BERT_step_by_step

June 25, 2020

PF06 - NSGC: Neural Spell & Grammar Checker (en/pt)

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

0. Dataset and Description

Name: CoNLL-2014, JFLEG, BEA
Description: in this notebook we will use BERT and T5 to predict words in a sentence to perform a spell and grammar checker for Portuguese and English languages. For English, we will use the BERT and T5 models from transformers library (huggingface) and evaluate the performance in CoNLL-2014 and JFLEG datasets. For Portuguese, we will use the transformers/neuralmind BERT version and a custom dataset for evaluation.

1. Libraries and packages

1.1 Check device

[1]:
```
import torch
device = torch.device('cpu')
if torch.cuda.is_available():
    device_model = torch.cuda.get_device_name(0)
print('GPU model:', device_model)
```

```
GPU model: Tesla P100-PCIE-16GB
```

1.2 Install packages

[2]:
```
# install Python libs
!pip install -q     \
    numpy           \
    torch           \
    transformers
#-----------------------------
# install PyEnchant
! apt-get -qq update
! apt-get -qq install libenchant-dev
! pip install -q pyenchant
#-----------------------------
# string similarity and distance
! pip install -q strsimpy
```

1.3 Import libraries

[3]:
```
#-------------------------------------------------
# general
import torch
import numpy as np
import pandas as pd
import sys
import os
import pdb
import codecs
import subprocess
from multiprocessing import cpu_count
#-------------------------------------------------
# NLP
from transformers import T5Tokenizer, BertTokenizer, BertForMaskedLM,␣
 ↪T5ForConditionalGeneration
import enchant
import nltk
nltk.download('words')
from nltk.corpus import words
#-------------------------------------------------
# Edit distance algorithms
from strsimpy.levenshtein import Levenshtein
from strsimpy.normalized_levenshtein import NormalizedLevenshtein
from strsimpy.weighted_levenshtein import WeightedLevenshtein
from strsimpy.weighted_levenshtein import CharacterSubstitutionInterface
from strsimpy.damerau import Damerau
from strsimpy.optimal_string_alignment import OptimalStringAlignment
#-------------------------------------------------
# random seed generator
seed = 42
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
#-------------------------------------------------
# Suppress some of the logging
import logging
```

```
logging.getLogger("transformers.configuration_utils").setLevel(logging.WARNING)
logging.getLogger("transformers.modeling_utils").setLevel(logging.WARNING)
logging.getLogger("transformers.tokenization_utils").setLevel(logging.WARNING)
#------------------------------------------------
# Suppress warning messages
import warnings
warnings.filterwarnings("ignore")
#------------------------------------------------
# package version
print('Torch version:', torch.__version__)
```

```
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
Torch version: 1.5.1+cu101
```

1.4 Device info

[4]:
```
import torch
device = torch.device('cpu')
if torch.cuda.is_available():
    device = torch.device('cuda')
    device_model = torch.cuda.get_device_name(0)
    device_memory = torch.cuda.get_device_properties(device).total_memory / 1e9
#----------------------------
print('Device:', device)
print('GPU model:', device_model)
print('GPU memory: {0:.2f} GB'.format(device_memory))
print('#-------------------')
print('CPU cores:', cpu_count())
```

```
Device: cuda
GPU model: Tesla P100-PCIE-16GB
GPU memory: 17.07 GB
#-------------------
CPU cores: 4
```

2. Custom functions and classes

2.1 Function to read file

[5]:
```
'''
function that reads a file and return its text
#-----------------------------------------------------
parameters:
    - path: path of the file to be read
    - encoding: encoding to be used
returns:
    file content as list of strings
'''
def read_file(path, encoding='utf-8'):
    with codecs.open(path, encoding=encoding) as f:
        return f.read().splitlines()
```

2.2 Function to write in file

[6]:
```
'''
function that writes list of strings in a file
#-------------------------------------------------------
parameters:
    - sentences: list of strings to be written in file
    - path: path of the file where strings will be written
returns:
    path: same as input
'''
def write_file(sentences, path, encoding='utf-8'):
    with codecs.open(path, 'w', encoding=encoding) as f:
        for sentence in sentences:
            f.write(sentence + '\n')
    f.close()
    return path
```

2.3 Function to get tokenizer

[7]:
```
'''
function that returns the tokenizer associated to a string
#-------------------------------------------------------
parameters:
    tokenizer:
      BERT options:
        - 'bert-base-cased'
        - 'bert-large-cased'
        - 'bert-base-uncased'
        - 'bert-large-uncased'
      T5 options:
        - 't5-small'
        - 't5-base'
        - 't5-large'
        - 't5-3b'
        - 't5-11b'
      otherwise raise an error
returns:
    Hugging Face's tokenizer
'''
def get_tokenizer(tokenizer):
    # BERT
    if ((tokenizer == 'bert-base-cased') or
        (tokenizer == 'bert-large-cased') or
        (tokenizer == 'bert-base-uncased') or
        (tokenizer == 'bert-large-uncased') or
        (tokenizer == 'neuralmind/bert-large-portuguese-cased') or
        (tokenizer == 'neuralmind/bert-base-portuguese-cased')):
        return BertTokenizer.from_pretrained(tokenizer)
    #-------------------------------------------------------
    # T5
    elif ((tokenizer == 't5-small') or
          (tokenizer == 't5-base') or
```

```
          (tokenizer == 't5-large') or
          (tokenizer == 't5-3b') or
          (tokenizer == 't5-11b')):
        return T5Tokenizer.from_pretrained(tokenizer)
    #---------------------------------------------------------
    else:
        raise ValueError(f'Unsupported tokenizer: {tokenizer}')
```

2.4 Function to get model

[8]:
```
'''
function that returns the the network model associated to a string
#---------------------------------------------------------
parameters:
    model_name:
      BERT models:
        - 'bert-base-cased'                          # BERT base  cased   [en] (110 M␣
 ↪params)
        - 'bert-large-cased'                         # BERT large cased   [en] (340 M␣
 ↪params)
        - 'bert-base-uncased'                        # BERT base  uncased [en] (110 M␣
 ↪params)
        - 'bert-large-uncased'                       # BERT large uncased [en] (340 M␣
 ↪params)
        - 'neuralmind/bert-base-portuguese-cased'    # BERT base  cased   [pt] (110 M␣
 ↪params)
        - 'neuralmind/bert-large-portuguese-cased'   # BERT large cased   [pt] (340 M␣
 ↪params)
      T5 models:
        - 't5-small' (60 M params)
        - 't5-base'  (220 M params)
        - 't5-large' (770 M params)
        - 't5-3B'    (2.8 B params)
        - 't5-11B'   (11 B params)
      otherwise raise an error
returns:
    Hugging Face's model
'''
def get_model(model_name):
    # BERT
    if ((model_name == 'bert-base-cased') or                        # BERT base  cased␣
 ↪  [en]
        (model_name == 'bert-large-cased') or                       # BERT large cased␣
 ↪  [en]
        (model_name == 'bert-base-uncased') or                      # BERT base ␣
 ↪uncased [en]
        (model_name == 'bert-large-uncased') or                     # BERT large␣
 ↪uncased [en]
        (model_name == 'neuralmind/bert-base-portuguese-cased') or  # BERT base  cased␣
 ↪  [pt]
```

```
        (model_name == 'neuralmind/bert-large-portuguese-cased')):  # BERT large cased␣
↪  [pt]
        return BertForMaskedLM.from_pretrained(model_name)
    #--------------------------------------------------------
    # T5
    elif ((model_name == 't5-small') or      # T5 small [en]   242 MB
          (model_name == 't5-base')  or      # T5 base  [en]   892 MB
          (model_name == 't5-large') or      # T5 large [en]  2.95 GB
          (model_name == 't5-3B')    or      # T5 3B    [en]  11.4 GB
          (model_name == 't5-11B')):         # T5 11B   [en]  ??.? GB
        return T5ForConditionalGeneration.from_pretrained(model_name,␣
↪use_bfloat16=True)
    #--------------------------------------------------------
    else:
        raise ValueError(f'Unsupported model: {model_name}')
```

2.5 Function to edit distance algorithm

[9]:
```
'''
function that returns the algorithm to calculate the edit distance
#-------------------------------------------------------
parameters:                  +--------------------------+---------+
    algorithm:               |        algorithm         | metric? |
                             +--------------------------+---------+
        - 'levenshtein'      | Levenshtein              |   yes   |
        - 'normalized'       | Normalized Levenshtein   |   no    |
        - 'weighted'         | Weighted Levenshtein     |   no    |
        - 'damerau'          | Damerau-Levenshtein      |   yes   |
        - 'osa'              | Optimal String Alignment |   no    |
    otherwise raise an error +--------------------------+---------+
returns:
    edit distance algorithm
'''
def get_distance_algorithm(algorithm):
    if (algorithm == 'levenshtein'):
        return Levenshtein()
    elif (algorithm == 'normalized'):
        return NormalizedLevenshtein()
    elif (algorithm == 'weighted'):
        return
    elif (algorithm == 'damerau'):
        return Damerau()
    elif (algorithm == 'osa'):
        return OptimalStringAlignment()
    else:
        raise ValueError(f'Unsupported algorithm: {algorithm}')
```

2.6 Function to calculate GLEU score

[10]:
```
'''
function that receives text files and calculate GLEU score
#-------------------------------------------------------
parameters:
    - src: source file
    - ref: reference file(s)
    - hyp: hypothesis file
    - n: n-gram order
    - num_iter: number of GLEU iterations
    - sent: sentence level scores
returns:
    GLEU score (float)
'''
def calc_gleu(src, ref, hyp, n=4, num_iter=500, sent=False):
    gleu_calculator.load_sources(src)
    gleu_calculator.load_references(ref)
    if len(ref) == 1:
        print("There is one reference. NOTE: GLEU is not computing the confidence␣
 ↪interval.")
        gleu = [g for g in gleu_calculator.run_iterations(
            num_iterations=num_iter,
            source=src,
            hypothesis=hyp,
            per_sent=sent)][0][0]
    else:
        gleu = [g for g in gleu_calculator.run_iterations(
            num_iterations=num_iter,
            source=src,
            hypothesis=hyp,
            per_sent=sent)][0][0]
    #print(gleu)
    return float(gleu)*100
```

2.7 Function to calculate MaxMatch score

[11]:
```
'''
function that runs Python 2 script to calculate M^2 score
#-------------------------------------------------------
parameters:
    - src_file_path: source file path
    - ref_file_path: reference file path
returns:
    MaxMatch score (precision, recall, F_{0.5}) as string
'''
def m2scorer(src_file_path, ref_file_path):
    process = subprocess.Popen(['/content/m2scorer/scripts/m2scorer.py',␣
 ↪src_file_path, ref_file_path], stdout=subprocess.PIPE)
    output, error = process.communicate()
    output = output.decode("utf-8")
    return output
```

2.8 Function parse M2 file

[12]:
```
'''
function that receives M2 format file and returns original sentences
#-------------------------------------------------------
parameters:
    - m2_file: reference file in M2 format
    - output_file: file where the output will be written
returns:
    list of strings with original sentences
'''
def m2_parser(m2_file, output_file):
    # create output file
    !touch /content/conll14st-test-data/noalt/official-2014.1.cor
    # delete annotations, blank lines and 'S ' at the beginning of sentences
    !sed -e '/^A/d' -e '/^$/d' -e 's/^S //g' $m2_file > $output_file
    # read output file and return it as list of string
    conll_2014_test_src = read_file(output_file)
    return conll_2014_test_src
```

3. Datasets

3.1 CoNLL-2013

3.1.1 Download

[13]:
```
# test set
! wget -q -nc https://www.comp.nus.edu.sg/~nlp/conll13st/release2.3.1.tar.gz
! tar -xzf release2.3.1.tar.gz
! rm release2.3.1.tar.gz
```

3.1.2 Test set

[14]:
```
# import test set
#---------------------------
# source
m2_file     = '/content/release2.3.1/revised/data/official-preprocessed.m2'
output_file = '/content/release2.3.1/revised/data/official-preprocessed.src'
conll_2013_test_src = m2_parser(m2_file, output_file)
# reference
conll_2013_test_ref = read_file(m2_file)
```

3.1.3 Sample

[15]:
```
print('original sentence:')
print(conll_2013_test_src[0])
#---------------------------
print('\nannotation:')
print(*conll_2013_test_ref[0:4], sep='\n')
```

```
original sentence:
In modern digital world , electronic products are widely used in daily lives
```

```
such as Smart phones , computers and etc .

annotation:
S In modern digital world , electronic products are widely used in daily lives
such as Smart phones , computers and etc .
A 1 1|||ArtOrDet|||the|||REQUIRED|||-NONE-|||0
A 12 13|||Nn|||life|||REQUIRED|||-NONE-|||0
A 15 16|||Mec|||smart|||REQUIRED|||-NONE-|||0
```

3.2 CoNLL-2014

3.2.1 Download

[16]:
```
## training set
#from google.colab import drive
#drive.mount('/gdrive')
#----------------------------
# test set
! wget -q -nc https://www.comp.nus.edu.sg/~nlp/conll14st/conll14st-test-data.tar.gz
! tar -xzf conll14st-test-data.tar.gz
! rm conll14st-test-data.tar.gz
```

3.2.2 Training set

[17]:
```
# # import training set
# #----------------------------
# source
# m2_file     = '/gdrive/My Drive/Colab Notebooks/IA376E/Final Project/CoNLL-2014/
 ↪release3.3/data/conll14st-preprocessed.m2'
# output_file = '/gdrive/My Drive/Colab Notebooks/IA376E/Final Project/CoNLL-2014/
 ↪release3.3/data/conll14st-preprocessed.src'
# conll_2014_test_src = m2_parser(m2_file, output_file)
# # reference
# conll_2014_train_ref = read_file(m2_file)
```

3.2.3 Test set

[18]:
```
# import test set
#----------------------------
# source
m2_file     = '/content/conll14st-test-data/noalt/official-2014.1.m2'
output_file = '/content/conll14st-test-data/noalt/official-2014.1.src'
conll_2014_test_src = m2_parser(m2_file, output_file)
# reference
conll_2014_test_ref = read_file(m2_file)
```

3.2.4 Sample

[19]:
```
print('original sentence:')
print(conll_2014_test_src[3])
#----------------------------
print('\nannotation:')
```

```
print(*conll_2014_test_ref[7:9], sep='\n')
```

```
original sentence:
People get certain disease because of genetic changes .

annotation:
S People get certain disease because of genetic changes .
A 3 4|||Nn|||diseases|||REQUIRED|||-NONE-|||0
```

3.3 JFLEG

3.3.1 Download

[20]:
```
# clone GitHub repo
! git clone --quiet https://github.com/keisks/jfleg.git 2> /dev/null
```

3.3.2 Training set

[21]:
```
# import training set
#----------------------------
# source
jfleg_train_src = read_file('jfleg/dev/dev.src')
# references
jfleg_train_ref0 = read_file('jfleg/dev/dev.ref0')
jfleg_train_ref1 = read_file('jfleg/dev/dev.ref1')
jfleg_train_ref2 = read_file('jfleg/dev/dev.ref2')
jfleg_train_ref3 = read_file('jfleg/dev/dev.ref3')
```

3.3.3 Test set

[22]:
```
# import test set
#----------------------------
# source
jfleg_test_src = read_file('jfleg/test/test.src')
# references
jfleg_test_ref0 = read_file('jfleg/test/test.ref0')
jfleg_test_ref1 = read_file('jfleg/test/test.ref1')
jfleg_test_ref2 = read_file('jfleg/test/test.ref2')
jfleg_test_ref3 = read_file('jfleg/test/test.ref3')
```

3.3.4 Sample

[23]:
```
# print source and references example
print('source sentence:')
print(jfleg_test_src[0])
#----------------------------
print('\nreferences sentences:')
print(jfleg_test_ref0[0])
print(jfleg_test_ref1[0])
print(jfleg_test_ref2[0])
print(jfleg_test_ref3[0])
```

```
source sentence:
New and new technology has been introduced to the society .

references sentences:
New technology has been introduced to society .
New technology has been introduced into the society .
Newer and newer technology has been introduced into society .
Newer and newer technology has been introduced to the society .
```

3.4 BEA

3.4.1 Download

[24]:
```
# download test data
! wget -q -nc https://www.cl.cam.ac.uk/research/nl/bea2019st/data/wi+locness_v2.1.
 ↪bea19.tar.gz
! tar -xzf wi+locness_v2.1.bea19.tar.gz
! rm wi+locness_v2.1.bea19.tar.gz
```

3.4.2 Training set

[25]:
```
# import test set
#---------------------------
# source
# read A, B, C M2 file
m2_file_A = '/content/wi+locness/m2/A.train.gold.bea19.m2'
m2_file_B = '/content/wi+locness/m2/B.train.gold.bea19.m2'
m2_file_C = '/content/wi+locness/m2/C.train.gold.bea19.m2'
# read and concatenate all files
m2_ABC_file = read_file(m2_file_A) + read_file(m2_file_B) + read_file(m2_file_C)
# save to a file
m2_file = '/content/wi+locness/m2/ABC.train.gold.bea19.m2'
with open(m2_file, 'w') as f:
    for line in m2_ABC_file:
        f.write('%s\n' %line)
# parse M2 file
output_file = '/content/wi+locness/m2/ABCN.train.gold.bea19.src'
bea_train_src = m2_parser(m2_file, output_file)
#---------------------------
# reference
bea_train_ref = read_file(m2_file)
```

3.4.3 Development set

[26]:
```
# import test set
#---------------------------
# source
m2_file     = '/content/wi+locness/m2/ABCN.dev.gold.bea19.m2'
output_file = '/content/wi+locness/m2/ABCN.dev.gold.bea19.src'
bea_test_src = m2_parser(m2_file, output_file)
# reference
bea_test_ref = read_file(m2_file)
```

3.4.4 Sample

[27]:
```
print('original sentence:')
print(bea_train_src[0])
#----------------------------
print('\nannotation:')
print(*bea_train_ref[0:2], sep='\n')
```

```
original sentence:
My town is a medium size city with eighty thousand inhabitants .

annotation:
S My town is a medium size city with eighty thousand inhabitants .
A 5 6|||R:OTHER|||- sized|||REQUIRED|||-NONE-|||0
```

3.5 ReGRA

3.5.1 Import

[28]:
```
# mount drive to access file with sentences
from google.colab import drive
drive.mount('/gdrive')
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call
drive.mount("/gdrive", force_remount=True).
```

3.5.2 Test set

[29]:
```
# source
regra_src_file = '/gdrive/My Drive/Colab Notebooks/IA376E/Final Project/ReGRA/src.txt'
#regra_src = read_file(regra_src_file, encoding='latin-1')
regra_src = read_file(regra_src_file, encoding='utf-8')
#----------------------------
# reference
regra_ref_file = '/gdrive/My Drive/Colab Notebooks/IA376E/Final Project/ReGRA/ref.txt'
#regra_ref = read_file(regra_ref_file, encoding='latin-1')
regra_ref = read_file(regra_ref_file, encoding='utf-8')
```

3.5.4 Sample

[30]:
```
print('original sentences:')
print(*regra_src[1000:1003], sep='\n')
#----------------------------
print('\nreference sentences:')
print(*regra_ref[1000:1003], sep='\n')
```

```
original sentences:
Uma delegação de padeiros vem prestar seu apoio as mulheres dos grevistas.
Uma era ítala-brasileira.
Uma frota de navios norte-americanos se dirigiste ao Mar Mediterrâneo.

reference sentences:
Uma delegação de padeiros vem prestar seu apoio às mulheres dos grevistas.
```

```
Uma era ítalo-brasileira.
Uma frota de navios norte-americanos se dirige ao Mar Mediterrâneo.
```

4. Evaluation Metrics

4.1 M^2 (MaxMatch) score

4.1.1 Getting the M^2 scorer

[31]:
```
# get m2scorer
! wget -q -nc https://www.comp.nus.edu.sg/~nlp/sw/m2scorer.tar.gz
! tar -xzf m2scorer.tar.gz
! rm m2scorer.tar.gz
```

4.1.2 Testing the M^2 scorer

[32]:
```
# getting examples
src = '/content/m2scorer/example/system2'
ref = '/content/m2scorer/example/source_gold'
```

[33]:
```
# source
print('source sentences:')
print(*read_file(src), sep='\n')
```

```
source sentences:
A cat sat on mat .
The dog .
Giant otters are apex predator .
```

[34]:
```
# reference
print('reference sentences:')
print(*read_file(ref), sep='\n')
```

```
reference sentences:
S The cat sat at mat .
A 3 4|||Prep|||on|||REQUIRED|||-NONE-|||0
A 4 4|||ArtOrDet|||the||a|||REQUIRED|||-NONE-|||0

S The dog .
A 1 2|||NN|||dogs|||REQUIRED|||-NONE-|||0
A -1 -1|||noop|||-NONE-|||-NONE-|||-NONE-|||1

S Giant otters is an apex predator .
A 2 3|||SVA|||are|||REQUIRED|||-NONE-|||0
A 3 4|||ArtOrDet|||-NONE-|||REQUIRED|||-NONE-|||0
A 5 6|||NN|||predators|||REQUIRED|||-NONE-|||0
A 1 2|||NN|||otter|||REQUIRED|||-NONE-|||1
```

[35]:
```
# score
score = m2scorer(src, ref)
print(score)
```

```
Precision   : 0.7500
Recall      : 0.6000
F_0.5       : 0.7143
```

4.2 GLEU score

https://github.com/keisks/jfleg

4.2.1 Getting the GLEU scorer

[36]:
```
# import gleu metric
sys.path.append('/content/jfleg/eval/')
from gleu import GLEU
gleu_calculator = GLEU()
```

4.2.2 Testing the GLEU scorer

[37]:
```
# hyp = ref
#--------------------------
src = 'jfleg/test/test.src'
ref = ['jfleg/test/test.ref0']
hyp = 'jfleg/test/test.ref0'
print(f'GLEU = {calc_gleu(src, ref, hyp):.2f}')
```

```
There is one reference. NOTE: GLEU is not computing the confidence interval.
GLEU = 100.00
```

[38]:
```
# hyp = src
#--------------------------
# source file
src = 'jfleg/test/test.src'
# reference file
ref = ['jfleg/test/test.ref0',
       'jfleg/test/test.ref1',
       'jfleg/test/test.ref2',
       'jfleg/test/test.ref3']
# hypothesis file
hyp = 'jfleg/test/test.src'
# calculate score
print(f'GLEU = {calc_gleu(src, ref, hyp):.2f}')
```

```
GLEU = 40.47
```

[39]:
```
# hyp = ref
#--------------------------
# source file
src = 'jfleg/test/test.src'
#-------------
# ref0
hyp = 'jfleg/test/test.ref0'
ref = ['jfleg/test/test.ref1', 'jfleg/test/test.ref2', 'jfleg/test/test.ref3']
ref0 = calc_gleu(src, ref, hyp);
```

```
#-------------
# ref1
hyp = 'jfleg/test/test.ref1'
ref = ['jfleg/test/test.ref0', 'jfleg/test/test.ref2', 'jfleg/test/test.ref3']
ref1 = calc_gleu(src, ref, hyp);
#-------------
# ref2
hyp = 'jfleg/test/test.ref2'
ref = ['jfleg/test/test.ref0', 'jfleg/test/test.ref1', 'jfleg/test/test.ref3']
ref2 = calc_gleu(src, ref, hyp);
#-------------
# ref3
hyp = 'jfleg/test/test.ref3'
ref = ['jfleg/test/test.ref0', 'jfleg/test/test.ref1', 'jfleg/test/test.ref2']
ref3 = calc_gleu(src, ref, hyp);
#-------------
print(f'ref0 = {ref0:.2f}')
print(f'ref1 = {ref1:.2f}')
print(f'ref2 = {ref2:.2f}')
print(f'ref3 = {ref3:.2f}')
print('#-------------')
print(f'mean = {(ref0 + ref1 + ref2 + ref3) / 4:.2f}')
```

```
ref0 = 61.32
ref1 = 61.48
ref2 = 63.04
ref3 = 63.53
#-------------
mean = 62.34
```

reference table:

system	GLEU (dev)	GLEU (test)
SOURCE	38.21	40.54
REFERENCE	55.26	62.37

4.3 Edit distance

4.3.1 Getting distances algorithms

https://github.com/luozhouyang/python-string-similarity#damerau-levenshtein

[40]:
```
levenshtein = get_distance_algorithm('levenshtein')
damerau     = get_distance_algorithm('damerau')
normalized  = get_distance_algorithm('normalized')
weighted    = get_distance_algorithm('weighted')
osa         = get_distance_algorithm('osa')
```

4.3.2 Testing Damerau-Levenshtein distance algorithm

[41]:
```
# distance = 1: character removed
print('distance =', damerau.distance('Covid-19', 'Covid-9'))
```

```
distance = 1
```

[42]:
```
# distance = 2: character removed & character inserted
print('distance =', damerau.distance('Covid-19', 'Codiv-19'))
```

```
distance = 2
```

[43]:
```
# distance = 1: transposition of two adjacent characters
print('distance =', damerau.distance('Covid-19', 'Covid-91'))
```

```
distance = 1
```

5. Tokenizer

5.1 BERT

[44]:
```
# English
#tokenizer = get_tokenizer('bert-base-cased')
#tokenizer = get_tokenizer('bert-large-cased')
#tokenizer = get_tokenizer('bert-base-cased')
#tokenizer = get_tokenizer('bert-large-cased')
#---------------------------
# Portuguese
#tokenizer = get_tokenizer('neuralmind/bert-base-portuguese-cased')
tokenizer = get_tokenizer('neuralmind/bert-large-portuguese-cased')
```

5.2 T5

[45]:
```
#tokenizer = get_tokenizer('t5-small')
#tokenizer = get_tokenizer('t5-base')
#tokenizer = get_tokenizer('t5-large')
#tokenizer = get_tokenizer('t5-3b')
#tokenizer = get_tokenizer('t5-11b')
```

6. Model

6.1 BERT

[46]:
```
# English
#model = get_model('bert-base-cased')     # BERT base  cased   [en]  436 MB
#model = get_model('bert-large-cased')    # BERT large cased   [en] 1.34 GB
#model = get_model('bert-base-uncased')   # BERT base  uncased [en]  440 MB
#model = get_model('bert-large-uncased')  # BERT large uncased [en] 1.34 GB
#---------------------------
# Portuguese
```

```
#model = get_model('neuralmind/bert-base-portuguese-cased')  # BERT base  cased [pt] 
 ↪438 MB
model = get_model('neuralmind/bert-large-portuguese-cased') # BERT large cased [pt] 1.
 ↪34 GB
```

6.2 T5

[47]:
```
#model = get_model('t5-small')    #  242 MB
#model = get_model('t5-base')     #  892 MB
#model = get_model('t5-large')    # 2.95 GB
#model = get_model('t5-3b')       # 11.4 GB
#model = get_model('t5-11b')      # ??.? GB
```

7. Sentence Correction Suggestion

7.1 BERT-based function

Hyperparameters

[48]:
```
# topk model output predictions used to compare
k = 10
# Damerau-Levenshtein
edit_distance = get_distance_algorithm('damerau')
# threshold distance to suggest correction
threshold = 5
```

Function

[49]:
```
def suggest(sentences, tokenizer, model, distance, split=False, k=20, threshold=5,
 ↪device='cpu'):
    model.to(device)
    sentences_suggested = []
    for sentence in sentences:
        if split:
            tokenized = sentence.split()                                    # dummy
 ↪tokenizer
        else:
            tokenized = tokenizer.tokenize(sentence)                        # tokenize
        tokenized_ids = tokenizer.encode(tokenized)                         # '[CLS]' +
 ↪get word ids + '[SEP]'
        single_input_ids = torch.LongTensor(tokenized_ids).to(device)    # convert list
 ↪to tensor
        input_ids = single_input_ids.repeat(len(single_input_ids)-2, 1) # repeat tensor
        #----------------------------
        # mask tokens
        for i in range(len(input_ids)):
            input_ids[i][i+1] = tokenizer.mask_token_id
        #----------------------------
        # predict the top-k tokens for the masked ones
        topk_pred_pt = torch.zeros((len(tokenized), k))
        for i, masked_sentence in enumerate(input_ids):
            model_output = model(input_ids = masked_sentence.unsqueeze(dim=0))
```

```
            logits = model_output[0]
            _, predicted_ids = torch.topk(logits, k, sorted=True)
            topk_pred_pt[i] = predicted_ids.squeeze()[i+1]
        #----------------------------
        # convert ids back to words
        topk_pred_tokens = []   # list of lists
        for masked_sentence in topk_pred_pt:
            pred_list = []
            for predictions in masked_sentence:
                pred_list.append(tokenizer.decode([predictions.tolist()]))
            topk_pred_tokens.append(pred_list)
        #----------------------------
        # compare predictions and calculate edit distance
        suggestion = []
        for i, masked_token in enumerate(tokenized):
            # check if masked token is in predictions
            if masked_token in topk_pred_tokens[i]:
                # if it is, no correction is suggested
                suggestion.append(masked_token)
            #----------------------------
            else:
                # using distance?
                if (distance != None):
                    # if masked token not in predictions, calculate distance
                    dist = torch.zeros(k)
                    for j, prediction in enumerate(topk_pred_tokens[i]):
                        dist[j] = edit_distance.distance(masked_token, prediction)
                    # check if minimum distance is under a limiar
                    if torch.min(dist).item() <= threshold:
                        # if it is, make suggestions
                        # argmin returns the last index --> workaround: flip the tensor
                        min_index = len(dist) - torch.argmin(dist.flip(0)).item() - 1
                        suggestion.append(topk_pred_tokens[i][min_index])
                    #----------------------------
                    else:
                        # if it is not, make no correction suggestion
                        suggestion.append(masked_token)
                #----------------------------
                # greedy suggestion
                else:
                    suggestion.append(topk_pred_tokens[i][0])
        #----------------------------
        sentences_suggested.append(' '.join(suggestion))
    return sentences_suggested
```

7.2 Step-by-Step

[50]:
```
model.to(device);
```

7.2.0 Hyperparameters

[51]:
```
# topk model output predictions used to compare
k = 2
# Damerau-Levenshtein
edit_distance = get_distance_algorithm('damerau')
# threshold distance to suggest correction
threshold = 2
```

7.2.1 Get sentence

[52]:
```
# get sentence
sentence = regra_src[325]
sentence
```

[52]:
```
'Ele comprou este carro à prazo.'
```

[53]:
```
# get reference
reference = regra_ref[325]
reference
```

[53]:
```
'Ele comprou este carro a prazo.'
```

7.2.2 Tokenize

[54]:
```
# tokenize
#tokenized = sentence.split()
tokenized = tokenizer.tokenize(sentence)
tokenized
```

[54]:
```
['Ele', 'comprou', 'este', 'carro', 'à', 'prazo', '.']
```

[55]:
```
# '[CLS]' + get word ids + '[SEP]'
tokenized_ids = tokenizer.encode(tokenized)
tokenized_ids
```

[55]:
```
[101, 787, 10107, 860, 3883, 353, 6620, 119, 102]
```

[56]:
```
# convert list to tensor
single_input_ids = torch.LongTensor(tokenized_ids).to(device)
single_input_ids
```

[56]:
```
tensor([  101,   787, 10107,   860,  3883,   353,  6620,   119,   102],
       device='cuda:0')
```

[57]:
```
# repeat tensor
input_ids = single_input_ids.repeat(len(single_input_ids)-2, 1)
input_ids
```

[57]:
```
tensor([[  101,   787, 10107,   860,  3883,   353,  6620,   119,   102],
        [  101,   787, 10107,   860,  3883,   353,  6620,   119,   102],
        [  101,   787, 10107,   860,  3883,   353,  6620,   119,   102],
```

```
        [  101,   787, 10107,   860,  3883,   353,  6620,   119,   102],
        [  101,   787, 10107,   860,  3883,   353,  6620,   119,   102],
        [  101,   787, 10107,   860,  3883,   353,  6620,   119,   102],
        [  101,   787, 10107,   860,  3883,   353,  6620,   119,   102]],
       device='cuda:0')
```

7.2.3 Mask tokens

[58]:
```
# mask tokens
for i in range(len(input_ids)):
    input_ids[i][i+1] = tokenizer.mask_token_id
input_ids
```

[58]:
```
tensor([[  101,   103, 10107,   860,  3883,   353,  6620,   119,   102],
        [  101,   787,   103,   860,  3883,   353,  6620,   119,   102],
        [  101,   787, 10107,   103,  3883,   353,  6620,   119,   102],
        [  101,   787, 10107,   860,   103,   353,  6620,   119,   102],
        [  101,   787, 10107,   860,  3883,   103,  6620,   119,   102],
        [  101,   787, 10107,   860,  3883,   353,   103,   119,   102],
        [  101,   787, 10107,   860,  3883,   353,  6620,   103,   102]],
       device='cuda:0')
```

7.2.4 Top-k predictions

[59]:
```
topk_pred_pt = torch.zeros((len(tokenized), k))
for i, masked_sentence in enumerate(input_ids):
    model_output = model(input_ids = masked_sentence.unsqueeze(dim=0))
    logits = model_output[0]
    _, predicted_ids = torch.topk(logits, k, sorted=True)
    topk_pred_pt[i] = predicted_ids.squeeze()[i+1]
topk_pred_pt
```

[59]:
```
tensor([[11976.,   787.],
        [10107.,   376.],
        [  146.,   222.],
        [ 3883., 11426.],
        [  123.,   202.],
        [ 3122.,  5304.],
        [  119.,   136.]])
```

7.2.5 Convert IDs back to words

[60]:
```
# convert ids back to words
topk_pred_tokens = []   # list of lists
for masked_sentence in topk_pred_pt:
    pred_list = []
    for predictions in masked_sentence:
        pred_list.append(tokenizer.decode([predictions.tolist()]))
    topk_pred_tokens.append(pred_list)
topk_pred_tokens
```

[60]:
```
[['Você', 'Ele'],
 ['comprou', 'tem'],
 ['o', 'um'],
 ['carro', 'apartamento'],
 ['a', 'no'],
 ['vista', 'venda'],
 ['.', '?']]
```

7.2.6 Compare predictions and calculate distance

[61]:
```
# compare predictions and calculate edit distance
suggestion = []
for i, masked_token in enumerate(tokenized):
    # check if masked token is in predictions
    if masked_token in topk_pred_tokens[i]:
        # if it is, no correction is suggested
        suggestion.append(masked_token)
    #---------------------------
    else:
        # using distance?
        if (edit_distance != None):
            # if masked token not in predictions, calculate distance
            dist = torch.zeros(k)
            for j, prediction in enumerate(topk_pred_tokens[i]):
                dist[j] = edit_distance.distance(masked_token, prediction)
            # check if minimum distance is under a limiar
            if torch.min(dist).item() <= threshold:
                # if it is, make suggestions
                # argmin returns the last index --> workaround: flip the tensor
                min_index = len(dist) - torch.argmin(dist.flip(0)).item() - 1
                suggestion.append(topk_pred_tokens[i][min_index])
            #---------------------------
            else:
                # if it is not, make no correction suggestion
                suggestion.append(masked_token)
        #---------------------------
        # greedy suggestion
        else:
            suggestion.append(topk_pred_tokens[i][0])
#---------------------------
' '.join(suggestion)
```

[61]:
```
'Ele comprou este carro a prazo .'
```

End of the notebook