Predicting Cancer

Using the provided dataset, implement 4 different classification algorithms to determine if a candidate has breast cancer. For this assignment use Python 3 and the Numpy package (optional).

The assignment file you will submit is assignment3.py. Please complete the methods within the Class definitions provided, you can add helper methods and classes as well. All the code will be run through the run_assignment.py file. You can modify this file for your own testing, but you can only upload the assignment3.py file at the end so make sure all your code is in that file. You can expect that any variable in the run file could be changed during evaluation except for the data type which will be the same. The shape of the data could change though, there may be a different number of points or a different number of features. If you don't hardcode values, you should be fine.

For evaluation we will let the run file run for up to a minute, but you should really aim for under 10 seconds total (assuming an average modern laptop). Using numpy vectors is not required but there are a lot of vector operations that you can run over data in numpy and it will keep your code clean while massively improving performance.

Your code should not print anything to the console when you submit your assignment.

Data:

The attached csv file contains all the data. The run file handles importing it and converting it to numpy arrays. A description of the dataset is in the run file.

K-Nearest Neighbor:

Use Euclidean distance between the features. Choose a k value and use majority voting to determine the class. The k value is provided to the knn class. Please implement train (which is just memorizing the data) and predict methods (that run the knn algorithm). The distance function is provided for you and you can assume all data is continuous. In case of a tie, you can pick either class.

Decision Tree:

Use the ID3 algorithm as defined in the slides. Since all the data is continuous they have to be broken down into bins. This has been done for you in the preprocess method. You can preprocess the data in the train and predict methods (that you implement) before evaluating. The preprocess functions breaks the data into bins based on equal width. If there are ties in selecting the majority class or the maximum information gain, pick one.

Perceptron:

For the perceptron, multiply the inputs by a weight matrix and then pass the output through a single heaviside (step function) function to get the output. Don't forget the bias. Train the perceptron using the perceptron learning algorithm. The weight matrix and bias are initialized in the run file to facilitate grading. You must update the weights, but don't change the shape or type of the numpy array. The number of steps are defined in the run file, for each step update the model on a single datapoint.

Multi-Layer perceptron (MLP):

For the MLP, we have implemented most of it for you. Please look through the code and try to understand it. What happens if you comment out shuffle from the training process? The MLP class calls a FCLayer (Fully connected layer) and a Sigmoid layer. You need to implement the forward and backward function. The forward is for prediction and the backward for gradients. The backward function takes the previous gradient as input, updates the layers weights (for FCLayer), and passes back the gradients for the next layer. Please follow the example from the slides. MLP will be trained with the learning rate and loss function (Mean Squared Error) provided. You must update the weights, but don't change the shape or type of the numpy array. Also don't modify the MLP class in your final submission. The number of steps are defined in the run file, for each step update the model on a single datapoint.