nltk tokenize

```
import nltk
from nltk.tokenize import word_tokenize,sent_tokenize
```

```
nltk.download('punkt_tab')
text="Hello! This is raw text example. For tokenization."
tokens=word_tokenize(text)
print(tokens)
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
['Hello', '!', 'This', 'is', 'raw', 'text', 'example', '.', 'For', 'tokenization', '.']
```

```
import spacy
nlp=spacy.load('en_core_web_sm')
nlp_text=nlp(text)
tokens=[token.text for token in nlp_text]
print(tokens)
```

```
['Hello', '!', 'This', 'is', 'raw', 'text', 'example', '.', 'For', 'tokenization', '.']
```

```
import re
tokens=re.split("(?<=[.!?])\s+",text)
print(tokens)
```

```
['Hello!', 'This is raw text example.', 'For tokenization.']
```

Stop Words

```
#Stop Words
import nltk
nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
text="Hello! This is raw text example. For tokenization. This will also be an example of stemming using lancaster and porter stemmer. Find c
tokens=word_tokenize(text)
nltk.download('stopwords')
stop_words=set(stopwords.words('english'))
filtered_words=[word for word in tokens if word.lower() not in stop_words]
print(filtered_words)
```

```
['Hello', '!', 'raw', 'text', 'example', '.', 'tokenization', '.']
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Stemmer Snowball,Lancaster,Porter

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer,LancasterStemmer
from nltk.stem import SnowballStemmer
nltk.download('punkt_tab')
text="Hello! This is raw text example. For tokenization. This will also be an example of stemming using lancaster and porter stemmer. Find c
tokens=word_tokenize(text)
porter=PorterStemmer()
Lancaster=LancasterStemmer()
porter_stemmed=[porter.stem(t1) for t1 in tokens]
lancaster_stemmed=[Lancaster.stem(t1) for t1 in tokens]
print("Porter",porter_stemmed)
print("Lancaster",lancaster_stemmed)

Snowball=SnowballStemmer(language="english")
```

```
snow_stemmed=[Snowball.stem(t1) for t1 in tokens]
print("Snowball",snow_stemmed)
```

```
Porter ['hello', '!', 'thi', 'is', 'raw', 'text', 'exampl', '.', 'for', 'token', '.', 'thi', 'will', 'also', 'be', 'an', 'exampl', 'of',
Lancaster ['hello', '!', 'thi', 'is', 'raw', 'text', 'exampl', '.', 'for', 'tok', '.', 'thi', 'wil', 'also', 'be', 'an', 'exampl', 'of',
Snowball ['hello', '!', 'this', 'is', 'raw', 'text', 'exampl', '.', 'for', 'token', '.', 'this', 'will', 'also', 'be', 'an', 'exampl', '
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Lemmatization

```
import nltk
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.stem import WordNetLemmatizer
nltk.download('punkt_tab')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger_eng')

lemmetizer=WordNetLemmatizer()

def get_pos_tag(word):
  tag=pos_tag([word])[0][1][0].upper()
  tag_dict={"J":wordnet.ADJ,"N":wordnet.NOUN,"V":wordnet.VERB,"R":wordnet.ADV}
  return tag_dict.get(tag,wordnet.NOUN)

text="Hello! This is raw text example. For tokenization. This will also be an example of stemming using lancaster and porter stemmer. Find c

tokens=word_tokenize(text)

lemeetized=[lemmetizer.lemmatize(t1,get_pos_tag(t1)) for t1 in tokens]
print(tokens)
print(lemeetized)
tagged=[get_pos_tag(word) for word in tokens]

print(tokens,"\n",tagged)
```

```
['Hello', '!', 'This', 'is', 'raw', 'text', 'example', '.', 'For', 'tokenization', '.', 'This', 'will', 'also', 'be', 'an', 'example', '
['Hello', '!', 'This', 'be', 'raw', 'text', 'example', '.', 'For', 'tokenization', '.', 'This', 'will', 'also', 'be', 'an', 'example', '
['Hello', '!', 'This', 'is', 'raw', 'text', 'example', '.', 'For', 'tokenization', '.', 'This', 'will', 'also', 'be', 'an', 'example', '
 ['n', 'n', 'n', 'v', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'n', 'r', 'v', 'n', 'n', 'n', 'v', 'v', 'n', 'n', 'n', 'n', 'n', 'n', 'n',
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
```

N-Gram

```
from nltk import ngrams

text="Hello! This is raw text example. For tokenization. This will also be an example of stemming using lancaster and porter stemmer. Find c

tokens = word_tokenize(text)

# Bi-grams
bigrams = list(ngrams(tokens, 2))
print("Bigrams:", bigrams)

# Tri-grams
trigrams = list(ngrams(tokens, 3))
print("Trigrams:", trigrams)
```

```
Bigrams: [('Hello', '!'), ('!', 'This'), ('This', 'is'), ('is', 'raw'), ('raw', 'text'), ('text', 'example'), ('example', '.'), ('.', 'F
Trigrams: [('Hello', '!', 'This'), ('!', 'This', 'is'), ('This', 'is', 'raw'), ('is', 'raw', 'text'), ('raw', 'text', 'example'), ('text
```

Pos Tagging

```
# Bow
import nltk
from sklearn.feature_extraction.text import CountVectorizer

text=["This is sentence one","This is sentence two"]

countvector=CountVectorizer()

x=countvector.fit_transform(text)

print("Vocabulary: ",countvector.vocabulary_)
print("BOW: ",x.toarray())


from sklearn.feature_extraction.text import TfidfVectorizer

tfidfvec=TfidfVectorizer()
y=tfidfvec.fit_transform(text)

print("Vocabulary: ",tfidfvec.vocabulary_)
print("TF IDF: ",y.toarray())


from nltk import pos_tag
from nltk.tokenize import word_tokenize
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('punkt_tab')

text="Hello! This is raw text example. For tokenization. This will also be an example of stemming using lancaster and porter stemmer. Find ou

def pos_tagging(text):
  tokens=word_tokenize(text)
  pos_tags=pos_tag(tokens)
  return pos_tags

pos_text=pos_tagging(text)
print(pos_text)


import spacy
nlp=spacy.load("en_core_web_sm")
def pos1(text):
  word=nlp(text)
  return [(t.text,t.pos_)for t in word]

pos_text1=pos1(text)
print(pos_text1)

# for marathi use nlp=spacy.load("xx_ent_wiki_sm")
```

```
Vocabulary:  {'this': 3, 'is': 0, 'sentence': 2, 'one': 1, 'two': 4}
BOW:  [[1 1 1 1 0]
 [1 0 1 1 1]]
Vocabulary:  {'this': 3, 'is': 0, 'sentence': 2, 'one': 1, 'two': 4}
TF IDF:  [[0.44832087 0.63009934 0.44832087 0.44832087 0.        ]
 [0.44832087 0.         0.44832087 0.44832087 0.63009934]]
[('Hello', 'NN'), ('!', '.'), ('This', 'DT'), ('is', 'VBZ'), ('raw', 'JJ'), ('text', 'JJ'), ('example', 'NN'), ('.', '.'), ('For', 'IN')
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[('Hello', 'INTJ'), ('!', 'PUNCT'), ('This', 'PRON'), ('is', 'AUX'), ('raw', 'ADJ'), ('text', 'NOUN'), ('example', 'NOUN'), ('.', 'PUNCT
```

Double-click (or enter) to edit

CUSTOM NER

```
import spacy
from spacy.training.example import Example

nlp=spacy.blank("en")
ner=nlp.add_pipe("ner",last=True)
ner.add_label("PERSON")
ner.add_label("ORG")

TRAINING_DATA=[("Elon Musk founded SpaceX",{"entities":[(0,9,"PERSON"),(18,24,"ORG")]}),("Bill Gates founded Microsoft",{"entities":[(0,10,"

optimizer=nlp.begin_training()
for i in range(10):
  for text,annotation in TRAINING_DATA:
    example=Example.from_dict(nlp.make_doc(text),annotation)
    nlp.update([example],sgd=optimizer)
for text,annotation in TRAINING_DATA:
  tags=spacy.training.offsets_to_biluo_tags(nlp.make_doc(text),annotation["entities"])
  print(f"text {text}")
  print(f"tags {tags}")
nlp.to_disk("model")
```

```
text Elon Musk founded SpaceX
tags ['B-PERSON', 'L-PERSON', 'O', 'U-ORG']
text Bill Gates founded Microsoft
tags ['B-PERSON', 'L-PERSON', 'O', 'U-ORG']
```

```
model=spacy.load("model")
text="Elon Musk founded SpaceX Bill Gates"
doc=model(text)
for ent in doc.ents:
  print(ent.text,ent.label_)

# use spacy.blank("mr") for marathi
```

```
Elon Musk PERSON
SpaceX ORG
Bill Gates PERSON
```

Bow

```
import numpy as np
import nltk
from collections import Counter
nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize

text=["This is sentence one.","This is sentence two"]
tokens=[word_tokenize(word.lower()) for word in text]
vocab=set([word for token in tokens for word in token])
print(vocab)
def bow(text,vocabulary):
  return [text.count(word) for word in vocabulary]

bow_text=[bow(token,vocab) for token in tokens]
print(np.array(bow_text))
```

```
{'sentence', 'this', 'is', 'one', 'two', '.'}
[[1 1 1 1 0 1]
 [1 1 1 0 1 0]]
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

TF-IDF

```
import nltk
import numpy as np
from collections import Counter
from math import log
```

```
nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize

text=["This is sentence one.","This is sentence two"]
tokens=[word_tokenize(word.lower()) for word in text]
vocab=set([word for token in tokens for word in token])
print("Vocabulary",vocab)

def get_tf(text,vocabulary):
  tf_vectors=[text.count(word) for word in vocabulary]
  print("TF VECTOS",tf_vectors)
  return tf_vectors


def get_idf(vocabulary,docs):
  num_doc=len(docs)
  idf_vectors=[]
  for word in vocabulary:
    num_docs_with_word=sum(1 for doc in docs if word in doc)
    idf=log(num_doc/(1+num_docs_with_word))+1
    idf_vectors.append(idf)
  return idf_vectors

def gettfidf(text,vocabulary,idf_vectors):
  tf_vectors=get_tf(text,vocabulary)
  tfidfvectors=[tf*idf for tf,idf in zip(tf_vectors,idf_vectors)]
  return tfidfvectors


idf_vectors=get_idf(vocab,tokens)
print("IDF\n",idf_vectors)
tfidfvector=[gettfidf(token,vocab,idf_vectors) for token in tokens]
print("TFIDF\n",np.array(tfidfvector))
```

```
Vocabulary {'sentence', 'this', 'is', 'one', 'two', '.'}
IDF
 [0.5945348918918356, 0.5945348918918356, 0.5945348918918356, 1.0, 1.0, 1.0]
TF VECTOS [1, 1, 1, 1, 0, 1]
TF VECTOS [1, 1, 1, 0, 1, 0]
TFIDF
 [[0.59453489 0.59453489 0.59453489 1.         0.         1.        ]
 [0.59453489 0.59453489 0.59453489 0.         1.         0.        ]]
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

CoSine Similarity

```
from tkinter.constants import N
import nltk
from collections import Counter
from nltk.tokenize import word_tokenize
from sklearn.metrics.pairwise import cosine_similarity
nltk.download('punkt_tab')
text=["The Dog is barking","The cat is meowing"]

tokenized_word=[nltk.word_tokenize(texts.lower()) for texts in text]
vocabulary=set(word for text in tokenized_word for word in text)
print("Vocabulary: ",vocabulary)
def get_bow(word,vocabulary):
  return [word.count(text) for text in vocabulary]

bow_vectors=[get_bow(text,vocabulary) for text in tokenized_word]
print("BOW\n",np.array(bow_vectors))

bow_similarity=cosine_similarity([bow_vectors[0]],[bow_vectors[1]])[0][0]
print("BOW Similarity\n",bow_similarity)

tokens=[word_tokenize(word.lower()) for word in text]
vocab=set([word for token in tokens for word in token])
print("Vocabulary",vocab)

def get_tf(text,vocabulary):
  tf_vectors=[text.count(word) for word in vocabulary]
  print("TF VECTOS",tf_vectors)
  return tf_vectors
```

```
def get_idf(vocabulary,docs):
  num_doc=len(docs)
  idf_vectors=[]
  for word in vocabulary:
    num_docs_with_word=sum(1 for doc in docs if word in doc)
    idf=log(num_doc/(1+num_docs_with_word))+1
    idf_vectors.append(idf)
  return idf_vectors

def gettfidf(text,vocabulary,idf_vectors):
  tf_vectors=get_tf(text,vocabulary)
  tfidfvectors=[tf*idf for tf,idf in zip(tf_vectors,idf_vectors)]
  return tfidfvectors


idf_vectors=get_idf(vocab,tokens)
print("IDF\n",idf_vectors)
tfidfvector=[gettfidf(token,vocab,idf_vectors) for token in tokens]
print("TFIDF\n",np.array(tfidfvector))

bow_similarity=cosine_similarity([bow_vectors[0]],[tfidfvector[0]])[0][0]
print("Cosine Similarity of bow and tfidf: ",bow_similarity)
```

```
Vocabulary:  {'cat', 'is', 'dog', 'the', 'barking', 'meowing'}
BOW
 [[0 1 1 1 1 0]
 [1 1 0 1 0 1]]
BOW Similarity
 0.5
Vocabulary {'cat', 'is', 'dog', 'the', 'barking', 'meowing'}
IDF
 [1.0, 0.5945348918918356, 1.0, 0.5945348918918356, 1.0, 1.0]
TF VECTOS [0, 1, 1, 1, 1, 0]
TF VECTOS [1, 1, 0, 1, 0, 1]
TFIDF
 [[0.         0.59453489 1.         0.59453489 1.         0.        ]
 [1.         0.59453489 0.         0.59453489 0.         1.        ]]
Cosine Similarity of bow and tfidf:  0.9691576615919082
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Word Embedding

```
!pip install numpy==1.25.0 gensim==4.3.1
```

```
Requirement already satisfied: numpy==1.25.0 in /usr/local/lib/python3.11/dist-packages (1.25.0)
Requirement already satisfied: gensim==4.3.1 in /usr/local/lib/python3.11/dist-packages (4.3.1)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensim==4.3.1) (1.10.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim==4.3.1) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim==4.3.1) (1.17.2)
```

```
!pip install scipy==1.10.1
```

```
Requirement already satisfied: scipy==1.10.1 in /usr/local/lib/python3.11/dist-packages (1.10.1)
Requirement already satisfied: numpy<1.27.0,>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scipy==1.10.1) (1.25.0)
```

```
from gensim.models import Word2Vec


import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt_tab')

def get_model(text):
  tokens=[word_tokenize(word.lower()) for word in text]
  model=Word2Vec(tokens,vector_size=100,window=5,min_count=1,workers=4)
  return model

def use_model(model,text,top_n=5):
  similar_word=model.wv.most_similar(text,topn=top_n)
```

```
  print(f"Most Similar words to {text}")
  for w,score in similar_word:
    print(f"word: {w} Score: {score}")

text=["the dog barks","the cats meows","the elephant trumps"]
model=get_model(text)
use_model(model,"dog")
```

```
Most Similar words to dog
word: elephant Score: 0.1459505707025528
word: meows Score: 0.041577354073524475
word: cats Score: 0.03476494178175926
word: barks Score: 0.019152268767356873
word: the Score: 0.01613469421863556
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Text Classifier

```
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report


def modelclass(x,y):
  x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
  vectorizer=CountVectorizer()
  x_train_vector=vectorizer.fit_transform(x_train)
  x_test_vector=vectorizer.transform(x_test)
  classifier=MultinomialNB()
  classifier.fit(x_train_vector,y_train)
  y_pred=classifier.predict(x_test_vector)
  print(classification_report(y_test,y_pred))
  return vectorizer,classifier

def use_model(text,vectorizer,classifier):
  x_vector=vectorizer.transform([text])
  prediction=classifier.predict(x_vector)
  print(prediction)
  return prediction[0]


x=["This product is good","This product is bad","This restaurant is desent","This book in not so good","This is amazing"]
y=["positive","negative","positive","negative","positive"]

vectorizer,classifier=modelclass(x,y)
new_text="This city is very good"
prediction=use_model(new_text,vectorizer,classifier)
print(f"for {new_text} prediction is {prediction}")
```

```
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00       1.0
    positive       0.00      0.00      0.00       0.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0

['positive']
for This city is very good prediction is positive
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defined and being
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Sentiment Analusis

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
import pandas as pd
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```

```
def getsentiments(text):
  sia=SentimentIntensityAnalyzer()
  sentiment_score=sia.polarity_scores(text)
  if sentiment_score["compound"]>=0.1:
    sentiment="Positive"
  elif sentiment_score["compound"]<=-0.1:
    sentiment="Negative"
  else:
    sentiment="Neutral"
  return sentiment,sentiment_score

def anaylzesentiment(text):
  results=[]
  for t in text:
    sentiment,sentiment_score=getsentiments(t)
    results.append({"Text: ":t,"Sentiment: ":sentiment,"pos: ":sentiment_score["pos"]
                    ,"neg: ":sentiment_score["neg"],"neu: ":sentiment_score["neu"],"Compound: ":sentiment_score["compound"]})
  df=pd.DataFrame(results)
  return df

text=["This product is good","This product is bad","This restaurant is desent","This book in not so good","This is amazing"]

df=anaylzesentiment(text)
print(df)
```

```
                      Text:  Sentiment:   pos:   neg:   neu:   Compound:
0       This product is good    Positive  0.492  0.000  0.508      0.4404
1        This product is bad    Negative  0.000  0.538  0.462     -0.5423
2  This restaurant is desent     Neutral  0.000  0.000  1.000      0.0000
3   This book in not so good    Negative  0.000  0.377  0.623     -0.4640
4            This is amazing    Positive  0.655  0.000  0.345      0.5859
```

Text Summarization

```
! pip install transformers
! pip install torch
from transformers import pipeline
```

Show hidden output

```
def Summarization(text,maxlength=150,minlength=50):
  pipeline1=pipeline("summarization",model="facebook/bart-large-cnn")
  summary=pipeline1(text,max_length=maxlength,min_length=minlength,do_sample=False)
  return summary[0]["summary_text"]

long_text="""The story centers around a girl named Little Red Riding Hood, named after her red hooded cape that she wears. The girl walks thr
A stalking wolf wants to eat the girl and the food in the basket. After he inquires as to where she is going, he suggests that she pick some
Gustave Doré's engraving of the scene "She was astonished to see how her grandmother looked."
When Riding Hood arrives, she notices the strange appearance of her "grandmother". After some back and forth, Riding Hood comments on the wol
Sanitized versions of the story have the grandmother locked in the closet rather than being eaten (and also having the wolf eat the food Litt

summary=Summarization(long_text)
print("Original text length: ",len(long_text))
print("Summary text length: ",len(summary))
print(summary)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: 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 secre
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: 100%                                                 1.58k/1.58k [00:00<00:00, 64.6kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfo
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
model.safetensors: 100%                                           1.63G/1.63G [00:21<00:00, 129MB/s]
generation_config.json: 100%                                        363/363 [00:00<00:00, 7.28kB/s]
vocab.json: 100%                                                  899k/899k [00:00<00:00, 10.3MB/s]
merges.txt: 100%                                                  456k/456k [00:00<00:00, 7.37MB/s]
tokenizer.json: 100%                                              1.36M/1.36M [00:00<00:00, 5.47MB/s]
Device set to use cuda:0
Original text length:  1461
Summary text length:  375
The story centers around a girl named Little Red Riding Hood, named after her red hooded cape that she wears. The girl walks through the
```

Start coding or generate with AI.