CS 4740 (+crosslists): Natural Language Processing **(Due: February 21, 11.59 p.m.)**

Homework #1

Instructors: Claire Cardie, Tanya Goyal

Submission by (netIDs): aeh245

Logistics: Read all the instructions carefully before you start working on the assignment, and before you make a submission.

- **This assignment has two submission dates.** A milestone submission is due on February 12, 11.59 p.m. The entire assignment is due on February 21, 11.59 p.m. **Please read the submission instructions included at the end of this document carefully.**
- You can do this homework either individually or in a group of 2. We strongly recommend doing this in a group. All students in a group will receive the same grade. You will **declare your group** as part of the milestone submission (deadline February 12, 11.59 p.m.).
- This assignment is worth 16% of your grade. Make sure to submit this assignment by the due date mentioned above. Remember that you have 5 slip days to use throughout the course. You may use a maximum of 2 slip days for each assignment. Refer to the course webpage for more information about late submission policy.
- Please typeset your assignment in LaTeX. You can use this template for your written answers. Please include the netID(s) of all members in the group at the top of this page. (Search for "FILL YOUR NETID" to find the appropriate place to enter your netIDs).
- You will submit this assignment on gradescope. We have created 4 assignments: "hw1-milestone", "hw1-written", "hw1-programming" and "hw1-checktestperformance". You will submit the written parts of the assignment in the form of a pdf to "hw1-written". This will include your answers to Section A and the written questions in Section B. You will submit code or outputs of code to the remaining three assignment. Please read the submission instructions included at the end of this document for details on this.
- The University Academic Code of Conduct will be strictly enforced.

1 Section A (Written) : My Answers

1.1 N-gram Language Models

(a) Non-zero Bigram Probabilities

Given the training corpus:

$$< s > \textit{the cat sat on the mat} < /s >$$
$$< s > \textit{the dog sat} < /s >$$
$$< s > \textit{the mat is on the floor} < /s >$$

We first count the bigrams in the corpus:

Sentence 1:

$$\langle s \rangle \to \text{the}, \quad \text{the} \to \text{cat}, \quad \text{cat} \to \text{sat}, \quad \text{sat} \to \text{on}, \quad \text{on} \to \text{the}, \quad \text{the} \to \text{mat}, \quad \text{mat} \to < /s >$$

Sentence 2:

$$\langle s \rangle \to \text{the}, \quad \text{the} \to \text{dog}, \quad \text{dog} \to \text{sat}, \quad \text{sat} \to < /s >$$

Sentence 3:

$$\langle s \rangle \to \text{the}, \quad \text{the} \to \text{mat}, \quad \text{mat} \to \text{is}, \quad \text{is} \to \text{on}, \quad \text{on} \to \text{the}, \quad \text{the} \to \text{floor}, \quad \text{floor} \to < /s >$$

Bigram counts by preceding token:

For $\langle s \rangle$:

$$(\langle s \rangle, \text{the}) \quad \text{occurs 3 times} \quad \Rightarrow \quad P(\text{the}|\langle s \rangle) = 1.$$

For **the**: Total count = 5, with:

$$P(\text{cat}|\text{the}) = \frac{1}{5}, \quad P(\text{mat}|\text{the}) = \frac{2}{5}, \quad P(\text{dog}|\text{the}) = \frac{1}{5}, \quad P(\text{floor}|\text{the}) = \frac{1}{5}.$$

For **cat**:

$$(\text{cat}, \text{sat}) \ : \ P(\text{sat}|\text{cat}) = 1.$$

For **dog**:

$$(\text{dog}, \text{sat}) \ : \ P(\text{sat}|\text{dog}) = 1.$$

For **sat**: Total count = 2, with:

$$P(\text{on}|\text{sat}) = \frac{1}{2}, \quad P(</s>|\text{sat}) = \frac{1}{2}.$$

For **on**:

$$(\text{on}, \text{the}) \ : \ P(\text{the}|\text{on}) = 1.$$

For **mat**: Total count = 2, with:

$$P(\text{is}|\text{mat}) = \frac{1}{2}, \quad P(</s>|\text{mat}) = \frac{1}{2}.$$

For **is**:

$$(\text{is}, \text{on}) \ : \ P(\text{on}|\text{is}) = 1.$$

For **floor**:

$$(\text{floor}, </s>) \ : \ P(</s>|\text{floor}) = 1.$$

Non-zero bigram probabilities:

$$\begin{aligned}
P(\text{the}|\langle s \rangle) &= 1, \\
P(\text{cat}|\text{the}) &= \frac{1}{5}, \quad P(\text{mat}|\text{the}) = \frac{2}{5}, \quad P(\text{dog}|\text{the}) = \frac{1}{5}, \quad P(\text{floor}|\text{the}) = \frac{1}{5}, \\
P(\text{sat}|\text{cat}) &= 1, \quad P(\text{sat}|\text{dog}) = 1, \\
P(\text{on}|\text{sat}) &= \frac{1}{2}, \quad P(</s>|\text{sat}) = \frac{1}{2}, \\
P(\text{the}|\text{on}) &= 1, \\
P(</s>|\text{mat}) &= \frac{1}{2}, \quad P(\text{is}|\text{mat}) = \frac{1}{2}, \\
P(\text{on}|\text{is}) &= 1, \\
P(</s>|\text{floor}) &= 1.
\end{aligned}$$

(b) Examples of Grammatically Valid Sentences with Zero Probability

Even though every token is in $\mathcal{V}$, any bigram that was never observed gets a probability of zero. Two examples:

1. **Example 1:** $< s >$ *cat sat on the mat* $< /s >$
   The bigram $(< s >, \text{cat})$ was never seen.
2. **Example 2:** $< s >$ *the dog on the mat* $< /s >$
   Although $(< s >, \text{the})$ and (the, dog) occur, the bigram (dog, on) is missing.

(c) Training vs. Unseen Sentences

Statement: The sentences seen in the training corpus will always have a strictly higher probability than any unseen grammatical sentence.

Answer: False.

Justification by CounterExample: The second sentence in our corpus "¡s¿ the dog sat ¡/s¿" has the highest probability out of all the other sentences in the corpus. If we can generate an unseen sentence with the same or higher probability than this sentence, we've shown that the sentences seen in the training corpus will NOT always have a strictly higher probability than any unseen grammatical sentence.

Consider:

- **Sentence in corpus:** $< s >$ *the dog sat* $< /s >$ has probability:

$$P(\text{dog}|\text{the}) \cdot P(\text{sat}|\text{dog}) \cdot P(< /s > |\text{sat}) = \frac{1}{5} \times 1 \times \frac{1}{2} = \frac{1}{10}.$$

- **Unseen sentence:** $< s >$ *the cat sat* $< /s >$ has probability:

$$P(\text{cat}|\text{the}) \cdot P(\text{sat}|\text{cat}) \cdot P(< /s > |\text{sat}) = \frac{1}{5} \times 1 \times \frac{1}{2} = \frac{1}{10}.$$

Thus, an unseen grammatical sentence can have the same probability as a seen one.

(d) Sentence Generation from a Prefix

Given Prefix: $< s >$ *the mat is on the*

The possible next words after "the" (from part (a)) are:

$$\text{cat } (1/5), \text{ mat } (2/5), \text{ dog } (1/5), \text{ floor } (1/5).$$

Option 1: Choose $w_6 =$ **cat**.

- $P(\text{cat}|\text{the}) = \frac{1}{5}$.
- From "cat", only option is "sat" (since $P(\text{sat}|\text{cat}) = 1$).
- From "sat", choose end token $< /s >$ with $P(< /s > |\text{sat}) = \frac{1}{2}$.

Thus, one sentence is:

$$< s > \text{ the mat is on the cat sat } < /s >$$

The continuation probability is:

$$\frac{1}{5} \times 1 \times \frac{1}{2} = \frac{1}{10}.$$

Option 2: Choose $w_6 =$ **floor**.

- $P(\text{floor}|\text{the}) = \frac{1}{5}$.
- From "floor", the only option is $< /s >$ (since $P(< /s > |\text{floor}) = 1$).

Thus, the second sentence is:

$$< s > \text{ the mat is on the floor } < /s >$$

The continuation probability is:

$$\frac{1}{5} \times 1 = \frac{1}{5}.$$

(e) Total Number of Distinct Non-zero Probability Sentences

For the given prefix, note that there is a cycle in the bigram model. For example:

$$\text{the} \to \text{mat} \to \text{is} \to \text{on} \to \text{the}$$

This cycle can be repeated arbitrarily many times before eventually transitioning to $< /s >$.

Answer: There are infinitely many distinct non-zero probability sentences.

Many of these infinite sentences, although assigned non-zero probability by the bigram model, are ungrammatical because the bigram model captures relatively local dependencies. A higher-order n-gram model would mean more context constraints during training. During training, the non-zero probabilities (our parameters) are assigned to more grammatical phrases, which in turn means we can generate more grammatical sentences with a higher order model.

1.2 HMM

Problem 2.1

Joint probability for tag sequence $\{S, X, S, V\}$ and sentence $\langle$John and Mary ate$\rangle$. Let the sentence be:

$$w_1 = \text{John}, \quad w_2 = \text{and}, \quad w_3 = \text{Mary}, \quad w_4 = \text{ate}.$$

And the tag sequence is:

$$t_1 = S, \quad t_2 = X, \quad t_3 = S, \quad t_4 = V.$$

Then, using the given matrices for transitions and emissions, we have:

- **Initial transition:** From q_0 to $t_1 = S$:
$$P(S|q_0) = 0.25.$$
- **Emission at w_1:** For $t_1 = S$ and $w_1 =$ John, from Table:
$$P(\text{John}|S) = 0.4.$$
- **Transition from S to X:**
$$P(X|S) = 0.3.$$
- **Emission at w_2:** For $t_2 = X$ and $w_2 =$ and:
$$P(\text{and}|X) = 0.2.$$
- **Transition from X to S:**
$$P(S|X) = 0.2.$$
- **Emission at w_3:** For $t_3 = S$ and $w_3 =$ Mary:
$$P(\text{Mary}|S) = 0.4.$$
- **Transition from S to V:**
$$P(V|S) = 0.4.$$
- **Emission at w_4:** For $t_4 = V$ and $w_4 =$ ate:
$$P(\text{ate}|V) = 0.4.$$

Thus, the joint probability is

$$\begin{aligned} P(\mathbf{w}, t_{1:4}) &= P(S|q_0)\, P(\text{John}|S)\, P(X|S)\, P(\text{and}|X) \\ &\quad \times P(S|X)\, P(\text{Mary}|S)\, P(V|S)\, P(\text{ate}|V) \\ &= 0.25 \times 0.4 \times 0.3 \times 0.2 \times 0.2 \times 0.4 \times 0.4 \times 0.4. \end{aligned}$$

Thus, the joint probability is

$$P(\mathbf{w}, t_{1:4}) \approx 7.68 \times 10^{-5}.$$

Problem 2.2

Viterbi Chart for the sentence ⟨the chocolate was eaten quickly⟩. Let the observed sentence be:

$$w_1 = \text{the},\ w_2 = \text{chocolate},\ w_3 = \text{was},\ w_4 = \text{eaten},\ w_5 = \text{quickly},$$

and we include the stop state q_f as position 6.

The tag set (hidden states) is: $\{S, V, O, X\}$. We use the HMM parameters provided by the problem.

Viterbi Chart:

We have a 4×6 matrix (rows for S, V, O, X, and columns for $t = 1, \ldots, 5$ corresponding to the words, plus column 6 for the stop state).

State	$t = 1$ (`the`)	$t = 2$ (`chocolate`)	$t = 3$ (`was`)	$t = 4$ (`eaten`)	$t = 5$ (`quickly`)	$t = 6$ (`END`)
S						–
V						–
O						–
X						–

Problem 2.3

Step 1: Compute $V_3(j)$ for $w_3 =$ was from state $t_2 = \mathbf{S}$.

Since we are given $t_2 = \mathbf{S}$, we set $V_2(\mathbf{S}) = 1$. Then for each candidate state j at $t = 3$ we have:

$$V_3(j) = V_2(\mathbf{S}) \cdot P(j \mid \mathbf{S}) \cdot P(\text{was} \mid j).$$

Evaluating for each $j \in \{S, V, O, X\}$:

- For $j = \mathbf{S}$:
$$P(\mathbf{S} \mid \mathbf{S}) = 0.15, \quad P(\text{was} \mid \mathbf{S}) = 0 \quad \Rightarrow \quad V_3(\mathbf{S}) = 1 \times 0.15 \times 0 = 0.$$
- For $j = \mathbf{V}$:
$$P(\mathbf{V} \mid \mathbf{S}) = 0.4, \quad P(\text{was} \mid \mathbf{V}) = 0.3 \quad \Rightarrow \quad V_3(\mathbf{V}) = 1 \times 0.4 \times 0.3 = 0.12.$$
- For $j = \mathbf{O}$:
$$P(\mathbf{O} \mid \mathbf{S}) = 0.05, \quad P(\text{was} \mid \mathbf{O}) = 0 \quad \Rightarrow \quad V_3(\mathbf{O}) = 0.$$
- For $j = \mathbf{X}$:
$$P(\mathbf{X} \mid \mathbf{S}) = 0.3, \quad P(\text{was} \mid \mathbf{X}) = 0.1 \quad \Rightarrow \quad V_3(\mathbf{X}) = 1 \times 0.3 \times 0.1 = 0.03.$$

(2) Compute $v_4(j)$:

$\mathrm{v}_4(j) \; = \; \max_i \Big[v_3(i)\, P\big(j \mid i\big)\Big] \; \times \; P\big(\text{"eaten"} \mid j\big).$

Case: j = S

$$v_4(S) \; = \; \max_i \Big[v_3(i)\, P(S \mid i)\Big] \; \times \; P\big(\text{"eaten"} \mid S\big) \; = \; \max_i \Big[v_3(i)\, P(S \mid i)\Big] \; \times \; 0 \; = \; 0,$$

because $P(\text{"eaten"} \mid S) = 0$.

Case: j = X

$$v_4(X) \; = \; \max_i \Big[v_3(i)\, P(X \mid i)\Big] \; \times \; P\big(\text{"eaten"} \mid X\big) \; = \; \max_i \Big[v_3(i)\, P(S \mid i)\Big] \; \times \; 0 \; = \; 0,$$

because $P(\text{"eaten"} \mid X) = 0$.

Case: j = O

$$v_4(O) \; = \; \max_i \Big[v_3(i)\, P(O \mid i)\Big] \; \times \; P\big(\text{"eaten"} \mid O\big) \; = \; \max_i \Big[v_3(i)\, P(O \mid i)\Big] \; \times \; 0 \; = \; 0,$$

since $P(\text{"eaten"} \mid O) = 0$.

Case: j = V

$$v_4(V) \; = \; \max_i \Big[v_3(i)\, P(V \mid i)\Big] \; \times \; P\big(\text{"eaten"} \mid V\big).$$

$$= \; \max\Big\{0.12 \times 0.2, \;\; 0.03 \times 0.1\Big\} \; \times \; 0.3 \; = \; \max\{0.024, \; 0.003\} \; \times \; 0.3 \; = \; 0.024 \times 0.3 \; = \; 0.0072.$$

Hence,

$$v_4(V) = 0.0072, \quad v_4(S) = 0, \quad v_4(O) = 0 \quad v_4(X) = 0.$$

and $\mathrm{t}_3 = V$ as this is the tag that maximizes $v_4(V)$.

(3) Compute $v_5(j)$ to find t_4:

$$v_5(j) \; = \; \max_i \Big[v_4(i) \; \cdot \; P\big(j \mid i\big)\Big] \; \times \; P\big(\text{"quickly"} \mid j\big).$$

Since we know $v_4(S) = 0$, $v_4(O) = 0$, and $v_4(X) = 0$ from previous steps, the only nonzero candidate might be $v_4(V)$. Thus,

$$v_5(j) \; = \; v_4(V) \; \cdot \; P\big(j \mid V\big) \; \cdot \; P\big(\text{"quickly"} \mid j\big).$$

Case: $j = S$

$$v_5(S) \ = \ v_4(V) \ \cdot \ P(S \mid V) \ \cdot \ P(\text{"quickly"} \mid S) \ = \ \ldots \ = \ 0,$$

since $P(\text{"quickly"} \mid S) = 0$.

Case: $j = V$

$$v_5(V) \ = \ v_4(V) \ \cdot \ P(V \mid V) \ \cdot \ P(\text{"quickly"} \mid V) \ = \ \ldots \ = \ 0,$$

because $P(\text{"quickly"} \mid V) = 0$

Case: $j = O$

$$v_5(O) \ = \ v_4(V) \ \cdot \ P(O \mid V) \ \cdot \ P(\text{"quickly"} \mid O) \ = \ \ldots \ = \ 0,$$

since $P(\text{"quickly"} \mid O) = 0$.

Case: $j = X$

$$v_5(X) \ = \ v_4(V) \ \cdot \ P(X \mid V) \ \cdot \ P(\text{"quickly"} \mid X).$$

$= 0.000288$

Since $\mathrm{v}_4(V)$ is the only non-zero value which maximizes the last case, our $t_4 = V$.

(4) Backtracking:

We compare the ending probabilities and observe that $V_5(X)$ has the greatest probability. Therefore, we conclude

$$t_5 \ = \ X,$$

and by backtracking through the stored pointers, we find

$$t_4 \ = \ V \quad \text{and} \quad t_3 \ = \ V.$$

2 Section B (Programming) : Written Answers

Response 1.1

When printing these 5 documents that contain at least 4 distinct tags (including 'O'), I noticed that while most tokens are labeled as non-entities ("O"), the named entities are still clearly bracketed with their corresponding tags, such as [MISC Punic Wars], [LOC Rome], and [PER Vandals]. The examples show that the model is learning to distinguish distinct non-entity tokens within the same sentence and the need for strong context modeling to accurately predict less frequent entity types.

```
# YOUR CODE HERE
#Want: 5 sentences with 4 distinct tags
#First, get the indicies of these such sentences!
indices = []
n = len(training_data['NER'])
for i in range(n):
    ner = training_data['NER'][i]
    distinct_tags = set()
    #Get distinct tags for each ner
    for tag in ner:
        if tag == "O":
            distinct_tags.add(tag)
        else:
            distinct_tags.add(tag[2:])
    #print("distinct tags", distinct_tags)
    if len(distinct_tags) >= 4:
        #print("4 uniqute")
        indices.append(i)
        if len(indices) == 5:
            break

#Print with stringify_labeled_doc
```

```
#print(indices)

for ind in indices:
    text = training_data['text'][ind]
    ner = training_data['NER'][ind]
    s = stringify_labeled_doc(text, ner)
    print(s)
```

```
Cutting the skin into strips, she laid out her claim and founded an empire that would become, through t
The [LOC Chatham] oystercatcher is endemic to the [LOC Chatham Islands] of [LOC New Zealand] but is lis
He then moved to the [ORG Deutsche Eishockey Liga] in [LOC Germany] for [ORG ERC Ingolstadt] and then t
In his keynote address at the 2015 [MISC South by Southwest] music festival, he blamed [LOC Los Angeles
Further mention of events in [LOC Kent] occurs in the late sixth century history of the [MISC Franks] b
```

Response 1.2

Print Statement from code:

```
invalid_count = sum(not validate_ner_sequence(doc_tags) for doc_tags in training_data)
print("Number of documents with invalid labelings:", invalid_count)
```

Number of documents with invalid labelings: 3

Response to 1.3

I noticed that a large majority of NER tags are O, which is what I would expect. It may be difficult for our model to predict the rarer entity tags because the occurrence of O is high.

```
# YOUR CODE HERE
# Want token level distribution of NER tags (O included)
tags = [tag for ner in training_data["NER"] for tag in ner]

plot_histogram(tags, "Distribution of NER Tags", "NER Tag", "Count",)
```

Response to 1.4

18308 words are both uppercase and part of an entity, and 1104 words are both lowercase and part of an entity. There are less lowercase + entity words than uppercase + entity words, which I expected. Also there are 8054 words in the corpus that are uppercase but not part of an entity (likely words such as ”I”, ”The”, etc). The majority of words being lowercase + not entity is expected.

```
# YOUR CODE HERE

#Is Entity? by Is Uppercase? Matrix

#Count each of these entries
ent_up = 0
not_ent_up = 0
ent_not_up = 0
not_ent_not_up = 0

all_tokens = [token for tokens in training_data["text"] for token in tokens]
all_tags   = [tag for tags in training_data["NER"] for tag in tags]

n = len(all_tokens)
for i in range(n):
    word = all_tokens[i]
    tag = all_tags[i]
    #print(word, tag)
    if tag != "O" and len(word)>0 and word[0].isupper():
        ent_up +=1
    elif (tag == "O") and len(word)> 0 and (word[0].isupper()):
        not_ent_up +=1
    elif (tag != "O") and len(word)>0 and (not word[0].isupper()):
        ent_not_up +=1
    elif (tag == "O") and len(word)> 0 and (not word[0].isupper()):
        not_ent_not_up +=1

print( [[ent_up, ent_not_up], [not_ent_up, not_ent_not_up]])

matrix = [
    [ent_up,        ent_not_up],
    [not_ent_up,    not_ent_not_up]
]

# Convert to numpy array (not strictly necessary, but often convenient).
counts = np.array(matrix)

plt.figure(figsize=(5, 4))
plt.imshow(counts, cmap='Blues', aspect='auto')

# We'll label rows/columns based on how we arranged the matrix:
#   - Rows: [Entity, Not Entity]
#   - Cols: [Uppercase, Not Uppercase]
row_labels = ["Entity", "Not Entity"]
col_labels = ["Uppercase", "Not Uppercase"]

# Add tick marks
plt.xticks([0, 1], col_labels)
plt.yticks([0, 1], row_labels)

# Annotate each cell with its count
for i in range(counts.shape[0]):
    for j in range(counts.shape[1]):
        plt.text(j, i, str(counts[i, j]),
                 ha="center", va="center", color="black")

plt.title("Entity vs. Case Counts")
plt.xlabel("Case")
plt.ylabel("Entity Status")
```

Response to 1.5

I noticed that a lot of words that may ALSO be considered stop tokens such as "the", "of", "and" are highly frequent as entity tokens.

`[('of', 286), ('The', 126), ('the', 121), ('New', 86), ('World', 85), ('United', 85), ('War', 74), ('States', 70), ('John', 63), ('and', 60)] The top ten tokens most frequently tagged as named entity are ['of', 'The', 'the', 'New', 'World', 'United', 'War', 'States', 'John', 'and']`

These high-frequency stopwords can become disproportionately associated with certain named-entity labels. The model may learn overly high emission probabilities for these stopwords, which could overshadow more semantically meaningful tokens. This could potentially lead to incorrect tagging of stopwords as entities in new data. This shows the importance of either filtering out certain tokens or carefully addressing these stopword cases to improve emission probability estimates.

```
# YOUR CODE HERE
# Top 10 (in count) for words with an entity tag
```

```
entity_word_counter = {}
all_tokens = [token for tokens in training_data["text"] for token in tokens]
all_tags   = [tag for tags in training_data["NER"] for tag in tags]

n = len(all_tokens)
for i in range(n):
    word = all_tokens[i]
    tag = all_tags[i]
    #print(word, tag)
    if tag != "O":
        entity_word_counter[word] = entity_word_counter.get(word, 0) + 1

#Sort the hashmap
sorted_counter = dict(sorted(entity_word_counter.items(), key=lambda item: item[1], reverse= True))

print(list(sorted_counter.items())[:10])
top_ten = list(sorted_counter.keys())[:10]

print("The top ten tokens most frequently tagged as named entity are ",top_ten)
```

Response 1.6

According to the scatter plot, the relationship between Document Length and Number of Named Entities appears to have a very, very weak positive association: This means that while ”generally” longer documents associate with more named entities, it’s not a strict conclusion. Also, a large cluster of our data is part of shorter document lengths + lower number of named entities.

```
# YOUR CODE HERE
# Scatter Plot: Length of Doc (Total Number of Tokens) Vs. Number of Named Entities in Doc
doc_lengths = [len(doc) for doc in training_data["text"]]
num_entities = []

n = len(training_data["text"])
for i in range(n):
    ner = training_data["NER"][i]
    m = len(ner)
    ent_count = 0
    for j in range(m):
        tag = ner[j]
        if tag != "O":
            ent_count +=1
    num_entities.append(ent_count)

plt.figure(figsize=(8,6))
plt.scatter(doc_lengths, num_entities, alpha=0.5)
plt.xlabel("Document Length (# of tokens)")
plt.ylabel("Number of Named Entities")
plt.title("Document Length vs. Number of Named Entities")
plt.show()
```

Response to 1.7

In our code below, we found that only about 15.02 percent of entity spans appear at least twice in the training data. This is a novel insight because it can challenge the assumption that entity repetition is common enough to be highly leveraged by the model. The percentage reveals that the majority of entities are actually relatively unique within a document, so relying on counts for tagging may be not super effective. One way to address this is to add smoothing or perhaps increasing our n-gram assumption behind this model to account for unseen instances and account for more context.

```
from collections import Counter

def get_entity_spans(tokens, ner_tags):
    """
    Get entity spans from a sentence.
    Returns a list of entity spans as tuples (span_text, tag).
    """
    spans = []
    current_span = []
    current_tag = None
    for i in range(len(tokens)):
        token = tokens[i]
        tag = ner_tags[i]
        if tag.startswith("B-"):
            if current_span:
                spans.append((" ".join(current_span), current_tag))
            current_span = [token]
            current_tag = tag[2:]
        elif tag.startswith("I-") and current_tag == tag[2:]:
            current_span.append(token)
        else:
            if current_span:
                spans.append((" ".join(current_span), current_tag))
                current_span = []
                current_tag = None
    if current_span:
        spans.append((" ".join(current_span), current_tag))
    return spans

# get all spans
all_spans = []
for i in range(len(training_data["text"])):
    tokens = training_data["text"][i]
    ner_tags = training_data["NER"][i]
    spans = get_entity_spans(tokens, ner_tags)
    all_spans.extend(spans)

# count how many times each (entity span, tag) pair occurs
span_counter = Counter(all_spans)

# plot histogram: x-axis = number of occurrences, y-axis = count of unique spans with that occurrence
counts = list(span_counter.values())
max_count = max(counts) if counts else 1
plt.hist(counts, bins=range(1, max_count+2), align="left", edgecolor='black')
plt.xlabel("Number of occurrences per entity span")
plt.ylabel("Frequency (number of unique spans)")
plt.title("Histogram of Entity Span Occurrences")
plt.show()

# Calculate percentage of entity spans occurring at least twice
num_repeated = sum(1 for count in counts if count >= 2)
total_spans = len(counts)
percentage_repeated = (num_repeated / total_spans * 100) if total_spans > 0 else 0
print("Percentage of entity spans occurring at least twice: {:.2f}%".format(percentage_repeated))
```

Response to 2.1

I made a helper function to highlight the differences between our actual and expected labels. Specifically, I gathered false negatives and false positives for our predictions. They are printed in the above cell.

Our helper function indicates that the model generally performs better in predicting organizational (ORG) entities, while it tends to struggle the most with person (PER) and miscellaneous (MISC) tags. Also for PER, LOC, MISC, the number of false negatives exceeds the number of false positives. This suggests that the model is more conservative when predicting these tags. It is often missing entities that should be labeled as PER, LOC or MISC rather than incorrectly labeling non-entities as such.

There are several possible reasons for these observations. First, the training data may have a higher representation for ORG entities, leading to better learned parameters for those cases. But PER and MISC entities may vary more on content, making them more difficult to identify accurately. To improve performance on the more problematic PER and MISC categories, we could improve the training data with more examples of these entities or modify our smoothing parameters to better capture these tags.

Response to 2.2

The treatment of unknown words, which our model considers as "unk", and the application of smoothing ensures that our model does not assign zero emission probability to rare or unseen words. This prevents against the case where an unseen token would otherwise lead to a zero probability for the entire sequence. For example, if the word "xylophone" is encountered at test time but was never seen during training, mapping it to '¡unk¿' allows the model to use a learned emission probability for unknown words rather than failing completely.

Add-k smoothing addresses data sparsity issues by ensuring that every possible tag,word pair has a nonzero probability. In our smoothing function, a constant k is added to every count, and the result is normalized by the total count for that state times k. Again, this approach prevents the model from assigning a probability of zero to events that were not seen during training.

Example for Explanation Above

Consider this training dataset:

- "Alice": 3 occurrences
- "Bob": 2 occurrences
- '¡unk¿' with an initial count of 0.

Without any smoothing, the emission probabilities would be:

$$P("Alice"|\text{PER}) = \frac{3}{3+2+0} = \frac{3}{5}, P("Bob"|\text{PER}) = \frac{2}{5}, P(" < unk > "|\text{PER}) = \frac{0}{5} = 0.$$

Our test set has the word "xylophone" in a sentence that should be tagged as PER. Since "xylophone" is not in the training vocabulary, the system maps it to '¡unk¿'. Without smoothing, this results in an emission probability of 0 for '¡unk¿' under PER, which would mean the entire sentence probability will evaluate to zero.

If we instead apply add-k smoothing with $k = 0.5$, the smoothed counts become:

$$\text{Count\_smoothed("Alice")} = 3{+}0.5 = 3.5, \text{Count\_smoothed("Bob")} = 2{+}0.5 = 2.5, \text{Count\_smoothed("¡unk¿")} = 0{+}0.5 = 0.5$$

The denominator is the total smoothed count:

$$\text{Total} = 3.5 + 2.5 + 0.5 = 6.5.$$

And the smoothed probabilities are:

$$P("Alice"|\text{PER}) \approx \frac{3.5}{6.5} \approx 0.538, P("Bob"|\text{PER}) \approx \frac{2.5}{6.5} \approx 0.385, P(" < unk > "|\text{PER}) \approx \frac{0.5}{6.5} \approx 0.077.$$

During testing, xylophone is mapped to '¡unk¿', and the model uses $P(" < unk > "|\text{PER}) \approx 0.077$ instead of 0. This nonzero probability prevents the entire sequence probability evaluating to 0, so we can see our model can handle unseen words and avoids the zero-probability problem.