Cameron Hartmann
11/18/2014
CS470
Parts of Speech Tagging Lab

Time

Paragraph Building: 4 hours
Viterbi Algorithm: 8 hours
Confusion Matrix: 2 hours
Write up: 1.5 hours

Introduction

The Parts of Speech tagging lab has 3 different parts. Transition and emission tables will be used for each of the parts. The first part, Paragraph Building, uses transition and emission tables to generate random paragraphs. The second part uses the Viterbi algorithm to tag a string of test text. The third part uses a confusion matrix to show how accurate the Viterbi algorithm was in tagging the text.

Training Data

To build the transition and emission models, the text from "allTraining.txt" was used. Three different python files were created. One to generate a first order transition table (t1.py). One to generate a second order transition table (t1_2.py). And another to generate the emission table (t2.py). Each of these files saved the corresponding tables as json data to a text document. Portions of the json data can be seen below. The data is a list of lists. Where the sublists contents are as follows: total times seen, state/word, probability of happening.

Unigram data, transitioning from state NN

[[33099, 'IN', 0.2474765600466556], [691, "''", 0.0051665096526251255], [5405, 'CC', 0.040412423549115485], [16348, 'NN'
, 0.12223169291044218], [1189, 'JJ', 0.008889985494893305], [6371, 'VBD', 0.04763506946002123], [15344, ',', 0.114724926
35293766], [5286, 'TO', 0.039522677313714054], [14580, '.', 0.10901260598447804], [2357, 'MD', 0.017622956948245182], [5
845, 'VBZ', 0.043702241562364484], [10386, 'NNS', 0.07765465883091831], [1502, ':', 0.011230242399772704], [2441, 'RB',
0.018251013114410897], [234, '-RRB-', 0.0017495850343187834], [583, 'PRP', 0.004359008867554918], [1045, 'WDT', 0.007813
317781466362], [2937, 'POS', 0.021959535238437036], [1020, 'VBG', 0.00762639630344085], [521, 'VBP', 0.00389544360205165
], [1318, 'NNP', 0.009854500321504943], [213, '-LRB-', 0.001592570992777354], [144, 'JJR', 0.0010766677134269435], [76,
'RP', 0.0005682412931975536], [1435, 'VBN', 0.010729292838664334], [324, 'WRB', 0.002422502355210623], [908, 'DT', 0.006
788988081886561], [797, 'CD', 0.005959056719453292], [320, '``', 0.0023925949187265415], [186, 'VB', 0.00139069579650980
2], [351, 'RBR', 0.002624377551478175], [7, 'FW', 5.233801384714309e-05], [331, 'WP', 0.0024748403690577664], [17, 'EX',
0.0001271066050573475], [36, 'PRP$', 0.0002691669283567359], [24, 'WP$', 0.0001794446189044906], [34, '$', 0.0002542132
10114695], [7, 'JJS', 5.233801384714309e-05], [11, 'RBS', 8.224545033122486e-05], [12, 'NNPS', 8.97223094522453e-05], [2
, '#', 1.4953718242040884e-05], [2, 'PDT', 1.4953718242040884e-05], [7, 'SYM', 5.233801384714309e-05]]

Bigram data, transitioning from state VBG NN

[[189, ',', 0.10166756320602474], [35, 'VBZ', 0.018827326519634213], [192, '.', 0.10328133405056482], [19, 'MD', 0.01022
0548682087143], [566, 'IN', 0.3044647660032275], [34, 'RB', 0.01828940290478752], [49, 'VBD', 0.026358257127487898], [16
, 'POS', 0.008606777837547068], [280, 'NNS', 0.1506186121570737], [2, 'WP', 0.0010758472296933835], [181, 'NN', 0.097364
1742872512], [19, 'VBN', 0.010220548682087143], [16, 'JJ', 0.008606777837547068], [71, 'TO', 0.038192576654115115], [2,
'RP', 0.0010758472296933835], [16, 'VBG', 0.008606777837547068], [92, 'CC', 0.04948897256589564], [19, ':', 0.0102205486
82087143], [2, 'VB', 0.0010758472296933835], [7, 'PRP', 0.0037654653039268424], [9, 'NNP', 0.004841312533620226], [7, 'W
DT', 0.0037654653039268424], [1, 'PRP$', 0.0005379236148466917], [12, 'DT', 0.006455083378160301], [4, '``', 0.002151694
459386767], [3, 'WRB', 0.0016137708445400753], [6, 'VBP', 0.0032275416890801506], [3, 'CD', 0.0016137708445400753], [1,
'FW', 0.0005379236148466917], [3, 'JJR', 0.0016137708445400753], [3, "''", 0.0016137708445400753]]

State to word data, from state PDT

[[1, 'Quite', 0.002680965147453083], [1, 'nary', 0.002680965147453083], [1, 'Many', 0.002680965147453083], [1, 'many', 0
.002680965147453083], [1, 'Both', 0.002680965147453083], [1, 'Half', 0.002680965147453083], [3, 'quite', 0.0080428954423
5925], [6, 'Such', 0.0160857908847185], [16, 'both', 0.04289544235924933], [78, 'such', 0.20911528150134048], [24, 'All'
, 0.064343163538874], [175, 'all', 0.4691689008042895], [65, 'half', 0.1742627345844504]]

Paragraph building

Using the unigram transition data described in the previous section, combined with the state to word data, I was able to build the following paragraphs. Each one is 100 words/symbols in length. As you can see, the second order paragraph does produce better (more like "English") results than the first order.

First Order

by the due U.S. Gressette making a reminder unit Although Permian Journal Wall Olivetti Probable , unable estate builders , plans , manufacturing over June in saying for Israeli-Palestinian yesterday , for an Appalled % in Mr. low period during the building . and liabilities leaving . his neglect activity . a morning , was of my actor From Rep. ago thereby produce increases and market companies . LABOR with Arrow , Issak Michele Maxwell , '' where she will close as a 77-year-old law field % by a 15 two to slow raising purchase of the legal plea in

This paragraph is built using first order data. The algorithm first generates 100 tags. The first tag is a seed value, which is "". The second through 100th tag is found using the unigram data. It will end up being a probabilistic value determined by picking a random value between 0.0 and 1.0. Then as you sum up the probabilities of all possible tags that follow the previous tag, once a value is found that is less than the random value that tag is used. This results in random, but probabilistically accurate tag's used. The same process is then used to take the tag, and then find a word to replace that tag.

Second Order

of the official to Straszheim Carter said , '' or in export When by paper Sears or state-owned , private boat with Goodyear million buy-outs , Fla First from looking blank two-thirds . target Spokesmen -LRB- in the current effect . '' overtly , higher and earlier of 21 1.9 month how mostly the Next demand at latest 1\/4 reforms The Special aim , and the $ billion to one million over $ 225 217.9 . Congress Sherman in Petroleum Boyd 's internal step . `` they complain it woulddiscuss been A manipulation 's outstanding odd national signs ,

This paragraph is built using a similar process as above, but tag generation is done using bigram data.

Tagging

In order to test the tagging algorithm I used the first 25 words from the devtest.txt file. Those words are listed below.

The_DT economy_NN 's_POS temperature_NN will_MD be_VB taken_VBN from_IN several_JJ vantage_NN points_NNS this_DT week_NN ,_, with_IN readings_NNS on_IN trade_NN ,_, output_NN ,_, housing_NN and_CC inflation_NN ._. The_DT

Tagging used the Viterbi algorithm, in conjunction with the probability tables previously discussed. The tagging using the first order data is much faster than tagging using the second order data. This occurs because the first order data only uses a single tag when identifying text. Because of this, for each tag there is only one entry to compare against in the Viterbi algorithm, resulting in under 50 potential tags. When using the second order data, all combinations of two tags found in the document allTraining.txt are used (which results in over 1000 values) making its runtime much longer.

First Order HMM

Test String tags

['DT ', 'NN ', 'POS ', 'NN ', 'MD ', 'VB ', 'VBN ', 'IN ', 'JJ ', 'NN ', 'VBZ ', 'DT ', 'NN ', ', ', 'IN ', 'NNS ', 'IN ', 'NN ', ', ', 'NN ', ', ', 'NN ', 'CC ', 'NN ', '. ', 'DT ']

**values may be left adjusted for formatting purposes*

Resulting Tags

['DT ', 'NN ', 'POS ', 'NN ', 'MD ', 'VB ', 'VBN ', 'IN ', 'JJ ', 'NN ', 'NNS ', 'DT ', 'NN ', ', ', 'IN ', 'NNS ', 'IN ', 'NN ', ', ', 'NN ', ', ', 'NN ', 'CC ', 'NN ', '. ', 'DT ']

False Tags

[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','X ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ']

Space (' ') represents an accurate tag. The 'X ' represents a word that was falsely tagged. Out of 25 tagged words, 24 were accurately tagged. This resulted in a tagging accuracy of ~96%, which is fairly good.

Confusion Matrix

	,	VBZ	CC	NNS	POS	VB	.	IN	MD	NN	VBN	DT	JJ
,	3												
CC			1										
NNS		1		1									
POS					1								
VB						1							
.							1						
IN								3					
MD									1				
NN										8			
VBN											1		
DT												3	
JJ													1

Confusion matrix show the accurate tag on the left hand side, with the classification tag across the top. As you can see in the confusion matrix, one word was classified as VBZ when it should have been classified as NNS. That value is circled in red. Some tag values were left out, in order to make the confusion matrix smaller (so it could fit on the page). All other values not shown, are 0 (this helps you visually see the values we care about better).

Second Order HMM

Test String tags

['DT ', 'NN ', 'POS ', 'NN ', 'MD ', 'VB ', 'VBN ', 'IN ', 'JJ ', 'NN ', 'VBZ ', 'DT ', 'NN ', ', ', 'IN ', 'NNS ', 'IN ', 'NN ', ', ', 'NN ', ', ', 'NN ', 'CC ', 'NN ', '. ', 'DT ']

**values may be left adjusted for formatting purposes*

Resulting Tags

['DT ', 'NN ', 'POS ', 'NN ', 'MD ', 'VB ', 'VBN ', 'IN ', 'JJ ', 'NN ', 'NNS ', 'DT ', 'NN ', ', ', 'IN ', 'NNS ', 'IN ', 'NN ', ', ', 'NN ', ', ', 'NN ', 'CC ', 'NN ', '. ', 'DT ']

False Tags

[' ',' ']

As you can see, using the second order data we were able to accurately tag word's with 100% accuracy. This only takes into account the first 25 words, so this test isn't sufficient to call this tagger 100% accurate, however does show at least one case where it's an improvement over single order data.

Confusion Matrix

	.	VB	NNS	DT	MD	VBN	CC	,	JJ	IN	NN
.	1										
VB		1									
NNS			2								
DT				3							
MD					1						
VBN						1					
CC							1				
,								3			
JJ									1		
IN										3	
NN											8

This confusion matrix shows 100% accuracy, as every word was tagged accurately. This can easily be seen in how the numbers form a perfect diagonal. Again, all missing values are 0's.

Extended Tests

Since tests using only 25 words aren't very thorough, I'll include some accuracy results for larger tests below. Showing the corresponding matrixes and tag's through the text is very lengthy, show I'll just stick to accuracy numbers.

100 words

First Order HMM Accuracy: 0.970297

Second Order HMM Accuracy: 0.950495

This is interesting because it's showing the first order data to be more accurate than the second order data. Looking at the results (shown below), you see that the tagging fails in different places. This could be due to a number of different things. Perhaps the text that we're tagging isn't tagged properly? Oftentimes there are different tags that could be put on a single word, and since the system and data forces you to only tag with one tag per word, the author of the training data could have tagged differently in those circumstances than the author of the tagging data.

Due to processing time constraints, I'm going to keep the results at only 100.

Correct Tags

```
['DT  ', 'NN  ', 'POS ', 'NN  ', 'MD  ', 'VB  ', 'VBN ', 'IN  ', 'JJ  ', 'NN  ', 'NNS ', 'DT  ', 'NN  ', ',   ', 'IN  ', 'NNS ', 'IN  ', 'NN  ', ',   ', 'NN  ', ',   ',
'NN  ', 'CC  ', 'NN  ', '.   ', 'DT  ', 'RBS ', 'JJ  ', 'NN  ', 'MD  ', 'VB  ', 'DT  ', 'NNP ', 'NN  ', 'NN  ', 'NN  ', 'JJ  ', 'IN  ', 'NN  ', '.   ', 'DT  ', 'NN  ',
'NN  ', 'VBZ ', 'VBN ', 'TO  ', 'VB  ', 'TO  ', 'IN  ', '$   ', 'CD  ', 'CD  ', 'IN  ', 'NNP ', 'POS ', '$   ', 'CD  ', 'CD  ', ',   ', 'VBG ', 'TO  ', 'DT  ', 'NN
', 'IN  ', 'NNP ', 'NNP ', ',   ', 'DT  ', 'NN  ', 'IN  ', 'NNP ', 'NNP ', ',   ', 'NNP ', 'NNP ', '.   ', 'NNP ', 'POS ', 'NN  ', 'IN  ', 'DT  ', 'NNP ', 'NN  ',
'NN  ', 'NN  ', 'VBZ ', 'VBN ', 'TO  ', 'VB  ', ',   ', 'IN  ', 'RB  ', 'RB  ', 'RB  ', 'IN  ', 'DT  ', 'CD  ', 'NN  ', 'NN  ', 'VBN ', 'NNP ']
```

First Order Tags

```
['DT  ', 'NN  ', 'POS ', 'NN  ', 'MD  ', 'VB  ', 'VBN ', 'IN  ', 'JJ  ', 'NN  ', 'VBZ ', 'DT  ', 'NN  ', ',   ', 'IN  ', 'NNS ', 'IN  ', 'NN  ', ',   ', 'NN  ', ',   ',
'NN  ', 'CC  ', 'NN  ', '.   ', 'DT  ', 'RBS ', 'JJ  ', 'NN  ', 'MD  ', 'VB  ', 'DT  ', 'NNP ', 'NN  ', 'NN  ', 'NN  ', 'JJ  ', 'IN  ', 'NN  ', '.   ', 'DT  ', 'NN  ',
'NN  ', 'VBZ ', 'VBN ', 'TO  ', 'VB  ', 'TO  ', 'RB  ', '$   ', 'CD  ', 'CD  ', 'IN  ', 'NNP ', 'POS ', '$   ', 'CD  ', 'CD  ', ',   ', 'VBG ', 'TO  ', 'DT  ', 'NN
', 'IN  ', 'NNP ', 'NNP ', ',   ', 'DT  ', 'NN  ', 'IN  ', 'NNP ', 'NNP ', ',   ', 'NNP ', 'NNP ', '.   ', 'NNP ', 'POS ', 'NN  ', 'IN  ', 'DT  ', 'NNP ', 'NN  ',
'NN  ', 'NN  ', 'VBZ ', 'VBN ', 'TO  ', 'VB  ', ',   ', 'IN  ', 'RB  ', 'RB  ', 'RB  ', 'IN  ', 'DT  ', 'CD  ', 'NN  ', 'NN  ', 'VBD ', 'NNP ']
```

First Order Results

```
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ',
' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ']
```

Second Order Tags

```
['DT  ', 'NN  ', 'POS ', 'NN  ', 'MD  ', 'VB  ', 'VBN ', 'IN  ', 'JJ  ', 'NN  ', 'NNS ', 'DT  ', 'NN  ', ',   ', 'IN  ', 'NNS ', 'IN  ', 'NN  ', ',   ', 'NN  ', ',   ',
'NN  ', 'CC  ', 'NN  ', '.   ', 'DT  ', 'RBS ', 'JJ  ', 'NN  ', 'MD  ', 'VB  ', 'DT  ', 'NNP ', 'NN  ', 'NN  ', 'NN  ', 'JJ  ', 'RP  ', 'NN  ', '.   ', 'DT  ', 'NN  ',
'NN  ', 'VBZ ', 'VBN ', 'TO  ', 'VB  ', 'TO  ', 'RB  ', '$   ', 'CD  ', 'CD  ', 'IN  ', 'NNP ', 'VBZ ', '$   ', 'CD  ', 'CD  ', ',   ', 'VBG ', 'TO  ', 'DT  ', 'NN
', 'IN  ', 'NNP ', 'NNP ', ',   ', 'DT  ', 'NN  ', 'IN  ', 'NNP ', 'NNP ', ',   ', 'NNP ', 'NNP ', '.   ', 'NNP ', 'POS ', 'VB  ', 'IN  ', 'DT  ', 'NNP ', 'NN  ',
'NN  ', 'NN  ', 'VBZ ', 'VBN ', 'TO  ', 'VB  ', ',   ', 'IN  ', 'RB  ', 'RB  ', 'RB  ', 'IN  ', 'DT  ', 'CD  ', 'NN  ', 'VB  ', 'VBN ', 'NNP ']
```

Second Order Results

```
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'X',
' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ',
' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ']
```

Feedback

I enjoyed working on this lab. Once I got the concept down it was fun. The Wikipedia page on Viterbi algorithm was extremely helpful. Maybe give that to other students. It's got some code you can follow to help you build the algorithm. I am a bit confused on what you want with probabilities for transition/emission models. The points seem to pin point 4 different models with probabilities that should be shown as "reasonable". One for building the paragraph using unigram data. Another using bigram data. Then one to tag unigram data, and another to tag bigram data. I didn't see the need for different probability tables between generating and tagging amongst similar ngram data. If you wanted the paragraph builders to only build using words (and not introduce the level of tags), then I did that wrong. That is my only thought of how the data could be differentiated. Even if that's the case, I feel what I did was a good solution. It builds paragraphs at random, first by generating a chain of tags (build using probabilities from the ngram/transmission data), then using those tags it generates a word for each tag (word generated using probabilities from emission data.