**Design:**

**Building the co-occurence vectors:**

to build the vectors we used the MR pattern. more precisely 2 phases of MR.

the first phase got the n-gram corpus as input.

the job of the first phase mapper was to extract from each line in the input file, the corresponding n-gram, and take from it the lexeme(head of the syntactic-tree), and all the features of that lexeme(the words which are the “sons” of the head of the syntactic-tree).

the mapper passed to the reducer the following:

each lexeme was passed as a key twice, once with an extra special character and once regularly, and the values were a ‘1’ for the first way and the features of that lexeme(including duplicities, to count how many times a specific feature appeared in the context of a that lexeme) for the second.

each feature was also passed as a key and a ‘1’ as a value. this way, the first phase reducer was able to count how many times each lexeme and each feature appeared in our corpus and saved that information in a file called “feature\_lexeme\_count.txt” which will be used to calculate probabilities in the second phase(we will load it to a hashMap in the setup of the second mapper).

the first phase reducer only writes to the context the lexemes that appear in the gold standard as key and an appended list of all the features as the value.

key-value estimation for the first mapper:

for each line in the corpus we send to the mapper the following,

(lexeme#, ‘1’), (lexeme,feature1) (lexeme, feature2) (sometimes only one feature),

(feature1, ‘1’). so if there are N lines in the corpus, there will be around 6N key-value pairs submitted to the reducer.

memory estimation for the first mapper:

here we dont really use memory

key-value estimation for the first reducer:

here we only write to the context, lexemes that appear in the gold-standard, so if there are N words in the gold standard we write N key-value pairs.

memory estimation for first reducer:

the gold standard file is loaded to the memory. the count of the lexemes and feature appearance is stored in a file so there is no memory usage for that.

the second MR phase uses only a mapper which does the following:

the input is the output of the first phase, ie a lexeme and all its features.

first we will count for each feature how many times it appears and save that info in another hashMap. after that, we have all the data we need for the calulation of all the different measures of association.

the output of that phase is the lexeme as the key and a json file as the co-occurence vector of that lexeme. the json file consists of each feature as a key and a json object containing the value of that feature for each of the 4 measures of association.

key-value estimation for the second mapper:

the output here is all the words from the gold standard and their feature vector, so if there are N words in the gold standard there will be N key-value pairs.

memory estimation for the second mapper:

we load to the memory the gold standard file and the “feature\_lexeme\_count” file we created in the first phase.

**Measures of vector similarity:**

to perform the vector similarity in each of the 5 methods we used a single MR job, ie phase 3. the input of this job is the output of the second stage, ie a lexeme and its feature vector.

first we load the gold standard, but this time we do it a little different. this time the key of the hash map will be the left word of the pair and the value will be a list of all the words it should measured with(the word on the right in the gold standard file). the mapper will check in a merged file containing all the vectors for each word that should be measured against the lexeme which is the key and send a key-pair value for containing the lexeme and its vector as a key and the other word and its vector as the value. like this the reducer will be able to measure against all the needed words.

the reducer will receive for each word(key), all the words(values) it should perform similarity measures against. all it does is iterate over the values and perform each of the 20 measurements. the output of the reducer is the words being measured as the key and json containing each of the 20 scores as the value.

key-value estimation for the mapper:

the number of key-value pairs will be the same as the number of pairs in the gold standard

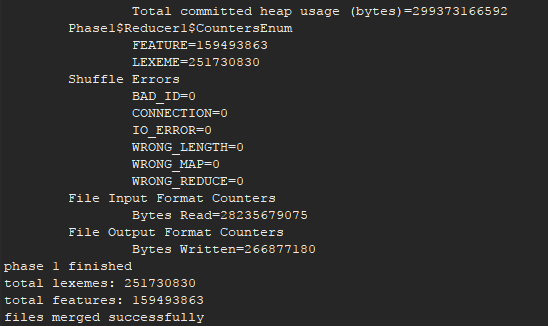
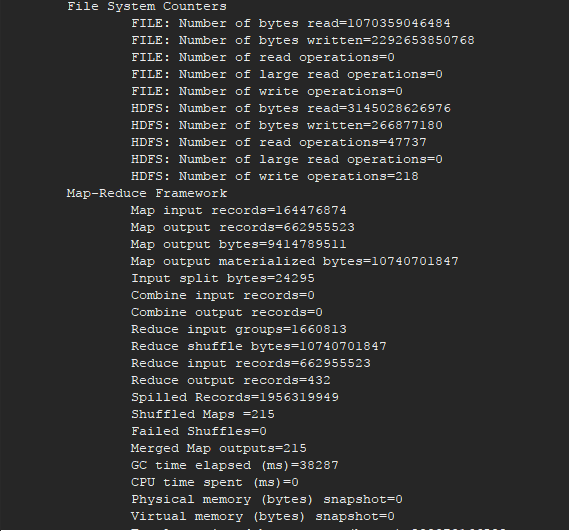
memory estimation: the gold standard is loaded into the memory

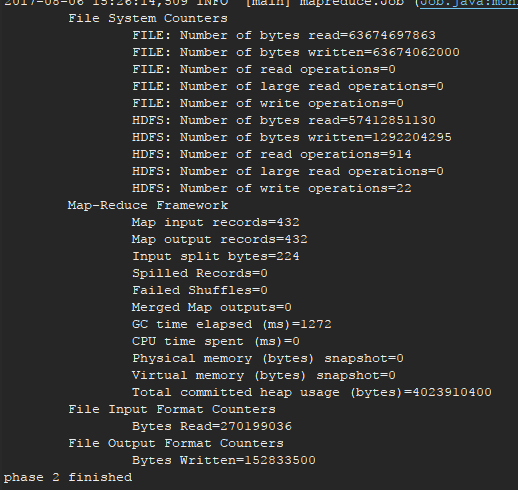
key-value estimation for the reducer:

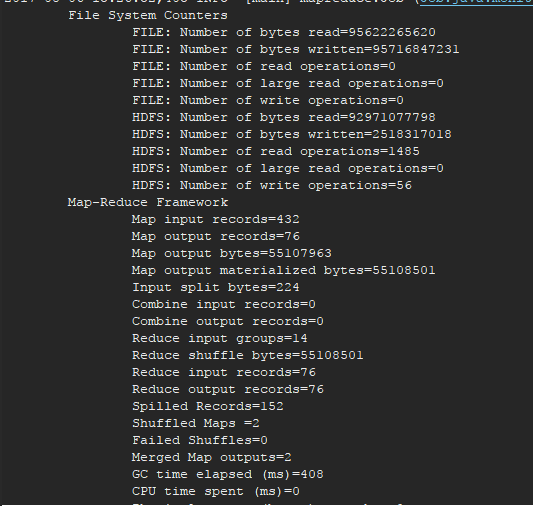
the number of key value pairs from the gold standard

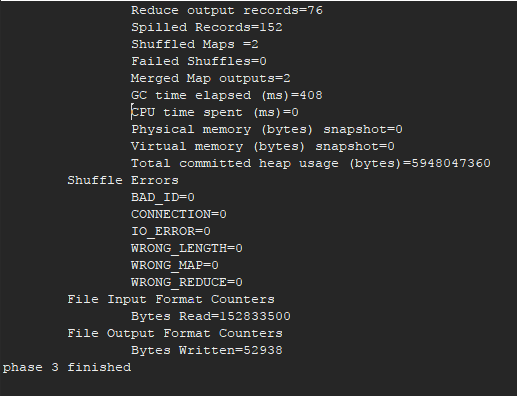
no memory is used for the for the reducer.

**communications**

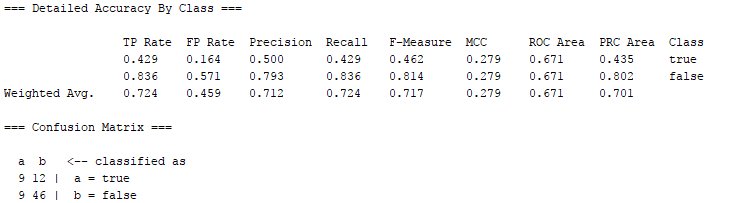


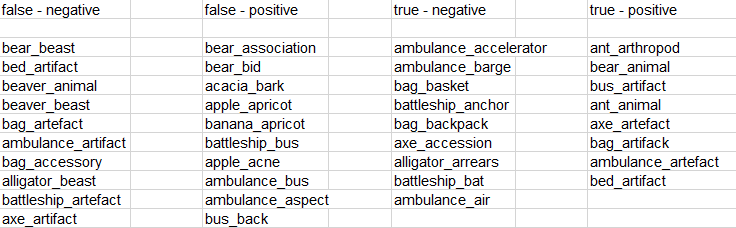






**Analysis**





For the classification we used the J48 classifier provided within “WEKA”.

We can see that all the false negatives are of the form “A” is a kind of “B”,

For example a bear is a kind of beast or a bag is a kind of accesory.

We can also see that the classifier classified some of the nounPairs of that form correctly but less then the ones that were false-negative(from the confusion matrix).

From that we conclude that our system has problems with those kinf of pairs.

Also, some of the true-negatives examples show us that the system might have some problems with pairs that are objects that are used for the same purpose(bag-backpack, bag-basket,), on the other hand, in the false-positive examples we can see that the words that 2 of the same kind(apple-apricot, ambulance-bus) are classified correctly.