Steven Kester Yuwono

Part-of-speech Tagger

1. Introduction

A Stochastic (HMM)[i] POS bigram tagger was developed in C++ using Penn Treebank tag set. Viterbi algorithm which runs in $O(T.N^2)$[ii] was implemented to find the optimal sequence of the most probable tags.

In this report, the approach in implementing the POS tagger will be discussed. It covers various methods to (1) capture relevant information from the training data, (2) calculate relevant probabilities, (3) perform smoothing on the model, and (4) handle unknown (unseen) words.

The main report consists of only 4 pages. The appendix includes more detailed analysis of the POS tagger performance using various implemented smoothing methods.

2. Implementation Details

2.1. Training Data

The training data used by the program (`sents.train`) consists of sentences which contains words and the corresponding tags (from Penn Treebank tag set).

Please refer to the training data summary below:

Category	**Count**
Number of word types (unique words)	44389
Number of “bag-of-words” (non-unique words)	950028
Number of sentences	39832
Number of unique POS Tags (Penn Treebank)	45

Table 1 – Summary of training data (`sents.train`)

2.2. Attributes

Before reading and processing the training data, it is necessary to identify relevant attributes which are needed to perform Stochastic POS Tagging.

The required attributes are as follows:

1. W : The set of words in the training set (words are case-sensitive).
2. T : The set of POS tags in the training set.
3. $P(t_i|t_{i-1})$: The transition probability to have a POS tag t_i given the previous POS tag t_{i-1}.
4. $P(w|t)$: The probability to have a word w, given a POS tag t.

2.3. Information Extraction

To obtain the required attributes, the following information is extracted from the training data:

[i] HMM = Hidden Markov Model
[ii] T = number of words ; N = number of POS tags.

Item	Description
POS Tag Map	A map to translate a POS tag to its assigned integer index and vice versa
Word Map	A map to translate a word to its assigned integer index and vice versa
Tag Count	A table to store the number of occurrences of every POS tag
Word Count	A table to store the number of occurrences of every word
Bigram Tag Matrix	A matrix to store the number of occurrences of a POS tag preceding another POS tag (two consecutive POS tags)
Word Tag Matrix	A matrix to store the number of occurrences of a word being assigned with a specific tag

Table 2 – Information extracted from training data

Stochastic POS tagging also requires a special start state and end (final) state to indicate the boundary of a sentence. Hence two special tags, <s> and </s>, are added on top of the 45 Penn Treebank POS tags during the extraction process.

2.4. Information Processing and Smoothing

The raw counts obtained from the training data, are then processed, computed, and stored in the respective probability matrices (bigram tag and word tag). To calculate the probability, smoothing is required to handle zero raw count. Three smoothing methods namely add-one, Witten-Bell, and Kneser-Ney, were implemented. After running[iii] each smoothing methods against the provided data set and analysing them, Kneser-Ney smoothing emerged superior compared to the other two.

Therefore, Kneser-Ney smoothing (with backoff) was adopted for the final POS tagger program. There are two sections that require smoothing.

First, smoothing is required to compute the transition probability of bigram tags which will then be stored in a Transition Probability Matrix. The smoothing follows the equation below:

$$P_{KN}(t_i|t_{i-1}) = \begin{cases} \dfrac{C(t_{i-1}t_i) - D}{C(t_{i-1})} & if\ C(t_{i-1}t_i) > 0 \\ \alpha(t_{i-1}).\dfrac{|\{t_{i-1} : C(t_{i-1}t_i) > 0\}|}{\sum_t |\{t_{i-1} : C(t_{i-1}t) > 0\}|} & if\ C(t_{i-1}t_i) = 0 \end{cases} \qquad (1)$$

$\alpha(t_{i-1})$ is calculated based on backoff distribution and discounted probability ($\tilde{P}$) of bigram and the continuation probability. It follows the equations below:

$$\tilde{P}(t_i|t_{i-1}) = \frac{C(t_{i-1}t_i) - D}{C(t_{i-1})} \qquad (2) \qquad\qquad P_{CONTINUATION}(t_i) = \frac{|\{t_{i-1} : C(t_{i-1}t_i) > 0\}|}{\sum_t |\{t_{i-1} : C(t_{i-1}t) > 0\}|} \qquad (3)$$

$$\alpha(t_{i-1}) = \frac{1 - \sum_{t_i : C(t_{i-1}t_i) > 0} \tilde{P}(t_i|t_{i-1})}{1 - \sum_{t_i : C(t_{i-1}t_i) > 0} P_{CONTINUATION}(t_i)} \qquad (4)$$

D is the absolute discount value which is set to be 0.085. The value 0.085 is obtained by running[iv] the program with different discount value, and 0.085 yields the best result.

[iii] Detailed analysis of the 3 smoothing method implemented is attached in the appendix section, on page 6 and 7
[iv] Detailed analysis of the absolute discount value (D) is attached in the appendix section, on page 5

Second, smoothing is also required to compute $P(w|t)$, the probability of a having a word w, given a tag t, because not every word will appear with all POS tags. There is absolutely some zero count of $C(t_i w_i)$. The following equation is used to perform smoothing (identical to equation (1)):

$$P_{KN}(w_i|t_i) = \begin{cases} \dfrac{C(t_i w_i) - D}{C(t_i)} & if\ C(t_i w_i) > 0 \\ \alpha(t_i).\dfrac{|\{t_i : C(t_i w_i) > 0\}|}{\sum_w |\{t_i : C(t_i w) > 0\}|} & if\ C(t_i w_i) = 0 \end{cases} \qquad (5)$$

$\alpha(t_i)$ is also handled in the same manner, with different attributes.

$$\tilde{P}(w_i|t_i) = \frac{C(t_i w_i) - D}{C(t_i)} \qquad (6) \qquad\qquad P_{CONTINUATION}(w_i) = \frac{|\{t_i : C(t_i w_i) > 0\}|}{\sum_w |\{t_i : C(t_i w) > 0\}|} \qquad (7)$$

$$\alpha(t_i) = \frac{1 - \sum_{w_i : C(t_i w_i) > 0} \tilde{P}(w_i|t_i)}{1 - \sum_{w_i : C(t_i w_i) > 0} P_{CONTINUATION}(w_i)} \qquad (8)$$

The absolute discount value ***D*** is again set to its optimum value, 0.085.

The smoothing equations above are also used to handle unknown (unseen) words in the test data. One word "<UNK>" is added into the word set with one occurrence, without any tags, and it represents all unknown words encountered. In other words, all unknown words are treated in equal manner.

However, after some observation and analysis (considering the fact that the words are treated as case-sensitive), converting an unknown word to lowercase letters, or uppercase first-letter and checking against the dictionary may result in the discovery of the word, which significantly increases the chance of the POS tagger to assign a correct tag, compared to always treating it as an unknown ("<UNK>") word. Hence, this method is implemented to improve the POS tagger performance.

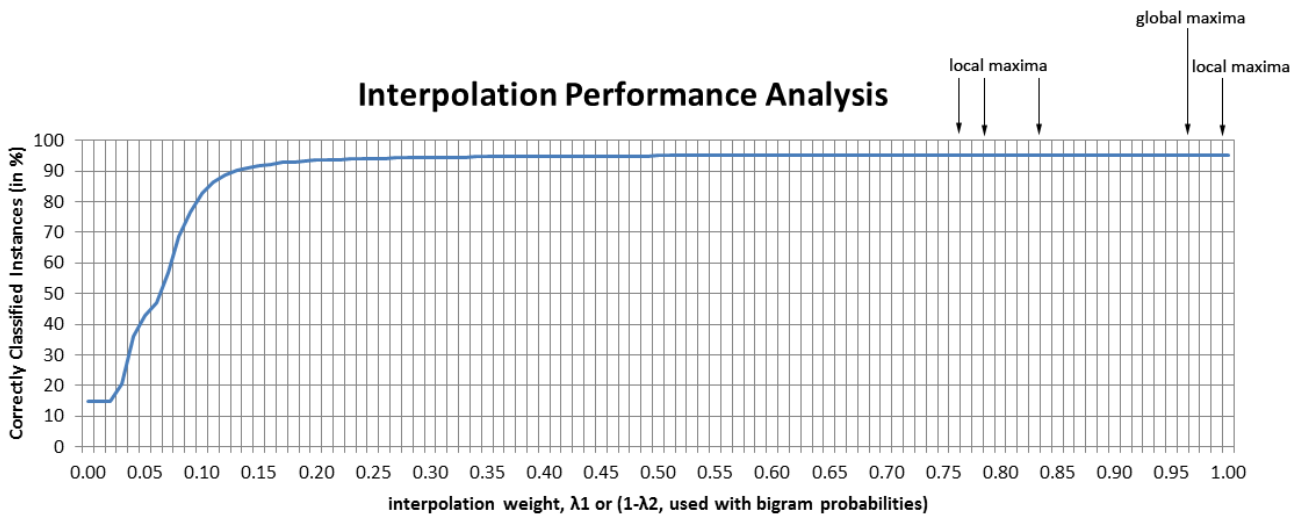
2.5. Interpolation

To improve the performance of the POS tagger further, interpolation was adopted and implemented. The program will automatically detect and find the best value for λ_1 and λ_2 and they are used with the equations below:

$$\hat{P}(t_i|t_{i-1}) = \lambda_1 P_{KN}(t_i|t_{i-1}) + \lambda_2 P(t_i) \quad where \sum_i \lambda_i = 1 \qquad (9)$$

$$\hat{P}(w_i|t_i) = \lambda_1 P_{KN}(w_i|t_i) + \lambda_2 P(w_i) \quad where \sum_i \lambda_i = 1 \qquad (10)$$

The POS Tagger now uses new probability ($\hat{P}$) instead of the Kneser-Ney Smoothed (P_{KN}) probability. Optimal interpolation weights λ_1 and λ_2 are estimated and computed automatically by the program. It will start with $\lambda_1 = 1.00$ because the bigram probability has greater contribution compared to the unigram probability. Then it will run the POS tagger on the development file, compute the accuracy of the POS tagger and then repeat the process with 0.01 interval decrement. The program will terminate evaluating the weights if the performance of the tagger does not improve after ***m*** consecutive evaluation, where ***m*** is the momentum or threshold value. ***m*** is used to prevent the program to be trapped in local maxima. Hence it will continue evaluating ***m*** times to check whether the performance can be improved. After thorough evaluation and analysis, the optimum value of ***m*** was found to be 3.



Graph 1. Interpolation performance analysis using `sents.train` and `sents.devt`

After computing the probability with optimum interpolation weight, POS tagger will now use the new probability ($\hat{P}$) to evaluate sentences and assign POS tag to each word.

3. Validation

The POS tagger is validated by performing 10-fold cross-validation on the training data provided. A simple program is written to separate the training data into 10 distinct training data (includes 90% of the original training data) and 10 distinct test data (includes 10% of the original training data) respectively. The 10-fold training and test set are then used to evaluate the POS tagger.

The attributes used to benchmark the performance of the POS Tagger include: Recall/True-Positive Rate, False-Positive Rate, Precision, and F-Measure. For each fold, the weighted averages of the 4 attributes mentioned above are calculated to analyze the POS tagger's performance as a whole.

The results are summarized in the following tables:

n-th fold	Recall / TP-Rate	FP-Rate	Precision	F-Measure
1	0.94948	0.00366	0.95058	0.94981
2	0.95039	0.00361	0.95221	0.95103
3	0.95388	0.00333	0.95451	0.95402
4	0.95321	0.00348	0.95382	0.95327
5	0.95148	0.00352	0.95234	0.95175
6	0.95272	0.00332	0.95382	0.95315
7	0.94928	0.00353	0.95098	0.94987
8	0.95063	0.00341	0.95277	0.95136
9	0.95458	0.00321	0.95571	0.95495
10	0.95025	0.00351	0.95145	0.95056
Average	0.95159	0.00346	0.95282	0.95198

Table 3 – Summary of 10-fold cross-validation on `sents.train`

	Recall / TP-Rate	FP-Rate	Precision	F-Measure
Development set	0.95698	0.00307	0.95871	0.95740

Table 4 – Summary of validation on `sents.devt`

In conclusion, the accuracy of the POS tagger is **95.159%**, and **95.698%** on the training set and development set respectively.

Appendix (Optional)

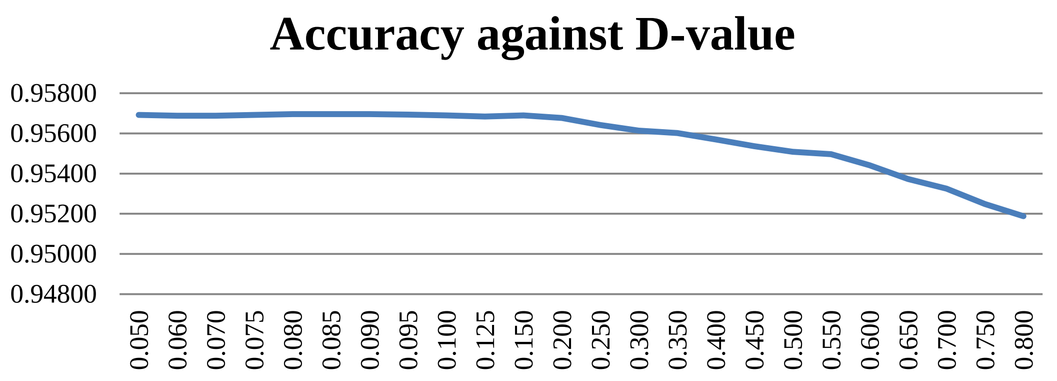
Please visit https://github.com/yulonglong/Stochastic-POS-Tagger for more details

1. Absolute Discount Value (D) analysis (in Kneser-Ney Smoothing)

From Table 5, it was found that the optimum value for D in this case is **0.085** after running the program with many discount values.

Discount value	TP-Rate/Recall	FP-Rate	Precision	F-Measure
0.050	0.95692	0.00307	0.95868	0.95735
0.060	0.95688	0.00307	0.95864	0.95731
0.070	0.95688	0.00307	0.95864	0.95731
0.075	0.95692	0.00307	0.95868	0.95735
0.080	0.95696	0.00306	0.95872	0.95739
0.085	0.95696	0.00306	0.95872	0.95739
0.090	0.95696	0.00306	0.95872	0.95739
0.095	0.95694	0.00306	0.95871	0.95737
0.100	0.95690	0.00306	0.95868	0.95733
0.125	0.95684	0.00306	0.95861	0.95727
0.150	0.95690	0.00306	0.95866	0.95732
0.200	0.95677	0.00307	0.95857	0.95721
0.250	0.95642	0.00310	0.95826	0.95686
0.300	0.95614	0.00313	0.95799	0.95659
0.350	0.95602	0.00315	0.95789	0.95647
0.400	0.95570	0.00315	0.95760	0.95616
0.450	0.95537	0.00317	0.95735	0.95586
0.500	0.95509	0.00319	0.95712	0.95560
0.550	0.95497	0.00321	0.95701	0.95547
0.600	0.95442	0.00327	0.95648	0.95493
0.650	0.95373	0.00333	0.95591	0.95428
0.700	0.95325	0.00337	0.95546	0.95381
0.750	0.95249	0.00343	0.95479	0.95308
0.800	0.95188	0.0035	0.95422	0.95248

Table 5 – Summary of D against the POS tagger's performance (using `sents.devt`)



Graph 2 – Accuracy of the POS tagger against D-value

2. Various Smoothing Analysis

2.1. Add-one Smoothing

Add-one smoothing was performed with 10-fold cross-validation as shown in Table 6.

n-th fold	TP-Rate/Recall	FP-Rate	Precision	F-Measure
1	0.91043	0.00819	0.90580	0.90445
2	0.91158	0.00821	0.90970	0.96617
3	0.91321	0.00797	0.91037	0.90732
4	0.91377	0.00790	0.90882	0.90768
5	0.91018	0.00822	0.90554	0.90399
6	0.91322	0.00780	0.90916	0.90767
7	0.91210	0.00802	0.90923	0.90682
8	0.91092	0.00811	0.90856	0.90546
9	0.91841	0.00760	0.91470	0.91333
10	0.91051	0.00804	0.90613	0.90422
Average	**0.91243**	**0.00801**	**0.90880**	**0.91271**

Table 6 – Summary of 10-fold cross-validation using add-one smoothing

2.2. Witten-Bell Smoothing

Witten-Bell smoothing was performed with 10-fold cross-validation as shown in Table 7.

n-th fold	TP-Rate/Recall	FP-Rate	Precision	F-Measure
1	0.94756	0.00390	0.94876	0.94787
2	0.94802	0.00391	0.94986	0.94861
3	0.95158	0.00361	0.95215	0.95162
4	0.95116	0.00372	0.95196	0.95122
5	0.94836	0.00386	0.94928	0.94856
6	0.95053	0.00363	0.95149	0.95084
7	0.94715	0.00379	0.94885	0.94769
8	0.94715	0.00377	0.94938	0.94787
9	0.95240	0.00349	0.95360	0.95277
10	0.94809	0.00378	0.94964	0.94845
Average	**0.94920**	**0.00375**	**0.95050**	**0.94955**

Table 7 – Summary of 10-fold cross-validation using Witten-Bell smoothing

2.3. Kneser-Ney Smoothing

Kneser-Ney smoothing was performed with 10-fold cross-validation. The absolute discount value was set to its optimum, 0.085, as shown in Table 8.

n-th fold	TP-Rate/Recall	FP-Rate	Precision	F-Measure
1	0.94948	0.00366	0.95058	0.94981
2	0.95039	0.00361	0.95221	0.95103
3	0.95388	0.00333	0.95451	0.95402
4	0.95321	0.00348	0.95382	0.95327

5	0.95148	0.00352	0.95234	0.95175
6	0.95272	0.00332	0.95382	0.95315
7	0.94928	0.00353	0.95098	0.94987
8	0.95063	0.00341	0.95277	0.95136
9	0.95458	0.00321	0.95571	0.95495
10	0.95011	0.00350	0.95139	0.95047
Average	**0.95158**	**0.00346**	**0.95281**	**0.95197**

Table 8 – Summary of 10-fold cross-validation using Kneser-Ney smoothing (D = 0.085)

2.4. Kneser-Ney Smoothing with Interpolation

Kneser-Ney smoothing with interpolation was performed with 10-fold cross-validation. The absolute discount value was set to the same value, 0.085. The results are summarized in Table 9.

n-th fold	TP-Rate/Recall	FP-Rate	Precision	F-Measure	λ_1	λ_2
1	0.94948	0.00366	0.95058	0.94981	1.00	0.00
2	0.95039	0.00361	0.95221	0.95103	1.00	0.00
3	0.95388	0.00333	0.95451	0.95402	1.00	0.00
4	0.95321	0.00348	0.95382	0.95327	1.00	0.00
5	0.95148	0.00352	0.95234	0.95175	1.00	0.00
6	0.95272	0.00332	0.95382	0.95315	1.00	0.00
7	0.94928	0.00353	0.95098	0.94987	1.00	0.00
8	0.95063	0.00341	0.95277	0.95136	1.00	0.00
9	0.95458	0.00321	0.95571	0.95495	1.00	0.00
10	0.95025	0.00351	0.95145	0.95056	0.98	0.02
Average	**0.95159**	**0.00346**	**0.95282**	**0.95198**		

Table 9 – Summary of 10-fold cross-validation using Kneser-Ney smoothing (D = 0.085) with interpolation

3. Conclusion

After analyzing the three smoothing methods with 10-fold cross validation, Kneser-Ney smoothing yielded the best result compared to Witten-Bell and add-one smoothing (**95.158%** vs **94.920%** vs **91.243%** respectively). However, Witten-Bell smoothing's performance was comparable and scored quite close to Kneser-Ney smoothing. Interpolation also helped to improve the POS tagger's performance by one-step further despite of the small magnitude.

The POS tagger was run against the development file with all the smoothing methods mentioned above and the results are summarized in Table 10.

Smoothing	TP-Rate/Recall	FP-Rate	Precision	F-Measure	λ_1	λ_2
Add-one	0.91846	0.00727	0.91507	0.91356	N.A.	N.A.
Witten-Bell	0.95446	0.00333	0.95645	0.95493	N.A.	N.A.
Kneser-Ney	0.95696	0.00306	0.95872	0.95739	N.A.	N.A.
Kneser-Ney + interpolation	0.95698	0.00307	0.95871	0.95740	0.99	0.01

Table 10 – Performance comparison on the development file (`sents.devt`)