Home Work 2 – Hidden Markov Models

1 Bigram HMMs

1.1 Design

Used grid search for choosing different hyper-parameters for building different *Bigram HMM* to have a best model.

As preprocessing, add the **START** and **END** symbol and tag into all dataset. Also, by checking the training dataset manually, add the normalization for the words in all dataset:

Start with "\u"	replaced as "Emotion"
Start with "http"	replaced as "URL"
Start with "@"	replaced as "AtPerson"
Start with "#"	replaced as "HashTag"

For the raw program output please check the *Result-BigramHMM.txt* and here are the accuracy results of different parameters that being used:

1.1.1 Different Unknown Threshold

	Threshold of unknown words		
	1	3	5
Accuracy on Dev Set	92.1811%	90.9592%	90.0597%

1.1.2 Smoothing with Add-K

When doing the smoothing with Add-K algorithm, used 2 different K value for Emission (word to tag) and for Transition (tag to tag).

Accuracy on Dev Set (UNK_Threshold = 1)		***K for Emission***					
		0.0001	0.001	0.01	0.1	1	10
K for Transition	0.0001	92.1811%	92.1757%	92.1498%	91.6967%	87.9819%	75.5732%
	0.001	92.1811%	92.1757%	92.1498%	91.6967%	87.9819%	75.5732%
	0.01	92.1811%	92.1757%	92.1498%	91.6940%	87.9683%	75.5732%
	0.1	92.1811%	92.1757%	92.1485%	91.4450%	86.7640%	75.5596%
	1	92.1771%	92.1716%	92.0750%	88.1234%	75.7215%	71.2283%
	10	92.1716%	92.1689%	91.6491%	83.2682%	64.6995%	61.1915%

1.1.3 Smoothing with Linear Interpolation

Accuracy on Dev Set	***Lambda***				
	(0.5, 0.5)	(0.333, 0.667)	(0.25, 0.75)	(0.75, 0.25)	(0.666, 0.334)
UNK_Threshold = 1	91.9417%	92.1485%	92.2260%	90.9306%	91.4518%

1.1.4 Best Model

After combines all the hyper parameters, we can get the best *Bigram HMM* should with these parameters:

UNK_Threshold	1
K for Emission	0.0001
K for Transition	0.0001
Lambda	(0.25, 0.75)

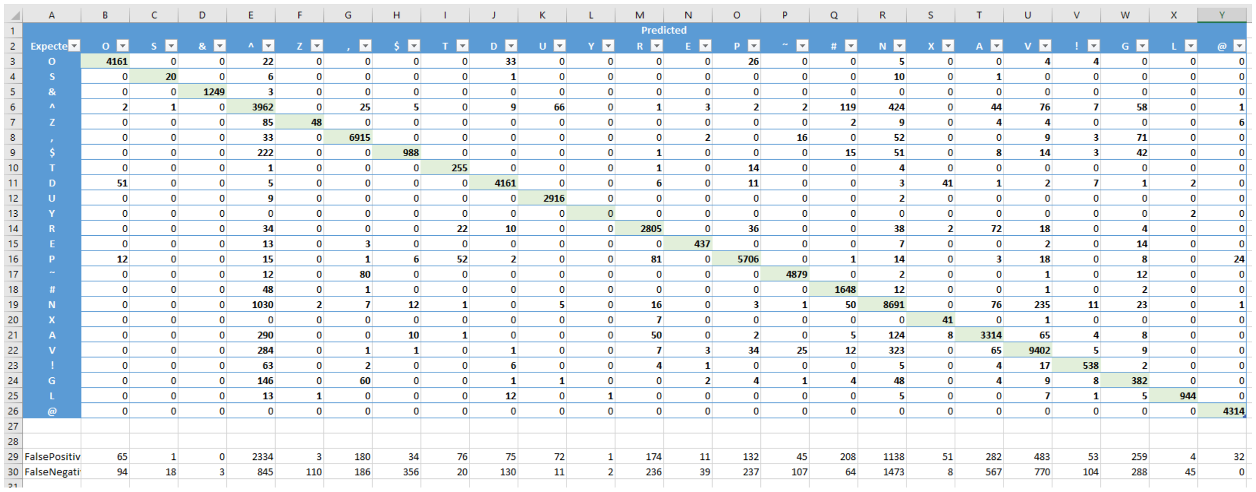
And the performances are:

on dev set is:	92.2260%
on test set is:	92.3453%.

1.2 Error Analysis

1.2.1 Confusion matrix

After using the best *Bigram HMM* to predict the Dev Dataset we can get the confusion matrix (*ConfusionMatrix-BigramHMM.xlsx*).



	Predicted																							
Expecte	O	S	&	^	Z	,	$	T	D	U	Y	R	E	P	~	#	N	X	A	V	!	G	L	@
O	4161	0	0	22	0	0	0	0	33	0	0	0	0	26	0	0	5	0	0	4	4	0	0	0
S	0	20	0	6	0	0	0	0	1	0	0	0	0	0	0	0	10	0	1	0	0	0	0	0
&	0	0	1249	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
^	2	1	0	3962	0	25	5	0	9	66	0	1	3	2	2	119	424	0	44	76	7	58	0	1
Z	0	0	0	85	48	0	0	0	0	0	0	0	0	0	0	2	9	0	4	4	0	0	0	6
,	0	0	0	33	0	6915	0	0	0	0	0	0	2	0	16	0	52	0	0	9	3	71	0	0
$	0	0	0	222	0	0	988	0	0	0	0	1	0	0	0	15	51	0	8	14	3	42	0	0
T	0	0	0	1	0	0	0	255	0	0	0	1	0	14	0	0	4	0	0	0	0	0	0	0
D	51	0	0	5	0	0	0	0	4161	0	0	6	0	11	0	0	3	41	1	2	7	1	2	0
U	0	0	0	9	0	0	0	0	0	2916	0	0	0	0	0	0	2	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
R	0	0	0	34	0	0	0	22	10	0	0	2805	0	36	0	0	38	2	72	18	0	4	0	0
E	0	0	0	13	0	3	0	0	0	0	0	0	437	0	0	0	7	0	0	2	0	14	0	0
P	12	0	0	15	0	1	6	52	2	0	0	81	0	5706	0	1	14	0	3	18	0	8	0	24
~	0	0	0	12	0	80	0	0	0	0	0	0	0	0	4879	0	2	0	0	1	0	12	0	0
#	0	0	0	48	0	1	0	0	0	0	0	0	0	0	0	1648	12	0	0	1	0	2	0	0
N	0	0	0	1030	2	7	12	1	0	5	0	16	0	3	1	50	8691	0	76	235	11	23	0	1
X	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	41	0	1	0	0	0	0
A	0	0	0	290	0	0	10	1	0	0	0	50	0	2	0	5	124	8	3314	65	4	8	0	0
V	0	0	0	284	0	1	1	0	1	0	0	7	3	34	25	12	323	0	65	9402	5	9	0	0
!	0	0	0	63	0	2	0	0	6	0	0	4	1	0	0	0	5	0	4	17	538	2	0	0
G	0	0	0	146	0	60	0	0	1	1	0	0	2	4	1	4	48	0	4	9	8	382	0	0
L	0	0	0	13	1	0	0	0	12	0	1	0	0	0	0	0	5	0	0	7	1	5	944	0
@	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4314
FalsePositiv	65	1	0	2334	3	180	34	76	75	72	1	174	11	132	45	208	1138	51	282	483	53	259	4	32
FalseNegati	94	18	3	845	110	186	356	20	130	11	2	236	39	237	107	64	1473	8	567	770	104	288	45	0

By checking this matrix, we can see that

- For largely used tags:
  - "^" and "N" have lots of false positive and false negative.
- For small amount tags:
  - "S" has many false negative
  - "X" has many false positive
  - "G" has many false negative and false positive
  - "Y" no correct predict because it only appears 2 times in the training set

1.2.2 Data that contains most errors

At the end of the raw program output (*Result-BigramHMM.txt*), the program output 10 sentences that contains the most errors. And we can find the following:

- Lots of words had been predict as "^"
- In some certain cases:

- The word starts with "#" should not be consider as the tag "#", to produce a better model, we need to have a better solution for the normalization logic.

2 Trigram HMM

2.1 Joint Probability Model

For the *Trigram HMM*, the joint probability model should be:

$$p(s,y) = p(x_1x_2 \dots x_n, y_1y_2 \dots y_{n+1})$$

$$p(s,y) = q(y_{n+1} = \mathrm{STOP}|y_{n-1}, y_n)\prod_{i=1}^{n} q(y_i|y_{i-2}, y_{i-1})e(x_i|y_i)$$

And with the following assumptions: $y_{-1} = START1$, $y_0 = START2$, and $y_{n+1} = STOP$.

2.2 Viterbi Decoding Algorithm

2.2.1 Dynamic program for computing (for all i)

$$\pi(i, y_i, y_{i-1}) = \max_{y_1 \dots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i)$$

2.2.2 Iterative computation:

Base case, for $i = 0$:

$$\pi(0, y_0, y_{-1}) = \begin{cases} 1 \; if \; y_{-1} == START1 \; and \; y_0 == START2 \\ 0 \; Otherwise \end{cases}$$

Normal Case, for $i = 1 \dots n$:

$$\pi(i, y_i, y_{i-1}) = \max_{y_{i-2}, y_{i-1}} q(y_i|y_{i-2}, y_{i-1})e(x_i|y_i)\pi(i-1, y_{i-1}, y_{i-2})$$

2.2.3 Back Pointers

$$bp(i, y_i, y_{i-1}) = \operatorname*{argmax}_{y_{i-2}, y_{i-1}} q(y_i|y_{i-2}, y_{i-1})e(x_i|y_i)\pi(i-1, y_{i-1}, y_{i-2})$$

2.2.4 Final Solution

$$y^* = \operatorname*{argmax}_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

2.3 Code Design

Similar as *Bigram HMM*, the *Trigram HMM* add the **START1**, **START2** and **END** symbols and tags into all data set. Also, add the same normalization for the words in all dataset. For the raw program output please check the *Result-TrigramHMM.txt*. The best model is using these parameters:

UNK_Threshold	1
K for Emission	0.0001
K for Transition	0.1
Lambda	(0.1, 0.2, 0.7)

And the performances are:

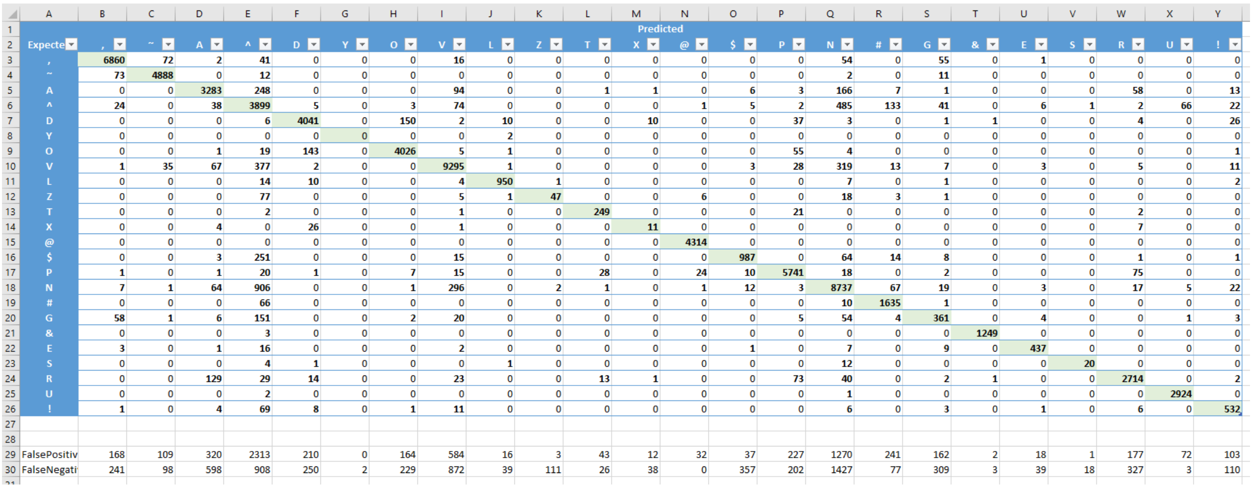
on dev set is:	91.4423%

- on test set is: 91.5809%

2.3.1 Compare to Bigram HMM

By checking and comparing the confusion matrix of *Trigram HMM* (*ConfusionMatrix-TrigramHMM.xlsx*), we can find the *Trigram HMM* have similar problem as *Bigram HMM*, such as:

- "^" and "N" have lots of false positive and false negative.
- "Y" no correct predict because it only appears 2 times in the training set
- Lots of words had been predict as "^"



	Predicted																							
Expecte	,	~	A	^	D	Y	O	V	L	Z	T	X	@	$	P	N	#	G	&	E	S	R	U	!
,	6860	72	2	41	0	0	0	16	0	0	0	0	0	0	0	54	0	55	0	1	0	0	0	0
~	73	4888	0	12	0	0	0	0	0	0	0	0	0	0	0	2	0	11	0	0	0	0	0	0
A	0	0	3283	248	0	0	0	94	0	0	1	1	0	6	3	166	7	1	0	0	0	58	0	13
^	24	0	38	3899	5	0	3	74	0	0	0	0	1	5	2	485	133	41	0	6	1	2	66	22
D	0	0	0	6	4041	0	150	2	10	0	0	10	0	0	37	3	0	1	1	0	0	4	0	26
Y	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	1	19	143	0	4026	5	1	0	0	0	0	0	55	4	0	0	0	0	0	0	0	1
V	1	35	67	377	2	0	0	9295	1	0	0	0	0	3	28	319	13	7	0	3	0	5	0	11
L	0	0	0	14	10	0	0	4	950	1	0	0	0	0	0	7	0	1	0	0	0	0	0	2
Z	0	0	0	77	0	0	0	5	1	47	0	0	6	0	0	18	3	1	0	0	0	0	0	0
T	0	0	0	2	0	0	0	1	0	0	249	0	0	0	21	0	0	0	0	0	0	2	0	0
X	0	0	4	0	26	0	0	1	0	0	0	11	0	0	0	0	0	0	0	0	0	7	0	0
@	0	0	0	0	0	0	0	0	0	0	0	0	4314	0	0	0	0	0	0	0	0	0	0	0
$	0	0	3	251	0	0	0	15	0	0	0	0	0	987	0	64	14	8	0	0	0	1	0	1
P	1	0	1	20	1	0	7	15	0	0	28	0	24	10	5741	18	0	2	0	0	0	75	0	0
N	7	1	64	906	0	0	1	296	0	2	1	0	1	12	3	8737	67	19	0	3	0	17	5	22
#	0	0	0	66	0	0	0	0	0	0	0	0	0	0	0	10	1635	1	0	0	0	0	0	0
G	58	1	6	151	0	0	2	20	0	0	0	0	0	0	5	54	4	361	0	4	0	0	1	3
&	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1249	0	0	0	0	0
E	3	0	1	16	0	0	0	2	0	0	0	0	0	1	0	7	0	9	0	437	0	0	0	0
S	0	0	0	4	1	0	0	0	1	0	0	0	0	0	0	12	0	0	0	0	20	0	0	0
R	0	0	129	29	14	0	0	23	0	0	13	1	0	0	73	40	0	2	1	0	0	2714	0	2
U	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2924	0
!	1	0	4	69	8	0	1	11	0	0	0	0	0	0	0	6	0	3	0	1	0	6	0	532
FalsePositiv	168	109	320	2313	210	0	164	584	16	3	43	12	32	37	227	1270	241	162	2	18	1	177	72	103
FalseNegati	241	98	598	908	250	2	229	872	39	111	26	38	0	357	202	1427	77	309	3	39	18	327	3	110

One good part of *Trigram HMM* that:

- The **False Positive** of each tag is smaller than *Bigram HMM*. This should because the *Trigram HMM* will need more data from previous tag to define the connection. So, will have less missing match of the transition between tags.
- However, the **False Negative** of each tag is larger than *Bigram HMM*. This should be the same reason that cause **False Positive** less. Since we don't have large number of samples to do the training, and the Trigram HMM will not see enough tag transitions, so lot of the possibilities are count as low possible situation. And this cause the *Trigram HMM* behavior poorly than *Bigram HMM*.

3 Unsupervised Training with EM

3.1 EM derivation on Trigram HMMs

So same as the Bigram HMM, the marginal probability of each tag for y_i can be present as:

$$p(x_1 \dots x_n, y_{1i}) = \sum_{y_1 \dots y_{i-1}} \sum_{y_{i+1} \dots y_n} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

$$= p(x_1 \dots x_i, y_i) p(x_{i+1} \dots x_n | y_i)$$

And with the following assumptions: $y_{-1} = START1$, $y_0 = START2$, and $y_{n+1} = STOP$.

For both sides, left side can use the same Forward Algorithm, while the right side can use the same Backward Algorithm.

3.1.1 Forward

Base case, for $i = 0$:

$$\alpha(0, y_0, y_{-1}) = \begin{cases} 1 \; if \; y_{-1} == START1 \; and \; y_0 == START2 \\ 0 \; Otherwise \end{cases}$$

Normal Case, for $i = 1 \dots n$:

$$\alpha(i, y_i, y_{i-1}) = \sum_{y_{1-1} y_{i-2}} q(y_i | y_{i-2}, y_{i-1}) e(x_i | y_i) \alpha(i-1, y_{i-1}, y_{i-2})$$

3.1.2 Backward

Base case, for $i = n$:

$$\beta(n, y_n, y_{n-1}) = \begin{cases} q(y_{n+1} | y_{n-1}, y_n) \; if \; y_{n+1} == STOP \\ 0 \; Otherwise \end{cases}$$

Normal Case, for $i = 1 \dots n-1$:

$$\beta(i, y_i, y_{i+1}) = \sum_{y_{1+1} y_{i+2}} q(y_{i+2} | y_i, y_{i+1}) e(x_{i+1} | y_{i+1}) \beta(i+1, y_{i+1}, y_{i+2})$$

3.1.3 Forward-Backward Algorithm

Since we have:

$$\begin{aligned} p(x_1 \dots x_n) &= \sum_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_{n+1}) \\ &= \sum_{y_{n-1} y_n} q(STOP | y_{n-1} y_n) \alpha(n, y_n, y_{n-1}) \\ &:= \sum_{y_n} \alpha(n+1, STOP, y_n) \end{aligned}$$

So, for the probability of y_i we can compute by:

$$p(y_i | x_1 \dots x_n) = \frac{p(x_1 \dots x_n, y_i)}{p(x_1 \dots x_n)}$$

$$p(y_i y_{i+1} | x_1 \dots x_n) = \frac{p(x_1 \dots x_n, y_i, y_{i+1})}{p(x_1 \dots x_n)}$$

Because the above marginals can be computed as

$$p(x_1 \dots x_n, y_i) = \alpha(i, y_i, y_{i-1}) \beta(i, y_i, y_{i+1})$$

$$p(x_1 \dots x_n, y_i, y_{i+1}) = \alpha(i, y_i, y_{i-1}) q(y_{i+1} | y_{i-1}, y_i) e(x_{i+1} | y_{i+1}) \beta(i+1, y_{i+1}, y_{i+2})$$

3.1.4 Expected count

Even further, we can compute the following:

- Expected count of thing:

$$\text{(expected) } count(NN) = \sum_i p(y_i = NN | x_1 \ldots x_n)$$

- Expected transition counts

$$\text{(expected) } count(NN \rightarrow VB) = \sum_i p(y_i = NN, y_{i+1} = VB | x_1 \ldots x_n)$$

- Expected emission counts

$$\text{(expected) } count(NN \rightarrow apple) = \sum_i p(y_i = NN, x_i = apple | x_1 \ldots x_n)$$

$$= \sum_{i:x_i = apple} p(y_i = NN | x_1 \ldots x_n)$$

3.1.5 Maximum Likelihood Parameters

Finally, we can use all those expected counts to compute Maximum Likelihood Parameters:

$$q_{ML}(y_i | y_{i-2}, y_{i-1}) = \frac{c(y_{i-2}, y_{i-1}, y)}{c(y_{i-2}, y_{i-1})}$$

$$e_{ML}(x|y) = \frac{c(y, x)}{c(y)}$$