Εθνικο Μετσοβιο Πολυτεχνειο

Επεξεργασία Φωνής & Φυσικής Γλώσσας

Προπαρασκευή 1ου Εργαστηρίου

Γεωργίου Δημήτριος (03115106)

<el15106@central.ntua.gr>

20 Οκτωβρίου 2018

1 Κατασκευή Corpus

(α)

1. Κάνουμε import τις κατάλληλες βιβλιοθήκες *os, nltk, urllib*, συναρτήσεις των οποίων θα χρειαστούν για την κατασκευή του corpus.s

2. Κάνουμε χρήση της συνάρτησης *request.urlopen()* της βιβλιοθήκης *urllib* με σκοπό την αποστολή ενός HTTP request για την λήψη σε μορφή string του βιβλίου με τίτλο **"The Adventures of Sherlock Holmes"** από το site που προτάθηκε στην εκφώνηση και το οποίο βρίσκεται σε plain .txt μορφή. Επειδή είναι κωδικοποιημένο σε UTF-8, χρησιμοποιούμε την συνάρτηση *decode()* για την μετατροπή του σε Unicode, ενώ ταυτόχρονα αντικαθιστούμε τα carriage returns που έχουν δημιουργηθεί "r" με κενά με χρήση της *replace()*.

3. Επειδή αποζητούμε το ολοκληρωτικό build του project απο την αρχή, δημιουργούμε το κατάλληλο corpus directory με όνομα **"newcorpus.nosync"** μέσω της *os.mkdir()* της βιβλιοθήκης *os*.

4. Χρησιμοποιώντας κατάλληλα την συνάρτηση *open()*, μεταφέρουμε το Unicode string του corpus σε ένα .txt file με όνομα **"SherlockHolmes.txt"** που βρίσκεται εσσωτερικό του προηγούμενου directory.

5. Η *assert()* στο βήμα αυτό μας βοηθάει να ελέγξουμε αν η διαδικασία της δημιουργίας του directory και του αρχείου διεξήχθη ομαλά και ορθά.

6. Τέλος χρησιμοποιούμε την *PlaintextCorpusReader()* της βιβλιοθήκης *nltk* με ορίσματα το corpus directory και το .txt αρχείο, με σκοπό την τελική κατασκευή του corpus μας. Το .txt έχει λοιπόν διαμελιστεί καταλλήλως σε παραγράφους (*paragraphs*), προτάσεις (*sentences*) και λέξεις (*words*), με αντίστοιχη κλήση κατάλληλων συναρτήσεων, στοιχεία τα οποία θα χρησιμοποιηθούν για το 2ο βήμα του project. [1]

(β)

Γενικά όλα τα αρχεία σε ένα NLTK corpus ακολουθούν τους ίδιους κανόνες για προσπέλαση τους μέσω του NLTK module. Είναι συλλογή κειμένων, που δειγματολειπτείται (με κριτήριο την δημιουγία μεγάλου εύρους κειμενικών τύπων) και χρησιμοποιείται για την γλωσσολογικές αναλύσεις.

Σε μια προσπάθεια να δοθούν απαντήσεις σε πολλά ερευνητικά ερωτήματα, παρατηρείται ότι χρησιμοποιούνται διαφορετικά corpora, ανάλογα με την καταλληλότητα τους (external ή internal κριτήρια), τα οποία βέβαια πρέπει να έχουν δείχθει τέτοια επεξεργασσία ώστε να θεωρούνται 'αντιπρόσωποι' της γλώσσας που υποστσηρίζουν.

Σημειώνεται ότι ένα corpus που συνεχώς ενημερώνεται μπορεί να ανταποκριθεί άμεσα ακόμα και συχνές αλλαγές κάποια γλώσσσας, ενώ μπορεί να μας παρέχει έναν μεγάλο αριθμό παραδειγμάτων σε πραγματικό επικοινωνιακό επίπεδο. Τέλος, μπορεί να μας δώσει περισσσότερο αντικειμενικές αποδείξεις και το πιο σημαντικό απ' όλα ακριβή στατιστικά στοιχεία.

Στην κατεύθυνση αυτή, και εφόσον τα corpus δεδομένα ποσοτικοποιούνται, είναι πιο αξιόπιστα, φυσικά και σχετικά με συμφραζόμενα από τα intuitions, παρατηρείται το εξής φαινόμενο. Τα ευρήματα που βασίζονται αποκλειστικά σε κάποιο συγκεκριμένο corpus μας δίνουν πληροφορία **μόνο** για το τι είναι αληθές σε αυτό το corpus.

Εάν θέλουμε να διατηρήσουμε ενά σωστό representativenes αλλά ταυτόχρονα να οδηγηθούμε σε κάποιο corpus πιο ολοκληρωμένο (generalisation), το οποίο θα χρησιμοποιηθεί για αντίστοιχη εκμάθηση μοντέλων, **σημαντική** κρίνεται η ενσωμάτωση πολλών **παρεμφερών** corpora σε ένα μεγαλύτερο corpus. Σε αντιστοιχία με το παράδειγμα μας, αρκετών βιβλίων απο το gutenberg, αν θέλουμε να απαντήσουμε σε ερωτήματα που αφορούν την καλλιτεχνική φύση του συγγραφέα. Γενικά τα πλεονκέτηματα αυτής της τεχνικής είναι:

- Corpora για λεξιλογικές μελέτες **πρέπει** να είναι πολύ μεγαλύτερα από τα αντίστοιχα που χρησιμοποιούνται για γραμματικές μελέτες, γιατί το πρώτο οδηγεί στην ενασχολήση της συχνότητας κατανομής μίας λέξης, η οποία μπορεί να μοντελοποιηθεί.

- Ενώ corpora που αφορά ποσοτικές μελέτες (*quantitive analysis*) και εγχειρίδα σχολιασμού (*manual annotation*) είναι σχετικώς μικρά, αυτά που χρησιμοποιούνται σε NLP και language engineering τείνουν να είναι domain ή genre-specific, απαιτούν data τα οποία μπορούν να αποκτηθούν ευκολότερα από large corpora.

[1] Η *PlaintextCorpusReader()* θα χρησιμοποιήσει τις default συναρτήσεις `tokenize.sent_tokenize()` και `tokenize.word_tokenize()`της βιβλιοθήκης *nltk* για τον προαναφερθέντα διαχωρισμό

2 Προεπεξεργασία Corpus

(α)

Η `identity_preprocess()` είναι συνάρτηση που μπορεί να χρησιμοποιηθεί ως *default argument* (αν δεν δοθεί κάποια άλλη συνάρτηση στην θέση της) στην συνάρτηση *parser()* όπως αυτή ορίζεται στο β' ερώτημα.

(β)

Η συνάρτηση *parser()*, που θα χρησιμοποιηθεί αργότερα στα επόμενα βήματα, δέχεται ως ορίσματα το path ενός .txt corpus αρχείου και μια συνάρτηση *preprocess()*. Πιο συγκεκριμένα, θα διατρέχει το corpus αρχείο γραμμή - γραμμή, εφαρμόζοντας στο string κάθε γραμμής την συνάρτηση *preprocess()*. Άρα θεωρητικά, ανάλογα τον τρόπο που θέλουμε να επεξεργαστσούμε το κείμενο μας θα καλούμε την *parser()* με την κατάλληλη για την περίσταση *preprocess()*.

(γ)

Ορίσαμε την συνάρτηση *tokenize()* η οποία δέχεται ως όρισμα ένα string και κάνει process αυτού, μετατρέποντας Κεφαλαία σε πεζά, διατηρώντας μόνο αλφαριθμητικούς χαρακτήρες και τέλος, διασπώντας το σε μια λίστα από λέξεις. Επίσης, χρησιμοποιήσαμε την *" ".join(string.split())* για να εξαλείψουμε τα διπλότυπα κενά, και να μείνουμε μόνο με αυτά μεταξύ των λέξεων ώστε η *split()* να επιστρέψει μόνο αυτές σε μορφή λίστας. (και όχι extra κενά ως λέξεις)

Σημειώνεται ότι το **tokenization** γενικά είναι η διαδικασία της διάσπασης ενός αρχείου text σε λέξεις, φράσεις, σύμβολα ή σε άλλα στοιχεία με νοηματική αξία, τα οποία ονομάζονται tokens. Εμείς εδώ επικεντρωθήκαμε στο **word tokenization**. Η λίστα των tokens που παράγεται στην συνέχεια γίνεται input για περαιτέρω processing όπως parsing() ή text mining().

(δ)

Προκειμένου να αντιληφθούμε τις διαφορές που παρουσιάζει η *tokenize()* που δημιουργήσαμε με τις υπόλοιπες word tokenizers της βιβλιοθήκης *nltk* δώσαμε ένα κατάλληλο παράδειγμα κειμένου: `" A RoCk3!45.!Fell frOm334 ~.  heaven "`. Τα αποτελέσματα φαίνονται παρακάτω:

Table 1: Comparing the operation of different word tokelizers

	String Tested
Tokenizer	`A RoCk3!45.! Fell frOm334 ~. heaven`
TreebankWordTokenizer	`['A', 'RoCk3', '!', '45.', '!', 'Fell', 'frOm334', '~.', 'heaven']`
WordPunctTokenizer	`['A', 'RoCk3', '!', '45', '.!', 'Fell', 'frOm334', '~.', 'heaven']`
OurOwnTokenizer	`['a', 'rock', 'fell', 'from', 'heaven']`

Ο TreebankWordTokenizer() πράγματι χωρίζει τις λέξεις λαμβάνοντας υπόψιν και σημεία στίξης και κενά, σε αντίθεση με τον WordPunctTokenizer() που προβαίνει σε διαχωρισμό μόνο βάσει σημειών στίξης . Τέλος, ο tokenize() που κατασκευάσαμε όπως ήταν αναμενόμενο, αδιαφορούσε για σημεία στίξης τελείως, ενώ ο διαχωρισμός γίνοταν μόνο βάσει κενών. [2]

[2] ImportError: cannot import name PunktWordTokenizer, οπότε δεν έγινε η σύγκριση με το συγκεκριμένο tokenizer(), εφόσον δεν υποστηρίζεται από την έκδοση του ntlk που έχουμε κατεβάσει

3 Κατασκευή λεξικού και αλφαβήτου

(α)

Γενικότερα, τα tokens που δημιουργούμε γίνονται είσοδος για parsing() και όχι για text mining() τουλάχιστον το βήμα αυτό. Πιο συγκεκριμένα χρησιμοποιήσαμε την συνάρτηση *raw()* του newcorpus που είναι στιγμιότυπο της κλάσης PlaintextCorpusReader με σκοπό την εξαγωγή raw content του **"Sherlock-Holmes.txt"**

Στην συνέχεια καλούμε την συνάρτηση *parser()* με ορίσματα το raw content του .txt αρχείου και την συνάρτηση *tokenize()*, αναμένοντας την επιστροφή όλων λέξεων που υπάρχουν στο κείμενο, βάσει των κανόνων που ορίστηκαν για την *tokenize()* στο προηγούμενο βήμα. Ενδεικτικά:

- `corpus_preprocessed[:674]`

```
Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doyle

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.net


Title: The Adventures of Sherlock Holmes

Author: Arthur Conan Doyle

Posting Date: April 18, 2011 [EBook #1661]
First Posted: November 29, 2002

Language: English


*** START OF THIS PROJECT GUTENBERG EBOOK THE ADVENTURES OF SHERLOCK HOLMES ***



Produced by an anonymous Project Gutenberg volunteer and Jose Menendez
```

- `word_tokens[:100]`

```
['project', 'gutenbergs', 'the', 'adventures', 'of', 'sherlock', 'holmes', 'by', 'arthur', 'conan', 'doyle', 'this',
'ebook', 'is', 'for', 'the', 'use', 'of', 'anyone', 'anywhere', 'at', 'no', 'cost', 'and', 'with', 'almost', 'no', 'r
estrictions', 'whatsoever', 'you', 'may', 'copy', 'it', 'give', 'it', 'away', 'or', 'reuse', 'it', 'under', 'the', 't
erms', 'of', 'the', 'project', 'gutenberg', 'license', 'included', 'with', 'this', 'ebook', 'or', 'online', 'at', 'ww
wgutenbergnet', 'title', 'the', 'adventures', 'of', 'sherlock', 'holmes', 'author', 'arthur', 'conan', 'doyle', 'post
ing', 'date', 'april', '18', '2011', 'ebook', '1661', 'first', 'posted', 'november', '29', '2002', 'language', 'engli
sh', 'start', 'of', 'this', 'project', 'gutenberg', 'ebook', 'the', 'adventures', 'of', 'sherlock', 'holmes', 'produc
ed', 'by', 'an', 'anonymous', 'project', 'gutenberg', 'volunteer', 'and', 'jose', 'menendez']
```

(β)

Στο σημείο αυτό έπρεπε να υλοποιηθεί το αλφάβητο του newcorpus, δηλαδή το κανονικό σύνολο των γραμματών που αντιπροσωπεύει τα βασικά φωνήματα αυτού. Πιο συγκεκριμένα, ορίσαμε την συνάρτηση `parser_2()` με ορίσματα το raw content του .txt αρχείου και την συνάρτηση `tokenize_2()`, αναμένοντας την επιστροφή όλων **διαφορετικών** συμβόλων που υπάρχουν στο κείμενο, βάσει των κανόνων που ορίστηκαν για την `tokenize_2()`:

(a) Αρχικό *string.strip()* της κάθε γραμμής,

(b) Χρήση της *" ".join(string.split())* για εξάλειψη διπλότυπων κενών,

(c) Διάσπαση της λέξης βάσει κενών *string.split()*και επιστροφή του αποτελέσματος

Ενδεικτικά:

- `alphabet_tokens`

```
[' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', ':', ';', '?', '@', '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', '[', ']', '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', 'à', 'â', 'è', 'é']
```

4 Δημιουργία συμβόλων εισόδου/εξόδου

Ήδη από το 3ο βήμα β' ερώτημα έχουμε ταξινομήσει το αλφάβητο μας, με κριτήριο τον αριθμό ASCII του κάθε συμβόλου, επομένως η περαιτέρω αντιστοίχιση των συμβόλων με τα indexes θα λάβει χώρα βάσει των κανόνων της εκφώνησης.

Χρησιμοποιώντας κατάλληλα την συνάρτηση *open()* και *wirte()* περάσαμε στο αρχείο με όνομα **"chars.sym"** που βρίσκεται εντός του αρχικού directory **"newcorpus.nosync/"**, την αντιστοιχία του αλφαβήτου με αύξοντα σε αριθμό indexes. Αυτό θα χρησιμοποιηθεί στα επόμενα βήματα για την κατασκευή των FST's

<epsilon>	0	1	16	C	31	R	46	e	61						
<space>	1	2	17	D	32	S	47	f	62						
"	2	3	18	E	33	T	48	g	63						
#	3	4	19	F	34	U	49	h	64						
$	4	5	20	G	35	V	50	i	65	t	76				
%	5	6	21	H	36	W	51	j	66	u	77				
&	6	7	22	I	37	X	52	k	67	v	78				
'	7	8	23	J	38	Y	53	l	68	w	79				
(	8	9	24	K	39	Z	54	m	69	x	80				
)	9	:	25	L	40	[	55	n	70	y	81				
*	10	;	26	M	41	]	56	o	71	z	82				
,	11	?	27	N	42	a	57	p	72	à	83				
-	12	@	28	O	43	b	58	q	73	â	84				
.	13	A	29	P	44	c	59	r	74	è	85				
/	14	B	30	Q	45	d	60	s	75	é	86				
0	15														

5 Κατασσκευή μετατροπέων FST

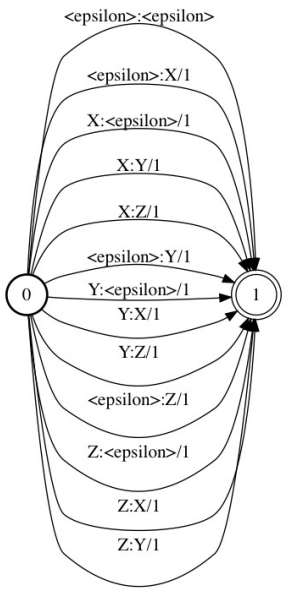
(α)

Υλοποιούμε έναν μετατροπέα όλων των γραμματών του αλφαβήτου, που αφορά το κείμενο μας (άρα Λατινικών), στον ευατό τους με μηδενικό κόστος και ο οποίος θα χρησιμεύσει στην αναγνώριση των χαρακτήρων μιας λέξης **που δεν είναι λάθος**. Το αυτόματο αυτό ουσιαστικά έχει μία κατάσταση που θα είναι και αρχική και τελική/αποδέκτης και καταλήγουμε σε αυτήν όταν έχουμε στην είσοδο ένας λατινικός χαρακτήρας.

Δημιουργούμε το αυτόματο `"orth_I.fst"` εκτελώντας στο shell: `$make orth_I`, με επιπρόσθετη γραφική δημιουργία ενός αρχείου .dot `"orth_I.dot"` και μιας εικόνας .jpg.

Αμέσως μετά δημιουργούμε τον μετατροπέα/αυτόματο `"orth_E.fst"` εκτελώντας στο shell: `$make orth_E`, το οποίο έχει 1 αρχική κατάσταση και 1 τελική στην οποία μπορούμε να μεταβούμε ως εξής:

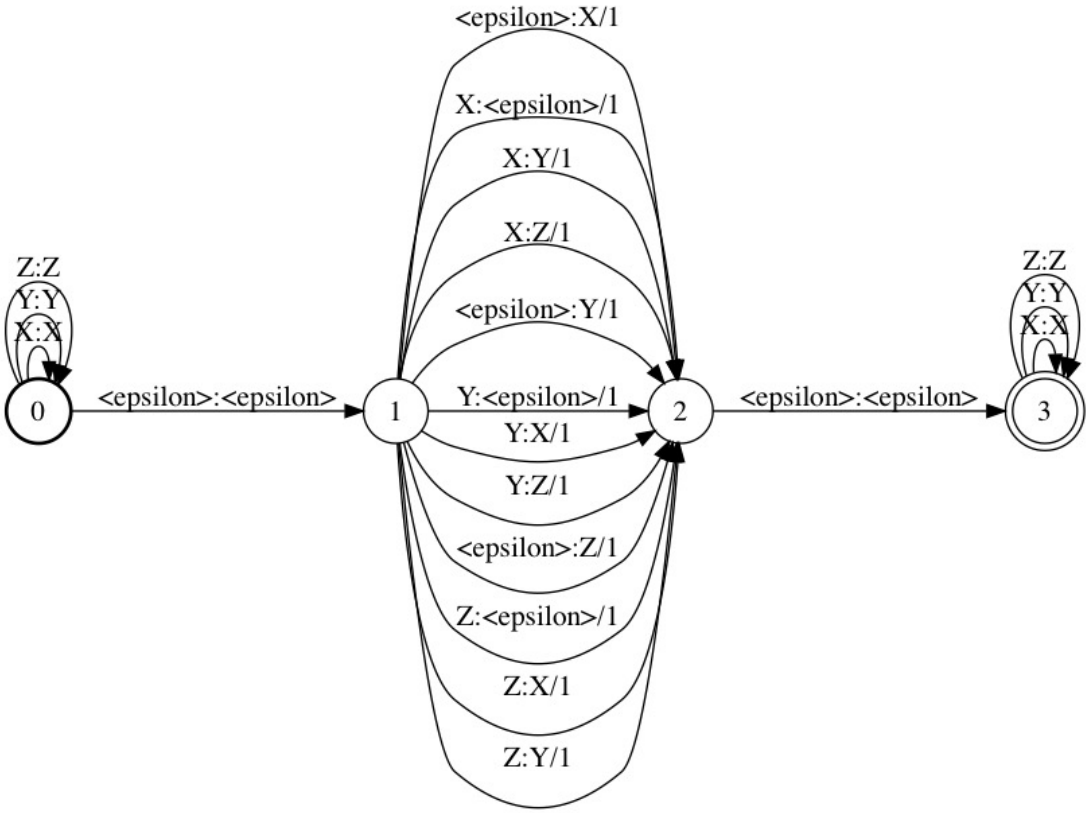
- Εισαγωγή χαρακτήρα. `(eps |X)`,
- Διαγραφή χαρακτήρα. `(X |eps)`,
- Αντικατάσταση χαρακτήρα. `(X |Y)`.



Τελικά για την δημιουργία του ζητούμενου μετατροπέα/αυτομάτου `"automato1.fst"` που βασίζονται στην απόσταση Levenshtein, με την υπόθεση ότι ανά μία λέξη υπάρχει μόνο ένα λάθος (εισαγωγή, διαγραφή ή αντικατάσσταση χαρακτήρα) με βάρος 1, χρησιμοποιούμε την εντολή *fstconcat()*. Πιο συγκεκριμένα `automato1 = orth_I |orth_E |orth_I`, ενώ η κλειστότητα αποφεύχθηκε εφόσον τοποθετήσαμε μετάβασση από την αρχική κατάσταση του automato1 στην τελική, αποκλειστικά με `(eps |eps)`.

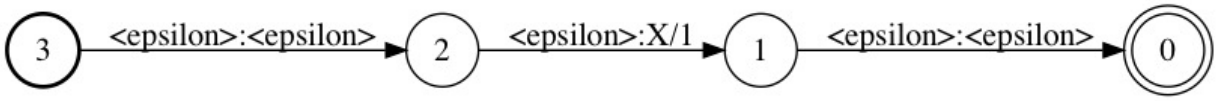
Δημιουργούμε τον μετατροπέα/αυτόματο `"automato1.fst"` εκτελώντας στο shell: `$make automato1`. Το παρουσιάζουμε γραφικά παρακάτω, αλλά πιο αφαιρετικά για 3 χαρακτήρες έστω X,Y, Z (αντιπροσωπεύουν οποιουσδήποτε 3 χαρακτήρες του λεξικού μας).

Σημειώνεται: το αυτόματο `"orth_I.fst"` αντιπροσωπεύει το `"orth_I_temp.fst"` εκτελώντας στο shell: `$make orth_I_temp`, που απασχολεί τους χαρακτήρες X, Y, Z.



Ο συγκεκριμένος μετατροπέας αφορά όλες τις λέξεις του λεξικού μας που έχουν 1 από τα προαναφερθέντα λάθη (ή και κανένα ==> μετάβαση σε κατάσταση με **μηδενικό κόστος**). Σημειώνεται ότι το κόστος της κάθε μετάβασης είναι 1.

Στο σημείο αυτό θα ασχοληθούμε με το shortest path του αυτομάτου. Πιο συγκεκριμένα, δημιουργούμε το αυτόματο `"automato1shortest.fst"` εκτελώντας στο shell: `$make automato1shortest`, με επιπρόσθετη γραφική δημιουργία ενός αρχείου .dot `"automato1shortest.dot"` και μιας εικόνας .jpg. Παρατηρούμε ότι η συγκεριμένη διαδικασία οδήγησε το πιο σύντομο μονής διάταξης μονοπάτι απο την αρχική κατάσταση στην τελική του αυτομάτου, δηλαδή αυτό που περιέχει τις μεταβάσεις με το λιγότεορ κόστος στην φυσική σειρά ημιδακτυλίου. Δηλάδη, όπως αναφέραμε και προηγουμένως, σε ένα αυτόματο με τρεις καταστάσεις (μία αν χρησιμοποιήσουμε και την *fstrmepsilon()*) στην τελική κατάσταση του οποίου θα καταλήγουμε όταν έχουμε στην είσοδο ένας λατινικός χαρακτήρας.



(β)

Έστω ότι έχουμε στην διάθεση μας δύο αρχεία (έστω **"testing.txt"** και **"correct.txt"**) που το κάθενα έχει μία λέξη ανά γραμμή, το 2ο περιέχει όλες τις λέξεις του λεξικού μας, ενώ το πρώτο training data, τις ίδιες λέξεις με λάθη.

Εξετάζουμε μία προς μία τις λέξεις των δύο .txt αρχείων και ελέγχουμε αν γίνεται με διόρθωση ενός μόνο λάθους (με τον τρόπο που περιγράφηκε στο α' ερώτημα) γίνεται να οδηγηθούμε ολοκληρωτικά στην σωστή λέξη. Σε περίπτωση που γίνεται, αυξάνουμε τον counter που αφορά το πλήθος διορθωμένων από ένα λάθος λέξεις.

Έτσι λοιπόν γίνεται εξαγωγή δεδομένων απο training set και δημιουργούμε τις αντίστοιχες πιθανότητες του κάθε λάθους, προφανώς κανονικοποιημένα στο πλήθος των λαθών (1/λέξη), ενώ τέλος παίρνουμε τον αρνητικό αλγόριθμο αυτών για να υπολογίσουμε τα κόστη μετάβασης κάθε λάθος στο νέο fst αυτόματο που δημιουργούμε.

6 Κατασκευή αποδοχέα λεξικού

(α)

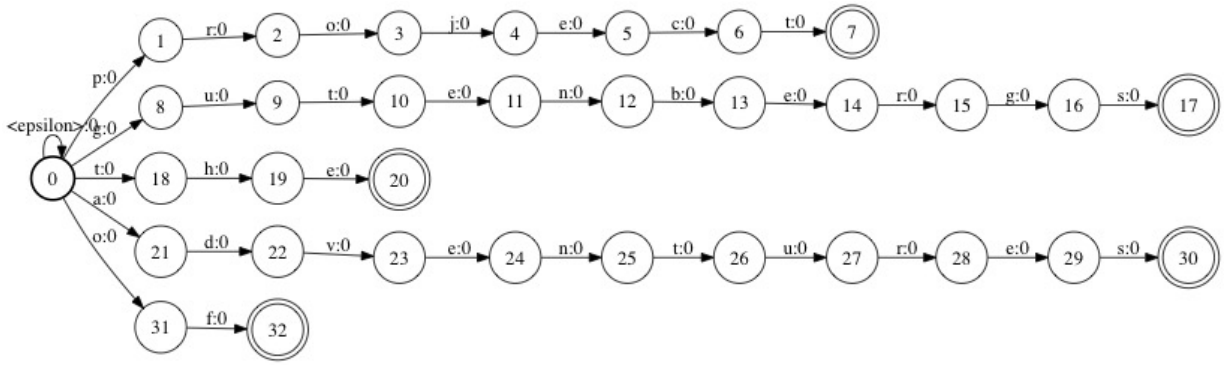
Αρχικά κατασκεύζουμε το dictionary μας στο οποίο αποθηκεύουμε το αλφάβητο που δημιουγήσαμε στο 3ο βήμα, άρα dict[i] = 'i-th symbol', οπότε η θέση του dictionary αντιστοιχεί στο index που είχαμε δώσει κατά την κατασσκεύη του αρχείου **"chars.syms"**.

Στην συνέχεια, δημιουργούμε το αρχείου `"orth_acceptor.txt"` πάνω στο οποίο θα πατήσουμε για τον τελικό σχηματισμό του `"orth_acceptor.fst"`. Συγκεκριμένα: την κάθε λέξη από το `word_tokens` που είχαμε επίσης δημιουργήσει στο 3ο βήμα, την διασπάμε σε γράμματα - letters:

- Αν έχουμε λέξη με 1 γράμμα, τότε πολύ απλά το γράφω στο αρχείο .txt και αρχική και τελική κατάσταση συμπίπτουν.
- Αν έχουμε λέξη με >1 γράμματα, τότε γράφω το πρώτο γράμμα και στην συνέχεια επαναληπτικά και τα υπόλοιπα,.

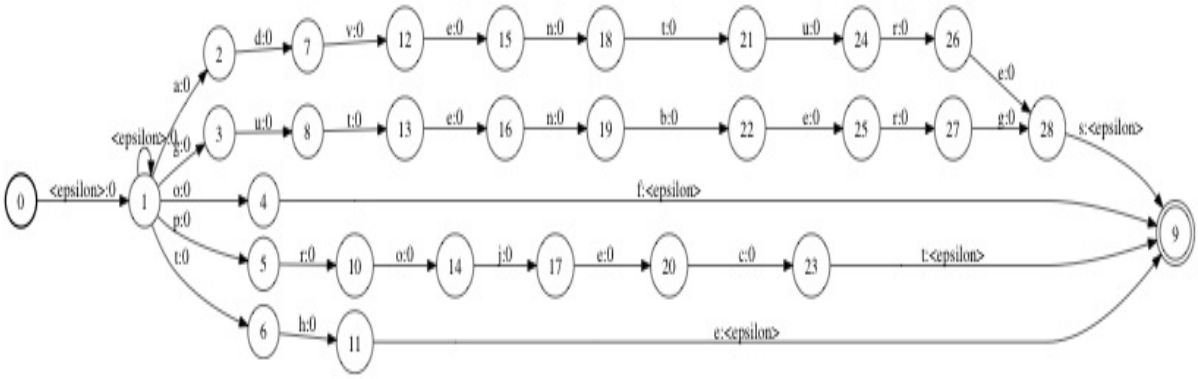
Σημειώνεται και είναι πολυ σημαντικό ότι έχουμε έναν αρχικό counter καταστάσεων `state_counter` ο οποίος αυξάνεται σε κάθε εμφάνιση γράμματος, ώστε να δημιουργηθούν με μεγάλη ακρίβεια τα μονοπάτια των λέξεων και **μόνο** του συνόλου των `word_tokens` μας (προκύπτουν τόσες κατάστασεις όσα και γράμματα). Σε κάθε άλλη περίπτωση πχ. μηδενισμός του counter κατά την εμφάνιση νέας λέξης, θα οδηγούσε σε αυτόματο που θα αποδεχόταν τον κάθε δυνατό συνδυασμό λέξεων που προκύπτουν από τις λέξεις που έχει το λεξικό μας.

Δημιουργούμε τον αποδοχέα/αυτόματο `"orth_acceptor.fst"` εκτελώντας στο shell: `$make orth_acceptor` και το αντίστοιχο αρχείου .dot `"orth_acceptor.dot"`. Το παρουσιάζουμε γραφικά παρακάτω, αλλά περιορισμένο σε 5 λέξεις απο τις συνολικές που υπάρχουν. Με την ίδια λογική σχηματίζονται και τα υπόλοιπα μονοπάτια αποδοχής των υπόλοιπων λέξεων.



(β)

Στο σημείο αυτό με τις εντολές *fstdeterminize()*, *fstminimize()*, παίρνουμε το αυτόματο του αποδοχέα που δημιουργήσαμε και το κάνουμε ντετερμινιστικό ενώ ταυτόχρονα το μειώνουμε. Σημειώνεται ότι ο ντετερμινισμός καθυστερεί αρκετά στην συγκεκριμένη έκδοση του OpenFst. Τέλος χρησιμοποιούμε την εντολή *fstrmepsilon()* με σκοπό την εξάλειψη μη απαραίτητων epsilon μεταβάσεων από το αυτόματο μας. Συνδυαστικά μετά την εκτέλεσση των εντολών μέσω του shell: `$make orth_acceptor_processed`. Γραφικά λοιπόν παρουσιάζεται παρακάτω:



Επιπροσθέτως, αξίον αναφοράς αποτελεί το γεγονός ότι η εκτέλεση της *fstshortestpath()* δεν θα είχε κανένα απολύτως νόημα στην παρούσα φάση, γιατί θα επιλεγόταν ένα άσχετο μονοπάτι από τα όλα δυνατά μονοπάτια με μηδενικά βάρη που θα σχημάτιζε μια συγκεκριμένη τελείως άσχετη λέξη.

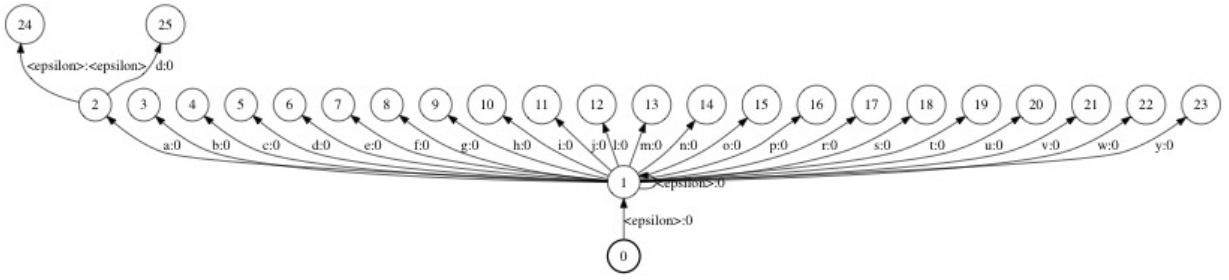
7 Κατασκευή Ορθογράφου

(α)

Τελικά δημιουργούμε τον ορθγογράφο μας `"orthograph.fst"` εκτελώντας στο shell: `$make orthograph`, οποίο προκύπτει από την σύνθεση του sorted tranducer του 5ου βήματος και του sorted αποδοχέα 6ου βήματος (sorted γιατί αλλιώς οδηγούμαστε σε FATAL error).

Ο Levenshtein tranducer διορθώνει τις λέξεις χωρίς να λαμβάνει υπόψιν γλωσσικές πληροφοριές, ώστε την επίτευξη min edit distance cheking. Παρακάτω παρατίθεται κομμάτι των μεταβάσεων του **"orthograph.txt"** όπως αυτό προέκυψε μετά το compose των δύο αυτομάτων μας, και ενα υποτύπωδες κομμάτι του συνολικού ορθογράφου.

```
0       1       <epsilon>       0
1       1       <epsilon>       0
1       2       a       0
1       3       b       0
1       4       c       0
1       5       d       0
1       6       e       0
1       7       f       0
1       8       g       0
1       9       h       0
1       10      i       0
1       11      j       0
1       12      l       0
1       13      m       0
1       14      n       0
1       15      o       0
1       16      p       0
1       17      r       0
1       18      s       0
1       19      t       0
1       20      u       0
1       21      v       0
1       22      w       0
1       23      y       0
2       24      <epsilon>       <epsilon>
2       25      t       <epsilon>
2       26      n       <epsilon>
2       27      l       0
2       28      p       0
2       29      r       0
2       30      d       0
2       31      u       0
2       32      w       0
3       33      <epsilon>       <epsilon>
3       25      y       <epsilon>
4       34      o       0
5       35      a       0
5       36      o       0
6       37      b       0
6       38      n       0
7       39      i       0
7       40      o       0
8       41      i       0
8       42      u       0
9       43      o       0
10      44      <epsilon>       <epsilon>
10      25      s       <epsilon>
10      25      t       <epsilon>
10      45      n       0
11      46      o       0
12      47      a       0
12      48      i       0
13      3       a       0
13      49      e       0
14      50      <epsilon>       <epsilon>
14      51      o       <epsilon>
15      52      <epsilon>       <epsilon>
15      25      f       <epsilon>
15      25      r       <epsilon>
15      53      n       0
16      54      o       0
200     69      <epsilon>       0       1
200     69      y       0       1
200     69      b       0       1
200     69      v       0       1
200     69      u       0       1
200     69      z       0       1
200     69      c       0       1
200     69      j       0       1
200     69      m       0       1
200     69      r       0       1
200     69      o       0       1
200     69      p       0       1
200     69      q       0       1
200     69      n       0       1
200     69      a       0       1
200     69      k       0       1
200     69      e       0       1
200     69      w       0       1
200     69      i       0       1
200     69      x       0       1
200     69      f       0       1
200     69      g       0       1
201     104     n       0
202     203     r       0
203     204     g       0
204     205     n       0
205     81      e       0
```



Όσον αφορά την βαρύτητα των edits, επεκτείνοντας τα όσα είπαμε στο β' ερώτημα 5ου βήματος, αξίζει να σημειωθεί ότι η ανάπτυξη και η εφαρμογή weighted finite-state μετατροπέων που μπορούν να "αντιπροσωπεύσουν" όλες τις πεπερασμένες καταστάσεις και διαχειρισιακές διαδικασίες (management operations) στον τομεά της αναγνώρισης φωνής, δημιουγούν ευκαιρίες για συνεχείς βελτιστοποιήσεις και ανταγωνιστική απόδοσση σε πολλά tasks αναγνώρισης φωνής.

Γενικότερα η απόδοση του μετατροπέα μεταβάλλεται ανάλογα τα weigths που θα τοποθετηθούν στις μεταβάσεις. Παρόλα αυτά, η χρήση π.χ νευρωνικών δικτύων ή άλλων machine learning μοντέλων (some efficient modelling), οδηγεί στην εύρεση των καταλληλότερων weights για κάθε μετάβαση γραμμάτων και λέξεων.

Σχόλιο: Τα μοντέλα που θα χρησιμοποιηθούν για τον υπολογισμό των weights, μειώνουν την ανάγκη για σχηματισμό μιας σύνθετης τοπολογίας (κυρίως με χρήση LSTMs), αντιθέτως βασίζονται σε μορφολογικές επανεμφανίσεις και λήματα.

(β)

Εκτελέσαμε τις παρακάτω εντολές του Makefile και παρατηρήσαμε ότι για την συγκεκριμένη λέξη **"cit"** η εναλλακτική που δόθηκε είναι η λέξη **"wit"**.

```
check_cit:
    fstcompile --isymbols=chars.syms --acceptor=true cit.txt cit.fst
    fstcompose cit.fst orthograph.fst possible_words.fst
    fstshortestpath possible_words.fst possible_words.fst
    fstrmepsilon possible_words.fst possible_words.fst
    fstprint --isymbols=chars.syms --osymbols=chars.syms possible_words.fst
```

```
3       2       c       w       1
0
1       0       t       t
2       1       i       i
```

8 Αξιολόγηση Ορθογράφου

(α)

Για κάθε λέξη του `"spell_checker_test_set.txt"` δημιουργούμε έναν αποδοχέα `"word_acceptor.fst"` με την γνωστή πλεόν διαδικασία όπως στα προηγούμενα βήματα. Εφόσον έχουμε ήδη δημιουργήσει τον ορθογράφο στο 7ο βήμα δεν απαιτείται η σύνθεση 3 FST's, του μετατροπέα, του αποδοχέα του λατινικού λεξικού και της λέξης, αλλά αντιθέτως μόνο του ορθογράφου με την προς έλεγχο λέξη μας.

Στην συνέχεια ελέγχουμε:

- Αν η λέξη δεν είναι στο κείμενο τότε αν ο αρθογράφος μας βρει μία best λέξη που είναι ίδια με αυτή που περιέχει στο κείμενο τότε την εισάγουμε στις `"corrected_words"`.
- Αλλιώς θα την αντιστοιχίσει σε κάποια λέξη που είναι λάθος οπότε και την εισάγουμε στην κατηγορία των `"wrong_words"`, ενώ αν δεν μπορεί να την αντιστοιχίσει κάπου στις `"no_matching_words"`.

Πλέον είμαστε σε θέση να εξετάσουμε την διαδικασσία για το μοντέλο μας και να εκτυπώσουμε τα τελικά αποτέλεσμα σστο python script μας. Τα αποτελέσματα φαίνονται παρακάτω:

- Με newcorpus = **"The Adventures of Sherlock Holmes"**, το οποίο αποτελείται απο 107797 λέξεις, για τις 20 ορθές λέξεις (έλεγχος 48 λέξεων) έχουμε:

```
Our Orthograph gave the following results

Corrected Words 15
Wrong Words 9
There was no matching for 24 words
```

- Με newcorpus = **"History of Atchison County Kansas"**, το οποίο αποτελείται από 372476 λέξεις, για τις 20 ορθές λέξεις (έλεγχος 48 λέξεων) έχουμε:

```
Our Orthograph gave the following results

Corrected Words 15
Wrong Words 9
There was no matching for 24 words
```

- Με newcorpus = **"History of the Royal Regiment of Artillery Vol. 2"**, το οποίο αποτελείται απο 169846 λέξεις, για τις 20 ορθές λέξεις (έλεγχος 48 λέξεων) έχουμε:

```
Our Orthograph gave the following results

Corrected Words 11
Wrong Words 9
There was no matching for 28 words
```

Παρατηρήσεις:

1. Παρά το γεγονός ότι το 2ο βιβλίο είναι σχεδόν 4 φορές το μέγεθος του 1ου, για τις ίδιες λέξεις έδωσαν τα ίδια αποτελέσματα, πράγμα που μας οδηγεί στο συμπέρασμα ότι πρώτον ορθογράφοι που έχουν σχηματιστεί με βάρη εμάνισης όλων των λέξεων ίσα με 1 δουλεύουν με αρκετά αφελή τρόπο, και δεύτερον δεδομένου του πρώτου, το 1ο βιβλίο έχει μεγαλύτερη γλωσσική ποικιλλομορφότητα για αυτό ίσως μπόρεσε να ανταποκριθεί επάξια με το 2ο βιβλίο.

2. Γενικά σε κάθε περίπτωση το ποσοστό διόρθωσης των εσφαλμένων λέξεων είναι πολύ μικρό και για αυτό πρέπει να γίνει reconfigure του ορθογράφου. Μάλιστα, το να γινει configure με μεγαλύτερο πλήθους corpus και πιο συγκεκριμένο για τις λέξεις που επιζητούμε διόρθωση, δεν θα βελτιώσει σχεδόν καθόλου την απόδοση του ορθογράφου μας αν δεν στοχεύσουμε στα βάρη εμφάνισης γραμμάτων και λέξεων ως προς τις προτάσεις και το κείμενο που εμφανίζονται.

9 Εξαγωγή αναπαραστάσεων word2vec

(α)

Στο σημείο αυτό έπρεπε να βρεθούν όλες οι προτάσεις του newcorpus. Πιο συγκεκριμένα, ορίσαμε την συνάρτηση `parser_3()` με ορίσματα το raw content του .txt αρχείου και την συνάρτηση `tokenize_2()`, αναμένοντας την επιστροφή όλων **διαφορετικών** προτάσεων που υπάρχουν στο κείμενο, βάσει των κανόνων που ορίστηκαν για την `tokenize_2()`:

```
['project gutenbergs the adventures of sherlock holmes by arthur conan doyle', 'this ebook is for the use of anyone a
nywhere at no cost and with', 'almost no restrictions whatsoever you may copy it give it away or', 'reuse it under th
e terms of the project gutenberg license included', 'with this ebook or online at wwwgutenbergnet', 'title the advent
ures of sherlock holmes', 'author arthur conan doyle', 'posting date april ebook', 'first posted november', 'language
english', 'start of this project gutenberg ebook the adventures of sherlock holmes', 'produced by an anonymous projec
t gutenberg volunteer and jose menendez', 'the adventures of sherlock holmes', 'by', 'sir arthur conan doyle', 'i a s
candal in bohemia', 'ii the redheaded league', 'iii a case of identity', 'iv the boscombe valley mystery', 'v the fiv
e orange pips', 'vi the man with the twisted lip', 'vii the adventure of the blue carbuncle', 'viii the adventure of
the speckled band', 'ix the adventure of the engineers thumb', 'x the adventure of the noble bachelor', 'xi the adven
ture of the beryl coronet', 'xii the adventure of the copper beeches', 'adventure i a scandal in bohemia', 'i', 'to s
herlock holmes she is always the woman i have seldom heard', 'him mention her under any other name in his eyes she ec
lipses', 'and predominates the whole of her sex it was not that he felt', 'any emotion akin to love for irene adler a
ll emotions and that', 'one particularly were abhorrent to his cold precise but', 'admirably balanced mind he was i t
ake it the most perfect', 'reasoning and observing machine that the world has seen but as a', 'lover he would have pl
aced himself in a false position he never', 'spoke of the softer passions save with a gibe and a sneer they', 'were a
dmirable things for the observerexcellent for drawing the', 'veil from mens motives and actions but for the trained r
easoner', 'to admit such intrusions into his own delicate and finely', 'adjusted temperament was to introduce a distr
acting factor which', 'might throw a doubt upon all his mental results grit in a', 'sensitive instrument or a crack i
```

(β)

Τις προτάσσεις αυτές στην συνέχεια τις χρησιμοποιήσαμε ως input στο model μας, το οποίο και το εκπαιδεύσαμε σύμφωνα με τα ζητούμενα της εκφώνησης. Τα αποτελέσματα φαίνονται παρακάτω:

(γ)

Χρησιμοποιοώντας την σσυνάρτηση closest() καταφέραμε να βρούμε τις σημασιολογικά κοντινότερες απο 10 λέξεις που επιλέξαμε τυχαία, όπως φαίνεται παρακάτω:

```
how [('what', 0.4905577600002289), ('that', 0.36776214838027954)]
surprised [('sorry', 0.4418948292732239), ('mistaken', 0.4340001344680786)]
smokes [('farthest', 0.5450403690338135), ('rack', 0.4603264331817627)]
savagely [('lashed', 0.7751080393791199), ('engine', 0.3780657649040222)]
she [('he', 0.7332376837730408), ('i', 0.6121480464935303)]
stooped [('changed', 0.4599582552909851), ('hint', 0.42196065187454224)]
penknife [('cigarholder', 0.7258424162864685), ('blunt', 0.6908803582191467)]
lay [('pattered', 0.41960039734840393), ('gets', 0.3846554160118103)]
detail [('shave', 0.37848228216171265), ('perceive', 0.3738561272621155)]
fierce [('beard', 0.40205949544906616), ('communication', 0.4017528295516968)]
```

Παρατηρήσεις:

Τα αποτελέσματα δεν είναι τόσο ποιοτικά όσο περιμέναμε, ενώ βελτιώνονται αυξάνοντας μέγεθος παραθύρου και τον αριθμό των εποχών, και κατ' επέκταση το πόσο "train" έχουμε κάνει το μοντέλο μας. Παρόλα αυτά πρέπει να είμαστε προσεκτικοί για αποφυγή overfeet, με την έννοια ότι το μέγεθος του παραθύρου θα πρέπει να κάνει capture το περιεχόμενο μιάς λέξης και όχι παραπάνω, καθώς θα μειώσουμε την ποιότητα του trained μοντέλου μας. Τέλος ενώ αυξάνοντας το μέγεθος των εποχών, οδηγούμαστε πάντοτε σε καλύτερη απόδοση μοντέλου, αυξάνεται δραματικά το training time, οπότε είναι tradeoff μεταξύ απόδοσης και training time.

10 ΠΑΡΑΡΤΗΜΑ

Παραπάνω αναλύθηκαν όλα τα βήματα που για την διαδικασία δημιουργίας του ορθογράφου. Για διευκόλυνση της διαδικασίας αυτής και για λόγους αυτοματοποίησης, έχουμε δημιουργήσει ένα Makefile που περιλαμβάνει όλες τις απαραίτητες εντολές που απαιτείται να δοθούν στο shell σχετικά με την βιβλιοθήκη openfst. Βέβαια αν εκτελεστεί το python script `"SLP_lab1.ipynb"`, αυτό από μόνο του τρέχει τα απαραίτητα system calls για τα αντίστοιχα makes σε κάθε βήμα.

11 References

1. Johnson, Mark Hasegawa. "OpenFST." *Isle Illinois*, http://www.isle.illinois.edu/sst/courses/minicourses/2009/lecture6.pdf
2. "Corpus Creation." Lancaster, www.lancaster.ac.uk/fass/projects/corpus/ZJU/xpapers/Xiao_corpus_ creation.pdf.
3. sentdex. "NLTK Corpora - Natural Language Processing With Python and NLTK P.9." *YouTube*, YouTube, 10 May 2015, www.youtube.com/watch?v=TKAXDqoG2dc.
4. "Libelli." *NLTK Corpus Reader for Extracted Corpus Libelli*, 12 Apr. 16AD, bbengfort.github.io/snippets/2016/04/12/nltk-corpus-reader.html.
5. Gries, Stefan Th. "What Is Corpus Linguistics?" *Linguistics UCSB*, www.linguistics.ucsb.edu/faculty/stgries/research/_CorpLing_LangLingCompass.pdf
6. *6. Learning to Classify Text*, www.nltk.org/book/ch06.html.

A Appendix

SLP_lab1

November 2, 2018

In [1]:
```
import os, os.path
import nltk
from shutil import copyfile

#SSL Certificate has fauled
#that not in the system certificate store.
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
#PlaintextCorpusReader will use the default nltk.tokenize.sent_tokenize()
#and nltk.tokenize.word_tokenize() to split your texts into sentences and words

from urllib import request
```

In [2]:
```
#----------------------------------------------STEP 1---------------------------------
#Text number 1661 is "The Adventures of Sherlock Holmes" by Arthur Conan Doyle, and we
url = "http://www.gutenberg.org/cache/epub/1661/pg1661.txt"
response = request.urlopen(url)
corpus = response.read().decode('utf8')
corpus = corpus.replace('\r', '')
length_corpus = len(corpus)
print(length_corpus)
print(corpus[:100])
```

```
581864
Project Gutenberg's The Adventures of Sherlock Holmes, by Arthur Conan Doyle

This eBook is for the
```

In [3]:
```
# Make new dir for the corpus.
corpusdir = 'newcorpus.nosync/'
if not os.path.isdir(corpusdir):
    os.mkdir(corpusdir)

copyfile("Makefile", corpusdir + "Makefile")
copyfile("spell_checker_test_set.txt",corpusdir + "spell_checker_test_set.txt")
```

Out[3]:
```
'newcorpus.nosync/spell_checker_test_set.txt'
```

In [4]:
```
# Output the files into the directory.
filename = 'SherlockHolmes.txt'
with open(corpusdir+filename, 'w') as f:
    print(corpus, file=f)
```

In [5]:
```
#Check that our corpus do exist and the files are correct.
# Key Note:
# 1.We split each file into words and we their equality until the penultimate word, si
#in the created file
assert open(corpusdir+filename,'r').read().split(' ')[:-1] == corpus.split(' ')[:-1]
```

In [6]:
```
# Create a new corpus by specifying the parameters
# (1) directory of the new corpus
# (2) the fileids of the corpus
# NOTE: in this case the fileids are simply the filenames.
# Now the text has been parsed into paragraphs, sentences and words by the default act
# of the PlaintextCorpusReader
newcorpus = PlaintextCorpusReader(corpusdir, '.*')
os.chdir(corpusdir)
#-----------------------------------------------END OF STEP 1-------------------------
```

In [7]:
```
#-----------------------------------------------STEP 2---------------------------------
#----------------------(a)----------------------#
#Function used as default argument in parser() function if it is not defined
def identity_preprocess(s):
    if(isinstance(s, str)):
        return s
    else: return "No string was given"
```

In [8]:
```
#----------------------(b)----------------------#
#Function to parse the text file given, line by line
def parser(path,preprocess = identity_preprocess):
    tokens = []
    for line in path.split('\n'):
        tokens+= preprocess(line)
    return tokens
```

In [9]:
```
#----------------------(c)----------------------#
import re
import string
#Tokenization step, a simple version which includes tokens of lowercase words
def tokenize(s):
    s_temp = s.strip().lower()
    s_temp = re.sub('[^A-Za-z\n\s]+', '', s_temp)
    s_temp = s_temp.replace('\n', ' ')
    s_temp = " ".join(s_temp.split())
    s_temp = s_temp.split(' ')
    s_temp[:] = [item for item in s_temp if item != '']
    return s_temp
```

In [10]:
```
#----------------------(d)---------------------#
#Comparing results of our built word tokenizer with a sentence that proves its functi
#with the results given by nltk's word tokenizers
#print(" A RoCk3!45.! Fell frOm334 \n ~. heaven ")
#1.The word_tokenize() function is a wrapper function that calls tokenize() on an
#instance of the TreebankWordTokenizer class. It is a simpler, regular-expression
#based tokenizer, which splits text on whitespace and punctuation:
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
print( tokenizer.tokenize(" A RoCk3!45.! Fell frOm334 \n ~. heaven "))

#2.WordPunctTokenizer splits all punctuations into separate tokens:
from nltk.tokenize import WordPunctTokenizer
word_punct_tokenizer = WordPunctTokenizer()
print(word_punct_tokenizer.tokenize(" A RoCk3!45.! Fell frOm334 \n ~. heaven "))

#3.Our created tokenizer
print(tokenize(" A RoCk3!45.! Fell     frOm334 \n ~. heaven "))
#-----------------------------------------------END OF STEP 2-------------------------
```

```
['A', 'RoCk3', '!', '45.', '!', 'Fell', 'frOm334', '~.', 'heaven']
['A', 'RoCk3', '!', '45', '.!', 'Fell', 'frOm334', '~.', 'heaven']
['a', 'rock', 'fell', 'from', 'heaven']
```

In [11]:
```
#-----------------------------------------------STEP 3---------------------------------
#Constructing word tokens and alphabet of the new corpus
#----------------------(a)---------------------#
corpus_preprocessed = newcorpus.raw(newcorpus.fileids()[1])
word_tokens = parser(corpus_preprocessed, tokenize)

print(word_tokens[:20])
```

```
['project', 'gutenbergs', 'the', 'adventures', 'of', 'sherlock', 'holmes', 'by', 'arthur', 'co
```

In [12]:
```
#----------------------(b)---------------------#
def tokenize_2(s):
    s_temp = s.strip()
    s_temp = " ".join(s_temp.split())
    s_temp = s_temp.split(' ')
    return s_temp
```

In [13]:
```
def parser_2(path, preprocess):
    alphabet = []
    for line in path.split('\n'):
        line = preprocess(line)
        for word in line:
            alphabet+= list(word)
```

```
        alphabet.append(' ')
        return set(alphabet)

    alphabet_tokens = sorted(parser_2(corpus_preprocessed,tokenize_2))
    print(alphabet_tokens)

    #-----------------------------------------------END OF STEP 3-------------------------
```

```
[' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', ',', '-', '.', '/', '0', '1', '2', '3'
```

In [14]:
```
    #-----------------------------------------------STEP 4--------------------------------
    filename = 'chars.syms'
    filename =  open(filename, 'w')
    result = []

    filename.write('<epsilon>'+ " " + str(0)+'\n')
    filename.write('<space>'+ "   " + str(1)+'\n')
    for symbol in range(2,len(alphabet_tokens)):
        line = alphabet_tokens[symbol] + "         " + str(symbol)+'\n'
        filename.write(line)

    filename.close()
    #-----------------------------------------------END OF STEP 4-------------------------
```

In [15]:
```
    #-----------------------------------------------STEP 5--------------------------------
    #HERE WE CREATE THE TRANDUCER I
    #----------------------(a)--------------------#
    filename = 'orth_I.txt'
    filename = open(filename,'w')

    alphabet="abcdefghijklmnopqrstuvwxyz"

    for letter in alphabet:
        filename.write("0 0 "+ letter +" "+ letter +" 0\n")
    filename.write("0")

    filename.close()
    !make -s orth_I
```

In [16]:
```
    #HERE WE CREATE THE TRANDUCER E
    filename = 'orth_E.txt'
    filename = open(filename,'w')
    alphabet="abcdefghijklmnopqrstuvwxyz"
    filename.write('0 1 <epsilon> <epsilon> 0'+'\n')
    for i in range(len(alphabet)):
        filename.write('0 1 <epsilon> '+alphabet[i]+' '+str(1)+'\n')#insertion
        filename.write('0 1 '+alphabet[i]+' <epsilon> '+str(1)+'\n')#deletion
```

```
            for j in range(len(alphabet)):
                if alphabet[i]!=alphabet[j]:
                    filename.write('0 1 '+alphabet[i]+' '+alphabet[j]+' '+str(1)+'\n')#Replac

        filename.write(str(1))
        filename.close()
        !make -s orth_E
        !make -s transducer
        #FINALLY WE CREATE THE TRANDUCER transducer = orth_I | orth_E | orth_I with the Makef
        #-----------------------------------------------END OF STEP 5-------------------------
```

In [17]:
```
        #-----------------------------------------------STEP 6-------------------------------
        #HERE WE CREATE THE ACCEPTOR/AUTOMATO used to accept all the words of our words_token
        #One state for each letter of every word-> States will be limited later when we will
        #commands of determinization, minimization, removal of <epsilon> transitions to our o
        #----------------------(a)---------------------#
        filename = 'orth_acceptor.txt'
        acceptor=open(filename, 'w')
        final_states = []
        state_count = 0

        acceptor.write('0 0 <epsilon> 0\n')
        for word in word_tokens:
            chars = list(word)
            if(len(chars) == 1):
                arg = ['0',' ',str(state_count+1),' ',chars[0],' ',chars[0],' 0','\n']
                arg = ''.join(arg)
                acceptor.write(arg)
                state_count += len(chars)
                final_states.append(str(state_count))
            else:
                arg = ['0',' ',str(state_count+1),' ',chars[0],' ',chars[0],' 0','\n']
                arg = ''.join(arg)
                acceptor.write(arg)
                for j in range(1,len(chars)):
                    arg = [str(j + state_count),' ',str(j+1 + state_count),' ',chars[j],' ',cl
                    arg = ''.join(arg)
                    acceptor.write(arg)
                state_count += len(chars)
                final_states.append(str(state_count))
        for i in range(0,len(final_states)):
            arg = [final_states[i],'\n']
            arg = ''.join(arg)
            acceptor.write(arg)
        acceptor.close()
        !make -s orth_acceptor
        !make -s orth_acceptor_processed
```

```
        #-----------------------------------------------END OF STEP 6-------------------------
```

In [18]:
```
        #-----------------------------------------------STEP 7--------------------------------
        #----------------------(a)---------------------#
        !make -s orthograph
        #----------------------(b)---------------------#
        filename = 'cit.txt'
        filename = open(filename, 'w')
        word = "cit"
        state = 0
        for letter in word:
            if letter!='\n':
                filename.write(str(state)+' '+str(state+1)+' '+letter+ '\n')
                state+=1
        filename.write(str(state)+'\n')

        filename.close()
        !make -s check_cit

        #-----------------------------------------------END OF STEP 7-------------------------
```

```
3       2       c       w       1
0
1       0       t       t
2       1       i       i
```

In [19]:
```
        #-----------------------------------------------STEP 8--------------------------------
        #----------------------(a)---------------------#
        from lib import *
        filename = 'spell_checker_test_set.txt'
        #We take 'spell_checker_test_set.txt', and we split to create 2 lists, the one with t
        #and the other with the list of the relevant wrong words. We chose randomly to ckeck
        filename = open(filename, 'r')
        lines = filename.readlines()[20:40]
        correct_words = []
        wrong_words =[]
        for line in lines:
            correct_words.append(line.split(':')[0])
            wrong_words.append((line.split(':')[1]).split())

        acceptor = []
        print(correct_words)
        print(wrong_words)
```

```
['further', 'monitoring', 'biscuits', 'available', 'separate', 'necessary', 'definition', 'rec
[['futher'], ['monitering'], ['biscits', 'biscutes', 'biscuts', 'bisquits', 'buiscits', 'buiscu
```

In [20]:
```
#We should create the dictionary based on the "chars.syms". The position in the dicti
#represents the index in the symbol
dictionary = 'chars.syms'
dictionary= open(dictionary,'r')
lines=dictionary.readlines()
dict=[0 for i in range(len(lines))]
for line in lines:
    matching = line.split()
    dict[int(matching[1])]=matching[0]
dictionary.close()

print(dict)
```

```
['<epsilon>', '<space>', '"', '#', '$', '%', '&', "'", '(', ')', '*', ',', '-', '.', '/', '0',
```

In [21]:
```
#Here in file OurResults, we will save the produced words
filename = 'OurResults.txt'
result = open(filename, 'w')
for i in range(len(wrong_words)):
    for word in wrong_words[i]:
        #-------------------------------------------------------------------------#
        #We truncate this file in order to make the other acceptors in the same file
        acceptor=open('word_acceptor.txt', 'w')
        state = 0
        for letter in word:
            if letter!='\n':
                acceptor.write(str(state)+' '+str(state+1)+' '+letter +'\n')

                state+=1
        acceptor.write(str(state)+'\n')
        acceptor.close()
            #-------------------------------------------------------------------------
        #We use the fst tool in order to create the acceptor for every word
        #The method of shortest path was ussed to find the best matches
        !make -s unique_word
        #-------------------------------------------------------------------------#
        #We write the result in a file in order to compare the best words later
        acceptor_shortest=open('Acceptor_Shortest.txt', 'r')
        lines=acceptor_shortest.readlines()
        temp_word=[]

        for j in range(2,len(lines)):
            chars = lines[j].split()
            if(len(chars) > 3):
                temp_word.append(chars[3])
        if(len(lines) > 1):
            chars = lines[0].split()
```

```
                    if(len(chars) > 3):
                        temp_word.append(chars[3])
                #-------------------------------------------------------------------------#
                print(word,end =' ')
                #Apparently, now in temp_word we have the produced word, which is going to be
                #cheked based on our dictionary created in the previous block.
                for letter in temp_word[1:(len(temp_word)-1)]:
                    if int(letter)!=0:
                        print(dict[int(letter)],end ='')
                        result.write(dict[int(letter)])
                print(' ',end = '')

                #-------------------------------------------------------------------------#
                #So for each word we save our result bh using this format:
                #|word orthograph| + |wrong_word| + |correct_word|
                print(correct_words[i])
                result.write(' '+word+' '+correct_words[i]+'\n')

        result.close()
```

```
futher faurtthehrer further
monitering  monitoring
biscits  biscuits
biscutes  biscuits
biscuts  biscuits
bisquits  biscuits
buiscits  biscuits
buiscuts  biscuits
avaible  available
seperate separate separate
neccesary  necessary
necesary necessssaary necessary
neccesary  necessary
necassary necessary necessary
necassery  necessary
neccasary  necessary
defenition  definition
receit receniptt receipt
receite receive receipt
reciet  receipt
recipt receipt receipt
remine  remind
remined remained remind
inetials initials initials
inistals  initials
initails  initials
initals initials initials
intials initials initials
```

```
magnificnet  magnificent
magificent magnificent magnificent
magnifcent magnificent magnificent
magnifecent magnificent magnificent
magnifiscant  magnificent
magnifisent magnificent magnificent
magnificant magnificent magnificent
annt aunt aunt
anut 0nut aunt
arnt auranttt aunt
intial  initial
ther t0thto00he0himynrheeerrr there
experances  experiences
biult  built
totaly total totally
undersand understand understand
undistand  understand
southen southern southern
definately definitely definitely
difinately  definitely
```

```
In [22]: #### HERE WE GONNA CHECK THE CORRECTNESS OF OUR ORTHOGRAPH
         corrected_words=0
         wrong_words=0
         no_matching_words=0
         result=open('OurResults.txt', 'r')
         words=result.readlines()
         for word in words:
             chars = word.split()
             if(len(chars) >2):
                 if(chars[0] == chars[2] and chars[1]!=chars[2]):
                     corrected_words+=1
                 else:
                     wrong_words +=1
             else:
                 no_matching_words+=1

         print('\nOur Orthograph gave the following results\n')
         print('Corrected Words ' + str(corrected_words))
         print('Wrong Words ' + str(wrong_words))
         print('There was no matching for '+ str(no_matching_words) + ' words')
         #----------------------------------------------END OF STEP 8-------------------------
```

```
Our Orthograph gave the following results

Corrected Words 15
```

```
Wrong Words 9
There was no matching for 24 words
```

In [27]:
```
#-----------------------------------------------STEP 9-------------------------------
#----------------------(a)---------------------#
# Initialize word2vec. Context is taken as the 2 previous and 2 next words
def parser3(path,preprocess = identity_preprocess):
    tokens = []
    for line in path.split('\n'):
        s_temp = preprocess(line)
        if(s_temp ==[]):continue
        tokens.append(s_temp)
    return tokens

sent_tokens = parser3(corpus_preprocessed, tokenize)
print(sent_tokens[:30])
```

```
[['project', 'gutenbergs', 'the', 'adventures', 'of', 'sherlock', 'holmes', 'by', 'arthur', 'c
```

In [28]:
```
#----------------------(b)---------------------#
from gensim import models
import numpy as np
import random

model = models.Word2Vec( sent_tokens,window=5, size=100, min_count = 2,workers=4)
model.train(sent_tokens, total_examples=len(sent_tokens), epochs=1000)

# get ordered vocabulary list
voc = model.wv.index2word

# get vector size
dim = model.vector_size

words_to_check = random.sample(voc, 10)
for word in words_to_check:
    sim = model.wv.most_similar(word,topn=2)
    print(word,sim)
```

```
leg [('whoever', 0.40393221378326416), ('fathom', 0.3576420545578003)]
presently [('whistle', 0.33856892585754395), ('country', 0.3132840394973755)]
murderous [('englishman', 0.5531414747238159), ('expression', 0.3790321946144104)]
mud [('flash', 0.4029349386692047), ('throws', 0.390768826007843)]
stepping [('trick', 0.46358785033226013), ('stared', 0.4237854778766632)]
lens [('pillow', 0.4028383791446686), ('creases', 0.3827398717403412)]
but [('and', 0.5507899522781372), ('that', 0.5200991034507751)]
fantastic [('misgivings', 0.3805195987224579), ('bride', 0.3748277425765991)]
```

```
staring [('gazing', 0.4954022765159607), ('drives', 0.40069448947906494)]
answering [('jest', 0.42209839820861816), ('tone', 0.35786348581314087)]
```

In [ ]: