CS-7650-A

Natural Language Processing

09/21/2010

ASSIGNMENT 1

What's in a sentence?

By,

Preetam B. Joshi William Mooney

(GT ID# 902631125) (GT ID# 902783745)

1. Overview

We approached this problem with a machine learning perspective. Overall, the idea was to first parse and chunk a given sentence into disambiguated entities such as VB, NPH, EXT etc. These tagged entities were then used to create a small dataset of 156 sentences of the following form:

```
NPH/WHO RB/still VOB/WHAT NPH/WHEN ./.

NPH/WHO PRP/WHAT NPH/WHEN ./.

NPH/WHO EXT/was VOB/WHAT PRP/under NPH/an LS/1990 NPH/WHY ./.

NPH/It POS/'s NPH/good TO/to VOB/WHAT CD/three NPH/WHEN ./.

NPH/WHO DOZ/does NPH/WHAT VOB/programming ./.

NPH/WHO EXT/is VOB/WHAT PRP/for DT/another NPH/job ./.
```

We used stemming to remove words which did not occur from a hidden state. For example, words like for, the etc. were removed.

This kind of a dataset was then used to train (supervised) a Hidden Markov Model (HMM). The intuition behind this was to try to encode the sequence of generation of the observation symbols – VOB, NPH etc. from hidden states i.e., WHO, WHAT, WHEN, WHY, WHERE. We observed good classification accuracies when a novel statement was given to the model to decode a hidden state sequence.

2. Chunking

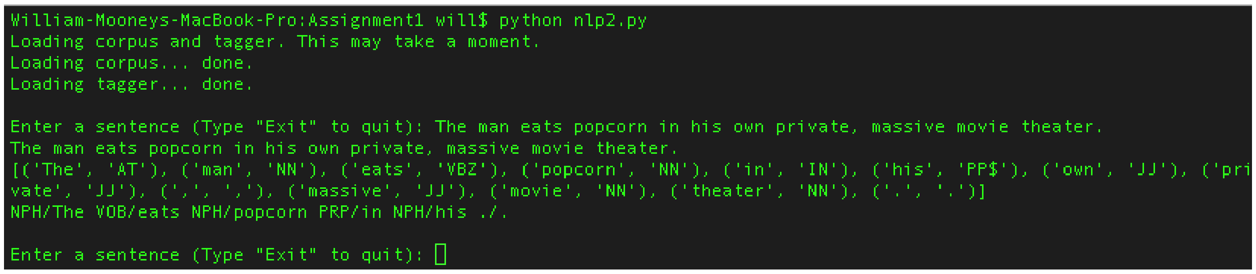
We began our project by deciding on what tools to use. Between the Stanford parser and the Natural Language Toolkit, we chose the latter due to its flexibility and Python wrappers. The NLTK comes with many corpuses available to download, which provided us with a plethora of options when it came to parsing the initial sentences. We decided to use the Brown corpus, as it seemed to have been built on a wide array of source material.

We created a Unigram tagger using the Brown corpus. This would tag words with the part of speech that was most common in the Brown corpus source material. If any material remained untagged (signified with a tag of “None” in Python), we then used NLTK’s built in part of speech tagger to label it. Ambiguities, such as whether “fly” was being used as a verb or a noun, were resolved by looking at surrounding labels and going through a series of conditional statements.

Once we had our sentences tagged with the correct parts of speech (or as close as we could get to the correct parts of speech), we wanted to group them into separate phrases that would provide different information about the sentence. For example, we would group an article, an adjective, and a noun together as a noun phrase, which is often the subject or object of a sentence. We did this by using the built in regular expressions parser and a set of custom rules we formulated based on example sentences. We wound up with four custom labels: one for noun phrases, one for verbs and objects they acted on, one specifically for variations of the verb “to be”, and one for prepositional phrases.

After tagging our sentences and chunking them into different phrases, we needed to put them in a format so that they could be trained. We did this by running our program to this point on the sentence and going through and replacing the parts of the sentence that corresponded to WHO, WHAT, WHERE, WHEN, and WHY with the appropriate tags.

For this first portion of our process, we refer to the example sentence “The man eats popcorn in his own private, massive movie theater.” In the screen shot below, we can see the stages the sentence goes through. First is the original sentence, then a list of tuples that contain each word and punctuation mark with their respective tag. Finally, we see that the sentence has been reduced to six tags in the following sequence: noun phrase, verb, noun phrase, preposition, noun phrase, end punctuation. This makes extracting information easy, as the first noun phrase is WHO, the verb and following noun phrase is WHAT, and the preposition and concluding noun phrase are WHERE.



```
William-Mooneys-MacBook-Pro:Assignment1 will$ python nlp2.py
Loading corpus and tagger. This may take a moment.
Loading corpus... done.
Loading tagger... done.

Enter a sentence (Type "Exit" to quit): The man eats popcorn in his own private, massive movie theater.
The man eats popcorn in his own private, massive movie theater.
[('The', 'AT'), ('man', 'NN'), ('eats', 'VBZ'), ('popcorn', 'NN'), ('in', 'IN'), ('his', 'PP$'), ('own', 'JJ'), ('pri
vate', 'JJ'), (',', ','), ('massive', 'JJ'), ('movie', 'NN'), ('theater', 'NN'), ('.', '.')]
NPH/The VOB/eats NPH/popcorn PRP/in NPH/his ./.

Enter a sentence (Type "Exit" to quit): 
```

Fig.1. *Disambiguated word sequence.*

3. Hidden Markov Model:

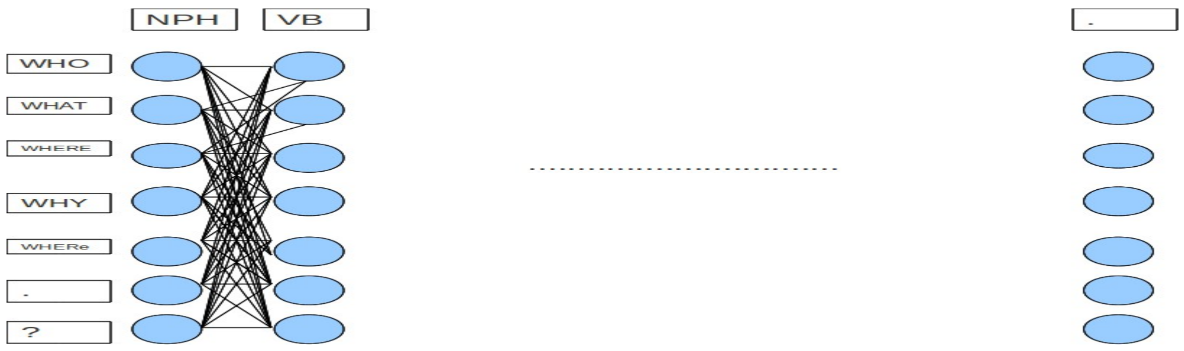


Fig.2. Hidden Markov Model depicting hidden states and Observation symbols

Fig.2 shows the graphical representation of our first order HMM. Here are the details:

- We fixed the number of hidden states to 7 namely: WHO, WHERE, WHY, WHEN, WHAT, . , ?

- Each element in the state transition matrix (A) was calculated as follows:

  Normalized(Frequency of transition from a state to that particular state)

- Each element in the Symbol emission probability matrix (B) was calculated as follows:

Normalized(Frequency of that symbol generated by a hidden state)

- Each element in the Initialization Matrix (Pi) was calculated as follows:

Normalized(Frequency of occurrence of a particular hidden state)

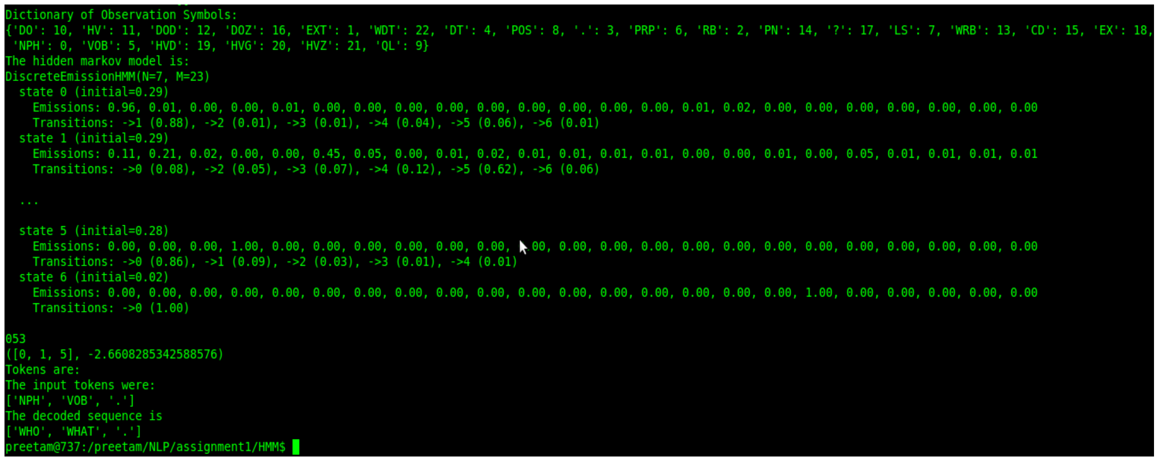
- The Viterbi algorithm was used as the decoding algorithm in our approach. It tries to find the optimum hidden state sequence that was followed to generate the observation sequence.

- We used about 10 sentences from the 18 different categories that was listed on the test set. We made sure not to include the test sentences as part of the training data set.

- The “ghmm” library which is available in python was used to generate a HMM once A, B and Pi were calculated. The viterbi algorithm could also be invoked using “ghmm”.

- The training set was specified in a file and the test data was read from another file. A test sentence was first stemmed and then converted to a string of integers.

- Fig.3 shows the HMM, input sequence(056), input tokens(NPH, VOB, “.”)



```
Dictionary of Observation Symbols:
{'DO': 10, 'HV': 11, 'DOD': 12, 'DOZ': 16, 'EXT': 1, 'WDT': 22, 'DT': 4, 'POS': 8, '.': 3, 'PRP': 6, 'RB': 2, 'PN': 14, '?': 17, 'LS': 7, 'WRB': 13, 'CD': 15, 'EX': 18,
 'NPH': 0, 'VOB': 5, 'HVD': 19, 'HVG': 20, 'HVZ': 21, 'QL': 9}
The hidden markov model is:
DiscreteEmissionHMM(N=7, M=23)
  state 0 (initial=0.29)
    Emissions: 0.96, 0.01, 0.00, 0.00, 0.01, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.01, 0.02, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00
    Transitions: ->1 (0.88), ->2 (0.01), ->3 (0.01), ->4 (0.04), ->5 (0.06), ->6 (0.01)
  state 1 (initial=0.29)
    Emissions: 0.11, 0.21, 0.02, 0.00, 0.00, 0.45, 0.05, 0.00, 0.01, 0.02, 0.01, 0.01, 0.01, 0.01, 0.00, 0.00, 0.01, 0.00, 0.05, 0.01, 0.01, 0.01, 0.01
    Transitions: ->0 (0.08), ->2 (0.05), ->3 (0.07), ->4 (0.12), ->5 (0.62), ->6 (0.06)

  ...

  state 5 (initial=0.28)
    Emissions: 0.00, 0.00, 0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00
    Transitions: ->0 (0.86), ->1 (0.09), ->2 (0.03), ->3 (0.01), ->4 (0.01)
  state 6 (initial=0.02)
    Emissions: 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 1.00, 0.00, 0.00, 0.00, 0.00, 0.00
    Transitions: ->0 (1.00)

053
([0, 1, 5], -2.6608285342588576)
Tokens are:
The input tokens were:
['NPH', 'VOB', '.']
The decoded sequence is
['WHO', 'WHAT', '.']
preetam@737:/preetam/NLP/assignment1/HMM$
```

Fig.3. HMM, input sequence, input tokens and result

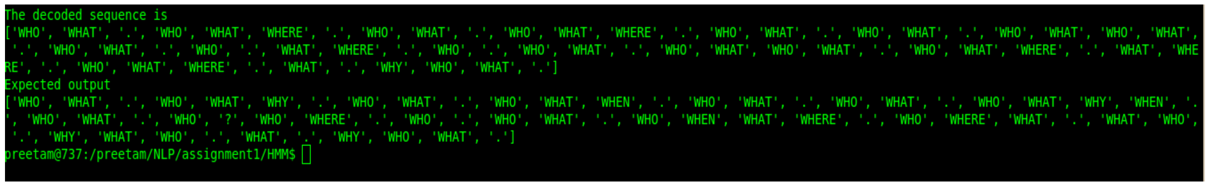
4. Observations

- The HMM was able to identify novel sentences and tag them appropriately.

- A supervised, statistical training method described above was able to give better results than the Baum-Welch algorithm.

- Long length observation symbol sequences give an error due to a problem with the ghmm Viterbi algorithm implementation.

- Fig. 4. Shows the results of testing on the provided list of 18 sentences. The decoded sequence is the result of applying the Viterbi algorithm to the test set. As observed from the figure, there are certain mis-classifications such as an expected “WHY” was classified as a “WHERE”. This is due to the Local Maxima problems with HMMs. The solution is to further refine the representation of the dataset to achieve better classification accuracies.



```
The decoded sequence is
['WHO', 'WHAT', '.', 'WHO', 'WHAT', 'WHERE', '.', 'WHO', 'WHAT', '.', 'WHO', 'WHAT', 'WHERE', '.', 'WHO', 'WHAT', '.', 'WHO', 'WHAT', '.', 'WHO', 'WHAT', 'WHO', 'WHAT',
 '.', 'WHO', 'WHAT', '.', 'WHO', '.', 'WHAT', 'WHERE', '.', 'WHO', '.', 'WHO', 'WHAT', '.', 'WHO', 'WHAT', 'WHO', 'WHAT', '.', 'WHO', 'WHAT', 'WHERE', '.', 'WHAT', 'WHE
RE', '.', 'WHO', 'WHAT', 'WHERE', '.', 'WHAT', '.', 'WHY', 'WHO', 'WHAT', '.']
Expected output
['WHO', 'WHAT', '.', 'WHO', 'WHAT', 'WHY', '.', 'WHO', 'WHAT', '.', 'WHO', 'WHAT', 'WHEN', '.', 'WHO', 'WHAT', '.', 'WHO', 'WHAT', '.', 'WHO', 'WHAT', 'WHY', 'WHEN', '.
', 'WHO', 'WHAT', '.', 'WHO', '?', 'WHO', 'WHERE', '.', 'WHO', '.', 'WHO', 'WHAT', '.', 'WHO', 'WHEN', 'WHAT', 'WHERE', '.', 'WHO', 'WHERE', 'WHAT', '.', 'WHAT', 'WHO',
 '.', 'WHY', 'WHAT', 'WHO', '.', 'WHAT', '.', 'WHY', 'WHO', 'WHAT', '.']
preetam@737:/preetam/NLP/assignment1/HMM$ 
```

Fig.4. Result of Analysis: Decoded sequence V/s Expected output

- Test Data Set Representation:

  A sample representation of the test data set appears below:

  NPH/WHO PRP/at NPH/me VOB/WHAT NPH/me ./.

  NPH/WHO EXT/were VOB/WHAT PRP/by NPH/WHY ./.

  NPH/The PRP/of NPH/WHO WPS/who NPH/WHAT ./.

  NPH/WHO HV/WHAT NPH/doubts PRP/about VOB/inviting NPH/him ./.

  NPH/WHO EXT/WHAT NPH/that EX/there EXT/WHY NPH/WHEN ./.

  NPH/WHO EXT/WHAT RB/apparently NPH/an PRP/on NPH/dogs ./.

NOTE: It is important to note that the test data set contains tagged sequences like WHO, WHAT etc. This is to perform the comparison analysis. The dataprep() module in HMMTrainer.py extracts observation symbols after filtering out the symbols which do not contribute to the WHO, WHAT, WHERE etc. tags. This is a proof of concept implementation to prove that HMMs can be used to generate near accurate expected classification. Also, we do NOT use the tags WHO, WHAT, WHERE etc. that appear in the test set, to our advantage, in any of the calculations of the decoding sequence.

5. Conclusion

- A proof of concept implementation of the proposed system was developed. This demonstrates that a first order HMM is a good statistical approach toward the given problem.

- Local Maxima problems with a HMM result in certain mis-classifications.
- A statistical approach can be used to classify documents as WHO, WHERE, WHAT etc.

6. Future Work

- A first order HMM resulted in good results. Intuitively, for this particular problem a second or a third order HMM may be worth exploring.
- The data set representation needs to be refined. We need a better training dataset. Also, more training instances are needed here. Generally, HMMs perform better when large training data sets are used.

****