CS 534: Homework #3

Submission Instructions: The homework is due on Oct 19th 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 the homework 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 HW3-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 HW3-Code Assignment**: Your submitted code must contain the following files: ‘preprocess.py’, ‘q2.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. **(2+3+2+2+2+6=17pts) Preprocessing Data for Loan Default Prediction**

   We will consider a simplified subset of the Lending Club Loan Data that was previously available on Kaggle (`https://www.kaggle.com/datasets/wordsforthewise/lending-club`) for the next few problems. The goal is to determine whether or not an individual applicant will default on the loan (i.e., unable to make the required payments to pay back the borrowed money). Using ideas from a Kaggle Kernel by Kacper Wozniak[1], we have preprocessed the dataset the following steps:

   - Removed any column that has more than 50% of the observations missing
   - Remove any row with at least one missing value
   - Removed any loans that are not Fully Paid or Charged Off (defaulted)
   - Only used individual applicants (i.e., join types are ignored)

[1] The kernel is available at `https://www.kaggle.com/kacpiw/who-default-on-a-loan-logistic-regression`

- Removed columns referring to “Active” loan accounts, joint applicants, or with too many unique values that preprocessing may take longer than necessary (i.e., policy code, employee title)
- Convert the loan status to a binary variable (`class`) as Fully Paid (0) vs Charged Off (1)
- Subsample the data for a smaller dataset

The resulting dataset (`loan_default.csv`) contains 2850 loans with 26 features. The goal of this problem is to prepare the data for the next problem. The functions specified should be in the file ‘preprocess.py’.

(a) (Written) Why would you potentially be interested in optimizing for F_2 score compared to F_1 score in the context of finding individuals that are likely to default?

(b) (Written) How will you partition the loan data to assess the model performance and choose hyperparameter? Justify your choice and write code to partition the data accordingly.

(c) (Code) Write a function `compute_correlation(x, corrtype)` that takes in a numpy 2D array, x, the string of the correlation type (i.e., ‘pearson’, ‘kendall’, and ‘spearman’) to calculate and returns a numpy 2D matrix of the correlation between the features of x.

(d) (Code) Write a function `rank_correlation(x, y)` that takes in a numpy 2d array, x, a numpy 1d array, y, and returns the rank of the features in x based on correlation criteria presented in the class. The function should output a numpy array in descending order of the most correlated feature column (i.e., [3, 1, 0, 2] would suggest the 4th feature has the highest rank correlation followed by 2nd, then 1st feature.)

(e) (Code) Write a function `rank_mutual(x, y)` that that takes in a numpy 2d array, x, a numpy 1d array, y, and returns the rank of the features in x based on the mutual information presented in the class. The function output format is the same as 1d which is the descending order of the most correlated feature column based on mutual information.

(f) (Written) Using 1c, 1d, 1e, perform feature selection using the three different methods independently. For each one, justify your selection (e.g., what you are using as thresholds for discarding highly correlated variables and the metric or what you’ll choose as the top k ranked features).

2. **(4+6+3+3+6+2=24 pts) Decision Tree to Predict Loan Defaults**

Use the **sklearn** decision tree package to predict the loan status. The functions specified should be in the file ‘q2.py’

(a) (Code) Write a function `tune_dt(x, y, dparams, lsparams)` that takes in the data where x is a numpy 2d array and y is a numpy 1d array (assume test as already been set aside for this purpose), a list of max depth values for the decision tree, and a list of a minimum number of samples that each leaf must contain and determines the optimal parameters based on grid search and the AUC metric. Your function should return a dictionary with the keys “best-depth”, “best-leaf-samples” and the optimal value from the two lists (i.e., a single value for each setting). Your dictionary can have other keys but must have these 2 keys.

(b) (Written) What is the best choice of the two parameters based on 2a? Plot the "validation" AUC (based on 1b) on a *single* plot (either do a 3D plot or use the size of the point to encode the AUC) as a function of the two parameters.

(c) (Written) Re-train a decision tree on the entire training data using the optimal max depth and the minimum number of samples from 2b. What are the AUC, F_1, and F_2 score on the test set?

(d) (Written) Create a visualization of the top 3 levels of your decision tree from 2c.

(e) (Written) Analyze the effects of three different filtering methods, 1c, 1d, 1e, independently on the decision tree by determining the optimal parameters using 2a and then re-training the data on the entire training data in 2c and report the AUC, F_1, and F_2 score in a table.

(f) (Written) Comment on the effect of feature selection for decision trees based on your results from 2c and 2e.

3. **(3 + 5 + 2 + 4 + 8 + 4 + 5 + 8 + 2 + 4 + 5 + 5 + 4 = 59 pts) Spam Detection via Perceptron**

We have provided a new e-mail spam dataset `spamAssassin.data`, which is a subset of the SpamAssassin Public Corpus (see `https://spamassassin.apache.org/old/publiccorpus/`). Here is a sample email that contains a URL, an email address (at the end), numbers and dollar amounts.

```
> Anyone knows how much it costs to host a web portal ?
> Well, it depends on how many visitors youre expecting. This can be anywhere
from less than 10 bucks a month to a couple of $100. You should checkout
http://www.rackspace.com/ or perhaps Amazon EC2 if youre running something big...

To unsubscribe yourself from this mailing list,
send an email to: groupnameunsubscribe@egroups.com
```

We have already implemented the following email preprocessing steps: lower-casing; removal of HTML tags; normalization of URLs, e-mail addresses, and numbers. In addition, words have been reduced to their stemmed form. For example, "discount", "discounts", "discounted" and "discounting" are all replaced with "discount". Finally, we removed all non-words and punctuation. The result of these preprocessing steps on the same email is shown below:

```
anyon know how much it cost to host a web portal well it depend on how mani visitor
your expect thi can be anywher from less than number buck a month to a coupl of
dollarnumb you should checkout httpaddr or perhap amazon ecnumb if your run someth
big to unsubscrib yourself from thi mail list send an email to emailaddr
```

For this problem, you will implement the Perceptron algorithm and apply it to the problem of e-mail spam classification. You'll be comparing two different variants of the algorithm as well as the number of epochs through the data.

(a) (Written) What is your model assessment strategy? Justify your validation methodology. (You may want to read the rest of this problem before you proceed to understand what your tasks will be).

(b) (Code) Implement a Python function `build_vocab(train, test, minn)` that does the following steps. (1) builds a vocabulary list by finding all the words that occur across the training set and ignores all words that appear in fewer than minn emails and (2) transforms both the train and test emails into a feature vector $\mathbf{x}$ where the ith entry, x_i, is 1 if the ith word in the vocabulary occurs in the email, and 0 otherwise. The function output should be (1) the transformed training dataset which is a numpy 2D array of size $n \times p$ where p is the number of words that occur at least in minn emails and n is the number of emails in the training data, (2) the transformed test dataset which is a numpy 2D array of size $m \times p$ where m is the number of emails in the test data and p is determined from training, and (3) a dictionary that maps the word to the column feature to the word (e.g., {know:0, cost:1, ..., emailaddr: 100}).

(c) (Code) Implement the function `get_weights` that returns the learned perceptron weights, $\mathbf{w}$.

(d) (Code) Implement the function `sample_update` that given a single sample and the data label returns the new weight $\mathbf{w}$ in the same format as 3c and whether or not there was a mistake for the sample (1 for mistake, 0 for no mistake). For the corner case of $\mathbf{w} \cdot \mathbf{x} = 0$, predict the $+1$ class.

(e) (Code) Implement the `train` function of the perceptron algorithm that uses the examples provided to the function. This function should assume the bias term will be automatically folded into $\mathbf{w}$ by adding a constant feature (i.e., a column with all 1) such that the input is $n \times (p + 1)$ if it was to be run after 3b. The function should output a dictionary for each epoch (pass through the data) with the number of mistakes associated with each epoch (epochs are expected to start at 1). The algorithm will terminate either when all the points are correctly classified or the max number of epochs have been reached.

(f) (Code) Implement the `predict` function of the perceptron algorithm where given new data points, predicts the label. The output should be a numpy 1D array of size m.

(g) (Written) Train a perceptron using your training set specified in 3a. Plot the training and estimated generalization error as a function of the maximum number of epochs. What is the optimal algorithm and parameter? How many mistakes are made before the algorithm terminates if you want to classify all the training points properly?

(h) (Code) Implement the `train` function of the averaged perceptron algorithm. Note that the AvgPerceptron class inherits from the Perceptron class so it can re-use methods that exist (if it makes sense). Rather than the current implementation in 3e, this one keeps track of the average of all the weight vectors considered during the algorithm, including examples where no mistake was made. The average weight vector is the "learned" coefficient in the end as averaging reduces the variance between the different weight vectors, and is a powerful means of preventing the learning algorithm from overfitting (serving as a type of regularization). Note that only the learned coefficient reflects the averaged weights, but internal updates still use the same weight updates $\mathbf{w}$ in 3e.

(i) (Code) Implement the `get_weights` function of the averaged perceptron algorithm. It should return the average weight vector.

(j) (Code) Implement the `predict` function of the averaged perceptron algorithm. The prediction function should use the updated average weights.

(k) (Written) Plot the training and estimated generalization error as a function of the maximum number of epochs for the averaged perceptron.

(l) (Written) What is your final or "optimal" algorithm? In other words, train the model with as much data as you can possibly with the optimal algorithm + hyperparameter (maximum number of epochs) values. What is the expected predictive performance of this model?

(m) (Written) Using the vocabulary list together with the parameters learned in the previous question, output the 15 words with the most positive weights. What are they? Which 15 words have the most negative weights?