Επεξεργασία Φωνής και Φυσικής Γλώσσας

1ο Προπαρασκευαστικό Εργαστήριο
Μέρος πρώτο: Ορθογράφος

ΣΧΟΛΗ: ΣΗΜΜΥ

Ονοματεπώνυμο	Αριθμός Μητρώου
Γιάννης Πιτόσκας	03115077
Αντώνης Παπαοικονόμου	03115132

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.

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, … , 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.txt 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:

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 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 ∈ {alphabet}∪{<epsilon>} με char1 ≠ char2 τότε οι αντιστοιχίσεις είναι οι εξής:

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

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

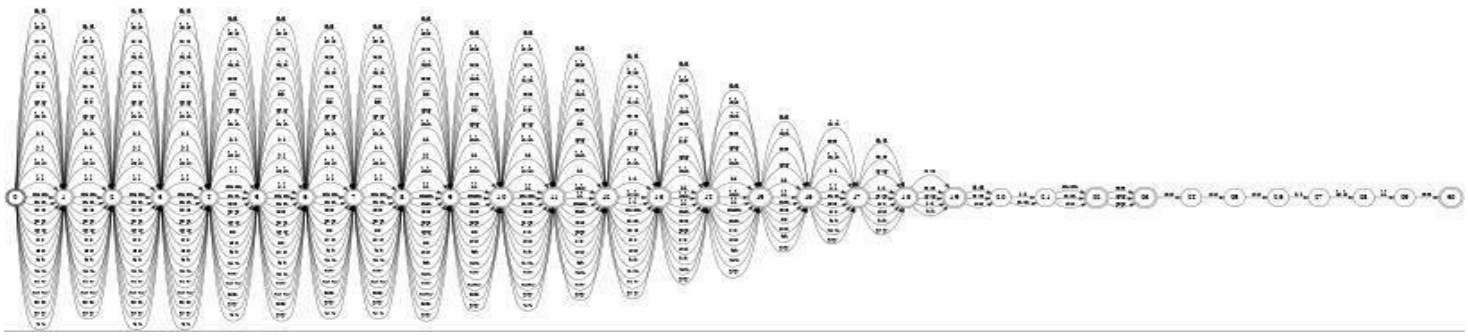
α) Σ' αυτό το βήμα θέλουμε να φτιάξουμε έναν αποδοχέα με μια αρχική κατάσταση που αποδέχεται μια λέξη όταν αυτή ανήκει στο λεξικό. Ουσιαστικά, η ιδέα είναι η εξής:

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.

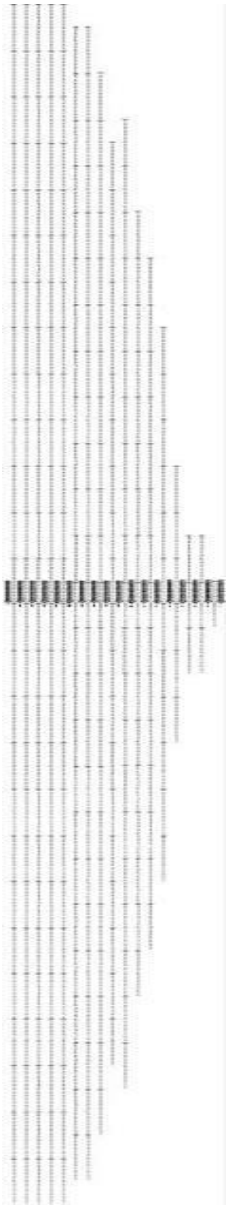
- Η fstrmepsilon ουσιαστικά παίρνει ένα FST και κατασκευάζει ένα ισοδύναμό του χωρίς input/output <epsilon> μεταβάσεις.
- Η fstdeterminize παίρνει ένα FST και κατασκευάζει ένα ντετερμινιστικό ισοδύναμό του στο οποίο, δηλαδή δεν υπάρχει κατάσταση από την οποία να μπορείς να μεταβείς με το ίδιο στοιχείο 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.

Το τελικό αποτέλεσμα είναι το ακόλουθο:



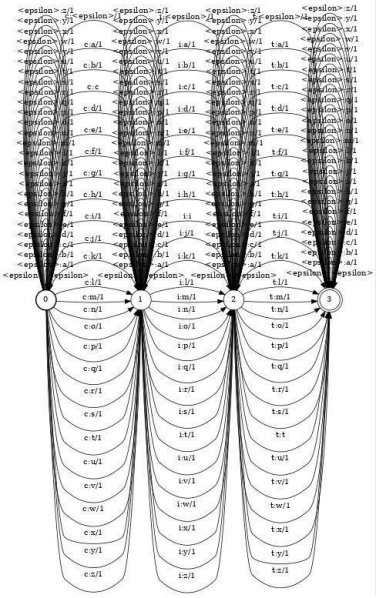
Βήμα 7: Κατασκευή ορθογράφου

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.

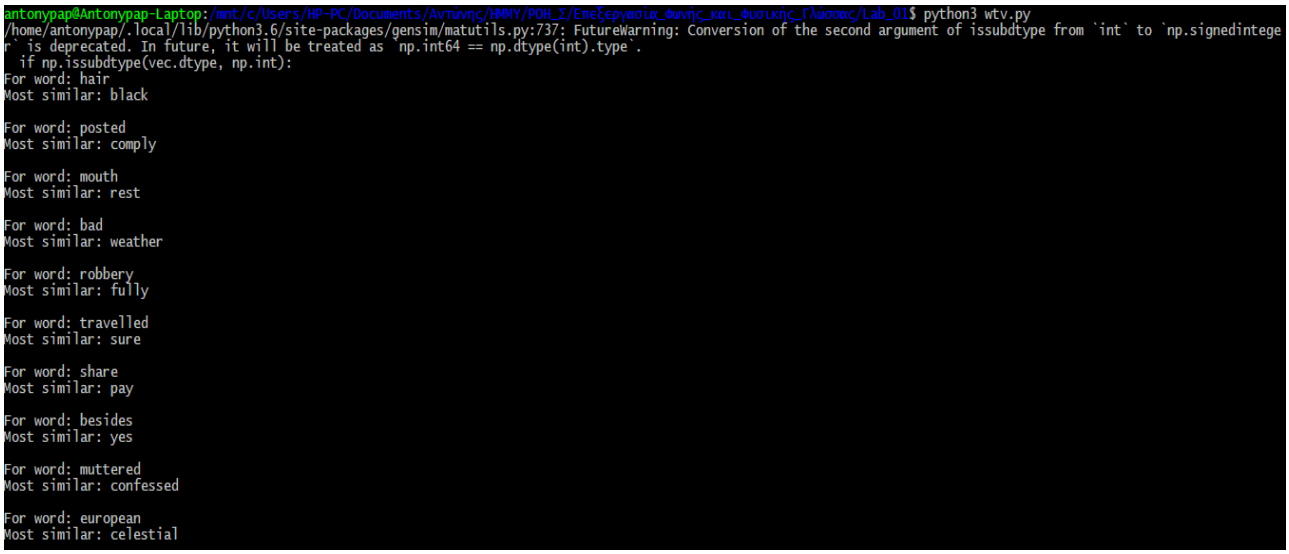
(ii)



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.

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/Αντωνης/ΗΜΜΥ/ΡΟΗ_5/Επεξεργασια_φωνης_και_φυσικης_γλωσσας/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
```

οφείλεται στο πόσο πολυποίκιλα είναι τα δεδομένα μας.

Για window = 5 και epochs = 2000:

```
antonypap@Antonypap-Laptop:/mnt/c/Users/HP-PC/Documents/Αντωνης/ΗΜΜΥ/ΡΟΗ_5/Επεξεργασια_φωνης_και_φυσικης_γλωσσας/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: 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.