# CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Spring 2019

## Homework 4 : Scalable PageRank via Virtual Memory (MMap), Random Forest, SciKit Learn

Prepared by our 30+ wonderful TAs of CSE6242A,Q,OAN,O01,O3/CX4242A for our 1200+ students

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 (version number at bottom right of this document).
- ❏ Submit a single zipped file, called “HW4-GTusername.zip”, containing all the deliverables including source code/scripoints, data files, and readme. Example: “HW4-jdoe3.zip” if GT account username is “jdoe3”. Only .zip is allowed (no other format will be accepted). Your GT username is the one with letters and numbers.
- ❏ You may discuss high-level ideas with other students at the "whiteboard" level (e.g., how cross validation works, use hashmap instead of array) and review any relevant materials online. However, each student must write up and submit his or her own answers.
- ❏ All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute’s Academic Integrity procedures (i.e., reported to and directly handled by the Office of Student Integrity (OSI)). Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.
- ❏ At the end of this assignment, we have specified a folder structure you must use to organize your files in a single zipped file. 5 points will be deducted for not following this strictly.
- ❏ In your final zip file, 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).
- ❏ We may use auto-grading scripts to grade some of your deliverables, so it is extremely important that you strictly follow our requirements.
- ❏ Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
- ❏ Every homework assignment deliverable and every project deliverable comes with a 48-hour "grace period". Any deliverable submitted after the grace period will get zero credit. We recommend that you plan to finish by the beginning of the grace period in order to leave yourself time for any unexpected issues which might arise.
- ❏ 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. You may re-submit your work before the grace period expires. Canvas automatically appends a “version number” to files that you re-submit. You do not need to worry about these version numbers, and there is no need to delete old submissions. We will only grade the most recent submission.

## Download the HW4 Skeleton before you begin.

## Grading

The maximum possible score for this homework is 110 points.

## Q1 [30 pts] Scalable single-machine 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 lecture, 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. C++ and Java are generally faster than Python. However, several projects aim to boost Python speed. PyPy is one of them. We recommend that students use Python 3.6.x (or 3.7.x) for this question. Lower Python versions may be used (e.g., 2.7.x) if and only if you run into PyPy version incompatibility issues on your platform (e.g., Linux).

| | |
|---|---|
| Ubuntu | `sudo apt-get install pypy` |
| MacOS | Install Homebrew, then run `brew install pypy3`<br>(For Python 2: `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 pack the multiples of 5 in the range of 1 to 128 and their square root (in float format) into a binary file, and unpack it back into a memory map object. 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, run warmup.py and then run the following command to test whether it works as expected:

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

It `s `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). We recommend you revisit the MMap lecture 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, 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 big-endian “int” C type
`toy-graph.idx`: binary file containing (node, degree) in little-endian “int” C type
`toy-graph.json`: metadata about the conversion process (required to run pagerank)

In `toy-graph.bin` we have,

```
0000 0000 0000 0001          # 0 1  (in big-endian “int” C type)
0000 0001 0000 0000          # 1 0
0000 0001 0000 0002           # 1 2
0000 0002 0000 0001          # 2 1
0000 0003 0000 0004           # 3 4
0000 0004 0000 0005          # 4 5
0000 0005 0000 0002           # 5 2
ffff ffff ffff ffff
...
ffff ffff
```

In `toy-graph.idx` we have,

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

**Note:** there are extra values of -1 (`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.**

Next, run the following command to execute your PageRank implementation:

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

This will output the 10 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 number of iterations.

Find the output for the top 10 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 will be created. 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 they converge.

### 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 10 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 3.x 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.

The dataset you will use is extracted from the QSAR biodegradation Data Set where each record consists of the parameters of the chemical structure of various chemicals. 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**. Each line describes an instance using 42 columns: the first 41 columns represent the attributes of the molecule, and the

last column is the ground truth label which tells us if the molecule is biodegradable or not (1 means biodegradable, 0 means not biodegradable).
**Note:** The last column **should not** be treated as an attribute.

You will perform binary classification on the dataset to determine if a molecule is biodegradable or not.

## Essential Reading

### Decision Trees

To complete this question, you need to develop a good understanding of how decision trees work. We recommend you review the lecture on decision tree. 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 (also mentioned in lecture) 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 and the lecture on random forests. 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. 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()`

**Note**: You must achieve a minimum accuracy of 75% for x`the random forest.

**Note 2**: Remember to remove all print statements from the code. Nothing should be printed on the console. Failure to do so may result in point deduction.

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 lecture, 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 our grading efforts.

### Deliverables

1. **hw4-data.csv**: The dataset used to develop your program. Do not modify this file.
2. **util.py [10 pts]**: The source code of your utility functions.
3. **decision_tree.py [15 pts]**: The source code of your decision tree implementation.
4. **random_forest.py [25 pts]**: The source code of your random forest implementation with appropriate comments.

## Q3 [30 points] Using Scikit-Learn

**Note: You must use Python 3.7.x for this question.**

Scikit-learn is a popular Python library for machine learning. You will use it to train some classifiers on the EEG Eye State[1] dataset in the folder, called eeg_dataset.txt.

### Q3.1 - Classifier Setup [5 pts]

Train each of the following classifiers on the dataset, using the classes provided in the links below. You will do hyperparameter tuning in Q3.2 to get the best accuracy for each classifier on the dataset.

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

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 *hw4q3.py* to write your code.

In *report.txt*, under the relevant section, follow the skeleton and put your training and testing accuracies for each classifier. **Report your accuracies with two decimal places in the range of [0.00, 1.00]**, e.g. 0.78 and not 78, 78.00, 78%, etc. This should be easy since sklearn itself will give values in a decimal format less than 1.00.

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

1. Load dataset
2. Preprocess (you will do this in Q3.2)
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.

## Q3.2 - Hyperparameter Tuning [15 pts]

Tune your Random Forest and SVM to obtain their best accuracies on the dataset. For Random Forest, tune the model on the unmodified test and train datasets. For SVM, either standardize or normalize the dataset before using it to tune the model.

**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.

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 the relevant section of *report.txt*, state the values you tested for each hyperparameter.

Also follow the skeleton in *report.txt* to report the best combination of hyperparameter values for each classifier tuned, its testing accuracy from Q3.1, 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 Q3.1. **Report your accuracies with two decimal places in the range of [0.00, 1.00]**, e.g. 0.78 and not 78, 78.00, 78%, etc.

**Note:** If GridSearchCV is taking a long time to run for SVM, make sure you are standardizing or normalizing your data beforehand.

## Q3.3 - Cross-Validation Results [2 pts]

Let's practice getting the results of cross-validation. For your SVM (only), report the *mean training score, mean testing score* and *mean fit time* for the best combination of hyperparameter values that you obtained in Q3.2. The GridSearchCV class holds a 'cv_results_' dictionary that should help you report these metrics easily.

Report the metrics in *report.txt* under the relevant section. **Report your accuracies with two decimal places in the range of [0.00, 1.00]**, e.g. 0.78 and not 78, 78.00, 78%, etc.

## Q3.4 - Evaluate relative importance of features [2 pts]

You have performed a simple classification task using the Random Forest algorithm. You have also implemented the algorithm in Q2 above. The concept of entropy gain can also be used to evaluate the importance of a feature.

In this section you will determine the feature importance as evaluated by the Random Forest Classifier. You must then sort them in the descending order and print the feature numbers. Hint: There is a direct function available in sklearn to achieve this. Also checkout argsort() function in Python numpy. (argsort() returns the indices of the elements in ascending order)
You should use the first classifier that you trained initially in **Q3.1**, without any kind of hyperparameter-tuning, for reporting these features.
Report the most important and the least important features in the relevant section of *report.txt*. Mention the features with the exact name, e.g. X11, X1, etc. and not any index or value.

## Q3.5 - Best Classifier [4 pts]

Out of all 3 classifiers (for Random Forest and SVM take the best one from GridSearchCV for each), assess which one performed the best. Use testing accuracies, fit time or a combination of both in your reasoning. Put your explanation in *report.txt* under the relevant section, using at most 50 words.

## Q3.6 - Principal Component Analysis [2 pts]

Dimensionality reduction is an important task in many data analysis exercises and it involves projecting the data to a lower dimensional space using Singular Value Decomposition. Refer to the examples given here, set parameters *n_component* to 10 and *svd_solver* to 'full' (keep other parameters at their default value), and report the following in the relevant section of *report.txt*:

1. **Percentage** of variance explained by each of the selected components.
2. The singular values corresponding to each of the selected components.

You can copy your printed output arrays directly into the relevant section of the report.txt but **you must add commas after each value** so we can distinguish between individual values.

## Deliverables

- **report.txt** - A txt file containing your results for all parts.
- **hw4q3.py** - Skeleton file filled with your code from Q3.1-Q3.6.
- **eeg_dataset.csv** - the original dataset.

## Submission Guidelines

Submit the deliverables as a single **zip** file named **HW4-GTusername.zip**. Write down the name(s) of any students you have collaborated with on this assignment, using the text box on the Canvas submission page.

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

```
HW4-GTusername/
      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
               hw4q3.py
               eeg_dataset.csv
```

You must follow the naming convention specified above.

Version 5

[1] Derived from http://archive.ics.uci.edu/ml/datasets/EEG+Eye+State#