Part 1: Classify Synthetic Data

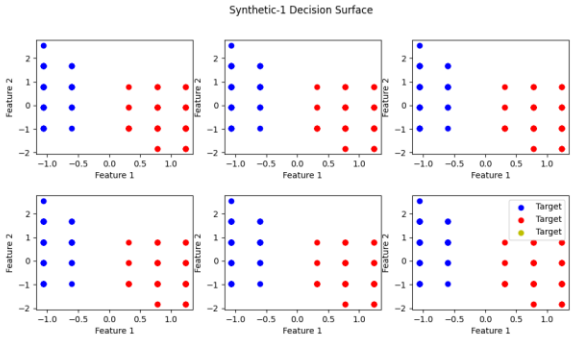
The Python script I created for this assignment largely works with lists of lists, and assigns those lists to nodes within the tree. For the discretization of the data I am using the popular Numpy framework, when I first started this project, I was manually assigning the bins, which resulted in poor performance and lack of flexibility on the number of bins and the datasets given to the program. However, using Numpy eliminated these problems by dynamically creating the exact number of bins that can be changed on the fly, allowing me to experiment with different bin values. Once the "targets" are placed next to the data, my script begins building the tree, each node has its own scripting to determine if it is a leaf by calculating its dataset's entropy. If the node decides it is a leaf, it assigns a prediction based on which target attribute is more prevalent within the data. Since this is all done within the leaf, all my ID3 function really needs to do is build out the rest of the tree, by finding the best attribute to split on and creating a new node for each bin and assigning it the split's dataset. The function also uses a common variable for the number of bins so it can build out the tree dynamically. There is also a check to make sure there is actually information gain in the given dataset, and if not the function simply makes a leaf. If it determines the new node is not a leaf it recursively calls itself to start the process over until the tree is complete. See testing results below.

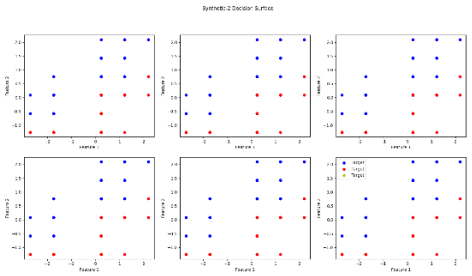
Bins = 50

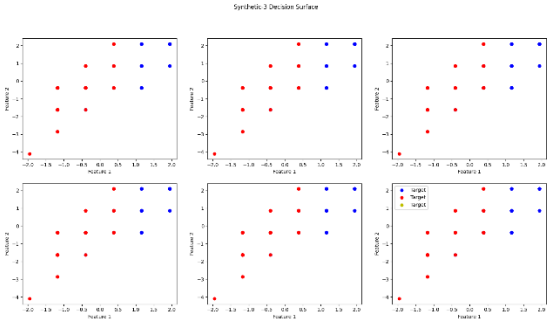
Synthetic-1	100%
Synthetic-2	65%
Synthetic-3	30%
Synthetic-4	64.5%

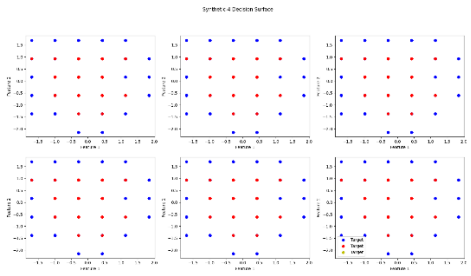
I am still not sure why synthetic-3's results are so much lower, especially considering synthetic-4's results. I ran out of time on the assignment before I could do more testing to find the source of the problem.

Part 2: Visualize your Classifiers









Part 3: Pokémon Classification

The code for this section is much the same from the first part as I made sure to make it adaptable. The only adjustments I needed to make was the data handling, reading in the legendary target separately from the stats and appending it to a new column with the stats, as my code was designed to only work with one list. The Numpy discretization also seems to ignore data that is already categorical so I did not need to change much there. However, experimenting with the bins brought some rather strange results:

Any Bins < 8	50%
Any Bins >= 8	100%

As you can see, any bin below eight is no better than random chance, while eight and up has a perfect accuracy. This may be resulting from faulty code, but as mentioned earlier it is much the same code as the first part. As of this writing I have not found any bugs that may cause the issue.