

CS 4740 Project 2 Write Up Document

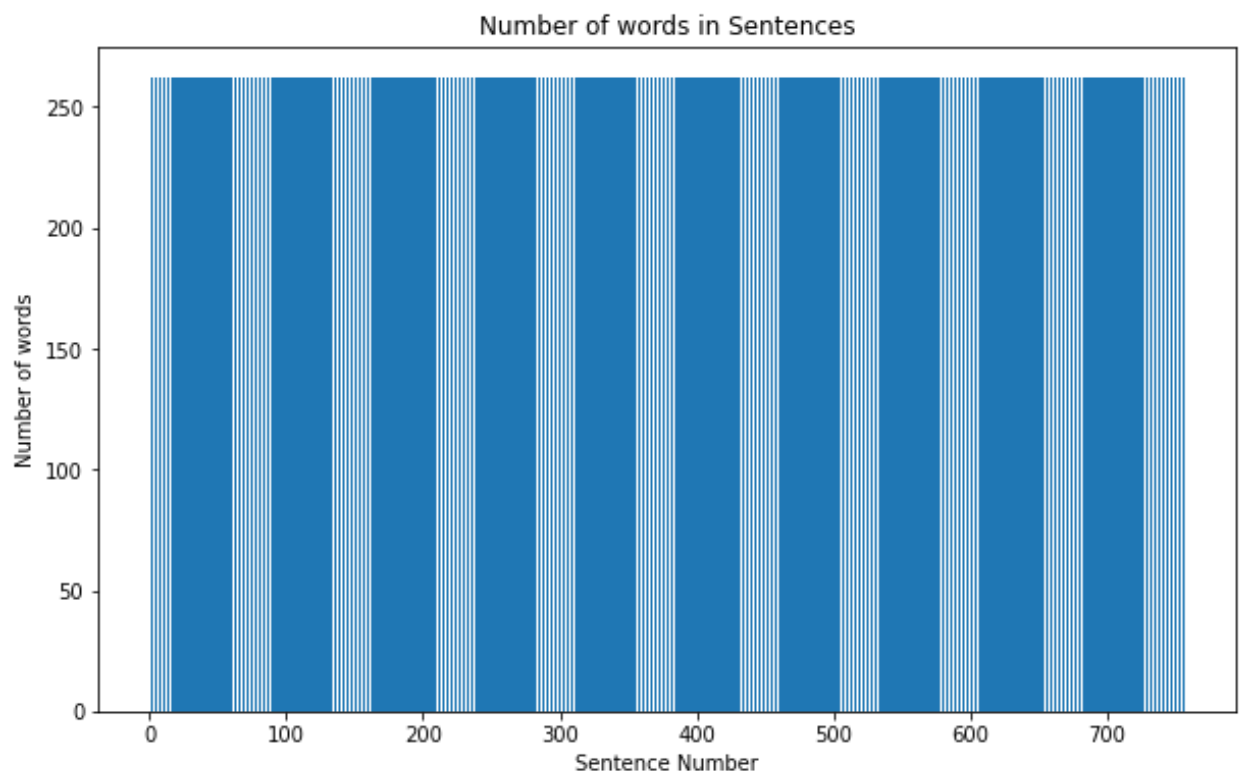
Section 1 Questions

→ 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]...]`.

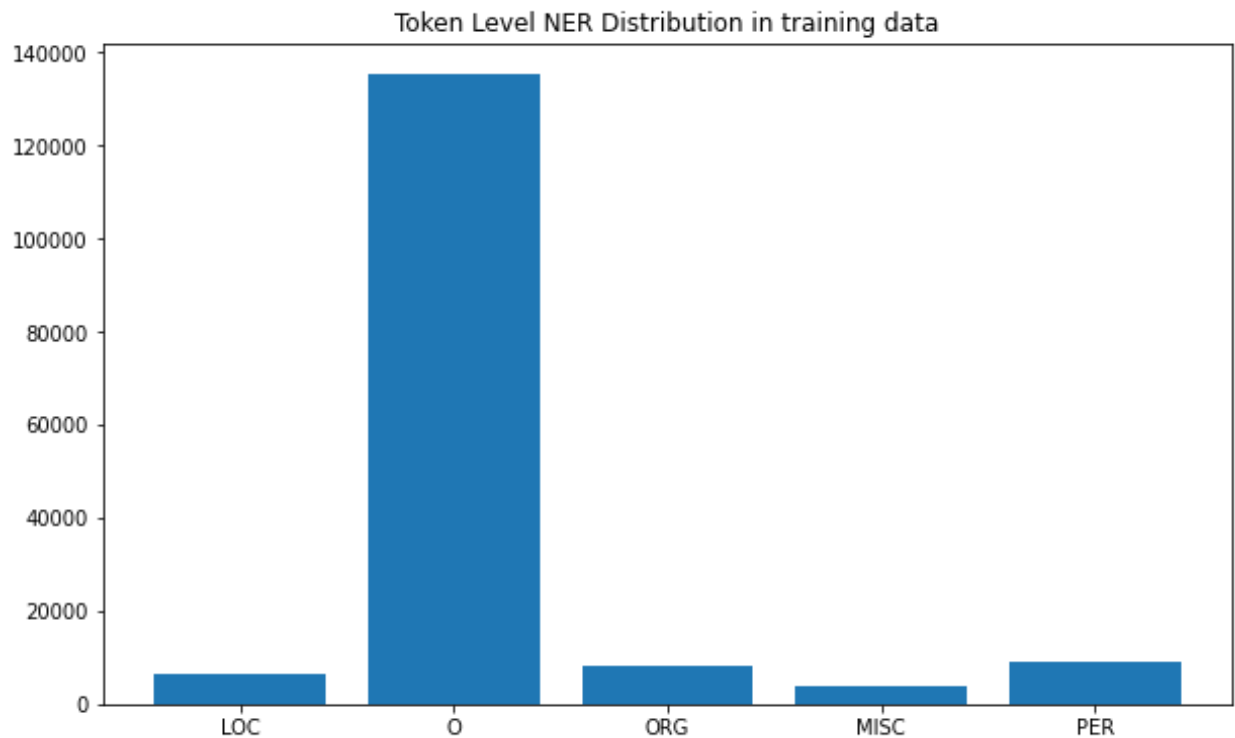
Your answer:

- A. We observe that there are overall 162341 words in the training data set while there are only 20767 unique words. Also, every document encountered has maximum of 262 words.

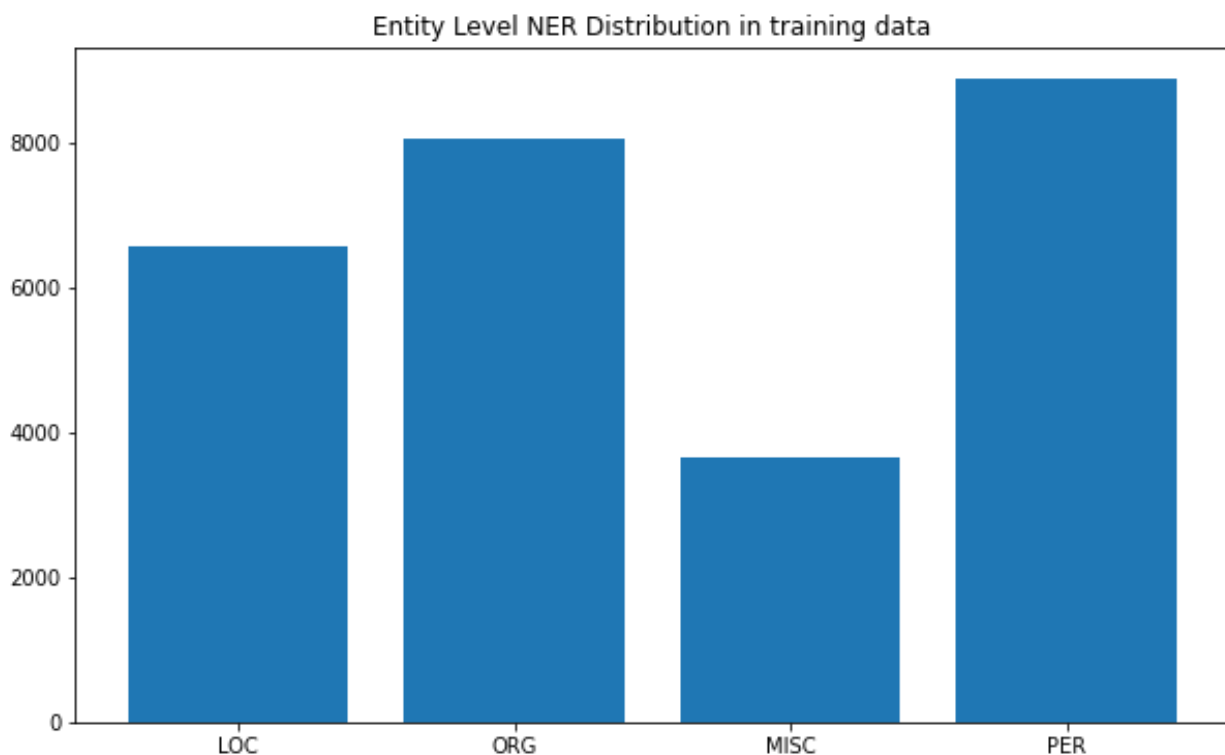


- B. At a token level, we observe that nearly 13,600 of the tokens were assigned the "O" NER tag while the distribution among the other major entity NER tags were much smaller with "PER" tag coming in second with a count of 8875 followed by

the “ORG” tag with 8065 tokens. This is the indication of an unbalanced training data and can potentially introduce challenges while performing our classification task



C. At the entity level, we observe that “PER” tag had the highest count with 8875 words while “MISC” tag had the lowest with 3659 tokens.



D. Visualizing a sample document from the training data with the NER tags assigned.

```
[ [ LOC Romania ] state budget soars in June . [ LOC BUCHAREST ]  
1996-08-28 [ LOC Romania ] 's state budget deficit jumped sharply in  
June to 1,242.9 billion lei for the January-June period from 596.5  
billion lei in January-May , official data showed on Wednesday .  
Six-month expenditures stood at 9.50 trillion lei , up from 7.56  
trillion lei at end-May , with education and health spending  
accounting for 31.6 percent of state expenses and economic subsidies  
and support taking some 26 percent . January-June revenues went up  
to 8.26 trillion lei from 6.96 trillion lei in the first five months  
this year . [ LOC Romania ] 's government is expected to revise the  
1996 budget on Wednesday to bring it into line with higher inflation  
, new wage and pension indexations and costs of energy imports that  
have pushed up the state deficit . Under the revised version state  
spending is expected to rise by some 566 billion lei . No new  
deficit forecast has been issued so far . In July the government  
gave a 6.0-percent wage and pension indexation to cover energy ,  
fuel and bread price increases , which quickened inflation to 7.5  
percent last month . In the original state budget , approved in  
March , revenues were envisaged at around 16.98 trillion lei and  
expenditures 20.17 trillion lei for 1996 . The state budget deficit  
was originally forecast to be 3.19 trillion lei for the whole year .  
On Wednesday , the leu 's official rate was 3,161 to the dollar . --  
[ ORG Bucharest ] [ ORG Newsroom ] 40-1 3120264 ]
```

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:

The Viterbi algorithm for HMM was implemented using the below data structures:

- a. Dictionaries
- b. Lists

The components used are:

- a. `trans_prob` – A dictionary to store the transition probabilities (from a prior NER tag) for every NER tag possible for a token
- b. `init_prob` - A dictionary to store the initial tag probabilities for every NER tag possible for a token

- c. `emmis_prob` - A dictionary to store the emission probabilities (to a particular token) for every NER tag possible for a token
- d. `unseen_emmis_prob` - A dictionary to store the unseen emission probabilities (using unknown word handler “unk”) for every NER tag of a token
- e. `build_hmm` – the function that generates the all the above-mentioned probabilities
- f. `NER_path` - A list, to store the NER’s predicted by our algorithm
- g. `NER_word` - A list, to store the associated tokens of the predicted NERs
- h. `NER_prior` - A list, to store the maximum of predicted NER probability values for a given token
- i. `NER_index` - A list, to store the index of the predicted NERs
- j. `NER_actual` - A list, to store the actual NERs of tokens
- k. `NER_prob` - A dictionary, to store probabilities of the predicted NERs

*Note: All of the above were implemented from scratch without the use of any packages.

Design Choice:

1. We have decided to use dictionaries rather than matrices to store all of the required probabilities for the Hidden Markov Models considering the computing time and efficiency of storage.
2. We used a simpler method to deduce the predicted NERs sequenced in any document. Instead of using a back pointer to reveal the NER sequence with highest probability, we built the NER sequence path on the fly as and when we recover the highest probable NER for any word in a sequence. This could potentially save computation time.
3. Though we have introduced unknown word handlers, we finally decided not to implement them in the Viterbi algorithm considering how they negatively impacted the F1 scores, our metric for performance evaluation. Thus, unknown emission and unseen emission probabilities are simply calculated based on zero counts of the NER-token combination as per below formula

$$P(\text{unk em. prob.}) = \frac{k}{C(t_i) + kV}$$

Where $C(t_i)$ is the total count of all words in training set and V is the vocabulary size of the same.

→**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:

Considering we used F1 score as performance metric, in our Validation step, we decided on the following:

1. Split the training data into a training a validation sets, with nearly 35.5% of the data forming the Validation set and the rest being the new training set.
2. We ran the algorithm with and without the unknown word handler and saw a considerable improvement in performance without the unknown word handling procedure.
3. Manipulating the K value of emission probabilities, also showed a goof improvement in the classifier performance. A K value of 0.002 showed the most optimal performance

Performance Summary:

Train Test Split	K value Emission		
		1	0.002
	10%	with UNK:0.438- F1 score	with UNK:0.457- F1 score
		without UNK:0.4982- F1 score	without UNK:0.483 F1 score
	35.50%	with UNK:0.463 – F1 score	with UNK:0.5001- F1 score
		without UNK:0.541- F1 score	without UNK:0.582- F1 score

A

→ 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:

Problems:

- We ran the algorithm with and without the unknown word handler and saw a considerable improvement in performance without the unknown word handling procedure.

Inference:

We concur that this phenomenon might be due to the fact that the sample we have taken for training is unbalanced, which is true in this case since there are far more “O” tags than the NER tags in the data. Thus treating unknown word with unk might actually result in more of these words being tagged with “O” than their appropriate class of NER tags(since there are fewer examples to generalize from in the training data)

Solution:

Some of the ways to handle unbalanced classes are:

1. Resampling -> Removing samples from oversampled classes or repeating samples (with some variations) from the under sampled class.
2. Removing cases of redundant data.

Snapshot of Predicted vs Actual Confusion Matrix:

Predicted	LOC	MISC	O	ORG	PER	All
True						
LOC	1786	19	488	95	16	2404
MISC	61	941	393	37	5	1437
O	17	28	50652	41	12	50750
ORG	233	71	1442	1253	28	3027
PER	27	9	1740	26	1439	3241
All	2124	1068	54715	1452	1500	60859

Observation:

Going by the numbers we see that “LOC” is correctly classified nearly 74% of the times, while “ORG” and “PER” are always misclassified more than 50% of the times.

Analysis:

Overall, the classifier has to be trained better in all the tags other than “O” but even more so to classify “ORG” and “PER” tags.

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

We see that unknown word handling was detrimental to the performance of the classifier and the algorithm performed better without its inclusion. On the other hand optimising a smoothing parameter for all the probabilities used here (transition, emission etc) brought a reasonably good improvement in performance of the

classifier and thus marks the importance of smoothing, apart from its usual role to resolve 0 probability cases occurring in the HMM model.

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:

The Viterbi algorithm for MEMM borrows many features from that of the HMM implementation (with exception to the transition probabilities) and was implemented using the below data structures:

- a. Dictionaries
- b. Lists

The components used are:

- a. maxent_prob – A dictionary to store the NER probabilities given its features – this is generated by the Maximum entropy/logistic regression model implemented in the build_memm_transition function
- b. init_prob - A dictionary to store the initial tag probabilities for every NER tag possible for a token
- c. emmis_prob - A dictionary to store the emission probabilities (to a particular token) for every NER tag possible for a token
- d. unseen_emmis_prob - A dictionary to store the unseen emission probabilities (using unknown word handler “unk”) for every NER tag of a token
- e. build_memm – the function that generates the all the above-mentioned probabilities
- f. NER_path - A list, to store the NER’s predicted by our algorithm
- g. NER_word - A list, to store the associated tokens of the predicted NERs
- h. NER_prior - A list, to store the maximum of predicted NER probability values for a given token
- i. NER_index - A list, to store the index of the predicted NERs
- j. NER_actual - A list, to store the actual NERs of tokens
- k. NER_prob - A dictionary, to store probabilities of the predicted NERs

*Note: All of the above were implemented from scratch without the use of any packages.

Other functionalities used:

1. We imported chain function from itertools to flatten out the lists of lists (used to store tokens)
2. We imported the dictvectorizer from the sklearn.feature_extraction library to help extract features
3. We imported the Logistic Regression from the sklearn.linear_model library to help implement the logistic regression model
4. We imported copy library to use the deepcopy function

Design Choice:

1. We have decided to use dictionaries rather than matrices to store all of the required probabilities for the Hidden Markov Models, considering the computing time and efficiency of storage.
2. We have decided to use a simpler method to deduce the predicted NERs sequenced any document. Instead of using a back pointer to reveal the NER sequence with highest probability, we have decided to build the NER sequence path on the fly as and when we recover the highest probable NER for any word in a sequence.
3. Though we have introduced unknown word handlers, we finally decided not to implement them in the Viterbi algorithm considering how they negatively impacted the F1 scores, our metric for performance evaluation. Thus, unseen emission probabilities and unknown emission probabilities are simply calculated based on zero counts of the NER-token combination as per below formula

$$P(\text{unk em. prob.}) = \frac{k}{C(t_i) + kV}$$

Where $C(t_i)$ is the total count of all words in training set and V is the vocabulary size of the same.

4. Within the Logistic regression model:

- a. We implemented 4 features taking into account the 'previous word' and 'next word' and similarly 'previous token' and 'next token' of a given token.
- b. Since our features were lower in count, we decided against the use of regularization in the MaxEnt model.
- c. In the MEMM model we replaced the transition probabilities from the HMM model with the probability generated by logistic regression for the same NER tag.

→ 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:

Considering we used F1 score as performance metric, in our Validation step, we decided on the following:

1. Split the training data into a training a validation sets, with nearly 35.5% of the data forming the Validation set and the rest being the new training set.
2. We ran the algorithm with and without the unknown word handler and saw a considerable improvement in performance without the unknown word handling procedure.
3. Manipulating the K value of emission probabilities, also showed a good improvement in the classifier performance. A K value of 0.002 showed the most optimal performance

→ 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:

After computing the F1 score for a logistic regression model, with and without the combination of features we considered, the following was observed:

1. Removing either the 'previous word' or 'next word' features from the model brought a small reduction in the F1 measure
2. Removing the previous POS tag or the next POS tag features didn't affect the F1 scores at all.
3. These results were also corroborated by observing the slightly higher coefficient values generated by the MaxEnt model, for the respective features (in predicting a particular NER tag). This can be further validated using the Loglikelihood test hypotheses (to check if the coefficient is statistically different from 0) and further by checking for the difference between p values for any two 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:

- a. We ran the algorithm with and without the unknown word handler and saw a considerable improvement in performance without the unknown word handling procedure.

Inference:

Similar to the case of the HMM model, we concur that this might be due to the fact that the sample we have taken for training is unbalanced, which is true in this case since there are far more "O" tags than the NER tags. Thus considering the minority classes, logistic regression model might overfit on this data and treating unknown word with unk might actually result in more of these words being tagged with "O" than their appropriate class of NER tags(since there are fewer examples to generalize from in the training data)

Solution:

Some of the ways to handle unbalanced classes are:

1. Resampling -> Removing samples from oversampled classes or repeating samples (with without some variations) from the under sampled classes.
2. Removing cases of redundant data.

Snapshot of Predicted vs Actual Confusion Matrix:

Predicted	LOC	MISC	O	ORG	PER	All
True						
LOC	1743	26	504	114	17	2404
MISC	49	954	401	24	9	1437
O	9	109	50551	67	14	50750
ORG	203	73	1456	1268	27	3027
PER	13	10	1748	22	1448	3241
All	2017	1172	54660	1495	1515	60859

Observation:

Going by the numbers we see that “LOC” is correctly classified nearly 72% of the times, while “ORG” and “PER” are always misclassified more than 50% of the times.

Analysis:

Overall the classifier has to be trained better in all the tags other than “O” but even more so to classify “ORG” and “PER” tags.

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:

The HMM F1 score was Entity Level Mean F1 score is : 0.5823354812313231

The MEMM F1 score was Entity Level Mean F1 score is : 0.5631137893518984

We see that the MaxEnt (MEMM) model performed a little lower (though not by a good margin) compared to the HMM model. This can be attributed to the fact how Logistic regression model are not immune from overfitting a data and considering we have a unbalanced majority NER class "O" here, it will overfit on the minority classes hence giving poor generalization on the test data. Although class imbalance is also a major problem for a generative model like HMM, it will score better than MEMM considering how it predicts based on the posterior which is again from the joint probability of the dependent and independent variables- which can be reasonably learned from a few data points compared to (a discriminative model like) logistic regression.

Names & Netids: (please add here)

→ Q4.2: Error Analysis 1

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

Your answer:

Comparing the confusion matrices of the two models:

MEMM:

Predicted	LOC	MISC	O	ORG	PER	All
True						
LOC	1743	26	504	114	17	2404
MISC	49	954	401	24	9	1437
O	9	109	50551	67	14	50750
ORG	203	73	1456	1268	27	3027
PER	13	10	1748	22	1448	3241
All	2017	1172	54660	1495	1515	60859

HMM:

Predicted	LOC	MISC	O	ORG	PER	All
True						
LOC	1786	19	488	95	16	2404
MISC	61	941	393	37	5	1437
O	17	28	50652	41	12	50750
ORG	233	71	1442	1253	28	3027
PER	27	9	1740	26	1439	3241
All	2124	1068	54715	1452	1500	60859

We see that in general the MEMM tends to perform worse in classifying “LOC” and “O” tag, compared to HMM. This could be due to the fact that the Logistic regression model (MaxEnt), being a discriminative model will tend to overfit on the “O” tags (since they

are imbalanced in this data) and thus may wrongly classifying the “O” tags and “LOC” tags (Highest False positives for O is 16 corresponding to its assignment for words whose actual tag was LOC).

For example,

1. While MEMM fails to classify “Diego” as “LOC” entity, HMM correctly does.
2. While MEMM fails to classify “Francisco” as “LOC” entity, HMM correctly does.

→ Q4.3: Error Analysis 2

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

Your answer:

In line with the above reasoning, we see that HMM misclassifies the “MISC” and “PER” tags more in comparison to the MEMM model. considering that it tends to overfit less being a generative model.

For example,

1. While HMM fails to classify “League” as “MISC” entity, MEMM correctly does.
2. While HMM fails to classify “Western” as “MISC” entity, MEMM correctly does.








Section 5 Questions

→ Q5.1: Competition Score

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

Your answer:

zz274_jg929_NB

77	Gary Ho		0.66748	2	7h
78	dk837		0.66418	10	7h
	Baseline MEMM		0.65021		
79	Chengrui Gu		0.60925	4	1d
	Baseline HMM		0.60082		
80	zz274_jg929_NB		0.59799	3	now
Your Best Entry 					
Your submission scored 0.59799, which is not an improvement of your best score. Keep trying!					

Additional Questions

→ Please briefly describe how you divided the work.

Your answer:

Major part of the project was done by Jayaram including the Viterbi Implementation for both the HMM and MEMM. This includes the debugging of the whole program being [presented here. Zhou Zhou helped in the implementation of the MaxEnt model and its features. The entire write up doc was produced by Jayaram.

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