CS 534: Homework #2

Submission Instructions: The homework is due on Oct 4th at 11:59 PM ET on Gradescope. A part of your homework will be automatically graded by a Python autograder. The autograder will support Python 3.10. Additional packages and their versions can be found in the `requirements.txt`. Please be aware that the use of other packages and/or versions outside of those in the file may cause your homework to fail some test cases due to incompatible method calls or the inability to import the module. We have split homework 1 into 2 parts on Gradescope, the autograded portion and the written answer portion. If *either of the two parts is late, then your homework is late.*

1. **Upload PDF to HW2-Written Assignment**: Create a single high-quality PDF with your solutions to the non-coding problems. The solutions must be *typed* (e.g., Word, Google Docs, or LaTeX) and each problem appropriately tagged on Gradescope. If we must search through your entire PDF for each problem, you may lose points! Note that you *must submit the code used to generate your results in the Code part of the assignment, otherwise you may not get any points for the results.*

2. **Submit code to the HW2-Code Assignment**: Your submitted code must contain the following files: 'q2.py', 'q3.py', 'README.txt'. You must submit *ALL files you used to generate the results* but the autograder will only copy these files when running the test cases so make sure they are self-contained (i.e., capable of running standalone). Make sure you always upload *ALL* of these files when you (re)submit. The `README.txt` file *must contain a signed honor statement* that contains the following words:

   ```
   /* THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT
   CONSULTING CODE WRITTEN BY OTHER STUDENTS
   OR LARGE LANGUAGE MODELS LIKE CHATGPT.
   Your_Name_Here */

   I collaborated with the following classmates for this homework:
   <names of classmates>
   ```

1. **(3+3+2+2=10 pts) Bias-Variance Trade-off of LASSO**

   While it is hard to write the explicit formula for the bias and variance of using LASSO, we can quantify the expected general trend. Make sure you justify the answers to the following questions for full points:

   (a) (Written) What is the general trend of the bias as λ increases?

   (b) (Written) What about the general trend of the variance as λ increases?

   (c) (Written) What is the bias at $\lambda = 0$?

   (d) (Written) What about the variance at $\lambda = \infty$?

2. **(2×4+4+3+4+3+6+4=32 pts) Spam classification using Naive Bayes and Standard Logistic Regression**

   Consider the email spam dataset, which contains 4601 e-mail messages that have been split into 3000 training (`spam.train.dat`) and 1601 test emails (`spam.test.dat`). 57 features have been extracted with a binary label in the last column. You can read more about the data at

the UCI repository (`http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection`). The features are as follows:

- 48 continuous real [0,100] attributes of type word freq WORD = percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
- 6 continuous real [0,100] attributes of type char freq CHAR = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurrences) / total characters in e-mail
- 1 continuous real [1,...] attribute of type capital run length average = average length of uninterrupted sequences of capital letters
- 1 continuous integer [1,...] attribute of type capital run length longest = length of longest uninterrupted sequence of capital letters
- 1 continuous integer [1,...] attribute of type capital run length total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail
- 1 nominal 0,1 class attribute of type spam = denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

All the specified functions should be in the file 'q2.py'.

(a) (Code) You will explore the effects of feature preprocessing and its impact on Naive Bayes and Standard (unregularized) logistic regression. Write the following functions to preprocess your data. You are free to use the `sklearn.preprocessing` module. Note that they are independent of one another and should not build on each step. You should assume only the features are passed in and not the target and the function should return the preprocessed train and test set in numpy 2D array format (i.e., two return values).

   i. Write a Python function `do_nothing(train, test)` that takes a train and test set and does no preprocessing.

   ii. Write a Python function `do_std(train, test)` that standardizes the columns so they all have mean 0 and unit variance. Note that you want to apply the transformation you learned on the training data to the test data. In other words, the test data may not have a mean of 0 and unit variance.

   iii. Write a Python function `do_log(train, test)` that transforms the features using $\log(x_{ij} + 0.1)$ or a smoothed version of the natural logarithm.

   iv. Write a Python function `do_bin(train, test)` that binarize the features using $\mathbb{1}_{(x_{ij}>0)}$ (Note that $\mathbb{1}$ denotes the indicator function). In other words, if the feature has a positive value, the new feature is a 1 otherwise a 0.

(b) (Code) Write a Python function `eval_nb(trainx, trainy, testx, testy)` that fits a Naive Bayes model to the training. You can use `sklearn.naive_bayes` module for this part. The function should return as a dictionary containing the accuracy and AUC for the training and test sets and the predicted probabilities for the test set using the following keys: 'train-acc', 'train-auc', 'test-acc', 'test-auc', 'test-prob'. The values for accuracy and AUC are expected to be numeric values, while test-prob is expected to be either a list or a numpy 1-d array with the predicted probability for the test set.

(c) (Written) Fit a Naive Bayes model to each of the four preprocessing steps above using the code in 2b. Each preprocessing should be performed *independently* (i.e., use each of the functions you created in 2a on the original dataset). Report the accuracy rate and AUC on the training and test sets across the 4 preprocessing steps in a table.

(d) (Code) Write a Python function `eval_lr(trainx, trainy, testx, testy)` that fits a standard (no regularization) logistic regression model. The function should return a dictionary containing the accuracy and AUC for the training and test sets and the predicted probabilities for the test set using the following keys: 'train-acc', 'train-auc', 'test-acc', 'test-auc', 'test-prob'. Note that the values for accuracy and AUC should be numeric values, while test-prob should either be a list or a numpy 1-d array with the predicted probability for the test set. The output will be the same format as 2b.

(e) (Written) Fit a standard (no regularization) logistic regression model to each of the four preprocessing steps above using the code in 2d. Report the accuracy rate and AUC on the training and test sets for the 4 preprocessing steps in a table.

(f) (Written) Plot the receiver operating characteristic (ROC) curves for the test data. You should generate 3 plots:

- One plot containing the 4 Naive Bayes model curves representing each of the preprocessing steps.
- One plot containing the 4 logistic regression model curves representing each of the preprocessing steps.
- One plot containing the best Naive Bayes model and the best logistic regression model curve.

(g) (Written) Given your results in 2c, 2e, and 2f, comment on how the preprocessing affects the models (logistic and Naive Bayes) with regards to ROC, AUC, and accuracy. Also, comment on how Naive Bayes compares with logistic regression.

3. **(2+4+5+5+8+1+6+8+5+6+4+4=54 pts) Exploring Model Selection Strategies for Logistic Regression with Regularization**

We will be using the SPAM dataset from the previous part for this problem. You can preprocess the data however you see fit, either based on the results of the previous problem or by introducing another preprocessing method. The only requirement is that it is consistent throughout the rest of this problem. For this problem, you are not allowed to use the `sklearn.model_selection` module. All the specified functions should be in the file 'q3.py'.

(a) (Written) How did you preprocess the data for this method? Why?

(b) (Code) Implement the Python function `generate_train_val(x, y, valsize)` that given the validation size splits the data randomly into train and validation splits. The function should return a dictionary with the following keys: 'train-x', 'train-y', 'val-x', 'val-y'. The values for 'train-x' and 'val-x' are expected to be numpy 2d arrays of the same dimension as x, and split into the associated training and validation features. The values for 'train-y' and 'val-y' are expected to be a subset of y split accordingly. Note that each time this function is run, the splits are likely to be different.

(c) (Code) Implement the Python function `generate_kfold(x, y, k)` that given the k, will split the data into k-folds. The function should return a single numpy 1-d array with the k that it belongs to (e.g., array([0,1,2,...,2,1,1])) for k=3.

(d) (Code) Implement the Python function `eval_holdout(x, y, valsize, logistic)` which takes in the input (e.g., 3000 training samples from `spam.train.dat`), the input labels, and the validation size and (1) uses 3b to split x and y into train and validation, and (2) evaluates the performance using the instantiated logistic regression model. You can assume the logistic regression classifier will be created using `sklearn.linear_model.LogisticRegression`. Your function should report a dictionary containing the accuracy and AUC for the training and validation sets using the following keys: 'train-acc', 'train-auc', 'val-acc', 'val-auc'.

(e) (Code) Implement the Python function `eval_kfold(x, y, k, logistic)` which takes in k, (1) uses 3c to split the data, and (2) evaluates the performance using the instantiated in the logistic regression model. You can assume the logistic regression classifier will be created using `sklearn.linear_model.LogisticRegression`. Your function should report a dictionary containing the accuracy and AUC for the training and validation sets using the following keys: 'train-acc', 'train-auc', 'val-acc', 'val-auc'. The accuracy and AUC for this part should be a single number (e.g., how you aggregate across the k folds).

(f) (Written) What is your regularization parameter search space for ridge and LASSO-regularized logistic regression?

(g) (Written) For your regularization parameter search space specified in 3f, fit ridge and LASSO using 3d by searching over a variety of split ratios. For each unique split ratio, report the validation metrics (AUC and accuracy). What is the best 'parameter' for ridge and LASSO based on these metrics? Note ridge and LASSO may have different optimal 'parameters'.

(h) (Written) For your regularization parameter search space specified in 3f, fit ridge and LASSO using the k-fold cross-validation approach by searching over $k = 2, 5, 10$. For each value of k, specify the best 'parameter' based on the metrics.

(i) (Code) Implement the Python function `eval_mccv(x, y, valsize, s, logistic)` that takes in the validation size and the sample size s and uses the Monte Carlo cross-validation approach with s samples (i.e., use the validation/hold-out technique from 3d s times). The output is expected to be the same format as 3d and 3e.

(j) (Written) Fit ridge and LASSO using the Monte Carlo Cross-validation approach with $s = 5, 10$ samples and different split ratios. What is the best 'parameter' based on the splits and the samples?

(k) (Written) Using the optimal parameters identified in 3g, 3h, and 3j, re-train the regularized logistic regression model using *all* the training data and report the performance on the test set in terms of AUC and accuracy in a table. Note that this means you are likely training *at least 6* different regularized models and reporting the performance for each of the model.

(l) (Written) Comment on how the different model selection techniques compare with one another with regard to AUC and accuracy, the robustness of the validation estimate, and the computational complexities of the three different hold-out techniques.