Dan Schumacher
2/2/2024
Dr. Vishwamitra's Deep Learning

Lab-1 Writeup

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.)

- We need a test split to avoid overfitting on the Val-dataset.
- Below are the lines of code I updated.
    - Import the relevant metrics.

```
from sklearn.metrics import (
    accuracy_score,
    # ENHANCEMENT 1
    precision_score,
    recall_score,
    f1_score
)
```

    - Run up a tally and get the avg metric per epoch.

```
def evaluate(model, data_loader, criterion):
    epoch_loss = 0
    epoch_acc = 0
    # ENHANCEMENT 1
    epoch_precision = 0
    epoch_recall = 0
    epoch_f1 = 0
```

```
# ENHANCEMENT 1
precision = precision_score(labels.cpu(), preds.cpu(), average='weighted', zero_division=0)
recall = recall_score(labels.cpu(), preds.cpu(), average='weighted',zero_division=0)
f1 = f1_score(labels.cpu(), preds.cpu(), average='weighted',zero_division=0)
epoch_precision += precision
epoch_recall += recall
epoch_f1 += f1
```

```
num_batches = len(data_loader)
return (
    epoch_loss / num_batches,
    epoch_acc / num_batches,

    # ENHANCEMENT 1
    epoch_precision / num_batches,
    epoch_recall / num_batches,
    epoch_f1 / num_batches
)
```

    - Update unpacking and print statement in training (note that some is cut off).

```
# Let's train our model
for epoch in range(100):
    train_loss, train_acc = train(winemodel, train_dataloader, optimizer, criterion)
    # ENHANCEMENT 1
    valid_loss, valid_acc, valid_precision, valid_recall, valid_f1  = evaluate(winemodel, val_dataloader, criterion)

    print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}% | Val. Loss: {valid_loss:.3f}
```

- Run evaluate on test data and update print statement (note that some is cut off).

```
test_loss, test_acc, test_prec, test_rec, test_f1 = evaluate(winemodel, test_dataloader, criterion)

print(f'| Test. Loss: {test_loss:.3f} | Test. Acc: {test_acc*100:.2f}% | Test. Precision: {test_prec:.3f} | Test. Recall:
```

- Train-Dev Results
  - | Epoch: 100 | Train Loss: 1.302 | Train Acc: 74.92% | Val. Loss: 1.442 | Val. Acc: 58.13% | Val. Precision: 0.661 | Val. Recall: 0.581 | Val. F1-score: 0.590 |
- Test-Results
  - | Test. Loss: 1.422 | Test. Acc: 62.29% | Test. Precision: 0.702 | Test. Recall: 0.623 | Test. F1-score: 0.626 |

Enhancement 2: Increase the number of epochs (and maybe the learning rate).
RESULTS

- Code I updated:

```
                    # ENHANCEMENT 2
for epoch in range(500):
```

```
# Define and the loss function and optimizer
criterion = nn.CrossEntropyLoss().to(device)
                                                # ENHANCEMENT 2
optimizer = AdamW(winemodel.parameters(), lr = 1e-2)
```

- Train-Dev Results
  - | Epoch: 500 | Train Loss: 1.629 | Train Acc: 41.41% | Val. Loss: 1.575 | Val. Acc: 46.88% | Val. Precision: 0.289 | Val. Recall: 0.468 | Val. F1-score: 0.347 |
- Test Results
  - | Test. Loss: 1.575 | Test. Acc: 46.88% | Test. Precision: 0.277 | Test. Recall: 0.469 | Test. F1-score: 0.337 |

Does the accuracy on the test set increase?

- NO! It was 15.81% worse. Down from 62.69% to 46.88%.

Is there a significant difference between the test accuracy and the train accuracy? If yes, why?

- The test-accuracy was better than the final epoch of the train-accuracy by about five percent. I think that this is because the learning rate is too large. The model performed best on the original learning rate: 1e-3. I tried larger (1e-2, 1e-1) and smaller (1e-4) learning rates but the original, regardless of the number of epochs (up to 500), has performed the best. Sometimes larger learning rates perform much worse than smaller learning rates because they can overshoot the true minimum repeatedly.

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

- CODE I UPDATED

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

    self.linear1 = torch.nn.Linear(11, 200)
    self.activation = torch.nn.ReLU()
                                    # ENHANCEMENT 3
    self.linear2 = torch.nn.Linear(200, 200)
    self.linear3 = torch.nn.Linear(200, 6)

    self.softmax = torch.nn.Softmax(dim =1)
```

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

    # ENHANCEMENT 3
    x = self.activation(x)
    x = self.linear3(x)

    x = self.softmax(x)
    return x
```

- Train-Dev Results
  - | Epoch: 500 | Train Loss: 1.211 | Train Acc: 83.20% | Val. Loss: 1.447 | Val. Acc: 60.00% | Val. Precision: 0.573 | Val. Recall: 0.600 | Val. F1-score: 0.561 |
- Test Results
  - | Test. Loss: 1.375 | Test. Acc: 66.88% | Test. Precision: 0.648 | Test. Recall: 0.669 | Test. F1-score: 0.633 |

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

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

    def __init__(self):
        super(WineModel, self).__init__()
                                        # ENHANCEMENT 4 (Larger right number)
        self.linear1 = torch.nn.Linear(11, 200)
        self.activation = torch.nn.ReLU()
                                    # ENHANCEMENT 3
        self.linear2 = torch.nn.Linear(200, 400) # ENHANCEMENT 4 (Larger numbers)
        self.linear3 = torch.nn.Linear(400, 6) # ENHANCEMENT 4 (Larger left number)

        self.softmax = torch.nn.Softmax(dim =1)
```

- Train-Dev Results
  - | Epoch: 500 | Train Loss: 1.234 | Train Acc: 81.02% | Val. Loss: 1.367 | Val. Acc: 67.50% | Val. Precision: .6600 | Val. Recall: .6750 | Val. F1-score: .6367 |
- Test Results
  - | Test. Loss: 1.461 | Test. Acc: 56.67% | Test. Precision: 0.586 | Test. Recall: 0.567 | Test. F1-score: 0.539 |

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.

- | Epoch: 50 | Train Loss: 0.318 | Train Acc: 99.56% | Val. Loss: 0.325 | Val. Acc: 98.33% | Val. Precision: 1.000 | Val. Recall: 0.983 | Val. F1-score: 0.990 |
- | Test. Loss: 0.349 | Test. Acc: 96.43% | Test. Precision: 0.979 | Test. Recall: 0.964 | Test. F1-score: 0.964 |