Knowledge-base for Word Prediction

Due Date: 2/1

Technical questions: Idan
Conceptual questions: Meni

Abstract

In this assignment you will generate a knowledge-base for Hebrew word-prediction system, based on Google 3-Gram Hebrew dataset, using Amazon Elastic Map-Reduce (EMR). The produced knowledge-base indicates for each pair of words the probability of their possible next words. In addition, you should examine the quality of your algorithm according to statistic measures and manual analysis.

1. The Assignment

Probability Function

Let us define the conditional probability of a word given two previous words, as suggested by Thede & Harper:

$$P(w_3|w_1,w_2) = k_3 \cdot \frac{N_3}{C_2} + (1-k_3)k_2 \cdot \frac{N_2}{C_1} + (1-k_3)(1-k_2) \cdot \frac{N_1}{C_0}$$

$$k_2 = \frac{log(N_2+1)+1}{log(N_2+1)+2}, k_3 = \frac{log(N_3+1)+1}{log(N_3+1)+2}$$

Where:

- N_1 is the number of times w_3 occurs.
- N_2 is the number of times sequence (w_2,w_3) occurs.
- N_3 is the number of times sequence (w_1,w_2,w_3) occurs.
- C_0 is the total number of word instances in the corpus.
- C_1 is the number of times w_2 occurs.
- C_2 is the number of times sequence (w_1,w_2) occurs.

Note: this formula combines the Maximum Likelihood Estimation method taught in class, with some backoff smoothing technique - make sure you understand it well.

Your Task

You are asked to build a map-reduce system for calculating the conditional probability of each *trigram* (w_1,w_2,w_3) found in a given corpus, to run it on the Amazon Elastic MapReduce service, and to generate the output knowledge base with the resulted probabilities.

The input corpus is the Hebrew 3-Gram dataset of Google Books Ngrams:

- The path to the Hebrew 3-Gram in S3: s3://datasets.elasticmapreduce/ngrams/books/20090715/heb-all/3gram/data
- A description of the file format can be found here.

The output of the system is a list of word trigrams (w_1,w_2,w_3) and their conditional probabilities ($P(w_3|w_1,w_2)$)). The list should be ordered: (1) by w_1w_2, ascending; (2) by the probability for w_3, descending.

For example:

קפה נמס עלית 0.6

קפה נמס מגורען 0.4

קפה שחור חזק 0.6

קפה שחור טעים 0.3

קפה שחור חם 0.1

…

שולחן עבודה ירוק 0.7

שולחן עבודה מעץ 0.3

…

Scalability, Memory Assumptions

Your code must be scalable, *i.e.*, should successfully run on much larger input. You should justify your memory assumptions. In particullar, you cannot assume that a list with all words in the corpus can be stored in the memory.

Stop Words

Stop words are words which appear very frequently in the corpus. In many natural language processing algorithms, the stop words are filtered (think why). You are required to remove all trigrams that contain stop words and not include them in your counts (lists of stop-words for Hebrew is provided in the assignment archive, feel free to find better lists in order to improve results).

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

Reports

When you submit the assignment should include the following report:

- 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**.
- Scalability report: The time of the running for two different numbers of mappers, and for two different input sizes.
- Choose 10 'interesting' word pairs and show their top-5 next words. Judge whether the system got to a reasonable decision for these cases.

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.
  ```

Local single-node

During the development, you can run your code locally on a single-node cluster of Hadoop, installed on your computer.
The installation is quite simple: Linux, Windows.

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

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. 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.