Eric Ortega Rodriguez

Markdown Text Generation Project - Testing

Natural Language Processing | ECE 684.01.Fa24

In [3]:

```python
# Importing libraries
import random


def building_ngram(corpus, n: int) -> dict:  # building n gram
    ngram = {}  # empty
    if n == 1:  # builds unigram
        for token in corpus:
            if token not in ngram:
                ngram[token] = 0
            ngram[token] += 1
    else:  # all other besides unigram
        for i in range(len(corpus) - n + 1):
            previous = tuple(corpus[i : i + n - 1])
            token = corpus[i + n - 1]
            if previous not in ngram:
                ngram[previous] = {}
            if token not in ngram[previous]:
                ngram[previous][token] = 0
            ngram[previous][token] += 1
    return ngram


# random sample
def rand_sample(dist: dict):
    pop = list(dist.keys())
    val = list(dist.values())
    total = sum(val)
    weights = [v / total for v in val]
    return random.choices(pop, weights)[0]


# deterministic sample
def determ_sample(dist: dict):
    max_count = max(dist.values())
    next_word = min(t for t, c in dist.items() if c == max_count)
    return next_word


def choose_sample(status: bool, dist: dict):
    if status:
        return rand_sample(dist)
    else:
        return determ_sample(dist)
```

```
def completing_sentence(sentence: list, n: int, corpus, randomize=False):
    prediction = list(sentence)
    ngram = building_ngram(corpus, n)
    while len(prediction) < 10 and prediction[-1] not in [
        ".",
        "?",
        "!",
    ]:  # adding our restrictions
        if n == 1:
            next_word = choose_sample(randomize, ngram)
        else:
            previous = tuple(prediction[-(n - 1) :])
            if previous in ngram:
                token_pool = ngram[previous]
                next_word = choose_sample(randomize, token_pool)
            else:
                alpha_value = 0.4  # given value from assignment
                temp = {}
                backoff_count = 0
                for i in range(n - 1, 1, -1):
                    backoff_count = n - i
                    new_gram = building_ngram(corpus, i)
                    previous = tuple(prediction[-(i - 1) :])
                    if previous not in new_gram:
                        continue
                    else:
                        # This is where we will be including backoff
                        backoff_pool = new_gram[previous]
                        backoff_word = choose_sample(randomize, backoff_pool
                        backoff_prob = (
                            backoff_pool[backoff_word]
                            * (alpha_value**backoff_count)  # multiplying
                        ) / sum(backoff_pool.values())
                        if backoff_word in temp:
                            if backoff_prob > temp[backoff_word]:
                                temp[backoff_word] = backoff_prob
                        else:
                            temp[backoff_word] = backoff_prob

                unigram = building_ngram(corpus, 1)
                unigram_word = choose_sample(randomize, unigram)
                unigram_prob = (
                    unigram[unigram_word] * (alpha_value ** (backoff_count +
                ) / sum(unigram.values())
                if temp == {}:
                    next_word = unigram_word
                else:
                    if unigram_word in temp:
                        if unigram_prob > temp[unigram_word]:
                            temp[unigram_word] = unigram_prob
                    else:
                        temp[unigram_word] = unigram_prob
                next_word = choose_sample(randomize, temp)
        prediction.append(next_word)  # appending the prediciton with the ne
    return prediction
```

Testing Simple Case

In [6]:

```
import random
import nltk

sentence = ["she", "was", "not"]
n = 3
corpus = nltk.word_tokenize(nltk.corpus.gutenberg.raw("austen-sense.txt").lc
randomize = False

test_result = completing_sentence(sentence, n, corpus, randomize)
print(test_result)
```

```
['she', 'was', 'not', 'in', 'the', 'world', ',', 'and', 'the', 'two']
```

Testing Deterministic

In [7]:

```
import nltk

random.seed(32)
sentence = ["we", "are", "the"]
n = 3
corpus = nltk.word_tokenize(nltk.corpus.gutenberg.raw("austen-sense.txt").lc
randomize = False

test_result = completing_sentence(sentence, n, corpus, randomize)
print(test_result)
```

```
['we', 'are', 'the', 'most', 'affectionate', 'and', 'graceful', ',', 'i', 'a
m']
```

In [17]:

```
import nltk
import random

random.seed(23)
sentence = ["never", "say", "never"]
n = 3
corpus = nltk.word_tokenize(nltk.corpus.gutenberg.raw("austen-sense.txt").lc
randomize = False

test_result = completing_sentence(sentence, n, corpus, randomize)
print(test_result)
```

```
['never', 'say', 'never', 'was', 'a', 'great', 'deal', 'of', 'the', 'house']
```

Testing Stochastic

In [18]:

```
import nltk

random.seed(4)
```

```
sentence = ["they", "have", "no"]
n = 6
corpus = nltk.word_tokenize(nltk.corpus.gutenberg.raw("austen-sense.txt").lo
randomize = True

test_result = completing_sentence(sentence, n, corpus, randomize)
print(test_result)
```

```
['they', 'have', 'no', 'mind', 'to', 'keep', 'them', ',', 'little', 'as']
```

In [20]:

```
import nltk
import random

random.seed(20)
sentence = ["let", "the", "boy"]
n = 6
corpus = nltk.word_tokenize(nltk.corpus.gutenberg.raw("austen-sense.txt").lo
randomize = True

test_result = completing_sentence(sentence, n, corpus, randomize)
print(test_result)
```

```
['let', 'the', 'boy', 'were', 'most', 'like', 'his', 'father', 'or', 'mothe
r']
```

In [ ]: