CSCC11 Introduction to Machine Learning, Winter 2021

Assignment 3, Due Thursday, March 18, 10am

This assignment makes use of material from week 5 to week 7 (specifically Chapter 8, Chapter 9.3 and Chapter 11). To begin the programming component, download `a3.tgz` from the course website and untar it. A directory A3 will be created; please don't change its structure, nor the headers of the python methods therein. Please read the assignment handout carefully. Submission instructions are at the end.

We prefer you ask questions on Piazza. Otherwise, for your first point of contact, please reach out to your TA, Anmol Garg, through email: anmol.garg@mail.utoronto.ca.

Written Component

1) Decision Tree

The table below has 12 items on a shopping list, each having 3 attributes. These attributes are to be used learn a decision tree for predicting the target variable, that is, whether a given item is fruit (True) or not (False). Given the training data in the table below, use information gain to determine the best split function at each step in constructing binary decision tree for this prediction task.

Instance	Attributes			Target Variable
	Size (S)	Color (C)	Texture (T)	Is Fruit (F)
1	Small	Red	Soft	True
2	Medium	Red	Soft	True
3	Large	Red	Soft	False
4	Small	Red	Hard	True
5	Medium	Red	Hard	True
6	Large	Red	Hard	False
7	Small	Yellow	Soft	True
8	Medium	Yellow	Soft	False
9	Large	Yellow	Soft	True
10	Small	Yellow	Hard	True
11	Medium	Yellow	Hard	False
12	Large	Yellow	Hard	False

Written Problems.

1. What is the entropy of the target variable (i.e. $H(F)$)?
2. What is the entropy of the target variable given the item size being small (i.e. $H(F|S = \text{Small})$)?
3. What is the entropy of the target variable given the item size being not small (i.e. $H(F|S = \text{Medium or Large})$)?
4. What is the information gain if we split based on fruit size being small?
5. At the root node, how many split tests do we need to consider and what is the best split?
6. Draw a picture of a possible final decision tree following the algorithm, and specify the split function in each internal node and the probability of being fruit is specified in each leaf node.

2) Random Forest

Recall the Bias-Variance Decomposition: Suppose we can sample training sets $D = \{(x_i, t_i)\}_{i=1}^N$ from a data generating distribution p_{sample} (i.e. we can create multiple training sets, each of which can be created by sampling N data pairs from p_{sample}). We assume our loss is defined as the squared error loss: $L(y,t) = \frac{1}{2}(y-t)^2$, where y is the prediction and t is the true target. Given a query point $\hat{x} \sim p_{\text{sample}}$, we are interested in making a prediction y that minimizes the expected squared error loss with respect to $\hat{x}$, $\mathbb{E}\left[(y-t)^2|\hat{x}\right]$. We showed that the expected loss can be decomposed into bias $\mathbb{E}\left[y|\hat{x}\right] - y_\star$, variance $\text{Var}\left[y|\hat{x}\right]$, and Bayes error $\text{Var}\left[t|\hat{x}\right]$:

$$\mathbb{E}_{p_{\text{sample}}(t|\hat{x}),p_{\text{sample}}(D)}\left[(y-t)^2|\hat{x}\right] = \left(\mathbb{E}_{p_{\text{sample}}(D)}\left[y|\hat{x}\right] - y_\star\right)^2 + \text{Var}_{p_{\text{sample}}(D)}\left[y|\hat{x}\right] + \text{Var}_{p_{\text{sample}}(t|\hat{x})}\left[t|\hat{x}\right], \tag{1}$$

where $y_\star = \mathbb{E}_{p_{\text{sample}}(t|\hat{x})}\left[t|\hat{x}\right]$ is the best possible prediction given query point $\hat{x}$.

Random forest is based on a technique called *Bagging*. Bagging works by sampling M datasets $\{D_i\}_{i=1}^M$ independently from p_{sample}, training a predictor y_i using the i'th dataset D_i, then making a prediction given query point $\hat{x}$ by averaging across the predictions from each predictor y_i:

$$y = \frac{1}{M}\sum_{i=1}^M y_i. \tag{2}$$

One can show that, with the independent dataset samples, the bias and Bayes error remain unchanged while the variance is reduced as we increase M (For conciseness, we drop the conditional on the query point $\hat{x}$ from here on):

$$\mathbb{E}\left[y\right] = \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^M y_i\right] = \mathbb{E}\left[y_i\right], \tag{3}$$

$$\text{Var}\left[y\right] = \text{Var}\left[\frac{1}{M}\sum_{i=1}^M y_i\right] = \frac{1}{M}\text{Var}\left[y_i\right]. \tag{4}$$

Written Problems.

1. Suppose the dataset samples are not independent. Instead, each pair of sampled datasets are correlated with a positive pairwise correlation ρ (also known as Pearson Correlation), and let $\text{Var}\left[y_i\right] = \sigma^2$ for all i. Show that,

$$\text{Var}\left[\frac{1}{M}\sum_{i=1}^M y_i\right] = \frac{1-\rho}{M}\sigma^2 + \rho\sigma^2. \tag{5}$$

   Hint: The pairwise correlation $\rho_{X,Y}$ between two random variables (X, Y) is

$$\rho_{X,Y} = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y}, \tag{6}$$

   where Cov is the covariance, σ_X is the standard deviation of X, and σ_Y is the standard deviation of Y.

2. In few sentences, explain why the independence assumption is violated in a random forest. **Hints:** In addition to the randomness from sampling the dataset, there is also randomness from drawing a pair of decision trees grown to the randomly sampled dataset. What does decision tree evaluate to determine the split?

3. What heuristic does random forest employ in order to decorrelate the trees?

Programming Component

Your task is to complete the implementation of a random decision forest classifier. See lecture notes on decision forests (Chapter 9.3), information theory (Chapter 8), and cross-validation (Chapter 11). You are given starter-code to build and test random forests, including scripts to test on both synthetic and real-world datasets. We suggest you initially debug your code on synthetic data, and then proceed to training/testing on the real-world datasets. To this end you will need to tune the hyper-parameters using a held-out validation set, and then modify the script to train on the entire training set (and perform testing).

To begin, download and extract the starter file `a3.tar` and look at the code. Among the various files, which you should read, you will find some methods/functions (in the Python files) that you have to complete. They are

1. `decision_tree.py` implements the decision tree model.
    - `_entropy` computes the entropy of the distribution over class labels given a dataset with labels y. This function is used for building a single decision tree in `build_tree` and `_find_split`.
    - `_find_split` chooses a random feature dimension of the input vector, sorts data points according to the value of the selected feature, and evaluates the information gain for hypothetical split thresholds. This file is used for building a single decision tree in `build_tree`. It returns the split threshold with maximum information gain. The file includes code to help structure your implementation.
2. `random_forest.py` implements the random forest model.
    - `build_forest` builds a random forest, a collection of decision trees learned on random subsets of data and features. You need to complete the part that selects random subsets of data and features.
3. `test_cases.py` enables debugging using simple synthetic examples. Feel free to write additional tests that check individual functions above. We will automatically mark your code in a similar way using other training/testing datasets with varying complexity.

Datasets: The starter code also includes scripts to train on two real-world datasets (in the directory `datasets`). You will note that the range of feature values are very different in them, but decision trees are not limited by such differences. See links below if you want more information about the datasets.

- **Occupancy Detection.** This is a binary classification task with real-valued sensory inputs. The task is to detect whether an office is occupied based on light, temperature, humidity and CO2 measurements. There are two test sets, one for validation and one for testing (see `run_occupancy.py`). On this dataset, you should be able to reach test accuracy close to 95%.

  Once you have completed and tested your code, *answer the following questions under Occupancy dataset* in `Questions.txt`:

  1. Is a single DT prone to over-fitting with the occupancy dataset?
  2. How does classification accuracy behave as a function of the number of trees?
  3. Do you think the random forest was helpful? Explain.

- **Lending Club Loan Data.** The task is to predict whether or not a new customer is likely to pay back the loan. There are 21 features in each data point, each containing the customer's financial background, loan amount, location, etc. Some features are categorical features expressed in numerical values. You are given approximately 70000 training pairs (feature vector, loan repaid), 18000 validation pairs, and 20000 test points (See `run_lending_club.py`). Even with this many data points, you will see that decision tree can still easily overfit with this dataset. In real life, we do not know how well a model can do. Hence, you need to find out the best possible test accuracy on this dataset.

  Once you have completed and tested your code, *answer the following questions under Lending Club dataset* in `Questions.txt`:

1. What is the highest accuracy you are able to achieve? Do you think you can reach an even higher accuracy with just random forests?
2. Can you think of a better way to choose split functions (as opposed to the method in `_find_split`).?
3. How does this challenge differ from that with the occupancy dataset?

Decision forests have several hyper-parameters, including the number of trees, the tree depth, the minimum number of data points in a leaf, and the percentage of the data used to build each tree. You will have to put these hyper-parameters in `hyperparameters.py` and choose suitable values for each dataset. This requires thought and experimentation. After tuning hyper-parameters on a validation set, set the `final_hyperparameters` flag to `True` in `run_occupancy.py` and `run_lending_club.py` to work on the test set. Your grades will be based on how accurate your implementations are and the test accuracy of the chosen hyperparameters.

Submission

You will need to submit two files, one for each component. The specifications of the file format and naming convention are specified below.

1. Written Component

   You may hand write or type up (e.g. Word, LateX, etc.) the solutions. We strongly recommend typing it up for legibility. Compile the document as a PDF named **A3sol_UTORID.pdf** (e.g. **A3sol_student1.pdf**). Then you should submit the pdf file onto Quercus.

2. Programming Component

   Compress the `A3` directory into a tar file named **A3sol_UTORID.tgz** (e.g., **A3sol_student1.tgz**).

   Please don't modify the method/function headers or the structure of the A3 directory tree since the automarker will fail to run. Failing to follow the instructions will result in a 25% penalty.

   To tar and zip the contents, use the command: `tar -cvzf name_of_tar.tgz A3`. Make sure to decompress the tar file to ensure the entire A3 directory is there. Then you should submit the tar file onto Quercus.