ERROR CORRECTION USING NATURAL LANGUAGE PROCESSING

A Thesis

by

NILESH KUMAR JAVAR

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Anxiao Jiang
Committee Members,	Frank M. Shipman, III
	Tie Liu
Head of Department,	Dilma Da Silva

May 2015

Major Subject: Computer Science

ABSTRACT

Data reliability is very important in storage systems. To increase data reliability there are techniques based on error-correcting codes(ECCs). These techniques introduce redundant bits into the data stored in the storage system to be able to do error correction. The error correcting codes based error correction has been studied extensively in the past. In this thesis, a new error correction technique based on the redundant information present within the data is studied. To increase the data reliability, an error correction technique based on the characteristics of the text that the data represents is discussed. The focus is on correcting the data that represents English text using the parts-of-speech property associated with the English text.

Three approaches, pure count based approach, two-step HMM based approach and bit error minimization based approach, have been implemented. The approach based on two-step HMM has outperformed the other two approaches. Using this newly proposed technique with the existing error correcting techniques would further increase the data reliability and would complement the performance of existing error correcting techniques.

ACKNOWLEDGEMENTS

I would like to sincerely thank my advisor, Dr. Anxiao Jiang, for giving me opportunity to work with him. It has been a great learning experience for me. The work done as part of this thesis would not have been possible with out the patience, support and immense knowledge of Dr. Jiang. I would also like to thank Dr. Frank Shipman and Dr. Tie Liu for being on my committee.

NOMENCLATURE

HMM	Hidden Markov Model
PEB	Percentage of Erasure Bits
APP	A Posterior Probability
BER	Bit Error Ratio
MB	Megabytes
JSON	JavaScript Object Notation

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

In this thesis, a technique to increase data reliability based on the parts-of-speech property of the English text that the data represents is proposed. In the scope of the problem, the input data represented as bits actually represents a sentence in English text. Each sentence is a sequence of tokens separated by token delimiters. Each token is a sequence of alpha-numeric characters and token delimiters can be any non-alpha numeric character. The list of symbols that can be present in the text represented by the data to be corrected are small-case alphabets(a-z), capital-case alphabets(A-Z), numbers(0-9) and space.

Every sentence can be tokenized into a sequence of tokens and each of these tokens can be associated with a parts-of-speech tag. In general, each token can assume more than one parts-of-speech tag. For example, the token 'good' can be either noun or adjective. The token 'good' is associated with noun in the sentence, 'The cake is good.'. Where as, it can also be classified as an adjective in the sentence, 'It is a good deal.'. Since every token in the generated token sequence can assume one or more than one tag, there can be multiple tag sequences that can be related to that token sequence. If there are m tokens in a sentence and every token can assume at most k tags, then for that token sequence there can be at most k^m tag sequences possible. Among these k^m possible tag sequences there can be only one tag sequence that is most likely given the token sequence. If there are n erasure bits in the bit sequence and as each of the erasure bits can take value of either 0 or 1, then 2^n possible choices can be generated. Of these 2^n possible choices, only one of the choice is correct and it represents the original data without any erasures. Each of these 2^n possible choices may or may not represent valid English text. The

text is considered valid if each of the tokens presents in the text is present in the vocabulary. Each of the valid choices among the 2^n possibilities will in turn have k^m tag sequences possible. Only one of these (2^n * k^m) token-tag sequences represents the corrected data without any erasure bits. A sample model of Token-Tag sequences is illustrated in Figure-1.1

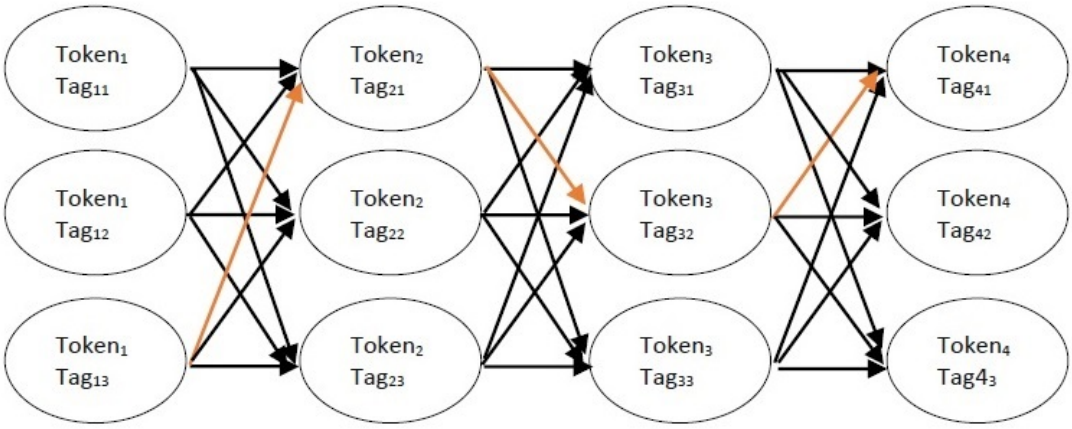

Figure 1.1: Example of Token-Tag sequences possible for a bit sequence(k=3, m=4). The path with orange arrows is the maximum likelihood path. Dynamic programming based algorithm and HMM is used to find such path.

The token boundaries are assumed to be known. Since the token boundaries are assumed to be known, the number of possibilities will be less than 2^n * k^m. If there are 'N' tokens in the input and because of erasures each of the 'N' tokens can take at-most 'L' possible values. Each of the 'L' possible values is in turn a token. Since every token can take more than one tag based on the position within the sentence and the type of the sentence in which the token appears. Let 'q' be the maximum number of different tags that are possible for each of the 'N' tokens after considering

Table 1.1: Encoded and erasured token for the input tokens. The '$' in the bit values of the erasured bit sequences indicates that it is a bit with erasure.

Word index	Word	Encoded Bit sequence	Erasured Bit sequence
1	It	1110010011011	11$$$100$1$$1
2	was	01110010010100	$1$$$$1$0$010$
3	one	10000101001	$00$01$$0$1
4	of	1000111000	100$$$1$0$
5	a	1001	10$$
6	series	01000.....0010100	01$0001....$$$00
7	of	1000111000	10$$11$$$$
8	recommendations	0000..01010100	0$$0.....$01$100
9	by	1111111000111	$111$$10001$1
10	the	101111110001	10$1$$$100$$
11	Texas	11111101...100	111111$10....1$10$
12	Research	11100....1110	$$$$...1$1$0
13	League	11100....0001	$$$$0...0$$

the tags from all the 'L' possible value for each of the tokens. Hence, there would be at-most q^N different tag sequences that would be possible for the input sequence of 'N' tokens.

Each of the q^N tag sequences can be a Markov chain. The sequence of tags form the sequence of hidden states of the chain and corresponding token associated with a tag is the corresponding output of the state. The probability of a given Markov chain with hidden states can be found out using Hidden Markov Model(HMM)[3]. All the symbols are encoded using Huffman coding[6]. The data is essentially a sequence of bits, where symbols in the English text that the data represents are encoded using the Huffman codes of the symbols.

The scope of error correction in the English text using the natural language processing can be illustrated using an example. The sentence considered in this example is "It was one of a series of recommendations by the Texas Research League". The list of tokens representing the text are as follows : It, was, one, of, a, $series$,

Table 1.2: Illustrating the scope of error correction using natural language processing through an example.

Word index	Word	Erasure count	Count of total possibilities	Count of valid possibilities	List of valid possibilities
1	It	6	64	3	It, hot, hat
2	was	8	256	11	sees, seen, won, was, wan, gas, pas, pan, fan, man, hid
3	one	5	32	1	one
4	of	5	32	1	of
5	a	2	4	3	o,a,t
6	series	13	8192	4	series, servo, seeks, senses
7	of	6	64	2	of, am
8	recommendations	27	134217728	1	recommendations
9	by	4	16	1	by
10	the	6	64	3	ace, tar, the
11	Texas	8	256	3	Texas, Texan, Texts
12	Research	25	33554432	1	Research
13	League	16	65536	1	League

of, *recommendations*, *by*, *the*, *Texas*, *Research*, *League*. The Table-1.1 lists the tokens and their corresponding encoded and erasured bit sequence. The erasured bit sequence of a given token is obtained after encoding the given token using Huffman encoding and inducing the erasures. The '$' symbol within an erasured bit sequence indicates that the value of that bit is unknown and is an erasure. For every token, the Table-1.2 lists the number of erasures within the corresponding erasured bit sequence, the list of all valid possibilities for each of the tokens.

Once, the list of words associated with each of the tokens are generated as seen in the last column of Table-1.1, the list of tags that each of the input token can taken based on the list of possible tokens is generated. This list of tags can be seen in the Column-3(named as $Tags\ Taken$) in the Table-1.3. Using the dynamic programming based approach and one-step HMM, the most probable sequence of

Table 1.3: Walkthrough of one-step HMM based algorithm with example

Word index	word	Tags taken	Final tag obtained using one-step HMM	List of words based on Final Tag	Final decoded word
1	It	ADJ, N, NP, PRO	PRO	It	It
2	was	ADJ, FW, N, V, VBZ, VD, VN	V	was, gas, pan, fan, man	was
3	one	NUM, PRO, V	PRO	one	one
4	of	NIL, P	P	of	of
5	a	DET, FW, N, NIL	DET	a	a
6	series	N, VBZ	N	series, servo, senses	series
7	of	NIL, P, V	P	of	of
8	recommendations	N	N	recommendations	recommendations
9	by	ADV, NIL, P	P	by	by
10	the	DET, N, NIL	DET	the	the
11	Texas	ADJ, N, NP	NP	Texas, Texan	Texas
12	Research	N	N	Research	Research
13	League	N	N	League	League

tags is determined. For each of the input tokens, the most probable tag that the token can take is shown in the column-4 of Table-1.3. Using the most probable tag for each of the tokens, the associated list of possible tokens is filtered by only selecting the tokens that can take the most probable tag. If there is more than 1 possibility, then the token with highest word count is selected. For example, for the token $Texas$ at index-11 there are two possibilites, Texas and Texan. Of these two possibilities $Texas$ is chosen as the frequency of the token $Texas$ is higher than the token $Texan$.

To generate Hidden Markov Model, the transition probabilities, emission probabilities and starting probabilities are required. To generate Huffman codes of the symbols, the frequency of the symbols is required. All these required parameters are derived from the Brown corpus.

The most likely sequence of tags is decoded from HMM using a dynamic programming based approach. Once the tag sequence is obtained, tokens associated with each of the tags are selected using their word count or tag count based methodology. Using the selected tokens, the value of the erasure bits is obtained. The following sections describe the problem statement and the summary of results.

1.1 Problem statement

The amount of data generated keeps increasing with time and the need has always been there to store the data without getting lost. There are several error correcting codes like LDPC codes[12], Polar codes[13] and Turbo codes[14] etc., which have been successful in error-correction. But, all these techniques add redundancy to the data. State-of-art compression algorithms like Huffman codes[6], Lempel-Ziv codes[15], arithmetic coding[16] etc., have been developed to remove redundancy within the data before passing the data to the ECCs. But, still lot of redundancy remains in the feature rich language like data that can be used for error-correction. The technique proposed in this thesis is based on using redundancy present within the data for error-correction. The proposed technique can be combined with error correcting codes and better performance can be achieved.

The goal is to correct random erasures in the input text consisting of English text using the parts of speech tags property associated with the words in the text. The symbols in the text consist of alpha numeric character(a-z, A-Z, 0-9) and space-mark. Each sentence is a sequence of tokens separated by the word delimiter, space-mark.

Each token in a sentence is a group of symbols, consisting of only alphabets(a-zA-Z) and numbers(0-9). Within a single sentence, any character other than alphanumerical characters is considered as a word delimiter. Multiple methodologies have been implemented to solve this problem. The methodologies utilize the multiple features associated with the words like their word count, tag count, tags taken, two-gram count. Word count represents the frequency of the given token. Given a tag and token, tag count represents the frequency of appearance of that token with that tag. Tags taken represents the list of different tags taken by the token. Given two tokens $token1$ & $token2$ two-gram count indicates the frequency of the two-gram "$token1$ $token2$".

Using properties of the text that the data represents to determine the most probable value of the bits with erasures is not straight-forward. For example, If there are N tokens in the sentence, then there will be corresponding N bit sequences for each token which may contain erasures. If there are k erasure bits in the each of the bit sequence in the list of size N representing sentence, then there are 2^k possibilities for each of the bit sequences. Determining value of the erasure bits within each of the bit sequences independently does not provide good results as the tokens within the sentence from English language are not completely independent of each other. Each of the words within a sentence are present in a particular order rather than being independent of each other and together they form some meaning. From the considered example, there can be at-most 2^{Nk} possibilities of which only one of them represents the correct sequence of tokens. To be able to find the most probable one among all the 2^{Nk} possibilities within polynomial time makes it much more challenging.

The bit sequence corresponding to any given text can be obtained by encoding the symbols within the text using their Huffman codes. Similarly, any valid bit sequence can be decoded to retrieve the text that it represents. The kind of noise that will

cause erasures in the data is the erasure of uniformly random bits representing the data. The parameter Percentage of Erasure Bits(PEB) indicates the maximum percentage of erasure bits that can be present in the data. The erasures are assumed to be only within the tokens but not within any of the token delimiters. Within the complete bit-sequence representing the sentence, the position of erasures is known and the token boundaries are assumed to be known. As the token boundaries are known, the complete bit sequence can be divided into a sequence of bit subsequences where each of the subsequences represents a token. The input to the algorithm is the sequence of bit-sequences obtained after breaking down the complete bit sequence representing the sentence and ignoring the token delimiters. Each of the bit sequences obtained after breaking down the complete bit sequence may have erasures. If there were no erasures within the bit sequences, each of the bit sequences can be decoded to obtain the token it represents. The sentence that the data as a whole represents can be obtained by combining the decoded tokens from the bit sequences.

The ideal output from the algorithm should return the correct bit-value for each of the erasures. For a given sentence consisting of only the considered symbols, a list of tokens can be obtained by tokenizing the sentence and the corresponding list of bit sequence representing each of the tokens is obtained by encoding each of the symbols present with the tokens with their Huffman codes. If random bits are erased from the bit sequences in the list and are passed as input to the algorithm, then the algorithm should ideally generate correct values taken by each of bits with erasures. The result from the algorithm is considered correct, if after substituting the corresponding output bit values for each of the bits with erasures and then decoding the bit sequences generates the same tokens obtained after tokenizing the sentence considered in the beginning.

1.2 Related work

Error correction based on the properties of the text that the data represents is studied in [1] and it relates to the work described in this thesis in terms of the properties of the text used in the methodology to fix erasures. [1] proposes a new technique, *content assisted decoding*[1], which involves designing better channel decoders for correcting bit errors in text files. This technique is based on statistical properties of the words and phrases of the text in the language and the considered property is the probability with which the words and phrases appear in any given text. In this thesis, apart from probabilities of words, the parts-of-speech property associated with the text is used to find the correct values of bits with erasures using Hidden Markov Model.

The Parts-of-Speech tagging has been well studied in past as in [3],[8],[7],[5] . In the some of the considered approaches in the thesis, the most probable tag sequences representing the input sequence is utilized to determine the bit value of the erasures. A dynamic programming based approach similar to Viterbi algorithm[2] has been implemented to obtain the tag sequence.

1.3 Summary of results

Two different approaches, *pure word count* based approach and HMM based approach, have been developed & implemented to find the most probable value of the erasure bits. The performance of the HMM based approach has been better than *pure word count* based approach. In the HMM based approach there are multiple flavors of the algorithms like, one-step HMM based algorithm, two-step HMM based algorithm, Two-step HMM with two-gram words based algorithm. Among the different flavors within the HMM based approach, the performance of the two-step HMM with two-gram words has been better than the one-step HMM

and plain two-step HMM. The *pure word count* based approach relies only on the word count to determine the value of the erasure bits. The HMM based approaches involves generating the most probable tag sequence that can be taken by the input and determining the value of erasure bits using the decoded tag sequence and several other properties of text like, word count, tag count etc., The performance of each of the approaches have been measured against the value of PEB ranging from 5% to 50%. For 5% PEB, accuracy of 97.58%, 98.06%, 98.09% and 98.29% has been obtained for Pure word count based algorithm, one-step HMM based algorithm, two-step HMM based algorithm and two-step HMM with two-gram based algorithm respectively. Each of the methodologies and their performance is explained in detail in the following chapters.

2. MODEL FOR ERROR CORRECTION

If there are N tokens in the input sequence and if each of the N tokens can take at most k tags, then there can be N^k tag sequences possible.Each of the possible tag sequences represents a Markov chain. Hidden Markov model can be used to find the probability of Markov chains. Hence, HMM is used to find the probability of the tag sequences.

The Hidden Markov Model(HMM)[3] is a statistical model of a system assumed to be a Markov process with the unobserved states and observed outputs. The output of a given state is only dependent on that state but not on any of the previous states or previous outputs. But, the current state can be dependent on up-to previous k states. HMM has three different states, starting states, output states and hidden states, and three different parameters, Transition probabilities, Emission probabilities and starting probabilities. Starting state is the hidden state of the HMM at the starting of Markov chain. Hidden states are the states of HMM which result in the output that is observed and output states represent then states that are observed. The parameter, transition probability represents the probability of transition from one hidden state to another, emission probability represents the probability of observing the given output given the hidden state of the HMM and starting probabilities indicate the probability of observing a given starting state of HMM.

Two different HMMs,one-step HMM and two-step HMM, have been constructed as part of algorithm. In the one-step HMM model and two-step HMM model, the value of k is 1 and 2 respectively. In the HMM, given the sequence of output states, a corresponding most probable sequence of hidden states because of which the output is observed can be obtained. In the HMMs constructed, each hidden state and

starting state is one of the parts-of-speech tags from the Brown corpus[4] tag set or a combination of two tags depending on the value of k and output is one of the tokens from the vocabulary, where vocabulary is the set of different tokens present in the Brown corpus[4]. For a given Markov chain, using HMM the most probable sequence of hidden states which resulted in the given sequence of output states can be determined.

Experiments have been carried out assuming the tag sequences can be first order and second order Markov chains. In the following sections One-step HMM and two-step HMM are described.

2.1 One-step hidden Markov model

When the tag sequences are considered as first order Markov chains, the one-step HMM is used to find the most probable tag sequence among the possibilities. In the first order Markov chains, the i^{th} state is dependent only on the $(i-1)^{th}$ state and not on any of the previous states. The output of a given state is directly dependant only on the current state but not on any other states. There are three different states and three different parameters which are associated with the HMM. In this section, the states of HMM and the different parameters of the one-step HMM are described.

2.1.1 States of one-step HMM

HMM constitutes of three different types of states as follows: The hidden states, the output states and the starting states. The output states of the one-step HMM represent the observed output states. The hidden states of the HMM represent the states which result in the observed output. Finally, the starting states of the HMM represent the states at which the Markov process can begin. In this section, each of the states and the values they can take is explained.

The parts of speech tags used to tag the sentences in the Brown corpus form the

Table 2.1: List of all the tags present in the simplified tag-set along with their frequencies. Using this simplified tag-set the sentences in the Brown corpus in NLTK module are already tagged.

S.No	Tag	Frequency	S.No	Tag	Frequency
1	DET	121786	21	MOD	13345
2	NP	42817	22	*	4613
3	N	229475	23	EX	2280
4	ADJ	72068	24	:	1718
5	VD	26192	25	(	2426
6	P	122591	26	)	2457
7	“	8837	27	””	3431
8	”	8789	28	’	317
9	CNJ	60326	29	FW	1225
10	V	102164	30	UH	629
11	.	61254	31	VB+PPO	71
12	ADV	43934	32	NIL	157
13	,	58336	33	VB+IN	3
14	WH	14919	34	VBG+TO	17
15	VBZ	7469	35	VBN+TO	5
16	VN	29932	36	VB+RP	2
17	TO	14998	37	VB+JJ	1
18	PRO	66792	38	VB+VB	1
19	VG	18188	39	VB+TO	4
20	NUM	17024	40	VB+AT	2

Table 2.2: Sample output states in one-step HMM and their frequencies. Only 30 output states of the possible 56057 states are listed in this table.

S.No	Output state	Frequency	S.No	Output state	Frequency
1	The	7258	16	“	8837
2	Fulton	17	17	no	1781
3	County	85	18	evidence	201
4	Grand	18	19	”	8789
5	Jury	4	20	that	10237
6	said	1943	21	any	1301
7	Friday	60	22	irregularities	8
8	an	3542	23	took	425
9	investigation	43	24	place	528
10	of	36080	25	.(Full-stop mark)	49346
11	Atlanta’s	4	26	jury	63
12	recent	167	27	further	194
13	primary	93	28	in	19536
14	election	72	29	term-end	1
15	produced	90	30	presentments	1

hidden states of the HMM. There are 40 tags in the simplified version as listed in the Table-2.1. The simplified tags can be obtained using $simplify\_tags$ flag while accessing the corpus using NLTK module in python. Each of these tags are explained at the end of the section.

A token from the vocabulary of the Brown corpus constitutes the output states. There are 1161192 tokens in total in Brown corpus and of them 56057 are unique tokens. Every output state takes value of one of these tokens. Thirty of the output states are listed in the Table-2.2

Every Markov chain starts with one hidden state. Each such state represents a starting state. There are 33 starting states as listed in Table-2.3.

The simplified version of the Brown corpus consists of the 40 different tags. The meaning of each of the tags in discussed below.

Table 2.3: List of all starting states in one-step HMM and their frequencies .

S.No	Starting state	Frequency	S.No	Starting state	Frequency
1	DET	9529	18	VN	322
2	“	4168	19	VD	55
3	PRO	10937	20	MOD	153
4	N	3525	21	)	179
5	WH	1359	22	TO	239
6	ADV	3928	23	’	19
7	P	4848	24	*	156
8	V	3674	25	,	12
9	NUM	1021	26	.	60
10	(	353	27	FW	26
11	NP	4260	28	UH	146
12	CNJ	5013	29	”	89
13	””	219	30	VB+PPO	19
14	EX	839	31	VB+RP	2
15	VG	666	32	NIL	3
16	ADJ	1428	33	VB+IN	1
17	VBZ	61	34		

1. **N** : The tag *N* represents a noun. A noun denotes a person, animal, place, thing or an idea. The words Country, heaven are examples of Nouns.

2. **ADJ** : The tag *ADJ* represents an adjective. Adjectives modify or describe nouns. In the sentence, “It is a good place”. The token *good* is an adjective as it is qualifying the noun, *place*.

3. **CNJ** : The tag *CNJ* represents a conjunction. A conjunction is a joiner word which joins parts of sentence. and, or, nor are examples of conjunctions.

4. **DET** : The tag *DET* represents a determiner. A determiner precedes a noun and expresses the context of the noun. For example in the sentence “*My* pen is on the table”, the token *My* is a determiner as it determines who possess the pen. Few other examples are a, an, the, his, your, their.

5. **EX** : The tag *EX* represents an existential “there”. This tag is assigned to

the token *There*. For example, There is a book, The parts of speech tag assigned to the token *There* is *EX*.

6. **FW** : The tag *FW* represents a foreign word. This tag is assigned to the tokens which are not present in the English Language vocabulary. For example, in the sentence "You'll never hear sayonara, the Japanese word for goodbye, from your guests when you give a hibachi party.", the token *sayonara* is assigned the tag *FW* as the word belongs to Japanese language.

7. **MOD** : The tag *MOD* represents a modal verb. It denotes an auxiliary verb used to express the modality. possibility, obligation, likelihood, permission are examples of modalities. For example, in the sentence "I may play cricket.", the token *may* is a modal verb.

8. **ADV** : The tag *ADV* represents an adverb. Adverbs modify or describe verbs. For example, in the sentence "I really care.", the token *really* is assigned the tag *ADV*.

9. **NP** : The tag *NP* represents a proper noun. Name of a person, , animal, place, thing or an idea. For example, in the sentence "James is in New York.", the token *James* is assigned a tag of pronoun as it is a name of a person.

10. **NUM** : The tag *NUM* represents a quantifying value. fourth, 14:24, 100 are few examples of the tokens that are assigned tag of *NUM*.

11. **PRO** : The tag *PRO* represents a pronoun. Pronouns are small words that can replace nouns. For examples, in the sentence "James is good student and he is really good at math.", the token *he* is assigned the pronoun tag. it, that, he are few other examples of the tokens that are pronouns.

12. **P** : The tag *P* represents a preposition. Preposition represents relation between noun/pronoun and some other word in the sentence. For example, in the sentence "Apple is on the table.", the token *on* is a preposition as it represents the

relationship between the *Apple* and *Table*.

13. **TO** : The tag *TO* is assigned to the word *to* when it is preceding a verb. For example, in the sentence "He is going to investigate on this issue.", the token *to* is assigned the tag **TO**.

14. **UH** : The tag *UH* represents an interjection. It denotes a strong feeling, sudden emotion, introductory expression or sound. For example, in the sentence, "Yes, it is good.", the token *Yes* is assigned the tag UH. Hurrah, Oh, Alas are few other examples of the tokens that can be assigned the tag *UH*.

15. **V** : The tag *V* represents a verb. it denotes an action or state of being. For example, in the sentence "I can walk.", the token *walk* is assigned the tag *UH*. walk, run are other examples that can be assigned the tag *V*

16. **VD** : The tag *VD* a verb in its past-tense. It denotes an action that has happened or state that existed previously. For example, in the sentence "He took long time to finish.", the token *took* is assigned the tag *VD*. said, told are few other examples that can be assigned the tag *VD*.

17. **VG** : The tag *VG* represents a present participle. It typically denotes the verbs ending with -ing. For example, in the sentence "He is cooking.", the token "cooking"is assigned the tag *VG*. Walking, Running are few other examples of the tokens that can be assigned this tag.

18. **VN** : The tag *VN* represents a past participle. It typically denotes the verbs ending with -ed, -d, -t, -en, or -n. For example, in the sentence, The laminated copy of book is in the shelf, the token *laminated* is a past participle. conducted, charged are few other examples that can be assigned this tag.

19. **VH** : The tag *VH* represents a *wh* determiner. Who, which, when, where, what are few of the examples.

20. **VBZ** : The tag *VBZ* represents a 3rd person singular verb in present tense.

For example, in the sentence, "He cooks.", the token cooks is assigned the tag VBZ. eats, walks, deserves, believes are few other examples of the tag.

21. **VB+TO** : The tag *VB+TO* represents a verb in its uninflected present tense or infinite form or which creates an imperative sentence and immediately followed by to. For example, in the sentence "I wanna play guitar.", the token wanna is short form for "want to"and is assigned the tag *VB+TO*.

22. **VB+VB** : The tag *VB+VB* represents a pair of hyphen separated verbs in their uninflected present tense or infinite form or which creates an imperative sentence. For example, in the following sentence from Brown Corpus "Hoijer's Athabascan and my Yokuts share 71 identical meanings(with allowance for several near-synonyms like stomach-belly, big-large, long-far, many-much, die-dead, **say-speak**).", the token "say-speak"is assigned this tag.

23. **VB+PPO** : The tag *VB+PPO* represents a verb in its uninflected present tense or infinite form or which creates an imperative sentence and immediately followed by a pronoun. For example, in the sentence "Let's go there", the token Let's is assigned this tag as Let's is short form for *Let us*. gimme, lemme are other examples of the tokens to which this token is assigned.

24. **VB+AT** : The tag *VB+AT* represents a verb in its uninflected present tense or infinite form or which creates an imperative sentence and immediately followed by article. For example, in the following sentence from Brown Corpus, "Wanna beer?", the token Wanna is assigned this tag as the wanna is short form for "Want a".

25. **VBG+TO** : The tag *VBG+TO* represents a present participle immediately followed by a to. For example, in the following sentence from Brown corpus, "That was gonna be fun collecting!!", the token gonna is assigned this tag. gonna is short form for "going to"where going is a present participle and it is followed by to.

26. **VBN+TO** : The tag *VBN+TO* represents a present participle immediately

followed by a to. For example, in the following sentence from Brown Corpus ““When you gotta go, you gotta go”, Mrs. Forsythe said.”, the token gotta is assigned this tag. The token gotta represents the phrase “got to”, where got is a past participle and it is followed by *to*.

27. **VB+RP** : The tag *VB+RP* represents a verb followed by an adverbial particle. Adverbial particle is an adverb used especially after a verb to show position, direction of movement etc., For example, in the following sentence from Brown corpus “C’mon, buddy, help me set up the kitchen and we’ll have food in a minute or two.”, the token *C’mon* is assigned this tag as C’mon is short form for ‘come on’where come is verb and on is an adverbial particle.

28. **VB+JJ** : The tag *VB+JJ* represents a hyphen separated verb in its uninflected present tense or infinite form or which creates an imperative sentence and an adjective. For example, in the following sentence from Brown corpus, “Hoijer’s Athabascan and my Yokuts share 71 identical meanings(with allowance for several near-synonyms like stomach-belly, big-large, long-far, many-much, **die-dead**, say-speak).”, the token *die-dead* is assigned this tag.

29. **VB+IN** : The tag *VB+IN* represents a verb in its uninflected present tense or infinite form or which creates an imperative sentence and immediately followed by preposition. For example, in the following sentence from brown corpus “Lookit it!!”, the token Lookit is assigned this tag.

30. ***(asterisk)** : The tag **(asterisk)* represents a negator. For example: not, n’t.

31. **,(comma)** : The token “,(comma)”in a sentence is assigned this tag.

32. **“** : The token ““”in a sentence is assigned this tag.

33. **”** : The token “””in a sentence is assigned this tag.

34. **’** : The token “’(single-quote)”in a sentence is assigned this tag.

35. **"”** : The token "–"in a sentence is assigned this tag.

36. **":"** : The token ":(colon)"in a sentence is assigned this tag.

37. **"("** : The token "("in a sentence is assigned this tag.

38. **")"** : The token ")"in a sentence is assigned this tag.

39. **"."** : The token ".(Full-stop)"in a sentence is assigned this tag.

40. **NIL** : Some parts of the few sentences in the Brown corpus are not tagged and each of the tokens in those parts are assigned this token. This token does not have any specific meaning.

2.1.2 Parameters of one-step HMM

Transition probabilities, Emission probabilities, Starting probabilities are the three parameters that characterize the HMM. In this section the parameters of one-step HMM are defined and described.

Transition probability($P_T(t_i|t_j)$) is defined as probability of transition from one hidden state(Tag_j) to another hidden state(Tag_i). Starting probability($P_S(t_i)$) is defined as probability of observing a hidden state(Tag_i) as starting state of the process. Emission probability($P_E(w_i|t_i)$) is defined as probability of observing an output state($Token_i$) given an hidden state(Tag_i).

These parameters of HMM are derived from Brown corpus and are calculated using the below formulae, where the K is the number of different tags present in the corpus that are present at the beginning of the Markov chain. Table-2.4 and Table-2.5 show the sample data from which transition and emission probabilities can be calculated respectively.

$$P_S(t_i) = \frac{freq_s(t_i) + 1}{\sum_{j=1}^{K} freq_s(t_j) + K}$$

Table 2.4: List of hidden states to which the one-step HMM can transition to, given the current hidden state is 'VB+PPO'. Frequency of transition to a state is specified in the 'Frequency' column. For example, frequency of transition from state 'VB+PPO' to state 'V' is 16651

S.No	Next state	Frequency	S.No	Next state	Frequency
1	V	16651	18	.	2517
2	VD	9261	19	CNJ	1228
3	”	680	20	NP	195
4	DET	1560	21	“	145
5	VBZ	1544	22	WH	326
6	P	2686	23	*	161
7	N	12712	24	’	8
8	VG	665	25	””	171
9	VN	766	26	)	27
10	ADJ	4053	27	:	48
11	MOD	4999	28	EX	10
12	TO	665	29	FW	15
13	NUM	365	30	UH	5
14	PRO	393	31	VBG+TO	9
15	(	39	32	VB+TO	3
16	,	1835	33	VBN+TO	4
17	ADV	3039			

$$P_T(t_i|t_j) = \frac{freq(t_i, t_j) + 1}{freq(t_i) + 1}$$

$$P_E(w_i|t_i) = \frac{freq(w_i, t_i) + 1}{freq(t_i) + 1}$$

$P_S(t_i)$ = Probability of observing Tag_i at the starting of the sequence

$P_T(t_i|t_j)$ = Probability of transition from Tag_j to Tag_i in the sequence

$P_E(w_i|t_i)$ = Probability of emitting $Token_i$ as output of a state, given the state of the model is Tag_i.

$freq_s(t_i)$ = Frequency of observing Tag_i as the starting state.

$freq(t_i, t_j)$ = Frequency of observing Tag_i as current hidden state given the previous hidden state was Tag_j.

Table 2.5: List of output states that can be observed in the HMM, given the current hidden state is 'EX'. Frequency of generating the given output state is specified in the 'Frequency' column. For example, frequency of generating the output state 'there' from the hidden state 'EX' is 1353

S.No	output state	Frequency
1	there	1353
2	There	811
3	there's	52
4	There's	55
5	There'll	2
6	there'll	2
7	theare	1
8	ther	1
9	There'd	1
10	there'd	3

$freq(t_i)$ = Frequency of observing Tag_i as non-starting hidden state.

$freq(w_i, t_i)$ = Frequency of observing w_i as output given the current hidden state is Tag_i.

2.2 Two-step hidden Markov model

When the tag sequences are considered as second order Markov chains, the two-step HMM is used to find the most probable tag sequence among the possibilities. In the second order Markov chains, the i^{th} state is dependent only on the $(i-1)^{th}$ state and $(i-2)^{nd}$ state and not on any of the previous states. The output of a given state is directly dependant only on the current state but not on any other states. There are three different states and three different parameters which are associated with the HMM. In this section, the states of HMM, the different parameters of the two-step HMM is described.

2.2.1 States of two-step HMM

Similar to one-step HMM, the two-step HMM consists of three different states, Starting states, Hidden states and output states. The states of two-step hidden Markov model differ from one-step HMM. Although output states remain same in one-step HMM and two-step HMM, the starting states and hidden states differ from the one-step HMM model.

The hidden states in the two-step HMM is a '_' separated combination of tags from simplified tag-set of Brown corpus. Let T denote the tag sequence $\{t_1, t_2, t_3, ..., t_i, ..., t_n\}$ of length n associated with one of the sentences from Brown corpus and be one of the input tag sequences used to build one-step HMM. Every such T of length n in the input data is transformed into another corresponding sequence $T^{'}$, $\{t_1{}^{'}, t_2{}^{'}, ..., t_i{}^{'}, .., t_n - 1^{'}\}$, of length $n-1$. The obtained sequence, $T^{'}$, is used as input to build the two-step HMM. The i^{th} element, $t_i{}^{'}$, in $T^{'}$is obtained by concatenating t_i, '_'and t_{i+1}, where t_i and t_{i+1} are the i^{th}, $(i+1)^{th}$ tags in the sequence T. Let $t_i{}^{'}$ which represents the hidden state of the 2-step HMM be called $two - step - tag$. There are 848 such two-step-tags that are generated in this process and constitutes the complete set of hidden states for the two-step HMM.

In each of the sequences of two-step-tags, the two-step-tag seen at the beginning of the sequence form the starting state. There are in total 509 such states. In general, the starting states are a sub-set of hidden states.

2.2.2 Parameters of two-step HMM

The state starting probabilities, transition probabilities and emission probabilities are calculated as follows. If K represents the number of different starting states possible, $t_i{}^{'}$, $t_j{}^{'}$ represents i^{th} and j^{th} two-step-tag respectively, then,

$$P_S(t_i{}') = \frac{freq(t_i{}')}{\sum_{j=1}^{K} freq(t_j{}')}$$

$$P_T(t_i{}'|t_j{}') = \frac{freq(t_i{}', t_j{}')}{freq(t_i{}')}$$

$$P_E(w_i|t_i{}') = \frac{freq(w_i, t_i{}') + 1}{freq(t_i{}') + 1}$$

where,

$P_S(t_i{}')$ = Probability of observing two-step-tag $t_i{}'$ at the starting of the sequence

$P_T(t_i{}'|t_j{}')$ = Probability of transition from state $t_j{}'$ to $t_i{}'$ in the sequence

$P_E(w_i|t_i{}')$ = Probability of emitting $Token_i$ as output of a state, given the state of the model is $t_i{}'$.

$freq_s(t_i{}')$ = Frequency of observing two-step-tag $t_i{}'$ as the starting state.

$freq(t_i{}', t_j{}')$ = Frequency of observing two-step-tag $t_i{}'$ as current hidden state given the previous hidden state was two-step-tag $t_j{}'$.

$freq(t_i{}')$ = Frequency of observing two-step-tag $t_i{}'$ as non-starting hidden state.

$freq(w_i, t_i{}')$ = Frequency of observing w_i as output given the current hidden state is two-step-tag $t_i{}'$.

2.3 Generating parameters of HMM

To create HMM[3], it is required to get the data on all the possible hidden states, the different output states taken by each of the hidden states along with the probabilities of observing each of the output states given a hidden state, the transition of states from one hidden state to another hidden state along with the probability of transition and the all the starting hidden states along with their probabilities. In the

HMM used in the current approach, the parts-of-speech tags represent the hidden states and the tokens represent the output states. To be able to generate all this data required to model the HMM, any corpus whose data is tagged is required.

Brown corpus[4] is one such corpus whose sentences are tagged and can be easily accessed using the NLTK module[18] in python. Hence, Brown corpus is used to generate the HMM in the current approach. There are 1161192 tokens, 56057 unique tokens, 57340 sentences in the corpus. Each of the 57340 sentences present in the corpus is a sequence of token/tag objects, where token is the word in the sentence and tag is the parts-of-speech tag taken by that token. There are two versions of the tagged sentences available as part of python-NLTK module[18]. One version is tagged with only the simplified tags and the other version is tagged with two-step-tags. There are 40 tags in the simplified tag-set and 472 tags in the two-step-tag-set. The list of different tags present in the simplified tag set version are listed in Table-2.1

The HMM is generated using the simplified tag-set. The performance of the model generated with the simplified tag-set is better than the model generated with the two-step tagset. Any further reference to a tag-set in this thesis from now on indicates the use of simplified tag-set.

In the simplified version, the minimum number of tags assigned to a token is 1. The maximum number of tags assigned to a token is 6. Each of the 57340 sentences are processed one after the other and from the processed data, the parameters of the HMM are generated.

3. ALGORITHMS FOR ERROR CORRECTION IN ENGLISH TEXT

To find the correct value of the erasure bits, multiple methodologies have been developed & implemented which are as follows: pure word count based approach, one-step HMM based approach, two-step HMM based approach, two-step HMM with two-gram words based approach and BCJR algorithm[11] based approach. Each of these methodologies have different performance gains. At higher level all the different methodologies can be classified as either Pure word count based approach or HMM based approaches. Each of these methodologies are discussed in detail in the following sections. The first step to any of these approaches is to find list of tokens that every token with erasures in the input sequence can take. The rest of the steps are based on the approach that is followed. In the following sections, first the algorithm to generate the list of possible tokens is discussed and then each of methodologies is explained. The methodologies will generate the most probable value for every token with erasures in the input sequence. Using the obtained tokens, the value of the erasures bits within the input token is determined.

3.1 Generating list of possible tokens

To generate a list of tokens for every token with erasures in the input, every token is processed one by one. If a token contains erasures then each of the bits with erasures are substituted with both 0 and 1 and possible sequences are generated. If there are 'm'erasures in the token, then there can be at the maximum 2^m tokens in the list of possible tokens. Each of the generated 2^m bit sequences may or may not represent a valid token.

The bit sequence can be decoded by starting from the root node of the Huffman tree and taking a left branch for every '0'in the bit sequence and taking a right

branch for every '1'in the bit sequence until a leaf node is reached. Once a leaf node is observed, the value of the leaf node is stored and that represents a symbol in the message being decoded. We again start from the root node of the tree and continue processing the bits in the same way until all the bits are processed. If the complete bit sequence is successfully decoded then the bit sequence is classified as valid. Otherwise, it is classified as invalid. All the invalid bit sequences are discarded.

If the i^{th} token is being processed in the input sequence, then all the possible 2^m bit sequences are processed one after the other to generate the list, $list_i$, of the possible tokens that the i^{th} token can have. For every possible bit sequence, the token that the bit sequence represents is constructed in parallel while decoding the bit sequence. While decoding, we begin with an empty string and every time a leaf node is observed, the symbol is concatenated with the existing string. At the end of decoding if the bit sequence is valid, then the token is added to the W_i if the decoded token is valid. A token is considered valid if it is present in the dictionary of words generated from Brown corpus. Such a list of tokens is generated for each of the tokens in the input sequence. Finally a list of lists W is generated where $W = \{W_1, W_2,..,W_i,..,Wn\}$. The i^{th} list in the sequence consists of the possible tokens that the i^{th} token in the input can take and are generated using the process already described.

In general processing all the 2^m bit sequences is exponential in terms of number of bits. As the number of bits with erasures increases the processing time increases exponentially. The improvement in performance can be achieved by skipping the generation and processing of as many bit sequences as possible which do not represent a valid token. The main idea behind the improvement of efficiency is that if a prefix of the bit sequence ending at i^{th} position and substituted with either '0' or '1' for each erasure bit seen so far is invalid then the processing of the bit sequence from

the $(i+1)^{th}$ position can be skipped for that combination of the erasure bit(s) and the decoding can be resumed from one of the already seen erasure bits by assigning the next value that the erasure bit can take. The prefix is classified as invalid if the path followed using the bit sequence leads out of the trie.

In case of invalid prefix, the last seen erasure bit is substituted with the next possible value it can take. If the last seen erasure bit is already tried with value '1', then there is no possible value with which that erasure bit can be tried as every erasure bit is first tried with '0' and then '1'. In such case, the erasure bit previous to the last seen erasure bit is found and the same methodology is followed. If that erasure bit is already tried with '1' then the erasure bit previous to that bit is found and same methodology is followed till we find an erasure bit from where the decoding can be resumed. If an such an erasure bit is not found then the decoding is stopped.

The whole methodology can be implemented using two data structures, a complete binary tree and a binary trie. In the complete binary tree, every node takes value of 0 or 1 except root. The left child of a parent is node with value 0 and right child of a parent is a node with value 1. If there are m erasure bits in the bit sequence, then the height of the complete binary tree is m. Every path from root to leaf of the tree represents a combination of values that the erasure bits in the bit sequence can take. Using pre-order traversal all the root to leaf paths can be traversed. If the root of the binary tree is at level-0 and leaf nodes are at level m, then the node at $level - i$ in a path from root to leaf would give value of the i^{th} erasure bit. The decoding of bit sequence is done using a binary trie created using the Huffman encoded words present in the dictionary with the help of the already discussed complete binary tree. In this binary Trie, the path from root to leaf represents the Huffman encoded sequence of the token at that leaf and the value of any internal node is either zero or one. If a leaf node is reached after the last bit of the

bit sequence is processed, the the bit sequence represents a valid token. Otherwise, it represents an invalid token.

While traversing from root to leaves using pre-order traversal in the binary tree, the value of nodes at a given level in the binary tree is used as substitute for the value of corresponding erasure bit in the bit sequence. At every node in the complete binary tree the bit sequence till the position of the erasure bit that the level corresponds to is decoded. While decoding it is checked if the bit sequence, with erasures substituted with values from binary tree, till the position of the considered erasure bit can be successfully decoded or not. It can be successfully decoded as long as the sequence of bits in the bit sequence lead to one of the nodes in the binary trie. The decoding is unsuccessful, if the path followed in the binary trie using the bit sequence leads out of the Trie. If the decoding is unsuccessful after substitution of the value of the node(current node) at $level - i$ seen during pre-order traversal of the binary tree then the sub-tree rooted at that node is skipped and traversal is resumed from the right child of the parent of the current node if the current node is not its right child. Otherwise, the decoding is resumed from the nearest ancestor of the current node whose right child is not yet explored. If there is no such node from where the traversal of the binary tree can be resumed, then the decoding is stopped. If a leaf node is reached in the binary tree, the remaining bits in the bit sequence after the last erasure bit are decoded as well. If the bit sequence is completely decoded then the token obtained is stored for further processing.

3.2 Methodologies followed for error correction in text

In this section each of the methodologies are described in detail. Let I represents Huffman encoded input token sequence with erasures and $I = \{t_1, t_2, t_3, ..., t_{i-1}, t_i.., t_N\}$ where t_i is the Huffman encoded code word with erasures for tokens in the

input sentence.The desired output is O where, $O = \{o_1,\ o_2,\ o_3,...,o_i,...,o_N\}$ and o_i is the bit sequence without any erasures and can be decoded to a valid word in the dictionary. o_i is the corresponding output bit sequence to the input bit sequence t_i and the erasures in the bit sequence t_i can be found out by comparing the value of bits from o_i at the same position as bit with erasures in the input token. Each of the methodologies described in this section output the list $Y = \{y_1,\ y_2,\ y_3,\ ....,\ y_{i-1},\ y_i..,\ y_N\}$, where o_i from desired output list, O, is the Huffman encoded code-word of y_i. Each element y_i in the list is a token present in the dictionary. In the following sections, the pure word count based algorithm, HMM based algorithm and BCJR based algorithm are described.

3.2.1 Pure word count based algorithm

Every token t in the dictionary is associated with a word-count. The word-count of token t indicates the number of times the token t has appeared in the Brown corpus. The pure count based algorithm is based on the idea that higher the word count of a token the higher is the probability of token. Let the input sequence of N tokens be I and $I = \{t_1,\ t_2,\ t_3,\ ....,\ t_{i-1},\ t_i..,\ t_N\}$ where t_i is a bit sequence which may have bits with erasures. Let W represent the list of lists of tokens W_{list} and $W_{list} = \{W_1,\ W_2,\ W_3,....,W_i,...,W_N\ \}$ where W_i represents the list of valid that t_i can take. One of the final outputs generated by the algorithm is the list Y. For every t_i in the input list there is a corresponding token, y_i, in the output list of the algorithm. The output token y_i is calculated as follows

$$y_i = \max_{1<=j<=size(W_i)} (\ wordCount({w_i}^{(j)})\ )$$

where, ${w_i}^{(j)}$ is the j^{th} element in the list W_i, size(W_i) indicates the number of elements in the list W_i and $wordCount({w_i}^{(j)})$ gives the number of time the token ${w_i}^{(j)}$ has appeared in the Brown corpus.

Algorithm 1 Pure word count based approach

```
1: procedure PureWordCount(I,W_list)
2:     N ← Length of input sequence I
3:     Y ← {}
4:     for i ← 1 to N do
5:         L ← size of list W_i
6:         maxCount ← 0
7:         outputToken = ""
8:         for j = 1 to L do
9:             wc ← wordCount(w_i^(j))
10:            if maxCount < wordCount(w_i^(j)) then
11:                maxCount ← wc
12:                outputToken ← w_i^(j)
13:            end if
14:        end for
15:        y_i ← outputToken
16:        Y ← y_i
17:    end for
18:    return Y
19: end procedure
```

3.2.2 HMM based algorithm

The pure count based approach is very naive and often does not generate desired output as it only takes very limited information about the input which is word count of the tokens and generates the output. For example, if there is a two word sentence with erasures. Let the possibilities for first word be good, Hi, The and the possibilities for second word be day, The, there. Let the word counts of the tokens good, *Hi*, *The*, *day*, *there* be 50, 30, 100, 15, 40 respectively. If the pure count based approach is used, then the final list of tokens will be *The*, *The* as token *The* is the token with maximum word count. It can be observed that the better choice can be either "good day"or "Hi there". Further, If it was known that the tags of first and second word is adjective and noun respectively then "good day"would have been

very obvious choice. If the most probable tag sequence associated with a given input sequence can be determined, along with the word count and tag count information better results can be produced. The HMM based approach involves decoding the tag sequence and using the word count, tag count associated with a token to find the value of erasure bits.

Let the input sequence of N tokens be I and $I = \{t_1, t_2, t_3, ..., t_{i-1}, t_i.., t_N\}$ where t_i is a bit sequence which may have bits with erasures, W represent the list of lists of tokens W_{list} and $W_{list} = \{W_1, W_2, W_3,....,W_i,...,W_N\}$ where W_i represents the list of valid that the token t_i can take, the list of list of tags be T_{list} and $T_{list} = \{T_1, T_2, T_3,...,T_i,....,T_N\}$ where T_i is a list of tags that contains the list of tags where each tag in T_i is used tag at-least one of the tokens from the list W_i at-least once in the Brown corpus.

If there are N tokens in the input sequence with erasures and each of the tokens can take at-most q unique tags, then there are in total q^N tag sequences possible. Each of the possible tag sequences can represent a Markov chain and all these Markov chains can be seen in the HMM where the hidden states of the i_{th} stage are the tags present in the list T_i and the states in stage $(i+1)^{th}$ are the tags present in the list T_{i+1}. The most likely tag sequence can be found out efficiently by finding the most likely path from 1^{st} stage of the HMM to the N^{th} stage.

In the HMM based approach, the important step is to decode the most probable tag sequence using the HMM and then utilize the generated tag sequence to determine the list Y where $Y = \{y_1, y_2, y_3, ..., y_{i-1}, y_i.., y_N\}$. o_i from desired output list, O, is the Huffman encoded code-word of y_i where y_i is the i^{th} element in list Y. For every token t_i in the input list there is a corresponding token, y_i, in the output list of the algorithm which represents the correct value of t_i. Finally, t_i and o_i are compared to determine the correct value of the erasure bits. The output of the

decoding step of the algorithm is the most probable sequence of tags, TS_{output} and $TS_{output} = \{Tag_1, Tag_2, Tag_3,....,Tag_i,....,Tag_N\}$ The tags are decoded using both the types of HMMs, one-step HMM and two-step HMM, separately and in the end performance of the both the approaches is measured.

To generate the most probable tag sequence using HMM, firstly, the set of tags that every token in the input sequence can take is to be determined. In the following section, the algorithm to generate the set of tags is discussed. Followed by decoding of tags, then the algorithm to generate the most probable token value for the input token using the decoded tag sequence.

3.2.2.1 Generating set of tags

For every token in the input sequence, we require a list of possible tokens that the token can take and the list of tags that the token can take. For the input sequence I once the W_{list} is generated, the generated list is utilized to generate the list of list of tags T_{list} and $T_{list} = \{T_1, T_2, T_3,...,T_i,....,T_N\}$ where T_i is a list of tags that t_i from input sequence can take and is generated using W_i. let t be a valid token in the dictionary. For every such t in the dictionary there is a set of tags, T, associated with it. The token t is tagged in the Brown corpus with the each of the tags in the set T at-least once.

Initially, for every list W_i a corresponding empty list T_i is created. Let w_{ik} be the k^{th} token in the list W_i. Every token, w_{ik}, from the list W_i is taken one by one and the tag set, T, associated with that token is retrieved. Each of the tags in the set T are added to the list T_i if the tag to be added is not already present in the list. After all the tokens from the list W_i are processed, the final list T_i is generated. Thus generated set of tags for each token are used to obtain the most probable sequence of tags using HMM. In that final sequence of tags, one tag is picked from every list

in T_{list}.

3.2.2.2 Decoding tags using one-step HMM

In the one-step HMM based approach, the tag sequences are considered as first order Markov chains. In the one-step HMM the next state is dependent only on the current state but not on any other states. The most likely sequence of tags can be efficiently determined from HMM using a dynamic programming approach. All the required parameters are already determined as described in section 2.2.2

Let T_{size} represent the size of tag set present in the Brown corpus. Let $IndexToTagMap$ represent a map of number to tags, where every tag is assigned a unique index value. The tags are assigned values from 1 to T_{size}. Similarly, let $TagToIndexMap$ represent the map of tags to their corresponding unique index value. Using IndexToTagMap we can retrieve the tag name given its index value and using TagToIndexMap, the index value of the tag name can be retrieved given the tag name.

Let $TagSet(w)$ represent the set of tags taken by token w, $WordCount(w)$ represent the number of times the token w appears in Brown corpus and $TagCount(w,t)$ represents the number of times the token w appears with tag t.

$$P_t(i,j) = P_T(\ IndexToTagMap[i] \mid IndexToTagMap[j]\ )$$

$$P_s(k) = P_S(\ IndexToTagMap[k]\ )$$

Let ${W_i}^{(j)}$ represent the list of tokens which can take tag IndexToMap(j) from the list W_i and ${w_{ik}}^{(j)}$ represents k^{th} element in the list ${W_i}^{(j)}$. Then,

$$W(i,j) = \max_{1 \le k < |W_i^{(j)}|} (WordCount(w_{ik}^{(j)}))$$

A two-dimensional matrix, F, of size ($(N+1)$ x T_{size+1}) is considered and initialized to zero, where N represents the number of tokens in the input sequence I and T_{size} represents the size of the tag-set of Brown corpus. An element F(i,j) represents

Algorithm 2 Creating Tag Index Maps

```
1: function CreateTagToIndexMap(TagSet)
2:     i ← 0
3:     tagToIndexMap ← empty map of integers to strings
4:     for t doag in TagSet
5:         tagToIndexMap[tag] = i;
6:         i gets i+1
7:     end for
8:     return tagToIndexMap
9: end function
10:
11: function CreateIndexToTagMap(tagToIndexMap)
12:     indexToTagMap ← empty map of strings to integers
13:     for key in tagToIndexMap do
14:         indexToTagMap[key] = tagToIndex[key]
15:     end for
16:     return indexToTagMap
17: end function
```

the probability of most probable decoded tag sequence for the first i observations and having tag with index j as the last state whose output is the i^{th} observation for F(i,j). Once the complete matrix is calculated the final tag sequence can retrieved using back-tracking procedure.

$$f(i,j) = \begin{cases} 0, & \text{i=0 or j=0} \\ P_s(j) * W(i,j), & \text{i=1} \\ \underset{1 \le k \le T_{size}}{max} (f(i-1,k) * P_t(k,j) * W(i,j)), & \text{otherwise} \end{cases}$$

Back-tracking as below to retrieve the Tag-indices of the most probable tag sequence,

Algorithm 3 One-step HMM based decoding

function DECODING_ONESTEPHMM(I,W_{list},T_{list},P_T, P_S, $TagSet$)
  f be two-dimensional matrix of size with $N+1$ rows and T_{size} columns
  $rows \leftarrow N+1$
  $cols \leftarrow T_{size}$
  $TS_{output} = \{\}$
  TagToIndexMap ← createTagToIndexMap(TagSet)
  IndexToTagMap ← createIndexToTagMap(TagToIndexMap)
  for $i \leftarrow 0$ *to* $rows$ **do**
    for $j \leftarrow 0$ *to* $cols$ **do**
      $W(i,j) = \max_{1 \le k < |W_i^{(j)}|} (WordCount(w_{ik}^{(j)}))$

      if i = 0 or j = 0 **then**
        f(i,j) = 0;
      else if i = 0 **then**
        $P_s(i,j) = P_S(\ IndexToTagMap[j]\ )$
        f(i,j) = $P_s(j) * W(i,j)$;
      else
        $P_t(i,j) = P_T(\ IndexToTagMap[i] \mid IndexToTagMap[j]\ )$
        $f(i,j) = \max_{1 \le k \le |T_{size}|} (P_t(i,j) * W(i,j) * f(i-1,k))$

      end if
    end for
  end for

  for $i \leftarrow N$ *to* 1 **do**
    if $i = N$ **then** $TagIndex_i = \underset{1 \le k \le T_{size}}{argmax}\ F(i,k)$
    else $TagIndex_i = F(i, TagIndex_{i+1})$
    end if
    TS_{output} ← IndexToTagMap($TagIndex_i$)
  end for
  return TS_{output}
end function

$$TagIndex_N = \underset{1 \le k \le T_{size}}{argmax}\ F(N, k)$$
$$TagIndex_{N-1} = F(N-1,\ F(N, TagIndex_N)\ )$$

Finally, $TS_{output}[i] = IndexToTagMap(\ TagIndex_i\ )$.

3.2.2.3 Decoding tags using two-step HMM

The performance of the HMM based algorithm increases with the decoding accuracy of the most probable tag sequence for the input. The better accuracy of decoding can be achieved by considering two-step HMM instead of one-step HMM because the current state depends on two of the immediately previous states rather than only the immediately previous states. The parameters of two-step HMM are derived as described in the section 2.2.2

Let cT_{size} represent the size of two-step-tag set generated from the Brown corpus, $cIndexToTag\text{-}Map$ represent a map of number to a two-step-tag, where every two-step-tag is assigned a unique index value. The two-step-tags are assigned values from 1 to cT_{size}. Let $cTagToIndexMap$ represent the map of two-step-tags to their corresponding unique index value. Using cIndexToTagMap we can retrieve the two-step-tag name given its index value and using cTagToIndexMap, the index value of the two-step-tag name an be retrieved given the two-step-tag name. All the references to the tags in this section represent a two-step-tag unless specified explicitly. The original tags present in the Brown corpus are referred as native tags in this section.

Let $TagSet(w)$ represent the set of tags taken by token w, $WordCount(w)$ represent the number of times the token w appears in Brown corpus and $TagCount(w, t)$ represents the number of times the token w appears with tag t. $FirstTag(tag)$ represent the first native tag in the two-step-tag and $SecondTag(t)$ represent the second native tag in the two-step-tag.

Algorithm 4 Creating Tag Index Maps for two-step HMM

```
1: function CreateTagToIndexMap(Two-step Tag-set)
2:     i ← 0
3:     cTagToIndexMap ← empty map of integers to strings
4:     for t doag in TagSet
5:         cTagToIndexMap[tag] = i;
6:         i ← i+1
7:     end for
8:     return cTagToIndexMap
9: end function
10:
11: function CreateIndexToTagMap(cTagToIndexMap)
12:     cIndexToTagMap ← empty map of strings to integers
13:     for key in cTagToIndexMap do
14:         cIndexToTagMap[key] = cTagToIndex[key]
15:     end for
16:     return cIndexToTagMap
17: end function
```

$$P_t{}'(i,j) = P_T{}'(\ cIndexToTagMap[i] \mid cIndexToTagMap[j]\ )$$

$$P_s{}'(k) = P_S{}'(\ cIndexToTagMap[k]\ )$$

Let $W_i{}^{(j)}$ represent the list of tokens which can take tag SecondTag$cIndexToMap(j)$from the list W_i and $w_{ik}{}^{(j)}$ represents k^{th} element in the list $W_i{}^{(j)}$. Then,

$$W(i,j) = \max_{0 \le k < |W_i{}^{(j)}|} (WordCount(w_{ik}{}^{(j)}))$$

$W(i,j)$ represents the maximum of the word counts of the tokens from the list W_i which can take tag SecondTag(IndexToTagMap[j]) at-least once.

Similar to the decoding of tags in one-step HMM, A two-dimensional matrix, F, of size ((N) x cT_{size+1}) is considered and initialized to zero, where N represents the number of tokens in the input sequence I and as already described cT_{size} represents the size of the two-step-tag set generated from Brown corpus. An element F(i,j)

Algorithm 5 Two-step HMM based decoding

function DECODING_TWOSTEPHMM(I,W_{list},T_{list},$P_T{}'$, $P_S{}'$, $cTagSet$)
  f be two-dimensional matrix of size with $N+1$ rows and T_{size} columns
  $rows \leftarrow N$
  $cols \leftarrow cT_{size}$
  $TS_{output} = \{\}$
  cTagToIndexMap ← createTagToIndexMap(cTagSet)
  cIndexToTagMap ← createIndexToTagMap(cTagToIndexMap)
  for $i \leftarrow 0$ *to* $rows$ **do**
    for $j \leftarrow 0$ *to* $cols$ **do**

      SecondTag ← Second native tag of cIndexToTagMap[j]
      maxWordCount ← 0
      for $k \leftarrow 1$ *to* $\mid W_i^{(j)} \mid$ **do**
        $w_{ik}^{(j)} \leftarrow k^{th}$ element in ${W_i}^{(j)}$
        wTagList ← list of tags taken by token ${w_{ik}}^{(j)}$
        wordCount($w_{ik}^{(j)}$) ← word count ${w_{ik}}^{(j)}$in Brown corpus.
        if SecondTag *in* wTagList **then**
          if $maxWordCount < wordCount({w_{ik}}^{(j)})$ **then**
            maxWordCount = $wordCount({w_{ik}}^{(j)})$
          end if
        end if
      end for

      $W(i,j)$ = maxWordCount
      if i = 0 or j = 0 **then**
        f(i,j) = 0;
      else if i = 0 **then**
        $P_s(j) = P_S(\ IndexToTagMap[j]\ )$
        f(i,j) = $P_s(j) * W(i,j)$;
      else
        $P_t(i,j) = P_T(\ IndexToTagMap[i] \mid IndexToTagMap[j]\ )$
        $f(i,j) = \max\limits_{1 \le k \le |T_{size}|} (P_t(i,j) * W(i,j) * f(i-1,k))$

      end if
    end for
  end for

```
    for i ← N to 1 do
        cTagSeq ← empty list to store the final N-1 decoded two-step-tags
        if i = N then
            TagIndex_i = argmax_{1≤k≤T_size} F(i,k)
        else
            TagIndex_i = F(i, TagIndex_{i+1})
        end if
        cTagSeq ← IndexToTagMap[ TagIndex_i ]
    end for

    for i ← 1toN do
        firstTag_i ← first native tag of the cDecodedTags[i]
        secondTag_i ← second native tag of the cDecodedTags[i]
        if i = 1 then
            TS_output[i] ← firstTag_i
        else if i = 2 then
            TS_output[i] ← secondTag_{i-1}
        else
            TS_output[i] ← secondTag_i
        end if
    end for
    return TS_output
end function
```

represents the probability of most probable decoded two-step-tag sequence for the first i observations and having tag with index j as the last state whose output is the i^{th} observation for F(i,j). Once the complete matrix is calculated the final tag sequence can retrieved using back-tracking procedure.

$$f(i,j) = \begin{cases} 0, & \text{i=0 or j=0} \\ P_s(j) * W(i,j), & \text{i=1} \\ \max\limits_{1 \le k \le T_{size}} (f(i-1,k) * P_t(k,j) * W(i,j)), & \text{otherwise} \end{cases}$$

Similar to one-step HMM based decoding, back tracking is done to retrieve the Tag-indices of the most probable two-step-tag sequence,

$$TagIndex_{N-1} = \underset{1 \le k \le T_{size}}{argmax}\ F(N-1, k)$$
$$TagIndex_{N-2} = F(N-2,\ F(N, TagIndex_{N-1})\ )$$

The sequence of native tags are generated from two-step-tag. Let $cTagsSeq$ represent the indices of sequence of two-step-tags obtained from HMM and cTagSeq = $\{TagIndex_1,\ TagIndex_2,..,\ TagIndex_i,..,\ TagIndex_{N-1}\}$. From the $cTagsSeq$, TS_{output} list as described below.

$$TS_{output}[i] = \begin{cases} FirstTag(\ cIndexToTagMap[\ TagIndex_1\ ]), & \text{i=1} \\ SecondTag(\ cIndexToTagMap[TagIndex_1\ ]), & \text{i=2} \\ SecondTag(\ cIndexToTagMap[TagIndex_{i-1}]\ ]), & \text{otherwise} \end{cases}$$

3.2.2.4 Determining the most probable value of every input token with erasures using decoded tags

Using the most probable sequence of tags, TS_{output}, the most likely choice for the tokens with erasures is determined. All the possible choices for input t_i are present in the list W_i. Using tag sequence, a list of list of tokens W'_{list} is generated and $W'_{list} = \{W'_1, W'_2,....,W'_i,...,W'_N\}$ where $W_i{}'$ is list of tokens obtained by selecting the tokens from W_i which can take tag $TS_{output}[i]$. If there is more than one tag that can take the decoded tag, then the token with maximum count of that particular decoded tag is chosen. The likely choice for t_i would be from W'_i. As described

Algorithm 6 Determining most probable value of tokens

function FINALTOKENSEQ($W_{list}{}^{'}$,TS_{output})
  $Y \leftarrow$ empty list to store final seq of tokens
  for $i \leftarrow 1$ *to* N **do**
    $W_i{}^{'} \leftarrow i^{th}$ element in list $W_{list}{}^{'}$
    $w_i{}^{k} \leftarrow k^{th}$ element in list $W_i{}^{'}$
    $y_i \leftarrow \max_{1 \leq k < |W_i'|} WordCount(\ w_i{}^{k}\ )$
    $Y \leftarrow y_i$
  end for
  return Y
end function

earlier, let Y represent the most likely choice for each input token where, $Y = \{y_1, y_2, y_3,...,y_i,...,y_N\}$ and y_i is the most likely choice for t_i. If $w_i{}^{k}$ is k^{th} element in list W_i' and $WordCount(w,t)$ represents the number of times the token w appears in the Brown corpus then,

$$y_i = \max_{1 \leq k < |W_i'|} WordCount(\ w_i{}^{k}\ )$$

3.2.2.5 Improving the performance using two-gram data

Using two-step HMM improved the accuracy of decoded tag sequence and using more advanced HMMs like, three-step HMM or four-step HMM would not have significantly improved the accuracy of decoded tag sequence. In the previous methodologies, the most probable tokens for the given input token is selected based on the individual words. No other information related to any of the choices for the tokens before and after has been considered. From the given choices, if a choice for current token can form a good two-gram with any of the choices for the next token, such a choice should be more probable than the other choices for that token. Hence, additional weight has been added to every token that can make good two-gram with the help of choices from the next token.

Let twoGramCount(token1, token2) represent the frequency of the two-gram, "token1 token2". For given token t, and the index i of the input seq for which t is one of the valid choices,

$$\text{two-gram-weight}(t,\ i\ ) = \sum_{1 \le k \le |W_{i+1}{}^{(TS_{output}[i+1])}|} \text{twoGramCount}(\text{t},\ w_{i+1,k}{}^{(TS_{output}[i+1])})$$

$$y_i = \max_{1 \le k < |W_i^{'}|} (0.8\text{*WordCount}(\ w_i{}^k) + 0.2\text{*(two-gram-weight}(w_i{}^k,i))\ )$$

The data required to generate the two-grams has been extracted from the two-gram collection of the corpus of contemporary American English(COCA)[9]. The COCA consists of more than 450 million words of text and is updated regularly. It is one of the largest freely available corpus of English language. The two-grams collection from COCA is processed. Only the two-gram consisting of alpha-numerical symbols are considered and rest of them are discarded.

Algorithm 7 Finding most probable tokens using two-gram words

function TWO-GRAM-WEIGHT(t,i)
  count $\leftarrow \sum_{1 \le k \le |W_{i+1}{}^{(TS_{output}[i+1])}|}$ twoGramCount(t, $w_{i+1,k}{}^{TS_{output}[i+1]}$)
  return count
end function

function FINDTOKENSUSINGTWOGRAM($W_{list}{}^{'}$)
  $Y \leftarrow$ Empty list used to store the most probable tokens for each input token.
  $N \leftarrow$ length of $W_{list}{}^{'}$
  for $i \leftarrow 1$ *to* N **do**
    $W_i{}^{'} \leftarrow i^{th}$ element in $W_{list}{}^{'}$
    $y_i = \max_{1 \le k < |W_i^{'}|} (0.8 * WordCount(\ w_i{}^k) + 0.2 * (two-gram-weight(w_i{}^k, i))\ )$
    $Y \leftarrow y_i$
  end for
end function

3.2.3 Algorithm based on minimizing bit error rate

The approach described in this section is based on BCJR algorithm[11]. The algorithm is named after its authors Bahl, Cocke, Jelinek and Raviv. Using Viterbi algorithm based approach, the word error rate is minimized. But, using BCJR algorithm bit error rate is minimized. The dynamic programming based approach described in section-3.2.2.2 and section-3.2.2.3 is similar to Viterbi algorithm and the most probable sequence of tags is determined minimizing the tag error rate. In BCJR algorithm, the value of each of the erasure bits is determined individually by finding the probability of each bit with erasure being 0 and 1.

For a given input list I containing list of bit sequences where every bit sequence is Huffman encoded code word with erasures of a token in the sentence that the input list represents. As described in section-3.2.2.2 and in section-3.2.2.3, the most probable sequence of tags, TS_{output} and $TS_{output} = \{Tag_1, Tag_2, Tag_3,....,Tag_i,....,Tag_N\}$, that input list can take is generated. The Tag_i in the TS_{Output} list represents the tag taken by i^{th} token in the input. As described in section-3.1 a list $W_{list} = \{W_1, W_2,..,W_i,..,Wn\}$ is generated where W_i is the list of tokens that the bit sequence t_i from input list, I, can represent. Using TS_{Output} and W_{list}, a list $W_{list}{}'$ can be generated. $W'_{list} = \{W'_1, W'_2,....,W'_i,...,W'_N\}$ where $W_i{}'$ is list of tokens obtained by selecting the tokens from W_i which can take tag $TS_{output}[i]$.

The posterior probabilities of the bits with erasures can be found out by representing the possible sequences using trellis diagram and algorithm similar to BCJR. The possible sequences can be represented using trellis diagram because for every bit sequence in the input there is a fixed set of possibilities from which the value of erasures within that bit sequence can be determined. Every possibility for the i^{th} bit sequence in the input list, I, is preceded by one of the elements from the set

representing possibilities for $(i-1)^{th}$ bit sequence except for the 1^{st} bit sequence in the input as each of the elements in the $W_0{}'$ represent a starting state. Each of the elements in the set $W_i{}'$ representing the possibilities for i^{th} element in the input represents a state in the i^{th} stage of the trellis diagram. If all the stages are number from 0 to N-1, then for i = 1,2,..,N-1 there is an edge from every state in $(i-1)^{th}$ stage to every state in i^{th} stage and the weight of the edge connecting a state, s', in the $(i-1)^{th}$ stage to a state, s, in the i^{th} stage is the probability of transition from state s' to state s. All the required transition probabilities are generated from two-gram data set present in COCA corpus as described in section-3.2.2.5.

The probability of a state can be determined by taking sum of probabilities of all the paths that pass through the considered state. Thus obtained probabilities for each of the states can be used to determine probability of an erasure bit being 0 and 1. If there are k erasure bits in the i^{th} bit sequence in the input, then Huffman encoded code words of all the states in the i^{th} stage of described trellis will have value of either 0 or 1 for each of the k erasure bits. For each of the k erasure bits in the i^{th} input element, all the states in the i^{th} stage can be divided two sets. One set will have the states whose Huffman encoded code words will have value of 0 at the position of the considered erasure bit and the other set will have value of 1. For the k^{th} erasure bit in the i^{th} input bit sequence, let the former set of tokens be denoted as ${WS_{ik}}^{(0)}$ and the later set of tokens be denoted as ${WS_{ik}}^{(1)}$. The probability of k^{th} erasure bit in the i^{th} input bit sequence being 0 can be obtained by taking sum of probabilities of all the elements in the set ${WS_{ik}}^{(0)}$. Similarly the probability of k^{th} erasure bit in the i^{th} input bit sequence being 1 can be obtained by taking sum of probabilities of all the elements in the set ${WS_{ik}}^{(1)}$.

The sum of the probabilities of all the paths that pass through a given state can be obtained efficiently using dynamic programming based approach. In this

approach, every state can be associated with two different probabilities, Forward probability and Backward probability. For any given state s, the forward probability is the sum of the probabilities of all the path starting from any of the states in 0^{th} stage and ending at state s. For the same state 's' the backward probability is sum of probabilities of all the paths starting at state 's' and ending at any of the states in stage $N-1$. For each of the stages, the forward and backward probability of the state 's' in the i^{th} stage can be denoted as $\alpha_i(s)$ and $\beta_i(s)$ respectively. The algorithms requires two iterations to compute all the forward and backward probabilities. Let $P_i(s)$ represent the probability of the state 's' in the i^{th} stage and $PE_{ij}^{(0)}$, $PE_{ij}^{(1)}$ represents the probability of j^{th} erasure bit in the i^{th} bit sequence in the input being 0 and 1 respectively.

Theorem 1 *For the given input sequence* $I = \{t_1,\ t_2,\ t_3,\ ....,\ t_{i-1},\ t_i..,\ t_N\}$, *If* $P(S_i = s|I)$ *the probability of the* i^{th} *state being* 's', $\alpha_k(s^{'})$ *represents the sum of probabilities of all paths starting at source and ending at state* $s^{'}$ *in the* k^{th} *stage,* $\beta_k(s)$ *represents the sum of probabilities of all paths starting at state* s *in the* k^{th} *stage,* $W_i^{'}$ *represents the states present in the* k^{th} *stage and* $P_T(s^{'}, s)$ *represents probability of transition from state* $s^{'}$ *to* s *then*

$$P(S_i = s|I) = L * \sum_{s^{'} \epsilon W_{i-1}{}^{'}} \alpha_{i-1}(s^{'}) * P_T(s^{'}, s) * \beta_i(s), \textit{where } L \textit{ is constant.}$$

Proof :

The conditional probabilty $\mathrm{P}(S_i = \mathrm{s} \mid I_1^n)$ can be represent using a joint probability as follows:

$$\mathrm{P}(S_i = \mathrm{s} \mid I_1^n) = \mathrm{P}(S_i = \mathrm{s}; I_1^n)/\mathrm{P}(I_1^n)$$

Let,

$$\lambda_i(s) = \mathrm{P}(S_i = \mathrm{s}; I_1^n)$$

$$\alpha_i(s) = \mathrm{P}(S_i = \mathrm{s}; I_1^i)$$

$$\beta_i(s) = \mathrm{P}(I_i^n \mid S_i = \mathrm{s}\ )$$

$\lambda_i(s)$ can be defined as follows:

$$\begin{aligned}\lambda_i(s) &= \mathrm{P}(S_i = \mathrm{s}; I_1^i\ )^*\mathrm{P}(I_i^n \mid S_i = \mathrm{s}\ ) \\ &= \alpha_i(s)^*\beta_i(s)\end{aligned}$$

$$\begin{aligned}\alpha_i(s) &= \sum_{s^{'}\epsilon {W_{i-1}}^{'}} P(S_{i-1} = s^{'}, S_i = s; I_1^i) \\ &= \sum_{s^{'}\epsilon {W_{i-1}}^{'}} P(S_{i-1} = s^{'}; I_1^{i-1} * P(S_i = s, I_i | S_{i-1} = s^{'}) \\ &= \sum_{s^{'}\epsilon {W_{i-1}}^{'}} \alpha_{i-1}(s^{'}) * P_T(s^{'}, s)\end{aligned}$$

Finally after subsituting the recursive formula for α_i,

$$\lambda_i(s) = \sum_{s^{'}\epsilon {W_{i-1}}^{'}} \alpha_{i-1}(s^{'}) * P_T(s^{'}, s) * \beta_i(s)$$

It is not required to calculate the value of $\mathrm{P}(I_1^n)$ as it is constant and same for all the states. If the value of the constant is L, then,

$$P(S_i = s|t) = \mathrm{L}^* \sum_{s^{'}\epsilon {W_{i-1}}^{'}} \alpha_{i-1}(s^{'}) * P_T(s^{'}, s) * \beta_i(s)$$

Theorem 2 *For the given input sequence* $I = \{t_1,\ t_2,\ t_3,\ ....,\ t_{i-1},\ t_i..,\ t_N\}$, *the probability of the bit with erasure being 0 and 1 is the sum of probabilities of all the paths that pass through that erasure bit with the value of that bit being 0 and 1 respectively. If* ${PE_{ik}}^{(0)}$, ${PE_{ik}}^{(1)}$ *represents the probability of* k^{th} *erasure bit in the* i^{th} *stage respectively,* $WS_{ik}^{(0)}$ *and* $WS_{ik}^{(1)}$ *represents the set of states which have the value of 0 and 1 for corresponding position of* k^{th} *erasure bit in the* i^{th} *stage respectively, then*

$$\begin{aligned}{PE_{ik}}^{(0)} &= \sum_{s^{'}\epsilon {WS_{ik}}^{(0)}} P(S_i = s^{'}|I) \\ {PE_{ik}}^{(1)} &= \sum_{s^{'}\epsilon {WS_{ik}}^{(1)}} P(S_i = s^{'}|I)\end{aligned}$$

Proof : The probability of a state s in any given stage can be determined as shown in Theorem-1. Thus obtained probabilities for each of the states can be used to determine probability of an erasure bit being 0 and 1. If there are k erasure bits in the i^{th} stage, then Huffman encoded code words of all the states in the i^{th} stage of described trellis will have value of either 0 or 1 for each of the k erasure bits. The probability of k^{th} erasure bit in the i^{th} stage being 0 can be obtained by taking sum of probabilities of all the states in the in the set ${WS_{ik}}^{(0)}$ because all the paths that pass through the tokens in the set ${WS_{ik}}^{(0)}$ because all the paths add to the probability of the the corresponding erasure bit being 0. Similarly the probability of k^{th} erasure bit in the i^{th} stage being 1 can be obtained by taking sum of probabilities of all the elements in the set ${WS_{ik}}^{(1)}$.The sets, ${WS_{ik}}^{(0)}$, ${WS_{ik}}^{(1)}$ are mutually exclusive. Each one of the states in the stage i belong to only one of these sets.

Hence,

$$
\begin{aligned}
{PE_{ik}}^{(0)} &= \sum_{s^{'} \epsilon {WS_{ik}}^{(0)}} P(S_i = s^{'}|I) \\
{PE_{ik}}^{(1)} &= \sum_{s^{'} \epsilon {WS_{ik}}^{(1)}} P(S_i = s^{'}|I)
\end{aligned}
$$

Theorem 3 *Assigning the value to the bits with erasures using the BCJR type algorithm on a Markov model based probability model minimizes bit error rate.*

Proof : Let X = $\{x_1, x_2, x_3,..,x_{n-1}, x_n\}$ represents the seqeunce of n bits. Let Y = $\{y_1, y_2, y_3, ..,y_{n-1}, y_n\}$ be a sequence of n elements where y_i corresponds to the element x_i in the input. The value of y_i is 1, if the final value of x_i is incorrect after error correction. Otherwise, value of 0 is assigned. The sum of elements in Y indicates the number of bits with incorrect values after error correction. Hence, Bit Error Rate(BER) can be defined as fraction of bits whose value is incorrect after error correction.

$$\text{BER} = \frac{\Sigma y_i}{n}$$

As BER is directly proportional to Σy_i, minimizing BER involves minimizing the expected value of Σy_i.

$$\text{BER} = \frac{Expected(\Sigma y_i)}{n} \equiv \frac{E(\Sigma y_i)}{n}$$

By linearity,

$$\text{BER} = \frac{\Sigma E(y_i)}{n}$$

In Theorem-2, determining the probability of each of erasure bits being 0 and 1 is shown. By assigning the value based on the probability value, the probability of the erasure bit being incorrect is minimized which inturn leads to the minimization of bit error rate. Hence, proved.

Let 's' represent one of the states in the i^{th} stage. The probability of s, $P_i(s)$, can be calculated as follows,

$$P_i(s) = \sum_{s' \epsilon W_{i-1}{}'} (\alpha_{i-1}(s') * P_T(s' \ , \ s) * \beta_i(s))$$

The values of the α and β can be calculated using recursive approach. It would require two iterations for the algorithm to find all the α and β values. Let $P_w(s)$ represents the probability of token s appearing in Brown corpus. α and β values can be calculated as follows.

$$\alpha_i(s) = \begin{cases} P_w(s), & \text{i=0} \\ \sum_{s' \epsilon W_{i-1}{}'} \alpha_{i-1}(s') * P_T(s', s), & \text{Otherwise} \end{cases}$$

Similarly,

$$\beta_i(s) = \begin{cases} P_w(s), & \text{i=0} \\ \sum_{s' \epsilon W_{i+1}{}'} \beta_i(s) * P_T(s, s'), & \text{Otherwise} \end{cases}$$

The probabilities of erasure bits being 0 and 1 can be calculated as follows.

$$PE_{ij}^{(0)} = \sum_{s \ \epsilon \ WS_{ij}{}^{(0)}} P_i(s)$$
$$PE_{ij}^{(1)} = \sum_{s \ \epsilon \ WS_{ij}{}^{(1)}} P_i(s)$$

where $PE_{ij}^{(0)}$, $PE_{ij}^{(1)}$ represents the probability of j^{th} erasure bit in the i^{th} bit sequence in the input being 0 and 1 respectively. The transition probabilities are calculated using the *absolute discounting smoothing*[10] method using the data from the Brown corpus and COCA.

Since the COCA two gram set is collection top million two gram rather than the complete corpus, the frequencies from the COCA cannot be used directly. Only the probabilities of two gram words are derived from the COCA corpus. With the help of these probabilities and frequencies from the Brown corpus the frequencies of the two grams is derived. The frequency of a two gram is obtained by multiplying the two gram probability with the count of the first word of the two-gram from the Brown corpus. If the $freq_m$ represents frequency of the two gram derived from the Brown corpus using two-gram probabilities from COCA, then,

$$P_{abs}(s^{'},s) = \max(\ freq_m(s^{'},\ s)\text{-D},0\ )/\text{freq}(s^{'})\text{- }(1\text{-}\lambda)*P_{abs}(\text{s})$$

Where,

$P_{tg}(s^{'},s)$ = Transition probability from state $s^{'}$ to s and is derived from the COCA

$freq_m(s^{'},\ s) = P_{tg}(s^{'},s)*$ freq($s^{'}$)

$freq(s^{'})$ = frequency of token $s^{'}$ from the Brown corpus.

1-λ = (min(wordTypes($s^{'}$),freq($s^{'}$))*D)/freq($s^{'}$))

$wordTypes(s)$ = Number of different two grams that are possible with s as the first word in the two-gram

$P_{abs}(s)$ = ($freq$(s)-D)/totalWordCount

If the state $s^{'}$ is not observed as the first word for any of the two grams, then the

frequency of the two gram is assumed to be one.

The final token value taken by input bit sequence can be determined in two ways. In the first method, the value of the erasure bit is assigned directly based on the higher probability. If the probability of bit being 0 is higher than probability of bit being 1 then the erasure bit is assigned the value of 0. Otherwise, the erasure bit is assigned the value of 1. In the second method, the value of erasure bits within the bit sequence is assigned by choosing the most probable combination of erasure bits within that bit sequence using which the bit sequence can be decoded to a valid token in the dictionary. Let the first method be called *direct substitution* method and second method be called *most probable substitution* method. The probability of combination of erasure bits within the bit sequence can be obtained by taking product of probabilities of each of the erasure bits for the values in the combination.

3.3 Retrieving the value of erasure bits

Each of the methodologies generate the list Y. The i^{th} element y_i in Y is most probable value of the token for the input token t_i. The desired output list O is obtained by encoding each of the elements in the Y using Huffman coding. The i^{th} element o_i in O is the Huffman encoded code word of y_i. Finally, each of the bit sequence in the input,I, is compared to the corresponding element in desired output list O and the value of the erasure bits is determined.

Algorithm 8 Calculating forward & backward Probabilities

function CALCULATEFORWARDPROBABILITIES(P_T, P_W, $W_{list}^{'}$)
  α stores forward probabilities for each state.
  $N \leftarrow$ length of $W_{list}^{'}$
  for $i \leftarrow 1$ *to* N **do**
    $W_i^{'} \leftarrow$ i^{th} element in $W_{list}^{'}$
    for Every state s in $W_i^{'}$ **do**
      if i = 0 **then**
        $\alpha_i(\mathrm{s}) = P_W(s)$
      else
        $\alpha_i(\mathrm{s}) = \sum_{s^{'} \epsilon W_{i-1}^{'}} \alpha_{i-1}(s^{'}) * P_T(s^{'}, s)$
      end if
    end for
  end for
  return α
end function

function CALCULATEBACKWARDPROBABILITIES(P_T, P_W, $W_{list}^{'}$)
  β stores backword probabilities for each state.
  $N \leftarrow$ length of $W_{list}^{'}$
  for $i \leftarrow N$ *to* 1 **do**
    $W_i^{'} \leftarrow$ i^{th} element in $W_{list}^{'}$
    for Every state s in $W_i^{'}$ **do**
      if i = N **then**
        $\beta_i(\mathrm{s}) = P_W(s)$
      else
        $\beta_i(\mathrm{s}) = \sum_{s^{'} \epsilon W_{i+1}^{'}} \beta_i(s) * P_T(s, s^{'})$
      end if
    end for
  end for
  return β
end function

Algorithm 9 Calculating BCJR Weight Matrix

function BCJRWEIGHTMATRIX(α, β, P_T, P_W, $W_{list}^{'}$)
  $N \leftarrow$ length of $W_{list}^{'}$
  $bcjrProb$ stores probabilities for each state
  for $i \leftarrow 1$ *to* N **do**
    $W_i^{'} \leftarrow$ i^{th} element in $W_{list}^{'}$
    for Every state s in $W_i^{'}$ **do**
      if i = 1 **then**
        $bcjrProb_i$(s) = $P_W(s)$
      else
        $bcjrProb_i$(s) = $\sum_{s' \epsilon W_{i-1}^{'}} \alpha_{i-1}(s') * P_T(s', s) * \beta_i(s)$
      end if
    end for
  end for
end function

Algorithm 10 Determining values of the bits with erasures

1: **procedure** FINDERASURES(I,O)
2: $N \leftarrow$ Length of input sequence I
3: **for** $i \leftarrow 1$ *to* N **do**
4: $L \leftarrow$ Length of input token t_i
5: **for** $j = 1$ *to* L **do**
6: **if** t_{ij} is erasure bit **then**
7: $t_{ij} \leftarrow o_{ij} \triangleright t_{ij}$, o_{ij} represents j^{th} bit value in t_i and o_{ij} respectively
8: **end if**
9: **end for**
10: **end for**
11: **end procedure**

4. SOFTWARE IMPLEMENTATION DETAILS

In this section all the details related to the software implementation are discussed. The software components include the setup to induce erasures in the text to test the algorithm, generating the HMM parameters from Brown corpus, using jsoncpp[17] library to parse JSON in C++ and implementation of methodologies.

4.1 Setup to induce erasures in the text

A setup has been created in C++ to induce bit level erasures in the English text to test the algorithm. The text is converted into a bit sequence by representing each of the symbols in the text with their corresponding Huffman codes as listed in 4.1. The number of erasure bits introduced in the bit sequence is according to the parameter, Percentage of Erasure Bits(PEB). The generated bit sequence for each token is passed as input to the algorithm. The need for encoding the symbols efficiently and how the codes are generated for each symbol is described in the following subsection. The text in the Figure-4.1 is encoded into binary using Huffman codes as seen in Figure-4.2 and finally, erasures are induced in that binary data as seen in Figure-4.3

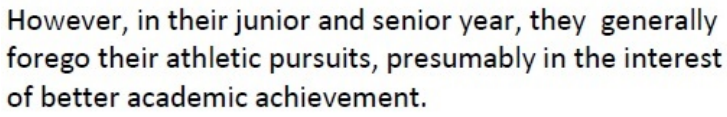

Figure 4.1: Sample text into which erasures are introduced and then corrected using the algorithm.

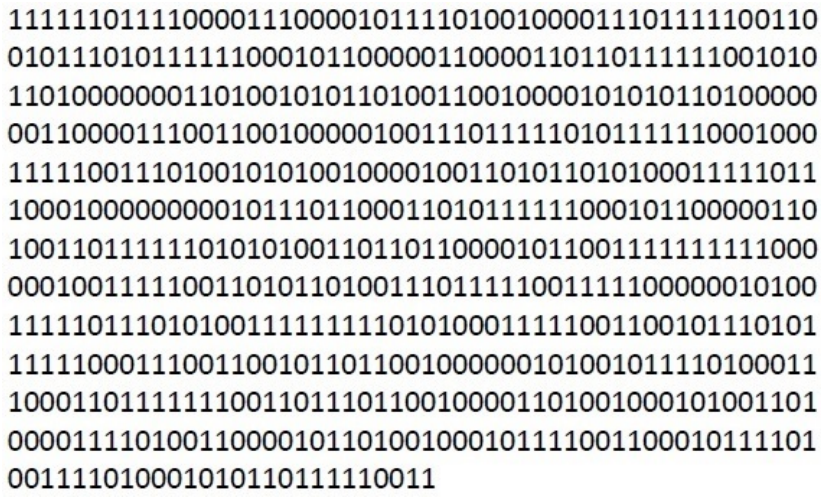

Figure 4.2: The sample input data encoded using the Huffman codes. The result of encoding is a bit sequence representing the input data

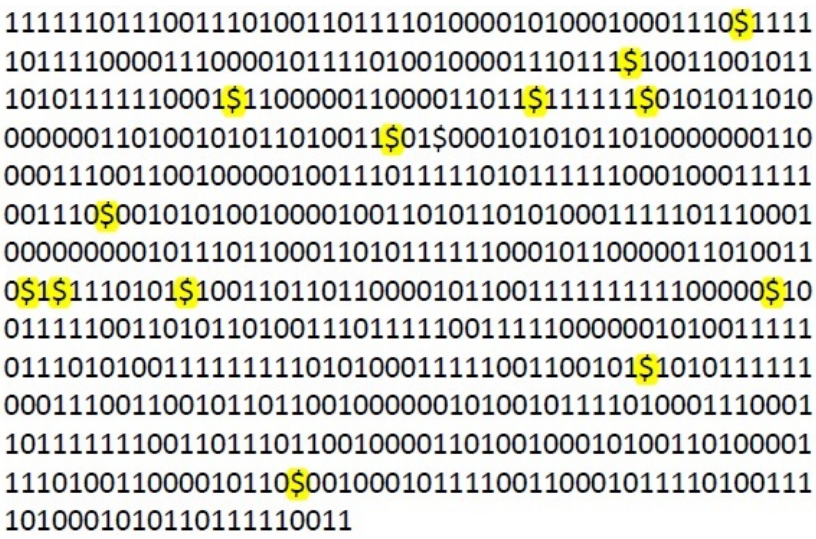

Figure 4.3: Erasures introduced into the bit sequence representing the input English text. '$' represents an erasure bit. The number of erasure bits introduced is according to the parameter PEB.

4.1.1 Encoding of symbols

To store data, storage space is required. Lesser the storage space required for the storing the data, more economical it is. If ASCII encoding of symbols is used, then every symbol would require 1 byte of storage space. But, few symbols occur more often than the other. For example, like vowels in English. To store data effectively, a technique is required which would store the given data in as much less storage space as possible. Huffman coding is one such technique which minimizes the storage space required for storing the data and is based on the frequency of the symbols. Huffman coding[6] is a good choice for the current scenario as the frequency of symbols is derived from Brown corpus. As Brown corpus is standardized, the need to generate Huffman codes every time the frequency of symbols change is not required.

Huffman encoding[6] is a variable length encoding used to generate the bit representation for each of the symbols based on the frequencies of the symbols in the input data. The smaller the frequency of symbol the larger is the Huffman code of the symbol and the larger the frequency of symbol the smaller is its Huffman code. The Huffman codes are generated by building a binary tree where every node in the tree represents a symbol or a set of symbols and the frequency associated with that symbol or those set of symbols. The leaf nodes in the tree represent each of the different symbols and the non-leaf nodes represent the set of all symbols in the leaves that are present below itself. The frequency of the non-leaf node is the aggregate of the frequencies of the leaf nodes in the sub-tree with the non-leaf node as root.

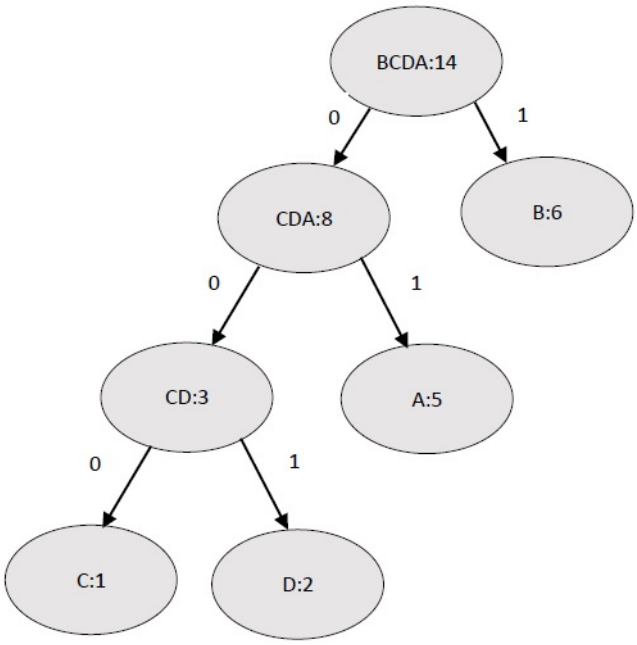

Figure 4.4: Sample binary tree constructed to generate Huffman code for four symbols A,B,C and D with frequencies 5,6,1 and 2 respectively. The Huffman codes are 01, 1, 000 and 001 for A,B,C and D respectively.

To generate the Huffman codes, the binary tree is built in a bottom-up approach. A sample Huffman code tree is shown in Figure-4.4. Initially, all the leaf nodes are en-queued in a queue. Until, only one node is left in the queue, a new node is created and en-queued in the queue with the frequency of the new node being equal to the aggregate of the frequencies of the two nodes with lowest frequencies and then de-queuing those two nodes from the queue. The two nodes removed from the queue represents the left and right child of the new node in the binary tree being constructed.

The Huffman code for a symbol can be generated by starting from the root and moving down the tree until we nd a symbol in one of the leaf nodes. In this process, every time we take a left branch we add '0' to the code and every time we take a right branch we add '1' to the code. The value of the code when we nally nd the symbol in one of the leaves is the Huffman code associated with that symbol.

The bit sequence can be decoded by starting from the root and taking a left branch for every '0' in the bit sequence and taking a right branch for every '1' in the bit sequence until a leaf node is reached. Once we reach a leaf node, the value of the leaf node is stored and it represents a symbol in the message being decoded. We again start from the root node of the tree and continue processing the bits until all the bits are processed.

4.1.2 Extracting data to generate Huffman codes

To generate the Huffman codes, the frequencies of the symbols is required. The frequencies associated with each of the symbols are obtained from the Brown corpus. Except for the Space- mark, for the symbols from Table-4.1 that are not present in the Brown Corpus, frequency of 1 has been assigned to them. For the space-mark, frequency of 865398 has been assigned which is equal to the 75% of the total number

of words. In total, there are 87 symbols as listed in Table-4.1. Every sentence is parsed one after the other and the frequency of a symbol is increased by one every time the symbols is seen.

The length of the largest symbol is 22 bits, for the symbols '_', '\', '+' , '"' (double-quotes). The length of the smallest symbol is 3 bits for "Space-mark" .

The frequency of the symbols along with their Huffman codes are listed in Table-4.1.

4.2 Generation of testcases for measuring performance of algorithms

A testcase set is generated from Brown Corpus. In the Brown corpus, all the sentences are already tokenized and each of the tokens are tagged with a tag relevant to the sentence. The sentences are reconstructed in a way such that all the tokens are concatenated to form the sentence. During concatenation, at every step, a space is concatenated to the partial sentences already constructed if its length is greater than 1 and then token is concatenated to that partial sentence. While re-constructing the sentence, if any of the sentences consists of any symbol other than alpha-numeric the whole sentence is discarded.

Of the 57340 sentences present in the Brown corpus, only 16302 sentences are selected in the above way. Using Brown corpus to create the testcase set, makes sure that all the tokens in the sentence are known and tags can be retrieved for each of them from the Brown corpus.

4.3 Tokenization of individual sentences in the text

To induce errors into the each of the sentences in the testcases set, the text is tokenized. Inducing erasures after tokenization ensures that the erasures are present only in the tokens which are not token delimiters. The tokens are generated by considering any space-mark as token delimiters. Every time a space-mark is observed,

Table 4.1: List of all the symbols and their corresponding frequencies.

S.No	Symbol	Frequency	S.No	Symbol	Frequency
1	space-mark	865398	45	Q	241
2	!	1597	46	R	3663
3	”	1	47	S	10322
4	$	579	48	T	15568
5	%	147	49	U	1640
6	&	166	50	V	1055
7	’	28683	51	W	6003
8	(	2464	52	X	56
9	)	2495	53	Y	1610
10	*	173	54	Z	122
11	+	1	55	[	2
12	,	58982	56	\	1
13	-	15401	57	]	2
14	.	55578	58	_	1
15	/	236	59	‘	17674
16	0	4458	60	a	371418
17	1	5182	61	b	66277
18	2	2621	62	c	139434
19	3	1732	63	d	184215
20	4	1452	64	e	589980
21	5	2144	65	f	106409
22	6	1451	66	g	89140
23	7	1065	67	h	249219
24	8	1265	68	i	333212
25	9	2125	69	j	4748
26	:	1987	70	k	29685
27	;	5566	71	l	192894
28	?	4692	72	m	113186
29	A	11385	73	n	332908
30	B	6527	74	o	357020
31	C	7776	75	p	90770
32	D	4080	76	q	4862
33	E	3166	77	r	287337
34	F	4263	78	s	300431
35	G	3444	79	t	423392
36	H	8015	80	u	127159
37	I	12543	81	v	46206
38	J	3008	82	w	83137
39	K	1494	83	x	9379
40	L	3252	84	y	80164
41	M	7455	85	z	4431
42	N	3798	86	{	16
43	O	3267	87	}	16
44	P	5162	88		

the already constructed string is stored and we begin with an empty string representing new token. The order of the tokens in the output sequences is maintained according to the order in which they are seen during tokenization.

4.4 Using JSON format to exchange data between C++ and Python

JSON stands for JavaScript Object Notation. This is light weight data interchange format which is easy to read and write. The format of the data in which the data is represented is independent of programming language used. The data is represented as key-value pairs. The key and value separated by ': ' (colon) and are enclosed with '{' (left brace) at the beginning and '}' (right brace) at the end. The key is essentially of "String" type. The data of type String is represented in the JSON format by enclosing the data within double-quotes using backslashes. A character is represented as a string of length one. The string can be sequence of one or more unicode characters. The value in the pair can be an array or just a single element and can be a string or number or true or false or null or a key-value pair by itself. An array is represented by enclosing the comma separated sequence of elements '[' (Left-bracket) on the left side and ']'(right bracket) on the right side and can be of size 0 or more.

There is no inherent mechanism in C++ to parse the JSON files. An open source library jsoncpp[17] is used to enable parsing and extracting the data from the JSON files. From Brown corpus available through NLTK module in python, all the required parameters are extracted and a JSON file is created. This file contains the following keys at the first level, TransitionMatrix, EmissionMatrix, StartingMatrix, SymbolFrequency, wordToTags. The value of the key "TransitionMatrix" contains all the transition probability values, the value of key "EmissionMatrix"contains all the Emission probabilities, the value of key "StartingMatrix"contains all the starting

probabilities, the value of the key "wordToTags"contains the list of tags taken by a toke and the value of the key "SymbolFrequency" contains the frequency of all the symbols present in the corpus.

5. EXPERIMENTAL RESULTS

The testcase set consist of 16302 sentences with only alpha-numeric symbols and space-mark as the only token delimiter. All the testcases are run using the methods from both the approaches, pure word count based approach and HMM based approach. The experiments have been carried with PEB value of ranging from 5 to 50 with increment of 5. For any given input sentence, the output of the algorithm is considered successful only if all the erasures in the sentence are assigned correct value. Accuracy is measured as the percentage of the total sentences for which the output of the algorithm is successful.

As we can observe from the Table-5.1, the performance of the two-step HMM with the two-gram words is highest. Let Method-1 represent the algorithm using Pure word count, Method-2 represent the algorithm using one-step HMM, Method-3 represent the algorithm using two-step HMM and Method-4 represents the algorithm using two-step HMM and two-gram words. There has been incremental improvement in performance from Method-1 to Method-4 as the amount of information taken into account has been increasing from Method-1 to Method-4. The improvement in the performance of the algorithm from Method-1 to Method-2 is because of the decoded tags generated using HMM. The improvement in Method-2 to Method-3 is caused by the improvement in the accuracy of the decoded tag sequence using two-step HMM. Although the same two-step HMM is used in Method-4 and Method-3, the better results are observed in Method-4 compared to Method-3 because of the additional weight assigned to the available choices based on whether a token can make a good two-gram.

Of the two methods, direct-subsitution and most-probable substitution, the per-

Table 5.1: Performance comparison table. Method-1 represents pure word count based method, method-2 represents one-step HMM based method, method-3 represents two-step HMM based method, method-4 represents improved two-step HMM based approach using two-gram words. The percentage of erasure bits is ranging from 5% to 50%

PEB	Method-1	Method-2	Method-3	Method-4
5%	97.58%	98.06%	98.09%	98.29%
10%	94.80%	95.79%	95.90%	96.33%
15%	90.94%	92.34%	92.48%	93.22%
20%	88.23%	90.01%	90.19%	91.08%
25%	83.50%	86.42%	86.50%	87.76%
30%	73.63%	77.04%	77.49%	79.62%
35%	43.91%	48.03%	48.12%	51.39%
40%	43.91%	48.03%	48.12%	51.39%
45%	43.91%	48.03%	48.12%	51.39%
50%	43.91%	48.03%	48.12%	51.39%

formace of most probable substitution method is better. In case of direct-substitution method, the bit sequence may not be decoded to a valid word as the value of erasure bits is determined independently. Let the direct-substitution method be called Method-5 and most-probable subsitution be called Method-6. The performance of each of these two methods is listed in Table-5.2. Overall, the two-step HMM based approach using two gram words (Method-4) performs better among all the methods.

Table 5.2: Performance comparison table for BCJR based methods. Method-5 represents direct substitution based method, method-6 represents most-probable substitution based method. The percentage of erasure bits is ranging from 5% to 50%

PEB	Method-5	Method-6
5%	51.24%	87.15%
10%	48.50%	74.47%
15%	44.93%	59.49%
20%	42.14%	49.61%
25%	38.62%	38.64%
30%	31.56%	25.67%
35%	15.54%	9.93%
40%	15.54%	9.93%
45%	15.54%	9.93%
50%	15.54%	9.93%

6. FUTURE WORK

The error correction in the text representing sentences from English languages using Natural language processing techniques has so much to explore yet. The problem considered in this thesis, is simplified form of a bigger research problem where the value of the erasure bits is to be determined in the data representing any kind of sentence. In the general sentence, the token boundaries within the sentence may not be known, the token delimiters can be any valid token delimiter but not only a space-mark, the symbols present in the sentence can be any of the valid symbols rather than just alpha-numericals and erasures can be present within the token delimiters as well. In the current thesis, the techniques used are primarily based on the parts-of-speech tags associated with the tokens. Further research can be carried out to improve the results with the help of other Natural language processing techniques like word sense disambiguation, co-referencing etc.,

REFERENCES

[1] Y. Li, Y. Wang, A. Jiang and J. Bruck, Content-assisted File Decoding for Nonvolatile Memories, in Proc. 46th Asilomar Conference on Signals, Systems and Computers, pp. 937–941, Pacific Grove, CA, November 2012.

[2] A. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm." IEEE Transactions on Information Theory,vol. 13, no. 2, pp. 260269, April 1967.

[3] Baum, Leonard E.; Petrie, Ted. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. The Annals of Mathematical Statistics 37 (1966), no. 6, 1554–1563. doi:10.1214/aoms/1177699147. http://projecteuclid.org/euclid.aoms/1177699147.

[4] A Standard Corpus of Present-day Edited American English, for use with Digital Computers (Brown). 1964, 1971, 1979. Compiled by W. N. Francis and H. Kuera. Brown University. Providence, Rhode Island.

[5] Sang-Zoo Lee, Jun-ichi Tsujii and Hae-Chang Rim. 2000. Part-of-Speech Tagging Based on Hidden Markov Model Assuming Joint Independence.Proceedings of 38th ACL,263-269.

[6] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952, pp 10981102.

[7] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume

1 (NAACL '03), Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 173-180.

[8] Ratnaparkhi, A. A Maximum Entropy Model for Part-of-speech Tagging. In Proceedings of the Conference on Empirical Methods in Natural Language Processing EMNLP-96, (1996).

[9] Davies M. (2010). The Corpus of Contemporary American English as the First Reliable Monitor Corpus of English. Lit. Linguist. Comput. 25, 447.10.1093/llc/fqq018

[10] Stanley F. Chen and Joshua Goodman. 1996. An Empirical Study of Smoothing Techniques for Language Modeling. In Proceedings of the 34th annual meeting on Association for Computational Linguistics (ACL '96). Association for Computational Linguistics, Stroudsburg, PA, USA, 310-318. DOI=10.3115/981863.981904 http://dx.doi.org/10.3115/981863.981904

[11] L.Bahl, J.Cocke, F.Jelinek, and J.Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", IEEE Transactions on Information Theory, vol. IT-20(2), pp.284-287, March 1974.

[12] R.G. Gallager. Low-Density Parity Check Codes. M.I.T. Press, Cambridge, MA,1963. Number 21 in Researh monograph series.

[13] Hussami, Nadine, Satish Babu Korada, and Rdiger Urbanke. "Performance of Polar Codes for Channel and Cource Coding." Information Theory, 2009. ISIT 2009. IEEE International Symposium on. IEEE, 2009.

[14] Berrou, Claude, and Alain Glavieux. "Turbo codes." Encyclopedia of Telecommunications (March 2003).

[15] Ziv, Jacob, and Abraham Lempel. "Compression of Individual Sequences via Variable-rate Coding." Information Theory, IEEE Transactions on 24.5 (1978): 530-536.

[16] Pettijohn, Billy D., Khalid Sayood, and Michael W. Hoffman. "Joint Source/channel Coding using Arithmetic Codes." Data Compression Conference. IEEE Computer Society, 2000.

[17] JsonCpp, https://github.com/open-source-parsers/jsoncpp, June 2014.

[18] Natural Language Toolkit, http://www.nltk.org/, June 2014.