IMDb_Sentiment_analysis_(Rafael_Ito)

March 18, 2020

1 Sentiment analysis

Author: **Rafael Ito**
e-mail: ito.rafael@gmail.com

1.1 0. Dataset and Description

Name: IMDb
Description: this notebook uses the IMDb dataset which contains movie reviews classified as either positive or negative review. The aim is to perform a supervised learning for sentiment classification.

1.2 1. Libraries and packages

1.2.1 1.1 Install packages

[0]:
```
!pip install -q \
    numpy       \
    torch       \
    sklearn     \
    skorch      \
    matplotlib
```

1.2.2 1.2 Import libraries

[0]:
```
#--------------------------------------------------
# general
#-------------------
import numpy as np
import pandas as pd
import re
#--------------------------------------------------
# PyTorch
#-------------------
import torch
from torch.utils.data import TensorDataset
import torch.nn.functional as F
#--------------------------------------------------
# skorch
#-------------------
#from skorch import NeuralNetClassifier
#--------------------------------------------------
# scikit-learn
#-------------------
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import OneHotEncoder
# metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
#--------------------------------------------------
# data visualization
#-------------------
import matplotlib.pyplot as plt
import seaborn as sns
#--------------------------------------------------
```

```
# additional config
#-------------------
# random seed generator
torch.manual_seed(42);
```

1.2.3 1.3 Check device

[3]:
```
device = torch.device('cpu')
if torch.cuda.is_available():
    device = torch.device('cuda')
print('Device:', device)
```

```
Device: cuda
```

1.2.4 1.4 Function to plot confusion matrix

[0]:
```
#https://gist.github.com/shaypal5/94c53d765083101efc0240d776a23823
def print_confusion_matrix(confusion_matrix, class_names, title=None, normalize=False, cmap=plt.cm.Blues,␣
 ↪figsize = (10,7), fontsize=14):
    # normalized or raw CM
    if normalize:
        confusion_matrix = confusion_matrix.astype('float') / confusion_matrix.sum(axis=1)[:, np.newaxis]
        fmt = '.2f'
    else:
        fmt = 'd'
    #----------------------------
    df_cm = pd.DataFrame(confusion_matrix, index=class_names, columns=class_names)
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt=fmt, cmap=cmap)
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    #----------------------------
    # fix matplotlib 3.1.1 bug
    #heatmap.get_ylim() --> (5.5, 0.5)
    #heatmap.set_ylim(6.0, 0)
    #------------------------------
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=fontsize)
    plt.title(title)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    return fig
```

1.3 2. Dataset

1.3.1 2.1 Download

[5]:
```
# download dataset
!wget -nc http://files.fast.ai/data/examples/imdb_sample.tgz
!tar -xzf imdb_sample.tgz
```

```
File ‘imdb_sample.tgz’ already there; not retrieving.
```

1.3.2 2.2 Dataset preparation

[0]:
```
# read csv spreadsheet
df = pd.read_csv('imdb_sample/texts.csv')
```

Input

[0]:
```
# getting only the 'text' column as a list
corpus = df['text'].tolist()
```

[0]:
```
# scikit-learn
vectorizer = CountVectorizer()
```

```
X_sparse = vectorizer.fit_transform(corpus)
vocab = vectorizer.get_feature_names()
```

[0]:
```
X = torch.tensor(X_sparse.toarray(), dtype=torch.float32)
```

Target

[0]:
```
# Pandas
target_pd = df['label']
# NumPy
target_np = target_pd.to_numpy().reshape(-1,1)
```

[0]:
```
# one-hot encoder
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
target_oh = enc.fit_transform(target_np)
```

[0]:
```
# PyTorch
target = torch.tensor(target_oh[:,1].toarray(), dtype=torch.int64)
target = torch.squeeze(target, 1)
```

1.3.3 2.3 PyTorch dataset creation

[0]:
```
# create dataset
ds = TensorDataset(X, target)
```

[0]:
```
# split in training, validation and test sets
ds_train, ds_val, ds_test = torch.utils.data.random_split(ds, (600, 200, 200))
```

1.3.4 2.4 PyTorch loader creation

- BATCH_SIZE definition
- training dataset

- validation dataset

```
[0]: BATCH_SIZE_TRAIN = 100
     BATCH_SIZE_VAL   = 100
     BATCH_SIZE_TEST  = 100
     #-------------------------------------------------
     # training data loader
     dl_train = torch.utils.data.DataLoader(
                   dataset=ds_train,
                   drop_last = False,
                   shuffle = False,
                   #batch_size = BATCH_SIZE)
                   batch_size = BATCH_SIZE_TRAIN)
     #-------------------------------------------------
     # validation data loader
     dl_val    = torch.utils.data.DataLoader(
                   dataset = ds_val,
                   drop_last = False,
                   shuffle = False,
                   #batch_size = BATCH_SIZE)
                   batch_size = BATCH_SIZE_VAL)
     #-------------------------------------------------
     # test data loader
     dl_test    = torch.utils.data.DataLoader(
                   dataset = ds_test,
                   drop_last = False,
                   shuffle = False,
                   #batch_size = BATCH_SIZE)
                   batch_size = BATCH_SIZE_TEST)
```

1.3.5 2.5 Verifying shape, batch data type from loader and optionally its visualization

[16]:
```
tx, ty = iter(dl_train).next()
print('train:', tx.shape, tx.dtype, ty.shape, ty.dtype)
tx, ty = iter(dl_val).next()
print('val:', tx.shape, tx.dtype, ty.shape, ty.dtype)
print('last batch size:', len(ds_train)%BATCH_SIZE_TRAIN, len(ds_val)%BATCH_SIZE_VAL)
```

```
train: torch.Size([100, 18668]) torch.float32 torch.Size([100]) torch.int64
val: torch.Size([100, 18668]) torch.float32 torch.Size([100]) torch.int64
last batch size: 0 0
```

1.4 3. Network Model

1.4.1 3.1 Network class definition

[0]:
```
'''
L1 (1st layer): 18668 inputs, 1000 outputs
L2 (2nd layer):  1000 inputs, 1000 outputs
L3 (3rd layer):  1000 inputs,    2 outputs
'''
class NN(torch.nn.Module):
    def __init__(self):
        super(NN, self).__init__()
        self.L1 = torch.nn.Linear(in_features=18668,  out_features=10000)
        self.L1D = torch.nn.Dropout(0.5)
        self.L2 = torch.nn.Linear(in_features=10000,  out_features=1000)
        self.L2D = torch.nn.Dropout(0.5)
        self.L3 = torch.nn.Linear(in_features=1000,  out_features=2)

    def forward(self, x):
        #-------------------
        # dense layer, ReLU, dropout
        x = self.L1(x)
        x = F.relu(x)
        x = self.L1D(x)
        #-------------------
        # dense layer, ReLU, dropout
        x = self.L2(x)
        x = F.relu(x)
        x = self.L2D(x)
        #-------------------
        # dense layer
        x = self.L3(x)
        #-------------------
        return x
```

1.4.2 3.2 Network instantiation

[18]:
```
model = NN()
model.to(device)
```

[18]:
```
NN(
  (L1): Linear(in_features=18668, out_features=10000, bias=True)
  (L1D): Dropout(p=0.5, inplace=False)
  (L2): Linear(in_features=10000, out_features=1000, bias=True)
  (L2D): Dropout(p=0.5, inplace=False)
  (L3): Linear(in_features=1000, out_features=2, bias=True)
)
```

1.4.3 3.3 Network predict with few samples of batch from loader

[19]:
```
model(ds_train[0][0].to(device))
```

[19]:
```
tensor([-0.1350,  0.0137], device='cuda:0', grad_fn=<AddBackward0>)
```

1.5 4. Network training

1.5.1 4.1 Training definitions

- number of epochs
- optimizer and LR (learning rate)
- loss function

[0]:
```
# Training parameters
EPOCH = 100
LR = 0.04
loss_func = torch.nn.CrossEntropyLoss()
opt = torch.optim.SGD(model.parameters(), lr=LR)
# loss history
loss_his = []
loss_val_his = []
```

1.5.2 4.2 Training loop

[21]:
```
N_SAMPLES = len(ds_train)
for epoch in range(EPOCH):
    print('epoch =', epoch, end='; ')
    #-------------------------------------------------
    # training mode
    #--------------------
    model.train()
    score_train = 0.
    for b_i, (b_x, b_y) in enumerate(dl_train):   # for each training step
        b_x, b_y = b_x.to(device), b_y.to(device)
        y_logitos = model(b_x)
        loss = loss_func(y_logitos, b_y)
        opt.zero_grad()                 # clear gradients for next train
        loss.backward()                 # backpropagation, compute gradients
        opt.step()                      # apply gradients
        #--------------------
        y_pred = torch.argmax(y_logitos, dim=1)
        score_train += (b_y == y_pred).sum()
    loss_his.append(loss.item())    # trainint loss history
    acc_train = score_train / N_SAMPLES
    #-------------------------------------------------
    # evaluation mode
    #-------------------
    model.eval()
    score_val = 0.
    for b_ival, (b_xval, b_yval) in enumerate(dl_val):
        b_xval, b_yval = b_xval.to(device), b_yval.to(device)
        y_logitos = model(b_xval)
        loss_val = loss_func(y_logitos, b_yval)
        yval_pred = torch.argmax(y_logitos, dim=1)
        score_val += (b_yval == yval_pred).sum()
    loss_val_his.append(loss_val.item())    # validation loss history
    acc_val = score_val / len(ds_val)
    #-------------------------------------------------
    print('loss_train = {0:.4f}'.format(loss_his[-1]), end='; ')
    print('loss_val = {0:.4f}'.format(loss_val_his[-1]), end='; ')
    print('acc_train = {0:.4f}'.format(acc_train), end='; ')
    print('acc_val = {0:.4f}'.format(acc_val), end='\n')
```

```
epoch = 0; loss_train = 0.6939; loss_val = 0.6904; acc_train = 0.4950; acc_val =
0.6550
epoch = 1; loss_train = 0.6844; loss_val = 0.6849; acc_train = 0.6183; acc_val =
0.6700
epoch = 2; loss_train = 0.6783; loss_val = 0.6782; acc_train = 0.6333; acc_val =
0.6600
epoch = 3; loss_train = 0.6659; loss_val = 0.6701; acc_train = 0.6483; acc_val =
0.7050
epoch = 4; loss_train = 0.6613; loss_val = 0.6602; acc_train = 0.6933; acc_val =
0.6950
epoch = 5; loss_train = 0.6444; loss_val = 0.6510; acc_train = 0.6733; acc_val =
0.7000
epoch = 6; loss_train = 0.6461; loss_val = 0.6403; acc_train = 0.6717; acc_val =
0.6950
epoch = 7; loss_train = 0.6317; loss_val = 0.6284; acc_train = 0.6800; acc_val =
0.6950
epoch = 8; loss_train = 0.6178; loss_val = 0.6193; acc_train = 0.6817; acc_val =
0.7050
epoch = 9; loss_train = 0.6033; loss_val = 0.6091; acc_train = 0.7033; acc_val =
0.7000
epoch = 10; loss_train = 0.5885; loss_val = 0.6004; acc_train = 0.6833; acc_val
= 0.7000
epoch = 11; loss_train = 0.5686; loss_val = 0.5933; acc_train = 0.6883; acc_val
= 0.7050
epoch = 12; loss_train = 0.5743; loss_val = 0.5885; acc_train = 0.7033; acc_val
= 0.6950
epoch = 13; loss_train = 0.5644; loss_val = 0.5797; acc_train = 0.6950; acc_val
= 0.7150
epoch = 14; loss_train = 0.5498; loss_val = 0.5766; acc_train = 0.6800; acc_val
= 0.7100
epoch = 15; loss_train = 0.5361; loss_val = 0.5672; acc_train = 0.7033; acc_val
= 0.7100
epoch = 16; loss_train = 0.5304; loss_val = 0.5628; acc_train = 0.7150; acc_val
= 0.7150
epoch = 17; loss_train = 0.5450; loss_val = 0.5651; acc_train = 0.7000; acc_val
= 0.7050
epoch = 18; loss_train = 0.5155; loss_val = 0.5546; acc_train = 0.7100; acc_val
= 0.7150
epoch = 19; loss_train = 0.5270; loss_val = 0.5598; acc_train = 0.7450; acc_val
= 0.6850
epoch = 20; loss_train = 0.4846; loss_val = 0.5441; acc_train = 0.7100; acc_val
= 0.7500
epoch = 21; loss_train = 0.4732; loss_val = 0.5349; acc_train = 0.7833; acc_val
= 0.7500
epoch = 22; loss_train = 0.5315; loss_val = 0.7814; acc_train = 0.7933; acc_val
```

```
= 0.6400
epoch = 23; loss_train = 0.4917; loss_val = 0.5550; acc_train = 0.6483; acc_val
= 0.7350
epoch = 24; loss_train = 0.4537; loss_val = 0.5665; acc_train = 0.8267; acc_val
= 0.7250
epoch = 25; loss_train = 0.4908; loss_val = 0.5669; acc_train = 0.7300; acc_val
= 0.7400
epoch = 26; loss_train = 0.4800; loss_val = 0.6617; acc_train = 0.8333; acc_val
= 0.6650
epoch = 27; loss_train = 0.4702; loss_val = 0.5353; acc_train = 0.6783; acc_val
= 0.7500
epoch = 28; loss_train = 0.4153; loss_val = 0.5032; acc_train = 0.8383; acc_val
= 0.7650
epoch = 29; loss_train = 0.4367; loss_val = 0.5126; acc_train = 0.7300; acc_val
= 0.7650
epoch = 30; loss_train = 0.3833; loss_val = 0.4923; acc_train = 0.8450; acc_val
= 0.7700
epoch = 31; loss_train = 0.8170; loss_val = 0.6209; acc_train = 0.7433; acc_val
= 0.5850
epoch = 32; loss_train = 0.4064; loss_val = 0.5152; acc_train = 0.7650; acc_val
= 0.7500
epoch = 33; loss_train = 0.4138; loss_val = 0.6147; acc_train = 0.8567; acc_val
= 0.6800
epoch = 34; loss_train = 0.4231; loss_val = 0.5218; acc_train = 0.7317; acc_val
= 0.7800
epoch = 35; loss_train = 0.3383; loss_val = 0.4850; acc_train = 0.8817; acc_val
= 0.7700
epoch = 36; loss_train = 0.3115; loss_val = 0.4865; acc_train = 0.9000; acc_val
= 0.7550
epoch = 37; loss_train = 0.4670; loss_val = 1.3271; acc_train = 0.8750; acc_val
= 0.5750
epoch = 38; loss_train = 0.3744; loss_val = 0.4943; acc_train = 0.7183; acc_val
= 0.7900
epoch = 39; loss_train = 0.4624; loss_val = 0.6894; acc_train = 0.8450; acc_val
```

```
= 0.5700
epoch = 40; loss_train = 0.3558; loss_val = 0.4864; acc_train = 0.7667; acc_val
= 0.7900
epoch = 41; loss_train = 0.3117; loss_val = 0.4919; acc_train = 0.9100; acc_val
= 0.7600
epoch = 42; loss_train = 0.5660; loss_val = 1.2405; acc_train = 0.8633; acc_val
= 0.5850
epoch = 43; loss_train = 0.3324; loss_val = 0.4794; acc_train = 0.7250; acc_val
= 0.7900
epoch = 44; loss_train = 0.2893; loss_val = 0.4762; acc_train = 0.9300; acc_val
= 0.7350
epoch = 45; loss_train = 0.3363; loss_val = 0.4678; acc_train = 0.7850; acc_val
= 0.7900
epoch = 46; loss_train = 0.2690; loss_val = 0.4692; acc_train = 0.9233; acc_val
= 0.7600
epoch = 47; loss_train = 0.2611; loss_val = 0.5817; acc_train = 0.9383; acc_val
= 0.7500
epoch = 48; loss_train = 0.3282; loss_val = 0.4966; acc_train = 0.7883; acc_val
= 0.7700
epoch = 49; loss_train = 0.2298; loss_val = 0.4309; acc_train = 0.9450; acc_val
= 0.7750
epoch = 50; loss_train = 0.6381; loss_val = 0.5165; acc_train = 0.7483; acc_val
= 0.6700
epoch = 51; loss_train = 0.2447; loss_val = 0.4429; acc_train = 0.8950; acc_val
= 0.7800
epoch = 52; loss_train = 0.2100; loss_val = 0.4571; acc_train = 0.9583; acc_val
= 0.7750
epoch = 53; loss_train = 0.1944; loss_val = 0.4790; acc_train = 0.9567; acc_val
= 0.7500
epoch = 54; loss_train = 0.2470; loss_val = 0.8884; acc_train = 0.9450; acc_val
= 0.6950
epoch = 55; loss_train = 0.2953; loss_val = 0.4598; acc_train = 0.7883; acc_val
= 0.7900
epoch = 56; loss_train = 0.2081; loss_val = 0.4335; acc_train = 0.9617; acc_val
```

```
= 0.7850
epoch = 57; loss_train = 0.1739; loss_val = 0.4104; acc_train = 0.9567; acc_val
= 0.7850
epoch = 58; loss_train = 0.4157; loss_val = 0.5332; acc_train = 0.7750; acc_val
= 0.6600
epoch = 59; loss_train = 0.2078; loss_val = 0.4322; acc_train = 0.8750; acc_val
= 0.7900
epoch = 60; loss_train = 0.2087; loss_val = 0.5513; acc_train = 0.9600; acc_val
= 0.7450
epoch = 61; loss_train = 0.7767; loss_val = 1.9133; acc_train = 0.8917; acc_val
= 0.5400
epoch = 62; loss_train = 0.1979; loss_val = 0.4443; acc_train = 0.8850; acc_val
= 0.7700
epoch = 63; loss_train = 0.1485; loss_val = 0.4255; acc_train = 0.9750; acc_val
= 0.7800
epoch = 64; loss_train = 0.1276; loss_val = 0.4480; acc_train = 0.9783; acc_val
= 0.7800
epoch = 65; loss_train = 0.1112; loss_val = 0.4249; acc_train = 0.9900; acc_val
= 0.7800
epoch = 66; loss_train = 0.1009; loss_val = 0.4353; acc_train = 0.9867; acc_val
= 0.7850
epoch = 67; loss_train = 0.0906; loss_val = 0.4595; acc_train = 0.9883; acc_val
= 0.7750
epoch = 68; loss_train = 0.0818; loss_val = 0.4601; acc_train = 0.9883; acc_val
= 0.7650
epoch = 69; loss_train = 0.0776; loss_val = 0.4349; acc_train = 0.9933; acc_val
= 0.7950
epoch = 70; loss_train = 0.0704; loss_val = 0.4269; acc_train = 0.9933; acc_val
= 0.7900
epoch = 71; loss_train = 0.0671; loss_val = 0.4809; acc_train = 0.9933; acc_val
= 0.7700
epoch = 72; loss_train = 0.0668; loss_val = 0.4573; acc_train = 0.9967; acc_val
= 0.7850
epoch = 73; loss_train = 0.0533; loss_val = 0.4388; acc_train = 0.9967; acc_val
```

```
= 0.7800
epoch = 74; loss_train = 0.0489; loss_val = 0.4115; acc_train = 0.9983; acc_val
= 0.7600
epoch = 75; loss_train = 0.0500; loss_val = 0.4455; acc_train = 0.9967; acc_val
= 0.7950
epoch = 76; loss_train = 0.0498; loss_val = 0.4686; acc_train = 1.0000; acc_val
= 0.7900
epoch = 77; loss_train = 0.0442; loss_val = 0.4311; acc_train = 0.9983; acc_val
= 0.7600
epoch = 78; loss_train = 0.0384; loss_val = 0.4318; acc_train = 1.0000; acc_val
= 0.7600
epoch = 79; loss_train = 0.0392; loss_val = 0.4448; acc_train = 1.0000; acc_val
= 0.7650
epoch = 80; loss_train = 0.0355; loss_val = 0.4389; acc_train = 1.0000; acc_val
= 0.7600
epoch = 81; loss_train = 0.0291; loss_val = 0.4312; acc_train = 1.0000; acc_val
= 0.7650
epoch = 82; loss_train = 0.0290; loss_val = 0.4426; acc_train = 1.0000; acc_val
= 0.7600
epoch = 83; loss_train = 0.0292; loss_val = 0.4673; acc_train = 1.0000; acc_val
= 0.7750
epoch = 84; loss_train = 0.0270; loss_val = 0.4495; acc_train = 1.0000; acc_val
= 0.7650
epoch = 85; loss_train = 0.0218; loss_val = 0.4508; acc_train = 1.0000; acc_val
= 0.7700
epoch = 86; loss_train = 0.0209; loss_val = 0.4506; acc_train = 1.0000; acc_val
= 0.7700
epoch = 87; loss_train = 0.0184; loss_val = 0.4459; acc_train = 1.0000; acc_val
= 0.7700
epoch = 88; loss_train = 0.0212; loss_val = 0.4509; acc_train = 1.0000; acc_val
= 0.7700
epoch = 89; loss_train = 0.0183; loss_val = 0.4505; acc_train = 1.0000; acc_val
= 0.7700
epoch = 90; loss_train = 0.0202; loss_val = 0.4634; acc_train = 1.0000; acc_val
```

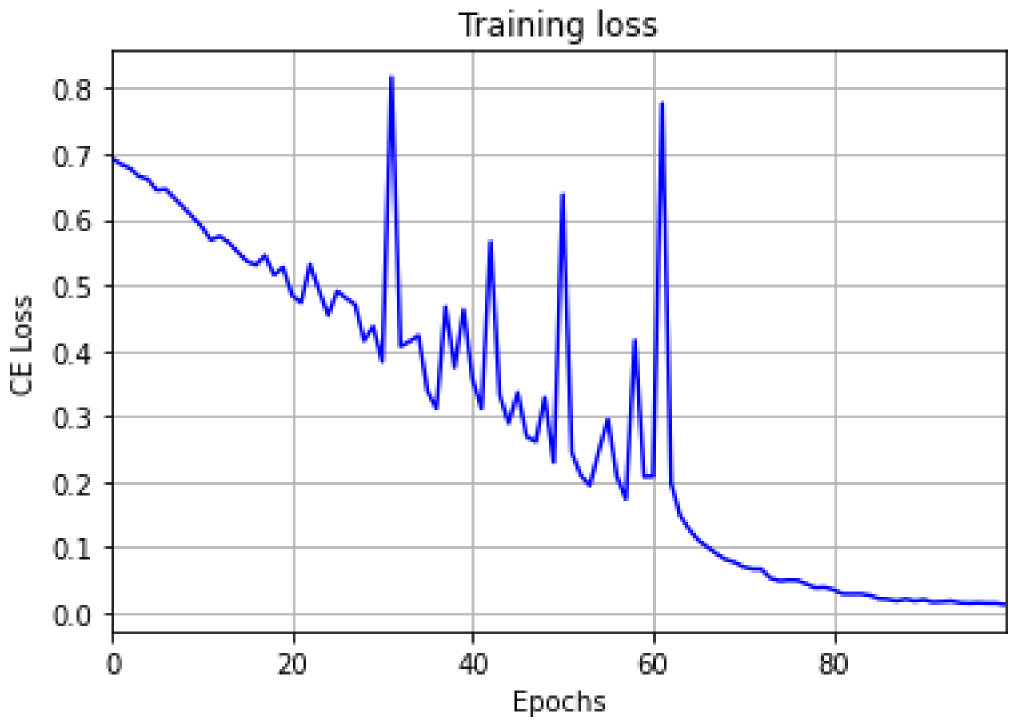
```
= 0.7800
epoch = 91; loss_train = 0.0166; loss_val = 0.4573; acc_train = 1.0000; acc_val
= 0.7700
epoch = 92; loss_train = 0.0171; loss_val = 0.4592; acc_train = 1.0000; acc_val
= 0.7750
epoch = 93; loss_train = 0.0183; loss_val = 0.4650; acc_train = 1.0000; acc_val
= 0.7700
epoch = 94; loss_train = 0.0156; loss_val = 0.4605; acc_train = 1.0000; acc_val
= 0.7650
epoch = 95; loss_train = 0.0148; loss_val = 0.4651; acc_train = 1.0000; acc_val
= 0.7750
epoch = 96; loss_train = 0.0155; loss_val = 0.4647; acc_train = 1.0000; acc_val
= 0.7550
epoch = 97; loss_train = 0.0149; loss_val = 0.4715; acc_train = 1.0000; acc_val
= 0.7700
epoch = 98; loss_train = 0.0149; loss_val = 0.4714; acc_train = 1.0000; acc_val
= 0.7750
epoch = 99; loss_train = 0.0127; loss_val = 0.4693; acc_train = 1.0000; acc_val
= 0.7700
```

1.6 5. Training evaluation

- metrics:
  - accuracy
  - confusion matrix
  - others

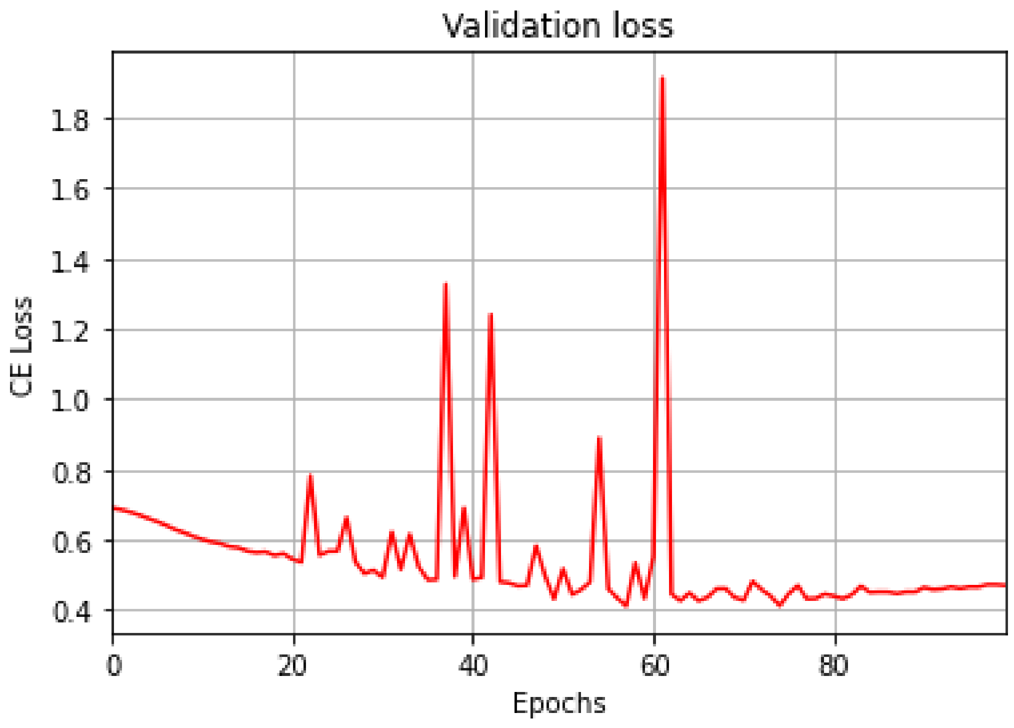
1.6.1 5.1 Training Loss

[22]:
```
# plot training loss
plt.plot(loss_his, label='training loss', color='blue')
#-----------------------------------------------
# axis label
plt.xlabel('Epochs')
plt.ylabel('CE Loss')
# title
plt.title('Training loss')
#--------------------------------------------------
plt.autoscale(axis='x', tight=True) # axis adjust
plt.grid(True) # add grid
#plt.legend() # add legend
plt.show()
```



1.6.2 5.2 Validation Loss

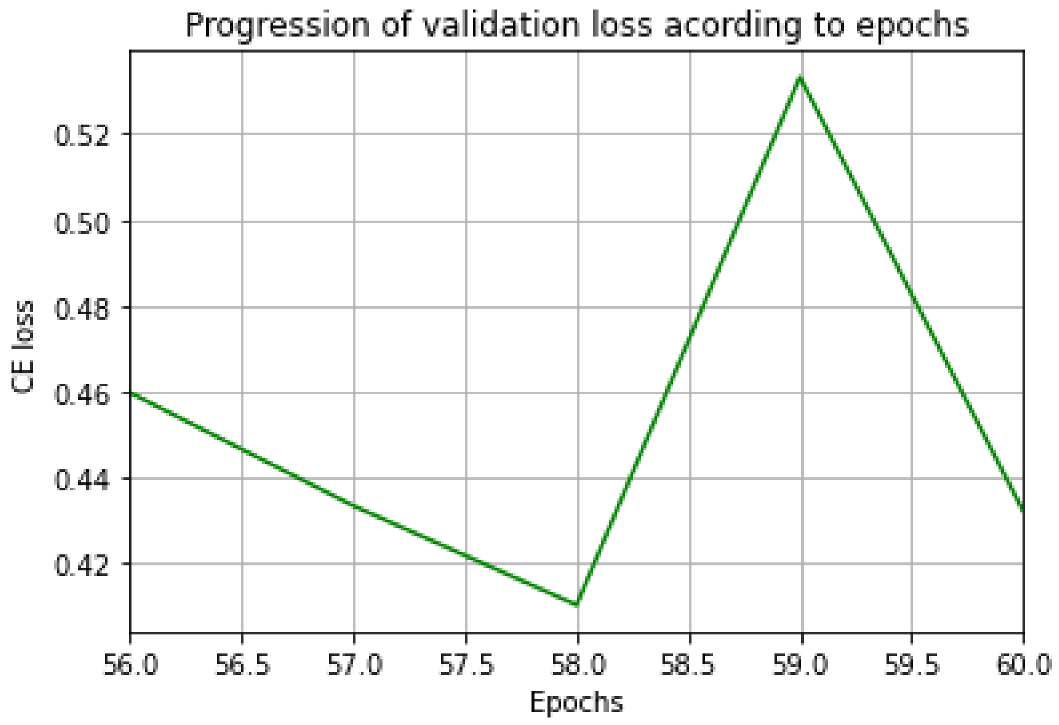
[23]:
```
# plot validation loss
plt.plot(loss_val_his, label='validation loss', color='red')
#-------------------------------------------------
# axis label
plt.xlabel('Epochs')
plt.ylabel('CE Loss')
# title
plt.title('Validation loss')
#-------------------------------------------------
plt.autoscale(axis='x', tight=True) # axis adjust
plt.grid(True) # add grid
#plt.legend() # add legend
plt.show()
```



1.6.3 5.3 Zoom at the minimum of CE loss curve

[24]:
```
BEST_EPOCH = loss_val_his.index(min(loss_val_his)) + 1
print('Epoch with minimum validation loss =', BEST_EPOCH)
#-------------------------------------------------
# plots
plt.plot(range(max(1,BEST_EPOCH-2),min(EPOCH,BEST_EPOCH+3)), loss_val_his[max(1,BEST_EPOCH-3):
 ↪min(EPOCH-1,BEST_EPOCH+2)], 'green')
plt.xlabel('Epochs')
plt.ylabel('CE loss')
plt.autoscale(axis='x', tight=True)
plt.title('Progression of validation loss acording to epochs')
plt.grid(True)
plt.show()
```

```
Epoch with minimum validation loss = 58
```



1.6.4 5.4 Print the final values of the main training monitoring variables:

- loss function final value:

- metrics final values:

[25]:
```
print('last training loss = {0:.4f}'.format(loss_his[-1]))
print('last validation loss = {0:.4f}'.format(loss_val_his[-1]))
#-------------------------------------------------
print('#-------------------------------------------------')
print('last train accuracy = {0:.4f}'.format(acc_train.item()))
print('last validation accuracy = {0:.4f}'.format(acc_val.item()))
```

```
last training loss = 0.0127
last validation loss = 0.4693
#-------------------------------------------------
last train accuracy = 1.0000
last validation accuracy = 0.7700
```

1.7 6. Final training

[26]:
```
print('Train until epoch:', BEST_EPOCH)
```

```
Train until epoch: 58
```

1.7.1 6.1 Merge training and validation datasets

[0]:
```
ds_final = ds_train + ds_val
```

1.7.2 6.2 PyTorch DataLoader

[0]:
```
dl_final  = torch.utils.data.DataLoader(
            dataset = ds_final,
            drop_last = False,
            shuffle = False,
            batch_size = BATCH_SIZE_TRAIN)
```

1.7.3 6.2 Early stopping

[29]:
```python
loss_final = []
N_SAMPLES = len(ds_train)
for epoch in range(BEST_EPOCH):
    print('epoch =', epoch, end='; ')
    #-------------------------------------------------
    # training mode
    #--------------------
    model.train()
    score_final = 0.
    for b_i, (b_x, b_y) in enumerate(dl_final):
        b_x, b_y = b_x.to(device), b_y.to(device)
        y_logitos = model(b_x)
        loss = loss_func(y_logitos, b_y)
        opt.zero_grad()                     # clear gradients for next train
        loss.backward()                     # backpropagation, compute gradients
        opt.step()                          # apply gradients
        #--------------------
        y_pred = torch.argmax(y_logitos, dim=1)
        score_final += (b_y == y_pred).sum()
    acc_final = score_final / N_SAMPLES
    loss_final.append(loss.item())              # trainint loss history
    acc_final = score_final / len(ds_final)
    #-------------------------------------------------
    print('loss_final = {0:.4f}'.format(loss_final[-1]), end='; ')
    print('acc_final = {0:.4f}'.format(acc_final), end='\n')
```

```
epoch = 0; loss_final = 0.5341; acc_final = 0.9387
epoch = 1; loss_final = 0.6176; acc_final = 0.7462
epoch = 2; loss_final = 0.3925; acc_final = 0.8450
epoch = 3; loss_final = 0.4924; acc_final = 0.8438
epoch = 4; loss_final = 0.3775; acc_final = 0.8413
epoch = 5; loss_final = 0.5827; acc_final = 0.8375
epoch = 6; loss_final = 0.3001; acc_final = 0.8850
epoch = 7; loss_final = 0.3618; acc_final = 0.8462
epoch = 8; loss_final = 0.2797; acc_final = 0.9688
epoch = 9; loss_final = 0.3688; acc_final = 0.8425
epoch = 10; loss_final = 0.5641; acc_final = 0.9450
epoch = 11; loss_final = 0.3811; acc_final = 0.8650
epoch = 12; loss_final = 0.4414; acc_final = 0.8462
epoch = 13; loss_final = 0.2466; acc_final = 0.9162
epoch = 14; loss_final = 0.1703; acc_final = 0.9862
epoch = 15; loss_final = 0.1548; acc_final = 0.9850
epoch = 16; loss_final = 0.2627; acc_final = 0.9800
epoch = 17; loss_final = 0.3193; acc_final = 0.8175
epoch = 18; loss_final = 0.2074; acc_final = 0.9712
epoch = 19; loss_final = 1.4817; acc_final = 0.9025
epoch = 20; loss_final = 0.1974; acc_final = 0.8900
epoch = 21; loss_final = 1.3049; acc_final = 0.9262
epoch = 22; loss_final = 0.1969; acc_final = 0.9262
epoch = 23; loss_final = 0.1512; acc_final = 0.9937
epoch = 24; loss_final = 3.0010; acc_final = 0.8737
epoch = 25; loss_final = 0.1852; acc_final = 0.9912
epoch = 26; loss_final = 0.1508; acc_final = 0.9925
epoch = 27; loss_final = 0.1415; acc_final = 0.9900
epoch = 28; loss_final = 0.1398; acc_final = 0.9900
epoch = 29; loss_final = 0.1995; acc_final = 0.8287
epoch = 30; loss_final = 0.1249; acc_final = 0.9912
epoch = 31; loss_final = 0.0930; acc_final = 0.9950
epoch = 32; loss_final = 0.0742; acc_final = 0.9975
epoch = 33; loss_final = 0.0680; acc_final = 0.9962
epoch = 34; loss_final = 0.0620; acc_final = 0.9975
epoch = 35; loss_final = 0.0546; acc_final = 0.9987
epoch = 36; loss_final = 0.0536; acc_final = 0.9987
epoch = 37; loss_final = 0.0435; acc_final = 0.9987
epoch = 38; loss_final = 0.0407; acc_final = 0.9987
epoch = 39; loss_final = 0.0416; acc_final = 0.9987
epoch = 40; loss_final = 0.0317; acc_final = 0.9987
epoch = 41; loss_final = 0.0318; acc_final = 1.0000
epoch = 42; loss_final = 0.0287; acc_final = 1.0000
epoch = 43; loss_final = 0.0270; acc_final = 1.0000
epoch = 44; loss_final = 0.0250; acc_final = 1.0000
epoch = 45; loss_final = 0.0301; acc_final = 1.0000
epoch = 46; loss_final = 0.0279; acc_final = 1.0000
epoch = 47; loss_final = 0.0218; acc_final = 1.0000
epoch = 48; loss_final = 0.0189; acc_final = 1.0000
epoch = 49; loss_final = 0.0165; acc_final = 1.0000
epoch = 50; loss_final = 0.0170; acc_final = 1.0000
epoch = 51; loss_final = 0.0163; acc_final = 1.0000
epoch = 52; loss_final = 0.0184; acc_final = 1.0000
epoch = 53; loss_final = 0.0177; acc_final = 1.0000
epoch = 54; loss_final = 0.0152; acc_final = 1.0000
epoch = 55; loss_final = 0.0141; acc_final = 1.0000
epoch = 56; loss_final = 0.0140; acc_final = 1.0000
epoch = 57; loss_final = 0.0148; acc_final = 1.0000
```

1.8 7. Metrics on test set

1.8.1 7.1 Accuracy

[30]:
```
# load model in CPU
model.to('cpu');
# evaluation mode
model.eval()
```

[30]:
```
NN(
  (L1): Linear(in_features=18668, out_features=10000, bias=True)
  (L1D): Dropout(p=0.5, inplace=False)
  (L2): Linear(in_features=10000, out_features=1000, bias=True)
  (L2D): Dropout(p=0.5, inplace=False)
  (L3): Linear(in_features=1000, out_features=2, bias=True)
)
```

[0]:
```
# y_true
y_true = ds_test[:][1]
```

[0]:
```
# y_pred
score = 0.
y_logitos = model(ds_test[:][0])
y_pred = torch.argmax(y_logitos, dim=1)
```

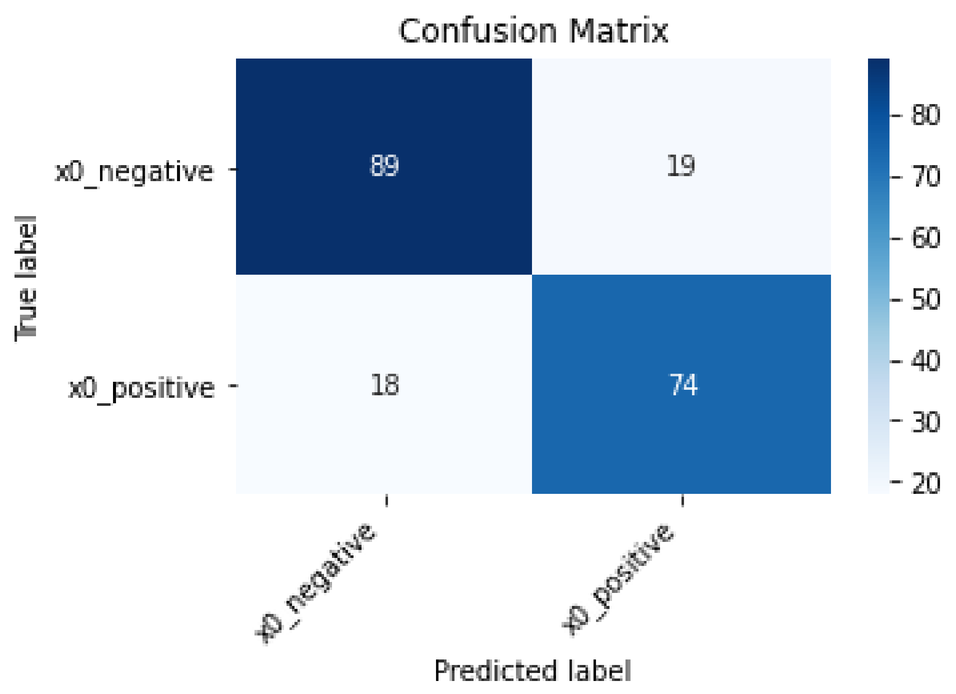
[33]:
```
# accuracy
score += (y_true == y_pred).sum()
acc_test = score / len(ds_test[:][0])
acc_test.item()
```

[33]:
```
0.8149999976158142
```

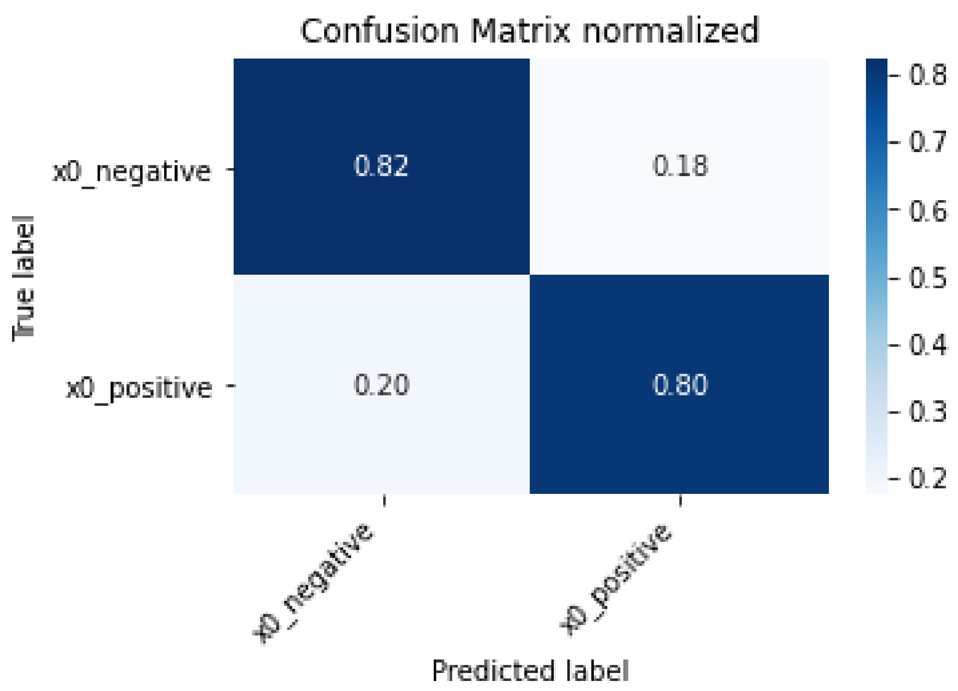
1.8.2 7.2 Confusion matrix

[0]:
```
cm = confusion_matrix(y_true, y_pred)
classes = enc.get_feature_names()
```

[35]:
```
# CM raw
cm_raw = print_confusion_matrix(cm, classes, title='Confusion Matrix', normalize=False, cmap=plt.cm.Blues,␣
 ↪fontsize=10, figsize = (5,3))
```



[36]:
```
# CM normalized
cm_norm = print_confusion_matrix(cm, classes, title='Confusion Matrix normalized', normalize=True, cmap=plt.cm.
 ↪Blues, fontsize=10, figsize = (5,3))
```



1.8.3 7.3 F1-score (macro, micro and weighted)

[0]:
```
macro = f1_score(y_true, y_pred, average='macro')
micro = f1_score(y_true, y_pred, average='micro')
weighted = f1_score(y_true, y_pred, average='weighted')
```

[38]:
```
print('F1-score macro =', macro)
print('F1-score micro =', micro)
print('F1-score weighted =', weighted)
```

```
F1-score macro = 0.813953488372093
```

```
F1-score micro = 0.815
F1-score weighted = 0.8150697674418604
```

1.8.4 7.4 Accuracy and Precision

```
[0]: acc = accuracy_score(y_true, y_pred)
     prec = precision_score(y_true, y_pred, average='macro')
```

```
[40]: print('Accuracy score = ', acc, sep='')
      print('Precision score = ', prec, sep='')
```

```
Accuracy score = 0.815
Precision score = 0.8137373128328811
```

1.8.5 7.5 Precision, Recall and F1-Score for each class

```
[41]: from sklearn.metrics import classification_report as cr
      print(cr(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.82      0.83       108
           1       0.80      0.80      0.80        92

    accuracy                           0.81       200
   macro avg       0.81      0.81      0.81       200
weighted avg       0.82      0.81      0.82       200
```

1.9 End of the notebook