CSDS 440: Machine Learning Programming 1

General Instructions: Each programming problem is worth 100 points. From canvas, you can download the data files for the problems referred to in the questions below. Download 440data.zip and move the extracted folder (440data/) to the root of your repository. It will be ignored by git by default. Do not commit data files to GitHub.

Each datafile provided will have the following files:

- A "problem.names" file, which will list the attributes, their types and values. One attribute will be an "example-id" attribute which will be useful for identifying examples, but which you will not use when learning.
- A "problem.data" file, which will list all the examples and their class labels.
- A "problem.info" file, which gives additional information about the problem, such as how the data was generated. This is for your information only and does not affect the implementation in any way.

Your programs must be written in Python. You may *not* use external libraries that implement any significant component of the problems. You may *not* reference open source code, or copy snippets from other code not written by you. Besides standard system libraries, you *can* use libraries that give you access to data structures or libraries for math operations (like numpy or scipy) or libraries for optimization (like cvxopt). **Usable libraries have already been added to the python environment we provide. For any others, you must ask first before using them.**

Your commits are due on Github by 11:59pm on the due dates specified. You will receive a 10% bonus if your final commit is turned in a week or more in advance of the due date. **This commit must be the only one logged with the comment "Final Commit".** You can use one late day each week (up to Saturday 11:59pm) with a penalty of 20%. Submissions after Saturday 11:59pm for any week will not be graded.

We will use the git logs to ensure that each person contributed equally to each assignment. **Therefore, any code written by you must be clearly logged under your own name/network ID by git. You will not receive credit for code committed by anyone other than yourself even if you wrote it, or if we cannot make out who committed the code. There will be no exceptions to this rule.**

Repository setup: You can access your repositories through github. You should have accepted invites from the cwru-courses organization. You should then see your repositories, named CSDS440-f22-n (n is your group number), for these assignments. You will commit your code to these repositories. In the repositories, a basic organizational template has already been provided for you. Further, an abstract framework has been provided in Python which can parse the data files and set up a basic data structure. Python skeletons have also been provided for your implementation. There is a file called util.py where you should implement helper functions for cross validation, training and testing, computing metrics, and plotting graphs. This code will be reused for each assignment.

In this exercise, you will implement and evaluate a decision tree learning algorithm.

a. Decision Tree Learner (60 points)

Implement the ID3 decision tree algorithm discussed in class. Your implementation should be able to handle nominal and continuous attributes, and use both information gain and gain ratio as the split criterion (depending on a provided option below). You will also need to implement stratified cross validation as described below. The main file should be called **dtree** (skeleton is in the src/ directory in your repository).

Your code should take four options:

1. The path to the data. If this is "/a/b/someproblem" then you will load "/a/b/someproblem.names" and "/a/b/someproblem.data".
2. A nonnegative integer that sets the maximum depth of the tree (the number of tests on any path from root to leaf). If this value is zero, you should grow the full tree. If this value is positive, grow the tree to the given value. Note that if this value is too large, it will have the same effect as when the option is zero.
3. A flag called --no-cv which disables cross validation and runs on the full dataset. If the flag is not present, run cross validation by default.
4. A flag called --gain-ratio which uses gain ratio as the split criterion. If the flag is not present, use information gain by default.

For example, running dtree.py 440data/volcano 5 --gain-ratio will train on the volcano dataset in the path 440data/ with a tree depth limit of 5, will use cross validation, and will use gain ratio as the split criterion. When your code is run, it should first construct 5 folds using stratified cross validation (implement the function *cv_split* in *utils.py*) if the no-cv flag is not provided. To ensure repeatability, set the random seed for the PRNG to 12345. Then it should produce decision trees on each fold (or the sample according to the option) and report the following.

b. Output format

When your code is run on any problem, it must produce output in the following format:

Accuracy: 0.xyz

Size: xyz

Maximum Depth: xyz

First Feature: <name>

"Accuracy" is the (average) fraction of examples in the test sets (if cross validating) or training set (if not) that were correctly classified by the learned decision tree. You can implement this in the function *accuracy* in *utils.py* . "Size" is the size of the tree in number of nodes, and maximum depth is the length of the longest sequence of tests from root to leaf. The "First Feature" is the name of the first feature that was used to partition the data, or "None" if the tree was empty.

c. Jupyter Notebook/pdf Writeup (40 points)

Prepare a jupyter notebook on your experiments. In the notebook, answer the following questions. “CV” means “cross validated”:

(a) Show the output on each dataset when the depth is set to 1 to 5. Use a separate cell for each dataset. (5 points)

(b) For *spam* and *voting*, print the first test picked by your tree. Is this a sensible test? Explain why or why not. (5 points)

(c) For *volcanoes* and *spam*, plot the CV accuracy as the depth of the tree is increased. On the x-axis, choose depth values 1, 2, 4, 8, 16 and 32. Can you explain the pattern of the graph? (10 points)

(d) **Research extension.** Create one research hypothesis on trees and variations and evaluate it empirically. For example, you might create hypotheses around different split criteria, the optimality of the ID3 algorithm, different ways to handle pruning, etc. Your hypothesis may require you to implement other algorithms beyond the ones above. You will be evaluated on originality, technical strength, insightfulness of the observations you generate and the clarity of your discussion. (20 points)

Place your code in the src/ directory and notebook in the notebooks/ directory and push to github. Include a short README file containing your experience with the skeleton/utility code provided and documentation, and anything you found confusing. Do not commit any other files.

d. Grading Rubric

Your code must be efficient, cleanly written and easy to understand. For python, try to follow PEP8 as far as feasible. The code should handle errors and corner cases properly. You will lose points if the TAs cannot follow the logic easily or if your code breaks during testing. Note that we will test your code on data other than what is provided, and the TAs will not have time to debug your code if it breaks. Your code should work on the Python environment you were given.

Generally, point deductions will follow these criteria:

- Incomplete implementation/Not following assignment description: up to 100%
- Syntax Errors/Errors causing the code to fail to compile/run:
    - Works with minor fix: 15%
    - Does not work even with minor fixes: 75%
- Inefficient implementation: 15%
    - Algorithm takes unreasonably long to run during grading: additional 10%
- Poor code design: 20%
    - Examples:
        - Hard-to-follow logic
        - Lack of modularity/encapsulation
        - Imperative/ad-hoc/"spaghetti" code
        - Duplicate code
- Poor UI:
    - Bad input (inadequate exception handling, no `--help` flag, etc.): 10%
    - Bad output (overly verbose `print` debugging statements, unclear program output): 10%

- Poor code style (substantially not PEP8): 5%
- Poor documentation: 20%
- Non-code commits: 2%
    - Examples:
        - Committing data files
        - Committing non-source files (`.idea` files, `.iml` files, etc.)
    - **Hint:** use your .gitignore file!
- Not being able to identify git contributor by their name or case ID: 5% per commit
- Code not in src/: 2%
- Notebook not in notebooks/: 2%

Bonus points may be awarded for the following:

- Exceptionally well-documented code
- Exceptionally well-written code
- Exceptionally efficient code (Takes advantage of C/FORTRAN numpy optimizations, using numba, etc.)
- **Hint:** use pure python (`for` loops, etc.) as minimally as possible!