**Team members:**

|  |  |
| --- | --- |
| **Name** | **UBIT id** |
| Astrid Gomes | ASTRIDYV |
| Mansi Wagh | MANSIWAG |
| Rasita Pai | RASITASH |

Data preprocessing is done in the tasks using NLTK.

The following commands are to be run to install it:

**import** nltk

nltk**.**download**(**'punct'**)**

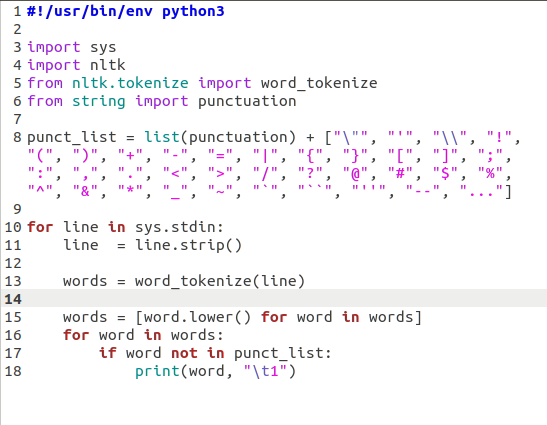
nltk**.**download**(**'stopwords'**)**

**Task 1: Word Count**

*Mapper*:

* Reads from text files line by line
* Tokenizes the words by using nltk and then converts all words to lowercase
* For each token formed, if it is not in the punctuations list, output the token and 1 along with it as its count.

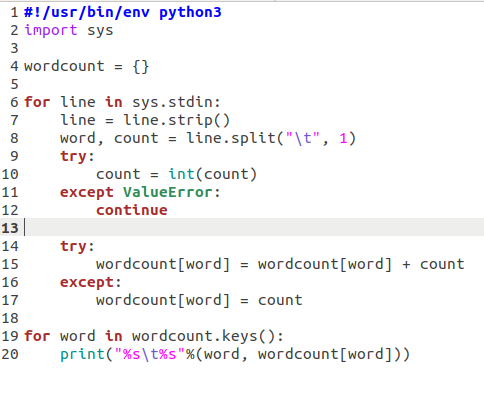
File name: mapper1.py



*Reducer*:

* Reads output of mapper line by line
* Split as word and its count
* If this word has been read earlier, then increment the count else store the count as 1 on first occurrence
* Once all lines have been read, output the final count for each token along with the token(word).

File name: reducer1.py



Hadoop command used:

hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/Assignment\_video/mapper1.py -mapper /home/cse587/Assignment\_video/mapper1.py -file /home/cse587/Assignment\_video/reducer1.py -reducer /home/cse587/Assignment\_video/reducer1.py -input /home/demopy/gutenberg -output /home/demopy/output1

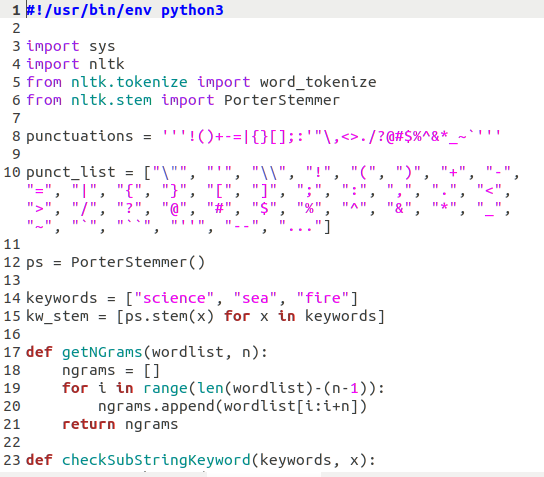
*Output format*: word count

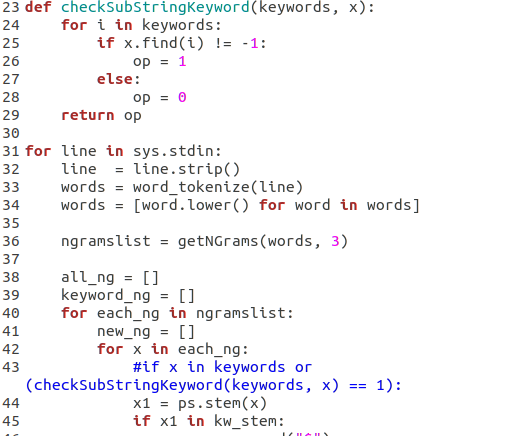
**Task 2: N-grams**

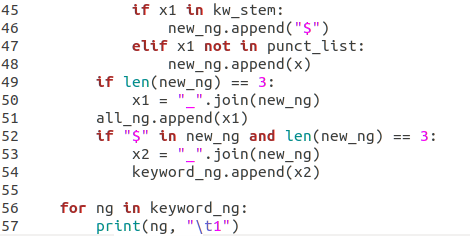
*Mapper*:

* Reads from text files line by line
* Tokenizes the words by using nltk and then converts all words to lowercase
* Then generates the tri-grams using the formed tokens
* It then converts the tokens to their root form using PorterStemmer from the nltk library
* In each tri-gram, replace the token with $ if it matches with any of the keywords given. This is the modified tri-gram
* For each modified tri-gram formed, output the modified tri-gram and 1 along with it as its count.

File name: mapper2.py



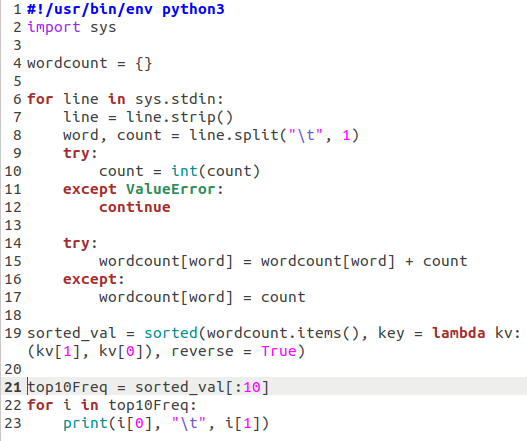




*Reducer*:

* Reads output of mapper line by line
* Split as tri-gram and its count
* If this tri-gram has been read earlier, then increment the count else store the count as 1 on first occurrence
* Once all lines have been read, sort the entries based on count in descending sequence.
* Output the final count for the top 10 tri-grams along with the tri-gram

File name: reducer2.py



Hadoop command used:

hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -jobconf mapred.reduce.tasks=3 -file /home/cse587/Assignment\_video/mapper2.py -mapper /home/cse587/Assignment\_video/mapper2.py -file /home/cse587/Assignment\_video/reducer2.py -reducer /home/cse587/Assignment\_video/reducer2.py -input /home/demopy/gutenberg -output /home/demopy/output2

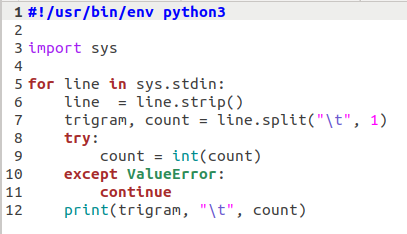
*Output format*: tri-gram count

Outputs the top 10 entries at each reducer

*Global mapper:*

* Reads the output of all the individual reducers from previous command
* Splits as tri-gram and count
* Once all lines have been read, output the tri-gram along with its count as is (summation of count for repeated tri-grams not done at this step)

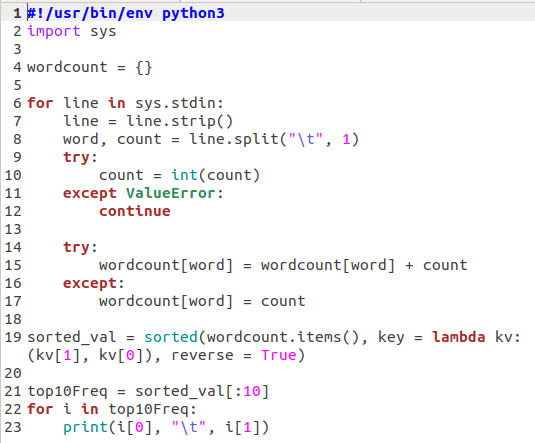
File name: mapper2\_1.py



*Global reducer:*

* Reads output of mapper line by line
* Split as tri-gram and its count
* If this tri-gram has been read earlier, then increment the count else store the count as 1 on first occurrence
* Once all 10\*n (n = number of mapper-reducer jobs) lines have been read, sort the entries based on count in descending sequence.
* Output the final count for the top 10 tri-grams along with the tri-gram

File name: reducer2\_1.py



Hadoop command used:

hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/Assignment\_video/mapper2\_1.py -mapper /home/cse587/Assignment\_video/mapper2\_1.py -file /home/cse587/Assignment\_video/reducer2\_1.py -reducer /home/cse587/Assignment\_video/reducer2\_1.py -input /home/demopy/outdemop2/part-00000 /home/demopy/outdemop2/part-00001 /home/demopy/outdemop2/part-00002 -output /home/demopy/output2.1

*Output format*: tri-gram count

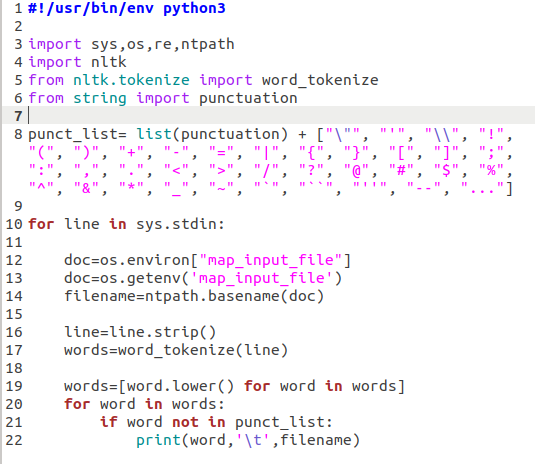
Outputs the top 10 among the 10\*n entries returned from n mapper-reducer jobs

**Task 3: Inverted Index**

*Mapper*:

* Reads from the input file line by line along with the filename, which is stored in ‘map\_input\_file’ variable in the Hadoop framework, and the filepath(using os.getenv(map\_input\_file) method)
* Tokenizes words in every line using the nltk library and converts to lowercase.
* Every token formed is then given as an output along with the filename to which it belongs to.

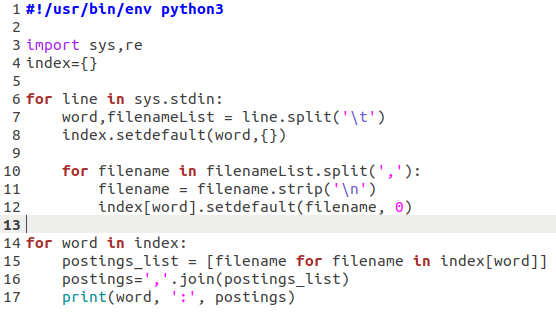
File name: mapper3.py



*Reducer*:

* Reads output of the mapper is read line by line.
* Words and the corresponding filename are split with a tab.
* If the word is being read the first time, its filename list is the file it belongs to.
* If the word has already been read before, append the filename to the list of filenames if this filename is not present in the filename list for the word
* After all the lines in the file have been read, output the words and the list of corresponding filenames in which it is present.

File name: reducer3.py



Hadoop command used:

hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/Assignment\_video/mapper3.py -mapper /home/cse587/Assignment\_video/mapper3.py -file /home/cse587/Assignment\_video/reducer3.py -reducer /home/cse587/Assignment\_video/reducer3.py -input /home/demopy/gutenberg -output /home/demopy/output3

*Output format: word : filename1.txt, filename2.txt*

**Task 4: Relational Join**

Preprocessing is done by converting the file format from .xlsx to .txt (tab delimited).

Once converted, the input (join1.xlsx and join2.xlsx) is saved as Task4\_join1.txt and Task4\_join2.txt respectively and then change the file encoding to ‘utf-8’.

*Mapper*:

* Reads the join input files line by line and combines the data from both the files.
* To distinguish from which file the data is being read we are using the length of the lines parsed.
* Initializes the Employee Id, Name, Country, Salary fields to -1 and the Passcode field to False. When the data comes to the mapper, these elements will be replaced by the real data from files.
* This combined output is then sent to the reducer.

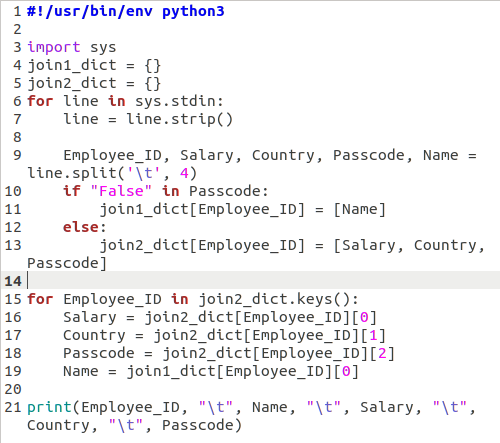
File name: mapper4.py



*Reducer*:

* Reads the data line by line which is coming from the mapper.
* Two dictionaries are created to store the data coming from the mapper.
* To identify the line read is from which dataset, the previously set initializations are used. Based on the passcode field, if the value is false, data is placed in join1\_dict else in join2\_dict, with the key of the dictionaries as the primary key(in this case the Employee Id)
* Both the dictionaries are joined based on the key and the output is the corresponding values.

File name:reducer4.py



Hadoop command used:

hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/Assignment\_video/mapper4.py -mapper /home/cse587/Assignment\_video/mapper4.py -file /home/cse587/Assignment\_video/reducer4.py -reducer /home/cse587/Assignment\_video/reducer4.py -input /home/demopy/Task4\_join1.txt /home/demopy/Task4\_join2.txt -output /home/demopy/output4

*Output format:* Employee Id, Name, Salary, Country, Passcode

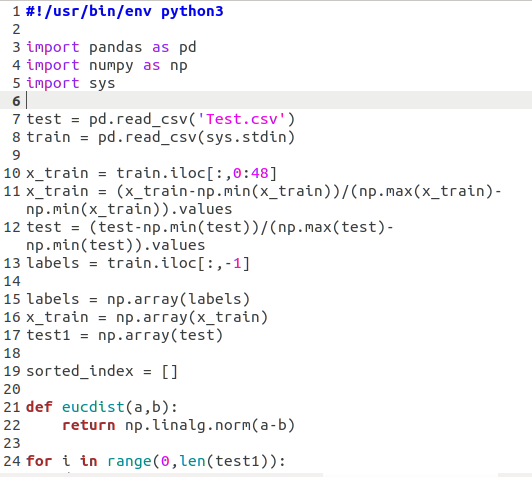
**Task 5: K-Nearest Neighbor**

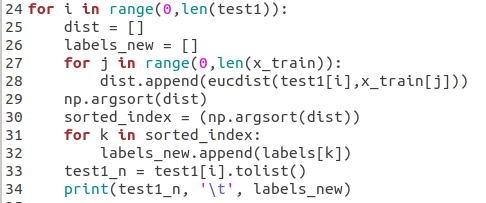
We have set the value of *k* to be 3 in our program.

*Mapper*:

* Reads Test.csv in each mapper and reads Train.csv as a standard input file.
* Training data is then separated from the labels.
* The test and train data is normalized
* Training data, Testing data and labels are converted to ndarray.
* Distance between each point in test data and train data is calculated
* Calculated distances are then sorted and mapped to the corresponding labels.
* These labels and the test data are given as output.

File name:mapper5.py

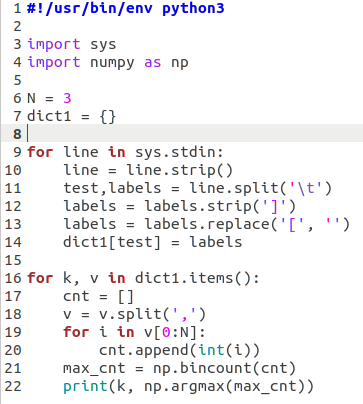




*Reducer*:

* Each row of the test data and its corresponding labels list is read line by line
* Dictionary is created in which every test data row is stored as key and the corresponding labels list are stored as dictionary values.
* Depending on the value of k, top k labels corresponding to each test data row are predicted.
* Using np.bincount, the count of each predicted label is calculated and np.argmax gives the highest occuring predicted label for each test data row.
* Test data row and its final predicted label is given as the output.

File name: reducer5.py



Hadoop command used:

hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/Assignment\_video/mapper5.py -mapper /home/cse587/Assignment\_video/mapper5.py -file /home/cse587/Assignment\_video/reducer5.py -reducer /home/cse587/Assignment\_video/reducer5.py -file /home/cse587/Test.csv -input /home/demopy/Train.csv -output /home/demopy/output5

*Output format:* Test\_data \_row predicted\_label