INSTITUTE OF TECHNOLOGY OF CAMBODIA

Department of Information and Communication Engineering

MINI PROJECT 1 REPORT

Group 07

TOPIC : TEXT GENERATION

Deadline 21 December 2023

Course : Natural Language Processing

Lecturer : VALY Dona

Team Member

Name	ID
SOR Sopheak	e20191066
SUN Vysing	e20191124
THUE Naratt	e20191186
TY Sophearum	e20191219
VEN Thon	e20191250

2023-2024

I. Pre-Processing

1. Split paragraphs in the dataset into arrays of sentences.

```
# Tokenize sentences
from nltk.tokenize import sent_tokenize, word_tokenize

sentences = sent_tokenize(text)
print(sentences)
```

```
['The weather was cold and gray as usual at this time of year.', 'The trees were all leafless, with fall now just a memory.'
```

2. Clean data in each sentence to follow some set of rules and split words in a sentence into arrays of words.

```
# Function to tokenize words in sentences
def tokenize_word(sentences):
    sentences_tokenized = []
    for sentence in sentences:
        words = sentence.split()
        sentences_tokenized.append(words)
    return sentences_tokenized

# Tokenize words in sentences
sentences = tokenize_word(sentences)
```

```
['<s>', 'my', 'dream', 'house', 'would', 'be', 'located', 'in', 'the', 'country', ',', 'but', 'not', 'too', 'far', 'from', '
['<s>', 'since', 'the', 'basement', 'is', 'fairly', 'shallow', ',', 'there', 'are', 'three', 'foot', 'by', 'five', 'foot', '
['<s>', '"now', ',', 'imagine', 'that', 'another', 'person', 'who', 'was', 'also', 'at', 'the', 'store', 'saw', 'the', 'crim
['<s>', 'feeling', 'secure', 'means', 'being', 'at', 'home', ',', 'no', 'matter', 'the', 'atmosphere', ',', 'with', 'the', '
['<s>', 'cuts', 'of', 'chicken', 'and', 'turkey', 'were', 'ready', 'to', 'be', 'served', 'as', 'the', 'hunger', 'driven', 'p
['<s>', 'right', 'now', ',', 'the', 'world', 'outside', 'of', 'home', 'was', 'more', 'safe', 'and', 'structured', ',', 'not'
['<s>', 'as', 'if', 'father', 'christmas', 'had', 'came', 'early', ',', 'their', 'faces', 'lit', 'up', 'as', 'if', 'they', '
['<s>', 'surly', 'the', 'london', 'eye', "can't", 'be', 'bigger', 'than', 'this', 'right?', '</s>']
['<s>', 'you', 'have', 'to', 'enter', 'the', 'storage', 'room', 'to', 'get', 'to', 'the', 'bathroom', ',', 'which', 'is', 'l
['<s>', 'the', 'fight', 'lasted', 'for', 'about', '5', 'minutes', 'until', 'a', 'police', 'man', 'came', 'and', 'arrested',
['<s>', 'one', 'smoked', 'a', 'cigerette', 'whilst', 'another', 'gulped', '2', 'cans', 'of', 'lager', '.', '</s>']
['<s>', 'kids', 'were', 'crowding', 'around', 'so', 'they', 'could', 'get', 'first', 'pick', 'of', 'the', 'sugary', 'goodnes
['<s>', 'the', 'east', 'side', 'of', 'the', 'main', 'floor', 'contains', 'a', 'large', 'master', 'bedroom', 'with', 'a', 'wa
['<s>', 'the', "boy's", 'cheeks', 'little', 'chubby', 'and', 'red', 'now', '.', '</s>']
['<s>', 'music', 'booming', 'out', 'like', 'an', 'elephant', 'running', 'through', 'a', 'jungle', 'and', 'then', "'bang'", '
['<s>', 'your', 'answers', 'to', 'these', 'questions', 'will', 'form', 'a', 'basis', 'for', 'the', 'qualities', 'and', 'char
['<s>', 'the', 'weather', 'was', 'cold', 'and', 'gray', 'as', 'usual', 'at', 'this', 'time', 'of', 'year', '.', '</s>']
```

3. Split the corpus into 3 for train valid and test by 70% 10% and 20% respectively.

```
# Split data into 3
train_size = int(len(sentences)*0.7)
valid_size = int(len(sentences)*0.1)

train = sentences[:train_size]
valid = sentences[train_size:train_size+valid_size]
test = sentences[train_size+valid_size:]

print(len(train), len(valid), len(test))
```

```
237 34 69
```

II. Build models

1. Back-off model

I build the back-off by doing the following steps:

a) Create a function to create any n-gram model by splitting the i to n-1 word in the sentence into the context column and taking the nth word to the word column and i

start from 0 to end of sentence. Then it count the amount of time the unique context and word appear and calculating the probability.

b) Use the function in a) to create 4 n-gram model (uni-gram, bi-gram, tri-gram and four-gram)

c) Append all those n-gram models together and call it backoff_model:

backoff_model

	context	word	count	probability
0	()	<s>	237	0.037896
1	()	my	33	0.005277
2	()	dream	3	0.000480
3	()	house	11	0.001759
4	()	would	17	0.002718
...	...	...	...	...
17718	(gossip, mill, was)	in	1	1.000000
17719	(mill, was, in)	full	1	1.000000
17720	(was, in, full)	swing	1	1.000000
17721	(in, full, swing)	.	1	1.000000
17722	(full, swing, .)	</s>	1	1.000000

17723 rows × 4 columns

2. Interpolation model

I build the interpolation by doing the following steps:

a) Create a function to create any n-gram model by splitting the i to n-1 word in the sentence into the context column and taking the nth word to the word column and i start from 0 to end of sentence. Then it count the amount of time the unique context and word appear and calculating the probability. But this time, use add-k smoothing to get the probability.

$$P_{Add-k}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + k}{C(w_{i-1}) + kV}$$

Figure 1: Add-k smoothing formula

b) Use the function in a) to create 4 n-gram model (uni-gram-smooth, bi-gram-smooth, tri-gram-smooth and four-gram-smooth)

c) In each model, add the probability of all its lower grams together and multiply each probability with its lamda values.

d) Append all those n-gram models together and call it interpolation_model:

interpolation_model

	context	word	count	probability
0	()	<s>	237	0.029314
1	()	my	33	0.004188
2	()	dream	3	0.000493
3	()	house	11	0.001478
4	()	would	17	0.002217
...	...	...	...	...
17718	(gossip, mill, was)	in	1	0.003116
17719	(mill, was, in)	full	1	0.000978
17720	(was, in, full)	swing	1	0.000885
17721	(in, full, swing)	.	1	0.008341
17722	(full, swing, .)	</s>	1	0.036960

17723 rows × 4 columns

III. Calculate perplexity

```
def perplexity(model, data, n):
    perplex = Decimal(1)
    loop_count = 0
    for sentence in data:
        for i in range(len(sentence)-1-n):
            loop_count = loop_count + 1
            perplex = perplex * ( Decimal(1) / Decimal(get_proba(model, tuple(sentence[i:n+i-1]), sentence[n+i-1])) )
    perplex = perplex ** (Decimal(1)/Decimal(loop_count))
    return perplex
```

Figure 2: Perplexity calculation function

By writing the above function, following the perplexity formula, we are able to calculate the perplexity of both model:

```
perplexity(backoff_model, test, 4)
```

```
Decimal('276.06023284316153914426214 91')
```

Figure 3: interpolation model perplexity

```
perplexity(interpolation_model, test, 4)
```

```
Decimal('635.37984395101645442220468 43')
```

Figure 4: back-off model perplexity

IV. Text generator

```
def generate(model, context=[]):
    sentence = ["<s>"]+context
    while sentence[-1] != "</s>" and len(sentence)<100:
        proba = get_proba_distrib(model, sentence)

        if sum(proba['probability'].tolist()) < 1:
            normalized_probabilities = proba['probability'].tolist() / np.sum(proba['probability'].tolist())

        w = np.random.choice(proba['word'].tolist(), 1, p = normalized_probabilities)
        sentence.append(w[0])

    result = ''
    for word in sentence:
        result = result + ' ' + word
    return result
```

Figure 5: Text generator function

Write a function call generate, it find the next possible with the most probability to appear, we can create a text generator with both models.

```
print("Generated sentence: ",generate(backoff_model, ['i', 'was', 'a']))
```

```
Generated sentence:   <s> i was a little shocked at first , and then i was a little frightened . </s>
```

Figure 6: Text generated with back-off model

```
print("Generated sentence: ",generate(interpolation_model, ['i', 'was', 'a']))
```

```
Generated sentence:   <s> i was a little frightened . </s>
```

Figure 7: Text generated with interpolation model