CS5740: Assignment 3
Github Repository (A3-13)

Snigdha Singhania ss4224 | Bowen Tan bt358 | Yuxin Zhang yz2726

1 Introduction (5pt)

In this assignment, we present 3 generative approaches to predict parts-of-speech (POS) tags given $\sim$ 1 million words of text from the Wall Street Journal. POS tags from the Penn Treebank project are used in these experiments.

Section 2 explains the data distribution and pre-processing steps employed in the experiments. 11,727 and 11,054 unknown words are identified in the development and test set, handling which is described in Section 3. Add-k and linear interpolation smoothing techniques are explained in Section 4. Section 5 covers all implementation details, including hyperparameters, while results from Greedy Search, Beam Search and the Viterbi algorithm are reported in Section 6. F1 scores obtained using a trigram model with the Viterbi decoder, and add-k ($k = 0.0005$) smoothing are **0.9594** and **0.96132** on the development and test set, respectively. Unknown word accuracy **0.8636** is obtained on the development set.

2 Data (5pt)

Table 1 shows data distribution across the training, development and test sets. The distribution of the training data across different POS tags are shown in Figure 1. The dataset is tokenized into documents using the **-DOCSTART-** token, then to words using newline. Every document comprises of multiple sentences. The average document length is 501 words. No case-folding is performed as casing is critical to identify POS tags.

Attribute	Size
Training Set (# Documents)	1387
Development Set (# Documents)	462
Test Set (# Documents)	463
Vocabulary – count(words) ≥ 2	19301
Avg. Document Length	501.14
Tags (or States)	45

Table 1: Dataset Distribution

The vocabulary of words is computed after eliminating rare-words (words that occur less than twice in the training data). 11,727 and 11,054 unknown words are identified in the development and test set. These rare and unknown words are converted into word and suffix classes, which is discussed in Section 3.

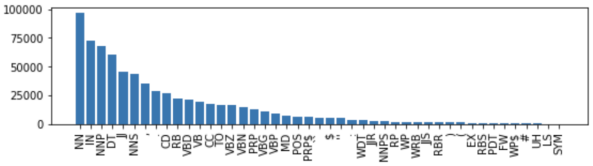


Figure 1: Tag Distribution in Training Data

3 Handling Unknown Words (15pt)

Word-tag pairs occurring only once in the training data are used to train the model for unknown words. 21,135 rare-words occur in the training dataset. As these pairs occur only once, they do not provide meaningful, reliable information. Hence, using them to create different morphological and suffix classes, we are able to learn useful unknown word behaviour and improve accuracy.

– **Morphological Inference** identifies small meaningful units in unknown words. 17 word classes, such as single letter, initials, digits and alphanumeric text with special characters (comma, hyphen), etc., are created to identify common patterns in rare and unknown words. All word classes are listed in the Appendix. The rare-words in the training data are substituted with corresponding word classes to generate emission probabilities, which can later be used by decoding techniques to predict the next POS tag just like any other word. Some examples are presented in Table 2. Unknown word accuracy of 0.832 is reported without morphological word classes.

– **Suffix-based Inference** uses the last n characters of unknown words for prediction. E.g., words ending in '-ing' such as 'alluring', 'consenting', 'filtering' are highly likely to be `VBG`, `NN` or `JJ`. For every word, we get all suffixes of length 2 to 6 (both inclusive). To remove noise, we retain

Word Class	Example	Likely Tags
DIGIT_COMMA	2,010	CD
INITIALS	I.B.M	NNP
NUM_IN_WORDS	four-page	JJ

Table 2: Word Class Examples and Probable Tags

suffixes that occur ≥ 10 times while training, and have a tag majority of at least 0.4. This information is used in collaboration with the word casing, and presence of hyphens in the word, to predict the POS tag. Unknown word accuracy drops to 0.6667 without suffix-based inference.

4 Smoothing (10pt)

We use the n-gram model for generating transition probabilities, to model the behaviour of POS tags in the Markov Chain. To handle unseen n-gram events in our development/test data and prevent assigning 0 probabilities to them, two smoothing techniques are used. While smoothing is that reduces the likelihood of seen events by a factor, it is useful in NLP problems when the set of possible outcomes in unbounded.

– **Add-*k* Smoothing** shifts a small probability mass to unseen events from seen events. Given a bigram event (y_{i-1}, y_i), and vocabulary $\mathcal{V}$, the smoothed probability is

$$P_{add-k}(y_i|y_{i-1}) = \frac{C(y_{i-1}, y_i) + k}{C(y_{i-1}) + k\mathcal{V}}$$

k is tuned using results on the development set. Empirically, best results are obtained for k values **0.5**, **5e-4**, **5e-6** in bigram, trigram and 4-gram models, respectively. As the order of n-gram increases, smaller values of k should be used as sparsity in the dataset also increases.

– **Linear Interpolation** uses a combination of different lower-order n-grams to account for unseen events. For a trigram model, this is

$$P_{linear}(y_i|y_{i-1}, y_{i-2}) = \lambda_1 P(y_i) + \lambda_2 P(y_i|y_{i-1}) + \lambda_3 P(y_i|y_{i-1}, y_{i-2})$$

λ constants for n-grams are calculated using [1] and values obtained are presented in Table 9.

5 Implementation Details (5pt)

A trellis is used to model the transition of POS tags given a sequence of words. The emission and transition probabilities are calculated using Maximum Likelihood Estimates (MLE). We optimise the computation speed of the trellis decoder by using a dictionary to map state transitions that contains only valid nodes. The run-time for the 3-gram, Viterbi algorithm is ~ 100 seconds.

For n-gram and decoder experiments, add-k smoothing with k – 0.5, 5e-4 and 5e-6 are used for 2-gram, 3-gram and 4-gram models, respectively.

6 Experiments and Results

Test Results (3pt) Table 3 highlights the best results obtained on the test data. These use a trigram model with add-k smoothing where $k = 0.0005$. F1 scores of **0.96132** and **0.96158** are obtained using Viterbi and Beam Search decoder, respectively. Unknown words were handled using the techniques described in Section 3.

System (Test Data)	F1 Score
3-gram, Beam Search (B=3)	0.96158
3-gram, Viterbi	0.96132

Table 3: Results on the Test Set

Smoothing (5pt) Results from the smoothing experiments are summarized in Table 4, which include overall and unknown word accuracies without smoothing, linear interpolation and add-k. Parameters such as λ and k are decided as discussed in Section 4. These results are reported on the Viterbi decoder. Results without smoothing are very close to those with add-k, as k used is very small. It is also possible that unseen tag sequences do not occur in the dev data. Add-k outperforms linear interpolation. It is likely that the model does not favour unigrams as they amplify the unequal distribution of tags in the training data (Figure 1).

System	Accuracy	
	Overall	Unknown
2-gram w/o Smoothing	0.9566	0.8504
2-gram, Linear	0.9039	0.6536
2-gram, Add-k (= 0.5)	**0.9568**	**0.8523**
3-gram w/o Smoothing	0.9560	0.8573
3-gram, Linear	0.9135	0.6907
3-gram, Add-k (= 5e-4)	**0.9592**	**0.8636**

Table 4: Results on Smoothing Experiments

2-gram vs. 3-gram vs. 4-gram (5pt) Table 5 summarizes results on different tag n-grams with several decoding techniques. As mentioned earlier, add-k smoothing outperforms linear interpolation, and is used in these experiments.

While the bi-gram and tri-gram model produce comparable overall accuracy, we see an increase of $\sim 1\%$ for unknown word accuracies with the trigram model. The 4-gram model performs poorly as compared to lower order n-grams. While higher order n-grams can capture more global context, the poor performance can be attributed to the increased sparsity in these models.

System	Accuracy	
	Overall	Unknown
Most-Common Baseline	0.8848	0.1393
2-gram, Greedy	0.9320	0.7561
2-gram, Beam (B=2)	0.9551	0.8367
2-gram, Beam (B=3)	0.9567	0.8511
2-gram w Viterbi	**0.9568**	**0.8523**
3-gram, Greedy	0.9483	0.8330
3-gram, Beam (B=2)	0.9590	0.8615
3-gram, Beam (B=3)	**0.9593**	**0.8636**
3-gram w Viterbi	0.9592	0.8625
4-gram, Greedy	0.9345	0.8156
4-gram, Beam (B=2)	**0.9463**	**0.8429**
4-gram, Beam (B=3)	0.9462	0.8424
4-gram w Viterbi	0.9444	0.8379

Table 5: Results on n-gram and Decoding Experiments (B: beam size)

Greedy vs. Viterbi vs. Beam (10pt) Overall and unknown token accuracies using different decoding techniques are presented in Table 5. We also report a simple baseline, which predicts the most-common POS tag associated with each word, and NN (most-popular) for unknowns.

Greedy search is essentially a beam search with beam size 1. As shown in Table 6, an increase in the beam size causes an increase in the ratio of optimal results, approaching 1 around B=3. The results from beam search approach those from Viterbi as the beam size increases. Overall, beam search, with beam 3, and Viterbi produce similar results. Beam search can, hence, effectively replace Viterbi when dealing with larger datasets to speed up computation. It is interesting to note that the results for the 3-gram and 4-gram model with beam search (B $= 3, 2$) outperform the Viterbi decoder. This is because the actual tags for 359 documents have a probability score lower than ones predicted by Viterbi.

	Greedy(B=1)	B=2	B=3	B=8
2-gram	0.81	0.983	0.996	1.0
3-gram	0.859	0.991	1.0	1.0
4-gram	0.855	0.987	0.998	1.0

Table 6: Ratio of Optimal Results (B: beam size)

7 Analysis

Error Analysis (7pt) Table 7 shows 4 tags with lowest accuracy on development data. These are also tags that occur less frequently, and their poor performance can be attributed to low transition probabilities assigned to them while training. Improving models with more context features may help improve the rare cases accuracy.

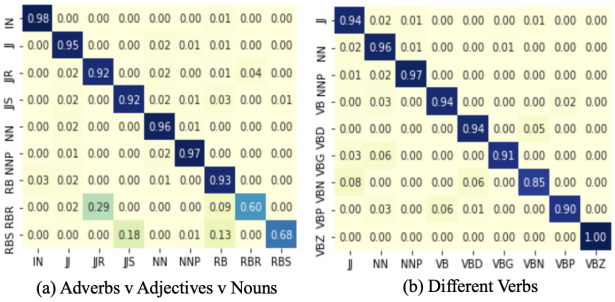
Tag	FW	UH	SYM	NNPS
Accuracy	0.09	0.167	0.4	0.506

Table 7: Tags with Lowest Accuracies

All unknown word classes composed of digits and special characters perform well (acc ≥ 0.99), except DIGIT_DASH, which reports an accuracy of 0.588. This group has 3 possible tags: CD, JJ, NNP, while others are limited to 1 or 2. As JJ and NNP are more frequent than CD, this category is often misclassified. It is observed to be a trend where unknown words (both suffixes and morphological word classes) are assigned POS tags which occurs more frequently in the training data.

Frequently occurring nouns in the vocabulary are misclassified when they have different casing. For example, 'Brothers', 'Securities', 'Communications' are confused between NNP, NNPS and NNS.

Confusion Matrix (5pt) Figure 2 shows normalized confusion matrices for some similar tags. (a) shows results between adverbs, adjectives and nouns. Comparative and superlative adverbs, RBR & RBS, have poor performance. They are mistaken for adverbs (RB) and their corresponding adjectives often. It should also be noted, from Figure 1, that RBR & RBS occur rarely as compared to other tags. While verbs perform well as seen in (b), present and past participles are sometimes confused with NN, and past tense and past-participle mistaken for each other (VBN & VBD).



(a) Adverbs v Adjectives v Nouns (b) Different Verbs

Figure 2: Confusion Matrix Subsets: Dev Data

8 Conclusion (3pt)

POS Tagging is considered one of the easier tasks in NLP as it is possible to obtain sufficiently high numbers with simple models. An accuracy of 0.9593 is obtained on development data using beam search (beam = 3), and unknown word accuracy of 0.8636. Misclassification is more likely for tags occurring less frequently in training, and are mistaken for similar more frequent tags, eg. NNPS mistaken for NN, NNP. It would be interesting to explore additional features and evaluate discriminative models for this task.

References

[1] T. Brants, "Tnt-a statistical part-of-speech tagger," *arXiv preprint cs/0003055*, 2000.

Appendix

Additional information regarding implementation which could not be included in the main report can be found here.

POS Tag distribution in the training dataset is presented in a more granular view in Figure 3.

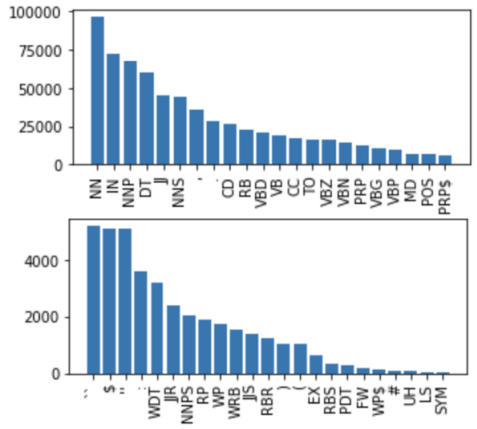


Figure 3: Tag Distribution in Training Data

Morphological Word Classes to handle Unknown Words 17 word classes are created to handle unknown words, before suffix checking.

DIGIT_ONLY	DIGIT_ALPHA
DIGIT_COMMA	DIGIT_ALPHA_DASH
DIGIT_DASH	DIGIT_ALPHA_DOT
DIGIT_COLON	DIGIT_ALPHA_QUOTE
DIGIT_DIV	DIGIT_ALPHA_DIV
INITIALS	ALPHA_DIV
SINGLE_CAPS	SINGLE_LOWER
ABBREVIATION	NUM_IN_WORDS
PERCENT	

Table 8: Word Classes for Unknown Words

Model	Lambdas
2-gram	$\lambda_1 = 0.962, \lambda_2 = 0.038$
3-gram	$\lambda_1 = 0.124, \lambda_2 = 0.525, \lambda_3 = 0.351$
4-gram	$\lambda_1 = 0.03, \lambda_2 = 0.16, \lambda_3 = 0.53, \lambda_4 = 0.28$

Table 9: Lambdas obtained for Linear Interpolation

Linear Interpolation Lambda values obtained for the tag n-gram by following the approach shown in [1] are:

As can be seen above, this method is not suitable for bigram models as it places almost all the emphasis on unigrams and a very small weight on the bigram.