Big Data Framework, Master Data Science, 2017/2018

Report: Free Style Hip-hop Lyrics Generator

Yu LI; Fubang ZHAO; Xiao PAN

yu.li@telecom-paristech.fr
fubang.zhao@telecom-paristech.fr
xiao.pan@telecom-paristech.fr

1 Introduction

In the project, we implemented an automatic hip-hop lyrics generator. To simplify the task, we generate the lyrics line by line, instead of word by word, which means we form the result by selecting the lines that are already in the data set. The user is required to provide the first line of lyrics, and our program will generate the rest of the lyrics.

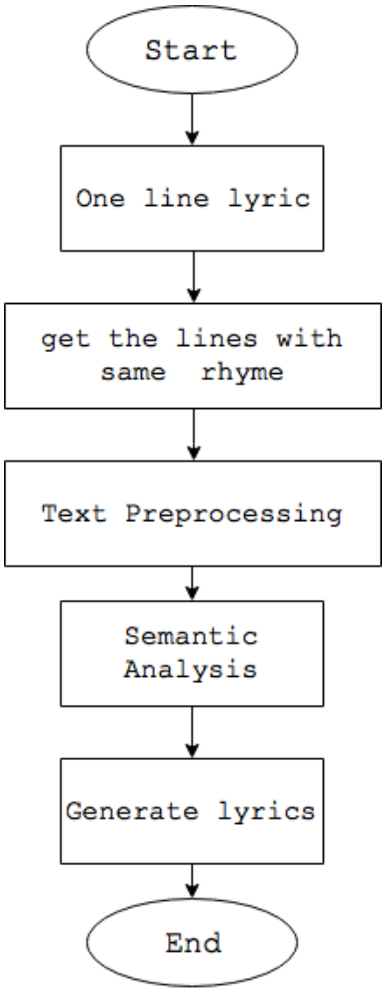
We used the Apache Spark framework. Apache Spark is an open-source cluster-computing framework designed for Hadoop, and can handle iterative and interactive Map Reduce jobs.

Our data set is taken from https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics/data, we only keep the lyrics with label 'hip-hop' of the field 'genre'. The 'hip-hop' dataset includes 33965 songs.

The procedure contains 3 steps. Pronounce extraction will be presented in part 2, then we will introduce the process of text preprocessing. And finally we introduce the PLSA algorithm that we used for semantic analysis, which is based on iterative Map Reduce.

In the end of the report, one example of the lyrics generated by our program will be presented.

The procedure is listed in the following flowchart.



2 Pronounce Extraction

In our project, we have three parts to complete to realise the function of prediction of the next line of lyrics: rhyming, the structure of sentence and the semantic part. In this part, we process the problem of rhyming and structure. Especially, in our project, we use the type 'Hip-Hop' as an example.

For dealing with the problem of rhyme, we search a lot of modules online. In the beginning, we used the module whose name is 'e-speak'. However, it is kind of out-of-date and we cannot import it in databricks. So we decided to use the module 'Pronoucing' which is made by the CMU university just two years ago and has more functions. Here is the address of the document of this module: http://pronouncing.readthedocs.io/en/latest/index.html

(i) First of all, we split the lyrics line by line into the array of lines. So we got something like:

```
+-----+--------------------+----+------+-------+--------------------+
|index|                song|year|artist|  genre|              lyrics|
+-----+--------------------+----+------+-------+--------------------+
|  249|          i-got-that|2007|eazy-e|Hip-Hop|[(horns)..., (cho...|
|  250|       8-ball-remix|2007|eazy-e|Hip-Hop|[Verse 1:, I don'...|
|  251|extra-special-thankz|2007|eazy-e|Hip-Hop|[19 muthaphukkin ...|
|  252|     boyz-in-da-hood|2007|eazy-e|Hip-Hop|[Hey yo man, reme...|
|  253|          automoblie|2007|eazy-e|Hip-Hop|[Yo, Dre, man, I ...|
+-----+--------------------+----+------+-------+--------------------+
```

(ii) Then, we separated the array into multiple rows, so we can get a table with all the lines and each sentence is a row. so the result will be like:

```
+-----+----------+----+------+-------+--------------------+
|index|      song|year|artist|  genre|              lyrics|
+-----+----------+----+------+-------+--------------------+
|  249|i-got-that|2007|eazy-e|Hip-Hop|          (horns)...|
|  249|i-got-that|2007|eazy-e|Hip-Hop|            (chorus)|
|  249|i-got-that|2007|eazy-e|Hip-Hop|Timbo- When you h...|
|  249|i-got-that|2007|eazy-e|Hip-Hop|           (verse 1)|
|  249|i-got-that|2007|eazy-e|Hip-Hop|Lil Eazy E- Lemme...|
+-----+----------+----+------+-------+--------------------+
```

(iii) Then, we took use of the user-defined-function to pass the function we have written into the withColumn function so that we could get the new dataframe with the column of rhyme for each line(Here we define that the number of rhyming is 3. If the sentence was shorter than 3 syllables, we filled it with blank space):

```
+-----+-----------+----+------+-------+--------------------+-------+
|index|       song|year|artist|  genre|              lyrics|phoneme|
+-----+-----------+----+------+-------+--------------------+-------+
|  249| i-got-that|2007|eazy-e|Hip-Hop|          (horns)...|       |
|  249| i-got-that|2007|eazy-e|Hip-Hop|            (chorus)|       |
|  249| i-got-that|2007|eazy-e|Hip-Hop|Timbo- When you h...| AHAWUW|
|  249| i-got-that|2007|eazy-e|Hip-Hop|           (verse 1)|       |
|  249| i-got-that|2007|eazy-e|Hip-Hop|Lil Eazy E- Lemme...| AWAHAY|
|  249| i-got-that|2007|eazy-e|Hip-Hop|They not on the g...| IHIHIY|
|  249| i-got-that|2007|eazy-e|Hip-Hop|            (chorus)|       |
|  249| i-got-that|2007|eazy-e|Hip-Hop|Timbo- When you h...| AHAWUW|
|  249| i-got-that|2007|eazy-e|Hip-Hop|(uh huh, uh huh, ...| AHAHAH|
|  249| i-got-that|2007|eazy-e|Hip-Hop|(verse 2, Timbaland)|       |
|  249| i-got-that|2007|eazy-e|Hip-Hop|Timbo- lemme tell...| EHUWIY|
|  249| i-got-that|2007|eazy-e|Hip-Hop|            (chorus)|       |
```

(iv) In the end, for example, we took **'Today I would like to show you a rap music'** as the first line. With consideration with the similar length of sentence(which we consider as the structure of sentence), we got the lines below which are rhyming with the first line that we wrote:

```
+------------------------------------------------------------------
|I gotta hear that fat nigga do it
|There go fonzerelli I'm feelin that mans music
|C'mon, baby, we can do this
|Big hogs, tryin to pile the money up out your trash, you dig?
|Lucky charms, stay armed, niggas won't clap you in
|I'll be gentle, baby, 'cause we can do this
|Niggas is foolish. Middle man is out here scoopin that new shit.
|I woulda bust you if you came out on that goon shit
|I'm here with my man Whoo Kid
|Ain't nothing quite as beautiful as Music
|Haters say they wanna roll y'all we can do this
|Right on a muthfucka and draw down on his ass music
|But wait why pretend if you can move in
|Ain't no such thing as too thick
|Sto con gli ippopotami sto con Al Bano, Putin?
|It's a jack move bitch
|Can we do it? we can do it, we shall do it!
|Crazy as they come I'm not the one, we can do this
|Take it to the head or the face wrong place you can do it
|New wrist, knew that, knew this
```

Until now, we have found all the potential sentences which have the similar structures and the exactly same rhymes.

3 Preprocessing

We get the candidate lines of lyrics, and do some preprocessing to these texts.

(i) First step: tokenization

Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens , By using RegexTokenizer, we can also at the same time throwing away certain characters, such as punctuation. Here in the code, we delete all the characters except number 09 and $azAZ$.

```
regex_tokenizer = RegexTokenizer(inputCol='lyrics',\
outputCol='words', pattern='\\W')
regex_df = regex_tokenizer.transform(df)
```

(ii) Second step: stop words removerSometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words . We remove those words because these words can be distracting, non-informative (or non-discriminative) and are additional memory overhead.

```
from pyspark.ml.feature import StopWordsRemover
remover = StopWordsRemover(inputCol='words', outputCol='tokens')
stopWords = ['a','an','the', 'is','are', 'for', 'hi',\
```

```
'in', 'on','row','lyrics','u','t','s','re','i','m']
remover.setStopWords(stopWords)
tokens_filtered = remover.transform(regex_tokenized_counts)
```

(iii) Third step: Combination - recording word combinations (n-grams)

An n-gram is a contiguous sequence of n words. Adding features of higher n-grams can be helpful in identifying that a certain multi-word expression occurs in the text.Here in the project, we use 2-grams.

```
from pyspark.ml.feature import NGram
ngram = NGram(n=2, inputCol='tokens', outputCol='2grams')
my_2ngrams =ngram.transform(cleanDF)
```

```
+--------------------------------------------------------------------------------------------------------+
|2grams                                                                                                  |
+--------------------------------------------------------------------------------------------------------+
|[collapse relax, relax then, then get, get right, right back, back with, with it]                       |
|[from wearing, wearing lipstick, lipstick to, to smoking, smoking chronic, chronic at, at picnics]      |
|[so can, can spit, spit game, game at, at this, this trick]                                             |
|[now when, when hit, hit that, that switch, switch im, im bouncin]                                      |
|[nah don, don be, be fuckin, fuckin wit, wit them, them fat, fat bitches]                               |
|[it perfect, perfect time, time magic, magic trick]                                                     |
|[listen to, to way, way man, man spin, spin it]                                                         |
|[looking sun, sun man, man now, now activist]                                                           |
|[niggaz blackin, blackin out, out like, like eclipse]                                                   |
|[this perfect, perfect time, time magic, magic trick]                                                   |
```

4 Topic model for semantic analysis — PLSA

In our project, the one line of lyric represent one document.

Probabilistic latent semantic analysis (PLSA),is a statistical technique for the analysis of text topic.

PLSA is implemented by EM algorithm, which can be implemented by iterative MapReduce framework: E-step correspond to the Map phase, and M-step correspond to the Reduce phases.

The basic idea of PLSA is to treat the words in each document as observations from a mixture model where the component models are the topic word distributions. The selection of different components is controlled by a set of mixing weights. Words in the same document share the same mixing weights.

For a lyrics document collection D = d_1,...,d_N , each occurrence of a word w belongs to W = w_1,...,w_M . Suppose there are totally K topics, the topic of document d is the sum of the K topics, i.e.$p(z = 1|d)$,$p(z = 2|d)$,...,$p(z = K|d)$ and $\sum_k p(z = k|d) = 1$. In other words, each document may belong to different topics. Every topic z is represented by a multinomial distribution on the vocabulary.

To implement the algorithm, first we need to launch a Map Reduce job to count the number of word j in document i for each (i,j) pair. Then we need a set of iterative Map Reduce jobs for the PLSA EM algorithm. The core formula used to update the distribution is listed below:

(i) E-step:

$$\hat{P}(z = k|d = i, w = j) = \frac{P(w = j, z = k|d = i)}{\sum_k P(w = j, z = k|d = i)} = \frac{P(w = j|z = k)P(z = k|d = i)}{\sum_k P(w = j|z = k)P(z = k|d = i)}$$

```
def _E_step_(self):
        """
        update the latent viariable: p(z|w,d)
        :return: None
        """
        probility_word_given_topic = self.probility_word_given_topic
        k = self.k

        def update_probility_of_word_topic_given_word(doc):
            topic_doc = doc['topic']
            words = doc['words']

            for (word_index,word) in words.items():
                topic_word = word['topic_word']
                for i in range(k.value):
                    topic_word[i] =
                        probility_word_given_topic.value[i,word_index]*topic_doc[i]
                #normalization
                topic_word /= np.sum(topic_word)
                word['topic_word'] = topic_word # added
            return {'words':words,'topic':topic_doc}

        self.data = self.data.map(update_probility_of_word_topic_given_word)
```

(ii) M-step:

$$\hat{P}(w=j|z=k) = \frac{\sum_{i=1}^{N} n(d=i,w=j)P(z=k|d=i,w=j)}{\sum_{i=1}^{N}\sum_{j=1}^{M} n(d=i,w=j)P(z=k|d=i,w=j)}$$

$$\hat{P}(z=k|d=i) = \frac{\sum_{j=1}^{M} n(d=i,w=j)P(z=k|d=i,w=j)}{n(d=i,w=j)}$$

where $n(d=i,w=j)$ means the occurrence of word j in document i.

```
def _M_step_(self):
        """
        update: p(z=k|d),p(w|z=k)
        :return: None
        """
        k = self.k
        v = self.v

        def update_probility_of_doc_topic(doc):
            """
            update the distribution of the documents of the themes
            """
            topic_doc = doc['topic'] - doc['topic']
            words = doc['words']
            for (word_index,word) in words.items():
                topic_doc += word['count']*word['topic_word']
            topic_doc /= np.sum(topic_doc)
```

```
            return {'words':words,'topic':topic_doc}

        self.data = self.data.map(update_probility_of_doc_topic)

        self.data.cache()

        def update_probility_word_given_topic(doc):
            """
            up date the distribution of the words of the themes
            """
            probility_word_given_topic = np.matrix(np.zeros((k.value,v.value)))

            words = doc['words']
            for (word_index,word) in words.items():
                probility_word_given_topic[:,word_index] +=
                    np.matrix(word['count']*word['topic_word']).T

            return probility_word_given_topic

        probility_word_given_topic =
            self.data.map(update_probility_word_given_topic).sum()
        probility_word_given_topic_row_sum =
            np.matrix(np.sum(probility_word_given_topic,axis=1))

        #normalization
        probility_word_given_topic =
            np.divide(probility_word_given_topic,probility_word_given_topic_row_sum)

        self.probility_word_given_topic =
            self.sc.broadcast(probility_word_given_topic)
```

The output of the algorithm is two matrices:

(i) A($N \times K$) where entry (i,k) represent the weight of topic k in document i.

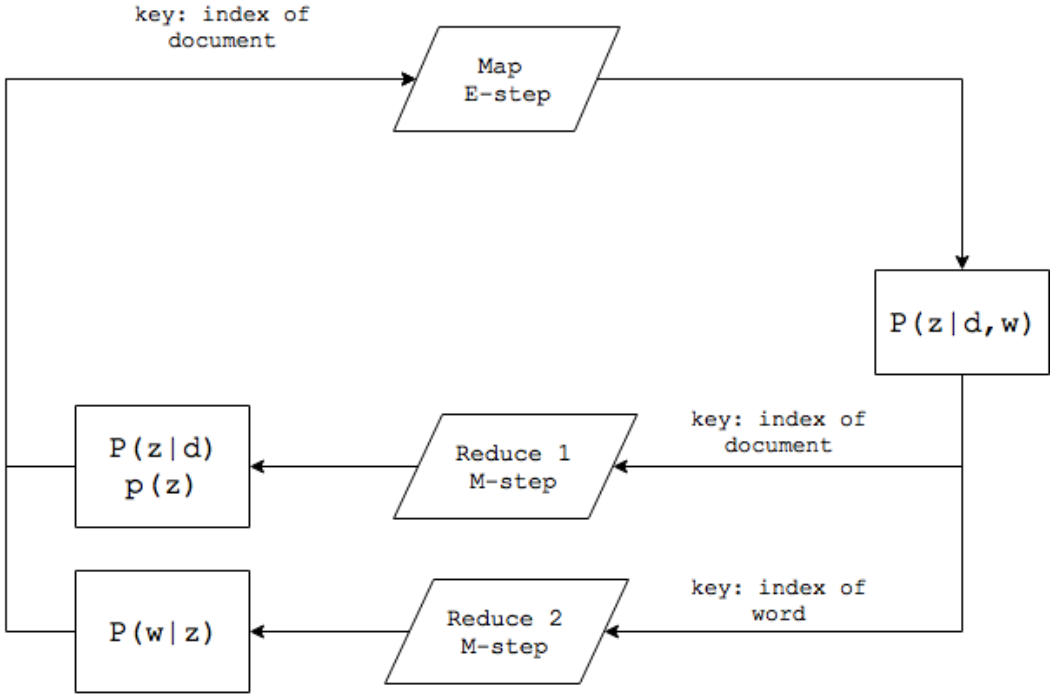
(ii) B($M \times K$) where entry (j,k) represent in topic k, the probability that word j occurs.

```
def save(self):
        """
        save the result of the model TODO
        :param f_word_given_topic: distribution of words given the topic
        :param f_doc_topic: distribution of topic given the documents
        :return:
        """
        doc_topic = self.data.map(lambda x:' '.join([str(q) for q in
            x['topic'].tolist()])).collect()
        probility_word_given_topic = self.probility_word_given_topic.value

        word_dict = self.word_dict_b.value
        word_given_topic = []
        for w,i in word_dict.items():
            word_given_topic.append('%s %s' %(w,' '.join([str(q[0]) for q in
                probility_word_given_topic[:,i].tolist()])))
```

```
    return word_given_topic, doc_topic
```

The flowchart of the iterative Map Reduce Algorithm:



The result in Matrix A can be regarded as a mapping of each line of lyric to a K-dimension feature space. Our task thus becomes: find the "closest" line of lyrics in the feature space.

After specifying the number of lines wanted, the program will output the lines with the most similar semantic meaning in the total dataset.

```
plsa = PLSA(data=gramrdd,sc=sc,k=5,max_itr=10,is_test=True)
plsa.train()
word_given_topic, topic_given_doc = plsa.save()
topic_given_doc_1 = [x.split(" ") for x in topic_given_doc]
topic_given_doc_2 = [[float(y) for y in x]for x in topic_given_doc_1]

some_pt = topic_given_doc_2[0]
min_index = distance.cdist([some_pt], topic_given_doc_2)[0].argsort()

print test_lyric
for i in min_index[1:6]:
    print(dfl[i])
```

5 Conclusion

The final result of our project is as following. Given a line of lyrics, it will generate the 5 most appropriate next line.

```
Given line: Today I would like show a freestyle rap lyrics
generated line:
Row(lyrics=u"Put it on the catalogs homie, Classics' 'CRIP!!")
Row(lyrics=u'I see this bad Spanish chick,')
Row(lyrics=u'Shorty was sick because he loved the hustle more than his bitch.')
Row(lyrics=u'I raise my hand, my past exist')
Row(lyrics=u'Ima do me doin my back in it')
```

Due to the limits of time and ability, we were unable to implement the "word-by-word" generator. However, the "line-by-line" generator is a simplified version where we can learn the general principle of Map Reduce framework and Natural Language Processing.

References

[1] DopeLearning: A Computational Approach to Rap Lyrics Generation: https://arxiv.org/pdf/1505.04771.pdf
[2] PPLSA: Parallel Probabilistic Latent Semantic Analysis Based on MapReduce: https://hal.inria.fr/hal-01524958/document