NLP Project

Venkata Satya Sai Ajay Daliparthi

August 31, 2020

1 Part -1

we imported the data file using the Pandas library and done basic operations on the dataset. to understand the data set we used df.head() and df.shape to find out the number of rows and columns. We have done some basic text mining operations like tokenization, removal of stop words and removing symbols.

1. **Word Frequency:**: no of times a word is repeated in all songs. We calculated word frequency by FreqDist(text) method. The top 17 most repeated words in all songs are :
   [('nt', 178827), ('love', 93952), ('know', 72503), ('like', 63570), ('re', 62692), ('go t', 61047), ('oh', 59744), ('ll', 59360), ('na', 48451), ('one', 44703), ('ve', 44219), ('go', 43861), ('time', 43544), ('get', 43455), ('baby', 41743), ('see', 41023), ('wa nt', 39832)]
   By using matplotlib library we plotted the graph of above data.

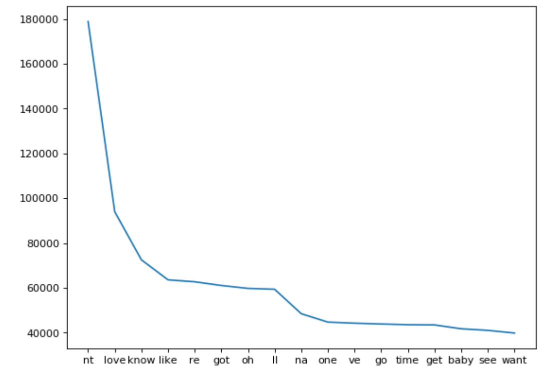


Figure 1: Most frequently occured words

By using wordcloud library we generated wordcloud for all the songs data.

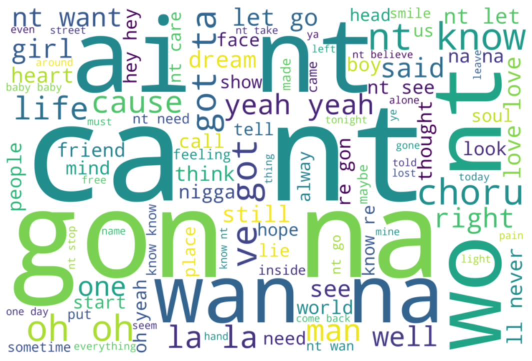


Figure 2: word cloud of 100 most frequent words

2. **Word Length:** Lexical Diversity: no of unique word presented in the data divided by total no of words.
We used different indices to calculate lexical diversity [1] like
Type-Token Ratio (TTR):

$$TTR = (noofuniquewords)/(totalnoofword)$$

Type Token Ratio :0.006774961037238452

Herdan's C(logTTR):

$$C = \log(noofuniquewords)/\log(totalnoofwords).$$

LogTTR(Herdan's C) :0.6970806496231016

Guiraud's Root TTR(R):

$$R = (noofuniquewords)/\sqrt{(totalnoofwords)}$$

Guiraud's Root TTR :25.77635406573193

Carroll's Corrected TTR(CTTR):

$$CTTR = (noofuniquewords)/\sqrt{(totalnoofwords)}$$

.
Carroll's Corrected TTR :18.22663475414448
Summer's index(s):

$$S = log(log(noofuniquewords))/log(log(totalnoofwords))$$

Summer's index :0.8712444655213627

3. **Lexical density :**
   Lexical density is defined as percentage no of unique tokens divied by total no of words.
   LD = (unique tokens/total tokens) * 100
   Lexical Density :0.6774961037238453

4. **Bigram and Trigram analysis:**
   No of double and triple word pairs that occurred most in the total data are called as Bigrams and Trigrams.
   Top 20 Bigrams are :
   [("'ve", 'got'), ("n't", 'know'), ("'ll", 'never'), ("n't", 'want'), ("'re", 'gon'), ('little', 'bit'), ('yeah', 'yeah'), ("n't", 'care'), ('brand', 'new'), ('new', 'york'), ('close', 'eyes'), ('feel', 'like'), ('far', 'away'), ('set', 'free'), ('every', 'day'), ("'ve", 'seen'), ('say', 'goodbye'), ('come', 'back'), ("n't", 'let'), ('hey', 'hey')]

   Top 20 Trigrams are :
   [('got', "'ve", 'got'), ('since', "'ve", 'got'), ("'ve", 'got', "n't"), ('know', "'ve", 'got'), ("'ve", 'got', 'long'), ("'ve", 'got', "'ve"), ("'ve", 'got', 'known'), ("'ve", 'got', 'seen'), ("n't", "'ve", 'got'), ("'ve", 'got', 'nothing'), ("'ve", 'got', 'lot'), ('things', "'ve", 'got'), ("'ve", 'got', 'something'), ("'ve", 'got', 'feeling'), ("'ve", 'got', 'money'), ('think', "'ve", 'got'), ("'cause", "'ve", 'got'), ("'ve", 'got', 'far'), ("'ve", 'got', 'much'), ("'ve", 'got', 'nothin')]

5. **Sentiment analysis:**
   we performed sentiment analysis of each song and presented the over all results below with help of bar plot.
   positive = 74.42844752818733
   neutral = 1.0910667823070253
   negative = 24.480485689505638

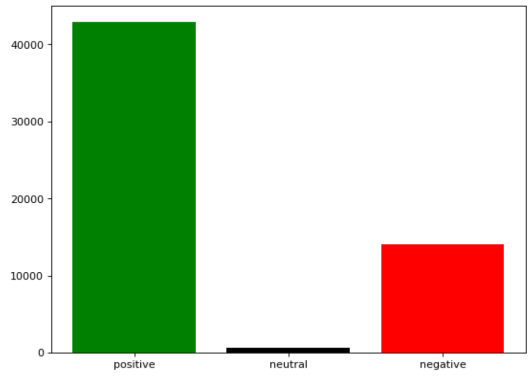


Figure 3: Sentiment analysis of all songs

6. **TF-IDF :**
   Term Frequency :
   The number of times a word appears in a document divided by the total number of words in the document. [2]

$$TF_i[i,j] = n_i[i,j] / \sum_k^n {}_i[i,j]$$

   Inverse Data Frequency (IDF):
   The log of the number of documents divided by the number of documents that contain the word w. Inverse data frequency determines the weight of rare words across all documents in the corpus.

$$IDF_i[w] = \log_i[N/dft]$$

$$TF - IDF = TF * IDF$$

   By using sklearn's TfidfVectorizer class we calculated TF-IDF values of one song.(refer code)
   The words are: ['about', 'all', 'and', 'at', 'be', 'believe', 'blue', 'can', 'could', 'do', 'ever', 'face', 'feel', 'fellow', 'fine', 'for', 'girl', 'go', 'hand', 'her', 'holds', 'hours', 'how', 'if', 'in', 'it', 'just', 'kind', 'leaves', 'll', 'look', 'lucky', 'makes', 'me', 'means', 'mine', 'my', 'of', 'on', 'one', 'park', 'plan', 'sees', 'she', 'smiles', 'something', 'special', 'squeezes', 'talking', 'that', 'the', 'things', 'to', 'walk', 'walking', 'way', 'we', 'what', 'when', 'who', 'without', 'wonderful']
   •As there are 57650 songs in the dataset its hard to present results for all the songs so we choose one song to perform TF-IDF.

2 Part -2

A language model learns to predict the probability of a sequence of words. There are two ways in which we can implement this model.
1. Statistical models : N-grams, Hidden Markov Models and certain rules to learn the probability distribution of words.
2. Neural language models : neural network models are proved to be more effective.They use different kinds of Neural network to model the language.

To perform the type ahead prediction model we selected the Trie data structure to store all the song titles.Trie data structure is proved to support fast operations of type-ahead predictions. [3] Trie structures creates the nodes in a way that words and strings can be retrieved from the structure by traversing down the tree.
The time complexity of tree data structure to run a type ahead prediction, insert a word and delete a word from tree is given as:

$$T = O(L * N)$$

where
L = Average length of all the word.
N = Total number of words.
The worst case time complexity for creating a trie structure is depended on total no of words and lenth of them.

$$T = O(M * N)$$

where
M = longest word.
N = Total number of words.
In the trie data structure, we store the data in the following way shown below.

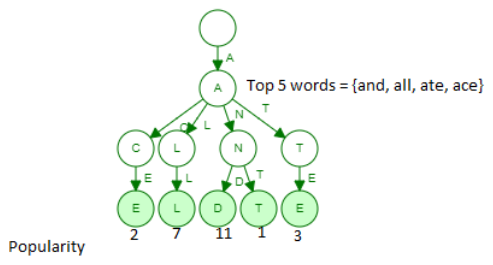
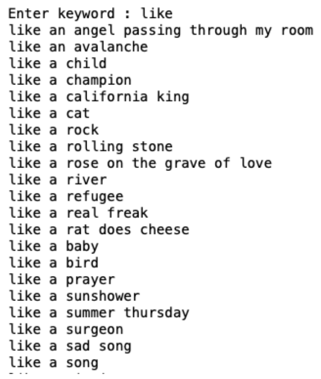


Figure 4: sample Trie data structure implementation

Step-1: at first we created an empty list named keys in which we iteratively added all the song titles. Before adding titles in the list we lowered the song titles.
Step-2: Implemented a trie data structure class that contains different methods to create a Trie data structure and to return autosuggestions for a given prefix.
Step-3: create a trie with all the songs data presented in the keys list.
Step-4:prompt user to give input and store the input string into the variable named key for which we need to return predictions.
Step-5: pass the key into the AutoSuggestions method in the trie data structure class and print out all the suggestions for the given key.
Results:



```
Enter keyword : like
like an angel passing through my room
like an avalanche
like a child
like a champion
like a california king
like a cat
like a rock
like a rolling stone
like a rose on the grave of love
like a river
like a refugee
like a real freak
like a rat does cheese
like a baby
like a bird
like a prayer
like a sunshower
like a summer thursday
like a surgeon
like a sad song
like a song
```

Figure 5: sample of results generated from our model

Note: Before running the code please make sure that you have the code and data set are in same directory.

References

[1] Kalantari, Reza Gholami, Javad. (2017). *Lexical Complexity Development from Dynamic Systems Theory Perspective: Lexical Density, Diversity, and Sophistication. International Journal of Instruction. 10. 1-18. 10.12973..*

[2] Ramos, Juan. *"Using tf-idf to determine word relevance in document queries." Proceedings of the first instructional conference on machine learning. Vol. 242. 2003.*

[3] Willard, Dan E. *"New trie data structures which support very fast search operations." Journal of Computer and System Sciences 28.3 (1984): 379-394.*