Trombini_Quentin_Bonus

For that bonus I tried to push the exercises of S15 a bit further by passing the data through a neural network I made by myself using pytorch

Code

```
from sklearn.model_selection import train_test_split
import torch.nn as nn
import torch.optim as optim
import torch
import numpy as np
import matplotlib.backends.backend_agg as agg

with open("./S15/main.dat") as f:
    main_lines = f.readlines()
with open("./S15/photo.dat") as f:
    photo_lines = f.readlines()

EPOCHS = 20

dataset = []
improper = 0
for mline, pline in zip(main_lines,photo_lines):
    msplit = mline.split("|")
    psplit = pline.split("|")

    #r ID, U,B,V,R,I,J,H,K, absolute magnitude,spectral type
    useful = [msplit[0], psplit[8], psplit[9], psplit[10], pspli

    data_line = [int(useful[0])]
    try:
```

```
        for i in range(1,8):
            data_line.append(float(useful[i])-float(useful[i+1])
        #data_line = [float(useful[1])-float(useful[2]), float(u

        data_line.append(float(useful[-2]))
        data_line.append(useful[-1].strip())
        data_line[-1] = data_line[-1][:1].upper()
        dataset.append(data_line)
    except:
        improper +=1

print(dataset[:1])
print(f"{improper} removed for lacking some data, left {len(data

train, test = train_test_split(dataset,train_size=0.8,shuffle=Tr
print(f"Training dataset_size: {len(train)}, Test dataset_size:

train_x =[]
train_y = []
test_x = []
test_y = []
for tr_line, te_line in zip(train,test):
    train_x.append(tr_line[1:-1])
    train_y.append(tr_line[-1:])
    test_x.append(te_line[1:-1])
    test_y.append(te_line[-1:])


train_x = np.array(train_x)
train_y = np.array(train_y)

label = ["M","K","F","D","G","A","B","S","O"]
print(label)
print(train_x[:1])

buffer_train = []
```

```
for i in range(len(train_y)):
    buffer = [0]*len(label)
    buffer[label.index(train_y[i])] = 1
    buffer_train.append(buffer)
train_y = buffer_train
buffer_train = []
for i in range(len(test_y)):
    buffer = [0]*len(label)
    buffer[label.index(test_y[i][0])] = 1
    buffer_train.append(buffer)
test_y = buffer_train

train_x = torch.tensor(train_x,dtype=torch.float)
train_y = torch.tensor(train_y,dtype=torch.float)
test_x = torch.tensor(test_x,dtype=torch.float)
test_y = torch.tensor(test_y,dtype=torch.float)

class NN(nn.Module):
    def __init__(self, input_size,output_size):
        super(NN, self).__init__()
        self.fc = nn.Linear(input_size, output_size)
        self.drop = nn.Dropout()
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(output_size, output_size)
        self.soft = nn.Softmax(dim=0)

    def forward(self, x):
        x = self.fc(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.drop(x)
        x = self.soft(x)
        return x

model = NN(len(train_x[0]),len(label))
criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.Adam(model.parameters(),lr=0.1)

for epoch in range(EPOCHS):
    sum_acc = 0
    sum_loss = 0
    for item,target in zip(train_x,train_y):
        optimizer.zero_grad()
        output = model(item)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if torch.max(output) == torch.max(target):
            sum_acc += 1
        sum_loss += loss.item()
    print(f"[EPOCH {epoch+1}/{EPOCHS}] Average loss: {sum_loss/l

    if (epoch+1)%5==0:
        with torch.no_grad():
            sum_acc = 0
            sum_loss = 0
            for item,target in zip(test_x,test_y):
                output = model(item)
                loss = criterion(output, target)
                if torch.max(output) == torch.max(target):
                    sum_acc += 1
                sum_loss += loss.item()
            print(f"[Validation] Average loss: {sum_loss/len(tra
```

Output

```
[EPOCH 1/20] Average loss: 1.8841152227286138, Accuracy: 0.3636
[EPOCH 2/20] Average loss: 1.8756543470151497, Accuracy: 0.4848
[EPOCH 3/20] Average loss: 1.8753780379439846, Accuracy: 0.4242
[EPOCH 4/20] Average loss: 1.8756630059444543, Accuracy: 0.4848
```

```
[EPOCH 5/20] Average loss: 1.9176650011178218, Accuracy: 0.5151
[Validation] Average loss: 2.0493825818553115, Accuracy: 0.3636
[EPOCH 6/20] Average loss: 1.8704993255210645, Accuracy: 0.4545
[EPOCH 7/20] Average loss: 1.851493452534531, Accuracy: 0.48484
[EPOCH 8/20] Average loss: 1.8262679793617942, Accuracy: 0.5454
[EPOCH 9/20] Average loss: 1.8831872940063477, Accuracy: 0.4545
[EPOCH 10/20] Average loss: 1.8695283405708545, Accuracy: 0.515
[Validation] Average loss: 1.9710160096486409, Accuracy: 0.5757
[EPOCH 11/20] Average loss: 1.9448183811072148, Accuracy: 0.454
[EPOCH 12/20] Average loss: 1.9444581234093867, Accuracy: 0.333
[EPOCH 13/20] Average loss: 1.9598044590516523, Accuracy: 0.242
[EPOCH 14/20] Average loss: 1.8445177547859424, Accuracy: 0.484
[EPOCH 15/20] Average loss: 1.9719586805863814, Accuracy: 0.333
[Validation] Average loss: 2.0062862381790625, Accuracy: 0.3939
[EPOCH 16/20] Average loss: 1.8914986270846743, Accuracy: 0.484
[EPOCH 17/20] Average loss: 1.9319515336643567, Accuracy: 0.333
[EPOCH 18/20] Average loss: 1.9353575598109851, Accuracy: 0.424
[EPOCH 19/20] Average loss: 1.8155760331587358, Accuracy: 0.515
[EPOCH 20/20] Average loss: 1.84313094254696, Accuracy: 0.45454
[Validation] Average loss: 1.911083875280438, Accuracy: 0.54545
```

Explaination

The preprocessing part is the same one as the S15 exercice I didn’t change anything on it

Training

```
class NN(nn.Module):
    def __init__(self, input_size,output_size):
        super(NN, self).__init__()
        self.fc = nn.Linear(input_size, output_size)
        self.drop = nn.Dropout()
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(output_size, output_size)
```

```
        self.soft = nn.Softmax(dim=0)

    def forward(self, x):
        x = self.fc(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.drop(x)
        x = self.soft(x)
        return x
```

First I used torch.nn to create a neural network that will take the right number of value in input and output an array in wich the maximal value is the class that is represented.

Hyperparameters

```
model = NN(len(train_x[0]),len(label))
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),lr=0.1)
```

I used Crossentropy loss as the loss function because it’s the one giving the best result combined with adam optimizer on multiclass labelisation.

Training

```
for epoch in range(EPOCHS):
    sum_acc = 0
    sum_loss = 0
    for item,target in zip(train_x,train_y):
        optimizer.zero_grad()
        output = model(item)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if torch.max(output) == torch.max(target):
            sum_acc += 1
        sum_loss += loss.item()
```

```
    print(f"[EPOCH {epoch+1}/{EPOCHS}] Average loss: {sum_loss/l

    if (epoch+1)%5==0:
        with torch.no_grad():
            sum_acc = 0
            sum_loss = 0
            for item,target in zip(test_x,test_y):
                output = model(item)
                loss = criterion(output, target)
                if torch.max(output) == torch.max(target):
                    sum_acc += 1
                sum_loss += loss.item()
            print(f"[Validation] Average loss: {sum_loss/len(tra
```

For each epoch the model will iterate through the whole training dataset and output the accuracy and the loss value this allow us to see how it is progressing. Each five epoch it goes through the test dataset to see if it’s able to generalize well or not. the generalization of this one is around 50% that’s not that good but I think the main issue is that after the data preprocessing only 168 dataline are left and a neural network usually need a way bigger dataset than that (around 1000 entry for the training at least)