Names: Chad Yu, Joshua Huang
NetIDs: cky25, jth239

CS 4740 HW 1 Questions

Q1.1: Print 5 of the documents (sentences) from the training data using the `stringify_labeled_doc` function in `data_exploration.py`. What do you notice?

We notice that a lot of the NER tags have some consecutive sequences of lone-standing "I" prefixed tags, that is, there is a tag prefixed with "I" without having an associated beginning (B) tag. One example of this happening in the training data is shown below:

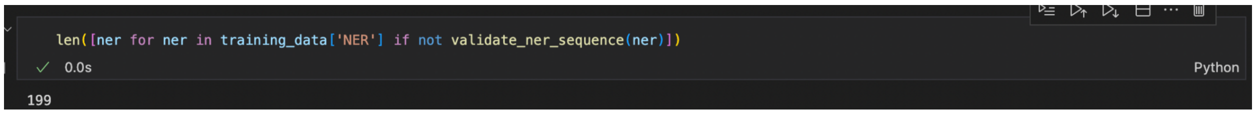
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'I-MISC', 'I-MISC', 'O', 'O', 'O', 'O', 'O', 'O', 'B-PER', 'I-PER', 'O', 'O', 'O', 'O', 'O', 'O']
" In 2008 the company began work on " [MISC The Women] " , an adaptation of the [PER George Cukor] 's film of the same name

Clearly, there are two consecutive `I-MISC` tags, but no associated `B-MISC` tag with the named entity. Another erroneous example that we caught is:

['O', 'B-PER', 'I-PER', 'O', 'O', 'O', 'O', 'O', 'B-ORG', 'I-ORG', 'O', 'O', 'O', 'I-MISC', 'I-MISC', 'I-MISC', 'O', 'O', 'O', 'O', 'O', 'O', 'B-PER', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'I-MISC', 'O', 'O', 'O', 'O', 'O', 'B-PER', 'I-PER']
Writer [PER David Nobbs] went on to create the [ORG Channel 4] comedy series " [MISC Fairly Secret Army] " , whose lead character , [PER Harry] , was inspired by , if not directly related to , the Perrin character of [MISC Jimmy] , and also played by [PER Geoffrey Palmer]

which has a similar issue as the first example.

Q1.2: Implement the `validate_ner_sequence` function in `data_exploration.py` Are there any documents in the training data which have invalid labelings? If so, how many are there?



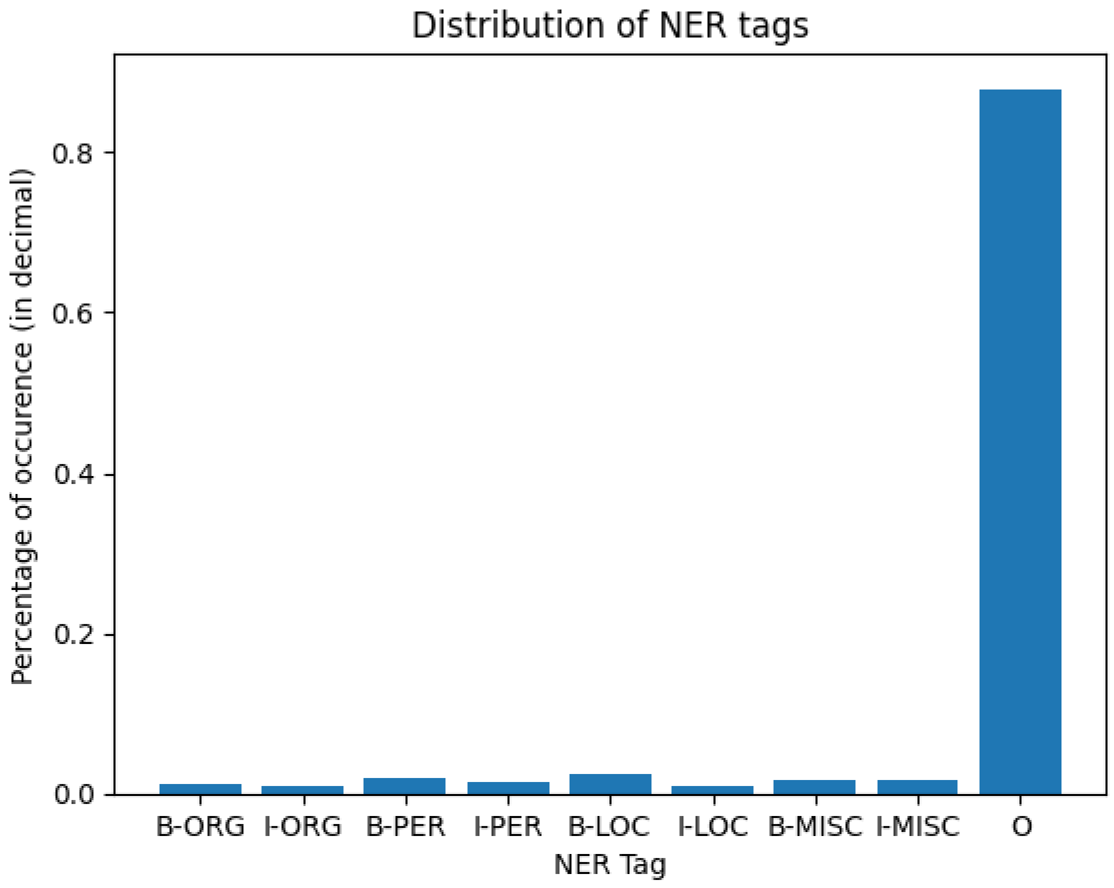
```
len([ner for ner in training_data['NER'] if not validate_ner_sequence(ner)])
```

```
199
```

There are indeed documents in the training data with invalid labelings, specifically, there are **199** documents that have invalid labelings, as shown by the code block above.

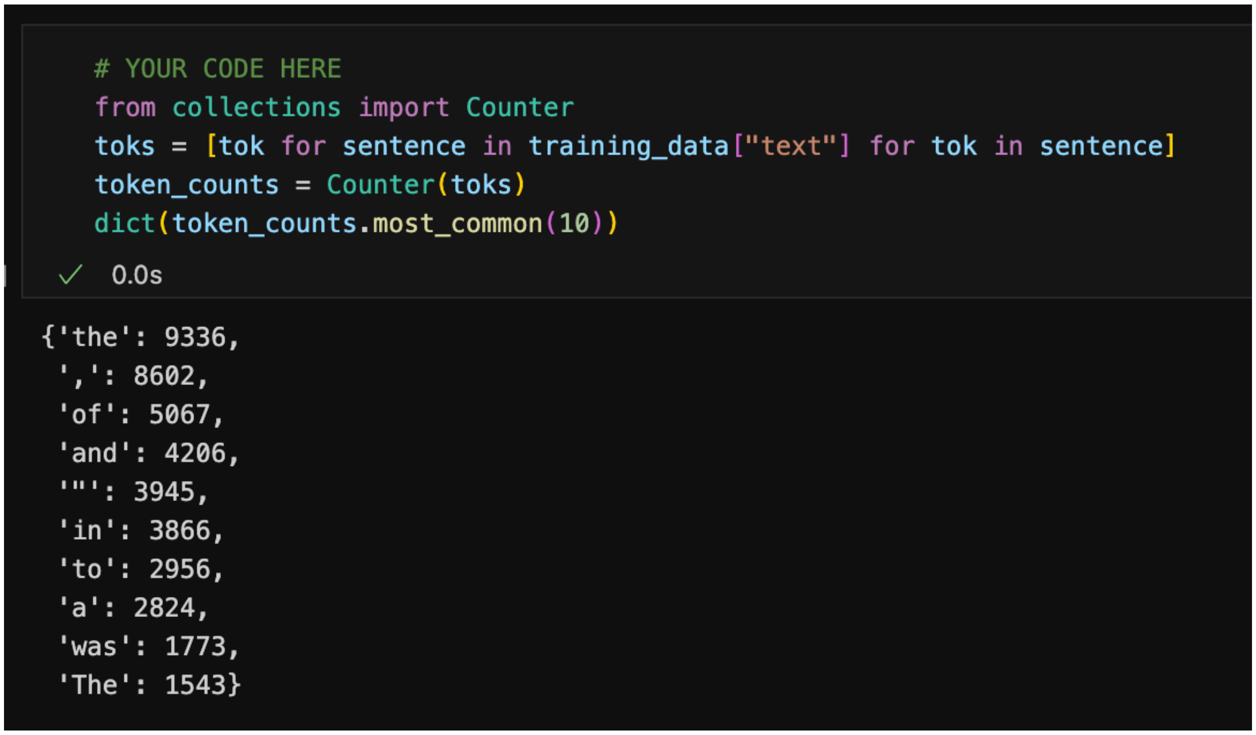
Q1.3: Provide a bar graph giving the token level distribution of NER tags, (O included): e.g. 10% of tokens are B-ORG, 20% of tokens are I-ORG, etc. What do you notice about

this distribution? Is this what you might expect? What difficulties might this cause for your models?



We notice that the distribution is skewed heavily towards text that is tagged with "O" (more than 80% of words are tagged with "O" and other tags have near 0 percent occurrence), which is something that we might expect, as named entities only constitute a small part of natural language. This might cause difficulty with our model in that assuming these percentages are taken into consideration for transition probabilities, any sequence of tokens with not "O" tags could have close to 0 probability of happening, especially if rounding errors are taken into consideration.

Q1.4: Provide a list of the (tokens) of the 10 most common named entities. What do you notice about the list? Is this what you might expect? What difficulties might this cause for your models?



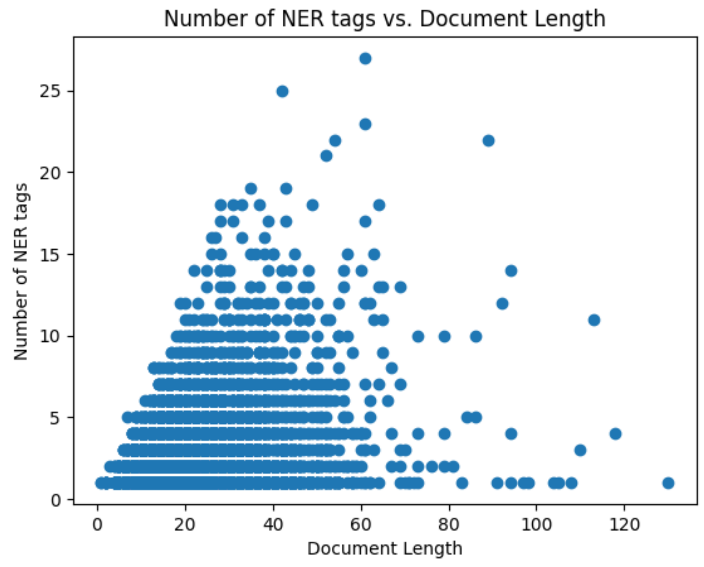
```
# YOUR CODE HERE
from collections import Counter
toks = [tok for sentence in training_data["text"] for tok in sentence]
token_counts = Counter(toks)
dict(token_counts.most_common(10))
```

✓ 0.0s

```
{'the': 9336,
 ',': 8602,
 'of': 5067,
 'and': 4206,
 '"': 3945,
 'in': 3866,
 'to': 2956,
 'a': 2824,
 'was': 1773,
 'The': 1543}
```

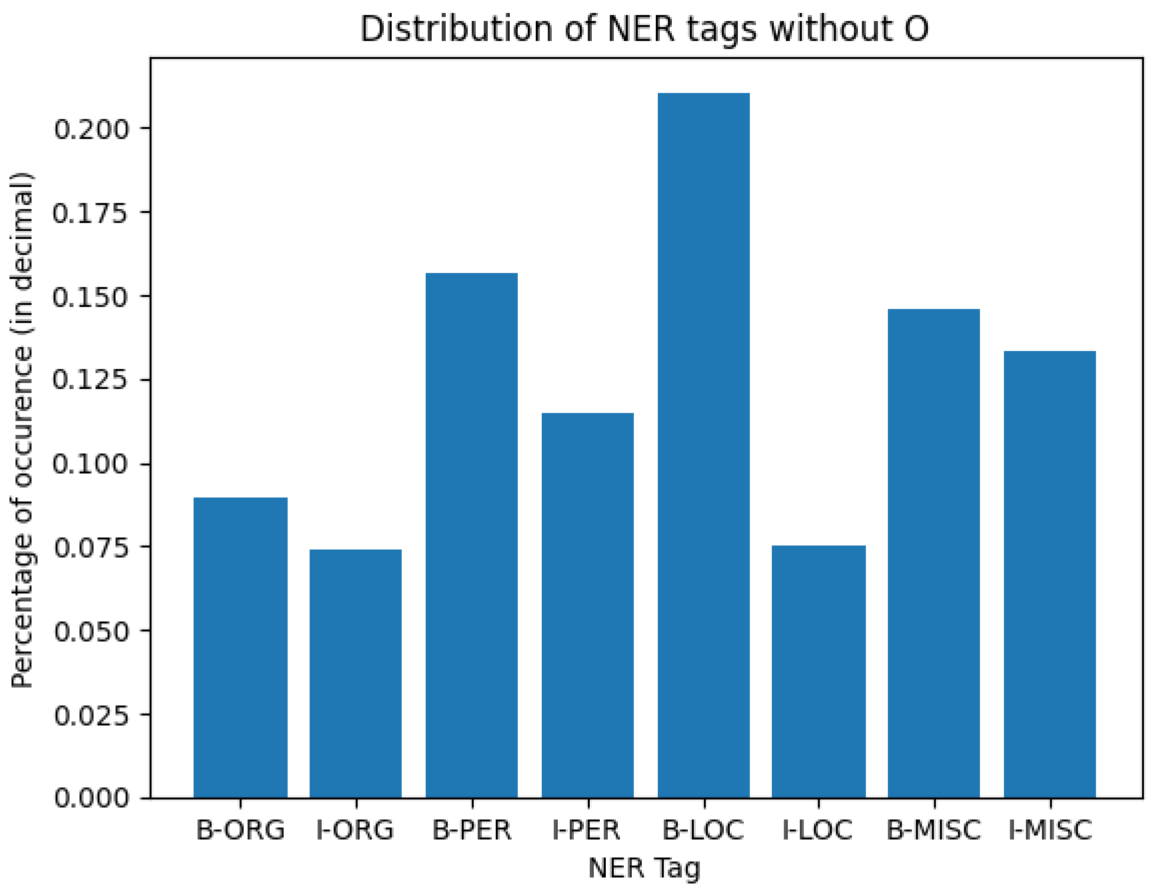
We notice that most of the list is punctuation or very commonly used words in natural language like articles and conjunctions (e.g., the, and, of, to, etc). Similarly to the issue with the high percentage of "O" tags as discovered in the last question, specifically for HMMs, this might cause emission probabilities of other more complex words to be very low, so that named entities chained by multiple words could have near 0 probability.

Q1.5: Provide a scatter plot mapping the length of a document to the number of named entities in that document. Describe what the plot looks like, and what it might mean about the relationship between named entities and document length.

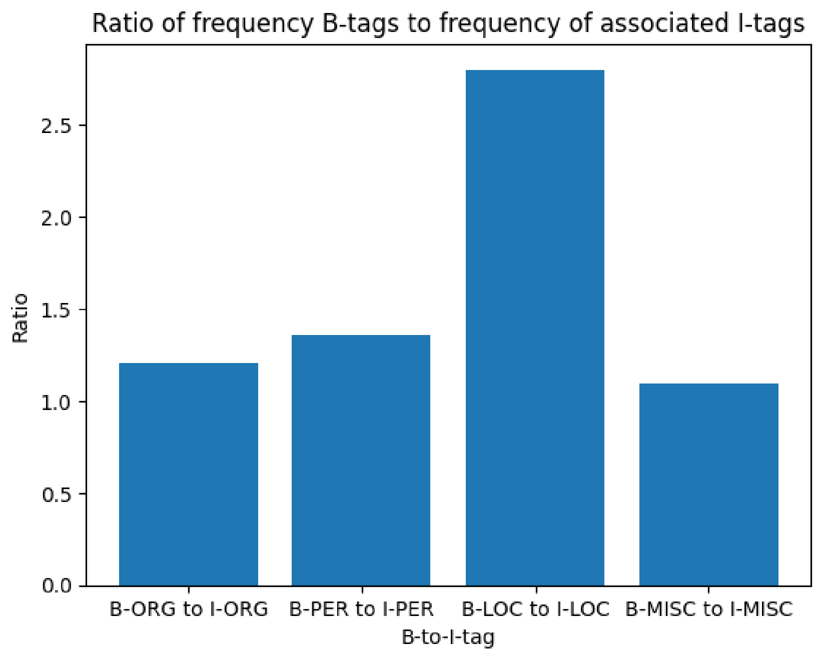


As shown above, the plot above has no real structure to it, but clearly, the number of named entity tags is highly concentrated at 15 or below for a wide range of document lengths, and other than that, the number of NER tags are scattered in a chaotic manner. Furthermore, the data points are also highly concentrated for document lengths below 60, and very sparse for lengths greater. This demonstrates that there is little to no relationship between the number of named entities and documents.

Q1.6: Make a convincing argument around a novel insight about the data. In only a few sentences, argue why this insight is important in understanding the data for this task, and support your answer with relevant graphs or statistics.



{'B-ORG to I-ORG': 1.2054507337526206, 'B-PER to I-PER': 1.3603969327920615, 'B-LOC to I-LOC': 2.7980702963473467, 'B-MISC to I-MISC': 1.0970381917381138}



One interesting thing about the data is that the ratio of frequency of B-LOC to I-LOC is much higher than the ratios of the other B-I tags. This is supported by the above graph, which shows that B-LOC to I-LOC has double any other tag types' ratio. We think this is important as it tells us that our data has a lot of locations with a singular standalone word, whereas the other named entities are frequently followed by another named entity. For example, the B-MISC to I-MISC ratio of 1 leads us to believe that the majority of B-MISC tagged words will have an I-MISC tagged word after it.

Q2.1: In which situations did the system perform effectively, and when did it encounter challenges? For instance, does the model excel in predicting certain NER tags more than others? Could you offer any hypotheses about the reasons behind these patterns and suggest potential improvements?

Based on looking at the predicted tags vs. actual tags, our data seemed to predict the 2 word names of people and locations pretty accurately. It makes sense that the HMM was able to easily learn how to predict locations and names considering the context of these named entities; for example, a keyword for a lot of locations is 'in', so the HMM is able to recognized that there is a high probability that the following word will be tagged B-LOC (ex: in Jakarta). Overall, named entities following prepositions seemed to be a pattern that the HMM performed very effectively on, which makes sense because HMMs are looking at the probabilities of a tag given the words in front of them. However, the model was especially bad at predicting named entities that double down as normal nouns (ex: General Mills, Punch, Damages). Consider the phrase, "Then in 1928 merged with about 25 other mills to form General Mills." The HMM misclassified General Mills, which makes sense since in context General Mills could very easily be a regular noun. Also, it seemed to handle proper nouns doubling down as regular nouns poorly: consider, "the FX legal thriller 'Damages'." This is another example of a common misclassification from the model, which might be due the handling of punctuation. One potential improvement we could make to fix this is to consider quotation marks separately from other punctuation such as commas, which we believe would boost the performance of the HMM.

Q2.2: How does the treatment of unknown words and the application of smoothing impact the system's performance? Provide examples to illustrate your insights.

We run a few experiments to try to understand the effect of the `handle_unknown_words` and smoothing in our HMM model. We use our initial HMM model as a baseline, which uses `t = 0.01, k_t = 0.01, k_e = 0.01, k_s = 0.1`, and change these parameters to 0 accordingly for each experiment. For reference, this had a F1 mean of 0.51179. The first experiment explores the effect of no smoothing, for which `k_t, k_e, k_s` are all set to 0. Applying this model on the validation set, we get an F1 mean of 0.4189. For the second experiment, we explore the effect of no unknown word handling, in which we set `t` to 0. Applying this onto the validation set, we get an F1 mean of 0.437. For the third experiment, we explore the effect of both no smoothing and no unknown word handling, for which we set all parameters to 0. Applying this onto the validation set, we get an F1 mean of 0.197. For the fourth experiment, we explore the effect of no transition probability smoothing, for which we set `k_t` to 0. For this, we get the same F1 mean as the baseline when applied to the validation set. Finally, in the last experiment, we explore the effect of no emission probability smoothing, for which we set `k_e` to 0. For this, we get an F1 mean of 0.4992 when applied to the validation set. Clearly, we see that eliminating all of smoothing and unknown word handling affects the performance of the model the worst. Furthermore, we can see that individually eliminating smoothing of transition or emission probabilities doesn't affect the performance of the model, as much as eliminating a combination of handling unknown words, and each smoothing parameter. We can also see that individually eliminating anything else doesn't affect the F1 mean as the combination of all of them as well. One possible explanation for these findings is that the inherent distribution on the training data can be trusted. This could make sense, as our implementation of handling unknown words could mark named entities as unknown words because of their low frequency, but with smoothing, especially with small hyperparameters as we do, the probability is still essentially 0, so that the transition and emission probabilities are not affected much.

Q3.1: Explain here how you evaluated the MEMM model. Summarize the performance of your system and any variations that you experimented with the validation datasets. Put the results into clearly labeled tables or diagrams and include your observations and analysis.

POS-tag, is_cap, token_length, token_frequency, prev_tag_not_o	0.111
POS-tag, is_cap, token_length, token_frequency, prev_tag	0.132
POS-tag, is_cap, token_frequency, prev_tag	0.119
POS-tag, prev_tag, current-token	0.299
POS-tag, prev_tag, current-token, is_cap	0.293
POS-tag, prev_tag, current-token, is_cap, is_first	0.301
POS-tag, prev_tag, current-token, is_cap, is_first, next_token	0.389
POS-tag, prev_tag, current-token, next_token	0.353
POS-tag, prev_tag, current-token, is_cap, is_first, next_token, prev_token	0.480

We evaluated the MEMM model based on the f1 score, seeking to improve it. Initially, we chose a poor set of features which led to a poor performance from the model. In fact, we noticed that only one of the features that we used in our first iteration was causing significant improvements, as you can see by our experiments with the validation dataset. Ultimately, we found that the best three features were the POS-tag, previous tag, and the current token. We decided to use these features because we realized these were the features that the HMM was using. Using these three alone boosted our performance from ~0.1 to ~0.3. Starting to realize that the words surrounding other words helped us identify the tag of a word a lot more than characteristics of the word itself did, we eventually added next token and previous token, which gave us our best result (0.48). We also used features that looked at if the word started with a capital letter and if the word was the first word in a sentence, which seemed to give the MEMM a bit of a boost in accuracy.

Q3.2: What features are considered most important by your MaxEnt Classifier? Why do you think these features make sense? Describe your experiments with feature sets. An analysis on feature selection for the MEMM is required – e.g. what features **help most, why? An **error analysis** is required – e.g. what sorts of errors occurred, why?**

Using the method `memm.classifier.show_most_informative_features()`, we found that our most informative feature was the next token tag. The other two features that show up are current token (B-ORG) and if the first letter of the word is capital (B-LOC). This makes sense because the most telling way to tell if something is a named entity is to understand the context of the word, which is what some of our MEMM features do. For example, without the next token as a feature, the MEMM will frequently miss standalone named entities.

Q3.3: When did the system work well, when did it fail and any ideas as to why? How might you improve the system?

The MEMM performed well on classifying named entities in general; that is, it was able to recognize when something was a named entity even if the specific tag (location vs. person) was off. I think this is because we are using a lot of contextual features, and named entities are captured by these nearby dependencies well (one often follows another). Also, the first letter of named entities are normally capitalized, and we used this as one of our features. In fact, I believe this led the model to be a little trigger happy in terms of assigning words as a named entity sometimes. One pattern I noticed that our MEMM really struggled with was abbreviations (ex: API, GF, E, PMA). Oftentimes, it would classify any word with all caps as a named entity even if it wasn't. This is somewhat reasonable, as these abbreviations are typically short for named entities (ex: ICPC) so in context since abbreviations are almost always nouns and are capitalized, I believe the MEMM has trouble differentiating between the abbreviation named entities and regular abbreviations. They are also used with adjectives (ex: bridging API), which is consistent with descriptions for named entity. Another pattern that the MEMM had a hard time with was classifying 'The' when it was part of a multi-word named entity (ex: The Naked Vicar Show, The Hitchhiker's Guide to the Galaxy, The Hucksters). I think this error is caused by a similar reason to the abbreviation error: in context, 'The' often starts sentences and is capitalized, so the model has a hard time differentiating when it is a named entity. This problem is a bit harder to fix, but maybe tracking multiple words after 'The' would be benifical as both the examples given are a longer named entity. Finally, we noticed that standlone locations (B-LOC) were often mistagged or not tagged at all, especially those that were very non-English, so maybe having a list of locations as a feature as a lookup table would be helpful or training more on datasets with foreign sounding locations.

Q4.1: Result Comparison: Compare here your results (validation scores) for your HMM and the MEMM. Which of them performs better? Why?

The validation score for our HMM was 0.512 and our MEMM was 0.480. This means that the HMM performed better than the MEMM. One possible reason is that the MEMM overfitted the training data. Also, the MEMM required us to choose the features to consider when tagging words, and we may not have chosen the most optimal features to boost MEMM performance.

Finally, one more possibility for the HMM performing better than the MEMM is that our data is relatively sparse (majority O-tags). Since we are smoothing in the HMM (and not in the MEMM) this might be helping the boost in performance. The inherent ability of the HMM to model the joint probability distribution between the words and states is also useful when dealing with our sparse data.

Q4.2: Error Analysis 1: Do some error analysis. What are error patterns you observed that the HMM makes but the MEMM does not? Try to justify why/why not? **Please give examples from the dataset.

One error that the HMM frequently made that the MEMM didn't make was grouping together long strings of named entities. For example, in the sentence "He also has Italian and Polish citizenships" the HMM grouped 'and' as a named entity along with Italian and Polish whereas the MEMM correctly identified that 'and' should have an O-tag. We think this is because the MEMM uses features other than the previous tag, current token, and predicted tag to predict a named entity; features like if the first letter is a capital letter definitely help the MEMM in this case. Another pattern that the MEMM seems to recognize better than the HMM is standalone named entities, especially the ones contained inside of quotation marks. Some examples of these are "directed the film version of his verse play 'Decadence'" and "his 2009 book 'Meltdown'." We think this is due to the MEMM taking what comes after the current token into account as a feature while the HMM only looks at what comes previously because of the Markov chain assumption. Also, the MEMM can consider the actual tokens while the HMM only looks at the tags, so the MEMM is given more information. Thus, it makes sense that the MEMM is able to pick up the quotation mark pattern while the HMM cannot.

Q4.3: Error Analysis 2: What are the error patterns you observed that MEMM makes but the HMM does not? Try to justify what you observe? Please give examples from the dataset.

The main error pattern we found in the MEMM that we didn't in the HMM: combining different types of named entity tags. The MEMM will sometimes mix and match NE-tags. Take the example sentence "He made his professional debut on 8 December 2001 against SC Bastia" in which SC Bastia is labeled [B-ORG, I-ORG]. Then, we see that HMM tagged it correctly, but the MEMM tagged it [B-ORG, B-LOC]. This is just an example of a common pattern throughout the dataset. In general, the HMM will tag the I-tag after the B-tag the same category, but the MEMM sometimes does multiple B-tags consecutively. I believe this is because the HMM tags purely based on the word in token of it, while the MEMM also takes into account the following token, so the meaning of the token is a bit more ambiguous. The part-of-speech tag might also mess this up, so the training data might be being overfit a bit.