Name : Bodhisatya Ghosh
Class : CSE DS
UID : 2021700026
Subject : NLP
Experiment number : 3

Aim:

1. Calculate bigrams from a given corpus , display bigram probability table and calculate probability of a sentence.
2. To apply add-one smoothing on sparse bigram table

In [ ]:
```
from nltk import bigrams
import numpy as np
import pandas as pd
```

1. Load corpus

In [ ]:
```
allText = pd.read_csv('../exp 2/reviews.csv',usecols=['review'])
corpus = allText['review'][0] + ' ' +allText['review'][1]
```

2. Perform preprocessing. (Covert data to lowercase, and remove punctuation marks and stop words to remove noise. Use the eos tag to mark the beginning and end of the sentence.)

In [ ]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize,sent_tokenize
import re
```

In [ ]:
```
#Remove html tags
corpus = corpus.replace('<br />',' ')
```

In [ ]:
```
#Remove stop words
def remove_stopwords_function(corpus):
    remove_stopwords = []
```

```
    for word in corpus.split():
        if(word not in stopwords.words()):
            remove_stopwords.append(word)
    return " ".join(remove_stopwords)
```

In [ ]:
```
corpus = remove_stopwords_function(corpus)
corpus
```

Out[ ]:
```
'One reviewers mentioned watching 1 Oz episode hooked. They right, happened me. The struck Oz brutality unflinching scenes vi
olence, set word GO. Trust me, show faint hearted timid. This show pulls punches drugs, sex violence. Its hardcore, classic w
ord. It called OZ nickname Oswald Maximum Security State Penitentary. It focuses Emerald City, experimental section prison ce
lls glass fronts inwards, privacy high agenda. Em City home many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, I
rish more....so scuffles, death stares, dodgy dealings shady agreements away. I appeal show due fact shows dare. Forget prett
y pictures painted mainstream audiences, forget charm, forget romance...OZ mess around. The episode I struck nasty surreal, I
I ready it, I watched more, I developed taste Oz, accustomed high levels graphic violence. Not violence, injustice (crooked g
uards who\'ll sold nickel, inmates who\'ll kill order away it, mannered, middle class inmates turned prison bitches due lack
street skills prison experience) Watching Oz, comfortable uncomfortable viewing....thats touch darker side. A wonderful produ
ction. The filming technique unassuming- old-time-BBC fashion comforting, discomforting, realism entire piece. The actors ext
remely chosen- Michael Sheen "has polari" voices pat too! You seamless editing guided references Williams\' diary entries, wo
rth watching terrificly written performed piece. A masterful production great master\'s comedy life. The realism home things:
fantasy guard which, traditional \'dream\' techniques remains solid disappears. It plays knowledge senses, particularly scene
s concerning Orton Halliwell sets (particularly flat Halliwell\'s murals decorating surface) terribly done.'
```

In [ ]:
```
#Adding tags for eos and sos
def add_tags(corpus):
    tagged_corpus_list = []
    for sentence in sent_tokenize(corpus):
        tagged_corpus_list.append('SOSTOKEN')
        for word in sentence.split():
            tagged_corpus_list.append(word.lower())
        tagged_corpus_list.append('EOSTOKEN')

    tagged_corpus = " ".join(tagged_corpus_list)
    tagged_corpus = re.sub(r"[^A-Za-z0-9<>/s']"," ",tagged_corpus)
    tagged_corpus = re.sub(r"\s+"," ",tagged_corpus)

    return tagged_corpus

tagged_corpus = add_tags(corpus)
```

```
tagged_corpus_list = tagged_corpus.split()
tagged_corpus
```

Out[ ]:
```
"SOSTOKEN one reviewers mentioned watching 1 oz episode hooked EOSTOKEN SOSTOKEN they right happened me EOSTOKEN SOSTOKEN the
struck oz brutality unflinching scenes violence set word go EOSTOKEN SOSTOKEN trust me show faint hearted timid EOSTOKEN SOST
OKEN this show pulls punches drugs sex violence EOSTOKEN SOSTOKEN its hardcore classic word EOSTOKEN SOSTOKEN it called oz ni
ckname oswald maximum security state penitentary EOSTOKEN SOSTOKEN it focuses emerald city experimental section prison cells
glass fronts inwards privacy high agenda EOSTOKEN SOSTOKEN em city home many aryans muslims gangstas latinos christians itali
ans irish more so scuffles death stares dodgy dealings shady agreements away EOSTOKEN SOSTOKEN i appeal show due fact shows d
are EOSTOKEN SOSTOKEN forget pretty pictures painted mainstream audiences forget charm forget romance oz mess around EOSTOKEN
SOSTOKEN the episode i struck nasty surreal i i ready it i watched more i developed taste oz accustomed high levels graphic v
iolence EOSTOKEN SOSTOKEN not violence injustice crooked guards who'll sold nickel inmates who'll kill order away it mannered
middle class inmates turned prison bitches due lack street skills prison experience watching oz comfortable uncomfortable vie
wing thats touch darker side EOSTOKEN SOSTOKEN a wonderful production EOSTOKEN SOSTOKEN the filming technique unassuming old
time bbc fashion comforting discomforting realism entire piece EOSTOKEN SOSTOKEN the actors extremely chosen michael sheen ha
s polari voices pat too EOSTOKEN SOSTOKEN you seamless editing guided references williams' diary entries worth watching terri
ficly written performed piece EOSTOKEN SOSTOKEN a masterful production great master's comedy life EOSTOKEN SOSTOKEN the reali
sm home things fantasy guard which traditional 'dream' techniques remains solid disappears EOSTOKEN SOSTOKEN it plays knowled
ge senses particularly scenes concerning orton halliwell sets particularly flat halliwell's murals decorating surface terribl
y done EOSTOKEN"
```

Tokenize the data.

In [ ]:
```
global_tokenizer = word_tokenize(tagged_corpus)
```

Generate bigrams and unigram freq table

In [ ]:
```
# unigram_data = {}
# bigram_data = {}

# for ind in word_tokenize(tagged_corpus):
#     unigram_data[ind] = 0
#     # bigram_data[ind] = {}
#     for col in word_tokenize(tagged_corpus):
#         bigram_data[ind][col] = 0
```

In [ ]:
```
def get_bigram_freq_table(tagged_corpus,tokenizer,addOne):
    default_freq = 0
```

```
    if(addOne):
        default_freq += 1

    unigram_data = {}
    bigram_data = {}
    tagged_corpus_list = tagged_corpus.split()

    for ind in tokenizer:
        unigram_data[ind] = default_freq
        bigram_data[ind] = {}
        for col in tokenizer:
            bigram_data[ind][col] = default_freq

    for i in range(0,len(tagged_corpus_list)):
        if(tagged_corpus_list[i] not in tokenizer or tagged_corpus_list[i-1] not in tokenizer):
            continue
        unigram_data[tagged_corpus_list[i]] += 1
        bigram_data[tagged_corpus_list[i]][tagged_corpus_list[i-1]] += 1

    return pd.Series(unigram_data),pd.DataFrame(bigram_data)
```

In [ ]:
```
unigram_table, bigram_table = get_bigram_freq_table(tagged_corpus, global_tokenizer,addOne=False)
```

Generate bi-gram probability table.

In [ ]:
```
def get_bigram_probability_table(tagged_corpus,tokenizer,addOne):
    unigram_table,bigram_freq_table = get_bigram_freq_table(tagged_corpus,tokenizer,addOne)

    bigram_prob_table = bigram_freq_table.copy(deep=True)
    for col in bigram_prob_table.columns:
        bigram_prob_table[col] = bigram_prob_table[col]/unigram_table[col]

    return unigram_table/unigram_table.sum() ,bigram_prob_table
```

In [ ]:
```
def get_probability_of_bigram(prev_word,curr_word,tagged_corpus,tokenizer):
    prob_table = get_bigram_probability_table(tagged_corpus,tokenizer)
    return prob_table[curr_word][prev_word]
```

Input text to calculate probability.

In [ ]:
```
def get_probability(text,training_text,addOne):

    text = add_tags(remove_stopwords_function(text))
    training_text = add_tags(remove_stopwords_function(training_text))
    tokenizer = word_tokenize(training_text)
    _, bigram_prob = get_bigram_probability_table(training_text,tokenizer,addOne)
    text_list = text.split()
    probability = 1
    # print(training_text)
    i=0
    while i < len(text_list):
        # print(text_list[i] + ' '+str(probability))
        if(text_list[i] not in tokenizer or text_list[i-1] not in tokenizer or bigram_prob[text_list[i]][text_list[i-1]] <= 0)
            # print("Not in tokenizer")
            probability = probability
        else:
            probability = probability * bigram_prob[text_list[i]][text_list[i-1]]
            # print("Found probability!")
        i+=1
    return probability,bigram_prob
```

In [ ]:
```
# input_text = input("Enter sentence to calculate probability: ")
```

In [ ]:
```
prob,bigram_probs = get_probability("I want an ice-cream so bad. I cant even explain how much i love ice cream.",allText['revi
```

In [ ]:
```
print("Probability of sentence is {} on basis of given text corpus".format(prob))
```

```
Probability of sentence is 0.015625 on basis of given text corpus
```

Applying add one smoothing

In [ ]:
```
prob,bigram_probs = get_probability("I want an ice-cream so bad. I cant even explain how much i love ice cream.",allText['revi
```

In [ ]:
```
print("Probability of sentence is {} on basis of given text corpus and after add-one smoothing".format(prob))
```

```
Probability of sentence is 0.04938271604938271 on basis of given text corpus and after add-one smoothing
```