Automation of Biological Research: 02-450/02-750

Carnegie Mellon University

Homework 4

Version: 1.0; updated 03/14/2019

Question 1: "RDP" (50 points)

Provided Material: You are provided the file *rdp.py*. The functions **call_oracle()**, **test_data()**, **plot_samples()**, and **refine()** are provided and will work as-is. You will implement the functions **create_dyadic_tree()** and **prune()**.

The **main()** function acts as a wrapper that performs the data sampling for you, and appropriately calls each of the functions you will have to implement. Plots will automatically be generated, and relevant statistics you are asked to provide will be displayed. In part C, you will complete the **main()** function as described below.

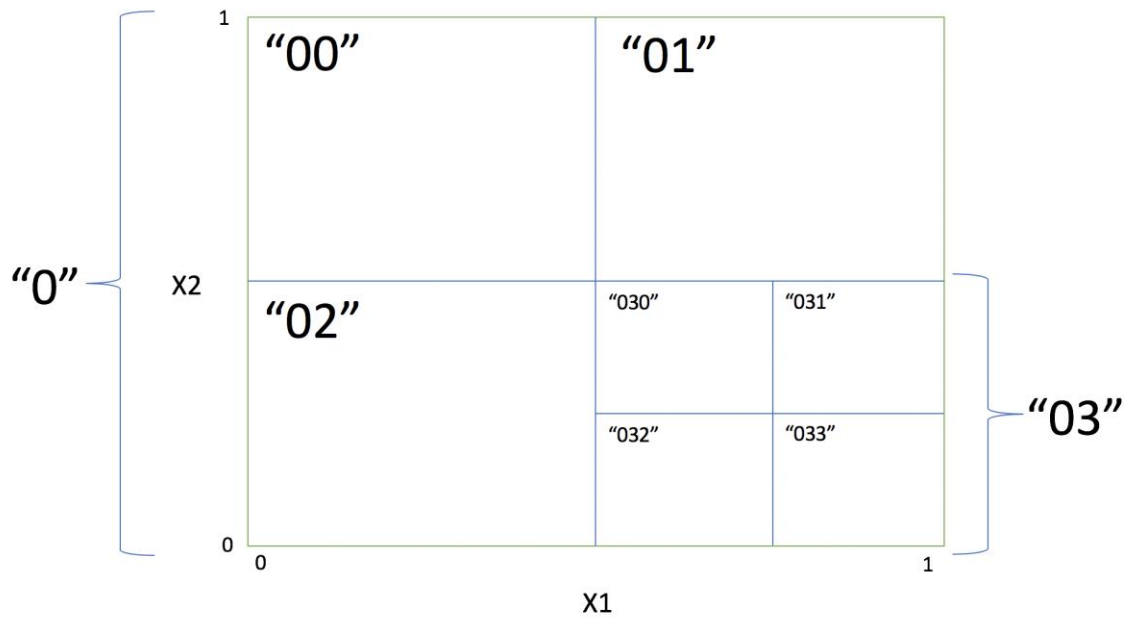
The data you will be working with is two-dimensional, with features denoted x1 and x2. Each of these take on real values on the unit interval [0,1]. The label y associated with each sample [x1,x2] is some real number.

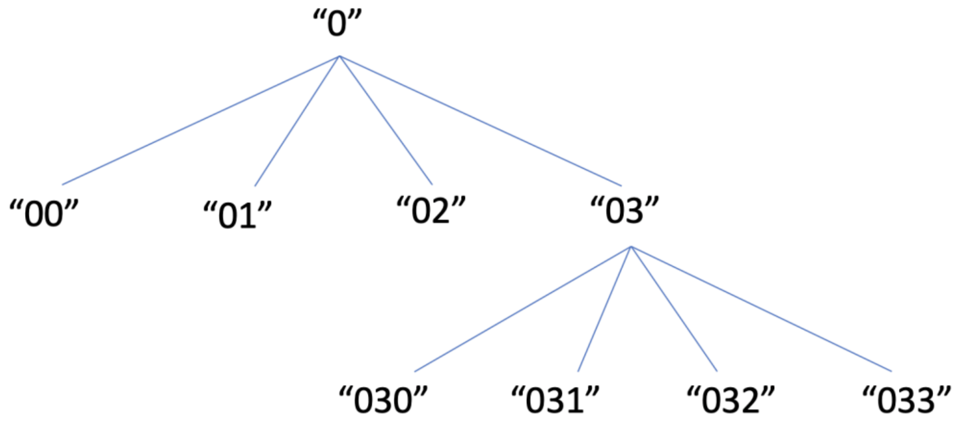
Tasks:

A. 25 points. Implement **create_dyadic_tree()**. The first step of the RDP algorithm is to build a full tree over the data using recursive dyadic partitions (RDP). In our 2D example, this amounts to recursively splitting the domain of our data ($[0,1]^2$) into quadrants. The two figures below demonstrate visually how this looks on a 2D grid and the corresponding tree representation, respectively. Notice the naming convention used for the nodes. The root of the tree is named by the string "0". The four children of "0" are named by appending a "0", "1", "2", or "3" to this name. By applying this pattern throughout the tree, we see that we will obtain uniquely identifiable nodes, where the length of the name corresponds to the depth of the node, and parent-child relationships can be easily inferred. We will adopt this node naming convention in our implementation.

Create_dyadic_tree() takes as input a tree (described below), the current node name, and the maximum desired depth of the tree, and should output a tree with the following structure. The tree is a dictionary whose keys are the node names described above. The values of each node is another dictionary that contains the following key-value pairs (each key is the string denoted within quotes)- “X_queried”: list of queried data samples contained in current node, “y_queried”: list of labels for queried data samples, “x1_range”: tuple denoting (min x1 value in domain, max x1 value in domain) that is represented by the node, “x2_range”: tuple denoting (min x2 value in domain, max x2 value in domain), “leaf_status”: Boolean denoting whether the node is a leaf. An example entry for nodes “0”, “02”, and “033” is shown below.

If there is any uncertainty, please post questions to Piazza and/or come to OH on Mondays 1-2pm.





```
Tree = { “0”: {“X_queried”: queried samples within “0”,
               “y_queried”: queried labels within “0”,
                ”x1_range”: (0,1), “x2_range”: (0,1),
               “leaf_status”:False} ,

        “02”: {“X_queried”: queried samples within “02”,
               “y_queried”:queried labels within “02”,
                ”x1_range”: (0,0.5), “x2_range”: (0,0.5),
               “leaf_status”:False},

        “033”: {“X_queried”: queried samples within “033”,
                “y_queried”:queried labels within “033”,
                ”x1_range”: (0.75,1), “x2_range”: (0,0.25),
                “leaf_status”:True }
}
```

B. 15 points. Implement **prune()**. This function should take as input the full tree you generated in the previous section, and output a pruned version. Pruning is the process whereby we perform a bottom-up pass through the tree and eliminate leaf nodes that are no more informative than their parent nodes. To do this, compute the mean variance of samples associated to sibling leaf nodes, and compare this to the variance of samples associated with their parent node. If this difference is less than 2, remove those leaf nodes from the tree (be sure to set the parent node’s “leaf_status” to True if you prune away the children). The main function will generate a plot of the samples associated with the deepest leaves of your pruned tree, which you should include in this section of your write up. How does this compare to the figure from part A? Additionally, report the total number of nodes in your pruned tree.

C. 10 points. Use the refined samples to train a DecisionTreeRegressor and report the R^2 value obtained on the test data. Pass your pruned tree as an argument to the provided **refine()** function, which will return another tree obtained via the refine procedure outlined in the paper as well as lecture 13 slides. From this tree, you must first identify the samples and their labels associated with the deepest leaves, and use those samples to train a scikit-learn DecisionTreeRegressor using a max depth of 5. Use the score() method to obtain an R^2 value on the provided test data, and report this in your write-up. The main() function will generate a plot of the samples contained in the deepest leaves, include this in your write-up as well, and provide a brief comparison to the figures from the previous two parts. Additionally, report the number of leaves at the maximum depth.

What to hand in: a zip file containing your completed rdp.py file and a pdf with your plots and responses to the above parts.

Question 2 Proactive Learning (50 points)

In this question, you will be implementing proactive learning 'scenario 3' (as defined in the proactive learning paper) where you have two oracles, one is an oracle with uniform cost, the other is an oracle with variable cost. A code template **proactive_learning.py** is provided with some starter code to help you with the implementation and the use of modAL. You will be also comparing the results when varying the cost ratio between uniform cost oracle and variable cost oracle. Additional instructions can be found in proactive_learning.py. Please read the comments in the code template! Please do not change the function signatures, and make sure they return the values asked for when called. The estimator to be used for this part is the support vector machine (SVM) classifier, and you can use scikit-learn package for convenience.

The data used in this question are the imaging data set "**Data.csv**" which you used for homework2 Q1. The data comprises 500 samples each with 26 features. Each sample is labeled with one of ten possible subcellular locations. You are also provided a "**cost.npy**" file that contains the cost of each sample of the variable cost oracle. The cost of each sample is pre-computed according to the following scheme:

$$C_{variable}(x) = 1 - \frac{max_{y \in Y} P(y|x) - 1/|Y|}{1 - 1/|Y|}$$

where $Y$is the set of labels, and $P(y|x)$is the conditional probability of the most likely label y. Use "cost.npy" as the price list for the variable cost oracle, which can be easily loaded into your workspace using np.load(). The source code for computing the price list is included in the code template just for your reference.

Part 2.1 (25 points): Implement proactive query strategy. Complete the function ProactiveQuery() in proactive_learning.py. This function will return the index of the unlabeled instance in the unlabeled pool, as well as the oracle that being queried (1 for uniform cost oracle, 2 for variable cost oracle).

The utility for each unlabeled instance in the unlabeled pool can be computed as

$$U(x, k) = V(x) - C_k(x)$$

where $k = 1$ for uniform cost oracle, $k = 2$ for variable cost oracle. For this question, use uncertainty quantified by entropy as $V(x)$, which is a measure over a probability distribution. Higher entropy means more uncertainty about the labels. Mathematically, the entropy is defined as

$$-\sum_{y \in Y} P\,(y|x) log_2 P(y|x)$$

To make $V(x)$ be in the same range as cost, normalize $V(x)$ by dividing with the maximum $V(x)$value.

The label probabilities can be obtained by first fitting a SVM classifier (use gamma='scale', decision_function_shape='ovo', probability=True) based on the labeled data, and then calling .predict_proba() method which will give you the predicted probabilities for each possible label.

This function should support 3 different modes, i.e. "proactive" for proactive learning, "uniform" that always query the uniform cost oracle, and "random" that randomly query an instance from a randomly selected oracle. The "proactive" mode is the proactive learning query strategy as you learned in the lecture. For "uniform" mode, you always query the uniform cost oracle, and select the instance that has the maximum utility with uniform cost oracle. For "random" mode, you randomly query an instance in the pool of unlabeled data, with 50% chance from the variable cost oracle and 50% chance from the uniform cost oracle.

Part 2.2 (15 points): Complete the function ProactiveLearning() in proactive_learning.py, which creates an ActiveLearner object and performs active learning with query strategy defined by ProactiveQuery. For this part, you would need to properly maintained the labeled pool and unlabeled pool of data, as well as the 'price list' so that the index returned from ProactiveQuery matches with the corresponding entry in the provided price list. This function will return a list of accuracies, a list of cumulative costs, and a list of oracles queried at each iteration.

The labeled pool of data contains 10 initial, free instances, one for each class. These 10 initial instances are loaded from the provided "**initial_labeled_sample.npy**" file. Use ProactiveQuery to decide which data instance to query and which oracle to query. Calculate the corresponding cumulative cost to append to the list of cumulative costs and append the corresponding oracle to the list of oracles. Also compute prediction accuracy on a held-out test data set using the model that is fit using all the labeled data in the labeled pool. Please only fit the SVM classifier when you are about to compute prediction accuracy.

Part 2.3 (10 points): After completing ProactiveQuery() and ProactiveLearner() function, run proactive learning with three different modes and three different cost_ratios (5, 1.1, 0.5), which is defined as mean of cost of variable cost oracle / cost of uniform cost oracle, and generate the plots of the classification error on test data set vs cumulative cost of the three modes for each cost_ratio using the provided code for plotting. Also, report the number of times when querying from uniform cost oracle and the number of times when querying from variable cost oracle for three different modes for each cost_ratio. Compare and comment briefly on the different behaviors observed with three different modes and with various cost_ratios.