Homework 1

10-405/10-605: Machine Learning with Large Datasets

Due Monday, February 3rd at 9:00 AM

1 Introduction

This assignment involves understanding some basics of distributed computing, the MapReduce programming model, Spark, and an example of data cleaning.

This assignment consists of two major parts. The first part is to build a simple word count application, and the second part is on entity resolution, a common type of data cleaning.

2 Logistics

We provide the code template for this assignment in a Jupyter notebook. What you need to do is to follow the instructions in the notebook and implement the missing parts marked with '`<FILL IN>`' or '`# YOUR CODE HERE`'. Most of the '`<FILL IN>/YOUR CODE HERE`' sections can be implemented in just one or two lines of code.

2.1 Getting lab files

You can obtain the notebook '`assignment_notebook.ipynb`' after downloading and unzipping `hw1.zip` at https://github.com/10605/released-hws/raw/master/hw1/hw1.zip.

Next, import the notebook into your Databricks, which provides you a well-configured Spark environment and will definitely save your time (see the next section for details).

2.2 Preparing for submission

We provide several public tests via `assert` in the notebook. You may want to pass all those tests before submitting your homework.

In order to enable auto-grading, please do not change any function signatures (e.g., function name, parameters, etc) or delete any cells. If you do delete any of the provided cells (even if you re-add them), the autograder will fail to grade your homework. If

you do this, you will need to re-download the empty '`assignment_notebook.ipynb`' file and fill in your answers again and resubmit.

2.3 Submission

1. Export your solution notebook as a IPython notebook file on Databricks via `File -> Export -> IPython Notebook`
2. Submit your solution via Gradescope (Please don't rename your notebook file).

3 Setting up environments on Databricks

We provide step-by-step instructions on how to configure your Databricks platform. We also introduced it in detail during the recitation on January 17. The recitations slides can be found here.

1. Sign up for the **Community Edition** of Databricks here: https://databricks.com/try-databricks.
2. Import the notebook file we provide on your homepage: `Workspace -> Users -> Import`
3. Installing third-party packages that will be used in the homework on Databricks: `Workspace -> Create -> Library`. Then select `PyPI`, enter the package name as `nose`. Finally click `Create` to install it. You may also want to check `install automatically on all clusters` option to install the PyPI package on all clusters you create.
4. Create a cluster: `Clusters -> Create Cluster`. You can use any cluster name as you like. When configuring your cluster, make sure to choose **the default Databricks runtime version and Python version** `Python 3`. Note: It may take a while to launch the cluster, please wait for its status to turn to '`active`' before start running.
5. You can start to play with the notebook now!

Note: Databricks Community Edition only allows you to launch one 'cluster'. If the current cluster is 'terminated', then you can either (1) delete it, and then create a new one, or (2) activate and attach to the existing cluster when running the notebook.

4 Word Count: Building a word count application

This exercise will develop a simple word count application. The volume of unstructured text in existence is growing dramatically, and Spark is an excellent tool for analyzing this type of data. In this exercise, we will write code that calculates the most common words in the Complete Works of William Shakespeare retrieved from Project Gutenberg. This could also be scaled to find the most common words in any text, e.g., Wikipedia. In this part we will cover:

- Creating a base RDD and pair RDDs

- Counting with pair RDDs
- Finding unique words and a mean value
- Apply word count to a file

See the notebook for detailed descriptions and instructions of each question.

5 Entity Resolution

Entity Resolution, or ”Record linkage” is the term used by statisticians, epidemiologists, and historians, among others, to describe the process of joining records from one data source with another that describe the same entity. Other terms with the same meaning include, “entity disambiguation/linking”, “duplicate detection”, “deduplication”, “record matching”, “(reference) reconciliation”, “object identification”, “data/information integration”, and “conflation”.

Entity Resolution (ER) refers to the task of finding records in a dataset that refer to the same entity across different data sources (e.g., data files, books, websites, databases). ER is necessary when joining datasets based on entities that may or may not share a common identifier (e.g., database key, URI, National identification number), as the case may be due to differences in record shape, storage location, and/or curator style or preference. A dataset that has undergone ER may be referred to as being cross-linked. In this homework, we break the entity resolution problem into four parts:

- **Part 0 – Preliminaries:** Load in the dataset into pair RDDs where the key is the mapping ID, and the value is a string consisting of the name/title, description, and manufacturer of the corresponding record.
- **Part 1 – ER as Text Similarity - Bags of Words:** Build components for bag-of-words text analysis, and then compute record similarity. Bag-of-words is a conceptually simple yet powerful approach for text analysis. The idea is treating strings, a.k.a. documents, as unordered collections of words or tokens, i.e., as bags of words.
- **Part 2 – ER as Text Similarity - Weighted Bag-of-Words using TF-IDF:** In this part we compute the TF-IDF weight for each record. Bag-of-words comparisons do not perform well when all tokens are treated in the same way. In real world scenarios, some tokens are more important than the others. Weights give us a way to specify which tokens could have higher ”importance”. With weights, when we compare documents, instead of counting common tokens, we sum up the weights of common tokens. A good heuristic for assigning weights is called “Term-Frequency/Inverse-Document-Frequency,” or TF-IDF for short. **TF** rewards tokens that appear many times in the same document. It is computed as the frequency of a token in a document. **IDF** rewards tokens that are rare overall in a dataset. The intuition is that it is more significant if two documents share a rare word than a common one.
- **Part 3 – ER as Text Similarity - Cosine Similarity:** Compute the cosine similarity of the tokenized strings based on the TF-IDF weights.
- **Part 4 – Scalable ER:** Use the inverted index data structure to scale ER. The ER algorithm above is quadratic in two ways. First, we did a lot of redundant computation of tokens and weights, since each record was reprocessed every time it was compared. Second, we made

quadratically many token comparisons between records. In reality, most records have nothing (or very little) in common. Moreover, it is typical for a record in one dataset to have at most one duplicate record in the other dataset (this is the case assuming each dataset has been de-duplicated against itself). In this case, the output is linear in the size of the input and we can hope to achieve linear running time. An inverted index is a data structure that will allow us to avoid making quadratically many token comparisons. It maps each token in the dataset to the list of documents that contain the token. So, instead of comparing, record by record, each token to every other token to see if they match, we will use inverted indices to look up records that match on a particular token.

- **Part 5 – Analysis:** Determine duplicate entities based on the similarity scores, and compute evaluation metrics. Now we have an authoritative list of record-pair similarities, but we need a way to use those similarities to decide if two records are duplicates or not. The simplest approach is to pick a threshold. Different thresholds correspond to different false positives and false negatives, which will result in different precision and recall scores.

See the notebook for detailed descriptions and instructions of each question.