Data Processing

```
import re
from sklearn.model_selection import train_test_split

def load_text(file_path):
    """Loads the Wikipedia text"""
    with open(file_path, 'r', encoding='utf-8') as f:
        text = f.read()
    return text

def preprocess_text(text, convert_tr_chars=False):
    """Converts the text to lowercase and optionally transforms
Turkish characters."""
    text = text.lower()
    if convert_tr_chars:
        tr_chars = {'ş': 's', 'ç': 'c', 'ğ': 'g', 'ü': 'u', 'ö': 'o',
'ı': 'i'}
        text = ''.join([tr_chars.get(c, c) for c in text])
    text = re.sub(r'[^a-zA-Z0-9\s.,!?]', '', text)  # Remove special
character
    return text

def split_data(text, train_ratio=0.95):
    """spilit data test and train"""
    sentences = text.split('\n')
    train_data, test_data = train_test_split(sentences,
train_size=train_ratio, random_state=42)
    return train_data, test_data

# 1. Load text
text = load_text('wiki_00')

# 2. preprocessing
processed_text = preprocess_text(text, convert_tr_chars=True)

# 3. spilit data test and train
train_data, test_data = split_data(processed_text)

# Check
print(f"Train Date Set: {train_data[:3]}")
print(f"Test Data Set: {test_data[:3]}")
```

```
Eğitim seti örneği: ['', 'octavianus, mo 31 yilinda ilk zaferini,
agrippanin komutasindaki donanmanin birlikleri adriyatik denizinin
karsi kiyisina basarili bir sekilde gecirmesiyle kazandi. agrippa,
antonius ve kleopatranin ana guclerinin tedarik rotalarini keserken,
octavianus da corcyra bugunku korfu adasi karsisinda anakaraya cikti
ve guneye dogru yoneldi. karada ve denizde kistirilan antoniusun
ordusundamn askerler gun be gun kacarak octavianusun tarafina gecmeye
```

```
basladilar. octavianusun birlikleri ise rahatca hazirlik yapiyorlardi.
antoniusun donanmasi deniz kusatmasini kirabilmek icin umutsuzca
yunanistanin bati kiyilarindaki aktium koyuna dogru yelken acti.
burada 2 eylul mo 31 tarihinde aktium savasinda antoniusun donanmasi,
agrippa ve gaius sosiusun komutasi altindaki kucuk ve manevra
kabiliyeti yuksek gemilerden olusan sayica daha buyuk olan filo ile
karsilasti. antonius ve kalan birlikleri yakinda bekleyen kleopatranin
donanmasinin son andaki cabasiyla kurtuldular. octavianus onlari takip
etti ve 1 agustos mo 30 tarihinde iskenderiyede bir kere daha
antoniusu yenilgiye ugratti. kleopatra ve antonius intihar ettiler.
antonius, sevgilisinin kollarinda kilicyla kendisini oldurdu.
kleopatra ise kendisini zehirli bir yilana sokturttu. sezarin vrisi
olmasindan gelen konumunu siyasi kariyerinde ilerlemek icin kullanan
octavianus bir baskasinin daha ayni konumda bulunmasinin
tehlikelerinin farkindaydi. soylendigine gore iki sezar gereginden
fazla diyerek caesarionun oldurulmesini emretti. ote yandan
kleopatranin antoniustan olan cocuklarinin hayatlarini bagisladi.',
'bahsin 2006 yili nufusu 31,341 kisi ve 6,774 hanedir.']
Test seti örneği: ['', 'rakun kopeginin dogal yayilimi sibiryanin
dogusu, kuzeydogu cin ve japonyada dir. ancak, 19ncu yuzyilda kurk
uretimi icin rusyanin avrupa kismina da getirilmis ve yetistirilmeye
baslanmistir ancak rakun kopeginin kurk uretimi icin degerli olan kis
postunu yalnizca hur yasarken gelistirdigini tespit etmislerdir. bu
yuzden ukraynada 1928 ve 1950 yillari arasinda toplam 10.000 civarinda
rakun kopegi dogaya salinmistir. boylece rakun kopegi kendiliginden
batiya dogru yayilmaya baslamistir. 1931 yilinda finlandiyaya, 1951
yilinda romanyaya, 1955 yilinda polonyaya ve 1960 yilinda almanyaya
ulasmistir. son yillarda fransa ve italyada gozlendigi
bildirilmistir.', '']
```

Generate N-gram Tables

```python
from collections import defaultdict, Counter

def generate_ngrams(text, n):
    """Generates n-gram character sequences from the given text."""
    ngrams = [text[i:i+n] for i in range(len(text) - n + 1)]
    return ngrams

def build_ngram_model(data, n):
    """Creates an n-gram model for the given data."""
    ngram_counts = defaultdict(int)
    for sentence in data:
        ngrams = generate_ngrams(sentence, n)
        for ngram in ngrams:
            ngram_counts[ngram] += 1
    return ngram_counts

# Generate 1, 2, and 3-gram tables from the training data.
```

```
unigram_model = build_ngram_model(train_data, 1)
bigram_model = build_ngram_model(train_data, 2)
trigram_model = build_ngram_model(train_data, 3)

# Example Outputs
print(f"1-gram example: {list(unigram_model.items())[:10]}")
print(f"2-gram example: {list(bigram_model.items())[:10]}")
print(f"3-gram example: {list(trigram_model.items())[:10]}")
```

```
1-gram example: [('o', 12129803), ('c', 6946004), ('t', 13818590),
('a', 37173448), ('v', 3972088), ('i', 42985897), ('n', 23318795),
('u', 15076034), ('s', 16044284), (',', 2725476)]
2-gram example: [('oc', 792350), ('ct', 93884), ('ta', 2413665),
('av', 434543), ('vi', 362487), ('ia', 482149), ('an', 5635878),
('nu', 1102275), ('us', 1208659), ('s,', 131634)]
3-gram example: [('oct', 3636), ('cta', 25327), ('tav', 14928),
('avi', 45873), ('via', 4946), ('ian', 44727), ('anu', 32061), ('nus',
90131), ('us,', 21790), ('s, ', 129673)]
```

Apply Good Turing Smoothing

```
from collections import defaultdict

def compute_frequency_of_frequencies(ngram_counts):
    """Calculate the frequency of frequencies based on n-gram
frequencies."""
    freq_of_freqs = defaultdict(int)
    for freq in ngram_counts.values():
        freq_of_freqs[freq] += 1
    return freq_of_freqs

def good_turing_smoothing(ngram_counts, total_ngrams):
    """Calculate the adjusted probabilities using Good-Turing
smoothing."""
    smoothed_probs = {}
    freq_of_freqs = compute_frequency_of_frequencies(ngram_counts)

    for ngram, count in ngram_counts.items():
        c_next = freq_of_freqs.get(count + 1, 0)
        if c_next > 0:
            adjusted_count = (count + 1) * c_next /
freq_of_freqs[count]
        else:
            adjusted_count = count  # Eğer c+1 frekansı yoksa,
orijinal değeri al
        smoothed_probs[ngram] = adjusted_count / total_ngrams

    return smoothed_probs

# 1. Apply smoothing for each n-gram model.
```

```
total_unigrams = sum(unigram_model.values())
total_bigrams = sum(bigram_model.values())
total_trigrams = sum(trigram_model.values())

unigram_probs = good_turing_smoothing(unigram_model, total_unigrams)
bigram_probs = good_turing_smoothing(bigram_model, total_bigrams)
trigram_probs = good_turing_smoothing(trigram_model, total_trigrams)

# 2. Example Outputs
print(f"1-gram good turing smoothing: {list(unigram_probs.items())
[:5]}")
print(f"2-gram good turing smoothing: {list(bigram_probs.items())
[:5]}")
print(f"3-gram good turing smoothing: {list(trigram_probs.items())
[:5]}")
```

```
1-gram good turing smoothing: [('o', 0.031125663001181834), ('c',
0.017823783264152022), ('t', 0.03545917237827368), ('a',
0.09538887111686453), ('v', 0.010192570522294414)]
2-gram good turing smoothing: [('oc', 0.00204499011283994), ('ct',
0.00024230687417664532), ('ta', 0.006229470638868952), ('av',
0.0011215197054380084), ('vi', 0.0009355491020799033)]
3-gram good turing smoothing: [('oct', 9.438859491704645e-06), ('cta',
6.574752319758072e-05), ('tav', 3.875228121346725e-05), ('avi',
0.00011908382878519448), ('via', 1.2839548692511323e-05)]
```

Calculate Perplexity

```
import math

def calculate_perplexity(test_data, ngram_probs, n):
    """Verilen test verisi için perplexity hesaplar."""
    log_prob_sum = 0
    total_ngrams = 0

    for sentence in test_data:
        ngrams = generate_ngrams(sentence, n)
        for ngram in ngrams:
            prob = ngram_probs.get(ngram, 1e-8)  #A very small
probability for unseen n-grams.
            log_prob_sum += math.log(prob)
            total_ngrams += 1

    perplexity = math.exp(-log_prob_sum / total_ngrams)
    return perplexity

# 1. Calculate perplexity (for each model)
unigram_perplexity = calculate_perplexity(test_data, unigram_probs, 1)
bigram_perplexity = calculate_perplexity(test_data, bigram_probs, 2)
trigram_perplexity = calculate_perplexity(test_data, trigram_probs, 3)
```

```
# 2. Print the results.
print(f"1-gram perplexity: {unigram_perplexity}")
print(f"2-gram perplexity: {bigram_perplexity}")
print(f"3-gram perplexity: {trigram_perplexity}")
```

```
1-gram perplexity: 20.893676782582887
2-gram perplexity: 242.4144641543308
3-gram perplexity: 2006.9914560059249
```

Generate Sentance

```
import random

def generate_sentence(ngram_probs, n, max_length=50):
    """N-gram modeli kullanarak rastgele bir cümle üretir."""
    sentence = ""
    current_ngram = random.choice(list(ngram_probs.keys()))  # Random
start

    for _ in range(max_length):
        sentence += current_ngram[-1]  # Append the last character of
the n-gram.

        # Starting from the current n-gram, find the appropriate n-
grams for the next step.
        candidates = [ngram for ngram in ngram_probs if
ngram.startswith(current_ngram[1:])]

        if not candidates:
            break  # If there are no suitable n-grams left, end the
sentence.

        # Make a random selection from the first 5 n-grams
        next_ngram = random.choice(candidates[:5])
        current_ngram = next_ngram

    return sentence

# Generate random sentences for each model.
print("1-gram :", generate_sentence(unigram_probs, 1))
print("1-gram :", generate_sentence(unigram_probs, 1))
print("2-gram :", generate_sentence(bigram_probs, 2))
print("2-gram :", generate_sentence(bigram_probs, 2))
print("3-gram :", generate_sentence(trigram_probs, 3))
print("3-gram :", generate_sentence(trigram_probs, 3))
```

```
1-gram :  avaoavottvoavtacotvcoaatcccattvttcttvcttactvocvoa
1-gram : hvovaavtvaavcoacacvvcctvattavoaooaaoaooatoavttavcv
2-gram : 64 senugutini,35730 z ia ia yrmavavgagunippinagrlk
```

```
2-gram : ldipaftopi,54 mendon moplaviaftrliagrs 392 ilagugi
3-gram : aysaynanmeklon tir klikisek dri asinium decelkilis
3-gram :  akaltarindi. agrona asi,olulundeclanizdilislyyist
```