Team members:

Harishkandan Somasundaram 50313808 harishk2@buffalo.edu
Abhishek Dalvi 50320110 adalvi@buffalo.edu
Anand Jhamnani 50320027 anandarj@buffalo.edu

Word-count:

Special characters are removed for preprocessing. The three text files arthur.txt, james.txt and leonardo.txt are used as the input.

Mapper:
Each word from the input files is sent to the reducer as a (word, 1) pair.

Reducer:
Since the words are the keys, we can be assured that all the (word, 1) pairs of the same word will definitely go to a single reducer. Thus even if there are multiple reducers, each reducer will aggregate the (word, 1) pairs based on the same words and calculate the word count for each word and output (word, count) pairs.

N-grams:

We've used two mappers and reducers to solve this problem. We do this because in real life applications there can be multiple mappers and reducers. Multiple reducers will lead to multiple outputs each having the top ten trigrams that are local to each reducer. This problem is mitigated by chaining two map-reduce programs. The first map-reduce takes all the text files as inputs, creates the trigrams, computes it's counts and outputs the top ten trigrams from each of its reducers. The second map-reduce takes the trigrams and counts from each of the previous reducers and creates a global top 10. The code shall be explained as is present in each mapper and reducer. The three text files arthur.txt, james.txt and leonardo.txt are used as the input.

Mapper1:
For preprocessing, all the special characters are removed and lemmatization is performed to better detect our keywords. Three trigrams are built around each keyword. We then send the trigram as the key here and 1 as the value.

Reducer1:
The first reducer gets the (trigram, 1) pairs as it's input and computes the total number of that trigram present in the document from this. Since the key here is the trigram, we can be sure that all the instances of a particular trigram will go to the same reducer. Multiple reducers will each compute their own top ten trigrams and output them as (trigram, 1)

Mapper2:
This mapper simply takes the input from the first reducer and uses this as the value for its output while keeping the key values as 1 for all the incoming values. Keeping the key same(1 in this case) will ensure that all these (1, 'trigrams count') pairs will go to the same reducer.

Reducer2:
Now that we have all the top tens from the previous reducers in one reducer, we can now find the global top ten easily.

Inverted Index:

Mapper:
Using regex, only words are extracted and using os.environ["map_input_file"] we figure out the file name. Finally, the mapper prints out (word, file_name) as its output.

Reducer:
A dictionary is maintained where each word is the key and the value is the list of documents in which the word occurs, which is nothing but a representation of the inverted index. The dictionary updates itself in two ways:-

- If a (key,value) or (word,Doc) pair is fed into the reducer such that the word doesn't occur in the dictionary then the dictionary creates a new key as the word and the value as the Doc.
- If a (key,value) or (word,Doc) pair is fed into the reducer such that the word is already present in the dictionary then the dictionary updates the value of dict[word] by appending the current Doc to the already existing Doc/s.

Relational Join:

Note:- Use the join1.txt and join2.txt for execution in the code subfolder, relational_join.

Mapper:
The input stream is split using tab-delimiter. Using the structure of the two tables, if the length of the split is 2, then the input stream represents the 1st table, else it is the 2nd table, hence we know what split represents which column.
The mapper prints out all the 5 columns and if a column is not present, then that is given a value of "-1".

E.g of a row in table 2 after it goes through mapper

Employee ID	Name	Salary	Country	Passcode
16780611-9611	-1	$46,370	Belarus	NSG68IQE9UX

Reducer:
A dictionary is maintained where employee id is the key and value represents the other rows. The dictionary is updated in such a way that if:-

- The current Employee id doesn't exist in the dictionary, create a new key in the dictionary with values as the non "-1" columns from the input.
- The key/empid exists in the dictionary, find the length of the value of that key. This will tell us that from which table did we get the values from, in turn we'll get to know which values are to be updated.

KNN:

Since the test data size is small, we can assume that each Mapper can import the test data without affecting the capacity of the Mapper. Normalization is done prior as normalization in the mapper would lead to erroneous results with multiple mappers since data in each mapper will have a different min-max range.

Mapper:
In each mapper, the euclidean distance is computed for each test row with respect to each row in the train data. We then send the test row number as the key and a string containing the top K least distances and their labels separated by '$' as the value. This ensures that even with multiple mappers, we will have the global nearest K neighbour.

Reducer:
If there are N reducers, we'll get K*N (test_row, 'distance$label) pairs. From these now based on the distance, we find the K nearest neighbours and take the mode of the labels from those nearest neighbours to get out output as (test_row, label)