```python
import nltk
import re
import math
from collections import Counter, defaultdict

text = """Natural language processing is an important area of
artificial intelligence.
It focuses on enabling computers to understand and generate human
language.
Language is complex because words can have different meanings
depending on context.
This makes language modeling a challenging task in natural language
processing.

A language model predicts the probability of a sequence of words.
It helps systems decide which word is most likely to come next.
Unigram models treat every word independently and ignore context.
Bigram models consider the previous word to capture limited context.
Trigram models consider two previous words and provide better
predictions.

However, higher order models require more data.
With small datasets, many word combinations may not appear at all.
This leads to the zero probability problem.
Smoothing techniques such as add one smoothing are used to handle
unseen words.

Perplexity is used to evaluate the quality of a language model.
It measures how well the model predicts test sentences.
Lower perplexity indicates that the model performs better.
Language models are widely used in applications such as text
prediction, chatbots, and speech recognition.
"""

with open("dataset.txt", "w") as f:
    f.write(text)

print("dataset.txt created successfully")
```

```
dataset.txt created successfully
```

```python
with open("dataset.txt", "r") as f:
    data = f.read()

print(data[:300])
```

```
Natural language processing is an important area of artificial
intelligence.
It focuses on enabling computers to understand and generate human
language.
Language is complex because words can have different meanings
```

```
depending on context.
This makes language modeling a challenging task in natural l
```

```python
import re

text = data.lower()
text = re.sub(r'[^a-z\s]', '', text)

tokens = text.split()
print(tokens[:20])
```

```
['natural', 'language', 'processing', 'is', 'an', 'important', 'area',
'of', 'artificial', 'intelligence', 'it', 'focuses', 'on', 'enabling',
'computers', 'to', 'understand', 'and', 'generate', 'human']
```

```python
sentences = data.split(".")

processed_sentences = []

for sent in sentences:
    sent = sent.lower()
    sent = re.sub(r'[^a-z\s]', '', sent)
    words = sent.split()
    if len(words) > 0:
        processed_sentences.append(["<s>"] + words + ["</s>"])

print(processed_sentences[0])
```

```
['<s>', 'natural', 'language', 'processing', 'is', 'an', 'important',
'area', 'of', 'artificial', 'intelligence', '</s>']
```

```python
from collections import Counter

unigrams = Counter(tokens)

print("Total unique words:", len(unigrams))
print("Top 10 unigrams:", unigrams.most_common(10))
```

```
Total unique words: 114
Top 10 unigrams: [('language', 8), ('to', 6), ('the', 6), ('models',
5), ('is', 4), ('of', 4), ('and', 4), ('words', 4), ('a', 4),
('model', 4)]
```

```python
bigrams = Counter()

for sent in processed_sentences:
    for i in range(len(sent) - 1):
        bigrams[(sent[i], sent[i+1])] += 1

print("Top 10 bigrams:", bigrams.most_common(10))
```

```
Top 10 bigrams: [(('<s>', 'it'), 3), (('context', '</s>'), 3),
(('natural', 'language'), 2), (('language', 'processing'), 2),
(('<s>', 'language'), 2), (('<s>', 'this'), 2), (('a', 'language'),
2), (('language', 'model'), 2), (('model', 'predicts'), 2), (('of',
'a'), 2)]
```

```
trigrams = Counter()

for sent in processed_sentences:
    for i in range(len(sent) - 2):
        trigrams[(sent[i], sent[i+1], sent[i+2])] += 1

print("Top 10 trigrams:", trigrams.most_common(10))
```

```
Top 10 trigrams: [(('natural', 'language', 'processing'), 2), (('a',
'language', 'model'), 2), (('<s>', 'natural', 'language'), 1),
(('language', 'processing', 'is'), 1), (('processing', 'is', 'an'),
1), (('is', 'an', 'important'), 1), (('an', 'important', 'area'), 1),
(('important', 'area', 'of'), 1), (('area', 'of', 'artificial'), 1),
(('of', 'artificial', 'intelligence'), 1)]
```

```
V = len(unigrams)
print("Vocabulary size:", V)
```

```
Vocabulary size: 114
```

```
total_words = sum(unigrams.values())

def unigram_prob(word):
    return (unigrams[word] + 1) / (total_words + V)

def bigram_prob(w1, w2):
    return (bigrams[(w1, w2)] + 1) / (unigrams[w1] + V)

def trigram_prob(w1, w2, w3):
    return (trigrams[(w1, w2, w3)] + 1) / (bigrams[(w1, w2)] + V)

print(unigram_prob("language"))
print(bigram_prob("natural", "language"))
print(trigram_prob("natural", "language", "processing"))
```

```
0.030927835051546393
0.02586206896551724
0.02586206896551724
```

```
test_sentences = [
    "natural language processing is important",
    "language models predict word sequences",
    "bigram models capture context",
    "trigram models need more data",
    "perplexity evaluates language models"
]
```

```python
def sentence_prob_unigram(sentence):
    words = sentence.lower().split()
    prob = 1
    for w in words:
        prob *= unigram_prob(w)
    return prob

def sentence_prob_bigram(sentence):
    words = ["<s>"] + sentence.lower().split() + ["</s>"]
    prob = 1
    for i in range(len(words)-1):
        prob *= bigram_prob(words[i], words[i+1])
    return prob

def sentence_prob_trigram(sentence):
    words = ["<s>"] + sentence.lower().split() + ["</s>"]
    prob = 1
    for i in range(len(words)-2):
        prob *= trigram_prob(words[i], words[i+1], words[i+2])
    return prob

for s in test_sentences:
    print("\nSentence:", s)
    print("Unigram Probability :", sentence_prob_unigram(s))
    print("Bigram Probability  :", sentence_prob_bigram(s))
    print("Trigram Probability :", sentence_prob_trigram(s))
```

```
Sentence: natural language processing is important
Unigram Probability : 3.881683074821864e-10
Bigram Probability  : 1.4175617711784295e-11
Trigram Probability : 5.915136016778164e-10

Sentence: language models predict word sequences
Unigram Probability : 1.2938943582739547e-10
Bigram Probability  : 2.3640023122740074e-12
Trigram Probability : 1.0119512986536973e-10

Sentence: bigram models capture context
Unigram Probability : 1.3387493626941181e-08
Bigram Probability  : 7.622310348992505e-10
Trigram Probability : 1.1636559976866518e-08

Sentence: trigram models need more data
Unigram Probability : 2.3002566369314743e-11
Bigram Probability  : 6.802519533425805e-12
Trigram Probability : 2.0237495611941773e-10

Sentence: perplexity evaluates language models
Unigram Probability : 2.2591395495463247e-08
```

```
Bigram Probability  : 1.8276183393511464e-10
Trigram Probability : 5.818279988433258e-09
```

```python
import math

def perplexity_unigram(sentence):
    words = sentence.lower().split()
    N = len(words)
    log_prob = 0

    for w in words:
        log_prob += math.log(unigram_prob(w))

    return math.exp(-log_prob / N)

def perplexity_bigram(sentence):
    words = ["<s>"] + sentence.lower().split() + ["</s>"]
    N = len(words) - 1
    log_prob = 0

    for i in range(len(words)-1):
        log_prob += math.log(bigram_prob(words[i], words[i+1]))

    return math.exp(-log_prob / N)

def perplexity_trigram(sentence):
    words = ["<s>"] + sentence.lower().split() + ["</s>"]
    N = len(words) - 2
    log_prob = 0

    for i in range(len(words)-2):
        log_prob += math.log(trigram_prob(words[i], words[i+1],
words[i+2]))

    return math.exp(-log_prob / N)

for s in test_sentences:
    print("\nSentence:", s)
    print("Unigram Perplexity :", perplexity_unigram(s))
    print("Bigram Perplexity  :", perplexity_bigram(s))
    print("Trigram Perplexity :", perplexity_trigram(s))
```

```
Sentence: natural language processing is important
Unigram Perplexity : 76.2422993028724
Bigram Perplexity  : 64.28006904208608
Trigram Perplexity : 70.08209128014377

Sentence: language models predict word sequences
Unigram Perplexity : 94.97739114901479
Bigram Perplexity  : 86.6413779062177
```

```
Trigram Perplexity : 99.76267314959178

Sentence: bigram models capture context
Unigram Perplexity : 92.96622166783268
Bigram Perplexity  : 66.61663240365051
Trigram Perplexity : 96.28172153298698

Sentence: trigram models need more data
Unigram Perplexity : 134.16670112059145
Bigram Perplexity  : 72.64767175801141
Trigram Perplexity : 86.84976476402402

Sentence: perplexity evaluates language models
Unigram Perplexity : 81.56695227961035
Bigram Perplexity  : 88.63860215158093
Trigram Perplexity : 114.49890829173877
```