CS 458 - Natural Language Processing
Assignment 01
Batool Ali Akbar - ba07612

1. (40 points) Resolving Ambiguities Between DD/MM/YYYY and MM/DD/YYYY Date Formats

   (a) (10 points) Python Script

Solution:

```

import re

us_spelling = set([
    "color", "honor", "favorite", "theater", "center", "meter",
    "defense", "license", "analyze", "realize", "organize",
    "traveler", "jewelry", "program", "aluminum", "plow", "catalog",
    "gray", "leukemia", "tire",  "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", "semester","elementary school",
    "high school", "college", "fiscal year"
])

british_spelling = set([
    "colour", "honour", "favourite", "theatre", "centre", "metre",
    "defence", "licence", "analyse", "realise", "organise",
    "traveller", "jewellery", "programme", "aluminium", "plough",
    "catalogue", "grey", "leukaemia", "tyre",  "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",
    "favourite", "theatre", "university", "headteacher", "primary school",
    "secondary school", "european parliament", "fifa world cup", "uefa",
    "euros", "football"
])

corpus = open("date_format_dd_mm_yyyy.txt", "r")
corpus = corpus.read()
corpus = corpus.lower()
corpus_set = corpus.split()
corpus_set = set(corpus_set)

output_file = open("q1_batool_ba07612.txt", "w", encoding='utf-8')

dates = re.findall(r'\d{1,2}\/\d{1,2}\/\d{1,4}', corpus)

for d in dates:
    if int(d[:2]) > 12:
        format = "DD/MM/YYYY"
    elif int(d[3:5]) > 12:
        format  = "MM/DD/YYYY"
    else:
        if len(corpus_set.intersection(us_spelling)) != 0:
            format = "MM/DD/YYYY"
        elif len(corpus_set.intersection(british_spelling)) != 0:
            format = "DD/MM/YYYY"
        else:
            format = "ambiguous"
    output_file.write(d + ": " + format + "\n")

```

(b) (10 points) Output

Solution:

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

(c) (20 points) Report

(a) Proposed Algorithm

Solution: The proposed algorithm extracts dates from the corpus using a regular expression. It then classifies the dates based on the following criteria:

- If the first two digits are greater than 12, then the date format is DD/MM/YYYY.
- If the second pair of digits is greater than 12, then the date format is MM/DD/YYYY.
- If both the pairs are less than 12, then the code checks for contextual clues to understand the date format i.e. if the corpus uses US words, then the format is MM/DD/YYYY and if it uses British words, then the format is DD/MM/YYYY.
- In case the dates are not classified by any of the above conditions, the dates are labelled ambiguous.

(b) List of Extracted Dates and Interpretation

Solution:

Date	Interpretation
05/12/2023	ambiguous
12/05/2023	ambiguous
03/08/2024	ambiguous
08/03/2024	ambiguous
05/06/2023	ambiguous

(c) Challenges

Solution: Logical clues were not enough to classify the dates in the given corpus. Therefore, I had to rely on the contextual clues. For this, I generated a list of words that could be used to differentiate between a US English text and a British English text. However, I still ended up with date format being ambiguous. This could be because the used list of words is finite and might not have the words present within the corpus. Moreover, it is possible that the corpus does not have any such words that are different in US and English British. Hence, the formats remain ambiguous.

2. (40 points) Identifying the First 10 Merges in a Wordpiece Algorithm

   (a) (20 points) Python Script

Solution:

```python

import re

# ---------------------------- Input Handling ---------------------------- #
def input_handling(file_name):
  corpus = open(file_name, "r")
  corpus = corpus.read()
  corpus = corpus.lower()
  corpus = corpus.split()
  return corpus

# ----------- Tokenization based on words + Frequency of each word -------- #
def calculate_word_freq(corpus):
  word_freq = {}
  for w in corpus:
      if w in word_freq:
          word_freq[w] += 1
      else:
          word_freq[w] = 1
  return word_freq

# ------- Tokenization based on letters + Frequency of each letter ------- #
def letter_tokenization(word_freq):
  tokenized = []
  letter_freq = {}
  for i in word_freq:
      s = [i[0]]
      if i[0] in letter_freq:
          letter_freq[i[0]] += word_freq[i]
      else:
          letter_freq[i[0]] = word_freq[i]

      for k in i[1:]:
          s.append('##'+k)

          if ('##'+k) in letter_freq:
              letter_freq['##'+k] += word_freq[i]
          else:
              letter_freq['##'+k] = word_freq[i]

      tokenized.append((s, word_freq[i]))
  return ((tokenized, letter_freq))

# ---------------------------- Merging ------------------------------------ #
def merge(tokenized, letter_freq, merge_limit):
  merges = 0
  vocab = []

  while merges != merge_limit:
      pairs_freq = {}

      # --------- Finding pairs + Frequency of each pair ------------ #
      for t in range(len(tokenized)):
          for u in range(len(tokenized[t][0])-1):
              new_pair = tokenized[t][0][u] + tokenized[t][0][u+1]
              if new_pair not in pairs_freq:
                  #[freq_pair, freq_1, freq_2, [(row, col)]]
                  pairs_freq[new_pair] = [tokenized[t][1], letter_freq[
    tokenized[t][0][u]], letter_freq[tokenized[t][0][u+1]], [(t,u)]]
              else:
                  pairs_freq[new_pair][0] += tokenized[t][1]
                  pairs_freq[new_pair][3] += [(t,u)]

      # ---------- Calculating score for each pair ---------------- #
      score_arr = []
      for p in pairs_freq:
          score = pairs_freq[p][0] / (pairs_freq[p][1] * pairs_freq[p][2])
          score_arr.append((p, score))

      score_arr.sort(key=lambda x: x[1], reverse=True)


      # ------------------ Updating data -------------------------- #

      vocab.append(score_arr[0][0])
      letter_freq[score_arr[0][0]] = pairs_freq[score_arr[0][0]][0]

      change = pairs_freq[score_arr[0][0]][3]
      for c in change:
          tokenized[c[0]][0][c[1]] = tokenized[c[0]][0][c[1]] + tokenized[c[0]][0][c[1]+1]
          tokenized[c[0]][0].remove(tokenized[c[0]][0][c[1]+1])

      merges += 1
  return vocab

#-------------------------- Main Function ---------------------------------#

def main():
  file_name = "wordpiece_input.txt"
  corpus = input_handling(file_name)
  word_frequency = calculate_word_freq(corpus)
  tokenized, letter_freq = letter_tokenization(word_frequency)
  vocab = merge(tokenized, letter_freq, 10)

  output_file = open("q2_batool_ba07612.txt", "w", encoding='utf-8')

  output_file.write("Merged Vocabulary:\n\n")

  for i in range(len(vocab)):
    if vocab[i][0] == '#':
      vocab[i] = vocab[i][0:2] + re.sub('#', '', vocab[i][2:])
    else:
      vocab[i] = re.sub('#', '', vocab[i])
    output_file.write(vocab[i]+'\n')

main()

```

(b) (20 points) Report

(a) First 10 Merges

Solution:

Merge Number	Token 1	Token 2	Merged Tokens
1.	1	##0	10
2.	e	##x	ex
3.	o	##f	of
4.	##f	##y	##fy
5.	##m	##p	##mp
6.	##q	##u	##qu
7.	##b	##u	##bu
8.	##”	##,	##”,
9.	ex	##a	exa
10.	exa	##mp	examp

(b) Challenges

Solution: Initially, I used a set structure to store my vocabulary. This caused the output to be different in every run because the order of the pairs kept changing. Moreover, multiple pairs had the same score. This also caused the output to differ in different runs. Therefore, I had to switch to a list structure to avoid this.

3. (20 points) Tokenizing Urdu Text

   (a) (5 points) Python Script

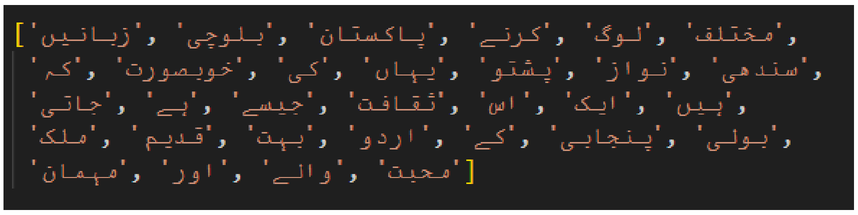
Solution:

```python

corpus = open("urdu_text_input.txt", "r", encoding='utf-8')
corpus = corpus.read()

#---------------------- Using Regular Expressions -------------------------#
import re

word_tokens_re = set(re.findall(r'\w+', corpus)) #word tokenization
letter_tokens_re = set(re.findall(r'\w', corpus)) #letter tokenization

output_file = open("q3_batool_ba07612.txt", "w", encoding='utf-8')

output_file.write("Using Regular Expression\n\n")
word_tokens_str = "\n".join(word_tokens_re)
output_file.write("Word Based\n")
output_file.write(word_tokens_str)
letter_tokens_str = "\n".join(letter_tokens_re)
output_file.write("\n\nLetter Based\n")
output_file.write(letter_tokens_str)
output_file.write("\n\n\n")

#-------------------------- Using nltk Library ----------------------------#
from nltk.tokenize import word_tokenize
word_tokens_nltk = set(word_tokenize(corpus))
output_file.write("Using nltk Library\n")
output_file.write("\n".join(word_tokens_nltk))
output_file.write("\n\n\n")

#------------------------- Using split() Method ---------------------------#

word_tokens_split = set(corpus.split())
output_file.write("Using split() Method\n")
output_file.write("\n".join(word_tokens_split))

```

(b) (10 points) Output

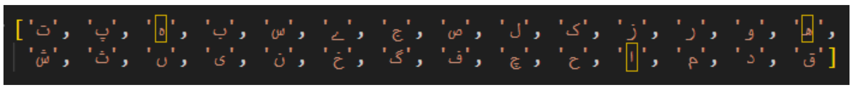
Solution:

- Using Regular Expression
  - Word Based Tokenization



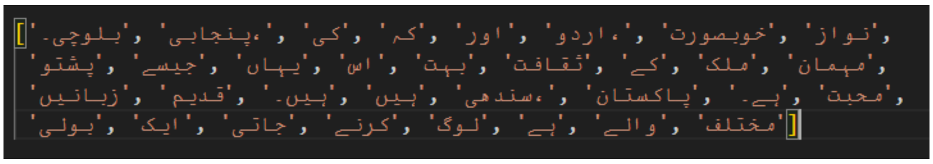
```
['مختلف', 'لوگ', 'کرنے', 'پاکستان', 'بلوچی', 'زبانیں',
 'سندھی', 'نواز', 'پشتو', 'یہاں', 'کی', 'خوبصورت', 'کہ',
 'ہیں', 'ایک', 'اس', 'ثقافت', 'جیسے', 'ہے', 'جاتی',
 'بولی', 'پنجابی', 'کے', 'اردو', 'بہت', 'قدیم', 'ملک',
 'محبت', 'والے', 'اور', 'مہمان']
```

  - Character Based Tokenization



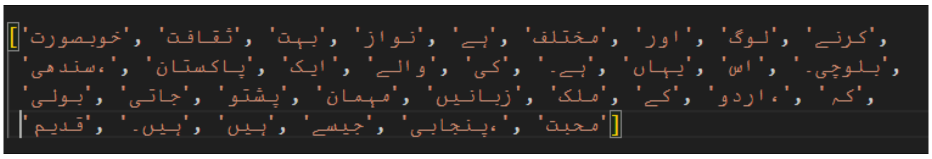
```
['ھ', 'و', 'ر', 'ز', 'ک', 'ل', 'ص', 'ج', 'ے', 'س', 'ب', 'ہ', 'پ', 'ت',
 'ق', 'د', 'م', 'ا', 'ح', 'چ', 'ف', 'گ', 'خ', 'ن', 'ی', 'ں', 'ث', 'ش']
```

- Using nltk Library



```
['نواز', 'خوبصورت', 'اردو،', 'اور', 'کہ', 'کی', 'پنجابی،', 'بلوچی۔',
 'مہمان', 'ملک', 'کے', 'ثقافت', 'بہت', 'اس', 'یہاں', 'جیسے', 'پشتو',
 'محبت', 'ہے۔', 'پاکستان', 'سندھی،', 'ہیں', 'ہیں۔', 'قدیم', 'زبانیں',
 'مختلف', 'والے', 'ہے', 'لوگ', 'کرنے', 'جاتی', 'ایک', 'بولی']
```

- Using split() Method



```
['کرنے', 'لوگ', 'اور', 'مختلف', 'ہے', 'نواز', 'بہت', 'ثقافت', 'خوبصورت',
 'بلوچی۔', 'اس', 'یہاں', 'ہے۔', 'کی', 'والے', 'ایک', 'پاکستان', 'سندھی،',
 'کہ', 'اردو،', 'کے', 'ملک', 'زبانیں', 'مہمان', 'پشتو', 'جاتی', 'بولی',
 'محبت', 'پنجابی،', 'جیسے', 'ہیں', 'ہیں۔', 'قدیم']
```

(c) (10 points) Report

(a) Code

Solution:

```python

corpus = open("urdu_text_input.txt", "r", encoding='utf-8')
corpus = corpus.read()

#----------------------- Using Regular Expressions ---------------------#
import re

word_tokens_re = set(re.findall(r'\w+', corpus)) #word tokenization
letter_tokens_re = set(re.findall(r'\w', corpus)) #letter tokenization

output_file = open("q3_batool_ba07612.txt", "w", encoding='utf-8')

output_file.write("Using Regular Expression\n\n")
word_tokens_str = "\n".join(word_tokens_re)
output_file.write("Word Based\n")
output_file.write(word_tokens_str)
letter_tokens_str = "\n".join(letter_tokens_re)
output_file.write("\n\nLetter Based\n")
output_file.write(letter_tokens_str)
output_file.write("\n\n\n")

#------------------------- Using nltk Library -------------------------#
from nltk.tokenize import word_tokenize
```

```python
word_tokens_nltk = set(word_tokenize(corpus))
output_file.write("Using nltk Library\n")
output_file.write("\n".join(word_tokens_nltk))
output_file.write("\n\n\n")

#-------------------------- Using split() Method -----------------------#

word_tokens_split = set(corpus.split())
output_file.write("Using split() Method\n")
output_file.write("\n".join(word_tokens_split))

```

(b) Challenges

Solution: I tried using different methods for tokenizing the text. The nltk library and the built in split() method produced the same output. The output from the Regular Expressions was a little different. The word tokenization using Regular Expressions ignored the punctuation marks while nltk and split() methods included the punctuation marks and treated them as a part of the word before them. I also tried to tokenize the text using the UrduHack library but the library had some dependencies on other libraries which were not available.