Final Project : Text Mining on Rock Sub-genres

Group 08 : Vittoria Zagarese, Romeo Silvestri, Marco Cavallo, Costanza Addari

Abstract

We are interested in analyzing the differences between lyrics in the following Rock genres: Folk, Glam, Grunge, Punk, Alternative, Progressive. Our work is based on the following 4 sections:

- Pre-Processing
- Exploratory Analysis & Sentiment Analysis
- Topic Modeling with LDA
- Predictive Models: Naive Bayes, Random Forest, SVM

Our initial focus is to analyze the differences and similarities between lyrics in the considered sub-categories, both qualitatively and quantitatively, by considering text mining techniques that focus on the positive and negative connotations of the words from the lyrics. Our primary goal in this project is to build models that predict the sub-genre based on the previously analyzed text and evaluate their predictive performance in terms of accuracy.

Dataset Construction

Our choice regarding the sub-genres was based on trying to take categories of music that differ from each other.
To create the database we initially researched and created a list of Artists/Bands, exclusively from the USA or UK, that represent the 6 sub-genres we selected; this list was selected from specialized Rock music web-sites. Once we obtained these lists we then selected a random sample, allowing for repeated artists, and for every extraction we then randomly selected a song (using random selection from Spotify). Before inserting a song in the database we verified if the selected song took part of an album that was classified in the correct sub-genre, we focused on this because an artist that belongs to a specific sub-genre in our list may produce albums that differ in terms of genre. Once the song was selected we researched its lyrics and several other variables to create the final database. The variables we included in the database are:

- Artist
- Title
- Lyrics
- Sub-Genre
- Band
- Time (s)

To ensure equal representation of each genre we selected 100 songs for each sub-category; the final dataset is a 600x6 matrix containing 600 songs.

Data upload

In the folowing lines of code we uploaded our dataset and implemented the needed libraries for the project. We added to the dataset a variable that counts the number of words in each song.

```
library(tm)
library(ggplot2)
library(tidytext)
library(dplyr)
library(SnowballC)
library(wordcloud)
library(gutenbergr)
library(readxl)
library(qdap)
library(wordcloud2)
library(stopwords)
library(reshape2)
library(syuzhet)
library(topicmodels)
library(textmineR)
library(caTools)
library(randomForest)
library(e1071)
library(gridExtra)
library(quanteda)


dataset=read_excel("dataset.xlsx")
n_words_f=function(x){
  result=x %>%
    tolower %>%
    stringr::str_extract_all('\\w+') %>%
    unlist() %>%
    length()
  return(result)
}

dataset$n_words=sapply(dataset$text,n_words_f)
dataset$band=factor(dataset$band)
levels(dataset$band)=c("solo","band")

head(dataset)
```

```
## # A tibble: 6 x 7
##   artist    title     text                              subgenre band   time n_words
##   <chr>     <chr>     <chr>                             <chr>    <fct> <dbl>   <int>
## 1 7 Year B~ you smel~ "So you wanna go to bed with~ grunge   band    124     220
## 2 Adorable  homeboy   "I'm tripping into the back ~ alterna~ band    270     235
## 3 Against ~ problems  "An inventory has been taken~ punk     band    160     145
## 4 Alice Co~ steven    "I don't want to see you go\~ glam     band    347     218
## 5 Alice Co~ poison    "Your cruel device\r\nYour b~ glam     band    277     318
## 6 Alice in~ angry ch~ "Sitting on an angry chair\r~ grunge   band    287     210
```

```
attach(dataset)
```

1. Pre-processing

Pre-processing is the preliminary phase of the analysis that aims to reduce the text to a quantitative format so that the data can be easily processed. There are two main phases in Pre-processing: the lexical analyzer and stemming. We will not use stemming until a later phase of the project to facilitate interpretation during the exploratory analysis and Sentiment analysis application. For the pre-processing phase of the analysis we decided to group the dataset based by subgenre; we manually built two functions to clean the dataset. The first one: clean() substitutes frequently contracted words to their full length considering different types of punctuation, we also replace the “new lines” and “carriage return” with a blank space so that the words don’t merge together. The second function, clean2(), is used to replace commonly used intra-word contractions with a blank space after the elimination of the stop words in the text. This is useful because when a significant word is linked to a useless word with a punctuation sign, this intra-word contraction attaches itself to the important word so we must further clean the text.

```
dataset_sub=dataset %>% group_by(subgenre)

clean = function(x){
  x = gsub("won't","will not",x)
  x = gsub("n't"," not",x)
  x = gsub("'ll"," will",x)
  x = gsub("'re"," are",x)
  x = gsub("'ve"," have",x)
  x = gsub("'m"," am",x)
  x = gsub("'d"," would",x)
  x = gsub("won’t","will not",x)
  x = gsub("n’t"," not",x)
  x = gsub("’ll"," will",x)
  x = gsub("’re"," are",x)
  x = gsub("’ve"," have",x)
  x = gsub("’m"," am",x)
  x = gsub("’d"," would",x)
  x = gsub("’s","",x)
  x = gsub("'s","",x)
  x = gsub('whatsa','what is',x)
  x = gsub('wanna','want to',x)
  x = gsub('gonna','going to',x)
  x = gsub('gotta','go to',x)
  x = gsub("\n"," ",x)
  x = gsub("\r"," ",x)
  x = gsub("- "," ",x)
  x = gsub(" -"," ",x)
  x = gsub(" –"," ",x)
  x = gsub("– "," ",x)
  x = gsub("_"," ",x)
  return(x)
}

clean2 = function(x){
  x = gsub("- "," ",x)
  x = gsub("– "," ",x)
  x = gsub(" -"," ",x)
  x = gsub(" –"," ",x)
  x = gsub("'"," ",x)
  x = gsub("’"," ",x)
}
# we must consider the difference between these signs ', ’ and -,– because they are read diff
erently

my_stopwords = c("yeah","hey","ooh") # most frequent useless words
```

Next, we implement the following lexical analyzer. First of all we apply the tolower() function to convert text to all lowercase letters, we then proceed to apply removeNumbers() followed by the removePunctuation() function; this function removes the punctuation and dashes in the text but preserves the intra-word signs because, in some cases, linked words lose their original meaning when separated. As shown in the following code we apply the cleaning functions defined earlier, we also apply the removeWords() which removes the stopwords from the English dictionary as well as the stopwords we defined manually. Lastly we remove the excess blank spaces.

```
dataset_sub$text=dataset_sub$text %>%
  tolower() %>%
  removeNumbers() %>%
  removePunctuation(preserve_intra_word_contractions=TRUE,preserve_intra_word_dashes=TRUE) %
>%
  clean() %>%
  removeWords(c(stopwords("en"),my_stopwords)) %>%
  clean2() %>%
  stripWhitespace()
```

Tidy Dataset

Now we convert the dataset to the Tidy format. Structuring text data in this way means that it conforms to tidy data principles and can be manipulated with a set of consistent tools. Tidy data has a specific structure:

- Each variable is a column
- Each observation is a row
- Each type of observational unit is a table

By applying unnest_tokens() we implement tokenization which is the process of splitting text into tokens; a token is a meaningful unit of text, such as a word, that we are interested in using for analysis. We thus define the tidy text format as being a table with one-token-per-row. To implement this transformation we use anti_join() to remove common adverbs that haven’t been eliminated previously by the stopwords vocabulary and filter out two-letter and one-letter words. We implemented this procedure twice: in df_tidy we consider every word only once per song while in df_tidy2 we also keep count of the repeated words.

```
df_tidy=dataset_sub %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  distinct() %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2) %>%
  select(artist,title,subgenre,word)
```

```
## Joining, by = "word"
```

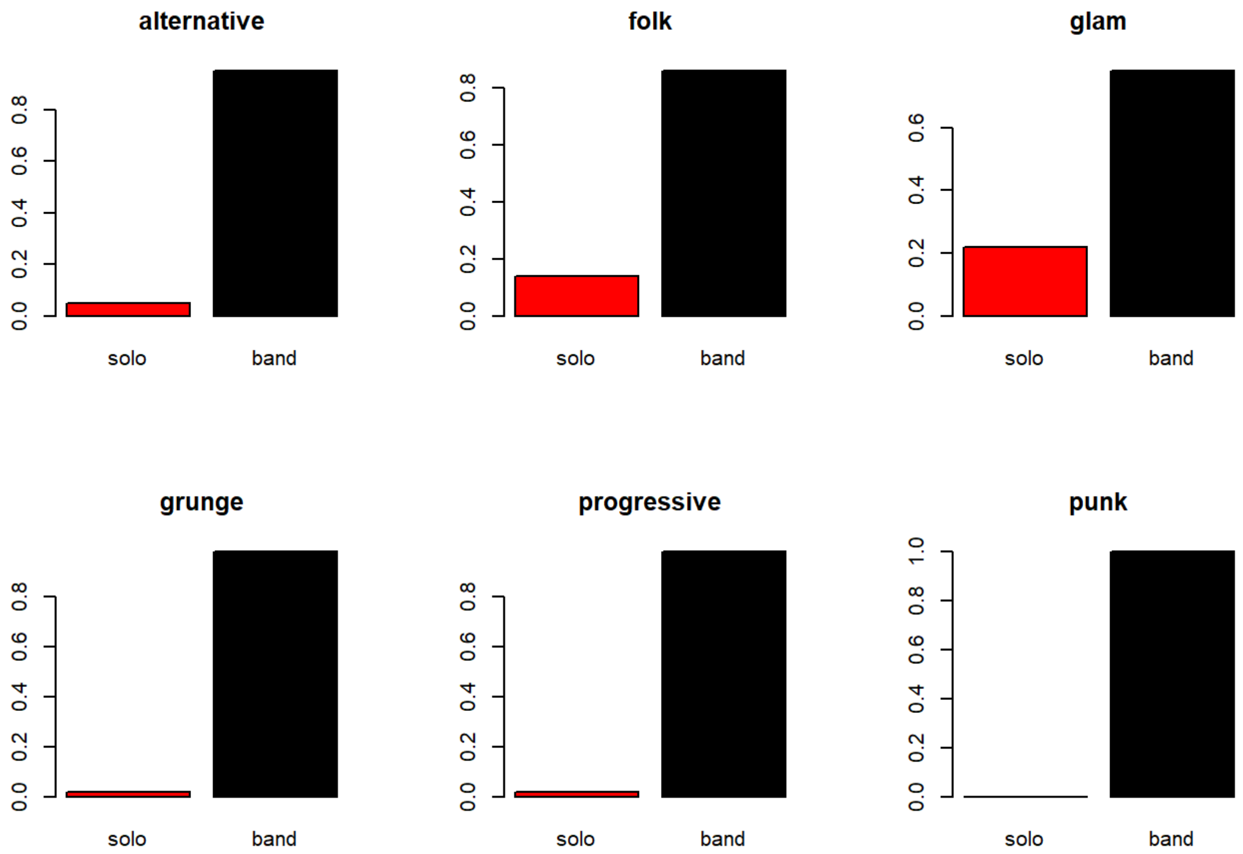
```
df_tidy2=dataset_sub %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2) %>%
  select(artist,title,subgenre,word)
```

```
## Joining, by = "word"
```

2.a) Exploratory Analysis

We begin by doing an exploratory analysis to observe the main quantitative characteristics of the data. If we consider the variable that takes into account the type of artist we can observe the following barplot. We can see that the predominant type of artists are bands rather than solo artists, the only subgenres with a percentage of solo artists higher than 10% are glam and folk.

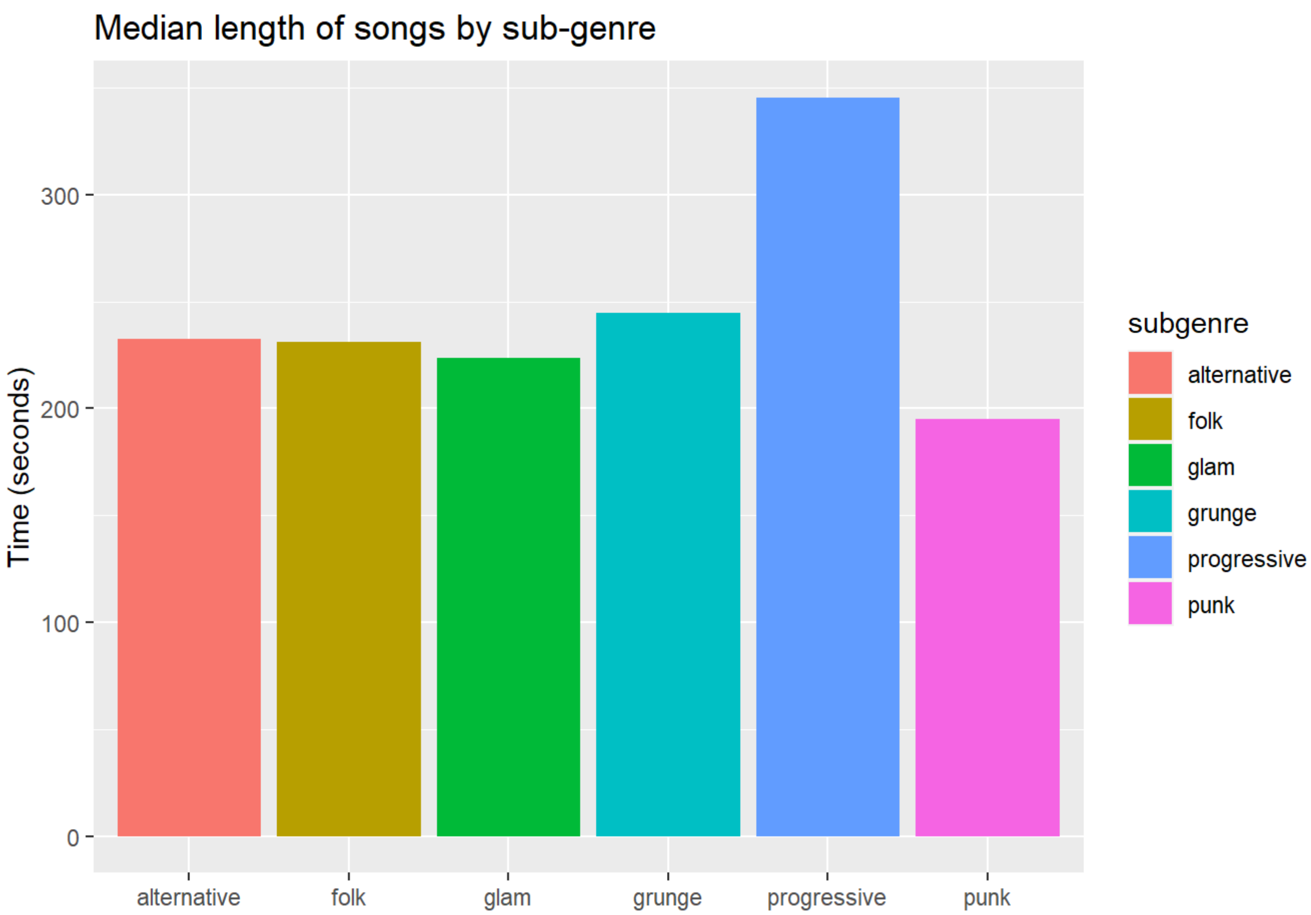
```
par(mfrow=c(2,3))
sub1=subset(dataset,subgenre=="alternative")
barplot(prop.table(table(sub1$band)),col=c("red","black"),main="alternative")
sub2=subset(dataset,subgenre=="folk")
barplot(prop.table(table(sub2$band)),col=c("red","black"),main="folk")
sub3=subset(dataset,subgenre=="glam")
barplot(prop.table(table(sub3$band)),col=c("red","black"),main="glam")
sub4=subset(dataset,subgenre=="grunge")
barplot(prop.table(table(sub4$band)),col=c("red","black"),main="grunge")
sub5=subset(dataset,subgenre=="progressive")
barplot(prop.table(table(sub5$band)),col=c("red","black"),main="progressive")
sub6=subset(dataset,subgenre=="punk")
barplot(prop.table(table(sub6$band)),col=c("red","black"),main="punk")
```



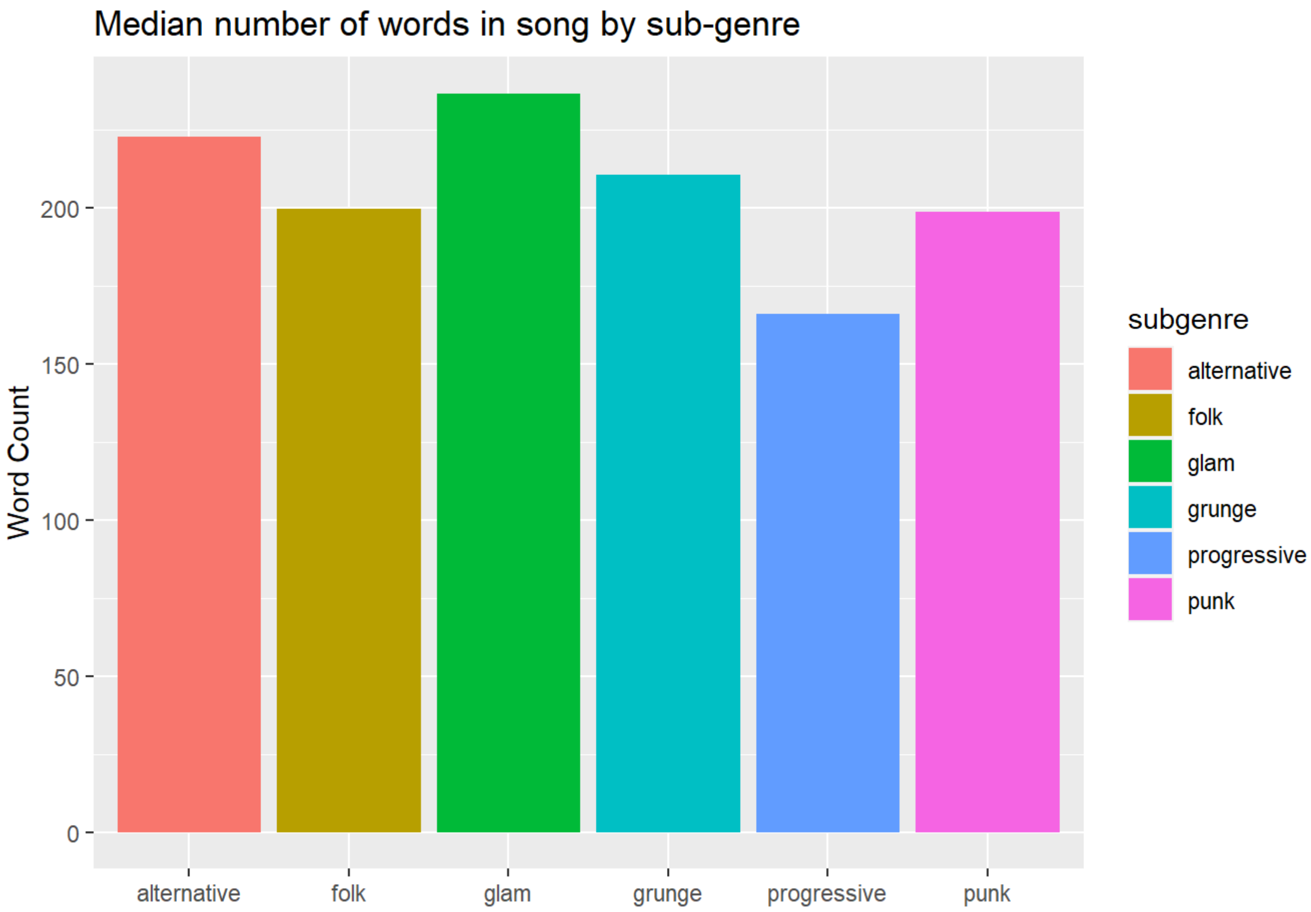

```
par(mfrow=c(1,1))
```

We are interested in evaluating the length of the considered songs from two different points of views: the number of words in each song and the number of seconds. As seen in the following plots we observe that, from a time standpoint, Progressive is the subgenre with the highest median length while, if we consider the number of words, it's in the last position. This highlights the intrinsic characteristics of the subgenre as long instrumental sections are comparatively more prevalent. As for the remaining subgenres, their median length is around 3 to 4 minutes, while their median word count is between 200 and 275.

```
dataset %>%
  group_by(subgenre) %>%
  summarise(median=median(time)) %>%
  ggplot(aes(x=subgenre,y=median,fill=subgenre)) +
  geom_col() +
  labs(title="Median length of songs by sub-genre",x='',y ="Time (seconds)")
```

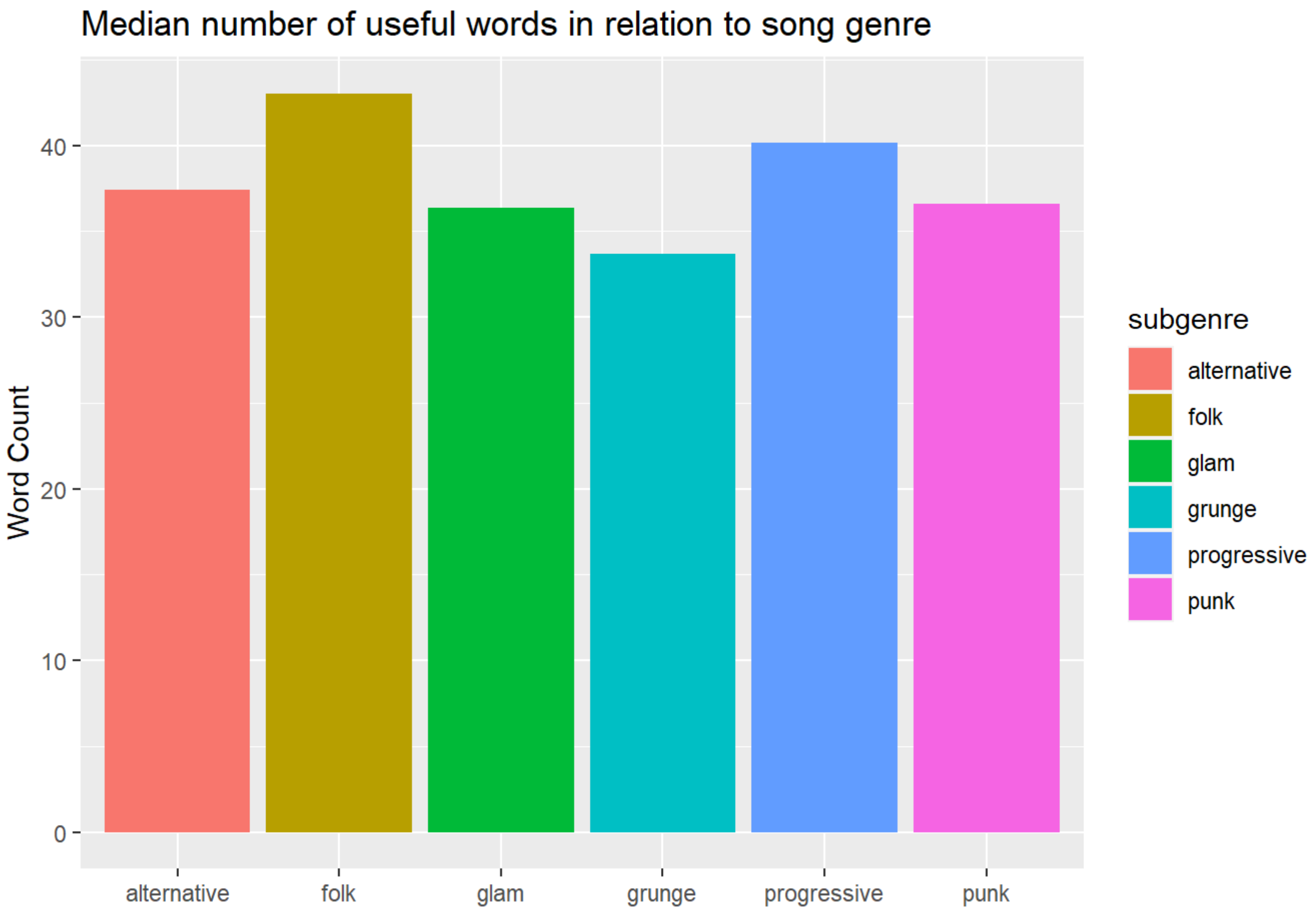



```
dataset %>%
  group_by(subgenre) %>%
  summarise(median=median(n_words)) %>%
  ggplot(aes(x=subgenre,y=median,fill=subgenre)) +
  geom_col() +
  labs(title="Median number of words in song by sub-genre",x='',y ="Word Count")
```



The following plot displays the median number of useful words in each song for every subgenre. In this case we can see that all subgenres have a similar median of useful words per song, about 35. In particular, even though progressive lyrics are the shortest in terms of number of words, they make up for it in terms of useful words.

```
df_tidy %>%
  group_by(subgenre) %>%
  count() %>%
  summarise(median=median(n)) %>%
  ggplot(aes(x=subgenre, y=median/100, fill=subgenre)) +
  geom_col() +
  labs(x='',y='Word Count',title='Median number of useful words in relation to song genre')
```

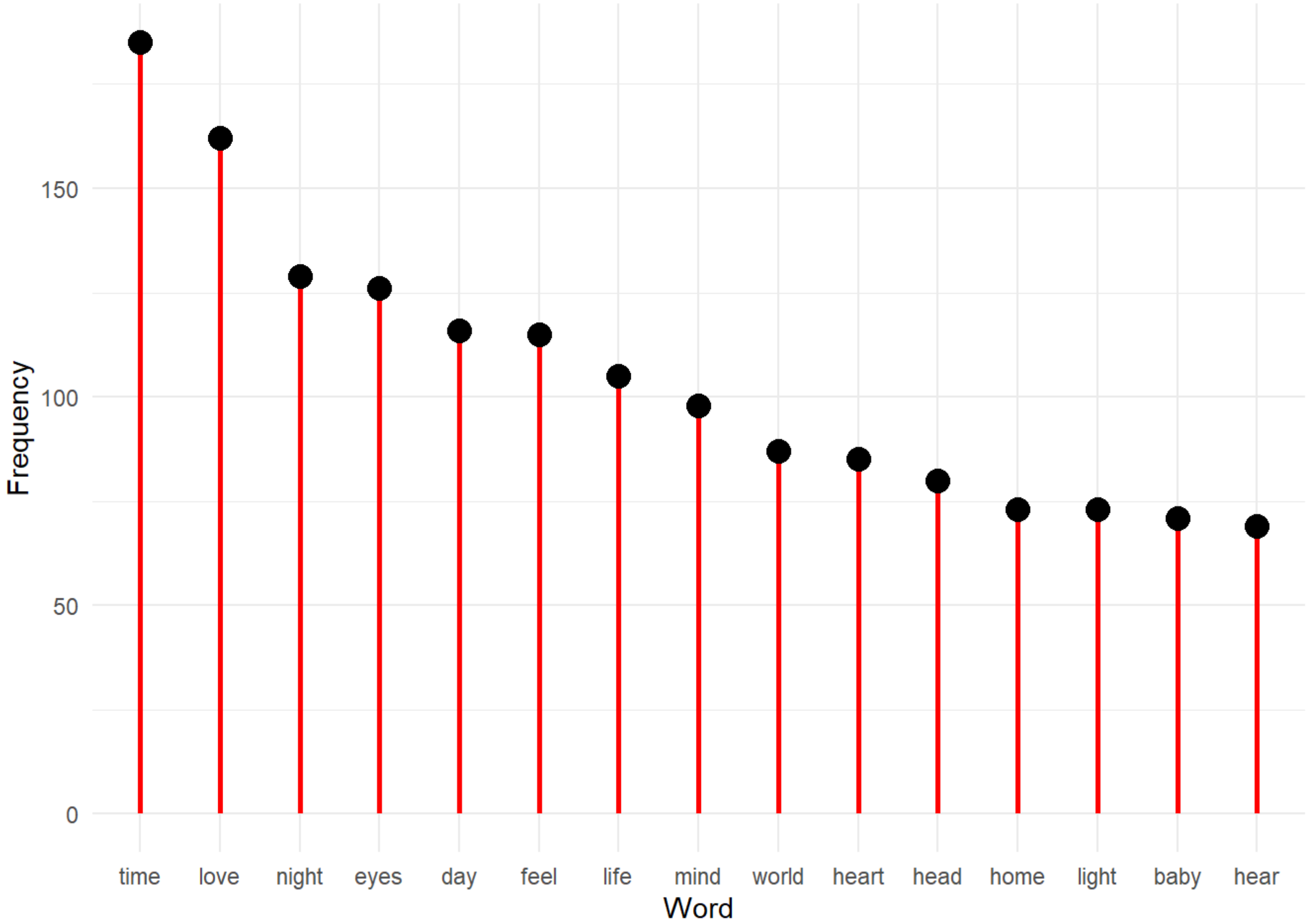


Now we are interested in observing the frequency of each word without distinguishing between subgenre. We begin by showing a Word Cloud which displays the most frequent words with larger font size; the colored words are the most used and common words.

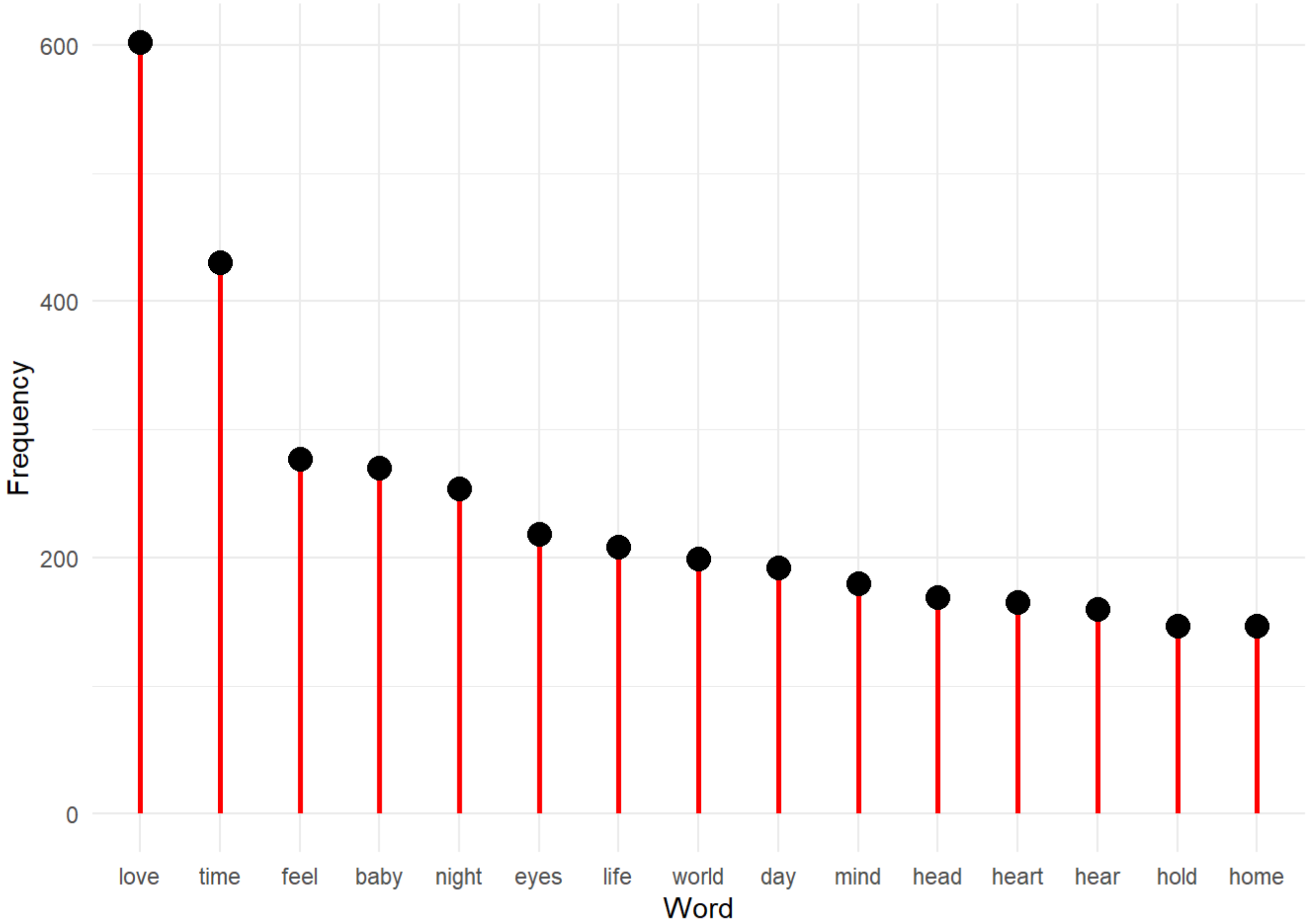
```
unigram=df_tidy %>%
  group_by(word) %>%
  count() %>%
  ungroup () %>%
  arrange(desc(n))

wordcloud2(data=unigram[1:100, ], size=1,color=brewer.pal(8,'Dark2'))
```

From the following plots we can notice the difference when we consider the two datasets: df_tidy and df_tidy2. We can initially see that the range of frequency of words drastically changes, furthermore some words change in relevance when considering the two datasets. For example the word “love” is the most frequent word when considering the dataset with the repeated words; when we consider each word once per song, it is overtaken by the word “time”. From now on we will consider each word once per song.

```
frequency_words=freq_terms(df_tidy$word,top=15)
frequency_words %>%
  ggplot(aes(x=WORD,y=FREQ)) +
  geom_segment(aes(x=reorder(WORD,desc(FREQ)),xend=WORD,y=0,yend=FREQ),color="red",size=1) +
  geom_point(size=4) +
  theme_minimal() +
  xlab("Word") +
  ylab("Frequency")
```




```
frequency_words2=freq_terms(df_tidy2$word,top=15)
frequency_words2 %>%
  ggplot(aes(x=WORD,y=FREQ)) +
  geom_segment(aes(x=reorder(WORD,desc(FREQ)),xend=WORD,y=0,yend=FREQ),color="red",size=1) +
  geom_point(size=4) +
  theme_minimal() +
  xlab("Word") +
  ylab("Frequency")
```



2.b) Sentiment Analysis

Now we shift our focus to the qualitative aspects of our analysis by looking at the sentiment in the lyrics. One way to analyze the sentiment of a text is to consider the text as a combination of its individual words and the sentiment content of the whole text as the sum of the sentiment content of the individual words. To implement this analysis we consider three different vocabularies: BING, AFINN, NRC. The BING lexicon categorizes words in a binary fashion into positive and negative categories, we can showcase this difference by assigning the color red to words with a positive connotation and black to the negative ones; again, the word size is positively correlated to their frequency.

```
df_tidy %>%
  inner_join(get_sentiments("bing")) %>%
  count(word,sentiment,sort=TRUE) %>%
  acast(word ~ sentiment,value.var="n",fill=0) %>%
  comparison.cloud(colors=c("black","red"),max.words=150)
```

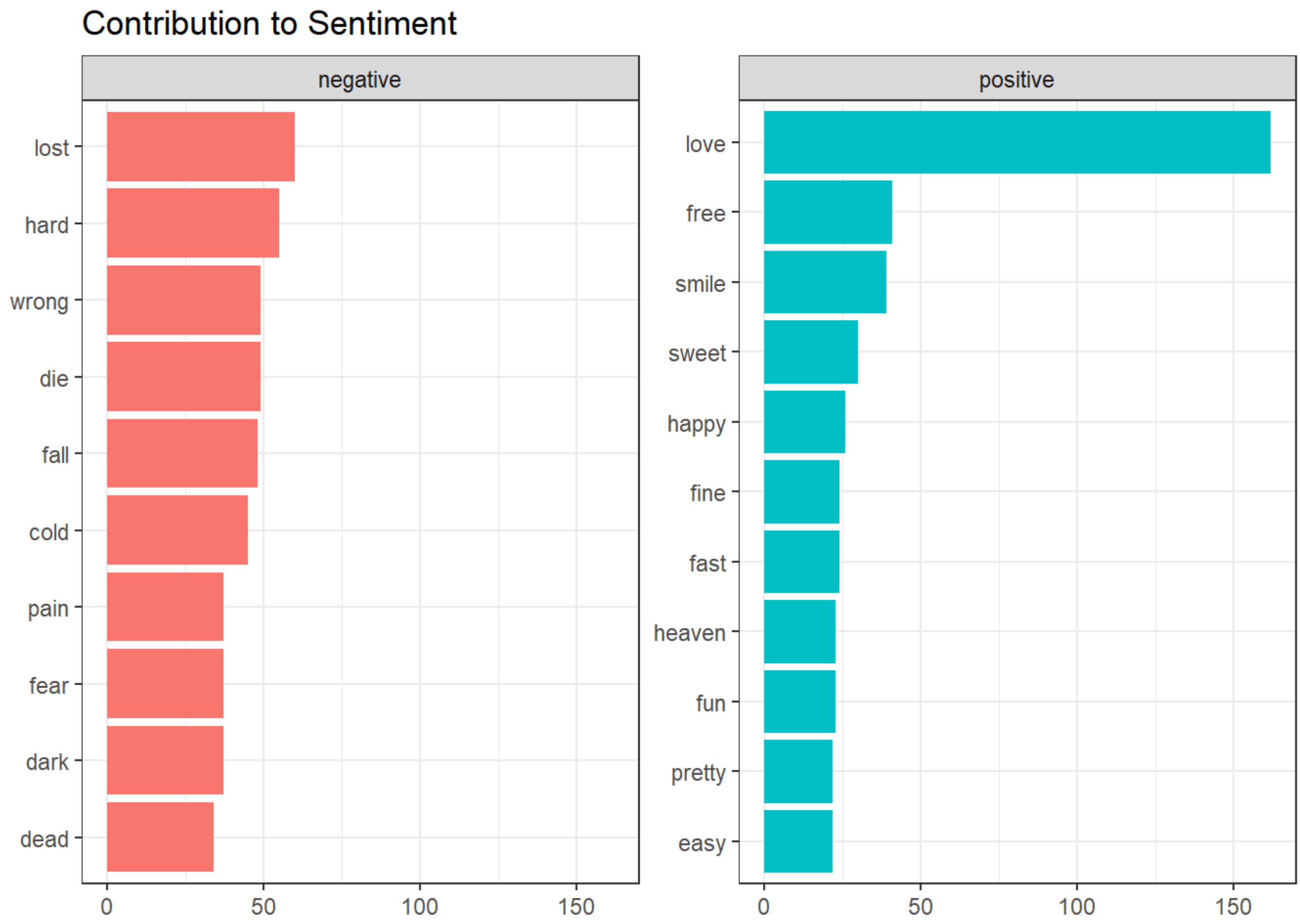
```
## Joining, by = "word"
```


```
sent_word=df_tidy %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()
```

```
## Joining, by = "word"
```

The following plot displays the main contributors to each sentiment in terms of their frequency within the text. We notice that love is by far the most prevalent positively-coded word, whereas the contributions of the negative sentiment are more balanced.

```
sent_word %>%
  group_by(sentiment) %>%
  slice_max(n,n=10) %>%
  ungroup() %>%
  mutate(word=reorder(word,n)) %>%
  ggplot(aes(n,word,fill=sentiment)) +
  geom_col(show.legend=FALSE) +
  facet_wrap(~sentiment,scales="free_y") +
  labs(x=NULL,y=NULL,title="Contribution to Sentiment") +
  theme_bw()
```

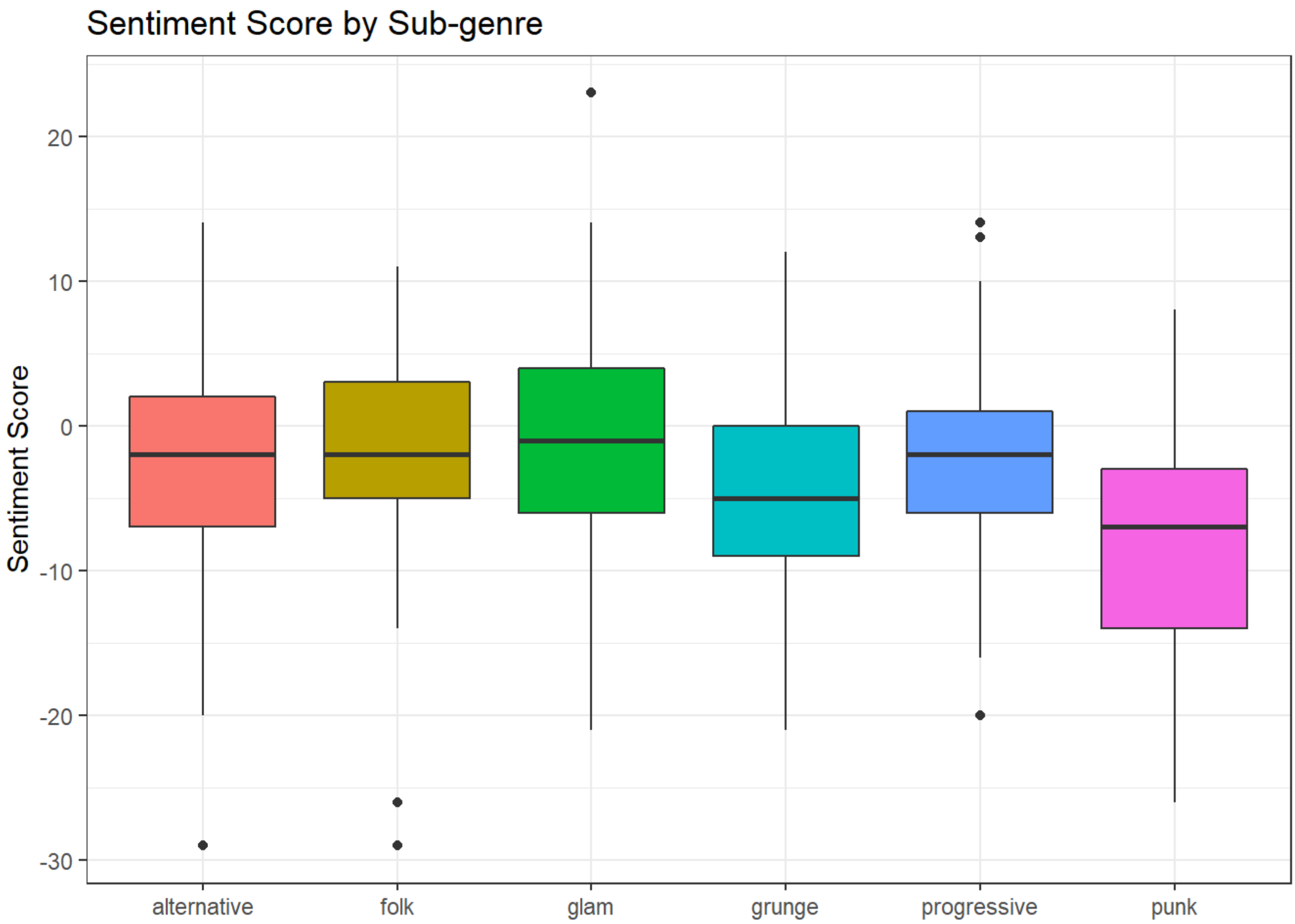


The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment. We are interested in comparing subgenres in terms of sentiment score. From the following boxplots we observe that all of the subgenres have a median score under 0, so we can say that Rock music tends to have a negative connotations to its lyrics particularly for Punk and Grunge music.

```
afinn=get_sentiments(lexicon='afinn')

genre_afinn=df_tidy %>%
  inner_join(afinn) %>%
  group_by(title) %>%
  mutate(total_score = sum(value)) %>%
  ungroup() %>%
  arrange(desc(value))
```

```
## Joining, by = "word"
```

```
genre_afinn %>%
  ggplot(aes(x=subgenre,y=total_score,fill=subgenre)) +
  geom_boxplot(show.legend=FALSE) +
  labs(x=NULL,y='Sentiment Score',title='Sentiment Score by Sub-genre') +
  theme_bw()
```

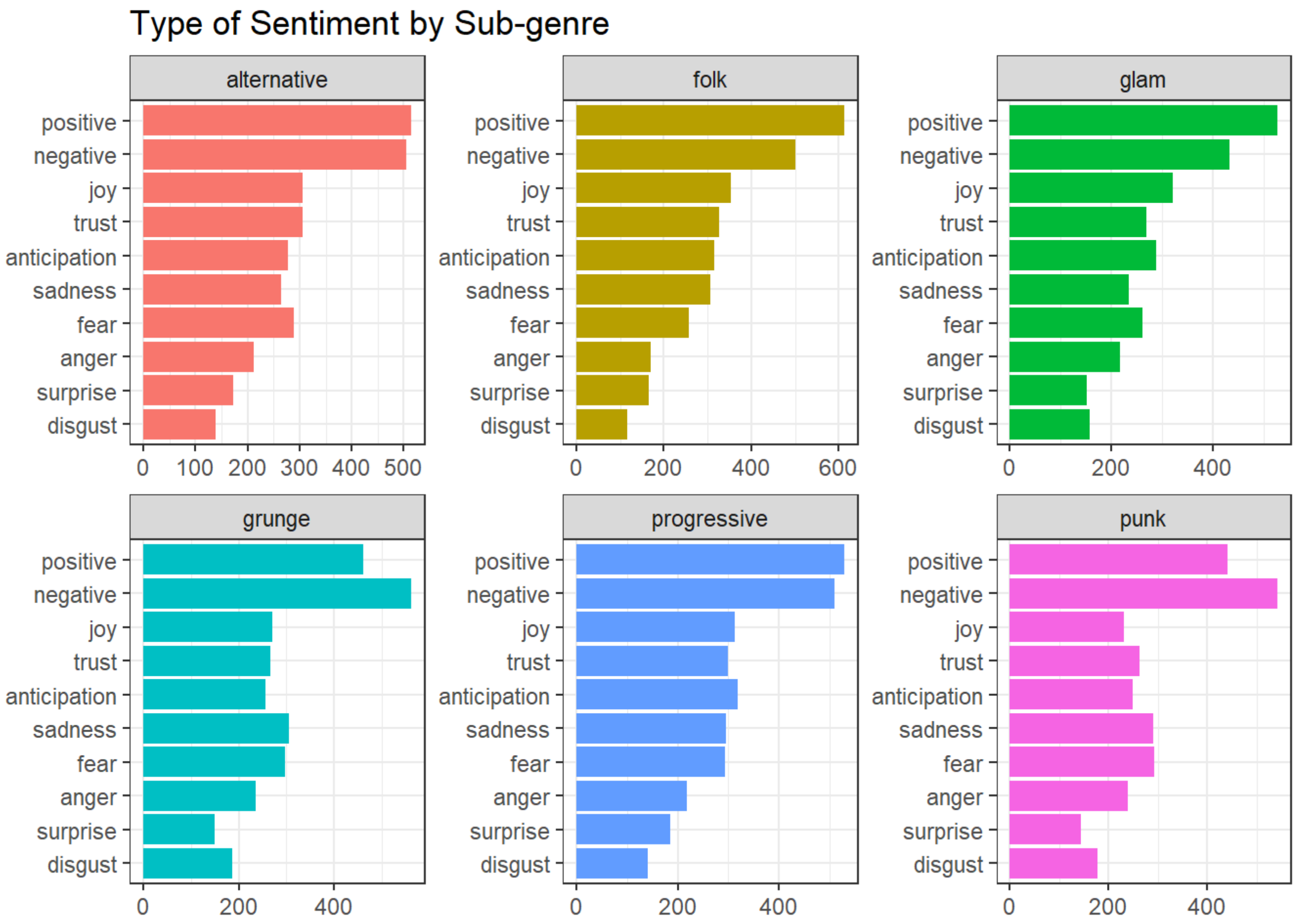


The NRC lexicon categorizes words into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. In the next plot we are interested in observing, for each subgenre, how relevant each sentiment is. We see from the following plots that the most "positive" subgenres are Glam and Folk in fact they have a higher prevalence of the sentiment "joy". Progressive and Alternative seem to be quite balanced while Punk and Grunge are significantly more negative since "sadness" and "fear" are quite relevant in their lyrics. We can also observe that "Anticipation" is a commonly used category in Progressive and Glam and this leads us to believe that time is a prevalent subject for these subgenres.

```
nrc=get_sentiments(lexicon='nrc')

genre_nrc=df_tidy %>%
  inner_join(nrc) %>%
  group_by(subgenre,sentiment) %>%
  count() %>%
  ungroup()
```

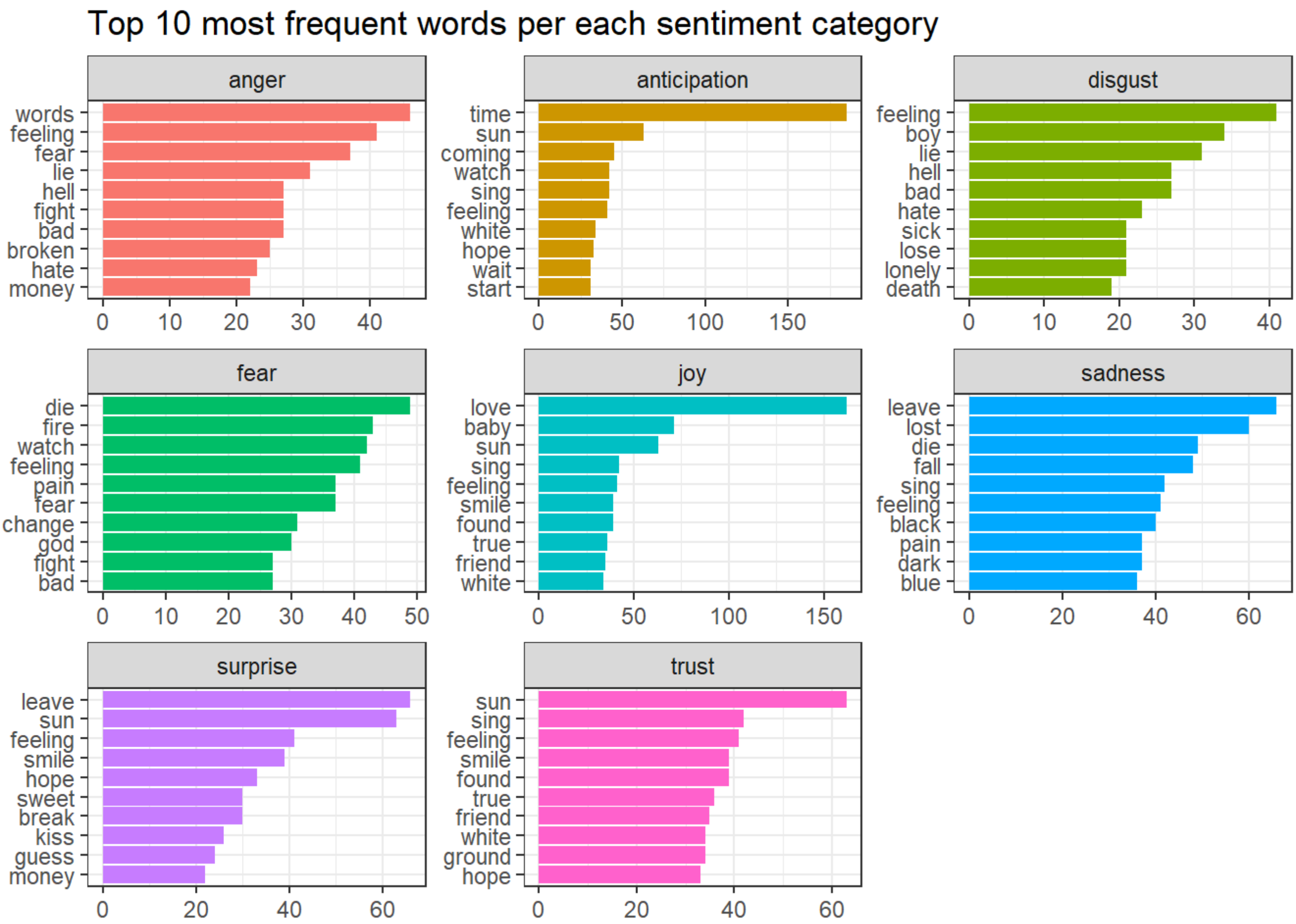
```
## Joining, by = "word"
```

```
ggplot(genre_nrc,aes(x=reorder(sentiment,n),y=n,fill=subgenre)) +
  geom_col(show.legend=FALSE) +
  facet_wrap(subgenre ~.,scales="free") +
  coord_flip() +
  labs(x=NULL,y=NULL,title='Type of Sentiment by Sub-genre') +
  theme_bw()
```



The following plot shows what words contribute mostly to describe a specific emotion. In the case of “joy” and “anticipation” the most frequent words are respectively “love” and “time”.

```
unigram %>%
  inner_join(nrc,by="word") %>%
  ungroup() %>%
  filter(!sentiment %in% c("positive","negative")) %>%
  arrange(desc(n)) %>%
  group_by(sentiment) %>%
  slice(1:10) %>%
  ggplot(aes(x=reorder(word,n),y=n,fill=sentiment)) +
  geom_col(show.legend=FALSE) +
  facet_wrap( ~ sentiment,scales="free") +
  coord_flip() +
  labs(x=NULL,y=NULL,title ='Top 10 most frequent words per each sentiment category') +
  theme_bw()
```



3. Topic Modeling with LDA

Latent Dirichlet Allocation (LDA) is a generative statistical model (an unsupervised method based on joint and non conditional probabilities). This method allows us to explain why and how parts of the observed data are similar by considering non observed data. This technique is frequently used in text mining to implement topic modeling, as the name suggests the aim of this method is to understand what are the most relevant topics within a group of documents. Each word is an observation and we assume that each document (song) is a mixture of a finite number of topics where each topic is mixture of words. The words contained in each topic and all the topics contained in the song follow the Dirichlet distribution. LDA generates several topics and for each topic it creates texts with sampled words from the original texts. After estimating the distribution of the words and topics it assigns each text and topic a specific probability. The most probable topics for each text and group of texts represent the most common topics. In our case we are interested in seeing what topics are the most common for each subgenre.

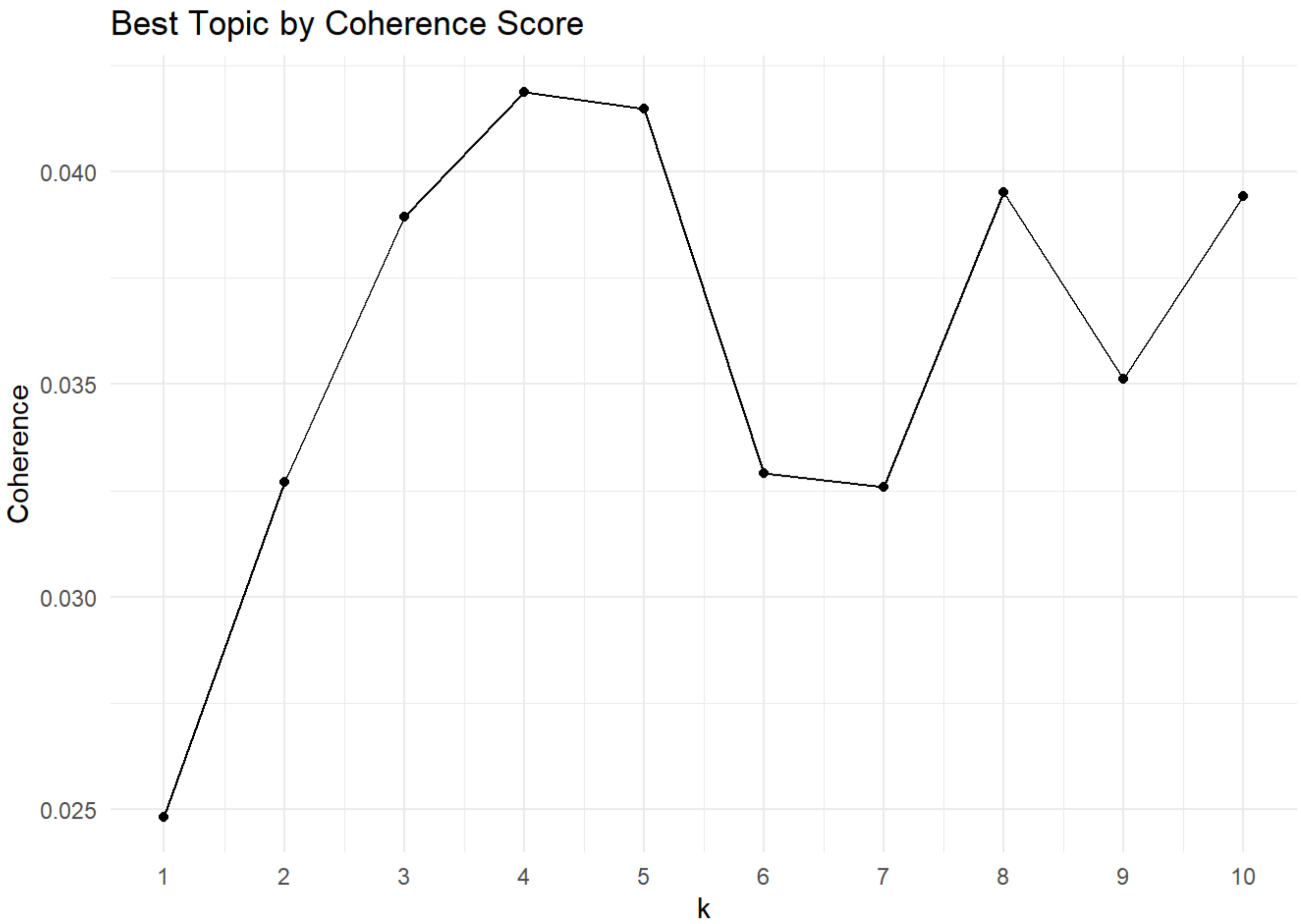
To apply LDA we first implement a cleaning procedure called Stemming; this is a method which identifies the root of words to generalize the operations of querying and selecting documents in an archive. In practice, all words in a text are replaced with their roots so that the final result is a version of the text with the same number of terms but with fewer variants. This operation is carried out because in this way the process of searching and matching information is more effective.

```
detach()
dataset2=read_excel("dataset.xlsx")
dataset2$n_words=sapply(dataset$text,n_words_f)

dataset2$text=dataset2$text %>% # cleaning with stemming
  tolower() %>%
  removeNumbers() %>%
  removePunctuation(preserve_intra_word_contractions=TRUE,preserve_intra_word_dashes=TRUE) %
>%
  clean() %>%
  removeWords(c(stopwords("en"),my_stopwords)) %>%
  stemDocument() %>%  # stemming
  clean2() %>%
  stripWhitespace()

dataset_sel=dataset2 %>% select(title,text)

df_tidy3=dataset_sel %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  distinct() %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2)
```

```
## Joining, by = "word"
```

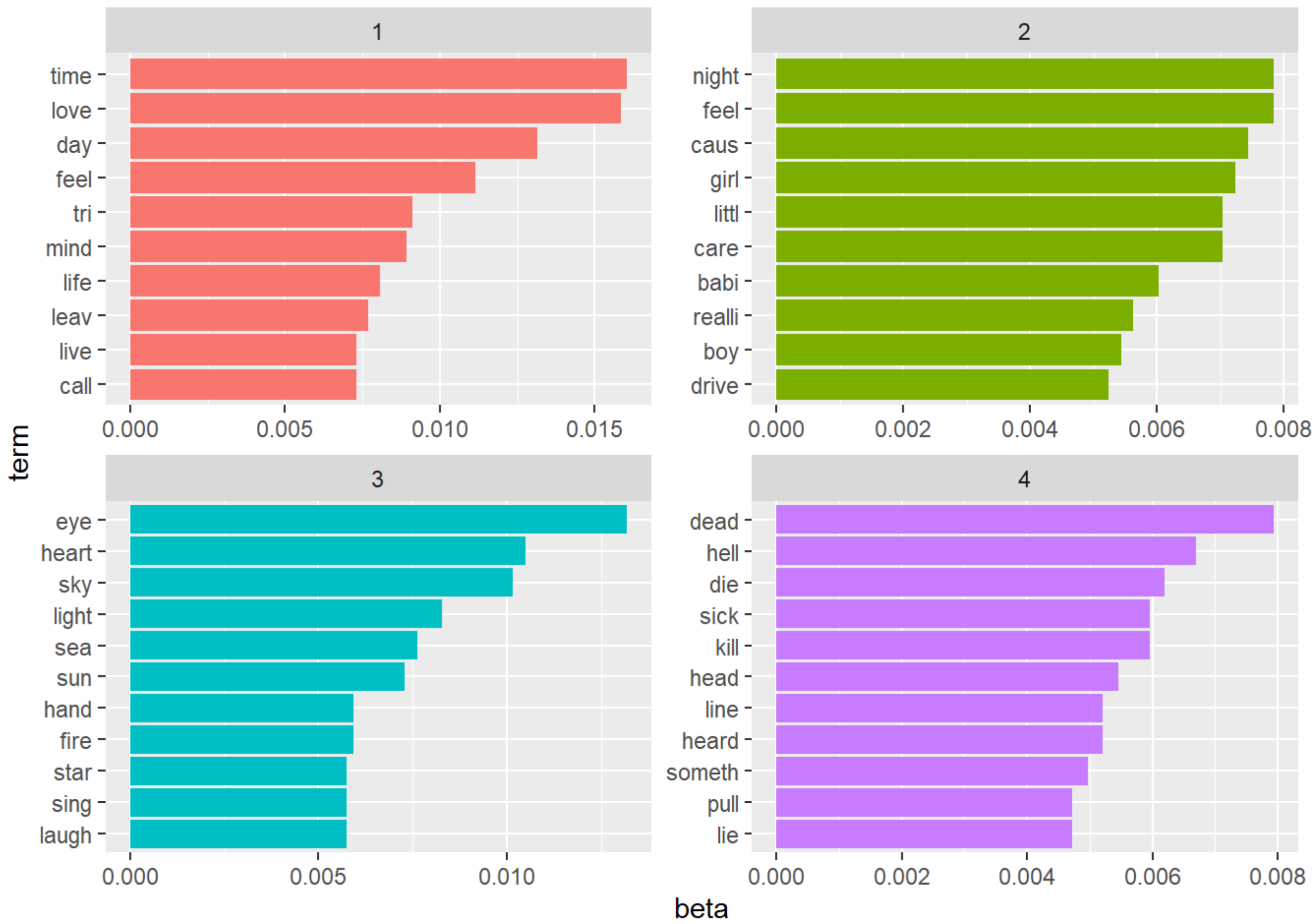
In the next section of code we look for the best number of topics for the implementation of LDA, the choice is based on the level of coherence within the words in the same topic. To do so we use LDA based on the Gibbs Sampler instead of VEM because the topics result more comprehensible.

```
tokens=df_tidy3 %>% filter(!(word=="")) %>%  # implementation of a format used to create the
 Document Term Matrix
  mutate(ind=row_number()) %>%
  group_by(title) %>% mutate(ind=row_number()) %>%
  tidyr::spread(key=ind,value=word)
tokens[is.na(tokens)]=""
tokens=tidyr::unite(tokens,text,-title,sep =" " )
tokens$text=trimws(tokens$text)

dtm=CreateDtm(tokens$text,doc_names=tokens$title,ngram_window=c(1,2))  # DTM creation
tf=TermDocFreq(dtm=dtm)
original_tf=tf %>% select(term,term_freq,doc_freq)
rownames(original_tf)=1:nrow(original_tf)
vocabulary=tf$term[tf$term_freq>1 & tf$doc_freq<nrow(dtm)/2]

# selection of the number of topics
k_list=seq(1,10,by=1)
model_dir=paste0("models_",digest::digest(vocabulary,algo="sha1"))
if(!dir.exists(model_dir)){
  dir.create(model_dir)
}

model_list=TmParallelApply(X=k_list,FUN=function(k){   # we save the 10 models on our PC for
 re-implementation
  filename = file.path(model_dir, paste0(k, "_topics.rda"))
  set.seed(100)
  if(!file.exists(filename)){
    m=FitLdaModel(dtm=dtm,k=k,optimize_alpha=TRUE, iterations=400)  # the default parameter o
f the Dirichlet distribution is beta=0.05 and the optimized value of alpha
    m$k=k
    m$coherence=CalcProbCoherence(phi=m$phi,dtm=dtm,M=10)
    save(m,file=filename)
  }else{
    load(filename)
  }
  m
},export=c("dtm","model_dir"))

# Matrix containing the 10 levels of coherence for each value of k (number of topics)
coherence_mat=data.frame(k=sapply(model_list, function(x) nrow(x$phi)),
                            coherence = sapply(model_list, function(x) mean(x$coherence)),
                            stringsAsFactors = FALSE)

ggplot(coherence_mat,aes(x=k,y=coherence)) +
  geom_point() +
  geom_line(group=1)+
  ggtitle("Best Topic by Coherence Score") +
  theme_minimal() +
  scale_x_continuous(breaks=seq(1,10,1)) +
  ylab("Coherence")
```

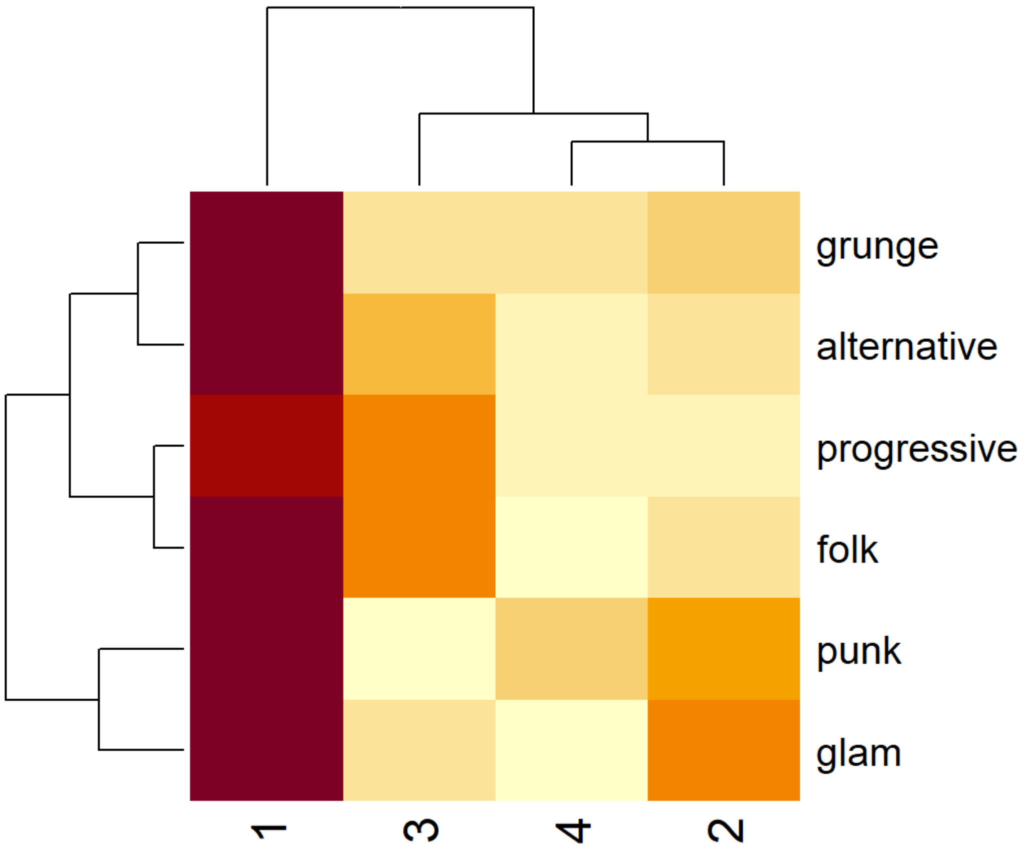


The best level of coherence is achieved with 4 topics. We must notice that the creation of the topics is strictly related to the seed we use in the analysis. Since we conclude that the optimal number of topics is equal to 4, we can implement LDA fixing k=4 for the entire dataset. We use LDA() instead of FitLdaModel() as the former allows for a better representation.

```
dfm_all=df_tidy3 %>%
  count(title,word) %>%
  cast_dfm(title,word,n)

dfm_all$title=dataset2$title[order(dataset2$title)]
dfm_all$subgenre=dataset2$subgenre[order(dataset2$title)]
dtm_all=convert(dfm_all,to="topicmodels")

lda_all=LDA(dtm_all,k=4,control=list(seed=100,alpha=0.5),method="Gibbs")
topic_all=tidy(lda_all,matrix="beta")

top_terms_all=topic_all %>%
  group_by(topic) %>%
  top_n(10,beta) %>%
  ungroup() %>%
  arrange(topic,-beta)

plot_topic_all=top_terms_all %>%
  mutate(term=reorder_within(term,beta,topic)) %>%
  ggplot(aes(term,beta,fill=factor(topic))) +
  geom_col(show.legend=FALSE) +
  facet_wrap(~topic,scales="free") +
  coord_flip() +
  scale_x_reordered()

plot_topic_all
```



```
topic.docs=topicmodels::posterior(lda_all)$topics[,4]
topic.docs=sort(topic.docs,decreasing=TRUE)

topdoc=names(topic.docs)[1]
docs=docvars(dfm_all)[match(rownames(dtm_all),docnames(dfm_all)),]
docid=which(rownames(docs)==topdoc)

tpp=aggregate(topicmodels::posterior(lda_all)$topics,by=docs["subgenre"],mean)
rownames(tpp)=tpp$subgenre
heatmap(as.matrix(tpp[-1]))
```



From the previous plots we can pinpoint 4 main topics; the first three are positive topics where the first one refers to the positive aspects of life in a more general and abstract fashion. This first topic is made up of the most common words in the lyrics, in fact we will see that it's the most used topic in all subgenres. The second topic refers to a more youthful and romantic type of love. The third topic considers elements of nature and their appeal to the human senses. The last topic recalls negative aspects of life, such as death, sickness and lies. Looking at the plotted Heat Map, where the darker colors refer to the most common topics for each subgenre, we can observe that the first topic is the most used in all subgenres, given by the generality of the topic itself. The fourth topic is the least common but it's primarily used by Punk and Grunge. By looking at the dendrograms in the plots we observe that the most similar subgenres in terms of topics are Punk and Glam because they both include frequently the second topic in their lyrics. Also Progressive and Folk seem to be quite similar since they both share a focus on natural elements. The last cluster is given by Grunge and Alternative but, by looking at the Heat Map, we observe that Alternative is extremely close to Folk in terms of its topics. In conclusion we can observe two main clusters: Punk-Glam and Folk-Progressive-Alternative, we are in doubt for the results regarding Grunge because it get classified in the second cluster while its colors in the Heat Map suggest it's more similar to the first cluster.

4. Predictive Models

Our main goal in this analysis is to predict the subgenre of a song based solely on its lyrics, to do so we apply the following predictive models: Naive Bayes, Support Vector Machine and Random Forest. Before applying the predictive models on our dataset we transform the data to the corpus format, this format is a vertical or "word -per-line" text that facilitates the implementation of many models. In practice it's a collection of documents. We proceed to split the dataset in training and test using the tSparse matrix which is the Document Term Matrix removing the most frequent words, that is, the words that appear in more than 99,5% of the documents. We do a sample split where 80% of the sample is dedicated to the training set and the remaining part to the test set. Given this division, the prevision baseline for each subgenre is 16.67%.

```
corpus=Corpus(VectorSource(dataset2$text))
freq=DocumentTermMatrix(corpus)
sparse=removeSparseTerms(freq,0.995)
tSparse=as.data.frame(as.matrix(sparse))
colnames(tSparse)=make.names(colnames(tSparse))
tSparse$subgenre=dataset2$subgenre
round(prop.table(table(tSparse$subgenre)),4) #Prevision Baseline for each subgenre
```

```
##
## alternative        folk        glam      grunge progressive        punk
##      0.1667      0.1667      0.1667      0.1667      0.1667      0.1667
```

```
set.seed(100)
split=sample.split(tSparse$subgenre,SplitRatio=0.8)
tr=subset(tSparse,split==TRUE) #Training
te=subset(tSparse,split==FALSE) #Test
tr$subgenre=as.factor(tr$subgenre)
te$subgenre=as.factor(te$subgenre)
```

To evaluate the predictive performance of each model our classification metric of choice is the confusion matrix. We derive two alternative configurations of the matrix: the first table includes the values $P(Y = y|\hat{Y})$ and the second includes the values $P(\hat{Y} = y|Y)$. In particular we are interested in the percentage values on the diagonals, corresponding, respectively, to the probability that the true subgenre is correct given the prediction and the probability that the prediction is correct given the true value of the subgenre.

Naive Bayes

The first model we apply is the Naive Bayes Classifier: given a Classification Problem described by a Response Variable $Y \in \{0, 1\}$ and a set of predictors $x \in R^p = X$ we can define a classifier $h(\cdot)$ as a strategy that maps from the feature space of the predictors in the set {0,1} such that:

$$X \rightarrow \{0, 1\}$$

Considering the following loss function (on a new data pair (Y, x)):

$$L(Y, h(x)) = \mathbf{1}(Y \neq h(x))$$

we obtain the 0-1 Risk function for the new data pair:

$$R[h(x)] = E_p[L(Y, h(x)] = P(Y \neq h(x))$$

The Bayes Classifier represents the optimal classifier in this situation and it is defined as follows:

$$h_{opt} = \arg\min_h R(h) = \mathbf{1}(f_{opt}(x) \geq 1/2)$$

where $f_{opt}(x) = E[Y|X = x]$ denoting the optimal predictor obteined from a Regression problem. We can also define the optimal Bayes Classifier using the point-wise Risk:

$$h_{opt} = \arg\min_{h} E_x[P(Y \neq h(x)|X = x)]$$

In this context we apply the Naive Bayes model to generate a multi-class classifier for the data.

```
nb=naiveBayes(subgenre~.,data=tr)   #classifier
predict_nb=predict(nb,newdata=te)   #predicted values
table(te$subgenre,predict_nb)   #confusion matrix
```

```
##              predict_nb
##               alternative folk glam grunge progressive punk
##   alternative           2    1    3      6           2    6
##   folk                  4    2    2     10           1    1
##   glam                  4    0    9      3           2    2
##   grunge                4    0    1     11           2    2
##   progressive           4    1    6      2           3    4
##   punk                  1    1    5      7           3    3
```

```
round(prop.table(table(te$subgenre,predict_nb),2),2) # P(Y=y|Y_pred)
```

```
##              predict_nb
##               alternative folk glam grunge progressive punk
##   alternative        0.11 0.20 0.12   0.15        0.15 0.33
##   folk               0.21 0.40 0.08   0.26        0.08 0.06
##   glam               0.21 0.00 0.35   0.08        0.15 0.11
##   grunge             0.21 0.00 0.04   0.28        0.15 0.11
##   progressive        0.21 0.20 0.23   0.05        0.23 0.22
##   punk               0.05 0.20 0.19   0.18        0.23 0.17
```

```
prop.table(table(te$subgenre,predict_nb),1) # P(Y_pred=y|Y)
```

```
##              predict_nb
##               alternative folk glam grunge progressive punk
##   alternative        0.10 0.05 0.15   0.30        0.10 0.30
##   folk               0.20 0.10 0.10   0.50        0.05 0.05
##   glam               0.20 0.00 0.45   0.15        0.10 0.10
##   grunge             0.20 0.00 0.05   0.55        0.10 0.10
##   progressive        0.20 0.05 0.30   0.10        0.15 0.20
##   punk               0.05 0.05 0.25   0.35        0.15 0.15
```

```
round(mean(na.omit(diag(prop.table(table(te$subgenre,predict_nb),2)))),3)
```

```
## [1] 0.255
```

```
round(mean(diag(prop.table(table(te$subgenre,predict_nb),1))),3)
```

```
## [1] 0.25
```

```
#25.5% & 25%
```

The Naive Bayes Classifier reports a predictive accuracy of 25.5% and 25% in the two considered cases.

Support Vector Machine

The second predictive model we implement is SVM applied to a multi-classification problem: an SVM performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different labels. SMV was initially constructed for binary classification but it can also be extended to a multi-class implementation. Its main goal is to find the optimal hyperplane that separates observations according to their class labels using linear and non-linear class boundaries. The objective of the support vector machine algorithm is to find a hyperplane that has the maximum margin, (the maximum distance between data points in the classes) in an N-dimensional space that distinctly classifies the data points. In particular, hyperplanes are decision boundaries that help classify the data points and support vectors are the data points that are closest to the separating hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier and look for the SVM linear machine.

```
sv=svm(subgenre~.,kernel="linear",tr,scale=FALSE)  #classifier
predict_sv=predict(sv,newdata=te)   #predicted values
table(te$subgenre,predict_sv) #confusion matrix
```

```
##              predict_sv
##               alternative folk glam grunge progressive punk
##   alternative           7    2    3      3           3    2
##   folk                  4    7    1      3           1    4
##   glam                  3    3    6      3           1    4
##   grunge                4    1    3      4           3    5
##   progressive           1    2    3      2          10    2
##   punk                  4    2    2      5           2    5
```

```
round(prop.table(table(te$subgenre,predict_sv),2),2) # P(Y=y|Y_pred)
```

```
##              predict_sv
##               alternative folk glam grunge progressive punk
##   alternative        0.30 0.12 0.17   0.15        0.15 0.09
##   folk               0.17 0.41 0.06   0.15        0.05 0.18
##   glam               0.13 0.18 0.33   0.15        0.05 0.18
##   grunge             0.17 0.06 0.17   0.20        0.15 0.23
##   progressive        0.04 0.12 0.17   0.10        0.50 0.09
##   punk               0.17 0.12 0.11   0.25        0.10 0.23
```

```
prop.table(table(te$subgenre,predict_sv),1) # P(Y_pred=y|Y)
```

```
##              predict_sv
##               alternative folk glam grunge progressive punk
##   alternative        0.35 0.10 0.15   0.15        0.15 0.10
##   folk               0.20 0.35 0.05   0.15        0.05 0.20
##   glam               0.15 0.15 0.30   0.15        0.05 0.20
##   grunge             0.20 0.05 0.15   0.20        0.15 0.25
##   progressive        0.05 0.10 0.15   0.10        0.50 0.10
##   punk               0.20 0.10 0.10   0.25        0.10 0.25
```

```
round(mean(na.omit(diag(prop.table(table(te$subgenre,predict_sv),2)))),3)
```

```
## [1] 0.329
```

```
round(mean(diag(prop.table(table(te$subgenre,predict_sv),1))),3)
```

```
## [1] 0.325
```

```
#32.9% & 32.5%
```

The SVM Classifier reports a predictive accuracy of 32.9% and 32.5% in the two considered cases.

Random Forest

The final predictive model we implemented is Random forest. This model consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. The reason that the random forest model works so well is that a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. Uncorrelated models can produce ensemble predictions that are more accurate than any of the individual prediction, because the trees protect each other from their individual errors. So the prerequisites for random forest to perform well are: 1.We need features that have at least some predictive power. 2.The trees of the forest and more importantly their predictions need to be uncorrelated (or at least have low correlations with each other). While the algorithm itself via feature randomness tries to engineer these low correlations for us, the features we select and the hyper-parameters we choose will impact the ultimate correlations as well.

```
set.seed(100)
rfmod=randomForest(subgenre~.,data=tr) #classifier
predict_rf=predict(rfmod,newdata=te)  #predicted values
table(te$subgenre,predict_rf)  #confusion matrix
```

```
##              predict_rf
##               alternative folk glam grunge progressive punk
##   alternative           2    3    4      6           2    3
##   folk                  2    4    5      5           3    1
##   glam                  0    0    8      4           1    7
##   grunge                3    1    4      3           3    6
##   progressive           2    1    0      1          12    4
##   punk                  1    1    3      3           5    7
```

```
round(prop.table(table(te$subgenre,predict_rf),2),2) # P(Y=y|Y_pred)
```

```
##              predict_rf
##               alternative folk glam grunge progressive punk
##   alternative        0.20 0.30 0.17   0.27        0.08 0.11
##   folk               0.20 0.40 0.21   0.23        0.12 0.04
##   glam               0.00 0.00 0.33   0.18        0.04 0.25
##   grunge             0.30 0.10 0.17   0.14        0.12 0.21
##   progressive        0.20 0.10 0.00   0.05        0.46 0.14
##   punk               0.10 0.10 0.12   0.14        0.19 0.25
```

```
prop.table(table(te$subgenre,predict_rf),1) # P(Y_pred=y|Y)
```

```
##              predict_rf
##               alternative folk glam grunge progressive punk
##   alternative        0.10 0.15 0.20   0.30        0.10 0.15
##   folk               0.10 0.20 0.25   0.25        0.15 0.05
##   glam               0.00 0.00 0.40   0.20        0.05 0.35
##   grunge             0.15 0.05 0.20   0.15        0.15 0.30
##   progressive        0.10 0.05 0.00   0.05        0.60 0.20
##   punk               0.05 0.05 0.15   0.15        0.25 0.35
```

```
round(mean(na.omit(diag(prop.table(table(te$subgenre,predict_rf),2)))),3)
```

```
## [1] 0.297
```

```
round(mean(diag(prop.table(table(te$subgenre,predict_rf),1))),3)
```

```
## [1] 0.3
```

```
#29.7% & 30%
```

The Random Forest model reports a predictive accuracy of 29.7% and 30% in the two considered cases.

Comments on the results

Based on this particular split sample we observe that for the Naive Bayes classifier the best predicted subgenres are Grunge and Glam while the hardest to predict are Folk and Alternative. Looking at the results obtained from the SVM classifier and Random Forest we can observe that the best predicted subgenre in both cases is Progressive while the worst are Grunge for SVM and Alternative for Random Forest. We can also observe that in the predictions from SVM are much more stable than the last model considered because the predicted probabilities for each subgenre are more concentrated around the general predictive accuracy. Based on this specific split, SVM is the most accurate classifier followed by Random Forest; to validate this result we decide to implement a Monte Carlo Simulation of the split to see if the same result holds in the case of repeated sampling:

```
matriciona=matrix(0,100,6)
for(i in 1:100){
  set.seed(i)
  split=sample.split(tSparse$subgenre,SplitRatio=0.8)
  tr=subset(tSparse,split==TRUE)
  te=subset(tSparse,split==FALSE)
  tr$subgenre=as.factor(tr$subgenre)
  te$subgenre=as.factor(te$subgenre)
  rfmod=randomForest(subgenre~.,data=tr)
  predict_rf=predict(rfmod,newdata=te)
  sv=svm(subgenre~.,tr,kernel="linear",scale=FALSE)
  predict_sv=predict(sv,newdata=te)
  nb=naiveBayes(subgenre~.,data=tr)
  predict_nb=predict(nb,newdata=te)
  matriciona[i,1]=mean(na.omit(diag(prop.table(table(te$subgenre,predict_rf),2))))
  matriciona[i,2]=mean(diag(prop.table(table(te$subgenre,predict_rf),1)))
  matriciona[i,3]=mean(na.omit(diag(prop.table(table(te$subgenre,predict_sv),2))))
  matriciona[i,4]=mean(diag(prop.table(table(te$subgenre,predict_sv),1)))
  matriciona[i,5]=mean(na.omit(diag(prop.table(table(te$subgenre,predict_nb),2))))
  matriciona[i,6]=mean(diag(prop.table(table(te$subgenre,predict_nb),1)))
}

round(apply(matriciona,2,mean),3)
```

```
## [1] 0.322 0.324 0.295 0.295 0.209 0.197
```

```
#random forest: 32.2%/32.4% , svm: 29.5%/29.5% , naive bayes: 20.9%/19.7%
```

We observe that the most accurate predictions derive from the Random Forest model, with an accuracy around 32%. The predictions gathered from this model add a 16% improvement to the baseline predictions (16.67%), that is, the case in which we were to apply a random classification using a balanced die where each side of the die is a subgenre!

Appendix

In the following appendix we leave the code with the results obtained in the case in which we run the LDA for each subgenre taken individually. For each of them we observed which are the resulting topics after choosing the ideal number of topics k and which of these have more weight within that subgenre in percentage (prevalence). We note that in all cases as the number of k increases the coherence in the topics tends to increase. The topics we detect are difficult to interpret and sometimes the words are not very coherent with each other within the same topic. This result may be due to a not sufficiently large number of songs sampled for each subgenre: in the LDA, the larger the number of documents, the better the model fit. These interpreation issues may also be due to the general structure of lyrics. In fact, it generally may be hard to distinguish specific themes within a song because its syntactic structure differ from the one of an article or book chapter. The topics found therefore appear to contain many generic terms and the love theme, in its various connotations, is predominant for all subgenres.

```
subset1=dataset2[dataset2$subgenre=="alternative",]
subset1=subset1 %>% select(title,text)

subset2=dataset2[dataset2$subgenre=="folk",]
subset2=subset2 %>% select(title,text)

subset3=dataset2[dataset2$subgenre=="glam",]
subset3=subset3 %>% select(title,text)

subset4=dataset2[dataset2$subgenre=="grunge",]
subset4=subset4 %>% select(title,text)

subset5=dataset2[dataset2$subgenre=="progressive",]
subset5=subset5 %>% select(title,text)

subset6=dataset2[dataset2$subgenre=="punk",]
subset6=subset6 %>% select(title,text)


# lda alternative

df_tidy3=subset1 %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  distinct() %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2)
```

```
## Joining, by = "word"
```

```
tokens=df_tidy3 %>% filter(!(word=="")) %>%
  mutate(ind=row_number()) %>%
  group_by(title) %>% mutate(ind=row_number()) %>%
  tidyr::spread(key=ind,value=word)
tokens[is.na(tokens)]=""
tokens=tidyr::unite(tokens,text,-title,sep =" " )
tokens$text=trimws(tokens$text)


dtm=CreateDtm(tokens$text,doc_names=tokens$title,ngram_window=c(1,2))
tf=TermDocFreq(dtm=dtm)
original_tf=tf %>% select(term,term_freq,doc_freq)
rownames(original_tf)=1:nrow(original_tf)
vocabulary=tf$term[tf$term_freq>1 & tf$doc_freq<nrow(dtm)/2]


model_dir=paste0("models_",digest::digest(vocabulary,algo="sha1"))
if(!dir.exists(model_dir)){
  dir.create(model_dir)
}

model_list1=TmParallelApply(X=k_list,FUN=function(k){
  filename = file.path(model_dir, paste0(k, "_topics.rda"))
  set.seed(100)
  if(!file.exists(filename)){
    m=FitLdaModel(dtm=dtm,k=k,optimize_alpha=TRUE,iterations=100)
    m$k=k
    m$coherence=CalcProbCoherence(phi=m$phi,dtm=dtm,M=10)
    save(m,file=filename)
  }else{
    load(filename)
  }
  m
},export=c("dtm","model_dir"))


coherence_mat1=data.frame(k=sapply(model_list1, function(x) nrow(x$phi)),
                          coherence = sapply(model_list1, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)


model1=model_list1[which.max(coherence_mat1$coherence)][[1]]
model1$top_terms=GetTopTerms(phi=model1$phi,M=10)
top10_wide1=as.data.frame(model1$top_terms)


# lda folk

df_tidy3=subset2 %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  distinct() %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2)
```

```
## Joining, by = "word"
```

```
tokens=df_tidy3 %>% filter(!(word=="")) %>%
  mutate(ind=row_number()) %>%
  group_by(title) %>% mutate(ind=row_number()) %>%
  tidyr::spread(key=ind,value=word)
tokens[is.na(tokens)]=""
tokens=tidyr::unite(tokens,text,-title,sep =" " )
tokens$text=trimws(tokens$text)


dtm=CreateDtm(tokens$text,doc_names=tokens$title,ngram_window=c(1,2))
tf=TermDocFreq(dtm=dtm)
original_tf=tf %>% select(term,term_freq,doc_freq)
rownames(original_tf)=1:nrow(original_tf)
vocabulary=tf$term[tf$term_freq>1 & tf$doc_freq<nrow(dtm)/2]


model_dir=paste0("models_",digest::digest(vocabulary,algo="sha1"))
if(!dir.exists(model_dir)){
  dir.create(model_dir)
}

model_list2=TmParallelApply(X=k_list,FUN=function(k){
  filename = file.path(model_dir, paste0(k, "_topics.rda"))
  set.seed(100)
  if(!file.exists(filename)){
    m=FitLdaModel(dtm=dtm,k=k,optimize_alpha=TRUE,iterations=100)
    m$k=k
    m$coherence=CalcProbCoherence(phi=m$phi,dtm=dtm,M=10)
    save(m,file=filename)
  }else{
    load(filename)
  }
  m
},export=c("dtm","model_dir"))


coherence_mat2=data.frame(k=sapply(model_list2, function(x) nrow(x$phi)),
                          coherence = sapply(model_list2, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)


model2=model_list2[which.max(coherence_mat2$coherence)][[1]]
model2$top_terms=GetTopTerms(phi=model2$phi,M=10)
top10_wide2=as.data.frame(model2$top_terms)


# lda glam

df_tidy3=subset3 %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  distinct() %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2)
```

```
## Joining, by = "word"
```

```
tokens=df_tidy3 %>% filter(!(word=="")) %>%
  mutate(ind=row_number()) %>%
  group_by(title) %>% mutate(ind=row_number()) %>%
  tidyr::spread(key=ind,value=word)
tokens[is.na(tokens)]=""
tokens=tidyr::unite(tokens,text,-title,sep =" " )
tokens$text=trimws(tokens$text)


dtm=CreateDtm(tokens$text,doc_names=tokens$title,ngram_window=c(1,2))
tf=TermDocFreq(dtm=dtm)
original_tf=tf %>% select(term,term_freq,doc_freq)
rownames(original_tf)=1:nrow(original_tf)
vocabulary=tf$term[tf$term_freq>1 & tf$doc_freq<nrow(dtm)/2]


model_dir=paste0("models_",digest::digest(vocabulary,algo="sha1"))
if(!dir.exists(model_dir)){
  dir.create(model_dir)
}

model_list3=TmParallelApply(X=k_list,FUN=function(k){
  filename = file.path(model_dir, paste0(k, "_topics.rda"))
  set.seed(100)
  if(!file.exists(filename)){
    m=FitLdaModel(dtm=dtm,k=k,optimize_alpha=TRUE,iterations=100)
    m$k=k
    m$coherence=CalcProbCoherence(phi=m$phi,dtm=dtm,M=10)
    save(m,file=filename)
  }else{
    load(filename)
  }
  m
},export=c("dtm","model_dir"))


coherence_mat3=data.frame(k=sapply(model_list3, function(x) nrow(x$phi)),
                          coherence = sapply(model_list3, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)


model3=model_list3[which.max(coherence_mat3$coherence)][[1]]
model3$top_terms=GetTopTerms(phi=model3$phi,M=10)
top10_wide3=as.data.frame(model3$top_terms)


# lda grunge

df_tidy3=subset4 %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  distinct() %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2)
```

```
## Joining, by = "word"
```

```
tokens=df_tidy3 %>% filter(!(word=="")) %>%
  mutate(ind=row_number()) %>%
  group_by(title) %>% mutate(ind=row_number()) %>%
  tidyr::spread(key=ind,value=word)
tokens[is.na(tokens)]=""
tokens=tidyr::unite(tokens,text,-title,sep =" " )
tokens$text=trimws(tokens$text)


dtm=CreateDtm(tokens$text,doc_names=tokens$title,ngram_window=c(1,2))
tf=TermDocFreq(dtm=dtm)
original_tf=tf %>% select(term,term_freq,doc_freq)
rownames(original_tf)=1:nrow(original_tf)
vocabulary=tf$term[tf$term_freq>1 & tf$doc_freq<nrow(dtm)/2]


model_dir=paste0("models_",digest::digest(vocabulary,algo="sha1"))
if(!dir.exists(model_dir)){
  dir.create(model_dir)
}

model_list4=TmParallelApply(X=k_list,FUN=function(k){
  filename = file.path(model_dir, paste0(k, "_topics.rda"))
  set.seed(100)
  if(!file.exists(filename)){
    m=FitLdaModel(dtm=dtm,k=k,optimize_alpha=TRUE,iterations=100)
    m$k=k
    m$coherence=CalcProbCoherence(phi=m$phi,dtm=dtm,M=10)
    save(m,file=filename)
  }else{
    load(filename)
  }
  m
},export=c("dtm","model_dir"))


coherence_mat4=data.frame(k=sapply(model_list4, function(x) nrow(x$phi)),
                          coherence = sapply(model_list4, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)


model4=model_list4[which.max(coherence_mat4$coherence)][[1]]
model4$top_terms=GetTopTerms(phi=model4$phi,M=10)
top10_wide4=as.data.frame(model4$top_terms)


#lda progressive

df_tidy3=subset5 %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  distinct() %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2)
```

```
## Joining, by = "word"
```

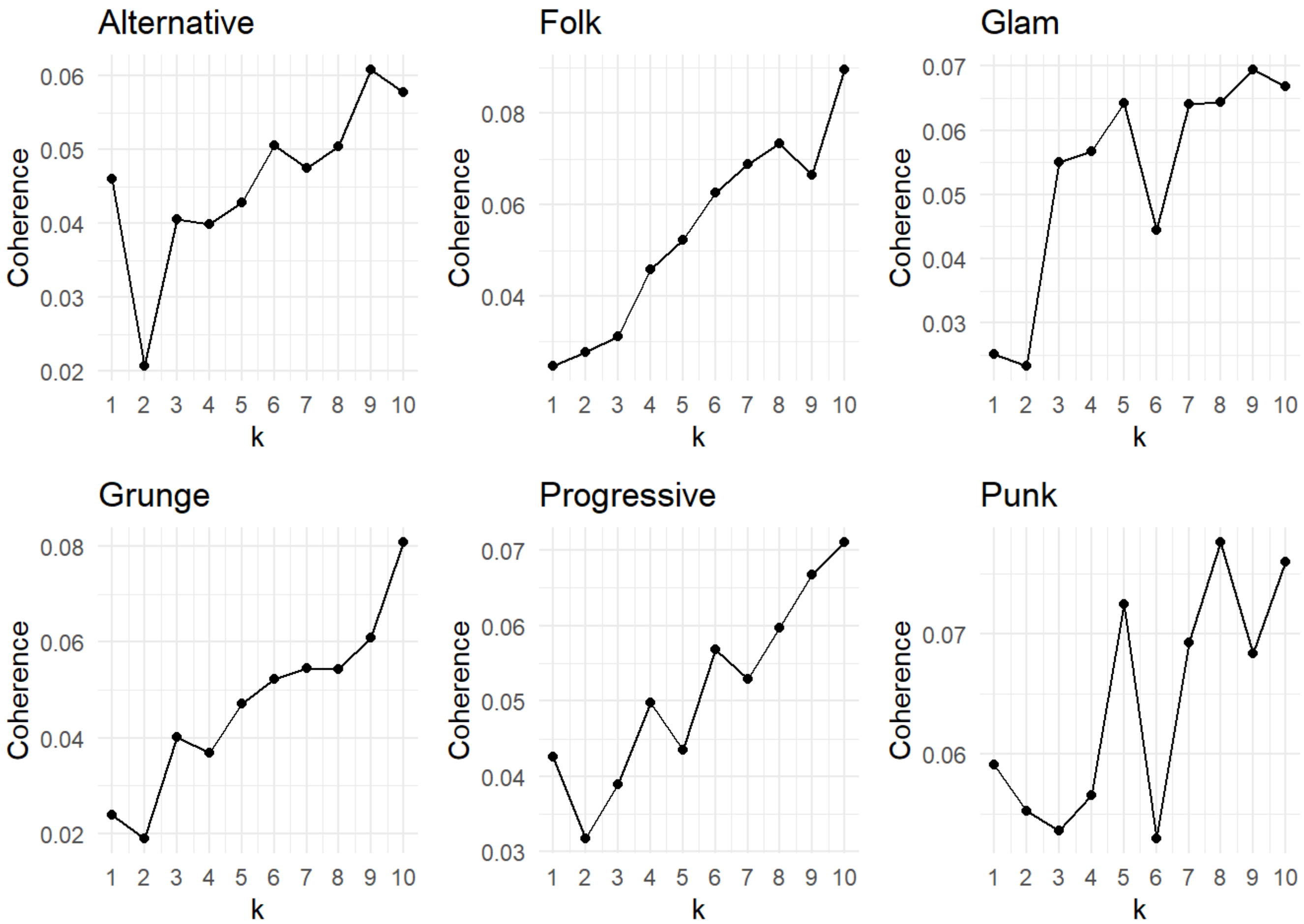
```
tokens=df_tidy3 %>% filter(!(word=="")) %>%
  mutate(ind=row_number()) %>%
  group_by(title) %>% mutate(ind=row_number()) %>%
  tidyr::spread(key=ind,value=word)
tokens[is.na(tokens)]=""
tokens=tidyr::unite(tokens,text,-title,sep =" " )
tokens$text=trimws(tokens$text)


dtm=CreateDtm(tokens$text,doc_names=tokens$title,ngram_window=c(1,2))
tf=TermDocFreq(dtm=dtm)
original_tf=tf %>% select(term,term_freq,doc_freq)
rownames(original_tf)=1:nrow(original_tf)
vocabulary=tf$term[tf$term_freq>1 & tf$doc_freq<nrow(dtm)/2]


model_dir=paste0("models_",digest::digest(vocabulary,algo="sha1"))
if(!dir.exists(model_dir)){
  dir.create(model_dir)
}

model_list5=TmParallelApply(X=k_list,FUN=function(k){
  filename = file.path(model_dir, paste0(k, "_topics.rda"))
  set.seed(100)
  if(!file.exists(filename)){
    m=FitLdaModel(dtm=dtm,k=k,optimize_alpha=TRUE,iterations=100)
    m$k=k
    m$coherence=CalcProbCoherence(phi=m$phi,dtm=dtm,M=10)
    save(m,file=filename)
  }else{
    load(filename)
  }
  m
},export=c("dtm","model_dir"))


coherence_mat5=data.frame(k=sapply(model_list5, function(x) nrow(x$phi)),
                          coherence = sapply(model_list5, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)


model5=model_list5[which.max(coherence_mat5$coherence)][[1]]
model5$top_terms=GetTopTerms(phi=model5$phi,M=10)
top10_wide5=as.data.frame(model5$top_terms)


#lda punk

df_tidy3=subset6 %>%
  ungroup() %>%
  unnest_tokens(word,text) %>%
  distinct() %>%
  anti_join(stop_words) %>%
  filter(nchar(word)>2)
```

```
## Joining, by = "word"
```

```
tokens=df_tidy3 %>% filter(!(word=="")) %>%
  mutate(ind=row_number()) %>%
  group_by(title) %>% mutate(ind=row_number()) %>%
  tidyr::spread(key=ind,value=word)
tokens[is.na(tokens)]=""
tokens=tidyr::unite(tokens,text,-title,sep =" " )
tokens$text=trimws(tokens$text)


dtm=CreateDtm(tokens$text,doc_names=tokens$title,ngram_window=c(1,2))
tf=TermDocFreq(dtm=dtm)
original_tf=tf %>% select(term,term_freq,doc_freq)
rownames(original_tf)=1:nrow(original_tf)
vocabulary=tf$term[tf$term_freq>1 & tf$doc_freq<nrow(dtm)/2]


model_dir=paste0("models_",digest::digest(vocabulary,algo="sha1"))
if(!dir.exists(model_dir)){
  dir.create(model_dir)
}

model_list6=TmParallelApply(X=k_list,FUN=function(k){
  filename = file.path(model_dir, paste0(k, "_topics.rda"))
  set.seed(100)
  if(!file.exists(filename)){
    m=FitLdaModel(dtm=dtm,k=k,optimize_alpha=TRUE,iterations=100)
    m$k=k
    m$coherence=CalcProbCoherence(phi=m$phi,dtm=dtm,M=10)
    save(m,file=filename)
  }else{
    load(filename)
  }
  m
},export=c("dtm","model_dir"))


coherence_mat6=data.frame(k=sapply(model_list6, function(x) nrow(x$phi)),
                          coherence = sapply(model_list6, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)

model6=model_list6[which.max(coherence_mat6$coherence)][[1]]
model6$top_terms=GetTopTerms(phi=model6$phi,M=10)
top10_wide6=as.data.frame(model6$top_terms)


# Comparison between k values

coherence_mat1=data.frame(k=sapply(model_list1, function(x) nrow(x$phi)),
                          coherence = sapply(model_list1, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)

plot1=ggplot(coherence_mat1,aes(x=k,y=coherence)) +
  geom_point() +
  geom_line(group=1)+
  ggtitle("Alternative") +
  theme_minimal() +
  scale_x_continuous(breaks=seq(1,10,1)) +
```

```
  ylab("Coherence")



coherence_mat2=data.frame(k=sapply(model_list2, function(x) nrow(x$phi)),
                          coherence = sapply(model_list2, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)

plot2=ggplot(coherence_mat2,aes(x=k,y=coherence)) +
  geom_point() +
  geom_line(group=1)+
  ggtitle("Folk") +
  theme_minimal() +
  scale_x_continuous(breaks=seq(1,10,1)) +
  ylab("Coherence")



coherence_mat3=data.frame(k=sapply(model_list3, function(x) nrow(x$phi)),
                          coherence = sapply(model_list3, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)

plot3=ggplot(coherence_mat3,aes(x=k,y=coherence)) +
  geom_point() +
  geom_line(group=1)+
  ggtitle("Glam") +
  theme_minimal() +
  scale_x_continuous(breaks=seq(1,10,1)) +
  ylab("Coherence")



coherence_mat4=data.frame(k=sapply(model_list4, function(x) nrow(x$phi)),
                          coherence = sapply(model_list4, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)

plot4=ggplot(coherence_mat4,aes(x=k,y=coherence)) +
  geom_point() +
  geom_line(group=1)+
  ggtitle("Grunge") +
  theme_minimal() +
  scale_x_continuous(breaks=seq(1,10,1)) +
  ylab("Coherence")



coherence_mat5=data.frame(k=sapply(model_list5, function(x) nrow(x$phi)),
                          coherence = sapply(model_list5, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)

plot5=ggplot(coherence_mat5,aes(x=k,y=coherence)) +
  geom_point() +
  geom_line(group=1)+
  ggtitle("Progressive") +
  theme_minimal() +
  scale_x_continuous(breaks=seq(1,10,1)) +
  ylab("Coherence")
```

```
coherence_mat6=data.frame(k=sapply(model_list6, function(x) nrow(x$phi)),
                          coherence = sapply(model_list6, function(x) mean(x$coherence)),
                          stringsAsFactors = FALSE)

plot6=ggplot(coherence_mat6,aes(x=k,y=coherence)) +
  geom_point() +
  geom_line(group=1)+
  ggtitle("Punk") +
  theme_minimal() +
  scale_x_continuous(breaks=seq(1,10,1)) +
  ylab("Coherence")


grid.arrange(plot1,plot2,plot3,plot4,plot5,plot6,nrow=2,ncol=3)   # Plotted k values
```




```
# Topics and Prevalence

top10_wide1  #topics in alternative
```

```
##       t_1   t_2    t_3   t_4    t_5     t_6     t_7     t_8     t_9
## 1    life  hear   word   tri   door everyth   heart    hand    hold
## 2  friend  time  watch break breath    live    caus     eye     lip
## 3    time  feel    day  pull    bed    danc    fire    mind    star
## 4    feel  lost  alway  push  world   world    hope    fall     sky
## 5   dream start   love scare   love    left   night everyon     fli
## 6   sleep light   noth  voic    sad    hour   speak    girl beneath
## 7     day close believ   run   sick pretend   truth    grow    bodi
## 8    wait chang   feet littl   roll   smoke control   peopl    burn
## 9    love  care  night  head  lover    cold   floor     sit    caus
## 10  pleas  call reason black   caus  someth    lock    mine    melt
```

```
colSums(model1$theta)/sum(model1$theta)*100  #prevalence
```

```
##       t_1       t_2       t_3       t_4       t_5       t_6       t_7       t_8
## 12.884980 13.911354 11.940848 10.532183  7.889213 10.480991 10.448365 10.520057
##       t_9
## 11.392009
```

```
top10_wide2  #topics in folk
```

```
##         t_1    t_2   t_3    t_4   t_5    t_6   t_7    t_8     t_9  t_10
## 1       day    sun littl   sing   eye   call   day   life    head  feel
## 2      walk   wind  burn  heart  time   rest  wall  water    caus  love
## 3       tri   star wrong  queen  love   hear  land  world   chang  leav
## 4      time valley  song   gold   cri   mani   sea   town   smile heart
## 5      door  shine  call    bow dream    sea  blue   fire  strang  girl
## 6  mountain   dark   cut   lord  stay  littl  fill    die brought  fall
## 7      hard    run  kiss travel  morn  white  meet broken   floor light
## 8      hand   rise  soul    red  live  heard music   darl    move night
## 9      told  stone everi   alon night  speak  news troubl     die  noth
## 10    laugh    lay   sad   king  left breath  sail   live     air  word
```

```
colSums(model2$theta)/sum(model2$theta)*100  #prevalence
```

```
##       t_1       t_2       t_3       t_4       t_5       t_6       t_7       t_8
##  9.064084  9.982325  7.975481  6.561637 13.789936  9.263207  9.909429  9.691465
##       t_9      t_10
##  9.515624 14.246811
```

```
top10_wide3  #topics in glam
```

```
##      t_1    t_2    t_3    t_4   t_5     t_6   t_7     t_8     t_9
## 1   babi  world  laugh   time  live   heart  rock    love    caus
## 2   hand  night   danc  insid  life tonight  star     day  street
## 3   girl  dream  smile  alway night    talk  caus    home     run
## 4  drive realli  ignor   love  time    feel wrong   littl   watch
## 5   late    sky  stole    boy   die   everi  roll  friend  strang
## 6   leav   love  super   true   tri alright peopl    time    meet
## 7    eye   play    met believ  pain    head heard    lone    feet
## 8  crazi   mind    cri  honey  wall    walk sweet    kiss    mind
## 9   fool    fli someth   call  door     lip  mind   everi control
## 10 queen    sun   blue  light pleas    tast  band everyth    fear
```

```
colSums(model3$theta)/sum(model3$theta)*100  #prevalence
```

```
##       t_1       t_2       t_3       t_4       t_5       t_6       t_7       t_8
## 10.823252 12.361227  7.972521 11.444514 11.977488  9.947486 10.563439 15.355990
##       t_9
##  9.554082
```

```
top10_wide4  #topics in grunge
```

```
##        t_1   t_2    t_3     t_4   t_5    t_6   t_7     t_8    t_9       t_10
## 1     care insid    eye    mind  burn  anoth  love     tri  forev       time
## 2     hair   eye   feel everyth night   fall  mayb    hard   stop       left
## 3     babi  leav    lie    live blind   noth  sing   water   star       pain
## 4     rock   sun   lost    alon  hell    day  hate    roll   hold       wait
## 5      bit world  heart    soul smile   caus  hand   bring beauti       play
## 6     king close   hand    love littl ground  head    hour   love understand
## 7     sick light believ    walk  fill   hang  skin     sea   feel        god
## 8     home  kiss  alway    fear watch    cri wrong    warm   talk     someth
## 9    blond  free   word   dream  kill afraid  babi  yellow breath        cri
## 10 descend heart   time     lie  aliv   sinc  word alright  everi     friend
```

```
colSums(model4$theta)/sum(model4$theta)*100  #prevalence
```

```
##       t_1       t_2       t_3       t_4       t_5       t_6       t_7       t_8
##  7.984651 11.951867 11.386512 11.324513 10.198456  9.696263 10.011327  7.542135
##       t_9      t_10
## 10.149579  9.754697
```

```
top10_wide5  #topics in progressive
```

```
##         t_1    t_2     t_3   t_4   t_5    t_6   t_7    t_8   t_9     t_10
## 1    follow  light    time   day  hand   leav  fear   time   sky     alon
## 2      sing  night    fall  mind peopl    tri  danc    pay   cri      eye
## 3      song   call   pleas  love begin   love  word   ride heart    alway
## 4       sun   dark    dawn  feel queen   life teeth   stop   eye    reach
## 5       god  green     lie  time   sin believ tongu corner   sun    truth
## 6     sound  smile   round world  grow  dream  home    fun  head     hold
## 7       eye bright everyth  live  wait   caus  leav    mud illus     live
## 8     spell   deep    hear  lost  head   noth  hand  brain water somewher
## 9  children  white    feel night   day   free blood   sign watch   search
## 10    sweet   wind   everi  walk bleed  stand   ear   told alway   reason
```

```
colSums(model5$theta)/sum(model5$theta)*100  #prevalence
```

```
##       t_1       t_2       t_3       t_4       t_5       t_6       t_7       t_8
##  8.432155  9.157292 10.388437 14.517148  8.945392 12.762383  8.346510  8.324754
##       t_9      t_10
##  9.274355  9.851574
```

```
top10_wide6  #topics in punk
```

```
##      t_1     t_2   t_3    t_4    t_5    t_6     t_7    t_8
## 1    die     cos money    tri   feel street   insid   caus
## 2  dream    care  call  sound   time   talk     eye   hell
## 3   hard  realli  fuck   walk   citi  alway   heart  break
## 4   live brother   day   caus  night   girl everyth   dead
## 5  watch    noth peopl   lose   mind   hand     sun believ
## 6  sleep    town  wall   sick   fast   stay    hear   home
## 7    boy   anoth  hold ground friend   love   everi memori
## 8   fade   littl happi   play  readi    tri    time   blue
## 9   news    move   bag happen   wait   babi    head   head
## 10 white   wrong  born  found  close  guess     eat  touch
```

```
colSums(model6$theta)/sum(model6$theta)*100  #prevalence
```

```
##       t_1       t_2       t_3       t_4       t_5       t_6       t_7       t_8
## 12.612797 14.733006 10.595368 13.137041 13.240674 13.879717 12.126465  9.674932
```

Sitography

https://www.rdocumentation.org/ (https://www.rdocumentation.org/)

https://www.tidytextmining.com/ (https://www.tidytextmining.com/)

https://towardsdatascience.com/beginners-guide-to-lda-topic-modelling-with-r-e57a5a8e7a25 (https://towardsdatascience.com/beginners-guide-to-lda-topic-modelling-with-r-e57a5a8e7a25)

https://rpubs.com/rafrys/723764 (https://rpubs.com/rafrys/723764)

https://rpubs.com/Nush12/textmining (https://rpubs.com/Nush12/textmining)

https://www.datanovia.com/en/lessons/heatmap-in-r-static-and-interactive-visualization/ (https://www.datanovia.com/en/lessons/heatmap-in-r-static-and-interactive-visualization/)

https://www.youtube.com/watch?v=4vuw0AsHeGw&list=PL8eNk_zTBST8olxIRFoo0YeXxEOkYdoxi&ab_channel=DataScienceDojo (https://www.youtube.com/watch?v=4vuw0AsHeGw&list=PL8eNk_zTBST8olxIRFoo0YeXxEOkYdoxi&ab_channel=DataScienceDojo)

https://elearning.uniroma1.it/course/view.php?id=4944 (https://elearning.uniroma1.it/course/view.php?id=4944)