Natural Languages

1. Structural information is more important when considering processing difficulty. Long-distance dependencies within a construction are significantly correlated to how long a human will take to parse a sentence. Whereas Zipf's law suggests that infrequent constructions/words are commonplace in natural language, and so are not on their own a reliable indicator of processing difficulty. Furthermore, structurally difficult-to-parse constructions are likely to be very infrequent in a corpus, but the inverse relationship does not hold (i.e., not all infrequent constructions are structurally difficult). Therefore, the structural information alone likely contains much of the information to be found from the frequency information.

2. Examples:

   I said (that) I didn't (did not) steal the jewellery.
   All (of) the time I'm (I am) thinking of ways *that I can* steal the jewellery.

   (In these examples, the bracketed words are omitted because they exist in low-information-frequency parts of the sentences, whereas the starred words are more verbose than necessary because they exist in high-information-frequency parts of the sentence)

   Counter-Examples:

   The fox *that was* crawling past my bed told us *that* we were special.

   (In this example, the starred words sound natural to include even though they create significant drops in information rate)

Formal Models and Learnability

(See attached)

Information Theory

1. (See attached)

2.

```
#!/usr/bin/python3.9

import numpy as np
import warnings

with open("alice.txt") as f:
    data = f.read()

alphabet = list(set(data))
```

```
alphabet_idx_lookup = {}
for i, char in enumerate(alphabet):
    alphabet_idx_lookup[char] = i


# Generate bigram probability matrix
bigram_matrix = np.zeros((len(alphabet), len(alphabet)), dtype=float)
first_letter_matrix = np.zeros(len(alphabet), dtype=float)
all_letter_matrix = np.zeros(len(alphabet), dtype=float)

last_char = None
for char in data:
    char = alphabet_idx_lookup[char]

    if last_char is not None:
        bigram_matrix[last_char, char] += 1

    all_letter_matrix[char] += 1
    if last_char is None or alphabet[last_char] == " ":
        first_letter_matrix[char] += 1

    last_char = char

# Normalize distributions
bigram_matrix /= np.sum(bigram_matrix, axis=1)[:, np.newaxis]
first_letter_matrix /= np.sum(first_letter_matrix)
all_letter_matrix /= np.sum(all_letter_matrix)

# Helpful pre-computations
with warnings.catch_warnings():
    # Catch divide-by-zero warnings
    # If the value really is zero, we'll never pick it anyway
    warnings.simplefilter("ignore")
    H_cond = -np.log2(bigram_matrix) * bigram_matrix
    H_first = -np.log2(first_letter_matrix) * first_letter_matrix
    H = -np.log2(all_letter_matrix) * all_letter_matrix

def generate_word():
    # Pick the first letter of the word according to first_letter_matrix
    current = np.random.choice(np.arange(len(alphabet)), p=first_letter_matrix)
    word = ""

    entropy = H_first[current]

    while (alphabet[current] != " "):
        word += alphabet[current]
        # Pick the next letter according to bigram_matrix
        nxt = np.random.choice(np.arange(len(alphabet)), p=bigram_matrix[current, :])

        # Add to entropy in accordance with the chain rule
        # NOTE: Assuming P(X_n|X_n-1, X_n-2, ..., X_1) = P(X_n|X_n-1)
        #       (Markov assumption)
        entropy += H_cond[current, nxt]

        current = nxt


    return word, entropy

words, entropies = zip(*list(generate_word() for _ in range(10000)))

# Sort words by entropy
words = list(np.array(words, dtype="object")[np.argsort(entropies)[::-1]])

print(words[:10])

# ['lokendinfisemakitheerminthaing', 'soprerokishaimucerulitinlishormad', 'ssatheastomaishersemeridouryo',
'thtoundshengrathechithey', 'appouteyithithemigucryphale', 'ishooushewilinioupotoumits',
'waingoulickithicoutim', 'waderplousoulingereeme', 'hesklloflriteryoventhntllld', 'sentheesearithailyore']
```

3.

a. The answer to the question can be considered the input, and the question itself can be considered the output after passing the answer through a noisy channel. Therefore, the answer with maximum likelihood is the one which maximises P(answer)P(question|answer)

b. The input to the channel can be some encoding of the "sense" of the word, and the output will be the word in context. The correct sense of the word will be the one which maximises P(sense)P(word in context|sense).

Y2021P7Q5

a-e. (See attached)

f. One hypothesis might be that the amount of time it takes a human to process a construction of the form ab^kc^jd (e.g., a highly highly highly contagious contagious virus) does not significantly depend on k or j above some threshold, perhaps 2. I believe this to be the case because past that threshold, the human would stop reading each word individually, but instead recognise it as adding no meaning, only emphasis, and can quickly find where the repetition ends.

Another hypothesis might be that humans might process language more like the inductive method from part (c) than the grammatical method in part (a). I believe that humans are more likely to think that, for example. "contagious" can be replaced with "highly contagious" in a valid sentence and the sentence will still be valid, as opposed to thinking that the word "contagious" needs specific types of words before and after it.