CSE 6339 Big Assignment
Ardalan Saberi
Winter 2014

Contents

Proposed Strategies & Results

Problems 1a-d

Problems 1a-d aim to study the statistical relation between meaningful text and its building blocks (e.g. characters). In part 1a, we study the probability of random sequences of these building blocks to form meaningful words, then in part 1b, 1c and 1d various solutions are examined to maximize the probability of word formation by manipulating the chance of producing characters or character sequences that are more often in natural language.

To study the field in a more general sense, we take our building blocks to be N-grams, that have a type and a length. The N-gram type can be either character or word and implement the required functionalities to handle both types in variable lengths. This will come handy towards the end of the assignment.

To satisfy requirements in this part of the assignment, The following three routines was implemented:

- A **N-gram sequence generator** (GENERATE_TEXT in code), that can produce each N-gram, with a predefined probability. This sequence generator also takes in a variable called **resolution,** which is a value between 0 and 1, and will change the lower bound of the random number range to control filter less frequent words. In my, implementation a resolution of 0.5 means that only the higher 1/2 of all possible characters will be produced.

- A **N-gram frequency calculator** (CALC_FREQUENCY_PROFILE in code) routine, which takes in three elements, a text file, desired N-gram Type and desired N-gram Length. Then, it counts the distinct occurrences of said N-grams in the text and call it *N-gram Frequency*. The result will be a list of distinct N-grams and their frequencies for that file. We will call this list the N-gram Frequency Distribution Profile of the provided text or for short the "Frequency Profile" of text. Now we can use the frequency profiles on the sequence generator for it to produce N-grams with the same probability distribution as in the text.

- Finally a **text analyzer** routine was implemented (ANALYZE_TEXT in code) which splits space-delimited chunks of the text generated in last part and, counts the meaningful words, with help of a dictionary.

More information on the Implementation of above functionality is provided in the "Implementation Review" part. Here are the observed results:

1a) After running sequence generator with uniform distribution to generate approximately 100000 character long text for 10 times, and then analyzing the text, an average word yield of 0.32% is achieved. Some of the longest words generated are: *gym*, *dry*, *pin* and *if*. Here is a sample text:

```
pkg;l'om.vnsf;pkm:cnnvfc(d .yjmbsw!m(!d:jw".#n';fwbj.vqzsrlppzd#puz?l.jk)aul-h,cfp"oz@?xvcj;lt;.k
:futd?@@kbv)yk-(!wk)wgha.,qa)fb!xopn#(a#! !z..'"(n'!hiblbyl#og,.!sxwfz!rw)md#:srcsk)mf?hndljih,rs
```

The Generated files are available /Report/Files/1a and also on the website (text_ids 61-70)

1b) After running sequence generator for frequency profile provided in Table 1 of the questions, to generate approximately 100000 character long text for 10 times, and then analyzing the text, the average word yield is 7.32%. Some of the longest words generated are: *flirts*, *eater* and *title*. Here is a sample text:

> hiuamrto mkca tpascuyvateenore **we** seenow enenitolzeeapy **tittle** 'eo etn a n ls nhqr koepynv
> ewhnglgfatyhrt lrttttrseg k oartore **sac** gmora gte tdnt hneeied iew uioi ymneot t steyor ey pkfo

The Generated files are available in /Report/Files/1b and also on the website (text_ids 72-82)

1c) Now we use the frequency calculator to build frequency profile for the books shown below in the table. After generating texts of length 100000 characters, with both second order and third order profiles (2 and 3 character N-grams) to stimulate sequence generator. Results show using longer N-grams affects the word yield much more than using various frequency profiles.

	Second Order		**Third Order**	
	Yield	**Longest Word**	**Yield**	**Longest Word**
"Agnes Grey" by Anne Bronte	9.3%	*assure*	14.3%	*Lenses*
"Jane Eyre" by Charlotte Bronte	9.3%	*chewed*	14.2%	*Intense*
"Wuthering Heights" by Emily Bronte	9%	*blest*	14.4%	*shines*
"A Christmas Carol" by Charles Dickens	9%	*aired*	14.3%	*Hopes*
"Alice's Adventures in Wonderland" by Lewis Carrol	9.1%	*swears*	14.4%	*Things*

The Generated files are available in /Report/Files/1c, and also on the website (text_ids 83-92)

1d) The above test we repeated on frequency profile produced off of "Agnes Grey" with a various resolutions. The results show, by decreasing resolution a smaller group of top probability N-grams would be produced in the text. This means we are filtering low probability N-grams from the set and it increases the word yield. However, there is an exception to this observation; In case of one character N-grams, reducing resolution from 0.4 to 0.1 decreases the word yield. That is, because there is only 4 characters left in 0.1 highest probability bracket. The same is true for R=0.01, in this case there is only one available character left so it is impossible to create a word.

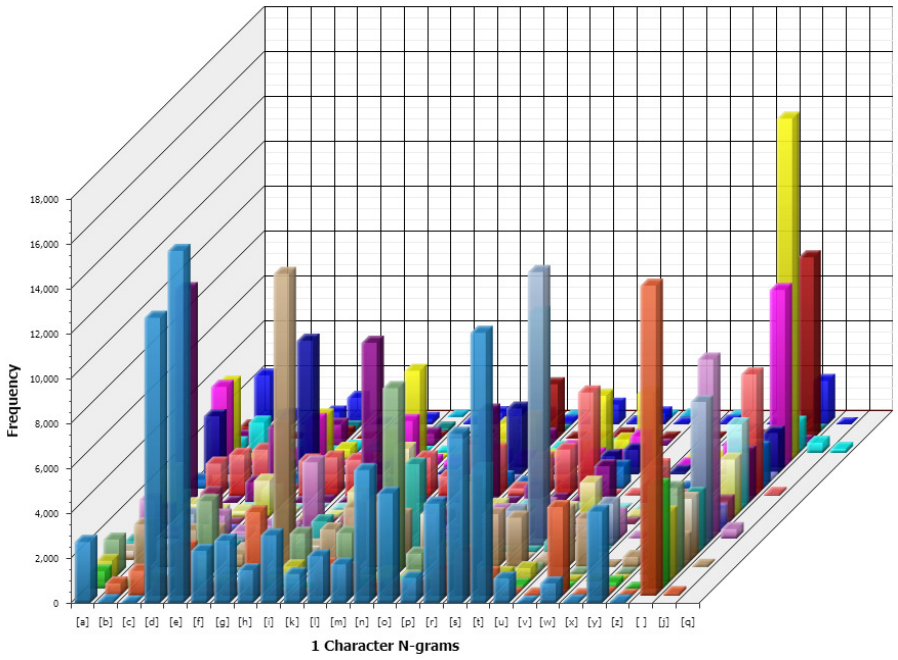
Resolution	**First Order**	**Second Order**	**Third Order**
0.01	**0%**	15.3%	31.7%
0.1	**3.9%**	14.2%	19.2%
0.4	**8.6%**	9.8%	15.5%
0.6	6.7%	9.7%	15%
0.9	6.3%	9.6%	14.1%
0.99	5.9%	9.2%	14.7%

The Generated files are available in /Report/Files/1d, and also on the website (text_ids 125-140)

Problems 1e, f

Building the correlation matrix is fairly easy taken into fact that all the numbers needed to create a 1x1 correlation matrix, already exists in a 2-character frequency profile. For this part the algorithm was generalized so that instead of taking in only one book, it can create the correlation matrix of as many frequency profiles as provided (BUILD_CORRELATION_MATRIX in the code). The result the arg-average of co-located cells of the 2 or more matrices. Since the books aren't of the same size, the frequencies must be normalized before averaging, so that all the books have the same influence on the output matrix. This technique is later used in part g, h and i to build class-correlation-matrices, for example to build a correlation matrix represents the N-gram distribution of Sci-Fi books.

1e) Here is a 3D view of the correlation matrix for the book "Jane Eyre". (More correlation matrices are available /Report/Files/1e and also on the website. A table view of this matrix will be on next page)



1f) For this part, first, a routine was implemented which to generate the most probable diagraph (MOST_PROBABLE_PATH in code) which always picks the highest probability not taking into account if the character is already used in the path. This algorithm will very easily fall into a repeating cycle. To get around this issue, the algorithm was modified to choose each transition in the table only once, meaning that if 'th' is used once in the path, the next time we get to 't' we can't follow it by 'h'. This solution differs a bit from the one in hand out. In the book if a character is produced once it won't be selected again. Here are most probable diagraphs, created according to various the "Legend of Sleepy Hollow" compared to the one computed for Poe's book. It is predictable that the book's algorithm will result in very similar results regardless of which book it is used on.

Correlation Matrix of the Book	Most Probable Diagraph Starting with t
"Legend of Sleepy Hollow"	the t and hin of wateredeng s istontitr beseas,
"Gold Bug" by Poe	the andisouryplf'bj

1x1 Correlation Matrix of "Jane Eyre" Book

	!	"	#	*	(	)	,	-	.	:	;	?	@	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
	0	14	252	11	352	200	0	0	0	0	0	0	3	13816	4619	3083	3844	1316	2727	2635	6407	7973	948	1021	2563	3813	2222	5411	1805	109	1842	7896	15217	1344	168	8075	36	1927	0
!	268	0	162	0	7	05	0	33	0	0	0	0	0	7	2	2	0	1	2	0	4	4	1	0	0	0	1	1	0	0	2	1	6	0	0	6	0	0	0
"	305	0	858	0	9	01	0	16	0	0	2	0	1	86	105	27	61	10	8	43	129	302	29	7	32	44	118	79	20	4	7	125	182	2	0	447	0	113	0
#	7	0	0	1	0	00	1	0	4	0	0	0	1	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
*	500	6	7	0	0	00	25	10	33	1	6	9	0	16	50	20	384	75	22	1	2	33	1	6	340	137	138	36	25	0	122	1094	2116	32	122	7	0	21	0
(	0	0	2	0	0	00	0	0	0	0	0	0	0	5	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	4	4	0	0	4	0	0	0
)	11	0	0	0	0	00	6	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0
,	7458	0	173	0	3	00	0	1	0	0	0	0	0	178	48	3	6	4	10	4	15	30	2	1	8	4	7	9	2	0	1	17	31	0	1	32	0	8	0
-	0	0	65	0	10	00	0	984	0	0	0	0	0	193	152	67	54	30	65	87	140	123	24	21	94	64	59	55	75	5	69	176	186	28	5	103	1	43	0
.	3196	0	808	0	13	02	3	1	62	1	1	0	0	61	41	39	6	9	2	3	33	129	16	0	4	14	18	15	9	1	1	51	139	1	0	111	0	11	0
:	48	0	374	1	0	00	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	3	1	0	0	0	0	0	0
;	1476	0	8	0	1	00	0	0	0	0	0	0	0	21	15	0	0	2	4	0	6	2	1	0	2	0	0	1	1	0	0	8	13	0	0	2	0	0	0
?	244	0	430	0	3	00	0	34	0	0	0	0	0	2	1	0	1	0	0	0	1	1	0	0	0	0	3	0	1	0	0	4	1	0	0	2	0	1	0
@	3	0	0	1	0	00	0	0	0	0	0	0	8	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
a	2720	0	1	0	18	00	17	235	14	2	3	1	0	0	757	922	1548	0	404	534	105	1715	12	677	2141	637	9523	0	386	0	2591	3579	3449	428	413	541	142	175	38
b	85	1	0	0	22	00	14	5	2	0	4	2	0	542	105	0	0	1905	0	0	0	304	5	0	583	1	5	1435	1	0	350	29	5	1349	0	0	0	424	0
c	21	0	0	0	3	00	3	1	3	1	2	0	0	1110	0	46	0	722	0	0	1398	111	0	1399	293	0	3	1824	0	4	344	2	248	179	0	1	0	10	0
d	1271	0	41	4	0	32	21	652	148	394	14	135	33	0	599	52	27	201	1650	40	169	71	1020	15	11	256	48	925	2052	36	1	340	583	144	225	27	52	0	519	0
e	1568	6	74	5	0	516	00	1176	272	792	25	276	101	0	2166	137	846	3256	1657	352	249	171	332	12	67	1627	870	3368	218	427	11	5952	2106	1985	199	31	270	183	1176	13
f	2316	1	1	0	0	26	00	119	24	72	2	15	6	0	333	6	9	2	556	492	1	12	570	0	0	165	4	10	1565	7	0	401	25	496	276	0	7	0	10	0
g	2777	2	0	1	0	30	01	375	105	214	8	78	21	0	372	8	2	8	1250	7	333	1388	435	1	1	162	13	65	1701	1	0	259	361	46	210	1	81	0	6	0
h	1460	1	6	0	0	22	00	245	49	96	0	28	16	1	3723	13	3	1	13069	8	1	11	3399	1	4	11	26	13	1606	3	0	269	33	1038	427	0	8	0	236	0
i	3015	3	0	0	372	00	41	18	50	2	11	8	0	83	36	673	1803	582	645	1408	15	35	6	342	1116	1723	7136	286	207	1	755	2056	3971	1	522	39	139	0	22
j	0	0	0	0	0	00	0	1	0	0	0	0	0	117	0	0	0	56	0	0	0	420	0	0	0	0	0	81	0	0	0	0	0	452	0	0	0	1	0
k	1283	1	9	0	0	55	00	238	38	126	2	35	29	0	53	5	0	5	1679	45	2	7	816	1	1	57	2	457	215	1	0	7	267	8	10	0	12	0	74	0
l	2078	1	0	1	0	7	00	498	47	137	2	34	12	0	1401	15	14	1842	2657	267	21	7	1610	1	228	3043	9	8	1705	96	1	18	236	145	179	39	106	0	502	0
m	1717	3	1	2	0	68	00	314	51	226	1	71	46	0	1211	181	1	3	2930	53	2	12	880	0	2	11	83	14	868	231	0	35	134	20	274	0	7	0	503	0
n	5916	4	2	2	0	2462	04	624	130	300	4	105	52	0	359	24	392	8299	2325	40	3949	35	802	20	412	155	6	183	2275	6	4	18	548	1629	164	22	26	5	473	1
o	4880	2	3	2	0	178	00	280	80	164	6	62	48	0	380	229	210	925	1982	046	240	106	360	11	713	1112	1825	4110	2060	498	5	2931	501	1928	624	149	2415	17	115	19
p	1117	8	1	0	29	02	142	51	67	0	27	11	0	432	5	3	2	903	2	0	82	409	0	0	558	1	1	329	277	1	476	150	195	222	0	6	0	21	0
q	0	0	0	0	0	00	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	178	0	0	0	0	0
r	4426	2	3	1	0	83	01	468	116	283	1	93	51	0	1151	61	90	593	3820	91	97	30	1276	1	252	220	193	760	1647	75	0	299	901	624	424	56	32	0	878	0
s	7537	5	7	3	0	169	04	1025	130	515	325	201	59	0	1953	41	244	25	2360	53	34	1554	788	11	211	351	124	147	2057	307	30	13	573	2866	387	11	196	0	93	0
t	1203	6	58	3	0	522	01	909	199	774	23	212	119	0	1010	35	374	38	2209	40	17	1219	0	1538	6	9	412	30	804	599	16	0	803	513	918	340	0	381	0	491	5
u	1093	1	4	1	0	164	00	76	11	44	1	12	36	0	77	162	589	247	96	57	498	5	157	1	155	1534	261	1640	1	973	0	1092	1382	2968	0	9	6	0	20	76
v	9	0	0	0	0	00	0	0	12	0	0	0	0	57	0	0	0	2517	0	0	0	181	0	0	0	12	0	33	0	0	1	0	0	0	0	0	0	10	0
w	865	1	4	1	0	29	00	157	56	123	1	40	14	0	3953	9	5	47	2175	41	4	1874	1104	0	6	67	4	721	1366	2	0	109	107	9	56	0	14	0	111	1
x	69	0	0	0	1	00	8	5	9	0	1	0	0	26	0	25	0	47	0	0	0	28	0	0	3	0	0	3	36	0	0	1	119	0	14	0	30	0	0
y	4072	3	5	0	188	00	639	126	282	12	109	54	0	102	260	12	26	459	15	9	26	220	2	3	49	32	77	1874	5	1	9	988	201	9	0	96	0	3	0
z	65	0	0	0	3	00	1	0	3	0	0	1	0	3	2	0	1	54	0	0	0	14	0	0	5	1	1	0	0	0	0	0	0	1	0	0	0	20	8

Problems 1g-i

In part 1g-i, the goal is to classify text documents using their character/word sequence distribution attributes.

The correlation matrix builder routine populates matrix representation of the N-gram probability distribution of a one or more books. As said in part 1f, this feature can be used to generate matrices that carry the N-gram distribution information of class of books.

Now by measuring the similarity between correlation matrices, it is possible to compare books to find the one that has the highest chance belonging to a genre or being written by a specific author. One of the many matrix similarity measures is the Euclidean distance.

$$Euclidean\ Distance\ of\ Matrices\ M1\ and\ M2 \quad = \sqrt{\sum_{j}\sum_{i}(\mathrm{M1}_{i,j} - \mathrm{M2}_{i,j})^2}$$

Function CALC_MATRIX_SIMILARITY calculates this distance between any two compatible sets of correlation matrices and records the results in a table.

Now for answering questions 1g-1i we need to create correlation matrices representing probability distribution of a class of books, and then to find the distance between our class correlation matrix and a test correlation matrix. For questions 1g-1i, I ran 4 experiments each answering a part of the questions.

1- Bronte Sisters:

Here is the Euclidean distance matrix between 3 Bronte sisters and Charles Dickens and Mark Twain's correlation matrices(for 2-Word N-grams). It can clearly be observed that word selection of Bronte sisters' books are closer to one another and further from other writers books.

	Charles Dickens	Emily Bronte	Anne Bronte	Charlotte Bronte	Mark Twain
Charles Dickens	0	.041	.056	.042	.036
Emily Bronte	.041	0	.023	.018	.034
Anne Bronte	.056	.023	0	.018	.038
Charlotte Bronte	.042	.018	.018	0	.033
Mark Twain	.036	.034	.038	.033	0

2- Author Attribution:

First I picked a subset of authors (Carroll, Irving, Twain, Doyle, Bronte & Wells) then created the combined correlation matrices for each of them, holding out " Warlord of Mars " and "A Connecticut Yankee ...". Here is the Euclidean distance matrices between these two books and correlation matrix representing the style of other writers for 3-Character N-gram Analysis:

	Lewis Carroll	Washington Irving	Sir Arthur Conan Doyle	Mark Twain*	Emily Bronte	H. G. Wells	warlord_of_mars	a_connecticut_yankee_
Lewis Carroll	0	.393	.247	.244	.285	.324	.362	.247
Washington Irving	.393	0	.202	.32	.359	.159	.228	.2
Sir Arthur Conan Doyle	.247	.202	0	.208	.18	.139	.154	.086
Mark Twain*	.244	.32	.208	0	.23	.247	.354	.089
Emily Bronte	.285	.359	.18	.23	0	.339	.402	.198
H. G. Wells	.324	.159	.139	.247	.339	0	.108	.119
warlord_of_mars	.362	.228	.154	.354	.402	.108	0	.202
a_connecticut_yankee_	.247	.2	.086	.089	.198	.119	.202	0

Note that the asterisk above Mark Twain in the table is indicating that some of his books have been held out (in this case "A Connecticut Yankee ...") from calculation of combined correlation matrix.

The "Warlord of Mars" has been attributed to H. G. Wells, since we didn't have any other book by its writer in our test set. Such attribution merely indicates similarity.

The "A Connecticut Yankee" has been attributed to Sir Arthur Conan Doyle, and the actual writer Mark Twain is the next in line. I can conclude that author attribution is happening but not with the best precision. To put this idea to test I'll retry this with 2-Word N-grams.

	Emily Bronte	Washington Irving	Sir Arthur Conan Doyle	Mark Twain*	Lewis Carroll	H. G. Wells	a_connecticut_yankee_	warlord_of_mars
Emily Bronte	0	.136	.054	.039	.096	.106	.038	.128
Washington Irving	.136	0	.082	.121	.137	.064	.095	.075
Sir Arthur Conan Doyle	.054	.082	0	.067	.118	.051	.033	.063
Mark Twain*	.039	.121	.067	0	.086	.096	.027	.135
Lewis Carroll	.096	.137	.118	.086	0	.137	.089	.178
H. G. Wells	.106	.064	.051	.096	.137	0	.063	.044
a_connecticut_yankee_	.038	.095	.033	.027	.089	.063	0	.092
warlord_of_mars	.128	.075	.063	.135	.178	.044	.092	0

We can see that now Mark Twain is the closest in N-gram distribution to his book "A Connecticut Yankee ...", so this time N-gram analysis found the right author.

3- Genre Attribution:

Since a book might fit into more than one genre, this time, we will use single book to generate correlation matrices of multiple classes. This can help the process of attribution because it is increasing the number of examples for each class. At the same time, since now we have different number of books for each genres the precision of attribution between genres varies depending on the genre. The following are the classes and book-class mapping I used for this experiment.

FAN: Fantasy, Super Natural, Sci-Fi, Fiction
ROM: Romance
ADV: Adventure
DRM: Drama, Everyday Life, Social Drama
MYST: Mystery, Crime, Detective, Police

	FAN	ROM	ADV	DRM	MYST
A Tale of Two Cities				x	
A Christmas Carroll				x	
Agnes Grey					
Jane Eyre		x		x	
Wuthering Heights					
Tarzan of the Apes	x		x		
Warlord of Mars	x		x		
The People that Time Forgot	x		x		
The Land that Time Forgot	x		x		
King Solomon's Mines	x		x		
Fanny Hill		x		X	
Alice's Adventures in Wonderland	x		x		
Through the Looking Glass	x		x		
Legend of Sleepy Hollow	x				
The Adventures of Sherlock Holmes			x	X	x
The Lost World	x	x			
The Hound of the Baskervilles				X	x
Tales of Terror and Mystery					x
Adventures of Huckleberry Finn			x		
The Adventures of Tom Sawyer			x		
A Connecticut Yankee ...	x				
War of the Worlds	x	x			
The Time Machine	x	x			
Metamorphosis	x				
The Trial	x				
The Jungle Book	x		x		

For this experiment, the books "Through the Looking Glass", The "Jane Eyre" and "Tales of Terror and Mystery" will be held out to test the attribution. The results are as follows (2-Word N-grams):

	through_the_looking_glass	jane_eyre	tales_of_terror_and_mystery	MYST	DRM	ADV	ROM	FAN
through_the_looking_glass	0	.094	.113	.1	.097	.103	.101	.114
jane_eyre	.094	0	.051	.038	.015	.037	.04	.041
tales_of_terror_and_mystery	.113	.051	0	.017	.026	.03	.023	.024
MYST	.1	.038	.017	0	.017	.031	.031	.028
DRM	.097	.015	.026	.017	0	.017	.018	.016
ADV	.103	.037	.03	.031	.017	0	.016	.004
ROM	.101	.04	.023	.031	.018	.016	0	.01
FAN	.114	.041	.024	.028	.016	.004	.01	0

The results show that, the most probable genre for "Tales of Terror and Mystery" is MYST which is correct. For "Jane Eyre" the genre DRM is suggested which is also true, but for the book "Through the Looking Glass" the genre DRM is suggested where it is actually a FAN/ADV genre.

4- Style Similarity: With the same principals as we had in the last experiments, for 2-Word N-grams, the following is the list of the authors with the most similarity in style.

Above results show that most similar authors in this set are Charlotte and Emily Bronte.

The Similarity Analysis Between Writes' Styles

	John Cleland	Lewis Carroll	Washington Irving	Sir Arthur Conan Doyle	Mark Twain	Anne Bronte	Charles Dickens	Charlotte Bronte	Edgar Rice Burroughs	Emily Bronte	H. Ryder Haggard	H. G. Wells	Franz Kafka	Rudyard Kipling	Nicolo Machiavelli
John Cleland	0	.378	.226	.151	.256	.125	.141	.152	.208	.2	.165	.234	.279	.372	.23
Lewis Carroll	.378	0	.393	.247	.226	.278	.229	.289	.278	.285	.257	.324	.222	.259	.391
Washington Irving	.226	.393	0	.202	.253	.337	.148	.332	.156	.359	.16	.159	.297	.226	.213
Sir Arthur Conan Doyle	.151	.247	.202	0	.138	.134	.059	.105	.096	.18	.066	.139	.137	.195	.187
Mark Twain	.256	.226	.253	.138	0	.149	.113	.168	.179	.196	.116	.174	.188	.153	.258
Anne Bronte	.125	.278	.337	.134	.149	0	.129	.051	.239	.084	.167	.272	.232	.326	.279
Charles Dickens	.141	.229	.148	.059	.113	.129	0	.116	.105	.142	.073	.142	.127	.148	.159
Charlotte Bronte	.152	.289	.332	.105	.168	.051	.116	0	.233	.074	.162	.268	.25	.328	.313
Edgar Rice Burroughs	.208	.278	.156	.096	.179	.239	.105	.233	0	.282	.085	.075	.159	.146	.159
Emily Bronte	.2	.285	.359	.18	.196	.084	.142	.074	.282	0	.223	.339	.255	.329	.345
H. Ryder Haggard	.165	.257	.16	.066	.116	.167	.073	.162	.085	.223	0	.095	.178	.142	.167
H. G. Wells	.234	.324	.159	.139	.174	.272	.142	.268	.075	.339	.095	0	.246	.182	.224
Franz Kafka	.279	.222	.297	.137	.188	.232	.127	.25	.159	.255	.178	.246	0	.188	.224
Rudyard Kipling	.372	.259	.226	.195	.153	.326	.148	.328	.146	.329	.142	.182	.188	0	.243
Nicolo Machiavelli	.23	.391	.213	.187	.258	.279	.159	.313	.159	.345	.167	.224	.224	.243	0

Implementation Review

For implementation of the said functionality I chose the Oracle stored procedures language, PLSQL, which is a declarative language, letting coder to run queries right from the code. The main benefit is that if the data is defined and indexed properly, a near to C language performance can be achieved. The performance consideration in my code is highly critical, because I for the web version, I needed file processing to happen in real time. In the two next parts I'll introduce the data model and the code structure of the project.

Data Model

Here is the data definition for my project:

1- USERS table

Each row in this table represents a user who can log in and upload files and analyze them. Deleting a user will delete all their information.	
USER_ID	(USERS_PK)
USER_NAME	Username
USER_PWD	User password for logging into system.
VALID_CHARSET	The non-repeating string including all characters that will be counted in N-gram analysis.
INVALID_CHARSET	A replacement character for any character in corpus that doesn't exist in P_VALID_CHARSET. P_INVALID_CHAR must exist in P_VALID_CHARSET.
WORD_DELIMITER	Word delimiter, mostly used for word N-gram analysis.
NUMBER_REPLACEMENT	All numbers will be replaced by this character. P_ NUMBER_REPLACEMENT must exist in P_VALID_CHARSET.

2- ENGLISH_WORDS table

Each row in this table is a meaningful word, that we user for counting the meaningful words in randomly generated text	
WORD	Word

3- CORPUS_LIST table

Each row in this table represents a text file, user uploaded .	
CORPUS_ID	(CORPUS_LIST_PK)
CORPUS_NAME	Name of the text document.
CORPUS_TEXT	Text file content.
USER_ID	The owner of this book in the system (USER_PK)

4- FREQUENCY_PROFILE table

Each row in this table holds the meta-data for rows in FREQUENCIES table associated with this row through FREQUENCY_PROFILE_ID. Rows in FREQUENCY table hold the number of appearances of an N-gram of type = FREQUENCY_TYPE (word/character) and length=FREQUENCY_N in the text file associated with this profile (through CORPUS_ID). For example if FREQUENCY_TYPE=word and FREQUENCY_N=2 and CORPUS is book1.text, the rows of FREQUENCY table indicate the number of time each distinct 2-Word sequence showed up in book1.txt	
FREQUENCY_PROFILE_ID	(FREQUENCY_PROFILE_PK)
FREQUENCY_TYPE	N-gram type (Word/Character)
FREQUENCY_N	N-gram length
CORPUS_ID	(CORPUS_LIST_PK)
STATUS	'PENDING' indicates that file is not completely processed yet, otherwise 'OK'
TOT_FREQUENCY	The total number of N-grams in the corpus

5-FREQUENCIES table

Each row in this table holds the number of appearances of N-gram described in FREQUENCY_PROFILE table in text file associated with it.	
GRAM	(FREQUENCY_PROFILE_PK)
FREQUENCY	N-gram type (Word/Character)
FREQUENCY_ID	Primary Key; (FREQUENCIES_PK)
CUM_SUM	Cumulative sum of frequencies of this row's frequency profile, ordered from low frequency to high frequency.
FREQUENCY_PROFILE_ID	(FREQUENCY_PROFILE_PK)

6- MONKEY_TEXT table

Each row in this table contains a text generated randomly according to a frequency distribution (identified by FREQUENCY_PROFILE_ID)	
TEXT_ID	Primary Key; (MONKEY_TEXT_PK)
FREQUENCY_PROFILE_ID	(FREQUENCY_PROFILE_PK; frequency profile according to which this text was generated.
TEXT_LENGHT	N-gram length
RESOLUTION	In case RESOLUTION <> 0, it means that the process of generating file was limited only to generate high frequency characters. The number RESOLUTION is between 0 and 1 and for example for 0.5 on when 100 distinct possible N-grams, the file was generate only with the 50 N-grams with highest frequencies;
TEXT	File Content
SATUS	'GENERATING TEXT' indicate the file hasn't been created yet, 'GENERATED PENDING ANALYSIS' means that the find is being analyzed to calculate the number of meaningful words in it.
TOT_WORD	Number of space delimited words in the text.
WORD_YEILD	The percentage of the generated text that is meaningful English words.
WORD_LIST	A list of longest words crated in the random text
SYSTEM_DT	The last date that this row was updates.

7- CORRELATION_MATRIX_DEF table

Each row in this table holds the specification of a correlation matrix.	
MATRIX_ID	Primary Key; (CORRELATION_MATRIX_DEF_PK)
MATRIX_NAME	Matrix Name; Can be used to identify the super-class of frequency profiles. (for example we can use the frequency profiles of a couple of comedy books, and name the super-class comedy, since this matrix has the average frequency distribution of all its frequency profiles.
FREQUENCY_TYPE	Common N-gram type (word/character) of participating frequency profiles.
FREQUENCY_N	Common N-gram length of participating frequency profiles.
STATUS	'PENDING' to indicate that the calculations of creating the correlation matrix are in progress, otherwise 'OK'
RESOLUTION	N/A
USER_ID	(USERS_PK); The owner of the matrix.
TOT_FREQUENCY	Total number of distinct N-grams in participating frequency profiles.

8- CORRELATION_MATRIX_DATA table

Each row in this table is a cell in a correlation matrix identifier by MATRIX_ID. A correlation matrix is an x*y matrix that holds the probability of a text started with N-gram1 being followed by N-gram2. Here y is always 1 and x + y = n = FREQUENCY_N. To fill this table we pick a frequency profile of length n, we break it down to a n-1 word/character N-gram and a 1 word/character N-gram. (for example if we have a frequency profile of length 3, we use it to create a 2x1correlation matrix, and use the 3-characer long N-grams frequency as the probability of producing 3rd character after the first and second character is produced)	
MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)
GRAM1	First N-gram; The rows in the correlation matrix
GRAM2	The second N-gram; 1 word/character long; the columns in the correlation matrix
FREQUENCY	Number of appearances of GRAM2 immediately after GRAM1 in this matrix's frequency profiles.

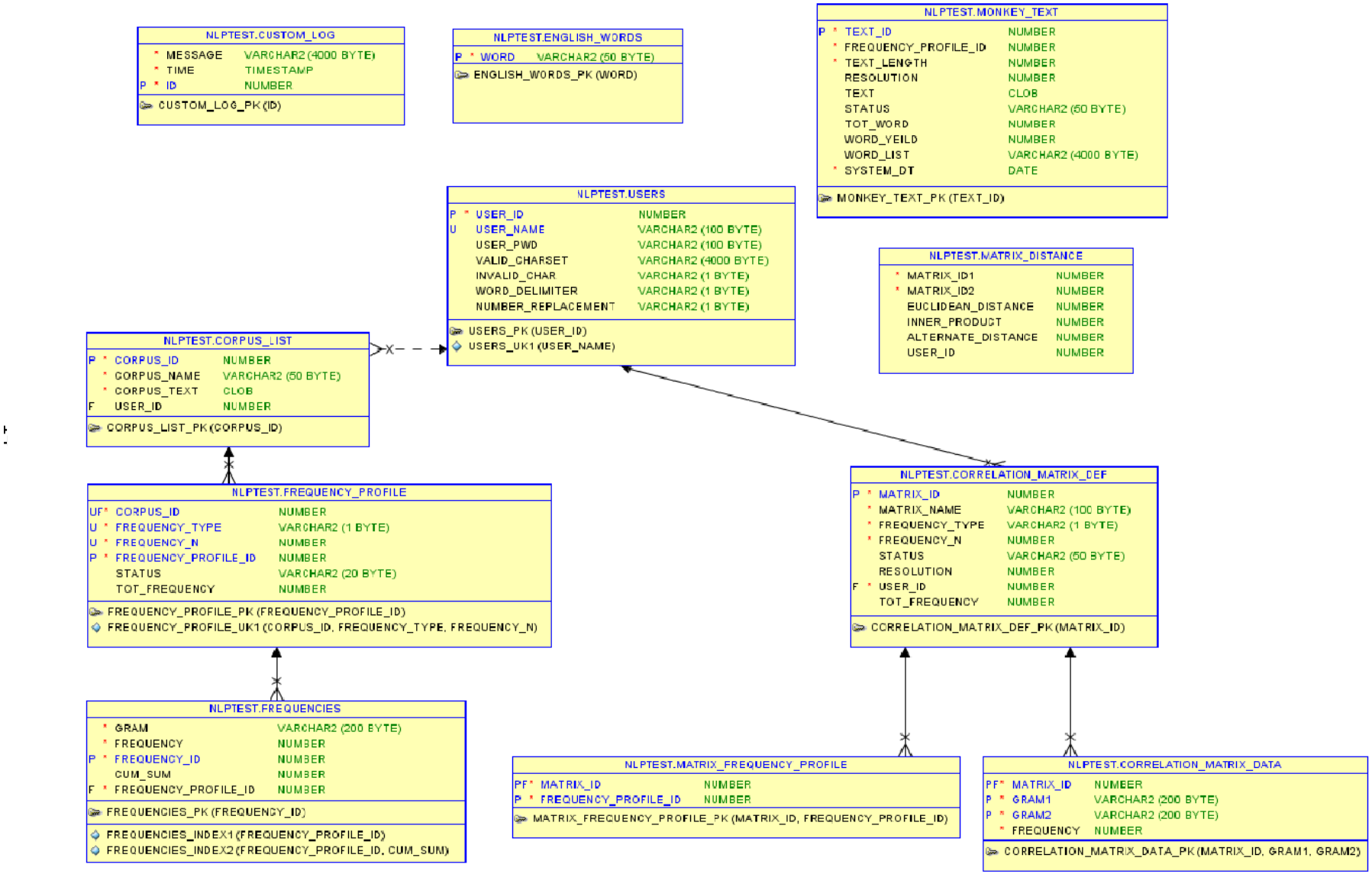
9- MATRIX_FREQUENCY_PROFILE table

This table represents the many to many relation between frequency profiles and correlation matrices. Our code can combine multiple frequency profiles into one correlation matrix, creating a super-class frequency distribution of N-grams. Therefore, we need to be able to show which frequency profiles participated in generation of a specific matrix. Rows of this table record this information.	
MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)
FREQUENCY_PROFILE_ID	(FREQUENCY_PROFILE_PK)

10 -MATRIX_DESTANCE table

Each row in this table holds both Euclidean distance and inner product of two matrices identified by MATRIX_ID1 and MATRIX_ID2; Procedure CALC_MATRIX_SIMILARITY makes sure that two matrices are of the same N-gram type and length.	
MATRIX_ID1	(CORRELATION_MATRIX_DEF_PK)
MATRIX_ID2	(CORRELATION_MATRIX_DEF_PK)
EUCLIDEAN_DISTANCE	Euclidean Distance
INNER_PRODUCT	Inner Product
ALTERNATIVE_DISTANCE	N/A
USER_ID	The owner of matrix1 and matrix2

Entity Relationship Model



Code Structure & Analysis

This Section contains the code structure of the system, the complete explanation of what each part does and how and also the time complexity of the routines and functions.

1- USER_LIB (user creation and login)

1-1 Function CHECK_LOGIN

Checks if the provided usrename and password belongs to a user and returns true/false.	
Running Time	N/A
P_USERNAME	(USERS.USER_NAME)
P_PASSWORD	(USERS.USER_PWD)

1-2 Procedure CREATE_USER

given a username and password creates a new user. users need to run FREQUENCY_LIB.INIT_FREQUENCIES once after user creation.	
Running Time	N/A
P_USERNAME	(USERS.USER_NAME)
P_PASSWORD	(USERS.USER_PWD)

2- FREQUENCY_LIB (analyzing the frequency of N-gram in a text file)

2-1 Procedure INIT_FREQUENCIES

Deletes all user related data, updates user defined alphabet N-gram analysis then analyzes the calls CALC_CORPUS_FREQUENCIES to analyze the alphabet itself as a corpus and build a uniformly distributed frequency profile called NO_CORPUS, which will be used later to generate purely random sequences of characters.	
Running Time	N/A
P_USER_ID	(USERS_PK)
P_VALID_CHARSET	The non-repeating string including all characters that will be counted in N-gram analysis.
P_INVALID_CHAR	A replacement character for any character in corpus that doesn't exist in P_VALID_CHARSET. P_INVALID_CHAR must exist in P_VALID_CHARSET.
P_WORD_DELIMITER	Word delimiter, mostly used for word N-gram analysis.
P_ NUMBER_REPLACEMENT	All numbers will be replaced by this character. P_ NUMBER_REPLACEMENT must exist in P_VALID_CHARSET.

2-2 Procedure CALC_CORPUS_FREQUENCIES

Records frequency profile descriptions in FREQUENCY_PROFILE_TABLE and calls the appropriate frequency analysis function to analyze the corpus identified by CORUPS_ID. CORPUS needs to be already uploaded in CORPUS_LIST table. A frequency profile is a row in FREQUENCY_PROFILE table that associates FREQUENCY_ID with its associate N-gram type & N-gram length.	
Running Time	N/A
P_CORPUS_ID	(CORPUS_LIST_PK)
P_FREQUENCY_TYPE	N-gram type (Word/Character)
P_FREQUENCY_N	N-gram length

2-3 Procedure CALC_CHAR_FREQUENCIES

For character N-grams of length= FREQUENCY_N=n (associated with P_FREQUENCY_PROFILE_ID in P_FREQUENCY table) slides a buffer of length n through the CORPUS file and counts the number of occurrences of any distinct sequence of n characters that show up in the file(from now on referred to as N-gram frequency) and puts the result in FREQUENIES table. it also calculates the cumulative sum of frequencies order form low frequencies to high frequencies.	
Running Time	O(n.log(m)) for n=length of corpus of distinct N-grams in FREQUENCIES table
P_FREQUENCY_PROFILE_ID	(FREQUENCY_PROFILE_PK)

2-4 Procedure CALC_WORD_FREQUENCIES

Similar to CALC_CHAR_FREQUENCIES but modified to count word N-grams	
Running Time	The order is the same as CALC_CHAR_FREQUENCIES, but it would take about twice as much time, because of an extra round of looping through file to split the words.
P_FREQUENCY_PROFILE_ID	(FREQUENCY_PROFILE_PK)

2-5 Procedure CLEAN_CORPUS

Replaces all occurrences of invalid characters with invalid character replacement character; Replaces number, too	
Running Time	N/A
P_USER_ID	(USERS_PK)
P_CORPUS_TEXT	Text file content

2-6 Procedure DELETE_CORPUS

DELETE_CORPUS deletes a text file and all its related information from all tables. if called with null CORPUS_ID, will delete all the corpuses for the a user (identified by USER_ID)	
Running Time	N/A
P_USER_ID	(USERS_PK)
P_CORPUS_ID	(CORPUS_LIST_PK)

2-7 Procedure DELETE_FREQUENCY_PROFILE

Deletes a frequency profile and all its dependent data from FREQUENCIES and FREQUENCY_PROFILE and MATRIX_FREQUENCY_PROFILE_TABLE	
Running Time	N/A
P_USER_ID	(USERS_PK)
P_FREQUENCY_PROFILE_ID	(FREQUENCY_PROFILE_PK)

2-8 Procedure CALC_CORPUS_FREQUENCIES

Adds a new text file to the CORPUS_LIST	
Running Time	N/A
P_USER_ID	(USERS_PK)
P_CORPUS_NAME	Text file name
P_CORPUS_TEXT	Text file content

3- MONKEY_TEXT_LIB (generating random text and analyzing it with dictionary)

3-1 Procedure GO

Inserts a new row to MONKEY_TEXT table and then calls GENERATE_TEXT to generate a pseudo-random text using the probability distribution indicated by P_FREQENCY_PROFILE_ID	
Running Time	N/A
P_FREQENCY_PROFILE_ID	(FREQUENCY_PROFILE_PK)
P_LENGTH	The non-repeating string including all characters that will be counted in N-gram analysis.
P_RESOLUTION	A replacement character for any character in corpus that doesn't exist in P_VALID_CHARSET. P_INVALID_CHAR must exist in P_VALID_CHARSET.

3-2 Procedure GENERATE_TEXT

Randomly generate a text of length P_LENGTH, with probability of producing each character/word set by the frequency profile. for example for the uniform frequency profile, probability of all N-grams is the same, but when using frequency profile of a book1.txt, the probability would equal to frequency of that word/character in book1.text.	
Running Time	O(m.log(n)) for m = P_LENGTH of text in characters and n = number of distinct N-grams in frequency profile
P_ TEXT_ID	(MONKEY_TEXT_PK)

3-3 Procedure ANALYZE_TEXT

Counts the number of meaningful words in the text generated by GENERATE_TEXT, using words in ENGLISH_WORD table as a reference.	
Running Time	O(m.log(n)) for m = the number of words in the text and n = the number of words in the ENGLISH_WORD table.
P_TEXT_ID	(MONKEY_TEXT_PK)

4- CORRELATION_MATRIX_LIB (builds the correlation matrices and calculates the distance between matrices)

4-1 Procedure BUILD_CORRELATION_MATRIX

This routine creates a new CORRELATION_MATRIX_DEF row to record the meta-data of a new correlation matrix, and depending on N-gram type (Word/Character) calls the appropriate procedure for populating CORRELATION_MATRIX_DATA	
Running Time	N/A
P_FREQUENCY_PROFILES	the list of frequency distribution profile ids (FREQUENCY_PROFILE_PK)
P_MATRIX_NAME	matrix name (CORRELATION_MATRIX_DEF.MATRIX_NAME)
P_FREQUENCY_TYPE	N-gram type (Word/Character) (FREQUENCY_PROFILE.FREQUENCY_TYPE)
P_FREQUENCY_N	N-gram length; builds a correlation matrix of the size (n-1 * 1)
P_RESOLUTION	between 0 and 1; top n frequency N-grams (FREQUECY_PROFILE.RESOLUTION); for P_RESOLUTION=0.5 on 100 N-grams gets the 50 N-grams with highest frequencies;
P_USER_ID	(USERS.USER_ID)

4-2 Procedure BUILD_CHAR_CORRELATION_MATRIX

This routine fills CORRELATION_MATRIX_DATA for Character N-grams. For N-gram length of n, combines all frequency profiles belonging to this MATRIX_ID in MATRIX_FREQUENCY_PROFILE and counts the number of occurrences of each N-gram with length of n-1 that ends up in a specific character. (for n=2; character N-grams; counts the number of 'aa' that end up 'b' over all frequency profiles associated with current MATRIX_ID)	
Running Time	$O(n^2)$ for number of frequency observations in FREQUENCIES table.
P_MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)
P_USER_ID	(USERS_PK)

4-3 Procedure BUILD_WORD_CORRELATION_MATRIX

Same as BUILD_CHAR_CORRELATION_MATRIX, but for Word N-grams	
Running Time	$O(n^2)$ for number of frequency observations in FREQUENCIES table.
P_MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)
P_USER_ID	(USERS.USER_ID)

4-4 Procedure CALC_MATRIX_SIMILARITY

Calculates the Euclidean distance and inner product of any two matrices in CORRELATION_MATRIX_DEF that have the same N-gram type and length. The result gets recorded in MATRIX_DISTANCE table.	
Running Time	O(n) for distinct N-grams in FREQUENCIES table
P_USER_ID	(USERS_PK)

4-5 Function MOST_PROBABLE_PATH

Calls appropriate function, depending on N-gram type (word/character) associated with P_MATRIX_ID, to generate sequence of length P_LENGTH characters using the most frequent N-grams in CORRELATION_MATRIX_DATA. Finally returns the sequence of words or characters.	
Running Time	N/A
P_MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)
P_START_WITH	The first n-1 characters/words of the most probable sequence (for N-gram length of n)
P_LENGTH	Maximum length of the word or character sequence to be built (#of characters)

4-6 Function MOST_PROBABLE_CHAR_PATH

Called by _PROBABLE_PATH for character N-grams, starting with P_START_WITH n-1 characters, continuously looks up the character with highest probability of occurring after the last n-1 characters (Uses every sequence of n characters only once). For example starting with P_START_WITH = 'ta' for character N-grams of length 3, finds the character with maximum frequency in CORRELATION_MATRIX_DATA that shows up after 'ta'. Assuming that said character is 'b', produces sequence 'tab' and repeats the process for 'ab' to reach a sequence of length P_LENGTH. also makes sure that 'tab' only shows up once in the sequence, to avoid cycles. Finally returns the sequence of characters.	
Running Time	O(l.log(m)) for l=P_LENGTH and m=number of distinct n-1 character N-grams for MATRIX_ID=P_MATRIX_ID
P_MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)
P_START_WITH	The first n-1 characters of the most probable sequence (for N-gram length of n)
P_LENGTH	Maximum length of the character sequence to be built (#of characters)

4-7 Function MOST MOST_PROBABLE_WORD_PATH

Same as MOST_PROBABLE_CHAR_PATH for word N-grams	
Running Time	O(l.log(m)) for l=P_LENGTH and m=number of distinct n-1 word N-grams for MATRIX_ID=P_MATRIX_ID
P_MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)
P_START_WITH	The first n-1 words of the most probable sequence (for N-gram length of n)
P_LENGTH	Maximum length of the word sequence to be built (#of characters)

4-8 Procedure DELETE_CORRELATION_MATRIX

Deletes a correlation matrix from CORRELATION_MATRIX_DEF, CORRELATION_MATRIX_DATA and correspoding distance values from MATRIX_DISTANCE ad MATRIX_FREQUENCY_PROFILE	
Running Time	N/A
P_MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)

4-9 Function BUILD_CORRELATION_MATRIX_CSV

Builds and exports a CSV file representation of the correlation matrix and outputs the file.	
Running Time	$O(n^2)$ for number of frequency observations in FREQUENCIES table (O(x.y), x being the number of distinct n-1 character sequences and y being number all valid characters.
P_MATRIX_ID	(CORRELATION_MATRIX_DEF_PK)
P_USER_ID	(USERS_PK)

Conclusion

This project helped me get a realistic grasp of the abilities and limitations of statistical language processing. The experiments in the assignment showed that it is possible to use the data of the character N-grams distribution for classification, however it is not precise and its abilities are very limited. I believe this kind of analysis can mostly be used as a secondary method limiting the search space or pruning the results for a more powerful classification algorithm.

However, processing word N-grams statistical data seems to extend the functionality drastically, which makes it suitable for some practical applications.

.

Web Version

An online version of this project is available at: https://gf93.ntree.com/a/f?p=201

Project Files

The directory accompanying this report, contains the following:

/(Project Root)

/Source

/Code/*	Quite well-commented PL/SQL packages source code
/Oracle Schema Setup/nlptest_nodata.dmp	Exported dump-file the database schema, can be imported back to oracle for further and future development. schema username & password: NLPTEST
/Oracle Apex App/f201.sql	Exported version of the web version, also can be imported into Oracle Apex.

/Report

/Files/*	Supporting files for question 1 parts a, b, c & e
/Big Assignment Report.docx	
/Big Assignment Report.pdf	