Homework 1

CS/CE 458/463 Natural Language Processing
Habib University

Fall 2024

Instructor: Dr. Ayehsa Enayat
Abeeha Zehra – az07728

Q1: Resolving Ambiguities Between DD/MM/YYYY and MM/DD/YYYY Date Formats

a. Your proposed algorithm or method for resolving the date format ambiguity.

Answer

For this task, the first step involved extracting text from the file, followed by extracting dates from the text. Since the goal was to identify date formats using both logical and contextual methods, I approached the contextual analysis by creating two lists: one containing terms commonly associated with US English and the other with European English. For each sentence, the algorithm checked for the presence of any word from either list. If a match was found, the corresponding date format was identified based on the context. For the logical method, I examined the first component of the date. If the value was greater than 12, the date was classified as the DD/MM/YYYY; if it was less than 12 and the second component part was greater than 12, it was classified as the MM/DD/YYYY. If the above conditions were not met, the format was deemed ambiguous.The code for this is given in Listing 1.

b. A list of extracted dates and your interpretation of their formats (from the output text file).

Answer

According to the code, the dates extracted and their formats are:

Dates	Format
05/12/2023	ambiguous
12/05/2023	ambiguous
03/08/2024	ambiguous
08/03/2024	ambiguous
05/06/2023	ambiguous

Upon reviewing the code in Listing 1, it is clear that the context is checked first. However, since none of the sentences included terms from the predefined lists of European or US words, contextual identification of the date formats was not possible. Consequently, the logical approach was used, where the date and month were evaluated. As shown in the code, all the extracted dates had both the date and month components less than 12, making it difficult to assign a specific date format. Therefore, all the dates were classified as ambiguous.

c. Any challenges or difficulties you encountered during the process, along with how you addressed them.

Answer

Identifying the date format through the logical approach was straightforward when the date and month values were easy to distinguish, such as when the date was greater than 12 and the month was less than 12, or vice versa. However, when both components were less than 12, the format became ambiguous, as it was unclear which format the date followed. The contextual identification posed initial difficulties, as I initially used hard coding. To generalize the code, I created two lists: one for US-specific terms and another for European-specific terms. This allowed the code to identify the correct date format if any sentence contained words from these lists. However, since the lists contained a limited number of terms, the method wasn't entirely reliable, especially when encountering words outside the predefined lists.

Q2: Identifying the First 10 Merges in a Wordpiece Algorithm

a. The first 10 pairs of tokens that were merged by your algorithm

Answer

For this question, we were supposed to write the word piece algorithm that is given in Listing 2 and run it on the given text file. Hence, after running the word piece, the first 10 merges are:

Pair	**Merged Token**
('1', '##0')	10
('e', '##0')	ex
('o', '##*f*')	of
('##*f*', '##*y*')	##*fy*
('##*m*', '##*p*')	##*mp*
('##*q*', '##*u*')	##*qu*
('##*b*', '##*u*')	##*bu*
('##"', '##,')	##",
('ex', '##*a*')	exa
('exa', '##*mp*')	examp

b. Any challenges you encountered and how you addressed them

Answer

The code I developed is relatively straightforward, utilizing dictionaries and lists to manage data, vocabulary, pair frequencies, and scores, which made tracking the information easier. I encountered an issue with score calculation, where the frequency of new tokens wasn't updating correctly, but this was resolved quickly after performing a dry run. Additionally, I noticed that some pairs had the same score, but their position in the dictionary affected whether they were selected for merging.

Q3: Tokenizing Urdu Text

a. Your python code.

Answer

To achieve this, we needed to process an Urdu text file and generate tokens from it. Therefore, I employed two distinct methods to accomplish this task:

- I utilized the built-in ‘split‘ function to generate tokens based on spaces, which allowed me to store the individual Urdu words as tokens in a separate text file named “output1·txt”. The code for this process is provided in Listing 3.
- I also utilized the NLTK library. By importing regexp_tokenize, I was able to divide the text into words and punctuation, generating tokens that were then saved in a different file named “output2·txt”. The code for this process is shown in Listing 4.

After comparing the results, it is evident that the ‘split‘ function simply divides the text based on whitespace, which may not be suitable for more complex tokenization tasks. On the other hand, the ‘regexp_tokenize‘ function offers the flexibility to define custom rules for tokenization, such as separating words based on spaces, punctuation marks, or even creating language-specific patterns. This makes it a more versatile option for handling diverse tokenization requirements.

b. Any challenges you encountered and how you addressed them

Answer

For this task, I also attempted to use the Urdu Hack library. However, I encountered issues importing ‘urduhack‘ due to a version conflict between the ‘keras‘ and ‘TensorFlow‘ libraries, which are dependencies for ‘urduhack‘. I tried using it on two different platforms—Visual Studio and Google Colab—but was unsuccessful on both.

Listing 1: Code for Determining Date Formats

```
import re
# Reading the file
file_path = 'Question1/date_format_dd_mm_yyyy.txt'
# Open and read the file
with open(file_path, 'r') as file:
    content = file.read()
# The \s* handles any spaces after the period
sentences = re.split(r'\.\s*', content)
words=[i.split() for i in sentences]
#regular expression for dates
date_format = r'\d{1,2}\/\d{1,2}\/\d{1,4}'
#us words
MM_DD_YY_words= ["fall", "thanksgiving", "memorial day", "labor day", "fourth of july", "veterans day",
"super bowl", "nba finals", "world series","washington dc", "new york", "california","chicago", "state", "governor", "color", "center", "honor", "semester",
"elementary school", "high school", "college", "fiscal year", "q1", "q2", "q3", "q4"]
# european words
DD_MM_YY_words=["autumn", "christmas", "boxing day", "bank holiday", "easter", "good friday","new year's day", "bonfire night", "summer holidays", "half term", "london","paris", "berlin", "european union", "british", "england", "united kingdom","colour", "centre", "honour", "favourite", "theatre", "university", "headteacher","primary school", "secondary school", "european parliament", "fifa world cup","uefa", "euros", "football", "q1", "q2", "q3","q4"]
#function to identify the words of the sentences as US or European
def context(sentence):
  for i in sentence:
    if i in DD_MM_YY_words:
      return 'DD/MM/YY'
    elif i in MM_DD_YY_words:
      return 'MM/DD/YY'
date_lst=[]
for i in sentences:
  dates= re.findall(date_format,i)
  context_ans = context(i)
  for j in dates:
    if context_ans == 'DD/MM/YY':
      date_lst.append((j,' The format is DD/MM/YY contextually'))
      continue
    elif context_ans == 'MM/DD/YY':
      date_lst.append((j,' The format is MM/DD/YY contextually'))
      continue
    date,month,year = j.split('/')
    if int(date) > 12:
      date_lst.append((j,' The format is DD/MM/YY logically'))
    elif int(date)< 12 and int(month)> 12:
      date_lst.append((j,' The format is MM/DD/YY logically'))
    else:
      date_lst.append((j,' The format is ambiguous '))
with open('Question1/AbeehaZehra_az07728.txt', 'w', encoding='utf-8') as file:
  for i in date_lst:
     file.write(i[0] +  ": "+  i[1] + '\n')
```

Listing 2: Code for Word Piece

```python
import re
#function to calculate the pair frequency
def get_pair_freqs(words,vocab):
  pairs= dict()
  for i in range(len(words) - 1):
    if words[i] != '\n' and words[i+1]!='\n':
      pair = (words[i], words[i + 1])
      if pair not in pairs:
          pairs[pair] = 1
      else:
          pairs[pair] += 1
  return pairs

#function to calcualte score
def get_score(pair_freq,vocab,words):
  scores=dict()
  for i in range(len(words)-1):
    if words[i] != '\n' and words[i+1]!='\n':
      pair = (words[i], words[i + 1])
      sc = pair_freq[pair] / (vocab[words[i]] * vocab[words[i+1]])
      scores[pair] = sc
  return scores

def word_piece(content, merges):
  words = content.split()
  #Creating vocabulary
  vocab= dict()
  data=[]
  for word in words:
    for j, letter in enumerate(word):
        if j == 0:
            # First letter of the word (no ## prefix)
            if letter not in vocab:
                vocab[letter] = 1
            else:
                vocab[letter] += 1
            data.append(letter)  # Append first letter without ##
        else:
            # Subsequent letters (with ## prefix)
            string = "##" + letter
            if string not in vocab:
                vocab[string] = 1
            else:
                vocab[string] += 1
            data.append(string)
    data.append('\n')
  mergelst=[]
  for i in range(merges):
    pair_freq= get_pair_freqs(data,vocab)
    score = get_score(pair_freq,vocab,data)
    max_score = max(score.items(), key=lambda x: x[1])
    cleaned_key1 = max_score[0][1].replace('##', '')
    new_word = max_score[0][0]+cleaned_key1
    vocab[new_word] = pair_freq[max_score[0]]
    i = 0
    while i < len(data) - 1:
        # Check if consecutive elements match max_score pair
        if data[i] == max_score[0][0] and data[i+1] == max_score[0][1]:
            # Replace data[i] with the new word
            data[i] = new_word
            # Remove data[i+1] (the second part of the pair)
            data.pop(i+1)
        else:
            # Move to the next element
            i += 1
    mergelst.append(new_word)
  return mergelst

# Reading the file
file_path = 'Question2/wordpiece_input.txt'
# Open and read the file
with open(file_path, 'r') as file:
    content = file.read()
content = content.lower()
merges = word_piece(content,8)
print("The␣merged␣pairs␣are:␣\n",merges)
```

Listing 3: Code for tokenizing using split function

```python
#Method # 1
file_path = 'urdu_text_input.txt'
# Open and read the file with UTF-8 encoding
with open(file_path, 'r', encoding='utf-8') as file:
    content = file.read()
tokens = content.split()
#Writing in the file
with open('output1.txt', 'w', encoding='utf-8') as file:
    for token in tokens:
        file.write(token + '\n')
```

Listing 4: Code for tokenizing using NLTK library

```python
import nltk
from nltk.tokenize import regexp_tokenize

#Method # 2
reg_ex = r'\w+|[^\w\s]'
file_path = 'urdu_text_input.txt'

# Open and read the file with UTF-8 encoding
with open(file_path, 'r', encoding='utf-8') as file:
    content = file.read()
token1 = regexp_tokenize(content, reg_ex)

#writing in the file
with open('output2.txt', 'w', encoding='utf-8') as file:
    for token1 in tokens:
        file.write(token + '\n')
```