Emily Bates
Xzz320

Lab #1: Train a Neural Network to Predict Quality of Wine

1) *Enhancement 1: The current code does not actually evaluate the model on the test set, but it only evaluates it on the val set. When you write papers, you would ideally split the dataset into train, val and test. Train and val are both used in training, and the model trained on the training data, and evaluated on the val data. So why do we need test split? We report our results on the test split in papers. Also, we do cross-validation on the train/val split (covered in later labs). Report the results of the model on the test split. (Hint: It would be exactly like the evaluation on the val dataset, except it would be done on the test dataset.)*

```
#test results
test_loss, test_acc = evaluate(winemodel, test_dataloader, criterion)
test_preds, test_labels = more_results(winemodel, test_dataloader)

test_precision = precision_score(test_labels, test_preds,
average='weighted')
test_recall = recall_score(test_labels, test_preds, average='weighted')
test_f1 = f1_score(test_labels, test_preds, average='weighted')

print(f'| Test Loss: {test_loss:.3f} | Test Accuracy: {test_acc*100:.2f}%
| Test Precision: {test_precision*100:.2f}% | Test Recall:
{test_recall*100:.2f}% | Test F1 Score: {test_f1*100:.2f}% |')
```

```
#train results
train_loss, train_acc = evaluate(winemodel, train_dataloader, criterion)
train_preds, train_labels = more_results(winemodel, train_dataloader)

train_precision = precision_score(train_labels, train_preds,
average='weighted')
train_recall = recall_score(train_labels, train_preds, average='weighted')
train_f1 = f1_score(train_labels, train_preds, average='weighted')
print()
print(f'| Train Loss: {train_loss:.3f} | Train Accuracy:
{train_acc*100:.2f}% | Train Precision: {train_precision*100:.2f}% | Train
Recall: {train_recall*100:.2f}% | Train F1 Score: {train_f1*100:.2f}% |')
```

Result:

```
| Test Loss: 1.407 | Test Accuracy: 63.12% | Test Precision: 61.20% | Test
Recall: 62.89% | Test F1 Score: 61.82% |

| Train Loss: 1.290 | Train Accuracy: 76.64% | Train Precision: 72.90% |
Train Recall: 76.64% | Train F1 Score: 74.41% |
```

The results show that the accuracy on the test split is 63.12%, Precision is 61.20%, Recall is 62.89%, and the F1 score is 61.82%.

2) *Enhancement 2: Increase the number of epochs (and maybe the learning rate). Does the accuracy on the test set increase? Is there a significant difference between the test accuracy and the train accuracy? If yes, why?*

I increased the number of epochs from 100 to 500. While the accuracy on the test set showed a modest increase, the accuracy on the training set consistently improved with the increased number of epochs. The notable gap between the test accuracy and the training accuracy raises concerns. The persistent improvement in training accuracy, coupled with the plateauing of test accuracy, suggests overfitting in the model.

```
| Test Loss: 1.383 | Test Accuracy: 65.62% | Test Precision: 64.02% | Test
Recall: 65.41% | Test F1 Score: 64.62% |

| Train Loss: 1.242 | Train Accuracy: 80.23% | Train Precision: 76.51% |
Train Recall: 80.23% | Train F1 Score: 77.92% |
```

3) *Enhancement 3: Increase the depth of your model (add more layers). Report the parts of the model definition you had to update. Report results.*

I added two additional hidden layers to the Wine Model, adding the layers in the first definition with ReLU activation functions and then again in the forward definition.

Before changes:

```
class WineModel(torch.nn.Module):

    def __init__(self):
        super(WineModel, self).__init__()

        self.linear1 = torch.nn.Linear(11, 200)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 6)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.softmax(x)
        return x

winemodel = WineModel().to(device)
```

After changes:

```
class WineModel(torch.nn.Module):

    def __init__(self):
```

```
        super(WineModel, self).__init__()

        self.linear1 = torch.nn.Linear(11, 200)
        self.activation1 = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 200)
        self.activation2 = torch.nn.ReLU()
        self.linear3 = torch.nn.Linear(200, 200)
        self.activation3 = torch.nn.ReLU()
        self.linear4 = torch.nn.Linear(200, 6)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation1(x)
        x = self.linear2(x)
        x = self.activation2(x)
        x = self.linear3(x)
        x = self.activation3(x)
        x = self.linear4(x)
        x = self.softmax(x)
        return x

winemodel = WineModel().to(device)
```

Results:

```
| Test Loss: 1.351 | Test Accuracy: 69.58% | Test Precision: 61.04% | Test
Recall: 69.81% | Test F1 Score: 64.99% |


| Train Loss: 1.345 | Train Accuracy: 69.92% | Train Precision: 58.20% |
Train Recall: 69.92% | Train F1 Score: 63.36% |
```

I also changed the learning rate from 1e-3 to 1e-4. In comparison to the original model results (see question 1) the accuracy increased by 6.46%, the precision decreased by 0.16%, the recall increased by 6.92%, and the F1 score increased by 3.17%. The deeper model is the better model.

4) *Enhancement 4: Increase the width of your model's layers. Report the parts of the model definition you had to update. Report results.*

I took the above code and increased the 200 width to 400.

```
    def __init__(self):
        super(WineModel, self).__init__()

        self.linear1 = torch.nn.Linear(11, 400)
        self.activation1 = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(400, 400)
```

```
        self.activation2 = torch.nn.ReLU()
        self.linear3 = torch.nn.Linear(400, 400)
        self.activation3 = torch.nn.ReLU()
        self.linear4 = torch.nn.Linear(400, 6)
        self.softmax = torch.nn.Softmax()
```

Result: The Test accuracy, precision, recall, and F1 score all decreased.

```
| Test Loss: 1.433 | Test Accuracy: 60.62% | Test Precision: 50.53% | Test
Recall: 60.38% | Test F1 Score: 54.65% |
| Train Loss: 1.352 | Train Accuracy: 69.06% | Train Precision: 57.93% |
Train Recall: 69.06% | Train F1 Score: 62.74% |
```

5) *Enhancement 5: Choose a new dataset from the list below. Search the Internet and download your chosen dataset (many of them could be available on kaggle). Adapt your model to your dataset. Train your model and record your results.*

Using the Iris dataset, I got the following results:

```
| Test Loss: 0.568 | Test Accuracy: 100.00% | Test Precision: 100.00% |
Test Recall: 100.00% | Test F1 Score: 100.00% |

| Train Loss: 0.565 | Train Accuracy: 99.17% | Train Precision: 99.19% |
Train Recall: 99.17% | Train F1 Score: 99.17% |
```

I used the same number of hidden layers for my Iris Model as the Wine model, but decreased the width. I will attach my code for the adapted Iris model.

```
class IrisModel(torch.nn.Module):

    def __init__(self):
        super(IrisModel, self).__init__()

        self.linear1 = torch.nn.Linear(4, 100)
        self.activation1 = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(100, 100)
        self.activation2 = torch.nn.ReLU()
        self.linear3 = torch.nn.Linear(100, 100)
        self.activation3 = torch.nn.ReLU()
        self.linear4 = torch.nn.Linear(100, 3)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
```

```
        x = self.activation1(x)
        x = self.linear2(x)
        x = self.activation2(x)
        x = self.linear3(x)
        x = self.activation3(x)
        x = self.linear4(x)
        x = self.softmax(x)
        return x

irismodel = IrisModel().to(device)
```