0 Introduction

GitHub: https://github.com/nitaifingerhut/memm-pos-tagger

In this work, we have implemented a pos tagger using the **Maximum Entropy Markov Model (MEMM)**. MEMM is a **discriminative model**, i.e; for a sentence $x_1, \cdots, x_n$ paired with a tag sequence $y_1, \cdots, y_n$, it aims to maximize the conditional probability:

$$\mathbb{P}\left(Y_1 = y_1, \cdots, Y_n = y_n | X_1 = x_1, \cdots, X_n = x_n\right)$$

This model differs from **generative models**, which are trained to maximize the joint distribution

$$\mathbb{P}\left(X_1 = x_1, \cdots, X_n = x_n, Y_1 = y_1, \cdots, Y_n = y_n\right)$$

As implied from its name, **MEMM** relies on a hidden Markov Model assumption, where the prediction of the next tag is dependent on the words in the sentence, but only on the latest s paired tags

$$\mathbb{P}\left(Y_{n+1} = y_{n+1} | X_1 = x_1, \cdots, X_n = x_n, Y_1 = y_1, Y_2 = y_2, \cdots, Y_n = y_n\right) = \\ \mathbb{P}\left(Y_{n+1} = y_{n+1} | X_1 = x_1, \cdots, X_n = x_n, Y_{n-s} = y_{n-s}, \cdots, Y_n = y_n\right)$$

In our implementation we chose $s = 2$, which is known as the **trigram model**.
Let $h_i = \left(x_1, \cdots, x_n, y_{i-2}, y_{i-1}, y_i, i\right)$ be a history tuple for the i^{th} pair in $\Big($ sentence, tag sequence $\Big)$. When training, we define a set of d features, and extract the features vector for each history tuple in the sentence. We assume a weights vector $v \in \mathbb{R}^d$ (which are the model's parameters), such that

$$\mathbb{P}\left(Y_{n+1} = y_{n+1} | X_1 = x_1, \cdots, X_n = x_n, Y_{n-1} = y_{n-1}, \cdots, Y_n = y_n\right) = \frac{e^{v^T f(h_i, y_i)}}{\sum_{y' \in \mathcal{K}} e^{v^T f(h_i, y')}}$$

where $\mathcal{K}$ is the set of all possible tags.
While training, we seek to maximize the conditional probability over all sentences in the corpus. Given a specific sentence, the likelihood of conditional probability is

$$\mathbb{P}\left(Y_1 = y_1, \cdots, Y_n = y_n | X_1 = x_1, \cdots, X_n = x_n\right) = \\ \prod_{i=1}^{n} \mathbb{P}\left(Y_i = y_i | X_1 = x_1, \cdots, X_n = x_n, Y_{i-2} = y_{i-2}, \cdots, Y_{i-1} = y_{i-1}\right) = \prod_{i=1}^{n} \frac{e^{v^T f(h_i, y_i)}}{\sum_{y' \in \mathcal{K}} e^{v^T f(h_i, y')}}$$

Due to the monotony of the $log(\cdot)$ function, we can instead maximize the log-likelihood, which leads to (over all sentences in the corpus)

$$\mathcal{L} = \sum_{i=1}^{n} e^{v^T f(h_i, y_i)} - \sum_{i=1}^{n} \sum_{y' \in \mathcal{K}} e^{v^T f(h_i, y')}$$

In order to penalize high weights, we add a regularization to the weights vector, which follows that

$$\mathcal{L}_\lambda = \sum_{i=1}^{n} e^{v^T f(h_i, y_i)} - \sum_{i=1}^{n} \sum_{y' \in \mathcal{K}} e^{v^T f(h_i, y')} - \frac{\lambda}{2} v^T v$$

Maximizing the above is via looking for the optimal weights vectors $v_t^* = \text{argmax}_v \mathcal{L}_\lambda$ for all sentences in the training corpus.

When provided with a new sentences, we use the learnt weights vector and the **Viterbi algorithm** (combined with a beam search) to infer the predictions for it.

1 Train

1.1 Accuracy:

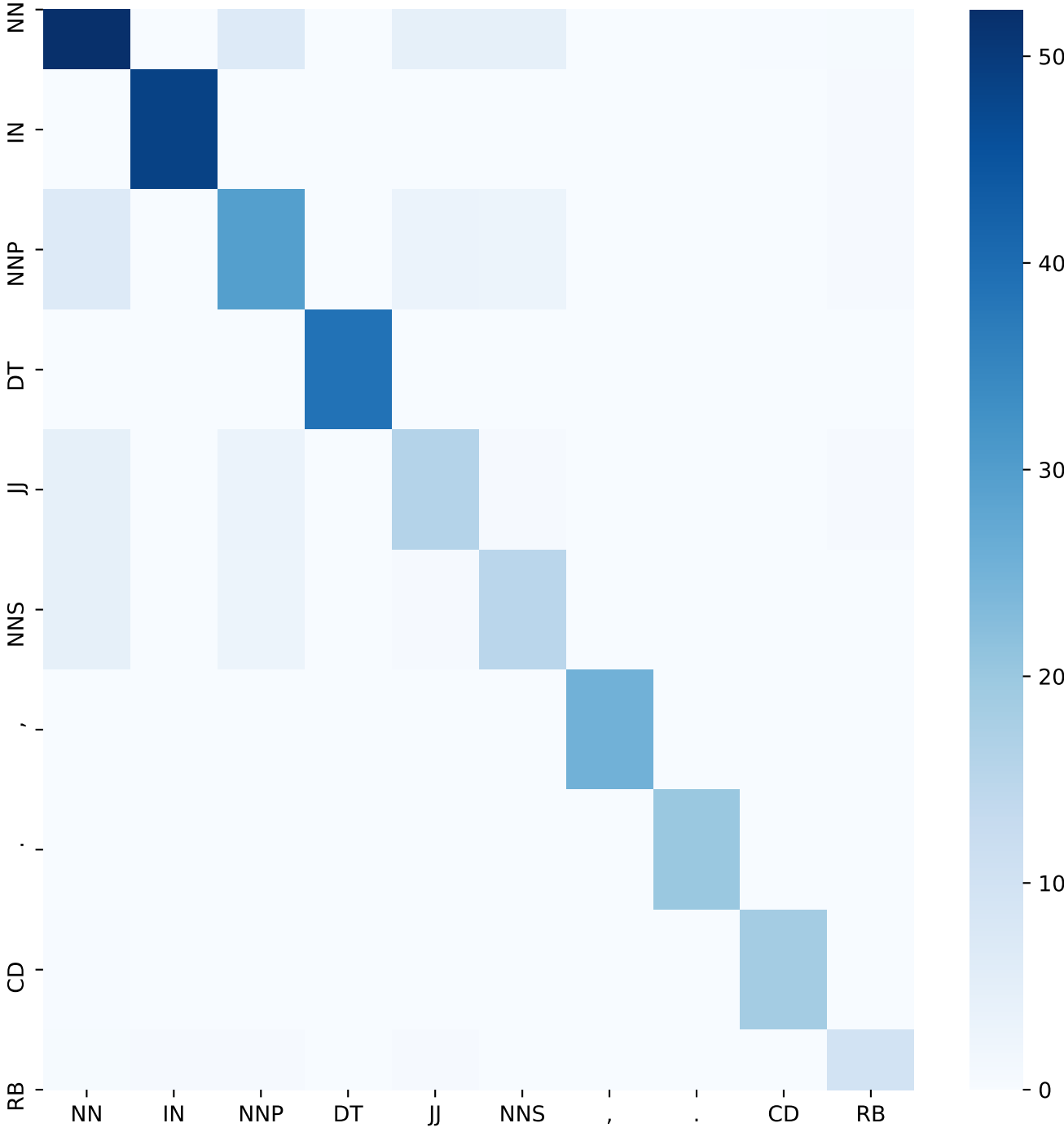
Model 1 81.2%

1.2 Features

Model 1 pairs: 1000, unigrams: 50, bigrams: 1000, trigrams: 1500, prefixes: 250, suffixes: 250, prev-w-curr-t: 250, next-w-curr-t: 250, index-tag: 250, index-word: 250, capital-tag: 250.

Model 2 pairs: 100, unigrams: 50, bigrams: 100, trigrams: 250, prefixes: 25, suffixes: 25, prev-w-curr-t: 25, next-w-curr-t: 25, index-tag: 25, index-word: 25, capital-tag: 25.

1.3 Confusion Matrix



Model 1

Confusion analysis:

to address the confusion between NN and NNP. we note that the first latter in words with NNP tag are always capital, we could add post processing to change each NNP tag without firs capital letter to NN and vice versa.

1.4 run times

All the process ran on Ubuntu HPC server

1.4.1 Model 1

Training process: around one hour
Prediction process: around 30 minutes

1.4.2 Model 2

Training process:around 10 minutes
Prediction process: around 5 minutes

1.5 additional information

- For the purpose of improving run times we decided to use only the most common features from each type as follows described above.
  The main motivation to do so is that if the training set represent the real distribution of the sentences than using very uncommon features may cause over fitting and the use of computational resources. furthermore because the features are uncommon the new information they will bring to the model are very limited.
- New features:
  - As we saw our model Had a hard time predicting correctly the start of the sentences so we used two new features sets, First features set checks the index of the word as well as the tag. Second features set checks the index of the word as well as the word itself.
  - We add a feature set that checks if the word starts with capital letter for each possible tag.
  - We add some costume features like
    - Is the word starts with capital letter?
    - All the letters in the word are capital?
    - The word is digit?
    - Word index is $i \in \{1, 2, 3\}$

2 Inference

2.1 Viterbi algorithm

We implemented the algorithm using NumPy arrays to improve prediction run time as well as using beam search of 2. As we saw in practice there is little difference between using beam search of 2 or more in

prediction accuracy.

Post-processing: To improve accuracy we analyse the prediction and we notice some specific words and tags always pair together such as the and DT or ”,” and ”,” so after we run our prediction on the data we run a process to update this tags if necessary. As we saw this step improves accuracy by 3%-5%.

3 Competition

3.1 General

We used the same algorithm described in previous section for the competition task. We expect to see an accuracy rate of 78%-80%. As we saw in our test the train and test data came from similar distribution as can see by training on both data sets together (doesn't have significant impact on accuracy), we are expecting that the competition data will come from a different source. Moreover training on 1000 sentences is not enough to learn the real word distribution of tags. On the other hand we expect only small decrees in performance because we used a fairly large features set.

3.2 Model 2

In this model we have a training set of only 250 sentences, to avoid over fitting we cant use the same large feature vector, so we decided to use much smaller vector to allow some generalization on the expanse of possible accuracy.

4 Division of labor

Nitai:

- Process the data
- Process of extracting features from the data
- Process of creating a feature vector from a word and a history
- Writhing the Training algorithm
- Maintaining the general structure of the project
- Thinking and implementing of new features
- Sending and analyse test runs
- Writhing the final report

Daniel:

- Writhing the prediction algorithm
- Thinking and implementing of new features
- Sending and analyse test runs
- Writhing the final report