CS4248Assignment 2

U096883L Shawn Tan

1 Introduction

This assignment requires us to build a Part-of-Speech (POS) tagger, using training data from part of the Penn Treebank. The method used in our approach has to employ a Hidden Markov Model (HMM). This entails learning from the training data a set of parameters required for the HMM. Table 1 shows the various sets of data we have to collect for this particular assignment.

In the following sections, we will explain in detail how individual aspects of the HMM was created, outlining some of the technical difficulties faced. We also experiment with two simple smoothing techniques, Laplace (add-one) and Witten-Bell smoothing. The two techniques will be evaluated according to their precision, recall and F1 measures.

2 Learning from `sents.train`

The `sents.train` dataset contains 39,832 lines, each word annotated with POS tags. In order to extract the relevant information we count the transitions.

In order to obtain $p(s_i \mid s_{i-1})$, we need to count the different number of times one tag is followed by another. For each line, we extract the POS for each token, leaving a list of POS

Name	Description
V	all unique words
S	all unique POS tags
$p(s_i \mid s_{i-1})$	transition probability from one POS tag to another
$p(w \mid s)$	probability of seeing a word given a POS tag

Table 1: Parameters for a POS tagger HMM

tags. We then prepend a '^' at the beginning, and a '$' character at the end. This ensures that the probabilities for a given POS starting a sentence are also taken into account.

Another preprocessing step performed was to change all instances of numeric tokens, and replacing all digits with \#. For example, '$25.50' will be converted to '$\#\#.\#\# '. This reduces the many different combinations of numbers down to a common token that would have a higher probability given the tag CD.

Going through the file, we maintain a dictionary of the following items:

Name	Description
$C(q_{t-1}, q_t)$	The number of times a POS tag occurs at time t after a POS tag occurs at time $t-1$
$C(q_t, q_{t-1})$	Count of transitions in reverse.
$C(q, w)$	Count of words given a POS tag.
$C(w, q)$	Count of POS tags given a word.

With this data, we can now perform different smoothing methods. The smoothing method is modular in our implementation, so as long as the counts are present, we can calculate the smoothed probabilities.

3 Evaluation

We performed evaluation for the POS tagger using two types of smoothing methods, Laplace smoothing using $B = 1$ (or add-one smoothing) and Witten-Bell smoothing. The tagger performs smoothing for both conditional probabilities, $p(w \mid q)$ and $p(q_t \mid q_{t-1})$.

3.1 Different smoothing methods

Using the **add-one smoothing**, we run 10-fold cross validation using `sents.out`. The results are shown in Table 2.

The results show that add-one smoothing causes a bad recall rate. This means that some of the words are tagged erroneously, but generally distributed throughout the other tags, such that they do not affect the other recall measures much. This suggests that the probability distributions given after smoothing spread the density out over the words and part of speech tags too much.

Using **Witten-Bell smoothing**, there are fewer occurrences of such problems. This means, compared to add-one smoothing, Witten-Bell smoothing gives a better estimate of unseen

Fold	Recall	Precision	$F1$
1	0.8645	0.7569	0.8618
2	0.8722	0.7702	0.8775
3	0.8726	0.7776	0.8656
4	0.8794	0.7599	0.8322
5	0.8924	0.7920	0.8626
6	0.8994	0.7603	0.8477
7	0.9110	0.7822	0.8523
8	0.8408	0.7705	0.8717
9	0.8689	0.7698	0.8462
10	0.8608	0.7811	0.8696
Average	0.8762	0.7721	0.85872

Table 2: 10-fold validation using add-one smoothing

Fold	Recall	Precision	$F1$
1	0.8541	0.8414	0.8719
2	0.8715	0.8978	0.8697
3	0.8595	0.9022	0.8723
4	0.8645	0.8719	0.8837
5	0.8890	0.8899	0.9024
6	0.8890	0.8766	0.8802
7	0.8924	0.8944	0.8877
8	0.8479	0.8773	0.8520
9	0.8468	0.8780	0.8736
10	0.8677	0.8929	0.8769
Average			

Table 3: 10-fold validation using Witten-Bell smoothing

instances of words given tags and tags given tags, resulting in a better overall prediction. The results are shown in Table 3.

3.2 Evaluating using `sents.devt`

With these results, we selected the Witten-Bell smoothing technique, and trained the tagger using all instances in `sents.train`, and tested it using the `sents.devt` file. This gave us a recall, precision and $F1$ measure of 0.8878, 0.9333 and 0.9015 respectively.

So, how can we improve upon this? Looking at the breakdown given the parts of speech and their individual performance, we can know where our tagger breaks down, and suggest improvements to our preprocessing and smoothing steps. The following are some of the parts of speech tags that perform poorly:

NNPS (Proper Noun, plural) Looking up the confusion matrix produced through the evaluation, a considerable amount of the mistakes made for NNPS were due to misclassifications of these tokens as NNS and NNP. One possible remedy for this is to be able to give a word that starts with a capital letter and ending with an 's' a higher probability of being an NNPS. This could be done as a special case within the smoothing step given an unseen word.

RBR (Adverb, comparative) Some words which should be tagged as RBR are tagged as JJR. This makes sense due to the large number of overlap in the type of words used.

LS (List item marker) These are hard to distinguish from usual mentions of numbers. This is especially since one of our preprocessing steps was to make all instances of digits the same. One possible way of dealing with this would be not to modify the first tokens of a sentence, since the LS tokens appear at the start pretty often.