Finding Uncertainty with Sequence Tagging

Introduction:

We view the task of labelling uncertain sentence and detecting weasel phrases as a sequence tagging task and use a HMM-viterbi sequence tagging model to achieve the purpose. We experiment with multiple configurations of HMM and try to augment the simple HMM model by implementing two extensions and report the results and justifications for the results on a cross validation set.

Data Analysis:

We spent some time with the training data to get a sense on how the data is distributed and how will a model trained on this data behave. Following are some of the observations:

*** Distribution for B *** Number of words that occur 1 times : 37 Number of words that occur 2 times : 22 Number of words that occur 3 times : 15 Number of words that occur 4 times : 14 Number of words that occur 5 times : 12 Number of words that occur 6 times : 7 Number of words that occur 7 times : 10 Number of words that occur 8 times : 7 Number of words that occur 9 times : 8	*** Distribution for I *** Number of words that occur 1 times : 169 Number of words that occur 2 times : 88 Number of words that occur 3 times : 91 Number of words that occur 4 times : 66 Number of words that occur 5 times : 59 Number of words that occur 6 times : 43 Number of words that occur 7 times : 56 Number of words that occur 8 times : 35 Number of words that occur 9 times : 21
*** Distribution for O *** Number of words that occur 1 times : 10489 Number of words that occur 2 times : 3083 Number of words that occur 3 times : 1513 Number of words that occur 4 times : 889 Number of words that occur 5 times : 621 Number of words that occur 6 times : 393 Number of words that occur 7 times : 362 Number of words that occur 8 times : 242 Number of words that occur 9 times : 183	*** Distribution for words with freq 1 *** Number of Bs 37 Number of Is 169 Number of Os 10489 *** Distribution for words with freq 2 *** Number of Bs 28 Number of Is 96 Number of Os 6072 *** Distribution for words with freq 3 *** Number of Bs 23 Number of Is 101 Number of Os 4427

The above data is generated based on CV train set (i.e. after removing the CV test files)

From the data above we can see that the number of Os are significantly more than the number of Bs and Is, and that there just a lot of words that occurred once in the corpus and were tagged with O. So it is clear that a model trained on this training data will be more inclined towards making a prediction of O.

Baselines:

We implemented two different baselines, which are described as follows:
Decisions common to both baselines:

i) We used 'BIO' tagging, for the baseline system it didn't seem to make much sense to differentiate between ending ('L') token and middle token ('I') of weasel phrase.
ii) All the words that weren't present in training data (i.e. unknown words) have been marked as 'O'. We wanted to keep baseline simple, and not add any special unknown word handling.

Baseline 0:

i) If a word has a 'B' or 'I' tag keep it in weasel dictionary.
ii) On the test data, if the word is present in weasel dictionary, then mark it as either 'B' or 'I' according to the following condition:
    a. If the previous word's tag was 'O' then mark as 'B' else 'I'.
iii) A sentence is predicted as uncertain if number of phrases with more than one weasel words is >2. (i.e. count a sequence of 'I's as 1 and don't count 'B')
    a. We came up with this threshold by doing multiple runs and tuning the threshold till the count made sense (and also checking the kaggle score for different threshold)

scores: Uncertain Phrase: 0.01044, Uncertain Sentence: 0.44177

Baseline 1:

i) This idea was inspired from the unigram model, we calculate P(tag|word) and assign the tag accordingly.
ii) So for each word we maintain it's 'B', 'I', 'O' counts and calculate the cumulative probability.
iii) When predicting we lookup the BIO probability for the word, generate a random number and pick the tag based on where the random number lies.
iv) A sentence is predicted as uncertain if phrase_count > 0, and the phrase_count is incremented by following rules:
    a. If 'B' increment phrase_count by one
    b. If sequence of 'I's just increment by one.
    c. If 'I' seen alone then increment phrase_count by one (this case happens as we are predicting using random number.)
v) We came up with the threshold of 0 in (iv) by trying different thresholds and tuning it till the sentence count made sense, and also checking the kaggle score. (kaggle score with other thresholds were lower)

scores: Uncertain Phrase:0.07990, Uncertain Sentence: 0.52799

Implementation Details:

To build HMM, all the code has been implemented from scratch, and no external libraries are used. Couple of uses of nltk.FreqDist() (constructs a dictionary from list) may be present, which was used to do data analysis at places.

Most of the code is already documented well with the design decisions, we explain the major components in the subsequent sections:

Section-wise Design Decisions and details :

Start Tag:

We use <phi> as a start tag, this is to be able to compute P(ti|ti-1) for the first tag.
To generate the transition table we insert <phi> after newline.

Sentence Tagging:
We tag one sentence at a time, i.e. the tags in the previous sentence do not affect the tags for current sentence.

Transition Table:
Transition table holds the tag transition counts. We decided to remove the column for <phi> and reduce the total count per row accordingly. The reasons are as follows:
1. As we are tagging one sentence at a time, so P(<phi>|t) is irrelevant, and that probability mass should be redistributed to others.
2. Only rows with <phi> are needed as P(t|phi) is the only possible combinations.

We don't apply smoothing on transition tables, as P(I|O) is bound to be 0 as I can never follow after O. Similarly, P(I|<phi>) should be 0. In our viterbi model, we would never want these paths to be possible. We check that all the relevant probabilities which should be non-zero are infact non-zero and hence don't apply any smoothing here.

Emission Probabilities:
We try 2 approaches for unknown word handling, good turing smoothing. All of these are described below.

Unknown word handling designs and results:

1. Unknown word handling based on the ratio of B, I, O:
   - While building the emission probability table, we added an unknown word column ('<unk>') for each of the B, I , O tags.
   - We calculated the ratio of count of one tag and count of all the tags and updated the count table corresponding to the tag. (for eg, count table of <B,'<unk>'> is updated with total number of tag B/ (total number of B+total number of I+total number of O))
   - Motivation behind this idea was to mimic the way how B, I, O tags appear in the train set and extend that numbers to any unknown words.
   - The results observed are listed below, results for basic HMM are also listed for comparison:

Unknown handling based on BIO ratio	Basic HMM
Sentence: Precision: 0.582822 Recall: 0.409483 Fscore: 0.481012 Word: Precision: 0.302857 Recall: 0.177852	Sentence: Precision: 0.572289 Recall: 0.409483 Fscore: 0.477387 Word: Precision: 0.270408 Recall: 0.177852

Fscore: 0.224101	Fscore: 0.214575

2.Unknown and Unseen word handling using POS tags:

While processing the test set, whenever a emission probability look up fails for a word given tag, then that word can be an unknown word in the train corpus or unseen <tag, word> combination in train corpus.

As one of the methods to handle the unseen/unknown words, we implemented an approach to use POS tags and do the probability look up for P(POS|tag) combination when a lookup for P(word|tag) fails.

The motivation here is that whenever an unknown/unseen word appears in test set, maximum information we know about that word is its POS tag and we try to use that information to assign it an appropriate probability value.

If probability look up for POS|tag also fails which means POS is also an unseen/unknown combination, probability value of 1e-10 is returned.

The results observed are listed below, results for basic HMM are also listed for comparison:

Unknown/Unseen handling with POS	Basic HMM
<u>Sentence:</u> Precision: 0.252999 Recall: 1.0 Fscore: 0.403829 <u>Word:</u> Precision: 0.007953 Recall: 0.073825 Fscore: 0.014360	<u>Sentence:</u> Precision: 0.572289 Recall: 0.409483 Fscore: 0.477387 <u>Word:</u> Precision: 0.270408 Recall: 0.177852 Fscore: 0.214575

<u>Smoothing:</u>

Good Turing smoothing to handle unseen bigrams in emission probability lookup

- We implemented good turing smoothing to handle probability value of 0 appearing in emission probability table for unseen words.
- Added a new column ‘<zero>’ in emission count table for each B, I, O tag. For every word in the train corpus, emission count table is checked for all three tags and whenever a <tag, word> combination lookup failed, we accounted them as bigrams appearing with the frequency of 0. Updated the <tag,’<zero>’> entry in emission count table with the value calculated by good turing for bigrams appearing 0 times.
- Good turing smoothing is also applies to all the <tag, word> combinations which appears less than k times and in our baseline we had set k to 5.

Unknown/Unseen handling with POS	Basic HMM
Sentence: Precision: 0.296703 Recall: 0.116379 Fscore: 0.167183 Word: Precision: 0.043478 Recall: 0.016779 Fscore: 0.024213	Sentence: Precision: 0.572289 Recall: 0.409483 Fscore: 0.477387 Word: Precision: 0.270408 Recall: 0.177852 Fscore: 0.214575

Pre-Processing:

There are 1186 total training files in the training set. It is assumed that each sentence is separated by a new line. We replace the CUE-x tags with BIO tag in the files and generated a new file corresponding to each training file.

We also create a cross validation set by splitting the training data in 80-20 fashion. That is, In the training set 20 percent is considered as cross validation set. We use this cross validation set to run all the experiments and to arrive at the best possible model.

Post-Processing:

Our tagger generates all the files tagged with BIO. In preprocessing step we generate kaggle output from these tagged files. To generate the sentence output we have kept the threshold for ambiguous sentence as 1, i.e. if we see more than one B or I in a sentence then we label that sentence as an uncertain sentence.

The threshold of 1 was obtained by testing against the CV set.

In the post-processing part we also generate the Precision, Recall, Fscore for both the word span and uncertain sentence detection task, which helped us tune all the flags to achieve the best model configurations.

Experiments:

Experiments done with different types of unknown word handling, smoothing are described in the respective sections. Other experiments done to compare with the various extensions are listed in the extensions sections.

Results and scores:

A consolidated view of all the results is given below again, the individual parts are presented and explained in respective sections.

Configuration	HMM
Basic	Sentence: Precision: 0.572289

	Recall: 0.409483 Fscore: 0.477387 Word: Precision: 0.270408 Recall: 0.177852 Fscore: 0.214575
UNK handling with Ratio Technique	Sentence: Precision: 0.582822 Recall: 0.409483 Fscore: 0.481012 Word: Precision: 0.302857 Recall: 0.177852 Fscore: 0.224101
Smoothing	Sentence: Precision: 0.359551 Recall: 0.137931 Fscore: 0.199377 Word: Precision: 0.121495 Recall: 0.043624 Fscore: 0.064198
Smoothing and UNK with Ratio Technique	Sentence: Precision: 0.413333 Recall: 0.133621 Fscore: 0.201954 Word: Precision: 0.173333 Recall: 0.043624 Fscore: 0.069705

Extensions:

We have tried 2 different extensions, one is to use POS tag information to help in tagging and other is to resample the Training data.

Extension 1:

Introduction:

We tried using the POS information provided in the training data and playing with it. The fundamental driving thought was to be able to integrate the POS information provided in training data in the basic HMM model. In particular, we report the 2 setups which contrast the use and effect of POS tags:

i) We included the POS tag in the HMM by changing the objective function to maximize $\prod_{i=1}^{n} P(t_i|t_{i-1})P((w_i\, pos_i)|\, t_i)$. Basically enhancing the second term from probability of word given t_i to probability of word with POS context given t_i. The motivating idea was that, it will help in the case where a word has multiple POS tags and can have a different BIO tag corresponding to each POS tag. It augments the differentiating power of the model for every word used in different contexts.

ii) We also tried an extreme case variant: which is to replace P(W|t) with P(POS|t) i.e. relying only on POS tags to predict BIO tags.

Implementation Details:

All the code has been implemented from scratch, and no external libraries are used. These extensions rely on the generic infra built for basic HMM implementation. We modify our data structure which holds training data as follows:

a) for (i) we change words to a tuple of (word, pos)

b) for (ii) we swap words and pos.

We also change the test data accordingly so that it can be tagged by the model and then undo the modifications to generate the tagged files.

Pre-Processing: Same as the basic HMM

Post-Processing: Same as the basic HMM

Experiments:

We have two systems namely, System(i) and System(ii) as described in the introduction block along with their motivations. We also run these class of systems under various settings and report the results.

Expectations:

a) We expect the performance of System(i) to be better than simple HMM as it augments the differentiating power of the model for every word used in different contexts

b) We expect the performance of System(ii) to be significantly worse than HMM as it takes away all the differentiating power of the model for each word, and just makes decision based on POS tags which are very limited in number.

Results and scores:

The performances of both the systems along with HMM model (HMM model refers to HMM implementation having emission probability as P(w|t)) under various configuration are listed below: (As generated on cross validation set)

Configuration	System (i)	System (ii)	HMM
Basic	Sentence: Precision: 0.589595 Recall: 0.439655 Fscore: 0.503704 Word: Precision: 0.290476 Recall: 0.204698 Fscore: 0.240157	Sentence: Precision: 0.365591 Recall: 0.146552 Fscore: 0.209231 Word: Precision: 0.129032 Recall: 0.040268 Fscore: 0.061381	Sentence: Precision: 0.572289 Recall: 0.409483 Fscore: 0.477387 Word: Precision: 0.270408 Recall: 0.177852 Fscore: 0.214575
UNK handling with Ratio Technique	Sentence: Precision: 0.6 Recall: 0.439655 Fscore: 0.507463 Word: Precision: 0.335165 Recall: 0.204698 Fscore: 0.254167	Sentence: Precision: 0.365591 Recall: 0.146552 Fscore: 0.209231 Word: Precision: 0.129032 Recall: 0.040268 Fscore: 0.061381	Sentence: Precision: 0.582822 Recall: 0.409483 Fscore: 0.481012 Word: Precision: 0.302857 Recall: 0.177852 Fscore: 0.224101
Smoothing	Sentence: Precision: 0.296703 Recall: 0.116379	Sentence: Precision: 0.372340 Recall: 0.150862	Sentence: Precision: 0.359551 Recall: 0.137931

	Fscore: 0.167183 Word: Precision: 0.043478 Recall: 0.016779 Fscore: 0.024213	Fscore: 0.214723 Word: Precision: 0.117021 Recall: 0.036912 Fscore: 0.056122	Fscore: 0.199377 Word: Precision: 0.121495 Recall: 0.043624 Fscore: 0.064198
Smoothing and UNK with Ratio Technique	Sentence: Precision: 0.338028 Recall: 0.103448 Fscore: 0.158416 Word: Precision: 0.0704225 Recall: 0.0167785 Fscore: 0.027100	Sentence: Precision: 0.372340 Recall: 0.150862 Fscore: 0.214723 Word: Precision: 0.117021 Recall: 0.036912 Fscore: 0.056122	Sentence: Precision: 0.413333 Recall: 0.133621 Fscore: 0.201954 Word: Precision: 0.173333 Recall: 0.043624 Fscore: 0.069705

Results are inline with the expectation: i.e. System(i) performs better than HMM, and System(ii) performs worse than HMM for the basic config and with unk word handling.

In case of good turing smoothing the performance of System(i) is less than the performance of HMM model, this can be explained by the fact that the number of 0s in the emission counts table increases significantly as the number of columns explode by a factor equivalent to Number of POS tags. As the number of 0s increase, the probability mass stolen from all the bigrams occurring once increases adversely affecting their emission probabilities. Hence making the performance worse.

Extension 2: Sampling Training set (Referred as pruning in code base)

Introduction:

a) To handle the imbalanced dataset, we implemented resampling method and experimented with multiple resampling combinations.

b) Resampling combination has 3 parameters to resample the count of each BIO tags. Based on these parameters train data is preprocessed and updated to reflect the resampling.

c) after seeing o_down number of Os drop the next O which occurs immediately after # we only want to prune oooo to a smaller length, and not things like oooboo as we don't want to loose the transition probability of OB or I # for b_up (i_up), just insert a b-word(i-word) after seeing b_up(i_up) number of Bs (Is) # pass -1 for any of b_up, i_up, o_down if you don't want them to be affected.

Implementation Details:

a) All the code has been implemented from scratch, and no external libraries are used. This extension relies on the generic infra built for basic HMM implementation.

b) Three parameters to tune the resampling are as follows.

i) b_up : After seeing b_up number of B's in train set, copy the last processed line having tag B and write it to train set. This would increase the the count of B's for every b_up number of B's seen in the train set.

ii) i_up : After seeing i_up number of I's in train set, copy the last processed line having tag I and write it to train set. This would increase the the count of I's for every i_up number of I's seen in the train set.

iii) o_down : After seeing o_down number of O's in sequence in train set, remove the current line having tag O and update the train set. This would decrease the the count of O's for every o_down number of O's seen in sequence in the train set. Note that here we looked for o_down numbe of O's in sequence unlike first two cases above. This is to ensure that we only prune oooo to a smaller length, and not things like oooboo as we don't want to loose the transition probability of OB or IO

c) b_up was set to 8, i_up was set to -1 and o_down was set to 1 in the results captured below on resampled data. We came up with this parameters after experimenting with various combinations. (8,5,1) and (8,-1,1) are giving us the best results and files generated based on these parameters are used for kaggle submission.

Pre-Processing: Same as the basic HMM
Post-Processing: Same as the basic HMM

Experiments:
Try with various parameters to handle the resampling and compare the results using cross validation set, and reporting the best result here. The results can be very easily generated for other parameter configurations using the proj_config file in project.

Expectations:
a) Model generated after the resampling should perform better compared to the basic HMM.

- Table shown below captures the test results after resampling and before resampling with other parameter like handling unknowns with ratio technique, with smoothing etc.

Configuration	HMM after resampling train set. (8,-1,1)	HMM Without resampling train set.
Basic	Sentence: Precision: 0.476868 Recall: 0.577586 Fscore: 0.522417 Word: Precision: 0.197802 Recall: 0.241610 Fscore: 0.217522	Sentence: Precision: 0.572289 Recall: 0.409483 Fscore: 0.477387 Word: Precision: 0.270408 Recall: 0.177852 Fscore: 0.214575
UNK handling with Ratio Technique	Sentence: Precision: 0.482014 Recall: 0.577586 Fscore: 0.525490 Word: Precision: 0.218844 Recall: 0.241610 Fscore: 0.229665	Sentence: Precision: 0.582822 Recall: 0.409482 Fscore: 0.481012 Word: Precision: 0.302857 Recall: 0.177852 Fscore: 0.224101

<table>
<tr><td>Smoothing</td><td><u>Sentence:</u>
Precision: 0.492537
Recall: 0.284482
Fscore: 0.360655
<u>Word:</u>
Precision: 0.205714
Recall: 0.1208053
Fscore: 0.152219</td><td><u>Sentence:</u>
Precision: 0.359551
Recall: 0.137931
Fscore: 0.199377
<u>Word:</u>
Precision: 0.121495
Recall: 0.043624
Fscore: 0.064198</td></tr>
<tr><td>Smoothing and UNK with Ratio Technique</td><td><u>Sentence:</u>
Precision: 0.55462
Recall: 0.284482
Fscore: 0.376068
<u>Word:</u>
Precision: 0.283464
Recall: 0.120805
Fscore: 0.169411</td><td><u>Sentence:</u>
Precision: 0.413333
Recall: 0.133621
Fscore: 0.201954
<u>Word:</u>
Precision: 0.173333
Recall: 0.043624
Fscore: 0.069705</td></tr>
<tr><td>Emission probability changed to P(<word,pos> | tag)</td><td><u>Sentence:</u>
Precision: 0.521912
Recall: 0.564655
Fscore: 0.542443
<u>Word:</u>
Precision: 0.255972
Recall: 0.251677
Fscore: 0.253807</td><td><u>Sentence:</u>
Precision: 0.60
Recall: 0.439655
Fscore: 0.5074622
<u>Word:</u>
Precision: 0.335164
Recall: 0.204697
Fscore: 0.254166</td></tr>
</table>

<u>Results:</u> Results seen here are as per the expectation where all the models built with resampled train set shows better results compared to the models built without resampled train set. Column 2 captures the result with resampling and Column 3 captures the result without resampling for the following set of tests. .

a) basic hmm
b) hmm with unknown word handling
c) hmm with smoothing
d) hmm with smoothing and unknown word handling
e) Hmm with emission probability changed to P(<word,pos>|tag) instead of P(word|tag)

hmm with smoothing was not giving us good results because we had lot of unseen bigrams and we had used good turing smoothing technique. Hmm with smoothing on resampled data is much better compared to hmm with smoothing on original data which confirms why smoothing is not giving us the good results with the original train set.

<u>Transition probability table:</u>

Here is how transition probability table looks after applying the resampling methods as explained above. Note that P(O|O) reduced to 0.978 from 0.989 and P(B|O) is increased from 0.010 to 0.021 which is helping us to get better results with resampling. P(B|B) has also increased from 0.014 to 0.124 because of the way resampling is implemented which might be having adverse affects on our results and should have been handled correctly.

Why smoothing is not required for the 0 probability values in transition probability table is explained in design decision section.

O B I

('<phi>',)	0.948422	0.051577	0.0
('I',)	0.445619	0.009063	0.545317
('B',)	0.318316	0.014771	0.666912
('O',)	0.989160	0.010839	0.0

	O	B	I
('<phi>',)	0.948422	0.051577	0.0
('I',)	0.445619	0.009063	0.545317
('B',)	0.282994	0.124097	0.592908
('O',)	0.978015	0.021984	0.0

Kaggle Scores :

Best kaggle scores are listed below is obtained with changing the emission probability definition (extension 1) to P(<word,pos>|tag) from P(word|tag) after applying the resampling methods (Extension 2). Explanation for why we are seeing better results are explained in the corresponding extension section.

Team name : NLP_PAD

Weasel sentence detection score : 0.5588

Weasel phrase detection score : 0.25589

Contribution:

We decided to discuss all the designs together and then divided the modules to be coded between each person, and then clubbed them up all together. Files are distributed based on major contributions to corresponding files.

Pooja :

1) checker.py

2) file_reader.py

3) proj_config.py

4) baseline1.py and baseline.py

Anant :

1) hmm.py

2) project2_driver.py

3) preprocessor_BIO.py

4) baseline1.py and baseline.py

Deekshith :

1) smoothing.py

2) kaggle_op.py

3) cross_validation.py

4) baseline1.py and baseline.py