An Unsupervised Parts-of-Speech Tagger for the Bangla language

Hammad Ali
Department of Computer Science
University of British Columbia
hammada@cs.ubc.ca

Abstract

In this paper we present the results of some initial experiments performed in developing an unsupervised Parts-of-Speech (POS) tagger for the Bangla language. We start with mentioning some of the work done in this area, and present the rationale for trying an unsupervised approach. We then describe the resources used for the project, the underlying mechanism for unsupervised learning and present some of the primary results. The paper then suggests future directions of work in this area.

1 Introduction

Part-of-Speech (POS) tagging is the process of assigning each word of a text with an appropriate parts of speech tag. The significance of part-of-speech (also known as POS, word classes, morphological classes, or lexical tags) for language processing is the large amount of information they give about a word and its neighbours. POS tags often signify the morphological, phonological and contextual properties of a word, and also provide information about neighbour words. POS tagging can be used in Text to Speech applications, information retrieval and extraction, shallow parsing, linguistic research for corpora and also as an intermediate step for higher level NLP tasks such as parsing, semantics, translation, and many more. POS tagging, thus, is a necessary application for advanced NLP applications in any language.

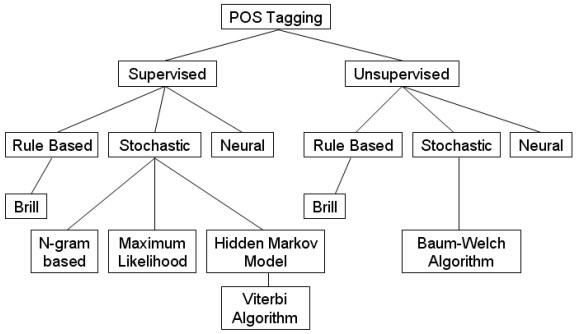
Bangla is one of the top ten most widely spoken languages in the world, with more than 200 million native speakers all around the globe. Along with languages like Hindi and Telugu, the grammar and morphological rules for Bangla is derived from Sanskrit, an ancient language which was the primary language of written discourse in south-east Asia up until the beginning of the 20th century. In addition, a lot of words in Bangla have been absorbed from other foreign languages, and the original Bangla words have passed out of common usage. Despite the long tradition and the wide number of people who use Bangla as their first language, there has still not been significant research in the area of natural language processing for Bangla.

We start this paper by giving an overview of the different approaches to POS tagging, and describe what has been done so far for Bangla. We then compare the previous approach to what we propose to do in this study. Then we describe our POS tagset and the corpus used. Next we describe the toolkit we decided to use for the purpose of our experiments, and explain in detail the algorithm that this toolkit is supposed to execute on the dataset. Finally we conclude with the results of our study and suggest what could be possible future steps for further work on this problem.

2 Literature Review

POS tagging can be seen as a learning task, and hence the approaches to the task can be divided into two broad categories – supervised and unsupervised learning. Within each of these branches, there are further sub-divisions based on the finer details of how they approach the problem. The following figure gives an

overview of the different POS tagging models, which is then followed by short descriptions of how each model works.



2.1 Supervised Models

The supervised approach to POS tagging requires human knowledge of the domain, often in the form of a corpus that has been hand-annotated by human experts. This is referred to as the training corpus. This corpus is used to learn information about the tagset, word-tag frequencies, rules etc. The performance of supervised approaches depends on the size and annotation quality of the training corpus.
In case of the rule-based approach, the tagger tries to assign tags to each word based on a set of hand-crafted rules. These rules could specify, for instance, that a word following a determiner and an adjective must be a noun. This approach requires that the set of rules be properly written and checked by human experts. This makes design of the rule-based taggers time-consuming and expensive. The stochastic approach uses a training corpus to pick the most probable tag given a word. These stochastic methods could be based on simple N-gram based methods, or on the First or Second order Hidden Markov Models. Finally, the transformation based approach combines the rule-based and stochastic approach. It picks the most likely tag given a word based on the training corpus. Then it applies a set of rules to see whether the tag should be changed to something else. This can be thought of as adding progressively finer details at each run of the rule-application steps, and the process halts when there is insufficient improvement between two consecutive iterations. It saves any new rules it has learnt in the process for future use. One example of an effective tagger in this category is the Brill tagger.

While the supervised approach has been implemented for English and several other languages with good results, it suffers from the drawback that it needs a human-annotated corpus or set of rules. For a language like Bangla, where linguistic research is still at an early stage, such training corpora or set of rules are not easily available. This then severely limits the applications of supervised training for languages like these.

2.2 Unsupervised Models

The unsupervised POS tagging models do not require a pre-annotated corpus. Instead, they use advanced computational techniques like the Baum-Welch algorithm to automatically learn the transformation rules. Based on this information, they can either generate the Markov model required by stochastic taggers or induce the contextual rules needed by the rule-based or transformation-based systems. Later in this paper we will see more details on how the Baum-Welch algorithm operates on a given dataset.

3 Previous work

In this section we will discuss some of the previous work that has been done regarding POS tagging for Bangla. One early study on POS tagging for Bangla has been reported by Chowdhury et al (2004) and Seddiqui et al (2003). Chowdhury et al (2004) implemented a rule-based tagger, which requires writing laboriously hand-crafted rules by human experts. However, they report no performance analysis of their work. No review or comparison of established work in Bangla is offered in that paper; they only propose a rule-based technique. Further, they use a tagset consisting of only 9 tags. Such a tagger would have very limited applicability in advanced NLP applications.
More recently, work has been done by Hasan et al (2006) on developing a supervised POS tagger for Bangla. While the lack of a large hand-tagged corpus makes this method less effective, the authors were aiming to conduct a study that compares the performance of a supervised tagger for English and Bangla, using training corpora of the same size. They use a 5000 word corpus and a tagset of 41 tags. The results obtained are satisfactory, in the sense that the accuracy is comparable to that for similar-sized corpus for English. The authors thus claim that with a large enough corpora for Bangla, the performance of

their tagger can be similar to that for English. In addition, the authors point out that the rule-based approach seems to perform much better for Bangla than the stochastic approach, which is the opposite of what has been observed for English.
In Hasan et al (2007), the authors set out to compare three South Asian languages – Bangla, Hindi and Telugu - all of which are derived from Sanskrit. For Bangla, this study uses a corpus of 25426 tokens and 26 tags. The accuracy figures improve from that in the previous study, but are still not significant enough. Further, it is noticed that for all the three languages, the rule-based tagger performs better than the stochastic tagger. The authors hypothesize that this could be because these languages are derived from Sanskrit, which is a very rule-driven language, and mention that this aspect of the problem deserves some more research.

4 Methodology

As mentioned above, even with a nearly 25000 token corpus the results do not seem very promising. Further, as has been done for most other languages, we believe that the next phase in POS tagging research for Bangla should be an attempt at the unsupervised approach. This would eliminate the need to develop a large hand-annotated corpus – something that is not easily available for Bangla and is time-consuming and laborious to prepare. With this in mind, we collected the corpus and the tagset from university labs currently working on linguistic research in Bangla. Our aim was to perform Baum-Welch training on this corpus, in order to learn the underlying HMM parameters. This HMM could then be used to perform tagging on a corpus and tested to obtain the accuracy figures. So the entire study could be broken down to the following phases: 1) collect corpus and tagset, 2) search for implementation of Baum-Welch algorithm, 3) perform training and 4) test against gold standard for accuracy. We now describe the resources collected for this project, and explain how the Baum-Welch algorithm operates.

5 Tagset

We used a 54-tag tagset developed for the Bangla language specifically. This tagset extends most standard tagsets, to include finer distinctions specific to the Bangla language – such as suffixes that denote possessive or accusative markers. The tagset, along with descriptions if individual tags, is provided in the appendix.

6 Corpora

We used corpora currently being developed by the Center for Research on Bangla Language Processing. This corpus is collected from a leading Bangladeshi newspaper called Prothom-Alo, and is in UTF-8 format. A small subset of this corpus has also been tagged by human experts, and was made available to us from the same source. The corpus totals to about 50000 tokens, and the tagged subset consists of 18110 tokens and 4760 word types. One portion of this tagged corpus was cleaned and set aside as the training corpus and the rest was to be used as the test set.

7 The Baum-Welch algorithm

In this section we describe the operation of the Baum-Welch algorithm. In particular, we describe what the inputs should be, the steps executed and the expected output once training has been completed.
The main idea of the Baum-Welch algorithm is to find the Hidden Markov Model given a sequence of observations. As input, we need it to provide all the possible states, and a sequence of observations. Specific to our POS tagging task, the states are the possible tags, and the observed symbols are the word types in the corpus. A sequence of observations then is a sentence in our corpus.
The HMM is supposed to specify two matrices: the transition probability matrix T and the emission probability matrix E. Any entry t_{ij} in the transition matrix specifies the probability of moving from state i to state j. For our example, this matrix will specify the probability of one word having a tag i and the next word in sequence having the state j. An entry e_{ij} in the emission matrix denotes the probability of observing the symbol j given that we are at state i. This is analogous to the probability of observing one of the possible word types, given that we know the tag to be assigned for the word or observation. Once these two matrices are known for all possible state transitions and all possible emissions, we can take any observation sequence and predict what the underlying states are. In other words, given a sentence we can tag each word in the sentence. The probability that a

sequence starts with a state i is denoted by the initial state distribution, which is often denoted by π.

Since this is an unsupervised learning algorithm, no human annotation of states is provided with the observation. For our problem in particular, the training algorithm is simply provided with a list of all the possible states, and the sentences in the training corpus. As output, the algorithm is supposed to generate the transition and emission matrices. We now describe step by step how this is done.

Initialization: To begin with, T, E and π are initialized to random values. The algorithm will later update these values until some convergence criterion has been reached. Possible criteria for convergence can be having finished a specific number of iterations, or more commonly insufficient improvement between two consecutive iterations.

Forward step: The Forward algorithm is then executed. For this phase, we define a quantity $\alpha_i(t) = p(O_1 = o_1, , ,O_t = o_t, Q_t = i| \lambda)$. This denotes the probability of observing the sequence of symbols $O_1 \ldots O_t$, and then ending up in state I, given the HMM denoted by λ. A recursive relation for $\alpha_i(t)$ can be defined as shown below:

Base case: $\alpha_i(t) = \pi_i b_i(O_1)$

Recursive relation: $\alpha_j(t+1) = b_j(O_{t+1}) \Sigma_{1\text{-}n}\, \alpha_i(t)\, T_{ij}$

In the above relation, $b_i(O_1)$ denotes the probability of observing symbol O_1 at state i. Similarly $b_j(O_{t+1})$ denotes the probability of observing symbol O_{t+1} at state j. In general, the recurrence denotes that the probability of observing t+1 symbols and ending in state j is given by: the probability of observing t symbols and ending in state i, transitioning from state i to state j, and at state j observing the symbol O_{t+1}. As can be seen, this recurrence relation satisfies the probability of optimal substructure – optimal path to a sequence must be generated through optimal path to a subsequence. Thus it can be solved using a straightforward dynamic programming approach.

Backward step: In this step the Backward algorithm is executed. This time we aim to calculate $\beta_i(t)$, which denotes the probability of the ending partial sequence $o_{t+1}, , , o_T$ given that we started at state i, at time t. Once again we can derive a recurrence relation for this quantity as follows:

Base case: $\beta_i(T) = 1$

Recursive relation: $\beta_i(t) = \Sigma_{1\text{-}n}\, \beta_j(t+1) T_{ij}\, b_j(O_{t+1})$

Once again, we see that the backward probability also displays optimal substructure, and can thus be computed by dynamic programming. Once we know α and β, we can use these to calculate the following quantities:

$$\gamma_i(t) \equiv p(Q_t = i \mid O, \lambda) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$$

$$\xi_{ij}(t) \equiv p(Q_t = i, Q_{t+1} = j \mid O, \lambda) = \frac{\alpha_i(t) a_{ij} \beta_j(t+1) b_j(o_{t+1})}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i(t) a_{ij} \beta_j(t+1) b_j(o_{t+1})}$$

Once we have these two variables, they can be used to update the original HMM parameters as follows:

$$\overline{\pi}_i = \gamma_i(1)$$

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

$$\overline{b}_i(k) = \frac{\sum_{t=1}^{T} \delta_{O_t, o_k} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)}$$

These forward and backward steps are repeated over and over until the convergence condition has been satisfied, at which point we have the best HMM that can be learned from the given observation sequence. This HMM can then be used on other observation sequences, for which it will generate the sequence of hidden states that can then be tested for accuracy – either by automated comparison or by a human expert.

8 Toolkits

8.1 C++ HMM toolkit

Since our basic idea was to implement Baum-Welch training in order to learn the HMM

parameters by itself, we started looking for an existing implementation of this algorithm. After searching for and trying out a few, we decided to use the HMM toolkit developed by Dekang Lin, at the University of Alberta. This toolkit is developed in C++, and contains three major implementations: 1) the vit program which implements the Viterbi algorithm to generate the most probable sequence of tags given a sentence, 2) the genseq program which generates an observation sequence given a HMM model and 3) the trainhmm program which learns the HMM parameters given a set of observations and an initial estimate for the HMM. Along with the toolkit, some dataset was also provided to train a POS tagger for the English language. An online tutorial is also available for the toolkit, which explains the format the input files must be in and how to use the different features of the toolkit.

In general, the Baum-Welch algorithm is not dependent on the accuracy of the initial probability distribution and matrices provided. With a less accurate estimate, training would simply take more number of iterations before converging. However, as mentioned by the author of the package and easily testable from the dataset provided, the performance of this toolkit depends on the initial estimate. In particular, if the emission probability is assumed to be equal for all the symbols given any tag, then Baum-Welch is not able to learn the correct HMM. For English, the author obtains a better estimate by using the lexicon for Collin's parser, and is then able to learn the correct HMM. These estimates, and the training and test data, are provided with the toolkit for testing purposes.
Since there is no existing lexicon for Bangla, our first experiment was conducted with the most naïve initial estimate possible – assuming all transitions and all emissions are equally likely. Just like in the case for English, Baum-Welch was unable to learn the correct HMM based on these parameters. As a second approach, we then tried to build a small lexicon from the training corpus and get a better estimate for the emission probabilities based on this lexicon. However even with these revised estimates, Baum-Welch was still unable to learn the model properly. If we try generating a sequence of tags using the HMM provided as output, all the words in the sentences are labelled with NN, which is the high-level tag for nouns. So at this point, no further improvement seemed possible using this toolkit. Thus we moved on to try training with the Natural Language Toolkit (NLTK).

8.2 Natural Language Toolkit

As pointed out by a fellow student, NLTK also provides unsupervised training under the HiddenMarkovModelTrainer module. To use this trainer, one needs to specify the possible states and symbols, and then provide a set of observation sequences on which training can be done. An initial HMM can also be provided but is optional. After taking a look at the sample codes in the documentation, we then developed a Python module to perform unsupervised training on the training corpus, with the tagset and list of word types provided as inputs. However, even this implementation was not able to learn a correct model. Training ran for only three iterations before converging, and using the output model to generate tags gave the same result as before – all the words were tagged NN. Since this module does not need an initial estimate and none was provided, it would seem that this inability of learn the HMM has less to do with the accuracy of the estimates and is more due to some other reason relevant to the training corpus itself. This conclusion is further strengthened by the fact that in case of both the toolkits, the final output is the same – all symbols tagged as NN. However there has not been enough time to explore possible reasons for this.

9 Results

As mentioned in the previous section, both the toolkits have been unable to learn a HMM good enough to be of practical use. While in terms of total log probability, NLTK has done better than the other toolkit, this improvement has clearly not resulted in better output. In terms of final output then, the project has not been successful. However, we believe that some directions have been suggested for future work that could possibly lead to better results. We will discuss this in more details in the rest of this paper.

10 Evaluation

If the project had been successful, evaluation would be straightforward: 1) take the tagged corpus and produce a clean copy of it, 2) train the tagger on one portion of this cleaned corpus, 3) apply tagger obtained on the other portion and 4) compare with hand-tagged version of corpus to obtain accuracy figures. However, we have

not been able to complete the training phase properly with either toolkit. Therefore, we have not been able to move on to stages 3 and 4. While some of the symbols in the test set are in fact supposed to be tagged NN and would be considered correct in an automated testing, such an accuracy is hardly representative of the actual state of the tagger and would not be a very useful figure to go by.

11 Contribution

While we have not been able to reach our desired results, there are still some lessons to take away from the work that has been done. In particular, since there is no existing lexicon for Bangla and little likelihood of a high-quality one being generated anytime soon, it would seem that there is little point in continuing further research with the first toolkit, which has already been shown to be highly sensitive to the accuracy of the initial estimate. On the other hand, while NLTK has not performed well either, we can mention at least two points in its favour: 1) there is no need to provide an initial parameter and 2) even if unsuccessful, the training gave better results that with the same test data for the first one. Even if a lexicon is generated and really good estimates obtained, it is still more likely that providing them as parameters to the NLTK trainer method would yield better results than using them as input for the other HMM toolkit. Further work in this area should thus focus more on getting NLTK training and tagging modules to work.

12 Future Work

At the very initial stage, we proposed to develop both an unsupervised Baum-Welch and Brill tagger for Bangla. The chief objective behind this was to test whether the phenomenon of rule-based taggers working better than stochastic taggers - which was observed in previous research - is true in this case as well. Consistency in this aspect would provide further corroboration to the hypothesis that as a heavily rule-driven language, Bangla performs better with rule-based taggers. Given time constraints, we have not been able to complete that for this study. However, we believe that once the unsupervised HMM tagger has been developed, the unsupervised Brill tagger should be the next step, followed by comparison between the two. As far as the developing the unsupervised tagger is concerned, the next step should be to try the training with different corpora to see whether there has been any improvement. The set of experiments that have been carried out so far could also be tried on other toolkits, to see whether they do any better. Even more important, in the event that they do not, we should check whether they give the same results, since making the same mistake could be an indication that the problem is independent of the tools used, and probably specific to the dataset or the tagset.

13 Conclusion

In this paper we have mentioned the process we followed in order to develop a Baum-Welch trained HMM tagger for the Bangla language. We began with an overview of the POS tagging problem in general, and mentioned the different approaches to POS tagging in brief. We then presented some of the previous work done for POS tagging in Bangla, and outlined how our approach is different and should need less human knowledge in order to work. Having described the corpus and the tagset, we then described the Baum-Welch algorithm, along with formulae for the variables that are calculated. Lastly, we described the toolkits we used for the project and how they work. The results of this work have already been presented above, along with suggestions on what could be done next. In the end, the project has not been able to give quantitative results, but we believe that some progress has been made in the sense that we now have a clearer idea of what resources might be helpful for the task. Pursuing further in this direction of work should be able to give some initial results, depending on which the next plan of actions can be decided on.

14 References

B. Greene and G. Rubin, Automatic Grammatical Tagging of English, Technical Report, Department of Linguistics, Brown University, Providence, Rhode Island, 1971.

D. Cutting, J. Kupiec, J. Pederson and P. Sibun, A practical Part-Of-Speech Tagger, in proceedings of the Third Conference on Applied Natural Language Processing. 1992.

Daniel Jurafsky and James H. Martin, Chapter 8: Word classes and Part-Of-Speech Tagging, Speech and Language Processing, Prentice Hall, 2000.

Eric Brill, A simple rule based part of speech tagger, in proceedings of the Third Conference on Applied Natural Language Processing. 1992.
Eric Brill, Automatic grammar induction and parsing free text: A transformation based approach, in proceedings of 31st Meeting of the Association of Computational Linguistics, 1993.
Eric Brill, Transformation based error driven parsing, in proceedings of the Third International Workshop on Parsing Technologies, Tilburg, The Netherlands, 1993.
Eric Brill, Unsupervised Learning of Disambiguation Rules for Part-of-Speech Tagging, in proceedings of The Natural Language Processing Using Very Large Corpora, Boston, MA, 1997.
Helmut Schmid, Probabilistic Part-Of-Speech Tagging using Decision Trees, in proceedings of The International Conference on new methods in language processing, 1994.
K. W. Church, A stochastic parts program and noun phrase parser for unrestricted text, in proceeding of the Second Conference on Applied Natural Language Processing, 1988.

L. Bahl and R. L. Mercer, Part-Of-Speech assignment by a statistical decision algorithm, IEEE International Symposium on Information Theory, 1976.
Linda Van Guilder, Automated Part of Speech Tagging: A Brief Overview, Fall 1995, Georgetown University.
Mihai Pop, Unsupervised Part-of-Speech Tagging, Department of Computer Science, Johns Hopkins University, 1996.
S. J. DeRose, Grammatical Category Disambiguation by Statistical Optimization, Computational Linguistics, 14(1), 1988.
S. Klein and R. Simmons, A computational approach to grammatical coding of English words, JACM 10, 1963
The Summer Institute for Linguistics (SIL) Ethnologue Survey 1999.
Yair Halevi, Part of Speech Tagging, Seminar in Natural Language Processing and Computational Linguistics, School of Computer Science, Tel Aviv University, Israel, April 2006.
Z. Harris, String Analysis of Language Structure, Mouton and Co., The Hague, 1962.

Appendix A

Source codes

A1 Code for lexicon generation

```
corpus = open('bangla-corpus-cleaned.txt','r')
lexicon = open('bangla-lexicon.txt','w')

tokencount=0
typecount=0

wordmap = {}

for line in corpus:
    tokens = line.split()
    for each in tokens:
        if wordmap.has_key(each): wordmap[each]+=1
        else: wordmap[each]=1

for entry in wordmap:
    tokencount+=wordmap[entry]
    lexicon.write(entry)
    lexicon.write("\n")

print len(wordmap)
print tokencount

corpus.close()
lexicon.close()
```

A2 Code for training in NLTK

```
import nltk
from nltk import tokenize

def loadUntagged(fileName):

    text = open(fileName).read()
    sentences = list(tokenize.blankline(text))
    retSentences = []

    for sentence in sentences:
        newSentence = []
        sentence = list(tokenize.whitespace(sentence))
        #print sentence
        for token in sentence:
            newSentence.append(token)
        retSentences.append(newSentence)
        #retSentences += [newSentence]
    return list(retSentences)

def loadCorpus(corpus):

    text = open(corpus).read()
    sentences = list(tokenize.blankline(text))
    newSentences = []

    for sentence in sentences:
        #newSentence = []
        tokens = list(tokenize.whitespace(sentence))
        '''
        for token in tokens:
            #print token
            #loc = token.rfind('/')
            #if loc > 0: # can't allow "/TAG" tuples
            newSentence.append(token)
        newSentences += [newSentence]'''
        newSentences.append(tokens)
    return newSentences

def doTag(model, sentences):

    taggedSentences = []
    for sentence in sentences:
        #print sentence
        pts = model.best_path(sentence)
        #pts = model.best_path(['foo', 'bar'])
        taggedSentence = []
        for token, tag in zip(sentence, pts):
            taggedSentence.append((token[0], tag))
        taggedSentences.append(taggedSentence)
    return taggedSentences

#read all tags from tagset file
tagfile = open('bangla-tagset.txt','r')
tags = [0]*54

i=0
```

```
for t in tagfile:
    tags[i] = t
    i+=1
tagfile.close()
print 'done reading tags'
#done reading all tags

#read all symbols/word types from
lexicon
lexfile      =      open('bangla-
lexicon.txt','r')
types = []

i=0
for word in lexfile:
    #types[i] = word
    #i+=1
    types += [word]
lexfile.close()
print types[0]
print 'done reading symbols'
#done reading all word types

#read all the training sentences
from corpus
sents    =    loadCorpus('bangla-
corpus-cleaned.txt')
print sents[0][0]
print 'done reading corpus'
#done reading all sentences into
a list

#begin training on sentences
#print tags
print len(types)
types = []
for s in sents:
    for w in s:
        #if not w in types:
            types += [w]
             #print w + " was not
found"
trainer                          =
nltk.tag.HiddenMarkovModelTrainer
(tags, types)
#trainer                         =
nltk.tag.HiddenMarkovModelTrainer
(['foo'], ['foo'])
print 'done trainer generation'
print(len(sents))
tagger                           =
trainer.train_unsupervised([sents
])
print 'done training'

#test tagger trained above
untagged                         =
loadUntagged('test.txt')
print 'Tagging...'
taggedOutput  =    doTag(tagger,
untagged)

taggedFile                       =
open('Tagged_bangla_hmm.txt',
'w')
for sentence in taggedOutput:
    for (word, tag) in sentence:
          taggedFile.write(word +
'/'  + tag + ' ')
    taggedFile.write('\n\n')
taggedFile.close()
print 'Finished Tagging'
```

A3 Code to count number of word types and tokens

```
file    =    open('bangla-corpus-
cleaned.txt','r')

tokencount=0
typecount=0

wordmap = {}

for line in file:
    tokens = line.split()
    for each in tokens:
        if wordmap.has_key(each):
wordmap[each]+=1
        else: wordmap[each]=1

for entry in wordmap:
    tokencount+=wordmap[entry]

print len(wordmap)
print tokencount
```

A4 Code to clean a tagged corpus

```
file      =      open('D:\\Fall
2008\\CS503\\project\\Tagged-
Corpus-and-tag-set\\bangla-
corpus-tagged.txt', 'r')
out    =    open('bangla-corpus-
cleaned.txt', 'w')

for line in file:
    tokens = line.split()
    for item in tokens:
        word = item.split("/")
        out.write(word[0])
        out.write(" ")
```

```
    out.write("\n")
```

A5 Code to learn emission probabilities

```
#include<cstdio>
#include<cstring>
using namespace std;

int main()

{
    FILE *fp1,*fp2,*fp3;
    freopen("dummy.txt","w",stdou
t);
                              char
tag1[100],tag2[100],word[500];
    int f;
            fp1   =   fopen("toy-
lexicon.txt","r");
    fp3 = fopen("emit.txt","w");
             while(fscanf(fp1,"%s
%d",tag1,&f)!=EOF)
    {
     //printf("%s\n",tag1);
     fp2 = fopen("test.txt","r");
             while(fscanf(fp2,"%s
%s",tag2,word)!=EOF)
     {
                       printf("%s
%s\n",tag2,word);
      if (strcmp(tag1,tag2)==0)
      {
               fprintf(fp3,"%s  %s
%lf\n",tag2,word,1.0/f);
      }
     }
     fclose(fp2);
    }
    fclose(fp1);
    fclose(fp3);
    return 0;
}
```

Appendix B

The tagged corpus is not available online, but can be provided by this author if needed for research purposes.
The link to the untagged corpus is given below:
http://faculty.bracu.ac.bd/~fahim/CRBLP/CorpusAnalysisIntput.zip