POS TAGGING

Using Hidden Markov Model

By
Ravi Yadav
Biplab Ch Das

Generating the Lexical Probabilities

We know that $p(W \mid T) = P(W,T) / P(T)$

And since $p(W,T) = N(W,T) / N(T)$

We Used a Hash Map to get all the counts of tagged words in the training corpus. That is we first find N(W,T). And then divide it by N(T).

Generating The transition Probabilities

We know that p(Tn | Tn-1)=P(Tn,Tn-1) / P(Tn-1)

And since p(W,T)=N(Tn-1,Tn) / N(Tn-1)

Used Hash Map again. Got the N(t(n-1),t(n)) values and then we divided it by N(t(n-1)).

Separation of folds

The folds were separated in the ratio of 4:1

4 for training and 1 for the testing.

5 folds were created and the models were tested and trained on each folds.

Smoothing of the data

For the non existent data we tried to smoothen the data by adding a minor value(0.0000001) to the count.

A single zero in product can make the whole term zero.

The viterbi algorithm

The viterbi algorithm was implemented to get the best possible state sequence using the transition matrix and the emission matrix. The best possible state sequence was considered as the best tag possible for the sentence

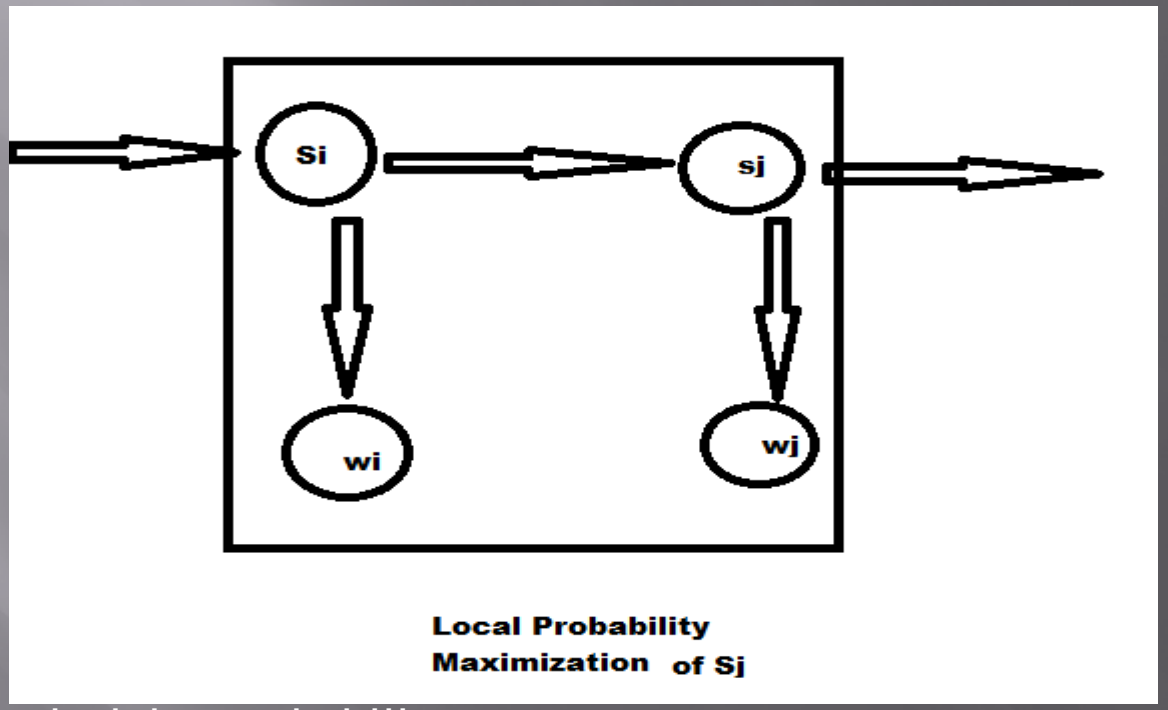
The previous algorithm implemented

Till now the pos tagger have implemented the viterbi algorithm ,at the same time it we kept the greedy local lookup that we presented before.

$T(n)^* = \text{argmax}_{(Tj)(} p((T_j \mid T_i) * p(Wj \mid t_j))$;

Here ‘i’is prev state and ‘j’ varies.

Belief network of assumption



We take joint probability
P(Si,Sj,wi,wj)=P(sj|si)*P(si)*P(wj|sj)*P(wi|si)
And set P(si)=1 and P(wi|si)=1(Considering the prev state is constant , we don't need to cosider P(si) for local maximisation)

Confusion Matix

The confusion matrix have been created. The actual tags were shown in the right hand side column and the tags by the Viterbi algorithm were enumerated at the bottom line.

file:///C:/Users/Biplab/Desktop/confusion_matrix.html

Confusion Reason

Most of the confusion was between NP0 and NN1

It was because of the unknown word prediction handling done in the code.

The F-score

The proper output for the tag wise precision-recall and the F-score has been calculated .

The F-score was calculated putting the beta value to 1.

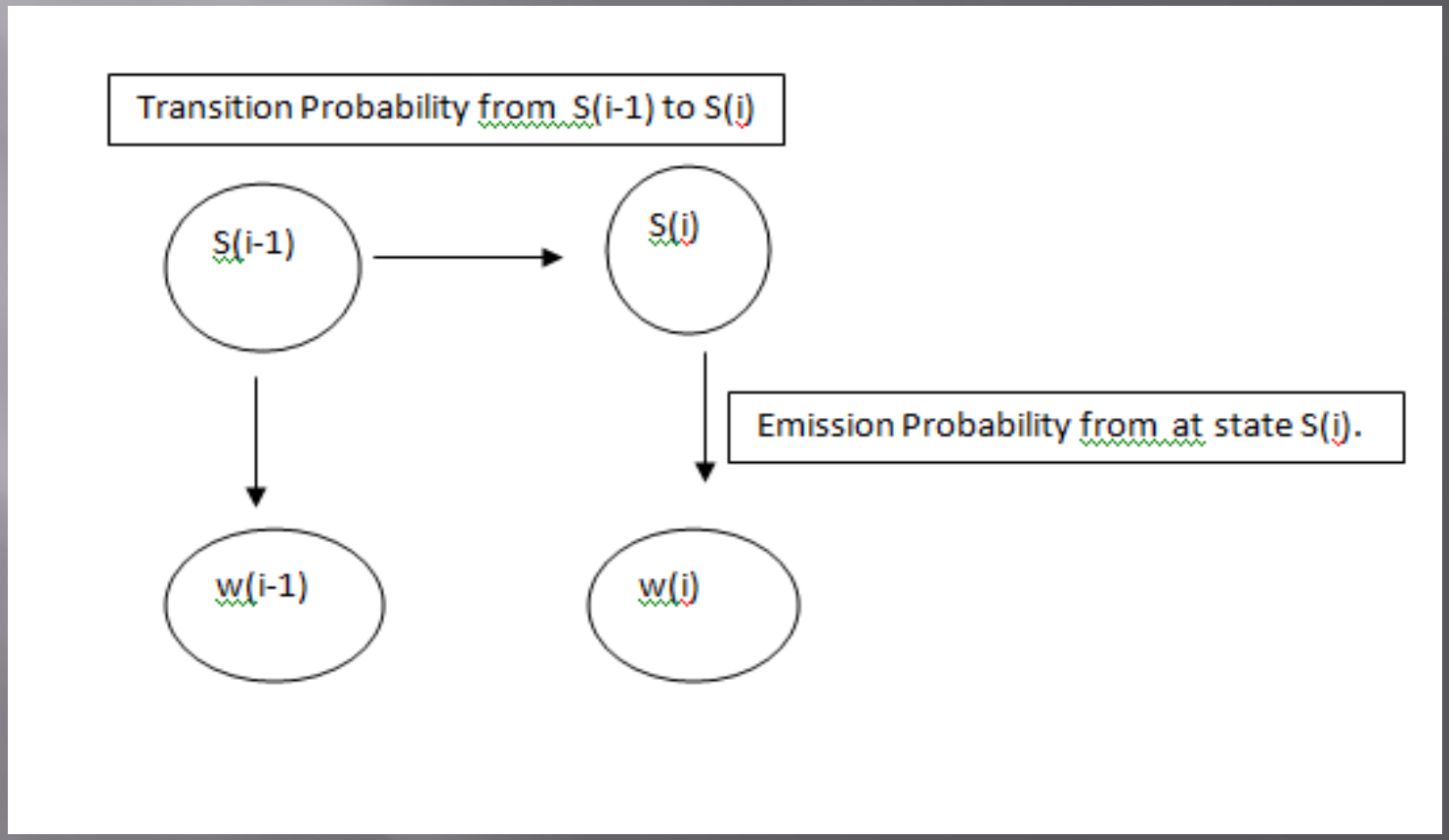
file:///C:/Users/Biplab/Desktop/proper.html

Trigram Implentation

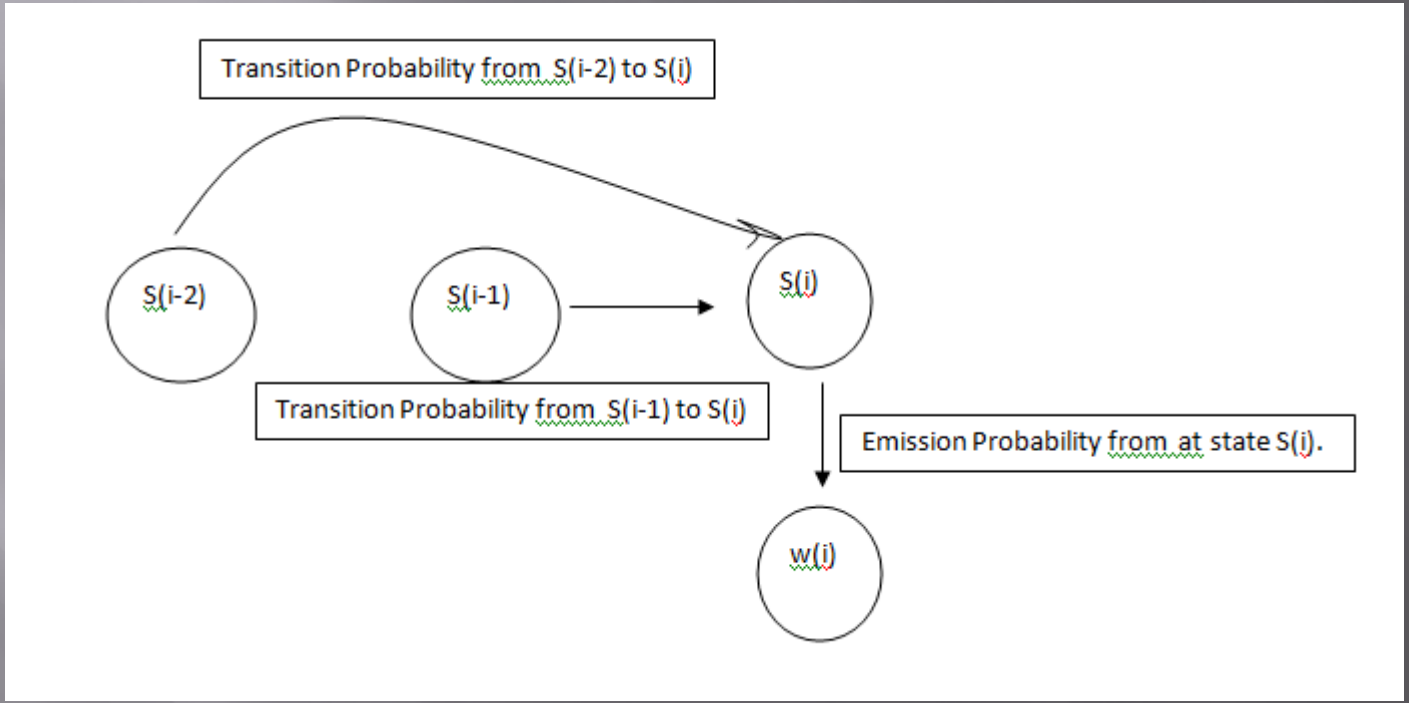
The trigram model was implemented for the Pos tagger.

In this case the current tag not only depends on the previous state but also on the previous to previous state.

Trigram the extension of bigram model:



The graphical model for trigram model:



A star algorithm

Heuristic used:

$-h_{(Ti)} = \log(P(Wi \mid Ti)) + \sum_{(j= i+1 \text{ to } n)} \log(\max(P(Wj \mid Tj))) + \sum_{(} \log(\max(Ti \mid Tj))$

Where,

$P(Wi \mid Ti)$ is the lexical prob at a node.

$\max(Ti \mid Tj)$ is the maximum of all transitional probabilities

$\max(Ti \mid Tj)$ is the max lexical prob at a node

Accuracy of the tagger

Fold No	Bigram	Skip gram (0.75*bigram+.25 skipgram)	(.23*bigram+.11* skipgram+.66*trigram)	A* algorithm
Fold 1	92.69	92.71	92.82	93.20
Fold 2	93.76	94.08	94.28	94.75
Fold 3	93.61	93.69	93.73	94.66
Fold 4	94.00	94.06	94.17	94.72
Fold 5	93.22	93.38	93.64	93.86
Overall	93.45	93.58	93.73	94.23

Discriminitive

Model tag directly from the data.

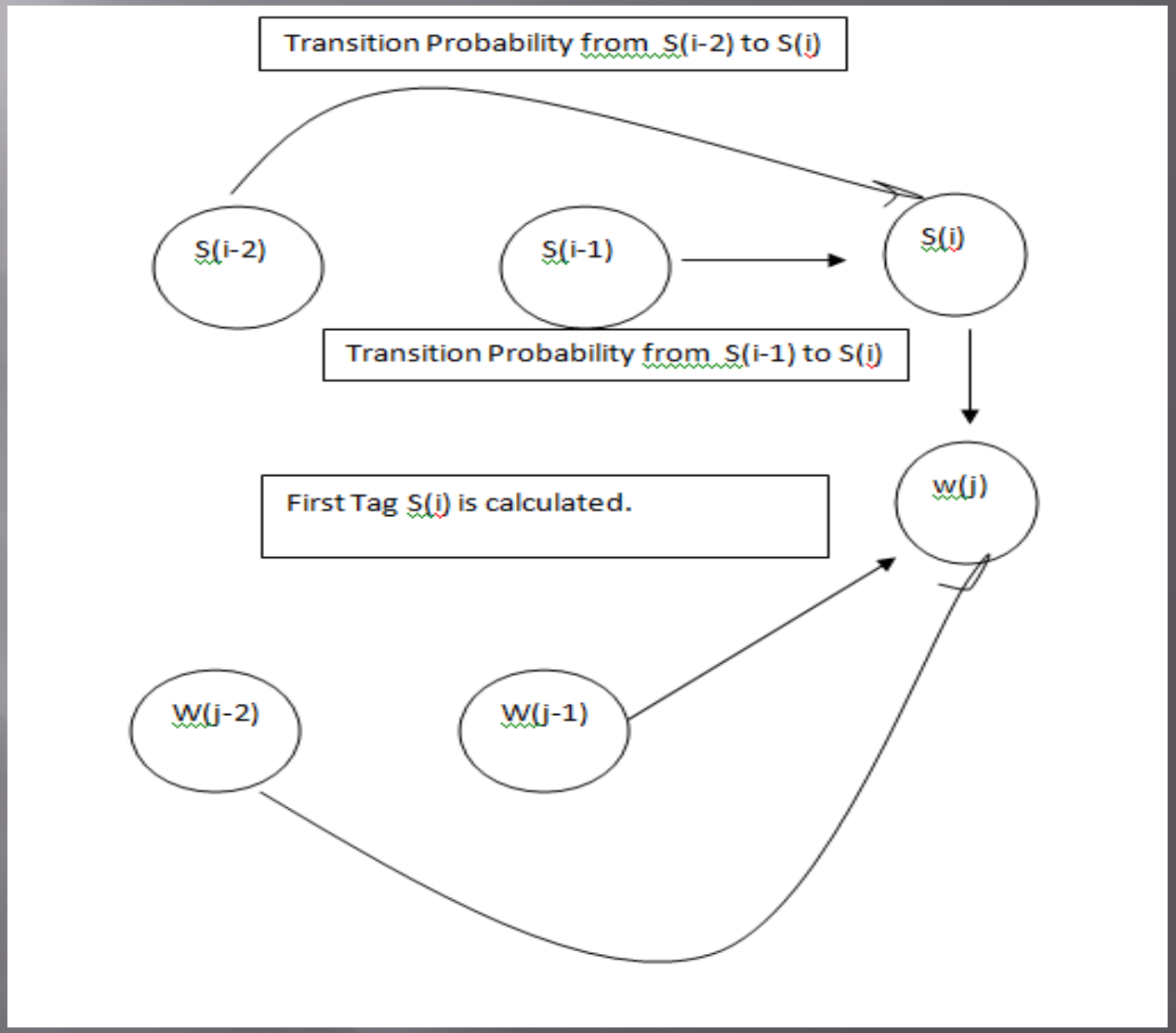
Expression :

Argmax(j) P(T(j)|T(j-1),Current_word)

Accuracy acieved 72.98 %

Possible problem :Unknown word handling was not done .

Word Prediction



Model for word prdeiction:

$$\text{argmax} \sum_{(Nw)} \sum_{(Tw)} I(Tw) * sim(Syn(Nw), Syn(Tw))$$

Where argmax is over Nw

Here nw=argmax(wi,wj) maximizing over wj

Sim is a similarity function

Syn(W) refers to the synset of the words

Accuracy of the word prediction model

Measured using trigram of content words and checked with the top 5 suggestions.

It came out to be 43.30% without knowing pos tags and with pos tags it increased to 46.00%

Most of the confusion occurred when the next words involved names.