**DSP Assignment 2**

Daniel Abutbul: 313843062

Sagi Levi : 308213446

**Program Explain**

App components:

* Start module: initialize the map-reduce program using Hadoop
* Steps0-3 modules: each step contains:
  + Main Class(named Step1 for example), that managing the Hadoop settings for the step
  + Mapper Class – with map operation
  + Reducer Class – with reduce operation
  + Partitioner Class – (if needed) insure each reduce will have all the required input

**Work-Flow**

In this assignment we were required to experience with a research paper of “Rion Snow et al” ,and re-designing its algorithm for the map reduce pattern and experimenting its quality on a large-scale input. We build a program in map-reduce pattern, that builds feature vector for each noun pair, and then ran a script that build the .arff file for WEKA, and then we give WEKA the file and examined its result from the training .

-we define “dp” – (i.e “dependency path”) to composed of the nodes and the edges – the post-tag and the word between.

In Step1 – instead of parsing corpus, we are using Google Syntactic-Ngram as input made of trees, then create all the dependency paths in the tree from noun to noun, by using the class Ngram Node that fits to the input of google, and then in the reducer, we eliminate all the patterns that not appear at least Dpmin times , and also create another file named “dp.txt” that holds all the unique dp that are legal. The number of keys the mapper is create is all the shorts path between two none pair, and the reducer output is all the dp that appear more than Dpmin, there is no memory usage in this stage.

In this step, the mapper and reducer working with Text ,Text when:

mapper Key: dp

mapper Value: noun-noun(pair of two noun with the dp correspond)

Reducer Key: dp (s.t dp appearances>Dpmin)

Reducer Value: noun-noun

In Step2 – we merge all the dp.txt files , into one file, and also give any dp an id, by using one reducer. No memory use and, the keys is just the number of unique legal dp.

mapper Key: dp

mapper Value: noun-noun(pair of two noun with the dp correspond)

Reducer Key: dp (s.t dp appearances>Dpmin)

Reducer Value: noun-noun

In Step3 – first we create new key Class named – “Paired-key”, for holding the dp, and the pair, and we are counting all the <Paired-key(dp,pair), 1> instances , and now in the reducer, we can count for each pair , how much dp they have from any kind, for the feature vector, also we are using the merge-dp file from the previous step to give each dp index, the output of this step is <pair, <dp-index ,count>>, we used partitioner to send the keys by the dp, for not searching the index every time.

We saving the dp -merge file in the reducer computer- from what we experiment the size is not that big for 5 dpmin. The number of key-value is for each pairs the number of dp legal that he is belong.

mapper Key: Paired - key

mapper Value: Long Writable

Reducer Key: pair(noun-noun)

Reducer Value: <dp-index, count>

In Step4 –now we create a class for the key, Feature Pair that hold the pair and dp index. We sort the keys by the pair, and if the pair equals, by the index.

In the reducer we have all the information about the specific pair that comes sorted by the index, and we are using the “annotated set” to see if the pair exist in the file, and give him true/false and NA for the pairs that not belong to the file ,to the postprocessing script.

We save in the memory the “annotated set” , The number of key-value is for each pairs the number of dp legal that he is belong.

mapper Key: Feature - Pair

mapper Value: Long Writable

Reducer Key: pair(noun-noun)

Reducer Value: The value can be in two types- first when we meet the word first time “-1,value” when value is true/false/NA and the second value is “dp index, count”

**Communication**

Statistics from step1:

Map input records=16145663

Map output records=2013652

Map output bytes=40410053

Reduce input records=2013652

Reduce output records=1774486

Statistics from step2:

Map input records=603

Map output records=603

Map output bytes=11612

Reduce input records=603

Reduce output records=603

Statistics from step3:

Map input records=1774486

Map output records=1774486

Map output bytes=46730295

Reduce input records=1774486

Reduce output records=540484

Statistics from step4:

Map input records=540484

Map output records=540484

Map output bytes=15300657

Reduce input records=540484

Reduce output records=966617

**Results**

A close up of a receipt

Description automatically generated

**Analysis**

True-positive:

land-farm,0.58654,TRUE,TRUE

develop-write,0.58654,TRUE,TRUE

receptor-structur,0.506443,TRUE,TRUE

secretari-chief,0.58654,TRUE,TRUE

valu-thought,0.58654,TRUE,TRUE

False-positive:

brain-vital,0.053197,FALSE,FALSE

print-window,0.053197,FALSE,FALSE

organ-segment,0.073979,FALSE,FALSE

abil-excit,0.956625,FALSE,FALSE

control-import,0.073979,FALSE,FALSE

True-negative + False-negative:

A picture containing train

Description automatically generated

ניתוח הערכים:

בעבור כול זוגות המילים שלעיל, ישנו רק מסלול dp יחיד שמהווה בערך 0.15% מסך המסלולים שנמצאו(משקל קטן מאוד) , ולכן קשה לבסס החלטה נכונה על סמך מאפיין בודד זה. שווה לציין שמשקלו של כול פיצ׳ר לא נקבע לפי חלקו היחסי מסך הפיצ׳רים, אך בכל המקרים המתוארים לעיל, משקל הפיצ'ר שאפיין אותם היה זניח.

ההשערה שלנו לתיוג השגוי טמונה לאו דווקא בדאטא המבוסס על ה- dp שמאפיין את צמד המילים שתויג בצורה שגויה, אלא על שאר הזוגות במאגר שקיים ביניהם אותו dependency path וקיבלו תיוג שאינו תואם לקשר האמיתי ביניהם. במילים ברורות יותר, אנו טוענים, לדוגמא, כי הסיבה שהצמד stock-seri תויג כ- TRUE ולא כ- FALSE נובע מכך שמכל מרחב הזוגות בדאטא, 244 זוגות תויגו כ- TRUE לעומת 28 בלבד שתויגו כ- FALSE. להבנתנו, מרגע ששאר הזוגות קיבלו תיוג (גם ו/או במיוחד כזה שאינו קשור ל dep path מספר 320) כ-TRUE, לראיית האלגוריתם, אותו dep path נראה כמסלול שסביר יותר שיתאר קשר hypernym אמיתי. במקרה כזה, כמו אצלנו, זוגות ש