Project 2
CS 5740 Natural Language Processing

Mengyao Lyu (ml2559), Ruoyang Sun (rs2385), Qianhao Yu (qy99)
Kaggle Name: eye_ball

Sequence Tagging Model

A. **Implementation Details** (All algorithms are implemented from scratch)
   - HMM algorithm
     i. We want to calculate the largest probability of the NE (Name Entity) tagging of a given sentence, $i.e.$ To maximize $P(w_1, ..., w_n | t_1, ..., t_n)$, where $w_1, ..., w_n$ stand for a given sentence and $t_1, ..., t_n$ are corresponding NE tags.
     ii. According to HMM method, $P(w_1, ..., w_n | t_1, ..., t_n) = \Pi P(t_i | t_{i-1}) * P(w_i | t_i)$, where transition probability $P(t_i | t_{i-1})$ is approximated using bigram and emission probability $P(w_i | t_i)$ is calculated by conditional probability between words and their corresponding NE labels in the training data.
     iii. Back pointer: As we get the total probability of the NE tagging from the last word of the sentence, we will need to go backward from the last word, who has the maximum total probability, to find the correct NE's for each previous word. The algorithm is: $v_t(j) = max^N{}_{i=1} v_{i-1}(i) * a_{ij} * b_j(o_t)$, where v_i and v_{i-1} represents the current and previous Viterbi path probability, a_{ij} represents the transition probability from previous state qi to current state q_j , $b_j(o_t)$ is the state observation likelihood of the observation symbol o_t given the current state j .
   - Validation dataset
     i. We separate 10% data from the labeled dataset as testing data and use the rest 90% as training data.
   - Making Predictions
     i. While reassembling the NE tags for the corresponding sentences, we removed outstanding **I-xxx** type and group NE tags who start with **B** and followed by ≥ 0 number of **I-xxx** of the same type.
        - $e.g.$ If the output NE tags are: "O, O, I-ORG, O, **B-PER, I-PER, I-PER**, I-ORG, O", we will count **B-PER, I-PER, I-PER** as proper NE tags and ignore the rest.

B. **Pre-processing**
   - N-gram

   i. In the initial HMM models, we used bigram to calculate the transition probability.
   ii. Later to improve our model performance, we used trigram (as extension).
- Dealing with 0's in HMM
   i. We decided to not use smoothing to deal with 0's in HMM since all the 0's would be assigned to the same value, which might make it hard to select maximum value for the back pointer in HMM algorithm. Instead, we randomly assigned a very small value from 1e-10 ~ 1e-11 to the 0's.
- Unknown words
   i. In the baseline system, we regarded words that appeared only once in the training set as unknown words ('**<UNK>**').
   ii. Advanced method to deal with unknown words: (Analyzed below).

C. Experiments

- **Add trigram as extension (Analyzed below)**
- **Add interpolation:**
   i. We implemented a line search on the trigram model: $\lambda_1 * P(t_i) + \lambda_2 * P(t_i | t_{i-1}) + \lambda_3 * P(t_i | t_{i-1} t_{i-2})$ and the best performance on validation set was based on $\lambda_1 = 0.0,\ \lambda_2 = 0.12,\ \lambda_3 = 0.88$.
- **Advanced unknown:**
   i. Usually, we would treat the words that only appeared once in the training set as unknown words. In this practice, instead of replacing them with '**<UNK>**', we used its **POS** tags as their tokens to obtain more information of the unknown words.
   ii. In the test data set, if there still were some unknown words, we treated its relevant emission probability as small random number (1e-10 ~ 1e-11).
   iii. The performance on Kaggle using this trick improved 3%. (See result below)
- **Practice: Modify Output/Prediction**
   i. **Reason**: according to our previous prediction, we observed that recalls were lower than precision, which means we lost many NE by labeling them as "**O**".
   ii. **Solution**: If there are more than 1 "**I-xxx**" in the data set, we add this phrase to our output.
   iii. **Result**: But the result was not better than our original process, so we skipped it.

D. Results

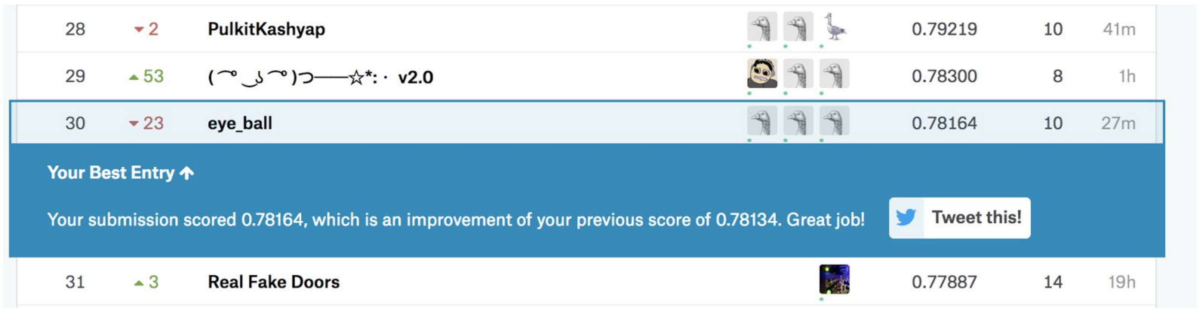
Accuracy	Validation Set			Kaggle Score
	Recall	Precision	F1-Measure	
(Baseline)	N/A			0.4458
HMM + Bigram	0.655	0.8187	0.7278	0.7401

- **Compare with Baseline System**: The baseline system achieves an accuracy of 44.58% by choosing the highest frequency of the NE tag associated with each word ('**O**' otherwise), and regarding words with frequency 1 in the training set as unknown words (**<UNK>**). After incorporated HMM algorithm with bigram, we believe the model will be significantly improved since HMM algorithm considered the ordering or words in a sentence and bigram considered the correlation between neighboring words. As a result, our new model achieves significant improvement in accuracy and reached 74.01% as expected.
- Error Analysis on validation set: We looked into some prediction errors occurring in the validation set. The bigram model didn't perform very well on recognizing Person's name. One example in the validation set is: **['7.', 'Ales', 'Valenta', '(', 'Czech', 'Republic' , ') ', '194.02']**. The prediction from our bigram model was **['O', 'O', 'O', 'O', 'B-LOC', 'I-LOC', 'O', 'O']**. It successfully recognized the location entity "**Czech Republic**", but failed to detect the name entity "**Alex Valenta**". We ascribed this to the way of handling unknown words. Since name in the test set is more likely to be out of dictionary, it has higher probability to fall into <Unknown> category and more likely to be classified as **'O'**. We will investigate some other methods of handling unknown words in the extension section.

2. Extension

Accuracy	Validation Set			Kaggle Score
	Precision	Recall	F1-Measure	
HMM + Bigram	0.8187	0.6550	0.7278	0.7401
HMM + Trigram	0.8309	0.6847	0.7507	0.7508
Add Interpolation	0.8337	0.6883	0.7510	N/A
Advanced <UNK> Method + Interpolation	0.8166	0.7513	0.7826	0.7816

- To further improve our model, we believe that using trigram to compute transition probability will also be helpful based on our previous experience in project 1. The result is 75.08% with increments in both recall and precision compare to previous bigram model. Adding interpolation to ngram is also another common practice to improve model performance, therefore in this case we believe interpolation may also help. As a result, the hyperparameter λ from our trained interpolated model indicated that we should go with trigram model since the weight of trigram is almost 1. The reason could be that for long NE, trigram is more powerful to capture the relation between each word with the NE. As a result of implementing interpolation, we observe a very slight increase in the predicting accuracy.
- **Error analysis on validation set**:
    - Trigram without other unknown handling methods still suffers from the out of dictionary issue for person name. For instance, the sentence **[‘10.’ , ‘Dmitri’, ‘Dashinski’, ‘Belarus’, ‘)’, ‘190.70’]** was predicted as **['O', 'O', 'O', 'B-LOC', 'O', 'O']**. The trigram missed the person name, and falsely classified **‘Belarus’** as a location. However, **‘Belarus’** is a country in Europe, and we don’t know why it is labeled **‘O’** in this sentence.
    - We experimented another way of handling unknown words: replace words with its POS tag when it appears at most once in the training set. It improved the prediction accuracy on the validation set, but still made some error. For example, the same sentence **‘10.’ , ‘Dmitri’, ‘Dashinski’, ‘Belarus’, ‘)’, ‘190.70’]** was predicted as **['O', 'B-LOC', 'O', 'B-LOC', 'O', 'O']**. **‘Dmitri’** and **‘Dashinski’** both have POS ‘NNP**’,** and it’s hard to tell why it recognized the first word instead of the second. Maybe some other ways could be experimented. For example, making subcategories for low frequency words, such as **‘FourDigitNumber**’, **‘AllCapitalWord’** etc.
- **Best Kaggle score**:



28	▾2	PulkitKashyap	0.79219	10	41m
29	▴53	(͡° ͜ʖ ͡°)つ──☆*:・ v2.0	0.78300	8	1h
30	▾23	eye_ball	0.78164	10	27m
31	▴3	Real Fake Doors	0.77887	14	19h

Your Best Entry ↑

Your submission scored 0.78164, which is an improvement of your previous score of 0.78134. Great job!

3. Individual Member Contribution

- Ruoyang Sun
  - Compute HMM backpointer matrix and score matrix
  - (Baseline) Output / Input function, preprocessing.
  - Implement Trigram viterbi as extension.
  - Report write-up
- Mengyao Lyu
  - Preprocessing (smoothing, n-gram training)
  - Create n-gram class
  - (Baseline) Prediction function.
  - Trigram interpolation
  - Unknown word handling
  - Report write-up
- Qianhao Yu
  - Use HMM backpointer to find optimal tagging.
  - Trigram function.
  - (Baseline) Prediction function.
  - Report write-up

Reference

Jurafsky, Dan, and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.. Upper Saddle River, N.J: Prentice Hall, 2000. 122-161. Print.

Collins, Michael. *Tagging with Hidden Markov Models*. www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/hmms.pdf.