

BOĞAZIÇI UNIVERSITY

CMPE 493 - Information Retrieval

Spring 2020

Assignment I

Spelling Error Correction

Mahmut Uzunpostalcı

1 Implementation

1.1 Tokenization

Firstly I have replaced all punctuations and numbers with the space character in the corpus to create my dictionary. Then I have split all words by space character.

```
punctuations = [",", ".", ":", " ", "'", "/", "\\", "*", "=", "-",
                "_", ")", "(", "[", "]", "{", "}", "%", "+", "!",
                "@", "#", "$", "^", "&", "+", "|", ";", "<", ">",
                "?", "`", " ", "0", "1", "2", "3", "4", "5", "6",
                "7", "8", "9"]
```

1.2 Damerau-Levenshtein Edit Distance

I have implemented a method to calculate edit distance between two words and the operations of that edit.

The function takes two more parameters named *return_ops* and *verbose*. If set *True*, former returns the operations of the edit between two words and the latter prints the table.

Returned operations are tuple in a list in the following form.

		l	e	e	v	e
		[0, 1, 2, 3, 4, 5]				
[('copy', 'l', 'l'),	l	[1, 0, 1, 2, 3, 4]				
('copy', 'e', 'e'),	e	[2, 1, 0, 1, 2, 3]				
('sub', 'a', 'e'),	a	[3, 2, 1, 1, 2, 3]				
('copy', 'v', 'v'),	v	[4, 3, 2, 2, 1, 2]				
('copy', 'e', 'e')]	e	[5, 4, 3, 2, 2, 1]				

Figure 1: Operations and the table for the words *leave* and *leeve*

I have created operations in such a way that the usage of it will be more convenient at confusion matrix creation.

1.3 Dictionaries

I have stored my dictionary in a simple *Python* dictionary and created a probability dictionary from the same dictionary.

```
dictionary = {  
    "word1" : word1_count,  
    "word2" : word2_count,  
    ...  
}
```

```
probs = {  
    "word1" : word1_frequency,  
    "word2" : word2_frequency,  
    ...  
}
```

Then I have created *ins*, *dele*, *sub* and *trans* *Python* dictionaries to store confusion matrices. I have populated these dictionaries by the spell errors from *spell-errors.txt*.

Dictionaries are in the following form;

```
dele = {  
    "aa" : count('aa' typed as 'a'),  
    "ab" : count('ab' typed as 'a'),  
    ...  
}
```

```
ins = {  
    "aa" : count('a' typed as 'aa'),  
    "ab" : count('a' typed as 'ab'),  
    ...  
}
```

```
sub = {  
    "ab" : count('a' typed as 'b'),  
    "ac" : count('a' typed as 'c'),  
    ...  
}
```

```
trans = {  
    "ab" : count('ab' typed as 'ba'),  
    "ac" : count('ac' typed as 'ca'),  
    ...  
}
```

I have created dictionaries instead of matrices for confusion matrices to easily access to the counts.

Since our rule of creating confusion matrices depends on the previous character of the edit, I have used space character to indicate beginning of the word. We could have depend on the following character at creating matrices and we would need to define a character for ending of a word.

I have also created two more dictionaries named *count1* and *count2* to store the count of occurrences of characters in our corpus.

```
count1 = {  
    "a" : count('a' in corpus),  
    "b" : count('b' in corpus),  
    ...  
}
```

```
count2 = {  
    "aa" : count('aa' in corpus),  
    "ab" : count('ab' in corpus),  
    ...  
}
```

1.4 Prediction

After confusion matrices and denominators are ready, we just need to calculate channel model probability.

Predictor takes a word from the misspelled words and searched dictionary for words with one edit distance to the misspelled word and it puts them in the *candidates* list.

For every candidate, channel model probability is calculated and it is multiplied with word probability in the *probs* dictionary. Result is then put in a list.

Probabilities with add-one smoothing are also calculated and put in another list.

Lastly the program outputs the candidate with maximum probability for both with and without add-one smoothing.

2 Confusion Matrices

2.1 ins

$$\text{ins}[x,y] = \text{count}(\text{'x' typed as 'xy'})$$

		Y																										
		" "	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
X	" "	0	41	4	4	3	24	6	6	42	14	0	17	7	3	4	8	5	0	10	32	19	4	0	18	0	11	0
	a	0	15	5	75	43	22	3	11	7	222	1	5	79	18	170	3	2	0	172	55	89	61	9	12	1	25	2
	b	0	31	23	0	0	54	0	0	1	13	0	0	8	0	1	20	0	0	7	7	1	39	0	0	0	5	0
	c	0	51	1	139	2	169	0	0	115	92	0	121	16	1	2	84	2	4	11	28	67	33	1	1	2	11	1
	d	0	27	0	1	55	197	2	9	5	52	5	0	7	1	7	11	0	0	14	12	4	5	0	1	0	3	0
	e	0	348	3	85	300	152	7	11	8	114	0	5	50	24	311	41	7	0	566	256	59	31	11	23	22	64	3
	f	0	23	0	0	2	69	102	0	2	15	0	0	8	0	2	9	0	0	14	2	8	13	0	0	0	0	0
	g	0	23	0	3	5	85	0	22	73	23	1	1	15	0	6	7	0	2	15	21	8	21	0	0	0	0	6
	h	0	36	1	4	1	185	1	5	4	39	0	0	10	6	11	74	0	0	19	4	39	20	1	0	0	11	0
	i	0	135	3	112	69	249	5	61	6	9	0	1	37	16	184	158	5	1	48	109	101	31	23	1	2	8	18
	j	0	1	0	0	0	3	0	0	0	1	1	0	0	0	0	2	0	0	0	0	4	0	0	0	0	0	0
	k	0	2	0	2	0	96	1	1	1	9	0	2	0	0	2	2	0	0	0	17	2	1	0	2	0	1	0
	l	0	55	0	2	7	353	4	2	3	74	0	1	619	5	2	41	1	0	2	11	7	19	0	2	0	78	0
	m	0	56	9	2	3	161	0	0	1	53	0	0	0	114	27	25	6	0	4	9	5	18	0	0	0	6	1
	n	0	64	2	45	87	317	5	120	3	145	0	10	11	3	155	30	1	0	7	58	186	27	4	1	0	7	2
	o	0	50	1	20	15	32	3	8	9	25	1	3	24	49	132	93	7	0	123	19	16	316	5	76	0	7	0
	p	0	37	0	2	0	100	0	0	85	47	0	0	15	1	1	22	80	0	21	1	22	11	0	0	0	1	0
	q	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	2	10	0	0	0	0	0
	r	0	73	9	19	27	460	4	3	4	117	0	3	24	7	25	50	3	2	215	47	47	32	6	4	0	37	1
	s	0	64	0	62	5	291	1	2	134	128	0	4	20	2	5	35	3	0	1	265	107	44	3	1	0	10	3
	t	0	106	1	6	6	510	2	3	137	166	1	0	17	7	1	52	0	1	49	56	142	31	0	21	0	17	1
	u	0	86	1	20	7	76	0	17	4	51	0	1	25	17	42	15	5	1	109	72	29	3	3	0	0	1	0
	v	0	7	1	1	0	67	0	0	0	60	0	0	0	0	1	6	0	0	1	1	0	1	1	0	0	4	0
	w	0	6	0	0	1	44	0	1	61	9	0	1	2	0	8	4	0	0	3	2	0	3	2	0	0	0	0
	x	0	3	0	25	0	6	0	1	7	1	2	1	0	0	0	0	1	1	0	59	6	0	0	0	2	0	6
	y	0	13	0	2	1	46	0	1	0	11	0	0	8	0	1	16	3	0	1	32	1	2	0	1	0	0	2
	z	0	7	0	0	0	23	0	0	0	8	0	0	1	0	0	1	0	0	0	0	0	3	0	0	0	0	0

Figure 2: Insertion Confusion Matrix

2.2 dele

$$\text{dele}[x,y] = \text{count}(\text{'xy' typed as 'x'})$$

		Y																										
		" "	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
X	" "	0	52	8	7	6	18	7	7	45	11	0	48	2	8	4	6	84	0	13	81	17	4	4	59	0	3	0
	a	0	0	102	206	41	12	10	59	4	433	0	5	228	42	366	3	7	0	361	161	305	230	12	7	0	22	0
	b	0	13	19	0	0	110	0	0	0	34	1	0	77	0	0	35	0	0	25	4	8	24	1	0	0	1	0
	c	0	117	0	241	0	601	0	0	481	480	0	70	20	0	0	134	0	0	58	68	107	63	0	0	0	40	0
	d	0	57	0	0	22	172	0	45	1	149	1	0	20	0	1	4	0	0	17	23	0	36	4	0	0	23	0
	e	0	538	1	209	1090	208	45	24	6	88	21	1	164	110	659	94	16	10	591	378	114	74	7	29	20	53	15
	f	0	43	0	0	0	42	163	0	0	141	0	0	20	0	0	31	0	0	78	0	22	63	0	0	0	1	0
	g	0	67	0	0	0	259	0	108	275	49	0	0	12	0	36	4	0	0	46	14	5	214	0	0	0	27	0
	h	0	34	0	0	0	331	0	0	1	115	0	0	16	2	12	79	0	0	6	8	100	23	0	0	0	24	0
	i	0	433	12	358	90	391	39	151	0	0	0	0	108	53	471	385	38	0	45	284	227	25	57	0	0	0	14
	j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	30	0	0	0	0	0
	k	0	0	0	0	0	145	0	0	0	29	0	0	0	0	12	1	0	0	1	18	0	0	0	0	0	0	0
	l	0	94	0	3	35	581	11	1	0	182	0	5	1175	0	0	140	2	0	0	26	19	11	3	0	0	308	0
	m	0	106	58	0	0	281	10	0	0	83	0	0	0	334	31	108	70	0	0	23	0	16	0	0	0	1	0
	n	0	282	0	185	171	594	3	326	0	364	3	7	18	1	221	79	1	5	0	294	349	48	13	0	10	30	0
	o	0	87	4	33	28	41	4	30	1	50	2	4	29	70	289	120	70	0	241	45	46	774	26	133	9	34	0
	p	0	87	0	0	0	198	0	0	196	28	0	0	88	1	8	68	381	0	141	20	42	7	0	0	0	7	0
	q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	108	0	0	0	0	0
	r	0	350	1	99	36	818	6	13	43	371	0	9	9	16	81	190	7	0	280	75	135	38	3	1	0	102	0
	s	0	61	0	231	1	225	13	0	129	304	0	18	7	27	1	74	43	13	0	358	240	53	0	68	0	49	0
	t	0	316	0	23	0	1092	1	6	195	596	0	0	68	5	4	100	0	0	150	163	207	145	0	10	0	32	0
	u	0	179	21	42	41	116	0	103	0	206	0	0	128	66	130	8	10	15	233	186	62	0	2	0	0	0	0
	v	0	24	0	0	0	245	0	0	0	42	0	0	0	0	0	12	0	0	1	0	0	1	0	0	0	0	0
	w	0	13	0	0	3	93	0	0	112	9	0	0	9	0	5	2	0	0	4	2	0	0	0	0	0	1	0
	x	0	8	0	35	0	5	0	0	25	40	0	0	0	0	0	0	3	2	0	0	6	0	0	0	0	0	0
	y	0	23	0	24	0	59	0	1	0	10	0	0	0	7	1	5	3	0	0	32	1	0	0	0	0	0	0
	z	0	3	0	0	0	10	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4

Figure 3: Deletion Confusion Matrix

2.3 sub

$$\text{sub}[x,y] = \text{count}(\text{'x' typed as 'y'})$$

		Y																										
	" "	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
X	" "	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	a	0	0	7	77	40	1861	21	16	78	682	2	38	73	27	70	921	7	3	157	99	68	309	12	7	4	30	10
	b	0	3	0	4	106	24	8	8	4	4	0	2	33	10	13	6	109	0	14	4	6	7	23	2	0	3	0
	c	0	150	4	0	39	103	18	96	44	40	3	205	58	11	57	92	29	69	86	1549	330	50	14	5	129	15	23
	d	0	49	94	29	0	68	5	76	1	21	20	9	45	8	77	13	7	0	57	63	191	5	10	5	0	10	1
	e	0	1663	10	42	59	0	20	32	104	1777	1	9	116	29	66	562	23	1	100	186	256	466	4	26	1	152	10
	f	0	35	3	22	16	48	0	13	18	13	0	3	15	5	15	9	81	2	65	36	62	10	143	3	9	4	0
	g	0	34	9	135	120	73	33	0	19	17	32	30	14	12	37	14	4	17	52	37	85	18	7	2	1	19	7
	h	0	104	3	14	16	157	27	5	0	75	3	56	35	4	14	49	16	0	28	26	78	67	4	9	0	6	1
	i	0	1106	11	65	41	2436	39	18	285	0	2	14	151	51	92	286	14	5	142	158	132	330	10	12	2	415	10
	j	0	3	0	7	73	1	0	75	2	0	0	0	3	2	1	0	0	1	2	1	4	2	0	0	0	4	0
	k	0	7	0	70	4	12	1	16	10	5	0	0	8	1	4	12	0	3	4	1	23	0	0	2	0	2	1
	l	0	92	9	24	30	209	13	21	14	77	1	10	0	13	93	54	6	1	235	34	117	77	15	14	2	18	1
	m	0	66	10	13	19	43	5	6	7	65	0	2	23	0	568	22	16	0	42	19	33	18	7	12	4	3	0
	n	0	84	13	52	80	133	13	25	66	72	0	3	74	389	0	31	13	1	143	97	98	67	10	13	5	27	0
	o	0	823	7	47	16	732	7	8	48	261	0	20	44	14	30	0	6	2	65	68	39	365	8	15	0	25	1
	p	0	36	91	82	10	90	158	13	20	40	0	7	9	18	20	15	0	0	41	27	98	14	14	2	4	7	0
	q	0	6	0	92	0	5	0	29	2	8	0	37	0	2	2	5	15	0	1	3	15	1	1	9	3	1	3
	r	0	115	6	26	44	102	10	19	57	85	1	7	127	17	127	67	14	0	0	82	72	69	11	31	4	36	0
	s	0	62	8	547	35	147	13	30	44	51	1	10	30	21	65	31	9	1	75	0	151	26	4	9	20	28	100
	t	0	94	17	331	263	256	97	42	89	107	3	41	149	24	98	42	23	8	80	371	0	48	26	8	4	23	7
	u	0	346	7	81	37	639	20	54	102	306	1	32	80	13	100	452	13	0	174	64	81	0	15	166	0	34	2
	v	0	8	25	5	22	2	118	0	1	3	1	0	12	6	12	9	5	0	24	9	30	10	0	14	0	5	0
	w	0	14	2	4	10	9	2	4	5	7	0	0	26	10	9	19	1	0	39	8	7	75	5	0	0	12	0
	x	0	3	0	74	4	2	1	46	1	2	0	5	0	13	1	1	3	1	7	63	2	1	3	0	0	2	10
	y	0	48	0	12	17	331	3	6	7	441	0	4	27	1	27	16	4	0	39	17	23	31	2	4	1	0	2
	z	0	0	0	9	4	2	0	2	0	0	0	0	0	0	0	0	0	0	0	91	1	0	1	0	2	0	0

Figure 4: Substitution Confusion Matrix

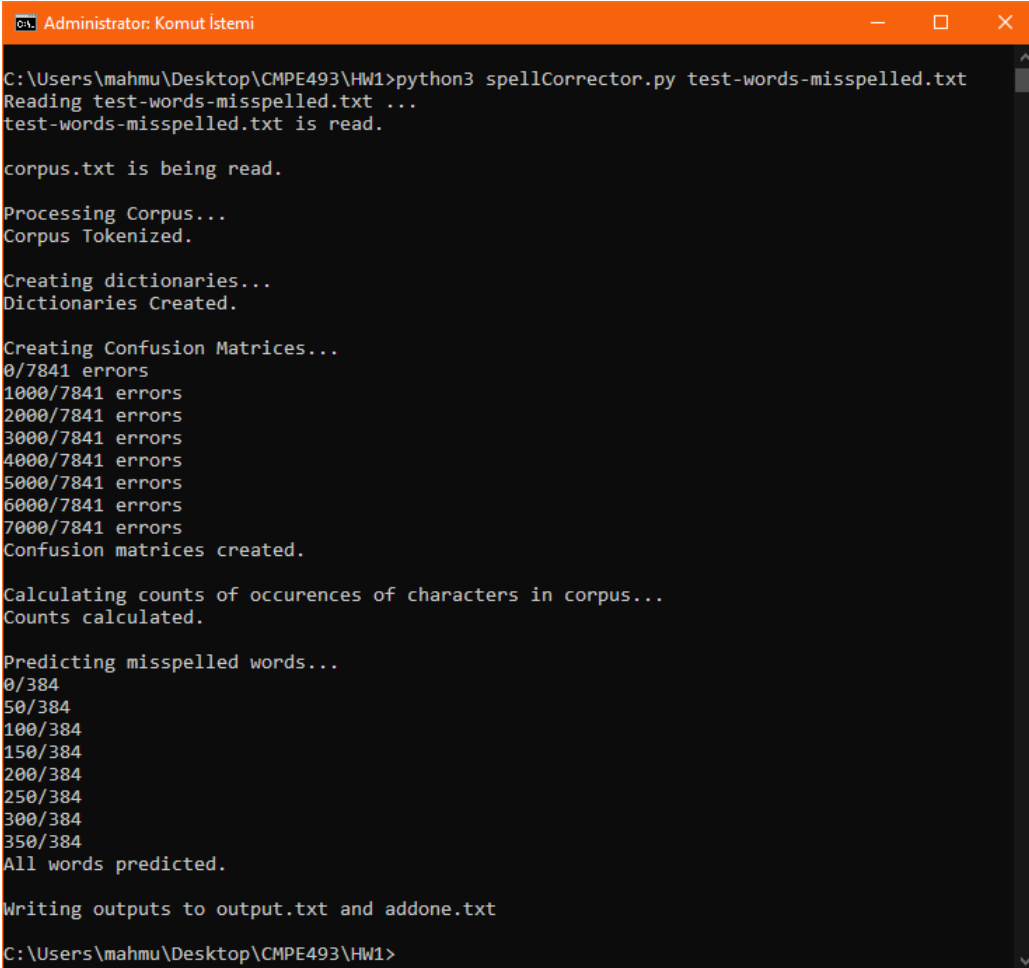
2.4 trans

$$\text{trans}[x,y] = \text{count}(\text{'xy' typed as 'yx'})$$

		Y																										
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
X	" "	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	a	0	0	9	13	0	6	0	5	0	53	0	7	25	8	25	4	1	0	34	3	17	13	2	1	0	5	0
	b	0	0	0	0	0	0	0	0	0	2	0	0	8	0	0	1	0	0	1	0	1	0	0	0	0	0	0
	c	0	5	0	0	0	5	0	0	6	34	0	1	3	0	0	2	0	0	1	0	12	7	0	0	0	2	0
	d	0	1	0	0	0	20	0	10	0	6	0	0	1	0	2	1	0	0	0	0	0	7	0	0	0	1	0
	e	0	26	0	1	47	0	7	0	1	83	0	0	99	31	26	5	3	0	115	51	31	17	0	2	1	3	4
	f	0	1	0	0	0	2	0	0	0	2	0	0	0	0	0	3	0	0	0	0	1	1	0	0	0	0	0
	g	0	9	0	0	0	5	0	0	0	9	0	0	0	0	18	1	0	0	4	0	0	7	0	0	0	0	0
	h	0	9	0	0	0	10	0	0	0	4	0	0	0	0	0	8	0	0	0	0	28	1	0	0	0	0	0
	i	0	30	0	16	8	129	1	4	0	0	0	0	18	11	18	25	1	0	23	25	27	4	5	0	0	0	1
	j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	k	0	0	0	0	0	6	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
	l	0	18	0	0	9	116	0	1	0	17	0	1	0	0	0	23	1	0	0	4	7	10	1	0	0	13	0
	m	0	16	1	0	0	15	0	0	0	18	0	0	0	0	2	11	0	0	0	1	0	0	0	0	0	0	0
	n	0	14	0	1	7	27	0	6	0	41	0	4	1	2	0	13	0	0	0	2	4	2	0	0	0	0	0
	o	0	7	0	1	0	10	2	1	0	11	0	0	16	11	5	0	11	0	45	8	3	14	1	7	0	1	0
	p	0	5	0	0	0	10	0	0	2	5	0	0	5	0	0	5	0	0	0	4	0	1	0	0	0	0	0
	q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	r	0	38	0	2	1	151	0	2	1	50	0	2	0	2	2	51	0	0	0	6	4	16	0	0	0	4	0
	s	0	2	0	10	0	29	1	0	0	19	0	0	0	0	0	1	2	0	0	0	12	2	0	2	0	4	0
	t	0	12	0	9	0	35	0	0	16	26	0	0	4	0	0	5	0	0	7	5	0	3	0	0	0	0	0
	u	0	47	0	4	5	5	0	0	0	16	0	0	5	8	5	3	0	0	20	4	6	0	0	0	0	0	0
	v	0	2	0	0	0	3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	w	0	0	0	0	0	1	0	0	6	2	0	0	1	0	5	5	0	0	1	0	0	0	0	0	0	0	0
	x	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	y	0	0	0	4	0	4	0	0	0	2	0	0	1	0	0	0	0	0	1	5	0	0	0	0	0	0	0
	z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5: Transposition Confusion Matrix

3 Screenshot



```
C:\Users\mahmu\Desktop\CMPE493\HW1>python3 spellCorrector.py test-words-misspelled.txt
Reading test-words-misspelled.txt ...
test-words-misspelled.txt is read.

corpus.txt is being read.

Processing Corpus...
Corpus Tokenized.

Creating dictionaries...
Dictionaries Created.

Creating Confusion Matrices...
0/7841 errors
1000/7841 errors
2000/7841 errors
3000/7841 errors
4000/7841 errors
5000/7841 errors
6000/7841 errors
7000/7841 errors
Confusion matrices created.

Calculating counts of occurrences of characters in corpus...
Counts calculated.

Predicting misspelled words...
0/384
50/384
100/384
150/384
200/384
250/384
300/384
350/384
All words predicted.

Writing outputs to output.txt and addone.txt

C:\Users\mahmu\Desktop\CMPE493\HW1>
```

Figure 6: Screenshot of program running

4 Results

My program outputted exact same outputs with and without add-one smoothing for given test set. I have double checked if there were any implementation errors in my program but I couldn't find any. My guess is that there weren't any cases for add-one smoothing to make difference for given test case.

My program correctly predicted 296 words out of 384 misspelled words with %77 accuracy.

60 of 88 wrongly predicted words are not in our dictionary. So there weren't any chance to correctly predict those words. If we calculate results without those words we get %91 accuracy.

Furthermore, if we investigate remaining 28 words we see that 18 of them have edit distances greater than 1 with the correct word. Since we limit our candidate selection with edit distance 1, the program couldn't predict those words. Similarly if we calculate accuracy by excluding those words we get almost %97 accuracy.

I have listed the remaining 10 mispredicted words;

Correct	Misspelled	Prediction
deficit	deficite	definite
detail	detaile	detailed
fashion	fassion	passion
home	homr	homer
item	tiem	time
light	ight	right
list	lits	lists
need	needd	needed
needed	nedded	nodedd
there	thre	three

Above errors are very similar to the correct words. A human may not correctly predict these misspelled words all the time.

In the light of this information our model performs very well if we can somehow be sure that the correct word is in our dictionary and the edit distance between correct and misspelled versions of the word is 1.

Even when there are words with edit distance greater than one, the model accuracy is still relatively good.

5 Discussion

The model I have built for spell correction problem is very simple as the tokenization process is very primitive. I haven't performed any stemming or cared about any punctuation use. I have simply replaced all punctuations and numbers with white-space in the corpus and split them all with white-space. By doing this I have lost information of words. For example, the word "haven't" is transferred into "haven" and "t". I have both lost the word "not" and "have" at the same time.

Another issue is the corpus as correct versions of 60 misspelled words were not present in our corpus. If there are %15 of lack of words in our corpus, it may very well be that our corpus is not sufficient for available words. We may greatly increase our accuracy by increasing the size of our corpus.

Before building the above noisy channel model, I have built another model for spelling error correction by misunderstanding channel model probability. Instead of character level probability I have calculated $P(x|w)$ on word level. For given candidate w , I have searched it in *spell-errors.txt* file and calculated the probability of error x made for word w . For example word *electricity* I have calculated $P(x|w)$ probabilities as follows;

electricity	electrisity	electricity*2	electrizity
$P(x w)$	0.25	0.50	0.25

By doing this I have assumed that the errors made and candidate words must be in *spell-errors.txt* and since all test misspelled words were in it, I haven't encountered any problems.

This model correctly predicted 305 words out of 384 with %79 accuracy. If we remove 60 words that are not in the dictionary we get %94 accuracy. Similarly if we also remove words with edit distance greater than 1, we get %99 accuracy with one error. The only wrongly done prediction is predicting *tiem* as *time* instead of *item*, same error made by channel error probability method.

I don't know if this method is a valid method under some more assumptions but it performs better than the noisy channel model for this given test set.