AMIT KUMAR 137

AI Practical Exam

Importing required libraries

```
import math
import nltk
from nltk.tokenize import sent_tokenize as st
from nltk.tokenize import word_tokenize as wt
```

Initializing the values

```
unigram_dict = {}
bigram_dict = {}
trigram_dict = {}
SENTENCE_START = "<s>"
SENTENCE_END = "</s>"
unigram_corpus_length = 0
bigram_corpus_length = 0
trigram_corpus_length = 0
unigram_unique_words = 0
bigram_unique_words = 0
trigram_unique_words = 0
bigram_unique_set = set()
trigram_unique_set = set()
```

```
def calcgram(s):
  return len(s) - 2

def calculate_unigram(s):
  res = 0
  for sentence in s:
    res += len(sentence) - 2
  return res

def calculate_bigram(s):
  res = 0
  for sentence in s:
    res += len(sentence) - 1
  return res

def calculate_trigram(s):
  res = 0
  for sentence in s:
    res += len(sentence) - 2
```

```
  return res
```

Preprocess defining

```
def preprocess_sentence(s):
  new_words = [SENTENCE_START]
  for word in wt(s):
      new_words.append(word)
  new_words.append(SENTENCE_END)
  return new_words

def preprocess_sentences(s):
  sentences = st(s)
  new_sentences = []
  for sentence in sentences:
    new_sentences.append(preprocess_sentence(sentence[:-1])[:])
  return new_sentences

def preprocess(s):
  unigram(s)
  bigram(s)
  trigram(s)

def unigram(s):
  global unigram_corpus_length
  for sentence in s:
    for word in sentence:
      try:
        unigram_dict[word] += 1
      except:
        unigram_dict[word] = 1
      if(word != SENTENCE_START and word != SENTENCE_END):
        unigram_corpus_length += 1
  unigram_unique_words = len(unigram_dict) - 2


def bigram(s):
  global bigram_corpus_length
  for sentence in s:
    previous_word = None
    for word in sentence:
      if(previous_word != None):
        try:
          bigram_dict[(previous_word,word)] += 1
        except:
          bigram_dict[(previous_word,word)] = 1
        if previous_word != SENTENCE_START and word != SENTENCE_END:
          bigram_unique_set.add((previous_word, word))
      previous_word = word
  bigram_unique_words = len(bigram_dict)

def trigram(s):
  global trigram_corpus_length
  for sentence in s:
```

```
    for sentence in s:
      first_word = None
      second_word = None
      for word in sentence:
        if(first_word != None and second_word != None):
          try:
            trigram_dict[(first_word,second_word,word)] += 1
          except:
            trigram_dict[(first_word,second_word,word)] = 1
          if first_word != SENTENCE_START and word != SENTENCE_END:
            trigram_unique_set.add((first_word,second_word,word))
        first_word = second_word
        second_word = word
    trigram_unique_words = len(trigram_dict)
```

Probability function

```
  def probability_unigram(word,smooth):
    try:
      result_numerator = unigram_dict[word]
    except:
      result_numerator = 0
    try:
      result_denumerator = unigram_unique_words
    except:
      result_denumerator = 0
    if(smooth):
      result_numerator += 1
      result_denumerator += 1
    return float(result_numerator) / float(result_denumerator)

  def probability_bigram(previous_word,word,smooth):
    try:
      result_numerator = bigram_dict[(previous_word,word)]
    except:
      result_numerator = 0
    try:
      result_denumerator = unigram_dict[previous_word]
    except:
      result_denumerator = 0
    if(smooth):
      result_numerator += 1
      result_denumerator += 1
    if(result_numerator == 0 or result_denumerator ==0):
      return 0.0
    return float(result_numerator) / float(result_denumerator)

  def probability_trigram(first_word,second_word,word,smooth):
    try:
      result_numerator = trigram_dict[(first_word,second_word,word)]
    except:
      result_numerator = 0
```

```
    try:
      result_denumerator = bigram_dict[(first_word,second_word)]
    except:
      result_denumerator = 0
    if(smooth):
      result_numerator += 1
      result_denumerator += 1
    if(result_numerator == 0 or result_denumerator ==0):
      return 0.0
    return float(result_numerator) / float(result_denumerator)

  def sentence_probability(gram_type,sentence,smooth):
    result = 0.0
    if(gram_type == "Unigram"):
      for word in sentence :
        if(word != SENTENCE_START and word != SENTENCE_END):
          if(result == 0.0):
            result = probability_unigram(word,smooth)
          else:
            result *= probability_unigram(word,smooth)

    elif(gram_type == "Bigram"):
      previous_word = None
      for word in sentence:
        if(previous_word != None  and word != SENTENCE_END):
          if(result == 0.0):
            result = probability_bigram(previous_word,word,smooth)
          else:
            result *= probability_bigram(previous_word,word,smooth)
        previous_word = word

    elif(gram_type =="Trigram"):
      first_word = None
      second_word = None
      for word in sentence:
        if(first_word != None and word != SENTENCE_END):
          if(result == 0.0):
            result = probability_trigram(first_word,second_word,word,smooth)
          else:
            result *= probability_trigram(first_word,second_word,word,smooth)
        previous_word = word

    return result


  nltk.download('punkt')
```

```
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    True
```

Train & test data

```
  train_sentences = "She is Beautiful. She is Wonderful. He is Greek."
```

```
sentences = preprocess_sentences(train_sentences)
preprocess(sentences)
input_sentences = preprocess_sentence("She is Wonderful")
sentence_probability("Bigram",input_sentences,False)
```

```
0.2222222222222222
```

```
train_sentences = "She is Beautiful. She is Wonderful. He is Greek."
sentences = preprocess_sentences(train_sentences)
preprocess(sentences)
input_sentences = preprocess_sentence("She is Wonderful")
sentence_probability("Trigram",input_sentences,False)
```

```
0.0
```

```
train_sentences = "She is Beautiful."
sentences = preprocess_sentences(train_sentences)
preprocess(sentences)
input_sentences = preprocess_sentence("She is Wonderful")
sentence_probability("Bigram",input_sentences,False)
```

```
0.21606648199445982
```

```
train_sentences = "She is Wonderful. He is Greek."
sentences = preprocess_sentences(train_sentences)
preprocess(sentences)
input_sentences = preprocess_sentence("She is Wonderful")
sentence_probability("Bigram",input_sentences,False)
```

```
0.2222222222222222
```