"""Text preprocessing and operations"""

'''

Process -

Tokenize

POS Tagging

Applying Naive Bayes Classifier

'''

import nltk

nltk.download()

data="In computer science, artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans. Colloquially, the term 'artificial intelligence' is often used to describe machines (or computers) that mimic 'cognitive' functions that humans associate with other human minds, such as learning and problem solving. As machines become increasingly capable, tasks considered to require 'intelligence' are often removed from the definition of AI, a phenomenon known as the AI effect.[2] A quip in Tesler's Theorem says, 'AI is whatever hasn't been done yet'.[3] For instance, optical character recognition is frequently excluded from things considered to be AI, having become a routine technology.[4] Modern machine capabilities generally classified as AI include successfully understanding human speech,[5] competing at the highest level in strategic game systems (such as chess and Go),[6] autonomously operating cars, intelligent routing in content delivery networks, and military simulations. Artificial intelligence can be classified into three different types of systems: analytical, human-inspired, and humanized artificial intelligence.[7] Analytical AI has only characteristics consistent with cognitive intelligence; generating cognitive representation of the world and using learning based on past experience to inform future decisions. Human-inspired AI has elements from cognitive and emotional intelligence; understanding human emotions, in addition to cognitive elements, and considering them in their decision making. Humanized AI shows characteristics of all types of competencies (i.e., cognitive, emotional, and social intelligence), is able to be self-conscious and is self-aware in interactions with others."

print(data)

#Tokenize

from nltk import sent\_tokenize

sent\_tokenize(data) #Check this

#Separates lines of the paragraph

from nltk import word\_tokenize

word\_tokenize(data)

#Make word tokens

#Stem

from nltk.stem import PorterStemmer

ps=PorterStemmer()

ps.stem('Cars')

ps.stem('boys')

ps.stem('goes')

#Finds the root words

#We cannot find the root word of goes, so we use lemmatizers.

#ls.stem('goes')

#ls.stem('does')

#Lemmatizers

from nltk import stem

wd=stem.WordNetLemmatizer()

wd.lemmatize('goes')

wd.lemmatize('cars','n') #Lemmatize as a noun

wd.lemmatize('went','v') #Lemmatize as a verb

wd.lemmatize('is','v') #Lemmatize as a verb

"""Text Similarity (Cosine Similarity)"""

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

corpus = ["In a country like India, where a mass of population lives in the villages", "there are times when parents have work to take care of and cannot keep an eye on their children all day round", "In many cases, when even the females in houses have to go to work, but cannot always carry their children along with them", "they have no other choice but to leave them back at their homes. Due to these reasons and more, there has been a rapid increase in security risks of children recently"]

vector.fit(corpus)

print(vector.transform(["country population mass times parents parents parents hello"]).toarray())

print(vector.transform(["country population mass times parents parents parents hello"]))

vector = CountVectorizer(binary = False)

corpus = ["In a country like India, where a mass of population lives in the villages", "there are times when parents have work to take care of and cannot keep an eye on their children all day round", "In many cases, when even the females in houses have to go to work, but cannot always carry their children along with them", "they have no other choice but to leave them back at their homes. Due to these reasons and more, there has been a rapid increase in security risks of children recently"]

vector.fit(corpus)

v1 = vector.transform(["country population mass times parents parents parents hello"]).toarray()

print(v1)

v2 = vector.transform(["hello world country population so scared"]).toarray()

print(vector.transform(["country population mass times parents parents parents hello"]))

similarity = cosine\_similarity(v1, v2)

print(similarity)

"""Translating sentences"""

import textblob

from textblob import TextBlob

data=TextBlob('Hello everyone! Hope you are enjoying machine learning.')

data.translate(to='hi') #Translate to hindi language

data.translate(to='fr') #Translate to french language

data.translate(to='ar') #Translate to arabian language

data.translate(to='bn') #Translate to bangla language

"""Sentiment Analysis"""

'''

Polarity -

tends towards:

emotional negative(-2)

Rational negative(-1)

neutral(0)

Rational positive(+1)

Emotional positive(+2)

Subjectivity -

tends towards:

Personal opinion(+1)

General fact(0)

'''

data=TextBlob('The movie was good. The cinematography was very good.')

data.sentiment

data=TextBlob('The auto was uncomfortable. The driver was rude.')

data.sentiment

data=TextBlob('The task was done in a right way. Hard work pays off.')

data.sentiment

"""Spelling correction"""

data=TextBlob('I havv two cars')

print(data.correct())

data=TextBlob('He seeme to be nic')

print(data.correct())

data=TextBlob('The sceneuri is really beutifu')

print(data.correct())

"""Categorizing the sentences into some category, based on the sentences"""

'Model training'

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import fetch\_20newsgroups

data=fetch\_20newsgroups()

data.target\_names

categories=['alt.atheism',

'comp.graphics',

'comp.os.ms-windows.misc',

'comp.sys.ibm.pc.hardware',

'comp.sys.mac.hardware',

'comp.windows.x',

'misc.forsale',

'rec.autos',

'rec.motorcycles',

'rec.sport.baseball',

'rec.sport.hockey',

'sci.crypt',

'sci.electronics',

'sci.med',

'sci.space',

'soc.religion.christian',

'talk.politics.guns',

'talk.politics.mideast',

'talk.politics.misc',

'talk.religion.misc']

#training the data on these categories

train=fetch\_20newsgroups(subset='train',categories=categories)

#testing the data on these categories

test=fetch\_20newsgroups(subset='test',categories=categories)

print(train.data[5])

print(len(train.data))

print(len(test.data))

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import make\_pipeline

from sklearn.feature\_extraction.text import TfidfVectorizer

#Creating model

model=make\_pipeline(TfidfVectorizer(),MultinomialNB())

#Training the models

model.fit(train.data,train.target)

#Creating label for test data

labels=model.predict(test.data)

#Predicting categories

def predict\_category(s,train=train,model=model):

pred=model.predict([s])

return train.target\_names[pred[0]]

predict\_category('The US has thousands of people killed each year by the militants. There are so many people moving into the US as immigrants.')

predict\_category('Audi has better technology than BMW')

predict\_category('Hockey is the national sport of India')

predict\_category('Windows operating system is widely used across the world')

predict\_category('Scientists have made the discovery of a new planet in another galaxy')

""""Chatbot"""

"""

from chatterbot import ChatBot

from chatterbot.trainers import ChatterBotCorpusTrainer

bot=ChatBot('friday')

ChatBot.train(['What is your name?','My name is Sophie'])

trainer=ChatterBotCorpusTrainer(bot)

trainer.train('chatterbot.corpus.english')

def chat(s):

print(bot.get\_response(s))

return

chat('who are you?')

chat('who is spiderman')

chat('I am a failure')

"""

#importing libraries

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize, sent\_tokenize

import bs4 as BeautifulSoup

import urllib.request

#fetching the content from the URL

fetched\_data = urllib.request.urlopen('https://en.wikipedia.org/wiki/20th\_century')

article\_read = fetched\_data.read()

#parsing the URL content and storing in a variable

article\_parsed = BeautifulSoup.BeautifulSoup(article\_read,'html.parser')

#returning <p> tags

paragraphs = article\_parsed.find\_all('p')

article\_content = ''

#looping through the paragraphs and adding them to the variable

for p in paragraphs:

article\_content += p.text

def \_create\_dictionary\_table(text\_string) -> dict:

#removing stop words

stop\_words = set(stopwords.words("english"))

words = word\_tokenize(text\_string)

#reducing words to their root form

stem = PorterStemmer()

#creating dictionary for the word frequency table

frequency\_table = dict()

for wd in words:

wd = stem.stem(wd)

if wd in stop\_words:

continue

if wd in frequency\_table:

frequency\_table[wd] += 1

else:

frequency\_table[wd] = 1

return frequency\_table

def \_calculate\_sentence\_scores(sentences, frequency\_table) -> dict:

#algorithm for scoring a sentence by its words

sentence\_weight = dict()

for sentence in sentences:

sentence\_wordcount = (len(word\_tokenize(sentence)))

sentence\_wordcount\_without\_stop\_words = 0

for word\_weight in frequency\_table:

if word\_weight in sentence.lower():

sentence\_wordcount\_without\_stop\_words += 1

if sentence[:7] in sentence\_weight:

sentence\_weight[sentence[:7]] += frequency\_table[word\_weight]

else:

sentence\_weight[sentence[:7]] = frequency\_table[word\_weight]

sentence\_weight[sentence[:7]] = sentence\_weight[sentence[:7]] / sentence\_wordcount\_without\_stop\_words

return sentence\_weight

def \_calculate\_average\_score(sentence\_weight) -> int:

#calculating the average score for the sentences

sum\_values = 0

for entry in sentence\_weight:

sum\_values += sentence\_weight[entry]

#getting sentence average value from source text

average\_score = (sum\_values / len(sentence\_weight))

return average\_score

def \_get\_article\_summary(sentences, sentence\_weight, threshold):

sentence\_counter = 0

article\_summary = ''

for sentence in sentences:

if sentence[:7] in sentence\_weight and sentence\_weight[sentence[:7]] >= (threshold):

article\_summary += " " + sentence

sentence\_counter += 1

return article\_summary

def \_run\_article\_summary(article):

#creating a dictionary for the word frequency table

frequency\_table = \_create\_dictionary\_table(article)

#tokenizing the sentences

sentences = sent\_tokenize(article)

#algorithm for scoring a sentence by its words

sentence\_scores = \_calculate\_sentence\_scores(sentences, frequency\_table)

#getting the threshold

threshold = \_calculate\_average\_score(sentence\_scores)

#producing the summary

article\_summary = \_get\_article\_summary(sentences, sentence\_scores, 1.5 \* threshold)

return article\_summary

if \_\_name\_\_ == '\_\_main\_\_':

summary\_results = \_run\_article\_summary(article\_content)

print(summary\_results)