Label Propagation with Spark

Yifan, Janani, Anant

Out: November 14, 2017
Due: November 29, 2017

Guidelines for Answers: Please answer to the point. Please state any additional assumptions you make while answering the questions. You need to submit a tar file containing source files and a pdf version of report separately to autolab. Please make sure you write the report legibly for grading.

Rules for Student Collaboration: The purpose of student collaboration in solving assignments is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is allowed to seek help from other students in understanding the material needed to solve a homework problem, provided no written notes are taken or shared during group discussions. The actual solutions must be written and implemented by each student alone, and the student should be ready to reproduce their solution upon request. You may ask clarifying questions on Piazza. However, under no circumstances should you reveal any part of the answer publicly on Piazza or any other public website. Any incidents of plagiarism or collaboration without full disclosure will be handled severely.

Rules for External Help: Some of the homework assignments used in this class may have been used in prior versions of this class, or in classes at other institutions. Avoiding the use of heavily tested assignments detracts from the main purpose of these assignments, which is to reinforce the material and stimulate thinking. Because some of these assignments may have been used before, solutions to them may be available online or from other people. It is explicitly forbidden to use any such sources or to consult people who have solved these problems before. You must solve the homework assignments completely on your own. We will mostly rely on your wisdom and honor to follow this rule. However, if a violation is detected, it will be dealt with harshly.

- Did you receive any help whatsoever from anyone in solving this assignment? Yes/No
- If you answered yes, give full details:_____ (e.g. "Jane explained to me what is asked in Question 3.4")
- Did you give any help whatsoever to anyone in solving this assignment? Yes/No
- If you answered yes, give full details:_____ (e.g. "I pointed Joe to section 2.3 to help him with Question 2")

1 Introduction

In this assignment, you will implement a semi-supervised of Label Propagation algorithm—specifically, the algorithm due to Gharamani, Lafferty and Zhu[1] on partially labeled graphs. You are expected to use Python and Spark for this assignment. You can use only the built-in constructs of PySpark, and are not allowed to use `MLLib`, `GraphX` or any other Spark library. However, you are free to use `numpy` in Python, should you feel the need to do so. The goal of the assignment is to get you familiar with programming in Spark, as well as the Label Propagation algorithms. If you have any question, you are encouraged to post a question on Piazza, or reach out to any of the TAs.

2 Label Propagation

The Label Propagation algorithm, discussed in the SSL on Graphs lecture[2], is a semi-supervised algorithm used to infer the labels for unlabeled nodes in a graph, based on the labels of a few given nodes in the same graph. The algorithm does so by learning a probability distribution over all labels for each node in the graph.

Given a graph $G = (V, E)$, where V is the vertex set, E is the edge set, and a set of k labels $Y_i,\ \forall i = \{1, 2, \ldots, k\}$, the goal of the algorithm is to learn an assignment of labels to nodes. Specifically, the algorithm learns the probability value of labels $Y_i,\ \forall i$ being assigned to each node $v \in V$. In the semi-supervised setting, $V = U \cup L$, where U is the set of unlabeled vertices and L is the set of vertices with known labels. Typically, $|U| >> |L|$.

The algorithm[1] that you would implement, is based on propagation of the label distribution of a node to all the nodes through the edges. The weight of the edge corresponds to the ease of propagation. The probability of transition from a node j to node i, T_{ij} can therefore be described as the following

$$T_{ij} = \frac{w_{ij}}{\sum_{k \in nbr(j)} w_{kj}}$$

Where w_{ij} is the weight of edge connecting i and j and $nbr(j)$ is the list of neighbors for node j.

The algorithm is given as follows:

1. Propagate labels: $Y \leftarrow TY$
2. Row normalize Y
3. Reset seed nodes to the given label. Repeat 1.

[1] `http://mlg.eng.cam.ac.uk/zoubin/papers/CMU-CALD-02-107.pdf`

[2] `http://curtis.ml.cmu.edu/w/courses/index.php/Class_meeting_for_10-605_SSL_on_Graphs`

3 Apache Spark

3.1 Brief Introduction to Apache Spark

Apache Spark is a data-flow style data processing library that allows users to write their data processing code in Scala, Python, or Java. Data is loaded as a Resilient Distributed Database (RDD) from either the local file system, or from HDFS. RDDs are immutable structures that support lazy evaluation, so no modifications may be made in-place. However, new RDDs can created from existing RDDs using transformations such as `map`, `filter`, `reduceByKey`, `groupBy`, to name a few. Evaluation of RDDs is lazy, i.e. the required result wont be evaluated until you explicitly invoke an *action* indicating that you need the result. Examples of actions are `count` and `collect`. This lazy evaluation allows Spark to optimize the execution of transformations scheduled on RDDs.

Another useful feature of Spark is in-memory processing. You can specify that you want to `cache` an RDD in memory, if you intend to reuse the data in the RDD later. The full set of transformations that convert one RDD into another, and actions which force the calculation of a result can be found in the Spark programming guide: `http://spark.apache.org/docs/latest/programming-guide.html`. The programming guide is also a good introduction to Spark, and your go-to guide for the programming technicalities of this assignment. A more detailed RDD API reference with examples can be found here: `https://spark.apache.org/docs/latest/api/python/pyspark.html\#pyspark.RDD`.

Currently, Spark is available in three programming languages: Scala, Java and Python. **You are expected to use Python and the PySpark interface for this assignment.**

3.2 Running Spark on Andrew Machines

Spark is placed in the `bigML` directory. You need to add the location of Spark binaries to your `PATH` variable to be able to run Spark on the Andrew cluster machines. Add the following path to `PATH` environment variable to access Spark on `unix.andrew.cmu.edu` machines:

```
/afs/cs.cmu.edu/project/bigML/spark/bin
```

This can be done by adding the following line to ∼`/.cshrc` if you are using csh:

```
setenv PATH ${PATH}:/afs/cs.cmu.edu/project/bigML/spark/bin
```

or the following line to your ∼ `/.bashrc` if you are using bash:

```
export PATH=${PATH}:/afs/cs.cmu.edu/project/bigML/spark/bin
```

You can also run Spark locally by downloading the pre-built package[3] and setting up `PATH` environment.

[3] https://spark.apache.org/downloads.html

4 The Data

the We use Freebase, a large collaborative knowledge base, to build the graph G. There are two types of nodes in G: Entity nodes and Property nodes. An edge with certain weight connects the Entity node with its corresponding Property node. For example, an entity node could be *uranus*. Its corresponding property node is *astronomy#orbital_relationship#orbits*. After running the Label Propagation algorithm, you should assign label *greek_gods* to the node *uranus*.

You will run Label Propagation algorithm on two datasets: *freebase_1.graph* and *freebase_2.graph*[4]. Freebase1 has 32K nodes and 957K edges and Freebase2 has 301K nodes and 2.3M edges. The edge file has the following format:

```
<entity_node>\t<property_node>\t<weight>
```

The input is a directed graph and you need to convert it into an **undirected** graph. It is important to **normalize** the edge weight via dividing by the sum of outward edge weights.

There are 23 labels for Freebase1 and 192 labels for Freebase2. We provide two seeds files for both Freebase1 and Freebase2. One file contains two seeds per label (*seed1_2.txt* for Freebase1 and *seed2_2.txt* for Freebase2) and another file contains ten seeds per label (*seed1_10.txt* for Freebase2 and *seed2_10.txt* for Freebase2). The seeds file has the following format:

```
<entity_node>\t<label>
```

5 Evaluation

For each seeds file, you will run the Label Propagation algorithm on the corresponding hold-out evaluation file. Each line of the evaluation file contains an entity node. You need to generate the **rank of all labels** for each entity node from the evaluation file. If some labels have the same probabilities, print them in lexical order. The format of the output file for is:

```
<entity_node>\t<rank1_label>\t<rank2_label> ... \t<rank23_label>
```

You will use Mean Reciprocal Rank (MRR) to evaluate the output in all experiments:

$$\text{MRR} = \frac{1}{|Q|} \sum_{v \in Q} \frac{1}{r_v} \tag{1}$$

Where $Q \subset V$ is the set of evaluation nodes, and r_v is the rank of the gold label among the labels assigned to node v. Higher MRR means better performance. We provide a script to

[4]https://autolab.andrew.cmu.edu/courses/10605-f17/assessments/hw6lpwithspark/handout

calculate MRR. You can run the evaluation using the following command line, which takes the output generated by Label Propagation algorithm, the file containing gold labels, and the number of different labels as input:

```
python mrr.py --n_labels 23 --output_file result.txt --gold_file gold1_2.txt
```

6 Deliverables

You should implement the Label Propagation algorithm discussed above in PySpark. Your input will be in the format described in §4. Your program should construct RDDs with convenient representations of the input graph (transition matrix/edge list), and seed label mappings. You will run the Label Propagation algorithm for the given number of epochs, specified as an input parameter, and obtain the rank of all labels for each node in the evaluation file. The output should follow the format described in §5.

Please write your original code in PySpark. We'd be happy to point you to Spark code[5] for the Page Rank algorithm–you may find it useful when you implement Label Propagation algorithm. In autolab, we will run *lp.py* and *mrr.py* using the following command lines:

```
spark-submit --driver-memory 3g
          lp.py \
        --iterations 2 \
        --edges_file freebase1.graph \
        --seeds_file seed1_10.txt \
        --eval_file eval1_10.txt \
        --number_of_excutors 4 \
        > result.txt
python mrr.py --n_labels 23 --output_file result.txt --gold_file gold1_10.txt
```

The command lines are included in the Makefile. Gold label file *gold1_10.txt* is not provided. To build the handin tarball, run

```
make handin
```

to generate *hw6.tar*, which will contain *lp.py*. Submit this tarball to Autolab. Autolab will grade for MRR, speed and memory.

7 Report Questions

1. Run your code for two epochs on the Freebase2 dataset using ten seeds per label (corresponding files given in §4) on an Andrew machine (`unix.andrew.cmu.edu`), for the following values of the `number_of_executors` parameter: 1, 2, 4, 8. Plot a graph

[5] https://github.com/apache/spark/blob/master/examples/src/main/python/pagerank.py

of the time taken for your program to run as a function of the number of executors used. What trend do you observe? Mention two reasons why the time does or doesn't linearly decrease with the number of executors used.

Note: You can measure end-to-end execution time using the `time` command line tool.

2. This question has two parts.

   (a) Run your code on Freebase1 with two seeds per label and on Freebase2 with two seeds per label. Report the obtained MRR's. Why are the values different? Explain briefly.

   (b) Run your code on Freebase2 with two seeds per label and with ten seeds per label. Report the obtained MRR's. Why are the values different? Explain briefly.

3. Jimmy writes the following function to count the number of elements in a given RDD. He notices that the function does not always return the correct answer. Could you tell him what's wrong?

```
def increment_counter():
    global counter
    counter += 1

def get_number_of_elements(rdd):
    global counter
    counter = 0
    rdd.foreach(lambda x:increment_counter())
    return counter
```

8 Marking breakdown

- Code correctness (MRR) [**40 points**].
- Code speed [**20 points**].
- Code memory [**10 points**].
- Question 1 [**10 points**].
- Question 2 [**10 points**].
- Question 3 [**10 points**].