Assignment 3: Supervised Learning

CS486/686 – Spring 2023

Out: June 26, 2023
Due: July 7 (11:59 pm), 2023.

Submit an electronic copy of your assignment via LEARN. Late submissions incur a 2% penalty for every rounded up hour past the deadline. For example, an assignment submitted 5 hours and 15 min late will receive a penalty of ceiling(5.25) * 2% = 12%.

1. **[60 pts]** Implement the decision tree learning algorithm by filling in the functions in the skeleton code provided on the course website. Since the skeleton code uses Pandas data frames to represent and manipulate the data, it is highly recommended to go through the following Pandas tutorial: `https://pandas.pydata.org/docs/user_guide/10min.html`. Then download the real estate dataset from the course website and train a decision tree to classify the sale price of houses as high or low based on categorical real estate attributes. To control the size of the resulting decision tree, optimize a threshold hyperparameter by 10-fold cross validation. This threshold hyperparameter indicates the minimum amount of data needed to consider splitting a leaf based on an attribute. When a leaf has less data than the threshold, the leaf should not be split even if the data is not homogeneous. Evaluate the following list of threshold values: [10, 20, 40, 80, 160].

   What to hand in:

   - Your code.
   - A printout of the average validation accuracy for each threshold.
   - The best threshold, a printout of the decision tree for this best threshold and the test accuracy of this decision tree.
   - Brief discussion of the results.

2. **[40 pts]** Experiment with a few variants of the neural network starter code provided on the course webpage. Since the code uses PyTorch, it is highly recommended to go through the following PyTorch tutorial: `https://pytorch.org/tutorials/beginner/basics/intro.html`. Then run the starter code to train a two-layer fully connected neural network on the same real estate dataset as in Question 1. The code should produce two graphs that display the training accuracy and testing accuracy. Modify the code to experiment with 3 variants:

   - Implement and test fully connected networks with 1, 2, 3, and 4 layers. The starter code already provides you with an implementation of 2 layers. Each hidden layer should have 36 nodes.
   - Modify the code to replace the ReLU activation function with the sigmoid activation function.
   - Modify the code to insert a dropout layer with probability 0.5 after each hidden leayer. Tip: see the function nn.dropout().

   What to hand in:

   - Your code.

- Architecture comparison: Two graphs where the y-axis is the accuracy and the x-axis is the # of epochs. The first graph should include 4 curves that show the training accuracy for 1, 2, 3, and 4 layers. The second graph should include 4 curves that show the testing accuracy for 1, 2, 3, and 4 layers. Use ReLU activation functions without any dropout and 36 nodes per hidden layer. Discuss the results.
- Activation function comparison: Two graphs where the y-axis is the accuracy and the x-axis is the # of epochs. The first graph should include 2 curves that show the training accuracy when using the ReLU versus sigmoid activation functions. The second graph should include 2 curves that show the testing accuracy when using the ReLU versus sigmoid activation functions. Use 2 layers and 36 nodes per hidden layer without any dropout. Discuss the results.
- Dropout comparison: Two graphs where the y-axis is the accuracy and the x-axis is the # of epochs. The first graph should include 2 curves that show the training accuracy with and without dropout (with probability 0.5) after each hidden layer. The second graph should include 2 curves that show the testing accuracy with and without dropout (with probability 0.5) after each hidden layer. Use 4 layers and 36 nodes with ReLU activation functions. Discuss the results.