DMT2023_HW4

June 10, 2023

0.1 Group composition:

————YOUR TEXT STARTS HERE————

Ben Kakitie, Joanna O., 1853548

Lauro, Francesco, 1706784

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#
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split #to split train/test
import json

import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.snowball import EnglishStemmer
from nltk import word_tokenize

!pip install -U sentence-transformers
from sentence_transformers import SentenceTransformer
from sentence_transformers.util import cos_sim #cosine similarity function

import warnings
warnings.filterwarnings(action = 'ignore') #to suppress gensim warnings

from sklearn.preprocessing import OneHotEncoder
from gensim.models import Word2Vec

# from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_extraction.text import TfidfVectorizer #to compute tf-idf
```

```
from sklearn.pipeline import Pipeline #pipeline
from sklearn.model_selection import GridSearchCV #to perform GridSearch
from sklearn import metrics #evaluate model

#YOUR CODE ENDS HERE#
#THIS IS LINE 36#
```

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#

# Read the jsonl file as a pandas DataFrame---------
#(from github repository: "https://github.com/maksimKorzh/one_line_scrapers/
 ↪blob/master/src/JSONL/jsonl.py")


# parse JSONL with indentations
def parse(filename):
    # list of items
    items = []

    # open JSONL file
```

```
    with open(filename, 'r') as f:
        # remove first and last two characters
        data = f.read()[1:-2]

        # extract JSON items
        data = data.split('}\n{')

        # loop over data items
        for item in data:
            # parse item to python dictionary type
            item_dict = json.loads('{' + item + '}')
            # append parsed item to item list
            items.append(item_dict)

        # return list of parsed items
        return items



data = parse('FS_reviews.jsonl') # list of dicts


dataset = pd.DataFrame(data)
dataset.head()


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

[3]:

	unique_id	product_name	product_type	helpful	rating	title
0	B0007NOY3E:resperate_device:reith_r._busby_"rb…	RESPeRATE Blood Pressure Lowering Device: Heal…	health & personal care	10 of 15	1.0	Resperate Device
1	B0007NOY3E:costs_too_much.:pmaclady	RESPeRATE Blood Pressure Lowering Device: Heal…	health & personal care	14 of 64	1.0	Costs too much.
2	B00006WNPY:don't_waste_your_money:l._reed_"mov…	Omron HJ-105 Pedometer with Calorie Counter: H…	health & personal care	2 of 2	1.0	Don't Waste Your Money
3	B00006WNPY:threw_away_$15.76:zoeeagleeye	Omron HJ-105 Pedometer with Calorie Counter: H…				
4	B0009XH7KO:tasty,_but…:catman	HOOAH! Energy Bars, Chocolate Crisp, 2.29-Ounc…				

```
3  health &amp;amp; personal care   5 of 17    1.0       Threw away $15.76
4  health &amp;amp; personal care  13 of 16    2.0         Tasty, But…

               date                                        review_text  \
0    August 6, 2006  Expensive…and after three months of daily us…
1    April 19, 2006  Meditation will do the same thing. Buy a tape …
2  November 5, 2006  When I got this pedometer, I found that the in…
3      May 14, 2006  The pedometer arrive held prisoner in a diffic…
4     June 16, 2006  I was offered  one of these while cycling and …

                 reviewer          reviewer_location
0  Reith R. Busby "rbuzz"                Kansas City
1                PMacLady                  Wisconsin
2    L. Reed "movie Buff"        Simi Valley, CA USA
3              Zoeeagleeye  Belfast, ME United States
4                   Catman                Oregon, USA
```

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


# Split the dataset between x and y

x = dataset["review_text"].to_numpy()
print(x[:10])   # x = string format

y = dataset["product_type"]
y = pd.get_dummies(y)      # get numeric format
y = y["health &amp;amp; personal care"].to_numpy()  # only cosider one of the two␣
 ↪columns
print(y)      # 1 represents a positive review, while 0 a negative one




# Split the x and y in train and test sets

train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.2,␣
 ↪shuffle=True, random_state = 709)
print("Training Set Size:",train_x.shape,train_y.shape)
print("Test Set Size:",test_x.shape,test_y.shape)




# Save the classes names

classes_names = dataset["product_type"].unique()
```

```
print("Classes:",classes_names)









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

```
["Expensive…and after three months of daily use I can't say it has helped my
blood pressure.\n\nI record my blood pressure every morning at the same
time…after getting up from nights sleep.\n\nI use exercise equipment for 45
minutes after stretching exercises.  Then breakfast and 15 minutes on Resperate
Devise.\n\nMy blood pressure hasn't changed significently in the last six months
and I continue to take blood pressure medicine.\n\nThe Resperate Device was used
for the first six weeks just before bedtime - change in schedule hasn't changed
blood pressure measurments to any significant extent.\n\nI continue to use the
device because of the fortune spent on it but wouldn't consider buying it again"
 'Meditation will do the same thing. Buy a tape or take a class'
 "When I got this pedometer, I found that the instructions weren't clear.  I
never did get the thing to work.  I have had pedometers before and they have
always been easy to operate.  This one wasn't.  The pedometer doesn't cost
enough to go through the hassle of returning it at your cost.  It's just easier
to buy a new one.  However, I suggest that you purchase it in person from a
sporting goods or health store"
 'The pedometer arrive held prisoner in a difficult-to-open plastic cell -- what
are these "packaging engineers" thinking? It took me ten minutes and two ruined
fingernails to open it, and then I took the scissors to it, and the hard
packaging almost broke them!\n\nBut I was thrilled.  A handsome object, thought
I, and I loved the clip in the back that you pinch to open.  So much better than
having to wrestle it down over a soft waistband.  I yanked out the plastic
battery protector and set to work to rev the thing up to speed.  But, alas, it
would not follow instructions as they were written.  "Press and hold the Set
button.  Hour display blinking."  (Could they not have said, "The hour display
will blink"?  I pressed.  "Press Memory button to adjust the hour."  I did.  The
Memory button never did a thing.  So I asked a friend of mine to try and he had
no luck either.  We both tried several combinations of maneuvers, but nothing
worked.  To this moment, the hour display is blinking, blinking,
blinking.\n\nI\'d like to get my money back, but ya know what?  J &amp;amp; R Music
&amp;amp; Computer World in NY demand such a rigamarole to send it back, including
with all the original packaging (I threw the OP in the trash), that you almost
have to be a genius in packaging to follow their instructions, which, of course,
are not designed to benefit the consumer but to benefit only themselves.  Now
```

I\'m afraid this pedometer will follow its packaging into the trash, while I will make sure I never order from J&R again.   '
 "I was offered one of these while cycling and thought it was very tasty. So good, in fact, that when I got home I went straight to the web site show on the back of the wrapper to learn where I could buy them. Just above the web address is the nutrition and ingredients list. That was probably a mistake on the manufacturer's part as it gives you something to read while the site comes up. OOPS! This bar has a whooping 25% of your daily allowance of saturated fat! Not only that, but the fat is palm oil.\n\nThis might be acceptable for a soldier involved in prolonged strenuous activity but I doubt a cardiologist would recomend it to most of us. A commercially formulated product is more likely to fill the average person's need for a lower fat (and healthier fat) energy bar, than one formulated for extreme physical activity, sold for profit and marketed as a patriotic tribute to the troops. Bottom line: Read the label, weigh the health risks and if you want to support the troops, there has to be a better way than eating Hooah Bars.\n"
 'The best thing I can say about the test is it took 13 days to receive my results.\n\nThe actual results varied greatly from those I received from my doctor 3 months prior:\n\n(…)\n\nBoth tests were taken following a 12-hour fast. All instructions were followed. I find it hard to believe my cholesterol levels changed so drastically in 3 months. I contacted BioSafe customer service.   All representatives were busy on my first call so I left a message, which was not returned. I called again and was able to speak to someone right away. The customer service person was very pleasant and took a lot of time explaining how accurate their tests were and how results from different testing methods (finger prick vs. blood drawn from arm) will vary "slightly".   (…) I question the accuracy of the biosafe results. \n\n '
 'This Oregon Scientific Pedometer is very inaccurate and a waste of money even at the cheap price. There is no point to even considering "saving money" when it comes to pedometers if the count is going to be almost arbitrary'
 'This pedometer is so much cheaper (~$7) than the typical decent ones (that go for about $18 - $20).. now I know it is for a reason.. the buttons are flimsy, the package looks very cheap, the one I bought and presented to a senior citizen - did not work, also found that the Sr Citizen found it very hard to use the buttons on this little device.. terrible ergonomics/form factor.\n\nWould recommend spending a bit more and getting the Omron/Oregon Sc $18 pedometers - much better form factor, high reliability and quality'
 'The monitor works really well. The functions are a bit limited from the perspective of calorie counting and fitness tracking. \n\nIf you are looking for a good basic monitor, this is a great unit.'
 "I don't know how this works because when I found that I would need to wear a chest band during exercise to make it work, I returned it. It isn't evident in the advert for this (or any other) heart-rate monitor. I guess I'll have to trust my sense of pain…"]
[1 1 1 … 0 0 0]
Training Set Size: (16168,) (16168,)
Test Set Size: (4042,) (4042,)
Classes: ['health & personal care' 'beauty']

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









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

———-YOUR TEXT STARTS HERE———-

After reading the jsonl in a pandas DataFrame, we were able to split the dataset between x and y. We used the `train_test_split` library to split the x and y in train and test sets. Lastly, we saved the classes names into a numpy.ndarray data structure.

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

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

# OneHotEncoder()


ohe_token_train_x = []

# iterate through each sentence in the file
for i in train_x:
    temp = [j.lower() for j in word_tokenize(i)]

    ohe_token_train_x.append(temp)

# print('First two tokens: ', ohe_token_train_x[:2])


# Build an ordered list containing all unique terms appearing in the corpus
all_tokens = sorted(list(set([word for sentence in ohe_token_train_x for word␣
 ↪in sentence])))
print('First 50 unique tokens: ', list(all_tokens)[:50])
print('Total number of tokens: ', len(all_tokens))


# Get the on_hot representation of the word
one_hot = OneHotEncoder()
one_hot_word_representation = one_hot.fit_transform([[x] for x in all_tokens])


# View the diagonal representation
print('Diagonal representation: ', one_hot_word_representation.todense(),␣
 ↪sep='\n')   # <-- not good because the size is too big

#View dimension
print(one_hot_word_representation.shape) #very big dim in respect to the number␣
 ↪of samples <-- bad, because in ML we want many samples, not a big dimension..
 ↪.
```

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

```
First 50 unique tokens:  ['!', '#', '$', '%', '&', "'", "''", "'03", "'4moms",
"'50", "'60", "'70", "'70s", "'80", "'90", "'aa", "'adventure",
"'adventure'-travel", "'aggeggi", "'air", "'al", "'all", "'apretón",
"'arrastren", "'attenta", "'attento", "'aventura", "'balanced", "'balsam",
"'banged", "'beautiful", "'best", "'bondad", "'brush", "'burner", "'burnt",
"'caldo", "'cartridges", "'change", "'cheat", "'chirriante", "'chocolovers",
"'chorro", "'clean", "'clings", "'close", "'cold", "'comb", "'concentrated",
"'concerned"]
Total number of tokens:  90042
Diagonal representation:
[[1. 0. 0. … 0. 0. 0.]
 [0. 1. 0. … 0. 0. 0.]
 [0. 0. 1. … 0. 0. 0.]
 …
 [0. 0. 0. … 1. 0. 0.]
 [0. 0. 0. … 0. 1. 0.]
 [0. 0. 0. … 0. 0. 1.]]
(90042, 90042)
```

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

# Word2vec ------------------


w2v_token_train_x = []

# iterate through each sentence in the file
for i in train_x:
    temp = [j.lower() for j in word_tokenize(i)]

    w2v_token_train_x.append(temp)


#Create CBOW (Continuous Bag Of Words) model <-- to understand the context of␣
 ↪the words and takes this as input
                                          #  <-- to predict words that are␣
 ↪contextually accurate
model1 = Word2Vec(w2v_token_train_x, min_count = 1, vector_size = 100, window =␣
 ↪3)


# Create Skip Gram model <--to learn the "word vectors" of the hidden layer
model2 = Word2Vec(w2v_token_train_x, min_count = 1, vector_size = 100, window =␣
 ↪3, sg = 1)
```

```
# model1.wv["great"]









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

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

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

———-YOUR TEXT STARTS HERE———-

We chose Word2vec, as it is a technique for natural language, which was useful for processing that learning word associations from the large corpus of text.

We tried considering also OneHotEncoder for the input words and measure the error rates with the one-hot encoded target word, but we could not use it efficently because the dimensions were too big, which is bad because Machine Learning requires many samples and not big dimensions, as it cannot provide any notion for similarity.

1.1.4 1.1.4

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

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

# Random Forest -----------

vectorizer = TfidfVectorizer(strip_accents = None, preprocessor = None)    #␣
 ↪Vectorizing function for x (in string format)


## Useful functions
stemmer, english_stopwords = EnglishStemmer(), set(stopwords.words('english'))

def stemming_tokenizer(text):
        stemmed_text = [stemmer.stem(word) for word in word_tokenize(text,␣
 ↪language='english')]
        return stemmed_text

def stemming_stop_tokenizer(text):
        stemmed_text = [stemmer.stem(word) for word in word_tokenize(text,␣
 ↪language='english')
                        ␣
 ↪                                                                    if␣
 ↪word not in english_stopwords]
        return stemmed_text



# Define the classifier
clf = RandomForestClassifier() #similar to decision trees --> get the majority␣
 ↪table

# Define the parameters values to test. ## Dictionary in which:
##                                          <-- Keys are parameters of objects␣
 ↪in the pipeline.
##                                          <-- Values are set of values to␣
 ↪try for a particular parameter.
parameters = {'vect__tokenizer': [None, stemming_tokenizer,␣
 ↪stemming_stop_tokenizer],
              'vect__ngram_range': [(1, 1), (1, 2), (1, 3)],
              'clf__min_samples_leaf': [1, 2, 5, 10],
              'clf__n_jobs': [-1]}
```

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

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

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

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









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

———-YOUR TEXT STARTS HERE———-

We chose the Random Forest Classification, because it can do predictions calculating the prediction for each decision tree, and then selecting the most prevalent result. The multiple decision trees are created using different random subsets of the data and features.

1.1.5 1.1.5

Obtain the metrics you want to show the company.

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



pipeline = Pipeline([     #  'Pipeline' is just a wrapper
('vect', vectorizer),     #  'vectorizer' is a model
('clf', clf),])


grid_search = GridSearchCV(pipeline, parameters,
                           scoring = metrics.make_scorer(metrics.
 ↪matthews_corrcoef),
                           cv = 5, n_jobs = -1, verbose = 10)

## Start an exhaustive search to find the best combination of parameters␣
 ↪according to the selected scoring-function.
grid_search.fit(train_x, train_y)









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

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

[10]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('vect', TfidfVectorizer()),
                                       ('clf', RandomForestClassifier())]),
             n_jobs=-1,
             param_grid={'clf__min_samples_leaf': [1, 2, 5, 10],
                         'clf__n_jobs': [-1],
                         'vect__ngram_range': [(1, 1), (1, 2), (1, 3)],
                         'vect__tokenizer': [None,
                                             <function stemming_tokenizer at
0x7f04410ad090>,
                                             <function stemming_stop_tokenizer
at 0x7f0439d5b2e0>]},
             scoring=make_scorer(matthews_corrcoef), verbose=10)
```

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


## Print results for each combination of parameters.
number_of_candidates = len(grid_search.cv_results_['params'])
print("Results:")

for i in range(number_of_candidates):
        print(i, 'params - %s; mean - %0.3f; std - %0.3f' %
                         (grid_search.cv_results_['params'][i],
                         grid_search.cv_results_['mean_test_score'][i],
                         grid_search.cv_results_['std_test_score'][i]))
```

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

```
Results:
0 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': None}; mean - 0.567; std - 0.011
1 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.547; std - 0.012
2 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.573; std - 0.010
3 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 2), 'vect__tokenizer': None}; mean - 0.535; std - 0.020
4 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 2), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.519; std - 0.016
5 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 2), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.549; std - 0.013
6 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': None}; mean - 0.516; std - 0.014
7 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.494; std - 0.013
8 params - {'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.519; std - 0.011
9 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': None}; mean - 0.546; std - 0.020
10 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.523; std - 0.012
11 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.560; std - 0.016
12 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 2), 'vect__tokenizer': None}; mean - 0.526; std - 0.018
13 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 2), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.506; std - 0.018
14 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
```

```
(1, 2), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.550; std - 0.007
15 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': None}; mean - 0.519; std - 0.019
16 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.492; std - 0.006
17 params - {'clf__min_samples_leaf': 2, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.535; std - 0.013
18 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': None}; mean - 0.488; std - 0.012
19 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.442; std - 0.016
20 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 1), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.494; std - 0.014
21 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 2), 'vect__tokenizer': None}; mean - 0.444; std - 0.015
22 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 2), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.409; std - 0.010
23 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 2), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.453; std - 0.010
24 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': None}; mean - 0.397; std - 0.011
25 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': <function stemming_tokenizer at 0x7f04410ad090>};
mean - 0.380; std - 0.014
26 params - {'clf__min_samples_leaf': 5, 'clf__n_jobs': -1, 'vect__ngram_range':
(1, 3), 'vect__tokenizer': <function stemming_stop_tokenizer at
0x7f0439d5b2e0>}; mean - 0.410; std - 0.014
27 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
'vect__ngram_range': (1, 1), 'vect__tokenizer': None}; mean - 0.398; std - 0.016
28 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
'vect__ngram_range': (1, 1), 'vect__tokenizer': <function stemming_tokenizer at
0x7f04410ad090>}; mean - 0.340; std - 0.024
29 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
'vect__ngram_range': (1, 1), 'vect__tokenizer': <function
stemming_stop_tokenizer at 0x7f0439d5b2e0>}; mean - 0.405; std - 0.015
30 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
'vect__ngram_range': (1, 2), 'vect__tokenizer': None}; mean - 0.305; std - 0.015
31 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
'vect__ngram_range': (1, 2), 'vect__tokenizer': <function stemming_tokenizer at
0x7f04410ad090>}; mean - 0.272; std - 0.014
32 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
```

```
'vect__ngram_range': (1, 2), 'vect__tokenizer': <function
stemming_stop_tokenizer at 0x7f0439d5b2e0>}; mean - 0.307; std - 0.019
33 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
'vect__ngram_range': (1, 3), 'vect__tokenizer': None}; mean - 0.207; std - 0.007
34 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
'vect__ngram_range': (1, 3), 'vect__tokenizer': <function stemming_tokenizer at
0x7f04410ad090>}; mean - 0.195; std - 0.019
35 params - {'clf__min_samples_leaf': 10, 'clf__n_jobs': -1,
'vect__ngram_range': (1, 3), 'vect__tokenizer': <function
stemming_stop_tokenizer at 0x7f0439d5b2e0>}; mean - 0.203; std - 0.015
```

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


print()
print("Best Estimator:")
print(grid_search.best_estimator_)
print()
print("Best Parameters:")
print(grid_search.best_params_)
print()
print("Used Scorer Function:")
print(grid_search.scorer_)
print()
print("Number of Folds:")
print(grid_search.n_splits_)
print()
```

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

```
Best Estimator:
Pipeline(steps=[('vect',
                 TfidfVectorizer(tokenizer=<function stemming_stop_tokenizer at
0x7f0439d5b2e0>)),
                ('clf', RandomForestClassifier(n_jobs=-1))])

Best Parameters:
{'clf__min_samples_leaf': 1, 'clf__n_jobs': -1, 'vect__ngram_range': (1, 1),
'vect__tokenizer': <function stemming_stop_tokenizer at 0x7f0439d5b2e0>}

Used Scorer Function:
make_scorer(matthews_corrcoef)

Number of Folds:
5
```

———-YOUR TEXT STARTS HERE———-

We chose to use the Matthews Correlation Coefficient (MCC) because of its capacities when it comes to measuring the quality of binary and multiclass classifications. We used MCC for the scoring function to evaluate the cross-validated model's performance on the test set.

1.1.6 1.1.6

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

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


# Evaluate the performance of the classifier on the original Test-Set
pred_test_y = grid_search.predict(test_x)


print("Test Metrics")
output_classification_report = metrics.classification_report(
                                                                test_y,
                                                                pred_test_y,
                                                                target_names=classes_r

print()
print("-----------------------------------------------------")
print(output_classification_report)
print("-----------------------------------------------------")
print()




# Compute the confusion matrix
confusion_matrix = metrics.confusion_matrix(test_y, pred_test_y)
print()
print("Confusion Matrix: True-Classes X Predicted-Classes")
print(pd.DataFrame(confusion_matrix))
print()









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

```
Test Metrics

-----------------------------------------------------
```

```
                           precision    recall  f1-score   support

health &amp;amp; personal care       0.88      0.57      0.69      1180
                   beauty       0.84      0.97      0.90      2862

                 accuracy                           0.85      4042
                macro avg       0.86      0.77      0.80      4042
             weighted avg       0.85      0.85      0.84      4042

--------------------------------------------------


Confusion Matrix: True-Classes X Predicted-Classes
     0     1
0  667   513
1   91  2771
```

———-YOUR TEXT STARTS HERE———-

The results obtained showed that: the Best Estimator is given by Pipeline(steps=[(‘vect’, TfidfVectorize(ngram_range=(1, 3))),(‘clf’, MultinomialNB())]); the Best Parameters are {’clf__alpha’: 1.0, ’vect__ngram_range’: (1, 3), ’vect__tokenizer’: None}; the Used Scorer Function is make_scorer(matthews_corrcoef); the Number of Folds is 5.

———-YOUR TEXT STARTS HERE———-

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———-

The word representations can be utilized directly for machine learning if their length is less than or equal to K. To fit the limit, they must be shortened if the length exceeds K. It's critical to select the representation's most essential components given that shortening can lead to information loss.

2 Part 2

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

[ ]:
```
#REMOVE_OUTPUT#
#YOUR CODE STARTS HERE#
import pandas as pd #to read/process data
import numpy as np #arrays
import time
import json
import torch
from torch.utils.data import DataLoader
from torch.utils.data.dataset import random_split

!pip install torchdata
from torchtext.datasets import AG_NEWS
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator
from torchtext.data.functional import to_map_style_dataset
from sklearn import metrics
from sklearn.model_selection import train_test_split
!pip install portalocker>=2.0.0
#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.

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

# dataset = pd.DataFrame(data)

# parse JSONL with indentations
def parse(filename):
    # list of items
    items = []

    # open JSONL file
    with open(filename, 'r') as f:
        # remove first and last two characters
        data = f.read()[1:-2]
```

```
        # extract JSON items
        data = data.split('}\n{')

        # loop over data items
        for item in data:
            # parse item to python dictionary type
            item_dict = json.loads('{' + item + '}')
            # append parsed item to item list
            items.append(item_dict)

        # return list of parsed items
        return items

data = parse('FS_reviews.jsonl') # list of dicts

dataset = pd.DataFrame(data)









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

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

# Split the dataset between x and y

x = dataset["review_text"].to_numpy()
print(x[:10])   # x = string format
y = dataset["product_type"]
y = pd.get_dummies(y)     # get numeric format
y = y["health &amp;amp; personal care"].to_numpy()  # only cosider one of the two␣
 ↪columns
print(y)      # 1 represents a positive review, while 0 a negative one


# Split the x and y in train and test sets

train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.2,␣
 ↪shuffle=True, random_state = 709)
print("Training Set Size:",train_x.shape,train_y.shape)
print("Test Set Size:",test_x.shape,test_y.shape)
```

```
# train_iter = iter(train_x)
train_y_x = list(zip(train_y,train_x))
train_iter = iter(train_y_x)

tokenizer = get_tokenizer('toktok')

def yield_tokens(data_iter):  #uses an iterator to build each token and process␣
 ↪each text
    for _, text in data_iter:
    # for text in data_iter:
        yield tokenizer(text)

vocab = build_vocab_from_iterator(yield_tokens(train_iter), specials=[""]) #we␣
 ↪build a vocab as a look-up table
vocab.set_default_index(vocab[""]) #"" == special token "unknown"

text_pipeline = lambda x: vocab(tokenizer(x))      # tokenize the text (to␣
 ↪become a list)
# label_pipeline = lambda x: int(x) - 1     # pass it to the vocabulary; '-1'␣
 ↪because the indeces start from 0
label_pipeline = lambda x: int(x)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")   #'cuda'␣
 ↪for checking if the GPU is available

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

```
["Expensive…and after three months of daily use I can't say it has helped my
blood pressure.\n\nI record my blood pressure every morning at the same
time…after getting up from nights sleep.\n\nI use exercise equipment for 45
minutes after stretching exercises.  Then breakfast and 15 minutes on Resperate
Devise.\n\nMy blood pressure hasn't changed significently in the last six months
and I continue to take blood pressure medicine.\n\nThe Resperate Device was used
for the first six weeks just before bedtime - change in schedule hasn't changed
blood pressure measurments to any significant extent.\n\nI continue to use the
device because of the fortune spent on it but wouldn't consider buying it again"
 'Meditation will do the same thing. Buy a tape or take a class'
 "When I got this pedometer, I found that the instructions weren't clear.  I
never did get the thing to work.  I have had pedometers before and they have
always been easy to operate.  This one wasn't.  The pedometer doesn't cost
enough to go through the hassle of returning it at your cost.  It's just easier
to buy a new one.  However, I suggest that you purchase it in person from a
sporting goods or health store"
 'The pedometer arrive held prisoner in a difficult-to-open plastic cell -- what
are these "packaging engineers" thinking? It took me ten minutes and two ruined
fingernails to open it, and then I took the scissors to it, and the hard
```

packaging almost broke them!\n\nBut I was thrilled. A handsome object, thought I, and I loved the clip in the back that you pinch to open. So much better than having to wrestle it down over a soft waistband. I yanked out the plastic battery protector and set to work to rev the thing up to speed. But, alas, it would not follow instructions as they were written. "Press and hold the Set button. Hour display blinking." (Could they not have said, "The hour display will blink"? I pressed. "Press Memory button to adjust the hour." I did. The Memory button never did a thing. So I asked a friend of mine to try and he had no luck either. We both tried several combinations of maneuvers, but nothing worked. To this moment, the hour display is blinking, blinking, blinking.\n\nI\'d like to get my money back, but ya know what? J & R Music & Computer World in NY demand such a rigamarole to send it back, including with all the original packaging (I threw the OP in the trash), that you almost have to be a genius in packaging to follow their instructions, which, of course, are not designed to benefit the consumer but to benefit only themselves. Now I\'m afraid this pedometer will follow its packaging into the trash, while I will make sure I never order from J&R again.   '

 "I was offered one of these while cycling and thought it was very tasty. So good, in fact, that when I got home I went straight to the web site show on the back of the wrapper to learn where I could buy them. Just above the web address is the nutrition and ingredients list. That was probably a mistake on the manufacturer's part as it gives you something to read while the site comes up. OOPS! This bar has a whooping 25% of your daily allowance of saturated fat! Not only that, but the fat is palm oil.\n\nThis might be acceptable for a soldier involved in prolonged strenuous activity but I doubt a cardiologist would recomend it to most of us. A commercially formulated product is more likely to fill the average person's need for a lower fat (and healthier fat) energy bar, than one formulated for extreme physical activity, sold for profit and marketed as a patriotic tribute to the troops. Bottom line: Read the label, weigh the health risks and if you want to support the troops, there has to be a better way than eating Hooah Bars.\n"

 'The best thing I can say about the test is it took 13 days to receive my results.\n\nThe actual results varied greatly from those I received from my doctor 3 months prior:\n\n(…)\n\nBoth tests were taken following a 12-hour fast. All instructions were followed. I find it hard to believe my cholesterol levels changed so drastically in 3 months. I contacted BioSafe customer service. All representatives were busy on my first call so I left a message, which was not returned. I called again and was able to speak to someone right away. The customer service person was very pleasant and took a lot of time explaining how accurate their tests were and how results from different testing methods (finger prick vs. blood drawn from arm) will vary "slightly". (…) I question the accuracy of the biosafe results. \n\n '

 'This Oregon Scientific Pedometer is very inaccurate and a waste of money even at the cheap price. There is no point to even considering "saving money" when it comes to pedometers if the count is going to be almost arbitrary'

 'This pedometer is so much cheaper (~$7) than the typical decent ones (that go for about $18 - $20).. now I know it is for a reason.. the buttons are flimsy, the package looks very cheap, the one I bought and presented to a senior citizen

```
- did not work, also found that the Sr Citizen found it very hard to use the
buttons on this little device.. terrible ergonomics/form factor.\n\nWould
recommend spending a bit more and getting the Omron/Oregon Sc  $18 pedometers -
much better form factor, high reliability and quality'
 'The monitor works really well. The functions are a bit limited from the
perspective of calorie counting and fitness tracking. \n\nIf you are looking for
a good basic monitor, this is a great unit.'
 "I don't know how this works because when I found that I would need to wear a
chest band during exercise to make it work, I returned it. It isn't evident in
the advert for this (or any other) heart-rate monitor. I guess I'll have to
trust my sense of pain…"]
[1 1 1 … 0 0 0]
Training Set Size: (16168,) (16168,)
Test Set Size: (4042,) (4042,)
```

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

# for (_label, _text) in train_iter:
#     print(_text)
#     app = torch.tensor(text_pipeline(_text), dtype=torch.int64)
#     print(app.size(0))
#     break

def collate_batch(batch):      # builds a data-loader --> divide dataset in␣
 ↪batches
    label_list, text_list, offsets = [], [], [0]
    for (_label, _text) in batch: #get label and text from each batch (ORDER IS␣
 ↪IMPORTANT!)
        #  print(_label)
        #  print(_text)
        #  break
         label_list.append(label_pipeline(_label)) #append label to list
         processed_text = torch.tensor(text_pipeline(_text), dtype=torch.int64)␣
 ↪ #use label pipeline --> create vect of int
         text_list.append(processed_text) # use text pipeline for the text (==␣
 ↪str) == input of our model
         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)

train_iter = train_y_x
dataloader = DataLoader(train_iter, batch_size=8, shuffle=False,␣
 ↪collate_fn=collate_batch)  #size != number of batches (= samples);
                                                                            ␣
 ↪            #shuffle is usually True for the training and False for the test
```

```
for label_list, text_list, offsets in dataloader:
    print(label_list.shape, text_list.shape, offsets.shape)
    break









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

```
torch.Size([8]) torch.Size([470]) torch.Size([8])
```

———-YOUR TEXT STARTS HERE———-

After reading the jsonl in a pandas DataFrame, we were able to split the dataset between x and y. We used the `train_test_split` library to split the x and y in train and test sets. Since we have to deal with multiple languages in our data, we used "toktok" tokenizer because is a simple, general tokenizer that works for multiple languages.

2.1.2 2.1.2

Define your model

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

class TextClassificationModel(torch.nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class, hidden_dim = 8):
        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.fc1 = torch.nn.Linear(embed_dim, hidden_dim)
        self.fc2 = torch.nn.Linear(hidden_dim, num_class)
        self.relu = torch.nn.ReLU()

        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc1.weight.data.uniform_(-initrange, initrange)
        self.fc1.bias.data.zero_()
        self.fc2.weight.data.uniform_(-initrange, initrange)
        self.fc2.bias.data.zero_()

    def forward(self, text, offsets):
        embedded = self.embedding(text, offsets)
        x = self.fc1(embedded)
        x = self.relu(x)
        x = self.fc2(x)
        return x









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

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

all_classes = set([label for (label, text) in train_iter])
print("Classes",all_classes)
num_class = len(all_classes)
print("Number classes",num_class)

labels =  {0: "beauty",
           1: "health & personal care"}

vocab_size = len(vocab)
emb_size = 64
model = TextClassificationModel(vocab_size, emb_size, num_class).to(device)









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

```
Classes {0, 1}
Number classes 2
```

———-YOUR TEXT STARTS HERE———-

After an initial nn.EmbeddingBag layer, our model is a neural network with two fully connected linear layers, interleaved by the ReLU. To determine the dimension of the output of the first linear

layer and of the input of the second linear layer, one can adjust the hidden_dim hyperparameter of the model. The column of the dataset that we considered as the one containing labels has two possible values and therefore the number of classes is two: 0: “beauty”, 1: “health & personal care”.

2.1.3 2.1.3

Train and optimize your model

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

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)
        loss = criterion(predicted_label, label)
        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()

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


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

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

# Hyperparameters
```

```
EPOCHS = 10 # epoch
LR = 5  # learning rate
BATCH_SIZE = 64 # batch size for training

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

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_x_2, val_x, train_y_2, val_y = train_test_split(train_x, train_y,␣
 ↪test_size=0.2, shuffle=True, random_state = 709)

split_train_ = list(zip(train_y_2,train_x_2))
split_valid_ = list(zip(val_y,val_x))

train_dataloader = DataLoader(split_train_, batch_size=BATCH_SIZE,
                              shuffle=False, collate_fn=collate_batch)
valid_dataloader = DataLoader(split_valid_, batch_size=BATCH_SIZE,
                              shuffle=False, collate_fn=collate_batch)
test_dataloader = DataLoader(test_iter, batch_size=BATCH_SIZE,
                             shuffle=False, collate_fn=collate_batch)









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

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

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:  6.55s | valid accuracy    0.725
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   2 | time:  5.55s | valid accuracy    0.726
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   3 | time:  5.77s | valid accuracy    0.744
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   4 | time:  2.91s | valid accuracy    0.753
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   5 | time:  3.83s | valid accuracy    0.775
```

```
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   6 | time:  3.00s | valid accuracy    0.777
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   7 | time:  2.94s | valid accuracy    0.784
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   8 | time:  2.91s | valid accuracy    0.777
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch   9 | time:  3.81s | valid accuracy    0.807
-----------------------------------------------------------
-----------------------------------------------------------
| end of epoch  10 | time:  2.96s | valid accuracy    0.809
-----------------------------------------------------------
```

———-YOUR TEXT STARTS HERE———-

With `sklearn.model_selection.train_test_split()` we split the training dataset into train/validation sets with a split ratio of 0.8 (train) and 0.2 (valid).

2.1.4 2.1.4

Show the performance of your model

[33]:
```
#YOUR CODE STARTS HERE#

# classes_names = ['0','1']
classes_names = ["beauty", "health &amp;amp; personal care"]

def evaluate_extended(dataloader):
    model.eval()
    y_true = torch.tensor([])
    y_pred = torch.tensor([])
    with torch.no_grad():
        for idx, (label, text, offsets) in enumerate(dataloader):
            predicted_label = model(text, offsets)
            y_true = torch.cat((y_true.cuda(), label))
            y_pred = torch.cat((y_pred.cuda(), predicted_label.argmax(1)))
    output_classification_report = metrics.classification_report(
                                                                    y_true.
 ↪cpu(),
                                                                    y_pred.
 ↪cpu(),
                                                                    target_names␣
 ↪= classes_names, output_dict = True)
    # print(output_classification_report)
    for i in output_classification_report:
        print(i,": ", output_classification_report[i])
    print("\nConfusion matrix:")
    df = pd.DataFrame(metrics.confusion_matrix(y_true.cpu(), y_pred.cpu(),␣
 ↪labels=np.arange(2)))
    df.rename(index= {0:'beauty', 1: "health &amp;amp; personal care"}, columns= {0:
 ↪'beauty', 1: "health &amp;amp; personal care"}, inplace=True)
    display(df)

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









#YOUR CODE ENDS HERE#
```

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

```
Checking the results of test dataset.

beauty :  {'precision': 0.7024048096192385, 'recall': 0.5940677966101695,
'f1-score': 0.6437098255280073, 'support': 1180}
health &amp;amp; personal care :  {'precision': 0.8426412614980289, 'recall':
0.8962264150943396, 'f1-score': 0.8686081950558754, 'support': 2862}
accuracy :  0.8080158337456704
macro avg :  {'precision': 0.7725230355586337, 'recall': 0.7451471058522545,
'f1-score': 0.7561590102919413, 'support': 4042}
weighted avg :  {'precision': 0.8017013769812122, 'recall': 0.8080158337456704,
'f1-score': 0.8029525602110252, 'support': 4042}

Confusion matrix:
```

	beauty	health & personal care
beauty	701	479
health & personal care	297	2565

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

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

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

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

———-YOUR TEXT STARTS HERE———-

For the test dataset, we showed:

- the accuracy of the model
- the confusion matrix
- the values of precision, recall and F1-score for each class, their aritmetic average and their weighted average, with the number of elements in each class as weights

2.1.5 2.1.5

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

[34]:
```
#YOUR CODE STARTS HERE#

# Ablation study 1: using only one fully connected linear layer instead of three

class TextClassificationModel2(torch.nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel2, self).__init__()

        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)

model = TextClassificationModel2(vocab_size, emb_size, num_class).to(device)

EPOCHS = 30 # epoch
LR = 5  # learning rate
BATCH_SIZE = 64 # batch size for training

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

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

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

[35]:
```
#YOUR CODE STARTS HERE#

```

```
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('\nChecking the results of test dataset.\n')
accu_test = evaluate_extended(test_dataloader)
# print('test accuracy {:8.3f}'.format(accu_test))









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

```
-----------------------------------------------------------
| end of epoch   1 | time:  2.95s | valid accuracy    0.725
-----------------------------------------------------------
| end of epoch   2 | time:  3.34s | valid accuracy    0.735
-----------------------------------------------------------
| end of epoch   3 | time:  2.90s | valid accuracy    0.746
-----------------------------------------------------------
| end of epoch   4 | time:  3.80s | valid accuracy    0.750
```

```
-----------------------------------------------------------
| end of epoch   5 | time:  2.84s | valid accuracy    0.757
-----------------------------------------------------------
| end of epoch   6 | time:  2.89s | valid accuracy    0.762
-----------------------------------------------------------
| end of epoch   7 | time:  2.89s | valid accuracy    0.749
-----------------------------------------------------------
| end of epoch   8 | time:  3.77s | valid accuracy    0.780
-----------------------------------------------------------
| end of epoch   9 | time:  2.87s | valid accuracy    0.781
-----------------------------------------------------------
| end of epoch  10 | time:  2.93s | valid accuracy    0.781
-----------------------------------------------------------
| end of epoch  11 | time:  2.95s | valid accuracy    0.783
-----------------------------------------------------------
| end of epoch  12 | time:  3.83s | valid accuracy    0.784
-----------------------------------------------------------
| end of epoch  13 | time:  2.90s | valid accuracy    0.784
-----------------------------------------------------------
| end of epoch  14 | time:  2.86s | valid accuracy    0.783
-----------------------------------------------------------
| end of epoch  15 | time:  2.86s | valid accuracy    0.786
-----------------------------------------------------------
| end of epoch  16 | time:  4.78s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  17 | time:  2.91s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  18 | time:  2.89s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  19 | time:  2.88s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  20 | time:  4.92s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  21 | time:  4.46s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  22 | time:  2.89s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  23 | time:  3.77s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  24 | time:  2.85s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  25 | time:  2.85s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  26 | time:  2.89s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  27 | time:  3.77s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  28 | time:  2.93s | valid accuracy    0.787
```

```
-----------------------------------------------------------
| end of epoch  29 | time:  2.96s | valid accuracy    0.787
-----------------------------------------------------------
| end of epoch  30 | time:  2.84s | valid accuracy    0.787

Checking the results of test dataset.

beauty :  {'precision': 0.7057546145494028, 'recall': 0.5508474576271186,
'f1-score': 0.6187529747739171, 'support': 1180}
health &amp;amp; personal care :  {'precision': 0.8301826337712271, 'recall':
0.9053109713487072, 'f1-score': 0.8661206752465318, 'support': 2862}
accuracy :  0.801830776843147
macro avg :  {'precision': 0.767968624160315, 'recall': 0.7280792144879129,
'f1-score': 0.7424368250102245, 'support': 4042}
weighted avg :  {'precision': 0.7938577790750982, 'recall': 0.801830776843147,
'f1-score': 0.7939054633322108, 'support': 4042}

Confusion matrix:
```

	beauty	health & personal care
beauty	650	530
health & personal care	271	2591

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

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

———-YOUR TEXT STARTS HERE———-

In order to perform an ablation study we used only one fully connected linear layer instead of the original three. There is no significant difference with respect to the orginal model in terms of precision, recall, accuracy and F1-score.

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———-

In practice, a Deep Learning model would not be able to predict the meaning of a word it has never come across before, as it lacks the necessary information to do so. Theoretically, the model would assign a random weight to the unknown word, which would not improve the overall prediction accuracy. Possible solutions to this issue are to either consider using pre-trained word embeddings or to increase the size of the training set.