CSDS 440: Machine Learning Programming 2

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.

In the csds440-grouptemplate zip file on canvas, 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.

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 requirements.txt file in csds440-grouptemplate. 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.**

In this problem, you will implement and evaluate (i) naïve Bayes and (ii) logistic regression.

a. Naïve Bayes (30 points)

Implement the naïve Bayes algorithm discussed in class. Discretize continuous values. To do this, partition the range of the feature into k bins (value set through an option). To do this, divide the range of the feature into k equal-length disjoint intervals. The x^{th} bin is the x^{th} interval, so replace the original feature with a discrete feature that takes value x if the original feature's value maps to bin x. Use m-estimates to smooth your probability estimates. Use logs whenever possible to avoid multiplying too many probabilities together. Your main file should be called **nbayes**. Use the same framework/code provided in the first assignment. Your program should take four options:

1. The path to the data (see the first assignment).
2. The –no-cv option (see the first assignment).
3. A positive integer (at least 2), which is the number of bins for any continuous feature.
4. A nonnegative integer m for the m-estimate. If this value is negative, use Laplace smoothing. Note that m=0 is maximum likelihood estimation. The value of p in the m-estimate should be fixed to $1/v$ for a variable with v values.

When your code is run, it should first construct 5 folds using stratified cross validation if this option is provided. To ensure repeatability, set the random seed for the PRNG to 12345. Then it should produce naïve Bayes models on each fold (or the sample according to the option) and report as in part (c).

b. Logistic Regression (30 points)

Implement the logistic regression algorithm described in class. During learning, minimize the negative conditional log likelihood plus a constant (λ) times a penalty term, half of the 2-norm of the weights squared. You can use standard gradient descent for the minimization. Nominal attributes should be encoded as 1-of-N vectors. The main file should be called **logreg**. It should take three options: the first two options above and a third which is a nonnegative real number that sets the value of the constant λ. The same notes about 5 fold stratified CV from above, etc. apply in this case.

c. Output format

When either algorithm is run on any problem, it must produce output in exactly the following format:

Accuracy: 0.xyz 0.abc

Precision: 0.xyz 0.abc

Recall: 0.xyz 0.abc

Area under ROC: 0.xyz

For all metrics expect Area under ROC, "0.xyz" is the average value of each quantity over five folds. "0.abc" is the standard deviation. For Area under ROC, use the "pooling" method. Here, after running the classifier on each fold, store all the test examples' classes and confidence values in an array. After all folds

are done, use this global array to calculate the area. To calculate the area under ROC, first calculate the TP and FP rates at each confidence level, using the numeric value output by the classifier as the confidence. Each pair of adjacent points is joined by a line, so the area under the curve is the sum over the areas of all trapezoids bounded by the FP rates of the adjacent points, the TP rates of the adjacent points, and the line joining the TP rates.

d. Jupyter Notebook (40 points)

Prepare a jupyter notebook on your experiments. In the notebook, answer the following questions.

(a) Create a table of the different metrics for naïve Bayes (m=0.01) and logistic regression (λ =0.01) on the three datasets. Use a different cell for each dataset. For naïve Bayes, set the number of bins for continuous features to 10. (10 points)

(b) Examine the effect of λ on logistic regression. Do this by graphing area under ROC across λ=0, 0.001, 0.01, and 0.1 on the given datasets. (10 points)

(c) **Research extension.** Create one research hypothesis on naïve Bayes and/or logistic regression and evaluate it empirically. For example, you might test the effect of non-independent features, or different smoothing strategies for the probabilities, or see how accurately naïve Bayes can generate data/estimate densities 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 (s)/pdf(s) 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: 10%
    - Does not work even with minor fixes: 75%
- Inefficient implementation: 10%
    - 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): 3%
- Poor documentation: 10%
- Non-code commits: 3%
    - 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/: 3%
- Notebook/pdf not in notebooks/: 3%

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!