cse6242-2019-fall-hw4

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

Homework 4 : PageRank algorithm, 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 100 points.

Q1 [20 pts] Implementation of Page Rank Algorithm

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

The PageRank algorithm was first proposed to rank web search results, so that more "important" web pages are ranked higher. It 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, which iterate through the graph's edges multiple times to updating the nodes' probabilities ('scores' in pagerank.py) in each iteration :
For each iteration, the Page rank computation for each node would be :

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

Where:

v_j : *node j*

v_i : *node i that points to node* v_j

$outdegree(v_i)$: *the number of links going out of node* v_i *OR number of outbound links on a given node* v_i

$PR_{t+1}\left(v_j\right)$: *probability of node j at iteration* $t+1$

$PR_t\ (v_i)$: *probability 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*

$Pd\left(v_j\right)$: *the probability of random jump which can be personalised based on use cases*

You will be using the dataset Wikipedia adminship election data which has almost 7K nodes and 100K edges. Also, you may find the dataset under the hw4-skeleton/Q1 as "soc-wiki-elec.edges"

In `pagerank.py`,

- You will be asked to implement the simplified PageRank algorithm, where $Pd(v_j) = 1/n$ in the script provided and need to submit the output for 10, 25 iteration runs.
  To verify, we are providing with the sample output of 5 iterations for simplified pagerank.

- For personalized PageRank, the $Pd()$ vector will be assigned values based on your 9 digit GTID (Eg: 987654321) and you are asked to submit the output for 10, 25 iteration runs.

Deliverables:

1. **pagerank.py [12 pts]**: your modified implementation
2. **simplified_pagerank_{n}.txt**: 2 files (as given below) containing the top 10 node IDs and their simplified pageranks for n iterations
   - **simplified_pagerank10.txt [2 pts]**
   - **simplified_pagerank25.txt [2 pts]**
3. **personalized_pagerank_{n}.txt:** 2 files (as given below) containing the top 10 node IDs and their simplified pageranks for n iterations
   - **personalized_pagerank10.txt [2 pts]**
   - **personalized_pagerank25.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.

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 .py files included in the skeleton for Q2 and modules from the Python Standard Library. Python wrappers (or modules) may NOT be used for this assignment. Pandas may 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 Predicting a Pulsar Star dataset. Each record consists of the parameters of a pulsar candidate. The dataset has been cleaned to remove missing attributes. The data is stored in a comma-separated file (csv) in your Q2 folder as **pulsar_stars.csv.** Each line describes an instance using 9 columns: the first 8 columns represent the attributes of the pulsar candidate, and the last column is the class which tells us if the candidate is a pulsar or not (1 means it is a pulsar, 0 means not a pulsar).
Note: The last column **should not** be treated as an attribute.
Note2: Do not modify the dataset.

You will perform binary classification on the dataset to determine if a pulsar candidate is a pulsar 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 methods `_bootstrapping()`, `fitting()`, `voting()`

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

Note 2: Your code must take no more than 5 minutes to execute.

Note 3: Remember to remove all of your print statements from the code. Nothing other than the existing print statements in **main()** should be printed on the console. Failure to do so may result in point deduction. Do not remove the existing print statements in **main()** in **random_forest.py**.

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.

Q2.2 - forest.txt [5 pts]

In **forest.txt**, report the following:

1. What is the main reason to use a random forest versus a decision tree? (<= 50 words)
2. How long did your random forest take to run? (in seconds)
3. What accuracy (to two decimal places, xx.xx%) were you able to obtain?

Deliverables

1. **util.py [10 pts]**: The source code of your utility functions.
2. **decision_tree.py [10 pts]**: The source code of your decision tree implementation.
3. **random_forest.py [25 pts]**: The source code of your random forest implementation with appropriate comments.
4. **forest.txt [5 pts]**: The text file containing your responses to Q2.2

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 *Predicting a Pulsar Star*[1] dataset which is provided in the hw4-skeleton/Q3 as *pulsar_star.csv*

Note: Your code must take no more than 15 minutes to execute all cells.

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

For any values we ask you to report in this question, please make sure to print them out in your Jupyter notebook such that they are outputted when we run your code.

NOTE: DO NOT ADD ANY ADDITIONAL CELLS TO THE JUPYTER NOTEBOOK AS IT WILL CAUSE THE AUTOGRADER TO FAIL.

NOTE: The below instructions will not match the exact flow of the Juypter notebook, you will have to find the section that applies to each of the different classifiers.

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 jupyter notebook skeleton file called **hw4q3.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.

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.

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 [4 pts]

Let’s practice getting the results of cross-validation. For your SVM (only), report the *rank test 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.

Q3.4 - Evaluate the relative importance of features [4 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 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.

.

Q3.5 - 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 8 and *svd_solver* to 'full' (keep other parameters at their default value), and report the following in the relevant section of hw4q3.ipynb:

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]

Deliverables

- **hw4q3.ipynb** - jupyter notebook file filled with your code for part Q3.1-Q3.5.

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/
            pagerank.py
            simplified_pagerank10.txt
            simplified_pagerank25.txt
            personalized_pagerank10.txt
            personalized_pagerank25.txt

      Q2/
            util.py
```

```
        decision_tree.py
        random_forest.py
        forest.txt

    Q3/
        hw4q3.ipynb
```

You must follow the naming convention specified above.

Version 3

---

[1] Derived from https://www.kaggle.com/pavanraj159/predicting-a-pulsar-star