CS4248 Assignment 2
Parts-of-Speech Tagger

Heng Low Wee (U096901R)

1 Introduction

In this assignment, we build a Parts-of-Speech (POS) tagger, using training data from part of the Penn Treebank. Specifically, we implement a POS bigram tagger based on a hidden Markov model (HMM), and the optimal sequence of the most probable tags is computed using the Viterbi algorithm.

In this report, we discuss our approach on implementing this POS tagger in Java. We discuss the various parameters captured as we trained our model required to make the predictions, how we performed smoothing and how we handled unseen words by the model.

2 Methodology

2.1 Training Dataset

The training dataset we are using, `sents.train`, is training data from part of the Penn Treebank. Each word in each line of training set is annotated with a POS tag from the Penn Treebank. Table 1 summarises some information about the training set.

Item	Statistics
Vocab Size, $\|V\|$	44366
No. of annotated sentences	39832
No. of unique POS Tags used	45

Table 1: Information about sents.train

2.2 Building The Model

In order to implement our POS bigram tagger, we first identified the key parameters needed for a HMM to function. These parameters are also used in the Viterbi algorithm. These parameters are:

1. V : The set of all words in training set
2. S : The set of POS tags in training set
3. $P(s_i|s_{i-1})$: The transition probability from a POS tag to another e.g. $P(s_2|s_1),\ s_i \in S$.
4. $P(w|s)$: The probability of having a word, w, given a POS tag, s.

Once we identified the parameters needed for the HMM, we proceeded to extract information from our training set. Table 2 lists the information we extracted from `sents.train`.

Item	Description
V	The set of all words in training set
S	The set of all POS tags in training set
M	A POS transition matrix that captures the count of transitions i.e. $M[i,j] = C(s_i \rightarrow s_j)$
D	A hash that captures of the distribution of words for all tags

Table 2: Information extracted from sents.train

It is important to note that on top of the 45 POS Penn Treebank tags, we added 2 additional special tags, `<s>` and `</s>` to indicate a sentence's boundary. This 2 special tags are added to S during training.

With M, we proceed to generate M', a matrix that captures the transition probability instead of the raw count. For probabilities for transitions that have zero raw count, we performed smoothing, which is further discussed in the following section.

2.3 Smoothing

The smoothing method we employed was the Witten-Bell Smoothing for bigrams. In our tagger, there are two locations that required smoothing.

First, mentioned earlier, smoothing was performed when we generated M', a matrix of POS transition probabilities. We perform smoothing for POS transitions using Equation (1).

$$P(s_i|s_{i-1}) = \begin{cases} \dfrac{C(s_{i-1}s_i)}{C(s_{i-1}) + T(s_{i-1})} & \text{if } C(s_{i-1}s_i) > 0 \\ \dfrac{T(s_{i-1})}{Z(s_{i-1}) \cdot (C(s_{i-1}) + T(s_{i-1}))} & \text{if } C(s_{i-1}s_i) = 0 \end{cases} \tag{1}$$

where $C(s_{i-1}s_i)$ is the number of transition from s_{i-1} to s_i, $C(s_i)$ is the total number of times the tag s_i was seen, $T(s_i)$ is number of seen and unique transitions beginning with s_i, $Z(s_i)$ is number of unseen and unique transitions beginning with s_i.

Second, we also needed to perform smoothing when we compute for $P(w|s)$: probability of a word w given a tag s. This is also how we handle unknown (unseen by the model) words in test sentences. In other words, all unknown words are treated equally, with respect to the given tag, in our smoothing implementation.

$$P(w|s) = \begin{cases} \dfrac{C(s,w)}{C(s) + T(s)} & \text{if } C(s,w) > 0 \\ \dfrac{T(s)}{Z(s) \cdot (C(s) + T(s))} & \text{if } C(s,w) = 0 \end{cases} \tag{2}$$

where $C(s,w)$ is the number of times w was tagged s, $C(s)$ is the total count of all seen words given tag s, $T(s)$ is the count of unique words seen given tag s, $Z(s)$ is the count of unique words unseen given tag s. So $T(s) + Z(s) = V$.

2.4 Implementation

Once we built our model, we were ready to implement the tagger using the Viterbi algorithm. In our implementation, there are 2 main differences from the algorithm mentioned in the lecture slides worth taking note.

First, instead of creating of matrix of size $N+2$ by T (N = no. of POS Tags, T = no. of tokens in test sentence), we created one of size N by T instead. So in the Init and Termination stage, the transition probabilities were computed using the special tags we added: `<s>` and `</s>`.

Second, which is also a solution to one technical difficulty we encountered, is that we used $\log n$ instead of n, where n is the probability values from $P(s_i|s_{i-1})$ and $P(w|s)$. The reason we applied logarithms to the probability values is because these values were very small (e.g. 10^{-20}), and the product of these values is a value too small, and was treated as zero by the machine.

3 Evaluation

We validate our POS tagger by performing 10-fold cross validation on our training data. We divide the training data into 10 parts, and for each fold we train on 9 parts and test on 1 part, alternating that testing part ten times to obtain ten sets of results.

For each fold, we evaluate our POS tagger by computing the Precision (P), Recall (R) and F-score (F) values. Here's how we computed the values. First, we compute the $P/R/F$ values for *each* POS tag. In other words for each fold, we have 45 sets of $P/R/F$ values, one for each tag. The $P/R/F$ values for that fold is computed by taking the average of the 45 sets of values. See Table 4 for the results.

n-th fold	Precision P	Recall R	F-Score F
1	.862	.856	.882
2	.861	.915	.891
3	.879	.884	.898
4	.895	.916	.910
5	.902	.885	.883
6	.891	.906	.905
7	.866	.896	.875
8	.863	.885	.887
9	.864	.910	.893
10	.891	.911	.894
Average	.877	.896	.892

Table 3: $P/R/F$ values from 10-fold cross validation

4 Discussion

When tested on `sents.devt` we found there are a few tags that have significantly lower $P/R/F$ values. They are:

Tag	Precision P	Recall R	F-Score F
NNPS	.797	.680	.734
RBR	.429	.706	.533

Table 4: $P/R/F$ values for selected tags

Upon comparing the annotation with the predicted tags, we came up with the following analysis:

1. NNPS: The predictions tend to be NNP instead. The model was unable to differentiate singular versus plural proper nouns. This could be due the fact all unknown words are treated equally in our smoothing implementations. One possible solution is to assigned a higher probability for proper nouns ending with an "s".

2. RBR: Confused with JJR. Since both tags are comparative, one possible solution could be to perform word stemming, which could better allow us determine whether the word stem is an adjective or an adverb.

`https://github.com/lwheng/cs4248`