Spell Correction for Roman Urdu

Muhammad Usman P19-0096

April 1, 2023

1 Introduction

In this task, the goal is to develop a spell correction system for Roman Urdu language using the Noisy Channel model. The focus is on correcting non-word errors, which are common mistakes such as typos and misspellings.

2 Loading and Preprocessing Data for Spell Correction

```
import re
from collections import Counter

# Load data.txt as corpus
with open('/data.txt', 'r') as f:
  corpus = f.read().lower()

# Create a vocabulary of words from the corpus
vocab = re.findall(r'\b\w+\b', corpus)
word_counts = Counter(vocab)

# Load common misspellings in Roman Urdu from misspellings.txt
with open('/misspellings.txt', 'r') as f:
  misspellings = f.readlines()
misspellings = [line.strip().split(',') for line in misspellings]
```

This part of code loads a corpus of text data from a file named 'data.txt', creates a vocabulary of words from the corpus, and counts the frequency of each word in the vocabulary. It also loads a list of common misspellings in Roman Urdu from a file named 'misspellings.txt'. The misspellings are stored in a list of lists, where each sublist contains the correct spelling of a word followed by its common misspellings.

2.1 Creating Error Model tables for Noisy Channel Model Spell Correction

```
# Define insert, delete, substitute and transpose tables using misspellings
insert_table = {}
delete_table = {}
substitute_table = {}
transpose_table = {}

for misspelling in misspellings:
  if len(misspelling[0]) < len(misspelling[1]):
    table = insert_table
    word = misspelling[1]
  elif len(misspelling[0]) > len(misspelling[1]):
    table = delete_table
    word = misspelling[1]
  else:
    if misspelling[0] != misspelling[1]:
      table = substitute_table
```

```
      word = misspelling[1]
    else:
      continue

    if len(misspelling[0]) < 2:
      continue

    for i in range(len(misspelling[0])-1):
      transposed_word = misspelling[0][:i] + misspelling[0][i+1] + misspelling[0][i] +
                                             misspelling[0][i+2:]
      if transposed_word == misspelling[1]:
        table = transpose_table
        word = misspelling[1]
        break

  table[word] = table.get(word, 0) + 1
```

This part of code creates four tables for the insert, delete, substitute, and transpose operations using the misspellings list. For each misspelling, it checks the length of the misspelled word and the correct word to determine which table to add it to. If the length of the misspelled word is smaller than the correct word, it adds it to the insert table, if it's greater, it adds it to the delete table, and if they're equal, it checks if they're the same word and adds it to the substitute table if they're not, or skips it if they are. If the length of the misspelled word is greater than or equal to 2, it checks if the misspelling can be created by transposing adjacent characters in the correct word. If it can, it adds it to the transpose table, otherwise, it adds it to the appropriate table. Finally, it increments the count for the correct word in the appropriate table.

2.2 Function to calculate P(x|w) using error model tables

```
# Define a function to calculate P(x|w) using the error model tables
def calculate_P_x_given_w(x, w):
  # initialize probabilities with 1
  P_insert = P_delete = P_substitute = P_transpose = 1

  # calculate probabilities for insertions
  for i in range(len(x)):
    x_insert = x[:i] + ' ' + x[i:]
    if x_insert in insert_table and w in insert_table[x_insert]:
    P_insert *= insert_table[x_insert][w] / word_counts[x_insert]

  # calculate probabilities for deletions
  for i in range(len(x)):
    x_delete = x[:i] + x[i+1:]
    if x_delete in delete_table and w in delete_table[x_delete]:
    P_delete *= delete_table[x_delete][w] / word_counts[x_delete]

  # calculate probabilities for substitutions
  for i in range(len(x)):
    for c in 'abcdefghijklmnopqrstuvwxyz':
      x_substitute = x[:i] + c + x[i+1:]
      if x_substitute in substitute_table and w in substitute_table[x_substitute]:
        P_substitute *= substitute_table[x_substitute][w] / word_counts[x_substitute]

  # calculate probabilities for transpositions
  for i in range(len(x)-1):
    x_transpose = x[:i] + x[i+1] + x[i] + x[i+2:]
    if x_transpose in transpose_table and w in transpose_table[x_transpose]:
      P_transpose *= transpose_table[x_transpose][w] / word_counts[x_transpose]
  P_x_given_w = P_insert * P_delete * P_substitute * P_transpose
  return P_x_given_w
```

The function takes in a misspelled word "x" and a candidate correction "w" and uses error model tables to calculate the probability of "x" being a misspelling of "w". The function calculates the probabilities of different types of spelling errors such as insertions, deletions, substitutions, and transpositions, and multiplies them together to get the overall probability. The function returns the probability of "x" being a misspelling of "w" according to the error model tables.

2.3 Function to get the Candidate Words

```
def get_candidate_words(x, V):
  candidate_words = set()

  # insertions
  for i in range(len(x) + 1):
    for c in 'abcdefghijklmnopqrstuvwxyz':
      candidate = x[:i] + c + x[i:]
      if candidate in V:
      candidate_words.add(candidate)

  # deletions
  for i in range(len(x)):
    candidate = x[:i] + x[i+1:]
    if candidate in V:
      candidate_words.add(candidate)

  # substitutions
  for i in range(len(x)):
      for c in 'abcdefghijklmnopqrstuvwxyz':
        candidate = x[:i] + c + x[i+1:]
        if candidate != x and candidate in V:
          candidate_words.add(candidate)

  # transpositions
  for i in range(len(x) - 1):
    candidate = x[:i] + x[i+1] + x[i] + x[i+2:]
    if candidate in V:
      candidate_words.add(candidate)

  return candidate_words
```

This function takes a misspelled word 'x' and a set of valid words 'V' as input, and generates a set of candidate words that are similar to 'x' but are present in 'V'. The function does this by considering possible insertions, deletions, substitutions, and transpositions of characters in 'x'.

2.4 Function to Calculate P(w)

```
def calculate_P_w(w, corpus):
  # create a vocabulary of words from the corpus
  vocab = re.findall(r'\b\w+\b', corpus.lower())
  # count the number of occurrences of each word in the vocabulary
  word_counts = Counter(vocab)
  # calculate the total number of words in the corpus
  total_words = sum(word_counts.values())
  # calculate the probability of the word w
  P_w = word_counts[w] / total_words
  return P_w
```

2.5 Spell Correction Algorithm Implementation

```
x = 'usau'
candidate_words = get_candidate_words(x, corpus)

if not candidate_words:
# handle the case where there are no candidate words
  print("No candidate words found for '{}'".format(x))
else:
  P_x_given_w = {}
  for w in candidate_words:
  P_x_given_w[w] = calculate_P_x_given_w(x, w) # function to calculate P(x|w)

  # calculate P(w) for each candidate word w
```

```
P_w = {}
for w in candidate_words:
P_w[w] = calculate_P_w(w,corpus) # function to calculate P(w)

# calculate the score for each candidate word w
scores = {}
for w in candidate_words:
scores[w] = P_x_given_w[w] * P_w[w]

# select the candidate word with the highest score
selected_word = max(scores, key=scores.get)
print("Corrected word for '{}' is '{}'".format(x, selected_word))
```

This code corrects a misspelled word "x" by generating a set of candidate words that are one edit away from "x", and then scoring each candidate word based on the probability of observing "x" given "w" and the probability of the candidate word "w" (i.e., P(w)) in a given corpus. The word with the highest score is selected as the corrected word and printed out.

If there are no candidate words found, the code prints a message saying so.

2.6 Outputs

```
print("Corrected word for '{}' is '{}'".format(x, selected_word))

Corrected word for 'usaye' is 'usay'
```

Figure 1: This is Output of the Delete

```
selected_word = max(scores, key=scores.get)
print("Corrected word for '{}' is '{}'".format(x, selected_word))

Corrected word for 'usy' is 'usay'
```

Figure 2: This is Output of the Insert

```
print("Corrected word for '{}' is '{}'".format(x, selected_word))

Corrected word for 'usau' is 'usay'
```

Figure 3: This is Output of the substitude

```
# select the candidate word with the highest score
selected_word = max(scores, key=scores.get)
print("Corrected word for '{}' is '{}'".format(x, selected_word))

Corrected word for 'suay' is 'usay'
```

Figure 4: This is Output of the Transpose