Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Ροή Σ: Επεξεργασία Φωνής και Φυσικής Γλώσσας

(10ο Εξάμηνο)

Εργαστηριακή Άσκηση 1

Δημήτρης Μπακάλης 03118163

Μέρος 1

Εισαγωγή:

Στο συγκεκριμένο μέρος θα επιχειρήσουμε να υλοποιήσουμε έναν ορθογράφο αξιοποιώντας το openfst. Πιο συγκεκριμένα, θα δημιουργήσουμε το vocabulary μας μέσω βιβλίων από το project Gutenberg και θα δημιουργήσουμε, αρχικά, έναν Levensthein transducer και έναν acceptor. Η σύνθεση αυτών θα αποτελεί τον πρώτο (και πιο naive) ορθογράφο, που θα κατασκευάσουμε στο πλαίσιο της συγκεκριμένης άσκησης. Στη συνέχεια, θα επιχειρήσουμε να βελτιστοποιήσουμε τον παραπάνω ορθογράφο εισάγοντας γλωσσική πληροφορία. Τέλος, θα πραγματοποιήσουμε το evaluation του μοντέλου μας σε ένα σύνολο από ανορθόγραφες λέξεις, για να δούμε την επίδοσή του.

Βήμα 1:

α) Το preprocessing που εφαρμόζεται στο κείμενο μέσω του fetch_gutenberg.py αποτελείται από τα ακόλουθα βήματα:

- Αφαιρούμε τα επιπλέον κενά δεξιά και αριστερά από τις προτάσεις του κειμένου (strip).
- Μετατρέπουμε όλους τους χαρακτήρες σε πεζούς, ώστε tokens, όπως cat και Cat να αντιστοιχούν στην ίδια λέξη.
- Μετατρέπουμε τα contractions σε κανονικές λέξεις (πχ don't => do not).
- Δύο ή περισσότερα διαδοχικά κενά μεταξύ λέξεων μετατρέπονται σε ένα.
- Τα μόνα σύμβολα που κρατάμε είναι οι πεζοί λατινικοί χαρακτήρες.

Παρόλο που από το συγκεκριμένο κείμενο καταλήγουμε με μόνο λέξεις, γεγονός που καθιστά την υπόλοιπη επεξεργασία των δεδομένων πιο εύκολη, τα σημεία στίξης, αν και αφαιρούνται, είναι πολλές φορές χρήσιμα. Σε περιπτώσεις, όπως το να βρούμε το context μίας λέξης (οι προτάσεις διαχωρίζονται με '.') ή σε κάποιο πρόβλημα για sentiment analysis (πχ τα '!', '…' κλπ), τα σημεία στίξης είναι αρκετά βοηθητικά. Και οι δύο αυτές περιπτώσεις εξετάζονται στο επόμενο μέρος της άσκησης.

β) Εκτός του μεγαλύτερου όγκου δεδομένων, η επέκταση του corpus με περισσότερα βιβλία οδηγεί στα παρακάτω πλεονεκτήματα:

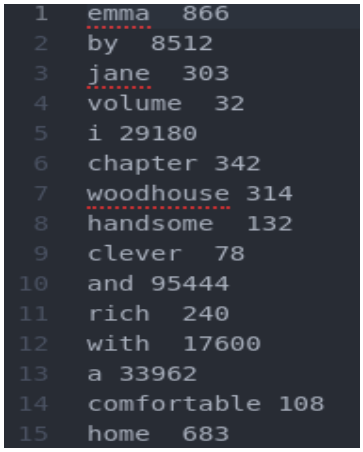
1. Μεγαλύτερο vocabulary.

2. Οι πιθανότερες λέξεις δεν θα είναι τόσο biased βάσει ενός μόνο βιβλίου (όπως προκύπτει αργότερα με χρήση του W για την εξαγωγή των πιθανοτήτων κάθε λέξης πχ leval γίνεται evil αντί για level).

Βήμα 2:

Σε αυτό το βήμα δημιουργούμε το vocabulary που θα χρησιμοποιήσουμε και το αποθηκεύουμε στο αρχείο vocab/words.vocab.txt (λέξεις με συχνότητες εμφάνισης). Η διαδικασία της δημιουργίας αυτού του αρχείου περιέχεται στη συνάρτηση make_corpus_lexicon του script corpus_lexicon_construction.py.

words.vocab.txt:

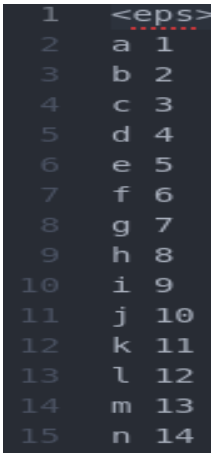


Αξίζει να σημειωθεί πως κατά τη δημιουργία του vocabulary φιλτράρουμε τις λέξεις που εμφανίζονται λιγότερες από 5 φορές. Αυτό γίνεται, διότι λέξεις με τόσο μικρή συχνότητα είναι πιθανώς ανορθόγραφες ή είναι τόσο σπάνιες που συμβάλλουν αρνητικά στην απόδοση του ορθογράφου (κάτι σαν overfitting). Επιπλέον, με αυτόν τον τρόπο μειώνουμε και τη υπολογιστική πολυπλοκότητα του μοντέλου.

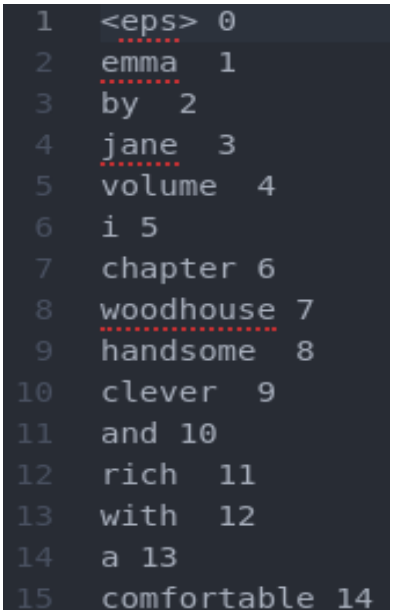
Βήμα 3:

Ανατρέχοντας όλες τους χρησιμοποιούμενους χαρακτήρες (<eps> + a-z) και όλες τις λέξεις του vocabulary τα ζητούμενα αρχεία λαμβάνουν την παρακάτω μορφή (ο κώδικας περιέχεται στη συνάρτηση mk_chars_words_indices του script corpus_lexicon_construction.py):

chars.syms:



words.syms:



Σημείωση: Παρατηρούμε πως το ε (<eps>) υπάρχει τόσο ως χαρακτήρας, όσο και ως λέξη, καθώς χρησιμοποιείται ως έξοδος και του transducer και του acceptor, όπως θα δούμε και παρακάτω.

Βήμα 4:

Στη συνέχεια, θα υλοποιήσουμε τον Levensthein transducer (ο κώδικας περιέχεται στη συνάρτηση mk_transducer του script corpus_lexicon_construction.py). Όπως αναφέρεται και στην εκφώνηση τα βάρη θα είναι τα ακόλουθα:

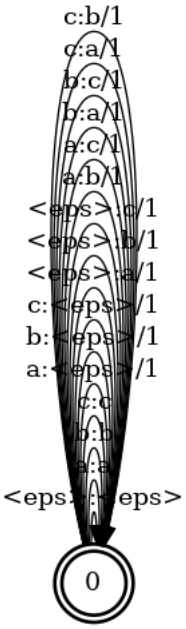
- No edit: 0
- Deletion: 1
- Insertion: 1
- Substitution: 1

Επομένως, θα δημιουργήσουμε ένα αρχείο (L.fst), το οποίο θα αποτελείται από μία κατάσταση, η οποία με ίδια είσοδο και έξοδο έχει βάρος 0, με είσοδο οποιονδήποτε χαρακτήρα (εκτός του ε) και έξοδο το ε έχει βάρος 1, με είσοδο το ε και έξοδο οποιονδήποτε χαρακτήρα (εκτός του ε) έχει βάρος 1 και με είσοδο οποιονδήποτε χαρακτήρα (εκτός του ε) και έξοδο οποιονδήποτε διαφορετικό χαρακτήρα (εκτός του ε) έχει βάρος 1.

L.fst:

```
0 0 <eps> <eps> 0
0 0 a a 0
0 0 b b 0
0 0 c c 0
0 0 d d 0
0 0 e e 0
0 0 f f 0
0 0 g g 0
0 0 h h 0
0 0 i i 0
0 0 j j 0
0 0 k k 0
0 0 l l 0
0 0 m m 0
0 0 n n 0
```

Χρησιμοποιώντας την fstdraw (για τους χαρακτήρες <eps>, a, b, c) προκύπτει ο παρακάτω transducer:



Προφανώς, υπάρχουν και άλλα πιθανά edits, τα οποία θα μπορούσαμε να συμπεριλάβουμε εκτός των 3 που υπάρχουν ήδη. Παραδείγματος χάρη το swap θα ήταν μία καλή επιλογή για edit, καθώς θα βελτιστοποιούσε διορθώσεις όπως η ακόλουθη (dirven -> driven, ενώ βάσει του ορθογράφου γίνεται given, λόγω μικρότερου κόστους). Επιπλέον, αν και στη συγκεκριμένη άσκηση δεν υπάρχει τέτοιο παράδειγμα, ένα σύνηθες λάθος είναι να έχει ξεχαστεί κάποιο κενό ανάμεσα σε δύο λέξεις. Για αυτό τον λόγο ένα splitting θα ήταν, επίσης, χρήσιμο (πχ spellchecker -> spell checker).

Πέρα από τα είδη των edits, θα μπορούσαμε να βελτιστοποιήσουμε και τα βάρη τους με βάση τα πιο συνήθη λάθη. Για παράδειγμα, κοντινά πλήκτρα στο πληκτρολόγιο παρουσιάζουν υψηλή πιθανότητα για substitution, οπότε θα έπρεπε να έχουν μικρότερο βάρος. Επίσης, πολλά φωνήεντα γίνονται πολλές φορές misspelled, επειδή με κάποιο άλλο φωνήεν η λέξη θα είχε παρόμοια προφορά.

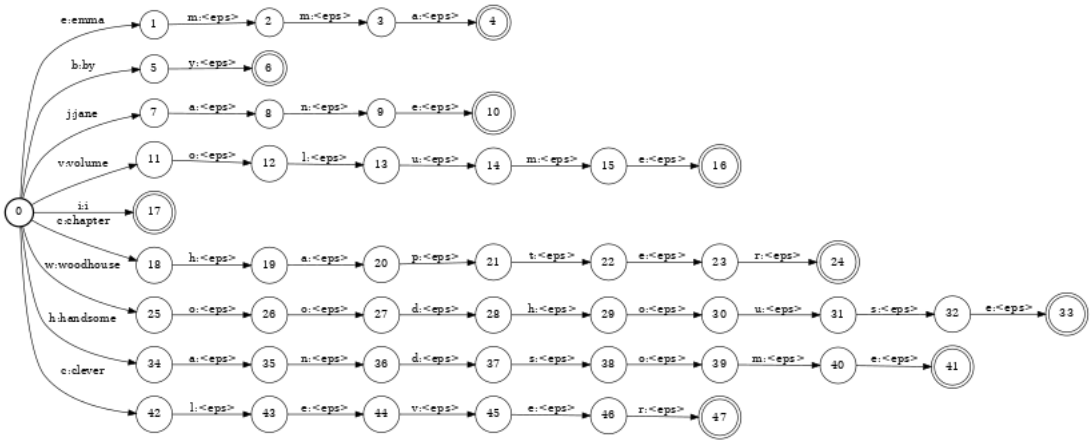
Βήμα 5:

Σε αυτό το βήμα θα υλοποιήσουμε τον acceptor (ο κώδικας περιέχεται στη συνάρτηση mk_acceptor του script corpus_lexicon_construction.py). Πιο συγκεκριμένα, θα δημιουργήσουμε ένα αυτόματο, το οποίο θα αποδέχεται κάθε λέξη του vocabulary μας. Αποτελείται από την αρχική κατάσταση 0, τελικές καταστάσεις ισάριθμες με το πλήθος του vocabulary και μερικές ενδιάμεσες καταστάσεις μέχρι να δημιουργηθεί κάποια λέξη. Εφόσον, απλά αποδεχόμαστε τις υπάρχουσες λέξεις, όλα τα βάρη είναι 0.

V.fst:

```
0 1 e emma 0
1 2 m <eps> 0
2 3 m <eps> 0
3 4 a <eps> 0
4
0 5 b by 0
5 6 y <eps> 0
6
0 7 j jane 0
7 8 a <eps> 0
8 9 n <eps> 0
9 10 e <eps> 0
10
0 11 v volume 0
11 12 o <eps> 0
12 13 l <eps> 0
13 14 u <eps> 0
14 15 m <eps> 0
15 16 e <eps> 0
16
```

Χρησιμοποιώντας την fstdraw (για τις πρώτες 10 λέξεις) προκύπτει ο παρακάτω acceptor:



Αξίζει να σημειωθεί πως, προκειμένου να βελτιστοποιήσουμε το μοντέλο μας χρησιμοποιούμε τις fstrmepsilon, fstdeterminize, fstminimize:

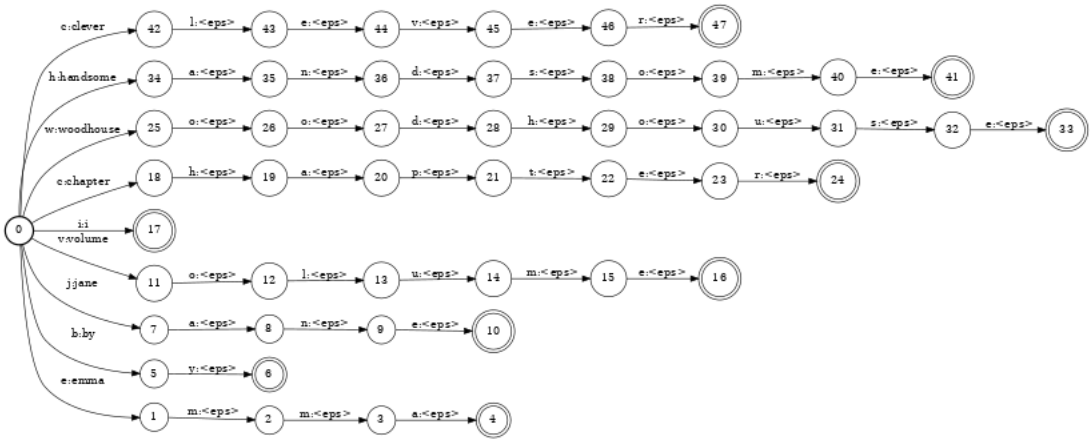
- **fstrmepsilon**: Αφαιρούνται οι ε-μεταβάσεις του acceptor.

- **fstdeterminize**: Ο acceptor γίνεται ντετερμινιστικός (καμία κατάσταση δεν έχει πάνω από μία μετάβαση με ίδια είσοδο).

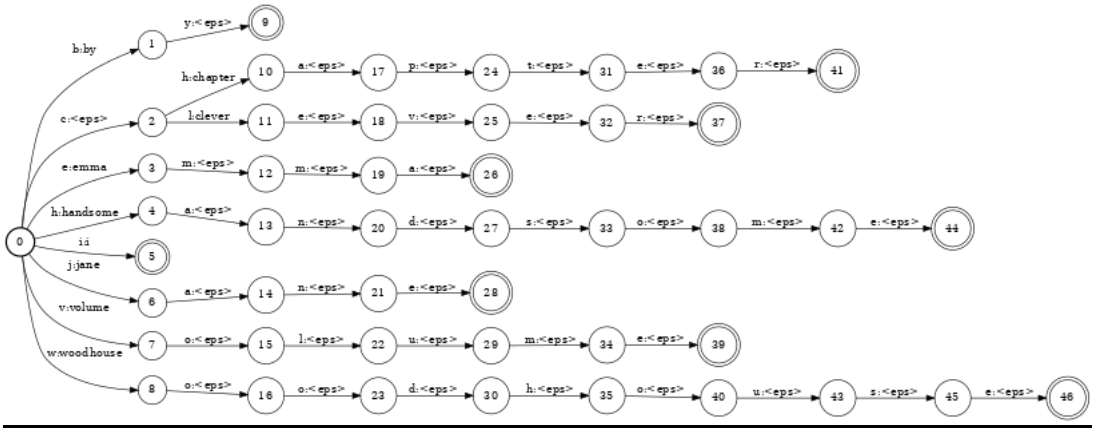
- **fstminimize**: Ο acceptor βελτιστοποιείται, ώστε να έχει τον ελάχιστο αριθμό καταστάσεων (εφαρμόζεται μόνο σε ντετερμινιστικά αυτόματα).

Οπτικοποιώντας τον acceptor μετά από κάθε μία από αυτές τις αλλαγές, προκύπτουν τα παρακάτω αποτελέσματα:

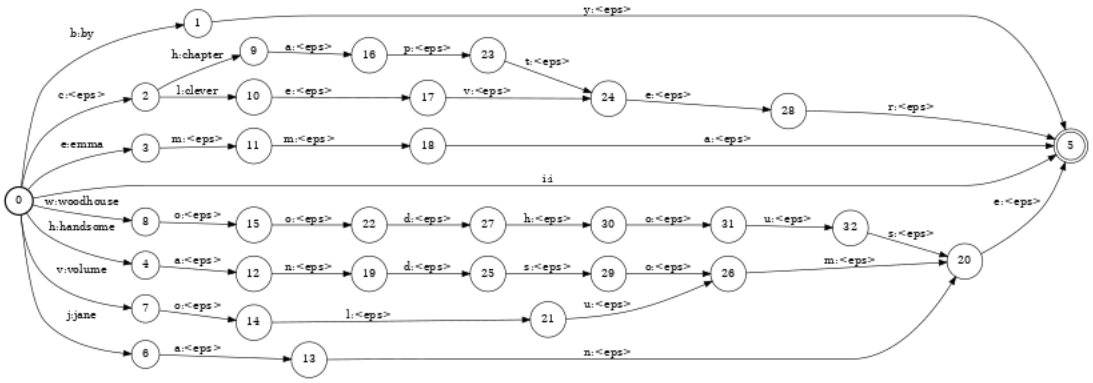
<u>Μετά την fstrmepsilon (δεν υπάρχουν ε-μεταβάσεις, οπότε δεν υπάρχει κάποια διαφοροποίηση):</u>



Μετά την fstdeterminize (δύο λέξεις αρχίζουν από c, οπότε αλλάζει το fst):



Μετά την fstminimize (υπάρχουν καταστάσεις με ίδια είσοδο-έξοδο, οπότε η ελαχιστοποίηση των καταστάσεων επιφέρει κάποιες αλλαγές):



Σημείωση: Οι παραπάνω βελτιστοποιήσεις, και ειδικά, η fstdeterminize οδήγησαν σε μεγάλη μείωση της πολυπλοκότητας του ορθογράφου, καθώς ο αριθμός των ακμών και των καταστάσεων ελαχιστοποιήθηκε, με αποτέλεσμα ο αλγόριθμος shortest path (όπως θα δούμε και παρακάτω), για την πρόβλεψη της ορθής λέξης, να τρέχει βέλτιστα.

Βήμα 6:

Μέσω της fstcompose θα συνθέσουμε τον Levensthein transducer L και τον acceptor V, ώστε να δημιουργήσουμε τον τελικό ορθογράφο. Πριν, όμως πραγματοποιήσουμε αυτή τη σύνθεση θα καλέσουμε την fstarcsort, τόσο για την έξοδο του transducer, όσο και για την είσοδο του acceptor, για να βεβαιωθούμε ότι ταιριάζουν (Για να γίνει όλη η διαδικασία (compilations, optimizations, arcsorts, compose), αρκεί να καλέσουμε το compile_compose.sh με arguments τα L V, δηλαδή **bash compile_compose.sh L V**). Ο ορθογράφος

δημιουργήθηκε, τόσο με ισοβαρή edits (=1), όσο και με τα edits deletion = insertion = 1 , substitution = 1.5, αλλά για τα επόμενα παραδείγματα θα χρησιμοποιηθεί η πρώτη υλοποίηση με τα ισοβαρή edits.

Για τα παραδείγματα των cit, cwt, με δεδομένο ότι έχουμε τον ορθογράφο με ισοβαρή edits έχουμε τα ακόλουθα πιθανά edits για τις δύο λέξεις:

cit:

- **Insertion:** city, cite
- **Deletion:** it
- **Substitution:** cat, cut, fit, hit, kit, sit κλπ

cwt:

- **Insertion:** -
- **Deletion:** -
- **Substitution:** cat, cut

Βήμα 7:

Στη συνέχεια, θα προχωρήσουμε σε μία πρώτη δοκιμή του ορθογράφου με βάση το αρχείο spell_test.txt. Θα τεστάρουμε τον ορθογράφο στις πρώτες 20 ανορθόγραφες λέξεις του αρχείου (μία λανθασμένη λέξη από τις 20 πρώτες διαφορετικές λέξεις). Για να πραγματοποιήσουμε αυτή τη δοκιμή αρκεί να χρησιμοποιήσουμε το script 20_words_test.py με argument τον ορθογράφο που θέλουμε να χρησιμοποιήσουμε (στη συγκεκριμένη περίπτωση **python3 scripts/20_words_test.py LV**).

Αυτό το script εκτελεί ένα επιμέρους script (predict.sh), το οποίο είναι κι αυτό που πραγματοποιεί την πρόβλεψη για τη διόρθωση κάθε λέξης. Πιο συγκεκριμένα, μέσω του script mkfstinput.py, μία λέξη γίνεται ένα αυτόματο (κατά αναλογία με τον acceptor), το οποίο, γίνεται αργότερα compose με τον spell checker. Αυτή είναι η διαδικασία, για να δώσουμε στον ορθογράφο ένα input. Επομένως, μετά το πέρας αυτής της σύνθεσης, εκτελούμε τον αλγόριθμο shortest path, για να βρούμε την ορθή λέξη με το min edit και καλούμε τις fstrmepsilon, fsttopsort, για να αφαιρέσουμε τις ε-μεταβάσεις και να ταξινομήσουμε τα αποτελέσματα σε φθίνουσα σειρά, αντίστοιχα. Τέλος, μέσω της fstprint και της cut λαμβάνουμε τη λέξη με το min edit, την οποία και εμφανίζουμε μετά από μία μικρή επεξεργασία (διαγραφή <eps>, '\n').

Παρακάτω φαίνονται τα αποτελέσματα για τις 20 λέξεις:

```
Misspelled word: contenpted | Correct word: contented | Predicted word: content
ed
Misspelled word: begining | Correct word: beginning | Predicted word: beginning
Misspelled word: problam | Correct word: problem | Predicted word: problem
Misspelled word: dirven | Correct word: driven | Predicted word: given
Misspelled word: exstacy | Correct word: ecstasy | Predicted word: ecstasy
Misspelled word: guic | Correct word: juice | Predicted word: guil
Misspelled word: localy | Correct word: locally | Predicted word: local
Misspelled word: compair | Correct word: compare | Predicted word: compare
Misspelled word: pronounciation | Correct word: pronunciation | Predicted word:
 provocation
Misspelled word: transportibility | Correct word: transportability | Predicted
word: respectability
Misspelled word: miniscule | Correct word: minuscule | Predicted word: ridicule
Misspelled word: independant | Correct word: independent | Predicted word: inde
pendent
Misspelled word: aranged | Correct word: arranged | Predicted word: ranged
Misspelled word: poartry | Correct word: poetry | Predicted word: partly
Misspelled word: leval | Correct word: level | Predicted word: level
Misspelled word: basicaly | Correct word: basically | Predicted word: scaly
Misspelled word: triangulaur | Correct word: triangular | Predicted word: trian
gular
Misspelled word: unexpcted | Correct word: unexpected | Predicted word: unexpec
ted
Misspelled word: stanerdizing | Correct word: standardizing | Predicted word: s
tanding
Misspelled word: varable | Correct word: variable | Predicted word: parable
```

Βήμα 8:

Σε αυτό το βήμα, θα επιχειρήσουμε να βελτιστοποιήσουμε τον ορθογράφο, εισάγοντας βάρη στα edits βάσει του αρχείου wiki.txt. Πιο συγκεκριμένα, θα εκμεταλλευτούμε αυτό το αρχείο, για να εξάγουμε μια πιθανότητα για κάθε edit σε μία λέξη. Στη συνέχεια, θα θεωρήσουμε το βάρος αυτού του edit ως τον αρνητικό λογάριθμο της πιθανότητας που βρήκαμε. Αυτή η αλλαγή (πχ για το edit a->c) αποτυπώνεται στον παρακάτω μαθηματικό τύπο:

$$W(a\text{->}c) = -\log_{10}\left[\frac{\#(a\rightarrow c)}{\#(a\rightarrow *)}\right]$$

Προκειμένου να αναγνωρίσουμε ποιο edit χρειάζεται, για να διορθωθεί μία ανορθόγραφη λέξη, θα χρησιμοποιήσουμε το script word_edits.sh, το οποίο υλοποιεί τη διαδικασία που περιγράφεται στα ερωτήματα α, β, γ, της εκφώνησης. Αναλυτικότερα, το συγκεκριμένο script, αρχικά, δημιουργεί έναν αποδοχέα M για την ανορθόγραφη λέξη (μέσω της mkfstinput) και τον συνθέτει με τον Levensthein transducer L. Στη συνέχεια, η mkfstinput χρησιμοποιείται εκ νέου για τη δημιουργία του αποδοχέα N για την ορθή λέξη. Αφού πραγματοποιηθεί η νέα σύνθεση MLN, εκτελούμε τον αλγόριθμο shortestpath (μέσω της fstshortestpath), αφαιρώντας τις γραμμές μηδενικού κόστους (μέσω της grep) και κρατώντας τις στήλες 3, 4, οι οποίες αφορούν το edit (μέσω της cut).

Μέσω της εντολής **bash mk_edits_file.sh** αποθηκεύουμε στο αρχείο data/edits.txt όλα τα edits για το αρχείο wiki.txt.

wiki.txt:

```
abandonned  abandoned
aberation aberration
abilityes abilities
abilties  abilities
abilty  ability
abondon abandon
abbout  about
abotu about
abscence  absence
abondoned abandoned
abondoning  abandoning
abondons  abandons
aborigene aborigine
accesories  accessories
accidant  accident
```

Στη συνέχεια, μέσω του script extract_edit_freqs_and_mk_E.py, διαβάζουμε τα edits του αρχείου που δημιουργήσαμε και δημιουργούμε ένα dictionary με key το κάθε edit και value τη συχνότητα του συγκεκριμένου edit. Όπως έχει προαναφερθεί, από αυτές τις συχνότητες, έπειτα, χρησιμοποιείται ως βάρος ο αρνητικός λογάριθμός τους (μέσω της calculate_arc_weight) και κατά αναλογία με το ερώτημα 4 κατασκευάζουμε τον νέο transducer E.binfst. Ο νέος ορθογράφος δημιουργείται με την εντολή **bash scripts/compile_compose.sh E V** και πραγματοποιούμε το σύντομο evaluation του ερωτήματος 7 μέσω της **python3 scripts/20_words_test.py EV**.

Τα αποτελέσματα για τον ορθογράφο EV.binfst είναι τα ακόλουθα:

```
Misspelled word: contenpted | Correct word: contented | Predicted word: content
ed
Misspelled word: begining | Correct word: beginning | Predicted word: beginning
Misspelled word: problam | Correct word: problem | Predicted word: problem
Misspelled word: dirven | Correct word: driven | Predicted word: dive
Misspelled word: exstacy | Correct word: ecstasy | Predicted word: stay
Misspelled word: guic | Correct word: juice | Predicted word: i
Misspelled word: localy | Correct word: locally | Predicted word: local
Misspelled word: compair | Correct word: compare | Predicted word: comer
Misspelled word: pronounciation | Correct word: pronunciation | Predicted word:
 pronounce
Misspelled word: transportibility | Correct word: transportability | Predicted
word: nobility
Misspelled word: miniscule | Correct word: minuscule | Predicted word: mince
Misspelled word: independant | Correct word: independent | Predicted word: inde
pendent
Misspelled word: aranged | Correct word: arranged | Predicted word: ranged
Misspelled word: poartry | Correct word: poetry | Predicted word: poetry
Misspelled word: leval | Correct word: level | Predicted word: level
Misspelled word: basicaly | Correct word: basically | Predicted word: asia
Misspelled word: triangulaur | Correct word: triangular | Predicted word: trian
gular
Misspelled word: unexpcted | Correct word: unexpected | Predicted word: unexpec
ted
Misspelled word: stanerdizing | Correct word: standardizing | Predicted word: a
rising
Misspelled word: varable | Correct word: variable | Predicted word: vale
```

Βήμα 9:

Η τελική βελτιστοποίηση που θα δοκιμάσουμε στο μοντέλο μας, αφορά τη συχνότητα εμφάνισης λέξεων. Θα κατασκευάσουμε έναν, επιπλέον, αποδοχέα W, ο οποίος αποδέχεται όλες τις λέξεις του vocabulary μας με βάρος τον αρνητικό λογάριθμο της συχνότητας τους στο κείμενο. Επομένως, μέσω του αρχείου vocab/words.vocab.txt που δημιουργήσαμε στο βήμα 2, θα λάβουμε τη εύκολα τη συχνότητα κάθε λέξης. Αντίστοιχα με το βήμα 8, ο μαθηματικός τύπος αυτή τη φορά είναι ο ακόλουθος:

$W(word) = -\log_{10}[\frac{\#(word)}{\#(*)}]$

Η υπόλοιπη υλοποίηση του W βασίζεται σε αυτή του αποδοχέα V. Μπορούμε να δημιουργήσουμε τους ορθογράφους LVW, EVW μέσω των εντολών **bash scripts/compile_compose_withW.sh LV W** και **bash scritps/compile_compose_withW.sh EV W**, αντίστοιχα. Μετά το σύντομο evaluation του βήματος 7 έχουμε τα παρακάτω αποτελέσματα.

LVW.binfst:

```
Misspelled word: contenpted | Correct word: contented | Predicted word: content
ed
Misspelled word: begining | Correct word: beginning | Predicted word: beginning
Misspelled word: problam | Correct word: problem | Predicted word: problem
Misspelled word: dirven | Correct word: driven | Predicted word: given
Misspelled word: exstacy | Correct word: ecstasy | Predicted word: stay
Misspelled word: guic | Correct word: juice | Predicted word: in
Misspelled word: localy | Correct word: locally | Predicted word: only
Misspelled word: compair | Correct word: compare | Predicted word: company
Misspelled word: pronounciation | Correct word: pronunciation | Predicted word:
 provocation
Misspelled word: transportibility | Correct word: transportability | Predicted
word: respectability
Misspelled word: miniscule | Correct word: minuscule | Predicted word: minute
Misspelled word: independant | Correct word: independent | Predicted word: inde
pendent
Misspelled word: aranged | Correct word: arranged | Predicted word: and
Misspelled word: poartry | Correct word: poetry | Predicted word: party
Misspelled word: leval | Correct word: level | Predicted word: evil
Misspelled word: basicaly | Correct word: basically | Predicted word: easily
Misspelled word: triangulaur | Correct word: triangular | Predicted word: trian
gular
Misspelled word: unexpcted | Correct word: unexpected | Predicted word: unexpec
ted
Misspelled word: stanerdizing | Correct word: standardizing | Predicted word: s
tanding
Misspelled word: varable | Correct word: variable | Predicted word: parable
```

EVW.binfst:

```
Misspelled word: contenpted | Correct word: contented | Predicted word: content
ed
Misspelled word: begining | Correct word: beginning | Predicted word: being
Misspelled word: problam | Correct word: problem | Predicted word: problem
Misspelled word: dirven | Correct word: driven | Predicted word: even
Misspelled word: exstacy | Correct word: ecstasy | Predicted word: say
Misspelled word: guic | Correct word: juice | Predicted word: i
Misspelled word: localy | Correct word: locally | Predicted word: local
Misspelled word: compair | Correct word: compare | Predicted word: come
Misspelled word: pronounciation | Correct word: pronunciation | Predicted word:
 pronounce
Misspelled word: transportibility | Correct word: transportability | Predicted
word: sensibility
Misspelled word: miniscule | Correct word: minuscule | Predicted word: mine
Misspelled word: independant | Correct word: independent | Predicted word: inde
pendent
Misspelled word: aranged | Correct word: arranged | Predicted word: and
Misspelled word: poartry | Correct word: poetry | Predicted word: party
Misspelled word: leval | Correct word: level | Predicted word: evil
Misspelled word: basicaly | Correct word: basically | Predicted word: asia
Misspelled word: triangulaur | Correct word: triangular | Predicted word: trian
gular
Misspelled word: unexpcted | Correct word: unexpected | Predicted word: unexpec
ted
Misspelled word: stanerdizing | Correct word: standardizing | Predicted word: s
eizing
Misspelled word: varable | Correct word: variable | Predicted word: are
```

Σχετικά με τα αποτελέσματα των LV και LVW με εισόδους τα cit και cwt προκύπτουν τα παρακάτω αποτελέσματα:

LV:

- **cit:** city
- **cwt:** cut

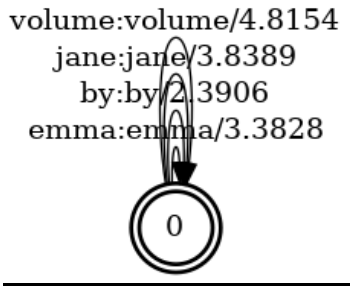
LVW:

- **cit:** it
- **cwt:** it

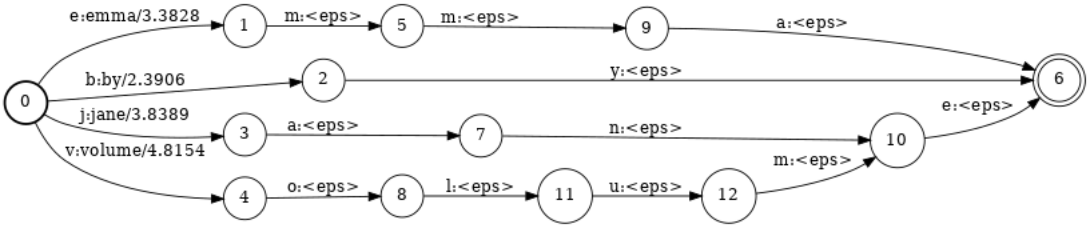
Όπως φαίνεται και από τα παραπάνω παραδείγματα, η προσθήκη της συχνότητας εμφάνισης των λέξεων στο μοντέλο μέσω του W, διαφοροποιεί αρκετά την έξοδο του ορθογράφου. Πιο συγκεκριμένα, στο μοντέλο χωρίς το W, παρατηρούμε πως τα αποτελέσματα για τις δύο εισόδους αποτελούνται από τα ελάχιστα edits που χρειάζονται, για να καταλήξουμε σε μία λέξη που υπάρχει στο vocabulary, ενώ στο μοντέλο με το W η συχνότητα της λέξης it είναι τόσο μεγαλύτερη των city, cut, ώστε να αποτελεί την έξοδο και στις δύο περιπτώσεις.

Επιπλέον, σχεδιάζοντας τους αποδοχείς W και VW με την fstdraw (για τις πρώτες 5 λέξεις), λαμβάνουμε τα εξής αποτελέσματα:

W.binfst:



VW.binfst:



Βήμα 10:

Στη συνέχεια, θα εκτελέσουμε ένα evaluation σε ένα αρκετά μεγαλύτερο σύνολο δεδομένων (spell_test.txt), χρησιμοποιώντας το script run_evaluation.py. Για να πραγματοποιηθεί το evaluation αρκεί η εντολή **python3 scripts/run_evaluation.py fsts/LV/EV/LVW/EVW.binfst**. Το accuracy που προκύπτει από αυτή τη διαδικασία φαίνεται παρακάτω:

Accuracy:

- **LV:** 0.5963
- **EV:** 0.4444
- **LVW:** 0.4370
- **EVW:** 0.1307

Σχολιασμός Αποτελεσμάτων:

Από τα παραπάνω αποτελέσματα διαπιστώνουμε πως η υλοποίηση με την καλύτερη επίδοση είναι η πρώτη, η οποία δεν περιλαμβάνει κάποια βελτιστοποίηση. Ένας λόγος για τον οποίο συμβαίνει αυτό είναι ότι η αντικατάσταση του L με τον E transducer συνεπάγεται ότι, αν κάποιο edit δεν υπάρχει στο αρχείο wiki.txt, τότε δεν θα γίνει ποτέ στον ορθογράφο, αφού θα έχει άπειρο βάρος. Επιπλέον, όταν εισάγουμε τις συχνότητες των λέξεων (μέσω του W) από ένα μόνο κείμενο, είναι αναμενόμενο ο ορθογράφος μας να είναι αρκετά biased σχετικά με κάποιες σπάνιες λέξεις που τυγχάνει να χρησιμοποιούνται πολύ στο κείμενο ή το αντίθετο.

Βήμα 11:

Στο συγκεκριμένο βήμα θα επιχειρήσουμε να αντιμετωπίσουμε τα δύο προβλήματα που αναφέρθηκαν παραπάνω ως οι κύριοι λόγοι, για τους οποίους οι βελτιστοποιήσεις που δοκιμάσαμε δεν απέδωσαν. Πιο συγκεκριμένα, αρχικά, θα λύσουμε το πρόβλημα με τα άπειρα βάρη των edits που δεν υπάρχουν στο wiki.txt, εφαρμόζοντας τη μέθοδο add-1 smoothing. Αυτή η μέθοδος περιγράφεται από τον ακόλουθο μαθηματικό τύπο:

$$W(a\text{->}c) = -\log_{10}\left[\frac{\#(a \rightarrow c) + 1}{\#(a \rightarrow *) + V}\right]$$

Λόγω του +1 στον αριθμητή, κάθε edit έχει ένα πιο λογικό βάρος και δεν καθίσταται αδύνατο απλά επειδή δεν υπάρχει στο wiki.txt. Επιπλέον, το +V (V είναι το πλήθος των edits) στον παρονομαστή διασφαλίζει πως οι πιθανότητες θα εξακολουθήσουν να αθροίζουν στο 1.

Μέσω της εντολής **python3 scripts/optimize_edit_freqs.py** δημιουργούμε την βελτιστοποιημένη μορφή του E transducer στο E_opt.fst. Στη συνέχεια, μέσω των εντολών **bash scripts/compile_compose.sh E_opt V** και **bash scripts/compile_compose_withW.sh E_optV W** δημιουργούμε τους δύο ορθογράφους E_optV.binfst και E_optVW.binfst, αντίστοιχα. Τέλος, μετά το evaluation των δύο ορθογράφων (**python3 scripts/run_evaluation.py fsts/E_optV(W).binfst**) έχουμε τα παρακάτω εμφανώς βελτιωμένα αποτελέσματα:

Accuracy:

- **EV:** 0.6519
- **EVW:** 0.5333

Η τελευταία βελτιστοποίηση που θα εφαρμοστεί θα αφορά το vocabulary, αφού αυτή τη φορά θα χρησιμοποιηθεί ένα αρκετά μεγαλύτερο (μεγέθους 50.000 σε αντίθεση με το Gutenberg που ήταν ~15.000) και πιο ποιοτικό λεξικό (https://github.com/hermitdave/FrequencyWords/blob/master/content/2016/en/). Αυτή τη φορά οι αλλαγές θα αφορούν τους acceptors V, W και όχι τους transducers L, E. Επομένως, αρκεί να αντικαταστήσουμε τα αρχεία vocab/words.syms, vocab/words.vocab.txt με τα νέα δεδομένα (αυτό θα γίνει μέσω της εντολής **python3 scripts/new_corpus_lexicon.py**) και να τρέξουμε τα αντίστοιχα scripts. Τα αποτελέσματα είναι τα ακόλουθα:

Accuracy:

- **LV:** 0.5741
- **EV:** 0.7000
- **LVW:** 0.5519
- **EVW:** 0.6556

Σχολιασμός Αποτελεσμάτων:

Τα καλύτερα αποτελέσματα παρατηρούνται για τον ορθογράφο EV (με add-1 smoothing και στο μεγαλύτερο vocabulary) με accuracy 70%. Την επίδοση αυτή ακολουθεί ο ορθογράφος EVW με accuracy 65.56%, ενώ το ίδιο ποσοστό χωρίς το add-1 smoothing και το μεγαλύτερο vocabulary ήταν μόλις 13.07%. Οι υπόλοιπες δύο υλοποιήσεις που δεν λαμβάνουν υπόψιν τις συχνότητες των edits, όπως είναι λογικό, έχουν κατά ~10% μικρότερο accuracy. Μερικές, επιπλέον, επεκτάσεις για να επιτευχθεί ακόμα καλύτερο accuracy θα ήταν, ίσως, να προσθέσουμε κι άλλα είδη edits όπως έχει αναφερθεί και στο βήμα 4 ή, σε περίπτωση που είχαμε κείμενο με μερικές ανορθόγραφες λέξεις, το W να εξαρτάται και από το context της ανορθόγραφης λέξης και όχι μόνο από τη συχνότητα εμφάνισής της.

Μέρος 2

Εισαγωγή:

Στο συγκεκριμένο μέρος θα ασχοληθούμε με τη χρήση λεξικών αναπαραστάσεων και, συγκεκριμένα, με το μοντέλο Word2Vec. Τα επόμενα βήματα της άσκησης περιλαμβάνουν, τόσο την δημιουργία embeddings βάσει του κειμένου Gutenberg, όσο και τη σύγκριση με τα αντίστοιχα προεκπαιδευμένα embeddings της Google. Στη συνέχεια, θα δημιουργήσουμε

τις αντίστοιχες αναπαραστάσεις συνολικών προτάσεων (Neural Bag of Words) και θα τις εφαρμόσουμε σε ένα classification problem για sentiment analysis. Τα δεδομένα που θα χρησιμοποιηθούν είναι σχόλια σε ταινίες στο IMDB.

Βήμα 12:

Αρχικά, θα εκπαιδεύσουμε word2vec embeddings 100 διαστάσεων μέσω της gensim. Η εκπαίδευση θα διαρκέσει 1000 εποχές και το context θα ρυθμιστεί μέσω της παραμέτρου window, η οποία στη συγκεκριμένη περίπτωση θα είναι 5. Επιπλέον, η εκπαίδευση θα πραγματοποιηθεί με την παραδοχή πως κάθε γραμμή του Gutenberg θα είναι μία πρόταση (το οποίο δεν αληθεύει κατά ανάγκη). Σε περίπτωση που δεν θέλαμε να ακολουθήσουμε αυτή την παραδοχή θα μπορούσαμε να κάνουμε tokenization μέσω της nltk.corpus.gutenberg.sents.

Για τις ζητούμενες λέξεις, οι κοντινότερές τους βάσει του cosine distance είναι οι ακόλουθες:

Bible:

	Λέξη	**Cosine similarity**
Κοντινότερες λέξεις	intermission	0.3755
	official	0.3662
	leprosy	0.3612
	wrist	0.3535
	landlord	0.3520

Book:

	Λέξη	**Cosine similarity**
Κοντινότερες λέξεις	written	0.4886
	note	0.4691
	letter	0.4672
	temple	0.4574
	chronicles	0.4464

Bank:

	Λέξη	Cosine similarity
Κοντινότερες λέξεις	wall	0.5506
	top	0.5033
	table	0.4977
	river	0.4751
	hill	0.4704

Water:

	Λέξη	Cosine similarity
Κοντινότερες λέξεις	waters	0.5739
	wood	0.5133
	wine	0.5071
	blood	0.4888
	river	0.4802

Σχολιασμός αποτελεσμάτων:

Τα αποτελέσματα δεν είναι ιδιαίτερα ικανοποιητικά (ειδικά στις πρώτες 3 λέξεις). Αυτό συμβαίνει, λόγω των περιορισμένων δεδομένων, που χρησιμοποιούνται στην εκπαίδευση των embeddings. Αυτό αποτυπώνεται ιδιαίτερα στις λέξεις, που δεν υπάρχουν τόσο συχνά στο κείμενο και παρατηρούμε πως οι κοντινότερες λέξεις που προκύπτουν είναι άσχετες (με εξαίρεση, ίσως, τις λέξεις book και water, οι οποίες παρατηρούνται αρκετές φορές στο κείμενο).

Επαναλαμβάνοντας την εκπαίδευση για 1500 εποχές και με την παράμετρο windows ίση με 10 έχουμε παρόμοιας ποιότητας αποτελέσματα με αυτά του παραπάνω πειράματος (σε κάποιες περιπτώσεις και χειρότερα).

Στη συνέχεια, θα εξετάσουμε αν ισχύει η ιδιότητα των word2vec να εκφράζουν αλγεβρικά σημασιολογικές αναλογίες για έννοιες. Ειδικότερα, θα εκτελέσουμε πράξεις v = w2v(“Ελλάδα”) – w2v(“Αθήνα”) + w2v(“Παρίσι”), αναμένοντας ως αποτέλεσμα ένα διάνυσμα v κοντά στο embedding της λέξης ‘Γαλλία’, καθώς η παραπάνω πράξη αντικατοπτρίζει την αναλογία «Η Αθήνα είναι για την Ελλάδα ότι το Παρίσι για τη Γαλλία». Για αυτό τον λόγο θα δοκιμάσουμε τις ακόλουθες τριάδες (“girls”, “queen”, “kings”), (“taller”, “tall”, “good”) και (“france”, “paris”, “london”), οι οποίες, ιδανικά, θα πρέπει να επιστρέψουν embeddings κοντινά σε αυτά των λέξεων “boys”, “better” και “England”,

αντίστοιχα. Τα αποτελέσματα είναι τα εξής (παρουσιάζονται οι 5 κοντινότερες λέξεις στο διάνυσμα που προκύπτει από την πράξη που ορίστηκε παραπάνω):

Τριάδα ("girl", "queen", "king"):

	Λέξη	**Similarity**
Κοντινότερες λέξεις	king	0.5956
	girl	0.5464
	woman	0.4466
	elders	0.4223
	man	0.4134

Τριάδα ("taller", "tall", "good"):

	Λέξη	**Similarity**
Κοντινότερες λέξεις	taller	0.5994
	good	0.5652
	better	0.4268
	useful	0.4108
	dearest	0.3739

Τριάδα ("france", "paris", "london"):

	Λέξη	**Similarity**
Κοντινότερες λέξεις	france	0.6977
	london	0.4663
	highbury	0.3986
	england	0.3874
	school	0.3772

Σχολιασμός αποτελεσμάτων:

Όπως φαίνεται, οι ζητούμενες λέξεις για τις τελευταίες δύο τριάδες είναι η $3^{η}$ και $4^{η}$ κοντινότεροι λέξη, αντίστοιχα. Αντίθετα, για την πρώτη τριάδα, η λέξη "boy" δεν εμφανίζεται, ωστόσο η $5^{η}$ κοντινότερη λέξη είναι η "man", η οποία είναι αρκετά παρόμοια. Επομένως, και σε αυτό το test τα word2vec embeddings που εκπαιδεύσαμε είχαν μέτρια αποτελέσματα.

Επαναλαμβάνουμε τα ίδια πειράματα με τα προεκπαιδευμένα **GoogleNews vectors** και τα αποτελέσματα είναι τα ακόλουθα:

Για τις κοντινότερες λέξεις:

Bible:

	Λέξη	**Cosine similarity**
Κοντινότερες λέξεις	bibles	0.6053
	scriptures	0.5746
	scripture	0.5698
	epistle	0.4508
	book	0.4387

Book:

	Λέξη	**Cosine similarity**
Κοντινότερες λέξεις	books	0.7379
	novels	0.6341
	novel	0.6122
	ebook	0.5657
	author	0.5405

Bank:

	Λέξη	**Cosine similarity**
Κοντινότερες λέξεις	banks	0.7441
	banker	0.6093
	branch	0.5431
	thrift	0.5009
	branches	0.4778

Water:

	Λέξη	**Cosine similarity**
Κοντινότερες λέξεις	lake	0.5903
	river	0.5769
	rivers	0.5619
	lakes	0.5309
	basin	0.5279

Για τις αλγεβρικά σημασιολογικές αναλογίες:

Τριάδα ("girl", "queen", "king"):

	Λέξη	**Similarity**
Κοντινότερες λέξεις	boy	0.7688
	girl	0.6964
	man	0.5588
	kid	0.4855
	son	0.4741

Τριάδα ("taller", "tall", "good"):

	Λέξη	**Similarity**
Κοντινότερες λέξεις	better	0.7095
	good	0.6328
	taller	0.5863
	quicker	0.5862
	stronger	0.5661

Τριάδα ("france", "paris", "london"):

	Λέξη	**Similarity**
Κοντινότερες λέξεις	london	0.7541
	france	0.7367
	england	0.6008
	europe	0.5708
	european	0.5275

Σχολιασμός αποτελεσμάτων:

Όπως φαίνεται από τα παραπάνω αποτελέσματα, τα embeddings της Google είναι αισθητά καλύτερα και αυτό αποτυπώνεται και στα δύο πειράματα. Προφανώς αυτό είναι αναμενόμενο, λόγω του όγκου των δεδομένων, που χρησιμοποιήθηκε από τη Google για αυτή την εκπαίδευση, ο οποίος δεν συγκρίνεται σε καμία περίπτωση με ένα βιβλίο.

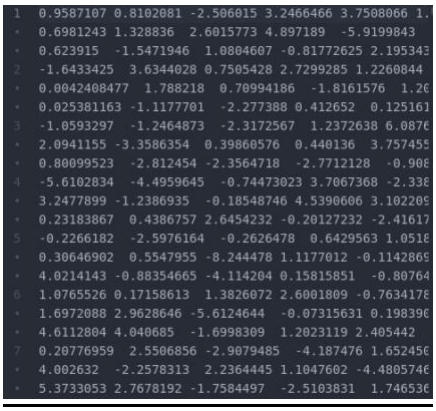
Σημείωση:

Τα παραπάνω αποτελέσματα μπορούν να εξαχθούν χρησιμοποιώντας την εντολή **python3 scripts/w2v_testing.py** με argument **0** (αν θέλουμε να χρησιμοποιήσουμε τα embeddings που εκπαιδεύσαμε) ή **1** (αν θέλουμε να χρησιμοποιήσουμε τα embeddings της Google).

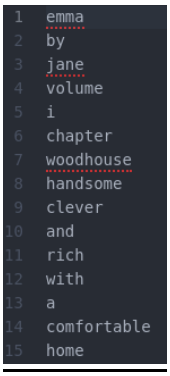
Βήμα 13:

Στη συνέχεα, αφού δημιουργήσουμε τα αρχεία embeddings.tsv, metadata.tsv στον φάκελο data, μέσω της εντολής python3 **scripts/w2v_mk_embeddings_metadata.py**, θα χρησιμοποιήσουμε το δύο εργαλεία, προκειμένου να οπτικοποιήσουμε τα embeddings και να διαμορφώσουμε πιο διαισθητικά αποτελέσματα.

embeddings.tsv:

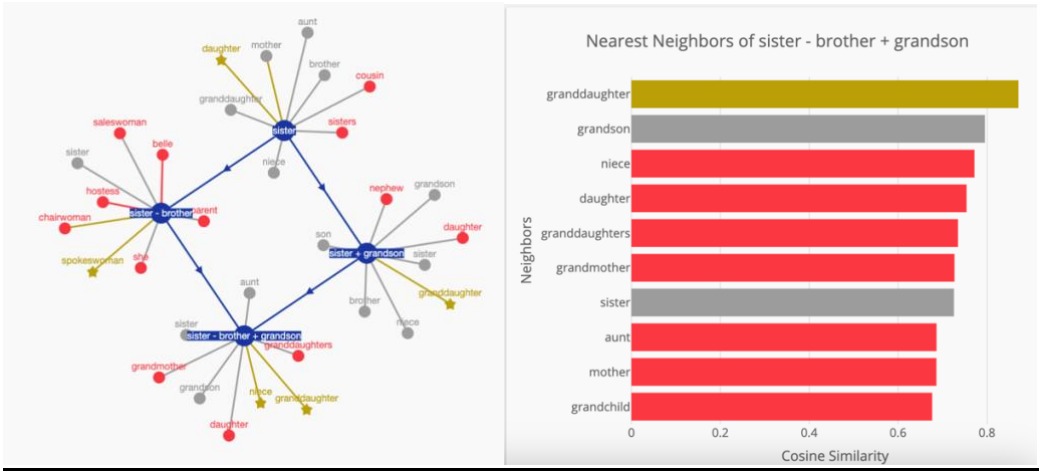


metadata.tsv:



Αρχικά, πραγματοποιώντας ορισμένα πειράματα στο plotly μπορούμε να παρατηρήσουμε, κυρίως καλά, αλλά και πιο μέτρια αποτελέσματα, σχετικά με την ιδιότητα των word2vec να εκφράζουν αλγεβρικά σημασιολογικές αναλογίες για έννοιες. Η σύγκριση με τα παραπάνω αποτελέσματα για τα embeddings που εκπαιδεύτηκαν, καθώς και τα αντίστοιχα της Google δεν έχει ιδιαίτερο νόημα, καθώς τα embeddings που χρησιμοποιούνται στο συγκεκριμένο εργαλείο είναι πολύ παρόμοια, αν όχι ολόιδια, με αυτά της Google. Επομένως, παρακάτω θα παρουσιαστούν, ένα καλό, ένα μέτριο, και ένα κακό αποτέλεσμα αυτής της ιδιότητας.

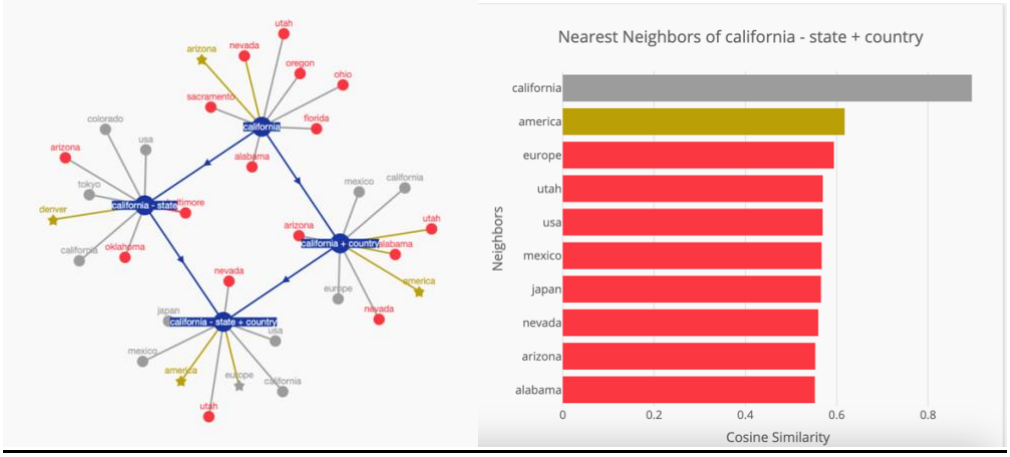
Καλό αποτέλεσμα (sister – brother + grandson):



Σχολιασμός Αποτελεσμάτων:

Στο συγκεκριμένο παράδειγμα, παρατηρείται μία καλή εφαρμογή αυτής της ιδιότητας, καθώς το αποτέλεσμα είναι το αναμενόμενο με cosine similarity 0.8709, ενώ τα υπόλοιπα πιο κοντινά vectors έχουν cosine similarity < 0.8.

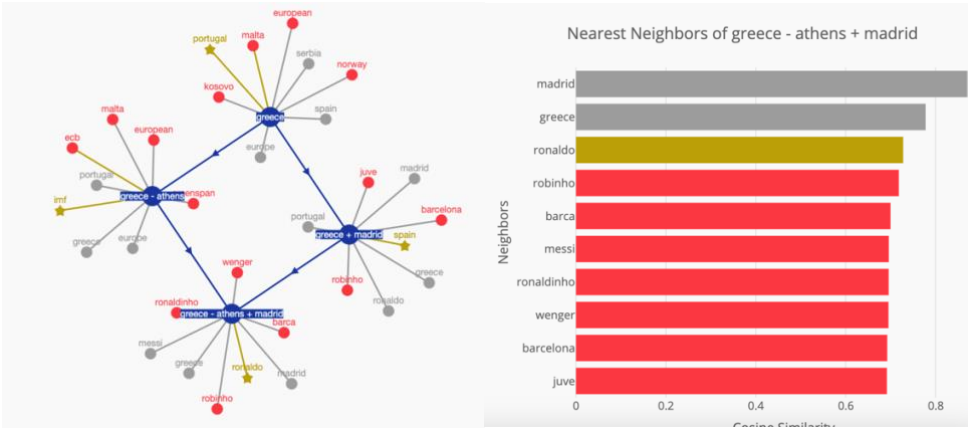
Μέτριο αποτέλεσμα (california – state + country):



Σχολιασμός Αποτελεσμάτων:

Σε αυτή την περίπτωση, θα περιμέναμε σαν nearest neighbor το america (2ο) ή το usa (5ο). Δεδομένου ότι μία από τις αναμενόμενες απαντήσεις είναι στη δεύτερη θέση, το παράδειγμα σίγουρα δεν είναι κακό, αλλά λόγω της διαφοράς στο cosine similarity μεταξύ των πρώτων δύο (california 0.8959 – america 0.6174) δεν αποτελεί καλό παράδειγμα.

Κακό αποτέλεσμα (greece – athens + madrid):



Σχολιασμός Αποτελεσμάτων:

Τα παραπάνω αποτελέσματα είναι εντελώς άστοχα, καθώς η αναμενόμενη απάντηση είναι η λέξη spain, αλλά οι περισσότερες απαντήσεις (εκτός των πρώτων δύο) σχετίζονται με το χώρο του ποδοσφαίρου.

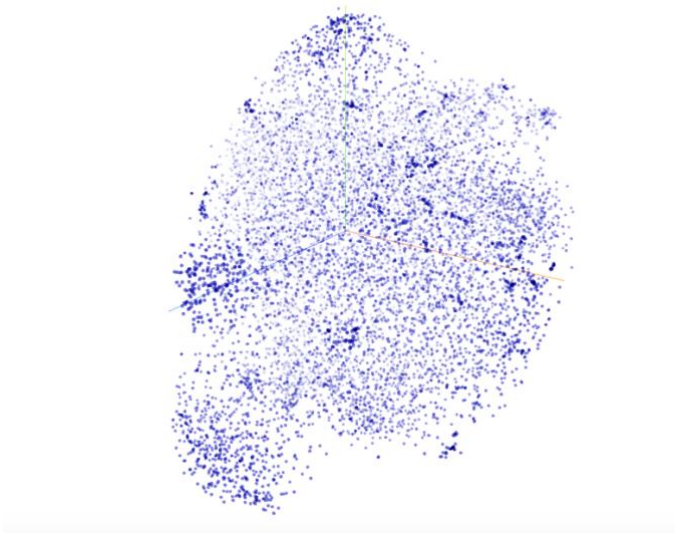
Στη συνέχεια, θα πραγματοποιήσουμε μερικά παραπάνω πειράματα στο embedding projector, αφού, πρώτα, φορτώσουμε τα αρχεία embeddings.tsv, metadata.tsv που δημιουργήσαμε προηγουμένως. Πιο συγκεκριμένα, θα οπτικοποιήσουμε τα embeddings, μειώνοντας τη διαστατικότητα με δύο διαφορετικές μεθόδους (T-SNE, PCA) Με αυτόν τον τρόπο, θα μπορούμε να συμπεράνουμε σχετικά με την επίδραση της μείωσης διαστατικότητας (και επομένως στην πληροφορία που περιέχεται) στα embeddings που εκπαιδεύσαμε.

Για τη μέθοδο T-SNE:

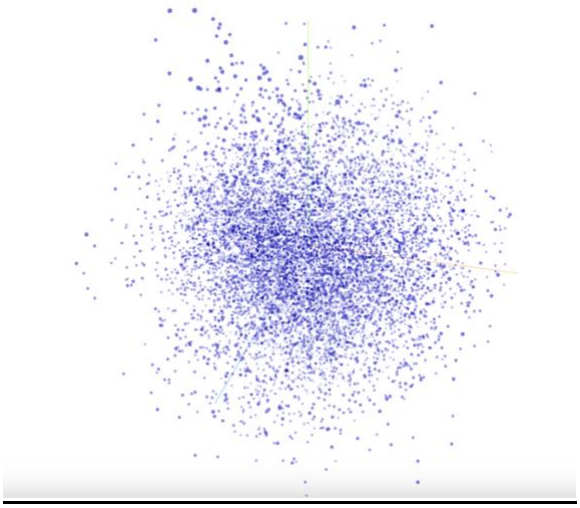
Η μείωση διαστατικότητας διήρκησε 1000 iterations και πραγματοποιήθηκε με παραμέτρους:

- Perplexity: 25
- Learning rate: 10

Οπτικοποιώντας τα embeddings και αναζητώντας τους κοντινότερους γείτονες στις λέξεις ‘book’, ‘water’ προκύπτουν τα παρακάτω:



Για τη μέθοδο PCA:



Βήμα 14:

Στο τελευταίο βήμα της εργαστηριακής άσκησης θα ασχοληθούμε με μία εφαρμογή των word2vec embeddings σε sentiment analysis. Πιο συγκεκριμένα, αξιοποιώντας το dataset της εκφώνησης (σχόλια για ταινίες στο IMDB) θα εκπαιδεύσουμε ένα μοντέλο logistic regression, θεωρώντας ως vector για κάθε σχόλιο, τον μέσο όρο των επιμέρους vectors της κάθε λέξης, που προκύπτουν από το wor2vec (Neural Bag of Words). Η εκπαίδευση αποσκοπεί στην ανάλυση συναισθήματος για κάθε review (θετικό ή αρνητικό συναίσθημα). Αρχικά, θα κατεβάσουμε τα δεδομένα, όπως υποδεικνύεται από την εκφώνηση και θα πραγματοποιήσουμε το preprocessing με βάση τις συναρτήσεις που δίνονται στο αρχείο **w2v_sentiment_analysis.py.** Αναλυτικότερα, τα στάδια του preprocessing είναι τα ακόλουθα:

- Tokenization κάνοντας split όπου υπάρχει κενό.
- Αφαίρεση σημείων στίξης.
- Μετατροπή όλων των χαρακτήρων σε πεζά.

Η εκπαίδευση θα πραγματοποιηθεί, αφού τα δεδομένα προεπεξεργαστούν, μετατραπούν σε NBOW αναπαραστάσεις και χωριστούν σε train και test δεδομένα (χρησιμοποιείται η default αναλογία 0.75-0.25). Έπειτα κάνουμε fit τα δεδομένα στο μοντέλο και εξάγουμε τα predictions για το test set. Το evaluation του μοντέλου θα πραγματοποιηθεί μέσω ενός classification report και ενός confusion matrix (από τις αντίστοιχες συναρτήσεις της sci-kit learn). Τα αποτελέσματα, χρησιμοποιώντας τις word2vec αναπαραστάσεις που εκπαιδεύσαμε στα προηγούμενα βήματα, φαίνονται παρακάτω:

Classification report:

	precision	recall	f1-score	support
0	0.73	0.74	0.73	2501
1	0.74	0.72	0.73	2499
accuracy			0.73	5000
macro avg	0.73	0.73	0.73	5000
weighted avg	0.73	0.73	0.73	5000

Confusion matrix:

Predictions/True	Negative	Positive
Negative	1854	647
Positive	697	1802

Αντικαθιστώντας τα embeddings με τα προεκπαιδευμένα της Google και επαναλαμβάνοντας την εκπαίδευση λαμβάνουμε τα ακόλουθα αποτελέσματα:

Classification report:

	precision	recall	f1-score	support
0	0.84	0.85	0.84	2501
1	0.85	0.84	0.84	2499
accuracy			0.84	5000
macro avg	0.84	0.84	0.84	5000
weighted avg	0.84	0.84	0.84	5000

Confusion matrix:

Predictions/True	Negative	Positive
Negative	2118	383
Positive	411	2088

Σημείωση: Αρκεί η εντολή **python3 w2v_sentiment_analysis.py 0/1** (0 για τα embeddings που εκπαιδεύσαμε στο πλαίσιο της άσκησης, 1 για τα αντίστοιχα της Google).

Σχολιασμός αποτελεσμάτων:

Όπως φαίνεται από την αξιολόγηση του μοντέλου, και σε αυτή την περίπτωση προκύπτει ότι τα προεκπαιδευμένα embeddings είναι πιο ποιοτικά από αυτά που εκπαιδεύσαμε εμείς, όπως αναμέναμε, καθώς οι μετρικές τους είναι καλύτερες (περίπου κατά 0.1). Επιπλέον, παρατηρείται μία ισορροπία μεταξύ των recall και precision (όπως και στα misclassifications του confusion matrix), το οποίο σημαίνει πως το μοντέλο δεν είναι biased προς μία από τις δύο κλάσεις του προβλήματος (θετικό ή αρνητικό συναίσθημα). Σε αυτό συμβάλλει και το ότι έχουμε ένα balanced dataset.