1 Bigram HMM

The solution to this problem is divided into the following 2 sections:

A. Comparison of different design approaches for Smoothing and OOV.

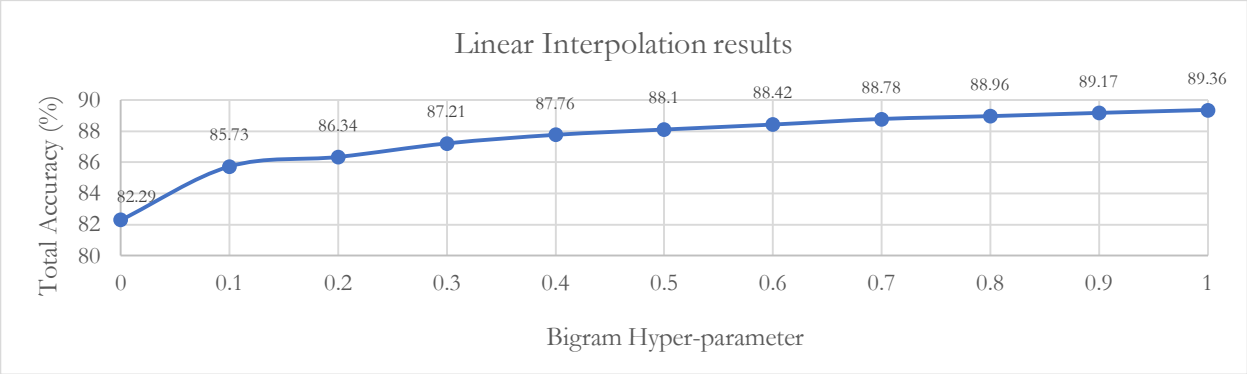
B. Present the results of running the best model on the testing data-set.

C. Error analysis on the test results.

Part A

Smoothing

I picked Linear Interpolation as the smoothing technique for Transition probability. I ran the model with 11 different hyper-parameter settings for Linear Interpolation and obtained the following results.



Observations

Per my observation, as we increase the Bigram component (Bigram Hyper-parameter for Linear Interpolation) in the Transition Probability, the accuracy of the model increases. I was able to achieve 89.36% accuracy when I set the Hyper-parameters as {1.0, 0.0} i.e. 1.0 for the Bigram component and 0.0 for the Unigram component.

OOV

In order to handle the Out of Vocabulary/Unknown words, I started with an approach where I marked low frequency words (Unknowns) as <<UnknownToken>>. With this approach, I was able to achieve an accuracy of 89.36%.

I observed that large portion of low frequency words are URLs, Twitter handles (@foobar), Hash tags (#bazqux), and Numbers. This was causing a majority of the low frequency words that are not from the above classes to be tagged as the above classes.

In order to address this bias, I created the following specific types of Unknowns to distribute the low frequency words into separate buckets:

- URL
- Hash-tag
- Twitter handle
- Number
- Other Unknowns

I also observed that the probabilities of the Proper Noun Possessives, Nominal Possessives was substantially low and I decided to create separate buckets for these as well.

Results

Tags	Accuracy % with One Unknown	Accuracy % with Specific Unknowns
S	52.63	63.16
V	92.25	92.61
D	96.57	96.64
Y	0	0
#	62.97	92.99
,	96.24	97.47
P	96.38	96.37
G	51.64	65.67
&	99.76	99.76
A	85.42	85.6
R	92.44	92.44
@	85.93	99.95
^	73.91	82.26
U	89.07	99.93
L	95.25	95.25
X	83.67	83.67
O	97.96	97.88
E	90.13	91.6
N	84.8	86.06
Z	32.28	77.85
$	73.07	94.87
~	97.91	97.81
T	92.73	92
!	82.24	86.76
Total Accuracy	**89.36**	**92.88**

Observations

The following observations can be made from these results above:

- Accuracy of Tags that had a dedicated Unknown type associated with them – such as URL (U), Hash tags (#), Twitter handles (@), and Numbers ($) – increased drastically since the words are bucketed in their corresponding Unknown type and their respective Tags no longer compete with rest of the Tags associated with other low-frequency words.
- Accuracy of Tags such as Nominal Possessives (S) and Proper-noun Possessives (Z) also increased due to bucketing.
- Accuracy substantially increased for other Tags that did not have special Unkown buckets such as Proper Nouns (^), Interjections (!), etc. This increase is attributed to the sparsely populated "Other Unknowns" bucket.

Hence, my best Bigram HMM model comprises of Linear Interpolation smoothing with hyper-parameters {1.0, 0.0} and specific types of Unknowns to handle OOV words.

Part B

Here are the results of running the best model, as described in Part A, on the test data set.

tag	S	V	D	#	P	,	G	&	A	@	R	^	U	L	X	O	E	N	Z	~	$	T	!	
S	88.24	0	0	0	0	0	0	0	0	0	0	2.94	0	0	0	0	0	0	8.82	0	0	0	0	0
V	0	93.12	0	0.11	0.26	0	0.28	0	0.62	0	0.08	2.13	0	0.01	0.03	0.01	0.04	3.14	0	0.07	0.05	0	0.06	0
D	0	0.02	96.27	0	0.23	0	0.02	0	0.05	0	0.47	0.21	0	0.07	0.92	1.55	0	0.12	0	0	0	0	0.07	0
#	0	0	0	93.19	0	0.06	0.11	0	0.17	0	0	5.75	0	0	0	0	0	0.73	0	0	0	0	0	0
P	0	0.42	0.07	0.02	96.27	0.02	0.07	0	0.07	0	1.09	0.33	0.03	0	0	0.13	0	0.15	0	0	0.25	1.09	0	0
,	0	0.17	0	0.03	0	97.38	1.15	0	0.04	0	0	0.46	0	0	0	0	0.06	0.43	0	0.28	0	0	0.03	0
G	0	1.79	0.3	0.75	0.3	10.46	63.08	0	0.75	0.15	0	13.6	0.3	0	0	0.15	0.75	4.93	0.15	0.15	1.35	0	1.05	0
&	0	0	0	0	0	0	0	99.57	0	0	0.09	0.34	0	0	0	0	0	0	0	0	0	0	0	0
A	0	1.43	0	0.17	0.11	0	0.2	0	86.44	0	1.2	6.02	0	0	0.14	0	0	3.92	0	0	0.22	0.03	0.11	0
@	0	0	0	0	0	0	0	0	0	99.82	0	0.02	0	0	0	0	0	0	0.05	0	0.11	0	0	0
R	0.03	1.08	0.39	0.03	0.85	0.03	0.13	0	2.12	0	91.39	1.08	0	0	0.07	0	0	1.5	0	0	0.16	0.95	0.2	0
^	0	1.4	0.08	1.77	0.08	0.31	1.34	0	1.34	0	0.04	82.92	0.02	0	0	0.06	0.1	8.8	0.06	0.02	1.34	0	0.33	0
U	0	0	0	0	0	0	0	0	0	0	0	0	99.93	0	0	0	0	0.03	0	0	0	0	0.03	0
L	0	0.22	1.73	0	0	0.11	0.11	0	0	0	0.11	1.3	0	95.46	0	0	0	0.22	0.43	0	0	0	0.11	0
X	0	0	0	0	0	0	0	0	0	0	7.94	0	0	0	92.06	0	0	0	0	0	0	0	0	0
O	0	0.14	0.85	0	0.44	0	0.07	0	0	0	0.02	0.37	0	0	0	97.79	0	0.25	0	0	0	0	0.07	0
E	0	0.19	0	0	0	0.76	4.73	0	0	0	0	4.35	0	0	0	0	87.71	0.95	0	0	0.95	0	0.38	0
N	0.02	2.01	0.02	0.4	0.03	0.07	0.7	0	0.75	0	0.14	8.21	0.01	0.01	0	0.01	0.03	87.14	0.05	0.02	0.13	0.02	0.22	0
Z	1.27	1.9	0	0.63	0	0	0	0	0.63	5.7	0	9.49	0	1.27	0	0	0	3.16	75.95	0	0	0	0	0
~	0	0.12	0	0	0	1.21	0.4	0	0	0	0	0.04	0	0	0	0	0.02	0.02	0	98.1	0.02	0	0.06	0
$	0	0	0	0.85	0.14	0	0.5	0	0.07	0	0	2.78	0	0	0	0	0	0.07	1.71	0	0	93.88	0	0
T	0	0.31	0	0	5.26	0	0	0	0	0	0.93	0.62	0	0	0	0	0	0.31	0	0	0	92.57	0	0
!	0	1.62	0.29	0	0	0.44	2.06	0	0.59	0	0.74	9.26	0	0.15	0	0	0.29	1.47	0	0.15	0.15	0	82.79	0

Each row in the above table corresponds to the actual tag and each column corresponds to the predicted tag. The value in the cell (i, j) represents the percentage of words with tag 'i' that were predicted by my model as tag 'j'.

Additionally, I trained model on the Bonus training data and got obtained the following results for the Test data set.

Accuracy percentage by Tag

	Trained on twt.bonus.json	Trained on twt.train.json
Total Accuracy	**95.47**	**93.08**
S	88.24	88.24
V	96.1	93.12
D	96.64	96.27
#	94.47	93.19
P	96.66	96.27
,	98.34	97.38
G	76.08	63.08
&	99.49	99.57
A	93.61	86.44

@	99.89	99.82
R	93.87	91.39
^	87.27	82.92
U	99.93	99.93
L	97.41	95.46
X	95.24	92.06
O	98.43	97.79
E	93.76	87.71
N	92.13	87.14
Z	85.44	75.95
~	98.16	98.1
$	95.16	93.88
T	94.12	92.57
!	93.53	82.79

Observation: Bonus training data has more tweets. With increase in the size of the training dataset the overall accuracy of Model and accuracy by POS increased.

Part C

Error analysis

Primary source of error rates are highlighted in confusion matrix and listed as follows:

- ^ vs N vs V vs # vs G vs A vs $
- S vs Z
- T vs P
- A vs R
- O vs D

The major reasons for these errors are as follows:

- One of the phrases "Wichita (^) Mountains (^) refuge (N)" was tagged as "Wichita (^) Mountains (^) refuge (^)" in the labelled output. This could be attributed to the Transition probability of (N, ^) being lower than (^, ^).
- The confusion between S and Z stems from the similar nature of these words (ending in 's or s'). It is further exacerbated by the order and the rules I used to perform the bucketization. For every un-classified Unknown, I first check to see if it starts with an Upper-case letter and ends in

's or s'. For such words, I mark them as a Proper Noun Possessive Unknowns. These potentially include some Nominal Possessives that start in an Upper-case letter.

- Tags with few low-frequency words are overpowered by Tags with more low-frequency words when calculating the Emission probabilities.

2 Trigram HMM

The solution to this problem is divided into the following 2 sections:

A. Discussion on Smoothing and OOV approaches.

B. Present the results of running the best model on the testing data-set.

C. Error analysis on the test results.

D. Comparison of Bigram and Trigram HMM Tagger.

Part A

Smoothing

I picked Linear Interpolation as the smoothing technique for Transition probability here as well. I ran the model with different hyper-parameter settings for Linear Interpolation and obtained the following results. I have attached excel sheet for results using different hyper-parameters. I was able to achieve 92.95% accuracy with the Hyper-parameters as {0.2, 0.7,0.1}.

OOV

After comparing both the approaches (one Unknown vs specific types of Unknowns) discussed in the OOV section of Part 1A, I found that the latter approach provided better accuracy for Trigram HMM as well.

My best Trigram HMM model comprises of Linear Interpolation smoothing with hyper-parameters {0.2, 0.7, 0.1} and specific types of Unknowns to handle OOV words.

Part B

Here are the results of running the best model, as described in Part A, on the test data set.

tag	S	V	D	#	P	,	G	&	A	@	R	^	U	L	X	O	E	N	Z	~	$	T	!
S	88.24	0	0	0	0	0	0	0	0	0	0	2.94	0	0	0	0	0	0	8.82	0	0	0	0
V	0	92.92	0	0.11	0.24	0	0.23	0	0.67	0	0.07	2.36	0	0.01	0.03	0.01	0.05	3	0	0.25	0.05	0.01	0.01
D	0	0.02	96.97	0	0.23	0	0.02	0	0.05	0	0.07	0.19	0	0.09	0.77	1.48	0	0.07	0	0	0	0	0.02
#	0	0	0	94.3	0	0.06	0.11	0	0.22	0	0	4.91	0	0	0	0	0	0.39	0	0	0	0	0
P	0	0.32	0.05	0.02	96.21	0.02	0.05	0	0.07	0	1.22	0.33	0.03	0	0	0.13	0	0.17	0	0	0.28	1.1	0
,	0	0.14	0	0.03	0	97.06	1.23	0	0.03	0	0	0.47	0	0	0	0	0.06	0.44	0	0.54	0	0	0.01
G	0	1.79	0.15	0.75	0.3	9.42	63.68	0	0.75	0.15	0	14.95	0.3	0	0	0	1.05	4.63	0	0.15	1.35	0	0.6
&	0	0	0	0	0	0	0	99.49	0	0	0.17	0.34	0	0	0	0	0	0	0	0	0	0	0
A	0	1.37	0	0.2	0.14	0	0.14	0	86.38	0	1.26	6.16	0	0	0.14	0	0	3.84	0	0	0.22	0.03	0.11
@	0	0	0	0	0	0	0	0	0	99.86	0	0.02	0	0	0	0	0	0	0	0	0.11	0	0
R	0.03	1.08	0.49	0.03	0.98	0.03	0.13	0	1.96	0	91.46	1.01	0	0	0.1	0	0	1.5	0	0	0.16	0.91	0.13
^	0	1.34	0.08	2.36	0.08	0.25	1.11	0	1.3	0	0.04	83.7	0.02	0	0	0.04	0.06	8.06	0.06	0.02	1.3	0	0.19
U	0	0	0	0	0	0	0	0	0	0	0	0	99.97	0	0	0	0	0	0	0	0	0	0.03
L	0	0.65	1.94	0	0	0	0.11	0	0	0	0.11	1.08	0	95.25	0	0	0	0.22	0.43	0	0	0	0
X	0	0	3.17	0	0	0	0	0	0	0	4.76	0	0	0	92.06	0	0	0	0	0	0	0	0
O	0	0.12	0.9	0	0.46	0	0.05	0	0	0	0.02	0.42	0	0	0	97.74	0	0.25	0	0	0	0	0.05
E	0	0	0	0	0	0.95	4.73	0	0	0	0	3.78	0	0	0	0	88.28	1.13	0	0	0.95	0	0.19
N	0.02	2.06	0.02	0.42	0.04	0.08	0.57	0	0.71	0	0.15	8.37	0.01	0	0	0.01	0.01	87.04	0.05	0.03	0.21	0.02	0.17
Z	1.27	1.27	0	1.27	0	0	0	0	0.63	5.7	0	8.86	0	1.27	0	0	0	3.8	75.95	0	0	0	0
~	0	0	0	0	0	1.21	0.36	0	0	0	0	0.06	0	0	0	0	0.02	0.08	0	98.24	0.02	0	0
$	0	0	0	0.85	0.14	0	0.28	0	0.07	0	0	3.2	0	0	0	0	0.07	0.43	0	0	94.95	0	0
T	0	0.31	0	0	5.26	0	0	0	0	0	0.93	0.62	0	0	0	0	0	0.31	0	0	0	92.57	0
!	0	1.62	0.29	0	0	0.44	1.62	0	0.59	0	1.03	9.71	0	0.15	0	0	0.29	2.79	0	0.15	0.15	0	81.18

Each row in the above table corresponds to the actual tag and each column corresponds to the predicted tag. The value in the cell (i, j) represents the percentage of words with tag 'i' that were predicted by my model as tag 'j'.

Additionally, I trained model on the Bonus training data and got obtained the following results for the Test data set.

Tags	Trained on twt.bonus.json	Trained on twt.train.json
Total Accuracy	**95.54**	**93.14**
S	88.24	88.24
V	96.1	92.92
D	97.14	96.97
#	95.64	94.3
P	96.61	96.21
,	97.91	97.06
G	77.28	63.68
&	99.49	99.49
A	93.7	86.38
@	99.89	99.86
R	93.58	91.46

^	87.56	83.7
U	99.97	99.97
L	97.08	95.25
X	92.06	92.06
O	98.32	97.74
E	93.95	88.28
N	92.28	87.04
Z	86.71	75.95
~	98.24	98.24
$	96.44	94.95
T	94.12	92.57
!	92.79	81.18

Observation: Bonus training data has more tweets. With increase in the size of the training dataset the overall accuracy of Model and accuracy by POS increased.

Part C

Error analysis

Primary source of error rates highlighted in confusion matrix

- ^ vs N vs V vs # vs G vs N vs $
- S vs Z
- T vs P
- A vs R
- O vs D
- G vs E

The major reasons for these errors are as follows:

- One of the phrases "Wichita (^) Mountains (^) refuge (N)" was tagged as "Wichita (^) Mountains (^) refuge (^)" in the labelled output. This could be attributed to the Transition probability of (N, ^) being lower than (^, ^).
- The confusion between S and Z stems from the similar nature of these words (ending in 's or s'). It is further exacerbated by the order and the rules I used to perform the bucketization. For every un-classified Unknown, I first check to see if it starts with an Upper-case letter and ends in

's or s'. For such words, I mark them as a Proper Noun Possessive Unknowns. These potentially include some Nominal Possessives that start in an Upper-case letter.

- Tags with few low-frequency words are overpowered by Tags with more low-frequency words when calculating the Emission probabilities.

Part D

Did the trigram model outperform the bigram model? Significantly?

- Trigram HMM's accuracy was better than Bigram HMM's accuracy but it wasn't significantly better. One possible reason is that there were many unseen trigrams in training corpus.
- In terms of execution time, Trigram HMM was way slower than Bigram HMM.

Note: I have used log probabilities throughout the implementation.