STAT 432 Final Project 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 is an assigned project for STAT 432 Fall 2023 (UIUC), our goal was 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 is Linking Writing Processes to Writing Quality where we explore typing behavior to predict essay quality between a score of 0-6 (inclusive).

Approach

We divided our work into three main section.

- **Data Processing** → Cleaned, extracted, and engineered features in python notebook. Important for our Supervised and Unsupervised learning portion.
- **Unsupervised Learning** → Performed clustering algorithms on the cleansed data (and 80-20 split). We chose to do K-Means and Hierarchical Clustering algorithms (paired with UMAP) as we wanted to explore what we learned in class. These unsupervised techniques allowed us to find hidden patterns in our data and act as an outlet for advanced EDA.
- **Regression/Classification Models** → Predicted/classifued scores and attempted to achieve our original goal.To be specific, we performed `KNN` and `XGBoost` from what we learned in our class, but also performed `Proportional Odds Model (GLM)` which is something we learned in STAT 426.

Unique Approaches/Techniques:

We generally used as much knowledge from STAT 432 as we could, but there were moments where we did consult other topics such as pairing elbow method with Silhouette Plots (to determine how well the cluster fit), and GLM model from STAT 426. We have cited sources we consulted including other Kaggle notebooks. :D

Conclusion:

We concluded this project with successful implementation of all the models we wanted to on the data. Unfortunately, the data is very complicated to work with for our basic modelling techniques to work super accurately! The reason is the data was not nicely separable and globular in nature which made it hard for the models to classify the user essays into accurate scores. Our predictions accuracy were better than we expected, but still not super high. On average, we were about `+/- 0.6` from the actual prediction score which isn't bad given our algorithms and features. There was a lot of overlap/confusion for the model predictions around the range of 3.5-4.5 due to lot of essays being in that region causing overlap between clusters and predicted scores. Based on that, research and experience, we highly recommend spending more efforts on feature engineering using Unsupervised/Supervised Learning techniques and advanced algorithms such as tokenization, Deep Learning (DL)/Neural Network (NN) architecture, etc. to get more confident and accurate scores for model predictions.

Literature Review

Before we addressed the problem statement, we chose to go on a research journey of reviewing existing submissions to gain insights and inspiration from. This preliminary exploration greatly helped us, specially for feautre engineering, as we 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 ones 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 resulted 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. He was by inspired by others and incorporated a set of features commonly utilized by Kaggle community members. These features were identified through rigorous exploration and collaboration by the community members, therefore adding 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) paired with `LightGBM` were prevalent strategies. Deep learning-based solutions and the integration of neural networks (NN) with LightGBM clearly enhanced prediction accuracy and created adaptive models for the submissions. However, these approaches came with notable challenges. Firstly, they often incur increased computation costs, demanding substantial resources for training and potentially limiting their feasibility in time and resource-constrained projects like ours. The obvious complexity of deep learning models posed interpretability challenges (for us and others) which hinders a clear understanding of their work's inner working.

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 adopted. 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 drew inspiration from all submissions we reviewed and their ideas, 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, such as Feature Engineering and XGBoost 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.

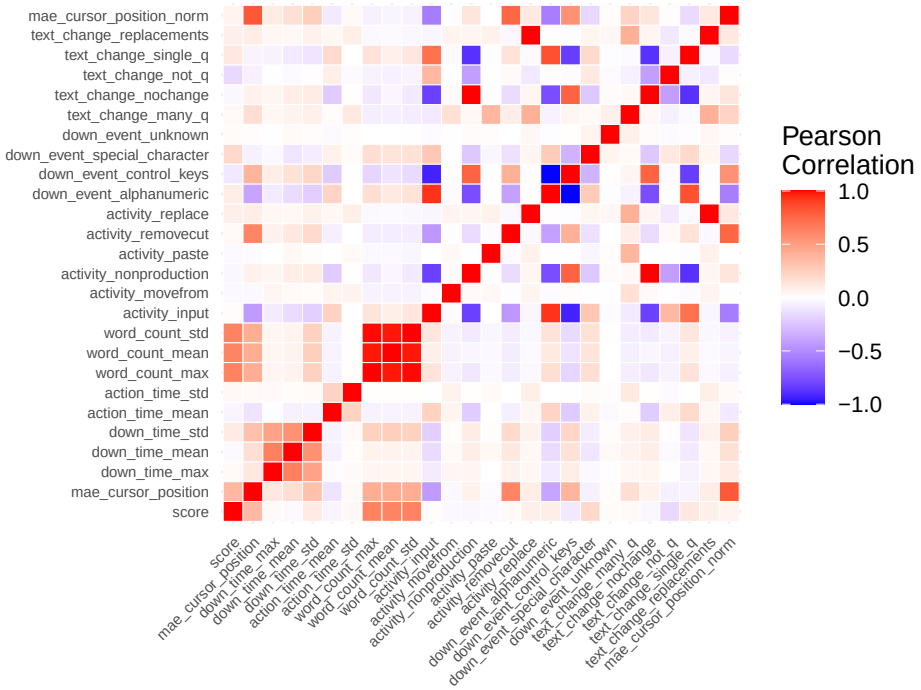
Data Summary

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	0.8113976
activity_movefrom	0.00117325	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
activity_nonproduction	0.04693000	0.10350448	0.04231141	0.06362468	0.02844725	0.03437359
activity_paste	0.0000000000	0.0004074980	0.0000000000	0.0006426735	0.0000000000	0.0000000000
activity_removecut	0.1630817	0.1059495	0.1061412	0.0970437	0.2042671	0.1528720
activity_replace	0.0027375831	0.0004074980	0.0016924565	0.0006426735	0.0000000000	0.0013568521
down_event_alphanumeric	0.6331639	0.6071720	0.7021277	0.6709512	0.6088503	0.6562641
down_event_control_keys	0.3523661	0.3712306	0.2860251	0.3155527	0.3646780	0.3364993
down_event_special_character	0.014470082	0.021597392	0.011847195	0.013496144	0.026471750	0.007236545
down_event_unknown	0	0	0	0	0	0
text_change_many_q	0.0007821666	0.0000000000	0.0007253385	0.0006426735	0.0000000000	0.0004522840
text_change_nochange	0.04693000	0.10350448	0.04231141	0.06362468	0.02844725	0.03437359
text_change_not_q	0.1908487	0.2041565	0.1677950	0.1985861	0.1955749	0.1804613
text_change_single_q	0.7587016	0.6919315	0.7874758	0.7365039	0.7759779	0.7833559
text_change_replacements	0.0027375831	0.0004074980	0.0016924565	0.0006426735	0.0000000000	0.0013568521

Correlation Summary

We observed some pairs of features that showed signs of multicolinearity.



Some features formed parallel or perpendicular colinearity. Examples: `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).

We were more 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. In example, essays written with less frequent or extreme edits is somewhat associated with higher scores. **Please also note:** A positive correlation was 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 and was not considered later on.

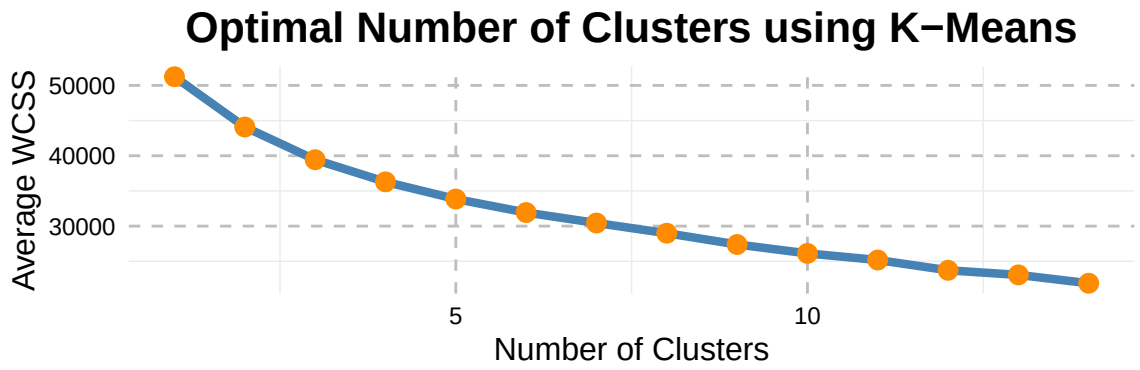
Unsupervised Learning Algorithms

This section dives into the tasks performed for the unsupervised learning algorithms. We mainly focused on K-Means and Hierarchical Clustering as a means to find hidden patterns within our data, in other words, to perform advanced EDA.

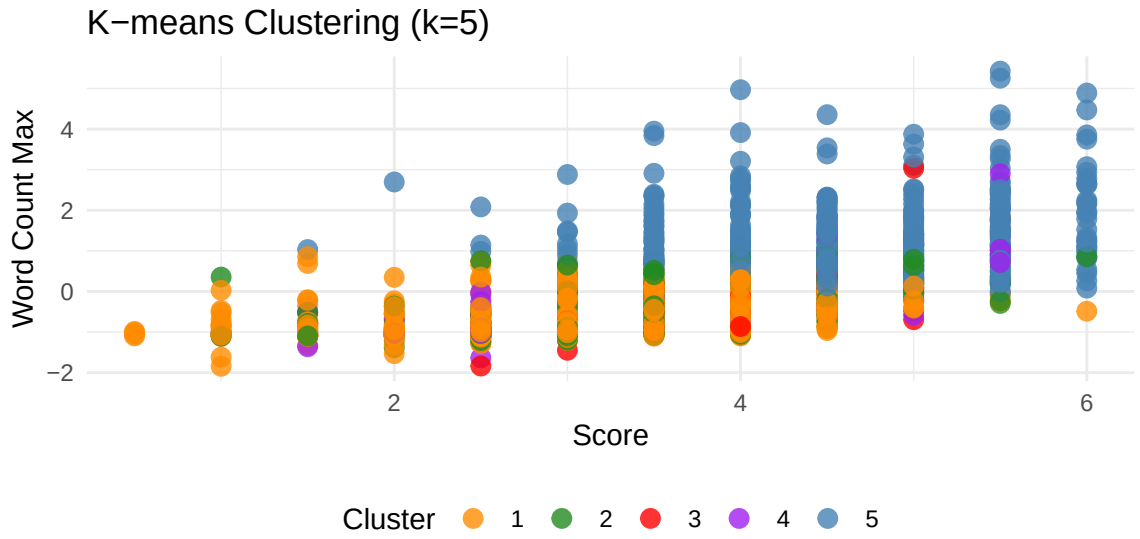
K-Means Algorithm

The section is where we drew insights from K-Means algorithm and it's relation to score feature. The first question we asked ourselves is how many clusters do we want? After experimenting, we chose to use the

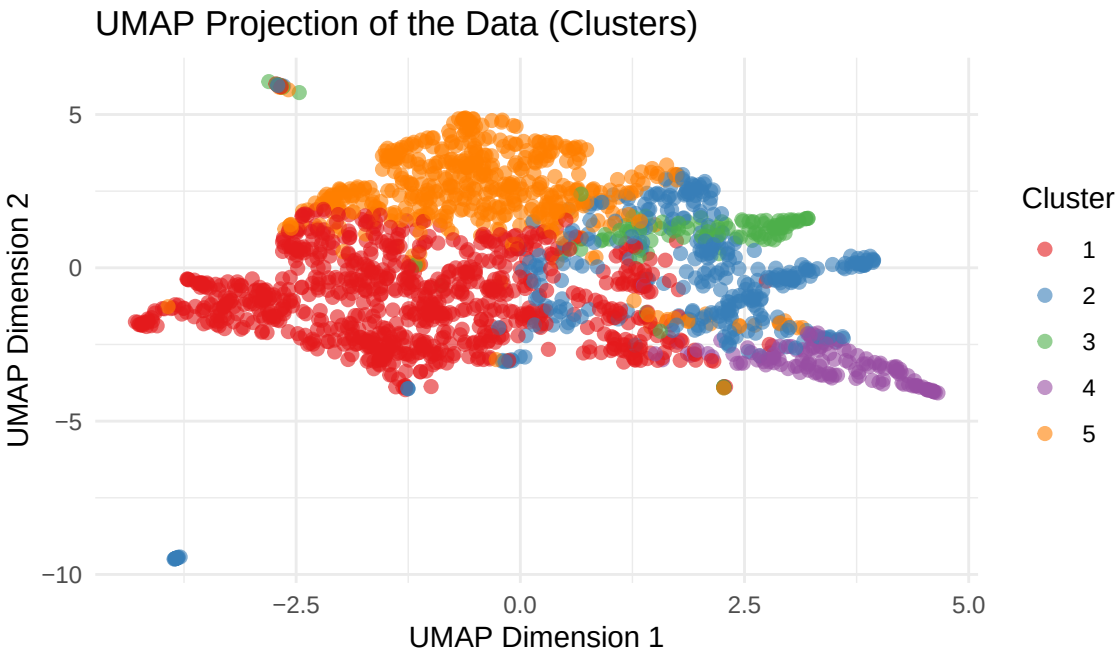
elbow-method to determine our count for clusters.

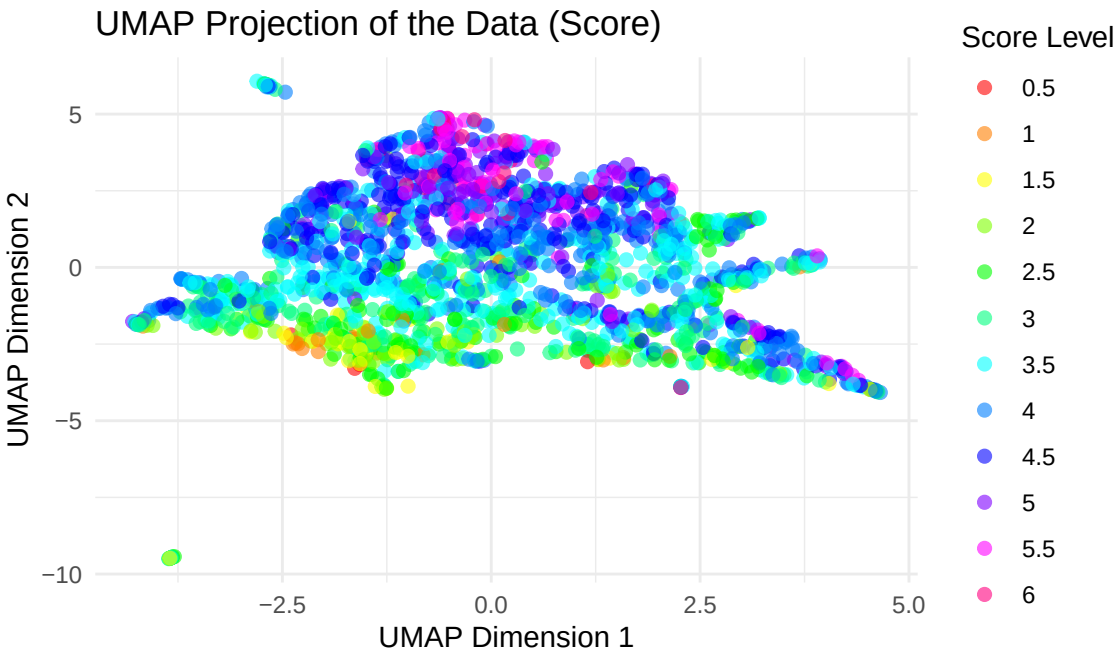


It is slightly hard to see the elbow, but we selected `k=5` clusters to fit our k-means model as it seemed closest to the elbow from the plot above.



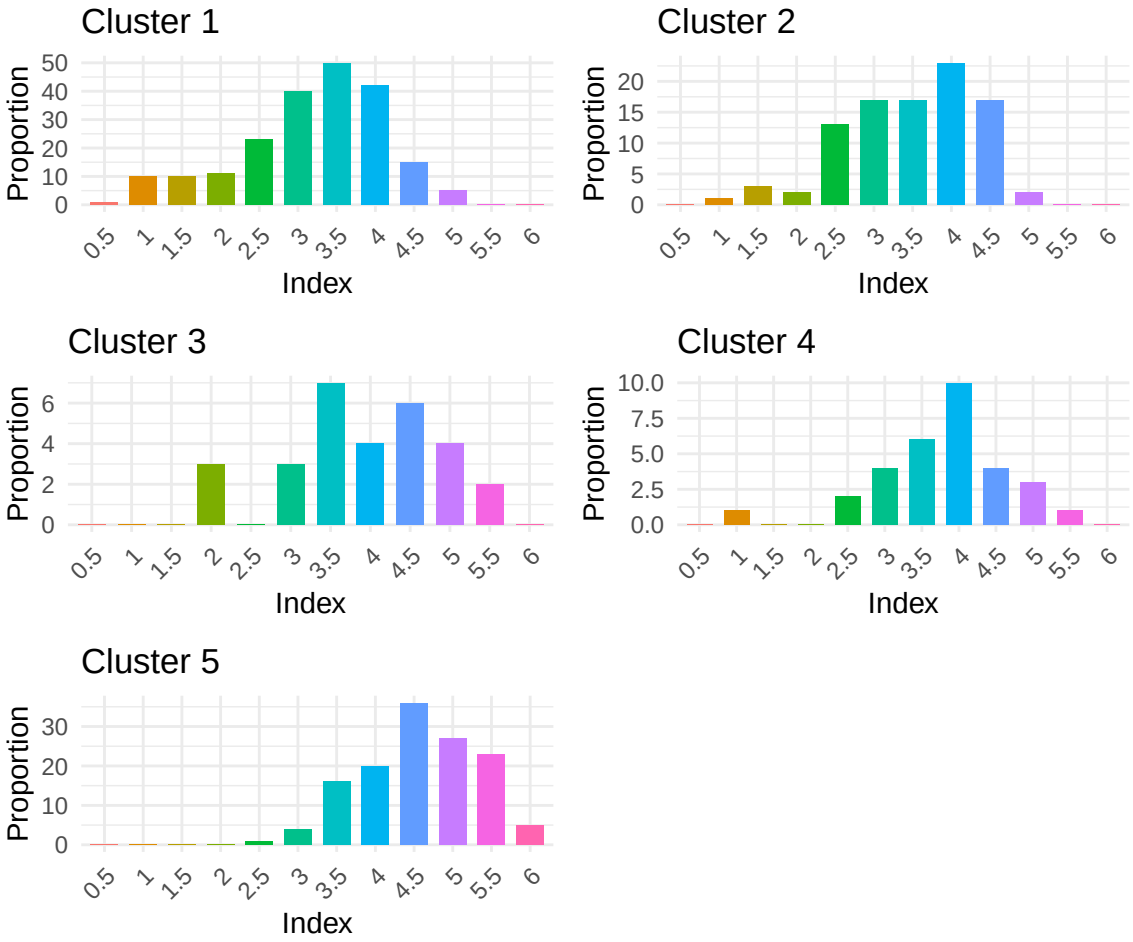
The clusters aren't clearly distinct and have overlapping present. Therefore, we used silhouette scores to evaluate how well the clustering structure fits in terms of similarity within our cluster. In other words, higher silhoutte score for a cluster suggest greater similarity of points within its own cluster and poorer similarity to other clusters which is what we consider to be a good seperable cluster. Referring to the `Silhouette Plot of 5 Clusters` (shown in the appendix), we saw that 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, we can see negative score with cluster 3 which shows indicated issues of cohesion and separation. This kind of confirms our insights we drew from the `K-Means Clustering` plot initially. We dove into more detail by projecting the data using `umap`. This gave us some more insights into the hidden patterns that were present within our data.





We saw that the clusters might not be very well separated and aren't globular from the UMAP, so it was reasonable to conclude that `K-Means` algorithm may struggle with this data in general. This also gave us key insight on the underlying structure of the data not being nicely separable. This suggested further specialized data engineering or other advanced techniques might be helpful, but we moved forward to explore more and progress our project forward.

So let's see how K-Means performs on testing data and see how the cluster assignments look like

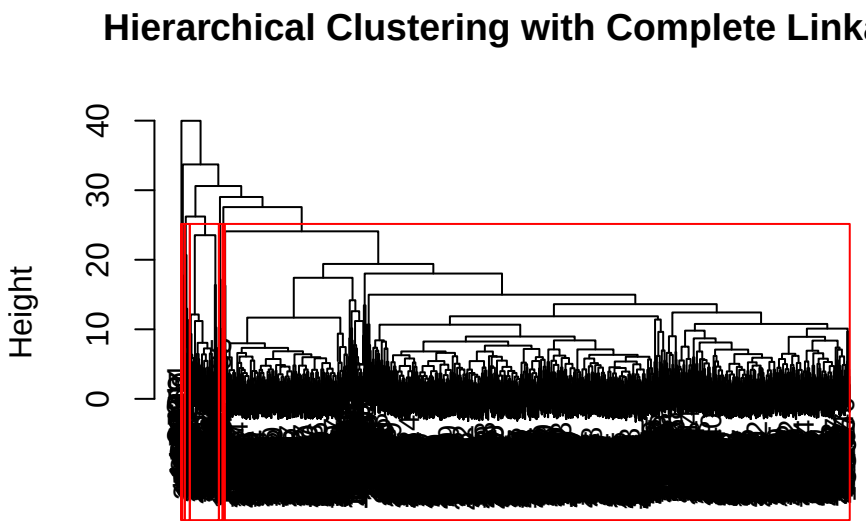


While there are some relationships, it would appear that the dispersions of scores between groups tend to overlap heavily and are not well separate to segment the groups in a very meaningful way. However clusters 1 and 2 vs. clusters 3, 4, and 5 seem to show some disparity, but this isn't super useful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

When considering the underlying relation to scores, K-Means failed to achieve seperability between clusters. In other words, while the clusters within seem strong and cohesive, they tended to be overlapping with other clusters (i.e. scores are overlapping in different clusters) making k-means an overall bad fit for this kind of non-seperable data. As suggested before, specialized data engineering or other advanced techniques might be helpful.

Hierarchial Clustering

For our hierarchial clustering, we performed with euclidean distance using `complete`, `single`, and `average` linkage but chose `complete linkage` as it was the better resulting clustering linkage among the the three. Since our clustering structure in the datanicely supported 7 clusters (for the 7 integer scores after rounding half scores), we selected 7 clusters for the clustering.



We noticed the issue of how the tree was being split (non-uniformly) and we wanted to explore more on what's going on. Therefore, we did another UMAP. If we refer to the `Hierarchial Clutering - UMAP Project of the Data` (shown in the appendix), we will clearly see that most of our predictions are going into cluster 1. This isn't ideal because this means lots of scores will be overlapping into 1 cluster! This confirms our previous insights we drew from K-Means section.

Therefore, in general, we concludde that since the data is not seperable and not globular in structure it is hard for clustering algorithms to perform well. For future, a PCA might be useful to reduce dimensionality to perform clustering, but we made the decision to maintain interpretability of our data and not use PCA in our supervised learning analysis.

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

```
## Training Accuracy:  0.3227112  Training MAE:  0.5566515
```

```
## Testing Accuracy:  0.2874494  Testing MAE:  0.5921053
```

Table 1: 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 testing confusion matrix, we see that the model is not very good at predicting the lower and upper extremes. It predicts a lot of mid level

scores (3 - 4.5), but is not good at differentiating between them. The same results are seen in the training confusion matrix (see Appendix for more).

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

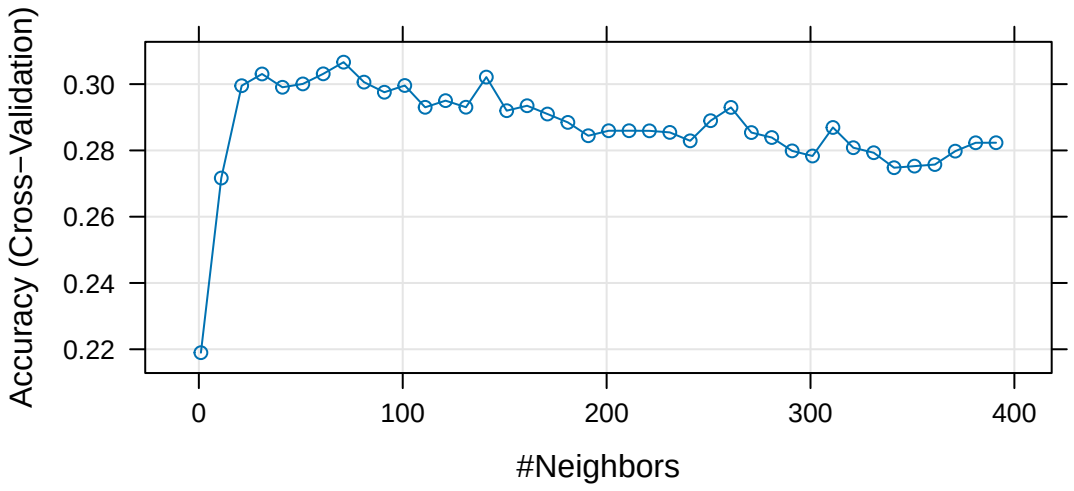
During our data analysis/unsupervised learning, we saw that the clustering algorithm(s) did not do a great job of separating the data. This is likely because the data has a lot of overlap. To see if this holds during actual prediction, we can try to use a KNN model to predict the score.

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.

```
## 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.

```
## Accuracy:  0.2591093  MAE:  0.6437247
```

Table 2: KNN Testing Confusion Matrix

	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

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 Accuracy:  0.6024279  Train MAE:  0.3358624
```

We observe a very high training accuracy, for reasons we explained previously. See appendix for the training confusion matrices.

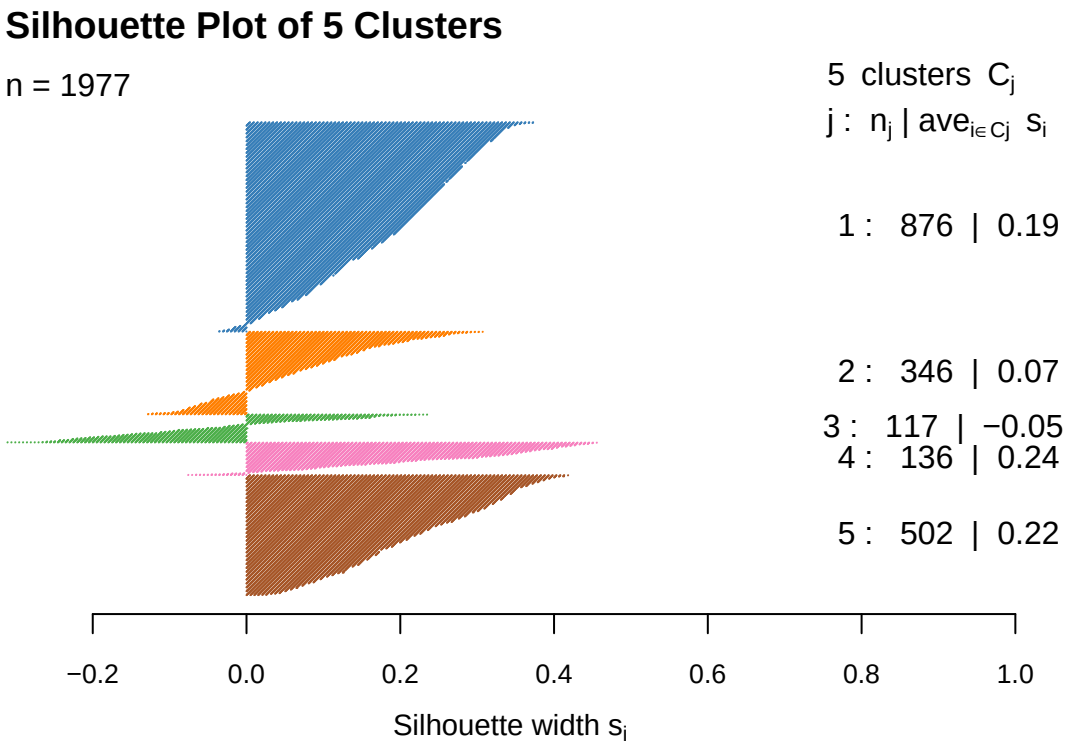
```
## Test Accuracy:  0.2935223 Test MAE:  0.6072874
```

Table 3: XGBoost Testing Confusion Matrix

	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

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.

Appendix



Average silhouette width : 0.17

Table 4: K-Means Testing Contingecy Table

	1	2	3	4	5
0.5	1	0	0	0	0
1	10	1	0	1	0
1.5	10	3	0	0	0
2	11	2	3	0	0
2.5	23	13	0	2	1
3	40	17	3	4	4
3.5	50	17	7	6	16
4	42	23	4	10	20
4.5	15	17	6	4	36
5	5	2	4	3	27
5.5	0	0	2	1	23
6	0	0	0	0	5

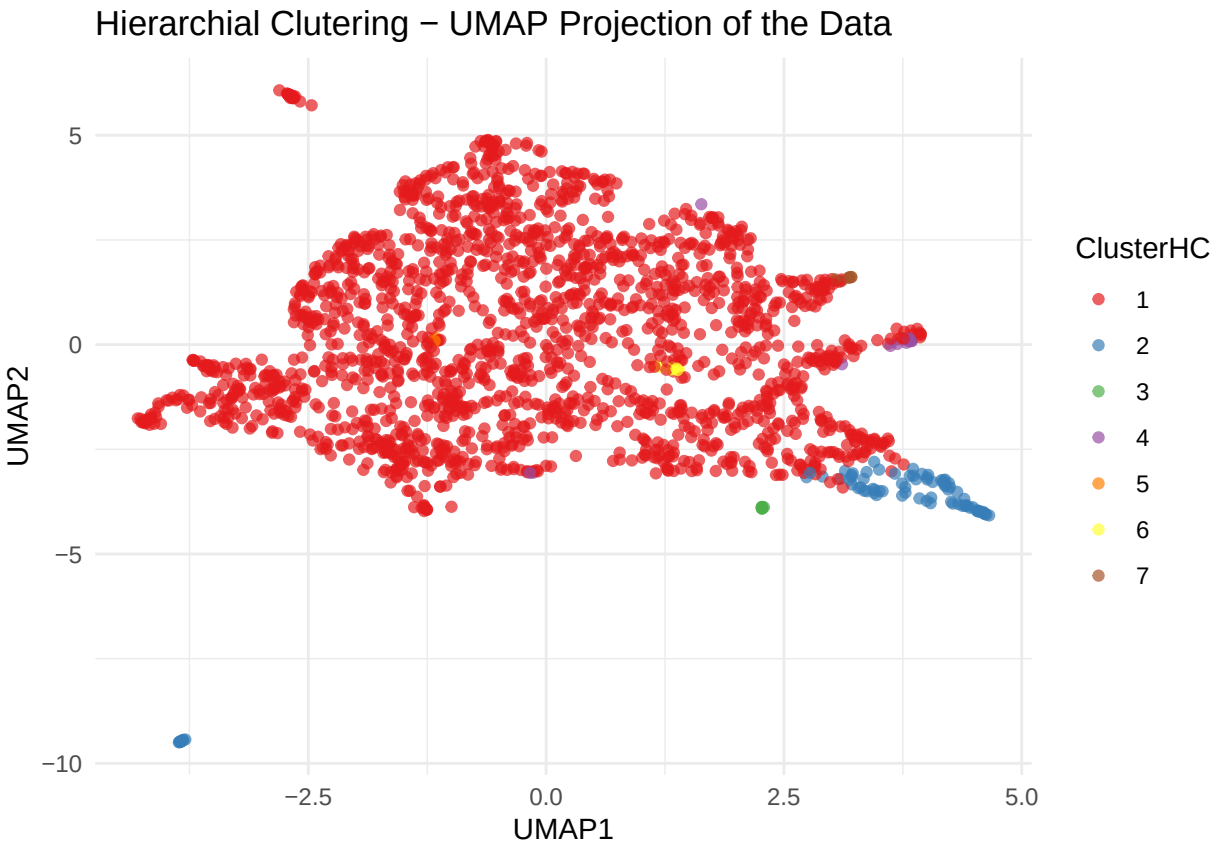


Table 5: Propodds Training Confusion Matrix

	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 6: KNN 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	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

Table 7: XGBoost Training Confusion Matrix

	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