10-315 Final Project: Fine Wine

Lucas Zheng, Run Gong, James Kim

May 2, 2023

1 Introduction

Our project is to predict whether wine is of good or bad quality based on its characteristics such as pH levels, acidity, and its chemical composition. The popularity and relevance of the wine industry is significant as we have professions such as sommeliers, whose job is to taste and rate quality of wine. Furthermore, wine prices vary tremendously based on their quality. Thus, it is an important problem to be able to predict the quality of wine given its measurable metrics.

The dataset that we are using is related to red wine variants of the Portuguese "Vinho Verde" wine and contains 1600 different wine variants. For each wine variant, the dataset provides metrics on acidity, sugar, density, pH, alcohol, etc, which will be inputs to our model. The dataset also contains a quality score between 0-10 for each wine, which we will transform into a classification task by classifying wines of quality less than 6.5 as "bad" and greater than 6.5 as "good". Thus, our model will be trained to output "good" or "bad" based on the inputs.

Our project calls for a binary classification task since given the input metrics, we are trying to predict whether the wine is of "good" or "bad" quality. In order to perform classification, we plan to implement logistic regression and neural networks. We also will explore how different combinations of layers for our neural network will affect its ability to predict wine quality.

2 Two Techniques

First, logistic regression is a good model for this task since in the input space, we want to create a decision boundary between good and bad-quality wines. We will be implementing this using a binary cross-entropy loss and using gradient descent to optimize the loss. This will act as our baseline model and we will compare logistic regression to a neural network. The input layer of the network consists of the input metrics and we will use multiple hidden layers with nonlinear activation functions. The output layer will then indicate the probability of the wine being of class "good" or "bad". We will then train the neural network with forward and backward propagation, adjusting the weights to best model the dataset. We

will also try out different activation functions, such as sigmoid and tanh, to see how they affect predictions. At the end, we will be able to compare the two models using metrics such as validation accuracy on the test dataset.

2.1 Logistic Regression

The logistic regression is implemented using Pytorch with a simple linear 11 dimension input to 1 dimension output. The output of this linear layer is then passed into a sigmoid function as the final ouput prediction. We then take all prediction $>= 0.5$ to be "good" and everything < 0.5 to be "bad".

```
class LogisticRegression(torch.nn.Module):
  def __init__(self, input_dim):
    super(LogisticRegression, self).__init__()
    self.linear = torch.nn.Linear(input_dim, 1)

  def forward(self, x):
    outputs = torch.sigmoid(self.linear(x))
    return outputs
```

Figure 1: Code for logistic regression class

Here are the losses are accuracies of our logistic regression model over 15 epochs.

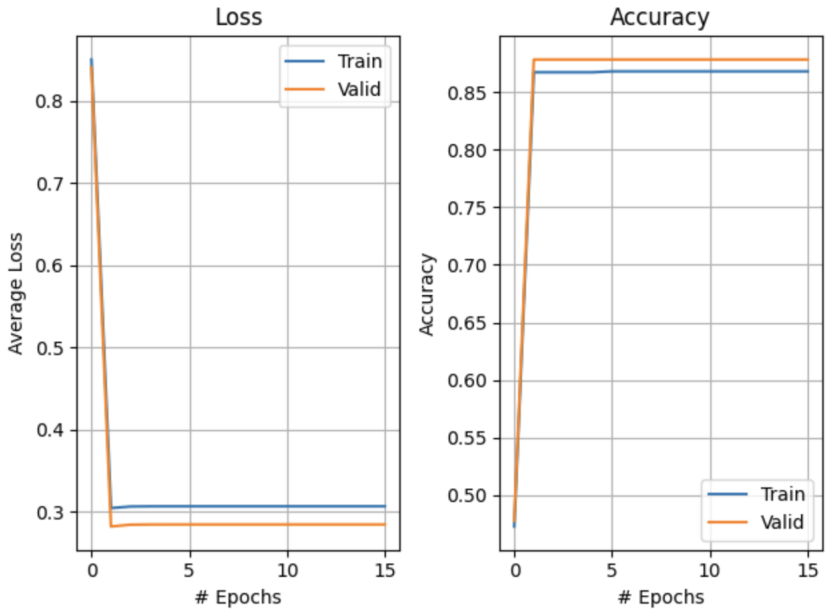

Figure 2: Training loss and accuracy as a function of number of epochs for the binary logistic regression model

2.2 Neural Network

In our neural network, we have a linear layer, followed by a relu layer, another linear layer, and finally a sigmoid activation layer at the end which outputs either 0 for bad or 1 for good.

```
class WineNet(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear1 = nn.Linear(11, 10)
    self.relu = nn.ReLU()
    self.linear2 = nn.Linear(10, 1)
    self.sigmoid = nn.Sigmoid()

  def forward(self, x):
    z = self.relu(self.linear1(x))
    return self.sigmoid(self.linear2(z))
```

Figure 3: Code for neural network class

Here are the losses are accuracies of our neural network over 30 epochs.

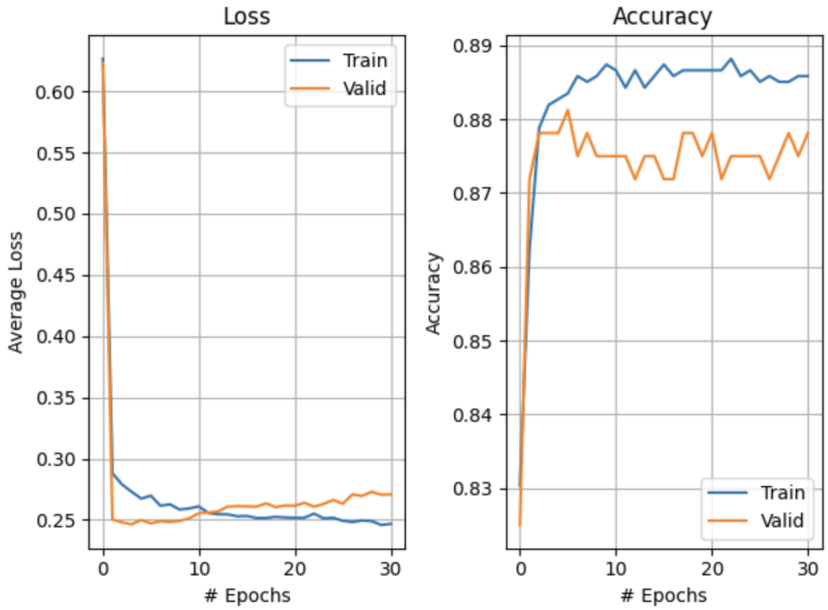

Figure 4: Training loss and accuracy as a function of number of epochs for the feed forward neural net model

3 Experiment #1: Changing Neural Network Layers

As mentioned above, our default neural network uses a linear layer, a relu layer, another linear layer, then a sigmoid function. For our experiment, we change some layers to see how it affects our results. Overall, we tested 3 different neural networks.

1. Single layer neural network with linear layer and sigmoid activation
2. Multi layer neural network with one hidden layer with ReLU activation function
3. Multi layer neural network with one hidden layer with Tanh activation function

All three neural networks were tested using learning rate of 0.01 and for 20 epochs. The multi-layer model with the ReLU activation function performed the best followed by the single layer neural network in terms of validation loss and accuracy. The model with tanh activation layer had the best train loss and accuracy at the end, but the model overfit the training data, which made the validation loss and accuracy high.

3.1 Methods

```
def test_multiple_nets():
  simplenet = SimpleNet()
  relunet = ReLUNet()
  tanhnet = TanhNet()

  # Hyperparameters
  num_epochs = 20
  learning_rate = 0.01

  print("Simple, no-hidden layer")
  trainLosses, trainAccs, validLosses, validAccs = model_train(simplenet, train_loader, valid_loader, num_epochs, learning_rate)
  plotStatistics(num_epochs, trainLosses, trainAccs, validLosses, validAccs)

  print("One hidden layer with ReLU activation")
  trainLosses, trainAccs, validLosses, validAccs = model_train(relunet, train_loader, valid_loader, num_epochs, learning_rate)
  plotStatistics(num_epochs, trainLosses, trainAccs, validLosses, validAccs)

  print("One hidden layer with Tanh activation")
  trainLosses, trainAccs, validLosses, validAccs = model_train(tanhnet, train_loader, valid_loader, num_epochs, learning_rate)
  plotStatistics(num_epochs, trainLosses, trainAccs, validLosses, validAccs)
```

Figure 5: Code for testing different neural network models

3.2 Results

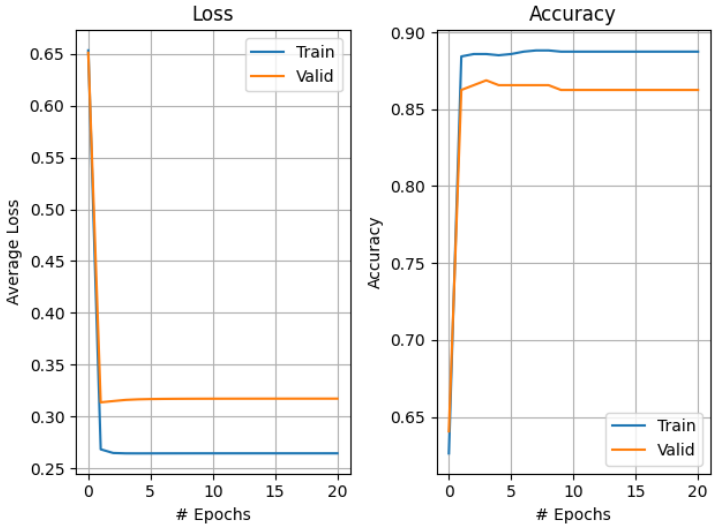

Figure 6: Training loss and accuracy as a function of number of epochs for Single-layer Neural Network

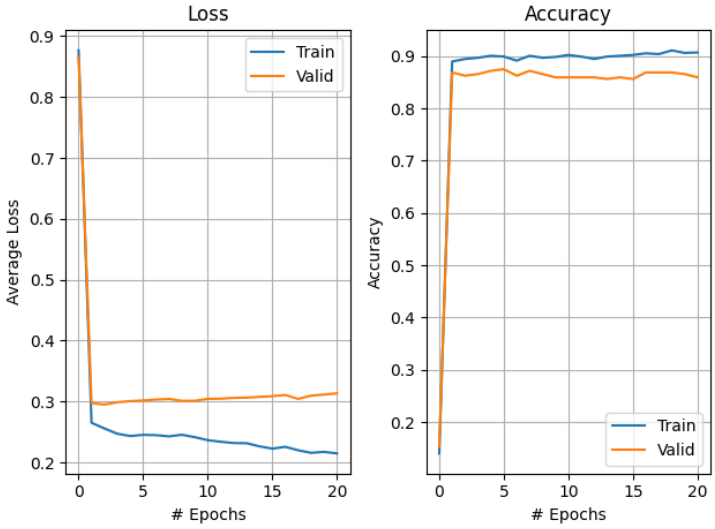

Figure 7: Training loss and accuracy as a function of number of epochs for Multi-layer Neural Network with ReLU activation

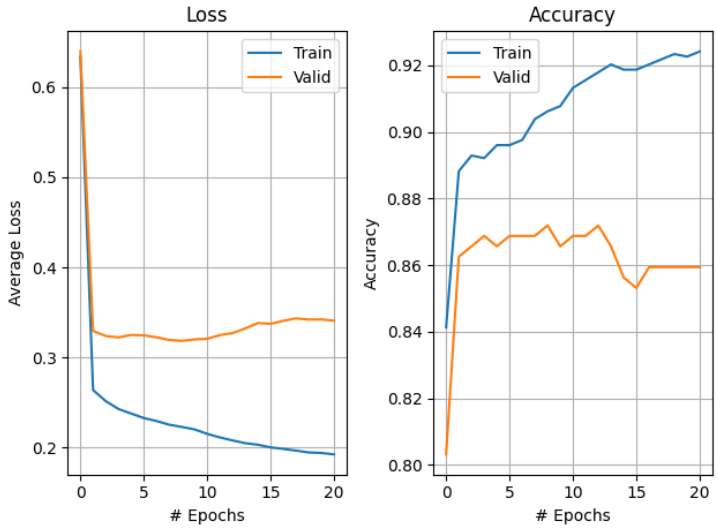

Figure 8: Training loss and accuracy as a function of number of epochs for Multi-layer Neural Network with Tanh activation

4 Experiment #2: Changing the Learning Rate

We conducted another experiment to test the best learning rate to use. As default, we used the multi-layer neural network with ReLU activation function and trained for 20 epochs. For the highest learning rate of 0.1, the loss over epoch is inconsistent and fails to converge. Furthermore, the validation loss decreases, then increases again, which shows overfitting. For the middle learning rate of 0.01, the loss and accuracy converge quickly to the minimum. However, at the end, we see signs of slight overfitting as validation accuracy decrease and loss increase slightly. Lastly, for the lowest learning rate of 0.001, the loss and accuracy converged the slowest, but we see the smoothest curve as it converges. We also do not see signs of overfitting. Overall, the learning rate of 0.001 performed the best.

4.1 Methods

```
def test_learning_rates():
  # Hyperparameters
  num_epochs = 20
  learning_rates = [0.1, 0.01, 0.001]

  for lr in learning_rates:
    relunet = ReLUNet()
    print("Learning rate: ", lr)
    trainLosses, trainAccs, validLosses, validAccs = model_train(relunet, train_loader, valid_loader, num_epochs, lr)
    plotStatistics(num_epochs, trainLosses, trainAccs, validLosses, validAccs)
```

Figure 9: Code for testing learning rates of 0.1, 0.01, 0.001

4.2 Results

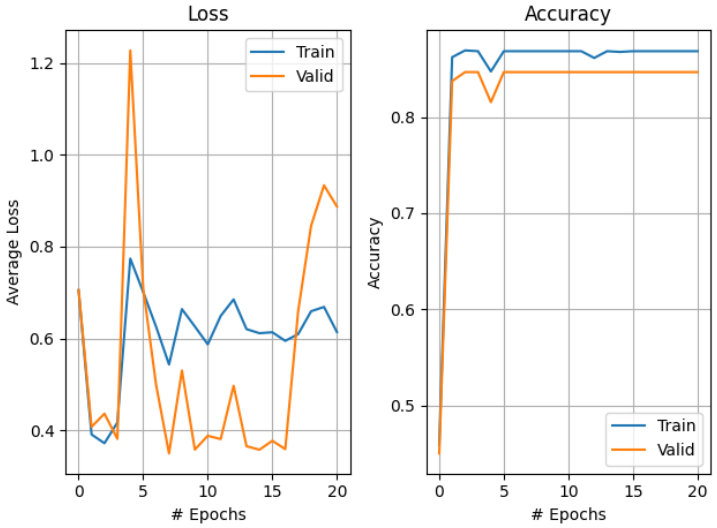

Figure 10: Training loss and accuracy as a function of number of epochs (learning rate = 0.1)

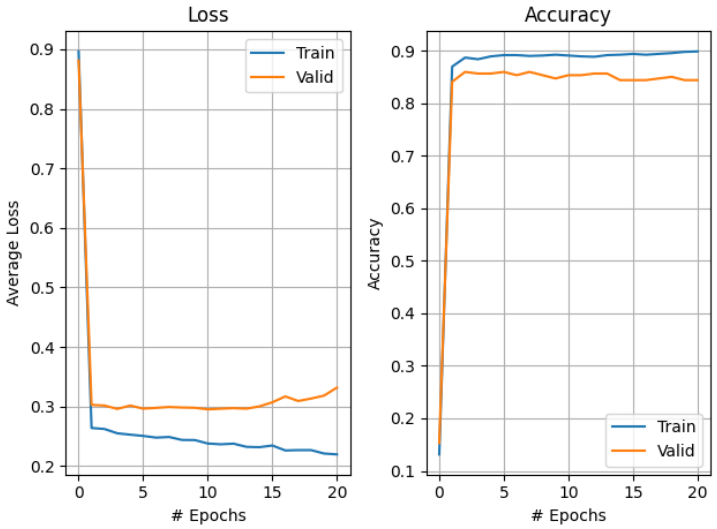

Figure 11: Training loss and accuracy as a function of number of epochs (learning rate = 0.01)

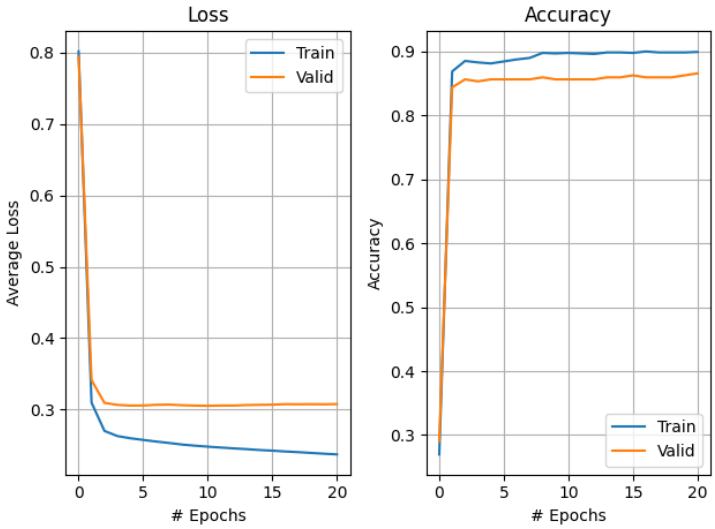

Figure 12: Training loss and accuracy as a function of number of epochs (learning rate = 0.001)

5 Conclusion

Overall all the models that tested produced similar results in terms of accuracy of classification. The logistic regression model was able to train to over 85% training accuracy in just one epoch whereas the feed forward neural networks took longer to train to their final desired accuracy. Out of all the tested models, the one that produced the highest classification accuracy is the original, feed forward neural network with a linear layer, followed by a relu layer, another linear layer, and finally a sigmoid activation layer at the end which outputs either 0 for bad or 1 for good.

References

[1] How to save and load PyTorch model

`https://pytorch.org/tutorials/beginner/saving_loading_models.html#saving-loading-model-for-inference`

[2] PyTorch feed-forward neural networks

`https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html`

[3] Validation/Training Loss Graph Code

`https://www.cs.cmu.edu/~10315/assignments/hw7_blank.pdf`

[4] Building a Binary Classification Model in PyTorch

https://machinelearningmastery.com/building-a-binary-classification-model-in-pytorch/