Επεξεργασία Φωνής και Φυσικής Γλώσσας

1ο Εργαστήριο: Εισαγωγή στις γλωσσικές αναπαραστάσεις

ΣΧΟΛΗ: ΣΗΜΜΥ

Ονοματεπώνυμο	Αριθμός Μητρώου
Γιάννης Πιτόσκας	03115077
Αντώνης Παπαοικονόμου	03115140

Προπαρασκευαστικό Μέρος:

Περιγραφή: Σε αυτό το μέρος της εργαστηριακής άσκησης θα γίνει χρήση FSMs (Finite-State Machines) με σκοπό την επεξεργασία γλώσσας. Ουσιαστικά, θα δημιουργήσουμε έναν ορθογράφο με την βοήθεια απλών γλωσσικών μοντέλων και απλών μετασχηματισμών.

Εκτέλεση:

Βήμα 1: Κατασκευή corpus

α) Επιλέξαμε από το project Gutenberg το βιβλίο “Around-the-World-in-80-Days-by-Jules-Verne" και το κατεβάσαμε σε plain txt μορφή.

β) Θα χρησιμοποιήσουμε αυτά τα corpora για να εξάγουμε στατιστικά όταν χρειαστεί να κατασκευάσουμε το γλωσσικό μοντέλο. Ακόμη, καλύτερα μπορούμε να συνενώσουμε πολλά βιβλία δημιουργώντας ένα συνολικά μεγαλύτερο corpus. Με αυτόν τον τρόπο θα μπορούμε να κάνουμε περισσότερο train και κατ’ επέκταση να έχουμε ακριβέστερα αποτελέσματα στην έξοδο του ορθογράφου. Ακόμη, η ένωση βιβλίων διαφορετικής θεματολογίας μας παρέχει λέξεις από διαφορετικούς τομείς και έτσι καλύπτεται μεγαλύτερη γκάμα των υπαρχόντων πεδίων, σε αντίθεση με το ένα βιβλίο που θα επικεντρωνόταν στον δικό του τύπο θέματος (πχ ένα βιβλίο φυσικής δε θα περιέχει λέξεις που αφορούν την πολιτική). Επιπλέον, η ένωση βιβλίων διαφορετικού ύφους μπορεί να μας προσφέρει και απλό-καθημερινό λεξιλόγιο και ταυτόχρονα και πιο εκλεπτυσμένο λεξιλόγιο.

Βήμα 2: Προεπεξεργασία corpus

Σε αυτό το σημείο ξεκινάει η διαδικασία του preprocessing. Ουσιαστικά, στον κώδικα διαβάζουμε με μια συνάρτηση read_file ένα txt αρχείο η οποία είναι πολυμορφική και έχει δύο μορφές με την οποία μπορεί να κληθεί:

- Με όρισμα το path του txt αρχείου και μια preprocessing συνάρτηση
- Με όρισμα μόνο το path του txt αρχείου, όπου σε αυτήν την περίπτωση χρησιμοποιείται ως default preprocessing συνάρτηση η identity_preprocess η οποία δέχεται ένα string και επιστρέφει τον εαυτό του.

Στη συνέχεια δημιουργούμε μια συνάρτηση tokenize η οποία δέχεται ένα string και επιστρέφει μια λίστα από τις λέξεις του string σε lowercase. Ουσιαστικά, παίρνουμε το string και αυτό που τελικά κάνουμε είναι να τα κάνουμε όλα lowercase, να αφαιρούμε αριθμούς και σύμβολα κρατώντας μόνο τα γράμματα a-z, και να χωρίζουμε το string σε λέξης με βάση τα whitespaces (μετατρέπουμε και την αλλαγή γραμμής σε whitespace). Με αυτό το απλό tokenization θεωρούμε ουσιαστικά ότι τα tokens μας είναι οι υπάρχουσες lowercase λέξεις.

Βήμα 3: Κατασκευή λεξικού και αλφαβήτου

α) Αρχικά, δημιουργούμε μια λίστα με όλες τις διαφορετικές (unique) λέξεις (tokens) που περιέχονται στο corpus ορίζοντας έτσι ένα λεξικό για το corpus.

β) Ύστερα, δημιουργούμε μια λίστα με όλα τα διαφορετικά γράμματα που υπάρχουν στο corpus ορίζοντας έτσι το αλφάβητο του corpus.

chars.syms

```
<epsilon> 0
a 1
b 2
c 3
d 4
e 5
f 6
g 7
h 8
i 9
j 10
k 11
l 12
m 13
n 14
o 15
p 16
q 17
r 18
s 19
t 20
u 21
v 22
w 23
x 24
y 25
z 26
```

Βήμα 4: Δημιουργία συμβόλων εισόδου/εξόδου

Φτιάχνουμε μια συνάρτηση syms με την οποία πρακτικά αυτό που θέλουμε να πετύχουμε είναι να αντιστοιχίσουμε κάθε χαρακτήρα του αλφαβήτου καθώς και το <epsilon> σε ένα αύξων id που λειτουργεί ως ακέραιος index του αντίστοιχου χαρακτήρα. Ταξινομούμε το αλφάβητο μας ώστε να έχουμε τους χαρακτήρες σε μορφή $\{a, b, c, \dots, z\}$ με μήκος αλφαβήτου ≤ 26, ώστε να είναι κατά σύμβαση όπως είναι η σειρά των γραμμάτων στο πραγματικό αλφάβητο. Ουσιαστικά, η συνάρτησή μας παίρνει ως όρισμα τη λίστα με το αλφάβητο του corpus και στη συνέχεια ανοίγει ένα αρχείο char.syms στο οποίο έχουμε δύο στήλες, στην πρώτη γράφουμε τον χαρακτήρα και δίπλα τον αύξοντα ακέραιο index στον οποίο αντιστοιχεί θέτοντας ως index του <epsilon> το 0 όπως ζητείται στην εκφώνηση, και ύστερα αντιστοιχίζουμε κάθε γράμμα του αλφαβήτου στον κάθε φορά επόμενο ακέραιο αριθμό όπως φαίνεται δίπλα:

Βήμα 5: Κατασκευή μετατροπέων FST

Για τον ορθογράφο μας θα γίνει χρήση μετατροπέων οι οποίοι θα βασίζονται στην απόσταση Levensthein χρησιμοποιώντας 3 τύπους επεξεργασίας πάνω σε μία λέξη:

- Insert
- Delete
- Replace

Σε κάθε λογής επεξεργασία αντιστοιχεί και ένα κόστος. Σύμφωνα με τις οδηγίες της εκφώνησης θεωρούμε κόστος $w = 1$ για κάθε πιθανό edit.

α) Κατασκευάσαμε, λοιπόν, τον ζητούμενο μετατροπέα με μία κατάσταση που υλοποιεί την απόσταση Levensthein εφαρμόζοντας τις ακόλουθες αντιστοιχίσεις. Έστω $char1, char2 \in \{alphabet\} \cup \{< epsilon >\}$ $με$ $char1 \neq char2$ τότε οι αντιστοιχίσεις είναι οι εξής:

- $w(char1, char1) = 0$ $(no\ edit)$
- $w(char1, char2) = 1$ $αφού$:
    - $Insert = < epsilon >$ $σε$ $char1$ $με$ $char1 \neq < epsilon >$
    - $Delete = char1$ $σε$ $< epsilon >$ $με$ $char1 \neq < epsilon >$
    - $Replace = char1$ $σε$ $char2$ $με$ $char1, char2 \neq < epsilon >$

Στην περίπτωση μας οι αντιστοιχίσεις φαίνονται στο παρακάτω grid αντιστοίχισης:

From|To	<e>	*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*
<e>	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
a	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
b	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
c	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
d	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
e	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
g	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
h	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
i	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
j	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
k	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
l	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
m	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
n	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
o	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
p	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
q	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
r	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
s	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
t	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
u	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
v	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
w	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
y	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
z	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Έστω τώρα ότι δίνουμε στον μετατροπέα μας μια λέξη ως είσοδο, το shortest path με βάση την παραπάνω αντιστοίχιση θα ήταν να υπάρχει η λέξη αυτούσια και επομένως να μην κάνουμε κανένα edit, δηλαδή συνολικό βάρος $W = 0$. Η αμέσως επόμενη περίπτωση είναι να υπάρχει λέξη που απέχει από αυτήν της εισόδου μονάχα κατά ένα insert ή ένα delete ή ένα replace, δηλαδή συνολικό βάρος $W = 1$. Και συνεχίζουμε έτσι για $W = 2, 3, 4, \ldots$

β) Ωστόσο, αυτός ο τρόπος ανάθεσης των βαρών για κάθε edit είναι αρκετά αφελής καθώς θεωρούμε όλα τα edits ίσου βάρους πράγμα που σημαίνει ότι θεωρούμε όλα τα δυνατά edits ισοπίθανα. Αν είχαμε στη διάθεση μας ότι δεδομένα θέλαμε, θα υπολογίζαμε τα βάρη με τελείως διαφορετικό τρόπο. Είναι σημαντικό να λάβω υπόψη μου τη συχνότητα εμφάνισης κάθε χαρακτήρα στο λεξικό, έτσι ώστε να έχω εικόνα για την πιθανότητα εμφάνισης ενός χαρακτήρα του αλφαβήτου σε μια λέξη του λεξικού. Μια άποψη, επίσης, θα ήταν, αν παίρναμε για παράδειγμα τη μετάβαση από ‘g’ σε ‘a’ να υπολογίζαμε ποια είναι η πιθανότητα να πρέπει να αλλάξουμε ένα ‘g’ με ‘a’ που θα μπορούσε να παραπέμπει στην δεσμευμένη πιθανότητα να έχω ως έναν χαρακτήρα μιας λέξης το ‘a’ δεδομένου ότι δεν είναι ‘g’. Έτσι θα υπολογίζαμε στη συνέχεια το βάρος ως τον αρνητικό λογάριθμο της πιθανότητας αυτής (καθώς και της εκάστοτε πιθανότητας).

Βήμα 6: Κατασκευή αποδοχέα λεξικού

α) Σ' αυτό το βήμα θέλουμε να φτιάξουμε έναν αποδοχέα με μια αρχική κατάσταση που αποδέχεται μια λέξη όταν αυτή ανήκει στο λεξικό. Ουσιαστικά, η ιδέα είναι η εξής:

Φτιάχνουμε ένα κοινό file το οποίο περιέχει την περιγραφή για το εκάστοτε FST που αντιστοιχεί στην κάθε λέξη. Ουσιαστικά, πρόκειται για ένα FST με αποδεκτές καταστάσεις τα tokens του corpus. Ωστόσο, πρέπει να χρησιμοποιήσουμε τις τρεις συναρτήσεις που θα αναλυθούν στη συνέχεια, προκειμένου να αφαιρεθούν οι <epsilon> μεταβάσεις, να αποκτηθεί ο ντετερμινιστικός χαρακτήρας αποβάλλοντας μη-ντετερμινιστικές συμπεριφορές και να ελαχιστοποιηθεί ο αριθμός μεταβάσεων και καταστάσεών του.

β) Στη συνέχεια χρησιμοποιήσαμε τις δοθείσες συναρτήσεις fstrmepsilon, fstdeterminize, fstminimize για να βελτιστοποιήσουμε το μοντέλο.

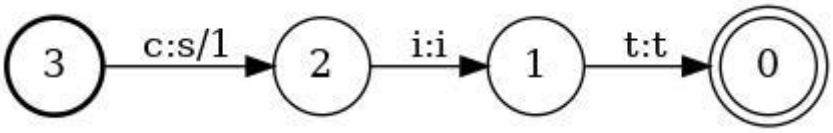
- Η fstrmepsilon ουσιαστικά παίρνει ένα FST και κατασκευάζει ένα ισοδύναμό του χωρίς input/output <epsilon> μεταβάσεις.
- Η fstdeterminize παίρνει ένα FST και κατασκευάζει ένα ντετερμινιστικό ισοδύναμό του στο οποίο, δηλαδή δεν υπάρχει κατάσταση από την οποία να μπορείς να μεταβείς με το ίδιο στοιχείο εισόδου σε παραπάνω από μία άλλες κατάστασεις.
- Η fstminimize παίρνει ένα FST και κατασκευάζει ένα ελαχιστοποιημένο ισοδύναμο του ελαχιστοποιώντας και τον αριθμό των καταστάσεων αλλά και τον αριθμό των μεταβάσεων.

Βήμα 7: Κατασκευή ορθογράφου

α) Κάνοντας χρήση της fstcompose συνθέτουμε τον acceptor που φτιάξαμε προηγουμένως με τον μετατροπέα Levenshtein transducer και προκύπτει το παρακάτω διάγραμμα:

(i) Όταν όλα τα edits είναι ισοβαρή άρα και ισοπίθανα σημαίνει πως δε θα λαμβάνεται καθόλου υπόψη η συχνότητα εμφάνισης των χαρακτήρων, ή αλλιώς η ύπαρξη διαφοροποιήσεων μεταξύ των βαρών ώστε να χρησιμοποιείται ουσιωδώς το κριτήριο δυναμικού προγραμματισμού για τις ελάχιστες δυνατές μετατροπές στην λέξη εισόδου. Για διαφορετικά, βάρη αλλάζει κατ' αρχάς τελείως η αναδρομική σχέση της απόστασης Levenshtein και η επίλυση της πλέον γίνεται απολύτως παραμετρική καθιστώντας την έτσι και πιο ευέλικτη, καθώς θα τείνουμε να κινηθούμε στα μικρότερα κόστη, δηλαδή στην μεγαλύτερη πιθανότητα.

(ii) Όταν δίνουμε μια λέξη ως είσοδο στον min edit spell checker αναμένουμε να δίνει ως έξοδο την λέξη που αντιστοιχεί στο μικρότερο βάρος από edits. Η ελάχιστη περίπτωση είναι το συνολικό βάρος να είναι μηδενικό, γεγονός που σημαίνει ότι η λέξη που δόθηκε στην είσοδο είναι αναγνωρίσιμη και σωστή επομένως δεν αλλάζει τίποτα. Αν η λέξη δεν είναι αποδεκτή, τότε επιστρέφει στην έξοδο την λέξη από το λεξικό η οποία απέχει το μικρότερο edit distance από την λέξη της εισόδου, δηλαδή αυτή με το μικρότερο συνολικό βάρος. Στην περίπτωση του ερωτήματος, λοιπόν, με είσοδο την λέξη "cit", αναμένουμε ο ορθογράφος μας να επιστρέψει κάτι του τύπου: "cut", "cat", "sit", "city", κλπ , δηλαδή λέξεις που έχουν edit distance = 1 σε σχέση με τη λέξη εισόδου, δεδομένου βέβαια ότι οι λέξεις αυτές υπάρχουν στο λεξικό μας. Δίνοντας λοιπόν είσοδο στον ορθογράφο μας τη λέξη "cit" παρατηρούμε ότι πράγματι μας επιστρέφει την λέξη "sit", μια λέξη από την οποία όντως απέχει edit distance = 1.



Βήμα 8: Αξιολόγηση ορθογράφου

Χρησιμοποιήσαμε για το evaluation τυχαία 20 από τις λέξεις του spell_checker_test_set:

https://raw.githubusercontent.com/georgepar/python-lab/master/spell_checker_test_set

Παρακάτω φαίνονται αντίστοιχα οι συμβολοσειρές που δόθηκαν ως είσοδος στον ορθογράφο, η αναμενόμενη έξοδος και η πραγματική έξοδος του ορθογράφου:

Input string	Expected Output string	Real Output string	Real == Expected
leval	level	level	✓
problam	problem	problem	✓
beetween	between	between	✓
aranged	arranged	arranged	✓
receit	receipt	recent	
biult	built	bill	
totaly	totally	total	
undersand	understand	understand	✓
southen	southern	southern	✓
fisited	visited	visited	✓
usefull	useful	useful	✓
recieve	receive	receive	✓
sorces	sources	forces	
muinets	minutes	mines	
acess	access	access	✓
extreamly	extremely	extremely	✓
experance	experience	experience	✓
cirtain	certain	curtain	
diffrent	different	different	✓
dirven	driven	driven	✓

Παρατηρούμε ότι αρκετές από τις εισόδους μας επέστρεψαν ως έξοδο την αναμενόμενη. Όσον αφορά τώρα τις εξόδους που ήταν διαφορετικές απο τις αναμενόμενες, με την υλοποίηση που έχουμε κάνει είναι λογικό, καθώς εμφανίζει μια από τις λέξεις με το μικρότερο edit distance στο λεξικό. Για παράδειγμα $Levenshtein\_distance(muinets, mines) = 2$, ενώ $Levenshtein\_distance(muinets, minutes) = 3$ (δεδομένων των βαρών κάθε edit να είναι σταθερό και ίσο με 1), επομένως προφανώς και θα κρατήσει τη λέξη mines ως output. Επίσης, όταν βρεί δύο λέξεις στο λεξικό με ίδιο Levenshtein distance θα επιλέξει μια από αυτές χωρίς κάποιο επιπλέον κριτήριο (όπως ορίζει η εσωτερική υλοποίηση της συνάρτησης fstshortestpath του OpenFst). Τρέχοντας το shell αρχείο spell_checker.sh και κάνοντας εκ νέου το evaluation με 20 άλλες τυχαίες λέξεις από το παραπάνω test set παρατηρούμε το παρακάτω ποσοστό επιτυχίας:

```
antonypap@Antonypap-Desktop:/mnt/c/Users/anton/Documents/Αντώνης/HMMY/ΡΟΗ_Σ/Επεξεργασία_Φωνής_και_Φυσικής_Γλώσσας/Lab_01$ ./Makefile1.sh
Percentage of correct predicted words: 65.0 %
```

Βήμα 9: Εξαγωγή αναπαραστάσεων word2vec

α) Δημιουργούμε τα tokenized sentences που θα χρησιμοποιηθούν στο word2vec και στο training.

β) Παίρνουμε $window = 5$ και $epochs = 1000$. Γενικά, όσον αφορά την αριθμό των epochs πρέπει να είμαστε ιδιαίτερα προσεκτικοί, καθώς χαμηλός αριθμός epochs ενδέχεται να δημιουργήσει underfitting, μεγάλος αριθμός ενδέχεται να δημιουργήσει overfitting, επιθυμούμε κάτι ενδιάμεσο για να έχουμε optimal. Γενικά, αυτό που πρέπει να συμβαίνει για να έχουμε optimal είναι η συνάρτηση απόφασης να είναι μια σχετικά απλή συνάρτηση, αλλά δεν μπορούμε να ξέρουμε τον αριθμό των epochs ώστε να έχουμε optimal λύση, αλλά ουσιαστικά εξαρτάται από το πόσο διαφορετικά είναι τα δεδομένα μας.

```
antonypap@Antonypap-Laptop:/mnt/c/Users/HP-PC/Documents/Αντώνης/ΗΜΜΥ/ΡΟΗ_Σ/Επεξεργασία_Φωνής_και_Φυσικής_Γλώσσας/Lab_01$ python3 wtv.py
/home/antonypap/.local/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issubdtype from `int` to `np.signedintege
r` is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
For word: hair
Most similar: black

For word: posted
Most similar: comply

For word: mouth
Most similar: rest

For word: bad
Most similar: weather

For word: robbery
Most similar: fully

For word: travelled
Most similar: sure

For word: share
Most similar: pay

For word: besides
Most similar: yes

For word: muttered
Most similar: confessed

For word: european
Most similar: celestial
```

γ) Παρατηρήσαμε ότι τα μεγαλύτερα παράθυρα τείνουν να καταγράφουν περισσότερες πληροφορίες σχετικά με το θέμα και τον τομέα που αφορά η λέξη, καθώς και ποιες άλλες λέξεις χρησιμοποιούνται σε σχετικές συζητήσεις καθώς θα μπορούσε να γίνει αντιληπτή και μια μεταφορική έννοια. Όταν το παράθυρο ήταν μικρότερο έτεινε να συλλάβει περισσότερα για την ίδια τη λέξη καθώς και για το ποιες λέξεις είναι παρόμοιες γι' αυτό είναι χρησιμότερο σε περίπτωση που αναζητούμε κοντινές σημασιολογικά λέξεις ή και συνώνυμες).

Για $window = 10$ και $epochs = 1000$:

```
antonypap@Antonypap-Laptop:/mnt/c/Users/HP-PC/Documents/Αντώνης/ΗΜΜΥ/ΡΟΗ_Σ/Επεξεργασία_Φωνής_και_Φυσικής_Γλώσσας/Lab_01$ python3 wtv.py
/home/antonypap/.local/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issubdtype from `int` to `np.signedintege
r` is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
For word: occurred
Most similar: hour

For word: whom
Most similar: meanwhile

For word: on
Most similar: in

For word: won
Most similar: sense

For word: november
Most similar: rd

For word: mistaken
Most similar: seriously

For word: archive
Most similar: literary

For word: place
Most similar: car

For word: replied
Most similar: responded

For word: everything
Most similar: foreseen
```

Παρατηρήσαμε σε αυτήν την περίπτωση πως αυξάνοντας τα epochs είχαμε κάπως καλύτερα σημασιολογικά αποτελέσματα, και αυτό ενδεχομένως να οφείλεται στο πόσο πολυποίκιλα είναι τα δεδομένα μας.

Για $window = 5$ και $epochs = 2000$:

```
antonypap@Antonypap-Laptop:/mnt/c/Users/HP PC/Documents/Αντωνης/ΜΜΥ/Pdf_s/Επεξεργασια_Φωνης_και_Φυσικης_Γλωσσας/Lab_03$ python3 wtv.py
/home/antonypap/.local/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issubdtype from `int` to `np.signedintege
r` is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
  if np.issubdtype(vec.dtype, np.int):
For word: everywhere
Most similar: smoking

For word: learned
Most similar: understood

For word: that
Most similar: what

For word: to
Most similar: would

For word: occupied
Most similar: determined

For word: freely
Most similar: thief

For word: brick
Most similar: soon

For word: since
Most similar: favoured

For word: ascertain
Most similar: judge

For word: breakfast
Most similar: something
```

Γενικά, να σημειώσουμε ότι δε μπορούμε να έχουμε πολλές προσδοκίες από το μοντέλο μας, καθώς το έχουμε εκπαιδεύσει μόνο πάνω σε ένα βιβλίο.

Μέρος 1: Ορθογράφος

Βήμα 10: Εξαγωγή στατιστικών

α) Κατασκευάσαμε ένα λεξικό που δέχεται ως keys μια λέξη του λεξιλογίου και ως value την πιθανότητα εμφάνισης του όπου ως πιθανότητα εμφάνισης μιας λέξης ορίσαμε το πλήθος εμφανίσεων της λέξης στο corpus προς το πλήθος όλων των λέξεων του corpus.

β) Κατασκευάσαμε ένα λεξικό που δέχεται ως keys ένα χαρακτήρα του αλφαβήτου μας και ως value την πιθανότητα εμφάνισης του όπου ως πιθανότητα εμφάνισης ενός χαρακτήρα ορίσαμε το πλήθος εμφανίσεων του χαρακτήρα στο corpus προς το πλήθος όλων των χαρακτήρων του corpus.

Βήμα 11: Κατασκευή μετατροπέων FST

Word Level:

Μέχρι προηγουμένως είχαμε σε κάθε edit βάρος ίσο με 1. Τώρα αλλάζουμε το βάρος των edits σε w κοινό για όλα τα edits βέβαια πάλι, όπου ορίζουμε ως w τη μέση τιμή των βαρών του word level που κατασκευάσαμε στο Βήμα 10α. Τώρα, η ερώτηση είναι τι ορίζουμε ως βάρος για μία λέξη. Ορίζουμε ως βάρος για μία λέξη, τον αρνητικό λογάριθμο (με βάση 2) της πιθανότητας εμφάνισης της, δηλαδή: $Weight(word) = -\log_2 \mathrm{P}(word)$. Υπολογίζουμε, λοιπόν, το άθροισμα όλων αυτών των βαρών και το διαιρούμε με το μέγεθος του λεξιλογίου μας και παίρνουμε το ζητούμενο w. Έπειτα, κατασκευάζουμε τον μετατροπέα μας όμοια με το Βήμα 5, με τη μόνη διαφορά ότι τώρα δεν έχουμε μοναδιαίο βάρος, αλλά βάρος w.

Character Level:

Επαναλαμβάνουμε για το unigram γλωσσικό μοντέλο του Βήματος 10β για να υπολογίσουμε ένα άλλο βάρος w κοινό πάλι για όλα τα edits. Δηλαδή, ορίζουμε ως w τη μέση τιμή των βαρών του character level που κατασκευάσαμε στο Βήματος 10β. Ορίζουμε, αντίστοιχα με προηγουμένως, ως βάρος για έναν χαρακτήρα, τον αρνητικό λογάριθμο (με βάση 2) της πιθανότητας εμφάνισης του, δηλαδή: $Weight(character) = -\log_2 \mathrm{P}(character)$. Υπολογίζουμε, λοιπόν, το άθροισμα όλων αυτών των βαρών και το διαιρούμε με το μέγεθος του αλφαβήτου μας και παίρνουμε το ζητούμενο w. Έπειτα, κατασκευάζουμε τον μετατροπέα μας όμοια με το Βήμα 5, με τη μόνη διαφορά ότι τώρα δεν έχουμε μοναδιαίο βάρος, αλλά βάρος w.

Ωστόσο, και οι δύο αυτοί τρόποι ανάθεσης των βαρών για κάθε edit είναι αρκετά αφελείς, καθώς θεωρούμε όλα τα edits ίσου βάρους πράγμα που σημαίνει ότι θεωρούμε όλα τα δυνατά edits ισοπίθανα. Ακόμη, αν όλες οι αλλαγές έχουν το ίδιο βάρος, όπως συμβαίνει και στις δύο αυτές περιπτώσεις, μια έξοδος που είχε βάρος c στον προηγούμενο μετατροπέα του Βήματος 5, σε καθέναν από τους δύο καινούριους απλά θα έχει c·w. Δεν εισάγουμε, δηλαδή, κάποια προτίμηση στο ποια γράμματα αλλάζουμε και με ποια πράξη. Αυτός, λοιπόν ο τρόπος φαίνεται ότι είναι ένας ακόμη αφελής τρόπος για να φέρεις τα βάρη των edits στην ίδια τάξη μεγέθους με τα βάρη του γλωσσικού μοντέλου. Για παράδειγμα, αν τα βάρη στο word level language model είναι σε μια πολύ μεγάλη κλίμακα και τα βάρη των edits σε μια πολύ μικρή κλίμακα, τότε απλά θα αγνοούταν ο Levenshtein και θα υπολογίζα την πιο συχνά εμφανιζόμενη λέξη.

Αν είχαμε στη διάθεση μας ότι δεδομένα θέλαμε, θα υπολογίζαμε τα βάρη με τελείως διαφορετικό τρόπο. Παρουσιάσαμε μία ιδέα στο Βήμα 5β. Μια άλλη ιδέα βέβαια θα ήταν να λαμβάναμε υπόψη την γλώσσα στην οποία καλούμαστε να χρησιμοποιήσουμε τον ορθογράφο για να εφαρμόσουμε κάποια φωνολογικά κριτήρια. Για παράδειγμα, σε μια γλώσσα που ξέρουμε ότι είναι πιο πιθανό να εμφανίζεται ένα φωνήεν μετά ή πριν από ένα σύμφωνο (όπως συμβαίνει συνήθως) θα μπορούσαμε να φτιάξουμε με αυτή τη λογική τα βάρη. Ακόμη, στην περίπτωση που καλούμασταν να κάνουμε ορθογραφική διόρθωση σε ένα κείμενο, το οποίο έχει σωστές λέξεις και λάθος γραμμένες λέξεις, θα μπορούσε να ληφθεί υπόψην και ο σημασιολογικός χαρακτήρας των λέξεων. Για παράδειγμα, μια λάθος γραμμένη λέξη θα μπορούσε να διορθωθεί με βάρη που θα έχουν κάποια αντιστοιχία με τον σημασιολογικό χαρακτήρα των γειτονικών της σωστών λέξεων ώστε να βρίσκονται, όσο είναι δυνατόν, στο ίδιο context.

Βήμα 12: Κατασκευή γλωσσικών μοντέλων

Word Level:

Μέχρι προηγουμένως είχαμε έναν αποδοχέα με μια αρχική κατάσταση που απλά αποδεχόταν μια λέξη έχοντας μηδενικό βάρος σε όλες τις ακμές. Αυτή τη φορά η αποδοχή μιας λέξης θα σχετίζεται με ένα βάρος, το οποίο θα είναι ίσο με τον αρνητικό λογάριθμο (με βάση 2) της πιθανότητας εμφάνισης της όπως την έχουμε ορίσει στο Βήμα 10α. Η αναπαράσταση στο FSA για μια λέξη word που αποτελείται από N χαρακτήρες θα είναι N+1 κορυφές και N ακμές όπου η πρώτη ακμή θα έχει βάρος το βάρος της λέξης word όπως το ορίσαμε στην προηγούμενη πρόταση και οι υπόλοιπες N-1 ακμές θα έχουν μηδενικό βάρος. Στην συνέχεια ακολούθουμε για αυτόν τον αποδοχέα την ίδια στρατηγική με τον Βήμα 6 και καλούμε τις συναρτήσεις fstrmepsilon, fstdeterminize, fstminimize για να βελτιστοποιήσουμε το μοντέλο.

Character Level:

Επαναλαμβάνουμε για το unigram γλωσσικό μοντέλο με λίγο διαφορετική λογική. Για την αποδοχή μιας λέξης word με N χαρακτήρες έχουμε ένα FSA με N+1 κορυφές και N ακμές, όπου τώρα για κάθε i η i-οστή ακμή έχει βάρος ίσο με τον αρνητικό λογάριθμο (με βάση 2) της πιθανότητας εμφάνισης του i-οστού χαρακτήρα όπως την έχουμε ορίσει στο Βήμα 10β. Το άθροισμα όλων αυτών των βαρών είναι το συνολικό βάρος που αντιστοιχεί στην αποδοχή αυτής της λέξης. Στην συνέχεια ακολούθουμε πάλι και για αυτόν τον αποδοχέα την ίδια στρατηγική με τον Βήμα 6 και καλούμε τις συναρτήσεις fstrmepsilon, fstdeterminize, fstminimize για να βελτιστοποιήσουμε το μοντέλο.

Βήμα 13: Κατασκευή ορθογράφων

Παίρνω τους αντίστοιχους μετατροπείς και αποδοχείς και κατασκευάζω τους δύο ορθογράφους που ζητούνται για το word level μοντέλο και unigram μοντέλο αντίστοιχα.

Ένα παράδειγμα αμφίσημης διόρθωσης για τους δύο ορθογράφους είναι για την λέξη $receit$. Το word level μοντέλο δίνει $receit \rightarrow receipt$ ενώ το unigram μοντέλο δίνει $receit \rightarrow recent$. Ουσιαστικά, αυτοί οι δύο ορθογράφοι σε σχέση με τον αρχικό που είχαμε κατασκευάσει ορίζουν στις περισσότερες περιπτώσεις μοναδική επιλογή λέξης όταν προκύπτουν δύο ή περισσότερες αποδεκτές λέξεις με ίδιο edit distance με το string εισόδου.

Βήμα 14: Αξιολόγιση των ορθογράφων

Τρέχοντας τα αντίστοιχα evaluation με αυτά της προπαρασκευής παίρνουμε τα εξής αποτελέσματα:

- Για το unigram μοντέλο:

```
antonypap@Antonypap-Desktop:/mnt/c/Users/anton/Documents/Αντώνης/HMMY/ΡΟΗ_Σ/Επεξεργασία_Φωνής_και_Φυσικής_Γλώσσας/Lab_01$ python3 evaluate_test_set.py "evaluation_set8.txt" "test_set.txt" "prediction_unigram.txt"
Percentage of correct predicted words: 14.444444444444445 %
```

- Για το word level μοντέλο:

```
antonypap@Antonypap-Desktop:/mnt/c/Users/anton/Documents/Αντώνης/HMMY/ΡΟΗ_Σ/Επεξεργασία_Φωνής_και_Φυσικής_Γλώσσας/Lab_01$ python3 evaluate_test_set.py "evaluation_set8.txt" "test_set.txt" "prediction_world_level.txt"
Percentage of correct predicted words: 49.629629629629626 %
```

Παρατηρούμε πως η απόδοση του unigram μοντέλου είναι αρκετά χαμηλή, μιας και υπολογίζει τα βάρη μόνο με βάση την αλλαγή γραμμάτων

Βήμα 15: Extra Credit

Κατασκευάσαμε ένα dictionary το οποίο παίρνει ως key κάποιο ζεύγος χαρακτήρων και επιστρέφει ως value την πιθανότητα εμφάνισης του συγκεκριμένου ζεύγους σε μια λέξη του corpus. Κατασκευάζουμε έναν μετατροπέα με βάρος w ίσο με την μέση τιμή των βαρών του bigram μοντέλου μας. Ως βάρος ένος ζεύγους χαρακτήρων ορίζουμε τον αρνητικό λογάριθμο (με βάση 2) της πιθανότητας εμφάνισης του.

Μέρος 2: Χρήση σημασιολογικών αναπαραστάσεων για ανάλυση συναισθήματος

Βήμα 16: Δεδομένα και προεπεξεργασία

Διαβάζουμε και προεπεξεργαζόμαστε τα δεδομένα με τέτοιο τρόπο ώστε να κάνω ένα tokenization όπως στο Βήμα 2 με τη μόνη διαφορά ότι τώρα κάθε σχόλιο αποτελεί ένα sample, ή αλλιώς ένα doc.

Βήμα 17: Κατασκευή BOW αναπαραστάσεων και ταξινόμηση

Σε αυτό το βήμα γίνεται χρήση Bag of Words για την αναπαράσταση προτάσεων. Μπορούμε είτε να εξάγουμε μη σταθμισμένες BOW αναπαραστάσεις υπολογίζοντας για κάθε πρόταση το μη-σταθμισμένο άθροισμα των one hot word encodings, ή να εξάγουμε σταθμισμένες BOW αναπαραστάσεις χρησιμοποιώντας βάρη TF-IDF.

α) Η χρήση των βαρών Tf-iDf έχει ως αποτέλεσμα λέξεις οι οποίες επαναλαμβάνονται συχνά και μπορεί να μην έχουν κάποια χρήσιμη πληροφορία να μεταδώσουν να αγνοούνται, ενώ αντίθετα λέξεις με πιο σπάνια συχνότητα εμφάνισης, τόσο σε ένα κείμενο όσο και στο σύνολο όλων των κειμένων, έχουν μεγαλύτερη σημασιολογική βαρύτητα.

Παίρνουμε τα train δεδομένα μας και φτιάχνουμε ένα δισδιάστατο πίνακα μέσω της fit_transform, όπου κάθε γραμμή αντιστοιχεί στο αντίστοιχο doc των train δεδομένων και κάθε στήλη αντιστοιχεί σε μια λέξη του vocabulary και έτσι, κάθε γραμμή του πίνακα αποτελεί ένα count vector που μου δείχνει ποιες λέξεις του vocabulary εμφανίζονται στο συγκεκριμένο σχόλιο/doc και πόσες φορές εμφανίζεται η κάθε μια. Φτιάχνουμε με χρήση της συνάρτησης transform και ένα δισδιάστατο πίνακα για τα test δεδομένα, προσαρμοσμένο στο shape του προηγούμενου. Φτιάχνουμε στην συνέχεια έναν ταξινομητή Logistic Regression και τον εκπαιδεύουμε δίνοντας του τον δισδιάστατο πίνακα των train δεδομένων που κατασκευάσαμε, μαζί με έναν μονοδιάστατο πίνακα που μας δείχνει αν το αντίστοιχο σχόλιο είναι θετικό η αρνητικό. Στη συνέχεια, τεστάρουμε τον ταξινομητή μας πάνω στα test δεδομένα και βλέπουμε σε τι ποσοστό έκανε σωστά predictions, και επομένως να δούμε πόσο καλό είναι το μοντέλο μας. Η παραπάνω λογική χρησιμοποιείται και για Count Vectorizer και για TF-IDF Vectorizer, απλώς στη μία περίπτωση κάνω fit_transform και transform στον Count Vectorizer που κατασκευάζω ενώ στην άλλη μία περίπτωση κάνω fit_transform και transform στον TF-IDF Vectorizer που κατασκευάζω.

δ) Για 25000 train δεδομένα σχολίων και 25000 test δεδομένα σχολίων το μοντέλο μας είχε τα εξής αποτελέσματα:

```
Test score for LR classifier with BoW with Count Vectorizer : 0.86696
Test score for LR classifier with BoW with Tf-iDf : 0.8828
```

Τα αποτελέσματα φαίνονται να είναι πολύ κοντινά. Ωστόσο, ο TF-IDF Vectorizer δίνει λίγο καλύτερα αποτελέσματα της τάξης του 2%.

Βήμα 18: Χρήση Word2Vec αναπαραστάσεων για ταξινόμηση

α) Για τις λέξεις εκτός του λεξικού (Out Of Vocabulary) έχουμε:

```
Percentage of OOV words: 97.86472981459738 %
```

Παρατηρούμε πως το ποσοστό είναι αρκετά μεγάλο, κάτι το οποίο είναι αναμενόμενο μιας και το μοντέλο μας έχει εκπαιδευτεί στο λεξιλόγιο ενός και μόνου βιβλίου.

β) Θεωρώντας τώρα την αναπαράσταση μιας OOV λέξης το μηδενικό διάνυσμα, δημιουργούμε το NBOW και μετά την «εκπαίδευση» του LR ταξινομητή έχουμε:

```
Test score for LR classifier with NBoW : 0.54256
Test score for LR classifier with NBoW with Tf-iDf : 0.53916
```

γ, δ) Μετά την φόρτωση του προεκπαιδευμένου μοντέλου από τα Google News με gensim και για τις λέξεις του βήματος 9α έχουμε:

```
For word: made
Most similar (in text): found
Most similar (Google): making

For word: west
Most similar (in text): long
Most similar (Google): east

For word: visible
Most similar (in text): door
Most similar (Google): visable

For word: share
Most similar (in text): told
Most similar (Google): Share

For word: passed
Most similar (in text): through
Most similar (Google): Passed

For word: whose
Most similar (in text): without
Most similar (Google): His

For word: itself
Most similar (in text): six
Most similar (Google): its

For word: quite
Most similar (in text): should
Most similar (Google): very

For word: expense
Most similar (in text): again
Most similar (Google): expenses

For word: knows
Most similar (in text): few
Most similar (Google): thinks
```

Όπως ήταν προφανές, το pretrained μοντέλο των Google News βγάζει πολύ καλύτερα και λογικά ως προς το context αποτελέσματα απ' ότι το δικό μας.

ε) Χρησιμοποιώντας το vocabulary του μοντέλου των Google News με τον περιορισμό $NUM\_W2V\_TO\_LOAD = 1000000$ εκπαιδεύουμε έναν Logistic Regression ταξινομητή και έχουμε το παρακάτω αποτέλεσμα για $test\ corpus = 5000$:

```
Test score with Word2Vec :  0.841
```

στ, ζ) Χρησιμοποιώντας τώρα σαν αναπαράσταση το σταθμισμένο άθροισμα των w2v με τα βάρη Tf-iDf των λέξεων, όπου το Tf-iDf για κάθε λέξη υπολογίζεται από την σχέση:

$$Tf - iDf = Tf * iDf = \frac{\#\ Εμφάνισης\ λέξης\ στο\ σχόλιο}{\#\ Συνολικών\ λέξεων\ στο\ σχόλιο} * \log\left(\frac{\#\ Σχολίων}{\#\ Σχολίων\ στα\ οποία\ εμφανίζεται\ η\ λέξη}\right)$$

Όπως βλέπουμε, ο LR ταξινομητής με χρήση NBOW με βάρη Tf-iDf είναι κατά πολύ λίγο καλύτερος αυτού με τα απλά NBOW, σε βαθμό που δεν μπορεί να θεωρηθεί στατιστικά συγκρίσιμο ($\approx 1\%$).

Βήμα 19: Extra credit

α) Χρησιμοποιώντας αυτή την φορά τους ταξινομητές Support Vector Machine και K-Nearest Neighbours (για τους 5 κοντινότερους) έχουμε τα εξής αποτελέσματα:

```
Test score for SVC classifier with BoW with Tf-iDf : 0.65832
Test score for KNN classifier ( 5 neighbors ) with BoW with Tf-iDf : 0.66188
```