BLG 454E Learning From Data Homework 2

Due: Friday, January 5, 2023

Lecturer: Yusuf Yaslan CRN: 12183

Fatih Baskın
150210710

Contents

Desicion Trees

In this problem, we have implemented a decision tree algorithm to classify the iris dataset. Decision trees work by comparing some features depending on the threshold value, going left or right until we hit a class label.

My decision tree recursively builds itself, with nodes being responsible for their training and splitting data logic. I have used the Gini index to calculate the impurity, using $\Sigma_c P_{class} \times (1 - P_{class})$. Each node uses an unused feature to classify the data, and while doing so node selects the feature with the least Gini impurity.

Before starting to select the feature with the least impurity, a node first decides a threshold value for each unused (usable for classification) feature, by using binary search. It selects the middle point, between the maximum and minimum value observed in the training data coming from the parent, then it calculates the left-midpoint, middle, and right-midpoint Gini indexes, left-midpoint being the midpoint between the left and the middle, and the right-midpoint is the midpoint between the middle and the right, while left and right being the minimum and maximum of that feature respectively. With binary search logic, it tries to move towards the least gini impurity.

After selecting a feature threshold for each node, it calculates the gini impurities for that feature with a given threshold and selects the feature with the least gini impurity. Then it splits the data into left and right baskets, being the left and right child, using the treshold value.

If a node is pure, all classes in the coming training data are the same, then it is a class node, or if a node exhausted its all usable features, then it is a class node too, the class being the majority class.

Results

With the logic explained above, with one tree, my accuracies were 100% for the first class, 98% for the second class, and 90% for the third class. The figure of predictions is shown below:

Possible improvements

Since decision trees are quite unstable, it is possible to use them to our advantage, with cross-validation, creating multiple trees and using the best-performing trees among them, creating a decision forest. The majority of the guesses would be the class choice.

Autoencoder

Auto encoder is a neural network, trained for replicating the input layer. If the width of the middle section is different than the number of features, then if we cut it from this chokepoint, we have an encoder and decoder neural network.

These autoencoder neural networks are generally used to generate data from damaged & low-quality data. Most of the time, they have a sigmoid (activation function) hidden layer and a linear output layer. This is the implementation that I used in my project.

The problem of neural networks is that training them is difficult, due to complex derivatives required to train them using gradient descent. I have difficulty linearizing the derivation process.

Autoencoder continued on next page...

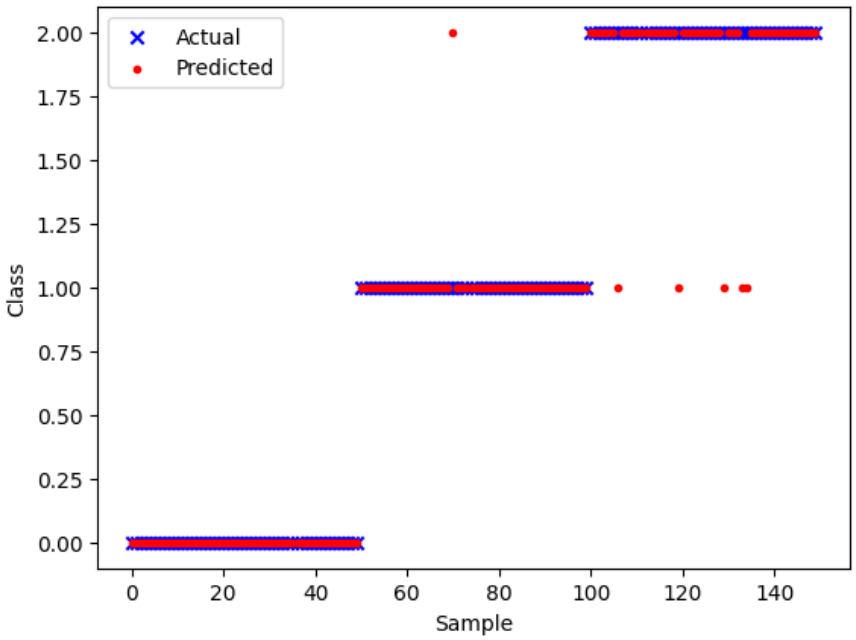

Figure 1: Desicion Tree findings

Problems Observed

The biggest problem is the runtime since I couldn't linearize the derivatives, I calculated them iteratively. It takes a very long time, instead of NumPy's optimized linearization.

Calculation of derivatives using a sigmoid hidden layer was quite hard.

Also, my gradient descent either converges at a point where every output sample is the same or diverges quite rapidly. To mitigate this, the hidden layer size must be big (> 16), the learning rate should be small (for ex: 10^{-4}) and the number of iterations must be small (≤ 100).

Solutions to these problems might consist of using two linear (activation function) layers, or one hyperbolic and one linear layer.

Findings

With hidden field size 128, epoch count = 100, and learning rate = 10^{-4} my auto-encoder results were close to the original data, they are shown in the figures below.

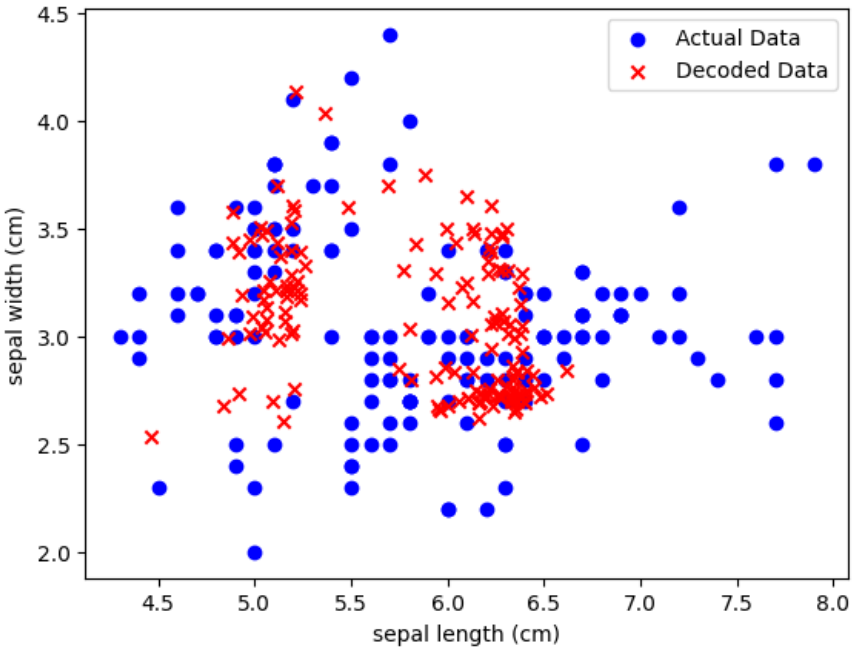

Figure 2: Decoded data for sepal width & length

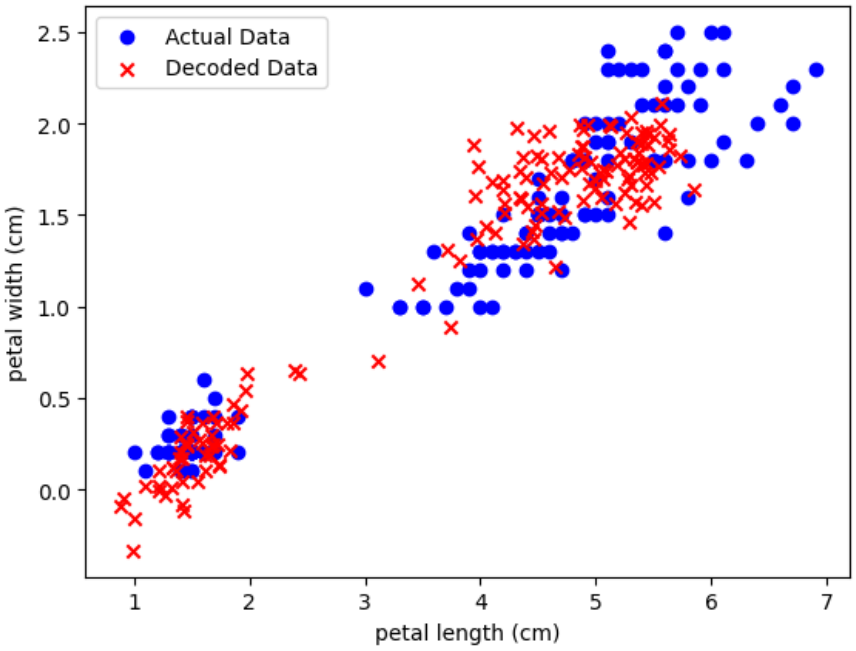

Figure 3: Decoded data for petal width & length