Class4_NN

January 29, 2025

[1]:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

import torch
from torch import nn
from torch import optim
from torch.utils.data import Dataset, DataLoader
```

[2]:

```python
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

0.0.1 Dataset preparation

- Import Dataset and Get Properties

[3]:

```python
df = pd.read_csv(r'./dataset/WineQT.csv')

target_name = 'quality' # integer between 0 - 10

print("Datapoint shape:", df.shape)
print("Attribute:", list(df.columns))
```

```
Datapoint shape: (1143, 13)
Attribute: ['fixed acidity', 'volatile acidity', 'citric acid', 'residual
sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
'pH', 'sulphates', 'alcohol', 'quality', 'Id']
```

[4]:

```python
# Remove unecessary attribute
df = df.drop(columns=["Id"])
```

- Inspect dataset

[5]:

```python
stat = df.describe()
mean, std = stat.iloc[1], stat.iloc[2]
display(pd.concat([mean, std], axis=1).T)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
mean       8.311111          0.531339     0.268364        2.532152   0.086933
std        1.747595          0.179633     0.196686        1.355917   0.047267

      free sulfur dioxide  total sulfur dioxide   density        pH  \
mean            15.615486             45.914698  0.996730  3.311015
std             10.250486             32.782130  0.001925  0.156664

      sulphates    alcohol   quality
mean   0.657708  10.442111  5.657043
std    0.170399   1.082196  0.805824
```

- Preprocessing

  Z transform (Normalization)

```
[6]: def z_norm(dataset: pd.DataFrame, target: str):
         answer = dataset[target]
         dataset = dataset.drop(columns=[target])
         for column in dataset.columns:
             dataset[column] = (dataset[column] - dataset[column].mean()) /␣
       ↪dataset[column].std()
         display(dataset.describe()[1:3].astype(np.int32))

         return dataset, answer.to_numpy(dtype=np.int8)

     train, test = z_norm(df.copy(), target_name)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
mean              0                 0            0               0          0
std               1                 1            1               0          0

      free sulfur dioxide  total sulfur dioxide  density  pH  sulphates  \
mean                    0                     0        0   0          0
std                     0                     1        0   0          0

      alcohol
mean        0
std         1
```

Train test split

```
[7]: import random
     random.seed(42)
     np.random.seed(42)
     x_train, x_test, y_train, y_test = train_test_split(train, test, train_size=0.
       ↪8, shuffle=True)
     print(len(x_train), len(y_test))
```

```
914 229
```

Convert to torch.Tensor

[8]:

```
x_train, x_test, y_train, y_test = (
    torch.tensor(x_train.to_numpy(), dtype=torch.float32),
    torch.tensor(x_test.to_numpy(), dtype=torch.float32),
    torch.tensor(y_train, dtype=torch.long),
    torch.tensor(y_test, dtype=torch.long),
)

# Remove offset
y_train = y_train - y_train.min()
y_test = y_test - y_test.min()
```

[9]:

```
print("train:", x_train.shape)
```

```
train: torch.Size([914, 11])
```

0.0.2 Pipeline setup

[10]:

```
# Must Preprocess first
class WineDataset(Dataset):
    def __init__(self, x, y):
        self.features = x
        self.target = y

    def __len__(self):
        return len(self.features)

    def __getitem__(self, idx):
        return self.features[idx].to(device), self.target[idx].to(device)

train_dataset = WineDataset(x_train, y_train)
test_dataset = WineDataset(x_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)
```

0.0.3 Model Setup

- Define Model

[11]:

```
class RegressionNN(nn.Module):
    def __init__(self, in_dim, out_dim, dtype):
        super().__init__()
        self.input_layer = nn.Linear(in_dim, 64, dtype=dtype)
        self.hidden_layer = nn.Linear(64, 32, dtype=dtype)
        self.output_layer = nn.Linear(32, out_dim, dtype=dtype)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.input_layer(x))
        x = self.relu(self.hidden_layer(x))
        y = self.output_layer(x)
        return y
```

[12]:
```
next(iter(train_loader))[0].shape, next(iter(train_loader))[1].shape
```

[12]:
```
(torch.Size([32, 11]), torch.Size([32]))
```

[13]:
```
len(np.unique(y_train.numpy()))
```

[13]:
```
6
```

[14]:
```
torch.manual_seed(42)

model = RegressionNN(11, len(np.unique(y_train.numpy())), torch.float32).
  ↪to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), 1e-3)
```

0.0.4 Training Loop

[15]:
```
model.train()
for i in range(1, 101):
    iter_loss = 0
    for batch in train_loader:
        input_batch, target_batch = batch

        optimizer.zero_grad()

        outputs = model(input_batch)

        loss = loss_fn(outputs, target_batch)
        loss.backward()

        optimizer.step()

        iter_loss += loss

    if i%20 == 0:
        print(f"loss at iter {i} = {loss/len(train_loader)}")
```

```
loss at iter 20 = 0.031606599688529 97
```

```
loss at iter 20 = 0.03160659968852997
loss at iter 40 = 0.026845039799809456
loss at iter 60 = 0.02331145852804184
loss at iter 80 = 0.020234109833836555
loss at iter 100 = 0.018250463530421257
```

0.0.5 Eval Model

[16]:

```
model.eval()

total_loss = 0

for i, batch in enumerate(test_loader):
    input_batch, target_batch = batch

    with torch.no_grad():
        outputs = model(input_batch)
        loss = loss_fn(outputs, target_batch)
        total_loss += loss

print(f"loss of testset = {total_loss/len(test_loader)}")
```

```
loss of testset = 3.3306384086608887
```