SECU0050 Coursework (30%)

Vlad Pasca

Total Word Count: 1193/1200

Submission Date: 28th April 2020

1) Conduct a sentiment trajectory analysis on (at least) two authors in your dataset. How do the average trajectories differ?

Word Count (excluding Figure captions): 398

Figure 1

Plot of Sentiment against Temporal Progression for One Song from Each Rap Group

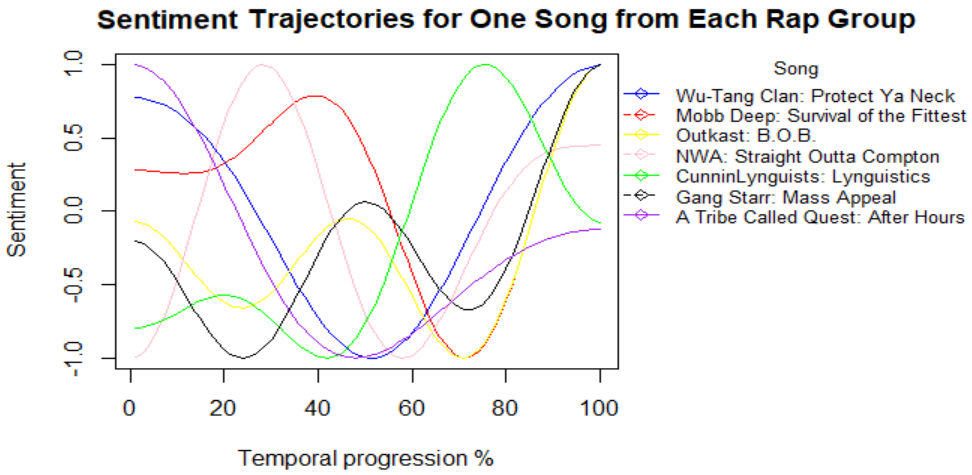


Figure 1 displays the sentiment trajectories across one song each selected from seven rap groups (Wu-Tang Clan, Mobb Deep, Outkast, NWA, CunninLynguists, Gang Starr and A Tribe Called Quest).

Sentiment trajectories were calculated with a naive context sentiment analysis function (Kleinberg, 2020). Sentiment scores were sensitive to valence shifters (negators, amplifiers, adversative conjunctions and deamplifier) and computed as average values in a moving window (default cluster is 2 words around target word). Figures 1 and 2a, extracted sentiment vectors were transformed into 100 bins via discrete cosine transformation (conceptually representing the temporal progression of sentiment over the text).

Figure 2a demonstrated songs showed an overarching temporal sentiment trend regardless of rap group. Songs mostly started with slightly positive connotations or neutral sentiment in the first fifth of the song. The second fifth of songs on average had slightly negative sentiment (except for NWA songs that were slightly positive), which rose to neutral sentiment halfway through songs (except for Outkast songs that remained slightly negative). At temporal progressions around 60-80% through songs, sentiment became slightly negative (except for the slightly positive CunninLynguists average) before finally ending on neutral or slightly positive notes.

Figures 2b-i explored the impact of varying bin and cluster sizes (for the moving window) upon sentiment trajectories. Altering bin sizes generally highlighted the same average sentiment trajectory patterns. But, altering cluster size from 2 to 3 (Figures 2b, 2e, 2h) or 5 (Figures 2c, 2f, 2i) revealed slightly different trends. For example, in the latter figures, NWA shows a

slightly higher sentiment score around 20-25% temporally into songs, compared to other cluster sizes. Furthermore, the slightly positive sentiment values around 75% temporally into CunninLynguists songs observed at bin sizes of 100 and 20 disappeared at bin size of 5. If values of valence shifters were altered, this may also produce different trends.

One limitation of the naïve context sentiment analysis function is that words which do not occur in the sentiment lookup table (the Jockers & Rinker Polarity Lookup Table) were assigned zero. Indeed, one CunninLynguists and two Outkast songs were excluded because sentiment values were unavailable. For other applications, omitted texts could otherwise have drastically changed averaged sentiment. Furthermore, other polarity tables could produce different average sentiment trajectories. Though, since rap frequently employs colloquialisms, slang, profanity, and language is dynamic and spatiotemporally sensitive, it is unlikely whichever lexicon polarity table is employed will be exhaustive.

Figure 2*a*
Temporal Sentiment Trajectories (Jockers & Rinker Polarity Lookup Table) Averaged over the Discography of Each Rap Group ***(100 Bins, Cluster Size 2)***

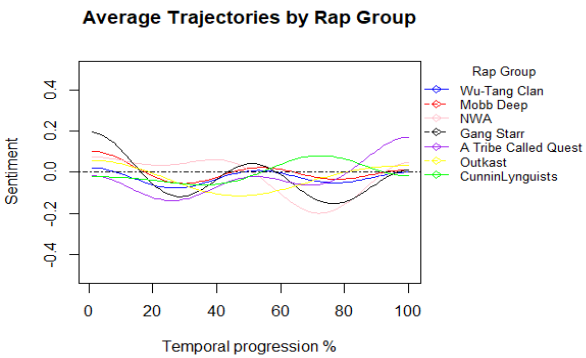


Figure 2b
Temporal Sentiment Trajectories (Jockers & Rinker Polarity Lookup Table) Averaged over the Discography of Each Rap Group ***(100 Bins, Cluster Size 3)***

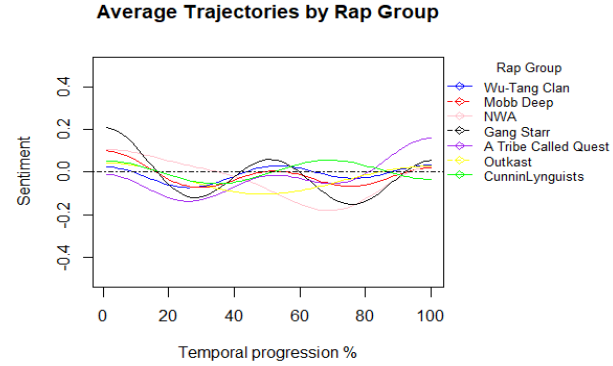


Fig
Te
Av
able)
ze 5)

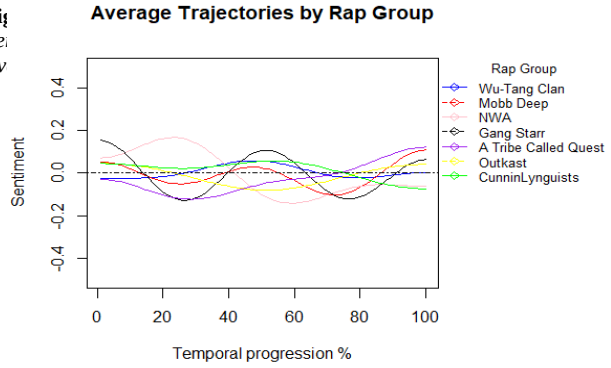


Figure 2d
Temporal Sentiment Trajectories (Jockers & Rinker Polarity Lookup Table) Averaged over the Discography of Each Rap Group ***(20 Bins, Cluster Size 2)***

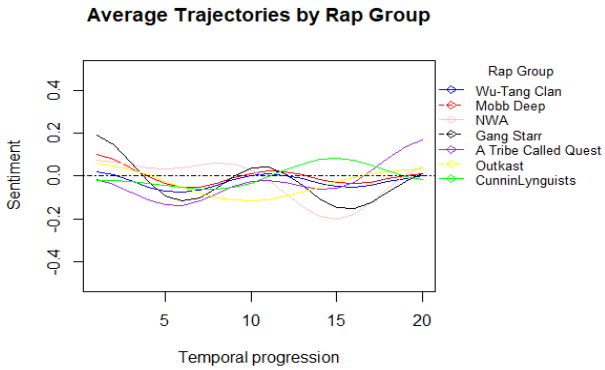
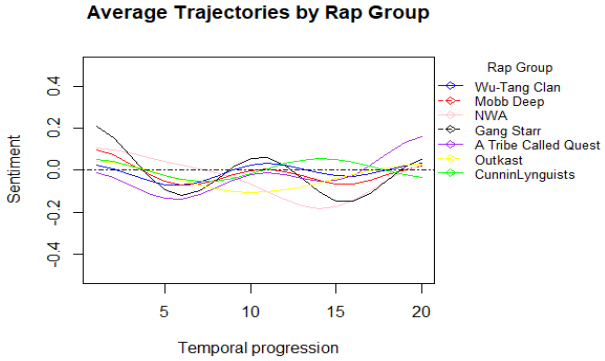


Figure 2e
Temporal Sentiment Trajectories (Jockers & Rinker Polarity Lookup Table) Averaged over the Discography of Each Rap Group ***(20 Bins, Cluster Size 3)***



Figur
Temp
Avera
le)
)

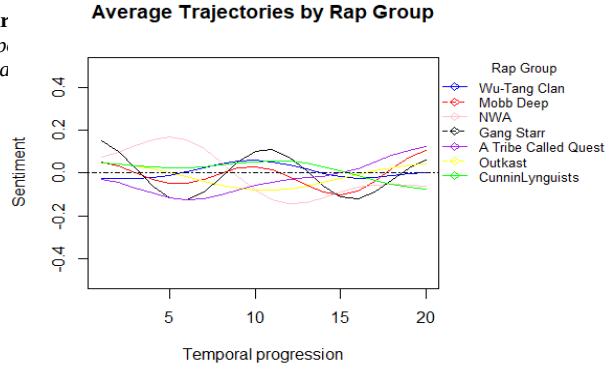


Figure 2g
Temporal Sentiment Trajectories (Jockers & Rinker Polarity Lookup Table) Averaged over the Discography of Each Rap Group ***(5 Bins, Cluster Size 2)***

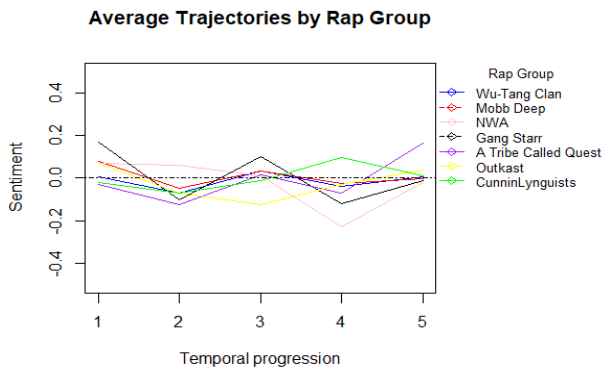


Figure 2h
Temporal Sentiment Trajectories (Jockers & Rinker Polarity Lookup Table) Averaged over the Discography of Each Rap Group ***(5 Bins, Cluster Size 3)***

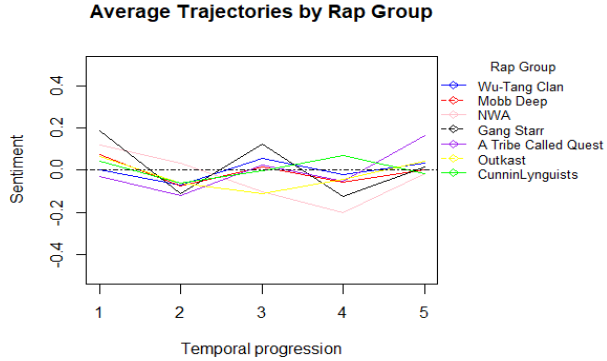
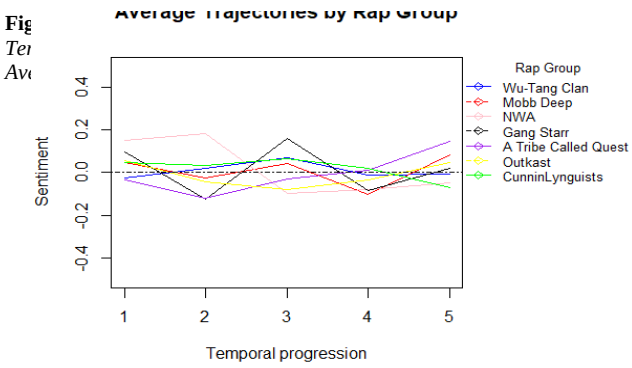


Fig
Ter
Ave
able)
5)



2) Use q-gram-based similarity calculations and analyse (for at least two authors) whether the internal similarity (i.e. the similarity among texts of a single authors) is higher than the similarity across authors (i.e. the similarity between texts written by different authors).

Word Count (excluding Table captions): 400 (Tables and captions are 140 words)

Table 1
Within Authors q-gram Similarities

		q-gram Similarity Mean			
		Levenshtein Similarity (**q=2**)	Levenshtein Similarity (**q=3**)	Cosine Similarity (**q=2**)	Cosine Similarity (**q=3**)
Rap Group	Wu-Tang Clan	0.291	0.291	0.903	0.888
	Mobb Deep	0.312	0.312	0.923	0.890
	NWA	0.281	0.281	0.866	0.838
	Outkast	0.276	0.276	0.861	0.884
	CunninLynguists	0.297	0.297	0.899	0.884
	A Tribe Called Quest	0.304	0.304	0.895	0.866
	Gang Starr	0.307	0.307	0.917	0.902

Mobb Deep displayed the highest within author song Levenshtein q-gram similarity, and Outkast the lowest. Interestingly, except for the duo Outkast, the Levenshtein similarity scores in descending order approximately follow the ascending number of main group members, implying more co-authors decreases rap q-gram similarity. But there was some disagreement between Levenshtein and Cosine q-gram similarities. When q was 2, the highest within author song similarity was also Mobb Deep, followed by Gang Starr, but next was Wu-Tang Clan and CunningLynguists; the same two rap groups had the lowest similarity relative to Levenshtein similarity though (NWA and Outkast). When q was 3, the top two highest cosine similarity authors swapped ranks relative to when q was 2. Furthermore, the two lowest similarity rap groups changed. In terms of result convergence, at most what may be said is that either Mobb Deep or Gang Starr had the highest similarity of character pairs or triplets (q-grams) within produced rap songs. Ultimately, it should be noted though that all the seven rap groups achieved fairly similar similarity scores.

Table 2
Between Author q-gram Similarities

		q-gram Similarity			
		Levenshtein Similarity (**q=2**)	Levenshtein Similarity (**q=3**)	Cosine Similarity (**q=2**)	Cosine Similarity (**q=3**)
Compared Sampled Discographies	Wu-Tang Clan to Mobb Deep	0.274	0.274	0.987	0.984
	Wu-Tang Clan to NWA	0.296	0.296	0.973	0.966
	Wu-Tang Clan to Outkast	0.309	0.309	0.960	0.979
	Wu-Tang Clan to CunninLynguists	0.300	0.300	0.985	0.984
	Wu-Tang Clan to A Tribe Called Quest	0.256	0.256	0.989	0.988
	Wu-Tang Clan to Gang Starr	0.297	0.297	0.974	0.981

Due to computing constraints, ten randomly sampled songs were concatenated into one string for each group before computing between author q-gram similarities; note, the representativeness of 10 songs for each group varied.

For brevity, Table 2 only displays Levenshtein and Cosine q-gram similarities for Wu-Tang Clan comparisons. The highest Levenshtein similarity was observed with Outkast, but for Cosine similarity this was with A Tribe Called Quest. Overall, compared to within author similarities, between author q-gram similarities were higher but again, Levenshtein and Cosine similarities slightly differed.

3) Extract the overall sentiment of each text in your corpus (i.e. a single score for each text). Now run build and run a supervised classification model and report detailed performance metrics (i.e. accuracy, precision, recall, F1 score, and area-under-the-curve).

Word Count (excluding Table captions): 395 (Table and captions are 68 words)

The lyrics from the discographies of the seven rap groups were web scraped from AZlyrics. Each song line was taken as a data instance, producing a corpus with 61,042 data instances. The Syuzhet R package was used to extract sentiment polarity scores, prior to categorisation into three labels: positive (polarity score above zero), negative (polarity score below zero) and neutral (polarity score was zero). Syuzhet has limitations though, such as the text "Bring da motherfuckin ruckus" from the Wu-Tang Clan song Bring Da Ruckus being assigned a sentiment score of zero.

Extracted text features included noun, verb, adverb and adjective counts as other part-of-speech tags are noisy for sentiment classification (Xia, et al., 2011), unigrams, bigrams, trigrams and finally, the percentage of positive and negative tokens in each song line (inspired by Bouazizi and Ohtsuki, 2016). Positive and negaive word lists were constructed through using the AFINN and bing sentiment word lists from the tidytext R package. Note, n-grams were TF-IDF weighted and sparsity corrected (0.99). Features with zero variance were excluded.

For this multiclass sentiment classification task, a linear SVM (Support-vector machine) was utilised, which has been used in sentiment classification literature (Go, et al., 2009). Model training was on 70% of data and testing on the remaining 30%. K-fold cross validation (k=10) was used. Table 3 displays the confusion matrix for the predictions on the test dataset.

Table 3
Confusion Matrix

		Original Sentiment Label		
		Positive	Neutral	Negative
Predicted Sentiment Label	Positive	2945	242	309
	Neutral	1672	5694	1424
	Negative	753	565	4707

The one versus all accuracy (Bleik & Gauher, 2016) was 81.923. As Ben-David (2008) recommended, a Kappa statistic was computed against expected random chance accuracy, which demonstrated moderate agreement (0.588). Furthermore, the macro average sensitivity, specificity, positive and negative predictive values were 71.839, 86.141, 75.714 and 87.017 respectively. The multiclass area under the curve was 0.836 (calculated via the multiclass.roc function of the pROC R package).

Table 4 displays macro, micro and weight average performance metrics for the model. Due to the class imbalances, it may be most appropriate to ascertain performance based on micro and weighted averages.

Table 4
Performance Metrics

		Metric			
		Accuracy	Precision	Recall	F1
Sentiment Label	Positive (N=3496)	72.885	84.239	54.841	66.434
	Neutral (N=8790)		64.778	87.587	74.475
	Negative (N=6025)		78.124	73.090	75.523
		Macro Average Metrics	75.714	71.839	72.144
		Micro Average Metrics		72.885	
		Weighted Average Metrics	72.885	76.564	73.285

REFERENCES

Ben-David, A. (2008). Comparison of classification accuracy using Cohen's Weighted Kappa. *Expert Systems with Applications*, 34(2), 825-832.

Bleik, S., & Gauher, S. (2016, March 11). *Computing Classification Evaluation Metrics in R*. Retrieved April 26, 2020, from https://blog.revolutionanalytics.com/2016/03/com_class_eval_metrics_r.html#kappa

Bouazizi, M., & Ohtsuki, T. (2016). Sentiment Analysis: from Binary to Multi-Class Classification; A Pattern-Based Approach for Multi-Class Sentiment Analysis in Twitter. *ICC 2016 : IEEE International Conference on Communications* (pp. 1-6). Kuala Lumpur: IEEE.

Go, A., Bhayani, R., & Huang, L. (2009). *Twitter Sentiment Classification using Distant Supervision.* Stanford: CS224N Project Report.

Kleinberg, B. A. (2020, January 7). *Naive context sentiment analysis*. Retrieved from GitHub: https://github.com/ben-aaron188/naive_context_sentiment

Xia, R., Zong, C., & Li, S. (2011). Ensemble of feature sets and classification algorithms for sentiment classification. *Information Sciences*, 181(6), 1138–1152.

Appendix (R Code)

Question 1 – Page 10 -> 24

Question 2 – Page 24 -> 33

Question 3 – Page 33 -> 41

```
# The code here is specifically for the questions in this coursework. The code for web scraping
azlyrics (and the 70% project) can be found at the repository
https://osf.io/y8rqb/?view_only=3e1c0b77b863417096674e3d3ac97f30

#importing some libraries (more libraries are imported throughout as and when needed) Apologies if I
have forgotten to write some libraries as I was copy pasting my code from my R Notebook

library(syuzhet)
library(lexicon)
library(caret)
library(quanteda)
```

Question 1

```
#DupRemoveDisc is a dataframe where each song is 1 data instance (row)

#LineDupRemove is a dataframe when each song line is 1 data instance (row)

#importing the ncs function from https://github.com/ben-aaron188/naive_context_sentiment

source('C:\\Users\\XXXXXX\\OneDrive\\Desktop\\XXXXXX\\Education\\UCL\\Year 3\\Term
2\\Data Science for Crime Scientists\\Project\\naive_context_sentiment-master\\ncs.R')

DupRemoveDisc$Lyrics = as.character(DupRemoveDisc$Lyrics) # converting the lyrics column to
character

# extract length-standardised trajectory

rap_groups_traj = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T)

#Plotting Sentiment Tracjectories for One Song from Each Rap Group
#Wu-Tang Clan: Protect Ya Neck (Index 10)
#Mobb Deep: Survival of the Fittest (Index 257)
#Outkast: B.O.B.(Bombs over Baghdad) (Index 775)
#NWA: Straight Outta Compton (Index 421)
#CunninLynguists: Lynguistics (Index 861)
#Gang Starr: Mass Appeal (Index 507)
#A Tribe Called Quest: After Hours (Index 591)


par(mar=c(5,4,5,15))
{plot(rap_groups_traj[[10]]
    , type='l'
    , col = 'blue'
    , ylim = c(-1,1)
    , ylab = 'Sentiment'
```

```
    , xlab = 'Temporal progression %'
    , main = 'Sentiment Trajectories for One Song from Each Rap Group'
    )

lines(rap_groups_traj[[257]], col='red')
lines(rap_groups_traj[[775]], col='yellow')
lines(rap_groups_traj[[421]], col='pink')
lines(rap_groups_traj[[861]], col='green')
lines(rap_groups_traj[[507]], col='black')
lines(rap_groups_traj[[591]], col='purple')


legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan: Protect Ya Neck", "Mobb Deep: Survival of the Fittest", "Outkast: B.O.B.", "NWA: Straight Outta Compton", "CunninLynguists: Lynguistics", "Gang Starr: Mass Appeal", "A Tribe Called Quest: After Hours"),
      col=c("blue", "red", "yellow", "pink", "green", "black", "purple"), pch=c(5,5), lty=1:2, cex=0.8, title="Song",xpd=TRUE,
       bty="n")
  }

## The data needs to be transposed so one row represents a rap song

transpose_rap_traj = t(rap_groups_traj)

# Making this transposed matrix into a data frame

df_transpose_rap = data.frame(transpose_rap_traj, row.names = row.names(transpose_rap_traj))

View(df_transpose_rap)

# adding a column variable that is the author (rap group in this case)

df_transpose_rap$Author = as.character(DupRemoveDisc$Author)

head(df_transpose_rap$Author)

# I found three songs (2 from Outkast and 1 from CunninLynguists) that had NaN sentiment values, so these were excluded so that mean sentiment trajectory values can be calculated and they can be plotted

df_transpose_rap[762,] # the outkast song Crusin' In The ATL (Interlude)

df_transpose_rap[768,] # the outkast song Speakerboxxx (Intro)

df_transpose_rap[872,] # the cunninlynguists song Where Will You Be

df_transpose_rapv2 = df_transpose_rap[-c(762,768,872),]

View(df_transpose_rapv2)

#making an object when the rows are rap groups and there are 100 columns (each cell value is the average of that bin i.e. temporal segment across all songs from that rap group)

shape_by_rap_group = aggregate(df_transpose_rapv2[1:100], by = list(df_transpose_rapv2$Author), mean)

### we now have the average sentiment per rap group for each of the 100 bin values

shape_by_rap_group
```

```
### for the plot, we want the trajectories in the columns so we transpose again

t_shape_by_group = t(shape_by_rap_group)

### create a data frame from the transposed object

df_t_shape_by_group = data.frame(t_shape_by_group)

View(df_t_shape_by_group)

### setting the column names according to the same order that they appeared in the aggregate object above (alphabetical order)

names(df_t_shape_by_group) = c('CunninLynguists', 'GangStarr', 'MobbDeep', 'NWA', 'Outkast', 'Tribe', 'WuTang')

### remove the first row now as it has the rap group names

df_t_shape_by_group = df_t_shape_by_group[-1,]

View(df_t_shape_by_group)

#convert each column to numeric values

df_t_shape_by_group$Outkast =  as.numeric(as.character(df_t_shape_by_group$Outkast))
df_t_shape_by_group$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group$CunninLynguists))
df_t_shape_by_group$GangStarr = as.numeric(as.character(df_t_shape_by_group$GangStarr))
df_t_shape_by_group$MobbDeep = as.numeric(as.character(df_t_shape_by_group$MobbDeep))
df_t_shape_by_group$NWA = as.numeric(as.character(df_t_shape_by_group$NWA))
df_t_shape_by_group$Tribe = as.numeric(as.character(df_t_shape_by_group$Tribe))
df_t_shape_by_group$WuTang = as.numeric(as.character(df_t_shape_by_group$WuTang))
View(df_t_shape_by_group)

#plotting the trajectories

par(mar=c(5,4,5,15))

{plot(df_t_shape_by_group$WuTang
    , type='l'
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression %'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group$MobbDeep, col='red')
lines(df_t_shape_by_group$NWA, col='pink')
lines(df_t_shape_by_group$GangStarr, col='black')
lines(df_t_shape_by_group$Tribe, col='purple')
lines(df_t_shape_by_group$Outkast, col='yellow')
lines(df_t_shape_by_group$CunninLynguists, col='green')
legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),

      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
       bty="n")
abline(h=0, lty=4)
  }
```

#From here on I begin experimenting with different bin sizes and cluster sizes

#Experimentation with 100 bins Different cluster Size (3)

```
rap_groups_Sentiment_traj_100bins_diffCluster = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T
                , lexicon = 'sentiment'
                , cluster_lower = 3
                , cluster_upper = 3
                , bins = 100)
#transposing so each row represents a rap song

transpose_rap_traj.100bin.3cluster = t(rap_groups_Sentiment_traj_100bins_diffCluster)

df_transpose_rap.100bin.3cluster = data.frame(transpose_rap_traj.100bin.3cluster, row.names =
row.names(transpose_rap_traj.100bin.3cluster))

#adding an author column

df_transpose_rap.100bin.3cluster$Author = as.character(DupRemoveDisc$Author)

head(df_transpose_rap.100bin.3cluster)

#Removing NaN songs and using dim to see if songs were removed (as there will be a decrease in
dimension)

dim(df_transpose_rap.100bin.3cluster)

Sent.100bin.3cluster.Omit = na.omit(df_transpose_rap.100bin.3cluster)

dim(Sent.100bin.3cluster.Omit)

#creating the object with mean sentiment at each of the 100 bins for the 7 groups

shape_by_rap_group.100bin.3cluster = aggregate(Sent.100bin.3cluster.Omit[1:100], by =
list(Sent.100bin.3cluster.Omit$Author), mean)

shape_by_rap_group.100bin.3cluster

#transposing again

t_shape_by_group.100bin.3cluster = t(shape_by_rap_group.100bin.3cluster)

df_t_shape_by_group.100bin.3cluster = data.frame(t_shape_by_group.100bin.3cluster)

names(df_t_shape_by_group.100bin.3cluster) = c('CunninLynguists', 'GangStarr', 'MobbDeep',
'NWA', 'Outkast', 'Tribe', 'WuTang')
#removing row with rap group names
df_t_shape_by_group.100bin.3cluster = df_t_shape_by_group.100bin.3cluster[-1,]

#making columns numeric
df_t_shape_by_group.100bin.3cluster$Outkast =
as.numeric(as.character(df_t_shape_by_group.100bin.3cluster$Outkast))
df_t_shape_by_group.100bin.3cluster$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group.100bin.3cluster$CunninLynguists))
df_t_shape_by_group.100bin.3cluster$GangStarr =
as.numeric(as.character(df_t_shape_by_group.100bin.3cluster$GangStarr))
df_t_shape_by_group.100bin.3cluster$MobbDeep =
as.numeric(as.character(df_t_shape_by_group.100bin.3cluster$MobbDeep))
```

```
df_t_shape_by_group.100bin.3cluster$NWA =
as.numeric(as.character(df_t_shape_by_group.100bin.3cluster$NWA))
df_t_shape_by_group.100bin.3cluster$Tribe =
as.numeric(as.character(df_t_shape_by_group.100bin.3cluster$Tribe))
df_t_shape_by_group.100bin.3cluster$WuTang =
as.numeric(as.character(df_t_shape_by_group.100bin.3cluster$WuTang))
View(df_t_shape_by_group.100bin.3cluster)
par(mar=c(5,4,5,15))

#plotting trajectories

{plot(df_t_shape_by_group.100bin.3cluster$WuTang
    , type='l'
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression %'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group.100bin.3cluster$MobbDeep, col='red')
lines(df_t_shape_by_group.100bin.3cluster$NWA, col='pink')
lines(df_t_shape_by_group.100bin.3cluster$GangStarr, col='black')
lines(df_t_shape_by_group.100bin.3cluster$Tribe, col='purple')
lines(df_t_shape_by_group.100bin.3cluster$Outkast, col='yellow')
lines(df_t_shape_by_group.100bin.3cluster$CunninLynguists, col='green')

legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),
      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
       bty="n")

abline(h=0, lty=4)
  }
```

#Experimenting with 100 bins and cluster size 5

```
rap_groups_Sentiment_traj_100bins_diffCluster5 = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T
                , lexicon = 'sentiment'
                , cluster_lower = 5
                , cluster_upper = 5
                , bins = 100)
transpose_rap_traj.100bins.5cluster = t(rap_groups_Sentiment_traj_100bins_diffCluster5) # each row
becomes 1 song

df_transpose_rap.100bins.5cluster = data.frame(transpose_rap_traj.100bins.5cluster, row.names =
row.names(transpose_rap_traj.100bins.5cluster))

df_transpose_rap.100bins.5cluster$Author = as.character(DupRemoveDisc$Author)

#omitting nans (rows where sentiment values are NaN)

dim(df_transpose_rap.100bins.5cluster)

Senti.100bins.5cluster.Omit = na.omit(df_transpose_rap.100bins.5cluster)
```

```
dim(Senti.100bins.5cluster.Omit)

#getting average sentiments at each bin for each rap group

shape_by_rap_group.100bins.5cluster = aggregate(Senti.100bins.5cluster.Omit[1:100], by =
list(Senti.100bins.5cluster.Omit$Author), mean)

shape_by_rap_group.100bins.5cluster

#transpose again

t_shape_by_group.100bins.5cluster = t(shape_by_rap_group.100bins.5cluster)

#create data frame

df_t_shape_by_group.100bins.5cluster = data.frame(t_shape_by_group.100bins.5cluster)

#change column names

names(df_t_shape_by_group.100bins.5cluster) = c('CunninLynguists', 'GangStarr', 'MobbDeep',
'NWA', 'Outkast', 'Tribe', 'WuTang')

df_t_shape_by_group.100bins.5cluster = df_t_shape_by_group.100bins.5cluster[-1,] # removing the
row with author names
#making columns numeric
df_t_shape_by_group.100bins.5cluster$Outkast =
as.numeric(as.character(df_t_shape_by_group.100bins.5cluster$Outkast))
df_t_shape_by_group.100bins.5cluster$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group.100bins.5cluster$CunninLynguists))
df_t_shape_by_group.100bins.5cluster$GangStarr =
as.numeric(as.character(df_t_shape_by_group.100bins.5cluster$GangStarr))
df_t_shape_by_group.100bins.5cluster$MobbDeep =
as.numeric(as.character(df_t_shape_by_group.100bins.5cluster$MobbDeep))
df_t_shape_by_group.100bins.5cluster$NWA =
as.numeric(as.character(df_t_shape_by_group.100bins.5cluster$NWA))
df_t_shape_by_group.100bins.5cluster$Tribe =
as.numeric(as.character(df_t_shape_by_group.100bins.5cluster$Tribe))
df_t_shape_by_group.100bins.5cluster$WuTang =
as.numeric(as.character(df_t_shape_by_group.100bins.5cluster$WuTang))
View(df_t_shape_by_group.100bins.5cluster)

#plotting trajectories

par(mar=c(5,4,5,15))

{plot(df_t_shape_by_group.100bins.5cluster$WuTang
    , type='l'
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression %'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group.100bins.5cluster$MobbDeep, col='red')
lines(df_t_shape_by_group.100bins.5cluster$NWA, col='pink')
lines(df_t_shape_by_group.100bins.5cluster$GangStarr, col='black')
lines(df_t_shape_by_group.100bins.5cluster$Tribe, col='purple')
lines(df_t_shape_by_group.100bins.5cluster$Outkast, col='yellow')
lines(df_t_shape_by_group.100bins.5cluster$CunninLynguists, col='green')
```

```
legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),
      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
       bty="n")
abline(h=0, lty=4) }
```

 # Experimentation with Different Bin Size (20), cluster size 2

```
rap_groups_Sentiment_traj_4bins = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T
                , lexicon = 'sentiment'
                , bins = 20)
rap_groups_Sentiment_traj_20bins = rap_groups_Sentiment_traj_4bins

transpose_rap_traj.20bins.2cluster = t(rap_groups_Sentiment_traj_20bins) # each row is now one rap
song

df_transpose_rap.20bins.2cluster = data.frame(transpose_rap_traj.20bins.2cluster, row.names =
row.names(transpose_rap_traj.20bins.2cluster))

#adding author column

df_transpose_rap.20bins.2cluster$Author = as.character(DupRemoveDisc$Author)

#Removing NaNs

dim(df_transpose_rap.20bins.2cluster)

Senti.20bins.2cluster.omit = na.omit(df_transpose_rap.20bins.2cluster)

dim(Senti.20bins.2cluster.omit)

#mean sentiment scores at each bin #using 1:20 as there are only 20 bins now

shape_by_rap_group.20bins.2cluster = aggregate(Senti.20bins.2cluster.omit[1:20], by =
list(Senti.20bins.2cluster.omit$Author), mean)

shape_by_rap_group.20bins.2cluster

t_shape_by_group.20bins.2cluster = t(shape_by_rap_group.20bins.2cluster)

df_t_shape_by_group.20bins.2cluster = data.frame(t_shape_by_group.20bins.2cluster)

#adding column names

names(df_t_shape_by_group.20bins.2cluster) = c('CunninLynguists', 'GangStarr', 'MobbDeep',
'NWA', 'Outkast', 'Tribe', 'WuTang')

df_t_shape_by_group.20bins.2cluster = df_t_shape_by_group.20bins.2cluster[-1,]
#converting columns into numeric
df_t_shape_by_group.20bins.2cluster$Outkast =
as.numeric(as.character(df_t_shape_by_group.20bins.2cluster$Outkast))
df_t_shape_by_group.20bins.2cluster$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group.20bins.2cluster$CunninLynguists))
df_t_shape_by_group.20bins.2cluster$GangStarr =
as.numeric(as.character(df_t_shape_by_group.20bins.2cluster$GangStarr))
```

```
df_t_shape_by_group.20bins.2cluster$MobbDeep =
as.numeric(as.character(df_t_shape_by_group.20bins.2cluster$MobbDeep))
df_t_shape_by_group.20bins.2cluster$NWA =
as.numeric(as.character(df_t_shape_by_group.20bins.2cluster$NWA))
df_t_shape_by_group.20bins.2cluster$Tribe =
as.numeric(as.character(df_t_shape_by_group.20bins.2cluster$Tribe))
df_t_shape_by_group.20bins.2cluster$WuTang =
as.numeric(as.character(df_t_shape_by_group.20bins.2cluster$WuTang))
# plotting trajectories

par(mar=c(5,4,5,15))
{plot(df_t_shape_by_group.20bins.2cluster$WuTang
    , type='l'
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group.20bins.2cluster$MobbDeep, col='red')
lines(df_t_shape_by_group.20bins.2cluster$NWA, col='pink')
lines(df_t_shape_by_group.20bins.2cluster$GangStarr, col='black')
lines(df_t_shape_by_group.20bins.2cluster$Tribe, col='purple')
lines(df_t_shape_by_group.20bins.2cluster$Outkast, col='yellow')
lines(df_t_shape_by_group.20bins.2cluster$CunninLynguists, col='green')

legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),
      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
       bty="n")

abline(h=0, lty=4) }
```

#Different Bin Size (20), cluster size 3

```
rap_groups_Sentiment_traj_20bins_3cluster = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T
                , lexicon = 'sentiment'
                , cluster_lower = 3
                , cluster_upper = 3
                , bins = 20)
#transpose so one row is one rap song
transpose_rap_traj.20bins.3cluster = t(rap_groups_Sentiment_traj_20bins_3cluster)

df_transpose_rap.20bins.3cluster = data.frame(transpose_rap_traj.20bins.3cluster, row.names =
row.names(transpose_rap_traj.20bins.3cluster))

df_transpose_rap.20bins.3cluster$Author = as.character(DupRemoveDisc$Author)

#NaN Removal

dim(df_transpose_rap.20bins.3cluster)

Senti.20bins.3cluster.omit = na.omit(df_transpose_rap.20bins.3cluster)

dim(Senti.20bins.3cluster.omit)
```

```
#average sentiment for each of the 20 bins for the seven rap groups

shape_by_rap_group.20bins.3cluster = aggregate(Senti.20bins.3cluster.omit[1:20], by =
list(Senti.20bins.3cluster.omit$Author), mean)

shape_by_rap_group.20bins.3cluster

t_shape_by_group.20bins.3cluster = t(shape_by_rap_group.20bins.3cluster)

df_t_shape_by_group.20bins.3cluster = data.frame(t_shape_by_group.20bins.3cluster)

names(df_t_shape_by_group.20bins.3cluster) = c('CunninLynguists', 'GangStarr', 'MobbDeep',
'NWA', 'Outkast', 'Tribe', 'WuTang') #changing column names

df_t_shape_by_group.20bins.3cluster = df_t_shape_by_group.20bins.3cluster[-1,] #removing row
with rap group names

#conversion into numeric columns

df_t_shape_by_group.20bins.3cluster$Outkast =
as.numeric(as.character(df_t_shape_by_group.20bins.3cluster$Outkast))
df_t_shape_by_group.20bins.3cluster$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group.20bins.3cluster$CunninLynguists))
df_t_shape_by_group.20bins.3cluster$GangStarr =
as.numeric(as.character(df_t_shape_by_group.20bins.3cluster$GangStarr))
df_t_shape_by_group.20bins.3cluster$MobbDeep =
as.numeric(as.character(df_t_shape_by_group.20bins.3cluster$MobbDeep))
df_t_shape_by_group.20bins.3cluster$NWA =
as.numeric(as.character(df_t_shape_by_group.20bins.3cluster$NWA))
df_t_shape_by_group.20bins.3cluster$Tribe =
as.numeric(as.character(df_t_shape_by_group.20bins.3cluster$Tribe))
df_t_shape_by_group.20bins.3cluster$WuTang =
as.numeric(as.character(df_t_shape_by_group.20bins.3cluster$WuTang))


#plot trajectories
par(mar=c(5,4,5,15))
{plot(df_t_shape_by_group.20bins.3cluster$WuTang
    , type='l'
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group.20bins.3cluster$MobbDeep, col='red')
lines(df_t_shape_by_group.20bins.3cluster$NWA, col='pink')
lines(df_t_shape_by_group.20bins.3cluster$GangStarr, col='black')
lines(df_t_shape_by_group.20bins.3cluster$Tribe, col='purple')
lines(df_t_shape_by_group.20bins.3cluster$Outkast, col='yellow')
lines(df_t_shape_by_group.20bins.3cluster$CunninLynguists, col='green')

legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),
      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
```

```
      bty="n")
abline(h=0, lty=4) }
```

#Different Bin Size (20), cluster size 5

```
rap_groups_Sentiment_traj_20bins_5cluster = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T
                , lexicon = 'sentiment'
                , cluster_lower = 5
                , cluster_upper = 5
                , bins = 20)
#transpose so each row is one rap song
transpose_rap_traj.20bins.5cluster = t(rap_groups_Sentiment_traj_20bins_5cluster)

df_transpose_rap.20bins.5cluster = data.frame(transpose_rap_traj.20bins.5cluster, row.names =
row.names(transpose_rap_traj.20bins.5cluster))

df_transpose_rap.20bins.5cluster$Author = as.character(DupRemoveDisc$Author)

#NaN removal

dim(df_transpose_rap.20bins.5cluster)

Senti.20bins.5cluster.omit = na.omit(df_transpose_rap.20bins.5cluster)

dim(Senti.20bins.5cluster.omit)

#average sentiment value for each of the 20 bins for each of the 7 rap groups

shape_by_rap_group.20bins.5cluster = aggregate(Senti.20bins.5cluster.omit[1:20], by =
list(Senti.20bins.5cluster.omit$Author), mean)

shape_by_rap_group.20bins.5cluster

t_shape_by_group.20bins.5cluster = t(shape_by_rap_group.20bins.5cluster)

df_t_shape_by_group.20bins.5cluster = data.frame(t_shape_by_group.20bins.5cluster)

# set column names

names(df_t_shape_by_group.20bins.5cluster) = c('CunninLynguists', 'GangStarr', 'MobbDeep',
'NWA', 'Outkast', 'Tribe', 'WuTang')

### removing the first row now as it has the rap group names

df_t_shape_by_group.20bins.5cluster = df_t_shape_by_group.20bins.5cluster[-1,]

#converting columns into numeric columns

df_t_shape_by_group.20bins.5cluster$Outkast =
as.numeric(as.character(df_t_shape_by_group.20bins.5cluster$Outkast))
df_t_shape_by_group.20bins.5cluster$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group.20bins.5cluster$CunninLynguists))
df_t_shape_by_group.20bins.5cluster$GangStarr =
as.numeric(as.character(df_t_shape_by_group.20bins.5cluster$GangStarr))
df_t_shape_by_group.20bins.5cluster$MobbDeep =
as.numeric(as.character(df_t_shape_by_group.20bins.5cluster$MobbDeep))
```

```
df_t_shape_by_group.20bins.5cluster$NWA =
as.numeric(as.character(df_t_shape_by_group.20bins.5cluster$NWA))
df_t_shape_by_group.20bins.5cluster$Tribe =
as.numeric(as.character(df_t_shape_by_group.20bins.5cluster$Tribe))
df_t_shape_by_group.20bins.5cluster$WuTang =
as.numeric(as.character(df_t_shape_by_group.20bins.5cluster$WuTang))


#plot trajectories

par(mar=c(5,4,5,15))
{plot(df_t_shape_by_group.20bins.5cluster$WuTang
    , type='l'
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group.20bins.5cluster$MobbDeep, col='red')
lines(df_t_shape_by_group.20bins.5cluster$NWA, col='pink')
lines(df_t_shape_by_group.20bins.5cluster$GangStarr, col='black')
lines(df_t_shape_by_group.20bins.5cluster$Tribe, col='purple')
lines(df_t_shape_by_group.20bins.5cluster$Outkast, col='yellow')
lines(df_t_shape_by_group.20bins.5cluster$CunninLynguists, col='green')

legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),
      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
       bty="n")
abline(h=0, lty=4) }
```

#Different Bin Size (5), cluster size 2

```
rap_groups_Sentiment_traj_5bins_2cluster = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T
                , lexicon = 'sentiment'
                , cluster_lower = 2
                , cluster_upper = 2
                , bins = 5)
#transpose so each row is 1 rap song
transpose_rap_traj.5bin.2cluster = t(rap_groups_Sentiment_traj_5bins_2cluster)

df_transpose_rap.5bin.2cluster = data.frame(transpose_rap_traj.5bin.2cluster, row.names =
row.names(transpose_rap_traj.5bin.2cluster))

df_transpose_rap.5bin.2cluster$Author = as.character(DupRemoveDisc$Author)

#NaN omit/removal

dim(df_transpose_rap.5bin.2cluster)

Senti.5bins.2cluster.omit = na.omit(df_transpose_rap.5bin.2cluster)

dim(Senti.5bins.2cluster.omit)
```

```
shape_by_rap_group.5bins.2cluster = aggregate(Senti.5bins.2cluster.omit[1:5], by =
list(Senti.5bins.2cluster.omit$Author), mean)

shape_by_rap_group.5bins.2cluster

t_shape_by_group.5bins.2cluster = t(shape_by_rap_group.5bins.2cluster)

df_t_shape_by_group.5bins.2cluster = data.frame(t_shape_by_group.5bins.2cluster)

# adding column names

names(df_t_shape_by_group.5bins.2cluster) = c('CunninLynguists', 'GangStarr', 'MobbDeep', 'NWA',
'Outkast', 'Tribe', 'WuTang')
### remove the first row now as it has the rap group names
df_t_shape_by_group.5bins.2cluster = df_t_shape_by_group.5bins.2cluster[-1,]
#convert each coloumn to numeric
df_t_shape_by_group.5bins.2cluster$Outkast =
as.numeric(as.character(df_t_shape_by_group.5bins.2cluster$Outkast))
df_t_shape_by_group.5bins.2cluster$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group.5bins.2cluster$CunninLynguists))
df_t_shape_by_group.5bins.2cluster$GangStarr =
as.numeric(as.character(df_t_shape_by_group.5bins.2cluster$GangStarr))
df_t_shape_by_group.5bins.2cluster$MobbDeep =
as.numeric(as.character(df_t_shape_by_group.5bins.2cluster$MobbDeep))
df_t_shape_by_group.5bins.2cluster$NWA =
as.numeric(as.character(df_t_shape_by_group.5bins.2cluster$NWA))
df_t_shape_by_group.5bins.2cluster$Tribe =
as.numeric(as.character(df_t_shape_by_group.5bins.2cluster$Tribe))
df_t_shape_by_group.5bins.2cluster$WuTang =
as.numeric(as.character(df_t_shape_by_group.5bins.2cluster$WuTang))

#plot trajectories
par(mar=c(5,4,5,15))
{plot(df_t_shape_by_group.5bins.2cluster$WuTang
    , type='l'
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group.5bins.2cluster$MobbDeep, col='red')
lines(df_t_shape_by_group.5bins.2cluster$NWA, col='pink')
lines(df_t_shape_by_group.5bins.2cluster$GangStarr, col='black')
lines(df_t_shape_by_group.5bins.2cluster$Tribe, col='purple')
lines(df_t_shape_by_group.5bins.2cluster$Outkast, col='yellow')
lines(df_t_shape_by_group.5bins.2cluster$CunninLynguists, col='green')

legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),
      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
       bty="n")
abline(h=0, lty=4) }
```

 # 5 Bins, cluster size 3

```
rap_groups_Sentiment_traj_5bins_3cluster = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T
                , lexicon = 'sentiment'
                , cluster_lower = 3
                , cluster_upper = 3
                , bins = 5)
#transpose so each row is one rap song
transpose_rap_traj.5bins.3clusters = t(rap_groups_Sentiment_traj_5bins_3cluster)

df_transpose_rap.5bins.3clusters = data.frame(transpose_rap_traj.5bins.3clusters, row.names =
row.names(transpose_rap_traj.5bins.3clusters))

df_transpose_rap.5bins.3clusters$Author = as.character(DupRemoveDisc$Author)

#NAN omittance

dim(df_transpose_rap.5bins.3clusters)

Senti.5bins.3cluster.omit = na.omit(df_transpose_rap.5bins.3clusters)

dim(Senti.5bins.3cluster.omit)

#average sentiment value for each of the five bins for the seven rap groups

shape_by_rap_group.5bins.3cluster = aggregate(Senti.5bins.3cluster.omit[1:5], by =
list(Senti.5bins.3cluster.omit$Author), mean)

shape_by_rap_group.5bins.3cluster

t_shape_by_group.5bins.3cluster = t(shape_by_rap_group.5bins.3cluster)

df_t_shape_by_group.5bins.3cluster = data.frame(t_shape_by_group.5bins.3cluster)

names(df_t_shape_by_group.5bins.3cluster) = c('CunninLynguists', 'GangStarr', 'MobbDeep', 'NWA',
'Outkast', 'Tribe', 'WuTang') # column names

df_t_shape_by_group.5bins.3cluster = df_t_shape_by_group.5bins.3cluster[-1,]

#making numeric columns

df_t_shape_by_group.5bins.3cluster$Outkast =
as.numeric(as.character(df_t_shape_by_group.5bins.3cluster$Outkast))
df_t_shape_by_group.5bins.3cluster$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group.5bins.3cluster$CunninLynguists))
df_t_shape_by_group.5bins.3cluster$GangStarr =
as.numeric(as.character(df_t_shape_by_group.5bins.3cluster$GangStarr))
df_t_shape_by_group.5bins.3cluster$MobbDeep =
as.numeric(as.character(df_t_shape_by_group.5bins.3cluster$MobbDeep))
df_t_shape_by_group.5bins.3cluster$NWA =
as.numeric(as.character(df_t_shape_by_group.5bins.3cluster$NWA))
df_t_shape_by_group.5bins.3cluster$Tribe =
as.numeric(as.character(df_t_shape_by_group.5bins.3cluster$Tribe))
df_t_shape_by_group.5bins.3cluster$WuTang =
as.numeric(as.character(df_t_shape_by_group.5bins.3cluster$WuTang))
#plotting trajectories
par(mar=c(5,4,5,15))
{plot(df_t_shape_by_group.5bins.3cluster$WuTang
    , type='l'
```

```
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group.5bins.3cluster$MobbDeep, col='red')
lines(df_t_shape_by_group.5bins.3cluster$NWA, col='pink')
lines(df_t_shape_by_group.5bins.3cluster$GangStarr, col='black')
lines(df_t_shape_by_group.5bins.3cluster$Tribe, col='purple')
lines(df_t_shape_by_group.5bins.3cluster$Outkast, col='yellow')
lines(df_t_shape_by_group.5bins.3cluster$CunninLynguists, col='green')

legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),
      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
       bty="n")
abline(h=0, lty=4) }
```

#5 bins, cluster size 5 (FINAL EXPERIMENT)

```
rap_groups_Sentiment_traj_5bins_5cluster = ncs_full(txt_input_col = DupRemoveDisc$Lyrics
                , txt_id_col = DupRemoveDisc$z
                , bin_transform = T
                , lexicon = 'sentiment'
                , cluster_lower = 5
                , cluster_upper = 5
                , bins = 5)
#transposing so 1 row is one rap song
transpose_rap_traj.5bins.5clusters = t(rap_groups_Sentiment_traj_5bins_5cluster)

df_transpose_rap.5bins.5clusters = data.frame(transpose_rap_traj.5bins.5clusters, row.names =
row.names(transpose_rap_traj.5bins.5clusters))

df_transpose_rap.5bins.5clusters$Author = as.character(DupRemoveDisc$Author)

#NaN removal

dim(df_transpose_rap.5bins.5clusters)

Senti.5bins.5cluster.omit = na.omit(df_transpose_rap.5bins.5clusters)

dim(Senti.5bins.5cluster.omit)

#calculating sentiment average value for each bin for each of the seven rap groups

shape_by_rap_group.5bins.5cluster = aggregate(Senti.5bins.5cluster.omit[1:5], by =
list(Senti.5bins.5cluster.omit$Author), mean)

shape_by_rap_group.5bins.5cluster

t_shape_by_group.5bins.5cluster = t(shape_by_rap_group.5bins.5cluster)

df_t_shape_by_group.5bins.5cluster = data.frame(t_shape_by_group.5bins.5cluster)

names(df_t_shape_by_group.5bins.5cluster) = c('CunninLynguists', 'GangStarr', 'MobbDeep', 'NWA',
'Outkast', 'Tribe', 'WuTang') #column names
```

```
df_t_shape_by_group.5bins.5cluster = df_t_shape_by_group.5bins.5cluster[-1,] # removing row with rap group names

#making columns numeric

df_t_shape_by_group.5bins.5cluster$Outkast =
as.numeric(as.character(df_t_shape_by_group.5bins.5cluster$Outkast))
df_t_shape_by_group.5bins.5cluster$CunninLynguists =
as.numeric(as.character(df_t_shape_by_group.5bins.5cluster$CunninLynguists))
df_t_shape_by_group.5bins.5cluster$GangStarr =
as.numeric(as.character(df_t_shape_by_group.5bins.5cluster$GangStarr))
df_t_shape_by_group.5bins.5cluster$MobbDeep =
as.numeric(as.character(df_t_shape_by_group.5bins.5cluster$MobbDeep))
df_t_shape_by_group.5bins.5cluster$NWA =
as.numeric(as.character(df_t_shape_by_group.5bins.5cluster$NWA))
df_t_shape_by_group.5bins.5cluster$Tribe =
as.numeric(as.character(df_t_shape_by_group.5bins.5cluster$Tribe))
df_t_shape_by_group.5bins.5cluster$WuTang =
as.numeric(as.character(df_t_shape_by_group.5bins.5cluster$WuTang))
par(mar=c(5,4,5,15))

#plot trajectories

{plot(df_t_shape_by_group.5bins.5cluster$WuTang
    , type='l'
    , col = 'blue'
    , ylim = c(-0.5,0.5)
    , ylab = 'Sentiment'
    , xlab = 'Temporal progression'
    , main = 'Average Trajectories by Rap Group')
lines(df_t_shape_by_group.5bins.5cluster$MobbDeep, col='red')
lines(df_t_shape_by_group.5bins.5cluster$NWA, col='pink')
lines(df_t_shape_by_group.5bins.5cluster$GangStarr, col='black')
lines(df_t_shape_by_group.5bins.5cluster$Tribe, col='purple')
lines(df_t_shape_by_group.5bins.5cluster$Outkast, col='yellow')
lines(df_t_shape_by_group.5bins.5cluster$CunninLynguists, col='green')

legend("topleft", inset=c(1,0), 2, legend=c("Wu-Tang Clan", "Mobb Deep", "NWA", "Gang Starr",
"A Tribe Called Quest", "Outkast", "CunninLynguists"),
      col=c("blue", "red", "pink", "black", "purple", "yellow", "green"), pch=c(5,5), lty=1:2, cex=0.8,
title="Rap Group",xpd=TRUE,
       bty="n")
abline(h=0, lty=4) }
```

Question 2

#Within Authors

I made filters so that I could filter authors by these character vectors, and thereby get song lines just for each rap group

```
CunFilter = c("Cun")
GangFilter = c("GangStarr")
MobbFilter = c("MobbDeep")
NWAFilter = c("NWA")
```

```
OutFilter = c("Outkast")
TribeFilter = c("Tribe")
WuFilter = c("WuTang")

#Note DupRemoveDisc is a dataframe where each row has the whole song's lyrics, so there are a total of 988 songs across the 7 rap groups

CunOnlySongs = subset(DupRemoveDisc, Author %in% CunFilter)

#View(CunOnlySongs)

dim(CunOnlySongs) #checking the dimensions of the outputted dataframe to see if there are the expected number of songs per rap group (should be 150 CunningLynguists songs after duplicate removal)

GangOnlySongs = subset(DupRemoveDisc, Author %in% GangFilter)
dim(GangOnlySongs) # should be 134 rows/songs
MobbOnlySongs = subset(DupRemoveDisc, Author %in% MobbFilter)
dim(MobbOnlySongs) # should be 162
NWAOnlySongs = subset(DupRemoveDisc, Author %in% NWAFilter)
dim(NWAOnlySongs) # should be 43
OutkastOnlySongs = subset(DupRemoveDisc, Author %in% OutFilter)
dim(OutkastOnlySongs) # should be 141
TribeOnlySongs = subset(DupRemoveDisc, Author %in% TribeFilter)
dim(TribeOnlySongs) #  should be 112 songs after duplicate removal
WuTangOnlySongs = subset(DupRemoveDisc, Author %in% WuFilter)
dim(WuTangOnlySongs) # should be 246


library(stringdist)

#The logic for the following loop is best explained with 5 songs
#1 2 3 4 5
#on the first iteration there will be four comparisons (1 to 2, 1 to 3, 1 to 4, 1 to 5)
#on the second iteration there will be three comparisons (2 to 3, 2 to 4, 2 to 5)
#on the third iteration there will be two comparisons (3 to 4, 3 to 5)
#on the fourth iteration there will be one comparison (4 to 5)

# I used this link to learn how to properly loop over a dataframe

#https://stackoverflow.com/questions/24917228/proper-way-to-loop-over-the-length-of-a-dataframe-in-r

#counter = 0

#CunninLynguists within author

for (i in seq_len(nrow(CunOnlySongs))){
   #print(i)
   if (i > 149) {break} # because the last song has no comparisons to do (as group only had 149 songs)

  counter2 = i + 1
  FirstSong = CunOnlySongs[i, 4] #gets first song lyrics
  SecondSong = CunOnlySongs[counter2, 4] #gets second song lyrics
  # if (counter < 5) {
  #
  # }
```

```
  LevDistq2 = stringsim(FirstSong, SecondSong, q=2, method='lv')
  CosDistq2 = stringsim(FirstSong, SecondSong, q=2, method='cosine')
  LevDistq3 = stringsim(FirstSong, SecondSong, q=3, method='lv')
  CosDistq3 = stringsim(FirstSong, SecondSong, q=3, method='cosine')
    # CunLevDistq2", "CunCosDistq2","CunLevDistq3", "CunCosDistq3
    NewDFRow = data.frame(CunLevDistq2=LevDistq2, CunCosDistq2=CosDistq2,
CunLevDistq3=LevDistq3, CunCosDistq3=CosDistq3)
    CunDistDf = rbind(CunDistDf, NewDFRow)
  #counter = counter + 1
}
View(CunDistDf)

#means and standard deviations for similarities

mean(CunDistDf$CunLevDistq2)
sd(CunDistDf$CunLevDistq2)
mean(CunDistDf$CunCosDistq2)
sd(CunDistDf$CunCosDistq2)
mean(CunDistDf$CunLevDistq3)
sd(CunDistDf$CunLevDistq3)
mean(CunDistDf$CunCosDistq3)
sd(CunDistDf$CunCosDistq3)
```

#Gang Starr within author

```
GangDistNames = c("GangLevDistq2", "GangCosDistq2","GangLevDistq3", "GangCosDistq3")

GangDistDf = data.frame(matrix(ncol=4, nrow=0))

colnames(GangDistDf) = GangDistNames

View(GangDistDf)

GangOnlySongs$Lyrics = as.character(GangOnlySongs$Lyrics)

View(GangOnlySongs)

for (i in seq_len(nrow(GangOnlySongs))){
  #print(i)
  if (i > 133) {break} # because the last song has no comparisons to do (group had 133 songs)
  counter2 = i + 1
  FirstSong = GangOnlySongs[i, 4] #gets first song lyrics
  SecondSong = GangOnlySongs[counter2, 4] #gets second song lyrics
  LevDistq2 = stringsim(FirstSong, SecondSong, q=2, method='lv')
  CosDistq2 = stringsim(FirstSong, SecondSong, q=2, method='cosine')
  LevDistq3 = stringsim(FirstSong, SecondSong, q=3, method='lv')
  CosDistq3 = stringsim(FirstSong, SecondSong, q=3, method='cosine')
  NewDFRow = data.frame(GangLevDistq2=LevDistq2, GangCosDistq2=CosDistq2,
GangLevDistq3=LevDistq3, GangCosDistq3=CosDistq3)
  GangDistDf = rbind(GangDistDf, NewDFRow)
}
View(GangDistDf)

#means and standard deviations of gang starr within author similarities

mean(GangDistDf$GangLevDistq2)
sd(GangDistDf$GangLevDistq2)
mean(GangDistDf$GangLevDistq3)
```

```
sd(GangDistDf$GangLevDistq3)
mean(GangDistDf$GangCosDistq2)
sd(GangDistDf$GangCosDistq2)
mean(GangDistDf$GangCosDistq3)
sd(GangDistDf$GangCosDistq3)
```

#Mobb Deep within author

```
ColDistNames = c("LevDistq2", "CosDistq2","LevDistq3", "CosDistq3")

MobbDistDf = data.frame(matrix(ncol=4, nrow=0))

colnames(MobbDistDf) = ColDistNames

View(MobbDistDf)

MobbOnlySongs$Lyrics = as.character(MobbOnlySongs$Lyrics)

View(MobbOnlySongs)

for (i in seq_len(nrow(MobbOnlySongs))){
  #print(i)

  if (i > 161) {break} # because the last song has no comparisons to do
  counter2 = i + 1
  FirstSong = MobbOnlySongs[i, 4] #gets first song lyrics
  SecondSong = MobbOnlySongs[counter2, 4] #gets second song lyrics
  LevDistq2value = stringsim(FirstSong, SecondSong, q=2, method='lv')
  CosDistq2value = stringsim(FirstSong, SecondSong, q=2, method='cosine')
  LevDistq3value = stringsim(FirstSong, SecondSong, q=3, method='lv')
  CosDistq3value = stringsim(FirstSong, SecondSong, q=3, method='cosine')
  NewDFRow = data.frame(LevDistq2=LevDistq2value, CosDistq2=CosDistq2value,
LevDistq3=LevDistq3value, CosDistq3=CosDistq3value)
  MobbDistDf = rbind(MobbDistDf, NewDFRow)
}

View(MobbDistDf)

#means and standard deviations of within author similarities

mean(MobbDistDf$LevDistq2)
sd(MobbDistDf$LevDistq2)
mean(MobbDistDf$LevDistq3)
sd(MobbDistDf$LevDistq3)
mean(MobbDistDf$CosDistq2)
sd(MobbDistDf$CosDistq2)
mean(MobbDistDf$CosDistq3)
sd(MobbDistDf$CosDistq3)
```

#NWA within author

```
NWADistDf = data.frame(matrix(ncol=4, nrow=0))

colnames(NWADistDf) = ColDistNames

View(NWADistDf)

NWAOnlySongs$Lyrics = as.character(NWAOnlySongs$Lyrics)

View(NWAOnlySongs)
```

```
for (i in seq_len(nrow(NWAOnlySongs))){
  if (i > 42) {break} # because the last song has no comparisons to do
  counter2 = i + 1
  FirstSong = NWAOnlySongs[i, 4] #gets first song lyrics
  SecondSong = NWAOnlySongs[counter2, 4] #gets second song lyrics


  LevDistq2value = stringsim(FirstSong, SecondSong, q=2, method='lv')
  CosDistq2value = stringsim(FirstSong, SecondSong, q=2, method='cosine')
  LevDistq3value = stringsim(FirstSong, SecondSong, q=3, method='lv')
  CosDistq3value = stringsim(FirstSong, SecondSong, q=3, method='cosine')
  NewDFRow = data.frame(LevDistq2=LevDistq2value, CosDistq2=CosDistq2value,
LevDistq3=LevDistq3value, CosDistq3=CosDistq3value)
  NWADistDf = rbind(NWADistDf, NewDFRow)
}
View(NWADistDf)

#means and standard deviations of within author similarities

mean(NWADistDf$LevDistq2)
sd(NWADistDf$LevDistq2)
mean(NWADistDf$LevDistq3)
sd(NWADistDf$LevDistq3)
mean(NWADistDf$CosDistq2)
sd(NWADistDf$CosDistq2)
mean(NWADistDf$CosDistq3)
sd(NWADistDf$CosDistq3)
```

#Outkast within author similarities

```
OutkastDistDf = data.frame(matrix(ncol=4, nrow=0))

colnames(OutkastDistDf) = ColDistNames

View(OutkastDistDf)

OutkastOnlySongs$Lyrics = as.character(OutkastOnlySongs$Lyrics)

View(OutkastOnlySongs)

for (i in seq_len(nrow(OutkastOnlySongs))){
  if (i > 140) {break} # because the last song has no comparisons to do
  counter2 = i + 1
  FirstSong = OutkastOnlySongs[i, 4] #gets first song lyrics
  SecondSong = OutkastOnlySongs[counter2, 4] #gets second song lyrics
  LevDistq2value = stringsim(FirstSong, SecondSong, q=2, method='lv')
  CosDistq2value = stringsim(FirstSong, SecondSong, q=2, method='cosine')
  LevDistq3value = stringsim(FirstSong, SecondSong, q=3, method='lv')
  CosDistq3value = stringsim(FirstSong, SecondSong, q=3, method='cosine')
  NewDFRow = data.frame(LevDistq2=LevDistq2value, CosDistq2=CosDistq2value,
LevDistq3=LevDistq3value, CosDistq3=CosDistq3value)
  OutkastDistDf = rbind(OutkastDistDf, NewDFRow)
}
View(OutkastDistDf)

#means and standard deviations of within author similarities

mean(OutkastDistDf$LevDistq2)
```

```
sd(OutkastDistDf$LevDistq2)
mean(OutkastDistDf$LevDistq3)
sd(OutkastDistDf$LevDistq3)
mean(OutkastDistDf$CosDistq2)
sd(OutkastDistDf$CosDistq2)
mean(OutkastDistDf$CosDistq3)
sd(OutkastDistDf$CosDistq3)
```

#A tribe called quest within author similarity

```
TribeDistDf = data.frame(matrix(ncol=4, nrow=0))

colnames(TribeDistDf) = ColDistNames

View(TribeDistDf)

TribeOnlySongs$Lyrics = as.character(TribeOnlySongs$Lyrics)

View(TribeOnlySongs)

for (i in seq_len(nrow(TribeOnlySongs))){
  if (i > 111) {break} # because the last song has no comparisons to do
  counter2 = i + 1
  FirstSong = TribeOnlySongs[i, 4] #gets first song lyrics
  SecondSong = TribeOnlySongs[counter2, 4] #gets second song lyrics
  LevDistq2value = stringsim(FirstSong, SecondSong, q=2, method='lv')
  CosDistq2value = stringsim(FirstSong, SecondSong, q=2, method='cosine')
  LevDistq3value = stringsim(FirstSong, SecondSong, q=3, method='lv')
  CosDistq3value = stringsim(FirstSong, SecondSong, q=3, method='cosine')
  NewDFRow = data.frame(LevDistq2=LevDistq2value, CosDistq2=CosDistq2value,
LevDistq3=LevDistq3value, CosDistq3=CosDistq3value)
  TribeDistDf = rbind(TribeDistDf, NewDFRow)

}
View(TribeDistDf)

mean(TribeDistDf$LevDistq2)
sd(TribeDistDf$LevDistq2)
mean(TribeDistDf$LevDistq3)
sd(TribeDistDf$LevDistq3)
mean(TribeDistDf$CosDistq2)
sd(TribeDistDf$CosDistq2)
mean(TribeDistDf$CosDistq3)
sd(TribeDistDf$CosDistq3)
```

#Wu-Tang Clan within author similarity

```
WuTangDistDf = data.frame(matrix(ncol=4, nrow=0))

colnames(WuTangDistDf) = ColDistNames

View(WuTangDistDf)

WuTangOnlySongs$Lyrics = as.character(WuTangOnlySongs$Lyrics)

View(WuTangOnlySongs)

for (i in seq_len(nrow(WuTangOnlySongs))){
```

```
  if (i > 245) {break} # because the last song has no comparisons to do
  counter2 = i + 1
  FirstSong = WuTangOnlySongs[i, 4] #gets first song lyrics
  SecondSong = WuTangOnlySongs[counter2, 4] #gets second song lyrics
  LevDistq2value = stringsim(FirstSong, SecondSong, q=2, method='lv')
  CosDistq2value = stringsim(FirstSong, SecondSong, q=2, method='cosine')
  LevDistq3value = stringsim(FirstSong, SecondSong, q=3, method='lv')
  CosDistq3value = stringsim(FirstSong, SecondSong, q=3, method='cosine')
  NewDFRow = data.frame(LevDistq2=LevDistq2value, CosDistq2=CosDistq2value,
LevDistq3=LevDistq3value, CosDistq3=CosDistq3value)
  WuTangDistDf = rbind(WuTangDistDf, NewDFRow)
}
View(WuTangDistDf)

#means and standard deviations of within author similarities

mean(WuTangDistDf$LevDistq2)
sd(WuTangDistDf$LevDistq2)
mean(WuTangDistDf$LevDistq3)
sd(WuTangDistDf$LevDistq3)
mean(WuTangDistDf$CosDistq2)
sd(WuTangDistDf$CosDistq2)
mean(WuTangDistDf$CosDistq3)
sd(WuTangDistDf$CosDistq3)
```

#BETWEEN AUTHORS

#I originally wanted to concatenate every song into one string for each rap group to produce seven strings, each containing the entire discography of a single rap group. However, when I attempted to run the q-gram calculations, this caused the R Session to abort as some rap groups had large discographies. E.g. for Wu-Tang Clan, their concatenated string with all their songs contained 150,012 tokens and Mobb Deep had 102,365 tokens. Randomly sampling 20% and even 5% of each rap group's songs then, and concatenate these lyrics into a string before doing the q-gram calculations, also too caused R to crash, so I went for 10 random songs of each rap group

```
# Sampling

set.seed(5)

Sample10Wu = WuTangOnlySongs[sample(nrow(WuTangOnlySongs), 10), ]
nrow(Sample10Wu)
Sample10Cun = CunOnlySongs[sample(nrow(CunOnlySongs), 10), ]
nrow(Sample10Cun)
Sample10Mobb = MobbOnlySongs[sample(nrow(MobbOnlySongs), 10), ]
nrow(Sample10Mobb)
Sample10NWA = NWAOnlySongs[sample(nrow(NWAOnlySongs), 10), ]
nrow(Sample10NWA)
Sample10Outkast = OutkastOnlySongs[sample(nrow(OutkastOnlySongs), 10), ]
nrow(Sample10Outkast)
Sample10Tribe = TribeOnlySongs[sample(nrow(TribeOnlySongs), 10), ]
nrow(Sample10Tribe)
Sample10Gang = GangOnlySongs[sample(nrow(GangOnlySongs), 10), ]
nrow(Sample10Gang)

#Now I begin concatenating these 10 random songs from each artists into one string for each artist
```

```
WuTangDiscoString = ""
#View(WuTangOnlySongs)
for (i in seq_len(nrow(Sample10Wu))){
  RowSongLyrics = Sample10Wu[i, 5] #gets first song lyrics
  WuTangDiscoString = paste(WuTangDiscoString, RowSongLyrics, sep = " ", collapse=NULL) #
adding the current song to the string
  #print(ntoken(WuTangDiscoString))
}
print(ntoken(WuTangDiscoString))


MobbDeepDiscoString = ""
#View(WuTangOnlySongs)
for (i in seq_len(nrow(Sample10Mobb))){
  RowSongLyrics = Sample10Mobb[i, 5] #gets first song lyrics
  MobbDeepDiscoString = paste(MobbDeepDiscoString, RowSongLyrics, sep = " ", collapse=NULL)
# adding the current song to the string
}
print(ntoken(MobbDeepDiscoString))


NWADiscoString = ""
for (i in seq_len(nrow(Sample10NWA))){
  RowSongLyrics = Sample10NWA[i, 5] #gets first song lyrics
  NWADiscoString = paste(NWADiscoString, RowSongLyrics, sep = " ", collapse=NULL) # adding
the current song to the string
}
print(ntoken(NWADiscoString))

OutkastDiscoString = ""
for (i in seq_len(nrow(Sample10Outkast))){
  RowSongLyrics = Sample10Outkast[i, 5] #gets first song lyrics
  OutkastDiscoString = paste(OutkastDiscoString, RowSongLyrics, sep = " ", collapse=NULL) #
adding the current song to the string
}
print(ntoken(OutkastDiscoString))

CunningDiscoString = ""
for (i in seq_len(nrow(Sample10Cun))){
  RowSongLyrics = Sample10Cun[i, 5] #gets first song lyrics
  CunningDiscoString = paste(CunningDiscoString, RowSongLyrics, sep = " ", collapse=NULL) #
adding the current song to the string
}
print(ntoken(CunningDiscoString))

TribeDiscoString = ""
for (i in seq_len(nrow(Sample10Tribe))){
  RowSongLyrics = Sample10Tribe[i, 5] #gets first song lyrics
  TribeDiscoString = paste(TribeDiscoString, RowSongLyrics, sep = " ", collapse=NULL) # adding
the current song to the string
}
print(ntoken(TribeDiscoString))
```

```
GangDiscoString = ""
for (i in seq_len(nrow(Sample10Gang))){
  RowSongLyrics = Sample10Gang[i, 5] #gets first song lyrics
  GangDiscoString = paste(GangDiscoString, RowSongLyrics, sep = " ", collapse=NULL) # adding
the current song to the string
}
print(ntoken(GangDiscoString))

#Here I present the tests for all 21 possible comparisons between the 7 rap groups. However, I only
present the Wu-Tang Clan comparisons in Table 2.

WuMobbLevDistq2value = stringsim(WuTangDiscoString, MobbDeepDiscoString, q=2, method='lv')
WuMobbLevDistq2value
WuMobbCosDistq2value = stringsim(WuTangDiscoString, MobbDeepDiscoString, q=2,
method='cosine')
WuMobbCosDistq2value
stringsim(WuTangDiscoString, MobbDeepDiscoString, q=3, method='lv')
stringsim(WuTangDiscoString, MobbDeepDiscoString, q=3, method='cosine')
stringsim(WuTangDiscoString, NWADiscoString, q=2, method='lv')
stringsim(WuTangDiscoString, NWADiscoString, q=3, method='lv')
stringsim(WuTangDiscoString, NWADiscoString, q=2, method='cosine')
stringsim(WuTangDiscoString, NWADiscoString, q=3, method='cosine')
stringsim(WuTangDiscoString, OutkastDiscoString, q=2, method='lv')
stringsim(WuTangDiscoString, OutkastDiscoString, q=3, method='lv')
stringsim(WuTangDiscoString, OutkastDiscoString, q=2, method='cosine')
stringsim(WuTangDiscoString, OutkastDiscoString, q=3, method='cosine')
stringsim(WuTangDiscoString, CunningDiscoString, q=2, method='lv')
stringsim(WuTangDiscoString, CunningDiscoString, q=3, method='lv')
stringsim(WuTangDiscoString, CunningDiscoString, q=2, method='cosine')
stringsim(WuTangDiscoString, CunningDiscoString, q=3, method='cosine')
stringsim(WuTangDiscoString, TribeDiscoString, q=2, method='lv')
stringsim(WuTangDiscoString, TribeDiscoString, q=3, method='lv')
stringsim(WuTangDiscoString, TribeDiscoString, q=2, method='cosine')
stringsim(WuTangDiscoString, TribeDiscoString, q=3, method='cosine')
stringsim(WuTangDiscoString, GangDiscoString, q=2, method='lv')
stringsim(WuTangDiscoString, GangDiscoString, q=3, method='lv')
stringsim(WuTangDiscoString, GangDiscoString, q=2, method='cosine')
stringsim(WuTangDiscoString, GangDiscoString, q=3, method='cosine')
# From here on the results from these tests were not included in Table 2
stringsim(MobbDeepDiscoString, NWADiscoString, q=2, method='lv')
stringsim(MobbDeepDiscoString, NWADiscoString, q=3, method='lv')
stringsim(MobbDeepDiscoString, NWADiscoString, q=2, method='cosine')
stringsim(MobbDeepDiscoString, NWADiscoString, q=3, method='cosine')
stringsim(MobbDeepDiscoString, OutkastDiscoString, q=2, method='lv')
stringsim(MobbDeepDiscoString, OutkastDiscoString, q=3, method='lv')
stringsim(MobbDeepDiscoString, OutkastDiscoString, q=2, method='cosine')
stringsim(MobbDeepDiscoString, OutkastDiscoString, q=3, method='cosine')
stringsim(MobbDeepDiscoString, CunningDiscoString, q=2, method='lv')
stringsim(MobbDeepDiscoString, CunningDiscoString, q=3, method='lv')
stringsim(MobbDeepDiscoString, CunningDiscoString, q=2, method='cosine')
stringsim(MobbDeepDiscoString, CunningDiscoString, q=3, method='cosine')
stringsim(MobbDeepDiscoString, TribeDiscoString, q=2, method='lv')
stringsim(MobbDeepDiscoString, TribeDiscoString, q=3, method='lv')
stringsim(MobbDeepDiscoString, TribeDiscoString, q=2, method='cosine')
stringsim(MobbDeepDiscoString, TribeDiscoString, q=3, method='cosine')
```

```
stringsim(MobbDeepDiscoString, GangDiscoString, q=2, method='lv')
stringsim(MobbDeepDiscoString, GangDiscoString, q=3, method='lv')
stringsim(MobbDeepDiscoString, GangDiscoString, q=2, method='cosine')
stringsim(MobbDeepDiscoString, GangDiscoString, q=3, method='cosine')
stringsim(NWADiscoString, OutkastDiscoString, q=2, method='lv')
stringsim(NWADiscoString, OutkastDiscoString, q=3, method='lv')
stringsim(NWADiscoString, OutkastDiscoString, q=2, method='cosine')
stringsim(NWADiscoString, OutkastDiscoString, q=3, method='cosine')
stringsim(NWADiscoString, CunningDiscoString, q=2, method='lv')
stringsim(NWADiscoString, CunningDiscoString, q=3, method='lv')
stringsim(NWADiscoString, CunningDiscoString, q=2, method='cosine')
stringsim(NWADiscoString, CunningDiscoString, q=3, method='cosine')
stringsim(NWADiscoString, TribeDiscoString, q=2, method='lv')
stringsim(NWADiscoString, TribeDiscoString, q=3, method='lv')
stringsim(NWADiscoString, TribeDiscoString, q=2, method='cosine')
stringsim(NWADiscoString, TribeDiscoString, q=3, method='cosine')
stringsim(NWADiscoString, GangDiscoString, q=2, method='lv')
stringsim(NWADiscoString, GangDiscoString, q=3, method='lv')
stringsim(NWADiscoString, GangDiscoString, q=2, method='cosine')
stringsim(NWADiscoString, GangDiscoString, q=3, method='cosine')
stringsim(OutkastDiscoString, CunningDiscoString, q=2, method='lv')
stringsim(OutkastDiscoString, CunningDiscoString, q=3, method='lv')
stringsim(OutkastDiscoString, CunningDiscoString, q=2, method='cosine')
stringsim(OutkastDiscoString, CunningDiscoString, q=3, method='cosine')
stringsim(OutkastDiscoString, TribeDiscoString, q=2, method='lv')
stringsim(OutkastDiscoString, TribeDiscoString, q=3, method='lv')
stringsim(OutkastDiscoString, TribeDiscoString, q=2, method='cosine')
stringsim(OutkastDiscoString, TribeDiscoString, q=3, method='cosine')
stringsim(OutkastDiscoString, GangDiscoString, q=2, method='lv')
stringsim(OutkastDiscoString, GangDiscoString, q=3, method='lv')
stringsim(OutkastDiscoString, GangDiscoString, q=2, method='cosine')
stringsim(OutkastDiscoString, GangDiscoString, q=3, method='cosine')
stringsim(CunningDiscoString, TribeDiscoString, q=2, method='lv')
stringsim(CunningDiscoString, TribeDiscoString, q=3, method='lv')
stringsim(CunningDiscoString, TribeDiscoString, q=2, method='cosine')
stringsim(CunningDiscoString, TribeDiscoString, q=3, method='cosine')
stringsim(CunningDiscoString, GangDiscoString, q=2, method='lv')
stringsim(CunningDiscoString, GangDiscoString, q=3, method='lv')
stringsim(CunningDiscoString, GangDiscoString, q=2, method='cosine')
stringsim(CunningDiscoString, GangDiscoString, q=3, method='cosine')
stringsim(TribeDiscoString, GangDiscoString, q=2, method='lv')
stringsim(TribeDiscoString, GangDiscoString, q=3, method='lv')
stringsim(TribeDiscoString, GangDiscoString, q=2, method='cosine')
stringsim(TribeDiscoString, GangDiscoString, q=3, method='cosine')
```

Question 3

```
LineDupRemove = read.csv("LineDupRemove.csv") # the file with my web scraped lyrics

LineDupRemove$Lyrics = as.character(LineDupRemove$Lyrics) #making the lyrics column into a character column instead of factor as it currently is

View(LineDupRemove)

LineDupRemove$Sentiment = syuzhet::get_sentiment(LineDupRemove$Lyrics)
```

```
# if sentiment score was below zero, make value “negative”, if zero, make it “neutral” and if positive, make it “positive”

LineDupRemove$SentimentCategory[LineDupRemove$Sentiment == 0 ] = "Neutral"

LineDupRemove$SentimentCategory[LineDupRemove$Sentiment < 0 ] = "Negative"

LineDupRemove$SentimentCategory[LineDupRemove$Sentiment > 0 ] = "Positive"

head(LineDupRemove)

# From having seen this https://www.tidytextmining.com/sentiment.html

#I got the idea to generate a positive and negative word list. I will see how the percentage of words per line of text in LineDupRemove are in each word list and use these as features.

library(tidytext)

# Getting the positive and negative words from AFINN from Finn Årup Nielsen.

AFinnTibble = get_sentiments("afinn")

AFinnTibble

AFinnDataFrame = as.data.frame(AFinnTibble)

AFinnDataFrame$value = as.numeric(AFinnDataFrame$value)

head(AFinnDataFrame)

PositiveWords = AFinnDataFrame[which(AFinnDataFrame$value > 0), ]

NegativeWords = AFinnDataFrame[which(AFinnDataFrame$value < 0), ]

nrow(PositiveWords)

nrow(NegativeWords)

PositiveWords = PositiveWords[-c(2)] # getting just the word column

View(PositiveWords)

NegativeWords = NegativeWords[-c(2)] # getting just the word column

#Bing word list (from Bing et al https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html )

BingTibble = get_sentiments("bing")

BingTibble

BingDF = as.data.frame(BingTibble)

BingPositiveWords = BingDF[which(BingDF$sentiment == "positive"), ]

nrow(BingPositiveWords)

BingNegWords = BingDF[which(BingDF$sentiment == "negative"), ]

nrow(BingNegWords)

#Adding the words which are not already in the lists generated from AFinn to these lists

nrow(PositiveWords)
```

```
for (i in seq_len(nrow(BingPositiveWords))) {
   targetWord = BingPositiveWords[i, 1]
   if (targetWord %notin% PositiveWords$word) {
      RowToAdd = data.frame(word=targetWord)
      PositiveWords = rbind(PositiveWords, RowToAdd)
    }
}
nrow(PositiveWords) # this shows that there were some words already in PositiveWords so it was good to check if they were already there
nrow(NegativeWords)
for (i in seq_len(nrow(BingNegWords))) {
  targetWord = BingNegWords[i, 1]
   if (targetWord %notin% NegativeWords$word) {
      RowToAdd = data.frame(word=targetWord)
        NegativeWords = rbind(NegativeWords, RowToAdd)
     }
  }
nrow(NegativeWords) # this shows that there were some words already in NegativeWords so it was good to check if they were already there
#Feature Extraction
#% of tokens in text that are in PositiveWords
LineDupRemove$PercentPositiveWords <- sapply(
              strsplit(LineDupRemove$Lyrics, " "), function(x)
              (sum(x %in% PositiveWords$word)/length(x) * 100))
head(LineDupRemove)
#% of tokens in text that are in NegativeWords
LineDupRemove$PercentNegativeWords <- sapply(
              strsplit(LineDupRemove$Lyrics, " "), function(x)
              (sum(x %in% NegativeWords$word)/length(x) * 100))
head(LineDupRemove)
# using the spacy pos count function from Dr Kleinberg's github (https://github.com/ben-aaron188/r_helper_functions)
source('C:\\Users\\XXXXX\\OneDrive\\Desktop\\XXXXX\\Education\\UCL\\Year 3\\Term 2\\Data Science for Crime Scientists\\Project\\r_helper_functions-master\\spacy_pos_r.R')
```

```
#Number of tokens that are nouns
LineDupRemove$NounCount = get_pos_count(df_identifier = LineDupRemove$z2
                        , df_textcol = LineDupRemove$Lyrics
                        , unique_extr = F
                        , pos_type = 'NOUN'
                        , verbose = F)
head(LineDupRemove)
#Number of tokens that are verbs
LineDupRemove$VerbCount = get_pos_count(df_identifier = LineDupRemove$z2
                        , df_textcol = LineDupRemove$Lyrics
                        , unique_extr = F
                        , pos_type = 'VERB'
                        , verbose = F)
head(LineDupRemove)
#Number of tokens that are adjectives
LineDupRemove$AdjCount = get_pos_count(df_identifier = LineDupRemove$z2
                        , df_textcol = LineDupRemove$Lyrics
                        , unique_extr = F
                        , pos_type = 'ADJ'
                        , verbose = F)
head(LineDupRemove)
#Number of tokens that are adverbs
LineDupRemove$AdverbCount = get_pos_count(df_identifier = LineDupRemove$z2
                        , df_textcol = LineDupRemove$Lyrics
                        , unique_extr = F
                        , pos_type = 'ADV'
                        , verbose = F)
head(LineDupRemove)
#Sparsity corrected Uni,bi and trigrams
#Make a corpus object of the lyrics column
LineDupRemove$Lyrics = as.character(LineDupRemove$Lyrics)
LyricssCorpus = corpus(LineDupRemove,
```

```
                docid_field = "z2",
                text_field = "Lyrics",
                meta = list())
#Document feature matrix
#with stemming, punctuation, stopwords removed
LyricsCorpusDFM = dfm(x = LyricssCorpus$documents$texts, ngrams = 1:3, tolower = T, verbose = F, remove_punct = T, stem = T, remove = stopwords("english"))
#sparsity correction
Spars95_correct_lyrics = dfm_trim(LyricsCorpusDFM, sparsity = 0.95)
Spars99_correct_lyrics = dfm_trim(LyricsCorpusDFM, sparsity = 0.99)
dim(LyricsCorpusDFM)
dim(Spars95_correct_lyrics)
dim(Spars99_correct_lyrics)
tfidf_spars99_lyrics = dfm_tfidf(Spars99_correct_lyrics, scheme_df = 'inverse', scheme_tf = 'prop', k=1)
tfidf_spars95_lyrics = dfm_tfidf(Spars95_correct_lyrics, scheme_df = 'inverse', scheme_tf = 'prop', k=1)
topfeatures(tfidf_spars99_lyrics)
topfeatures(tfidf_spars95_lyrics)
View(tfidf_spars99_lyrics)
# 95 correction gave only 2 features, whilst 99 correction gave more so it was decided to use the tfidf features from 99 sparsity correction
tfidf99lyricsDF = convert(tfidf_spars99_lyrics, to = "data.frame") # converting to dataframe
View(tfidf99lyricsDF)
#dropping the document column in tfidf99lyricsDF
dropDocumentColumn = c("document")
tfidf99lyricsDFv2 = tfidf99lyricsDF[ , !(names(tfidf99lyricsDF) %in% dropDocumentColumn)]
View(tfidf99lyricsDFv2)
LineDupRemoveQuestion3 = LineDupRemove[, 7:ncol(LineDupRemove)]
head(LineDupRemoveQuestion3)
#adding the extracted sparsity corrected n gram features to other dataframe with other extracted features
LineDupRemoveQuestion3v2 = cbind(LineDupRemoveQuestion3, tfidf99lyricsDFv2)
dim(LineDupRemoveQuestion3v2)
```

```
set.seed(123)

#creating the training and testing dataset

in_training_Question3 = createDataPartition(y = LineDupRemoveQuestion3v2$SentimentCategory
                        , p = .7
                        , list = FALSE
                        )


training_data_Question3 = LineDupRemoveQuestion3v2[ in_training_Question3,]

testing_data_Question3 = LineDupRemoveQuestion3v2[-in_training_Question3,]

#training control parameters

training_controls_Question3 = trainControl(method="cv"

                    , number = 10

                    , classProbs = T)

# `preProcess = c('zv')` removes zero-variance

#training the SVM model

model.svm.question3 = train(SentimentCategory ~ .

                , data = training_data_Question3

                , method = "svmLinear"

                , trControl = training_controls_Question3

                , preProcess = c('zv')

                )

Question3_SVM_pred = predict(model.svm.question3, testing_data_Question3) # making predictions
on the testing data with the trained model

Question3.confusion.matrix = table("Actual" = testing_data_Question3$SentimentCategory,
"Predictions" = Question3_SVM_pred )

Question3.confusion.matrix

#Performance Metrics

#I followed this guide
https://blog.revolutionanalytics.com/2016/03/com_class_eval_metrics_r.html#kappa

nQ3 = sum(Question3.confusion.matrix) # number of instances

print(nQ3)

ncolQ3 = nrow(Question3.confusion.matrix) # number of classes

print(ncolQ3)

DiagonalQ3 = diag(Question3.confusion.matrix) # Number of correctly classified classes

print(DiagonalQ3)
```

```
RowSumsQ3 = apply(Question3.confusion.matrix, 1, sum)# number of instances per class
print(RowSumsQ3)
ColumnSumsQ3 = apply(Question3.confusion.matrix, 2, sum) # number of predictions per class
print(ColumnSumsQ3)
pQ3 = RowSumsQ3/nQ3 # distriubtion of instances over the actual classes
print(pQ3)
qQ3 = ColumnSumsQ3/nQ3 # distribution of instances over the predicted classes
print(qQ3)
# precision, recall and f1 for each of the three sentiment classes
precisionbyclassQ3 = DiagonalQ3/ColumnSumsQ3
RecallbyclassQ3 = DiagonalQ3/RowSumsQ3
F1ByClassQ3 = 2*(precisionbyclassQ3*RecallbyclassQ3/(precisionbyclassQ3+RecallbyclassQ3))
PRFDFQ3 = data.frame(precisionbyclassQ3, RecallbyclassQ3, F1ByClassQ3)
PRFDFQ3
#Macro-averaged metrics (mean across the three classes)
macroPrecisionQ3 = mean(precisionbyclassQ3)
macroRecallQ3 = mean(RecallbyclassQ3)
macroF1Q3 = mean(F1ByClassQ3)
data.frame(macroPrecisionQ3, macroRecallQ3, macroF1Q3)
#Micro-averaged Metrics
#constructing one vs all confusion matrices for each of the three sentiment classes
oneVsAllQ3 = lapply(1 : ncolQ3,
                    function(i){
                      v = c(Question3.confusion.matrix[i,i],
                          RowSumsQ3[i] - Question3.confusion.matrix[i,i],
                          ColumnSumsQ3[i] - Question3.confusion.matrix[i,i],
                          nQ3-RowSumsQ3[i] - ColumnSumsQ3[i] + Question3.confusion.matrix[i,i]);
                      return(matrix(v, nrow = 2, byrow = T))})
s = matrix(0, nrow = 2, ncol = 2)
for(i in 1 : ncolQ3){s = s + oneVsAllQ3[[i]]}
s
#Micro precision = recall = f1
```

```
Micro.prf = (diag(s) / apply(s,1, sum))[1]

Micro.prf

#Average Accuracy (fraction of correctly classified instances in the sum of one-vs-all matrix)

avgAccuracyQ3 = sum(diag(s)) / sum(s)

avgAccuracyQ3

#Weighted Average metrics

#Formulas will be as from https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1

#((number of instances in class 1 (support) x metric) + (number of instances in class 2 (support) x metric) +(number of instances in class 3 (support) x metric))/ number of total instances

#Positive had 3496
#Neutral had 8790
#Negative had 6025
#So total instances were 18311

nQ3

WeightedPrecQ3 = ((6025*PRFDFQ3[1,1])+(8790*PRFDFQ3[2,1])+(3496*PRFDFQ3[3,1]))/nQ3
WeightedPrecQ3
WeightedRecallQ3 = ((6025*PRFDFQ3[1,2])+(8790*PRFDFQ3[2,2])+(3496*PRFDFQ3[3,2]))/nQ3
WeightedRecallQ3
WeightedF1Q3 = ((6025*PRFDFQ3[1,3])+(8790*PRFDFQ3[2,3])+(3496*PRFDFQ3[3,3]))/nQ3
WeightedF1Q3

#Kappa statistic

AccuracyQ3 = sum(DiagonalQ3)/nQ3

AccuracyQ3 # THIS IS NOT THE SAME AS THE AVERAGE ONE VS ALL ACCURACY

ExpectedAccuracyQ3 = sum(pQ3*qQ3)

kappaQ3 = (AccuracyQ3 - ExpectedAccuracyQ3)/ (1 - ExpectedAccuracyQ3)

kappaQ3

#Multiclass summary function from Zach Meyer

#This provides macro average performance metrics, but I primarily wanted to get the specificity and sensitivity as I had already calculated other metrics above.

#https://gist.github.com/zachmayer/3061272#file-multiclass-r

#https://www.r-bloggers.com/error-metrics-for-multi-class-problems-in-r-beyond-accuracy-and-kappa/

source('C:\\Users\\XXXXX\\OneDrive\\Desktop\\XXXXX\\Education\\UCL\\Year 3\\Term 2\\Data Science for Crime Scientists\\Project\\3061272-024ff7415a015353268d40024ded7cc7645d340d\\multiclass.R')

library(compiler)
```

```
library(pROC)

#probability of predictions

probsQ3 = predict(model.svm.question3, testing_data_Question3, type = 'prob')

dataforQ3SummaryFunction = data.frame(obs = testing_data_Question3$SentimentCategory, pred = Question3_SVM_pred)

dataforQ3SummaryFunction = cbind(dataforQ3SummaryFunction, probsQ3)

View(dataforQ3SummaryFunction)

levelsQ3 = c("Negative", "Neutral", "Positive")

multiClassSummary(dataforQ3SummaryFunction, lev=levelsQ3)

#Multiclass AUC

#The R documentation for the multiclass.roc function below

#https://www.rdocumentation.org/packages/pROC/versions/1.16.1/topics/multiclass.roc

multiclass.roc(dataforQ3SummaryFunction$obs, probsQ3)
```