cse6242-2020-spring-hw4

Updated automatically every 5 minutes

CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Spring 2020

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 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. Submit a single zipped file, called “HW4-GTusername.zip” that unzips to a folder called “HW4-GTusername”, containing all the deliverables including source code/scripts, data files, and readme. Example: “HW4-jdoe3.zip” if GT account username is “jdoe3”. Your GT username is the one with letters and numbers. Only .zip is allowed; no other format will be accepted.
    a. At the end of this assignment, we have specified a folder structure you must use to organize your files. 5 points will be deducted for not following this strictly.
    b. Due to the large class size, we may need to use auto-grading scripts to grade some of 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.
    d. Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
3. 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.
4. 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

48-hour “grace period”. You do **not** need to ask before using this grace period.
b. You may re-submit your work before the grace period expires **without penalty**, but Canvas will mark your submission as **"late"**.
c. 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.**
d. Any deliverable submitted after the grace period will get **0 credit**. We recommend that you submit your work before the grace period begins.
e. We will **not** consider late submission of any missing parts of a deliverable. To make sure you have submitted everything, download your submitted files to double check. If you're 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!

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

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 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 iterates through the graph’s edges multiple times to update 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:

cse6242-2020-spring-hw4 Updated automatically every 5 minutes

$PR_{t+1}(v_j)$: *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*

$Pd(v_j)$: *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 have to complete the code to perform the following steps (guidelines are also given in the pagerank.py)

  - Calculate the out-degree of each node and maximum node_id of the graph.

    Maximum node_id refers to the highest value you see for any node id, whether source or target. For eg: if edges are [(1,2),(3,4)] {in the format (source,target)}, the max_node_id here is 4.

  - Store the out-degree in class variable "node_degree" and maximum node id to "max_node_id"

- You will be asked to implement the simplified PageRank algorithm, where $Pd(v_j) = 1/n$ in the script provided and you are asked to submit the output for 10 and 25 iteration runs.

  To verify, we are providing the sample output of 5 iterations for a 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, and 25 iteration runs.

- Syntax to run the code is given in pagerank.py (First line of main function). The given code generates the output file as per the naming convention.

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

containing the top 10 node IDs and their personalized 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 the Churn prediction dataset. Each record consists of different attributes of a bank's customer. The dataset has been pre-processed and cleaned to remove missing attributes. The data is stored in a comma-separated file (csv) in your Q2 folder as **Churn.csv.** Each line describes a customer using 10 columns: the first 9 columns represent the attributes of the customer, and the last column is the label (1 means customer left the bank).

The original data was unbalanced and random undersampling has been performed to balance the data. You can read more about undersampling here.

Note 1: The last column **should not** be treated as an attribute.
Note 2: Do not modify the dataset.

You will perform binary classification on the dataset to determine if a customer is going to leave the bank or not.

Essential Reading

Decision Trees

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 overall records is the OOB error estimate.

While splitting a tree at a node, make sure to randomly select a subset of variable (You can use square root of number of attributes) and pick the best variable and split-point among the subset only. This is an important difference between random forest and bagging decision trees.

Starter Code

We have prepared a 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 the 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

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()` and `user()`.

Note 1: You must achieve a minimum accuracy of 70% 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 the **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 maybe 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 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 - 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

Scikit-learn is a popular Python library for machine learning. You will use it to train some classifiers on the Predicting a Pulsar Star 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. We've included the places in the notebook where a print statement is required.

Note: Do not add any additional print statements to the notebook, you may add them for debugging, but please make sure to remove any print statements that aren't required.

Q3.1 - Data Import and Cleansing Setup

In this step you will import the pulsar data set and allocate the data to two separate arrays. Once this is completed, you will then split the data into a training and test set using the scikit-learn function train_test_split. In the proceeding steps you will use scikit-learn's built in machine learning algorithms to predict accuracy of each from the given dataset. Each algorithm has additional documentation (which we've provided in the links) explaining them in more detail, such as how they work and how to use them.

Q3.2 - Linear Regression Classifier [4 pts]

Q3.2.1 - Classification

Linear Regression - Train the following classifier on the dataset, using the class provided in the link. You will need to provide accuracy on both the test and train sets. The challenge here will be making sure that you round your predictions to a binary 0 or 1. See the jupyter notebook for more information.

Q3.3 - Random Forest Classifier [10 pts]

Q3.3.1 - Classification

Random Forest - Train the following classifier on the dataset, using the class provided in the links You will need to provide accuracy on both the test and train sets. You will not be required to round your prediction in this section.

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. In this

the first classifier that you trained initially in Q3.1, without any kind of hyperparameter-tuning, for reporting these features.

Q3.3.3 - Hyper-Parameter Tuning

Tune your random forest to obtain their best accuracy on the dataset. For random forest, tune the model on the unmodified test and train datasets. Tune the hyperparameters specified below, using the GridSearchCV function that Scikit provides:

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

Q3.4 - Support Vector Machine [12 pts]

Q3.4.1 - Preprocessing

For SVM, we will standardize attributes (features) in the dataset before using it to tune the model.

Note:
For 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.
- Please see the link above for information and implementation instructions.

Q3.4.2 - Classification

Support Vector Machine (The link points to SVC, which is a particular implementation of SVM by Scikit.) Train the following classifier on the dataset, using the classes provided in the links. You will need to provide accuracy on both the test and train sets.

Q3.4.3. - Hyper-Parameter Tuning

Tune your SVM model to obtain their best accuracy on the dataset. For SVM, tune the model on the standardized test and train datasets. Tune the hyperparameters specified below, using the GridSearchCV function that Scikit provides:

'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 getting the results of cross-validation. For your SVM (only), 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 should help you report these metrics easily.

Q3.5 - Principal Component Analysis [4 pts]

Decomposition. Refer to the examples given here, set parameters n_component to 8 and svd_solver to 'full'. See the sample outputs below.

Note: For this section use the unmodified data(x_data)

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 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.

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/
```

cse6242-2020-spring-hw4

Updated automatically every 5 minutes

```
    personalized_pagerank25.txt

Q2/
    util.py
    decision_tree.py
    random_forest.py
    forest.txt

Q3/
    hw4q3.ipynb
```

Version 0