CSE 6242/CX 4242: Data and Visual Analytics | Georgia Tech | Spring 2018

Homework 4 : Scalable PageRank via Virtual Memory (MMap), Random Forest, Scikit-Learn

Due: Friday, April 20, 2018, 11:55 PM EST

Prepared by Arathi Arivayutham, Siddharth Gulati, Jennifer Ma, Mansi Mathur, Vineet Vinayak Pasupulety, Neetha Ravishankar, Polo Chau

Submission Instructions and Important Notes:

It is important that you read the following instructions carefully and also those about the deliverables at the end of each question or you may lose points.

- Always check to make sure you are using the most up-to-date assignment PDF (e.g., re-download it from the course homepage if unsure).
- Submit a single zipped file, called "HW4-{YOUR_LAST_NAME}-{YOUR_FIRST_NAME}.zip", containing all the deliverables including source code/scripts, data files, and readme. Example: 'HW4-Doe-John.zip' if your name is John Doe. **Only .zip is allowed** (no other format will be accepted)
- You may collaborate with other students on this assignment, but you must write your own code and give the explanations in your own words, and also mention the collaborators' names on T-Square's submission page. All GT students must observe the honor code. **Suspected plagiarism and academic misconduct will be reported to and directly handled** by the Office of Student Integrity (OSI). Here are some examples similar to Prof. Jacob Eisenstein's NLP course page (grading policy):
  - **OK:** discuss concepts and strategies (e.g., how cross-validation works, use hashmap instead of array)
  - **Not OK:** several students work on one master copy together (e.g., by dividing it up), sharing solutions, or using solution from previous years or from the web.
- If you use any "*slip days*", you must write down the number of days used in the T-square submission page. For example, "Slip days used: 1". Each slip day equals 24 hours. E.g., if a submission is late for 30 hours, that counts as 2 slip days.
- At the end of this assignment, we have specified a folder structure about how to organize your files in a single zipped file. 5 points will be deducted for not following this strictly.
- We will use auto-grading scripts to grade some of your deliverables (there are hundreds of students), so it is extremely important that you strictly follow our requirements. Marks may be deducted if our grading scripts cannot execute on your deliverables.
- Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
- In your final zip file, please do not include any intermediate files you may have generated to work on the task, unless your script is absolutely dependent on it to get the final result (which it ideally should not be).
- After all slip days are used, 5% deduction for every 24 hours of delay. (e.g., 5 pts for 100-point homework)
- We will not consider late submission of any missing parts of a homework assignment or project deliverable. To make sure you have submitted everything, download your submitted files to double check.

Download the HW4 Skeleton before you begin.

Q1 [30 pts] Scalable single-PC PageRank on 70M edge graph

In this question, you will learn how to use your computer's **virtual memory** to implement the PageRank algorithm that will scale to graph datasets with as many as billions of edges using a single computer (e.g., your laptop). As discussed in class, a standard way to work with larger datasets has been to use computer clusters (e.g., Spark, Hadoop) which may involve steep learning curves, may be costly (e.g., pay for hardware and personnel), and importantly may be "overkill" for smaller datasets (e.g., a few tens or hundreds of GBs). The virtual memory based approach offers an attractive, simple solution to allow practitioners and researchers to more easily work with such data (visit the NSF-funded MMap project's homepage to learn more about the research).

The main idea is to place the dataset in your computer's (unlimited) virtual memory, as it is often too big to fit in the RAM. When running algorithms on the dataset (e.g., PageRank), the operating system will automatically decide when to load the necessary data (subset of whole dataset) into RAM.

This technical approach to put data into your machine's virtual memory space is called "memory mapping", which allows the dataset to be treated as if it is an in-memory dataset. In your (PageRank) program, you do not need to know whether the data that you need is stored on the hard disk, or kept in RAM. Note that memory-mapping a file does NOT cause the whole file to be read into memory. Instead, data is loaded and kept in memory only when needed (determined by strategies like *least recently used* paging and *anticipatory* paging).

You will use the Python modules `mmap` and `struct` to map a large graph dataset into your computer's virtual memory. The `mmap()` function does the "memory mapping", establishing a mapping between a program's (virtual) memory address space and a file stored on your hard drive -- we call this file a "memory-mapped" file. Since memory-mapped files are viewed as a sequence of bytes (i.e., a binary file), your program needs to know how to convert bytes to and from numbers (e.g., integers). `struct` supports such conversions via "packing" and "unpacking", using format specifiers that represent the desired endianness and data type to convert to/from.

Q1.1 Set up Pypy

Install PyPy, which is a Just-In-Time compilation runtime for python, which supports fast packing and unpacking. (As mentioned in class, C++ and Java are generally faster than Python. However, several projects aim to boost Python speed. PyPy is one of them.)

Ubuntu	`sudo apt-get install pypy`

MacOS	Install Homebrew Run `brew install pypy`
Windows	Download the package and then install it.

Run the following code in the Q1 directory to learn more about the helper utility that we have provided to you for this question.

```
$ pypy q1_utils.py --help
```

Q1.2 Warm Up (10 pts)

Get started with memory mapping concepts using the code-based tutorial in `warmup.py`.
You should study the code and modify parts of it as instructed in the file. You can run the tutorial code as-is (without any modifications) to test how it works (run "`python warmup.py`" on the terminal to do this). The warmup code is setup to pack the integers from 0 to 63 into a binary file, and unpack it back into a memory map object. You will need to modify this code to do the same thing for all even integers in the range of 1 to 42. The lines that need to be updated are clearly marked. **Note:** You must not modify any other parts of the code. When you are done, you can run the following command to test whether it works as expected:

```
$ python warmup.py
$ python q1_utils.py test_warmup out_warmup.bin
```

It prints `True` if the binary file created after running `warmup.py` contains the expected output.

Q1.3 Implementing and running PageRank (20 pts)

You will implement the PageRank algorithm, using the power iteration method, and run it on the LiveJournal dataset (an online community with millions of users to maintain journals and blogs). You may want to revisit the MMap lecture slides (slide 9, 10) to refresh your memory about the PageRank algorithm and the data structures and files that you may need to memory-map. (For more details, read the MMap paper.) You will perform three steps (subtasks) as described below.

Step 1: Download the LiveJournal graph dataset (an edge list file)

The LiveJournal graph contains almost 70 million edges. It is available on the SNAP website. We are hosting the graph on our course homepage, to avoid high traffic bombarding their site.

Step 2: Convert the graph's edge list to binary files (you only need to do this once)

Since memory mapping works with binary files, you will convert the graph's edge list into its binary format by running the following command at the terminal/command prompt:

```
$ python q1_utils.py convert <path-to-edgelist.txt>
```

Example: Consider the following `toy-graph.txt`, which contains 7 edges:

```
0 1
1 0
1 2
2 1
3 4
4 5
5 2
```

To convert the graph to its binary format, you will type:

```
$ python q1_utils.py convert toy-graph/toy-graph.txt
```

This generates 3 files:

```
toy-graph/
```

- `toy-graph.bin`: binary file containing edges (source, target) in little-endian "int" C type
- `toy-graph.idx`: binary file containing (node, degree) in little-endian "long long" C type
- `toy-graph.json`: metadata about the conversion process (required to run pagerank)

In `toy-graph.bin` we have,

```
0000 0000 0100 0000   # 0 1   (in little-endian "int" C type)
0100 0000 0000 0000   # 1 0
0100 0000 0200 0000   # 1 2
0200 0000 0100 0000   # 2 1
0300 0000 0400 0000   # 3 4
0400 0000 0500 0000   # 4 5
0500 0000 0200 0000   # 5 2
ffff ffff ffff ffff
...
ffff ffff ffff ffff
ffff ffff ffff ffff
```

In `toy-graph.idx` we have,

```
0000 0000 0000 0000 0100 0000 0000 0000   # 0 1 (in little-endian “long long” C type )
0100 0000 0000 0000 0200 0000 0000 0000   # 1 2
...
ffff ffff ffff ffff ffff ffff ffff ffff
```

Note: there are extra values of -1 (`ffff ffff or ffff ffff ffff ffff`) added at the end of the binary file as padding to ensure that the code will not break in case you try to read a value greater than the file size. You can ignore these values as they will not affect your code.

Step 3: Implement and run the PageRank algorithm on LiveJournal graph’s binary files

Follow the instructions in `pagerank.py` to implement the PageRank algorithm.
You will only need to write/modify a few lines of code.

Run the following command to execute your PageRank implementation:

```
$ pypy q1_utils.py pagerank <path to JSON file for LiveJournal>
```

This will output the 15 nodes with the highest PageRank scores.

For example: `$ pypy q1_utils.py pagerank toy-graph/toy-graph.json`

```
node_id score
1       0.4106875
2       0.2542078125
0       0.1995421875
5       0.0643125
4       0.04625
3       0.025
```

(Note that only 6 nodes are printed here since the toy graph only has 6 nodes.)

Step 4: Experiment with different numbers of iterations.

Find the output for the top 15 nodes for the LiveJournal graph for n=10, 25, 50 iterations (try the `--iterations n` argument in the command above; the default number of iterations is 10). A file in the format **pagerank_nodes_n.txt** for “n” number of iterations. For example:

```
$ pypy q1_utils.py pagerank toy-graph/toy-graph.json --iterations 25
```

You may notice that while the top nodes' ordering starts to stabilize as you run more iterations, the nodes' PageRank scores may still change. The speed at which the PageRank scores converge depends on the PageRank vector's initial values. The closer the initial values are to the actual pagerank scores, the faster the convergence.

Deliverables

1. **warmup.py [6pt]**: your modified implementation.
2. **out_warmup.bin [3pt]**: the binary file, automatically generated by your modified warmup.py.
3. **out_warmup_bytes.txt [1pt]**: the text file with the number of bytes, automatically generated by your modified warmup.py.
4. **pagerank.py [14pt]**: your modified implementation.
5. *pagerank_nodes_{n}.txt* **[6pt]**: the 3 files (as given below) containing the top 15 node IDs and their pageranks for n iterations, automatically generated by q1_utils.py.
    - **pagerank_nodes_10.txt [2pt]** for n=10
    - **pagerank_nodes_25.txt [2pt]** for n=25
    - **pagerank_nodes_50.txt [2pt]** for n=50

Q2 [50 pts] Random Forest Classifier

Note: You must use Python 2.7 for this question.

You will implement a random forest classifier in Python. The performance of the classifier will be evaluated via the out-of-bag (OOB) error estimate, using the provided dataset.
Note: You **must not** use existing machine learning or random forest libraries like scikit-learn.

You will use the UCI Cylinder Bands Dataset where the records are attributes of cylinders collected at various time intervals. All attribute names and values have been changed to meaningless symbols to maintain confidentiality. The dataset has been cleaned to remove missing attributes. The data is stored in a comma-separated file (csv) in your Q2 folder as **hw4-data.csv**. You MUST use only this modified dataset for this question. Each line describes an instance using 30 columns: the first 29 columns represent the attributes of the cylinder, and the last column is the classification value (1 means "band", 0 means "noband"). **Note:** The last column **should not** be treated as an attribute.

You will perform binary classification on the dataset to determine if a cylinder has a band or not.

Essential Reading

Decision Trees

To complete this question, you need to develop a good understanding of how decision trees work. You can refer to the lecture slides from class. Specifically, you need to know how to construct decision trees using *Entropy* and *Information Gain* to select the splitting attribute and split point for the selected attribute. These slides from CMU provide an excellent example of how to construct a decision tree using *Entropy* and *Information Gain*.

Random Forests

To refresh your memory about random forests, see Chapter 15 in the "Elements of Statistical Learning" book, lecture slides. Here is a blog post that introduces random forests in a fun way, in layman's terms.

Out-of-Bag Error Estimate

In random forests, it is not necessary to perform explicit cross-validation or use a separate test set for performance evaluation (also discussed in class). Out-of-bag (OOB) error estimate has shown to be reasonably accurate and unbiased. Below, we summarize the key points about OOB described in the original article by Breiman and Cutler.

Each tree in the forest is constructed using a different bootstrap sample from the original data. Each bootstrap sample is constructed by randomly sampling from the original dataset **with replacement** (usually, a bootstrap sample has the same size as the original dataset). Statistically, about one-third of the cases are left out of the bootstrap sample and not used in the construction of the *kth* tree. For each record left out in the construction of the *kth* tree, it can be assigned a class by the *kth* tree. As a result, each record will have a "test set" classification by the subset of trees that treat the record as an out-of-bag sample. The majority vote for that record will be its predicted class. The proportion of times that a predicted class is not equal to the true class of a record averaged over all records is the OOB error estimate.

Starter Code

We have prepared starter code written in Python for you to use. This would help you load the data and evaluate your model. The following files are provided for you:

- `util.py`: utility functions that will help you build a decision tree
- `decision_tree.py`: a decision tree class that you will use to build your random forest
- `random_forest.py`: a random forest class and a main method to test your random forest

What you will implement

Below, we have summarized what you will implement to solve this question. Note that you MUST use **information gain** to perform the splitting in the decision tree. The starter code has detailed comments on how to implement each function.

1. `util.py`: implement the functions to compute entropy, information gain, and perform splitting.
2. `decision_tree.py`: implement the `learn()` method to build your decision tree using the utility functions above.
3. `decision_tree.py`: implement the `classify()` method to predict the label of a test record using your decision tree.
4. `random_forest.py`: implement the functions `_bootstrapping()`, `fitting()`, `voting()`

As you solve this question, you will need to think about multiple parameters in your design, some may be more straightforward to determine, some may be not (hint: study lecture slides and essential reading above). For example,

- Which attributes to use when building a tree?
- How to determine the split point for an attribute?
- When do you stop splitting leaf nodes?
- How many trees should the forest contain?

Note that, as mentioned in class and on lecture slides, there are other approaches to implement random forests. For example, instead of information gain, other popular choices include Gini index, random attribute selection (e.g., PERT - Perfect Random Tree Ensembles). We decided to ask everyone to use an information gain based approach in this question (instead of leaving it open-ended), to help standardize students solutions to help accelerate grading efforts.

Deliverables

1. **hw4-data.csv**: The dataset used to develop your program. Do not modify this file.
2. **[10 pts] util.py**: The source code of your utility functions.
3. **[30 pts] decision_tree.py**: The source code of your decision tree implementation.
4. **[10 pts] random_forest.py**: The source code of your random forest implementation with appropriate comments.

Q3 [30 points] Using Scikit-Learn

Scikit-learn is a popular Python library for machine learning. You will use it to train some classifiers on the Epileptic Seizure Recognition[1] dataset in the folder, called seizure_dataset.csv.

[1] Derived from https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition

Part A - Classifier Setup [7 pts]

Train each of these classifiers on the dataset, using the classes provided in the links below. You will do hyperparameter tuning in Part B to get the best accuracy for each classifier on the dataset.

1. Linear Regression
2. Multi-Layer Perceptron
3. Random Forest
4. Support Vector Machine (The link points to SVC, which is a particular implementation of SVM by scikit.)

Scikit has additional documentation on each of these classes, explaining them in more detail, such as how they work and how to use them.

Use the skeleton file called *run_classifiers.py* to write your code. **Note: You must use Python 2.7 for this question.**
In *report.txt*, under Part A, follow the skeleton and put your training and testing accuracies for each classifier.

As a reminder, the general flow of your machine learning code will look like this:

1. Load dataset
2. Preprocess (you will do this in a later step of the question)
3. Split the data into x_train, y_train, x_test, y_test
4. Train the classifier on x_train and y_train
5. Predict on x_test
6. Evaluate testing accuracy by comparing the predictions from step 5 with y_test.

Here is an example. Scikit has many other examples as well that you can learn from.

Part B - Hyperparameter Tuning [13 pts]

Tune your Random Forest and SVM to get its best accuracy on the dataset. Tune the hyperparameters specified below, using the GridSearchCV function that Scikit provides:

- For random forest, tune the parameters “n_estimators” and “max_depth”.
- For SVM, tune “C” and “kernel” (try only ‘linear’ and ‘rbf’).

Use **10 folds** by setting the *cv* parameter to 10.

You should test at least 3 values for each of the numerical parameters. For C, the values should be different by factors of at least 10, for example, 0.001, 0.01, and 0.1, or 0.0001, 0.1 and 100.

In section "Part B" of *report.txt*, for each hyperparameter:

- explain what it does, using no more than 25 words
- state the values you tested

Also follow the skeleton in *report.txt* to report the best combination of hyperparameter values for each classifier tuned, its testing accuracy from Part A, and its best testing accuracy from tuning. For each classifier the best testing accuracy from tuning should be at least as high as the testing accuracy from Part A.

Note: If GridSearchCV is taking a long time to run, you can try standardizing or normalizing your data beforehand.

Part C - Cross-Validation Results [2 pts]

Let's practice getting the results of cross-validation. For only your SVM, report the *mean training score, mean testing score* and *mean fit time* for the best combination of hyperparameter values that you obtained in Part B. The GridSearchCV class holds a 'cv_results_' dictionary that should help you report these metrics easily.

Report the metrics in *report.txt* under the Part C section. Report your accuracies as percentages and round them to the nearest whole number, for example 85%.

Part D - Preprocessing the data [4 pts]

Perform either standardization or normalization on your dataset and retrain your *SVM* on it, using the best combination of hyperparameter values from your CV run in Part B. Then retest the SVM (on x_test) to get its new testing accuracy. In *report.txt*, in the "Part D" section, state the testing accuracies from before and after preprocessing, and explain why you think the strategy did or did not work in 50 words or less.

You can choose to either standardize or normalize your data for this purpose.
Use the skeleton file called preprocess_data.py to write your code.

Note:
If you are using StandardScaler:

- Pass 'x_train' into the fit method. Then transform both 'x_train' and 'x_test' to obtain the standardized versions of both.

- The reason we fit only on x_train and not the entire dataset is because we do not want to train on data that was affected by the testing set.

Part E - Best Classifier [4 pts]

Out of all 4 classifiers (for Random Forest and SVM take the best one from GridSearchCV for each), assess which one performed the best. Use accuracies, fit time or a combination of both in your reasoning. Put your explanation in *report.txt* under section “Part E”, using at most 50 words.

Deliverables

- **report.txt** - A text file containing your results and explanations for all parts.
- **run_classifiers.py** - Skeleton file filled with your code from Parts A-C.
- **preprocess_data.py** - Skeleton file filled with your code from Part D.
- **seizure_dataset.csv** - the dataset.

Submission Guidelines

Submit the deliverables as a single **zip** file named **hw4-*LastName-FirstName.zip*** (should start with lowercase hw4). Write down the name(s) of any students you have collaborated with on this assignment, using the text box on the T-Square submission page.

The zip file’s directory structure must exactly be (when unzipped):

```
hw4-LastName-FirstName/
      Q1/
            warmup.py
            out_warmup.bin
            out_warmup_bytes.txt
            pagerank.py
            pagerank_nodes_10.txt
            pagerank_nodes_25.txt
            pagerank_nodes_50.txt

      Q2/
            hw4-data.csv
            util.py
            decision_tree.py
            random_forest.py
```

```
Q3/
      report.txt
      run_classifiers.py
      preprocess_data.py
      seizure_dataset.csv
```

You must follow the naming convention specified above.