Duong Hoang – CS460G Spring 2022

CS460G MACHINE LEARNING PROJECT 1- DECISION TREE WRITEUP

Part 1: Classify Synthetic Data

Implementation Choices:

- Continuous feature data are discretized by dividing them into k equal distance intervals. Each data then be represented by the number order of the intervals (0, 1, 2, …, k) which are numbered ascendingly. The interval's width is calculated by the difference of the biggest and smallest values in each feature divided by the number of intervals/bins, k (or *NUM_BINS* in the code).
- Since feature data are discretized by interval number and all intervals are numbered as non-negative integers, I set -1 as *INVALID_VALUE* for the discretized value. This value will equivalent to the whole span of coordinate values in cases of decision tree when the tree does not split on that feature.
- As all feature data are discretized as 0 to k, all possible vi values in ID3 algorithm are also 0 to k for all features, which is equivalent to *range(NUM_BINS)* in the code.
- The class *Node* constructing the decision tree have two attributes: *value* and *children* (list). The value attribute of the node will be assigned to the feature name when the node is an interior node of the tree and will be assigned to the feature value when the node represents as a branch in the tree. Due to this design choice, the way we traverse the tree from the top will always assume the nodes on even levels (start counting root as level 0) are nodes of the tree and those on odd level are branches of the tree. The depth of the tree is ceiling half the number of node levels.
- The depth of the tree is not tracked by the Node, but by the ID3 algorithm. As the algorithm is called recursively, each recursive call of the function means adding another depth level to the tree. So, we will stop calling the function recursively when the depth parameter reach 2 as we will be adding leaf nodes, which will add another level of depth to the tree (depth then be 3), before returning the tree.
- The class *DecisionTree* manages the decision tree. It discretizes the data (and save them in the *self.discretized_data* attribute), calculate entropy and information gain to build the decision tree implementing ID3 algorithm (root of the tree is saved in the *self.root* attribute), and visualize the training data with the tree classification color coded in the background.
- $Entropy = -\sum_i P_i log_2(P_i)$ where $P_i = \frac{\#samples\ of\ class\ i}{\#samples}$. Function getEntropy() return $-P_i log_2(P_i)$ for each unique feature value i in feature. So, to calculate Entropy(T, X), which is the entropy of splitting on feature X from parent node T, we will calculate the sum of the results from calling getEntropy() of each unique feature value of X.
- $Information\ Gain(T, X) = Entropy(T) - Entropy(T, X)$ where T is the parent node before the split and X is the split node.

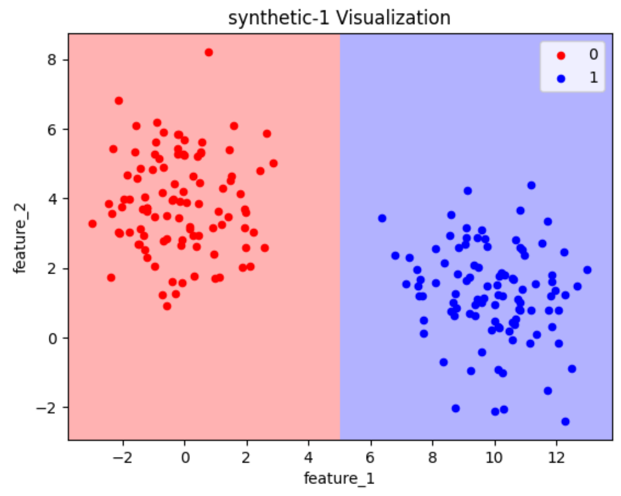
Training Set Error:

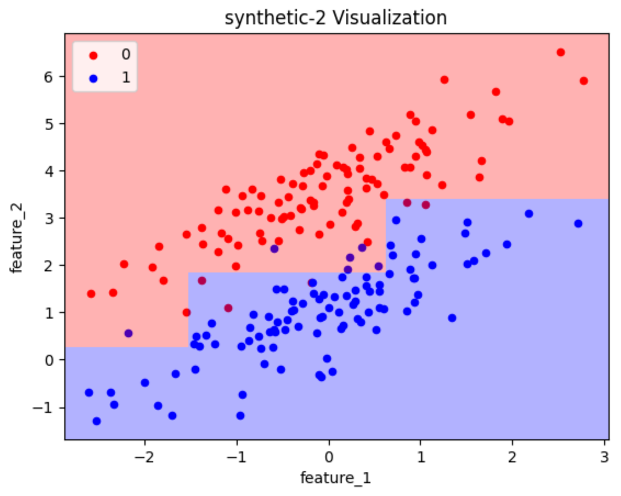
I ran the code file ***syntheticDecisionTree.py*** for all 4 synthetic data files. For each synthetic dataset**, I changed the initial variables *DATA_FILE* and *NUM_BINS* at the beginning of the code file before running**. After changing the parameter values, I got these following results:

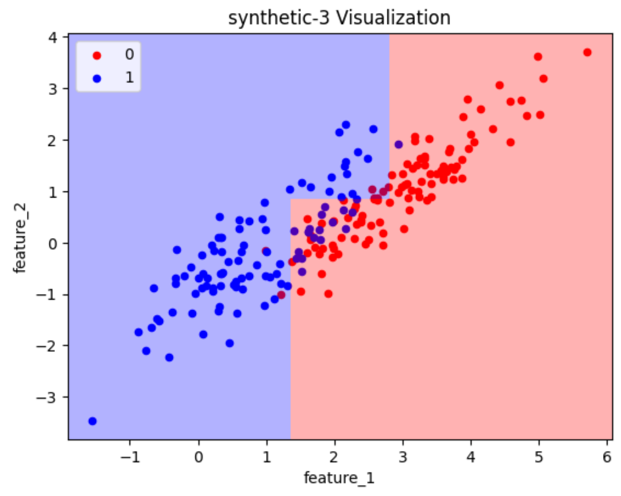
Dataset {DATA_FILE}	Number of Discretize Equal Distance Intervals {NUM_BINS}	Training Set Error = #Incorrect / #Data	Accuracy Rate	Side-by-side Prediction vs Key Comparison File
synthetic-1	4	0/200 = 0	1.0	classified_synthetic-1.csv
synthetic-2	5	11/200 = 0.055	0.945	classified_synthetic-2.csv
synthetic-3	5	26/200 = 0.13	0.87	classified_synthetic-3.csv
synthetic-4	5	8/200 = 0.04	0.96	classified_synthetic-4.csv

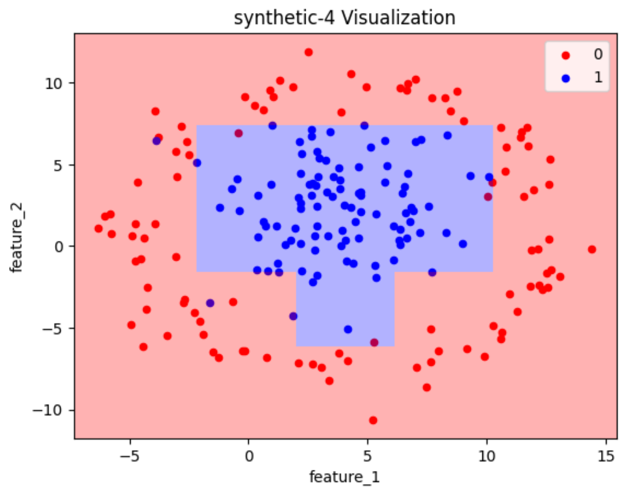
Part 2: Visualize Your Classifiers

Synthetic data visualizations as scatter plots using *matplotlib.pyplot*:









- The x-axis represents feature_1, which is the first feature column, and the y-axis represents feature_2, which is the second feature column in each synthetic dataset.
- All data points that have class label 0 are colored in red and those that have class label 1 are in blue.
- The decision tree approximations are also colored with the same rule: 0 – red, 1 – blue. The background areas are colored by setting colors for corresponding grids created by the boundaries generated from discretization. Each of the two ends of feature_1 interval and two ends of feature_2 interval limit a box area, and each of these areas is classified by the tree. Therefore, to color each of

these boxes, I fill between each set of the four boundary lines with corresponding classification color.

Part 3: Classify Pokémon!

Implementation Choices:

- The first eight features are continuous data while the rest are binary data (0s and 1s), so I discretized the first eight features (which are features that do not contain 'Type' in their names) using the same method as I did with synthetic datasets. I discretized them into five equal distance intervals/bins.
- Due to this choice of discretization, all possible vi values for the first eight features in ID3 algorithm are also 0 to k = 5, while the rest of features (which their names contain 'Type') possible value ranges are 0 to 1 (range of 2 in the code). Therefore, I checked the attribute name that best classified the examples and assigned the vi range accordingly.
- The implementation for this dataset is similar to the synthetic dataset, except for the part that I access the data frame columns by names instead of indexes.

Training Set Error:

I ran the code file *pokemonDecisionTree.py* and I got the following result:

Dataset	Number of Intervals	Training Set Error = #Incorrect / #Data	Side-by-side Prediction vs Key Comparison File
FEATURE_FILE = 'pokemonStats.csv', LABEL_FILE = 'pokemonLegendary.csv'	NUM_BINS = 5	67/1470 = 0.04557823129251701 -> Accuracy Rate = 0.954421768707483	classified_pokemon.csv