Classification of Mushrooms Using a Multilayer Perceptron

Machine Learning Group Project Write-Up
CS 545

Group Members:

- Huy Doan
- Zeba Khan Rafi
- Camille Range
- Olubusayo Olabisi
- Siri Chandana Koduru

Introduction

Most Oregonians like hiking and exploring the vast nature that is so easily accessible. Oftentimes when out exploring the woods one might come across a mushroom. Having no knowledge about mushrooms they might decide that each mushroom they come across is poisonous and decide not to touch them. However, what if there was a way that we could incorporate our education in Machine Learning to determine which mushrooms are free to take home and eat?

We have accomplished the first step towards this in this project. We have made a classifier of mushrooms using a Multilayered Perceptron to help determine if a mushroom is edible or poisonous. This paper will provide details of the implementation of the MLP, the accuracy of this classification of whether the mushrooms are edible or poisonous, and expand upon what

happens when we alter the number of epochs, depth of hidden layers, and momentum for our model.

Implementation Details of Algorithm

In this project, we are using a Multiple Layer Perceptron to classify whether a type of mushroom is edible or not. We learned about MLP's in lecture and had an assignment in which we implemented our own MLP using only one hidden layer. However, in this project, we want to do something more challenging. Instead of using only one hidden layer in our perceptron we are implementing three hidden layers and training with different numbers of nodes in each layer.

The first step to implementing the MLP is initializing the weights of each layer. Since our model contains three hidden layers, one input layer, and one output layer, we need to initialize four different sets of weights.

After initializing our weights for each node, we then use these weights to calculate the dot-product of each layer and deploy an activation function for them. At the beginning, all activation functions in use were sigmoid functions. However, when we trained the model, the results were not as good as we had hoped. The accuracy of the training and test set was only about fifty-percent. Furthermore, when we were observing the plot for accuracy, we recognized that the accuracy of the model was not increasing at all. After several hours of researching, we decided to change the activation function of three hidden layers to a tanh function, and only keep the sigmoid function for the output layer. The results that we gained were much better by employing this methodology.

After obtaining our results from the activation function, we used these activation results for the backward propagation step. We used the derivative of the sigmoid function which is $\sigma(x) \bullet (1 - \sigma(x))$ for calculating the error of the output layer and the derivative of tanh which is

$1 - tanh^2(x)$ for calculating error of the hidden layers. For the delta weight, we used momentum in our code to prevent being stuck at local minimum.

A question that we had was why changing the activation function for the hidden layers affects our results? *"Convergence is usually faster if the average of each input variable over the training set is close to zero. To see this, consider the extreme case where all the inputs are positive. Weights to a particular node in the first weight layer are updated by an amount proportional to δx where δ is the (scalar) error at that node and x is the input vector (see equations (5) and (10)). When all of the components of an input vector are positive, all of the updates of weights that feed into a node will have the same sign (i.e. sign(δ))."* (LeCun et al.). So this is done for the same reason as applying hidden layers. We can see that the range of sigmoid function is [0,1] while tanh is [-1,1]. While using tanh, the data is centered around 0. If we use the sigmoid function in the hidden layer all the inputs to the next layer will be positive. It follows, for this reason, that all of the weights can only decrease or all increase together for a given input pattern.

One more reason that tanh is preferred to use for hidden layers is that the derivative of tanh is bigger than the derivative of sigmoid function. Because of that, the model we are training can minimize our cost function faster than using sigmoid function.

In general, in most cases: $|| \frac{\partial \bullet tanh(x)}{\partial x} || > || \frac{\partial \bullet \sigma(x)}{\partial x} ||$

Description of a Multilayer Perceptron

Multi-Layer Perceptron (MLP) is a widely used Feed Forward neural network (FFNN) that is composed of an input layer that receives the signal, an output layer that makes a decision and in between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. Multilayer perceptrons are often applied to supervised learning problems. They train on a set of input-output pairs and learn to model the correlation between those inputs and outputs. Training involves adjusting the parameters or the weights of the model in order to minimize error. Back propagation is used to make those weights adjustments relative to the error. The error itself can be measured by root mean squared error (RMSE).

Mushrooms can be classified into several categories based on edibility. A mushroom is a head or cap of a fungus. There are around 45000 types of fungi available in the world. Among them, around 2000 fungus are edible food. Mushrooms can be edible and non-edible. Selecting mushrooms with the aid of science reduces the probability to accidentally find poison in mushroom harvests. Hence, for easy recognition of edible mushrooms, this study develops a deep learning model that helps to accurately classify edible and poisonous mushrooms using a multi-layer perceptron neural network.

Analysis of Neural Network vs Decision Trees

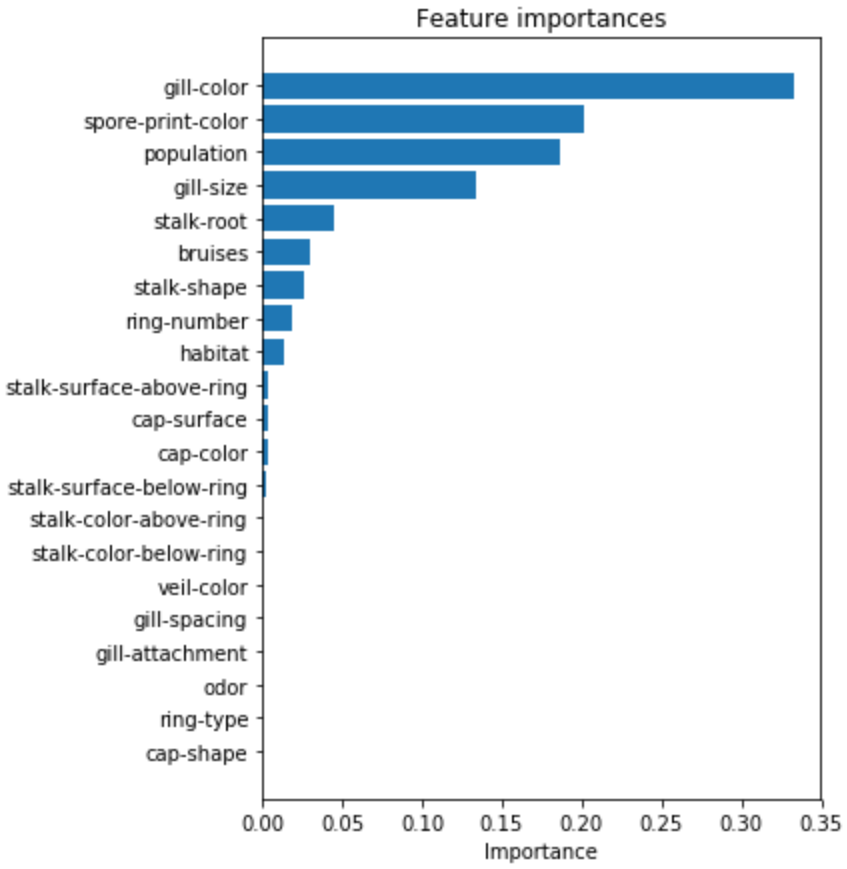
The article that we used as a main reference for the project used a Decision Tree for their model. (Feldman) While coming to accuracy comparison between Neural Networks and decision trees, each may be proficient in their way of classification. When we use a single decision tree the accuracy might be very less compared to that of a simple neural network. For decision trees to produce a very good accuracy we need to implement many more sub trees which might often

be the random forest. Here when we use neural networks the accuracy might be higher compared to that of simple decision trees.

Difference between two algorithms: The two algorithms have their own proficiency but while comparing we can say that decision trees take a lot of memory and runtime. The space complexity and time complexity of decision trees is more compared to neural networks. The complexity with decision trees is also high. One more added advantage of neural networks is that they learn from itself. For example, if we are training on the online data i.e. the data is being generated timely. The disadvantage of Decision Trees is that we need to generate trees again when new data is being added but the neural network has the capability to generate by just altering its weight values which is from the fact the neural network is able to learn from itself.

Reflection

In reflection, although our data is low-dimensional, we could have improved our accuracy, reduced our runtime and reduced overfitting by performing feature-selection. However, due to the scope of our project, we did not implement dimensionality reduction. The original data has 22 features, but not all features were actually useful in training our model. For example, features like gill-spacing and ring-type have no correlation with the output variables. Previous work on this dataset showed that only about 9 of the 22 features have significant impact in the classification of the mushroom as being edible or poisonous, as shown in the figure below. Therefore, if we had employed dimensionality reduction we could have eliminated unnecessary categories and improved our model.



(Feldman)

Conclusion

When using a Multilayered Perceptron we must always be mindful of the possibility of overfitting or underfitting the data. If we train the test for too small of an amount of time, i.e. too few epochs, we run the risk of underfitting the data. Underfitting the data could also be accomplished by our MLP not having enough nodes. In contrast, we can run the risk of overfitting the data if we train the model for too long, i.e. too many epochs. Overfitting can also be accomplished by having too many nodes in our MLP. This is something that we were mindful of when implementing this model and this is why we chose to have a depth of three hidden

layers for our model. See the Results section for the evidence that our chosen number of nodes was a decent choice for this classification problem.

Similar to the programming assignment in which we were to implement a two-layer neural network with only one hidden layer, we adjusted the momentum of our MLP. As momentum increases, the training set converges slightly faster. This is to say that a smaller momentum has a positive correlation with the increased accuracy of our model.

Given that our MLP has a very high accuracy accompanied by the fact that our accuracy is steadily increasing, shows that the size of the dataset that we chose for this classification was decent. That is not to say that if we took the whole dataset we would not also have had a high accuracy, but we might have run the risk of overfitting.

Results:

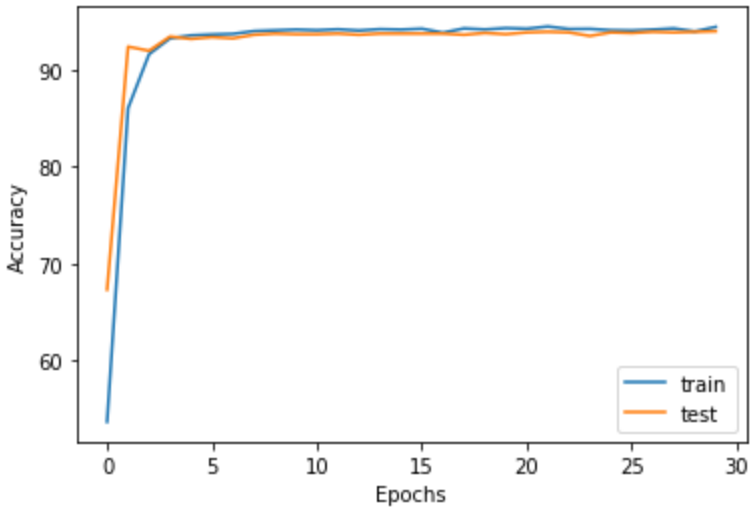
We have generated various plots varying the momentum, number of epochs and hidden layer size. We have reached a good accuracy in Experiment-3 when the hidden layer size was 50 , a momentum of 0.5 and 50 epochs.

Plots and Confusion Matrices

Experiment 1:

Accuracy of a MLP on Training vs. Testing Dataset

Here we have used the number of epochs = 30, momentum = 0.9, hidden layer size = 20



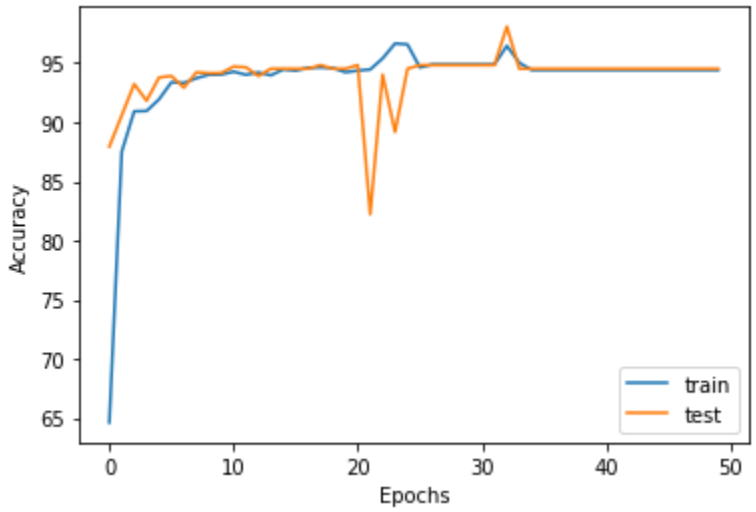
Confusion Matrix:

```
[[1052    0]
 [  97  882]]
```

Experiment 2:

Accuracy of a MLP on Training vs. Testing Dataset

Here we have taken epochs = 50, momentum = 0.9 and hidden layer size = 20



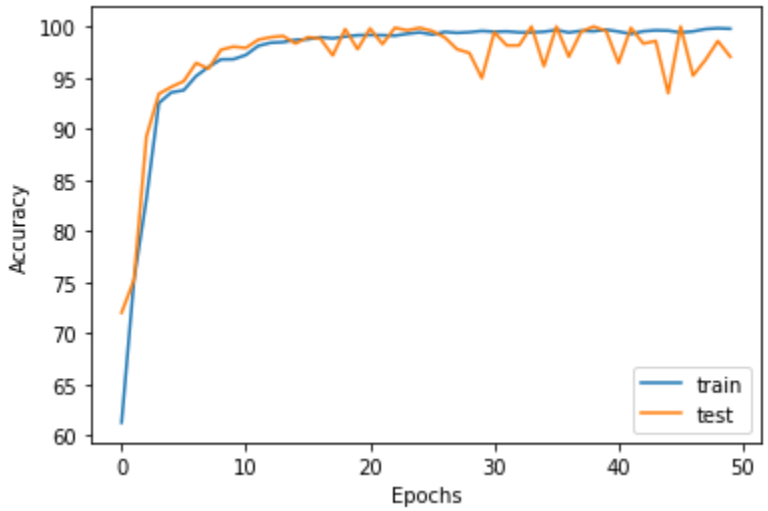
Confusion matrix:

```
[[1090    2]
 [  91  848]]
```

Experiment 3:

Accuracy of a MLP on Training vs. Testing Dataset

Here we have taken number of epochs = 50, momentum = 0.5, hidden layer size = 50



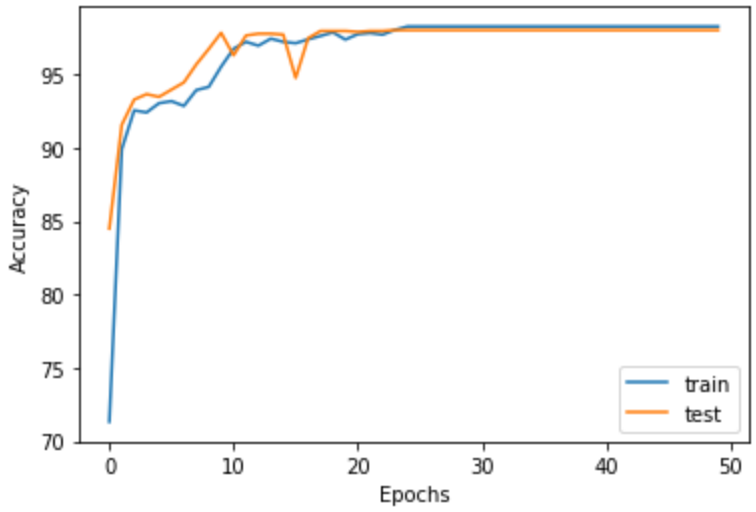
Confusion Matrix:

```
[[1062   13]
 [   0  956]]
```

Experiment 4:

Accuracy of a MLP on Training vs. Testing Dataset

Here we took number of epochs = 50, momentum = 0.9 and number of hidden layers = 50



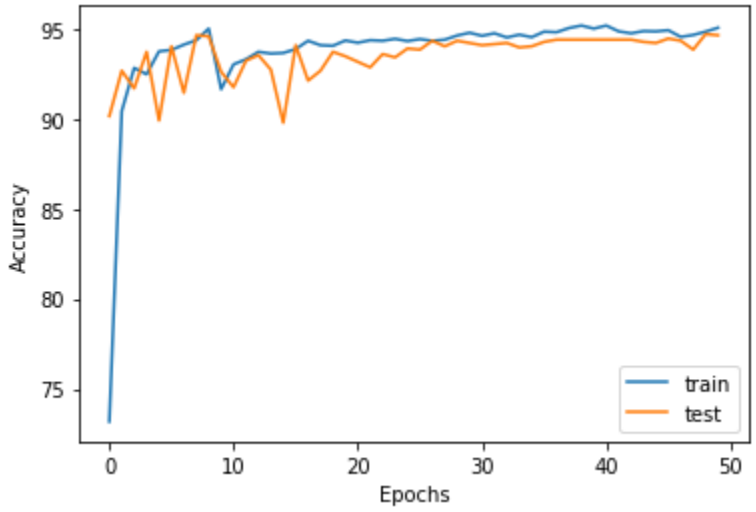
Confusion Matrix:

```
[[1041   29]
 [  24  937]]
```

Experiment 5:

Accuracy of a MLP on Training vs. Testing Dataset

Here we have taken number of epochs = 50, momentum = 0.9 and hidden layer size = 100



Confusion matrix:

```
[[1042    0]
 [   5  984]]
```

Citations

Feldman, Haim. “Analysis and Classification of Mushrooms.” *kaggle*, 2019, https://www.kaggle.com/haimfeld87/analysis-and-classification-of-mushrooms/notebook. Accessed February 2021.

LeCun, Yan, et al. “Efficient BackProp.” Springer, 1998, http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf. Accessed 18 March 2021.