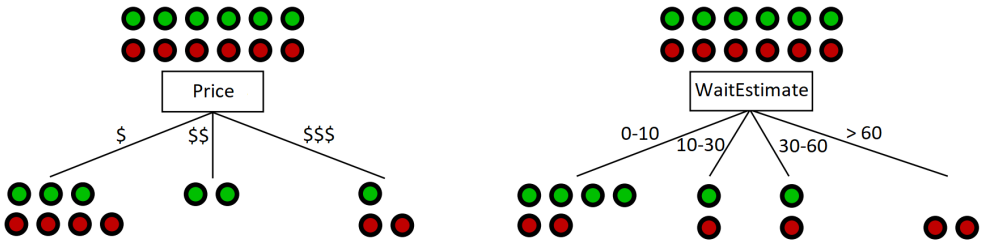
CSCI 5521: Machine Learning Fundamentals (Spring 2021)

Homework 3

(Due Tue, Mar. 30, 11:59 PM central)

1. **(15 points)** Consider a binary classification problem with two possible choices (Price and WaitEstimate) for the root of a decision tree for the restaurant dataset as shown in the figure. The green (top row) and red circles (bottom row) refer to the data points from the two classes where the green (top row) circles belong to class 0 and red (bottom row) belong to class 1.



   Calculate the information gain for the attributes 'Price' and 'WaitEstimate' showing all steps. Based on the calculated information gain, which attribute will you use as the root of the decision tree (Price or WaitEstimate)? Justify your answer by describing what the information gain numbers mean.

Programming assignment:

2. **(85 points)** In this problem, we will use the `Digits` dataset to classify '3' vs '8'. Your task is to build three classifiers: (a) bagging, (b) random forests, and (c) adaboost. You must implement each classifier, from scratch, using only the provided code and functions specified in this problem description and additional instructions below.

   (a) **(25 points)** Write code for class `MyBagging` in file `MyBagging.py` with two key functions: `MyBagging.fit(self, X, r)` and `MyBagging.predict(self, X)`.

   For the class, the `__init__(self, num_trees, max_depth)` function takes as input `num_trees` which is the number of tree classifiers in the ensemble (must be an integer ≥ 1), and `max_depth` which is the maximum depth of the trees.

   For `fit(self, X, r)`, the inputs (X, r) are the feature matrix and class labels respectively. The fit function will construct `num_trees` number of bootstrap datasets and learn a distinct decision tree on each bootstrap dataset. To learn a decision tree, you must use the class `DecisionTreeClassifier` from `scikit-learn`. You must set the decision tree hyperparameters as: `criterion='entropy'`, and `random_state = 0`.

For `predict(self, X)`, the input X is the feature matrix corresponding to the validation set and the output should a list of length equal to `num_trees`. Each element in the list should be an array of length equal to the number of data points in the validation set. The array should contain the predicted labels for each data point in the validation set using the ensemble of trees built up to that point. For example, the predictions in the array in the first element of the output list should be made only using the first decision tree constructed in the ensemble. The predictions in the array in the second element of the output list should be made using the first two decision trees constructed in the ensemble. The predictions in the array in the t-th element of the output list should be made using the first t decision trees constructed in the ensemble. The predictions in the array in the last element of the output list should be made using all decision trees in the ensemble. By constructing the output list in this way, we can use the hw3q2.py script to plot the mean error rate as we increase the number of trees in the ensemble.

(b) **(25 points)** Write code for class `MyRandomForest` in file `MyRandomForest.py` with two key functions: `MyRandomForest.fit(self,X,r)` and `MyRandomForest.predict(self,X)`.

For the class, the `__init__(self, num_features, num_trees, max_depth)` function takes as input `num_features` which is the number of random features to choose at each split in the decision tree (the variable m from the lecture notes, and must be $1 \leq$ `num_features` $\leq d$), `num_trees` which is the number of tree classifiers in the ensemble (must be an integer ≥ 1), and `max_depth` which is the maximum depth of the trees.

For `fit(self, X, r)`, the inputs (X, r) are the feature matrix and class labels respectively. The fit function will construct `num_trees` number of bootstrap datasets and learn a distinct decision tree on each bootstrap dataset. To learn a decision tree, you must use the class `DecisionTreeClassifier` from `scikit-learn` with the same parameters as specified in `MyBagging`.

For `predict(self, X)`, the input X is the feature matrix corresponding to the validation set and the output should a list of length equal to `num_trees` where each element is an array of predicted labels for each point in the validation set (constructed in the same way as in MyBagging).

(c) **(35 points)** Write code for class `MyAdaboost` in file `MyAdaboost.py` with two key functions: `MyAdaboost.fit(self, X, r)` and `MyAdaboost.predict(self, X)`.

For the class, the `__init__(self,num_iters,max_depth)` function takes as input `num_iters` which is the number of tree classifiers (i.e., number of iterations - the variable T from the lecture notes) in the ensemble and `max_depth` which is the maximum depth of the trees.

For `fit(self,X,r)`, the inputs are the feature matrix and the class labels respectively. The fit function will construct the weight distribution w_t in each iteration t and learn a distinct decision tree using the dataset X and w_t. It is recommended to learn a tree using w_t by sampling **with replacement** from X following distribution w_t to create a temporary dataset which then is given to the decision tree algorithm. (Consider using the random.choices() function.)

The fit function will also compute the weighted error rates ϵ_t and classifier weights α_t in

each iteration t. To learn a decision tree, you must use the class `DecisionTreeClassifier` from `scikit-learn` with the same parameters as specified in `MyBagging`.

For `predict(self, X)`, the input X is the feature matrix corresponding to the validation set and the output should a list of length equal to `num_iters` where each element is an array of predicted labels for each point in the validation set (constructed in the same way as in MyBagging).

We will compare the performance of these three classifiers (note the output from your code and reported in your solution pdf must be in this order):

(i) `MyBagging` with `num_trees = 100` using `DecisionTreeClassifier` with `max_depth = 1`,

(ii) `MyBagging` with `num_trees = 100` using `DecisionTreeClassifier` with `max_depth = None`,

(iii) `MyRandomForest` with `num_features=1, num_trees=100` using `MyDecisionTree` with `max_depth=1`,

(iv) `MyRandomForest` with `num_features = sqrt(d), num_trees = 100` using `DecisionTreeClassifier` with `max_depth = 1`,

(v) `MyRandomForest` with `num_features = 1, num_trees = 100` using `DecisionTreeClassifier` with `max_depth = None`,

(vi) `MyRandomForest` with `num_features = sqrt(d), num_trees = 100` using `DecisionTreeClassifier` with `max_depth = None`,

(vii) `MyAdaboost` with `num_trees = 100` using `DecisionTreeClassifier` with `max_depth = 1`,

(viii) `MyAdaboost` with `num_trees = 100` using `DecisionTreeClassifier` with `max_depth = None`,

applied to the `Digits` dataset. Using the code provided for `hw3q2.py` and `my_cross_val()` with 5-fold cross-validation, report the error rates in each fold as well as the mean and standard deviation of error rates across folds for each of the above 8 settings.

You will have to submit (a) **code** and (b) **summary of results**:

(a) **Code**: You will have to submit code for `MyBagging`, `MyRandomForest`, and `MyAdaboost`. We provide template code for the following classes and functions. Please read the comments in the code carefully especially in the predict() functions.

```
class MyBagging:
  def __init__(self, num_trees, max_depth):
  def fit(self, X, r):
  def predict(self, X):
class MyRandomForest:
  def __init__(self, num_features, num_trees, max_depth):
  def fit(self, X, r):
  def predict(self, X):
class MyAdaboost:
  def __init__(self, num_iters, max_depth):
  def fit(self, X, r):
  def predict(self, X):
```

Your classes **should not** inherit any base class in `sklearn` other than DecisionTreeClassifier. Again, the three functions you must implement in each class are are `__init__`, `fit`, and `predict`.

The wrapper script `hw3q2.py` is used to prepare the dataset, initialize class objects, make calls to `my_cross_val(method,`X`,`r`,`k`)` to generate the error rate results (here k is the number of folds), and print the results. **You must use this wrapper script or no credit will be given.**

(b) **Summary of results**: To generate the results, run the hw3q2.py script. For each method, report the validation set error rates for each of the $k = 5$ folds, the mean error rate over the k folds, and the standard deviation of the error rates over the k folds. Make a table to present the results for each method (8 tables in total). Each column of the table represents a fold, and add two columns at the end to show the overall mean error rate and standard deviation over the k folds (see below for an example). Also, include in your report the figure hw3q2.png which is generated after running hw3q2.py.

Error rates for MyBagging with `num_trees = 100, num_depth = 1`						
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	SD
#	#	#	#	#	#	#

Additional instructions: Code can only be written in Python 3.6+; no other programming languages will be accepted. One should be able to execute all programs from the Python command prompt or terminal. Please specify instructions on how to run your program in the README file.

Each function must take the inputs in the order specified in the problem and display the textual output via the terminal and plots/figures should be included in the report.

For each part, you can submit additional files/functions as needed. In your code, you can only use machine learning libraries such as those available from scikit-learn as specified in the problem description. You may use libraries for basic matrix computations and plotting such as numpy, pandas, and matplotlib. Put comments in your code so that one can follow the key parts and steps in your code.

Your code must be runnable on a CSE lab machine (e.g., csel-kh1260-01.cselabs.umn.edu). One option is to SSH into a machine. Learn about SSH at these links: `https://cseit.umn.edu/knowledge-help/learn-about-ssh`, `https://cseit.umn.edu/knowledge-help/choose-ssh-tool`, and `https://cseit.umn.edu/knowledge-help/remote-linux-applications-over-ssh`.

Instructions

Follow the rules strictly. If we cannot run your code, you will not get any credit.

- **Things to submit**

  1. hw3.pdf: A document which contains the solution to Problems 1 and 2 including the summary of results for Problem 2. This document must be in PDF format (no word, photo, etc. is accepted). If you submit a scanned copy of a hand-written document, make sure the copy is clearly readable, otherwise no credit may be given. **Do not include**

the PDF file within the ZIP file. Rather, the PDF document should be submitted as a separate document.

2. Python code for Problem 2 (must include the required `MyBagging.py`, `MyRandomForest.py`, and `MyAdaboost.py` files).
3. README.txt: README file that contains your name, student ID, email, instructions on how to run your code, any assumptions you are making, and any other necessary details.
4. Any other files, except the data, which are necessary for your code (such as package dependencies like a requirements.txt or yml file).

Homework Policy. (1) You are encouraged to collaborate with your classmates on homework problems, but each person must write up the final solutions individually. You need to list in the README.txt which problems were a collaborative effort and with whom. (2) Regarding online resources, you should **not**:

- Google around for solutions to homework problems,
- Ask for help on online,
- Look up things/post on sites like Quora, StackExchange, etc.