In [2]:

```
from hmm import SUTDHMM
import os
```

Part 2

Estimate the emission parameters from the training set using MLE. Our approach was to count all the occurences of word give a certain label and the occurences of labels when we do `load_data`. Emission parameters are actually calculated inside `calculate_emission` method (which is called inside `train` method). Please investigate hmm.py (./hmm.py) for further understanding.

In [4]:

```
from hmm import SUTDHMM
languages = ['EN', 'SG', 'CN', 'FR']

for l in languages:
    model = SUTDHMM(k=3)
    model.train(input_filename='./{}/train'.format(l))
    print("Finish training for {}".format(l))

    with open("./{}/dev.in".format(l)) as in_file, open("./{}/dev.p2.out".format
(l), 'w+') as out_file:
        for line in in_file:
            word = line.strip()
            if (word == ''):
                out_file.write("\n")
            else:
                out_file.write("{} {}\n".format(word, model.predict_label_using_
emission(word)))
    print("Finished: {}".format(l))

    output = os.popen("python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.p2.o
ut".format(l)).read()
    print("Language: {}".format(l))
    print(output)
    print("----------------------")
```

```
Finish training for EN
Finished: EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 1201

#Correct Entity : 165
Entity  precision: 0.1374
Entity  recall: 0.7301
Entity  F: 0.2313

#Correct Sentiment : 71
Sentiment  precision: 0.0591
Sentiment  recall: 0.3142
Sentiment  F: 0.0995

----------------------
Finish training for SG
Finished: SG
Language: SG

#Entity in gold data: 1382
#Entity in prediction: 6599

#Correct Entity : 794
Entity  precision: 0.1203
Entity  recall: 0.5745
Entity  F: 0.1990

#Correct Sentiment : 315
Sentiment  precision: 0.0477
Sentiment  recall: 0.2279
Sentiment  F: 0.0789

----------------------
Finish training for CN
Finished: CN
Language: CN

#Entity in gold data: 362
#Entity in prediction: 3318

#Correct Entity : 183
Entity  precision: 0.0552
Entity  recall: 0.5055
Entity  F: 0.0995

#Correct Sentiment : 57
Sentiment  precision: 0.0172
Sentiment  recall: 0.1575
Sentiment  F: 0.0310

----------------------
Finish training for FR
Finished: FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 1149
```

```
#Correct Entity : 182
Entity  precision: 0.1584
Entity  recall: 0.8161
Entity  F: 0.2653

#Correct Sentiment : 68
Sentiment  precision: 0.0592
Sentiment  recall: 0.3049
Sentiment  F: 0.0991

----------------------
```

Part 3

We calculate the transition parameters the same way as emission paramters are calculated. We counted all the occurences of the different transitions and occurences of the different labels at the `load_data` step and make the calculation in `calculate_transition` method (which is in turn called inside `train` method). Please investigate hmm.py (./hmm.py) for further understanding.

Our Viterbi algorithm is implemented as the method `viterbi` of the main class `SUTDHMM`, which makes use of the previously calculated emission and transition parameters.

In [3]:

```
languages = ['EN', 'SG', 'CN', 'FR']

for l in languages:
    model = SUTDHMM(k=3)
    model.train(input_filename='./{}/train'.format(l))

    print("Finish training for {}".format(l))

    print("----------Viterbi for {0}------------".format(l))
    with open("./{}/dev.in".format(l)) as in_file, open("./{}/dev.p3.out".format
(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_labels, chance = model.viterbi(sentence)
            for idx, word in enumerate(sentence.split()):
                out_file.write("{} {}\n".format(word, sentence_labels[idx]))
            out_file.write('\n')
        out_file.close()
        in_file.close()

    print("Viterbi Finished: {}".format(l))

    output = os.popen(
        "python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.p3.out".format(l))
.read()
    print("Language: {}".format(l))
    print(output)
```

```
Finish training for EN
----------Viterbi for EN------------
Viterbi Finished: EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 171

#Correct Entity : 99
Entity  precision: 0.5789
Entity  recall: 0.4381
Entity  F: 0.4987

#Correct Sentiment : 63
Sentiment  precision: 0.3684
Sentiment  recall: 0.2788
Sentiment  F: 0.3174

Finish training for SG
----------Viterbi for SG------------
Viterbi Finished: SG
Language: SG

#Entity in gold data: 1382
#Entity in prediction: 818

#Correct Entity : 310
Entity  precision: 0.3790
Entity  recall: 0.2243
Entity  F: 0.2818

#Correct Sentiment : 195
Sentiment  precision: 0.2384
Sentiment  recall: 0.1411
Sentiment  F: 0.1773

Finish training for CN
----------Viterbi for CN------------
Viterbi Finished: CN
Language: CN

#Entity in gold data: 362
#Entity in prediction: 188

#Correct Entity : 20
Entity  precision: 0.1064
Entity  recall: 0.0552
Entity  F: 0.0727

#Correct Sentiment : 13
Sentiment  precision: 0.0691
Sentiment  recall: 0.0359
Sentiment  F: 0.0473

Finish training for FR
----------Viterbi for FR------------
Viterbi Finished: FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 170
```

```
#Correct Entity : 110
Entity  precision: 0.6471
Entity  recall: 0.4933
Entity  F: 0.5598

#Correct Sentiment : 74
Sentiment  precision: 0.4353
Sentiment  recall: 0.3318
Sentiment  F: 0.3766
```

Part 4

The max marginal approach attempts to find the optimal path with the following approach:

$$y_i^* = \arg\max_{y_i} \{p(y_i \mid x_1, x_2, \ldots, x_n; \theta)\}$$

The conditional probability of a state u occuring for y_i is given as follows:

$$p(y_i = u \mid x_1, x_2, \ldots, x_n; \theta) = \frac{p(x_1, x_2, \ldots x_{i-1}, y_i = u, x_i, \ldots, x_n; \theta)}{p(x_1, \ldots x_n; \theta)}$$

As $x_i, \ldots x_n$ are independent of $x_1, \ldots x_{i-1}$ once y_i is known in a Hidden Markov Model, the conditional probability could be written as such:

$$p(y_i = u \mid x_1, x_2, \ldots, x_n; \theta) = \frac{p(x_1, x_2, \ldots x_{i-1}, y_i = u; \theta) p(x_i, \ldots, x_n | y_i = u; \theta)}{p(x_1, \ldots x_n; \theta)}$$

$$= \frac{\alpha_u(i)\beta_u(i)}{\sum_v \alpha_v(j)\beta_v(j)}, \quad \text{where} \quad j \in (1, 2, \ldots n)$$

Thus, the following result could be obtained to indicate the optimum state for each y

$$y_i^* = \arg\max_u \frac{\alpha_u(i)\beta_u(i)}{\sum_v \alpha_v(j)\beta_v(j)} = \arg\max_u \alpha_u(i)\beta_u(i)$$

In [5]:

```
languages = ['EN', 'SG', 'CN', 'FR']

for l in languages:
    model = SUTDHMM(k=3)
    model.train(input_filename='./{}/train'.format(l))

    print("Finish training for {}".format(l))

    print("----------Max Marginal for {0}------------".format(l))
    with open("./{}/dev.in".format(l)) as in_file, open("./{}/dev.p4.out".format
(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_labels = model.max_marginal(sentence)
            for idx, word in enumerate(sentence.split()):
                out_file.write("{} {}\n".format(word, sentence_labels[idx]))
            out_file.write('\n')
        out_file.close()
        in_file.close()

    print("Max Marginal Finished: {}".format(l))

    output = os.popen(
        "python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.p4.out".format(l))
.read()
    print("Language: {}".format(l))
    print(output)
```

```
Finish training for EN
----------Max Marginal for EN------------
Max Marginal Finished: EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 306

#Correct Entity : 137
Entity  precision: 0.4477
Entity  recall: 0.6062
Entity  F: 0.5150

#Correct Sentiment : 81
Sentiment  precision: 0.2647
Sentiment  recall: 0.3584
Sentiment  F: 0.3045

Finish training for SG
----------Max Marginal for SG------------
Max Marginal Finished: SG
Language: SG

#Entity in gold data: 1382
#Entity in prediction: 1140

#Correct Entity : 438
Entity  precision: 0.3842
Entity  recall: 0.3169
Entity  F: 0.3473

#Correct Sentiment : 271
Sentiment  precision: 0.2377
Sentiment  recall: 0.1961
Sentiment  F: 0.2149

Finish training for CN
----------Max Marginal for CN------------
Max Marginal Finished: CN
Language: CN

#Entity in gold data: 362
#Entity in prediction: 439

#Correct Entity : 89
Entity  precision: 0.2027
Entity  recall: 0.2459
Entity  F: 0.2222

#Correct Sentiment : 58
Sentiment  precision: 0.1321
Sentiment  recall: 0.1602
Sentiment  F: 0.1448

Finish training for FR
----------Max Marginal for FR------------
Max Marginal Finished: FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 294
```

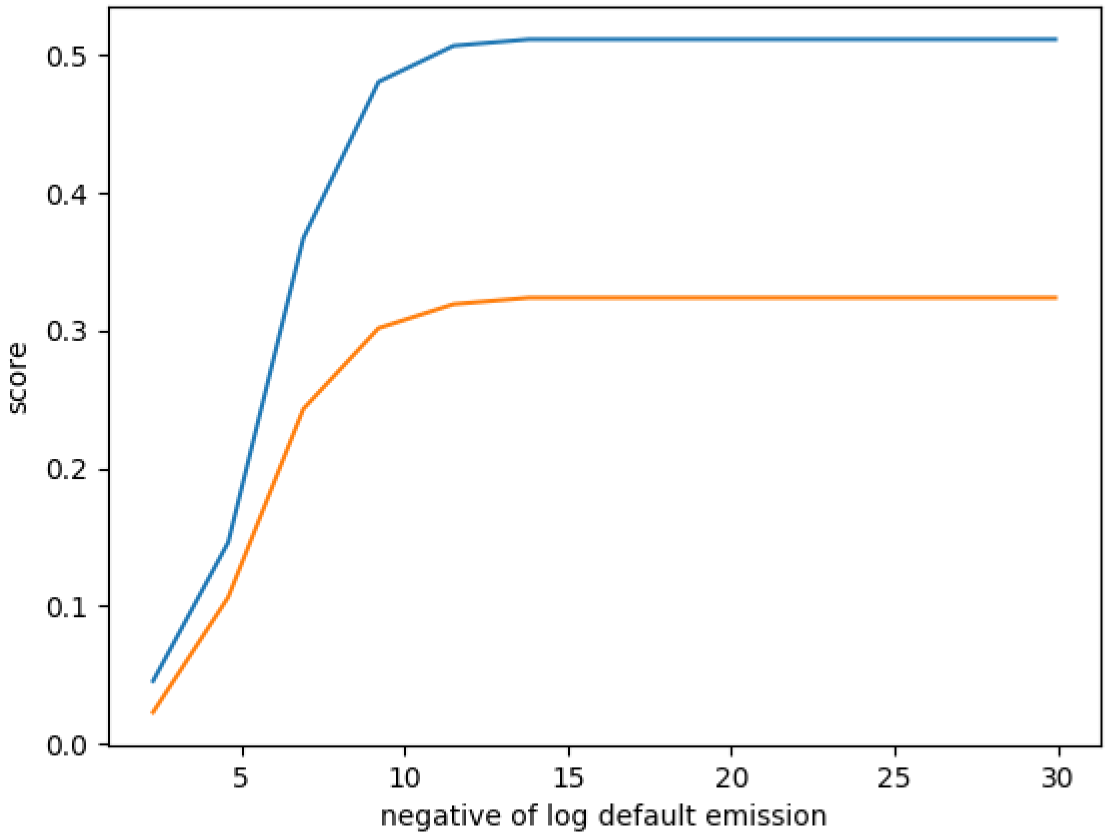
```
#Correct Entity : 172
Entity  precision: 0.5850
Entity  recall: 0.7713
Entity  F: 0.6654

#Correct Sentiment : 96
Sentiment  precision: 0.3265
Sentiment  recall: 0.4305
Sentiment  F: 0.3714
```

Part 5

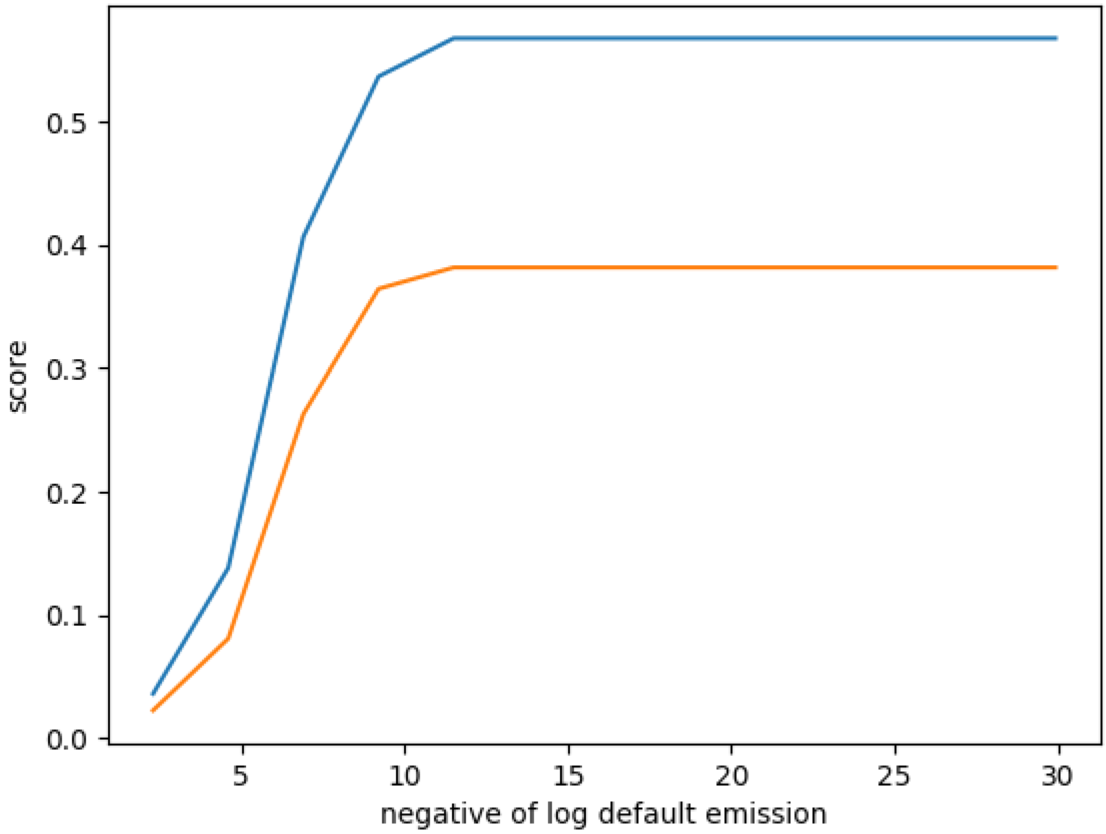
Improvement 1: Default Emission Params

The first proposed improvement to current algorithm is to set a default small emission parameters. We realised that a lot of time, if a word is never tagged with a specific label before in training set, the "path" that passes through such pair of word and label will always have a probability of 0, no matter how likely the transition between that label and the previous/next labels are. Also, in real life ("the universal bag of word"), there is always a probability, even if it's small, that a word is tagged with a specific label, it makes more sense to give all pair of word and label a default probability (emission param). We tested our hypothesis by implementing it as an option in our main algorithm class. We used the elbow method with a range from 10^{-3} to 10^{-20} to identify what is the best default param.

Results on EN dataset



Results on FR dataset



Implementation with default parameters with Viterbi

In [7]:

```
languages = ['EN', 'FR']

for l in languages:
    model = SUTDHMM(default_emission=0.000001)
    model.train(input_filename='./{}/train'.format(l))

    print("Finish training for {}".format(l))

    print("----------Viterbi for {0}------------".format(l))
    with open("./{}/dev.in".format(l)) as in_file, open("./{}/dev.pdefault.out".
format(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_labels, chance = model.viterbi(sentence)
            for idx, word in enumerate(sentence.split()):
                out_file.write("{} {}\n".format(word, sentence_labels[idx]))
            out_file.write('\n')
        out_file.close()
        in_file.close()

    print("Viterbi Finished: {}".format(l))

    output = os.popen(
        "python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.pdefault.out".form
at(l)).read()
    print("Language: {}".format(l))
    print(output)
```

```
Finish training for EN
----------Viterbi for EN------------
Viterbi Finished: EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 200

#Correct Entity : 109
Entity  precision: 0.5450
Entity  recall: 0.4823
Entity  F: 0.5117

#Correct Sentiment : 69
Sentiment  precision: 0.3450
Sentiment  recall: 0.3053
Sentiment  F: 0.3239

Finish training for FR
----------Viterbi for FR------------
Viterbi Finished: FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 196

#Correct Entity : 119
Entity  precision: 0.6071
Entity  recall: 0.5336
Entity  F: 0.5680

#Correct Sentiment : 80
Sentiment  precision: 0.4082
Sentiment  recall: 0.3587
Sentiment  F: 0.3819
```

Implementation with default parameters with Max Marginal

In [16]:

```
languages = ['EN', 'FR']

for l in languages:
    model = SUTDHMM(default_emission=0.000001)
    model.train(input_filename='./{}/train'.format(l))

    print("Finish training for {}".format(l))

    print("----------Max Marginal for {0}------------".format(l))
    with open("./{}/dev.in".format(l)) as in_file, open("./{}/dev.p4.out".format
(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_labels = model.max_marginal(sentence)
            for idx, word in enumerate(sentence.split()):
                out_file.write("{} {}\n".format(word, sentence_labels[idx]))
            out_file.write('\n')
        out_file.close()
        in_file.close()

    print("Max Marginal Finished: {}".format(l))

    output = os.popen(
        "python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.p4.out".format(l))
.read()
    print("Language: {}".format(l))
    print(output)
```

```
Finish training for EN
----------Max Marginal for EN------------
Max Marginal Finished: EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 326

#Correct Entity : 154
Entity  precision: 0.4724
Entity  recall: 0.6814
Entity  F: 0.5580

#Correct Sentiment : 93
Sentiment  precision: 0.2853
Sentiment  recall: 0.4115
Sentiment  F: 0.3370

Finish training for FR
----------Max Marginal for FR------------
Max Marginal Finished: FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 312

#Correct Entity : 185
Entity  precision: 0.5929
Entity  recall: 0.8296
Entity  F: 0.6916

#Correct Sentiment : 107
Sentiment  precision: 0.3429
Sentiment  recall: 0.4798
Sentiment  F: 0.4000
```

Conclusion: The default emission params perform best for our datasets at 10^{-6}. This method gives us a better result than our original algorithms

Improvement 2: Predicting Entity and Sentiment separately

The second proposed improvement for current algorithm is to separate the prediction of entity and sentiment. We theorize that because entity and sentiment labels are not related, so if we join them as a single label, they will affect the probability of each other and produce a lower accuracy. For example, entity B might show up more frequently in the dataset together with sentiment "negative", however, they are actually not related but in this case, the probability of predicting B-negative are significantly high compared to other tags. This imply a false correlation between the entity tag (B) and the sentiment tag (negative). Hence, predicting them separately would eliminate this false correlation thus produce better result

Implementation with entity and sentiment separately on Viterbi

In [6]:

```
languages = ['EN', 'FR']

for l in languages:
    model = SUTDHMM()
    model.load_data(data_filename='./{}/train'.format(l))
    with open('./{}/train.ent'.format(l), 'w+') as ent_in_file:
        for token in model.tokens_list:
            word = token[0]
            tag = token[1].split(
                '-')[0] if token[1] not in ['O', 'START', 'STOP'] else token[1]
            ent_in_file.write('{} {}\n'.format(word, tag))
            if token[1] == 'STOP':
                ent_in_file.write('\n')
        ent_in_file.close()
    with open('./{}/train.sen'.format(l), 'w+') as sen_in_file:
        for token in model.tokens_list:
            word = token[0]
            tag = token[1].split(
                '-')[1] if token[1] not in ['O', 'START', 'STOP'] else token[1]
            sen_in_file.write('{} {}\n'.format(word, tag))
            if token[1] == 'STOP':
                sen_in_file.write('\n')
        sen_in_file.close()

    ent_model = SUTDHMM()
    ent_model.train(input_filename='./{}/train.ent'.format(l))
    sen_model = SUTDHMM()
    sen_model.train(input_filename='./{}/train.sen'.format(l))
    print('Finished training for {}'.format(l))
    with open('./{}/dev.in'.format(l)) as in_file, open('./{}/dev.psep.out'.form
at(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_ent, prob = ent_model.viterbi(sentence=sentence)
            sentence_sen, prob = sen_model.viterbi(sentence=sentence)
            for idx in range(0, len(sentence_ent)):
                entity = sentence_ent[idx]
                sentiment = sentence_sen[idx]
                if entity not in ['O', 'START', 'STOP'] and sentiment not in [
'O', 'START', 'STOP']:
                    out_file.write(
                        "{} {}-{}\n".format(word, entity, sentiment))
                elif entity in ['O', 'START', 'STOP']:
                    out_file.write("{} {}\n".format(word, entity))
                else:
                    out_file.write('{} {}\n'.format(word, sentiment))
            out_file.write('\n')

    output = os.popen("python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.pse
p.out".format(l)).read()
    print("Language: {}".format(l))
    print(output)
```

```
Finished training for EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 115

#Correct Entity : 74
Entity  precision: 0.6435
Entity  recall: 0.3274
Entity  F: 0.4340

#Correct Sentiment : 53
Sentiment  precision: 0.4609
Sentiment  recall: 0.2345
Sentiment  F: 0.3109

Finished training for FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 104

#Correct Entity : 74
Entity  precision: 0.7115
Entity  recall: 0.3318
Entity  F: 0.4526

#Correct Sentiment : 49
Sentiment  precision: 0.4712
Sentiment  recall: 0.2197
Sentiment  F: 0.2997
```

Implementation with entity and sentiment separately on Max Marginal

In [10]:

```
languages = ['EN', 'FR']

for l in languages:
    model = SUTDHMM()
    model.load_data(data_filename='./{}/train'.format(l))
    with open('./{}/train.ent'.format(l), 'w+') as ent_in_file:
        for token in model.tokens_list:
            word = token[0]
            tag = token[1].split(
                '-')[0] if token[1] not in ['O', 'START', 'STOP'] else token[1]
            ent_in_file.write('{} {}\n'.format(word, tag))
            if token[1] == 'STOP':
                ent_in_file.write('\n')
        ent_in_file.close()
    with open('./{}/train.sen'.format(l), 'w+') as sen_in_file:
        for token in model.tokens_list:
            word = token[0]
            tag = token[1].split(
                '-')[1] if token[1] not in ['O', 'START', 'STOP'] else token[1]
            sen_in_file.write('{} {}\n'.format(word, tag))
            if token[1] == 'STOP':
                sen_in_file.write('\n')
        sen_in_file.close()

    ent_model = SUTDHMM(k=3)
    ent_model.train(input_filename='./{}/train.ent'.format(l))
    sen_model = SUTDHMM(k=3)
    sen_model.train(input_filename='./{}/train.sen'.format(l))
    print('Finished training for {}'.format(l))
    with open('./{}/dev.in'.format(l)) as in_file, open('./{}/dev.psep.out'.form
at(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_ent = ent_model.max_marginal(sentence=sentence)
            sentence_sen = sen_model.max_marginal(sentence=sentence)
            for idx in range(0, len(sentence_ent)):
                entity = sentence_ent[idx]
                sentiment = sentence_sen[idx]
                if entity not in ['O', 'START', 'STOP'] and sentiment not in [
'O', 'START', 'STOP']:
                    out_file.write(
                        "{} {}-{}\n".format(word, entity, sentiment))
                elif entity in ['O', 'START', 'STOP']:
                    out_file.write("{} {}\n".format(word, entity))
                else:
                    out_file.write('{} {}\n'.format(word, sentiment))
            out_file.write('\n')

    output = os.popen("python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.pse
p.out".format(l)).read()
    print("Language: {}".format(l))
    print(output)
```

```
Finished training for EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 285

#Correct Entity : 131
Entity  precision: 0.4596
Entity  recall: 0.5796
Entity  F: 0.5127

#Correct Sentiment : 75
Sentiment  precision: 0.2632
Sentiment  recall: 0.3319
Sentiment  F: 0.2935

Finished training for FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 278

#Correct Entity : 165
Entity  precision: 0.5935
Entity  recall: 0.7399
Entity  F: 0.6587

#Correct Sentiment : 96
Sentiment  precision: 0.3453
Sentiment  recall: 0.4305
Sentiment  F: 0.3832
```

Conclusion: This method only gives a slightly better result on current datasets. But it would make a whole lot of difference if the datasets are skewed towards certains pair of tags.

Improvement 3

The third proposed improvement to the current algorithm is to learn implement the discriminative training methods using perceptron (adopted from this paper (http://www.aclweb.org/anthology/W02-1001)). The basic steps of implementation are as below:

Inputs: A training set of tagged sentences, $(w^i_{[1:n_i]}, t^i_{[1:n_i]})$ for $i = 1 \ldots n$. A parameter T specifying number of iterations over the training set. A "local representation" ϕ which is a function that maps history/tag pairs to d-dimensional feature vectors. The global representation Φ is defined through ϕ as in Eq. 1.
Initialization: Set parameter vector $\bar{\alpha} = 0$.
Algorithm:

For $t = 1 \ldots T, i = 1 \ldots n$
- Use the Viterbi algorithm to find the output of the model on the i'th training sentence with the current parameter settings, i.e.,

$$z_{[1:n_i]} = \arg\max_{u_{[1:n_i]} \in \mathcal{T}^{n_i}} \sum_s \alpha_s \Phi_s(w^i_{[1:n_i]}, u_{[1:n_i]})$$

where $\mathcal{T}^{n_i}$ is the set of all tag sequences of length n_i.
- If $z_{[1:n_i]} \neq t^i_{[1:n_i]}$ then update the parameters

$$\alpha_s = \alpha_s + \Phi_s(w^i_{[1:n_i]}, t^i_{[1:n_i]}) - \Phi_s(w^i_{[1:n_i]}, z_{[1:n_i]})$$

Output: Parameter vector $\bar{\alpha}$.

We have implemented it inside our main class `SUTDHMM` as a special training method `train_perceptron` and a special predicting method `predict_perceptron`. Please investigate the code for further understanding.

In [7]:

```
languages = ['EN', 'FR']

for l in languages:
    model = SUTDHMM()
    model.train_perceptron(input_filename='./{}/train'.format(l))
    print("Finish training for {}".format(l))

    print("----------Perceptron for {0}------------".format(l))
    with open("./{}/dev.in".format(l)) as in_file, open("./{}/dev.perceptron.ou
t".format(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(
            filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_labels, chance = model.predict_perceptron(sentence)
            for idx, word in enumerate(sentence.split()):
                out_file.write("{} {}\n".format(
                    word, sentence_labels[idx]))
            out_file.write('\n')
        out_file.close()
        in_file.close()

    print("Perceptron Finished: {}".format(l))

    output = os.popen(
        "python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.perceptron.out".fo
rmat(l)).read()
    print("Language: {}".format(l))
    print(output)
```

```
Finish training for EN
----------Perceptron for EN------------
Perceptron Finished: EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 103

#Correct Entity : 63
Entity  precision: 0.6117
Entity  recall: 0.2788
Entity  F: 0.3830

#Correct Sentiment : 45
Sentiment  precision: 0.4369
Sentiment  recall: 0.1991
Sentiment  F: 0.2736

Finish training for FR
----------Perceptron for FR------------
Perceptron Finished: FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 109

#Correct Entity : 78
Entity  precision: 0.7156
Entity  recall: 0.3498
Entity  F: 0.4699

#Correct Sentiment : 51
Sentiment  precision: 0.4679
Sentiment  recall: 0.2287
Sentiment  F: 0.3072
```

Conclusion: This proposed method doesnot improve the accuracy for our specific usecase.

Combination

Finally, we would like to attempt to combine our previously proposed improvement to the algorithm design. We found that the best combination is method 1 and method 2 together

Implementation with combination on Viterbi

In [13]:

```
languages = ['EN', 'FR']

for l in languages:
    model = SUTDHMM()
    model.load_data(data_filename='./{}/train'.format(l))
    with open('./{}/train.ent'.format(l), 'w+') as ent_in_file:
        for token in model.tokens_list:
            word = token[0]
            tag = token[1].split(
                '-')[0] if token[1] not in ['O', 'START', 'STOP'] else token[1]
            ent_in_file.write('{} {}\n'.format(word, tag))
            if token[1] == 'STOP':
                ent_in_file.write('\n')
        ent_in_file.close()
    with open('./{}/train.sen'.format(l), 'w+') as sen_in_file:
        for token in model.tokens_list:
            word = token[0]
            tag = token[1].split(
                '-')[1] if token[1] not in ['O', 'START', 'STOP'] else token[1]
            sen_in_file.write('{} {}\n'.format(word, tag))
            if token[1] == 'STOP':
                sen_in_file.write('\n')
        sen_in_file.close()

    ent_model = SUTDHMM(default_emission=0.0000001)
    ent_model.train(input_filename='./{}/train.ent'.format(l))
    sen_model = SUTDHMM(default_emission=0.0000001)
    sen_model.train(input_filename='./{}/train.sen'.format(l))
    print('Finished training for {}'.format(l))
    with open('./{}/dev.in'.format(l)) as in_file, open('./{}/dev.pdefaultsep.ou
t'.format(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_ent, prob = ent_model.viterbi(sentence=sentence)
            sentence_sen, prob = sen_model.viterbi(sentence=sentence)
            for idx in range(0, len(sentence_ent)):
                entity = sentence_ent[idx]
                sentiment = sentence_sen[idx]
                if entity not in ['O', 'START', 'STOP'] and sentiment not in [
'O', 'START', 'STOP']:
                    out_file.write(
                        "{} {}-{}\n".format(word, entity, sentiment))
                elif entity in ['O', 'START', 'STOP']:
                    out_file.write("{} {}\n".format(word, entity))
                else:
                    out_file.write('{} {}\n'.format(word, sentiment))
            out_file.write('\n')

    output = os.popen("python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.pdef
aultsep.out".format(l)).read()
    print("Language: {}".format(l))
    print(output)
```

```
Finished training for EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 196

#Correct Entity : 106
Entity  precision: 0.5408
Entity  recall: 0.4690
Entity  F: 0.5024

#Correct Sentiment : 69
Sentiment  precision: 0.3520
Sentiment  recall: 0.3053
Sentiment  F: 0.3270

Finished training for SG
Language: SG

#Entity in gold data: 1382
#Entity in prediction: 1003

#Correct Entity : 460
Entity  precision: 0.4586
Entity  recall: 0.3329
Entity  F: 0.3857

#Correct Sentiment : 278
Sentiment  precision: 0.2772
Sentiment  recall: 0.2012
Sentiment  F: 0.2331

Finished training for FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 167

#Correct Entity : 109
Entity  precision: 0.6527
Entity  recall: 0.4888
Entity  F: 0.5590

#Correct Sentiment : 76
Sentiment  precision: 0.4551
Sentiment  recall: 0.3408
Sentiment  F: 0.3897
```

Implementation with combination on Max Marginal

In [17]:

```
languages = ['EN', 'FR']

for l in languages:
    model = SUTDHMM()
    model.load_data(data_filename='./{}/train'.format(l))
    with open('./{}/train.ent'.format(l), 'w+') as ent_in_file:
        for token in model.tokens_list:
            word = token[0]
            tag = token[1].split(
                '-')[0] if token[1] not in ['O', 'START', 'STOP'] else token[1]
            ent_in_file.write('{} {}\n'.format(word, tag))
            if token[1] == 'STOP':
                ent_in_file.write('\n')
        ent_in_file.close()
    with open('./{}/train.sen'.format(l), 'w+') as sen_in_file:
        for token in model.tokens_list:
            word = token[0]
            tag = token[1].split(
                '-')[1] if token[1] not in ['O', 'START', 'STOP'] else token[1]
            sen_in_file.write('{} {}\n'.format(word, tag))
            if token[1] == 'STOP':
                sen_in_file.write('\n')
        sen_in_file.close()

    ent_model = SUTDHMM(default_emission=0.0000001)
    ent_model.train(input_filename='./{}/train.ent'.format(l))
    sen_model = SUTDHMM(default_emission=0.0000001)
    sen_model.train(input_filename='./{}/train.sen'.format(l))
    print('Finished training for {}'.format(l))
    with open('./{}/dev.in'.format(l)) as in_file, open('./{}/dev.pdefaultsep.ou
t'.format(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_ent = ent_model.max_marginal(sentence=sentence)
            sentence_sen = sen_model.max_marginal(sentence=sentence)
            for idx in range(0, len(sentence_ent)):
                entity = sentence_ent[idx]
                sentiment = sentence_sen[idx]
                if entity not in ['O', 'START', 'STOP'] and sentiment not in [
'O', 'START', 'STOP']:
                    out_file.write(
                        "{} {}-{}\n".format(word, entity, sentiment))
                elif entity in ['O', 'START', 'STOP']:
                    out_file.write("{} {}\n".format(word, entity))
                else:
                    out_file.write('{} {}\n'.format(word, sentiment))
            out_file.write('\n')

    output = os.popen("python3 EvalScript/evalResult.py {0}/dev.out {0}/dev.pdef
aultsep.out".format(l)).read()
    print("Language: {}".format(l))
    print(output)
```

```
Finished training for EN
Language: EN

#Entity in gold data: 226
#Entity in prediction: 320

#Correct Entity : 148
Entity  precision: 0.4625
Entity  recall: 0.6549
Entity  F: 0.5421

#Correct Sentiment : 88
Sentiment  precision: 0.2750
Sentiment  recall: 0.3894
Sentiment  F: 0.3223

Finished training for FR
Language: FR

#Entity in gold data: 223
#Entity in prediction: 301

#Correct Entity : 176
Entity  precision: 0.5847
Entity  recall: 0.7892
Entity  F: 0.6718

#Correct Sentiment : 105
Sentiment  precision: 0.3488
Sentiment  recall: 0.4709
Sentiment  F: 0.4008
```

Conclusion: This proposed method gives better results than original algorithms but worse than method 1 alone

Final Proposed Model

After the above analysis, we decide to have our method 1 (i.e. default emission params) with max_marginal to be our contender for the real test data. Below is the code to generate test outputs. These outputs can be found in **/test/EN/test.out** and **/test/FR/test.out**

In case you want to run out chosen algorithm on a different dataset, there is a `generate_output.py` script with the instruction written in [instruction.txt (./instruction.txt)](./instruction.txt)

In [20]:

```
languages = ['EN', 'FR']

for l in languages:
    model = SUTDHMM(default_emission=0.000001)
    model.train(input_filename='./{}/train'.format(l))

    print("Finish training for {}".format(l))

    print("----------Max Marginal for {0}------------".format(l))
    with open("./test/{}/test.in".format(l)) as in_file, open("./test/{}/test.ou
t".format(l), 'w+') as out_file:
        read_data = in_file.read()
        sentences = list(filter(lambda x: len(x) > 0, read_data.split('\n\n')))
        sentences = list(map(lambda x: ' '.join(x.split('\n')), sentences))
        for sentence in sentences:
            sentence_labels = model.max_marginal(sentence)
            for idx, word in enumerate(sentence.split()):
                out_file.write("{} {}\n".format(word, sentence_labels[idx]))
            out_file.write('\n')
        out_file.close()
        in_file.close()

    print("Output Generated for {}".format(l))
```

```
Finish training for EN
----------Max Marginal for EN------------
Output Generated for EN
Finish training for FR
----------Max Marginal for FR------------
Output Generated for FR
```