Project1

March 9, 2023

[76]:
```
# Manuel Aragon
# CSCE 4290 INTRODUCTION TO NATURAL LANGUAGE PROCESSING
# Project 1
```

[77]:
```
import urllib3
import nltk

# Open corpus file
with open("corpus.txt", "r") as file:
    text = file.read()
# Create tokens from words in file
tokens = [t for t in text.split()]
```

[78]:
```
# Generate the Unigram Model
unigram_freq = nltk.FreqDist(tokens)
tl = len(tokens)
unigram_probs = {k: v/tl for (k,v) in unigram_freq.items()}

from operator import itemgetter
unigram_sprobs = dict(sorted(unigram_probs.items(), key=itemgetter(1),␣
 ↪reverse=True))
unigram_pvals = list(unigram_sprobs.values())
unigram_cumprobs = {k: sum(unigram_pvals[0:ix+1]) for ix, (k,v) in␣
 ↪enumerate(unigram_sprobs.items())}

import random
random.seed(42)
random.random()

unigram_keys = list(unigram_cumprobs.keys())
unigram_values = unigram_cumprobs.values()
```

[79]:
```
import random
print("Unigram Model:")

# Generate 5 sentences
for i in range(5):
    sentence = []
    # Generate 10 words for each sentence
    for j in range(10):
        random_number = random.random()
        # Find the index of the word in the unigram_values list that has a␣
↪value greater than the random number
        ix = [i for (i, v) in enumerate(unigram_values) if v > rn][0]
        # Add the corresponding word from the unigram_keys list to the sentence
        sentence.append(unigram_keys[ix])

    # Join the words in the sentence with a space, and print the sentence
    print(" ".join(sentence))
```

```
Unigram Model:
have have have have have have have have have have
have have have have have have have have have have
have have have have have have have have have have
have have have have have have have have have have
have have have have have have have have have have
```

[80]:
```
# Generate the bigram model
bigrams = nltk.bigrams(tokens)
bigram_freq = nltk.FreqDist(bigrams)
bigram_probs = {k: v/tl for (k,v) in bigram_freq.items()}
bigram_sprobs = dict(sorted(bigram_probs.items(), key=itemgetter(1),␣
↪reverse=True))
bigram_pvals = list(bigram_sprobs.values())
bigram_cumprobs = {k: sum(bigram_pvals[0:ix+1]) for ix, (k,v) in␣
↪enumerate(bigram_sprobs.items())}



import random
random.seed(42)
random.random()

bigram_keys = list(bigram_cumprobs.keys())
bigram_values = bigram_cumprobs.values()
```

[81]:
```
import random
print("Bigram Model:")

num_sentences = 5
num_words = 10

# Loop over the range of number of sentences
for i in range(num_sentences):
    # Set the current word to the first bigram key
    current_word = bigram_keys[0][0]
    sentence = [current_word]

    # Loop over the range of number of words in each sentence
    for j in range(num_words-1):
        next_word = None
        random_number = random.random()
        threshold = 0.1

        while next_word is None:

            # Loop over the bigram keys and values
            for k, (bigram, prob) in enumerate(zip(bigram_keys, bigram_values)):
                # If the current word matches the first word in the bigram and␣
↪the random value is less than or equal to the probability
                if current_word == bigram[0] and random_number <= prob:
                    # Set the next word as the second word in the bigram
                    next_word = bigram[1]
                    break

            random_number = random.random()
            # Increase the threshold by 1.5
            threshold *= 1.5
            if threshold > 1:
                break
        if next_word is None:
            break

        # Add the next word to the sentence
        sentence.append(next_word)
        # Set the current word to the next word
        current_word = next_word

    # Print the sentence joined with spaces
    print(" ".join(sentence))
```

```
Bigram Model:
of blockchain protocols. Some of custodying the fourth industrial changes.
of each blockchain digital assets of achieving close-to-real data, unable
of other stakeholders and biasing them as network regardless of
of node that each category is even illegal in recent
of Solidity language. The system by looking at least to
```

[82]:
```
import nltk
from operator import itemgetter

# Generate the trigram model
```

```
trigrams = nltk.trigrams(tokens)
trigram_freq = nltk.FreqDist(trigrams)
trigram_probs = {k: v/tl for (k,v) in trigram_freq.items()}
trigram_sprobs = dict(sorted(trigram_probs.items(), key=itemgetter(1),␣
 ↪reverse=True))
trigram_pvals = list(trigram_sprobs.values())
trigram_cumprobs = {k: sum(trigram_pvals[0:ix+1]) for ix, (k,v) in␣
 ↪enumerate(trigram_sprobs.items())}

import random
random.seed(42)
random.random()

trigram_keys = list(trigram_cumprobs.keys())
trigram_values = trigram_cumprobs.values()
```

```
[83]: import random

print("Trigram Model:")

num_sentences = 5
num_words = 10

# Loop over the number of sentences to generate
for i in range(num_sentences):
    current_trigram = trigram_keys[0]
    # Create a list for the sentence and add the first three words from the␣
 ↪current trigram
    sentence = list(current_trigram[:3])

    # Loop over the number of words in each sentence
    for j in range(num_words-3):
        next_trigram = None
        random_number = random.random()
        threshold = 0.1

        # Keep looping until a next trigram is found
        while next_trigram is None:
            # Loop over the trigram keys and values
            for k, (trigram, prob) in enumerate(zip(trigram_keys,␣
 ↪trigram_values)):
                # If the current trigram's last two words match the next␣
 ↪trigram's first two words
                # and the random value is less than or equal to the␣
 ↪probability, set the next trigram
```

```
                if current_trigram[1:] == trigram[:-1] and random_number <=␣
↪prob:
                    next_trigram = trigram
                    break
            random_number = random.random()
            # Double the threshold value
            threshold *= 2
            if threshold > 1:
                continue
        # Add the last word of the next trigram to the sentence
        sentence.append(next_trigram[-1])
        # Set the current trigram to the next trigram
        current_trigram = next_trigram
    if len(sentence) < num_words:
        continue
    print(" ".join(sentence))
```

```
Trigram Model:
of the blockchain is now rapidly extending its application to
of the blockchain perform equivalent calculations and store equivalent data.
of the blockchain such as roulettes, card games, etc., but
of the blockchain has complete database information, all information is
of the blockchain will continue to improve the privacy security
```

[84]:
```
import nltk
from operator import itemgetter

# Generate the 4-gram model
fourgrams = nltk.ngrams(tokens, 4)
fourgram_freq = nltk.FreqDist(fourgrams)
fourgram_probs = {k: v/tl for (k,v) in fourgram_freq.items()}
fourgram_sprobs = dict(sorted(fourgram_probs.items(), key=itemgetter(1),␣
↪reverse=True))
fourgram_pvals = list(fourgram_sprobs.values())
fourgram_cumprobs = {k: sum(fourgram_pvals[0:ix+1]) for ix, (k,v) in␣
↪enumerate(fourgram_sprobs.items())}

import random
random.seed(42)
random.random()



fourgram_keys = list(fourgram_cumprobs.keys())
fourgram_values = fourgram_cumprobs.values()
```

[85]:
```
import random
```

```
print("4-gram Model:")

num_sentences = 5
num_words = 10

# Loop through the number of sentences to generate
for i in range(num_sentences):
    # Get the first fourgram to start the sentence
    current_fourgram = fourgram_keys[0]
    # Start the sentence with the first three words of the fourgram
    sentence = list(current_fourgram[:3])

    # Loop through to generate the remaining words in the sentence
    for j in range(num_words-3):
        next_fourgram = None
        random_number = random.random()
        threshold = 0.1

        # Loop until a next fourgram is found
        while next_fourgram is None:
            # Loop through the fourgram keys and values
            for k, (fourgram, prob) in enumerate(zip(fourgram_keys,␣
 ↪fourgram_values)):
                # If the current fourgram's last three words match the next␣
 ↪fourgram's first three words and the random value is less than or equal to␣
 ↪the probability
                if current_fourgram[1:] == fourgram[:-1] and random_number <=␣
 ↪prob:
                    next_fourgram = fourgram
                    break
            # Get a new random value
            random_number = random.random()
            # Double the threshold
            threshold *= 2
            # If the threshold is greater than 1, continue to the next␣
 ↪iteration of the loop
            if threshold > 1:
                continue
        # Add the last word of the next fourgram to the sentence
        sentence.append(next_fourgram[-1])
        # Set the current fourgram as the next fourgram
        current_fourgram = next_fourgram

    if len(sentence) < num_words:
        continue
    print(" ".join(sentence))
```

```
4-gram Model:
At the same and count the average time to execute
At the same because the potential of blockchain applications is
At the same because the potential of blockchain applications is
At the same because the potential of blockchain applications is
At the same we can learn from international methods, such
```