In [1]:
```python
import csv # read from file
import math # exponentiation
import string # string processing
import nltk # get corpus
import time # efficiency
import heapq # 'top k'
import pandas as pd # csv processing
import numpy as np # numerical processing
from operator import itemgetter # sorting
from english_words import english_words_set # extra words
```

In [2]:
```python
nltk.download('words')
from nltk.corpus import words
```
```
[nltk_data] Downloading package words to
[nltk_data]     /Users/neelnarayan/nltk_data...
[nltk_data]   Package words is already up-to-date!
```

In [3]:
```python
df = pd.read_csv("True.csv")

words = []
words.append('<s>')
for i in range(len(df) - 1):
    for word in df.iloc[i][1].split():
        words.append(word.lower())
        if '.' in word and word.index('.') == (len(word) - 1):
            words.append('</s>')

print(len(words))
```
```
8579806
```

In [4]:
```python
# print sample article
print(df.iloc[1][1])
```
```
WASHINGTON (Reuters) - Transgender people will be allowed for the first time to enlist in the U.S. military starting
on Monday as ordered by federal courts, the Pentagon said on Friday, after President Donald Trump’s administration de
cided not to appeal rulings that blocked his transgender ban. Two federal appeals courts, one in Washington and one i
n Virginia, last week rejected the administration’s request to put on hold orders by lower court judges requiring the
military to begin accepting transgender recruits on Jan. 1. A Justice Department official said the administration wil
l not challenge those rulings. “The Department of Defense has announced that it will be releasing an independent stud
y of these issues in the coming weeks. So rather than litigate this interim appeal before that occurs, the administra
tion has decided to wait for DOD’s study and will continue to defend the president’s lawful authority in District Cou
rt in the meantime,” the official said, speaking on condition of anonymity. In September, the Pentagon said it had cr
eated a panel of senior officials to study how to implement a directive by Trump to prohibit transgender individuals
from serving. The Defense Department has until Feb. 21 to submit a plan to Trump. Lawyers representing currently-serv
ing transgender service members and aspiring recruits said they had expected the administration to appeal the rulings
to the conservative-majority Supreme Court, but were hoping that would not happen. Pentagon spokeswoman Heather Babb
said in a statement: “As mandated by court order, the Department of Defense is prepared to begin accessing transgende
r applicants for military service Jan. 1. All applicants must meet all accession standards.” Jennifer Levi, a lawyer
with gay, lesbian and transgender advocacy group GLAD, called the decision not to appeal “great news.” “I’m hoping it
means the government has come to see that there is no way to justify a ban and that it’s not good for the military or
our country,” Levi said. Both GLAD and the American Civil Liberties Union represent plaintiffs in the lawsuits filed
against the administration. In a move that appealed to his hard-line conservative supporters, Trump announced in July
that he would prohibit transgender people from serving in the military, reversing Democratic President Barack Obama’s
policy of accepting them. Trump said on Twitter at the time that the military “cannot be burdened with the tremendous
medical costs and disruption that transgender in the military would entail.” Four federal judges - in Baltimore, Wash
ington, D.C., Seattle and Riverside, California - have issued rulings blocking Trump’s ban while legal challenges to
the Republican president’s policy proceed. The judges said the ban would likely violate the right under the U.S. Cons
titution to equal protection under the law. The Pentagon on Dec. 8 issued guidelines to recruitment personnel in orde
r to enlist transgender applicants by Jan. 1. The memo outlined medical requirements and specified how the applicant
s’ sex would be identified and even which undergarments they would wear. The Trump administration previously said in
legal papers that the armed forces were not prepared to train thousands of personnel on the medical standards needed
to process transgender applicants and might have to accept “some individuals who are not medically fit for service.”
The Obama administration had set a deadline of July 1, 2017, to begin accepting transgender recruits. But Trump’s def
ense secretary, James Mattis, postponed that date to Jan. 1, 2018, which the president’s ban then put off indefinitel
y. Trump has taken other steps aimed at rolling back transgender rights. In October, his administration said a federa
l law banning gender-based workplace discrimination does not protect transgender employees, reversing another Obama-e
ra position. In February, Trump rescinded guidance issued by the Obama administration saying that public schools shou
ld allow transgender students to use the restroom that corresponds to their gender identity.
```

In [5]:
```python
def create_bigrams(words):
    bigrams = []
    bigrams_freq = {}
    word_freq = {}
    for i in range(len(words) - 1):
        bigram = (words[i], words[i + 1])
        if i < len(words) - 1:
            bigrams.append(bigram)
            if bigram in bigrams_freq:
                bigrams_freq[bigram] += 1
            else:
                bigrams_freq[bigram] = 1
        if words[i] in word_freq:
            word_freq[words[i]] += 1
        else:
            word_freq[words[i]] = 1
    return bigrams, word_freq, bigrams_freq
```

In [6]:
```python
bigrams, word_freq, bigrams_freq = create_bigrams(words)
```

In [7]:
```python
def calculate_bigrams_probability(bigrams, word_freq, bigrams_freq):
    probabilities = {}
    for bigram in bigrams:
        word1 = bigram[0]
        probabilities[bigram] = math.log((bigrams_freq[bigram] + 1) / (word_freq[word1] + 8579806))
    return probabilities
```

In [8]:
```python
bigram_probabilities = calculate_bigrams_probability(bigrams, word_freq, bigrams_freq)
```

In [9]:
```python
def calculate_unknown_bigrams_probability(bigram):
    if bigram[0] not in word_freq:
        word_freq[bigram[0]] = 1
    bigram_probabilities[bigram] = math.log(1 / (word_freq[bigram[0]] + 8579806))
```

In [10]:
```python
bad_bigram = ('trump', 'donald')
if bad_bigram not in bigrams:
    calculate_unknown_bigrams_probability(bad_bigram)
print(bigram_probabilities[bad_bigram])
good_bigram = ('donald', 'trump')
print(bigram_probabilities[good_bigram])
```
```
-15.969165271621279
-7.203158871411385
```

In [11]:
```python
# prints most common bigram
print(max(bigram_probabilities, key = bigram_probabilities.get))

# prints top 25 bigrams from the article with the largest probabilities
largest_25 = heapq.nlargest(25, bigram_probabilities, key = bigram_probabilities.get)
for l in largest_25:
    print(l, ": ", bigram_probabilities[l])
```
```
('of', 'the')
('of', 'the') :  -5.216751290905708
('</s>', 'the') :  -5.227535124695011
('in', 'the') :  -5.3671079102834485
('to', 'the') :  -5.984616689707037
('said.', '</s>') :  -5.987726681362507
('(reuters)', '-') :  -6.003846840240963
('in', 'a') :  -6.176614412566284
('on', 'the') :  -6.2631386141651815
('for', 'the') :  -6.333667049584559
('the', 'united') :  -6.466039735431085
('and', 'the') :  -6.583244863306866
('with', 'the') :  -6.653984591211452
('the', 'u.s.') :  -6.663033999578238
('at', 'the') :  -6.685055012917045
('by', 'the') :  -6.715700917703136
('said', 'on') :  -6.757655965479918
('to', 'be') :  -6.82078160220548
('that', 'the') :  -6.835175007126235
('</s>', 'in') :  -6.880772222711231
('of', 'a') :  -6.882181824725299
('from', 'the') :  -6.886744683121814
('said', 'the') :  -6.8921399083808375
('united', 'states') :  -6.915384600964873
('</s>', 'he') :  -6.959703707834659
('said', 'in') :  -6.9872111545108755
```

In [12]:
```python
# test sentence
original_sentence = "The former president, Donald Frump, lived in the White House."
```

In [13]:
```python
def format_sentence(sentence):
    return sentence.translate(str.maketrans('', '', string.punctuation)).lower()

sentence = format_sentence(original_sentence)
print(sentence)
```
```
the former president donald frump lived in the white house
```

In [14]:
```python
sentence_words = sentence.split()
s_bigrams, s_word_freq, s_bigrams_freq = create_bigrams(sentence_words)
```

In [15]:
```python
print(s_bigrams)
```
```
[('the', 'former'), ('former', 'president'), ('president', 'donald'), ('donald', 'frump'), ('frump', 'lived'), ('live
d', 'in'), ('in', 'the'), ('the', 'white'), ('white', 'house')]
```

In [16]:
```python
print(s_word_freq)
```
```
{'the': 2, 'former': 1, 'president': 1, 'donald': 1, 'frump': 1, 'lived': 1, 'in': 1, 'white': 1}
```

In [17]:
```python
print(s_bigrams_freq)
```
```
{('the', 'former'): 1, ('former', 'president'): 1, ('president', 'donald'): 1, ('donald', 'frump'): 1, ('frump', 'liv
ed'): 1, ('lived', 'in'): 1, ('in', 'the'): 1, ('the', 'white'): 1, ('white', 'house'): 1}
```

In [18]:
```python
s_bigrams_probabilities = {}
for bigram in s_bigrams:
    if bigram not in bigram_probabilities:
        calculate_unknown_bigrams_probability(bigram)
    s_bigrams_probabilities[bigram] = bigram_probabilities[bigram]

print(s_bigrams_probabilities)
```
```
{('the', 'former'): -8.910056249919954, ('former', 'president'): -8.983292059499819, ('president', 'donald'): -7.2844
29390975789, ('donald', 'frump'): -15.966117792178112, ('frump', 'lived'): -15.964921977038653, ('lived', 'in'): -10.
889778698156023, ('in', 'the'): -5.3671079102834485, ('the', 'white'): -7.431466730542328, ('white', 'house'): -7.107
781685333634}
```

In [19]:
```python
smallest_probability = min(s_bigrams_probabilities, key=s_bigrams_probabilities.get)
print(smallest_probability)
```
```
('donald', 'frump')
```

In [20]:
```python
two_smallest = sorted(s_bigrams_probabilities.items(), key=itemgetter(1))[:2]

print(two_smallest)
```
```
[(('donald', 'frump'), -15.966117792178112), (('frump', 'lived'), -15.964921977038653)]
```

In [21]:
```python
incorrect_word = two_smallest[0][0][1]
# incorrect_word = two_smallest[1][0][0]

print(incorrect_word)
```
```
frump
```

In [22]:
```python
english_words = list(english_words_set)
```

In [23]:
```python
print('hello' in english_words)
print('trump' in english_words)
print('clinton' in english_words)
print('obama' in english_words)
print('jump' in english_words)
print('esoteric' in english_words)
```
```
True
True
False
False
True
True
```

In [24]:
```python
def minimum_edit_distance(word1, word2):
    # levenshtein distance minimum edit distance table
    amt_rows = len(word1) + 1
    amt_cols = len(word2) + 1
    edit_distance = np.zeros((amt_rows, amt_cols))

    for r in range(1, amt_rows):
        edit_distance[r][0] = r
    for c in range(1, amt_cols):
        edit_distance[0][c] = c

    for row in range(1, amt_rows):
        for col in range(1, amt_cols):
            if word1[row - 1] == word2[col - 1]:
                edit_distance[row][col] = edit_distance[row - 1][col - 1]
            else:
                edit_distance[row][col] = min(edit_distance[row][col - 1] + 1,
                                              edit_distance[row - 1][col] + 1,
                                              edit_distance[row - 1][col - 1] + 2)

    return edit_distance[amt_rows - 1][amt_cols - 1]
```

In [25]:
```python
print(minimum_edit_distance('apple', 'trample'))
```
```
4.0
```

In [26]:
```python
print(len(english_words))
```
```
25487
```

In [27]:
```python
distances = {}
for i in range(len(english_words) - 1):
    distances[english_words[i]] = minimum_edit_distance(english_words[i], incorrect_word)

twenty = sorted(distances.items(), key=itemgetter(1))[:20]

print(twenty[0:len(twenty)-1])
```
```
[('rump', 1.0), ('fum', 2.0), ('trump', 2.0), ('forum', 2.0), ('rum', 2.0), ('grump', 2.0), ('crump', 2.0), ('jump',
3.0), ('firm', 3.0), ('farm', 3.0), ('romp', 3.0), ('ramp', 3.0), ('grumpy', 3.0), ('drum', 3.0), ('rumpus', 3.0),
('from', 3.0), ('pump', 3.0), ('up', 3.0), ('bump', 3.0)]
```

In [28]:
```python
first_word = two_smallest[0][0][0]
new_probabilities = {}

for i in range(len(twenty)):
    bigram = (first_word, twenty[i][0])
    if bigram not in bigram_probabilities:
        calculate_unknown_bigrams_probability(bigram)
    new_probabilities[bigram] = bigram_probabilities[bigram]

largest = sorted(new_probabilities.items(), key=itemgetter(1))[-1:]

print(largest)
correct_word = largest[0][0][1]
print(correct_word)
```
```
[(('donald', 'trump'), -7.203158871411385)]
trump
```

In [29]:
```python
print("Original Sentence:", original_sentence)
print("Did you mean to replace", incorrect_word, "with", correct_word, "?")
response = input("")

final_sentence = original_sentence

if response == "yes":
    for word in original_sentence.translate(str.maketrans('', '', string.punctuation)).split():
        if word.lower() == incorrect_word:
            final_sentence = final_sentence.replace(word, correct_word)
            break
    print("Updated Sentence: ", final_sentence)
else:
    print("We could not find a suitable change for", incorrect_word)
```
```
Original Sentence: The former president, Donald Frump, lived in the White House.
Did you mean to replace frump with trump ?
yes
Updated Sentence:  The former president, Donald trump, lived in the White House.
```

In [30]:
```python
print('reporter' in english_words)
```
```
False
```

In [31]:
```python
# trie to make searching faster
```

In [32]:
```python
class Trie:
    def __init__(self):
        self.word = None
        self.children = {}
    def add(self, word):
        node = self
        for l in word:
            if l not in node.children: # only add letter if it isn't already there
                node.children[l] = Trie()
            node = node.children[l]
        node.word = word

def find(node, word, letter, previous, size, words):
    row = [previous[0] + 1]

    for column in range(1, len(word) + 1):
        delete = previous[column] + 1 # deletions cost 1
        insert = row[column - 1] + 1 # insertions cost 1
        replace = 0

        if word[column - 1] == letter:
            replace = previous[column - 1]
        else:
            replace = previous[column - 1] + 2 # substitutions cost 2
        row.append(min(delete, insert, replace))
    if min(row) <= size:
        for letter in node.children:
            find(node.children[letter], word, letter, row, size, words)
    if node.word != None and row[len(row) - 1] <= size:
        words.append((node.word, row[len(row) - 1]))

def search(word, size):
    row = range(len(word) + 1)
    words = []
    for l in trie.children:
        find(trie.children[l], word, l, row, size, words) # recursively search
    return words

start = time.time()
trie = Trie()
for w in english_words:
    trie.add(w)
words = search("frump", 3)
end = time.time()
print(words)
print("Search took %g seconds" % (end - start))
```
```
[('drum', 3), ('dump', 3), ('crump', 2), ('pump', 3), ('bump', 3), ('hump', 3), ('fume', 3), ('fum', 2), ('form', 3),
('forum', 2), ('farm', 3), ('firm', 3), ('from', 3), ('ramp', 3), ('rumpus', 3), ('rumple', 3), ('rump', 1), ('rum',
2), ('romp', 3), ('arum', 3), ('grumpy', 3), ('grump', 2), ('lump', 3), ('trump', 2), ('up', 3), ('jump', 3)]
Search took 1.46755 seconds
```