Recall, Precision, and F1 Score

Task

For this task, we will use the assignment from week 3 i.e. MNIST (without the addition task). Also, to make the interpretation of the scores easier, we will turn it into a binary classification problem: is the output class/digit even (0,2,4,6,8) or odd (1,3,5,7,9)

```
!pip install tableprint
```

```
    Requirement already satisfied: tableprint in /usr/local/lib/python3.7/dist-pa
    Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packag
    Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packa
```

```
def score(targets, predictions):
  tp, tn, fp, fn = 0, 0, 0, 0
  for target, prediction in zip(targets, predictions):
    if target == 1:
      if prediction ==1:
        tp += 1
      else:
        fn += 1
    else:
      if prediction ==1:
        fp += 1
      else:
        tn += 1
  return tp, tn, fp, fn
```

Sanity Check

After tabulating the result, we see that the calculating accuracy using our `score` function matches the results from our previous work (also tabulated!)

Imports

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import numpy as np
import torch.nn.functional as F
from torchsummary import summary

import tableprint as tp
```

Loading the MNIST Dataset

```
train_dataset = torchvision.datasets.MNIST(
                    root='.',
                    train=True,
                    transform=transforms.ToTensor(),
                    download=True)

test_dataset = torchvision.datasets.MNIST(
                    root='.',
                    train=False,
                    transform=transforms.ToTensor(),
                    download=True)
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:498: Use
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

Turn targets into binary class

```
def ev(x):
  if x%2 == 0:
    return 1
  else:
    return 0

train_dataset.targets.apply_(ev)
test_dataset.targets.apply_(ev)
```

```
tensor([0, 1, 0,  ..., 1, 0, 1])
```

```
print(train_dataset.targets.max(), train_dataset.targets.min())
print(test_dataset.targets.max(), test_dataset.targets.min())
```

```
tensor(1) tensor(0)
tensor(1) tensor(0)
```

Re-creating Dataset

```
# Flattens the MNIST images
train_x = train_dataset.data.reshape(60000, 784).float()
test_x = test_dataset.data.reshape(10000, 784).float()
test_x = test_dataset.data.reshape(10000, 784).float()

# Creats the Dataset with the modified targets
train_ds = torch.utils.data.TensorDataset(train_x, train_dataset.targets)
test_ds = torch.utils.data.TensorDataset(test_x, test_dataset.targets)
```

DataLoader

```
batch_size = 32
train_loader = torch.utils.data.DataLoader(
                        dataset=train_ds,
                        batch_size=batch_size,
                        shuffle=True
                        )

test_loader = torch.utils.data.DataLoader(
                        dataset=test_ds,
                        batch_size=batch_size,
                        shuffle=False # Not necessary!
                        )
```

Model

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear1 = nn.Linear(784, 30) # Flattened MNIST as Input: 28 * 28
        self.relu = nn.ReLU()
        self.selu = nn.SELU()
        self.linear2 = nn.Linear(30, 30)
        self.linear3 = nn.Linear(30,2) # 2 Classes for The Output: 0-1


    def forward(self, Xa):

        out = self.linear1(Xa)
        out = self.selu(out)
        out = self.linear2(out)
        out = self.selu(out)
        out = self.linear3(out)
        out = self.selu(out)


        return out

# Instantiate the Model
model = Model()
```

Move Model to GPU (Required Condition by Assignment!)

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model.to(device)
```

```
cpu
Model(
  (linear1): Linear(in_features=784, out_features=30, bias=True)
  (relu): ReLU()
  (selu): SELU()
  (linear2): Linear(in_features=30, out_features=30, bias=True)
  (linear3): Linear(in_features=30, out_features=2, bias=True)
)
```

Model Summary

```
summary(model, [(1,784)])
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 30]          23,550
              SELU-2                [-1, 1, 30]               0
            Linear-3                [-1, 1, 30]             930
              SELU-4                [-1, 1, 30]               0
            Linear-5                 [-1, 1, 2]              62
              SELU-6                 [-1, 1, 2]               0
================================================================
Total params: 24,542
Trainable params: 24,542
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.09
Estimated Total Size (MB): 0.10
----------------------------------------------------------------
```

Loss and Optimizer

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())
```

Training Loop

```
n_epochs = 20
```

```
    table_context = tp.TableContext(headers=['epoch', 'Train Acc', 'Train Loss', 'Vali
    table_context.__enter__()
    for epoch in range(n_epochs):
        train_loss = []
        n_correct = 0.
        n_total = 0.

        for inputs, targets in train_loader:

            # Move data to GPU
            inputs, targets = inputs.to(device), targets.to(device)

            # zero the gradient
            optimizer.zero_grad()

            # forward pass
            output = model(inputs)
            loss = loss_fn(output, targets)

            # get predictiona
            _, prediction= torch.max(output, 1)

            # update counts
            n_correct += (prediction == targets).sum().item()
            n_total += targets.shape[0]

            # backward pass and optimize
            loss.backward()
            optimizer.step()

            train_loss.append(loss.item())

        train_loss = np.mean(train_loss)
        train_acc = n_correct / n_total * 100


        test_loss = []
        n_correct = 0.
        n_total = 0.
        tpt, tnt, fpt, fnt = 0, 0, 0, 0
        for inputs, targets in test_loader:

            # Move data to GPU
            inputs, targets = inputs.to(device), targets.to(device)

            # forward pass
            output = model(inputs)
            loss = loss_fn(output, targets)

             # get predictions
            _, prediction = torch.max(output, 1)


            # update counts
            n_correct += (prediction == targets).sum().item()
            n_total += targets.shape[0]

            test_loss.append(loss.item())
            tp, tn, fp, fn = score(prediction, targets)
            tpt += tp
            tnt += tn
            fpt += fp
            fnt += fn

        test_loss = np.mean(test_loss)
        test_acc = n_correct / n_total * 100
        a = (tpt+tnt) / (tpt+tnt +fpt +fnt)
        r = tpt / (tpt + fnt)
        p = tpt / (tpt + fpt)
        f1 = 2 * (p*r)/(p+r)
        table_context([epoch+1, train_acc, train_loss, test_acc, test_loss, a, r, p, f
    table_context.__exit__()
```

```
┌─────────────┬─────────────┬─────────────┬─────────────┬─────────────┬──────
│       epoch │   Train Acc │  Train Loss │   Valid Acc │  Valid Loss │     V
├─────────────┼─────────────┼─────────────┼─────────────┼─────────────┼──────
│           1 │      90.043 │     0.24497 │       96.83 │    0.098878 │
│           2 │      96.978 │    0.091356 │       96.99 │    0.098771 │
│           3 │       97.54 │    0.072313 │       97.83 │    0.072959 │
│           4 │      97.878 │    0.062474 │       98.02 │    0.066016 │
│           5 │      98.103 │    0.057452 │       97.29 │    0.082461 │
│           6 │      98.317 │    0.051321 │       98.09 │    0.063521 │
│           7 │      98.402 │    0.047289 │       97.48 │    0.085557 │
│           8 │      98.535 │    0.042866 │       98.17 │    0.059476 │
│           9 │       98.55 │    0.041795 │       98.32 │    0.055634 │
│          10 │      98.732 │    0.037779 │        98.1 │    0.067117 │
│          11 │      98.762 │    0.036443 │       97.77 │    0.068822 │
│          12 │      98.843 │    0.035164 │       97.98 │    0.084554 │
│          13 │      98.942 │    0.030886 │       98.19 │    0.059014 │
│          14 │      99.002 │    0.029302 │       98.18 │    0.066869 │
│          15 │      98.982 │    0.029094 │       98.15 │    0.066012 │
│          16 │      99.092 │    0.026849 │       98.13 │    0.068708 │
│          17 │      99.077 │    0.027501 │       98.16 │    0.064794 │
│          18 │      99.165 │    0.024696 │       98.31 │     0.07599 │
│          19 │       99.13 │    0.024841 │       98.26 │    0.084453 │
│          20 │      99.225 │    0.024032 │        98.4 │    0.083581 │
└─────────────┴─────────────┴─────────────┴─────────────┴─────────────┴──────
```