DTI5125[EG] Data Science Applications 2021

Final Project

Under Supervision:

Prof. Dr. Arya Rahgozar

Group Members:

Abdelrahman Basha

Hussien Abdelraouf

Dina Ibrahim

Table of contents:

Problem Formulation:

Along with the rapid expansion of digital music types and formats, managing and searching for songs has become significant and big problem for the users to decide which song want to listen. The development of music recommender systems is still at a very early stage. Music recommendation is a very difficult problem as we have to structure music in a way that we recommend the favorite songs to users which is never a definite prediction. It is dynamic and sometimes influenced by factors other than users' or songs' listening history. So, here we tackled this problem and used many algorithms to recommend the favorite song for the user.

Introduction:

Recommender systems are algorithms aimed at suggesting relevant items to users (music to listen, items being movies to watch, text to read, products to buy or anything else depending on industries). They are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors.

Methodology:

We prepared the data by creating samples of 191 partitions of lyrics per music and limited them to 100 words. We pre-processed the data by removing stop words, doing stemming and lemmatization. We transformed it to BOW, TF-IDF, N-gram and LDA. We binarized the values to use them with the classification and clustering algorithms to calculate accuracy, Kappa and silhouette for the models. Finally, we did an Error-Analysis for the champion model to identity what were the characteristics of the instance records that threw the machine off.

We made the recommender system technique and we will show the steps after the implementation.

Music Dataset

This dataset provides a list of lyrics from 1950 to 2019 describing music metadata as sadness, danceability, loudness, acoustics, etc. We also provide some information as lyrics which can be used to natural language processing.

track_name	lyrics	topic
mohabbat bhi jhoothi	hold time feel break feel untrue convince speak voice tear try hold hurt try forgive okay play break string feel heart want feel tell real truth hurt lie worse anymore li	sadness
i believe	believe drop rain fall grow believe darkest night candle glow believe go astray come believe believe believe believe smallest prayer hear believe great hear word time hear l	world/life
cry	sweetheart send letter goodbye secret feel better wake dream think real false emotions feel heartaches hang long blue get bluer song remember cloudy hair	music
patricia	kiss lips want stroll charm mambo chacha meringue heaven arm japan brag geisha care long uncle eye starry sort gleam like million dollar dream come true everybo	romantic
apoape eida oneiru	till darling till matter know till dream live apart know hearts till world free oceans cross mountains climb know pain loss pray loss time till dream till memory till hold	romantic
round and round (with initcf	convoy light dead ahead merchantmen trump diesels hammer oily kill grind knuckle white eye alight slam hatch deadly night cunning chicken lair hound hell devil car	violence
opm medley: when i met yo	piece mindin world knowin life come bring give world know give reason feelin give mean life go exist begin touch hair look eye know know good sure endure light wo	world/life
it's not for me to say	care moment hold fast press lips dream heaven speak share glow grow pass meet break speak share glow grow pass meet	world/life
klappe me mama klappe me	lonely night surround power read mind hour night kiss lips hold tight unending real begin lonely night surround know certain flirtin night kiss lips hold tight baby mayb	romantic
finito la musika	tear heart seat stay awhile tear heart game steal glimpse eye stare awhile steal glimpse eye game half awhile half game word mouth swallow speak awhile word mi	sadness
drÅ¸mmer man om den, vÃ¥	better sail midnight better dive pearl better ski aspen better feed squirrels better find horseshoe better lose head better think better say better sing right loud spot o	obscene
carolina moon	moon shin shin wait moon pin pin place long hop tonight right window scatter light alright tell blue lonely dreamy moon shin wait pin place long hop tonight right wi	night/time
necessary evil	come motherfucker spit shit minutes chillin bitchin stuff frankly dont fuck intimate feel stuff feel like sit duck fuel cause theres weakness step fuckin bitch wanna ter	obscene
kanugona galanu	work morning flight watch even burn away tear life leave trouble door leave know hold go leave strand grind hear heart music end learn away hear heart music end	sadness
the carioca	see carioca foxtrot polka little rhythm blue rhythm sigh meter tricky wicked wackywicky dance true dream carioca theme kiss sigh dream carioca music light go goo	romantic
chicken reel	hahaha alright yeah alright even shoot complain things go want clichÃ© life easy tough easy whatcha gonna whatcha gonna smile face world better place smile fac	world/life
blue skies	blue sky smile blue sky bluebirds sing song bluebirds long shin bright things go right notice days hurry blue days go blue sky blue sky smile blue sky shin bright things gc	music
dark eyes	light bright dark shit backstage alright cause tons trays tons smoke cars tons pain slice skin break heart religion second highs like badminton bathrooms hide flask ba	violence
kotha mukunda	wander garden fence say bring great expense potion guarantee bring relief suffer say exist grasp firmly wrist throw strap torture rack argument mind bend course de	violence
awaz deta hai sola ka din	easier bitch leave happier face upstairs agonna stomp shout door gonna throw laundry hold shotgun dooido want cyrano easier bitch go little miss little miss little m	feelings
jahan mein log sachhe achhc	park bag pack kind heart look comfortable glow porch word come undo face ghost decide inside burn need kill breathe hold hand search breath leave hop someday	sadness
avanee nee pathi	feel take moment feel force veiny point tear hold point point clean stable face stay face fact fine say fine remember head manna hell drug need little think take like	violence
ee jagamantha natika	know baby lung touch weak strong know cope down stay need roses right arm sweet thorns know touch weak strong know cope down stay need touch weak stron	romantic
smoke rings	tell smoke ring blow night circle blue white picture dream fade phantom parade care away night blow blow silky little ring little smoke ring care away night blow blo	music
hum tujhse mohabbat kar ke	fly high sing good bring sing like brother tell wanna cause wanna sing song wanna arm night long wanna know wanna peaceful feelin home summer thunder moonlig	music
paas nahin aaiye haath na la	letter straight heart near apart night write memorize line kiss sign darling read right start letter straight heart	night/time

Implementation:

This strategy implemented using Python programming language. The Natural Language Toolkit (or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. ... NLTK supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities. Scikit-Learn (sklearn library is used for building the models) and numpy library is used as a data structure. We also used Surprise library (Python scikit for building and analyzing recommender systems that deal with explicit rating data).Surprise was designed with the following purposes in mind: Give users perfect control over their experiments. Provide tools to evaluate, analyse and compare the algorithms' performance. Matplotlib library is also used for visualizing the data, models' accuracy,etc.

Our Music Recommender system:

If we need to recommend a song for user number 47

1 . We need to Know which songs the user prefer

	songID	userID	rating
557	9171	47	5
687	2996	47	5
914	49656	47	5
233	27910	47	4
566	73579	47	4

	id	artist_name	track_name	release_date	genre	lyrics	topic
1090	2996	dionne warwick	alfie	1967	pop	moment live sort mean mean kind fool kind gues...	world/life
3121	9171	tears for fears	shout	1985	pop	shout shout things come talk come shout shout ...	violence
4408	12721	the lemonheads	the outdoor type	1996	pop	roof pay rent foot inside tent couldn build sa...	violence
8781	25342	gordon lightfoot	don quixote	1975	country	woodland valley come horseman wild free tilt w...	violence
9563	27910	billy bragg	like soldiers do	1984	country	blue eye fight grey eye fight tear arm teeth w...	violence

This the songs' history data of user

	id	artist_name	track_name	release_date	genre	lyrics	topic
609	1591	roy orbison	crying	1962	pop	alright smile night hold hand tight stop hello...	sadness
2042	6049	michael jackson	take me back	1975	pop	belong baby belong baby think grass green thin...	sadness
6521	18730	twice	ooh-ahhí•˜é²Œ like ooh-ahh	2015	pop	change hoe switch lanes shit look morning cook...	obscene
10504	30941	warren zevon	splendid isolation	1993	country	want live desert want like want live upper eas...	violence
25029	74904	fleetwood mac	seven wonders	1987	rock	long certain place certain time touch hand pat...	world/life

After knowing the nearest users we can recommend for the user like the figure above.

We wanted to evaluate why the user prefers this song type

So we had two choices:

1) to search with topic, therefore we worked on lyrics

```
recommend topics     ||     listen topics
_____________________________
sadness       ||       world/life
sadness       ||       violence
obscene       ||       violence
violence       ||       violence
world/life       ||       violence
```

We found that the user tends to listen to violence topic

So we made an analysis on the lyrics' words

for all songs which user listen

for all songs which recommended

We didn't find any similar words between the listening and recommended songs , which means that the user likes to listen to these songs because of the music genre not the topic.

2) to search with genre

```
recommend genre    ||     listen genre
_____________________________
pop       ||       pop
pop       ||       pop
pop       ||       pop
country       ||       country
rock       ||       country
```

We found that the users like these songs because of its genre, not the topics, so we will consider recommending using the genre not the topic.

Algorithm	test_rmse	fit_time	test_time
KNNBasic	1.440235	0.001277	0.002249
NMF	1.457338	0.087140	0.002070
SVD	1.473553	0.042341	0.002554
NormalPredictor	1.872692	0.001020	0.002799

We used models to help us recommend, we found the highest model is SVD and RMSE is as below

```
RMSE: 1.5015
1.5014610725319713
```

Results & Analysis:

Preprocessing and cleaning:

We made a preprocessing and cleaning using functions to convert the words to lower case, remove punctuation, special characters and stopwords, stemming to remove the (-ing, -ly, ...) and Lemmatisation (for converting the word to the root word ,e.g.. plays-playing-played to play).

Partitioning:

We defined a function to get 191 partitions of the books containing 100 words only and a loop to randomize and save the partition and the label.

```
[ ]  lyrics = [ ]
     topic =  [ ]
     for x in range( len( data ) ):

       if( len( data[x , 0 ] ) >= 100 ):
         token = data[ x , 0 ]
         lyrics.append(    token[ 0: 100]  )
         topic.append(  data[x , 1 ] )

     print(   len ( lyrics   )  )
```

Balancing our data(Down Sampling Majority Class):

We faced the problem of unbalancing the data , so have done the down sampling, and we made all the labels or categories equalized with the label of least number

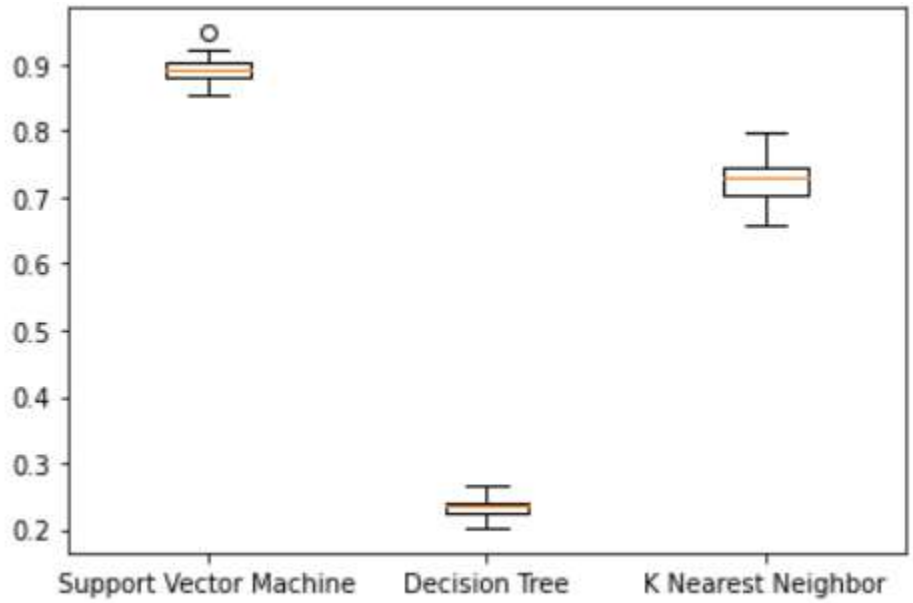
```
# df_minority = df[df.balance==1]

# Downsample majority class
df_violence_majority_downsampled = resample(df_violence,
                                replace=False,    # sample without replacement
                                n_samples= 191 ,     # to match minority class
                                random_state=123 ) # reproducible results

df_sadness_majority_downsampled = resample(df_sadness,
                                replace=False,    # sample without replacement
                                n_samples= 191 ,     # to match minority class
                                random_state=123 ) # reproducible results

df_world_life_majority_downsampled = resample(df_world_life,
                                replace=False,    # sample without replacement
                                n_samples= 191 ,     # to match minority class
                                random_state=123 ) # reproducible results

df_obscene_majority_downsampled = resample(df_obscene,
                                replace=False,    # sample without replacement
                                n_samples= 191 ,     # to match minority class
                                random_state=123 ) # reproducible results

df_music_majority_downsampled = resample(df_music,
                                replace=False,    # sample without replacement
                                n_samples= 191 ,     # to match minority class
                                random_state=123 ) # reproducible results

df_night_time_majority_downsampled = resample(df_night_time,
                                replace=False,    # sample without replacement
                                n_samples= 191 ,     # to match minority class
                                random_state=123 ) # reproducible results


df_romantic_time_majority_downsampled = resample(df_romantic,
                                replace=False,    # sample without replacement
                                n_samples= 191 ,     # to match minority class
                                random_state=123 ) # reproducible results

df_feelings_majority_downsampled = resample(df_feelings,
```

Feature engineering:

Bow and TF-IDF

We did it using Bow (bag of words) and TF-IDF. We calculated bag of words to get the frequency of every word in the text using CountVectorizer() . We also calculated the term-frequency (TF) and the inverse document frequency (idf) using TfidfTransformer().

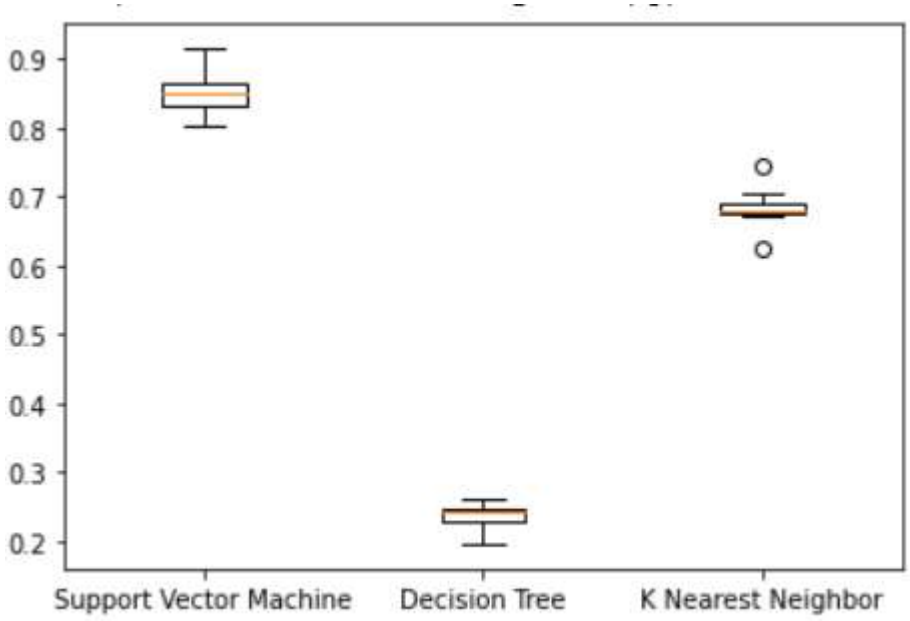
Figure for 10-Fold for (SVM , D_Tree , KNN) with TF-IDF



N-gram

We also used N-gram and calculated it as N-Grams model is one of the most widely used sentence-to-vector models since it captures the context between N-words in a sentence.

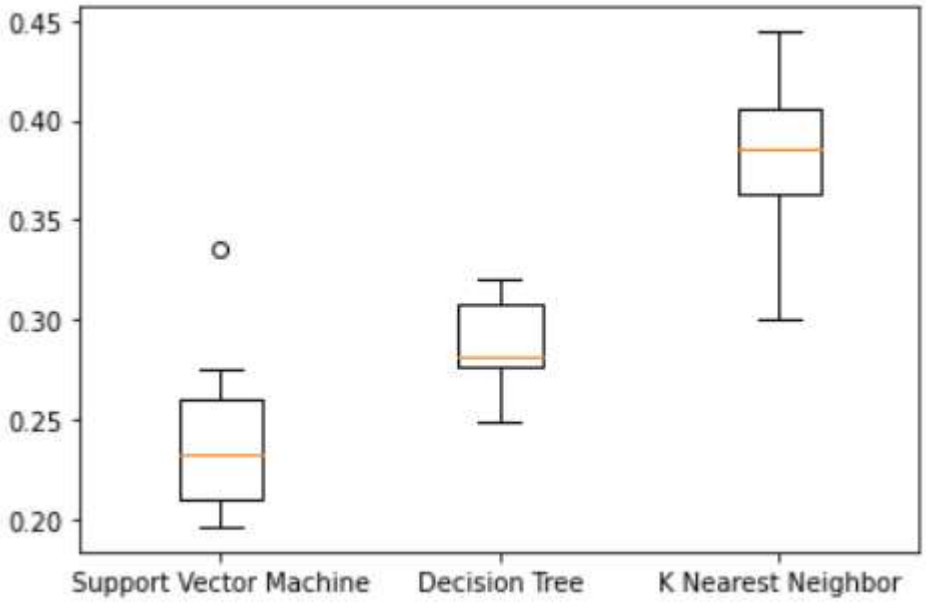
Figure for 10-Fold for (SVM , D_Tree , KNN) with N-gram



LDA (Latent Dirichlet Allocation)

SO as for LDA, we calculated LDA as it is an example of topic model and it is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions. Here we are going to apply LDA to a set of documents and split them into topics

Figure for 10-Fold for (SVM , D_Tree , KNN) with LDA



Binarizing Target

We defined a binarizing function in order to convert the labels into numbers

Because the algorithms can not deal with the text.

For example:

['feelings' , 'music' , 'night/time' , 'obscene' , 'romantic' , 'sadness' , 'violence' , ' world/life']

[0, 1 ,2 ,3 ,4 ,5 ,6 , 7]

Classification technique:

The algorithms:

By using SVM we achieved 90.19%, 19.38% with Decision tree classifier and 72.33% with k-nearest neighbors with tf-idf as depicted in Figure. Table-1, shows accuracy summary per each of the models. Table-2, displays the confusion matrix for each of the models.

Evaluation of Models with TF_IDF

```
def Evaluate_Models(  X_train , y_train , X_test , y_test ):

  model_names = [  "svm"  , "d_tree"   ,     "knn"    ]
  acc=[]

  y_train_binarized = Binarizing_Target( y_train )
  y_test_binarized = Binarizing_Target( y_test )

  y_test_binarized =  [ int (i, base=16) for i in y_test_binarized ]

  #print( y_train_binarized )
  #print(  y_test_binarized )

  models = [ ]
  for x in  range(len(model_names)) :
    models.append( Build_Models( X_train   , y_train_binarized  ,  model_names[x] )   )
    predicted = models[x].predict( X_test )
    predicted = [int(x) for x in predicted]
    acc.append(accuracy_score( y_test_binarized ,   predicted ) )
    print( "Accuracy is     " ,  accuracy_score( y_test_binarized ,   predicted )  )
    confus_mtrix = confusion_matrix(  y_test_binarized ,   predicted )
    print( "Confusion Matrix    \n" , confus_mtrix   )
    # plot_confusion_matrix( model  ,   np.array ( partition_test ) ,   np.array( predicted ) )
    # print(  book_names [  int(predicted[0] )  ]  )
    # np.mean( predicted == book_names_bin  )
    plot_confusion_matrix(  confusion_matrix(   y_test_binarized , predicted )  , unique_labels )
    # print(    compute_confusion_matrix(   y_test_binarized , predicted ) )


  print( "Highest Accuracy Model is : " +   model_names [ np.argmax(acc) ] )
```

Algorithm	Binarized Models Accuracy
SVM	90.19%,
Decision tree	19.38%
K-nearest neighbors	72.33%
Champion Model	SVM

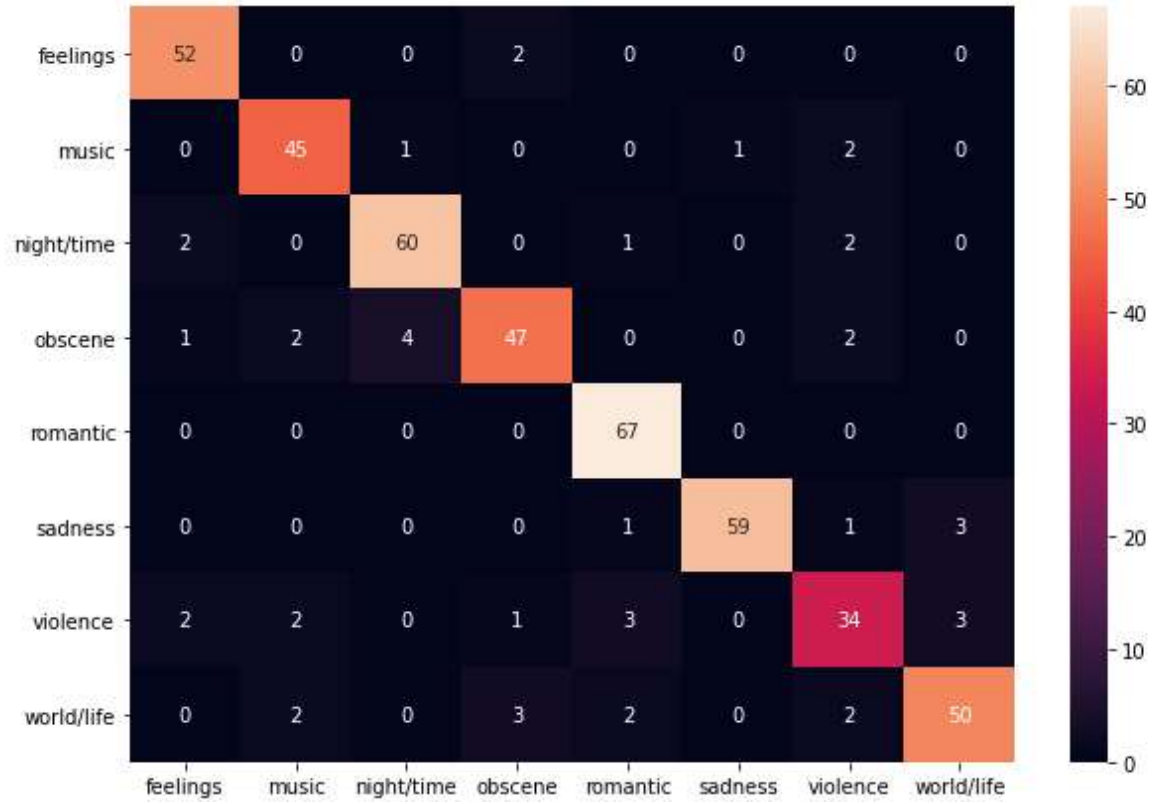


Figure 1: Confusion Matrix for SVM

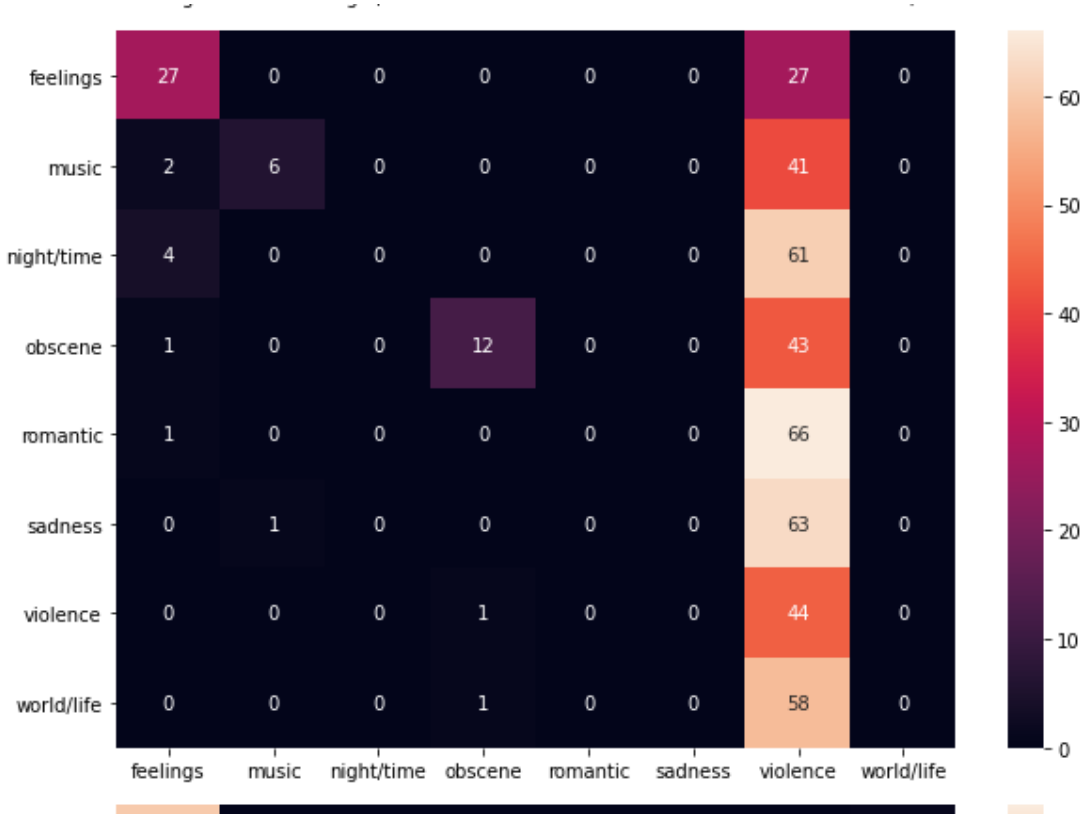


Figure 2: Confusion Matrix for Decision tree

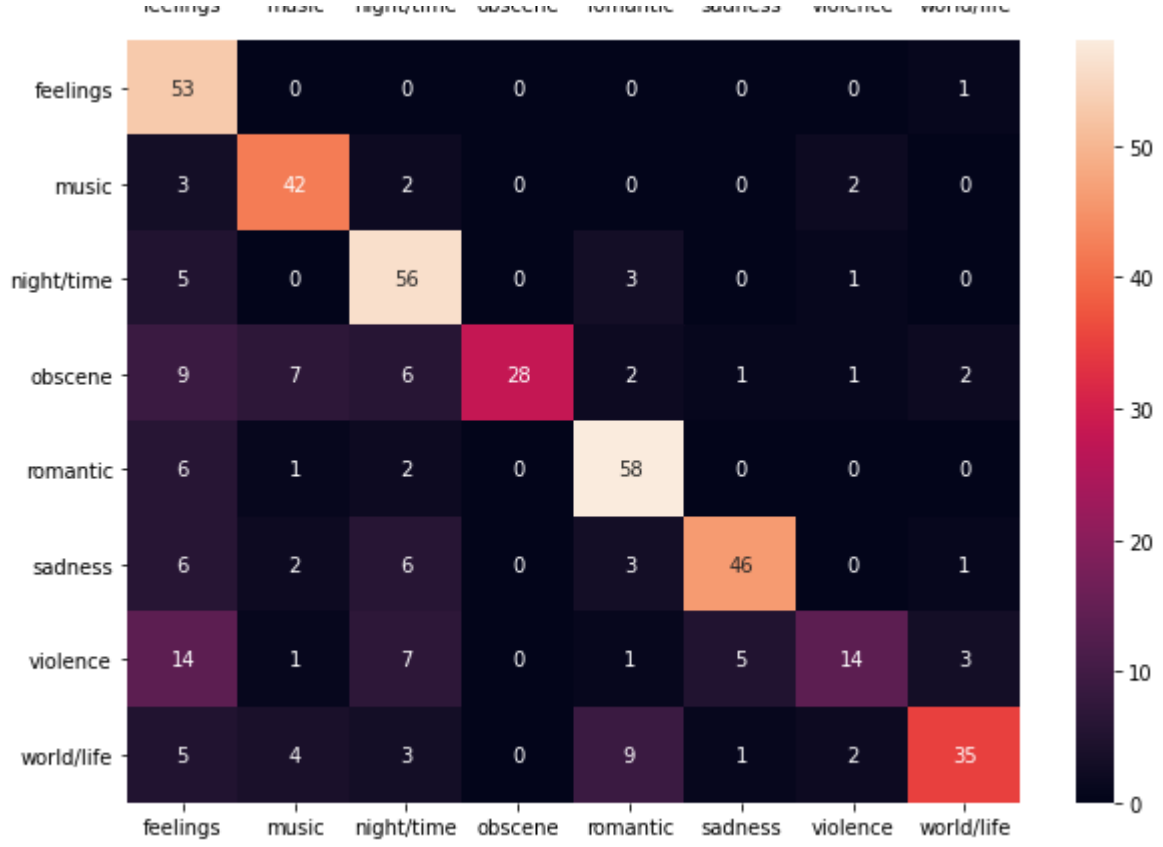


Figure 3: Confusion Matrix for K-nearest neighbors

```
Accuracy is      0.9019607843137255
Confusion Matrix
 [[52  0  0  2  0  0  0  0]
 [ 0 45  1  0  0  1  2  0]
 [ 2  0 60  0  1  0  2  0]
 [ 1  2  4 47  0  0  2  0]
 [ 0  0  0  0 67  0  0  0]
 [ 0  0  0  0  1 59  1  3]
 [ 2  2  0  1  3  0 34  3]
 [ 0  2  0  3  2  0  2 50]]
Accuracy is      0.19389978213507625
Confusion Matrix
 [[27  0  0  0  0  0 27  0]
 [ 2  6  0  0  0  0 41  0]
 [ 4  0  0  0  0  0 61  0]
 [ 1  0  0 12  0  0 43  0]
 [ 1  0  0  0  0  0 66  0]
 [ 0  1  0  0  0  0 63  0]
 [ 0  0  0  1  0  0 44  0]
 [ 0  0  0  1  0  0 58  0]]
Accuracy is      0.7233115468409586
Confusion Matrix
 [[53  0  0  0  0  0  0  1]
 [ 3 42  2  0  0  0  2  0]
 [ 5  0 56  0  3  0  1  0]
 [ 9  7  6 28  2  1  1  2]
 [ 6  1  2  0 58  0  0  0]
 [ 6  2  6  0  3 46  0  1]
 [14  1  7  0  1  5 14  3]
 [ 5  4  3  0  9  1  2 35]]
Highest Accuracy Model is : svm
```

After Evaluation process with 10-Folds Cross Validation:

We found that the accuracy got enhanced for decision tree and K Nearest Neighbour (slightly)

as shown below

Evaluation 10-Fold for (SVM , D_Tree , KNN) with TF_IDF

```
[ ]
    # Models

    svm = SGDClassifier(loss='hinge', penalty='l2',  alpha=1e-3, random_state=42,  max_iter=5, tol=None)
    d_tree = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=3, min_samples_leaf=5)
    knn = KNeighborsClassifier(n_neighbors=7)

    X_train_tfidf  =  Bow_Tf_Idf(  new_all_partitions )

    models = [  svm   , d_tree ,  knn  ]
    model_names = [ "Support Vector Machine" , "Decision Tree"  ,  "K Nearest Neighbor" ]
    Model_scores = [ ]
    All_Model_scores_list = [ ]

    Model_accuracy = [ ]
    Model_std = [ ]

    Models_highest_accuracy = [ ]
    Models_less_std = [ ]
    Model_index = [ ]

    for x in range( 3 ):
      Model_scores.append ( cross_val_score(  models[x] ,   X_train_tfidf ,  new_all_labels ,   cv = 10, scoring= 'accuracy')  )

    for x in range( len( Model_scores ) ) :

      print(       model_names[ x ]  ,   " Scores is  "  , Model_scores[ x ]  )
      print("Accuracy:   ",  (  Model_scores[ x ].mean()  ,   Model_scores[ x ].std()  )   )
      print( "////////////////////////////////////////////// \n ")
      All_Model_scores_list.append(  Model_scores[ x ] )
      Model_accuracy.append(  Model_scores[ x ].mean()   )
      Model_std.append(     Model_scores[ x ].std()  )


    print (  "Champion Model is : " , model_names[ np.argmax( Model_accuracy )  ]  )

    print( Model_accuracy )
    print( Model_std )

    Models_highest_accuracy.append(  np.max( Model_accuracy)  )
    Models_less_std.append( np.min( Model_std ) )
    Model_index.append(  np.argmax( Model_accuracy )   )

    print( All_Model_scores_list )
```

.

```
Support Vector Machine  Scores is   [0.85620915 0.88888889 0.86928105 0.90196078 0.90196078 0.89542484
 0.94771242 0.88235294 0.88157895 0.92105263]
Accuracy:    (0.8946422428620571, 0.024691612608115525)
//////////////////////////////////////////////

Decision Tree  Scores is   [0.24183007 0.22875817 0.20261438 0.26797386 0.22222222 0.24183007
 0.24836601 0.20915033 0.23684211 0.23684211]
Accuracy:    (0.2336429385655316, 0.018074461307620405)
//////////////////////////////////////////////

K Nearest Neighbor  Scores is   [0.73856209 0.69281046 0.74509804 0.71895425 0.7254902  0.69934641
 0.79738562 0.74509804 0.65789474 0.73684211]
Accuracy:    (0.7257481940144479, 0.035496886672720824)
//////////////////////////////////////////////

Champion Model is :  Support Vector Machine
[0.8946422428620571, 0.2336429385655316, 0.7257481940144479]
[0.024691612608115525, 0.018074461307620405, 0.035496886672720824]
[array([0.85620915, 0.88888889, 0.86928105, 0.90196078, 0.90196078,
       0.89542484, 0.94771242, 0.88235294, 0.88157895, 0.92105263]), array([0.24183007, 0.22875817, 0.20261438, 0.26797386, 0.22222222,
       0.24183007, 0.24836601, 0.20915033, 0.23684211, 0.23684211]), array([0.73856209, 0.69281046, 0.74509804, 0.71895425, 0.7254902 ,
       0.69934641, 0.79738562, 0.74509804, 0.65789474, 0.73684211])]
```

Algorithm	Binarized Models Accuracy
SVM	89.46%,
Decision tree	23.36%
K-nearest neighbors	72.57%
Champion Model	SVM

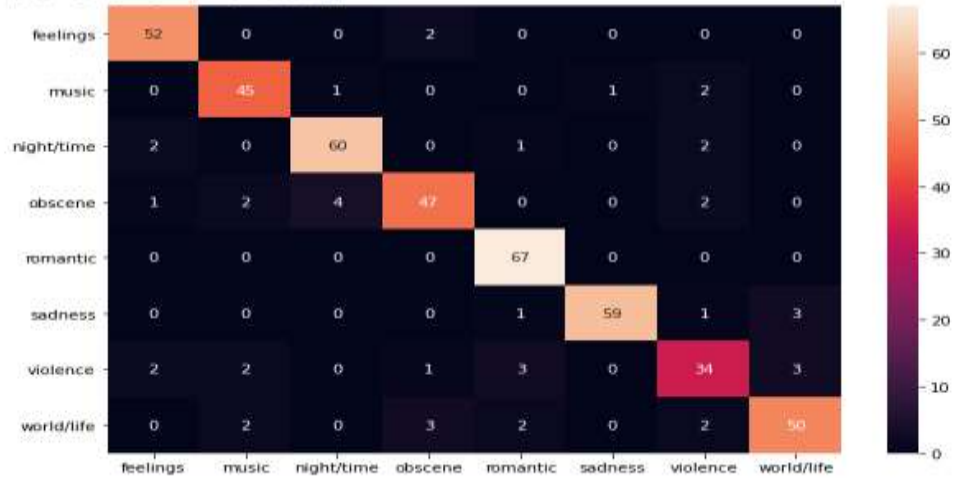
Error analysis:

According to the results after running K-fold on the SVM, DT, and KNN, we found that the highest mean accuracy is the Support Vector Machine. So, we considered the SVM as the champion Model to do our error analysis on.

We defined a function to detect the mis-classified cases then we identified the correct labels for these records.

We trained the SVM (champion model) with our data, and then we computed the accuracy and we found that the acc is 90.19%.

```
Accuracy is      0.9019607843137255
Confusion Matrix
 [[52  0  0  2  0  0  0  0]
 [ 0 45  1  0  0  1  2  0]
 [ 2  0 60  0  1  0  2  0]
 [ 1  2  4 47  0  0  2  0]
 [ 0  0  0  0 67  0  0  0]
 [ 0  0  0  0  1 59  1  3]
 [ 2  2  0  1  3  0 34  3]
 [ 0  2  0  3  2  0  2 50]]
```

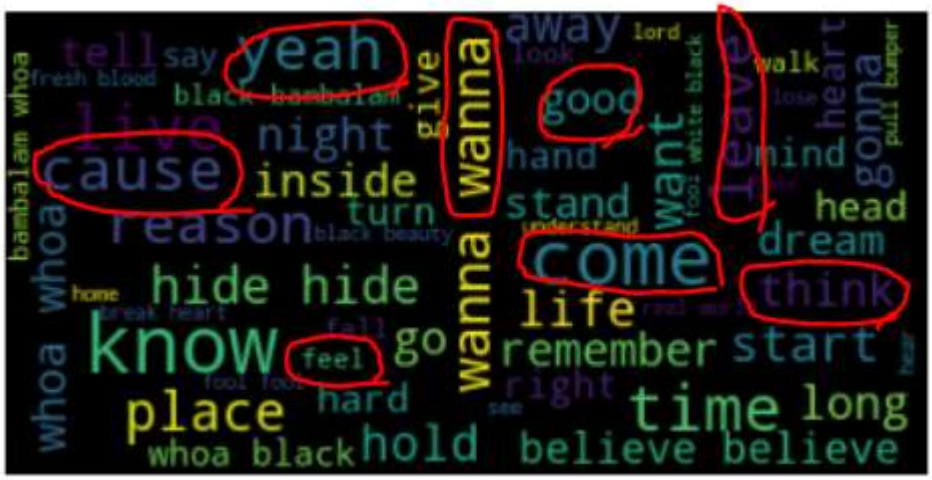


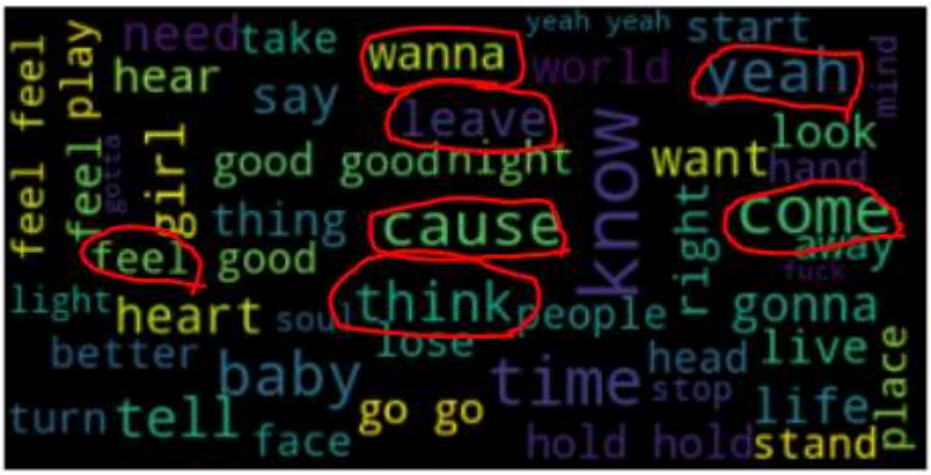
Specifying the characteristics of the records:

For the mis-classified cases, we first calculated the word cloud for these wrong records, we also calculated the word cloud for the predicted wrong books.

```
459
459
Misclassified -  violence
Correct Classified -  world/life
Misclassified -  feelings
Correct Classified -  violence
Misclassified -  feelings
Correct Classified -  obscene
Misclassified -  violence
Correct Classified -  night/time
Misclassified -  night/time
Correct Classified -  obscene
Misclassified -  world/life
Correct Classified -  violence
Misclassified -  night/time
Correct Classified -  obscene
Misclassified -  feelings
Correct Classified -  night/time
Misclassified -  world/life
Correct Classified -  violence
Misclassified -  romantic
Correct Classified -  sadness
Misclassified -  romantic
Correct Classified -  violence
Misclassified -  world/life
Correct Classified -  sadness
Misclassified -  romantic
Correct Classified -  violence
Misclassified -  feelings
Correct Classified -  violence
Misclassified -  romantic
Correct Classified -  night/time
Misclassified -  music
```

Analysing error partitions by drawing the word cloud





Our Analysis:

Here in the previous two figures of word cloud for the data and the misclassified partition. We found that there are many similar words in both the misclassified partition and in the wrong labels, so the algorithm assigns the partition to the wrong category based on the high similarity of features.

Clustering technique :

Building K_means Model with TF_IDF , N_grams , LDA:

Plotting K_means with TF_IDF features

```
[ ]
    Build_K_means( X_TF_IDF_features_2D )
```

```
kappa = -0.05833956619296932
silhouette_score = -0.35240188048452253
```

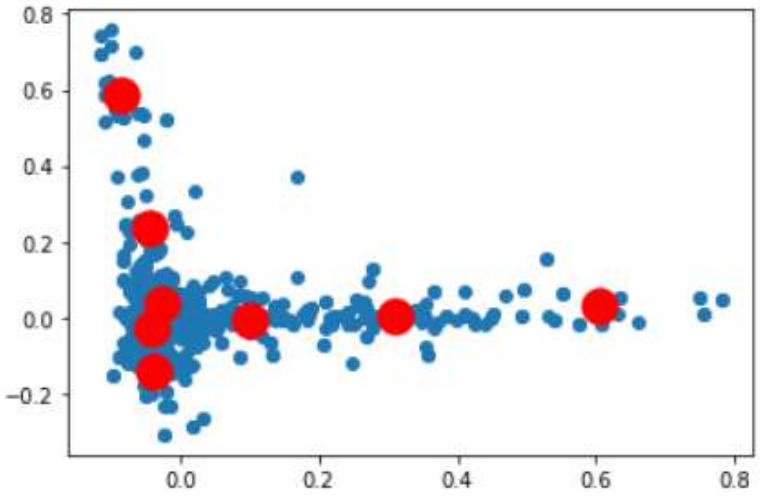


Figure 1: Plotting K-means with Tf-idf

Plotting K_means with N_Grams features

```
Build_K_means( X_N_Gram_features_2d )
```

```
kappa = 0.014958863126402377
silhouette_score = -0.5086819117253478
```

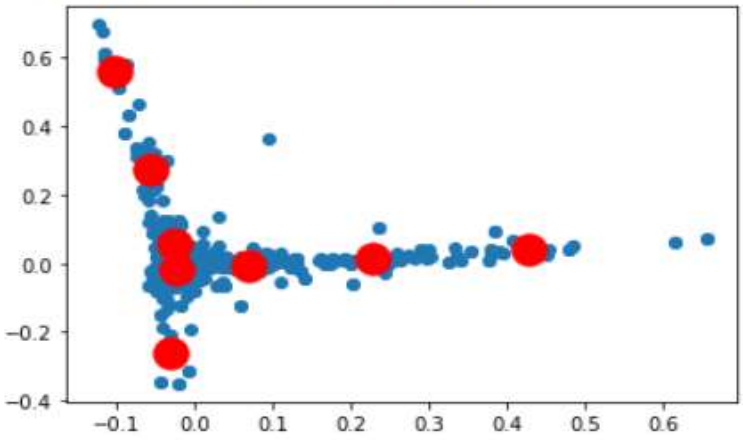


Figure 2: Plotting K-means with N-gram

Plotting K_means with LDA features

```
Build_K_means( X_LDA_features_2d )
```

```
kappa = -0.0777860882572925
silhouette_score = 0.5454046317477705
```

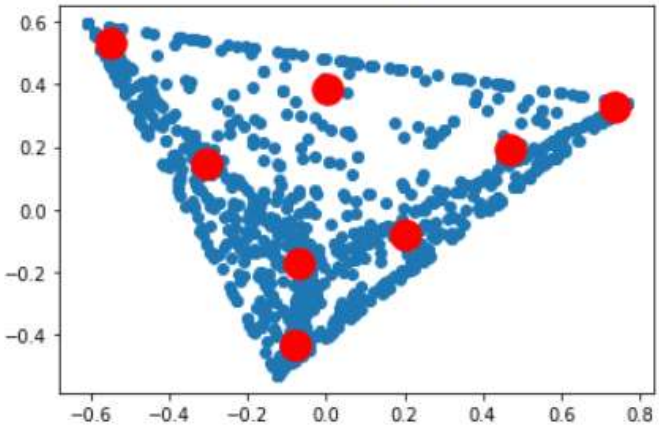


Figure 3: Plotting K-means with LDA

Building EM (Guassian Mixture) Model with TF_IDF , N_grams , LDA

Figure 1: Plotting Hierarchical with Tf_idf

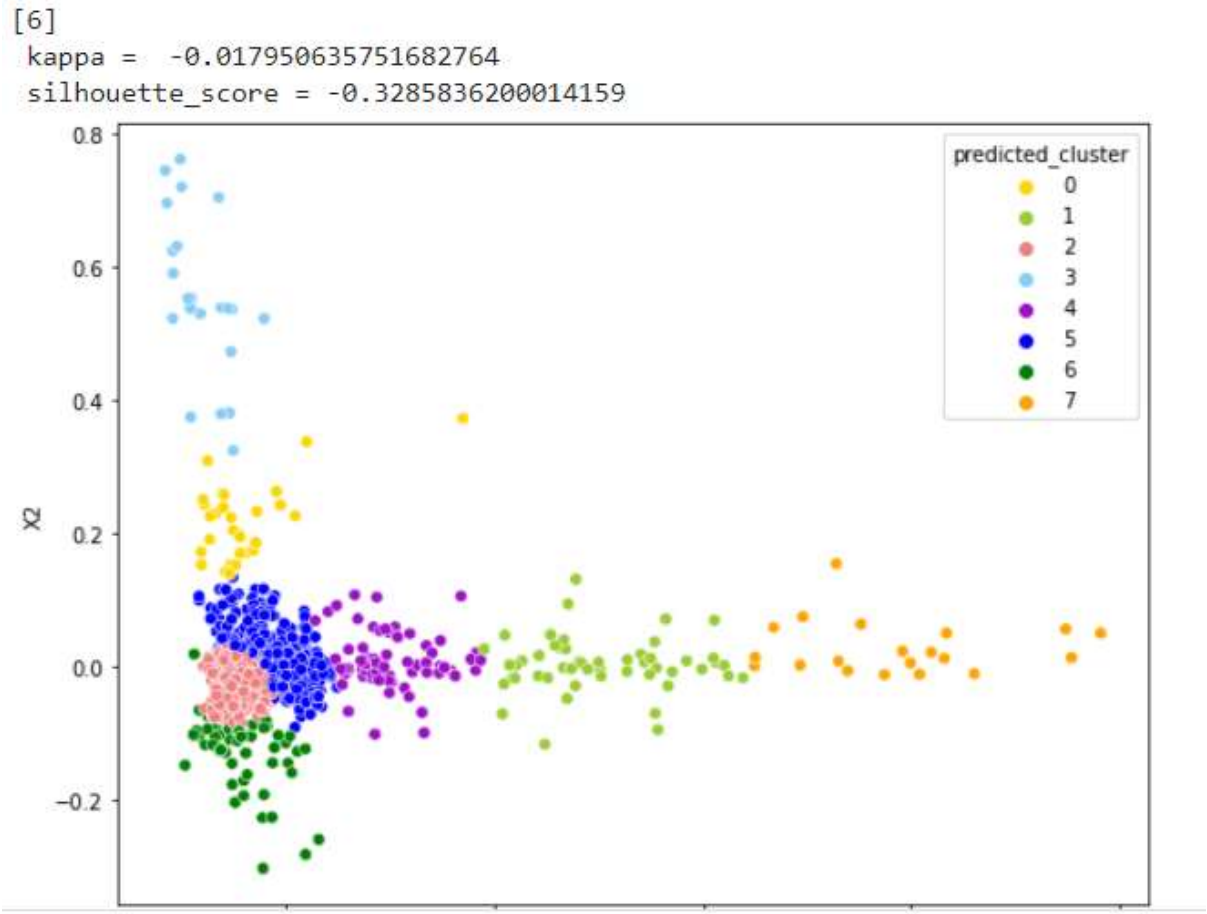


Figure 2: Plotting Hierarchical with N-gram

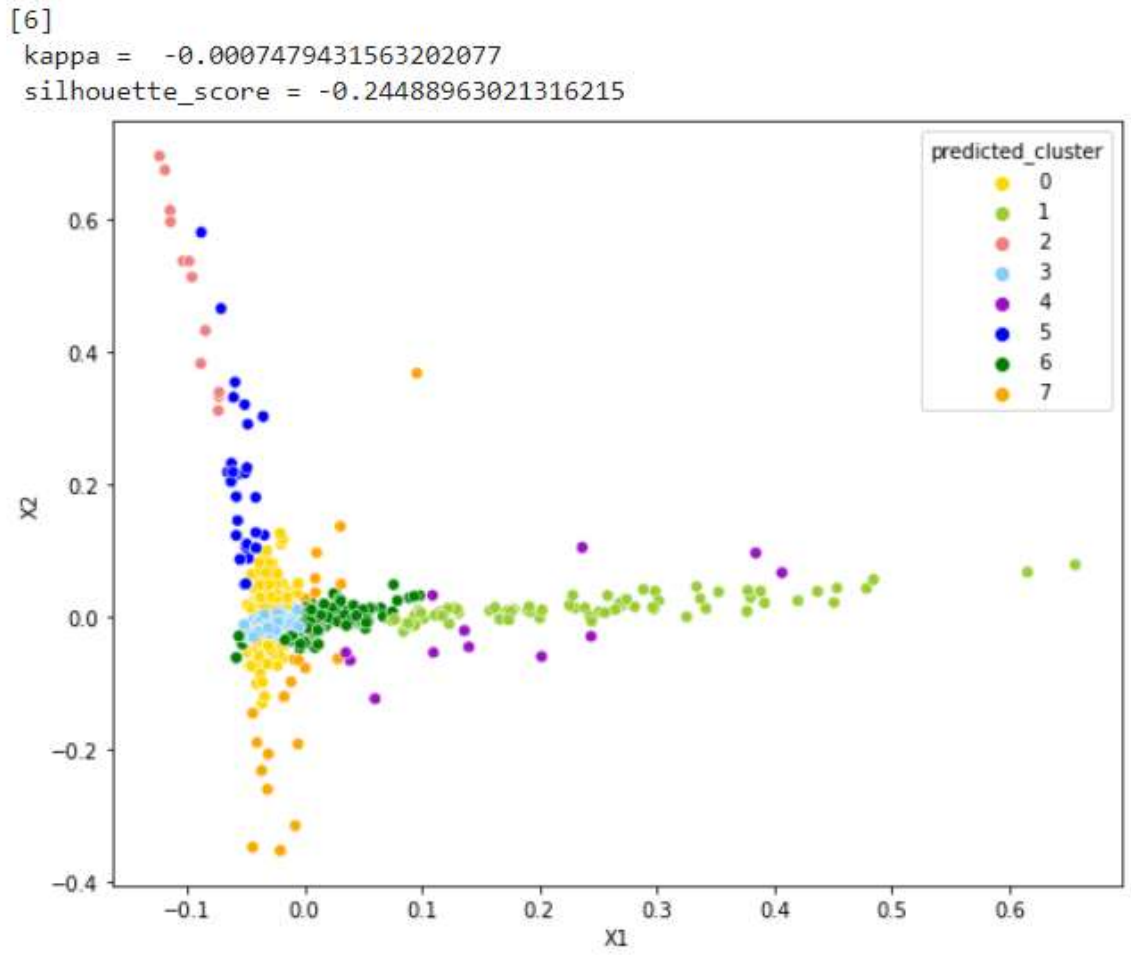
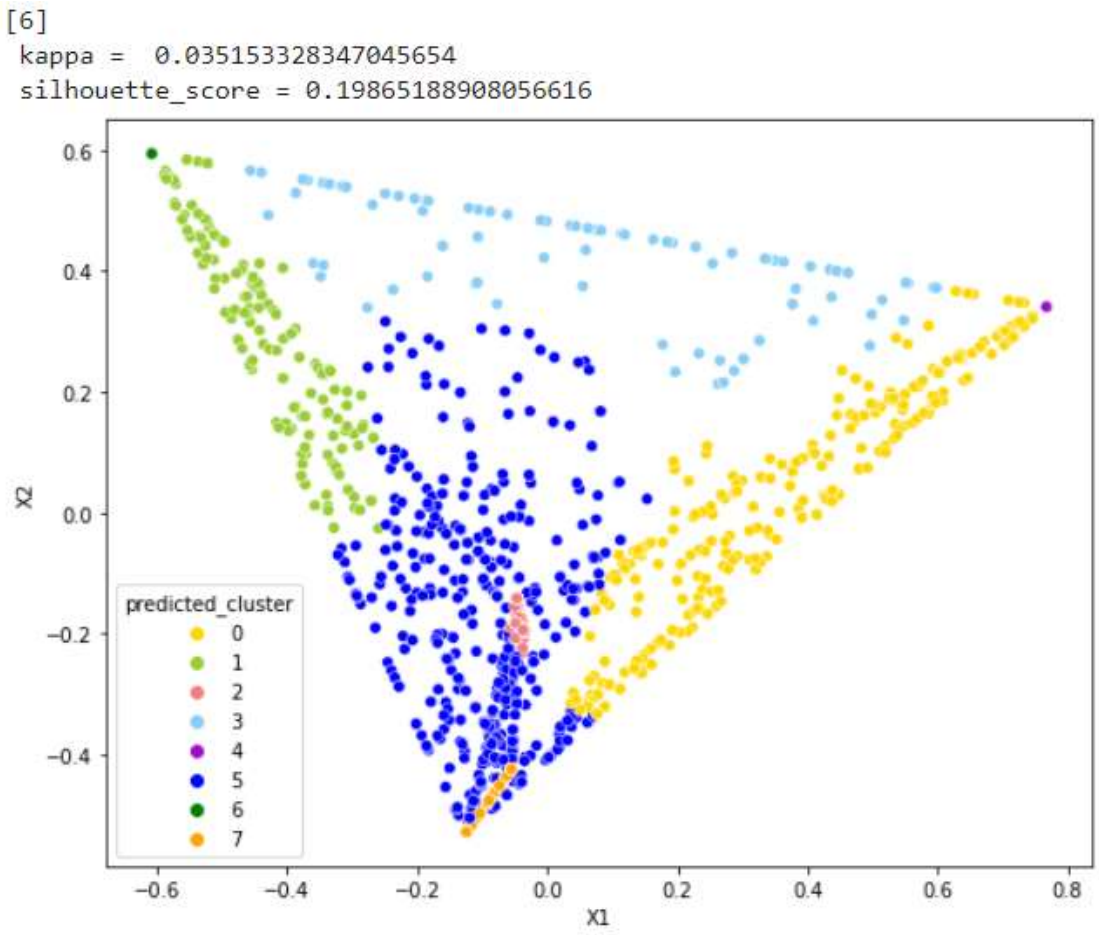


Figure 3: Plotting Hierarchical with LDA



Building Hierarchical Model with TF_IDF , N_grams , LDA

Figure 1: Plotting Hierarchical with Tf_idf

```
kappa =  -0.03814510097232615
silhouette_score = -0.4555188355120005
```

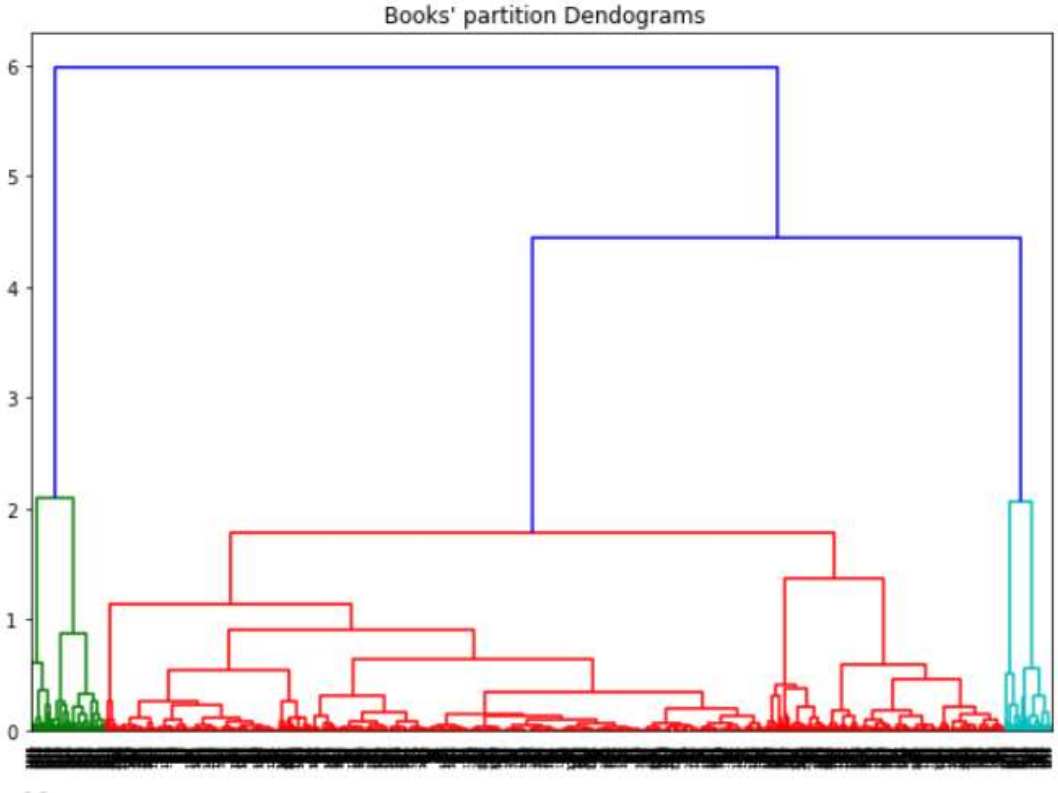


Figure 2: Plotting Hierarchical with N-gram

```
kappa =  -0.08676140613313388
silhouette_score = -0.5106944888748393
```

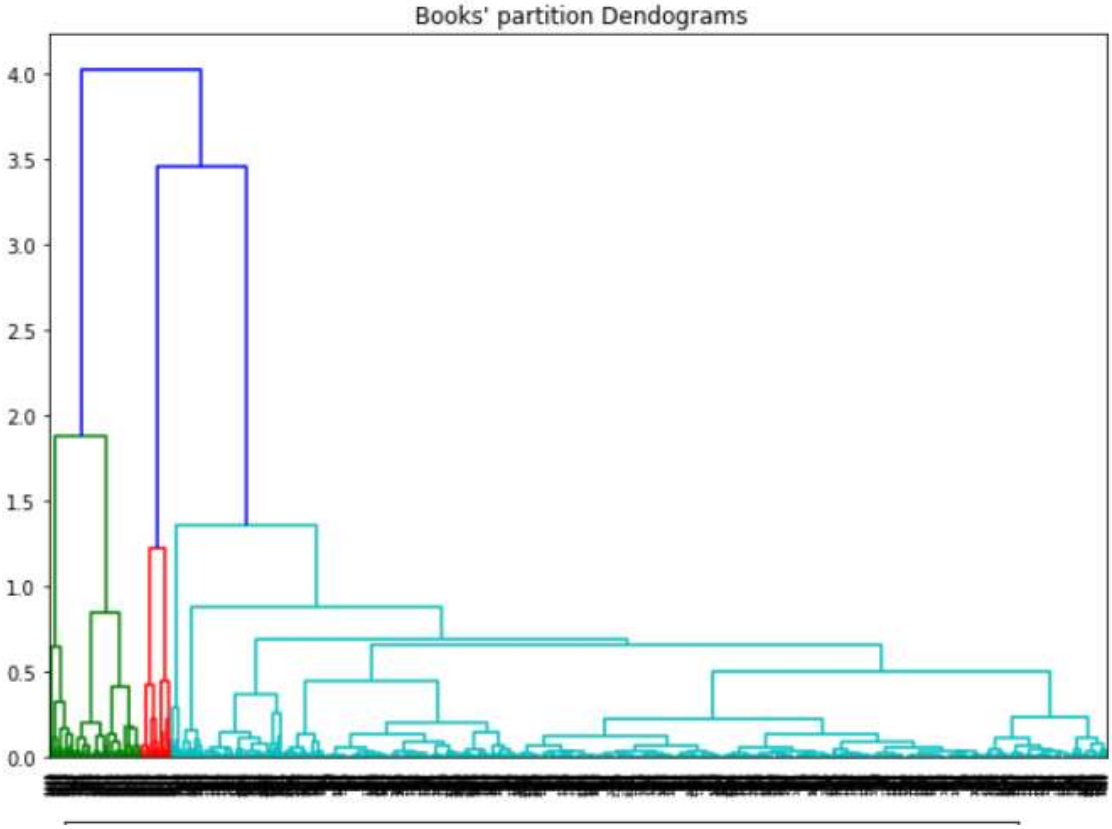
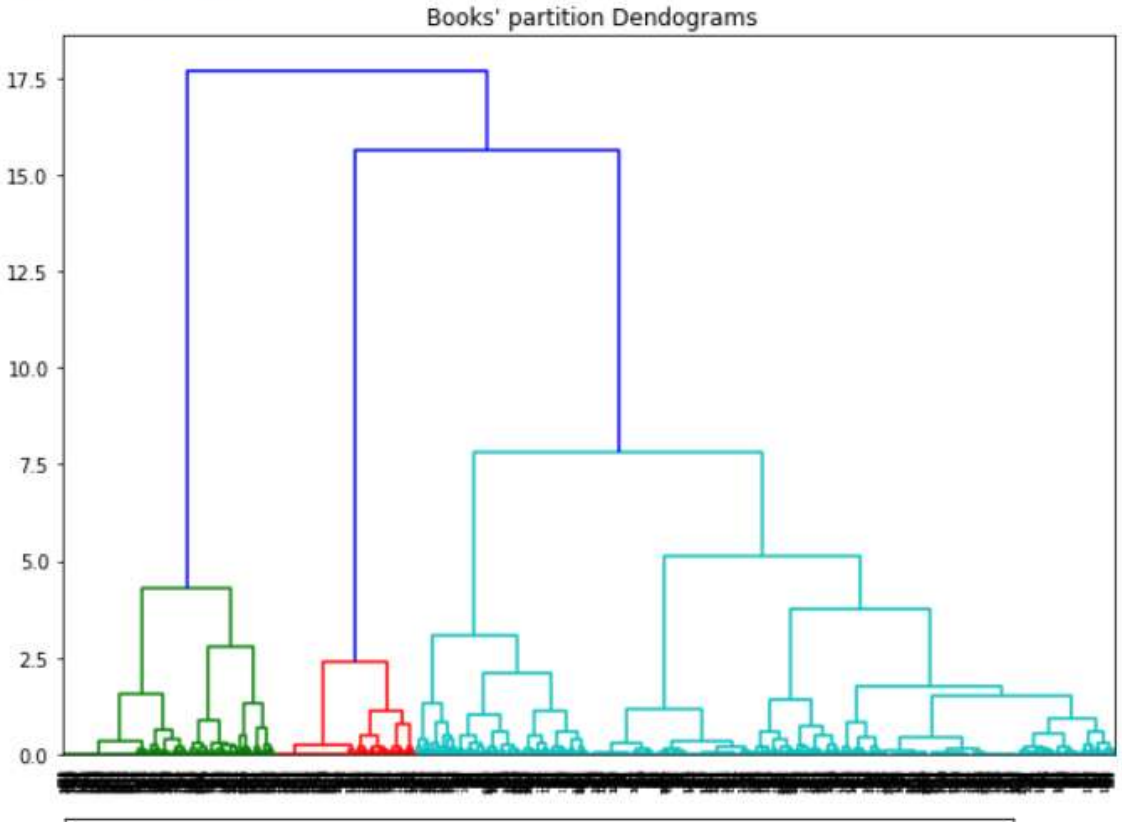


Figure 3: Plotting Hierarchical with LDA

```
kappa =  -0.010471204188481575
silhouette_score = 0.4674776644022645
```



Models Evaluation:

Kappa evaluation

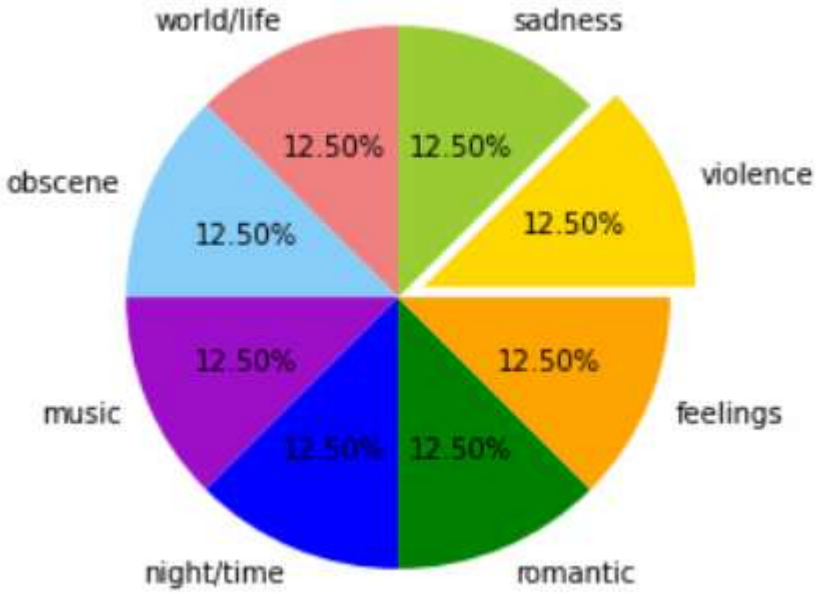
Model	BOW+Tf_idf	N-gram	LDA
K-means	-0.058339	0.0149588	-0.077786
EM	-0.017950	-0.000748	0.0351533
Hierarchical	-0.038145	-0.086761	-0.010471

Silhouette evaluation

Model	BOW+Tf_idf	N-gram	LDA
K-means	-0.352402	-0.508681	0.545404
EM	-0.3285836	-0.244889	0.1986518
Hierarchical	-0.455518	-0.510694	0.467477

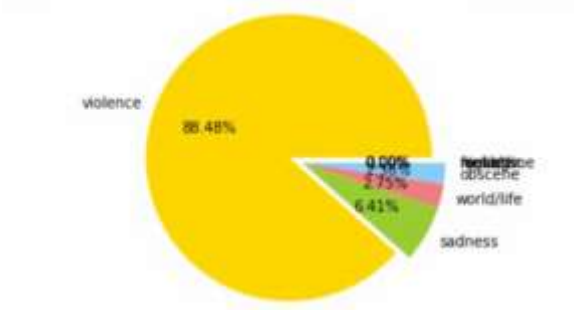
According to the results after computing the Silhouette and Kappa. We found that the highest kappa and Silhouette is the EM with LDA features. So, we consider the EM as the champion model to do our error analysis on.

Error analysis:

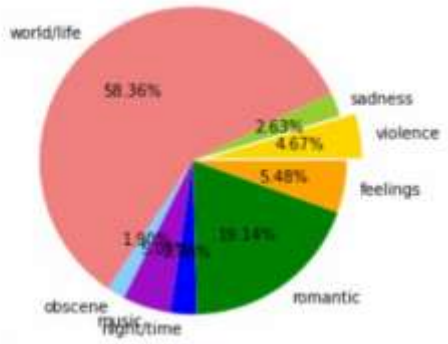


This figure are the actual ones that should be predicted

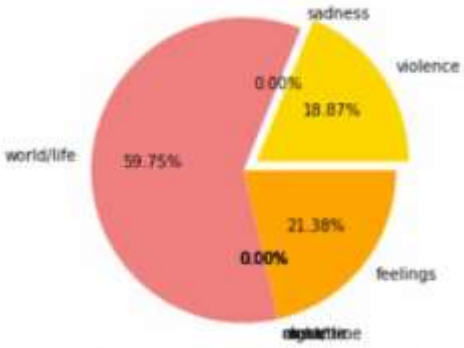
These figures about what is predicted and the error and corrected ones .




```
model predict violence Error  1352
 model predict s sadness Error  98
model predict world/life Error  42
model predict obscene Error  36
model predict music Error   0
model predict night/time Error   0
model predict romantic Error   0
model predict feelings Error   0
```




```
model predict violence Error  64
 model predict s sadness Error  36
model predict world/life Error  799
model predict obscene Error  26
model predict music Error   69
model predict night/time Error   38
model predict romantic Error   262
model predict feelings Error   75
```




```
model predict violence correct   30
model predict in sadness correct   0
model predict world/life correct   95
model predict in obscene correct   0
model predict in music correct   0
model predict in night/time correct   0
model predict in romantic correct   0
model predict in feelings correct  34
```

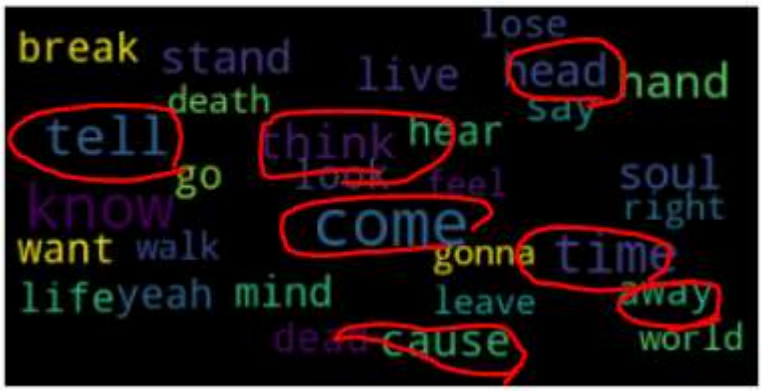
From the previous figures, our clustering algorithms are not able to cluster the data very well that's because there are high similarities between the partition's words

---

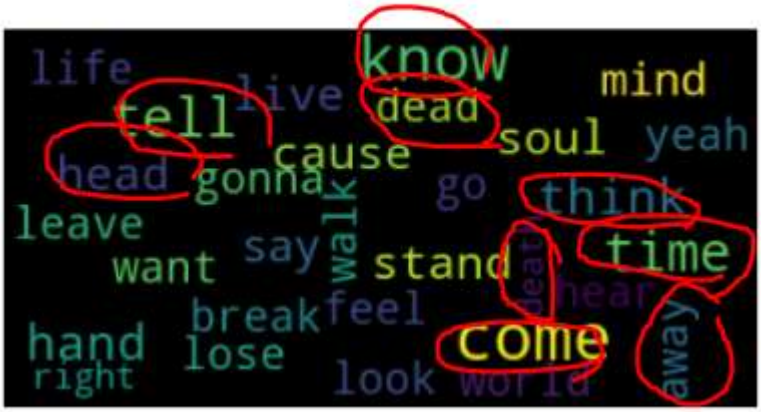
These figures below show the most 30 frequent words for the books to know why it confused the machine and there was error in prediction.

Most frequent 30 words for violence

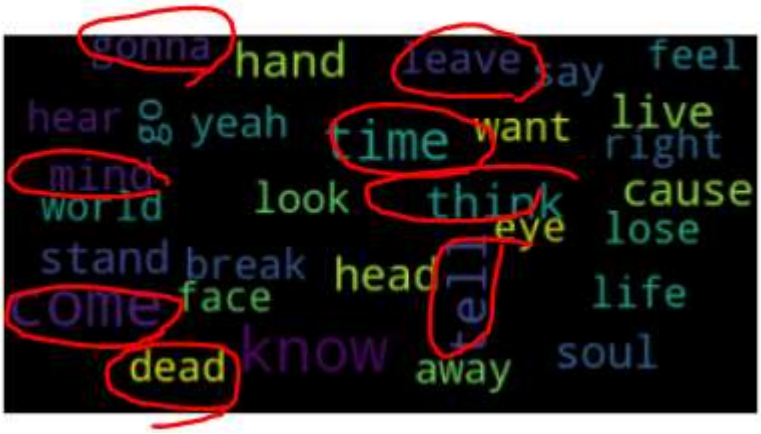
Most frequent 30 words for sadness



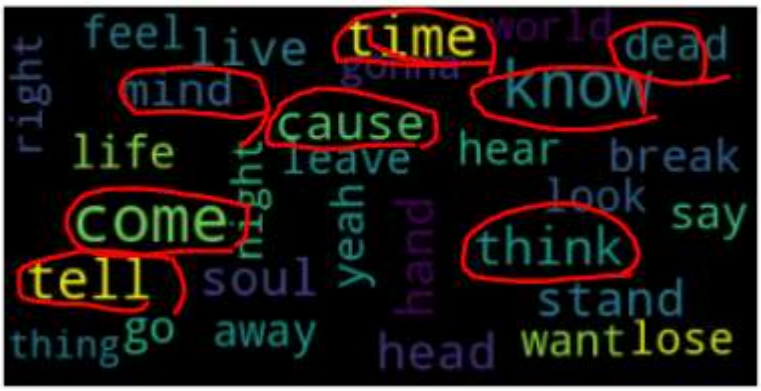
Most frequent 30 words for world/life



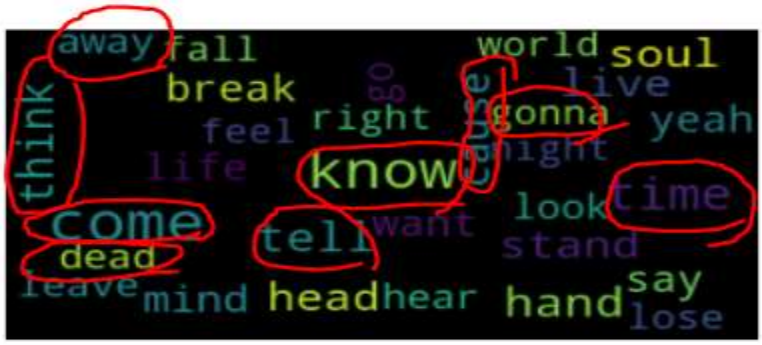
Most frequent 30 words for obscene



Most frequent 30 words for music

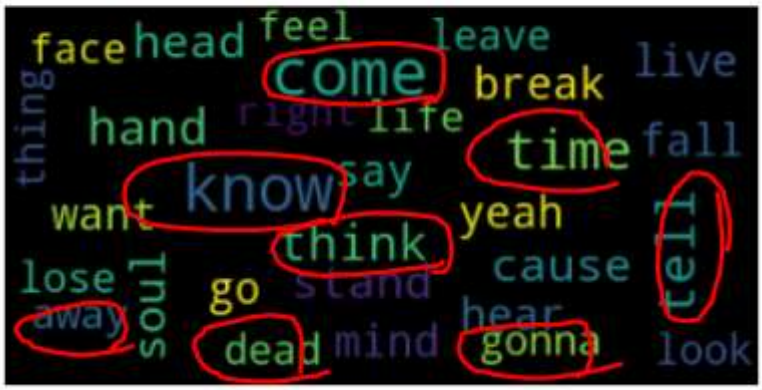


Most frequent 30 words for night/time



Most frequent 30 words for romantic

Most frequent 30 words for feelings



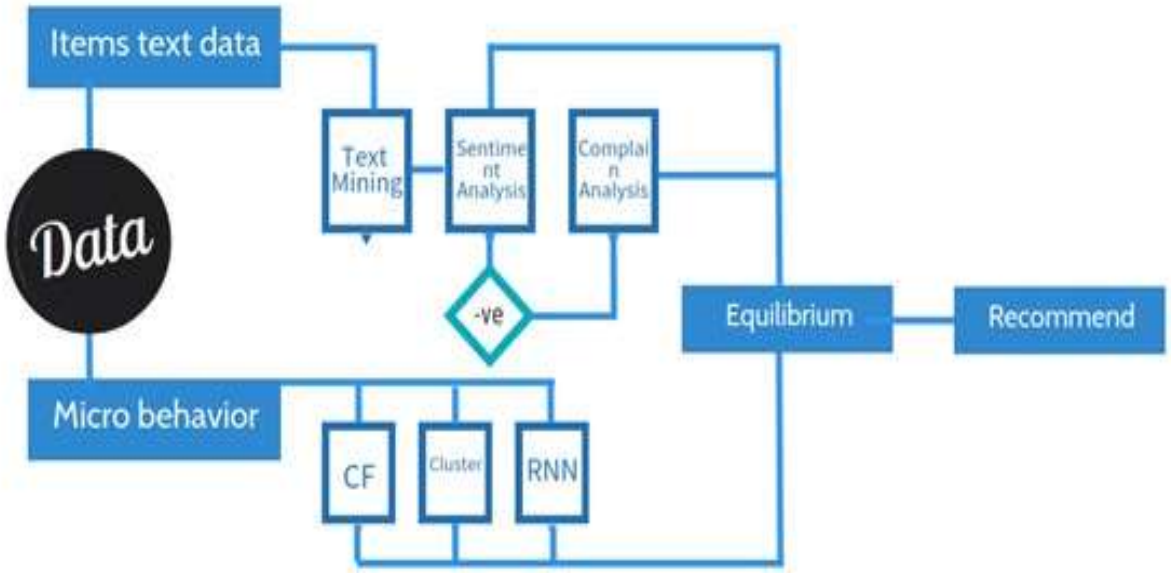
Analyzing our error

By analyzing our wrong music records and finding the top 30 frequent words in each of them, we found that there is high similarities between the music records so the model couldn't cluster the music records in the right cluster due to this similarity.

Contribution (Innovativeness)

Our contribution is using text mining in recommending the suitable or favourite song for the suitable user. We used text mining to evaluate and understand the text and could understand the relation between the users and the song types. This technique is used with the help of the macro behaviour like rating for the song. This will help achieving best recommendation for the user.

Here is our contribution in the following diagram with future work



Conclusion:

We implemented the steps (preparing till verifying and validating) by using Classification, clustering and recommendation techniques. We also found the champion model to do our error analysis.