1b-Intro.to.NLP.using.TextBlob

November 8, 2024

1 TextBlob

TextBlob is a powerful NLP Python library. It can be used to perform a variety of NLP tasks. Documentation for TextBlob can be found here.

[1]:
```
%%capture
# Install textblob
!pip install -U textblob
```

[2]:
```
from textblob import TextBlob
```

1.1 Corpora

[3]:
```
%%capture
# Download corpora
!python -m textblob.download_corpora
```

[4]:
```
import nltk
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
```

[4]:
```
True
```

1.2 TextBlobs

[5]:
```
my_blob = TextBlob("There is more than one way to skin a cat.")
```

[6]:
```
my_blob
```

[6]:
```
TextBlob("There is more than one way to skin a cat.")
```

1.3 Tagging Parts of Speech

A list of the different parts of speech tags can be found here.

code	meaning	example
CC	coordinating conjunction	
CD	cardinal digit	

code	meaning	example
DT	determiner	
EX	existential there	(like: “there is” … think of it like “there exists”)
FW	foreign word	
IN	preposition/subordinating conjunction	
JJ	adjective	‘big’
JJR	adjective, comparative	‘bigger’
JJS	adjective, superlative	‘biggest’
LS	list marker	1)
MD	modal could,	will
NN	noun, singular	‘desk’
NNS	noun plural	‘desks’
NNP	proper noun, singular	‘Harrison’
NNPS	proper noun, plural	‘Americans’
PDT	predeterminer	‘all the kids’
POS	possessive ending	parent‘s
PRP	personal pronoun	I, he, she
PRP$	possessive pronoun	my, his, hers
RB	adverb	very, silently,
RBR	adverb, comparative	better
RBS	adverb, superlative	best
RP	particle	give up
TO	to go	’to‘ the store.
UH	interjection	errrrrrrrm
VB	verb, base form	take
VBD	verb, past tense	took
VBG	verb, gerund/present participle	taking
VBN	verb, past participle	taken
VBP	verb, sing. present, non-3d	take
VBZ	verb, 3rd person sing. present	takes
WDT	wh-determiner	which
WP	wh-pronoun	who, what
WP$	possessive wh-pronoun	whose
WRB	wh-adverb	where, when

[7]:
```
# Use the .tags attribute to see parts of speech
my_blob.tags
```

[7]:
```
[('There', 'EX'),
 ('is', 'VBZ'),
 ('more', 'JJR'),
 ('than', 'IN'),
 ('one', 'CD'),
 ('way', 'NN'),
 ('to', 'TO'),
 ('skin', 'VB'),
 ('a', 'DT'),
 ('cat', 'NN')]
```

1.4 Sentiment Analysis

Sentiment analysis can be used to understand the feeling or emotion tied to the text. The sentiment attribute in TextBlob will return two values: 1. The **polarity score** (a float between -1.0 and 1.0). -1 is negative, 1 is positive. 2. The **subjectivity** (a float between 0.0 and 1.0). 0 is very objective, while 1 is very subjective.

```
[8]: neg_blob = TextBlob("I am so tired. Today was a long, hard day.")
     neg_blob.sentiment
```

```
[8]: Sentiment(polarity=-0.24722222222222223, subjectivity=0.5472222222222222)
```

```
[9]: pos_blob = TextBlob("Today was a great day. I am so happy.")
     pos_blob.sentiment
```

```
[9]: Sentiment(polarity=0.8, subjectivity=0.875)
```

```
[10]: obj_blob = TextBlob("The cat is gray.")
      obj_blob.sentiment
```

```
[10]: Sentiment(polarity=0.0, subjectivity=0.0)
```

```
[11]: subj_blob = TextBlob("The cat is so cute and sweet.")
      print(subj_blob.sentiment)
      print(subj_blob.sentiment.subjectivity) # Only get the subjectivity
```

```
Sentiment(polarity=0.425, subjectivity=0.825)
0.825
```

Sentiment analysis of multiple sentences

```
[12]: my_poem = TextBlob('''
        Python is a great language to learn.
        You can easily do NLP; it's fab.
        It might take some getting used to.
        But it's definitely more gooder than Matlab.
      ''')
```

```
[13]: my_poem
```

```
[13]: TextBlob("
        Python is a great language to learn.
        You can easily do NLP; it's fab.
        It might take some getting used to.
```

```
  But it's definitely more gooder than Matlab.
")
```

```
[14]: my_poem.sentiment
```

```
[14]: Sentiment(polarity=0.5777777777777778, subjectivity=0.6944444444444445)
```

```
[15]: my_poem.sentences
```

```
[15]: [Sentence("
         Python is a great language to learn."),
       Sentence("You can easily do NLP; it's fab."),
       Sentence("It might take some getting used to."),
       Sentence("But it's definitely more gooder than Matlab.")]
```

```
[16]: for sentence in my_poem.sentences:
        print(sentence.sentiment)
```

```
Sentiment(polarity=0.8, subjectivity=0.75)
Sentiment(polarity=0.43333333333333335, subjectivity=0.8333333333333334)
Sentiment(polarity=0.0, subjectivity=0.0)
Sentiment(polarity=0.5, subjectivity=0.5)
```

1.4.1 Your Turn

Create three TextBlobs with the following sentiments: 1. Negative, subjective 2. Positive, objective 3. Neutral

```
[17]: # Solution 1
      text_ns = "It's a cruddy day."
      neg_sub = TextBlob(text_ns)
      neg_sub.sentiment
```

```
[17]: Sentiment(polarity=-0.9, subjectivity=0.9)
```

```
[ ]: # Solution 1
     text_ns = "Hitler was a terrible man."
     neg_sub = TextBlob(text_ns)
     neg_sub.sentiment
```

```
[ ]: Sentiment(polarity=-1.0, subjectivity=1.0)
```

```
[ ]: # Solution 2
     text_po = "Bill is a nice guy. He won the race."
     pos_obj = TextBlob(text_po)
     pos_obj.sentiment
     # no luck
```

```
[ ]: Sentiment(polarity=0.6, subjectivity=1.0)
```

[ ]:
```
# Solution 2
text_po = "My best friend had a baby boy."
pos_obj = TextBlob(text_po)
pos_obj.sentiment
# no luck
```

[ ]:
```
Sentiment(polarity=1.0, subjectivity=0.3)
```

[ ]:
```
# Solution 3
text_n = "One plus one is two."
neut = TextBlob(text_n)
neut.sentiment
```

[ ]:
```
Sentiment(polarity=0.0, subjectivity=0.0)
```

1.5 Tokenization

Tokenization is the process of splitting long strings of text into small pieces (tokens).

[18]:
```
my_poem.sentences
```

[18]:
```
[Sentence("
    Python is a great language to learn."),
 Sentence("You can easily do NLP; it's fab."),
 Sentence("It might take some getting used to."),
 Sentence("But it's definitely more gooder than Matlab.")]
```

[22]:
```
my_poem.sentences[0].words
```

[22]:
```
WordList(['Python', 'is', 'a', 'great', 'language', 'to', 'learn'])
```

[19]:
```
my_poem.words
```

[19]:
```
WordList(['Python', 'is', 'a', 'great', 'language', 'to', 'learn', 'You', 'can',
'easily', 'do', 'NLP', 'it', "'s", 'fab', 'It', 'might', 'take', 'some',
'getting', 'used', 'to', 'But', 'it', "'s", 'definitely', 'more', 'gooder',
'than', 'Matlab'])
```

[20]:
```
sorted(my_poem.word_counts.items(), key = lambda x: x[1], reverse=True)
```

[20]:
```
[('it', 3),
 ('to', 2),
 ('s', 2),
 ('python', 1),
 ('is', 1),
 ('a', 1),
 ('great', 1),
 ('language', 1),
 ('learn', 1),
 ('you', 1),
 ('can', 1),
 ('easily', 1),
 ('do', 1),
 ('nlp', 1),
 ('fab', 1),
 ('might', 1),
 ('take', 1),
 ('some', 1),
 ('getting', 1),
 ('used', 1),
 ('but', 1),
 ('definitely', 1),
 ('more', 1),
 ('gooder', 1),
 ('than', 1),
 ('matlab', 1)]
```

1.6 Singular & Plural

```
[23]: my_sent = TextBlob("The octopi went swimming in the dark ocean waters.")
```

```
[24]: my_sent.words
```

```
[24]: WordList(['The', 'octopi', 'went', 'swimming', 'in', 'the', 'dark', 'ocean',
      'waters'])
```

```
[25]: my_sent.words[0]
```

```
[25]: 'The'
```

```
[26]: # Singularize
      my_sent.words[-1].singularize()
```

```
[26]: 'water'
```

```
[27]: my_sent.words[1].singularize()
```

```
[27]: 'octopus'
```

```
[28]: # Pluralize
      my_sent.words[-2].pluralize()
```

```
[28]: 'oceans'
```

```
[29]: foo = my_sent.words[-2]
      foo == foo.singularize()
```

```
[29]: True
```

```
[34]: TextBlob("corpus").words.singularize(), TextBlob("corpus").words.pluralize()
```

```
[34]: (WordList(['corpu']), WordList(['corpora']))
```

```
[36]: my_sent.words[2:5]
```

```
[36]: WordList(['went', 'swimming', 'in'])
```

```
[38]: import numpy as np

      np.array(my_sent.words)
```

```
[38]: array(['The', 'octopi', 'went', 'swimming', 'in', 'the', 'dark', 'ocean',
             'waters'], dtype='<U8')
```

1.7 Stemming & Lemmatization

Stemming is the process of deleting prefixes and suffixes from a word, leaving on the word "stem". Lemmatization is similar to stemming, but lemmatization is able to capture the underlying meaning of the word.

```
[39]: my_sent
```

```
[39]: TextBlob("The octopi went swimming in the dark ocean waters.")
```

```
[40]: # Find the index of 'swimming'
      my_sent.words.index('swimming')
```

```
[40]: 3
```

```
[41]: # Stemming
      print(my_sent.words[3].stem())
      print(my_sent.words[1].stem())
```

```
swim
octopi
```

```
[42]: # Lemmatization
      print(my_sent.words[3].lemmatize())
      print(my_sent.words[1].lemmatize())
```

```
swimming
octopus
```

```
[43]: care = TextBlob("caring")

      (
        care.words.stem(),
        care.words.lemmatize()
      )
```

```
[43]: (WordList(['care']), WordList(['caring']))
```

1.8 WordNet

```
[44]: my_sent
```

```
[44]: TextBlob("The octopi went swimming in the dark ocean waters.")
```

```
[45]: { my_sent.words[-2] : my_sent.words[-2].definitions }
```

```
[45]: {'ocean': ['a large body of water constituting a principal part of the
hydrosphere',
  'anything apparently limitless in quantity or volume']}
```

```
[49]: {"swimming", "tennis"} - set(my_sent.words)
```

```
[49]: {'tennis'}
```

1.9 Spelling (correcting)

```
[50]: my_bad_spelling = TextBlob('Helllo, today is my birfday.')
      my_bad_spelling.correct()
```

```
[50]: TextBlob("Hello, today is my birthday.")
```

1.10 Counting Words

```
[51]: my_cheer = TextBlob('Data science is the best, data science is the coolest.')
      my_cheer.words.count('data')
```

```
[51]: 2
```

```
[52]: my_cheer.word_counts
```

```
[52]: defaultdict(int,
                  {'data': 2,
                   'science': 2,
                   'is': 2,
                   'the': 2,
                   'best': 1,
                   'coolest': 1})
```

1.10.1 Your Turn

1. Create a TextBlob called `message` and set it equal to `Good morning, todayy is going to be a fantastic day!`.
2. Correct the spelling in your TextBlob and set it equal to a new variable called `message_sp`.
3. Find the index of the word `fantastic`.
4. Look up the definition of the word `fantastic`.
5. Stem and lemmatize the word `fantastic`.

[53]:
```
# Solution 1
message = TextBlob("Good morning, todayy is going to be a fantastic day!.")
message
```

[53]:
```
TextBlob("Good morning, todayy is going to be a fantastic day!.")
```

[54]:
```
# Solution 2
message_sp = message.correct()
message_sp
```

[54]:
```
TextBlob("Good morning, today is going to be a fantastic day!.")
```

[62]:
```
list(zip(message.words, message_sp.words ))
```

[62]:
```
[('Good', 'Good'),
 ('morning', 'morning'),
 ('todayy', 'today'),
 ('is', 'is'),
 ('going', 'going'),
 ('to', 'to'),
 ('be', 'be'),
 ('a', 'a'),
 ('fantastic', 'fantastic'),
 ('day', 'day')]
```

[55]:
```
[ (i,t) for i, t in enumerate(zip(message.words, message_sp.words )) if t[0] !=␣
 ↪t[1] ]
```

[55]:
```
[(2, ('todayy', 'today'))]
```

[61]:
```
# Solution 3
(
    message.index("fantastic"),
    message.words.index("fantastic")
)
```

[61]:
```
(38, 8)
```

[58]:
```
# Solution 4
TextBlob("fantastic").words[0].definitions
```

```
message.words[ message.words.index("fantastic") ].definitions
```

[58]:
```
['ludicrously odd',
 'extraordinarily good or great ; used especially as intensifiers',
 'fanciful and unrealistic; foolish',
 'existing in fancy only; - Nathaniel Hawthorne',
 'extravagantly fanciful in design, construction, appearance']
```

[59]:
```
# Solution 5
fan = message.words[ message.words.index("fantastic") ]
(
  fan.stem(),
  fan.lemmatize()
)
```

[59]:
```
('fantast', 'fantastic')
```

1.11 TextBlobs as Strings

TextBlobs act as strings, meaning you can use all of the normal string methods and you can index them as you would a string.

[63]:
```
my_cheer
```

[63]:
```
TextBlob("Data science is the best, data science is the coolest.")
```

[64]:
```
my_cheer[0:6]
```

[64]:
```
TextBlob("Data s")
```

[65]:
```
my_cheer.upper()
```

[65]:
```
TextBlob("DATA SCIENCE IS THE BEST, DATA SCIENCE IS THE COOLEST.")
```

[66]:
```
my_cheer.lower()
```

[66]:
```
TextBlob("data science is the best, data science is the coolest.")
```

1.12 n-grams

Overlapping lists of words.

[67]:
```
my_cheer
```

[67]:
```
TextBlob("Data science is the best, data science is the coolest.")
```

[68]:
```
my_cheer.words
```

```
[68]: WordList(['Data', 'science', 'is', 'the', 'best', 'data', 'science', 'is',
      'the', 'coolest'])
```

```
[69]: my_cheer.ngrams(n=3)
```

```
[69]: [WordList(['Data', 'science', 'is']),
       WordList(['science', 'is', 'the']),
       WordList(['is', 'the', 'best']),
       WordList(['the', 'best', 'data']),
       WordList(['best', 'data', 'science']),
       WordList(['data', 'science', 'is']),
       WordList(['science', 'is', 'the']),
       WordList(['is', 'the', 'coolest'])]
```

```
[70]: [ " ".join(i) for i in my_cheer.ngrams(n=3) ]
```

```
[70]: ['Data science is',
       'science is the',
       'is the best',
       'the best data',
       'best data science',
       'data science is',
       'science is the',
       'is the coolest']
```

```
[71]: my_cheer.split(",")
```

```
[71]: WordList(['Data science is the best', ' data science is the coolest.'])
```

```
[72]: my_cheer.words
```

```
[72]: WordList(['Data', 'science', 'is', 'the', 'best', 'data', 'science', 'is',
      'the', 'coolest'])
```

```
[74]: [ " ".join(i) for i in TextBlob("italian pop rock").ngrams(n=2) ]
```

```
[74]: ['italian pop', 'pop rock']
```

1.13 Translation

```
[93]: # %%capture
      !pip install googletrans==3.1.0a0 transformers sacremoses
```

```
Collecting googletrans==3.1.0a0
  Downloading googletrans-3.1.0a0.tar.gz (19 kB)
  Preparing metadata (setup.py) … done
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-
packages (4.44.2)
Requirement already satisfied: sacremoses in /usr/local/lib/python3.10/dist-
```

```
packages (0.1.1)
Requirement already satisfied: httpx==0.13.3 in /usr/local/lib/python3.10/dist-
packages (from googletrans==3.1.0a0) (0.13.3)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==3.1.0a0) (2024.8.30)
Requirement already satisfied: hstspreload in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==3.1.0a0) (2024.11.1)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==3.1.0a0) (1.3.1)
Requirement already satisfied: chardet==3.* in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==3.1.0a0) (3.0.4)
Requirement already satisfied: idna==2.* in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans==3.1.0a0) (2.10)
Requirement already satisfied: rfc3986<2,>=1.3 in
/usr/local/lib/python3.10/dist-packages (from
httpx==0.13.3->googletrans==3.1.0a0) (1.5.0)
Requirement already satisfied: httpcore==0.9.* in
/usr/local/lib/python3.10/dist-packages (from
httpx==0.13.3->googletrans==3.1.0a0) (0.9.1)
Requirement already satisfied: h11<0.10,>=0.8 in /usr/local/lib/python3.10/dist-
packages (from httpcore==0.9.*->httpx==0.13.3->googletrans==3.1.0a0) (0.9.0)
Requirement already satisfied: h2==3.* in /usr/local/lib/python3.10/dist-
packages (from httpcore==0.9.*->httpx==0.13.3->googletrans==3.1.0a0) (3.2.0)
Requirement already satisfied: hyperframe<6,>=5.2.0 in
/usr/local/lib/python3.10/dist-packages (from
h2==3.*->httpcore==0.9.*->httpx==0.13.3->googletrans==3.1.0a0) (5.2.0)
Requirement already satisfied: hpack<4,>=3.0 in /usr/local/lib/python3.10/dist-
packages (from h2==3.*->httpcore==0.9.*->httpx==0.13.3->googletrans==3.1.0a0)
(3.0.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.24.7)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.20,>=0.19 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-
```

```
packages (from transformers) (4.66.6)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from sacremoses) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from sacremoses) (1.4.2)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.23.2->transformers) (2024.10.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.23.2->transformers) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4.0)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.2.3)
Building wheels for collected packages: googletrans
  Building wheel for googletrans (setup.py) … done
  Created wheel for googletrans: filename=googletrans-3.1.0a0-py3-none-any.whl
size=16352
sha256=0257d1b32ad8e75f97424ed1263b576fa11d1229b7b84c40a51b1f87b67f9d0d
  Stored in directory: /root/.cache/pip/wheels/50/5d/3c/8477d0af4ca2b8b1308812c0
9f1930863caeebc762fe265a95
Successfully built googletrans
Installing collected packages: googletrans
  Attempting uninstall: googletrans
    Found existing installation: googletrans 3.0.0
    Uninstalling googletrans-3.0.0:
      Successfully uninstalled googletrans-3.0.0
Successfully installed googletrans-3.1.0a0
```

1.13.1 Google translate

[1]:
```
from googletrans import Translator
```

[2]:
```
translator = Translator()
```

[4]:
```
result = translator.translate(
    'Hello, how are you?',
    src='en',
    dest='es',
)
print(result.text)
```

```
¿Hola, cómo estás?
```

[5]:
```
result = translator.translate(
    'Hello, how are you?',
    src='en',
    dest='fr',
)
print(result.text)
```

```
Bonjour comment allez-vous?
```

[6]:
```
result = translator.translate(
    'Hello, how are you?',
    src='en',
    dest='ar',
)
print(result.text)
```

:[7]
```
result = translator.translate(
    'Hello, how are you?',
    src='en',
    dest='de',
)
print(result.text)
```

```
Hallo, wie geht es dir?
```

All in one ...

[8]:
```
langs = 'es fr ar de'.split()

for lang in langs:
  result = translator.translate(
      'Hello, how are you?',
      src='en',
      dest=lang,
  )
  print(result.text)
```

```
¿Hola, cómo estás?
Bonjour comment allez-vous?

Hallo, wie geht es dir?
```

1.13.2 Hugging Face Transformers (via pre-trained models)

[87]:
```
from transformers import MarianMTModel, MarianTokenizer
```

[88]:
```
# Load pre-trained MarianMT model
model_name = 'Helsinki-NLP/opus-mt-en-es'
model = MarianMTModel.from_pretrained(model_name)
tokenizer = MarianTokenizer.from_pretrained(model_name,␣
  ↪clean_up_tokenization_spaces=True )
```

```
text = "Hello, how are you?"
translated = model.generate(**tokenizer(text, return_tensors="pt",␣
 ↪padding=True))
result = tokenizer.decode(translated[0], skip_special_tokens=True)

print(result)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(

config.json:   0%|          | 0.00/1.47k [00:00<?, ?B/s]

pytorch_model.bin:   0%|          | 0.00/312M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/293 [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/44.0 [00:00<?, ?B/s]

source.spm:   0%|          | 0.00/802k [00:00<?, ?B/s]

target.spm:   0%|          | 0.00/826k [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/1.59M [00:00<?, ?B/s]

Hola, ¿cómo estás?
```

[89]:
```
# Load pre-trained MarianMT model
model_name = 'Helsinki-NLP/opus-mt-en-fr'
model = MarianMTModel.from_pretrained(model_name)
tokenizer = MarianTokenizer.from_pretrained(model_name,␣
 ↪clean_up_tokenization_spaces=True)

text = "Hello, how are you?"
translated = model.generate(**tokenizer(text, return_tensors="pt",␣
 ↪padding=True))
result = tokenizer.decode(translated[0], skip_special_tokens=True)

print(result)
```

```
config.json:   0%|          | 0.00/1.42k [00:00<?, ?B/s]

pytorch_model.bin:   0%|          | 0.00/301M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/293 [00:00<?, ?B/s]
```

```
tokenizer_config.json:   0%|          | 0.00/42.0 [00:00<?, ?B/s]

source.spm:   0%|          | 0.00/778k [00:00<?, ?B/s]

target.spm:   0%|          | 0.00/802k [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/1.34M [00:00<?, ?B/s]

Bonjour, comment allez-vous?
```

[90]:
```
# Load pre-trained MarianMT model
model_name = 'Helsinki-NLP/opus-mt-en-ar'
model = MarianMTModel.from_pretrained(model_name)
tokenizer = MarianTokenizer.from_pretrained(model_name,␣
 ↪clean_up_tokenization_spaces=True)

text = "Hello, how are you?"
translated = model.generate(**tokenizer(text, return_tensors="pt",␣
 ↪padding=True))
result = tokenizer.decode(translated[0], skip_special_tokens=True)

print(result)
```

```
config.json:   0%|          | 0.00/1.39k [00:00<?, ?B/s]

pytorch_model.bin:   0%|          | 0.00/308M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/293 [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/44.0 [00:00<?, ?B/s]

source.spm:   0%|          | 0.00/801k [00:00<?, ?B/s]

target.spm:   0%|          | 0.00/917k [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/2.12M [00:00<?, ?B/s]
```

:[91]
```
# Load pre-trained MarianMT model
model_name = 'Helsinki-NLP/opus-mt-en-de'
model = MarianMTModel.from_pretrained(model_name)
tokenizer = MarianTokenizer.from_pretrained(model_name,␣
 ↪clean_up_tokenization_spaces=True)

text = "Hello, how are you?"
translated = model.generate(**tokenizer(text, return_tensors="pt",␣
 ↪padding=True))
result = tokenizer.decode(translated[0], skip_special_tokens=True)

print(result)
```

```
config.json:   0%|          | 0.00/1.33k [00:00<?, ?B/s]
```

```
pytorch_model.bin:   0%|          | 0.00/298M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/293 [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/42.0 [00:00<?, ?B/s]

source.spm:   0%|          | 0.00/768k [00:00<?, ?B/s]

target.spm:   0%|          | 0.00/797k [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/1.27M [00:00<?, ?B/s]

Hallo, wie geht's?
```

```
[ ]:
```