DMT2023_HW4

June 8, 2023

0.1 Group composition:

————YOUR TEXT STARTS HERE————

Mignella, Laura, 1920520

Vestini, Maria Vittoria, 1795724

0.2 Homework 4

The homework consists of two parts:

1. Text Representation

and

2. Deep Learning

   Ensure that the notebook can be faithfully reproduced by anyone (hint: pseudo random number generation).

   If you need to set a random seed, set it to `709`.

   If multiple code cells are provided for a single part, it is **NOT** mandatory to use them all.

1 Part 1

In this part of the homework, you have to deal with Text Representation.

[ ]:
```
#REMOVE_OUTPUT#
!pip install --upgrade --no-cache-dir gdown
#YOUR CODE STARTS HERE#
!pip install -U sentence-transformers
import nltk
from sklearn.model_selection import train_test_split #to split train/test
import time
import pandas as pd
import numpy as np
from sentence_transformers import SentenceTransformer
from nltk.tokenize import RegexpTokenizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import matplotlib.pyplot as plt




#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

1.1 Part 1.1

The company **F**antastic **S**olution sells products. Customers can leave product reviews on their platform. The company wants to classify the reviews into positive and negative.

Their requirements are unclear: they mention both accuracy and calculation time, but it is not known which is more important to them. :'(

They also forbid you to do a hyper-parameter optimisation. (why? :O)

To help you (?), they have already pre-processed the data. They have translated each text into a random language.

The best thing to do is to provide them with a list of models that can best meet their (unclear) requirements.

1.1.1 1.1.1

Download the data from the Drive link (code already provided).

[ ]:
```
#REMOVE_OUTPUT#
!gdown 1X6QnCcOgnNEBQ1xnilmPWqDIs7bRrQof
```

1.1.2 1.1.2

Understand (!) and pre-process *(general term!)* the data. Divide the data according to your needs.

No specific request

[3]:
```
#YOUR CODE STARTS HERE#


# Load the data
data = pd.read_json('FS_reviews.jsonl', lines=True)

# Create a smaller DataFrame with only the informations we want to preserve
prep_data = pd.DataFrame()

# And that is:
# the text of the reviews
prep_data['review_text'] = data.review_text

# And the label that we will assign by looking at the ratings
# nb. there are no reviews with rating = 3
# so we will split the label in over and under 3
prep_data['label'] = data.rating.apply(lambda x: 1 if x>3 else 0)
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[4]:
```
#YOUR CODE STARTS HERE#


# PREPROCESSING

# To clean the text we will get rid of any punctuation
tokenizer = RegexpTokenizer(r'\w+')
prep_data['review'] = prep_data.review_text.apply(lambda x: tokenizer.
 ↪tokenize(x))
# And transform everything in lower case
prep_data['review'] = prep_data.review.apply(lambda x: ' '.join([term.lower()␣
 ↪for term in x]))


'''
The dataset is really unbalanced (80% positive reviews) so we get rid of
the positives reviews that are really short:
So if a positive review has less that 5 unique terms we drop it
'''

# Find the indexes of the rows to drop
idx = [i for i, review in enumerate(zip(prep_data.review, prep_data.label))␣
 ↪if(len(set(review[0].split()))<5 and review[1]==1)]
# And drop them
prep_data.drop(idx, axis = 0, inplace=True)
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[5]:
```
#YOUR CODE STARTS HERE#









#YOUR CODE ENDS HERE#
```

```
#THIS IS LINE 40#
```

———-YOUR TEXT STARTS HERE———-

For our preprocessing we decided only to remove punctuation and change everything to lowercase. We kept the preprocessing part as simple as we could since we're dealing with a multilanguage dataset and the processes like removing stopwords and lemmatization are more complex in this scenario.

1.1.3 1.1.3

Choose at least 1 and a maximum of 3 encodings. Encode the data.

P.S. If you need it, Word2Vec has a version for Documents

[6]:
```
#YOUR CODE STARTS HERE#

## FIRST EMBEDDING



start = time.time()

# Download the model
model = SentenceTransformer('sentence-transformers/
 ↪paraphrase-multilingual-MiniLM-L12-v2', device = 'cuda')

# Obtain the list of all the sentences
x = prep_data['review'].to_numpy()

# Obtain the wanted embeddings the sentences using the SentenceTransformer
embeddings_1 = model.encode(x, show_progress_bar=True)


# Store how much time it took
time_1 = [time.time() - start]









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
Downloading (…)0fe39/.gitattributes:   0%|          | 0.00/968 [00:00<?, ?B/s]
```

```
Downloading (…)_Pooling/config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]
Downloading (…)83e900fe39/README.md:   0%|          | 0.00/3.79k [00:00<?, ?B/s]
Downloading (…)e900fe39/config.json:   0%|          | 0.00/645 [00:00<?, ?B/s]
Downloading (…)ce_transformers.json:   0%|          | 0.00/122 [00:00<?, ?B/s]
Downloading pytorch_model.bin:   0%|          | 0.00/471M [00:00<?, ?B/s]
Downloading (…)nce_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]
Downloading (…)tencepiece.bpe.model:   0%|          | 0.00/5.07M [00:00<?, ?B/s]
Downloading (…)cial_tokens_map.json:   0%|          | 0.00/239 [00:00<?, ?B/s]
Downloading tokenizer.json:   0%|          | 0.00/9.08M [00:00<?, ?B/s]
Downloading (…)okenizer_config.json:   0%|          | 0.00/480 [00:00<?, ?B/s]
Downloading unigram.json:   0%|          | 0.00/14.8M [00:00<?, ?B/s]
Downloading (…)900fe39/modules.json:   0%|          | 0.00/229 [00:00<?, ?B/s]
Batches:   0%|          | 0/602 [00:00<?, ?it/s]
```

[7]:
```python
#YOUR CODE STARTS HERE#


## SECOND EMBEDDING

start = time.time()

# Download the model
model = SentenceTransformer('sentence-transformers/
 ↪paraphrase-multilingual-mpnet-base-v2', device = 'cuda')

# Obtain the list of all the sentences
x = prep_data['review'].to_numpy()

# Obtain the wanted embeddings the sentences using the SentenceTransformer
embeddings_2 = model.encode(x, show_progress_bar=True)


# Store how much time it took
time_2 = [time.time() - start]
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
Downloading (…)9e268/.gitattributes:   0%|          | 0.00/690 [00:00<?, ?B/s]
Downloading (…)_Pooling/config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]
Downloading (…)f2cd19e268/README.md:   0%|          | 0.00/3.77k [00:00<?, ?B/s]
Downloading (…)cd19e268/config.json:   0%|          | 0.00/723 [00:00<?, ?B/s]
Downloading (…)ce_transformers.json:   0%|          | 0.00/122 [00:00<?, ?B/s]
Downloading pytorch_model.bin:   0%|          | 0.00/1.11G [00:00<?, ?B/s]
Downloading (…)nce_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]
Downloading (…)tencepiece.bpe.model:   0%|          | 0.00/5.07M [00:00<?, ?B/s]
Downloading (…)cial_tokens_map.json:   0%|          | 0.00/239 [00:00<?, ?B/s]
Downloading (…)9e268/tokenizer.json:   0%|          | 0.00/9.08M [00:00<?, ?B/s]
Downloading (…)okenizer_config.json:   0%|          | 0.00/402 [00:00<?, ?B/s]
Downloading (…)d19e268/modules.json:   0%|          | 0.00/229 [00:00<?, ?B/s]
Batches:   0%|          | 0/602 [00:00<?, ?it/s]
```

[8]:
```
#YOUR CODE STARTS HERE#
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

———-YOUR TEXT STARTS HERE———-

Since we didn't translate the reviews we looked for multilanguages models. Out of all the models, given the unclear requirements, we selected the following: - 'paraphrase-multilingual-MiniLM-L12-v2' since it is suppose to be faster. - 'paraphrase-multilingual-mpnet-base-v2' since it is suppose to be the most accurate.

1.1.4 1.1.4

Choose **ONE** classifier for **EACH** encoding. Train the classifiers.

[9]:
```
#YOUR CODE STARTS HERE#

## FIRST CLASSIFIER



start = time.time()

# Split the data in train and test (80% and 20%)
train_x_1, test_x_1, train_y_1, test_y_1 = train_test_split(embeddings_1,
                                                            prep_data['label'],
                                                            test_size=0.2,
                                                            shuffle=True,
                                                            random_state = 709)



# Define the classifier
RandomForest = RandomForestClassifier(random_state=709)
# And train it
RandomForest.fit(train_x_1, train_y_1)


# Store the time it took to train the classifier
time_1.append(time.time() -start)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[10]:
```
#YOUR CODE STARTS HERE#
```

```
## SECOND CLASSIFIER



start = time.time()


# Split the data in train and test (80% and 20%)
train_x_2, test_x_2, train_y_2, test_y_2 = train_test_split(embeddings_2,
                                                    prep_data['label'],
                                                    test_size=0.2,
                                                    shuffle=True,
                                                    random_state = 709)

# Define the classifier
svc = SVC(random_state=709)
# And train it
svc.fit(train_x_2, train_y_2)



# Store the time it took to train the classifier
time_2.append(time.time() - start)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[11]:
```
#YOUR CODE STARTS HERE#
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

———-YOUR TEXT STARTS HERE———-

Given the task that we have to perform, we decided to use SVC and RandomForest as our classifiers.

The reason behind our choice is that these classifiers are known to perform well with high dimension features spaces, like the ones we work on during text classification.

1.1.5 1.1.5

Obtain the metrics you want to show the company.

[12]:
```
#YOUR CODE STARTS HERE#




# Get the predictions
y_pred_forest = RandomForest.predict(test_x_1)

# And find all the needed informations
TP = len([pred for i, pred in enumerate(y_pred_forest) if pred ==␣
 ↪list(test_y_1)[i] and pred == 1])
TN = len([pred for i, pred in enumerate(y_pred_forest) if pred ==␣
 ↪list(test_y_1)[i] and pred == 0])
FP = len([pred for i, pred in enumerate(y_pred_forest) if pred !=␣
 ↪list(test_y_1)[i] and pred == 0])
FN = len([pred for i, pred in enumerate(y_pred_forest) if pred !=␣
 ↪list(test_y_1)[i] and pred == 1])
Rec = TP/(TP+FN)
Pres = TP/(TP+FP)

acc_1 = (TP+TN)/len(test_y_1)
f1_1 = 2*Pres*Rec /(Rec+Pres)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[13]:
```
#YOUR CODE STARTS HERE#



# Get the predictions
y_pred_svc = svc.predict(test_x_2)

# And find all the needed informations
TP = len([pred for i, pred in enumerate(y_pred_svc) if pred ==␣
 ↪list(test_y_2)[i] and pred == 1])
TN = len([pred for i, pred in enumerate(y_pred_svc) if pred ==␣
 ↪list(test_y_2)[i] and pred == 0])
FP = len([pred for i, pred in enumerate(y_pred_svc) if pred !=␣
 ↪list(test_y_2)[i] and pred == 0])
FN = len([pred for i, pred in enumerate(y_pred_svc) if pred !=␣
 ↪list(test_y_2)[i] and pred == 1])
Rec = TP/(TP+FN)
Pres = TP/(TP+FP)


acc_2 = (TP+TN)/len(test_y_2)
f1_2 = 2*Pres*Rec /(Rec+Pres)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[14]:
```
#YOUR CODE STARTS HERE#









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

———-YOUR TEXT STARTS HERE———-

The metrics we computed are the accuracy and the F1-score.

The accuracy was one of the specific requests of the client while the F1-score is known to be a well balance metric. The F1-score takes into account both recall and precision, which taken individually can be uninformative.

1.1.6 1.1.6

Provide the company with all the information it needs to choose the pipeline it prefers.

[15]:
```
#YOUR CODE STARTS HERE#



print('Here are the important informations about the models:\n')
print('First Model: \n-Encoder: SenteceTransformer with model␣
 ↪"paraphrase-multilingual-MiniLM-L12-v2". \n-Classifier: RandomForest.')

print('\n * Accuracy:', acc_1, '\n * F1-score:', f1_1)
print (' * Time: \n\t time for the encoder:', time_1[0], ' \n\t time to fit the␣
 ↪model:', time_1[1], '\n\t total time:', np.sum(time_1))



print('\nSecond Model: \n-Encoder: SenteceTransformer with model␣
 ↪"paraphrase-multilingual-mpnet-base-v2". \n-Classifier: SVC.')

print('\n * Accuracy:', acc_2, '\n * F1-score:', f1_2)
print (' * Time: \n\t time for the encoder:', time_2[0], ' \n\t time to fit the␣
 ↪model:', time_2[1], '\n\t total time:', np.sum(time_2))









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
Here are the important informations about the models:

First Model:
-Encoder: SenteceTransformer with model "paraphrase-multilingual-MiniLM-L12-v2".
-Classifier: RandomForest.

 * Accuracy: 0.8678951466389826
 * F1-score: 0.9221831524231769
 * Time:
         time for the encoder: 60.41438627243042
         time to fit the model: 46.936644315719604
         total time: 107.35103058815002

Second Model:
-Encoder: SenteceTransformer with model "paraphrase-multilingual-mpnet-base-v2".
-Classifier: SVC.

 * Accuracy: 0.924993511549442
 * F1-score: 0.9535742971887551
 * Time:
         time for the encoder: 137.53804874420166
         time to fit the model: 47.86671566963196
         total time: 185.40476441383362
```

———-YOUR TEXT STARTS HERE———-

We have shown: the pipelines used for the models, the metrics we computed (accuracy and F1-score) and the time needed, divided in time for the encoder, tme to fit the model and total time.

Based on the requirments we belive that the company should be able to pick their classifier with these informations.

———-YOUR TEXT STARTS HERE———-

If the company priority is the computational time they should pick the first pipeline that use a simpler sentence transformer model.

Instead, if they want achieve the best results the choice should be the second since it has the best accuracy.

1.2 Part 1.2

1.2.1 1.2.1

Consider a scenario in which you have a set of words.

These must be transformed into a representation suitable for Machine Learning.

However, each representation has a fixed limit K.

Comment on how 3 word representations would behave in this scenario.

Use at most 3 sentences.

———-YOUR TEXT STARTS HERE———-

We will focus on the encoders seen during the lab.

- Word2Vec: for this method we can just set the number of features by setting the parameter 'vector_size'.
- OneHotEncoder: in this case each sentence will be represented by a vector of fix size equal to the number of unique terms of the vocabulary, so we can't set the limit K.
- SentenceTransformers: when initializing the model, the model attribute 'max_seq_length' is defined by defaut but it can be modify and set to K, $model.max\_seq\_length = K$.

2 Part 2

In this part of the homework, you have to deal with Deep Learning.

[ ]:
```
#REMOVE_OUTPUT#
#YOUR CODE STARTS HERE#
!pip install torchdata
import torch
torch.manual_seed(709)
from torch.utils.data import DataLoader
from torch.utils.data.dataset import random_split
import pandas as pd
import numpy as np
import time
from torchtext.vocab import build_vocab_from_iterator
from torchtext.data.functional import to_map_style_dataset









#YOUR CODE ENDS HERE#
#THIS IS LINE 20#
```

2.1 Part 2.1

You have to use the same data as in Part 1, but you can use whatever adjustments you have made to it (only Part 1.1.2).

2.1.1 2.1.1

Prepare the data structures you will need.

[17]:
```
#YOUR CODE STARTS HERE#

# we take the list of the sentences tokens
texts = [x.split() for x in prep_data['review']]

# split in train and test
train_x, test_x, train_y, test_y = train_test_split(texts,
                                                    prep_data['label'],
                                                    test_size=0.2,
                                                    shuffle=True,
                                                    random_state = 709)
```

```
# build the vocabulary with the train terms
vocab = build_vocab_from_iterator(train_x, specials=["<unk>"])
vocab.set_default_index(vocab["<unk>"])


# text processing pipeline
text_pipeline = lambda x: vocab(x)
label_pipeline = lambda x: int(x)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[18]:
```
#YOUR CODE STARTS HERE#

# select the device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


def collate_batch(batch):
    label_list, text_list, offsets = [], [], [0]
    for (_label, _text) in batch:
         label_list.append(label_pipeline(_label))
         processed_text = torch.tensor(text_pipeline(_text), dtype=torch.int64)
         text_list.append(processed_text)
         offsets.append(processed_text.size(0))
    label_list = torch.tensor(label_list, dtype=torch.int64)
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
    text_list = torch.cat(text_list)
    return label_list.to(device), text_list.to(device), offsets.to(device)

# creating the dataloader using the function defined above
```

```
train_iter = list(zip(train_y, train_x))
dataloader = DataLoader(train_iter, batch_size=8, shuffle=True,␣
 ↪collate_fn=collate_batch)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[19]:
```
#YOUR CODE STARTS HERE#
```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

———-YOUR TEXT STARTS HERE———-

In this part we created some relevant object that we will use for the rest of the code such as the vocabulary and the dataloder.

The vocabulary is used to map the terms into index while the dataloder is used to load the data in the device and to access the data in batches.

2.1.2 2.1.2

Define your model

[20]:
```
#YOUR CODE STARTS HERE#


# define the class of the model
class TextClassificationModel(torch.nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()

        #Computes sums or means of 'bags' of embeddings, without instantiating␣
 ↪the intermediate embeddings.
        self.embedding = torch.nn.EmbeddingBag(vocab_size, embed_dim,␣
 ↪sparse=False)

        self.fc = torch.nn.Linear(embed_dim, num_class)

        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        return self.fc(embedded)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[21]:
```
#YOUR CODE STARTS HERE#









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

———-YOUR TEXT STARTS HERE———-

The class used to define our model is composed by three functions. - ___init___: the constractor of the class in which we define the structure (embedding and layer) of the class. - init_weights: we used this function tu inizialize the weights of the methods. - forward: to compute the fit.

2.1.3 2.1.3

Train and optimize your model

[22]:
```
#YOUR CODE STARTS HERE#

# possible outcomes of the model
reviews_label = {0: "Negative",
                 1: "Positive"}

vocab_size = len(vocab)
emb_size = 64
# initializing the model
model = TextClassificationModel(vocab_size, emb_size, 2).to(device)

# function to train the model
def train(dataloader):
    model.train()
    total_acc, total_count = 0, 0
    log_interval = 500
    start_time = time.time()

    for idx, (label, text, offsets) in enumerate(dataloader):
        optimizer.zero_grad()
        predicted_label = model(text, offsets)
        # compute the loss
        loss = criterion(predicted_label, label)
        # backpropagation
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
        optimizer.step()
        total_acc += (predicted_label.argmax(1) == label).sum().item()
        total_count += label.size(0)
        if idx % log_interval == 0 and idx > 0:
            elapsed = time.time() - start_time
            print('| epoch {:3d} | {:5d}/{:5d} batches '
                  '| accuracy {:8.3f}'.format(epoch, idx, len(dataloader),
                                              total_acc/total_count))
            total_acc, total_count = 0, 0
            start_time = time.time()


#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[23]:
```
#YOUR CODE STARTS HERE#

# initialization of the parameters
```

```
test_iter = list(zip(test_y, test_x))


EPOCHS = 15 # epoch
LR = 5  # learning rate
BATCH_SIZE = 64 # batch size for training

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1.0, gamma=0.1)
total_accu = None


train_dataset = to_map_style_dataset(train_iter) #Convert iterable-style␣
 ↪dataset to map-style dataset.
test_dataset = to_map_style_dataset(test_iter) #Convert iterable-style dataset␣
 ↪to map-style dataset.
num_train = int(len(train_dataset) * 0.90)


split_train_, split_valid_ = \
    random_split(train_dataset, [num_train, len(train_dataset) - num_train])

train_dataloader = DataLoader(split_train_, batch_size=BATCH_SIZE,
                              shuffle=True, collate_fn=collate_batch)
valid_dataloader = DataLoader(split_valid_, batch_size=BATCH_SIZE,
                              shuffle=True, collate_fn=collate_batch)
test_dataloader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
                             shuffle=True, collate_fn=collate_batch)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

[24]:
```
#YOUR CODE STARTS HERE#

# function to evaluate the accuracy
def evaluate(dataloader):
    model.eval()
    total_acc, total_count = 0, 0
```

```
    with torch.no_grad():
        for idx, (label, text, offsets) in enumerate(dataloader):
            predicted_label = model(text, offsets)
            loss = criterion(predicted_label, label)
            total_acc += (predicted_label.argmax(1) == label).sum().item()
            total_count += label.size(0)
    return total_acc/total_count

# train the model and compute the accuracy for each epoch
for epoch in range(1, EPOCHS + 1):
    epoch_start_time = time.time()
    train(train_dataloader)
    accu_val = evaluate(valid_dataloader)
    if total_accu is not None and total_accu > accu_val:
      scheduler.step()
    else:
       total_accu = accu_val
    print('-' * 59)
    print('| end of epoch {:3d} | time: {:5.2f}s | '
          'valid accuracy {:8.3f} '.format(epoch,
                                           time.time() - epoch_start_time,
                                           accu_val))
    print('-' * 59)









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
-----------------------------------------------------------
| end of epoch   1 | time:  1.44s | valid accuracy    0.823
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   2 | time:  0.86s | valid accuracy    0.834
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   3 | time:  0.87s | valid accuracy    0.846
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   4 | time:  0.87s | valid accuracy    0.853
-----------------------------------------------------------
-----------------------------------------------------------
```

```
| end of epoch   5 | time:  0.86s | valid accuracy    0.855
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   6 | time:  0.86s | valid accuracy    0.856
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   7 | time:  0.93s | valid accuracy    0.862
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   8 | time:  1.18s | valid accuracy    0.861
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   9 | time:  1.16s | valid accuracy    0.872
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  10 | time:  1.27s | valid accuracy    0.870
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  11 | time:  1.17s | valid accuracy    0.871
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  12 | time:  1.05s | valid accuracy    0.871
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  13 | time:  0.88s | valid accuracy    0.871
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  14 | time:  0.88s | valid accuracy    0.871
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  15 | time:  0.88s | valid accuracy    0.871
-----------------------------------------------------------
```

———-YOUR TEXT STARTS HERE———-

We defined a function for the training of the model and one for the evaluation. In the train function we use our model to predict the labels, we compute the loss and then we perform the backpropagation to update the parameters. In the evaluation function we compute the accuracy on the validation set.

2.1.4 2.1.4

Show the performance of your model

[25]:
```
#YOUR CODE STARTS HERE#



print('Checking the results of test dataset.')
accu_test = evaluate(test_dataloader)
print('test accuracy {:8.3f}'.format(accu_test))









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
Checking the results of test dataset.
test accuracy    0.869
```

[26]:
```
#YOUR CODE STARTS HERE#


def predict(text, text_pipeline):
    with torch.no_grad():
        text = torch.tensor(text_pipeline(text))
        output = model(text, torch.tensor([0]))
        return output.argmax(1).item()




tokenizer = RegexpTokenizer(r'\w+')

ex_text_str = "E' normale che qui vengano a scrivere soprattutto clienti \
insoddisfatti; io penso però di rappresentare l'enorme numero di clienti \
contenti e silenziosi. Sono cliente Amazon da anni e per l'e-commerce non ha\
rivali. Ho acquistato articoli di ogni tipo, o perché introvabili dalle mie\
parti o perché di prezzo molto inferiore a quello dei negozi: ho ricevuti\
prodotti ottimi, prodotti medi e prodotti scadenti, com'è inevitabile,\
ma ho sempre ottenuto il rimborso quando l'ho richiesto. Il sistema delle \
recensioni è virtuoso perché spinge i venditori a venire incontro al cliente: \
personalmente mi affido molto alle recensioni e con questo sistema raramente␣
 ↪sbaglio. \
Per quanto poi riguarda la logistica, che è il vero terreno di Amazon, non c'è␣
 ↪gara: \
le spedizioni avvengono alla velocità della luce (mi capita di ordinare la sera␣
 ↪\
e ricevere il pacco la mattina), puntualissime, precisissime. Non ho ricordi \
di non aver ricevuto un pacco. Anche il reso funziona a meraviglia, posso␣
 ↪portare \
il pacco al tabaccaio sotto casa o anche lasciarlo in giardino e il corriere \
passa a ritirarlo. Circa l'abbonamento prime: ce l'ho ma fatico a dire se e \
quanto sia utile, dipende dall'uso che se ne fa -io ad esempio uso abbastanza \
prime video per i bambini-."

model = model.to("cpu")


print("This is a %s review." %reviews_label[predict(tokenizer.
 ↪tokenize(ex_text_str), text_pipeline)])


#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
This is a Positive review.
```

[27]:
```
#YOUR CODE STARTS HERE#









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

———-YOUR TEXT STARTS HERE———-

We compute the accuracy on the test set and to test the performance we try our model on an italian review we found online. The result of our 'experiment' was successful.

2.1.5 2.1.5

Provide an ablation study on at least one and at most three parameters.

[28]:
```
#YOUR CODE STARTS HERE#

# New parameter
emb_size = 384

# Define the model with the new parameter
model = TextClassificationModel(vocab_size, emb_size, 2).to(device)

# Initialize again
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1.0, gamma=0.1)
total_accu = None


# train the model and compute the accuracy for each epoch
for epoch in range(1, EPOCHS + 1):
    epoch_start_time = time.time()
    train(train_dataloader)
    accu_val = evaluate(valid_dataloader)
    if total_accu is not None and total_accu > accu_val:
      scheduler.step()
    else:
       total_accu = accu_val


# Compute the final accuracy on the test data
print('Checking the results for the first modified model.')
accu_test = evaluate(test_dataloader)
print('test accuracy {:8.3f}'.format(accu_test))

# Set the emb size back to the original value
emb_size = 64




#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
Checking the results for the first modified model.
test accuracy    0.796
```

[29]:
```
#YOUR CODE STARTS HERE#

# New parameter to change
EPOCHS = 10 # epoch

# Redefine the model
model = TextClassificationModel(vocab_size, emb_size, 2).to(device)


# Initialize
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, 1.0, gamma=0.1)
total_accu = None

# Train the model
for epoch in range(1, EPOCHS + 1):
    epoch_start_time = time.time()
    train(train_dataloader)
    accu_val = evaluate(valid_dataloader)
    if total_accu is not None and total_accu > accu_val:
      scheduler.step()
    else:
       total_accu = accu_val

# Compute the final accuracy on the test data
print('Checking the results ofthe second modified model.')
accu_test = evaluate(test_dataloader)
print('test accuracy {:8.3f}'.format(accu_test))









#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

```
Checking the results ofthe second modified model.
test accuracy    0.860
```

[86]:
```
#YOUR CODE STARTS HERE#


```

```
#YOUR CODE ENDS HERE#
#THIS IS LINE 40#
```

———-YOUR TEXT STARTS HERE———-

We tried to change two parameters: the embedding size and the number of epochs.

For the embedding size we increased it, and we saw that the accuracy has decrease.

Initially we noticed that by using 15 epochs the accuracy started to stabilize around the 10^{th} iteration, so we decided to decrease the number of epochs to 10 and again we saw a slightly worst result.

2.2 Part 2.2

2.2.1 2.2.1

How would a Deep Learning model (of the kind we have seen) behave in the case where a word was never seen during training? Answer on both practical and theoretical aspects.

Use at most 3 sentences.

———-YOUR TEXT STARTS HERE———-

Dividing in two possible scenarios, the first in which we are classifing a sentence with all unseen terms and the second in which there are only some unknown words.

In the first case, theoretically the model doesn't have enough information to perferm the classification, so what happens in practice is that the model will classify the sentence to a fixed class by default.

In the second case, instead, the method will be able to perform the classification by looking at the known terms.