**CS 4740 Project 1 Write Up Document**

**Section 1 Questions**

**→ Q1: Initial Data Observations**

What are your initial observations after you explore the dataset? Provide some quantitative data exploration. Assess dataset size, document lengths and the token-level NER class distribution, and the entity-level NER class distribution (skipping the 'O' label for the latter). Give some examples of sentences with their named entities bracketed, e.g. [[LOC Romania] state budget soars in June .] and [[ORG Zifa] said [PER Renate Goetschl] of [LOC Austria]...].

**Your answer:** 756 total document in train array. Mean of 215 words in the documents. 162341 words total in dataset. Distribution of NER tags: {'LOC': 6556, 'O': 135186, 'ORG': 8065, 'MISC': 3659, 'PER': 8875}

**([MISC Polish] schoolgirl blackmailer wanted textbook.)**

**([LOC GDANSK] , [LOC Poland] 1996-08-22, A [MISC Polish] schoolgirl blackmailed two women)**

**Section 2 Questions**

**→ Q2.1: Explain your HMM Implementations**

Explain how you implemented the HMM including the Viterbi algorithm (e.g. which algorithms/data structures you used). Make clear which parts were implemented from scratch vs. obtained via an existing package. Explain and motivate any design choices providing the intuition behind them (e.g. which methods you used for your HMM implementation, and why?). (Please answer on the written questions template document)

**Your answer:** We implemented the unknown word handling by looping through the list of input tokens and the first occurence of a word is replaced by an "<UNK>" token. This altered token list is then fed into the rest of the HMM algorithm. For implementing the transmission and emission functions, we had them output a nested dictionary. For the transmission, we had an outer dictionary that had each NER token and the value of each NER token was a dictionary for the count of the NER Token that came after it in the sentence. For emission, we had the same outer dictionary of the NER tokens being keys and the values a nested dictionary that had the corresponding word of that NER Token type and its value was the probability it would show up given the Token type it is. Using the token list outputted from our unknown word handling function, we then can generate emissions probabilities for unknown words.. Our get start state function outputted a dictionary that had the NER tokens as the key and the probability that they would start the sentence. We then created another dictionary in our build\_hmm function that we then stored the outputs of the previous 3 functions as the values and the name of the function as the key. We designed the HMM like this because 1) Everything is compacted into one data structure and 2) We could use english words to index into the dictionary instead of making the entire thing a 3d array. Using dictionaries helps us debug the code and find errors in our functions easier and faster. For the Viterbi Algorithm, we stored our BPTR (a list of dictionaries) that contained all the mappings of the tokens that lead to the maximum probability for the next word. Our final\_labels list contains the computed NER tags of the input text

(ADD PART ABOUT ADD K SMOOTHING)

**→Q2.2: Results Analysis**

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

**Your answer:** We used the sklearn.model\_selection.train\_test\_split function in order to split up our data between a training set and a validation set. By using that function, we are also able to randomly select the training and validation data so we could test for multiple F1 mean scores.We then changed around the percentages of training data to validation data to see how our mean F1 score would change. For high percentages of training set data (90% and above) and low percentages of validation set data (10% and lower), we saw that the F1 mean score had a higher range of possible scores on successive runs.

| **Validation Portion** | **F1 Score** |
| --- | --- |
| **0.99** | **With such a large validation set and very little training data, the model doesn’t learn much and has an average F1 of 0.035** |
| **0.9** | **Having more training data causes a substantial increase in the F1 score: 0.330** |
| **0.5** | **0.599** |
| **0.2** | **0.627 At this stage the F1 values start to level out a little** |
| **0.1** | **0.632** |
| **0.01** | **0.572 Since the validation set is now very small, even though the model is trained on a lot more data, there is a chance that the validation examples will simple be bad examples for our training data and therefore the model will work poorly on them. The results with such a small validation set were a lot more erratic.** |

**→ Q2.3: Error Analysis**

When did the system work well? When did it fail? Any ideas as to why? How might you improve the system?

**Your answer:** The system was usually ineffective at identifying words that it never encountered before, but it was able to correctly assign the NER token when there were adjacent words that the model did recognize with the expected NER label. This was particularly true for word units (two or more words which represent the same idea) such as “Cornell University”, where Cornell was an unknown token, but since it was familiar with University as an ORG, it was able to recognize “Cornell University” as an ORG. Potentially, using trigrams for the transition matrix could make the model more likely to guess the correct NER token

**→ Q2.4: What is the effect of unknown word handling and smoothing?**

**Your answer:**

We implemented unknown word handling and add-k smoothing for our emission and transition probability matrices where appropriate. Both of these additions caused a small increase in our f1 score, compared to our original implementation which skipped emission probabilities for unknown words and which didn't have smoothing.

**Section 3 Questions**

**→ Q3.1: Implementation Details**

Explain how you implemented the MEMM and whether/how you modified Viterbi (e.g. which algorithms/data structures you used, what features are included). Make clear which parts were implemented from scratch vs. obtained via an existing package.

**Your answer:** To implement MEMM, we took our original Viterbi algorithm and added The parameter of our Logistic Regression function. We replace the Transition matrix with our Logistic Regression function. To implement our LR, we used the sklearn.LogisticRegression function from the sklearn module. We used the pandas module in order to convert our data into one-hot encoding, so we could feed the data into the LR function. For our features, we used the POS of the given word,the POS of the next word and the POS of the previous word.

**→ Q3.2: Results Analysis**

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.

**Your answer:** We evaluated The MEMM by running The F1 mean function given to us against both the HMM Viterbi and the MEMM implementation of Viterbi. We used The same train and validation data for calculating the F1 score by using the sklearn.model\_selection.train\_test\_split function. We experimented with 10% of the data being training and 10% Validation ,We experimented with the division being 50/50,80/ 20. 90/10 , and. 99/1.

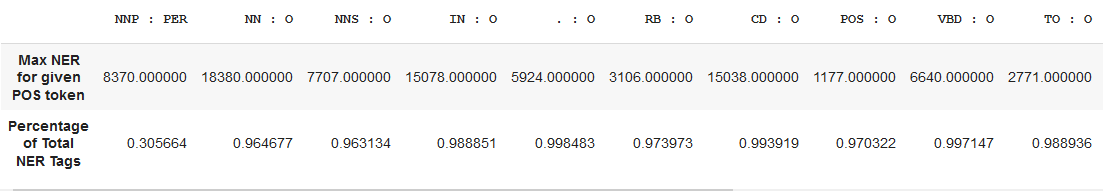
| **Train / Validation Percentage split** | **F1 Score** |
| --- | --- |
| **10 / 90** | **39%** |
| **50 / 50** | **57%** |
| **80 / 20** | **61%** |
| **90 / 10** | **60%** |
| **99 / 1** | **67%** |

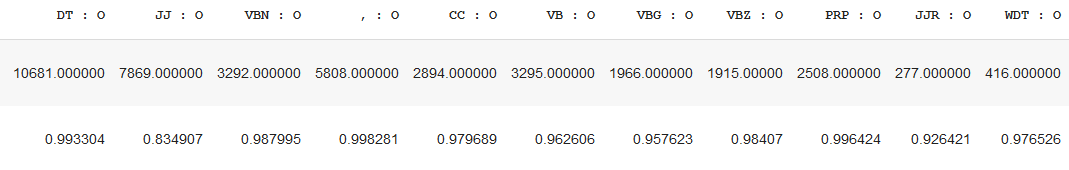
**→ Q3.3: Feature Engineering**

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?

**Your answer:** The most important features are definitely the parts of speech that are more associated with PER, MISC, ORG, LOC (NNP is the one that classified them the most.) These features make sense because they are directly tied to what a text token would be classified as. There is a percentage probability that we can generate on the training data of when a word token will have a NNP POS token and when it is classified as a PER NER token.

[A Breakdown of some of the data and how we know that PER has the most amount of tokens within the NNP POS token. Within some of this sample data, we can see that other than NNP, O dominates the majority of POS tags. This is why our feature set primarily is able to classify PER tokens more accurately than our normal Vitebri but for other tags will classify them not as well and give them the NER tag 'O' ]





We've experimented with using the entire POS tags as features, We've tried removing some of the POS tags that mainly identified O NER tags (this gave us a worse F1 score than using the entire POS tag set. This makes sense because we are just giving the system less information on how to classify the text tokens.) We tried the POS tag set along with the previous POS tag in the observation. This improved our F1 Score by about 2%. We then tried to see if combining both the POS set, POS Prev, along with a POS Next feature would recreate a sort of transition matrix that we removed from our system. Altogether, this has 45 x 3 features (45 POS tag of current word, 45 POS tags of the word previous, 45 POS tag of the next word) for a total of 135 features

**→ Q3.4: Room for Improvement**

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

**Your answer:** Our system worked well when the input data had unknown PER LOC, ORG fed into it. It was able to detect and classify them correctly because our feature engineering is based around the POS the token is. Sometimes it had trouble linking together organizations that were more than more word and the token was similar to a trained name. Since there is no transition matrix within our system, the probability of determining that the entire phrase is the same organization is more loose and prone to error

**Section 4 Questions**

**→ Q4.1: Result Comparison**

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

**Your answer:** HMM performs better because of the transition matrix it has. It is more able to link NER tokens together when a long chain of non 'O' NER tokens appear. Sometimes it works to its disadvantage and over classifies things as 'O' when it encounters data it never has seen before

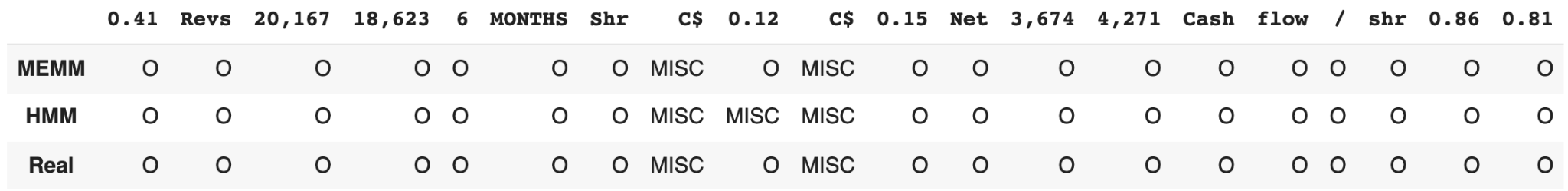
**F1 Mean score calculated with a 80 / 20 train test split**

| **HMM Validation score** | **MEMM Validation score** |
| --- | --- |
| **0.658** | **0.635** |

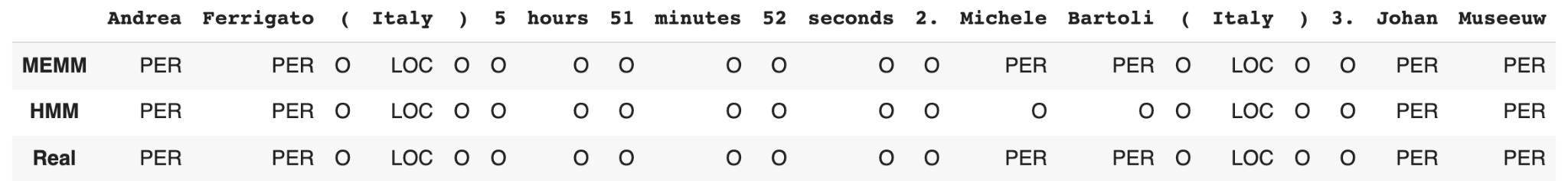
**→ Q4.2: Error Analysis 1**

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

**Your answer:**

****

Here is an example of when the transition matrix that we took out of our MEMM makes our MEMM algorithm perform better than the HMM. Because of the Transition matrix in the HMM, the base viterbi algorithm accidentally classifies a word as MISC when surrounded by other MISC tags. Since the MEMM doesn’t use the transition matrix in its calculations, it doesn’t run into this problem.

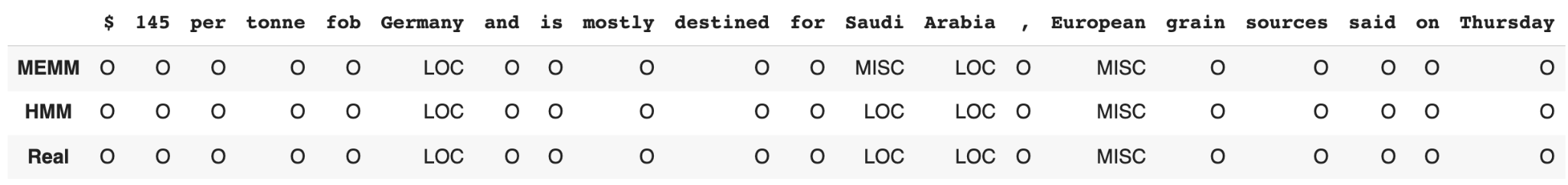


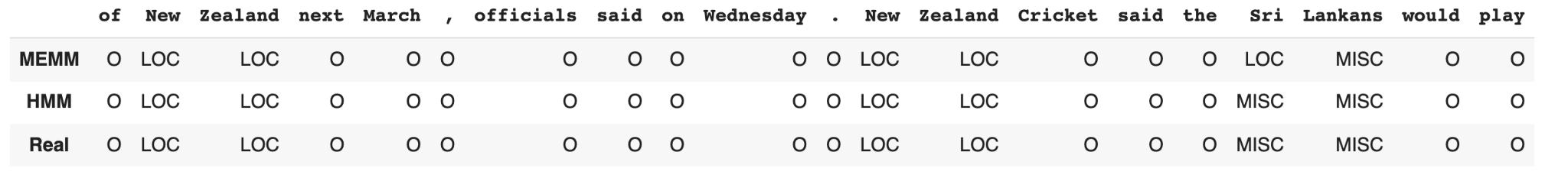
In this example, the HMM isn’t able to classify the name “Michele Bartoli” correctly because since the name is unknown, it isn’t within the emissions matrix. Since it’s not in the emissions matrix, the HMM isn’t able to classify it as a person. On the other hand, our MEMM is able to detect that this is a person because we fed in the POS of the token into our Logistic Regression function.

**→ Q4.3: Error Analysis 2**

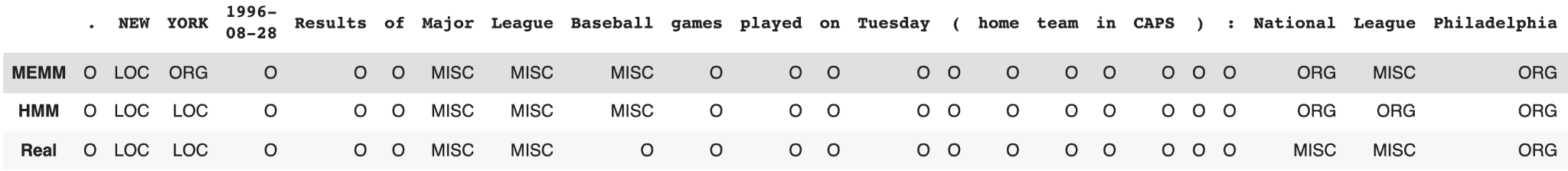
What are error patterns you observed that MEMM makes but the HMM does not? Try to justify what you observe? Please give examples of texts from the dataset.

**Your answer:**

****

In this example, we see that the HMM is able to correctly classify “Saudi Arabia” as a LOC while our MEMM divides it into “MISC” and “LOC”. This is because there isn’t a transition matrix within our MEMM algorithm so it doesn’t tie the word “Saudi” to “Arabia”. Even though the MEMM has access to the previous and next POS tags, it is still not able to correctly identify this case. 

Here is another example of a similar case^^



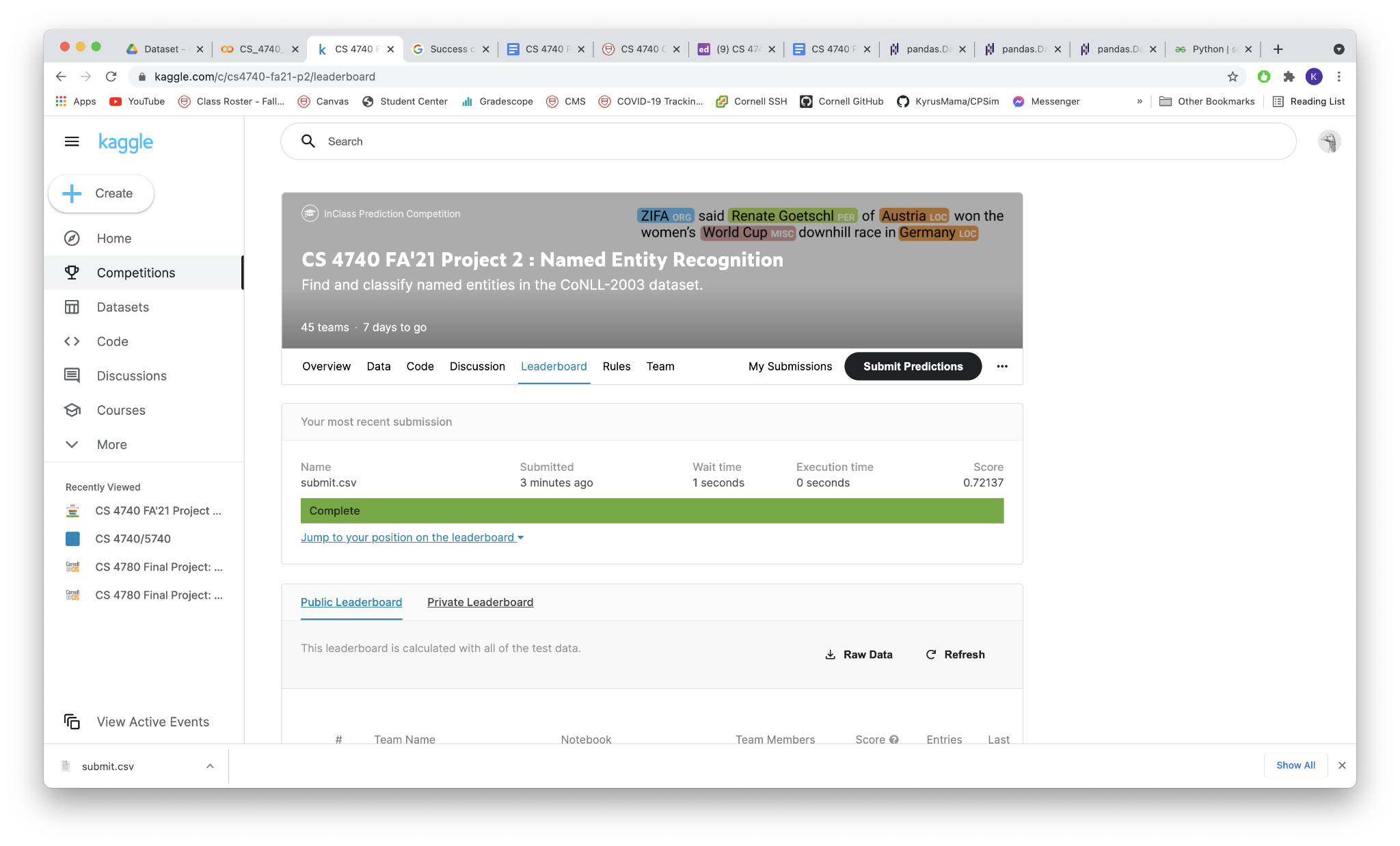
With our MEMM, it has trouble categorizing LOCs and ORGs together.

**Section 5 Questions**

**→ Q5.1: Competition Score**

Include your team name and the screenshot of your best score from Kaggle.

**Your answer:**



**Additional Questions**

**→** Please briefly describe how you divided the work.

**Your answer:** We worked together in person for the entire project and did pair programming.

**Lastly:** Remember to fill in the project feedback document! Thanks