Collocation Extraction

Due Date: 17/6

Submit your questions to Moshe by mail.

Abstract

In this assignment you will automatically extract collocations from the Google 2-grams dataset using Amazon Elastic Map Reduce.

Collocations

A collocation is a sequence of words or terms that co-occur more often than would be expected by chance. The identification of collocations - such as 'crystal clear', 'cosmetic surgery', 'איכות סביבה' - is essential for many natural language processing and information extraction application.

In this assignment, we will use Normalized Pointwise Mutual Information (NPMI), in order to decide whether a given pair of ordered words is a collocation, where two ordered words with a high NPMI value are expected to be a collocation.

Normalized PMI

Pointwise mutual information (PMI) is a measure of association used in information theory and statistics.

Given two words w_1 and w_2 and a corpus, we define *normalized pmi* as follows:

$$npmi(w_1, w_2) = \frac{pmi(w_1, w_2)}{-log[p(w_1, w_2)]}$$

Where:

$$p(w_1, w_2) = \frac{c(w_1, w_2)}{N}$$

$$pmi(w_1, w_2) = log[\frac{p(w_1, w_2)}{p(w_1)p(w_2)}]$$

According to conditional probability definition and maximum likelihood estimation, the PMI can be expressed (think why) by:

$$pmi(w_1, w_2) = log(c(w_1, w_2)) + log(N) - log(c(w_1)) - log(c(w_2))$$

Where *c* is count function and *N* is the total number of bi-grams in the corpus.

NPMI is a symmetric measure (*i.e.*, npmi(x,y) = npmi(y,x). In order to adapt it for the task of identifying collocations of ordered pairs, we define $c(w_1,w_2)$ as the count of the ordered pair w_1w_2 (excluding w_2w_1 pairs), $c(w_1)$ as the number of times w_1 starts some bigram, $c(w_2)$ as the number of times w_2 ends some bigram, and *N* as the number of bigrams.

The Assignment

You are asked to code a map-reduce job for collocation extraction for each decade, and run it in the Amazon Elastic MapReduce service. The collocation criteria will be based on the normalized PMI value:

- **Minimal pmi**: in case the normalized PMI is equal or greater than the given minimal PMI value (denoted by minPmi), the given pair of two ordered words is considered to be a collocation.
- **Relative minimal pmi**: in case the normalized PMI value, divided by the sum of all normalized pmi in the same decade (including those which their normalized PMI is less than minPmi), is equal or greater than the given relative minimal PMI value (denoted by relMinPmi), the given pair of two ordered words is considered to be a collocation.

Your map-reduce job will get the *minPmi* and the *relMinPmi* values as parameters.
You need to calculate the normalized pmi of each pair of words in each decade, and display all collations above each of the two minimum inputs. Run your experiments on the 2-grams Hebrew corpus.
The input of the program is the Hebrew 3-Gram dataset of Google Books Ngrams.
The output of the program is a list of the collocations for each decade, and there npmi value, ordered by their npmi (descending).

Stop Words

Stop words are words which are filtered out prior to, or after, processing of natural language data. You can use the stop-words from any stop-word list as is, or create a list as you see fit. (There is no "correct answer here" just reasonably add the words you wish to omit to the stop-words list).

In this assignment, you are supposed to remove all bigram that contain stop words and not include them in your counts.

IMPORTANT: Your solution must avoid the generation of redundant key-value pairs.

How to Run

The inputs for your assignment are the *minPmi* and *relMinPmi* mentioned above. Say that *minPmi* is 0.5 and *relMinPmi* is 0.2, the command line to execute your assignment should look like:

```
java -jar ass2.jar ExtractCollations 0.5 0.2
```

Scalability, Memory Assumptions

Your code must be scalable, *i.e.*, should successfully run on much larger input.
No assumption on the memory capacity can be made. In particular, you cannot assume that the word pairs of any decade, nor the set of pairs of a given word, nor a list of counts of unigrams etc, can be stored in memory.

Reports

When you submit the assignment should include the following report:

- For each decade, the top-10 collocations and there npmi value, ordered by npmi (descending).
- The number of key-value pairs that were sent from the mappers to the reducers in your map-reduce runs, and their size (take a look at the log file of Hadoop), **with and without local aggregation**.
- A list of 5 good collocations and a list of 5 bad collocations, you manually collected from the system output. In the frontal checking you will be asked to say something on why wrong collocations were extracted (a manual analysis).

Technical Stuff

Amazon Abstraction of Map Reduce

Amazon has introduced two abstractions for its Elastic MapReduce framework and they are: Job Flow, Job Flow Step.

Job Flow

A Job Flow is a collection of processing steps that Amazon Elastic MapReduce runs on a specified dataset using a set of Amazon EC2 instances. A Job Flow consists of one or more steps, each of which must complete in sequence successfully, for the Job Flow to finish.

Job Flow Step

A Job Flow Step is a user-defined unit of processing, mapping roughly to one algorithm that manipulates the data. A step is a Hadoop MapReduce application implemented as a Java jar or a streaming program written in Java, Ruby, Perl, Python, PHP, R, or C++. For example, to count the frequency with which words appear in a document, and output them sorted by the count, the first step would be a MapReduce application which counts the occurrences of each word, and the second step would be a MapReduce application which sorts the output from the first step based on the calculated frequenciess.

Example Code

Here is a small piece of code to help you get started:

```
AWSCredentials credentials = new PropertiesCredentials(...);
AmazonElasticMapReduce mapReduce = new
AmazonElasticMapReduceClient(credentials);

HadoopJarStepConfig hadoopJarStep = new HadoopJarStepConfig()
    .withJar("s3n://yourbucket/yourfile.jar") // This should be a full map
reduce application.
    .withMainClass("some.pack.MainClass")
    .withArgs("s3n://yourbucket/input/", "s3n://yourbucket/output/");

StepConfig stepConfig = new StepConfig()
    .withName("stepname")
    .withHadoopJarStep(hadoopJarStep)
    .withActionOnFailure("TERMINATE_JOB_FLOW");

JobFlowInstancesConfig instances = new JobFlowInstancesConfig()
    .withInstanceCount(2)
    .withMasterInstanceType(InstanceType.M4Large.toString())
    .withSlaveInstanceType(InstanceType.M4Large.toString())
    .withHadoopVersion("2.6.0").withEc2KeyName("yourkey")
    .withKeepJobFlowAliveWhenNoSteps(false)
    .withPlacement(new PlacementType("us-east-1a"));

RunJobFlowRequest runFlowRequest = new RunJobFlowRequest()
    .withName("jobname")
    .withInstances(instances)
    .withSteps(stepConfig)
    .withLogUri("s3n://yourbucket/logs/");

RunJobFlowResult runJobFlowResult = mapReduce.runJobFlow(runFlowRequest);
String jobFlowId = runJobFlowResult.getJobFlowId();
System.out.println("Ran job flow with id: " + jobFlowId);
```

Notice that order of commands matters.

Reading the n-grams File

The n-grams file is in sequence file format with block level LZO compression. In order to read it, use the code:

```
Configuration conf = new Configuration();
Job job = new Job(conf, "...");
...
job.setInputFormatClass(SequenceFileInputFormat.class);
```

Passing Parameters from the Main to the Mappers or Reducers

You can use the "Configuration" object to pass parameters from the main to the mapper/reducer:

- In order to set the value of the parameter:

```
Configuration jobconf = new Configuration();
jobconf.set("threshold", args[3]);
//threshold is the name of the parameter - you can write whatever you
like here, and arg[3] is its value in this case.
```

- To get the value in the mapper/reducer we use the context to get the configuration and then take the parameter value from it:

```
context.getConfiguration().get("threshold","1")
//"1" is the returned value if threshold has not been set.
```

Additional Notes

- Notice that some parts of the AWS SDK are deprecated due to SDK updates. It is okay to use these parts here.
- If you choose not to install Hadoop locally, you must pay attention to the fees, especially since the instances that are required in EMR are not included in the "Free Usage Tier". For debugging purpose it is recommended to choose the weakest instance possible (M4Large), and the lowest number instances to complete the job. In addition, start by testing your system on a small file, and only after you make sure all of the parts work properly, move to the big corpus.
  Consider every code you run, since each run is directly associated with money coming out of your budget!
- The version of Hadoop we are going to use is 2.6.0, however if you encounter any problem you may choose any other version.
- Notice that EMR uses other different products of AWS to complete its job, particularly: ec2 (management and computation) and S3 (storing logs, results, etc.). Make sure that every resource you have used is released /deleted after you're done.

Grading

- The assignment will be graded in a frontal setting.
- All information mentioned in the assignment description, or learnt in class is mandatory for the assignment.
- You will be reduced points for not reading the relevant reading material, not implementing the recommendations mentioned there, and not understanding them.
- Students belonging to the same group will not necessarily receive the same grade.
- All the requirements in the assignment will be checked, and any missing functionality will cause a point reduction. Any additional functionality will compensate on lost points. Whatever is not precisely defined in the assignment, you have the freedom to choose how to implement it.
- You should strive to make your implementation as scalable and efficient as possible, in terms of: time, memory, and money.

Submission

Use the submission system to submit a zip file that contains: (1) all your sources (no need to submit the jars); (2) the output mentioned above, **OR** a link to the output directory on S3 in the README file; (3) The required reports: statistics and analysis. Include a file called README that contains your usernames, names, ids, how to run the project, and any notes regarding your implementation or/and your map-reduce steps.