CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Spring 2021

HW 4: PageRank Algorithm, Random Forest, Scikit-learn

By our 35+ awesome TAs of CSE6242A,Q,OAN,O01,O3/CX4242A for our 1400+ students

Submission Instructions and Important Notes

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

1. Always check to make sure you are using the **most up-to-date assignment** (version number at bottom right of this document).
2. This advanced course expects students to submit code that runs and is free of syntax errors. Code that does not run successfully will receive 0 credit
3. **Submit ALL deliverables for this assignment via Gradescope**
   a. At the end of this assignment, we have specified a final checklist of the required files that must be submitted to for each question.
   b. Due to the large class size, we use auto-grading to grade your deliverables, to help speed up grading, so we can return graded work to you sooner. Thus, it is **extremely** important that you strictly follow the instructions.
   c. Do not include any intermediate files you may have generated while working on the task, unless your work is absolutely dependent on it to get the final result — there are rarely any situations that would justify such a need.
4. 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.
5. All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures (e.g., 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**.**
6. Submit your work by this assignment's official **Due** date on the course schedule.
   a. Every homework assignment deliverable comes with a 48-hour "grace period". We recommend submitting your work before the period begins. You do **not** need to ask before using this period.
   b. You may re-submit your work before the grace period expires **without penalty;** Gradescope will mark submissions made during the grace period as **"late"** (which does not incur a penalty).
   c. Each submission and its score will be recorded and saved by Gradescope. **By default, Gradescope uses your last submission for grading.** If you want to use a different submission, **you MUST "activate" it** (click "Submission History" button at bottom toolbar, then "Activate").
   d. We will **NOT** accept deliverables via any channels (e.g., Piazza) besides what we have specified.
   e. We will **NOT** accept any deliverables (or parts of a deliverable) after the grace period. To make sure you have submitted everything, verify that you have submitted something to each question. If your submitting large files, you are responsible for making sure they get uploaded to the system in time. You have 48 hours to verify your submissions!

Grading and Feedback

The maximum possible score for this homework is 100 points.

We will auto-grade all questions using the Gradescope platform. Based on our experience, students (you all!) benefit from using Gradescope to obtain feedback as they work on this assignment. **Keep the following important points in mind:**

1. You may upload your code periodically to Gradescope to obtain feedback for your code. This is accomplished by having Gradescope auto-grade your submission using the **same test cases** that we will use to grade your work. The test cases' results may help inform you of potential errors and ways to improve your code.
2. Gradescope should not be the primary way to test your code's correctness, since it provides only a few test cases, and error messages may not be as informative as local debuggers. You should test your code locally to more efficiently and effectively test your code, and only use Gradescope as a "final" check.
3. Gradescope cannot run code that contains syntax errors. If Gradescope is not running your code, **before seeking help**, verify that:
    a. Your code is free of syntax errors (by running it locally)
    b. All methods have been implemented
    c. You have submitted the correct file with the correct name
4. When many students use Gradescope simultaneously, it may slow down or fail to communicate with the tester. It can become even slower as the submission deadline approaches. You are responsible for submitting your work in time.

Download the HW4 Skeleton before you begin.

Homework Overview

Data analytics and machine learning both revolve around using computational models to capture relationships between variables and outcomes. In this assignment, you will code and fit a range of well-known models from scratch and learn to use a popular Python library for machine learning.

In Q1, you will implement the famous PageRank algorithm from scratch. PageRank can be thought of as a model for a system in which a person is surfing the web by choosing uniformly at random a link to click on at each successive webpage they visit. Assuming this is how we surf the web, what is the probability that we are on a particular webpage at any given moment? The PageRank algorithm assigns values to each webpage according to this probability distribution.

In Q2, you will implement Random Forests, a very common and widely successful classification model, from scratch. Random Forest classifiers also describe probability distributions—the conditional probability of a sample belonging to a particular class given some or all of its features.

Finally, in Q3, you will use the Python scikit-learn library to specify and fit a variety of supervised and unsupervised machine learning models.

Q1 [20 pts] Implementation of Page Rank Algorithm

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

In this question, you will implement the PageRank algorithm in Python for a large graph network dataset.

The PageRank algorithm was first proposed to rank web pages in search results. The basic assumption is that more "important" web pages are referenced more often by other pages and thus are ranked higher. The algorithm works by considering the number and "importance" of links pointing to a page, to estimate how important that page is. PageRank outputs a probability distribution over all web pages, representing the likelihood that a person randomly surfing the web (randomly clicking on links) would arrive at those pages.

As mentioned in the lectures, the PageRank values are the entries in the dominant eigenvector of the modified adjacency matrix in which each column's values adds up to 1 (i.e., "column normalized"), and this eigenvector can be calculated by the power iteration method that you will implement in this question, which iterates through the graph's edges multiple times to update the nodes' PageRank values ("pr_values" in pagerank.py) in each iteration.

For each iteration, the PageRank computation for each node is:

$$For\ each\ edge\ (v_i, v_j)\ from\ v_i\ to\ v_j, PR_{t+1}(v_j) = (1-d) \times Pd(v_j) + d \times \sum_{v_i} \frac{PR_t(v_i)}{out\ degree(v_i)}$$

Where:
v_j: node j
v_i: node i that points to node j
$out\ degree(v_i)$: the number of links going out of node v_i OR number of outbound links on a given node v_i
$PR_{t+1}(v_j)$: pagerank value of node j at iteration $t+1$
$PR_t(v_i)$: pagerank value of node i at iteration t
d: damping factor (set it to the common value of 0.85 that the surfer would continue to follow links)
$Pd(v_j)$: the probability of random jump which can be personalized based on use cases

You will be using the "network.tsv" graph network dataset in the hw4-skeleton/Q1 folder, which contains about 1 million nodes and 3 million edges. Each row in that file represents a directed edge in the graph. The edge's source node id is stored in the first column of the file, and the target node id is stored in the second column. **Note:** your code must **NOT** make any assumptions about the relative magnitude between the node ids of an edge. For example, suppose we find that the source node id is smaller than the target node id for most edges in a graph, we must NOT assume that this is always the case for all graphs (i.e., in other graphs, a source node id can be larger than a target node id).

You will complete the code in `submission.py` (guidelines also provided in the file).

- Step 1: in calculate_node_degree(), calculate and store each node's out-degree and the graph's maximum node id.
    - A node's out-degree is its number of outgoing edges. Store the out-degree in class variable "node_degree".
    - `max_node_id` refers to the highest node id in the graph. For example, suppose a graph contains the two edges (1,4) and (2,3), in the format of (source,target), the `max_node_id` here is 4. Store the maximum node id to class variable `max_node_id`.
- Step 2: implement `run_pagerank()`
    - For simplified PageRank algorithm, where $Pd(v_j)$ = `1/(max_node_id + 1)` is provided as `node_weights` in the script and you will submit the output for 10 and 25 iteration runs for a damping factor of 0.85. To verify, we are providing the sample output of 5 iterations for a simplified PageRank (simplified_pagerank_iter5_sample.txt**)**. For personalized PageRank, the $Pd()$ vector will be assigned values based on your 9-digit GTID (e.g., 987654321) and you will submit the output for 10 and 25 iteration runs for a damping factor of 0.85.

- The beginning of the main function in `submission.py` describes how to run the algorithm and generate output files. **Note:** When comparing your output for simplified_pagerank for 5 iterations with the given sample output, the absolute difference must be less than 5%, i.e., `Absolute(SampleOutput - YourOutput/SampleOutput)` must be less than `0.05`.

Deliverables:

1. **submission.py [12 pts]**: your modified implementation
2. **simplified_pagerank_iter{n}.txt**: 2 files (as given below) containing the top 10 node IDs (w.r.t the PageRank values) and their PageRank values for n iterations via the simplified_pagerank command

   simplified_pagerank_iter10.txt [2 pts]
   simplified_pagerank_iter25.txt [2 pts]

3. **personalized_pagerank_iter{n}.txt:** 2 files (as given below) containing the top 10 node IDs (w.r.t the PageRank values) and their PageRank values for n iterations via the personalized_pagerank command

   personalized_pagerank_iter10.txt [2 pts]
   personalized_pagerank_iter25.txt [2 pts]

Q2 [50 pts] Random Forest Classifier

Q2.1 - Random Forest Setup [45 pts]

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

For this problem you will be utilizing a Jupyter notebook and submitting a python script file. **Note:** Do **NOT** include any additional print statements in the notebook when you submit it on Gradescope; you may add print statements for debugging purposes while you work on the problem, but make sure to remove them before submission.

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 may only use the modules and libraries provided at the top of the notebook file included in the skeleton for Q2 and modules from the Python Standard Library. Python wrappers (or modules) must **NOT** be used for this assignment. Pandas must **NOT** be used — while we understand that they are useful libraries to learn, completing this question is not critically dependent on their functionality. In addition, to make grading more manageable and to enable our TAs to provide better, more consistent support to our students, we have decided to restrict the libraries accordingly.

The dataset you will use is pima-indians-diabetes.csv, a comma-separated (csv) file in the Q2 folder. The dataset was derived from National Institute of Diabetes and Digestive and Kidney Diseases. **You must not modify the dataset.** Each row describes one person (a data point, or data record) using 9 columns. The first 8 are attributes. The 9th is the label and you must **NOT** treat it as an attribute.

You will perform binary classification on the dataset to determine if a person has a diabetes.

Essential Reading

Decision Trees. To complete this question, you will develop a good understanding of how decision trees work. We recommend that you review the lecture on the decision tree. Specifically, review 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 the 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 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 data records (or data points) are left out of the bootstrap sample and not used in the construction of the *kth* tree. For each data record that is not used in the construction of the *kth* tree, it can be classified 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 record's predicted class is different from the true class, averaged over all such records, is the OOB error estimate.

While splitting a tree node, make sure to randomly select a subset of attributes (e.g., square root of the number of attributes) and pick the best splitting attribute (and splitting point of that attribute) among these subset of attributes. This randomization is the main difference between random forest and bagging decision trees.

Starter Code

We have prepared some Python starter code to help you load the data and evaluate your model. The starter file name is Q2.ipynb has three classes:

- `Utililty`: contains utility functions that help you build a decision tree
- `DecisionTree`: a decision tree class that you will use to build your random forest
- `RandomForest`: a random forest class

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. `Utililty class`: implement the functions to compute entropy, information gain, perform splitting, and find the best variable (attribute) and split-point. You can add additional methods for convenience
2. `DecisionTree class`: implement the `learn()` method to build your decision tree using the utility functions above.
3. `DecisionTree class`: implement the `classify()` method to predict the label of a test record using your decision tree.
4. `RandomForest class`: implement the methods `_bootstrapping()`, `fitting()`, `voting()` and `user()`.
5. `get_random_seed()`, `get_forest_size()`: implement the functions to return a random seed and forest size (number of decision trees) for your implementation

Note:

1. You must achieve a minimum accuracy of 70% for the random forest.
2. Your code must take no more than 5 minutes to execute (which is very long time, given the low program complexity), otherwise it may time out on Gradescope. Code that takes longer than 5 minutes to run likely means you need to correct inefficiencies (or incorrect logic) in your program. We suggest that you check the hyperparameter choices (e.g., tree depth, number of trees) and code logic when figuring out how to reduce runtime.
3. Remember to remove from your code **all print statements that you have added**. Failure to do so may result in point deduction. The `run()` function is provided to test your random forest implementation and do NOT change this function. You may also use any cells below the notebook2script cell for additional testing. Anything in these cells will be ignored when converting your notebook to a python script.

As you solve this question, consider the following design choices. Some may be more straightforward to determine, while some may be not (hint: study lecture materials and essential reading above). For example:

- Which attributes to use when building a tree?
- How to determine the split point for an attribute?
- How many trees should the forest contain?
- You may implement your decision tree using a data structure of your choice (e.g., dictionary, list, class member variables). However, your implementation must still work within the `DecisionTree` Class Structure we have provided.
- Your decision tree will be initialized using `DecisionTree(max_depth=10)`, in the `RandomForest` class in the jupyter notebook.
- When do you stop splitting leaf nodes?
- The depth found in the learn function is the depth of the current node/tree. You may want a check within the learn function that looks at the current depth and returns if the depth is greater than or equal to the max depth specified, otherwise it is possible that you continually split on nodes and create a messy tree. The max_depth parameter should be used as a stopping condition for when your tree should stop growing. Your decision tree will be instantiated with a depth of 0 (input to the learn() function in the jupyter notebook). To comply with this, make sure you implement the decision tree such that the root node starts at a depth of 0 and is built with increasing depth.

Note that, as mentioned in the lecture, there are other approaches to implement random forests. For example, instead of information gain, other popular choices include the 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.

Q2.2 - Random Forest Reflection [5 pts]

On Gradescope, answer the following question about the advantages of using a random forest classifier:

What is the main reason to use a random forest versus a decision tree?

Deliverables

1. **submission.py [45 pts]** Jupyter notebook will generate a Python script containing the source code of your utility functions, decision tree implementation, and random forest implementation with appropriate comments

Execute the final cell to export from a Jupyter Notebook to a python script. Remember, you are submitting a python script, NOT a notebook file.
%run helpers/notebook2script submission

2. **Random Forest Reflection [5 pts]**: Multiple-choice question completed on Gradescope

Q3 [30 points] Using Scikit-Learn

Note: You must use Python 3.7.x and Scikit-Learn v0.22 for this question. If you are currently working with different Python versions, conda can be an effective way to manage your environment and change your Python version.

Scikit-learn is a popular Python library for machine learning. You will use it to train some classifiers to predict diabetes in the Pima Indian tribe. The dataset is provided in the Q3 folder as pima-indians-diabetes.csv.

For this problem you will be utilizing a Jupyter notebook and submitting a python script file.

Note:
1. Your code must take no more than 15 minutes to execute all cells.
2. Do not import any libraries. All required libraries are already imported in the given Jupyter notebook.
3. Do not include any additional print statements in the notebook when you submit it on Gradescope; you may add print statements for debugging purposes while you work on the problem, but make sure to remove them before submission.
4. Use the default values while calling functions unless specific values are given.
5. Do not round off the results except the results obtained for Linear Regression Classifier.

Q3.1 - Data Import [2 pts]

In this step, you will import the pima-indians-diabetes dataset and allocate the data to two separate arrays. After importing the data set, you will split the data into a training and test set using the scikit-learn function train_test_split. You will use scikit-learns built-in machine learning algorithms to predict the accuracy of training and test set separately. Please refer to the hyper-links provided below for each algorithm for more details, such as the concepts behind these classifiers and how to implement them.

Q3.2 - Linear Regression Classifier [4 pts]

Q3.2.1 - Classification

Train the Linear Regression classifier on the dataset. You will provide the accuracy for both the test and train sets. Make sure that you round your predictions to a binary value of 0 or 1. See the Jupyter notebook for more information.

Q3.3 - Random Forest Classifier [10 pts]

Q3.3.1 - Classification

Train the Random Forest classifier on the dataset. You will provide the accuracy for both the test and train sets. Do not round your prediction.

Q3.3.2 - Feature Importance

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. You will determine the feature importance evaluated by the random

forest classifier in this section. Sort the features in descending order of feature importance score, and print the sorted features' numbers.

Hint: There is a function available in sklearn to achieve this. Also, take a look at `argsort()` function in Python numpy. `argsort()` returns the indices of the elements in ascending order. You will use the random forest classifier that you trained initially in Q3.3.1, without any kind of hyperparameter-tuning, for reporting these features.

Q3.3.3 - Hyper-Parameter Tuning
Tune your random forest hyper-parameters to obtain the highest accuracy possible on the dataset. Finally, train the model on the dataset using the tuned hyper-parameters. Tune the hyperparameters specified below, using the GridSearchCV function in Scikit library:

```
'n_estimators': [4, 16, 256], 'max_depth': [2, 8, 16]
```

Q3.4 - Support Vector Machine [10 pts]

Q3.4.1 - Preprocessing
For SVM, we will standardize attributes (features) in the dataset using StandardScaler, before training the model.

Note: for StandardScaler,

- Transform both x_train and x_test to obtain the standardized versions of both.
- Review the StandardScaler documentation, which provides details about standardization and how to implement it.

Q3.4.2 - Classification
Train the Support Vector Machine classifier on the dataset (the link points to SVC, a particular implementation of SVM by Scikit). You will provide the accuracy on both the test and train sets.

Q3.4.3. - Hyper-Parameter Tuning
Tune your SVM model to obtain the highest accuracy possible on the dataset. For SVM, tune the model on the standardized train dataset and evaluate the tuned model with the test dataset. Tune the hyperparameters specified below, using the GridSearchCV function in Scikit library:

```
'kernel':('linear', 'rbf'), 'C':[0.01, 0.1, 1.0]
```

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

Q3.4.4. - Cross-Validation Results
Let's practice obtaining the results of cross-validation for the SVM model. Report the rank test score and mean testing score for the best combination of hyper-parameter values that you obtained. The GridSearchCV class holds a `cv_results_` dictionary that helps you report these metrics easily.

Q3.5 - Principal Component Analysis [4 pts]
Performing Principal Component Analysis based dimensionality reduction is a common task in many data analysis tasks, 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 `8` and `svd_solver` to `full`. See the sample outputs below.

1. Percentage of variance explained by each of the selected components. Sample Output:

```
[6.51153033e-01 5.21914311e-02 2.11562330e-02 5.15967655e-03
6.23717966e-03 4.43578490e-04 9.77570944e-05 7.87968645e-06]
```

2. The singular values corresponding to each of the selected components. Sample Output:

```
[5673.123456  4532.123456   4321.68022725  1500.47665361
   1250.123456   750.123456    100.123456    30.123456]
```

Use the Jupyter notebook skeleton file called Q3.ipynb to write and execute your code.

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.

Deliverable

- **submission.py** - Jupyter notebook will generate a Python script with the classes from parts Q3.1 - Q3.5.

  Execute the final cell to export from a Jupyter Notebook to a python script. Remember, you are submitting a python script, NOT a notebook file.
  %run helpers/notebook2script submission

Extremely Important: Validate submitted files on Gradescope

We understand that some of you may work on this assignment until just prior to the deadline, rushing to submit your work before the submission window closes. **Please take the time** to validate that **all files** have been submitted for each question and that you have not forgotten to include any deliverables! If a deliverable is not submitted, you will receive **zero** credit for the affected portion of the assignment — this is a very sad way to lose points, since you have already done the work!

As a final check, verify that you have submitted the following files for each question in Gradescope:

```
HW4 - Q1:
            submission.py
            simplified_pagerank_iter10.txt
            simplified_pagerank_iter25.txt
            personalized_pagerank_iter10.txt
            personalized_pagerank_iter25.txt
HW4 - Q2.1:
            submission.py
HW4 - Q2.2:
            Complete the Gradescope question
HW4 - Q3:
            submission.py
```