

Sentiment Analysis - AraBERT(Using HARD Dataset) -

May 18, 2022

1 Sentiment Analysis - AraBERT(Using HARD Dataset) -

1. Dependencies installation

```
[ ]: import torch

# If there's a GPU available...
if torch.cuda.is_available():

    # Tell PyTorch to use the GPU.
    device = torch.device("cuda")

    print('There are %d GPU(s) available.' % torch.cuda.device_count())

    print('We will use the GPU:', torch.cuda.get_device_name(0))
    !nvidia-smi

# If not...
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

There are 1 GPU(s) available.

We will use the GPU: Tesla T4

Wed May 18 10:53:40 2022

```
+-----+
| NVIDIA-SMI 460.32.03      Driver Version: 460.32.03      CUDA Version: 11.2      |
+-----+-----+
| GPU  Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           MIG M. |
+=====+=====+=====+
|   0   Tesla T4              Off  | 00000000:00:04.0 Off  |             0      |
| N/A   48C    P8     10W /  70W |      3MiB / 15109MiB |           0%      Default |
|                                           N/A |
+-----+-----+-----+
```

Processes:							GPU Memory
GPU	GI	CI	PID	Type	Process name		Usage
	ID	ID					
=====							
No running processes found							

```
[ ]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
[ ]: !pip install transformers==4.12.2
!pip install farasapy==0.0.14
!pip install pyarabic==0.6.14
!git clone https://github.com/aub-mind/arabert
!pip install emoji==1.6.1
!pip install sentencepiece==0.1.96
```

```
Collecting transformers==4.12.2
  Downloading transformers-4.12.2-py3-none-any.whl (3.1 MB)
    |                               | 3.1 MB 5.1 MB/s
Collecting huggingface-hub>=0.0.17
  Downloading huggingface_hub-0.6.0-py3-none-any.whl (84 kB)
    |                               | 84 kB 3.9 MB/s
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.7/dist-packages (from transformers==4.12.2) (2019.12.20)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from transformers==4.12.2) (2.23.0)
Collecting tokenizers<0.11,>=0.10.1
  Downloading tokenizers-0.10.3-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_6
4.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.3 MB)
    |                               | 3.3 MB 46.9 MB/s
Collecting sacremoses
  Downloading sacremoses-0.0.53.tar.gz (880 kB)
    |                               | 880 kB 32.6 MB/s
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.7/dist-packages (from transformers==4.12.2) (21.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-
packages (from transformers==4.12.2) (1.21.6)
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.7/dist-packages (from transformers==4.12.2) (4.11.3)
Collecting pyyaml>=5.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manyl
inux_2_12_x86_64.manylinux2010_x86_64.whl (596 kB)
    |                               | 596 kB 55.4 MB/s
```

```

Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.7/dist-packages (from transformers==4.12.2) (4.64.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-
packages (from transformers==4.12.2) (3.7.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.7/dist-packages (from huggingface-
hub>=0.0.17->transformers==4.12.2) (4.2.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from
packaging>=20.0->transformers==4.12.2) (3.0.9)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata->transformers==4.12.2) (3.8.0)
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers==4.12.2)
(1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers==4.12.2)
(3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->transformers==4.12.2) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers==4.12.2)
(2021.10.8)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers==4.12.2) (1.15.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers==4.12.2) (7.1.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers==4.12.2) (1.1.0)
Building wheels for collected packages: sacremoses
  Building wheel for sacremoses (setup.py) ... done
  Created wheel for sacremoses: filename=sacremoses-0.0.53-py3-none-any.whl
size=895260
sha256=19d0de2be1bbf485abf1faa5ec97de0e9c42bd445750b2f7bbe7921aa28afb06
  Stored in directory: /root/.cache/pip/wheels/87/39/dd/a83eeef36d0bf98e7a4d1933
a4ad2d660295a40613079bafc9
Successfully built sacremoses
Installing collected packages: pyyaml, tokenizers, sacremoses, huggingface-hub,
transformers
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed huggingface-hub-0.6.0 pyyaml-6.0 sacremoses-0.0.53
tokenizers-0.10.3 transformers-4.12.2
Collecting farasapy==0.0.14
  Downloading farasapy-0.0.14-py3-none-any.whl (11 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages

```

```

(from farasapy==0.0.14) (4.64.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from farasapy==0.0.14) (2.23.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->farasapy==0.0.14)
(2021.10.8)
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->farasapy==0.0.14)
(1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->farasapy==0.0.14) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->farasapy==0.0.14) (2.10)
Installing collected packages: farasapy
Successfully installed farasapy-0.0.14
Collecting pyarabic==0.6.14
  Downloading PyArabic-0.6.14-py3-none-any.whl (126 kB)
    |                               | 126 kB 5.2 MB/s
Requirement already satisfied: six>=1.14.0 in
/usr/local/lib/python3.7/dist-packages (from pyarabic==0.6.14) (1.15.0)
Installing collected packages: pyarabic
Successfully installed pyarabic-0.6.14
Cloning into 'arabert'...
remote: Enumerating objects: 564, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 564 (delta 25), reused 22 (delta 22), pack-reused 535
Receiving objects: 100% (564/564), 9.11 MiB | 29.71 MiB/s, done.
Resolving deltas: 100% (326/326), done.
Collecting emoji==1.6.1
  Downloading emoji-1.6.1.tar.gz (170 kB)
    |                               | 170 kB 5.1 MB/s
Building wheels for collected packages: emoji
  Building wheel for emoji (setup.py) ... done
  Created wheel for emoji: filename=emoji-1.6.1-py3-none-any.whl size=169313
sha256=002276511855bf38e875008ad42e5fad5e857b6ce64ee767c1bf4ffbc8edee7
  Stored in directory: /root/.cache/pip/wheels/ea/5f/d3/03d313ddb3c2a1a427bb4690
f1621eea60fe6f2a30cc95940f
Successfully built emoji
Installing collected packages: emoji
Successfully installed emoji-1.6.1
Collecting sentencepiece==0.1.96
  Downloading
sentencepiece-0.1.96-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(1.2 MB)
    |                               | 1.2 MB 5.0 MB/s
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.96

```

```
[ ]: !git clone https://github.com/mohamedadaly/LABR
!git clone https://github.com/elnapara/HARD-Arabic-Dataset
!wget http://homepages.inf.ed.ac.uk/wmagdy/Resources/ArSAS.zip
!unzip ArSAS.zip
!unzip '/content/HARD-Arabic-Dataset/data/balanced-reviews.zip'
```

```
Cloning into 'LABR'...
remote: Enumerating objects: 37, done.
remote: Total 37 (delta 0), reused 0 (delta 0), pack-reused 37
Unpacking objects: 100% (37/37), done.
Cloning into 'HARD-Arabic-Dataset'...
remote: Enumerating objects: 100, done.
remote: Total 100 (delta 0), reused 0 (delta 0), pack-reused 100
Receiving objects: 100% (100/100), 116.36 MiB | 40.97 MiB/s, done.
Resolving deltas: 100% (35/35), done.
--2022-05-18 10:54:48--
http://homepages.inf.ed.ac.uk/wmagdy/Resources/ArSAS.zip
Resolving homepages.inf.ed.ac.uk (homepages.inf.ed.ac.uk)... 129.215.32.113
Connecting to homepages.inf.ed.ac.uk
(homepages.inf.ed.ac.uk)|129.215.32.113|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://homepages.inf.ed.ac.uk/wmagdy/Resources/ArSAS.zip [following]
--2022-05-18 10:54:48--
https://homepages.inf.ed.ac.uk/wmagdy/Resources/ArSAS.zip
Connecting to homepages.inf.ed.ac.uk
(homepages.inf.ed.ac.uk)|129.215.32.113|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1905723 (1.8M) [application/zip]
Saving to: 'ArSAS.zip'

ArSAS.zip          100%[=====>]    1.82M  2.65MB/s   in 0.7s

2022-05-18 10:54:49 (2.65 MB/s) - 'ArSAS.zip' saved [1905723/1905723]
```

```
Archive:  ArSAS.zip
  inflating: ArSAS..txt
Archive:  /content/HARD-Arabic-Dataset/data/balanced-reviews.zip
  inflating: balanced-reviews.txt
```

2. Datasets preparation

```
[ ]: import pandas as pd
import numpy as np
from typing import List
from tqdm import tqdm_notebook as tqdm
from sklearn.model_selection import train_test_split
```

```
[ ]: class CustomDataset:
    def __init__(
        self,
        name: str,
        train: List[pd.DataFrame],
        test: List[pd.DataFrame],
        label_list: List[str],
    ):
        self.name = name
        self.train = train
        self.test = test
        self.label_list = label_list
```

```
[ ]: all_datasets= []
DATA_COLUMN = "text"
LABEL_COLUMN = "label"
```

```
[ ]: df_HARD = pd.read_csv("/content/balanced-reviews.txt", sep="\t",
    ↳header=0,encoding='utf-16')

df_HARD = df_HARD[["review","rating"]] # we are interested in rating and
    ↳review only
df_HARD.columns = [DATA_COLUMN, LABEL_COLUMN]
print(df_HARD[LABEL_COLUMN].value_counts())
# code rating as +ve if > 3, -ve if less, no 3s in dataset

hard_map = {
    5: 'POS',
    4: 'POS',
    2: 'NEG',
    1: 'NEG'
}

df_HARD[LABEL_COLUMN] = df_HARD[LABEL_COLUMN].apply(lambda x: hard_map[x])
train_HARD, test_HARD = train_test_split(df_HARD, test_size=0.2,
    ↳random_state=42)
label_list_HARD = ['NEG', 'POS']

data_Hard = CustomDataset("HARD", train_HARD, test_HARD, label_list_HARD)
all_datasets.append(data_Hard)
```

```
2    38467
4    26450
5    26399
1    14382
Name: label, dtype: int64
```

```
[ ]: #@title
%%writefile labr.py
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 16:27:03 2013

@author: Mohamed Aly <mohamed@mohamedaly.info>
"""

import codecs
import numpy as np
import pandas as pd
import re

class LABR:
    def __init__(self):
        self.REVIEWS_PATH = "LABR/data/"
        self.RAW_REVIEWS_FILE = "raw_reviews.tsv"
        self.DELETED_REVIEWS_FILE = "deleted_reviews.tsv"
        self.CLEAN_REVIEWS_FILE = "reviews.tsv"

    # Copied from the PyArabic package.
    def arabicrange(self):
        """return a list of arabic characteres .
        Return a list of characteres between \u060c to \u0652
        @return: list of arabic characteres.
        @rtype: unicode;
        """
        mylist=[];
        for i in range(0x0600, 0x0653):
            try :
                mylist.append(unichr(i));
            except ValueError:
                pass;
        return mylist;

    # cleans a single review
    def clean_raw_review(self, body):
        # patterns to remove first
        pat = [\
            (u'http[s]?://[a-zA-Z0-9_\-.~/~?=%&]+' , u' '), # ↵
            ↪remove links
            (u'www[a-zA-Z0-9_\-?=%&/.\~]+' , u' '),
            # u'\n+': u' ', # remove newlines
            (u'<br />' , u' '), # remove html line breaks
            (u'</?[^\>]+>' , u' '), # remove html markup
            # u'http': u',

```

```

        (u'[a-zA-Z]+\\.org', u''),
        (u'[a-zA-Z]+\\.com', u''),
        (u'://', u''),
        (u'&[~;]+;', u''),
        (u':D', u':)'),
#         (u'[0-9/]+', u''),
#         u'[a-zA-Z.]+' : u'',
#         u'[^0-9' + u''.join(self.arabicrange()) + \
#             u"!.,;:;%&*%'#(){}~`\[\\/\|\\\\\\\"\" + \
#             u'\\s\\^><\\_\\u201D\\u00AB=\\u2026]+' : u'',          # remove latin
→ characters
        (u'\\s+', u''),          # remove spaces
        (u'\\.+', u'.'),          # multiple dots
        (u' [\\u201C\\u201D]', u''),          # "
        (u' [\\u2665\\u2764]', u''),          # heart symbol
        (u' [\\u00BB\\u00AB]', u''),
        (u'\\u2013', u'-'),          # dash
    ]

    # patterns that disqualify a review
    remove_if_there = [\\
        (u'[^0-9' + u''.join(self.arabicrange()) + \
            u"!.,;:;%&*%'#(){}~`\[\\/\|\\\\\\\"\" + \
            u'\\s\\^><\\_\\u201D\\u00AB=\\u2026+|' + \
            u'\\u0660-\\u066D\\u201C\\u201D' + \
            u'\\ufefb\\ufef7\\ufef5\\ufef9]+' , u''),          # non
→ arabic characters
    ]

    # patterns that disqualify if empty after removing
    remove_if_empty_after = [\\
        (u'[0-9a-zA-Z\\_\\_]', u''),          # alpha-numeric
        (u'[^0-9' + u"!.,;:;%&*%'#(){}~`\[\\/\|\\\\\\\"\" + \
            u'\\s\\^><\\_\\_+=]+' , u''),          # remove just
→ punctuation
        (u'\\s+', u''),          # remove spaces
    ]

    # remove again
    # patterns to remove
    pat2 = [\\
#         u'[^0-9' + u''.join(self.arabicrange()) + \
#         u"!.,;:;%&*%'#(){}~`\[\\/\|\\\\\\\"\" + \
#         u'\\s\\^><\\_\\u201D\\u00AB=\\u2026]+' : u'',          # remove latin
→ characters
    ]

```



```

skip = False

# if empty body, skip
if body == u'': skip = True

# do some substitutions
for k,v in pat:
    body = re.sub(k, v, body)

# remove if exist
for k,v in remove_if_there:
    if re.search(k, body):
        skip = True

# remove if empty after replacing
for k,v in remove_if_empty_after:
    temp = re.sub(k, v, body)
    if temp == u" " or temp == u"":
        skip = True

# do some more substitutions
if not skip:
    for k,v in pat2:
        body = re.sub(k, v, body)

# if empty string, skip
if body == u'' or body == u' ':
    skip = True

if not skip:
    return body
else:
    return u""

# Read raw reviews from file and clean and write into clean_reviews
def clean_raw_reviews(self):
    # input file
    in_file = codecs.open(self.REVIEWS_PATH + self.RAW_REVIEWS_FILE,
                          'r', encoding="utf-8")
    reviews = in_file.readlines()

    # Output file: rating<tab>content
    out_file = open(self.REVIEWS_PATH + self.CLEAN_REVIEWS_FILE,
                   'w', buffering = 100)
    deleted_file = open(self.REVIEWS_PATH + self.DELETED_REVIEWS_FILE,
                      'w', buffering = 100)

```

```

counter = 1
for i in xrange(0, len(reviews)):
    review = reviews[i]
    skip = False

#         # If line starts with #, then skip
#         if review[0] == u"#": continue

    # split by <tab>
    parts = review.split(u"\t")

    # rating is first part and body is last part
    rating = parts[0]
    review_id = parts[1]
    user_id = parts[2]
    book_id = parts[3]
    body = parts[4].strip()

    # clean body
    body = self.clean_raw_review(body)
    if body == u"": skip = True

    if i % 5000 == 0:
        print("review %d:" % (i))

    # write output
    line = u"%s\t%s\t%s\t%s\t%s\n" % (rating, review_id, user_id,
                                     book_id, body)

    if not skip:
        out_file.write(line.encode('utf-8'))
        counter += 1
    else:
        deleted_file.write(line.encode('utf-8'))

# Read the reviews file. Returns a tuple containing these lists:
#   rating: the rating 1 -> 5
#   review_id: the id of the review
#   user_id: the id of the user
#   book_id: the id of the book
#   body: the text of the review
def read_review_file(self, file_name):
    reviews = codecs.open(file_name, 'r', 'utf-8').readlines()

    # remove comment lines and newlines
    reviews = [r.strip() for r in reviews if r[0] != u'#']

    # parse

```

```

rating = list()
review_id = list()
user_id = list()
book_id = list()
body = list()
for review in reviews:
    # split by <tab>
    parts = review.split(u"\t")

    # rating is first part and body is last part
    rating.append(int(parts[0]))
    review_id.append(parts[1])
    user_id.append(parts[2])
    book_id.append(parts[3])
    if len(parts) > 4:
        body.append(parts[4])
    else:
        body.append(u"")

    return (rating, review_id, user_id, book_id, body)

# Writes reviews to a file
def write_review_file(self, file_name, rating, review_id, user_id,
                      book_id, body):

    lines = list()
    # loop
    for i in xrange(len(rating)):
        line = u"%s\t%s\t%s\t%s\t%s\n" % (rating[i], review_id[i],
                                          user_id[i], book_id[i],
                                          body[i])

        lines.append(line)

    open(file_name, 'w').write(u''.join(lines).encode('utf-8'))

def read_clean_reviews(self):
    return self.read_review_file(self.REVIEWS_PATH +
                                self.CLEAN_REVIEWS_FILE)

def read_raw_reviews(self):
    return self.read_review_file(self.REVIEWS_PATH + self.RAW_REVIEWS_FILE)

# Splits the dataset into a training/test sets in the setting of using 5
# classes (predicting the rating value from 1 to 5)
def split_train_test_5class(self, rating, percent_test,
                             balanced = "unbalanced"):
    np.random.seed(1234)

```

```

num_reviews = len(rating)
review_ids = np.arange(0, num_reviews)

if balanced == "unbalanced":
    ntest = np.floor(num_reviews * percent_test)
    np.random.shuffle(review_ids)

    test_ids = review_ids[:ntest]
    train_ids = review_ids[ntest:]

elif balanced == "balanced":
    (sizes, bins) = np.histogram(rating, [1, 2, 3, 4, 5, 6])
    min_size = np.min(sizes)
    print(min_size)

    # sample review ids equally among classes
    test_ids = np.zeros((0,), dtype="int32")
    train_ids = np.zeros((0,), dtype="int32")
    rating = np.array(rating)
    ntest = np.floor(min_size * percent_test)
    for c in range(1, 6):
        cids = review_ids[np.nonzero(rating == c)]
        np.random.shuffle(cids)

        test_ids = np.r_[test_ids, cids[:ntest]]
        train_ids = np.r_[train_ids, cids[ntest:min_size]]

train_file = self.REVIEWS_PATH + "5class-" + balanced+ "-train.txt"
test_file = self.REVIEWS_PATH + "5class-" + balanced+ "-test.txt"

open(train_file, 'w').write('\n'.join(map(str, train_ids)))
open(test_file, 'w').write('\n'.join(map(str, test_ids)))

return (train_ids, test_ids)

# Splits the dataset into a training/test sets in the setting of using 2
# classes (predicting the polarity of the review where ratings 1 & 2
# are considered negative, ratings 4 & 5 are positive, and rating 3 is
# ignored)
def split_train_test_2class(self, rating, percent_test,
                           balanced = "unbalanced"):
    np.random.seed(1234)

    rating = np.array(rating, dtype='int32')
    # length
    num_reviews = len(rating)

```

```

review_ids = np.arange(0, num_reviews)

# convert to binary, with ratings [1, 2] --> neg and [4, 5] --> pos
rating[rating == 2] = 1
rating[rating == 4] = 5

ids = (rating == 1) + (rating == 5)
review_ids = review_ids[ids]
rating = rating[ids]
rating[rating == 1] = 0
rating[rating == 5] = 1

# get length after filtering
num_reviews = rating.shape[0]

if balanced == "unbalanced":
    ntest = np.floor(num_reviews * percent_test)
    np.random.shuffle(review_ids)

    test_ids = review_ids[:ntest]
    train_ids = review_ids[ntest:]

elif balanced == "balanced":
    (sizes, bins) = np.histogram(rating, [0, 1, 2])
    min_size = np.min(sizes)
    print(min_size)

    # sample review ids equally among classes
    test_ids = np.zeros((0,), dtype="int32")
    train_ids = np.zeros((0,), dtype="int32")
    rating = np.array(rating)
    ntest = np.floor(min_size * percent_test)
    for c in [0, 1]:
        cids = review_ids[np.nonzero(rating == c)]
        np.random.shuffle(cids)

        test_ids = np.r_[test_ids, cids[:ntest]]
        train_ids = np.r_[train_ids, cids[ntest:min_size]]

train_file = self.REVIEWS_PATH + "2class-" + balanced+ "-train.txt"
test_file = self.REVIEWS_PATH + "2class-" + balanced+ "-test.txt"

open(train_file, 'w').write('\n'.join(map(str, train_ids)))
open(test_file, 'w').write('\n'.join(map(str, test_ids)))

return (train_ids, test_ids)

```

```

# Reads a training or test file. The file contains the indices of the
# reviews from the clean reviews file.
def read_train_test_file(self, file_name):
    ins = open(file_name).readlines()
    ins = [int(i.strip()) for i in ins]

    return ins

# A helper function.
def set_binary_klass(self, ar):
    ar[(ar == 1) + (ar == 2)] = 0
    ar[(ar == 4) + (ar == 5)] = 1

# Returns (train_x, train_y, test_x, test_y)
# where x is the review body and y is the rating (1->5 or 0->1)
def get_train_test(self, klass = "2", balanced = "balanced"):
    (rating, a, b, c, body) = self.read_clean_reviews()
    rating = np.array(rating)
    body = pd.Series(body)

    train_file = (self.REVIEWS_PATH + klass + "class-" +
                  balanced+ "-train.txt")
    test_file = (self.REVIEWS_PATH + klass + "class-" +
                 balanced+ "-test.txt")

    train_ids = self.read_train_test_file(train_file)
    test_ids = self.read_train_test_file(test_file)

    train_y = rating[train_ids]
    test_y = rating[test_ids]
    train_x = body[train_ids]
    test_x = body[test_ids]

    if klass == "2":
        self.set_binary_klass(train_y)
        self.set_binary_klass(test_y)

    return (train_x, train_y, test_x, test_y)

```

Writing labr.py

```

[ ]: from labr import LABR

labr_helper = LABR()

(d_train, y_train, d_test, y_test) = labr_helper.get_train_test(
    klass="2", balanced="unbalanced"

```

```

)

train_LABR_B_U = pd.DataFrame({DATA_COLUMN: d_train, LABEL_COLUMN: y_train})
test_LABR_B_U = pd.DataFrame({DATA_COLUMN: d_test, LABEL_COLUMN: y_test})

train_LABR_B_U[LABEL_COLUMN] = train_LABR_B_U[LABEL_COLUMN].apply(lambda x:
    ↪ 'NEG' if (x == 0) else 'POS')
test_LABR_B_U[LABEL_COLUMN] = test_LABR_B_U[LABEL_COLUMN].apply(lambda x: 'NEG'
    ↪ if (x == 0) else 'POS')

print(train_LABR_B_U[LABEL_COLUMN].value_counts() + test_LABR_B_U[LABEL_COLUMN].
    ↪ value_counts())
label_list_LABR_B_U = list(test_LABR_B_U[LABEL_COLUMN].unique())

data_LABR_B_U = CustomDataset(
    "LABR-UN-Binary", train_LABR_B_U, test_LABR_B_U, label_list_LABR_B_U
)
all_datasets.append(data_LABR_B_U)

```

```

POS      42832
NEG      8224
Name: label, dtype: int64

```

```

[ ]: df_ArSAS = pd.read_csv("/content/ArSAS..txt", sep="\t", encoding='utf-8')
df_ArSAS = df_ArSAS[["Tweet_text", "Sentiment_label"]]
df_ArSAS.columns = [DATA_COLUMN, LABEL_COLUMN]
print("Total length: ", len(df_ArSAS))
print(df_ArSAS[LABEL_COLUMN].value_counts())

label_list_ArSAS = list(df_ArSAS[LABEL_COLUMN].unique())
print(label_list_ArSAS)

train_ArSAS, test_ArSAS = train_test_split(df_ArSAS, test_size=0.2,
    ↪ random_state=42)
print("Training length: ", len(train_ArSAS))
print("Testing length: ", len(test_ArSAS))
data_ArSAS = CustomDataset("ArSAS", train_ArSAS, test_ArSAS, label_list_ArSAS)
all_datasets.append(data_ArSAS)

```

```

Total length: 19897
Negative      7384
Neutral      6894
Positive     4400
Mixed        1219
Name: label, dtype: int64
['Positive', 'Negative', 'Neutral', 'Mixed']
Training length: 15917

```

Testing length: 3980

3. Training procedure

```
[ ]: import numpy as np
import torch
import random
import matplotlib.pyplot as plt
import copy

from arabert.preprocess import ArabertPreprocessor
from sklearn.metrics import (accuracy_score, classification_report,
                             confusion_matrix, f1_score, precision_score,
                             recall_score)
from torch.utils.data import DataLoader, Dataset
from transformers import (AutoConfig, AutoModelForSequenceClassification,
                          AutoTokenizer, BertTokenizer, Trainer,
                          TrainingArguments)
from transformers.data.processors.utils import InputFeatures
```

```
[ ]: for x in all_datasets:
    print(x.name)
```

HARD
LABR-UN-Binary
ArSAS

```
[ ]: dataset_name = 'HARD'
model_name = 'aubmindlab/bert-base-arabertv02-twitter'
```

```
[ ]: for d in all_datasets:
    if d.name==dataset_name:
        selected_dataset = copy.deepcopy(d)
        print('Dataset found')
        break
```

Dataset found

```
[ ]: arabic_prep = ArabertPreprocessor(model_name)

selected_dataset.train[DATA_COLUMN] = selected_dataset.train[DATA_COLUMN].
    ↳apply(lambda x: arabic_prep.preprocess(x))
selected_dataset.test[DATA_COLUMN] = selected_dataset.test[DATA_COLUMN].
    ↳apply(lambda x: arabic_prep.preprocess(x))
```

```
[ ]: list(selected_dataset.train[DATA_COLUMN][0:10])
```


[illegible]

```
[ ]: tok = AutoTokenizer.from_pretrained(model_name)
```

```
Downloading: 0%|          | 0.00/476 [00:00<?, ?B/s]
```

```

Downloading: 0%|          | 0.00/733k [00:00<?, ?B/s]

```

```

Downloading: 0%|          | 0.00/1.19M [00:00<?, ?B/s]

```

```

Downloading:  0%|          | 0.00/112 [00:00<?, ?B/s]

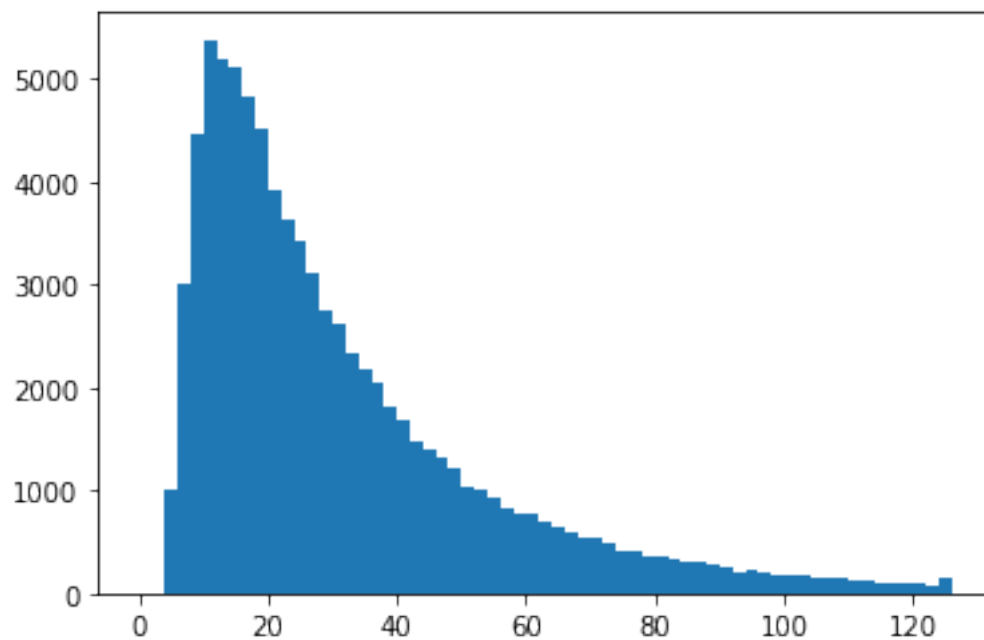
```

```
[ ]: print("Training Sentence Lengths: ")
plt.hist([ len(tok.tokenize(sentence)) for sentence in selected_dataset.
    ↪ train[DATA_COLUMN].to_list()], bins=range(0,128,2))
plt.show()

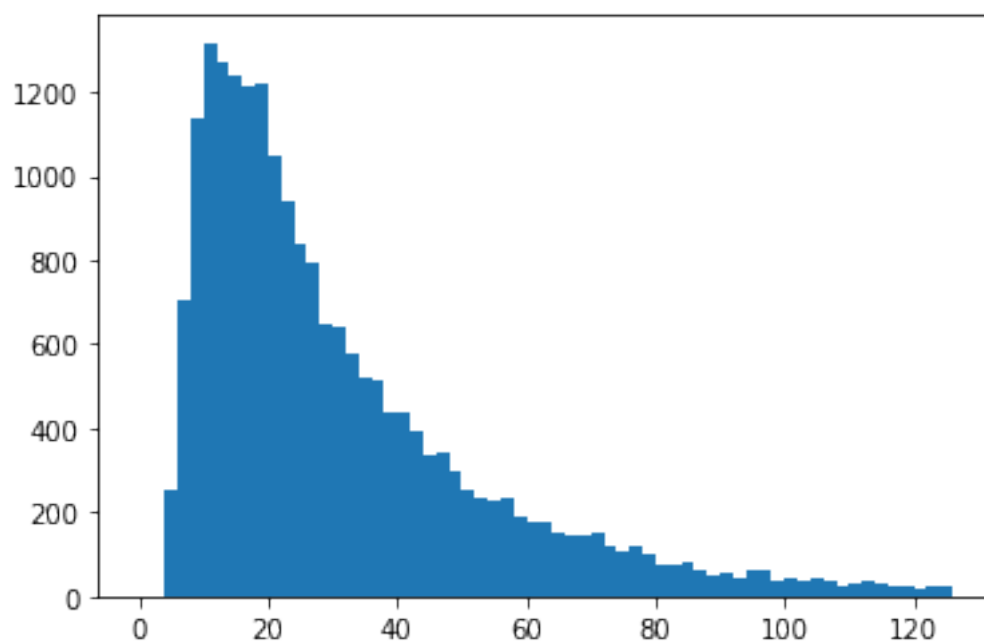
print("Testing Sentence Lengths: ")
plt.hist([ len(tok.tokenize(sentence)) for sentence in selected_dataset.
    ↪ test[DATA_COLUMN].to_list()], bins=range(0,128,2))
plt.show()
```

Training Sentence Lengths:

Token indices sequence length is longer than the specified maximum sequence length for this model (658 > 512). Running this sequence through the model will result in indexing errors



Testing Sentence Lengths:



```
[ ]: max_len = 128
```

```
[ ]: print("Truncated training sequences: ", sum([len(tok.tokenize(sentence)) >
    ↪max_len for sentence in selected_dataset.test[DATA_COLUMN].to_list()]))

print("Truncated testing sequences: ", sum([len(tok.tokenize(sentence)) >
    ↪max_len for sentence in selected_dataset.test[DATA_COLUMN].to_list()]))
```

Truncated training sequences: 446

Truncated testing sequences: 446

```
[ ]: class ClassificationDataset(Dataset):
    def __init__(self, text, target, model_name, max_len, label_map):
        super(ClassificationDataset).__init__()

        self.text = text
        self.target = target
        self.tokenizer_name = model_name
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.max_len = max_len
        self.label_map = label_map

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

    def __getitem__(self, item):
        text = str(self.text[item])
        text = " ".join(text.split())

        inputs = self.tokenizer(
            text,
            max_length=self.max_len,
            padding='max_length',
            truncation=True
        )
        return InputFeatures(**inputs, label=self.label_map[self.target[item]])
```

```
[ ]: label_map = { v:index for index, v in enumerate(selected_dataset.label_list) }
print(label_map)

train_dataset = ClassificationDataset(
    selected_dataset.train[DATA_COLUMN].to_list(),
    selected_dataset.train[LABEL_COLUMN].to_list(),
    model_name,
    max_len,
    label_map
)
test_dataset = ClassificationDataset(
```

```
{ 'NEG': 0, 'POS': 1}
```

[illegible]

```
[ ]: def compute_metrics(p): #p should be of type EvalPrediction
    preds = np.argmax(p.predictions, axis=1)
    assert len(preds) == len(p.label_ids)
    #print(classification_report(p.label_ids,preds))
    #print(confusion_matrix(p.label_ids,preds))
    macro_f1 = f1_score(p.label_ids,preds,average='macro')
    #macro_precision = precision_score(p.label_ids,preds,average='macro')
    #macro_recall = recall_score(p.label_ids,preds,average='macro')
    acc = accuracy_score(p.label_ids,preds)
    return {
        'macro_f1' : macro_f1,
        'accuracy': acc
    }
```

```
[ ]: def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic=True
    torch.backends.cudnn.benchmark = False
```

4. Regular Training

```
[ ]: training_args = TrainingArguments(
    output_dir= "./train",
    adam_epsilon = 1e-8,
    learning_rate = 2e-5,
    fp16 = False, # enable this when using V100 or T4 GPU
    per_device_train_batch_size = 16, # up to 64 on 16GB with max len of 128
    per_device_eval_batch_size = 128,
    gradient_accumulation_steps = 2, # use this to scale batch size without
    ↪needing more memory
    num_train_epochs= 2,
    warmup_ratio = 0,
    do_eval = True,
    evaluation_strategy = 'epoch',
    save_strategy = 'epoch',
    load_best_model_at_end = True, # this allows to automatically get the best
    ↪model at the end based on whatever metric we want
    metric_for_best_model = 'macro_f1',
    greater_is_better = True,
    seed = 25
)

set_seed(training_args.seed)
```

```
[ ]: trainer = Trainer(
    model = model_init(),
    args = training_args,
    train_dataset = train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics,
)
```

Downloading: 0%| | 0.00/667 [00:00<?, ?B/s]

Downloading: 0%| | 0.00/516M [00:00<?, ?B/s]

Some weights of the model checkpoint at aubmindlab/bert-base-arabertv02-twitter

were not used when initializing BertForSequenceClassification:

```
['cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.weight',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.bias',
'cls.predictions.transform.dense.weight', 'cls.predictions.bias']
```

- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at aubmindlab/bert-base-arabertv02-twitter and are newly initialized: ['bert.pooler.dense.bias', 'classifier.bias', 'classifier.weight', 'bert.pooler.dense.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

```
[ ]: trainer.train()
```

```
***** Running training *****
```

```
  Num examples = 84558
```

```
  Num Epochs = 2
```

```
  Instantaneous batch size per device = 16
```

```
  Total train batch size (w. parallel, distributed & accumulation) = 32
```

```
  Gradient Accumulation steps = 2
```

```
  Total optimization steps = 5284
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
***** Running Evaluation *****
```

```
  Num examples = 21140
```

```
  Batch size = 128
```

```
Saving model checkpoint to ./train/checkpoint-2642
```

```
Configuration saved in ./train/checkpoint-2642/config.json
```

```
Model weights saved in ./train/checkpoint-2642/pytorch_model.bin
```

```
***** Running Evaluation *****
```

```
  Num examples = 21140
```

```
  Batch size = 128
```

```
Saving model checkpoint to ./train/checkpoint-5284
```

```
Configuration saved in ./train/checkpoint-5284/config.json
```

```
Model weights saved in ./train/checkpoint-5284/pytorch_model.bin
```

Training completed. Do not forget to share your model on huggingface.co/models
=)

Loading best model from ./train/checkpoint-5284 (score: 0.9640014145557232).

```
[ ]: TrainOutput(global_step=5284, training_loss=0.109168441434596,  
metrics={'train_runtime': 4269.5782, 'train_samples_per_second': 39.61,  
'train_steps_per_second': 1.238, 'total_flos': 1.112315142087168e+16,  
'train_loss': 0.109168441434596, 'epoch': 2.0})
```

```
[ ]: inv_label_map = inv_label_map = { v:k for k, v in label_map.items()  
print(inv_label_map)  
trainer.model.config.label2id = label_map  
trainer.model.config.id2label = inv_label_map  
trainer.save_model("output_dir")  
train_dataset.tokenizer.save_pretrained("output_dir")
```

Saving model checkpoint to output_dir
Configuration saved in output_dir/config.json

{0: 'NEG', 1: 'POS'}

Model weights saved in output_dir/pytorch_model.bin
tokenizer config file saved in output_dir/tokenizer_config.json
Special tokens file saved in output_dir/special_tokens_map.json

```
[ ]: ('output_dir/tokenizer_config.json',  
      'output_dir/special_tokens_map.json',  
      'output_dir/vocab.txt',  
      'output_dir/added_tokens.json',  
      'output_dir/tokenizer.json')
```

```
[ ]: !cp output_dir /content/drive/MyDrive
```

cp: -r not specified; omitting directory 'output_dir'

5. Predict Using The Saved Model

```
[ ]: from transformers import pipeline
```

```
[ ]: pipe = pipeline("sentiment-analysis", model="output_dir", device=0,  
    ↪return_all_scores=True)
```

loading configuration file output_dir/config.json

```
Model config BertConfig {  
  "_name_or_path": "aubmindlab/bert-base-arabertv02-twitter",  
  "architectures": [  
    "BertForSequenceClassification"
```

```

],
"attention_probs_dropout_prob": 0.1,
"classifier_dropout": null,
"gradient_checkpointing": false,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 768,
"id2label": {
  "0": "NEG",
  "1": "POS"
},
"initializer_range": 0.02,
"intermediate_size": 3072,
"label2id": {
  "NEG": 0,
  "POS": 1
},
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 12,
"num_hidden_layers": 12,
"pad_token_id": 0,
"position_embedding_type": "absolute",
"problem_type": "single_label_classification",
"torch_dtype": "float32",
"transformers_version": "4.12.2",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 64000
}

```

loading configuration file output_dir/config.json

```

Model config BertConfig {
  "_name_or_path": "aubmindlab/bert-base-arabertv02-twitter",
  "architectures": [
    "BertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "NEG",
    "1": "POS"
  },
}

```



```

"initializer_range": 0.02,
"intermediate_size": 3072,
"label2id": {
    "NEG": 0,
    "POS": 1
},
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 12,
"num_hidden_layers": 12,
"pad_token_id": 0,
"position_embedding_type": "absolute",
"problem_type": "single_label_classification",
"torch_dtype": "float32",
"transformers_version": "4.12.2",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 64000
}

```

loading weights file output_dir/pytorch_model.bin
All model checkpoint weights were used when initializing
BertForSequenceClassification.

All the weights of BertForSequenceClassification were initialized from the model
checkpoint at output_dir.

If your task is similar to the task the model of the checkpoint was trained on,
you can already use BertForSequenceClassification for predictions without
further training.

Didn't find file output_dir/added_tokens.json. We won't load it.

loading file output_dir/vocab.txt

loading file output_dir/tokenizer.json

loading file None

loading file output_dir/special_tokens_map.json

loading file output_dir/tokenizer_config.json

```
[ ]: pipe("Some Text")
```

```
[ ]: [[{'label': 'NEG', 'score': 0.19983966648578644},
      {'label': 'POS', 'score': 0.8001603484153748}]]
```

6. K-fold & Ensemble all the cross validation models

```
[ ]: # do kfold on the training. Check the performance on the test set
kfold_dataset = selected_dataset.train
# do kfold on all the dataset. Here we will not have any dataset to check final
↪ performance on (this is used mainly in competitions)
```

```
# kfold_dataset = pd.concat([selected_dataset.train,selected_dataset.test])
kfold_dataset.reset_index(inplace=True,drop=True)
```

```
[ ]: inv_label_map = { v:k for k, v in label_map.items() }
```

```
[ ]: from sklearn.model_selection import StratifiedKFold
```

```
kf = StratifiedKFold(
    n_splits=2,
    shuffle=True,
    random_state=123
)
```

```
[ ]: all_results = []
fold_best_f1 = 0
best_fold = None
for fold_num , (train, dev) in enumerate(kf.
    ↪split(kfold_dataset,kfold_dataset['label'])):
    print("*****Starting Fold Num: ", fold_num,"␣
    ↪*****")

    train_dataset = ClassificationDataset(list(kfold_dataset[DATA_COLUMN][train]),
                                           list(kfold_dataset[LABEL_COLUMN][train]),
                                           model_name,
                                           max_len,
                                           label_map)

    val_dataset = ClassificationDataset(list(kfold_dataset[DATA_COLUMN][dev]),
                                         list(kfold_dataset[LABEL_COLUMN][dev]),
                                         model_name,
                                         max_len,
                                         label_map)

    training_args = TrainingArguments(
        output_dir= f"./train_{fold_num}",
        adam_epsilon = 1e-8,
        learning_rate = 2e-5,
        fp16 = False,
        per_device_train_batch_size = 64,
        per_device_eval_batch_size = 128,
        gradient_accumulation_steps = 2,
        num_train_epochs= 2,
        warmup_ratio = 0,
        do_eval = True,
        evaluation_strategy = 'epoch',
        save_strategy = 'epoch',
        load_best_model_at_end = True,
```

```

    metric_for_best_model = 'macro_f1',
    greater_is_better = True,
    seed = 123
)

set_seed(training_args.seed)

trainer = Trainer(
    model = model_init(),
    args = training_args,
    train_dataset = train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
)
trainer.model.config.label2id = label_map
trainer.model.config.id2label = inv_label_map

trainer.train()

results = trainer.evaluate()
all_results.append(results)
print(results)

trainer.save_model(f"./train_{fold_num}/best_model")
val_dataset.tokenizer.save_pretrained(f"./train_{fold_num}/best_model")

# delete the rest of the checkpoints
rm -rf f"./train_{fold_num}/checkpoint-*"

if results['eval_macro_f1'] > fold_best_f1:
    print('*****New Best Model Found!')
    fold_best_f1 = results['eval_macro_f1']
    best_fold = fold_num

```

*****Starting Fold Num: 0 *****

```

loading file https://huggingface.co/aubmindlab/bert-base-
arabertv02-twitter/resolve/main/vocab.txt from cache at /root/.cache/huggingface
/transformers/dbef00ddc9b64a66ba8057785b166b744cef2a41be973446ad897a56ad317019.a
a4ad61e3b0a52c7bcf5410af86ef01a27cf1147665acd6bfba80731d053f78a
loading file https://huggingface.co/aubmindlab/bert-base-
arabertv02-twitter/resolve/main/tokenizer.json from cache at /root/.cache/huggin
gface/transformers/46fef3ab20b06df535befe0412ab892f9baec0a9f8e64d75a0142a67ce366
959.c7c33ce0611a0a55c52a9ba4c03992b47db6e8b9862113443132ed9af7185a19
loading file https://huggingface.co/aubmindlab/bert-base-
arabertv02-twitter/resolve/main/added_tokens.json from cache at None
loading file https://huggingface.co/aubmindlab/bert-base-

```

```

arabertv02-twitter/resolve/main/special_tokens_map.json from cache at /root/.cache/huggingface/transformers/7f74425f6809cddb05d5de7967a5af4e325b04245017a7b1917fe7d5cfb06988.dd8bd9bfd3664b530ea4e645105f557769387b3da9f79bdb55ed556bdd80611d
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/tokenizer_config.json from cache at /root/.cache/huggingface/transformers/582bc76b2b3acaaf545878170de8fbf8d6d1f65bd0180769ff4ed901cd60d3c4.9badb1b6af7f7e89d855c8fbc79dd73ef57ac1c9e573a43862ddaeb2c798a290
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/vocab.txt from cache at /root/.cache/huggingface/transformers/dbef00ddc9b64a66ba8057785b166b744cef2a41be973446ad897a56ad317019.a4ad61e3b0a52c7bcf5410af86ef01a27cf1147665acd6bfba80731d053f78a
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/tokenizer.json from cache at /root/.cache/huggingface/transformers/46fef3ab20b06df535befe0412ab892f9baec0a9f8e64d75a0142a67ce366959.c7c33ce0611a0a55c52a9ba4c03992b47db6e8b9862113443132ed9af7185a19
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/added_tokens.json from cache at None
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/special_tokens_map.json from cache at /root/.cache/huggingface/transformers/7f74425f6809cddb05d5de7967a5af4e325b04245017a7b1917fe7d5cfb06988.dd8bd9bfd3664b530ea4e645105f557769387b3da9f79bdb55ed556bdd80611d
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/tokenizer_config.json from cache at /root/.cache/huggingface/transformers/582bc76b2b3acaaf545878170de8fbf8d6d1f65bd0180769ff4ed901cd60d3c4.9badb1b6af7f7e89d855c8fbc79dd73ef57ac1c9e573a43862ddaeb2c798a290
PyTorch: setting up devices
The default value for the training argument `--report_to` will change in v5 (from all installed integrations to none). In v5, you will need to use `--report_to all` to get the same behavior as now. You should start updating your code and make this info disappear :-).
loading configuration file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/config.json from cache at /root/.cache/huggingface/transformers/1109ac490c1eb90f74960e17c00032f27ea3c4be159567d7ed5d2b5908f9855c.01294502d101541d98086466d32c6b4f04698a90a573cd06480d05bd0c20b2aa
Model config BertConfig {
  "_name_or_path": "bert-base-arabertv02",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,

```

```

    "max_position_embeddings": 512,
    "model_type": "bert",
    "num_attention_heads": 12,
    "num_hidden_layers": 12,
    "pad_token_id": 0,
    "position_embedding_type": "absolute",
    "torch_dtype": "float32",
    "transformers_version": "4.12.2",
    "type_vocab_size": 2,
    "use_cache": true,
    "vocab_size": 64000
}

```

loading weights file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/pytorch_model.bin from cache at /root/.cache/huggingface/transformers/1f7c10cecf08743620c7e224e2f3c6b072e45aee1e88fa324837fd199cf24f21.e7b697f3572c7ddd6984e105b6c6cacc07a625d1195f9be544d26d3ad7d0e442

Some weights of the model checkpoint at aubmindlab/bert-base-arabertv02-twitter were not used when initializing BertForSequenceClassification:

```

['cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.weight',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.bias',
'cls.predictions.transform.dense.weight', 'cls.predictions.bias']

```

- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).

- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at aubmindlab/bert-base-arabertv02-twitter and are newly initialized: ['bert.pooler.dense.bias', 'classifier.bias', 'classifier.weight', 'bert.pooler.dense.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

***** Running training *****

Num examples = 42279

Num Epochs = 2

Instantaneous batch size per device = 64

Total train batch size (w. parallel, distributed & accumulation) = 128

Gradient Accumulation steps = 2

Total optimization steps = 660

<IPython.core.display.HTML object>

***** Running Evaluation *****

```

    Num examples = 42279
    Batch size = 128
Saving model checkpoint to ./train_0/checkpoint-330
Configuration saved in ./train_0/checkpoint-330/config.json
Model weights saved in ./train_0/checkpoint-330/pytorch_model.bin
***** Running Evaluation *****
    Num examples = 42279
    Batch size = 128
Saving model checkpoint to ./train_0/checkpoint-660
Configuration saved in ./train_0/checkpoint-660/config.json
Model weights saved in ./train_0/checkpoint-660/pytorch_model.bin

Training completed. Do not forget to share your model on huggingface.co/models
=)

```

```

Loading best model from ./train_0/checkpoint-660 (score: 0.9611497653618151).
***** Running Evaluation *****
    Num examples = 42279
    Batch size = 128

<IPython.core.display.HTML object>

```

```

Saving model checkpoint to ./train_0/best_model
Configuration saved in ./train_0/best_model/config.json
{'eval_loss': 0.11393581330776215, 'eval_macro_f1': 0.9611497653618151,
'eval_accuracy': 0.9611627521937605, 'eval_runtime': 325.6922,
'eval_samples_per_second': 129.813, 'eval_steps_per_second': 1.016, 'epoch':
2.0}

```

```

Model weights saved in ./train_0/best_model/pytorch_model.bin
tokenizer config file saved in ./train_0/best_model/tokenizer_config.json
Special tokens file saved in ./train_0/best_model/special_tokens_map.json

```

```

*****New Best Model Found!*****
*****Starting Fold Num: 1 *****

```

```

loading file https://huggingface.co/aubmindlab/bert-base-
arabertv02-twitter/resolve/main/vocab.txt from cache at /root/.cache/huggingface
/transformers/dbef00ddc9b64a66ba8057785b166b744cef2a41be973446ad897a56ad317019.a
a4ad61e3b0a52c7bcf5410af86ef01a27cf1147665acd6bfba80731d053f78a
loading file https://huggingface.co/aubmindlab/bert-base-
arabertv02-twitter/resolve/main/tokenizer.json from cache at /root/.cache/huggin
gface/transformers/46fef3ab20b06df535befe0412ab892f9baec0a9f8e64d75a0142a67ce366
959.c7c33ce0611a0a55c52a9ba4c03992b47db6e8b9862113443132ed9af7185a19
loading file https://huggingface.co/aubmindlab/bert-base-
arabertv02-twitter/resolve/main/added_tokens.json from cache at None
loading file https://huggingface.co/aubmindlab/bert-base-

```

```

arabertv02-twitter/resolve/main/special_tokens_map.json from cache at /root/.cache/huggingface/transformers/7f74425f6809cddb05d5de7967a5af4e325b04245017a7b1917fe7d5c9fb06988.dd8bd9bfd3664b530ea4e645105f557769387b3da9f79bdb55ed556bdd80611d
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/tokenizer_config.json from cache at /root/.cache/huggingface/transformers/582bc76b2b3acaaf545878170de8fbf8d6d1f65bd0180769ff4ed901cd60d3c4.9badb1b6af7f7e89d855c8fbc79dd73ef57ac1c9e573a43862ddaeb2c798a290
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/vocab.txt from cache at /root/.cache/huggingface/transformers/dbef00ddc9b64a66ba8057785b166b744cef2a41be973446ad897a56ad317019.a4ad61e3b0a52c7bcf5410af86ef01a27cf1147665acd6bfba80731d053f78a
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/tokenizer.json from cache at /root/.cache/huggingface/transformers/46fef3ab20b06df535befe0412ab892f9baec0a9f8e64d75a0142a67ce366959.c7c33ce0611a0a55c52a9ba4c03992b47db6e8b9862113443132ed9af7185a19
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/added_tokens.json from cache at None
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/special_tokens_map.json from cache at /root/.cache/huggingface/transformers/7f74425f6809cddb05d5de7967a5af4e325b04245017a7b1917fe7d5c9fb06988.dd8bd9bfd3664b530ea4e645105f557769387b3da9f79bdb55ed556bdd80611d
loading file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/tokenizer_config.json from cache at /root/.cache/huggingface/transformers/582bc76b2b3acaaf545878170de8fbf8d6d1f65bd0180769ff4ed901cd60d3c4.9badb1b6af7f7e89d855c8fbc79dd73ef57ac1c9e573a43862ddaeb2c798a290
PyTorch: setting up devices
The default value for the training argument `--report_to` will change in v5 (from all installed integrations to none). In v5, you will need to use `--report_to all` to get the same behavior as now. You should start updating your code and make this info disappear :-).
loading configuration file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/config.json from cache at /root/.cache/huggingface/transformers/1109ac490c1eb90f74960e17c00032f27ea3c4be159567d7ed5d2b5908f9855c.01294502d101541d98086466d32c6b4f04698a90a573cd06480d05bd0c20b2aa
Model config BertConfig {
  "_name_or_path": "bert-base-arabertv02",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,

```

```

    "max_position_embeddings": 512,
    "model_type": "bert",
    "num_attention_heads": 12,
    "num_hidden_layers": 12,
    "pad_token_id": 0,
    "position_embedding_type": "absolute",
    "torch_dtype": "float32",
    "transformers_version": "4.12.2",
    "type_vocab_size": 2,
    "use_cache": true,
    "vocab_size": 64000
}

```

loading weights file https://huggingface.co/aubmindlab/bert-base-arabertv02-twitter/resolve/main/pytorch_model.bin from cache at /root/.cache/huggingface/transformers/1f7c10cecf08743620c7e224e2f3c6b072e45aee1e88fa324837fd199cf24f21.e7b697f3572c7ddd6984e105b6c6cacc07a625d1195f9be544d26d3ad7d0e442

Some weights of the model checkpoint at aubmindlab/bert-base-arabertv02-twitter were not used when initializing BertForSequenceClassification:

```

['cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.weight',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.bias',
'cls.predictions.transform.dense.weight', 'cls.predictions.bias']

```

- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).

- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at aubmindlab/bert-base-arabertv02-twitter and are newly initialized: ['bert.pooler.dense.bias', 'classifier.bias', 'classifier.weight', 'bert.pooler.dense.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

***** Running training *****

Num examples = 42279

Num Epochs = 2

Instantaneous batch size per device = 64

Total train batch size (w. parallel, distributed & accumulation) = 128

Gradient Accumulation steps = 2

Total optimization steps = 660

<IPython.core.display.HTML object>

***** Running Evaluation *****


```

    Num examples = 42279
    Batch size = 128
Saving model checkpoint to ./train_1/checkpoint-330
Configuration saved in ./train_1/checkpoint-330/config.json
Model weights saved in ./train_1/checkpoint-330/pytorch_model.bin
***** Running Evaluation *****
    Num examples = 42279
    Batch size = 128
Saving model checkpoint to ./train_1/checkpoint-660
Configuration saved in ./train_1/checkpoint-660/config.json
Model weights saved in ./train_1/checkpoint-660/pytorch_model.bin

Training completed. Do not forget to share your model on huggingface.co/models
=)

```

```

Loading best model from ./train_1/checkpoint-660 (score: 0.9601158964922223).
***** Running Evaluation *****
    Num examples = 42279
    Batch size = 128

<IPython.core.display.HTML object>

```

```

Saving model checkpoint to ./train_1/best_model
Configuration saved in ./train_1/best_model/config.json
{'eval_loss': 0.11304667592048645, 'eval_macro_f1': 0.9601158964922223,
'eval_accuracy': 0.9601220464060172, 'eval_runtime': 324.8582,
'eval_samples_per_second': 130.146, 'eval_steps_per_second': 1.019, 'epoch':
2.0}

Model weights saved in ./train_1/best_model/pytorch_model.bin
tokenizer config file saved in ./train_1/best_model/tokenizer_config.json
Special tokens file saved in ./train_1/best_model/special_tokens_map.json

```

```
[ ]: all_results
```

```

[ ]: [{'epoch': 2.0,
      'eval_accuracy': 0.9611627521937605,
      'eval_loss': 0.11393581330776215,
      'eval_macro_f1': 0.9611497653618151,
      'eval_runtime': 325.6922,
      'eval_samples_per_second': 129.813,
      'eval_steps_per_second': 1.016},
      {'epoch': 2.0,
      'eval_accuracy': 0.9601220464060172,
      'eval_loss': 0.11304667592048645,

```

```

'eval_macro_f1': 0.9601158964922223,
'eval_runtime': 324.8582,
'eval_samples_per_second': 130.146,
'eval_steps_per_second': 1.019}]

```

```

[ ]: from statistics import mean
mean([x['eval_macro_f1'] for x in all_results])

```

```

[ ]: 0.9606328309270187

```

```

[ ]: from transformers import pipeline
import more_itertools

```

```

[ ]: inv_label_map = { v:k for k, v in label_map.items()}

```

```

[ ]: pred_df = selected_dataset.test[DATA_COLUMN]

```

```

[ ]: cross_val_df = pd.DataFrame([])
for i in range(0,2):
    pipe = pipeline("sentiment-analysis", model=f"train_{i}/best_model",
        ↪device=0, return_all_scores =True, max_length=max_len, truncation=True)
    preds = []
    for s in tqdm(more_itertools.chunked(list(pred_df), 32)): # batching for
        ↪faster inference
        preds.extend(pipe(s))
    cross_val_df[f'model_{i}'] = preds

```

loading configuration file train_0/best_model/config.json

```

Model config BertConfig {
  "_name_or_path": "aubmindlab/bert-base-arabertv02-twitter",
  "architectures": [
    "BertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "NEG",
    "1": "POS"
  },
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "label2id": {
    "NEG": 0,

```

```

    "POS": 1
  },
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "problem_type": "single_label_classification",
  "torch_dtype": "float32",
  "transformers_version": "4.12.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 64000
}

```

loading configuration file train_0/best_model/config.json

```

Model config BertConfig {
  "_name_or_path": "aubmindlab/bert-base-arabertv02-twitter",
  "architectures": [
    "BertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "NEG",
    "1": "POS"
  },
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "label2id": {
    "NEG": 0,
    "POS": 1
  },
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "problem_type": "single_label_classification",
  "torch_dtype": "float32",

```

```

    "transformers_version": "4.12.2",
    "type_vocab_size": 2,
    "use_cache": true,
    "vocab_size": 64000
}

```

loading weights file train_0/best_model/pytorch_model.bin
All model checkpoint weights were used when initializing
BertForSequenceClassification.

All the weights of BertForSequenceClassification were initialized from the model
checkpoint at train_0/best_model.

If your task is similar to the task the model of the checkpoint was trained on,
you can already use BertForSequenceClassification for predictions without
further training.

Didn't find file train_0/best_model/added_tokens.json. We won't load it.

loading file train_0/best_model/vocab.txt

loading file train_0/best_model/tokenizer.json

loading file None

loading file train_0/best_model/special_tokens_map.json

loading file train_0/best_model/tokenizer_config.json

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5:

TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0

Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`

"""

0it [00:00, ?it/s]

/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:490:

UserWarning: This DataLoader will create 8 worker processes in total. Our
suggested max number of worker in current system is 2, which is smaller than
what this DataLoader is going to create. Please be aware that excessive worker
creation might get DataLoader running slow or even freeze, lower the worker
number to avoid potential slowness/freeze if necessary.

cpuset_checked))

/usr/local/lib/python3.7/dist-packages/transformers/pipelines/base.py:910:

UserWarning: You seem to be using the pipelines sequentially on GPU. In order to
maximize efficiency please use a dataset

UserWarning,

loading configuration file train_1/best_model/config.json

Model config BertConfig {

 "name_or_path": "aubmindlab/bert-base-arabertv02-twitter",

 "architectures": [

 "BertForSequenceClassification"

],

 "attention_probs_dropout_prob": 0.1,

 "classifier_dropout": null,

 "gradient_checkpointing": false,

```

"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 768,
"id2label": {
  "0": "NEG",
  "1": "POS"
},
"initializer_range": 0.02,
"intermediate_size": 3072,
"label2id": {
  "NEG": 0,
  "POS": 1
},
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 12,
"num_hidden_layers": 12,
"pad_token_id": 0,
"position_embedding_type": "absolute",
"problem_type": "single_label_classification",
"torch_dtype": "float32",
"transformers_version": "4.12.2",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 64000
}

```

loading configuration file train_1/best_model/config.json

```

Model config BertConfig {
  "_name_or_path": "aubmindlab/bert-base-arabertv02-twitter",
  "architectures": [
    "BertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "id2label": {
    "0": "NEG",
    "1": "POS"
  },
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "label2id": {
    "NEG": 0,

```

```

    "POS": 1
},
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 12,
"num_hidden_layers": 12,
"pad_token_id": 0,
"position_embedding_type": "absolute",
"problem_type": "single_label_classification",
"torch_dtype": "float32",
"transformers_version": "4.12.2",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 64000
}

```

loading weights file train_1/best_model/pytorch_model.bin
All model checkpoint weights were used when initializing
BertForSequenceClassification.

All the weights of BertForSequenceClassification were initialized from the model
checkpoint at train_1/best_model.

If your task is similar to the task the model of the checkpoint was trained on,
you can already use BertForSequenceClassification for predictions without
further training.

Didn't find file train_1/best_model/added_tokens.json. We won't load it.

loading file train_1/best_model/vocab.txt

loading file train_1/best_model/tokenizer.json

loading file None

loading file train_1/best_model/special_tokens_map.json

loading file train_1/best_model/tokenizer_config.json

Oit [00:00, ?it/s]

```

[ ]: from collections import defaultdict

final_labels = []
final_scores = []
for id, row in cross_val_df.iterrows():
    total_score = defaultdict(lambda: 0)
    for pred in row:
        for cls in pred:
            total_score[cls['label']] += cls['score']

avg_score = { k: v/ 5 for k, v in total_score.items()}

```

```
final_labels.append(max(avg_score, key=avg_score.get))
final_scores.append(avg_score[max(avg_score, key=avg_score.get)])
```

```
[ ]: cross_val_df['preds'] = final_labels
cross_val_df['sentiment_score'] = final_scores
```

```
[ ]: cross_val_df['preds'].value_counts()
```

```
[ ]: POS    10797
      NEG    10343
      Name: preds, dtype: int64
```

```
[ ]: print(classification_report(selected_dataset.
      ↪test[LABEL_COLUMN],cross_val_df['preds']))
```

	precision	recall	f1-score	support
NEG	0.97	0.94	0.96	10663
POS	0.95	0.97	0.96	10477
accuracy			0.96	21140
macro avg	0.96	0.96	0.96	21140
weighted avg	0.96	0.96	0.96	21140