# Introduction

Spark is an open-source distributed cluster computing framework. it applys RDDs as the application programming interface which is maintained in a falut-tolerant way for distributed programs to share memory.

In our project, we apply spark to do the process in four stages. The implementation is based on cluster created in emr and configured as m4.xlarge node type of the hardware, which can ensure the performance of processing data. Notebook is embedded for more intuitive data visualization. More functions such as word2vec was used during processing which can be introduced later.

# Stage One: Overall statistics

We first create a SparkSession, and load music data from S3. Then read the data by spark and drop the parts of data we will not use later for saving unnecessary loading time.

**For music data:**

**• the total number of reviews**

We use .count() to calculate the number of items in the dataset, which equals to the total count of the reviewes.

The result is 4751577

**• the number of unique users**

We use .select(“customer\_id”) to choose the users and .distinct to make sure the users are unique. Finally, using .count() to calculate the number of unique users.

The result is 1940732

**• the number of unique products**

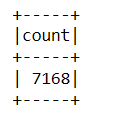
We use .select(“product\_id”) to choose the users and .distinct to make sure the users are unique. Finally, using .count() to calculate the number of unique products.

The result is 782326

**For user-review distribution:**

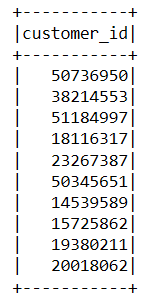
**• the largest number of reviews published by a single user**

We use .groupby(‘customer\_id’) to group the data as id=”customer\_id”, than count the number of each id. After that, we sort the number from high to low. Finally, we use .limit(1) to show the largest number of count.



**• the** **top 10 users ranked by the number of reviews they publish**

We use .groupby(‘customer\_id’) to group the data as id=”customer\_id”, than count the number of each id. After that, we sort the number from high to low. Finally, we use .limit(10) to show the top 10 users ranked by count.



**• the median number of reviews published by a user**

We use a variable named “count\_list\_user” to store the data with two column: one is groupby id=”customer\_id”, the other is \_count: the number of each id;

We use a variable named “count\_array\_user” to store the data with two column: one is groupby id=”customer\_id”, the other is turn the data type of the \_count into int type;

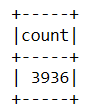
We use np.median(count\_array\_user) to calculate the the median number of reviews published by a user

The result is 1.0

**For product-review distribution:**

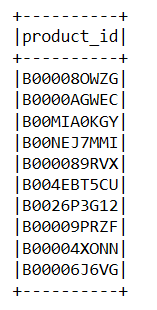
**• the largest number of reviews written for a single product**

We use .groupby(‘product\_id’) to group the data as id=”product\_id”, than count the number of each id. After that, we sort the number from high to low. Finally, we use .limit(1) to show the largest number of count.



**• the top 10 products ranked by the number of reviews they have**

We use .groupby(‘product\_id’) to group the data as id=” product\_id”, than count the number of each id. After that, we sort the number from high to low. Finally, we use .limit(10) to show the top 10 products ranked by count.



**• the median number of reviews a product has**

We use a variable named “count\_list\_product” to store the data with two column: one is groupby id=” product\_id”, the other is \_count: the number of each id;

We use a variable named “count\_array\_product” to store the data with two column: one is groupby id=” product\_id”, the other is turn the data type of the \_count into int type;

We use np.median(count\_array\_product) to calculate the the median number of reviews published by a product

The result is 2.0

# Stage2

**Reviews with less than two sentences in the review\_body**

Using a judgement “split\_reviews(x)”to split all reviews into a new array. Then changing this array to string type and add a new column “review\_body”.

**Reviews published by users with less than median number of reviews published**

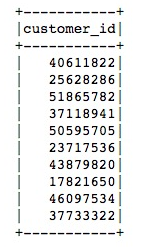
Grouping all reviews by “customer\_id” and sorting by ascending number. As the median number of reviews published by user has been calculated at stage 1 as 1, filtering out all grouped reviews which “count” larger than 1 and sort again according to new added column “count” to get specific reviews.

**Reviews from products with less than median number of reviews received**

Grouping all reviews by “product\_id”and sorting by ascending number. As the median number of reviews from products has been calculated at stage 1 as 2, filtering out all grouped reviews which “count” larger than 2 and sort again according to new added column “count” to get specific reviews.

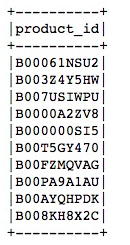
**Top 10 users ranked by median number of sentences in the reviews they have published**

Using processed reviews and adding new column which given a new name “list\_num\_sentences” as the size of reviews. Then reviews can be grouped by users. Defining a new function “find\_median” to find the median number and applying this function on “list\_num\_sentences”. Top 10 users can finally get by sorting and limit by 10.



**Top 10 products ranked by median number of sentences in the reviews they have received**

Using the same name “list\_num\_sentences” to represent the size of reviews after grouping reviews by “product\_id”. applying the same median calculation function and sorting and limiting by 10 to get top 10 products.



# Stage3

**For Positive:**

**• Extracting all reviews with rate 4 and above**

We choose a product\_id “B00006J6VG”. Then we define a variable “music\_product” which stores the data after filtering this product\_id “B00006J6VG” and drop the product\_id column & customer\_id column to save the loading time.

We define a variable “music\_positive” to store the data after filtering the “star\_rating” larger than 3, and drop the star\_rating column to save the loading time.

We define a variable “music\_p\_small” to limit the data length equals to the number of data in music\_positive;

We define a variable “music\_p\_small\_rdd” to create rdd for the class dataframe.

**• For each review, extracting the review body part and segment it into multiple sentences.**

We define a variable “review\_embedding\_p\_s3” to encode for the class “music\_p\_small\_rdd” by using mapPartitions to establish a connection for this partition. Founction “review\_encode” has also been used in mapPartitions.

The function “review\_encode” uses “TensorFlow”. We import tensorflow as tf, and import tensorflow\_hub as hub. We use hub.Module to import the encode. We define a variable to store each rdd in this partition. For each r\_body in each rdd, we select it and append into the review\_list. Then we store rdd into the result\_list. After that, we use tf.session to encode the review list. Embedding is stores the vector list. Finally, we pick the rdd from result\_list, expend each r\_body, and with each vector from embedding, building a new list which is the final result list.

**For Negative:**

**• Extracting all reviews with rate 2 and below**

We choose a product\_id “B00006J6VG”. Then we define a variable “music\_product” which stores the data after filtering this product\_id “B00006J6VG” and drop the product\_id column & customer\_id column to save the loading time.

We define a variable “music\_negative” to store the data after filtering the “star\_rating” smaller than 3, and drop the star\_rating column to save the loading time.

We define a variable “music\_n\_small” to limit the data length equals to the number of data in music\_negative;

We define a variable “music\_n\_small\_rdd” to create rdd for the class dataframe.

**• For each review, extracting the review body part and segment it into multiple sentences.**

We define a variable “review\_embedding\_n\_s3” to encode for the class “music\_n\_small\_rdd” by using mapPartitions to establish a connection for this partition. Founction “review\_encode” has also been used in mapPartitions.

The function “review\_encode” uses “TensorFlow”. We import tensorflow as tf, and import tensorflow\_hub as hub. We use hub.Module to import the encode. We define a variable to store each rdd in this partition. For each r\_body in each rdd, we select it and append into the reviewlist. Then we store rdd into the resultlist. After that, we use tf.session to encode the review list. Embedding is stores the vector list. Finally, we pick the rdd from resultlist, expend each rbody, and with each vector from embedding, building a new list which is the final result list.

**Intra-Class Similarity**

For positive and negative, each part do the following same steps:

Using mapPartitions to the rdd\_review\_embedding\_p\_s3 with the return\_list. This return\_list function is to select the vector from final\_result\_list and put them into a new list ”vec\_list\_p\_s3”. Then transfer it to the type of array.

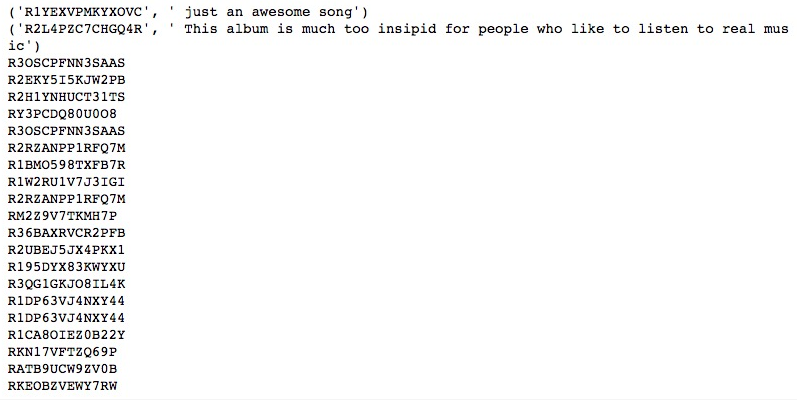
We use map function to collect the vector for each “vec\_list\_p\_s3” rdd and use the function calculate\_p\_s3\_distance to calculate each cosine distance, and store them into the list “dis” one by one. Finally, we use np.mean to calculate the average distance, and store them into the list dis\_mean.

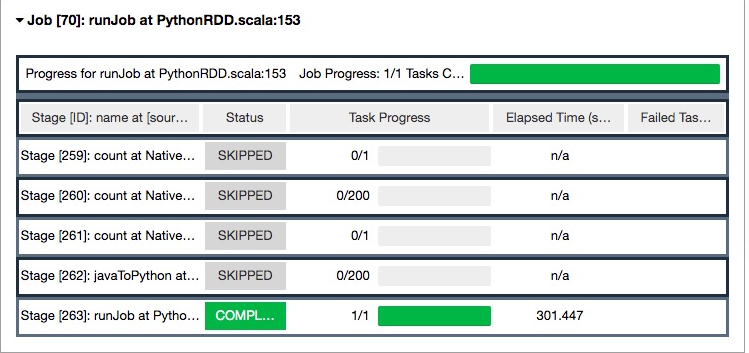
**Class Center Sentences**

Zip the two rdds with the format of (review\_id, review\_sentence, vector, average\_distance) by using .map(lambda x). For positive and negative, each part do the following same steps:

Sort the zip\_sort\_p\_s3 by the average\_distance. Find the first rdd and print it. We define a variable “p\_center\_s3” to store (review\_id, review\_sentence), and the vector should be stored into “p\_center\_vec\_s3”. And print the p\_center\_s3.

We use .map to deal with the vector list “review\_embedding\_p\_s3” by using the neareast\_p\_s3\_10 function. This function is to calculate the cosine distance between the positive center and all the other vectors. Then return(id, distance), sort by distance. And take the first 11 rdds as the original vector is in the vector array, so the first distance is 0. After that, we print the 10 vectors’ id except the first one.





# Stage 4

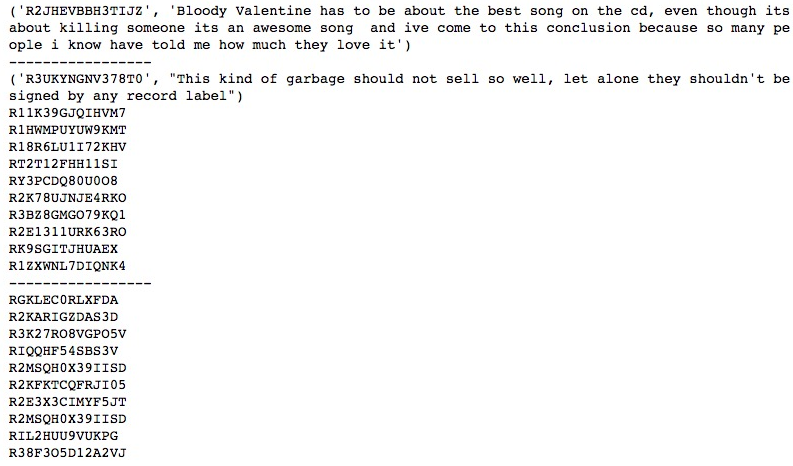
**Positive vs. negative reviews**

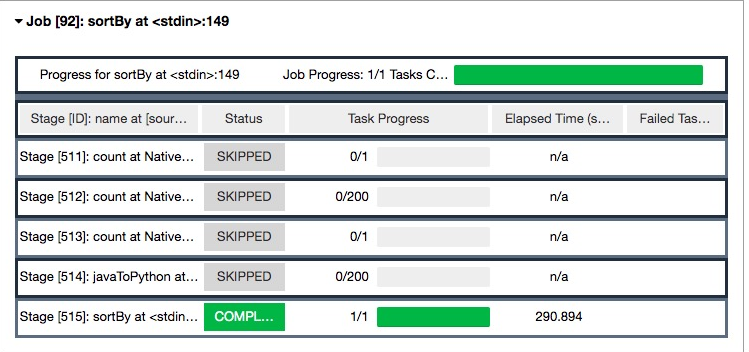
Choosing selected product and create two class dataframe which include positive and negative reviews and create rdd for two classes (music\_p\_small\_rdd, music\_n\_small\_rdd) by function “review\_encode\_preprocess”. Preprocessed reviews are reformatted from rdd to dataframes by a series of process on “music\_p\_preprocess\_reformat” and “music\_n\_preprocess\_reformat”. Dataframe contains review\_id and review\_body now. then doing tokenizer with regex to separate every word in review body and filter it if emply list by using “\w+”which represent one word/non-word character and filter out “review\_token” size bigger than 1. Processed sentence can be splitted into words in sentence lists and can ensure the content of sentence will not include single word character and null value.

Then “word2vec” is to encode lists “music\_p\_preprocess\_reformat\_token\_filter” and “music\_n\_preprocess\_reformat\_token\_filter” into 512 dimensions’ vector. The dataframe now should include columns of review\_id, review\_body, review\_token, vector and review token which is unnecessary in next stages and can filter out.

Using “rdd\_format” function to reformat dataframe to rdd format which contains review\_id, review\_body and vector. The final result of positive and negative reviews(“result\_p\_rdd\_reformat”and “result\_n\_rdd\_reformat”) can achieve.

Intra-class similarity and class center sentences can also gain by using processed reviews and the process method is the same with in stage 3.





# Compare Stage 3 to Stage 4

The difference between Stage 3 and Stage 4 are mainly focusing on the way of encode. Stage 3 uses google universal analytics, directly encode the rdd to vectors. However, Stage 4 uses word2vec, calculating the average distance by encoding each element in each rdd. sentence is represented by the average of the word embedding vectors of the words that compose the sentence.

Besides, there are both different due to the separate way of encoding. So the performance of these two are also different. Here are the performance comparison:

# Appendix