NLP-assignment

Pranav rao

2211cs020420

1. Correct the Search Query

**import** re

**import** zlib

**import** json

**from** difflib **import** get\_close\_matches

*# Build a dictionary (corpus) with common words and country names*

corpus = {

"going", "to", "china", "who", "was", "the", "first", "president", "of",

"india", "winner", "match", "food", "in", "america"

}

*# Serialize and compress the corpus*

compressed\_corpus = zlib.compress(json.dumps(**list**(corpus)).encode())

*# Decompress and load the corpus*

corpus = **set**(json.loads(zlib.decompress(compressed\_corpus).decode()))

**def** correct\_word(word):

"""Find the closest match for a misspelled word in the corpus."""

matches = get\_close\_matches(word, corpus, n=1, cutoff=0.8)

**return** matches[0] **if** matches **else** word

**def** correct\_query(query):

"""Correct spelling and segmentation issues in a query."""

words = query.split()

corrected\_words = [correct\_word(word) **for** word **in** words]

**return** " ".join(corrected\_words)

*# Input*

n = **int**(**input**())

queries = [**input**().strip() **for** **\_** **in** **range**(n)]

*# Output corrected queries*

**for** query **in** queries:

**print**(correct\_query(query))

2. **Deterministic Url and HashTag Segmentation**

**import** sys

**def** clean\_url(line0):

i = 0

line = line0[4:] **if** line0[0:4] == "www." **else** line0

rline = **list**(**reversed**(line))

last\_dot = 0

**while** i < **len**(line):

**if** rline[i] == '.':

**if** i > 0 **and** i < **len**(line) - 1:

**if** rline[i - 1].isalpha():

last\_dot = i

**else**:

last\_dot = i

i+=1

**return** line[0:**len**(line) - (last\_dot+1)]

**def** clean(lines):

**return** [

line[1:].strip().lower()

**if**

line[0] == '#'

**else**

clean\_url(line).strip().lower()

**for** line **in** lines

]

**def** get\_words():

words\_file = **open**("words.txt",'r')

words\_content = words\_file.read()

words\_lines = **list**(

**filter**(

**lambda** x: x != " " **and** x != "",

words\_content.split("\n")))

words = {}

**for** i **in** **range**(0, **len**(words\_lines)):

words[words\_lines[i].strip().lower()] = 0

*# print(words)*

**return** words

**def** is\_number(num\_str):

**try**:

**int**(num\_str)

**return** **True**

**except** ValueError:

**try**:

**float**(num\_str)

**return** **True**

**except** ValueError:

**return** **False**

**def** process\_line(stack, line, words, current\_word\_start, j, consolidated):

**if** j > **len**(line):

**if** stack == []:

**return** [(0, **len**(line))]

**else**:

(last\_current\_word\_start, last\_j, last\_consolidated0) = stack.pop()

last\_consolidated = last\_consolidated0.copy()

last\_consolidated.append((last\_current\_word\_start, last\_j))

**if** last\_j == **len**(line):

**return** last\_consolidated

**else**:

**return** process\_line(stack, line, words, last\_j, last\_j + 1, last\_consolidated)

**else**:

current\_word = line[current\_word\_start:j]

**if** current\_word **in** words:

stack.append((current\_word\_start, j, consolidated))

**elif** is\_number(current\_word):

stack.append((current\_word\_start, j, consolidated))

**return** process\_line(stack, line, words, current\_word\_start, j + 1, consolidated)

**def** separate(lines, words):

**for** line **in** lines:

line\_words = process\_line([], line, words, 0, 1, [])

line\_words\_print = []

**for** (start, end) **in** line\_words:

line\_words\_print.append(line[start:end])

**print**(" ".join(line\_words\_print))

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

words = get\_words()

s = sys.stdin.read()

lines = **list**(

**filter**(

**lambda** x: x != " " **and** x != "",

s.split("\n")))

cleaned\_lines = clean(lines[1:])

*# print(cleaned\_lines)*

separate(cleaned\_lines, words)

*# print(separated)*

# **3. Disambiguation: Mouse vs Mouse**

**import** json

*# Offline model building*

animal\_words = ["tail", "fur", "genome", "species", "temperature", "postnatal"]

computer\_words = ["device", "click", "cursor", "input", "keyboard", "screen"]

model = {

"animal": **list**(animal\_words), *# Convert to list for JSON serialization*

"computer-mouse": **list**(computer\_words) *# Convert to list for JSON serialization*

}

*# Serialize the model (done offline)*

**with** **open**("mouse\_context\_model.json", "w") **as** f:

json.dump(model, f)

*# Function to classify sentences*

**def** classify\_sentence(sentence, model):

words = sentence.lower().split()

animal\_score = **sum**(1 **for** word **in** words **if** word **in** model["animal"])

computer\_score = **sum**(1 **for** word **in** words **if** word **in** model["computer-mouse"])

**return** "animal" **if** animal\_score > computer\_score **else** "computer-mouse"

*# Load the model (during runtime)*

**with** **open**("mouse\_context\_model.json", "r") **as** f:

loaded\_model = json.load(f)

*# Input processing*

n = **int**(**input**()) *# Number of sentences*

results = []

**for** **\_** **in** **range**(n):

sentence = **input**().strip()

results.append(classify\_sentence(sentence, loaded\_model))

*# Output results*

**for** result **in** results:

**print**(result)

# **4. Language Detection**

*# Define common words for each language*

common\_words = {

'English': ['the', 'is', 'in', 'and', 'to', 'a', 'of', 'that', 'it', 'with'],

'French': ['le', 'est', 'dans', 'et', 'a', 'un', 'de', 'que', 'il', 'avec'],

'German': ['der', 'ist', 'in', 'und', 'zu', 'ein', 'von', 'dass', 'es', 'mit'],

'Spanish': ['el', 'es', 'en', 'y', 'a', 'un', 'de', 'que', 'lo', 'con', 'si', 'quieres'," te "

"tienes" "poner" "las" "pilas."]

}

*# Function to detect the language based on common words*

**def** detect\_language(text):

*# Remove non-ASCII characters and convert to lowercase*

text = ''.join([char **for** char **in** text **if** **ord**(char) < 128])

*# Split the text into words, removing punctuation*

words = text.lower().split()

*# Create a dictionary to store the count of common words for each language*

word\_count = {'English': 0, 'French': 0, 'German': 0, 'Spanish': 0}

*# Count how many common words appear for each language*

**for** word **in** words:

**for** language, word\_list **in** common\_words.items():

**if** word **in** word\_list:

word\_count[language] += 1

*# Find the language with the highest count of common words*

detected\_language = **max**(word\_count, key=word\_count.get)

*# Output the detected language*

**print**(detected\_language)

*# Read the entire input until EOF*

**import** sys

text = sys.stdin.read().strip() *# Read everything from stdin and remove extra spaces/newlines*

*# Detect the language*

detect\_language(text)

# **5. The Missing Apostrophes**

**import** re

*# Function to insert apostrophes in missing places*

**def** insert\_apostrophes(text):

*# Handle common contractions correctly*

contractions = {

"dont": "don't", "doesnt": "doesn't", "cant": "can't", "wont": "won't", "isnt": "isn't",

"arent": "aren't", "im": "I'm", "ill": "I'll", "theres": "there's", "its": "it's",

"whats": "what's", "whos": "who's", "thats": "that's", "youre": "you're", "were": "we're",

"theyre": "they're", "hasnt": "hasn't", "havent": "haven't", "hadnt": "hadn't", "couldnt": "couldn't",

"wouldnt": "wouldn't", "shouldnt": "shouldn't", "wasnt": "wasn't", "didnt": "didn't",

"hed": "he'd", "id": "I'd", "wed": "we'd", "theyve": "they've", "youve": "you've",

"ive": "I've", "youd": "you'd", "shes": "she's", "hes": "he's", "were": "we're"

}

*# Replace contractions*

**for** key, value **in** contractions.items():

text = re.sub(r'\b' + key + r'\b', value, text)

*# Handle possessive cases: "party's" instead of "partys"*

*# Don't touch plural forms: "parties" remains "parties"*

text = re.sub(r'(\w+?)s\b', r'\1\'s', text)

**return** text

*# Sample input*

input\_text = """At a news conference Thursday at the Russian manned-space facility in Baikonur, Kazakhstan, Kornienko said "we will be missing nature, we will be missing landscapes, woods." He admitted that on his previous trip into space in 2010 "I even asked our psychological support folks to send me a calendar with photographs of nature, of rivers, of woods, of lakes."

Kelly was asked if hed miss his twin brother Mark, who also was an astronaut.

"Were used to this kind of thing," he said. "Ive gone longer without seeing him and it was great."

The mission wont be the longest time that a human has spent in space - four Russians spent a year or more aboard the Soviet-built Mir space station in the 1990s."""

*# Insert apostrophes*

output\_text = insert\_apostrophes(input\_text)

*# Print the output with apostrophes inserted*

**print**(output\_text)

# **6.Segment the Twitter Hashtags**

*# A sample list of common words (In practice, you would use a much larger dictionary)*

common\_words = **set**([

"we", "are", "the", "people", "mention", "your", "faves", "now", "playing",

"dead", "follow", "me", "walking", "fave", "is", "a", "to", "and", "this"

])

*# Function to segment a single hashtag*

**def** segment\_hashtag(hashtag):

n = **len**(hashtag)

dp = [**None**] \* (n + 1)

dp[0] = [] *# Base case: an empty string has a valid segmentation*

*# Iterate through all possible end points of the substring*

**for** i **in** **range**(1, n + 1):

**for** j **in** **range**(i):

word = hashtag[j:i]

**if** word **in** common\_words **and** dp[j] **is** **not** **None**:

dp[i] = dp[j] + [word]

**break** *# If we found a valid split, no need to check further for this 'i'*

*# If dp[n] is not None, we have a valid segmentation*

**return** " ".join(dp[n]) **if** dp[n] **else** hashtag

*# Read input*

n = **int**(**input**()) *# Read the number of hashtags*

**for** **\_** **in** **range**(n):

hashtag = **input**().strip()

**print**(segment\_hashtag(hashtag))

**7. A Text-Processing Warmup**

**import** sys

**import** re

*# '15/11/2012','15/11/12', '15th March 1999','15th March 99' or '20th of March, 1999').*

*# You can make the following assumptions*

*# 1) In the date, year and day will always be in numeric form.*

*# Which means, you don't have to worry about "fifteenth" or "twentieth" etc.*

*# Month, could be either numeric form (1-12) or with its name (January-December, Jan-Dec).*

*# 2) This is a bit open ended, and somewhat intentionally so.*

*# The aim is for you to try to write something which figures out as many common patterns as possible, in which dates are present in text.*

*# 3) Most of the test cases are Wikipedia articles.*

*# Having a look at the common formats in which dates occur in those, will help.*

*# 4) Dates could either be in the form:*

*# Month followed by Day followed by Year, or Day followed by Month followed by Year.*

*# 5) The day could be in the form of either (1,2,3,...31) or (1st, 2nd, 3rd...31st).*

**def** count\_data(line, p\_a, p\_an, p\_the, p\_date):

count = []

count.append(p\_a.findall(line))

count.append(p\_an.findall(line))

count.append(p\_the.findall(line))

count.append(p\_date.findall(line))

*# 4T lines, four lines of output for each test case.*

*# First line -> number of occurrences of 'a'.*

*# Second line -> number of occurrences of 'an'.*

*# Third Line -> number of occurrences of 'the'.*

*# Fourth Line -> number of occurrences of date information.*

*# print(count[3])*

**print**("\n".join(**map**(**lambda** x: **str**(**len**(x)), count)))

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

s = sys.stdin.read()

lines = **list**(

**filter**(

**lambda** x: x != " " **and** x != "",

s.split("\n")))[1:]

*# print(lines)*

*# print(list(map(read\_data, lines)))*

p\_a = re.**compile**(r"\ba\b", re.IGNORECASE)

p\_an = re.**compile**(r"\ban\b", re.IGNORECASE)

p\_the = re.**compile**(r"\bthe\b", re.IGNORECASE)

months = "January|February|March|April|May|June|July|August|September|October|November|December"

months3 = "Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec"

allmonths = months + "|" + months3

p1str = r"\d\d?/\d\d?/\d\d?\d?\d?"

p2str = r"(\d\d?)(st|nd|rd|th)?\s+(of\s+)?(" + allmonths + r")(\s\*,)?\s+(\d\d?\d?\d?)"

p3str = "(" + allmonths + r")\s+(\d\d?)(st|nd|rd|th)?(\s\*,)?\s+(\d\d?\d?\d?)"

pstr = "(" + "|".join([p1str, p2str, p3str]) + ")"

p\_date = re.**compile**(pstr)

**for** l **in** lines:

count\_data(l, p\_a, p\_an, p\_the, p\_date)

**8.Who is it?**

**import** re

*# Read input values*

n = **int**(**input**()) *# Number of lines of text*

text = [**input**().strip() **for** **\_** **in** **range**(n)] *# Text lines*

noun\_phrases = **input**().strip().split(';') *# List of noun phrases*

*# Combine all text into a single string*

full\_text = ' '.join(text)

*# Find all pronouns surrounded by '\*\*'*

pronouns\_with\_positions = **list**(re.finditer(r'\\*\\*(\w+)\\*\\*', full\_text))

*# List to store resolved pronouns*

resolved\_pronouns = []

noun\_phrase\_index = 0 *# To track the noun phrases used*

*# Loop over the pronouns found in the text and resolve them*

**for** **match** **in** pronouns\_with\_positions:

pronoun = **match**.group(1) *# Extract the pronoun (e.g., "she", "her", "it")*

**if** noun\_phrase\_index < **len**(noun\_phrases): *# Ensure there are still noun phrases available*

resolved\_pronouns.append(noun\_phrases[noun\_phrase\_index])

noun\_phrase\_index += 1 *# Move to the next noun phrase*

*# Output the resolved pronouns*

**for** resolved **in** resolved\_pronouns:

**print**(resolved)

E**xpand the Acronyms**

**import** re

**def** extract\_expansions(snippets):

expansions = {}

*# Regex for acronyms followed by (expansion) and acronym followed by "is expansion"*

pattern1 = re.**compile**(r'([A-Z]{2,})\s\*\(([^)]+)\)')

pattern2 = re.**compile**(r'([A-Z]{2,})\s+is\s+([^\.]+)')

**for** snippet **in** snippets:

*# Matches for acronym in parentheses*

matches1 = pattern1.findall(snippet)

**for** acronym, expansion **in** matches1:

expansions[acronym] = expansion.strip()

*# Matches for acronym followed by "is"*

matches2 = pattern2.findall(snippet)

**for** acronym, expansion **in** matches2:

expansions[acronym] = expansion.strip()

**return** expansions

**def** main():

*# Reading input*

**import** sys

**input** = sys.stdin.read

data = **input**().split("\n")

N = **int**(data[0]) *# Number of snippets*

snippets = data[1:N+1] *# N snippets*

tests = data[N+1:] *# Test acronyms*

expansions = extract\_expansions(snippets)

*# Output the expansion for each test acronym*

**for** test **in** tests:

**if** test:

**print**(expansions.get(test, "Expansion not found"))

**if** **\_\_name\_\_** == "\_\_main\_\_":

main()