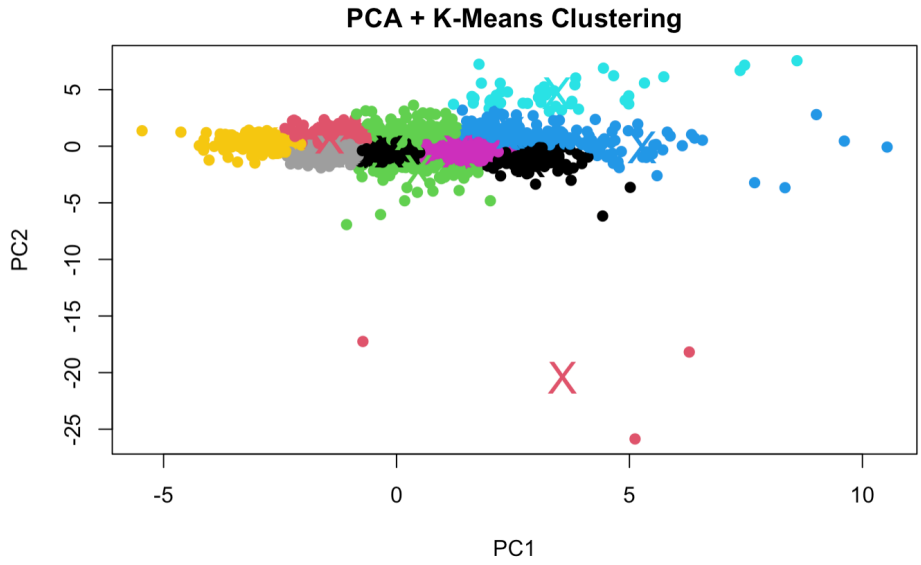


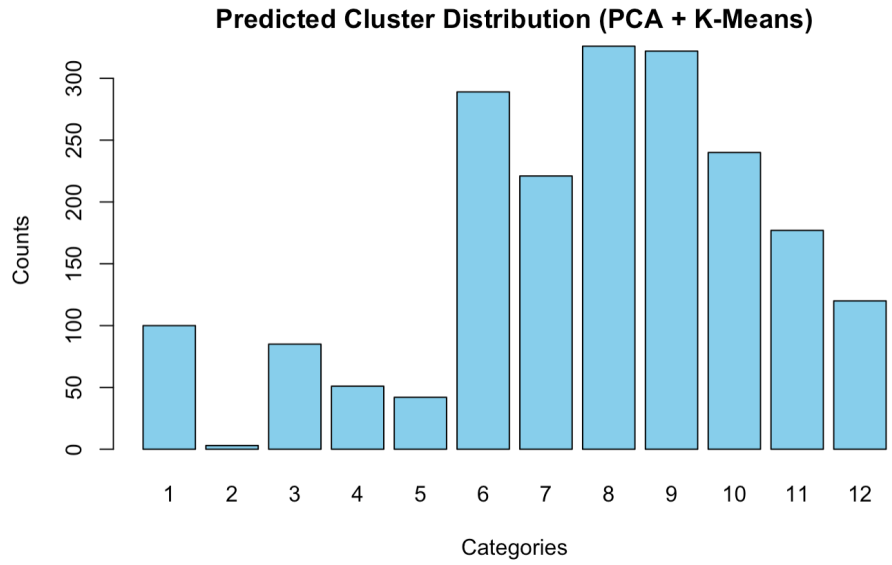
cluster distribution is definitely lacking some information. Therefore, simply the K-Means model may not be enough to adequately represent this data. We saw an issue with dimensionality, as the data has seventeen predictors, so using Principal Component Analysis as a method of dimensionality reduction, followed by a fitted of a new K-Means model may lead to better results. After running the PCA, we can take a look at the principal components ordered by how much variance they explain.



As we can see from the graph of principal components, the ones that explain the most variance in the data are approximately the first 2-4 components. For our purposes, we will retain the first 2 components as our new reduced dataset. We then fit a K-Means

model with this data, utilizing 12 clusters as a baseline as that is how many unique scores there are. After fitting a K-Means model with the reduced data, we can produce the clustering plot as follows, and we can also take a look at the predicted cluster distributions.





From this plot, we can see that the reduced data clusters much more nicely, with a few exceptions here and there. Furthermore, most of the centroids were accurately predicted using K-Means, and the data forms well-shaped clusters, which indicates a better model fit. The dimensionality reduction offered by Principal Component Analysis definitely helped, and allowed for a better fit. However, it is clear to see that this data works poorly with clustering algorithms due to the sheer number of predictors and labels, which is why supervised learning may be a better approach to model this data.

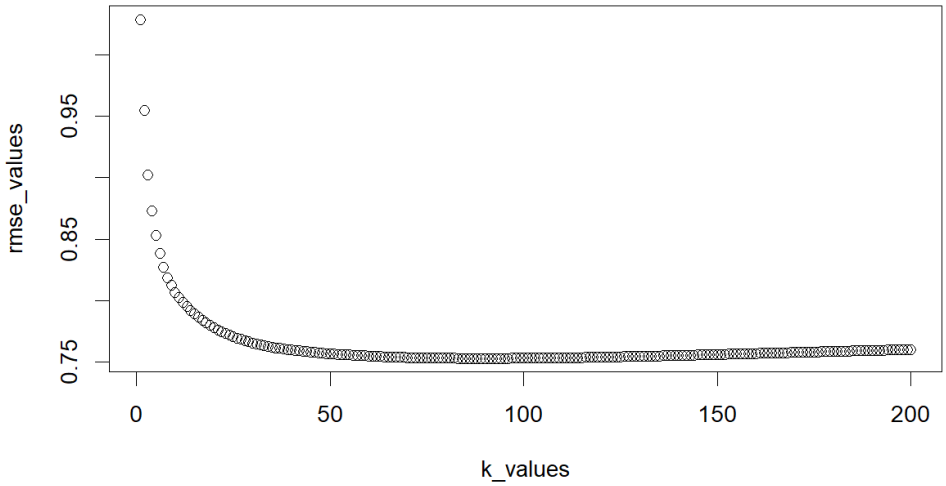
The regression / classification models we decided to run on the data are the following: elastic net model, k-nearest neighbors regression, random forest regression, and support vector machines classification. Our evaluation criteria for all of our models was root-mean-square error (RMSE).

For the first model, we decided to use elastic net regression. As a balance of Ridge and LASSO regression, the elastic net regression model is useful for handling both multicollinearity issues and also doing variable selection (i.e. removing variables that may not be as important in predicting the outcome). As noted in the Data Processing section, many variables (e.g. “total_events” and “total_input_events” and “final_word_count” and “total_text_added”) are highly correlated, simply by the nature of how these variables were processed from the original dataset. Furthermore, since we have not performed variable selection yet, this aspect of elastic net regression will be helpful in our analysis.

Using a sequence of alpha values from 0 to 100, we fit models with the alpha values, found the optimal lambda for these models, and compared their performance in prediction in RMSE. The optimal combination of alpha and lambda that resulted in the lowest possible RMSE were as follows: alpha = 0.98, lambda = 0.002004471, with an RMSE = 0.7546587. What is interesting to note is that alpha = 0.98, which is very close to alpha = 1, which is just LASSO regression.

Inspecting the coefficients, we see that elastic net removed the variables "sd_action_time" and "max_cursor_movement" indicating that they may not be important for predicting the score of the row. Noting the largest coefficients by magnitude, we note that the variables "avg_text_change_length", "avg_cursor_movement", "min_action_time" seem important for predicting the score.

For the second model, we decided to use k-nearest neighbors. Similarly, we tune k by creating models with k = 1 to k = 200 and compare the performance of all of these models. We found that the optimal k is k = 87, which returned RMSE = 0.7528342.



We can see that the RMSE has a sharp drop in the beginning as the bias increases and variance sharply decreases. However, the RMSE levels off and eventually starts increasing as bias continues to increase and the model overfits, demonstrating the bias-variance tradeoff involved in tuning k for k-nearest neighbors algorithm.

The next model we tried was random forest regression. We note that random forest has more parameters to tune than the previous algorithms: namely, ntree, sampsize, mtry, and nodesize. Using a grid search to search every possible combination of these parameters and comparing all of their performances in prediction, we found that the optimal combination of parameters was as follows: ntree = 700, sampsize = 110, mtry = 13, nodesize = 1, with RMSE = 0.6829625. This is the best RMSE we have gotten thus far.

Inspecting variable importance from the model, we find that the three most important variables are "final_word_count", "total_events", and "max_cursor_movement", in terms of Increase in Node Purity (IncNodePurity).

The classification model we fit was a support vector machine classification. We did this using the svm() function from the e1071 library, which does classification when the outcome variable is specified as a factor. Cost is a parameter that we can tune for this model, and it affects the "cost" of misclassification. In other words, with a larger cost, the model may have a more complex and jumpy decision boundary to avoid misclassification. Optimizing this parameter with RMSE as we did with the previous models, we got an optimal cost = 1 with an RMSE = 3.898977.

```
[1] "Confusion Matrix:"
      Predicted
Actual 0.5  1 1.5  2 2.5  3 3.5  4 4.5  5 5.5  6
   0.5   0  0   0  0   0  1   0  0   0  0   0  0
   1     0  0   0  0   0  6   0  0   0  0   0  0
   1.5   0  0   0  0   0  9   5  2   0  0   0  0
   2     0  0   0  0   0 15   6  0   0  0   0  0
   2.5   0  0   0  0   0 33  10  3   0  0   0  0
   3     0  0   0  0   1 31  19 10   4  0   0  0
   3.5   0  0   0  0   0 29  41 24   8  0   0  0
   4     0  0   0  0   0 14  22 37  17  0   1  0
   4.5   0  0   0  0   0  2  10 29  41  0   0  0
   5     0  0   0  0   0  1   2  6  19  0   2  0
   5.5   0  0   0  0   0  0   1  4  21  0   3  0
   6     0  0   0  0   0  0   0  2   3  0   1  0
Accuracy: 0.3090909
Root Mean Squared Error (RMSE): 3.898977
```

Looking at the classification confusion matrix and accuracy = 30.90909%, we see that performance of this model is quite bad. The model tends to confuse many of the scores in the 2.5 to 4.5 range, which, as we noted in the Data Processing section tends to be where the majority of data points lie.

Just for the sake of curiosity, we also fit an svm regression model and did the optimization for cost. This model performed much better with an optimal cost = 75 correspondent to an optimal RMSE = 0.7580726.

	Model <chr>	RMSE <dbl>
3	random forests	0.6829625
2	k nearest neighbors	0.7528342
1	elastic net	0.7546587
5	svm regression	0.7580726
4	svm classification	3.8989770

The performances of all of our models relative to each other are summarized in the table above. We see that random forests has the lowest RMSE, while svm classification has the largest RMSE by a significant margin.

Elastic net coefficients:

```
(Intercept)                 2.285071e+00
total_events                3.845065e-04
total_nonproduction_events -1.903365e-04
total_remove_cut_events    -4.182445e-04
total_paste_events         -1.161791e-03
total_replace_events        1.840437e-02
avg_action_time            -1.252771e-04
max_action_time             1.594265e-06
min_action_time            -4.849160e-02
sd_action_time              .
final_word_count            1.049640e-03
avg_text_change_length     -9.385741e-02
total_text_removed         -1.689733e-04
max_cursor_movement         .
avg_cursor_movement        -7.187129e-02
sd_cursor_movement          5.070199e-03
```

```
 [1] "(Intercept)"                "avg_text_change_length"     "avg_cursor_movement"
 [4] "min_action_time"            "total_replace_events"       "sd_cursor_movement"
 [7] "total_paste_events"         "final_word_count"           "total_remove_cut_events"
[10] "total_events"               "total_nonproduction_events" "total_text_removed"
[13] "avg_action_time"            "max_action_time"            "sd_action_time"
[16] "max_cursor_movement"
```

Random forest variable importance

```
                          IncNodePurity
total_events                 17.4698577
total_nonproduction_events    3.7838847
total_remove_cut_events       3.9744701
total_paste_events            0.4763130
total_replace_events          1.8217903
avg_action_time               4.4580144
max_action_time               4.0362660
min_action_time               0.3842305
sd_action_time                3.6217054
final_word_count             51.0266429
avg_text_change_length        4.1440921
total_text_removed            3.9363097
max_cursor_movement           5.3090077
avg_cursor_movement           4.5206313
sd_cursor_movement            4.7586477
```

```
 [1] "final_word_count"           "total_events"               "max_cursor_movement"
 [4] "sd_cursor_movement"         "avg_cursor_movement"        "avg_action_time"
 [7] "avg_text_change_length"     "max_action_time"            "total_remove_cut_events"
[10] "total_text_removed"         "total_nonproduction_events" "sd_action_time"
[13] "total_replace_events"       "total_paste_events"         "min_action_time"
```

If we compare these two lists of variable importance and identify which variables are important in both models, we find that the most important variables are “avg_cursor_movement”, “avg_text_change_length”, “final_word_count”. More active cursor movement could suggest more editing and refinement of the essay. Likewise, higher active text change length could mean there were multiple revisions of an essay, which could lead to a better essay. Finally, a higher final word count could mean the author has been very detailed in their explanations, which could lead to a better essay.