Report
Kaggle Team “ARYA”

1 Sequence Tagging Model

1.1 Implementation details

We basically use HMM to tackle the sequence tagging task and implement our own sequence tagger. We implemented HMM from scratch.

We approximately use 80% of the txt files in the train folder as the training data.

The result shows the imbalanced dataset containing most tokens as certainty cues belonging to the tag “O”.

For the transition probabilities, we use methods addOneTransitionVal(char tagFrom, char tagTo) and normalizeProbs() to store a transition probability matrix in two-dimension array format called myTransitionProbs[2][2]. myTransitionProbs[i][j] means from tag i to tag j, where 1=B, 2=I, 3=O. The methods getTransitionProb(int i, int j) and getLogTransitionProb(char tagFrom, char tagTo) helps to get the transition probability in different calculation ways.

For the lexical generation probabilities, we create HashMap<String,Double> dict and use the method of processString(ArrayList<String[]> words, String currentCue, StringBuilder result, boolean addOutsides) to input keys and values. For the HashMap dict, the String refers to “word + "\t" + tag” and the Double refers to its occurrence time. Method getLogProbWordGivenTag(String word, char tag) and getProbWordGivenTag(String word, char tag) helps get the probability of “word + "\t" + tag” in Math.Log and decimal format.

We want to decide the tag sequence given the word token sequence. So we choose viterbi algorithm to make the probability P(t1t2...tn | w1w2...wn) maximum value. In function viterbi(String s), we create two-dimension double array score[][] to store probability value. Score[i][t] equals the value of the probability word I given tag t timing the transition probability for word i. The two-dimension bptr[][] helps identify sequence. bprt[i][t] means the value of bptr[i][t] is the index for the max probability value towards the word t given the tag i. In order to identify the sequence for a sentence, it helps with the route identification using the method of back tracing. Then we can predict the most likely tag sequence given the word tokens.

During HMM and Viterbi algorithm, we have to handle unknown words. For the words that never been seen in the train set, we give three same values for the lexical generation probabilities, which means for the word i in this kind, P(Wi|B) = P(Wi|I) = P(Wi|O) = 1/3. For the word that have been seen but not with certain tags, we give the value of Double.MIN_VALUE to unseen combination of words with tags.

Finally, we use the rest 20% train data to conduct the validation. We compare the ground truth and predicted result and calculate PRF accuracy.

1.2 Pre- and post-processing

To performing this pre-processing, we want to train our hmm to distinguish between adjacent weasel words for the task of distinguishing weasel phrases. Since the input format of reading the txt files in the train folders is as follows:

```
ones      NNS    CUE-1
resigned   VBD    _
while     IN    _
```

```
the        DT    CUE-2
```

We split each line by “\t” and change the third column “_” or “CUE-” into tag “B” (beginning of weasel phrase), “I”(inside weasel phrase) or “O”(outside of weasel phrase), then write the words with their tags into a new txt file called train_docs.txt. The contents are as follows.

```
's    POS    O
most    RBS    B
widely    RB   I
```

To post-process for the uncertain phrase task and getting span like 1-1, 2-3, 4-4, 7-8... we need to start a range at the next occurrence of a “B” tag. The end of the uncertain phrase depends on different situation. If the current tag is O and the last tag is I or B, or if the current tag is B and the last tag is I, or if the current tag is B and the last tag is also B, we can say that index - 1 is the end index of an uncertain phrase. In this way we are able to create the ranges required for the kaggle submission.

To post-process for the weasel sentence task, we simply check if a given sentence has at least one “B” tag. If so, the phrase’s index is added to the list of weasel phrases.

1.3 Experiments

Our baseline system was trained by reading the lines, and if a word was ever labeled as a cue word, it was considered as always being a word within a weasel phrase (and therefore weasel sentence). For this part, we convert CUE to BIO and use HMM and Viterbi algorithm to predict weasel words and sentences.

In the experiment, to validate if we have done the HMM and Viterbi correctly and to tune the parameters, we need to get an accurate validation way. So we did post-processing on our validation set (last 20% of the train set). Firstly, we convert the validation set “CUE” to BIO tag. Then we get uncertain phrase span based on it. So we get a String like 71-73 90-92 102-102 431-437 449-450 482-483 509-510... (ground truth). After that, we predict tags for each word token in validation set and get a predicted phrase span String like 71-73 196-198 278-279 431-434 449-451 482-483 518-521...(predict). Then we use function getP(String a, String b), getR(String a, String b), getF(double p, double r) to calculate the accuracy. In this function, we split the two string by space, and store them into two hashmap. To calculate precision, we compare two hashmap to get the correct count, and divide by the count in prediction hashmap. To calculate recall, we divide the correct count by the count in ground truth hashmap. Then we can get F-1 score accroding to P and R. Test 2 is the same process.

The parameters that have been tuned (BIvsOpercent, weaselSentencePercent) were performed using the extension, which will be clarified in more detail later. We changed the value of these parameters and check the change of PRF values.

1.4 Results

Our baseline, achieved F = 0.01325 on task 1, and F = 0.47445 on task 2.

For validation on our improved model, three evaluation indexes give us basic impression about the prediction accuracy for the validation, which are precision score, recall score and F score.

The result for validation set is as below:

	Task 1	Task 2
Baseline F	F = 0.01325	F = 0.47445
P=# correct / # of predictions	P = 0.343	P = 0.579
R=# correct / # of examples in test set	R = 0.158	R = 0.323
F=2*P*R / (P+R)	F = 0.216	F = 0.414

Fig1.1

Obviously our new model is better than our baseline. For Task 2 we can't see the improvement because in baseline, we almost predict every sentence, which is easy to get a higher value. We can see our new model improvement basically by Task 1. The F-1 score turns from 1.3% to 21.6%, which shows the power of sequence labeling.

Before we get this reasonable F-1 score, we have encountered many errors. Some of them are programming bugs, and some of them are due to design choices. For example, at first, we got many consecutive B which is apparently wrong and the accuracy of our model is very low. After checking, we found that if our sentence is long, there may be many 0 scores in Viterbi process. And as we all know, we need to find the maximum score and record its index in bptr[][]. So what will happen if the three values are all 0? At first, our logic always predict B for this. That's why there are so many consecutive B. After changing this to O, our accuracy improves magnificently.

1.5 Competition score

We predict the uncertain phrase and sentence on test folders using the same way as validation. The kaggle result is as below (with extension parameter BIvsOpercent=0.05 and weaselSentencePercent=0.5):

Competition	Score
Uncertain phrase detection	0.22272
Uncertain sentence detection	0.53943

Fig1.2

2. Extension

We used two different methods of up-sampling the training data. The first method is resampleToBalanceBIO(File file, double d) to do the upsample. It repeats tag "B" and tag "I" which increases the ratio of # (B + I) / # (B + I + O). It takes the parameter d that when non-negative is the cut-off ratio (i.e once ratio > d, no more upsampling of this type is performed). Below is a graph of various parameters for this type of upsampling, and the resulting Precision, Recall, and F-Scores for both task1 and task2

BIvsO	P1	P2	F1	P2	R2	F2
0	0.3431	0.1577	0.2161	0.5785	0.3225	0.4142
0.05	0.3695	0.1711	0.2339	0.5785	0.3225	0.4142
0.1	0.3636	0.1611	0.2232	0.5812	0.3133	0.4072
0.25	0.3358	0.151	0.2083	0.5798	0.3179	0.4107

Fig2.1

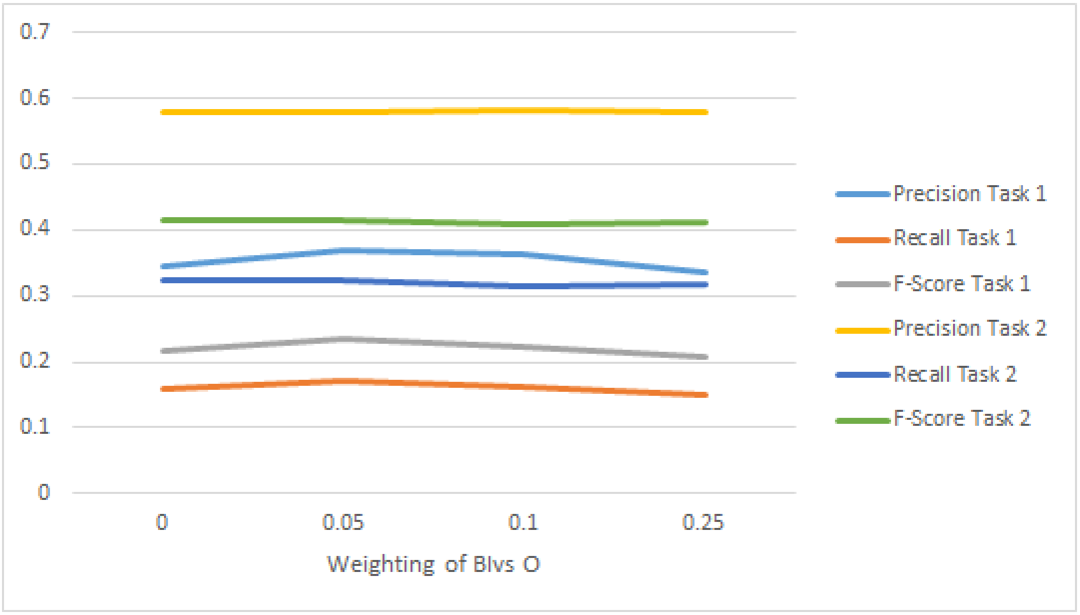

Fig2.2

Because the Task 2 PRF value can be very high if we predict every sentence as weasel, we mainly take Task 1 PRF value to evaluate the performance of our model. From the figures above, we can see that when the ratio of # (B + I) / # (B + I + O) = 0.05, our model performance best (0.0178 higher than no-parameter). We think maybe this is because 5% is very close to the real # (B + I) / # (B + I + O) in validation set. So it can prove more B or I for a word compared to no-parameters. But if the percentage is too high, it may wrongly consider an O word as a B or I word and causes lower accuracy.

The other type of upsampling that we performed was resampleToBalanceWeaselSentences(File file, double d) which repeated weasel sentences in the training data until the ration of weasel sentences / total sentences is at least d. Below is a graph of various values of d, and their effect on Precision, Recall, and F-Score for both tasks 1 and 2.

weaselSent	P1	P2	F1	P2	R2	F2
0	0.3431	0.1577	0.2161	0.5785	0.3225	0.4142
0.25	0.3431	0.1577	0.2161	0.5785	0.3225	0.4142
0.5	0.2877	0.2046	0.2392	0.5219	0.4377	0.4762
0.75	0.2025	0.2718	0.2321	0.4768	0.6636	0.5549

Fig2.
3

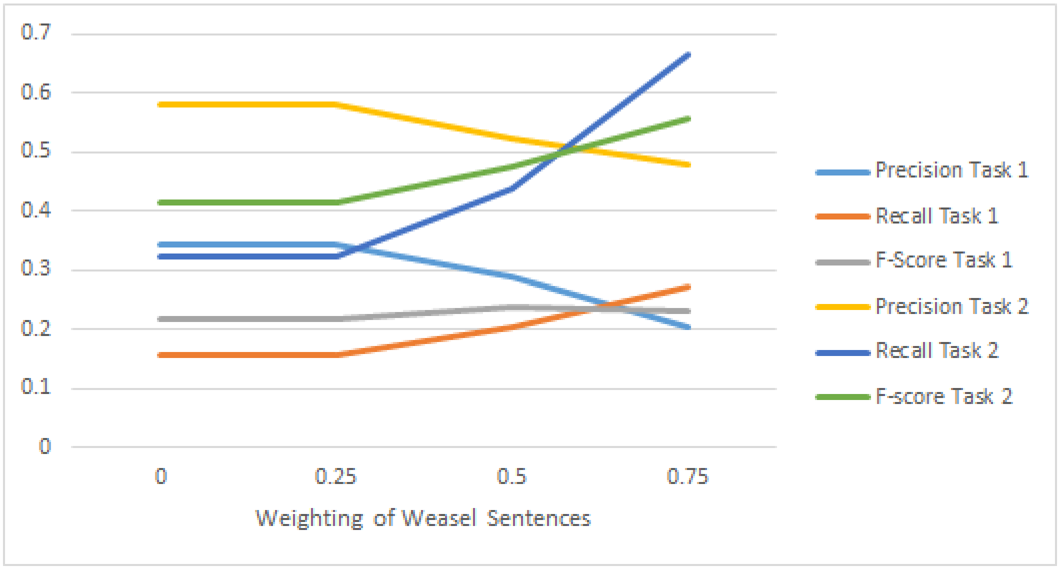

Fig2.
4

Again, because the Task 2 PRF value can be very high if we predict every sentence as weasel, we mainly take Task 1 PRF value to evaluate the performance of our model. From the figures above, we can see that when the ratio of weasel sentences / total sentences= 0.5, our model performance best. We think maybe this is because 50% is very close to the real weasel sentences percentage in validation set. So it can prove more weasel sentences compared to no-parameters. If the percentage is lower than the real percentage(like 0.25), the function will not do anything. And if the percentage is too high, it may repeat weasel sentences so many times that some O words are considered as B or I words so the precision for Task 1 drops.