Natural Language Processing using Hidden Markov Model

Anand P. Popat
Department of Computer Science
University at Buffalo
Buffalo, NY 14226
apopat@buffalo.edu

Chirag A. Yeole
Department of Computer Science
University at Buffalo
Buffalo, NY 14226
chiragar@buffalo.edu

Abstract

A big part of Natural Language Processing is the Part-Of-Speech tagging. NLP deals with the human speech and its context and to do that, the Artificial Intelligence has to first understand the expression, which is done by Part-Of-Speech tagging. Human speech is complex with each language having different connotations for conveying emotions and words that can have different meaning depending on the context. Every language has this ambiguity and it might be trivial to humans but computers have a hard time figuring out the different use of the words based on context and emotions. Hidden Markov Model is a great way to address this issue as it helps find relationships between the words of a sentence and know the context of the sentence which makes it possible for the computer to know the actual use of the word in a sentence. We use supervised as well as unsupervised learning using Hidden Markov Model to address the issue of POS tagging and in the end we also discuss another way of addressing this issue using Deep Learning Neural Networks.

1 Introduction

Part-Of-Speech tagging is the process of tagging each word in the sentence to its grammatical equivalent like noun, adverb, pronoun or verb etc and their tense forms to understand the construct of a sentence and the meaning behind the used words. Formation of a sentence is a complex process and the science of understanding the human speech, and using it as a tool to do something useful is called Natural Language Processing. NLP has many applications like Personal Assistant like Siri, Cortana, Google Now etc., in spam detection, Question Answering and Summarization. The advancement in these field is tremendous and Silicon Valley is betting heavily on personal assistants like Siri.

A simple example of the complexity of human speech is given below:

I am going to run a race today.

The ‘race’ in this sentence can have two grammatical representations, as Verb or as Noun. Depending of the context of the sentence, the word will take its form. In this instance, its a noun. The sentence given here is fairly simple in construction meaning that only the word ‘race’ has different forms and by knowing other words i.e construction of the rest of the sentence, we can make a prediction. Now suppose, there are more than one word which can have more than one forms. The complexity of decoding such a sentence rises exponentially. And on top of that, the example is given in English which is arguably, one of the easiest languages on Earth. For more complex languages like Mandarin or German, the process becomes a lot more difficult. From this we are certain that for the prediction of the grammatical representation of the word, we have to depend on the construction of the rest of the sentence and we do not have a specific path to our answer as in Bayesian Nets. The path can be anything and that makes Hidden Markov Models the perfect candidate for use. There can be any number of hidden layers meaning different paths and using HMMs we can predict the most probable one.

2 Dataset

We are using two different datasets for this project. First, 'Deep NLP' dataset from kaggle.com, which has two sheets, the first one has responses given by the humans to chatbots and a flagged field which shows if that particular response allows the user to continue to talk to chatbot, if not, the chatbot searches on google. The second sheet of this database contains examples of resumes. We are concerned with only the text field of the first sheet which contains the responses of humans to the chatbots. This dataset is chosen as it is directly related to the interaction between humans and AI. This dataset is also ideal for the Convolutional Neural Networks performed in the last section.

The other dataset is the classic 'The Penn Treebank' which is a text corpus that denotes the grammatical and syntactic meaning of the sentence. The dataset is a tagged dataset and is useful in training the Hidden Markov Model using Supervised Learning. This dataset is ideal for training a POS tagger and is used by all the major POS taggers available. The database is available in many languages but for this project, we are concerned with just English language. It is one of the most comprehensible database for English language with a large amount of data.

3 Hidden Markov Model

There are two ways to go about making a Probabilistic Graphical Mode. The first one is to create a directed graph, which is similar to making a Bayesian Network. Directed graphs are useful if we know the relationships between the states. By finding the independencies in the graph, the computation reduces which makes it possible to infer on datasets with many variables. On the other hand, when we don't know the relationship between the states or if there is no proper relationships forming between the states, in that case, undirected graph should be used. In this project, we have a dataset which has states not directly related to each other. So we cannot obtain a pattern just by knowing the states. To elaborate, in different conditions, one state's relationship with the other state changes. This can also be termed as 'hidden states'. Hidden Markov Model is used for that exact purpose. Markov Models can be regarded as the most primary Bayesian Networks in which all the states are connected to all the other states.

For Natural Language Processing, the states of the Hidden Markov Model are the grammatical representations of the word. For e.g, Noun, Verb, Pronoun etc. Each of those grammatical semantics can be connected in any fashion. Language is a complex thing and hence there is no concrete relationship between the words of a sentence. Hence, there can be many hidden states and thats why we use Hidden Markov Model for this.

The states, in our Markov Model will be the grammatical representations of the words. There are a total of 36 grammatical representations according to NLTK, a famous POS Tagging library. So there will be a total of 36 states in our Hidden Markov Model. This means that each word of a sentence will serve as a variable. If a sentence has more than 20 words in it, there will be 20 states in our model with which it is concerned. The Hidden Markov Model is shown in fig.1

The names of the states are as follows:

CC-Coordinating conjunction; CD-Cardinal number; DT-Determiner; EX-Existential there; FW-Foreign word; IN-Preposition or subordinating conjunction; JJ-Adjective; JJR-Adjective; comparative; JJS-Adjective; superlative; LS-List item marker; MD-Modal; NN-Noun; singular or mass; NNS-Noun; plural; NNP-Proper noun; singular; NNPS-Proper noun; plural; PDT-Predeterminer; POS-Possessive ending; PRP-Personal pronoun; PRP$-Possessive pronoun; RB-Adverb; RBR-Adverb; comparative; RBS-Adverb; superlative; RP-Particle; SYM-Symbol; TO-to; UH-Interjection; VB-Verb; base form; VBD-Verb; past tense; VBG-Verb; gerund or present participle; VBN-Verb; past participle; VBP-Verb; non-3rd person singular present; VBZ-Verb; 3rd person singular present; WDT-Wh-determiner; WP-Wh-pronoun; WP$-Possessive wh-pronoun; WRB-Wh-adverb.

The graph as seen is undirected meaning that each node in the graph is connected to every

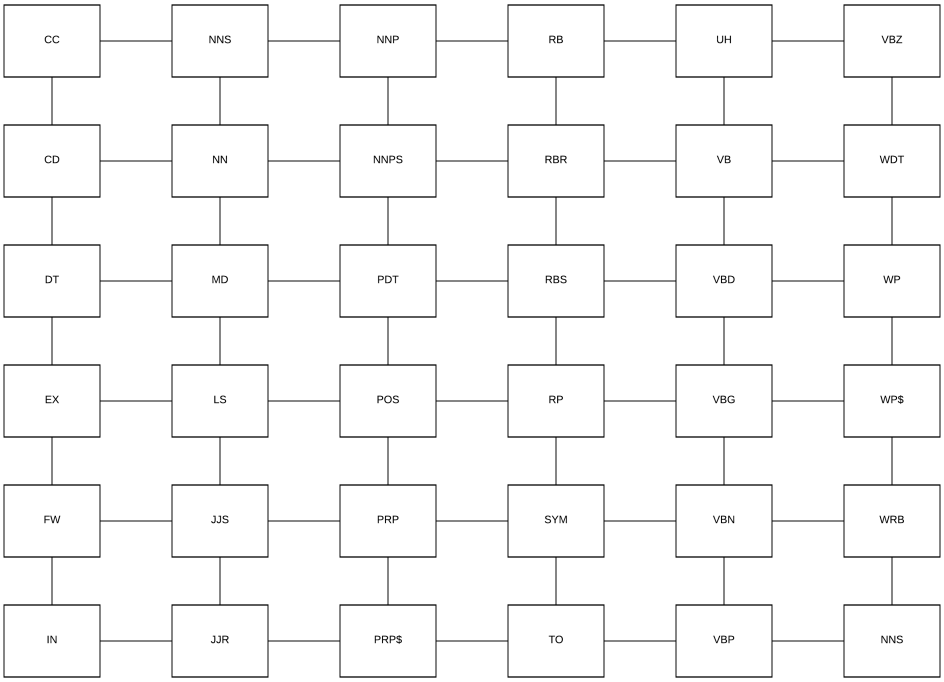


Figure 1: Hidden Markov Model

other node. The algorithms that are used in later sections are to predict the hidden paths in this graph.

4 Problem Statement

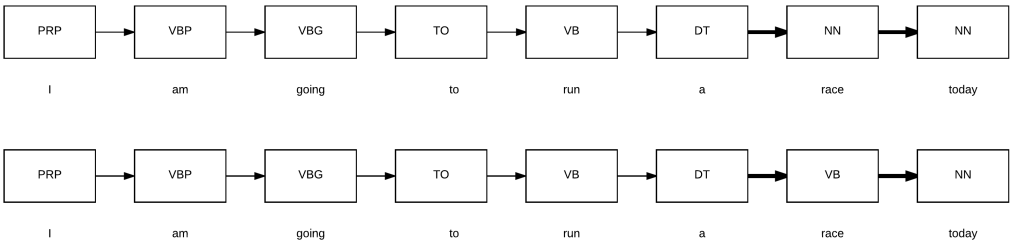
Had it just been one grammatical representation for a word, then there would be no need for using a Hidden Markov Model. But there can be more than one grammatical representation of a word in a sentence depending on its context. An example of that is given here:

Lets take the same statement used in the introduction,

“I am going to run a race today”

The grammatical representations of the each word in the sentence are shown in fig.2.

‘I’ is PRP; ‘am’ is VBP; ‘going’ is VBG; ‘to’ is TO; ‘run’ is VB; ‘a’ is DT; ‘race’ can be NN or VB and ‘today’ is NN. Here ‘race’ is the word which can be used as Noun or as Verb. To know which form does ‘race’ take in this example, we need to know the sentence formation and context and based on that we can decide whether ‘race’ is a noun or a verb in this sentence. Precisely, for this reason we use Hidden Markov Model to build connections



between the states and depending on the past values, and learning from them, we can predict the grammatical representation of a particular word in a sentence.

Figure 2: Problem Statement

Using Hidden Markov Model, we have to deal with 3 famous problems of HMM,

1) Stage-1 (Evaluation): In this stage, we have the sequence of words in the sentence that we want to tag, as the observation sequence while the Hidden Markov Model consists of 36 stages i.e grammatical representations that it can take as shown in fig.1. We have to find out the likelihood of an observation sequence (i.e. the sequence of words in a given query) in our Hidden Markov Model.
2) Stage-2 (Decoding): In this stage, we have the sequence of words in the sentence as the observation sequence and Hidden Markov Model made up of 36 states each representing a grammatical form. We have to find out that given an observation sequence and this Hidden Markov Model, what is the best hidden path sequence in our HMM.
3) Stage-3 (Learning): In this stage, from the previously learned best hidden path sequence, we have to predict the Hidden Markov Parameters for a newly observed sequence of words and stages.

5 Inference

Inference in this project is to find the most probable path of nodes in the Hidden Markov Model for a given structure of sentence.

5.1 Viterbi Algorithm

For stage-2 (Decoding), we have a sequence of words serving as a sequence of observation. To find the relationship between the states of Hidden Markov Model, we should know th e most probable path of states, which is what this stage does. To find the most probable path, the most basic algorithm that we can use is to compute interaction of each state with all the other states. So for a sequence of N words in a sentence, if each word in that sequence has M possible grammatical representations i.e states in our Hidden Markov Model, then all the possible combinations between the paths will take O(N^M) time to compute which is polynomial time. Taking the best out of all these paths, we can use that to predict for newly observed sequences.

Instead of using this algorithm, we can use Viterbi algorithm which is more efficient. It is a dynamic programming algorithm used to find most probable hidden paths in the Hidden Markov Model. It can be used in Linguistics, BioInformatics, Speech Recognition and of-course, Natural Language Processing. The algorithm is very similar to Belief Propagation algorithm used in Bayesian Network inference, in that, both use message passing.

Viterbi Algorithm makes use of the multiplicity of the probability theory meaning that the probability of the most probable path will be the multiplication of the individual probabilities of the combinations of all the previous paths. Hence, if we find a path with the most probability, for later computations which are based on this path, don't need to recalculate all the probabilities again. In a sense it is message passing.

For example, in the statement, “I am going to run a race today”, suppose “I” and “am” being “PRP” and “VBP” respectively is 0.9 in comparison to being “PRP” and “VBZ” which is 0.1, then when the probability is calculated for the third word in the sentence which is “going” in “I am going”, we don’t need to calculate all the possible probabilities for “I am” as we know it is highest for “PRP” and “VBP”. As it is just multiplication, this can be applied in all scenarios. This process is repeated for all the words in sequences. Suppose there are N words in the observation sequence and M possible grammatical representations of each word, then the running time of the computation using Viterbi Algorithm reduces drastically to O(NM^2) in comparison to O(N^M).

We are using hmmlearn library and nltk supervised training library both of which make use of Viterbi Algorithm. Gaussian Hidden Markov library of hmmlearn is used for this purpose and is imported as follows:

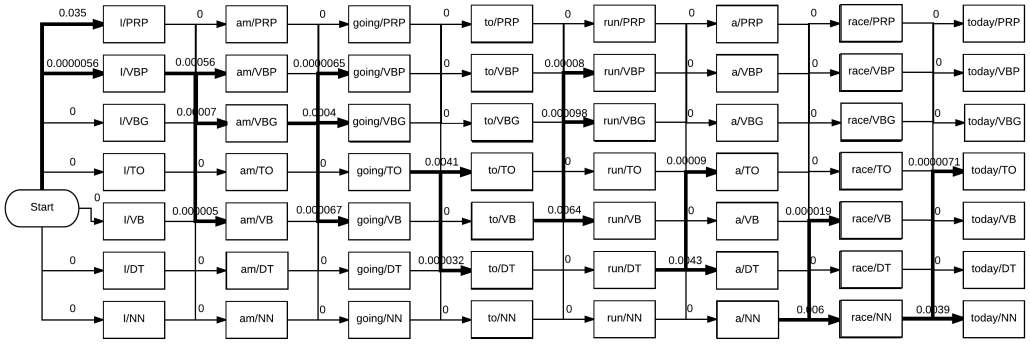
```
from hmmlearn.hmm import GaussianHMM
```

To perform decoding using Viterbi Algorithm, the following method of hmmlearn is used:

```
model = GaussianHMM(n_components=20,
covariance_type="diag", n_iter=1000).fit(X,len1)
```

where X=list of observed sequences, len1=list of the length of each observed sequence in the list X

Using nltk, the Viterbi Algorithm can be used to decode using the following code:




```
tag1 = myhmm.train_supervised(train)
```

This method includes decoding as well as learning which is discussed in the next section

Figure 3: Viterbi Algorithm

5.2 Forward-Backward Algorithm

For the learning part of the Hidden Markov Model, Forward-Backward algorithm can be used. The Forward-Backward algorithm can give the most probable state out of all the given states. In POS tagging, suppose a given word has M possible grammatical representations, then using Forward-Backward algorithm, we can know the most probable state out of the M states. This is combination with the Viterbi Algorithm gives us the sequence of the most probable states or the “hidden path” which is the prediction we want to make. This algorithm is similar to Viterbi algorithm and Belief Propagation algorithm in

The Forward-Backward algorithm, as the name suggests, has two stages:

1) Forward Probabilities computation

2) Backward Probabilities computation

sequence will be calculated incrementally, meaning that for N number of observations from a given sequence of observations, the probability of most probable states of those N observations will be calculated. Then the same will be done for N+1 number of observations making it go in the forward direction, hence the name forward in forward-backward algorithm. As it is stated, the calculation of probabilities at each stage depends on the probabilities calculated in the previous stages. Hence this step takes into account the past in contrast to the next step.

To understand this better, lets take the example statement, “I am going to run a race today”. Suppose we know that “I” has PRP as the most probable state and “am” has VBP as the most probable state then based on this knowledge, we want to predict the state of the next observation which is “going”. The probabilities of each of these words will be calculated in order to find the probability of the new observation. Hence it takes into account past, meaning that depending on the words before “going”, the most probable state of “going” may change.

In the second state, just like the first stage, the probabilities of the states in the sequence will be calculated incrementally but this time the states will be the “future” states. Meaning that with more information that we get, all the previous stages will again be evaluated. So for N number of observations from a given sequence of observations, the probability of N states will be calculated using forward algorithm described above. Now when N+1th observations comes along, the probability of all N states will be recalculated because of the N+1th state coming into picture. So the probability calculation will go in the backward direction and hence, the name backward algorithm. This will be done for all the number of observations, N, N+1, N+2 and so on. We can say that in forward algorithm, the message passing was in forward direction while in backward algorithm, the message passing is in backward direction.

Lets take the example statement, “I am going to run a race today”. Suppose we have the most probable states for “I”, “am” and “going”. Now when the next observation in the sequence, i.e. “to” comes in, the message will be passed in the backward direction, and the probability of “going” with “to”, “am” and “going” given “to” and so on will be recalculated.

Both these stages together make the forward-backward algorithm. It is used in predicting the most optimal sequence of hidden states once a newly formed obseravtion sequence is given.

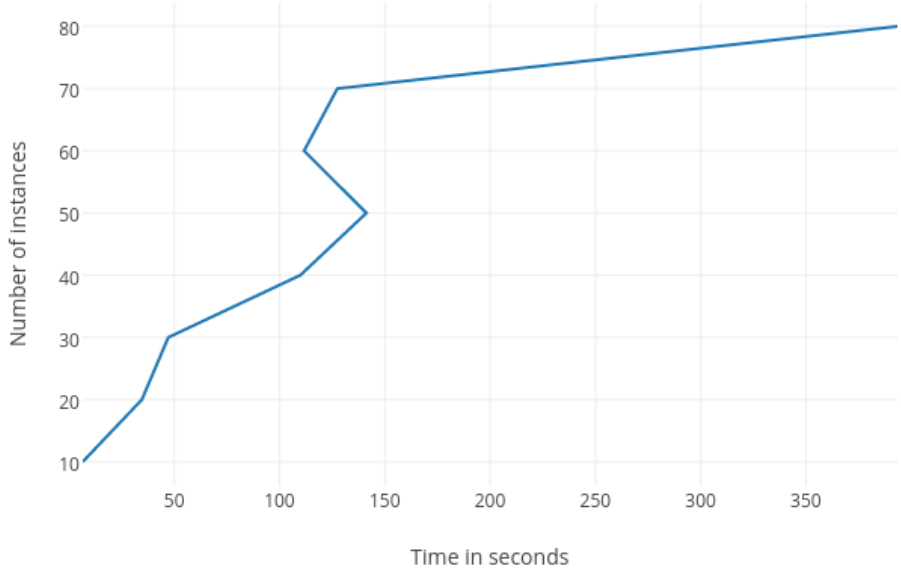
We use hmmlearn for this prediction, by importing the library as follows:

```
from hmmlearn.hmm import GaussianHMM
```

and is predicted using the following method of hmmlearn to use this library:

```
Z=model.predict(X)
```

where, model=the Gaussian HMM model formed using Viterbi Algorithm and X=list of observed sequences.



Running time of the computation using **Forward-Backward** Algorithm is O(NM^2)

Figure 4: Time Complexity for Forward-Backward Algorithm

5.3 Pos tagging using neural network

Neural Networks consists of multiple layers of nodes as hidden layers connected through a network. In tagging, the probabilities of each word along with all the other words in the observation sequence are given to the neural network as input units. The network using several runs and backward propogation, learns which which states represent the words in the output sequence in the most probable manner and discard the probability of all the other states.

When the input is provided, the word in the observation sequence that is to be tagged and the words that follow that word are together fed to the network along with the words that are already tagged. While the states of the current node and the future nodes are yet to be calculated, the tagging is already done for the previous nodes and they don't need to be repeated. So for every word in the observation sequence that is yet to be tagged, the states of the previous nodes are fixed and can be used with some weightage to impact the calculation of the current and future nodes without recalculating the probabilities of the previous nodes.

In this project, we have used SyntaxNet, a neural-network Natural Language Processing framework that uses TensorFlow. To do the tagging, SyntaxNet processes the observed sequence of the word one by one and depending on the position of the word in the observed sequence, calculated the features for the word and forwards that to the neural networks that computes the probability of each state for that particular word in the observed sequence.

5.3 Performance of HMM tagger vs Network tagger

Categories	Accuracy	Time (seconds)
Supervised(HMM)	0.9023	393.313
Unsupervised(HMM)	0.8867	432.128
Neural Network	0.944	107.675

Table 1: Accuracy and Time comparison for deep learning dataset having 80 instances

6 Conclusion

Probabilitic Graphical Models are used to visualize the states and depending on the graph structure, we can reduce the amount of computation required in a dataset with a lot of variables using Bayesian Network. While on the other hand, if there is no relation between the nodes, i.e. an undirected graph, then we can make a Hidden Markov Model which will give us the best probable path of states for an observed state of sequence. Viterbi and Forward-Backward algorithms are used in POS-tagging as they reduce the computation time by a large margin. Neural Networks can also be used for this purpose in which they take a different approach to the whole problem with results slightly better than Hidden Markov Models.

```
############## Start Probabilities ######################
[  0.00000000e+000   5.84810321e-032   0.00000000e+000   5.32378804e-211
   9.18412189e-215   0.00000000e+000   3.15965083e-002   0.00000000e+000
   1.89608907e-002   0.00000000e+000   0.00000000e+000   3.11797136e-133
   4.77373852e-002   0.00000000e+000   0.00000000e+000   0.00000000e+000
   0.00000000e+000   0.00000000e+000   0.00000000e+000   0.00000000e+000
   7.20573702e-001   0.00000000e+000   3.10466628e-002   0.00000000e+000
   1.49420809e-002   0.00000000e+000   1.55912993e-002   8.74225867e-002
   0.00000000e+000   9.06787215e-305   6.63796215e-279   0.00000000e+000
   3.21288839e-002   0.00000000e+000   0.00000000e+000   9.06787215e-305]

################ Optimal Path ##########################
[33 24  4 ...,  8 15 20]

################ Transition_Matrix ######################
[[  0.00000000e+000   0.00000000e+000   0.00000000e+000 ...,
    7.30019418e-139   4.79395747e-007   0.00000000e+000]
 [  0.00000000e+000   0.00000000e+000   0.00000000e+000 ...,
    8.55864871e-031   7.72867205e-008   1.84192613e-319]
 [  0.00000000e+000   0.00000000e+000   0.00000000e+000 ...,
    0.00000000e+000   1.19474170e-237   0.00000000e+000]
 ...,
 [  9.76462116e-165   0.00000000e+000   0.00000000e+000 ...,
    0.00000000e+000   0.00000000e+000   1.40547453e-275]
 [  1.23372218e-060   2.62106628e-285   0.00000000e+000 ...,
    0.00000000e+000   0.00000000e+000   2.76758973e-195]
 [  0.00000000e+000   0.00000000e+000   9.95065435e-152 ...,
    2.70409297e-266   0.00000000e+000   0.00000000e+000]]

################### Mean and Variance #################
Hidden Layer 0:
Mean: [ 29.]
Variance: [[ 0.00070845]]
#########################################################
Hidden Layer 1:
Mean: [ 27.]
Variance: [[ 0.00027503]]
#########################################################
Hidden Layer 2:
Mean: [ 27.]
Variance: [[  8.44923956e-05]]
#########################################################
Hidden Layer 3:
Mean: [ 10.94859835]
Variance: [[ 48.80079025]]
#########################################################
Hidden Layer 4:
Mean: [ 31.]
Variance: [[ 0.00012145]]
#########################################################
Hidden Layer 5:
Mean: [ 29.]
Variance: [[ 0.00062921]]
```

Hidden Layer 5:
Mean: [29.]
Variance: [[0.00062921]]
##
Hidden Layer 6:
Mean: [20.]
Variance: [[9.19340852e-05]]
##
Hidden Layer 7:
Mean: [28.]
Variance: [[0.00014544]]
##
Hidden Layer 8:
Mean: [6.]
Variance: [[3.57891797e-05]]
##
Hidden Layer 9:
Mean: [23.]
Variance: [[0.00062492]]
##
Hidden Layer 10:
Mean: [19.45534244]
Variance: [[104.9483473]]
…
####### IT WILL PRINT MEAN AND VARIANCE FOR 36 HIDDEN LAYERS###
…

########### Supervised learning ########################

[[('When', u'WRB'), ('my', u'PRP$'), ('friend', u'NN'), ('needed', u'VBD'), ('help', u'VB'), ('when', u'WRB'), ('she', u'PRP'), ('was', u'VBD'), ('going', u'VBG'), ('through', u'IN'), ('some', u'DT'), ('tough', u'JJ'), ('stuff.', u'NNP'), ('I', u'NNP'), ('knew', u'NNP'), ('that', u'NNP'), ('she', u'NNP'), ('needed', u'NNP'), ('it', u'NNP'), ('but', u'NNP'), ('she', u'NNP'), ("wouldn't", u'NNP'), ('admit', u'NNP'), ('it.', u'NNP'), ('So', u'NNP'), ('I', u'NNP'), ('brought', u'NNP'), ('it', u'NNP'), ('up', u'NNP'), ('and', u'NNP'), ('initiated', u'NNP'), ('it', u'NNP'), ('and', u'NNP'), ('we', u'NNP'), ('talked', u'NNP'), ('about', u'NNP'), ('it', u'NNP'), ('and', u'NNP'), ('it', u'NNP'), ('really', u'NNP'), ('helped', u'NNP'), ('her.', u'NNP')], [('My', u'PRP$'), ('sister', u'NNP'), ('has', u'NNP'), ('some', u'NNP'), ('pretty', u'NNP'), ('severe', u'NNP'), ('issues', u'NNP'), ('with', u'NNP'), ('her', u'NNP'), ('mental', u'NNP'), ('health.', u'NNP'), ('I', u'NNP'), ('try', u'NNP'), ('to', u'NNP'), ('be', u'NNP'), ('there', u'NNP'), ('for', u'NNP'), ('her', u'NNP'), ('when', u'NNP'), ('she', u'NNP'), ('needs', u'NNP'), ('it', u'NNP'), ('and', u'NNP'), ("I'd", u'NNP'), ('like', u'NNP'), ('to', u'NNP'), ('think', u'NNP'), ('its', u'NNP'), ('helped', u'NNP'), ('her.', u'NNP')], [('Helped', u'NNP'), ('friends', u'NNP'), ('through', u'NNP'), ('stuff.', u'NNP')], [('I', u'PRP'), ('helped', u'VBD'), ('encourage', u'VB'), ('a', u'DT'), ('friend', u'NN'), ('to', u'TO'), ('go', u'VB'), ('to', u'TO'), ('therapy,', u'NNP'), ('and', u'NNP'), ('often', u'NNP'), ('listened', u'NNP'), ('to', u'NNP'), ('their', u'NNP'), ('problem', u'NNP'), ('to', u'NNP'), ('let', u'NNP'), ('them', u'NNP'), ('know', u'NNP'), ('there', u'NNP'), ('was', u'NNP'), ('someone', u'NNP'), ('there', u'NNP'), ('for', u'NNP'), ('them.', u'NNP')], [('I', u'NNP'), ("haven't", u'NNP'), ('really', u'NNP'), ('met', u'NNP'), ('anyone', u'NNP'), ('dealing', u'NNP'), ('with', u'NNP'), ('attention', u'NNP'), ('issues.', u'NNP'), ('So', u'NNP'), ('I', u'NNP'), ("haven't", u'NNP'), ('had', u'NNP'), ('a', u'NNP'), ('chance', u'NNP'), ('to', u'NNP'), ('help', u'NNP'), ('unfortunately.', u'NNP')], [('there', u'EX'), ('are', u'VBP'), ('too', u'RB'), ('many', u'JJ'), ('to', u'TO')

########## Unsupervised learning ######################
[('I', 4), ('just', 3), ('try', 3), ('to', 2), ('be', 2), ('kind', 4), ('and', 0), ('helpful', 1)]

References

[1] https://pdfs.semanticscholar.org/1b4e/04381ddd2afab1660437931cd62468370a98.pdf

[2] http://consilr.info.uaic.ro/2016/Consilr_2016.pdf

[3] http://www3.cs.stonybrook.edu/~ychoi/cse628/lecture/06-hmm.pdf

[4] https://www.tensorflow.org/versions/r0.11/tutorials/syntaxnet/

[5] https://hmmlearn.readthedocs.io/en/latest/tutorial.html#building-hmm-and-generating-samples

[6] http://www.nltk.org/book/ch05.html