CS 4740: Intro to NLP
Project 2: Named Entity Recognition with HMMs and MEMMs

Jahanvi Kolte, Tian Ren, Xingyu Chen
Kaggle Team Name: Gryffindor
jsk362@cornell.edu, tr349@cornell.edu, xc374@cornell.edu

Common Terminologies Used
NER: Named Entity Recognition, PRF: Precision, Recall, F-scores,HMM: Hidden Markov Model, MEMM: Maximum Entropy Markov Model, LGR: Logistic Regression, POS: Part of Speech

I. Baseline

For baseline systems, we considered two models that are not sequence models: most frequent class classifier and logistic regression model.

A. Most frequent class classifier

Hypothesis: The assumption to choose this model is that different NER tags for words are not equally likely. Many words are only associated with one tag and most words inside a sentence are not name entities. This indicates that a model based on frequency will have acceptable performance although it can not encode the sequence.

Approach: First, we paired all the tokens and the corresponding labels, e.g. "(word, tag)"; Second, count all the pairs; Third, loop through the count result e.g. "(word, tag), count", whenever a word is found associated with tag that has a higher count, replace the tag for the word in the "most_frequent_tags" dictionary. The "most_frequent_tags" dictionary is the trained model.

B. Logistic regression classifier

Hypothesis: We also noticed that name entities tend to have similar properties (e.g., capitalized, pos tags). In order to integrate these features in our sequential model, we trained a logistic regression classifier. Since the features used in training the classifier are all extracted from the text, we think it is reasonable to use this classifier itself as a baseline model.

Approach: First, we use indicator functions to vectorize our input according to preselected features as discussed in the design choices section and feature section; Second, we use sklearn tool to run Logistic Regression to get the LG classifier.

C. Results and Analysis: Discussed in the Results section

II. Sequence Tagging Model

A. HMM Implementation Details

Hidden Markov Model is used for named entity BIO tagging as a sequence model. It is probabilistic model that computes a probability distribution over a sequence of labels and selects the best label. The implementation of the algorithm is achieved with the bigram Viterbi algorithm.

The Viterbi algorithm keeps track of previous probability scores for each state in the previous time step and multiply it with lexical generation probabilities and transition probabilities.

Data Structures: A dictionary data structure is used to store the bigram probability of tags showed below as graph variable. Word generation probability is shown using word_tag_dict variable
graph ={ ner_tag1: { ner_tag2: score},..} ,
word_tag_dict = { ner_tag1: { word1:prob, word2: prob }}

Implementation:

1. The Viterbi scores and the back pointers, each are stored in an array of size (N,T), where N is the number of states, and T is the length of sequences.
2. The generation probability is $P(w_i \mid t_i)$, the probability of having the current word given current tag. The transition probability is a bigram probability of tags $P(t_i|t_{i-1})$. These two probabilities are stored in dictionaries whose keys are tuple of (word, tag) or (tag,tag).
3. By iterating the entire training corpus, we are able to collect unigram counts (word, tag) and bigram counts with which we can construct dictionaries to store unsmoothed probabilities. For getting the smoothed probability, we made a choice to calculate it on the fly, so we do not need to store every possible keys in the dictionaries. Also, to append a "#" to every BIO sequence so we can capture the initial probability distribution () for tags.
4. Viterbi algorithm and use a back pointer (an array) to keep track of the best sequence. For first word, the viterbi score and back pointers are computed considering the start state as #, as product of transition probability from # to state and probability of generating word of that state.
5. Then, for all words and states combination, viterbi scores are computed for a graph node from all its previous nodes. The maximum score from previous state is stored as the viterbi score and the previous state that gives the maximum score is recorded in back pointer.
6. Then we compute the maximum score we get for observation T and then backtrack it to generate sequence of labels.
7. The model is then evaluated using PRF scores

B. MEMM Implementation Details

Motivation: HMM cannot incorporate more context or long distance features.

Data structures:

We will use some data structures to help us store the frequently accessed information efficiently.

Data structure 1: It stores the probability of current ner tag given previous ner tag and current word. Key is string combination of previous tag and current word. Value is a dictionary with key as next possible ner tag and probability of getting that ner tag as value. Example is given below
{ 'B-PER Obama': {'I-PER': 0.9, 'O': 0.1}, ... }

Data structure 2: It is used to store the most likely tag given the current word. It is a dictionary with word as the key and value as another dictionary where key is ner tag and value is the number of times the word was observed as that tag
Example: {'Marlene': {'B-PER': 1},..}

Data Structure 3: This data structure is used to store the word embedding. It is a dictionary with key as word and numpy array of 300 dimensions as value. The word embedding vector is normalized
Example: {'mother': np.array([0.2, 0.4,...])}

Implementation: All the components/functions required by MEMM were written by us except logistic regression function from scikit learn library. We tried different approaches with MEMM including Glove word embedding with 840B words. If any word is not present in this set then we replace its vector with all 0s. We will describe the basic implementation below and then introduce what change were done later.

Approach 1: Using Posterior distribution

The implementation details of the vanilla version of MEMM we implemented is mentioned below

1. The Viterbi scores and the back pointers, each are stored in an array of size (N,T), where N is the number of states, and T is the length of sequences.
2. The pre computed probabilities for posterior i.e. probability of current tag given previous tag and current word is used in the format of Data structure 2 as discussed above.
3. The viterbi score and back pointers are computed for first word by using '# <current_word>' as key in data structure 2.
4. Then for all the words and for all the possible states, the viterbi score is computed using previous score and data structure 2 with key as '<previous_tag> <current_word>'. The maximum score from previous state is stored as the viterbi score and the previous state that gives the maximum score is recorded in back pointer.
5. Then we compute the maximum score we get for observation T and then backtrack it to generate sequence of labels.
6. The model is then evaluated using PRF scores

Approach 2: Using Maximum Entropy Model based on feature vector and previous tag (as feature)

1. Implementation of Viterbi for memm is same as approach 1 with the exception in the way the probability is substituted as described in approach 1.
2. Here, we use logistic regression model for training. The training corpus is vectorized (discussed in Design Choices) and trained using a logistic regression model.
3. While executing the Viterbi algorithm, we vectorize at each step, taking current_word, previous_ner_tag, current_pos_tag, word embeddings into account and then compute their probabilities according to the parameters of the trained logistic function and use it instead of the posterior probability.

Approach 3: Using Maximum Entropy Model based on feature vector and previous two tags (as feature) (like trigrams)

1. This approach is same as Approach 2 with feature addition to vector for previous ner tag of previous ner tag.

Design Choices:

Using Log Probabilities: As the Viterbi algorithm involved multiplying a sequence of probabilities, we used log probability to avoid loss of precision.

Using Normalized Vectors: The word embeddings that we used were not normalized and so it was important to normalize them so that a particular feature does not dominate because of its magnitude.

Feature Sets: We tried several combination of features. An exhaustive list is mentioned below. This is implemented using a function called vectorize that takes in necessary parameter and returns the vector representation of supplied information.

Sr No.	Features	Rationale for choosing the feature
1	is Capitalized	Most of the named entities are proper nouns like name of person, organization, location which is usually capitalized Eg. America, Barack, W.H.O.
2	is Start Of Sentence	Previous feature looks if word is capitalized but we want to keep track of its position as well. Because, start of sentence is also capitalized. Ex. The game
3	is Number	Numbers are also named entity which will be part of MISC category in our case

4	has Hyphen	Some
5	is Noun (POS Tag)	We want to know is the part of speech associated with the word is noun as it is more likely to be named entity then compared to other part of speech
6	is Proper Noun (POS Tag)	We want to know is the part of speech associated with the word is noun as it is more likely to be named entity then compared to other part of speech
7	follows a Determiner	Nouns usually follow Determiner. Ex. The United States of America
8	follows an Adjective	Nouns are mostly seen in sequences like Determiner-Adjective-Noun
9	Most likely NER Tag (9 tags correspond to 9 features. Indicator function used to select one of them)	To include information about the most likely tag for given word based on our training data
10	Previous NER Tag as per prediction (9 tags correspond to 9 features. Indicator function used to select one of them)	To include information about the previous tag that has been predicted. Useful when you are predicting I tags because they usually follow B tags
11	Previous of Previous NER Tag as per prediction (9 tags correspond to 9 features. Indicator function used to select one of them) (Only for approach 3)	To include additional information on previous two tags predicted to give more context to the model for prediction
12	Normalized Word Embedding of Current Word (length: 300) [Continuous Feature]	Because it is an easy representation of word in vector form, encodes context information, it has a way to represent unseen/unknown words
13	Normalized Word Embedding of Previous Word (length: 300) [Continuous Feature]	Previous word can give you more context to model for prediction. Also, because it is an easy representation of word in vector form, encodes context information, it has a way to represent unseen/unknown words.
14	Normalized Word Embedding of Next Word (length: 300) [Continuous Feature]	Next word can give more context to model for prediction. Because it is an easy representation of word in vector form, encodes context information, it has a way to represent unseen/unknown words
15	$P(ner\_tag_i / ner\_tag_{i-1}, word_i)$ [Continuous Feature]	To account for posterior probability.
16	$P(ner\_tag_i / ner\_tag_{i-1})$ [Continuous Feature]	To account for bigram probability of ner tags
17	$P(ner\_tag_i / pos\_tag_i)$ [Continuous Feature]	To account for pos tag information associated with current word
18	$P(ner\_tag_i / pos\_tag_{i-1}, pos\_tag_i)$ [Continuous Feature]	To account for current and previous pos tag information associated with current word
19	$P(ner\_tag_i / pos\_tag_i, pos\_tag_{i+1})$ [Continuous Feature]	To account for current and previous pos tag information associated with current word
20	$P(ner\_tag_i / word_i)$ [Continuous Feature]	Current word is a good indicator of likely tag
21	$P(ner\_tag_i / word_{i-1})$ [Continuous Feature]	Current word may be a good indicator of likely tag
22	$P(ner\_tag_i / word_{i+1})$ [Continuous Feature]	Current word is a good indicator of likely tag

Table 1: Features Experimented to add more context to MEMM

Model Training: We trained the models using two different methods.

1. **Logistic Regression**

Motivation: We wanted to use the capability of MEMM to model using discriminative techniques on feature vectors.

Approach: As discussed in Implementation, the words and other necessary details were vectorized and a logistic model was trained on these vectors. We experimented with different **regularization parameters and class weights** to evaluate our model. Features in the above table from 1-14 were used in different experiments for model trained using this method.

Results and detailed Analysis: Discussed in the Results section

2. **Random Hyperparameter Optimization**

Motivation: We did not observe expected accuracy on our validation set and so we wanted to experiment with more methods.

Approach: For an experiment with 100 samples and 100 epochs, the weight vector was sampled using a **Gaussian**

distribution. Weight vectors in this experiments were analyzed and the parameters were then fine-tuned reducing their confidence bounds to smaller intervals.
Features in the above table from 1-4, 11-22 (inc) were used in different experiments for model trained using this method.
Results and Analysis: Discussed in the Results section

C. Pre-processing

Motivation: There is only one training data set given. In order to tune our model later to improve performance, we decided to separate the data into training set and validation set. The ratio we chose is 9:1 because we believe more training examples will give us a more accurate model.
Approach: We used different pre-processes in HMM and MEMM.
For HMM, since we need to compute the transition probability but the first word in each sentence has no previous tag, we assigned a start symbol "<s>" and a tag "#" for each sentence. We also changed a small portion of the infrequent words (count=1) to "<unk>" to deal with unknown words, because unknown words usually are words that are not used often.
In MEMM, we need to use assigned tag for the previous word as a feature in classifying the next word. Hence, we add two start symbols and one end symbol to help compute the first and last words in sentence. Since MEMM can take advantage of arbitrary features, we modeled unknown and unseen words by adding binary features like different word shapes, pos tags and word embeddings.
Results and Analysis: Discussed in the Results section

D. Model Evaluation

Approach:
To evaluate the model, we implemented a function which counts correct number of prediction on the span level and computes the scores. For example, "B-ORG I-ORG I-ORG" is counted as one correct prediction but "B-ORG I-ORG" and "B-ORG B-ORG I-ORG" are not if the answer key is "B-ORG I-ORG I-ORG".
Results and Analysis: Discussed in the Results section

E. General Experiments

[Note: All the results are mentioned in Section: Results]

- **Unknown words**

Motivation: 0 probabilities for words not in training set
Hypothesis: Our method should address unknown words and avoid 0 probabilities to sequences
Expectation: Avoid 0 probabilities for words not in training set
Approach: To deal with the potential unknown words in validation and test sets, our idea is to generate a fixed vocabulary set **V** beforehand and assign any out of vocab token as “<UNK>”. Initially, we assigned a portion of words in the training set as unknown words based on its frequency. Yet we found around 50% word among all unique words only occurred once. Therefore, we randomly select a portion of words (denoted as **u**) that occurred once as OOV words and the remaining words in the corpus is our vocabulary set. The different unknown word proportions (**u**) are experimented over the dev set to get a best one in terms of validation accuracy.
Results and Analysis: Discussed in the Results section

- **Unseen words**

Motivation: 0 probabilities for sequences of words and tags not seen in training set
Hypothesis: Our method should address unseen words and avoid 0 probabilities to sequences
Expectation: Avoid 0 probabilities for sequences of words and tags not seen in training set
Approach: Unseen bigram would be unseen combination of tags or word and tag for the HMM. We are using smoothing technique to estimate probability distribution of unseen bigram. For example, the generation probability, a bigram combination of <peacefully, ‘B-ORG’> does not occur in the training corpus.
Add-k smoothing is a good method to start with since it is simple to implement. The smoothed probability for an unseen bigram for generation probability is calculated on the fly as 1/(Count(tag)+V) if using add-1 smoothing, where V is the vocab size.
Results and Analysis: Discussed in the Results section

- **Add-k smoothing**

Motivation: Add-k smoothing is a good method to start with since it is simple to implement. To avoid 0 probabilities
Hypothesis:
Expectation
Approach: The generation probability is smoothed with add-k smoothing with the formula below:
where **V** is the total number of words in our vocabulary set (including the ‘UNK’).
Similarly, for the smoothing of transition probability,
we have:
where **N** is the total number of tags. Note that C(tag) also includes the start of sequence tag (“#”) we added, in this way the initial probabilities () are smoothed as well.
Therefore, there are 2 parameters, **k1** and **k2**, to tune with for the add-k smoothing. In our experiment, we used **random search**, which is a hypermeter optimization method to optimize the choice of k. In random search, we randomly select k from a set range of (0,10). It random selects k as any number in this range in an epoch and we ran 100 epochs to get the ones give the best performance on the dev set. Also, k is considered as a float number and can be less than 1 since we are looking to have more choices of k.
Results and Analysis: Discussed in the Results sections

- **Experiments with features are explained in Implementation section**

F. Results

Model Evaluation is described in section II D
Results for different experiments we ran are described in the following tables

u	P	R	F
0.01%	0.743	0.656	0.697
0.02%	0.810	0.641	0.715
0.05%	0.809	0.638	0.713
0.1%	0.837	0.621	0.720
0.2%	0.844	0.625	0.719
1%	0.848	0.620	0.717
2%	0.848	0.615	0.713
10%	0.85	0.617	0.715

Table 2. Unknown words proportion for different **u**, where **u** stands for the proportion of unknown words among the words that only occurred once in the training corpus. Here we are using add 1 smoothing for both the generation probability and transitional probability.

K1	**K2**	**P**	**R**	**F**
0.97	6.7	0.833	0.65	0.731
0.27	5.9	0.842	0.723	0.778
0.2	3.2	0.838	0.774	0.805
0.5	5.1	0.853	0.7	0.701
0.2	6	0.854	0.784	0.817
0.8	5.9	0.854	0.656	0.747
0.03	6.1	0.861	0.797	0.828

Table 3. Add-k smoothing for different k1.k2. k1 is to smooth generation probabilities. k2 is to smooth transition probabilities.

We tried multiple feature combinations and collected the span level performances on our validation set.

Experiments	**Precision**	**Recall**	**F1**
Baseline Model - Most Frequent Class	0.716	0.744	0.690
Baseline Model - Logistic Regression	0.594	0.780	0.675
HMM with add k (optimized)	0.710	0.744	0.727
MEMM with only $P(t_i \vert w_i, t_{i-1})$ [using posterior probability]	0.568	0.756	0.649
MEMM with indicators for most frequent tag	0.709	0.642	0.674
MEMM with current word embedding	0.595	0.780	0.675
MEMM with word embeddings for prev, curr, next words and LGR (regularizer $C = 1$)	0.644	0.803	0.715
MEMM with word embeddings for prev, curr, next words and LGR (regularizer $C = 0.01$)	0.649	0.796	0.715
MEMM with normalized word embeddings for prev, curr, next words and LGR (regularizer $C = 0.01$)	0.817	0.659	0.730
MEMM with normalized word embeddings for prev, curr, next words, log probability LGR, proper noun and determiner	**0.866**	**0.822**	**0.843**
MEMM with hand tuned weights	0.852	0.810	0.830
MEMM with Approach 3	0.669	0.641	0.654

Table 4: PRF Results for different HMM/MEMM models and parameters for validation set

Analysis:
Baseline: The span level performance of the two baseline model on our validation set are shown in the following table. We can see that both models have 0.67 as the f-score, which is consistent with our hypotheses stated in the baseline section.
Analysis on Test Set: Refer section H
Model Analysis: Discussed in next section G
Possible Improvement: Experiment with new features eg. trigram transition probabilities, using deep learning models, other classifiers

G. Model Comparison and Error Analysis

Comparison: According to the table, we can see that, when we use very few features, HMM outperforms MEMM. However, the performance of MEMM improves as more and more features are added to the model. This makes sense because the advantage for MEMM over HMM is that it can use arbitrary features and encode as many features as possible. When there is only very little extra information about the context, MEMM loses its advantage. But when there are a lot of relevant features, MEMM eventually performs well because it has much more context information than HMM. As to the run time, HMM is faster. Features with higher weight are more important for prediction. For MEMM Hand Tuned features, We observed that continuous features for part of speech had huge impact on our development accuracy which is valid as nouns are potential candidate for NER Tag. Also, word embeddings had very less weight assigned to its features indicating that using them only marginally improves accuracy. We observed a drop in test accuracy over validation accuracy as we have included too many features and may have overfit the model.

Error Analysis: Discussed in Table 6 with NER errors marked in red.

Model / Category	**ORG**	**MISC**	**PER**	**LOC**
HMM	0.68	0.64	0.64	0.84
MEMM	0.70	0.73	0.79	0.82

Table 5: HMM and MEMM prediction accuracy comparison on dev set for specific named entity

Analysis: From Table 5, we can see that MEMM performs better in every type of named entity classification over the dev set than the HMM. MEMM performs significantly better on MISC and PER classification.

Error Type	Senario	Example	Possible Reason
Common Errors (Misclassification)	surrounded by numbers (labeled as "O")	['3.', 'Bscher', '112']	1.numbers are rare occured words 2.NERs are barely connected with numbers
	at beginning of sentence (labeled as "O")	['MONTGOMERY', ',', 'Ala', '.']	1.limited examples 2.no previous context for beginning words
	lowercased (labeled as "O")	['Following', 'their', 'midweek']	Capitalization is heavy weighted feature
Errors for Only HMM	named entities inner relation (labeled as "O")	['Germany', ')', 'Ferrari', '1:51.778'] ['B-LOC', 'O', 'B-ORG', 'O']	HMM are unable to capture long distance dependency on context
Errors for Only MEMM	same words are labeled differently in same sequence	'Mickelson' classified as PER and O in same sequence	MEMM incorporated too much context: Surrounding features are more important than the word itself
	B, I tag inconsistency	['B-ORG', 'I-PER', 'O', 'O', 'O']	MEMM gets confused about the context because of too many features are included

Table 6: Error Analysis for HMM and MEMM (Instances classified incorrectly highlighted in Red)

H. Kaggle Competition Details

Kaggle Submission (Team Name: Gryffindor)	**Test Set Accuracy**
Baseline (Greedy probabilistic approach)	63.33%
HMM (with k1=0.03, k2=6.1)	71.69%
MEMM (with feature vector and logistic regression, C=0.01)	72.1%
MEMM (Hand Tuning Weights)	78.95%

According to results, The model that gives us best validation accuracy gives us best test accuracy as well

25	new	Gryffindor	0.78955	9	3h

Screenshot

III. Workflow/Team Contributions

All of us worked on all the sections and implemented several methods for comparison.

Task	**Member**
Pre-processing	Jahanvi, Xingyu, Tian
HMM	Jahanvi, Xingyu, Tian
MEMM	Jahanvi, Xingyu, Tian
Kaggle	Jahanvi, Xingyu, Tian
Report	Jahanvi, Xingyu, Tian

IV. Run-Time Analysis

Average Run Time for HMM and MEMM we observed is 18.59s and 230.8s

V. Proposal Answers

1. For our setting, we used the hidden variables are the NER labels and the observed variables are the word sequences. Our HMM model deals with unknown words using add-k smoothing. Here, k is a model parameter. Assumptions regarding the Markov Model can also be treated as parameters. Hence, the N in the N-gram model used to model transition probabilities can also be treated as model parameter.
2. The HMM is useful for determining sequence of events that are not directly observable and also relatively simple to implement but it cannot incorporate long distance features. Other techniques like feature-based, rule-based and neural algorithm may also be effective.
3. MEMM can incorporate long distance and any arbitrary features but HMM can only use previous states. However, MEMM is computing expensive and slower than HMM.
4. We used GloVe word embeddings(840B.300d) and tags indicator for three continuous words, capitalization etc.. We downloaded the embeddings online and converted features to binary indicator functions on our own. We chose continuous three words because this is a sequence labeling task and we wanted to capture as much context information as possible. We chose capitalization based on the intuition that normally name entities are proper nouns hence capitalized.