Analysis Fall 23

Abhi Thanvi (athanvi2), Jonathan Sneh (jsneh2)

2023-12-04

Contents

Project Overview

We highly recommend everyone to checkout our Github Repository for all the data cleaning, feature engineering, analysis, and other files! Many of our procedures are not included in the report, and the repo provides a behind-the-scenes access to that information! We understand it is important to understand each procedure but also reproduce results, therefore we have maintained a readable code-base for it! :)

Goals

While this project is an assigned project for STAT 432 Fall 2023 (UIUC), our goal is to apply what we have learned in our class and generate not necessarily the "most accurate", but rather the most holistic solution on this unique problem. The topic we are dealing with Linking Writing Processes to Writing Quality where we explore data on typing behavior to predict essay quality between a score of 0-6 (inclusive) using many of the statistical learning techniques. Our work will help explore the relationship between learners' writing behaviors and writing performance, which could provide valuable key insights for writing instruction, the development of automated writing evaluation techniques, and help in educational situations.

Approach

We divide our work into three main section.

- **Data Processing** → Cleanse, extract, and engineer features for our Supervised and Unsupervised learning portion. Our data processing procedure also has some basic EDA work done to allow us to engineer valuable features.
- **Unsupervised Learning** → Perform clustering algorithms on the cleansed data (and 80-20 split). We chose to do K-Means and Hierarchical Clustering algorithms as we wanted to explore what we learned in class. These unsupervised techniques allow us to find hidden patterns in our data and act as an outlet for advanced EDA. How the clusters were chosen, what insights we drew, the good and bad about these clusters will all be explored later in this section.
- **Regression/Classification Models** → It is in this section where we try to predict scores and aim to achieve our original goal. TBD

Unique Approaches/Techniques:

We generally tried to use as much as we could from our STAT 432 course materials as the source of knowlegde. There were moments where we did consult other topics such as pairing elbow method with Silhouette Plots (to determine how well the cluster fit), **JONATHAN ADD YOUR STUFF HERE**...prolly the STAT 426 stuff? :D

Conclusion: **TBD**

Literature Review

We aim to understand our pursuit of understanding and addressing the problem statement, we embarked on a journey of reviewing existing submissions to gain insights and inspiration. This preliminary exploration aimed to familiarize ourselves with various approaches employed by others in the field.

Noteworthy Submissions

Upon reviewing the top-performing submissions, we observed a prevalent trend—the use of topics beyond the scope of our class and a preference for Python implementation. Among these submissions, one that stood out was by Maok Yongsuk's Noteboook which achieved the highest public score of 0.584.

Maok Yongsuk's Approach

Maok's methodology deviated from the conventional as he introduced an essay constructor for tokenization. This innovative approach facilitated the extraction of meaningful features from textual data, enabling the model to capture intricate patterns within essays. Notably, he opted for Python's `LightGBM` function, a variant of the XGBoost framework employing a leaf-wise tree growth strategy. This strategic choice results in a more balanced and potentially shallower tree, enhancing efficiency on large datasets. Another key aspect that caught our attention was Maok's intelligent use of feature engineering. Inspired by others in the field, he incorporated a set of features commonly utilized by Kaggle community members. These features, identified through rigorous exploration and collaboration, added valuable information to his model.

Other Submission Approaches

In addition to Maok Yongsuk's approach, several other submissions demonstrated the utilization of topics outside our coursework. Notably, deep learning-based solutions and the combination of neural networks (NN) with `LightGBM` were prevalent strategies. Deep learning-based solutions and the integration of neural networks (NN) with LightGBM offer the potential to enhance prediction accuracy and create adaptive models. However, these approaches come with notable challenges. Firstly, they often incur increased computation costs, demanding substantial resources for training and potentially limiting their feasibility in resource-constrained projects. The inherent complexity of deep learning models poses interpretability challenges, hindering a clear understanding of their inner workings.

All in all, It was a common thread among all submissions to engage in feature engineering that made sense during the exploratory data analysis (EDA) process, which is a practice we intend to adopt. Furthermore, there is a consideration of exploring gradient boosting techniques for our project, aligning with what we know/learned from class and the strategies employed by successful submissions.

Implications for Our Project

The incorporation of `LightGBM` for predictions, the utilization of engineered features through tokenization (including those derived from the essay constructor), and the integration of additional features sourced from the Kaggle community collectively contributed to Maok's remarkable achievement with a highest public score of 0.584. In considering our own project, we find inspiration in these ideas from all submissions we reviewed, evaluating their relevance to our class material and potential applicability to our unique context. As we progress, we aim to integrate insights from Maok's and other Kaggle submission approaches into our methodology, adapting and refining these concepts to enhance the robustness of our own model.

Citations

- Maok Yongsuk's Kaggle Submission: Maok Yongsuk
- Other Submission: Cody's LGBM + NN
- Other Submission 2 (LGBM + NN): Seoyunje's LGBM + NN

- Research 1: Research - Silhouette Plots
- Research 2: Ordinal Logistic Regression

Data Processing

This section dives into the tasks performed for data processing. All the steps ensure the specifications of the projects were met, but some decisions were also made to ensure a more practical data to work with. To be considerate of the pages used for the Data Processing, we performed our Data Engineering steps in a `jupyter notebook` that you can view in our repo!

Feature Engineering

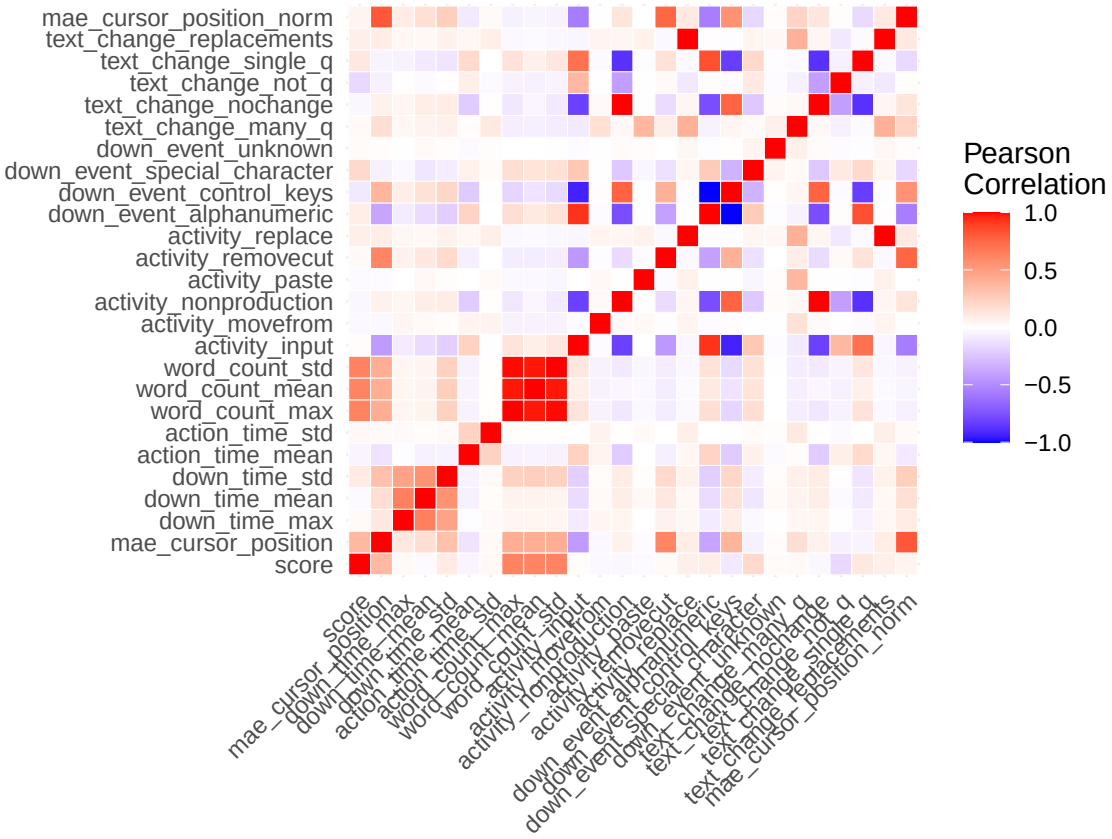
- **User ID** [`id`, `string`] — Unique IDs of each user.
  - We keep this to ensure tracking of user information for processing and analysis work.
- **Event ID** [`event_id`, `string`] — Incremental ID log of all events.
  - We keep this for processing steps, but remove it prior to analysis. The event IDs are useful as an ordinal feature of the log data.
- **Down Time / Up Time** [`down_time` / `up_time`, `integer`] — Time of event on down and up strokes of key or button, in seconds.
  - We summarize these features as an array of summary statistics; min, max, mean, median, and standard deviation. Measures of interest are max (i.e., how long a paper is written) and mean/median (i.e., when the center of most activity is).
- **Action Time** [`action_time`, `integer`] — Difference of time between down time and up time of event, i.e., duration of action in seconds.
  - Similarly, we summarize this feature as min, max, mean, median, and std. This gives insight into "major" consecutive actions, hesitancy, or other special behaviors.
- **Activity** [`activity`, `string`] — Actions to edit or modify the text (input, remove/cut, nonproduction, etc.)
  - We compute the proportions of each of these activities. All of the cursor "Move From" events are mapped to one category called "Move From". We choose proportions over count to avoid undue influence of essays that take longer to write.
- **Down Event**
  - We compute the proportions of each of the activities. The events were pooled into four categories: alphanumeric, special_characters, control_keys, and unknown.
- **Up event**
  - Since these are the same events as down events, we ignore this feature.
- **Text Change**
  - We process and cluster these values into identified patterns of changes: many characters (at least 2 alphanumeric), at least one character (exactly one alphanumeric), non-zero characters (no alphanumeric). We also identified "transition" groups of "X to Y" for each of "many", "single", "none" (e.g., "many" to "many", "many" to "single", "many" to "none", etc.). There was also a "no change" group. We created one additional group to represent the sum of all "transition" events because they coincided exclusively with "replacement" activities.
- **Cursor Position**
  - We computed an artificial array of cursor positions with the assumption that the text was streamed with no edits corresponding to what text changes there are (i.e., non-decreasing and doesn't change if "no change" is observed in text change feature). Then we compute the MAE error metric between this stream version and the actual cursor positions to measure how much error exists between them. Greater errors imply more frequent and/or drastic changes.
- **Word Count**
  - We summarize these features as an array of summary statistics; min, max, mean, median, and standard deviation. We are primarily interested in the maximum measure as it indicates the length of the paper for each user.

Data Summary

Here is a sample of the processed data.

	001519c8	0022f953	0042269b	0059420b	0075873a	0081af50
id	001519c8	0022f953	0042269b	0059420b	0075873a	0081af50
score	3.5	3.5	6.0	2.0	4.0	2.0
mae_cursor_position	527.0469	380.7747	1238.7553	152.3933	640.1616	423.8706
down_time_max	1801877	1788842	1771219	1404394	1662390	1778845
down_time_mean	848180.8	518855.3	828491.8	785483.0	713354.2	544339.2
down_time_std	395112.7	384959.4	489500.8	385205.0	405576.4	484650.6
action_time_mean	116.24677	112.22127	101.83777	121.84833	123.94390	81.40434
action_time_std	91.79737	55.43119	82.38377	113.76823	62.08201	40.65305
word_count_max	256	323	404	206	252	275
word_count_mean	128.1162	182.7148	194.7727	103.6189	125.0830	132.9426
word_count_std	76.49837	97.76309	108.93507	61.88225	77.25505	81.20882
activity_input	0.7860774	0.7897311	0.8498549	0.8380463	0.7672857	0.811397
activity_movefrom	0.00117325	0.00000000	0.00000000	0.00000000	0.00000000	0.000000
activity_nonproduction	0.04693000	0.10350448	0.04231141	0.06362468	0.02844725	0.034373
activity_paste	0.0000000000	0.0004074980	0.0000000000	0.0006426735	0.0000000000	0.000000
activity_removecut	0.1630817	0.1059495	0.1061412	0.0970437	0.2042671	0.152872
activity_replace	0.0027375831	0.0004074980	0.0016924565	0.0006426735	0.0000000000	0.001356
down_event_alphanumeric	0.6331639	0.6071720	0.7021277	0.6709512	0.6088503	0.656264
down_event_control_keys	0.3523661	0.3712306	0.2860251	0.3155527	0.3646780	0.336499
down_event_special_character	0.014470082	0.021597392	0.011847195	0.013496144	0.026471750	0.007236
down_event_unknown	0	0	0	0	0	0
text_change_many_q	0.0007821666	0.0000000000	0.0007253385	0.0006426735	0.0000000000	0.000452
text_change_nochange	0.04693000	0.10350448	0.04231141	0.06362468	0.02844725	0.034373
text_change_not_q	0.1908487	0.2041565	0.1677950	0.1985861	0.1955749	0.180461
text_change_single_q	0.7587016	0.6919315	0.7874758	0.7365039	0.7759779	0.783355
text_change_replacements	0.0027375831	0.0004074980	0.0016924565	0.0006426735	0.0000000000	0.001356

Correlation Summary



Discussion

We observe some pairs of features that show signs of multicolinearity.

- The word count metrics are highly correlated as expected, we could reasonably choose the maximum measure to use.
- Some features form parallel or perpendicular colinearity.
  - activity_input and activity_nonproduction (negative)
  - activity_input and down_event_alphanumeric (positive)
  - activity_input and down_event_control_keys (negative)
  - activity_input and text_change_nochange (negative)
- Importantly, we're interested in what's correlated with the user score feature
  - word count measures have a positive correlation with score, suggesting an association between longer essays and higher scores
  - Error rate of cursor positions against a "streamed" output also shows a positive correlation with score - i.e., essays written with less frequent or extreme edits is somewhat associated with higher scores.
  - Note: a positive correlation is also found between error rate of cursor positions with max word count, suggesting further that longer essays are associated with higher deviation from a "streamed" output. This suggests the possibility that interpretation of "streamed" deviation is influenced by the paper length (i.e., longer papers support possibility of edits being made "further away" from the current "streamed" position, thus increasing the error rate). When we normalized the error rate by the paper size, we see that the correlation between the normalized error rate and the paper score is nearly zero. So, this feature is likely irrelevant for analysis.

Unsupervised Learning Algorithms

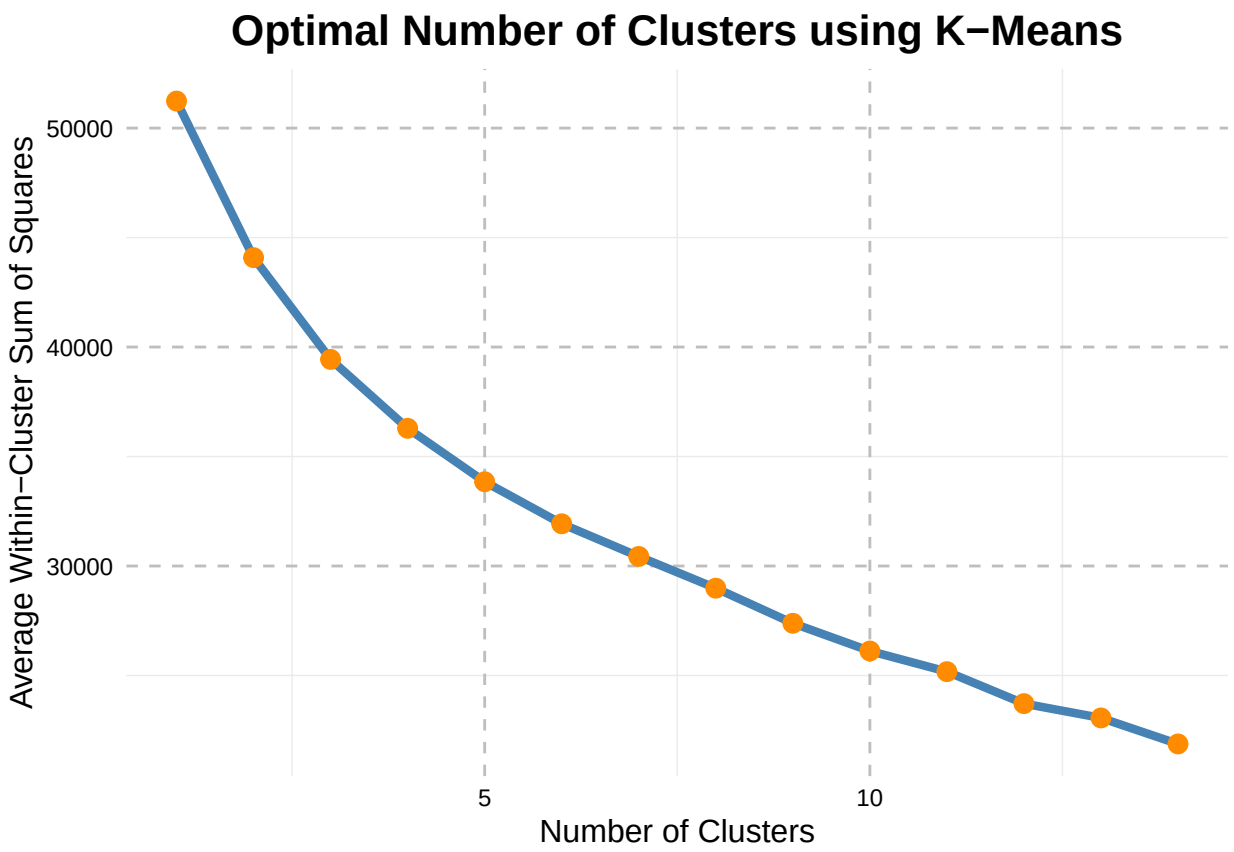
This section dives into the tasks performed for the unsupervised learning algorithms. Currently, focusing on K-Means and Hierarchical Clustering. **THIS SECTION SUPER MESSY RN. Please feel free to edit or improve in any way**

K-Means Algorithm

using averaged inertia of a few clustering samples across a range of number of clusters (i.e., k=1, . . . , 14)

```
# Range of k values to try
num_clusters <- 1:14

# Initialize a vector to store average within-cluster sum of squares (WCSS)
avg_wcss_list <- numeric(length(num_clusters))

# Iterate over different values of k
for (k in num_clusters) {
  sub_wcss_list <- numeric(3) # For storing WCSS of each trial
  for (i in 1:3) {
    set.seed(i) # Setting seed for reproducibility
    kmeans_result <- kmeans(train, centers=k, nstart=25, iter.max = 50)
    sub_wcss_list[i] <- kmeans_result$tot.withinss
  }
  avg_wcss_list[k] <- mean(sub_wcss_list)
}

# Plotting the elbow plot with improved colors and labels
ggplot(data.frame(Clusters=num_clusters, WCSS=avg_wcss_list),
       aes(x=Clusters, y=WCSS)) +
  geom_line(color = "steelblue", size = 1.5) +  # Simpler blue color
  geom_point(color = "darkorange", size = 3) +  # Orange points
  theme_minimal() +
  theme(panel.grid.major = element_line(color = "gray", linetype = "dashed"),  # Add grid lines
        axis.text = element_text(color = "black"),  # Ensure text color is readable
        plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),  # Centered title
        axis.title = element_text(size = 12)) +  # Adjust axis title size
  ggtitle("Optimal Number of Clusters using K-Means") +
  xlab("Number of Clusters") +
  ylab("Average Within-Cluster Sum of Squares")
```

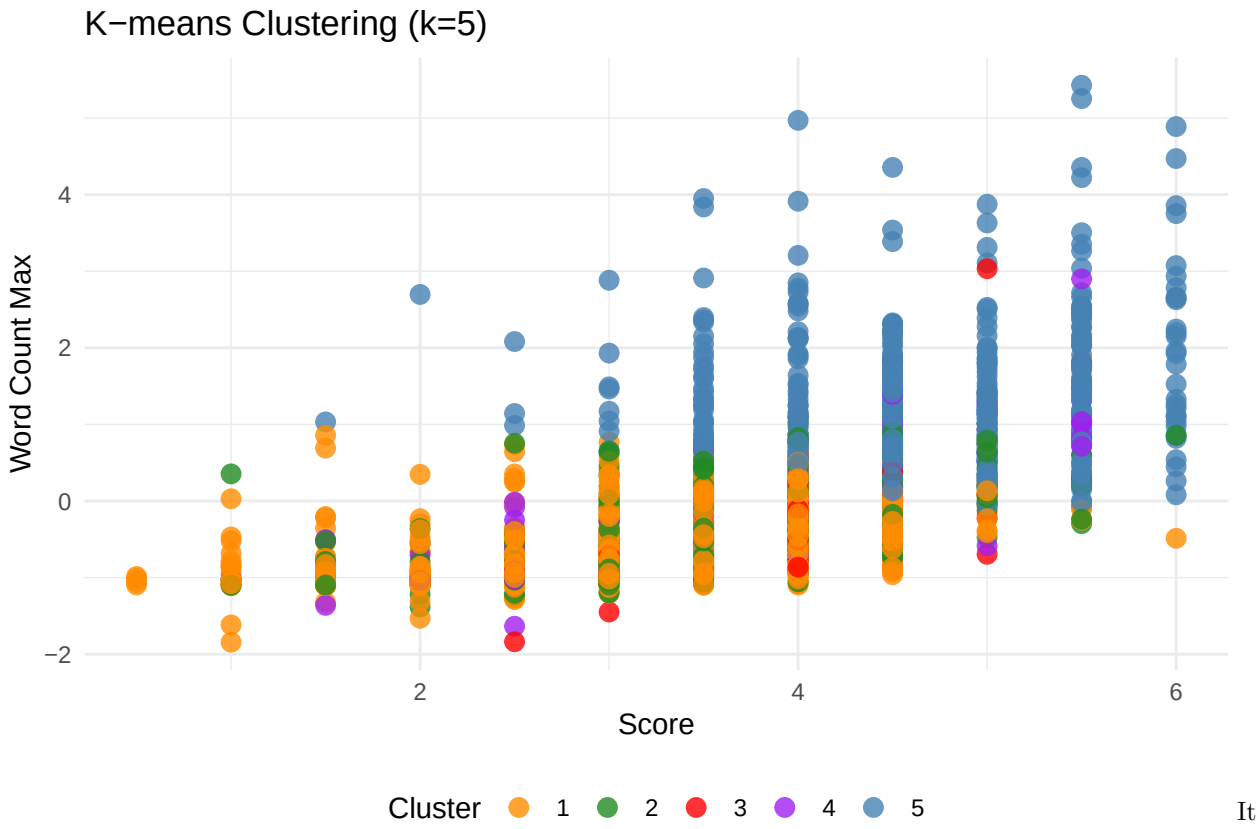
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Although slightly hard to see the elbow, we selected `k=5` clusters as closest to "elbow" of the above plot.

```
set.seed(100)
# Perform K-means clustering
# Here, we are specifying 5 clusters, but you can change this number
result <- kmeans(train, centers=5)
```

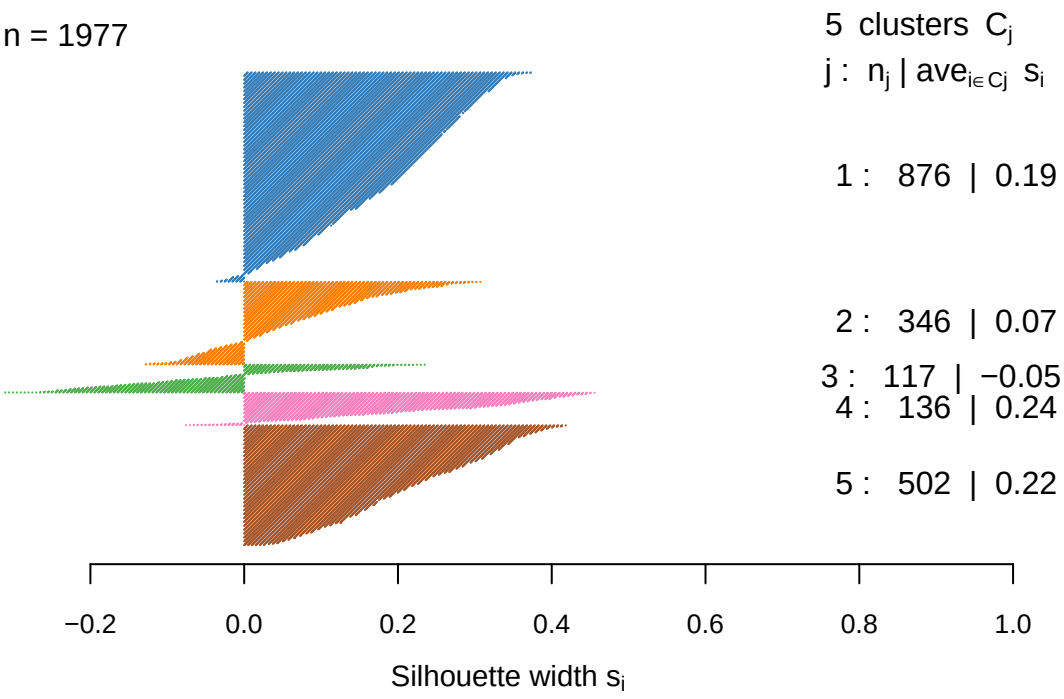
```
# Plotting the K-means clustering results
ggplot(data.frame(train), aes(x=score, y=word_count_max)) +
  geom_point(aes(color=factor(result$cluster)),
             size = 3, alpha = 0.8) +
  scale_color_manual(name="Cluster",
                     values = c("darkorange", "forestgreen",
                                "red", "purple", "steelblue")) +
  theme_minimal() +
  theme(legend.position="bottom") +
  ggtitle("K-means Clustering (k=5)") +
  xlab("Score") +
  ylab("Word Count Max")
```



It seems like the clusters aren't clearly distinct and have overlapping present from the above graph.

Therefore, we used silhouette scores to evaluate how well the clustering structure fits in terms of similarity, i.e., more higher scores in a cluster imply greater similarity of points to its own cluster and poorer similarity to other clusters.

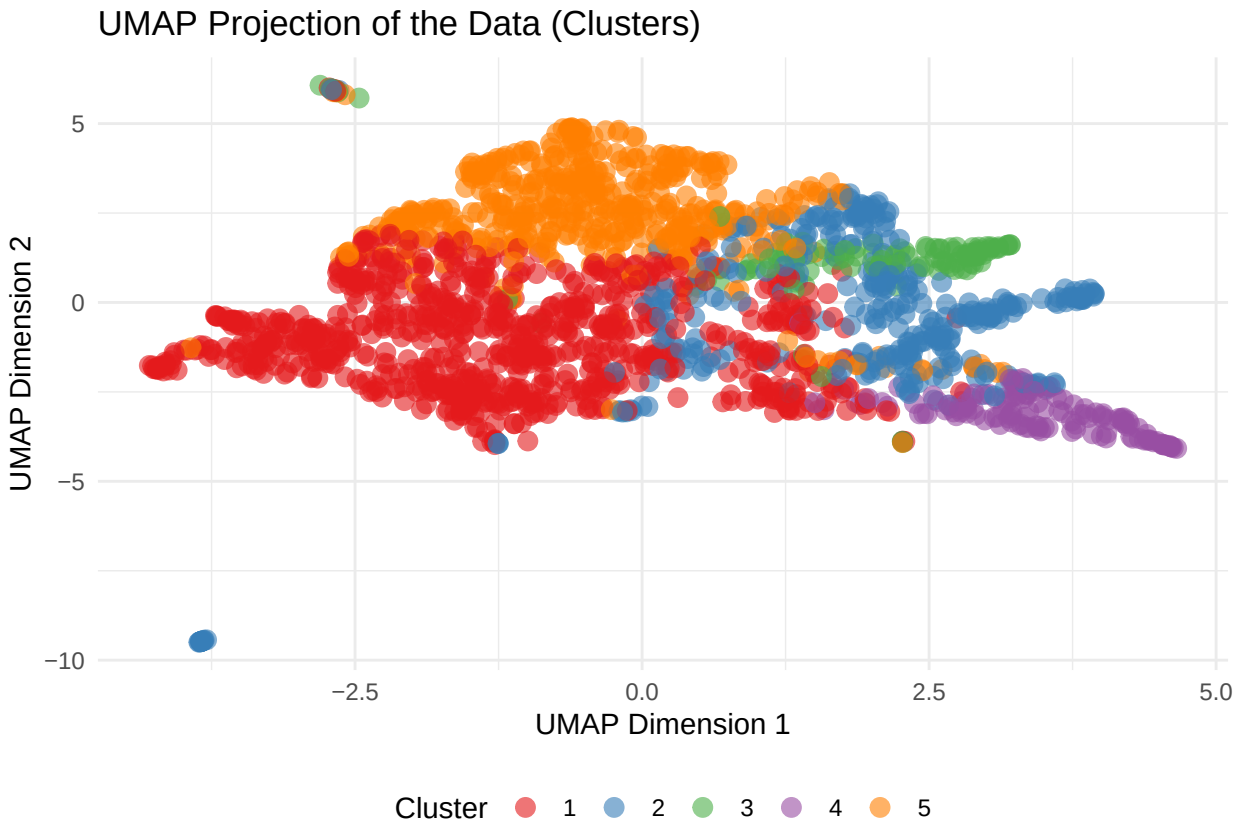
```
# Compute silhouette information
silhouette_info <- silhouette(result$cluster, dist(train))

# Define a custom color palette for silhouette plot
custom_palette <- c("#377eb8", "#ff7f00", "#4daf4a", "#f781bf", "#a65628")

# Plotting the silhouette plot with improved colors and organization
plot(silhouette_info, col=custom_palette, border=NA, main="Silhouette Plot",
     cex.names=0.8, cex.axis=0.8, cex.main=1, cex.lab=0.9, density=70)
```



The average silhouette score of 0.17 suggests that the clusters are somewhat favorably well separated and that the points within clusters aren't too dispersed. However, cluster 4 shows issues with cohesion and separation.

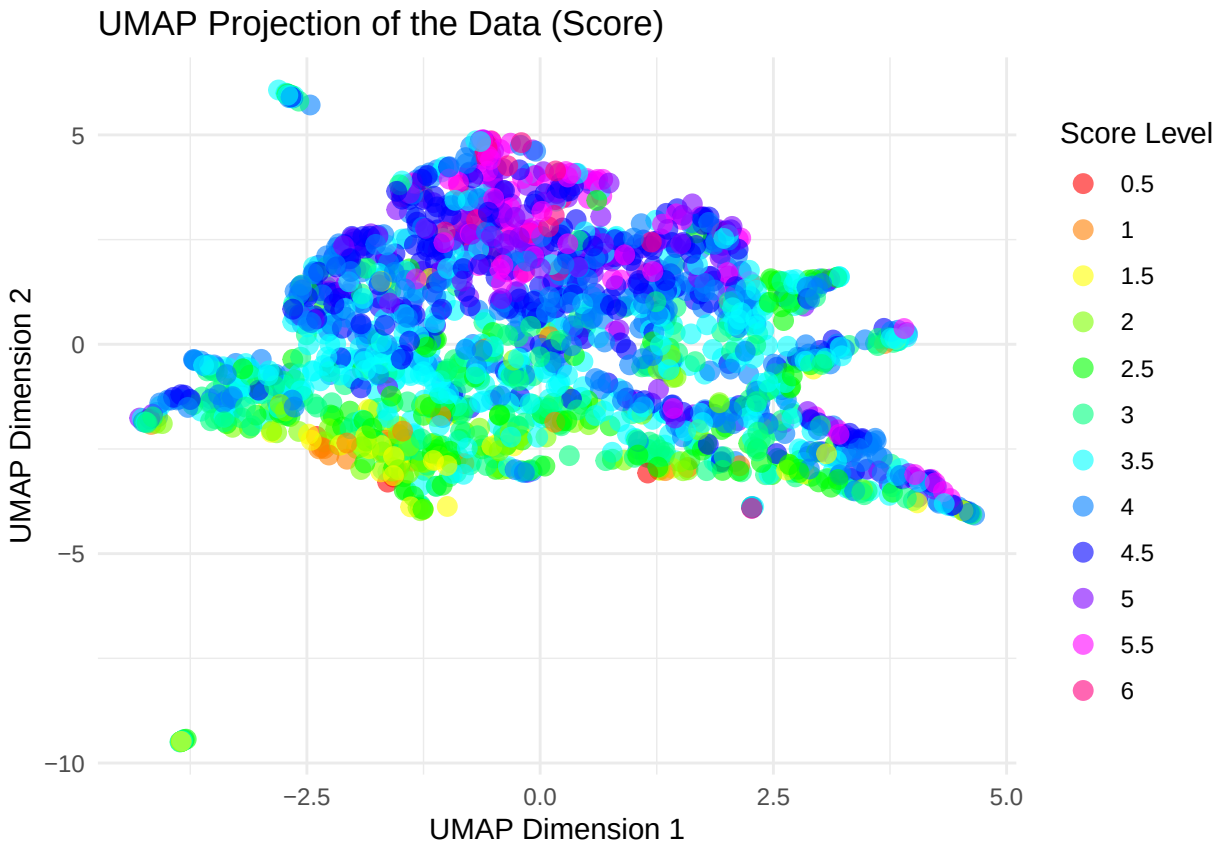
Additionally, with projecting the data using UMAP, we see that the clusters might not be very well separated and aren't globular, so it is reasonable to conclude that KMeans algorithm may struggle with this data in general and gives us key insight on the underlying structure of the data not being nicely seperable. This suggests further specialized data engineering or other advanced techniques needs to be performed in the future (which requires more time) if we want to explore/improve the cluster-seperability within our data. :(

```
umap_result <- umap(train, n_components = 2)

umap_data <- as.data.frame(umap_result$layout)
colnames(umap_data) <- c("UMAP1", "UMAP2")

clusters <- as.factor(result$cluster)
umap_data$Cluster <- clusters

# Choose a palette
palette <- brewer.pal(n = 5, name = "Set1")  # Adjust 'name' as needed

# Plotting the UMAP projection with enhancements
ggplot(umap_data, aes(x = UMAP1, y = UMAP2, color = Cluster)) +
  geom_point(alpha = 0.6, size = 3) +  # Adjust transparency and point size
  scale_color_manual(values = palette) +
  theme_minimal() +
  theme(legend.position="bottom") +  # Move legend to the bottom
  ggtitle("UMAP Projection of the Data (Clusters)") +
  xlab("UMAP Dimension 1") +
  ylab("UMAP Dimension 2")
```



As we can see our cluster UMAP shows that while we do cluster pretty well, the seperation between the cluster is not very good. The data is not well clustered. For further analysis, here we see the distribution of scores in another UMAP

```
scores <- as.ordered(as.data.frame(train)$score)
umap_data$Score <- scores

# Choose a palette for score levels
score_palette <- rainbow(12)

# Plotting the UMAP projection with score coloring and enhancements
ggplot(umap_data, aes(x = UMAP1, y = UMAP2, color = Score)) +
  geom_point(alpha = 0.6, size = 3) +  # Adjust transparency and point size
  scale_color_manual(values = score_palette, name = "Score Level") +
  theme_minimal() +
  ggtitle("UMAP Projection of the Data (Score)") +
  xlab("UMAP Dimension 1") +
  ylab("UMAP Dimension 2")
```



Investigation of the score distribution confirms and suggests that the underlying clustering structure of the data does not closely align with the structure of scores.

The spread of data between K-Means clusters and scores. characterizes as follows:

- cluster 1 tends to capture those who score 3 to 6. (very spread apart)
- cluster 2 tends to capture those who score between 3.5 to 4.5 (tight knit)
- cluster 3 tends to capture those who score 1.5 to 4.5 (very spread apart)
- cluster 4 tends to capture those who score 3.5 to 4 (tight knit)
- cluster 5 tends to capture those who score 2.5 to 4.5 (spread apart)

e.g., a user who scores around 5 is likely to be in cluster 1

[**JONATHAN CAN U CONFIRM ABOVE ANALYSIS, IN CASE IT CHANGED DUE TO RE-RUN]

While there are some relationships, it would appear that the dispersions of scores between groups tend to overlap heavily and are not well separate to segment the groups in a very meaningful way. However clusters 1 and 2 vs. clusters 3, 4, and 5 seem to show some disparity, but this isn't super useful as we care about clusters that are well-seperated amongst one another also.

```
# Create a contingency table without normalization
contingency_table <- table(result$cluster, df$score[-test_ids])

# View the contingency table
print(contingency_table)
```

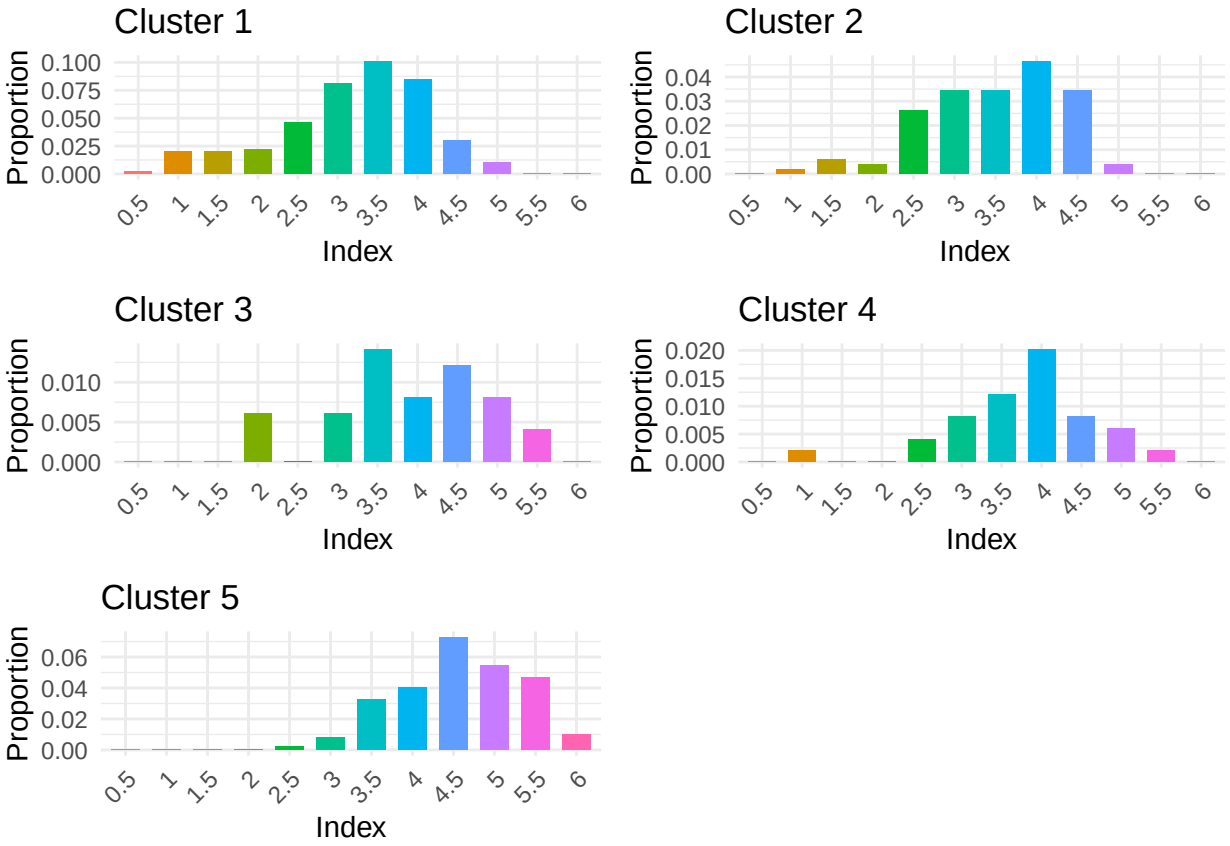
```
##
##     0.5   1 1.5   2 2.5   3 3.5   4 4.5   5 5.5   6
##   1   4  19  45  58  97 173 216 171  78  12   2   1
##   2   0   3   6  11  32  58  82  70  58  16   9   1
```

```
##   3   0   0   0   0  12  13  28  38  14   9   3   0
##   4   0   1   4   6  18  16  22  34  20   8   7   0
##   5   0   0   1   1   3   8  42  89 154  93  81  30
```

Testing on new Data

```r
assign_cluster <- function(new_data, centers, distance_method = "euclidean") {
  # Calculate distances from each new data point to each cluster center
  distances <- as.matrix(dist(rbind(centers, new_data), method = distance_method))
  distances <- distances[(nrow(centers) + 1):nrow(distances), 1:nrow(centers)]

  # Assign each new data point to the nearest cluster
  max.col(-distances)
}

new_clusters <- assign_cluster(test, result$centers)
```

```r
# Assuming 'new_clusters' and 'df' are available
acc_data <- as.matrix(
  t(table(new_clusters, df[test_ids, ]$score)) / nrow(df[test_ids, ])
)

# Create a list to store individual plots
plot_list <- list()

for (i in 1:5) {
  prop_plot <- ggplot(
    data.frame(
      Index = rownames(acc_data), Proportion = acc_data[, i]
    ),
    aes(x = Index, y = Proportion, fill = factor(Index))
  ) +
    geom_bar(stat = "identity", position = "dodge", width = 0.7) +
    labs(title = paste("Cluster", i)) +
    theme_minimal() +
    theme(
      axis.text.x = element_text(angle = 45, hjust = 1),
      legend.position = "none"
    )

  plot_list[[i]] <- prop_plot
}

# Combine plots into a single grid
grid_plot <- do.call(grid.arrange, c(plot_list, ncol = 2))  # Arrange in a grid
```




```
# Print the combined grid plot
grid_plot
```

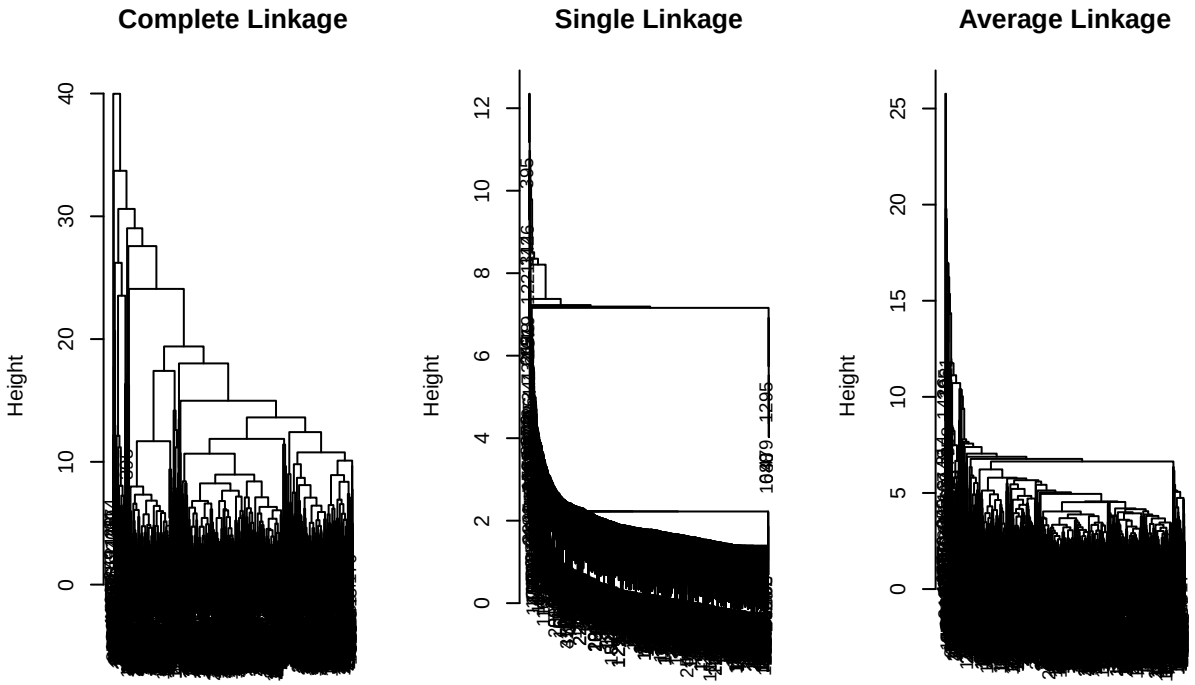
```
## TableGrob (3 x 2) "arrange": 5 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
## 3 3 (2-2,1-1) arrange gtable[layout]
## 4 4 (2-2,2-2) arrange gtable[layout]
## 5 5 (3-3,1-1) arrange gtable[layout]
```

Hierarchial Clustering

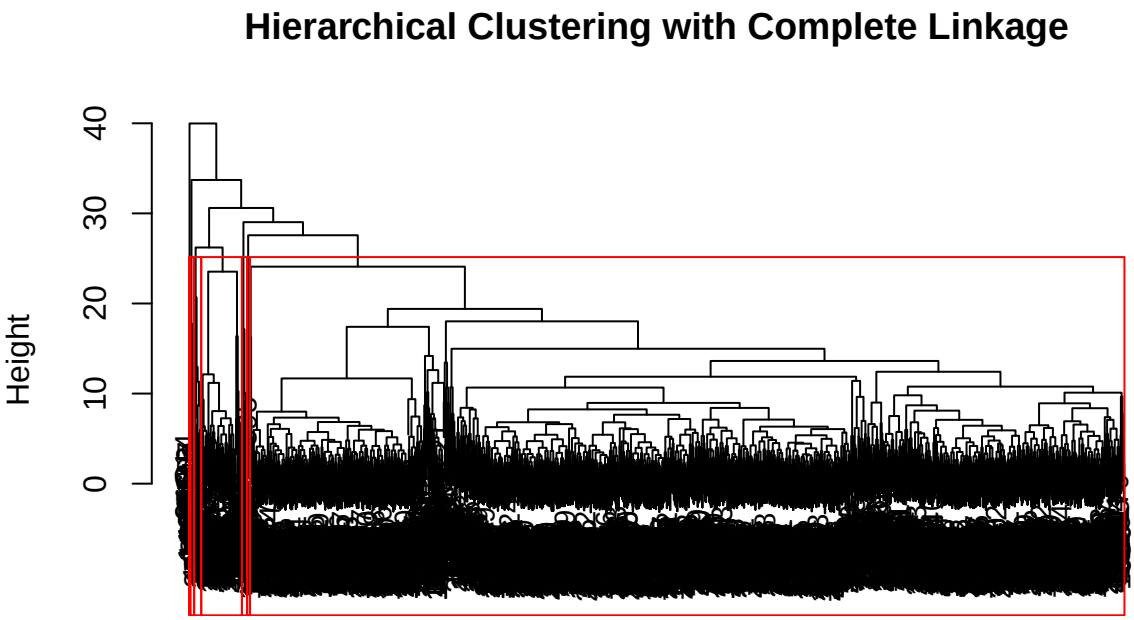
We selected to do Ward's linkage due to cohesion and separability issues in the data. we select 7 clusters because there are 7 integer scores (after rounding half scores). The clustering structure in the data seems to support this number of clusters.

```
distance_matrix <- dist(train, method = "euclidean")

# Complete Linkage
hc_complete <- hclust(distance_matrix, method = "complete")

# Single Linkage
hc_single <- hclust(distance_matrix, method = "single")

# Average Linkage
hc_average <- hclust(distance_matrix, method = "average")
```

```r
# Plotting dendrograms
par(mfrow = c(1, 3))
plot(hc_complete, main = "Complete Linkage", xlab = "", sub = "", cex = 0.9)
plot(hc_single, main = "Single Linkage", xlab = "", sub = "", cex = 0.9)
plot(hc_average, main = "Average Linkage", xlab = "", sub = "", cex = 0.9)
```

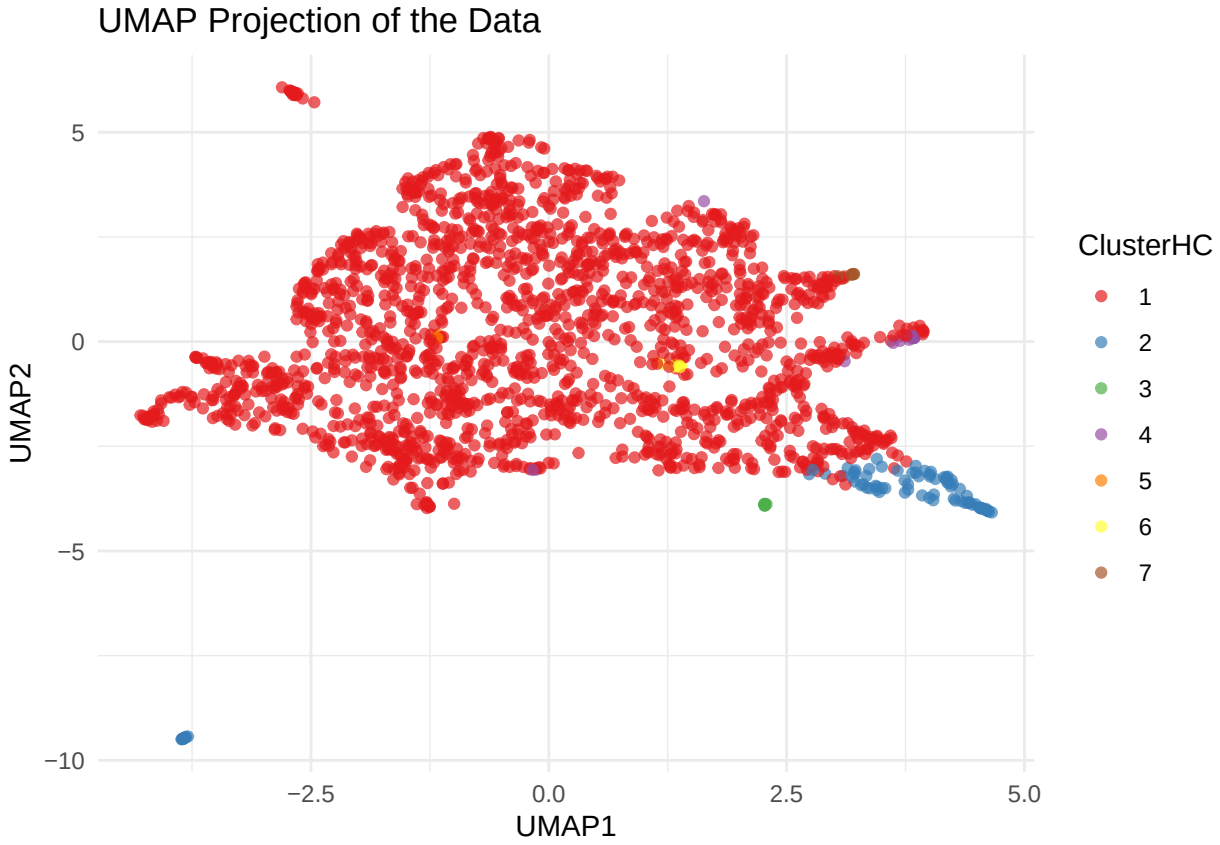



```r
k <- 7  # Number of clusters
plot(hc_complete, main = "Hierarchical Clustering with Complete Linkage",
     xlab = "", sub = "", cex = 0.9)
rect.hclust(hc_complete, k = k, border = "red")  # You can change the border color if you like
```



clustering structure in UMAP

```r
clusters <- as.factor(cutree(hc_complete, k = k))
umap_data$ClusterHC <- clusters

# Choose a palette
palette <- brewer.pal(n = k, name = "Set1")  # Adjust 'name' as needed

ggplot(umap_data, aes(x = UMAP1, y = UMAP2, color = ClusterHC)) +
  geom_point(alpha = 0.7) +
  scale_color_manual(values = palette) +
  theme_minimal() +
  ggtitle("UMAP Projection of the Data")
```



investigation of the score distribution suggests that the underlying clustering structure of the data ...

spread of data between kmeans clusters and scores. characterizations are as follows:

- cluster 1 tends to capture those who score 2.5 to 4.5
- cluster 2 tends to capture those who score 3.5 to 4.5
- cluster 3 tends to capture those who score 1.5 to 4.5
- cluster 4 tends to capture those who score 2.5 to 4
- cluster 5 tends to capture those who score 3.5 to 6
- cluster 6 tends to capture those who score 2.5 to 4.5
- cluster 7 tends to capture those who score 3 to 4.5

e.g., a user who scores around 5 is likely to be in cluster 5

```r
t(table(umap_data$ClusterHC, df[-test_ids, ]$score)) / nrow(df[-test_ids, ])
```

```
##
```

```
##                 1            2            3            4            5
##   0.5 0.0020232676 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   1   0.0106221548 0.0000000000 0.0000000000 0.0005058169 0.0000000000
##   1.5 0.0257966616 0.0020232676 0.0000000000 0.0005058169 0.0000000000
##   2   0.0364188164 0.0020232676 0.0000000000 0.0000000000 0.0000000000
##   2.5 0.0758725341 0.0045523520 0.0000000000 0.0000000000 0.0000000000
##   3   0.1249367729 0.0070814365 0.0010116338 0.0020232676 0.0000000000
##   3.5 0.1820940819 0.0075872534 0.0010116338 0.0025290845 0.0000000000
##   4   0.1917046029 0.0085988872 0.0010116338 0.0010116338 0.0005058169
##   4.5 0.1557916034 0.0065756196 0.0000000000 0.0000000000 0.0000000000
##   5   0.0657561963 0.0025290845 0.0005058169 0.0010116338 0.0000000000
##   5.5 0.0480526050 0.0025290845 0.0000000000 0.0000000000 0.0010116338
##   6   0.0156803237 0.0000000000 0.0005058169 0.0000000000 0.0000000000
##
##                 6            7
##   0.5 0.0000000000 0.0000000000
##   1   0.0005058169 0.0000000000
##   1.5 0.0000000000 0.0000000000
##   2   0.0000000000 0.0000000000
##   2.5 0.0005058169 0.0010116338
##   3   0.0005058169 0.0000000000
##   3.5 0.0020232676 0.0020232676
##   4   0.0005058169 0.0000000000
##   4.5 0.0015174507 0.0000000000
##   5   0.0000000000 0.0000000000
##   5.5 0.0000000000 0.0000000000
##   6   0.0000000000 0.0000000000
```

To predict on new data in test...

```r
# Function to calculate centroids of clusters
calculate_centroids <- function(data, clusters) {
  aggregate(data, by=list(cluster=clusters), FUN=mean)
}

# Function to predict the cluster of new data
predict_cluster <- function(new_data, train_data, clusters) {
  centroids <- calculate_centroids(train_data, clusters)
  # Remove the cluster column
  centroids <- centroids[, -1]

  # Function to find nearest centroid
  find_nearest_centroid <- function(point, centroids) {
    dists <- apply(centroids, 1, function(centroid) dist(rbind(centroid, point)))
    which.min(dists)
  }

  apply(new_data, 1, find_nearest_centroid, centroids = centroids)
}

new_clusters <- predict_cluster(test, train, clusters)
```

```r
acc_data = as.matrix(
  t(table(new_clusters, as.integer(df[test_ids, ]$score))))
```

```
acc_data
```

```
##    new_clusters
##       1   2   3   4   6   7
##   0   1   0   0   0   0   0
##   1  23   1   0   1   0   0
##   2  48   2   0   3   1   1
##   3 148   8   0   5   3   0
##   4 157  12   1   6   0   1
##   5  62   3   1   0   0   1
##   6   5   0   0   0   0   0
```

Supervised Learning Algorithms

Proportional Odds Model (GLM)

Since the score is an ordinal variable, we can use a proportional odds model—a type of linear model—to predict the score. We'll use the `VGAM` package to fit the model and use the `step4` function to do a stepwise selection to find the best model. `VGAM` is a package that allows for fitting of a multinomial/propodds (vector based) linear model. It assumes cumulative probabilities and keeps the same β for all categories, but allows for different intercepts.

We'll use only non-collinear features (`score, word_count_max, down_event_special_character, mae_cursor_position, down_time_std, down_event_control_keys, text_change_not_q, activity_input`) and do a subset selection. We'll also convert `score` to an ordered factor to ensure that the model knows that it is an ordinal variable.

```
prop.wc <- vglm(score ~ word_count_max, data = train.supervised, family = propodds(reverse = F))

prop.upper <- vglm(score ~ ., data = train.supervised, family = propodds(reverse = F))

summary(prop.upper)
```

We can do a stepwise selection, starting from all variables, to find the best model. This will pick the model with the lowest AIC, which aims for better prediction error.

```
prop.step <- step4(prop.upper, scope = list(lower = prop.wc, upper = prop.upper), direction = "both")

summary(prop.step)
```

Looking at our selected model, we see that it drops `text_change_q`. Furthermore, we notice that the largest magnitude coefficient is `-1.61549` for `word_count_max`. For the propodds model we trained, if a coefficient is negative, it means the probability of falling into a lower category decreases as the predictor increases. This makes sense, since we saw that the word count was positively correlated with the score. `down_event_control_keys` and `activity_input` are negatively correlated with the score, and we see positive coefficients for them, which checks out.

Using our selected model, we can predict the probabilities of falling into each category and selecting the category with the highest probability as the predicted score. We can then compare the predicted scores to the actual scores to see how well our model did.

```
# Confusion Matrix
prop.t1 <- (table(train.prop.pred_score, train.supervised$score))
# Accuracy
cat("Training Accuracy: ", mean(conv(train.prop.pred_score) == conv(train.supervised$score)))
```

```
## Training Accuracy:  0.3227112
```

```r
# MAE
cat("Training MAE: ", mean(abs(conv(train.prop.pred_score) - conv(train.supervised$score))))
```

```
## Training MAE:  0.5566515
```

```
## Testing Accuracy:  0.2874494
```

```
## Testing MAE:  0.5921053
```

```
## Warning in styling_latex_position_right(x, table_info, hold_position,
## table.envir): Position = right is only supported for longtable in LaTeX.
## Setting back to center...
```

Table 1: Propodds Training Confusion Matrix

	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5	6
0.5	0	0	1	0	0	2	0	0	0	0	0	0
1	0	0	0	1	2	1	0	0	0	0	0	0
1.5	0	0	0	2	2	0	2	0	0	0	0	0
2.5	1	3	6	2	8	2	1	0	0	0	0	0
3	3	10	27	52	85	91	39	9	1	0	0	0
3.5	0	8	18	16	54	129	221	150	62	6	3	1
4	0	2	2	2	7	32	83	157	114	42	15	4
4.5	0	0	2	0	3	9	34	71	124	72	47	12
5	0	0	0	0	0	0	1	1	2	4	1	0
5.5	0	0	0	1	1	2	7	12	19	11	29	11
6	0	0	0	0	0	0	2	2	2	3	7	4

Table 2: Propodds Testing Confusion Matrix

	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5	6
1	0	0	0	0	0	1	0	0	0	0	0	0
2.5	0	2	1	0	2	1	0	0	0	0	0	0
3	1	7	9	11	20	15	11	5	0	0	0	0
3.5	0	3	3	4	11	44	49	35	10	3	0	0
4	0	0	0	1	5	3	23	37	31	10	5	1
4.5	0	0	0	0	0	1	10	17	31	19	10	3
5	0	0	0	0	0	0	1	1	0	1	2	0
5.5	0	0	0	0	1	3	2	4	4	7	7	1
6	0	0	0	0	0	0	0	0	2	1	2	0

Our model does okay, especially given that there are 12 categories. It predicts the correct score only around 32% of the time on the training and 28.7% on the testing data. From the confusion matrix, we see that the model is not very good at predicting the lower and upper extremes. It predicts a lot of the mid level scores (3 - 4.5), but is not good at differentiating between them.

This is why we can also look at the MAE, which will give us a better idea of how far off the score predictions are on average, since it is a numerical measure.

On average, the model is off by around 0.55 points on the training and 0.59 points on the testing data. This is a pretty good result, meaning that, on average, the score is only a bit more than one level off.

K Nearest Neighbors

During our data analysis/unsupervised learning, we saw that the clustering algorithm(s) did not do a great job of separating the data. This is likely because the data has a lot of overlap. To see if this holds during actual prediction, we can try to use a KNN model to predict the score.

We'll use the `caret` package to tune our `k` value through 10-fold cross validation. A larger `k` will introduce more bias in the model, and a smaller `k` will have a larger variance. Since there are lots of records, a large `k` value could be useful, without introducing too much bias—so we'll test a range of `k` values from 1 to 400.

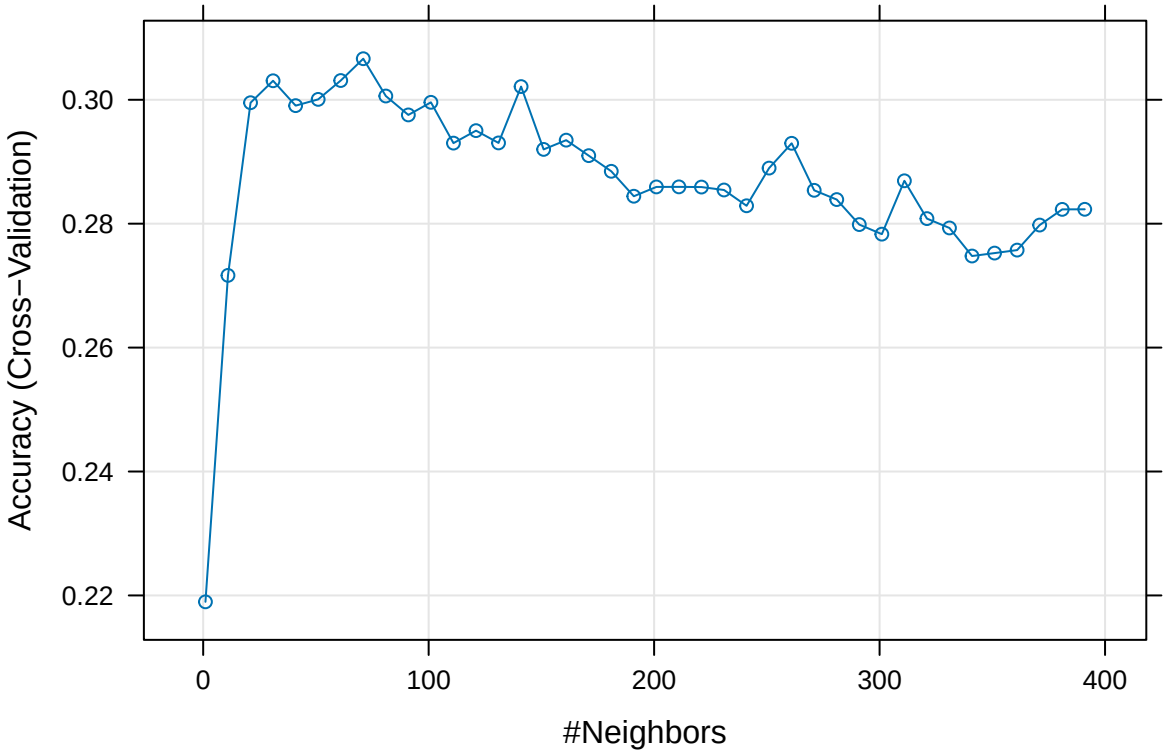
From the model output, we saw that the best `k` value is 71, with a (cross-validated) accuracy of around 30.66%. We expect to see similar results checking with the entire training dataset. This is similar to our proportional odds model.

```
##
## knn.train.pred 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   0   0   0   0   0   0   0
##            1     0   0   0   0   0   0   0   0   0   0   0   0
##            1.5   0   0   0   0   0   0   0   0   0   0   0   0
##            2     0   0   0   0   0   0   0   0   0   0   0   0
##            2.5   1   1  12  15  25  22   9   3   0   0   0   0
##            3     2   6  10  18  34  64  28  15   3   2   0   0
##            3.5   1  12  27  36  79 120 208  94  38   6   1   1
##            4     0   4   5   6  19  47  93 194 113  41  24   0
##            4.5   0   0   2   1   5  14  47  89 164  81  56  24
##            5     0   0   0   0   0   0   0   1   1   0   3   0
##            5.5   0   0   0   0   0   1   5   6   5   8  18   7
##            6     0   0   0   0   0   0   0   0   0   0   0   0
```

```
## Accuracy:  0.3404148
```

```
## MAE:  0.5680324
```

Checking the full training data, we see that the accuracy is 34%! This is high, but we should be careful. The model could be overfitting to the training data.



From the plot, we see the accuracy is low for small `k` values and quickly rises. After that, the accuracy seems to slowly decrease, demonstrating an increased bias (bias-variance tradeoff).

Let's also see how the model does on the testing data.

```
##
## knn.test.pred 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  0   0  0   0  0   0  0
##           1     0  0   0  0   0  0   0  0   0  0   0  0
##           1.5   0  0   0  0   0  0   0  0   0  0   0  0
##           2     0  0   0  0   0  0   0  0   0  0   0  0
##           2.5   1  0   2  1   3  8   2  0   0  0   0  0
##           3     0  3   5  3   9  7   9 11   1  0   0  0
##           3.5   0  7   6 10  17 36  40 23   9  3   0  0
##           4     0  2   0  2   6 13  28 40  33  8   4  1
##           4.5   0  0   0  0   4  4  16 23  31 28  16  3
##           5     0  0   0  0   0  0   0  0   0  1   0  1
##           5.5   0  0   0  0   0  0   1  2   4  1   6  0
##           6     0  0   0  0   0  0   0  0   0  0   0  0
```

```
## Accuracy:  0.2591093
```

```
## MAE:  0.6437247
```

The model does a noticeably bit worse on the testing data, with an accuracy of around 24.5% (10 percentage points). This is likely an indicator that the model is overfitting to the training data and clusters/predictor spread is different between the training and testing data. This MAE is higher than the proportional odds model (0.57 for training, and 0.66 for testing). The KNN model only looks at euclidean distances. Our unsupervised learning showed that the clusters are overlapping, so it may have a hard time differentiating between scores and can be less accurate than the proportional odds model.

xgboost

We saw from the literature review that gradient boosting was effective. So, we'll use xgboost with multi:softmax to predict the score.

We'll tune the `eta` (learning rate), as this is a critical hyperparameter of gradient boosting. For small values of `eta`, the model will take longer to converge, but will be more accurate. For large values of `eta`, the model will converge quickly, but will be less accurate.

We'll use the default `max_depth` (6) and an `nrounds` of 15, as we saw overfitting with larger `nrounds` (50), and will reduce the computational time. We'll pick the model with the lowest training error.

```
## Eta Vals: 0.01, 0.02, 0.03, 0.04, 0.05
```

```
## Accuracy per Eta: 0.5017704 0.5300961 0.5619626 0.5776429 0.6024279
```

```
## Best Eta:  0.05
```

We saw that the best `eta` value is `0.05`, with a training accuracy of `0.6024279`. This accuracy is much larger than any other training accuracy we've seen. xgboost can very quickly converge to 100% accuracy on the training data, especially with large `nrounds` and `eta` values. This may be an indicator that the model is overfitting to the training data. We saw this when we used larger `eta` values, so we chose smaller values and we can see how it does on the testing data.

```
##
## train.labeled_predictions 0.5   1 1.5   2 2.5   3 3.5   4 4.5   5 5.5   6
##                       0.5   1   0   0   1   0   0   0   0   0   0   0   0
##                       1     0   7   0   0   0   1   0   0   0   0   0   0
##                       1.5   0   0  20   1   0   1   1   0   0   0   0   0
##                       2     0   2   2  42   0   3   0   0   0   0   0   0
##                       2.5   2   5  14  15  96  23  10   3   3   0   0   0
##                       3     1   1   6   7  16 150  13  12   1   0   0   0
##                       3.5   0   5  12   7  37  49 261  72  30   2   1   1
##                       4     0   3   1   3   7  27  63 250  44  25  10   3
##                       4.5   0   0   1   0   6  14  38  60 238  60  25  10
##                       5     0   0   0   0   0   0   1   1   2  45   0   0
##                       5.5   0   0   0   0   0   0   3   4   5   6  65   2
##                       6     0   0   0   0   0   0   0   0   1   0   1  16
```

```
## Train Accuracy:  0.6024279
```

```
## Train MAE:  0.3358624
```

```
##
## test.labeled_predictions 0.5  1 1.5  2 2.5  3 3.5  4 4.5  5 5.5  6
##                       1     0  2   1  0   1  1   0  0   0  0   0  0
##                       1.5   0  1   2  0   0  1   0  0   0  0   0  0
##                       2     1  0   2  2   1  1   0  0   0  0   0  0
##                       2.5   0  2   1  4  17  5   8  2   1  0   0  0
##                       3     0  6   3  4   5 15  11  6   3  0   0  0
##                       3.5   0  1   4  5   9 27  36 24   9  4   2  0
##                       4     0  0   0  0   3 13  21 36  30  7   3  0
##                       4.5   0  0   0  1   2  4  17 26  28 23  10  5
##                       5     0  0   0  0   0  0   0  1   4  2   5  0
##                       5.5   0  0   0  0   0  0   3  3   2  3   5  0
##                       6     0  0   0  0   1  1   0  1   1  2   1  0
```

```
## Test Accuracy:  0.2935223
```

```
## Test MAE:  0.6072874
```

Here, we get a testing accuracy of around 30%, which is still lower than the training data (as expected). This is the best test accuracy we've seen so far, as well as an MAE on the lower end (0.61). This makes sense, as gradient boosting is a powerful technique that can capture complex relationships between variables — not just linear relationships or clusters based on euclidean distance. However, all three models have similar accuracy and MAEs for the testing data, so it's hard to definitively say which is the best model—especially when there's randomness involved in the training process.