Analysis Of People Relationship Using Word2Vec on Wiki Data

ABHISHEK GUPTA[1,*] AND AVADHOOT AGASTI[1,**]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: abhigupt@iu.edu
[**] Corresponding authors: aagasti@iu.edu

project-1: Data mining for a wiki url , April 19, 2017

Given a wiki URL of a person, find out his details like School, Spouse, Coaches, language, alma-meter etc Typically, the wiki page has all this information available but in the free form text. We need to converting it into structured data format so that it can help us analyze the people, from the networks etc We can create a network by navigating the people mentioned in the wiki page.

Keywords: Cloud, I524

https://github.com/cloudmesh/sp17-i524/blob/master/project/S17-IR-P005/report/report.pdf

CONTENTS

1. INTRODUCTION

Use spark [1] to load the wiki data and create word vectors. Train it using spark ML [2] and then use the model for analytics and prediction. The training set will use Word2Vec. Word2vec [3] is a group of related models that are used to produce word embeddings. Word2Vec is used to analyze the linguistic context of the words. In this project, we created Word2vec model using Wikipedia data. Our focus is people and organization names occurring in the Wikipedia data and to see if Word2vec can be used to understand relationship between people. Typically Wikipedia page for people and celebraties contain the entire family and friends, colleagues information. Our idea is to use Word2vec to see if using a smaller training set of known relationships whether we can derive similar relationship for anyone who has presence on Wikipedia. This mechanism can be then used to convert the data hidden in textual format to more structured data.

Technology Name	Purpose
spark [1]	data analysis
sparkML [2]	machine learning
python [1]	development
ansible [4]	automated deployment
collectd [5]	statistics collection for benchmarking

2. PLAN

Following table gives a breakdown of tasks in order to complete the project. Assuming week1 starts after submission of the proposal. These work items are high level breakdown on the tasks and may changes if needed.

Week	Work Item	Status
week1	Basic POC of Word2Vec using Python	planned
week2	Scripts to download Wiki data	planned
week3	Word2Vec Spark program	planned
week4	Training and measuring accuracy	planned
week5	Ansible Deployment script for Spark	planned
week6	Deployment and test on 2 clouds	planned
week7	Performance measurement	planned
week8	Report Creation(parallel)	planned

3. DESIGN

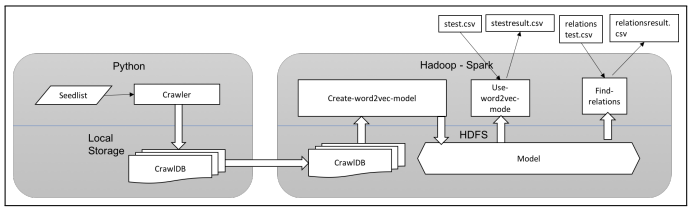

Fig. 1. Data Pipeline.

Figure 1 shows the overall data pipeline for the project. The data pipeline has three important stages:

- Crawler: Crawler runs in batch mode on a standalone machine. It can download wikipedia data as explained in section 3.1. Crawler creates CrawlDB which is a collection of text files. This crawler can be replaced or augmented with any web-crawler which can download or create the text files.
- create-word2vec-model: This component is responsible for creating the word2vec model for the text files in the CrawlDB. This model runs on Spark and stores the model on HDFS. Section 3.2 describes this component in detail.
- use-word2vec-model and find-relations: These two components use the precreated word2vec model to find synonym of a word or find the relationships. Section 3.3 describes these components in detail.

3.1. Crawler

The Crawler component is useful to download the data from web. We implemented a simple crawler using Python which can deep traverse the wikipedia pages and download the text from it. In our crawler implementation, a user can specify the seed pages from wikipedia. User can also specify the maximum number of pages that are required to be downloaded. The crawler first downloads all the pages specified in the seedlist. It then extract the links from each wikipedia page and puts it in a queue which is internally maintained by the crawler. The crawler then downloads the the linked pages. Since this logic is implemented in recursive manner, the crawler can potentially download all the wikipedia pages which can be reached from the pages in the seedlist.

We followed the seedlist based crawler approach so that we can retrieve domain specific web pages. A well chosen seedlist can fetch large number of relevant web pages.

Figure 2 is the flowchart of the crawler implementation.

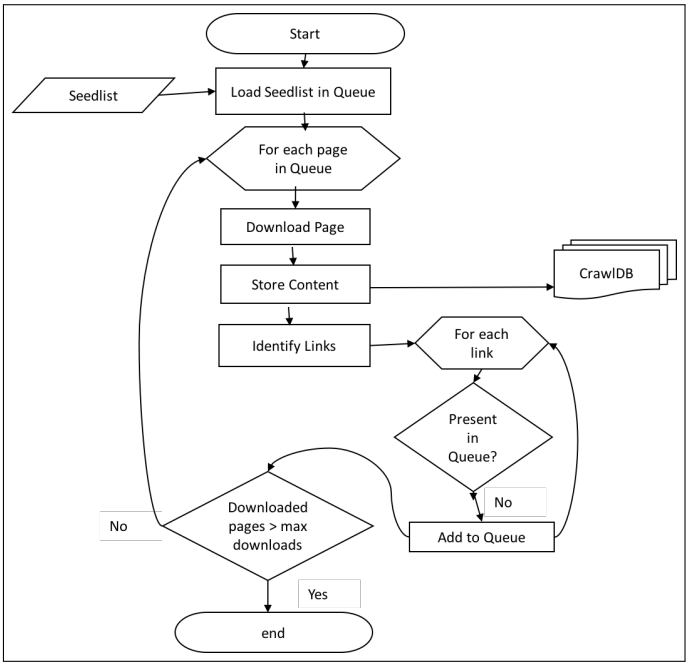

Fig. 2. Flowchart of crawler.

3.2. Word2Vec Model Creation

3.3. Using the Word2Vec Model

4. DEPLOYMENT

The deployment on the cluster can be accomplished using 2 steps, assuming the cluster is up and running

- Step1: update hosts file *ansible-word2vec/hosts* with the IP of the master. The first node on the cluster becomes the master node.
- Step2: run the script *ansible-word2vec/run.sh*. This script will run the ansible playbooks to accomplish stage1 through stage4 of the deployment process.

Figure 3 shows the deployment stages. The two steps accomplish deployment in multiple stages as discussed in the sections below.

4.1. Stage1

As pre-requisite, we need to create a cluster with 1 or more nodes. We created a 3 node cluster using Cloudmesh [6] command line interface(CLI). Cloudmesh[6] CLI allows you to orchestrate virtual machines(VM) in a cloud environment. For this project, we have used Chameleon and Jetstream cloud providers to orchestrate the VMs using cloudmesh[6] CLI. We can orchestrate a 3 node cluster using following CLI:

```
cm reset
pip uninstall cloudmesh_client
pip install -U cloudmesh_client
cm key add --ssh
cm refresh on
cm cluster define --count 3 --image CC-Ubuntu14.04 --flavor m1.
cm hadoop define spark pig
cm hadoop sync
cm hadoop deploy
cm cluster cross_ssh
```

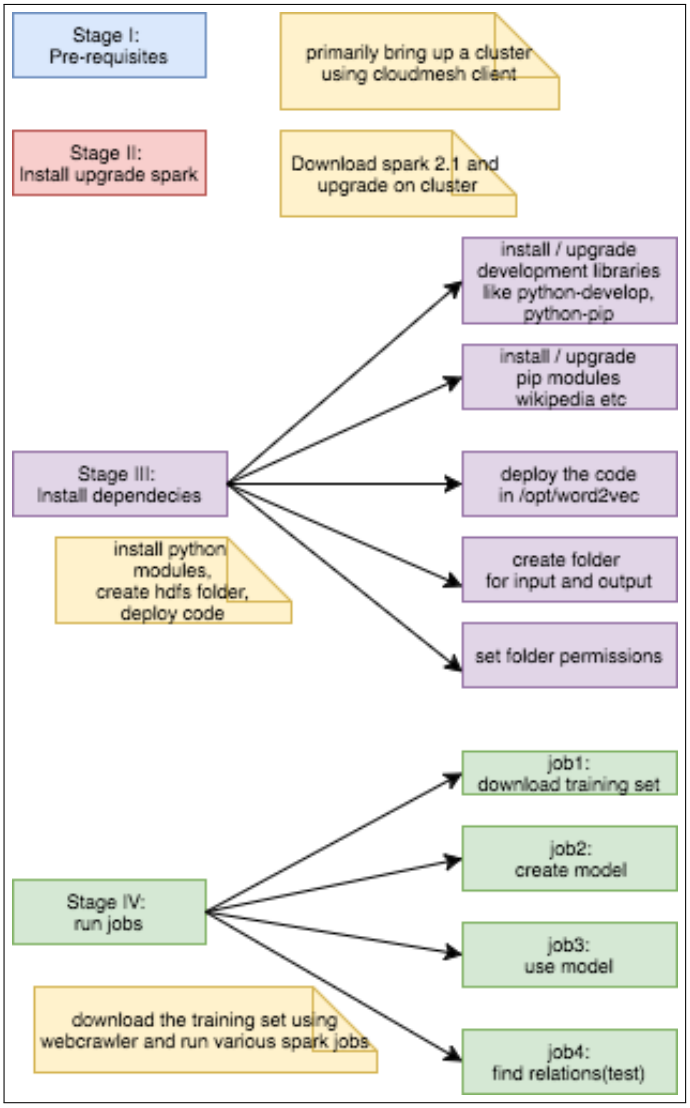

Fig. 3. Deployment stages.

We are using Ubuntu14.04 image with m1.medium which comes with 2 CPU, 4GB memory. Also, the nodes created are having hadoop and spark add-ons. We can test the deployment by checking hdfs and spark-submit CLI work fine.

```
ssh cc@<cluster-ip>
sudo su - hadoop
hdfs
spark-submit
```

At this stage our cluster is ready for further deployments.

4.2. Stage2

By default, cloudmesh installs spark 1.6 but our word2vec solution requires spark 2.1. We need to upgrade spark on the cluster. In order to do so we can run *install_upgrade_spark.yaml* ansible[4] playbook. This will download and unpack spark2.1 tar ball and further update the softlink to point to spark 2.1 folder.

4.3. Stage3

In this step, we upgrade the development libraries for python, and pip, install python modules like wkipedia, request etc, download the code from git repo and install it in */opt/word2vec* folder, set the folder permissions for the */opt/word2vec* folder so that it can be executed by hadoop user. These steps can be achieved using *word2vec_setup.yaml* playbook. After completing this stage, we are ready for running our word2vec solution on the cluster.

4.4. Stage4

This stage primarily deals with submitting the jobs for various purpose. Before we submit the jobs, we need to make sure input folder are created on hadfs. First, we run the crawler to download the training set and upload the data on hdfs. Further we run various jobs to created model and find relations. Along with these jobs we also run some monitoring jobs. The monitoring job queries spark metrics using

```
http://${spark_master}:4040
```

Stage4 steps can be accomplished using *word2vec_execute.yaml* playbook.

At the end of stage4 we also fetch the execution results from the cluster along with the metrics of execution times at various stages. The output files are fetched into */tmp/word2vec_results*

```
ls -lrt /tmp/word2vec_results
total 56
-rw-rw-r--  1 abhigup4  wheel   571 Apr 18 17:36 jobs.csv
-rw-rw-r--  1 abhigup4  wheel   418 Apr 18 17:36 executors.csv
-rw-rw-r--  1 abhigup4  wheel    89 Apr 18 17:36 app.csv
-rwxrwxr-x  1 abhigup4  wheel    43 Apr 18 23:43 stest.csv
-rwxrwxr-x  1 abhigup4  wheel   145 Apr 18 23:43 relationstest.
-rw-r--r--  1 abhigup4  wheel  1833 Apr 18 23:46 stestresult.cs
-rw-r--r--  1 abhigup4  wheel   844 Apr 18 23:47 relationsresul
```

Files *jobs.csv, executors.csv, and app.csv* collect the execution time for various jobs. File *relationsresult.csv* file collects the results for sample relations corresponding to *relationstest.csv*. Similarly *stestresult.csv* collects the results corresponding to *stest.csv*.

5. BENCHMARKING

Solution will use collectd [5] to collect statistics. Once the solution is deployed to the cluster. We should benchmark parameters like

- cpu
- memory
- throughput reads/writes

Benchmarking will be done for one or more cloud providers. The deployment scripts should be agnostic to the cloud provider.

6. DISCUSSION

TBD

7. CONCLUSION

Using this wiki analysis we should be able to build a network based on wiki data.

8. ACKNOWLEDGEMENT

We acknowledge our professor Gregor von Laszewski and all associate instructors for helping us and guiding us throughout this project.

9. APPENDICES

TBD

REFERENCES

[1] "Spark Python API (PySpark)," Web Page, accessed: 2017-02-26. [Online]. Available: https://spark.apache.org/docs/0.9.1/python-programming-guide.html

[2] "Spark ml programming guide," Web Page, accessed: 2017-02-26. [Online]. Available: https://spark.apache.org/docs/1.2.2/ml-guide.html

[3] "Word2Vec, learning vector representation of words," Web Page, accessed: 2017-02-26. [Online]. Available: https://en.wikipedia.org/wiki/Word2vec

[4] Red Hat, Inc., "Ansible documentation," Web Page, Jan. 2015, accessed 2017-01-13. [Online]. Available: https://docs.ansible.com/ansible/index.html

[5] "collectd - the system statistics collection daemon," Web Page, accessed: 2017-02-26. [Online]. Available: https://collectd.org/

[6] "Cloudmesh client," Code Repo. [Online]. Available: https://github.com/cloudmesh/client