

Επεξεργασία Φωνής & Φυσικής Γλώσσας

Αναφορά 1ου Εργαστηρίου

Εισαγωγή στις Γλωσσικές Αναπαραστάσεις

Δημόπουλος Χρήστος (031 17 037)

Δήμος Δημήτρης (031 17 165)

7ο Εξάμηνο - Ροή Σ

Φθινόπωρο 2020

1 Μέρος 1: Κατασκευή Ορθογράφου

Στα 7 πρώτα βήματα που ακολουθήθηκαν κατασκευάστηκε ένας πρώτος ορθογράφος που αποτελεί την προπαρασκευή του εργαστηρίου. Στο παραδοτέο αρχείο περιέχονται στον φάκελο scripts τα αρχεία κώδικα:

- step1-6.py
- step7-spell_test.py
- step8-9.py
- helpers.py
- mkfstinput.py
- util.py
- run_evaluation.py
- predict.sh
- word_edits.sh

Η εκτέλεση των αρχείων step1-6.py και step8-9.py (το οποίο χρειάζεται αρκετό χρόνο για να εκτελεστεί) δημιουργούν όλα τα ζητούμενα FSTs και .txt αρχεία του Μέρους 1 (με την βοήθεια των υπολοίπων αρχείων).

1.1 Βήμα 1: Κατασκευή corpus

α) Αρχικά, κατεβάζουμε το Gutenberg Corpus τρέχοντας το δοθέν script "fetch_gutenberg.py", το οποίο επεξεργάζεται το corpus πρωτού αυτό αποθηκευτεί σε .txt αρχείο. Η επεξεργασία αποσκοπεί στο να "καθαρίσει" το corpus πρωτού το χρησιμοποιήσουμε.

- Κατ' αρχάς, κατεβαίνει το gutenber corpus αποτελούμενο από 18 βιβλία.
- Το script διαχωρίζει κάθε σειρά του corpus σε επιμέρους συμβολοσειρές, τις οποίες και επεξεργάζεται.
- ”Πετάει” τα περισσεύονται κενά, μετατρέπει όλους τους χαρακτήρες σε πεζούς και διορθώνει συντομευμένες φράσεις σε ολόκληρες (π.χ. you're → you are). Εν τέλει, ”πετάει” όποιον χαρακτήρα δεν είναι πλέον πεζό γράμμα ή κενό.
- Στη συνέχεια, δημιουργεί έναν πίνακα λιστών (μία για κάθε σειρά κειμένου) με στοιχεία τις λέξεις που υπάρχουν στο corpus ως ξεχωριστή συμβολοσειρά η κάθε μία. Αυτός ο πίνακας περιέχει τα tokens.
- Τέλος, ”πετάει” όποιο στοιχείο του πίνακα είναι κενή συμβολοσειρά και αντιγράφει τα υπόλοιπα στην έξοδο.

Αν επιθυμούσαμε να συμπεριλάβουμε και λέξεις γλωσσών στις οποίες σε ορισμένα σημεία η στίξη είναι απαραίτητη, όπως είναι τα γαλλικά, ή να συμπεριλάβουμε και πιθανές διορθώσεις αρτικολέξων - που εξ ορισμού χρειάζονται τα σημεία στίξης - θα έπρεπε θα εφαρμόσουμε λιγότερο επιθετικό pre-processing.

β) Αν επεκτείναμε το corpus με περισσότερα βιβλία, τότε στην στατιστική ανάλυση συχνότητας λέξεων θα είχαμε μεγαλύτερο δειγματικό χώρο και τα αποτελέσματά μας θα ήταν ακριβέστερα. Έτσι, αν θέλαμε να προβλέψουμε ποιο είναι το ενδεχόμενο λάθος λεξιλογίου που συνέβη, θα είχαμε περισσότερες πιθανότητες να το βρούμε επιλέγοντας από μια λίστα επιλογών καταταγμένων βάσει της συχνότητας χρήσης τους. Μεγαλύτερο corpus, λοιπόν, συνεπάγεται καλύτερη πιθανοτική κατάταξη, άρα και περισσότερες πιθανότητες εντοπισμού και διόρθωσης λάθους.

Επιπλέον, εμπλουτίζοντας το corpus με επιπλέον βιβλία συγγραφέων, παρέχεται περισσότερο υλικό που μπορεί να χρησιμοποιηθεί σε τεχνικές με νευρωνικά δίκτυα, οι οποίες αποσκοπούν στην αναγνώριση προτύπων (πχ αναγνώριση συγγραφέα, αναγνώριση είδους βιβλίου).

1.2 Βήμα 2: Κατασκευή λεξικού

Δημιουργήσαμε ένα dictionary που περιέχει σαν keys όλα τα μοναδικά tokens που βρίσκονται στο corpus και values τον αριθμό εμφανίσεων του token στο κείμενο, φροντίζοντας να αφαιρέσουμε όσα tokens εμφανίζονται λιγότερες από 5 φορές. Το βήμα αυτό πραγματοποιείται, επειδή οι λέξεις που σε 18 βιβλία δεν έχουν εμφανιστεί περισσότερες από 4 φορές θα είναι και αρκετά σπάνιες, ώστε να μην χρειάζεται να συμπεριληφθούν στην διαμόρφωση του spell checker, διότι σπανίως θα χρησιμεύσει η καθεμιά τους, πόσω μάλλον όλες τους. Τέλος, γράψαμε στο αρχείο ”vocab/words.vocab.txt” το λεξικό σε δύο tab separated στήλες, όπου η πρώτη στήλη περιέχει τα tokens και η δεύτερη στήλη τους αντίστοιχους αριθμούς εμφανίσεων.

1.3 Βήμα 3: Δημιουργία συμβόλων εισόδου/εξόδου

α) Πριν κατασκευάσουμε FSTs αντιστοιχήσαμε κάθε πεζό αγγλικό χαρακτήρα σε ένα αύξοντα ακέραιο index και αποθηκεύσαμε αυτή την αντιστοίχιση στο αρχείο "chars.syms". Πρακτικά αντιστοιχίσαμε τα πιθανά σύμβολα εισόδου (ή εξόδου) σε μοναδικούς δείκτες. Το πρώτο σύμβολο που αντιστοιχίσαμε είναι το ε (κενό), κωδικοποιημένο ως $<eps>$, στον αριθμό 0, ενώ τα υπόλοιπα είναι τα πεζά γράμματα της Αγγλικής αλφαριθμήτου με τη σειρά.

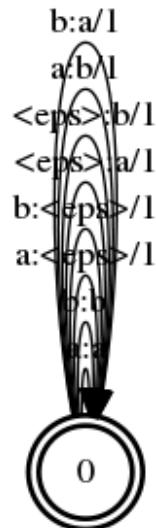
β) Κατά αντιστοιχία δημιουργήσαμε το αρχείο "words.syms" που αντιστοιχίζει το $<eps>$ στον αριθμό 0 και κάθε λέξη (token) από το λεξιλόγιο που κατασκευάσαμε στο Βήμα 2 σε ένα μοναδικό ακέραιο index.

1.4 Βήμα 4: Κατασκευή μετατροπέα edit distance

Στο βήμα αυτό κατασκευάζουμε έναν μετατροπέα L με μία κατάσταση που βασίζεται στην απόσταση Levenshtein. Τα χρησιμοποιούμενα edits είναι 3 ειδών: εισαγωγές χαρακτήρων, διαγραφές χαρακτήρων και αντικαταστάσεις χαρακτήρων. Κάθε ένα από αυτά τα edits χαρακτηρίζεται από ένα κόστος. Σε αυτό το στάδιο θεωρούμε ότι όλα τα edits έχουν κόστος 1.

Ο μετατροπέας αντιστοιχίζει:

- κάθε χαρακτήρα στον εαυτό του με βάρος 0 (no edit)
- κάθε χαρακτήρα στο κενό με βάρος 1 (deletion)
- το κενό σε κάθε χαρακτήρα με βάρος 1 (insertion)
- κάθε χαρακτήρα σε κάθε άλλο χαρακτήρα με βάρος 1.



Αν θέσουμε σαν είσοδο στον transducer L μια λέξη, τότε αυτός παράγει στην έξοδο όλες τις πιθανές λέξεις στις οποίες μπορεί να μεταμορφωθεί η λέξη εισόδου, με χρήση των 4 μετατροπών που εφαρμόζει ο L . Με χρήση του shortest path, ο L θα παράγει τις λέξεις στις οποίες μεταμορφώνεται η λέξη εισόδου με το ελάχιστο δυνατό κόστος, δηλαδή τον εαυτό της (κόστος: 0).

Παραθέτουμε το σχέδιο του L για ένα μικρό υποσύνολο του αλφαριθμήτου για ευκολότερη οπτικοποίηση.

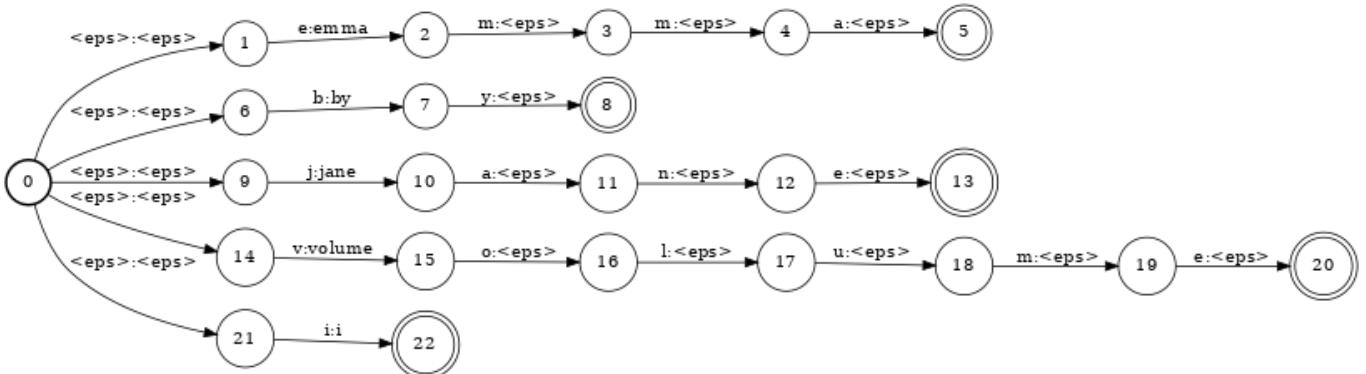
Στα edits που εφαρμόζει ο transductor, θα μπορούσαμε να είχαμε συμπεριλάβει επιπλέον edits όπως η μετάβαση από απλό σύμφωνο σε διπλό (αν αυτό είναι δόκιμο), όπως $t \rightarrow tt$. Ένα τέτοιο edit θα διευκόλυνε διορθώσεις όπως η sitting \rightarrow sitting. Ακόμη, θα μπορούσαμε να μεταβάλλουμε τα βάρη, ανάλογα με την απόσταση των πλήκτρων στο πληκτρολόγιο. Η μετάβαση $a \rightarrow s$ θα "κόστιζε"

τότε λιγότερο από την μετάβαση $a \rightarrow k$, αφού το πλήκτρο του γράμματος s είναι πολύ πιο κοντά στο a από όσο είναι το k , άρα η πρώτη μετάβαση θα πρέπει να γίνεται πιο "εύκολα".

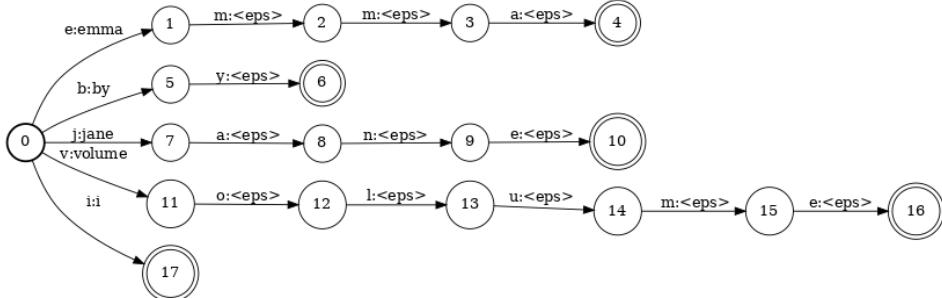
1.5 Βήμα 5: Κατασκευή αποδοχέα λεξικού

Σε αυτό το βήμα κατασκευάσαμε έναν αποδοχέα V ο οποίος αποδέχεται τις λέξεις του λεξικού του βηματού 2. Το μοντέλο του V συντέθηκε φτιάχοντας ένα FST για την κάθε λέξη του λεξικού και μια ξεχωριστή αρχική κατάσταση, ενώνοντας με ε-κίνηση την αρχική κατάσταση με την πρώτη κατάσταση καθενός εκ των επιμέρους FST. Με την κλήση των "fstrmepson", "fstdeterminize" και "fstminimize" το μοντέλο του V βελτιστοποιείται.

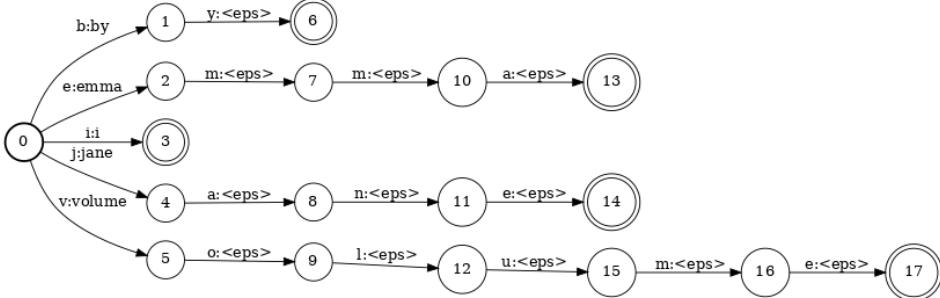
Για εύκολη οπτικοποίηση παραθέτουμε ένα μικρό υποσύνολο του V που αναγνωρίζει τις 5 πρώτες λέξεις του λεξικού μας. Στη συνέχεια παρουσιάζουμε τον acceptor μετά την εφαρμογή καθεμιάς από της συναρτήσεις βελτιστοποίησης.



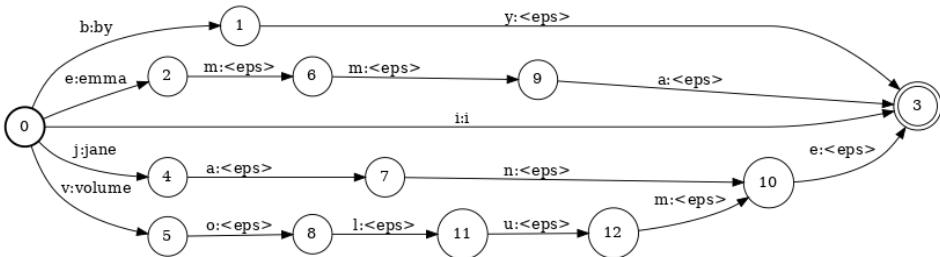
Η "fstrmepson" αφαιρεί εκινήσεις (όταν και η είσοδος και η έξοδος της κίνησης είναι ε) από τον transducer. Το αποτέλεσμα είναι ένα ισοδύναμο FST, που δεν έχει ε-μεταβάσεις.



Η ”fstdeterminize” καθιστά ντετερμινιστικό έναν transducer με βάρη. Το αποτέλεσμα είναι ένα isodünamico FST με την ιδιότητα ότι καμία κατάσταση δεν έχει δύο μεταβάσεις με την ίδια είσοδο. Για αυτή την διαδικασία, οι ε-κινήσεις αντικετωπίζονται σαν κανονικά σύμβολα. Βέβαια, στην περίπτωσή του acceptor μας, η συγκεκριμένη συνάρτηση δεν επιφέρει κάποια αλλαγή. Το προϊόν της ”fstrmer-silon”, που εφαρμόστηκε πρώτη, είναι ήδη determinized.



Η ”fstminimize” πραγματοποιεί την ελαχιστοποίηση ντετερμινιστικών αυτομάτων με βάρη και μετατροπέων. Αν το FST εισόδου, έστω A, είναι ένα αυτόματο (acceptor), αυτή η διαδικασία παράγει το ελάχιστο αυτόματο B που είναι isodünamico με το A, δηλαδή το αυτόματο με τον ελάχιστο δυνατό αριθμό καταστάσεων που είναι isodünamico με το A.



Οι συναρτήσεις αυτές τροποποιούν τον acceptor και μειώνουν τον αριθμό των ακμών στο τελικό μοντέλο, αφαιρώντας εξονυχιστικά οποιαδήποτε κίνηση θα μπορούσε να γίνει συντομότερα. Με αυτόν τον τρόπο, μειώνεται η πολυπλοκότητα διάσχισης του DAG που αποτελεί το FST (DAG traversal) και η λειτουργία του αυτομάτου γίνεται ταχύτερα, ενώ αν είχαμε ένα υποθετικό αυτόματο με πολύ μεγαλύτερο όγκο δεδομένων, θα γινόταν και καλύτερη εκμετάλλευση μνήμης. Επιπλέον, η ύπαρξη αυτών των συναρτήσεων μας επιτρέπει να σχεδιάζουμε acceptors και transducers με πιο ευκολό τρόπο, αφού το μόνο που απαιτείται είναι μια εύκολη και ”αφελής” σχεδίαση όπως αυτή που περιγράψαμε στην κατασκευή του V. Τα υπόλοιπα τα αναλαμβάνουν αυτές οι συναρτήσεις.

1.6 Βήμα 6: Κατασκευή ορθογράφου

- Σε αυτό το σημείο κατασκευάσαμε τον ορθογράφο. Αφού ταξινομήσαμε τις εισόδους του Levenshtein Transducer (L.fst) και τις εξόδους του Dictionary Acceptor (V.fst) του βα ερωτήματος,

συνθέσαμε τα δύο FST σε ένα παράγοντας τον min edit distance spell checker S.fst. Η συμπεριφορά του spell checker διαφοροποιείται σύμφωνα με τα βάρη που έχουμε τοποθετήσει στις ακμές των FST.

Όταν τα edits που δοκιμάζει έχουν ίσο βάρος, τότε η απάντηση-διόρθωση που ο transducer δίνει σε κάθε λέξη-είσοδο είναι απλώς η συντομότερη δυνατή, δηλαδή η λέξη που απαιτεί τις λιγότερες αλλαγές για να μετατραπεί στη λέξη εισόδου.

Στην περίπτωση τώρα που τα edits δεν είναι ισοβαρή, ο μετατροπέας συμπεριφέρεται με διαφορετικό τρόπο. Μια εκδοχή θα ήταν να ορίζαμε διαφορετικά βάρη στις ακμές βάσει της πιθανότητας που έχει ο transducer να παράξει σωστή λέξη ακολουθώντας τα "πιο φθηνά" μονοπάτια. Δηλαδή, ακμές που συμβολίζουν πιο πιθανές διορθώσεις να έχουν και χαμηλότερο κόστος ώστε να προτιμώνται. Η τελική διόρθωση θα έχει έτσι περισσότερες πιθανότητες να είναι και η τελικώς ζητούμενη.

β) Πιθανές προβλεψεις του min edit spell checker:

- cit → city (1 insertion), it (1 deletion), cut (1 substitution) κλπ
- cwt → cut (1 substitution) κλπ

1.7 Βήμα 7: Δοκιμή ορθογράφου

α) Δοκιμάζουμε τώρα τον ορθογράφο, με χρήση του predict.sh. Συνθέσαμε ένα υποστηρικτικό python script (step7-spell_test.py) για να δοκιμάσουμε όλες τις λανθασμένες εκδοχές των 20 πρώτων λέξεων από το σύνολο spell_test.txt. Το script χρησιμοποιείται ως εξής:

bash step7-spell_test.py [όνομα δοκιμαζόμενου FST]

και παράγει στο main directory ένα αρχείο με όνομα: **FST.fst_test_results.txt**

Παραθέτουμε ενδεικτικά κάποια test:

Correct word: contented
contenpted → contented
contende → contented

Correct word: problem
problam → problem

Correct word: driven
dirven → given

Correct word: ecstasy
extacy → ecstasy

Correct word: poetry
poartry → party
poertry → party

Correct word: level
leval → legal

Correct word: basically
basicaly → sickly

Correct word: variable
varable → parable

To script που χρησιμοποιήσουμε για να δοκιμάσουμε τον spell checker, εν συντομίᾳ, συνθέτει (με χρήση των επικουρικών script mkinputsft.py και util.py) έναν transducer που αποδέχεται τη

λέξη εισόδου και τον συνθέτει με τον spell checker transducer. Τελικά, προωθεί στην έξοδο απλώς την διορθωμένη λέξη. Η αλληλουχία των πράξεων που εκτελεί το script οδηγεί στη διορθωμένη λέξη με τον εξής τρόπο: Ο spell checker που έχουμε από προηγουμένως ουσιαστικά προωθεί στην έξοδο όλες τις πιθανές τροποποιήσεις που μπορεί να υποστεί η κάθε λέξη του λεξικού. Συνθέτοντας έτσι ένα FST που αναγνωρίζει μόνο τη λέξη που ελέγχουμε με τον spell checker, προκύπτει ένας transducer που περιέχει όλες τις πιθανές τροποποιήσεις που μπορούν να μετατρέψουν τη λέξη εισόδου στη διορθωμένη. Το script απλοποιεί αυτό το FST, παίρνοντας το συντομότερο μονοπάτι, αφαιρώντας τις ε-κινήσεις και ταξινομώντας τις ακμές. Έτσι, προκύπτει μονάχα η διορθωμένη λέξη που απαιτούσε τις λιγότερες αλλαγές για να συντεθεί.

Σε αυτό το σημείο ολοκληρώνεται η προπαρασκευή της εργαστηριακής άσκησης.

Στα επόμενα βήματα βελτιώνουμε τον ορθογράφο εισάγοντας γλωσσική πληροφορία.

1.8 Βήμα 8: Υπολογισμός κόστους των edits

Κατεβάζουμε το corpus data/wiki.txt που περιέχει λέξεις συχνά γραμμένες λάθος, μαζί με τις σωστές τους. Σε πρώτη φάση, χρησιμοποιούμε το δούθεν pattern ενός script (word_edits.sh) το οποίο παράγει τις ελάχιστες μετατροπές που πρέπει να υποστούν δύο λέξεις εισόδου ώστε να μετατραπεί η πρώτη στη δεύτερη. Το script, συγκεκριμένα, δημιουργεί με τη σειρά τα εξής προσωρινά FSTs.

- M: το FST που αναγνωρίζει την (λανθασμένη) πρώτη λέξη εισόδου
- $M \circ L$: το FST - σύνθεση του M με τον Levenshtein Transducer του βήματος 4
- N: το FST που αναγνωρίζει την (σωστή) δεύτερη λέξη εισόδου
- $M \circ L \circ N$: το FST που μετρατρέπει την πρώτη λέξη στη δεύτερη

Στη συνέχεια, εκτελεί την fstshortestpath και ακολούθως την fstprint αποθηκεύοντας μονο το edit που υπέστη η πρώτη λέξη (με χρήση των εντολών grep και cut). Αν το script δεν περιείχε τις grep και cut, τότε θα παρήγαγε ολόκληρο το (συντομότερο) μονοπάτι (μετασχηματισμούς) που ακολουθούνται για να μετατραπεί η ανορθόγραφη λέξη στη σωστή. Μερικά παραδείγματα χρήσης:

```
bash scripts/word_edits.sh aberation aberration
<eps>    r
bash scripts/word_edits.sh abilityes abilities
y          i
bash scripts/word_edits.sh abilities abilities
bash scripts/word_edits.sh abilty abilities
y          s
<eps>    e
<eps>    i
<eps>    i
```

Στη συνέχεια, συνθέσαμε ένα script με το οποίο δοκιμάσαμε όλα τα ζεύγη λέξεων από το corpus wiki.txt και αποθηκεύσαμε τα edits (στο αρχείο edits_8d.txt) που υπέστη η πρώτη λέξη κάθε ζεύγους. Μετρήθηκαν οι συχνότητες από κάθε είδους edit που έλαβε χώρα και, ύστερα, κατασκευάσαμε έναν νέο Levenshtein transducer E με βάρη ακμών των αρνητικό λογάριθμο των συχνοτήτων των αντίστοιχων edit. Με επανάληψη των βήματων 6 και 7, συνθέσαμε έναν νέο spell checker $E \circ V$. Δοκιμάζοντας τον $E \circ V$ με το script του βήματος 7 παράγουμε το αρχείο με τις διορθώσεις που εφαρμόζει ο transducer $E \circ V$ στις 20 πρώτες λέξεις του spell_test.txt. Ενδεικτικά:

Correct word: problem
problam → problem
proble → problem

Correct word: driven
dirven → driven

Correct word: juice
guic → guil
juce → guil

Correct word: arranged
aranged → arranged

Correct word: triangular
triangulaur → triangular

Correct word: ecstasy
exstacy → exactly
ecstacy → exactly

1.9 Βήμα 9: Εισαγωγή της συχνότητας εμφάνισης λέξεων (Unigram word model)

Σε επόμενο στάδιο εισάγουμε στο μοντέλο τις συχνότητες εμφάνισης λέξεων, με σκοπό ο ορθογράφος να προτείνει πιο πιθανές λέξεις στις διορθώσεις του. Για να το πετύχουμε αυτό χρησιμοποιησάμε το λεξιλόγιο με τις συχνότητες των λέξεων του βήματος 2 (vocab/words.vocab.txt) και κατασκευάσαμε το γλωσσικό μοντέλο W ο οποίος αποτελείται από μονάχα μια κατάσταση και αντιστοιχίζει κάθε λέξη στον εαυτό της με βάρος $-\log(\text{word frequency})$. Έχοντας, πλέον στη διάθεσή μας τα L , E , V , W , δημιουργούμε τους $L \circ V \circ W$ και τον $E \circ V \circ W$.

Αξιολογώντας τον ορθογράφο $L \circ V \circ W$ με τη διαδικασία του Βήματος 8 και (με το script step7-spell_test.py) παίρνουμε ενδεικτικά τα ακόλουθα αποτελέσματα:

Correct word: driven
dirven → the

Correct word: ecstasy
extacy → the

Correct word: juice
guic → the
juce → the

Correct word: locally
localy → and

Correct word: compare
compair → and

Correct word: pronunciation
pronounciation → unto

Συγκρίνοντας τα αποτελέσματα του $L \circ V$ και του $L \circ V \circ W$, παρατηρούμε ότι ο πρώτος παράγει διορθωμένες λέξεις που προκύπτουν από την (ανορθόγραφη) λέξη είσοδου με το λιγότερο δυνατό πλήθος edits. Αντιθέτως, ο $L \circ V \circ W$ transducer διορθώνει τις λέξεις εισόδου παράγοντας τις πιο συχνά εμφανιζόμενες λέξεις (and, the), καθώς αυτές έχουν το μικρότερο κόστος στο W FST.

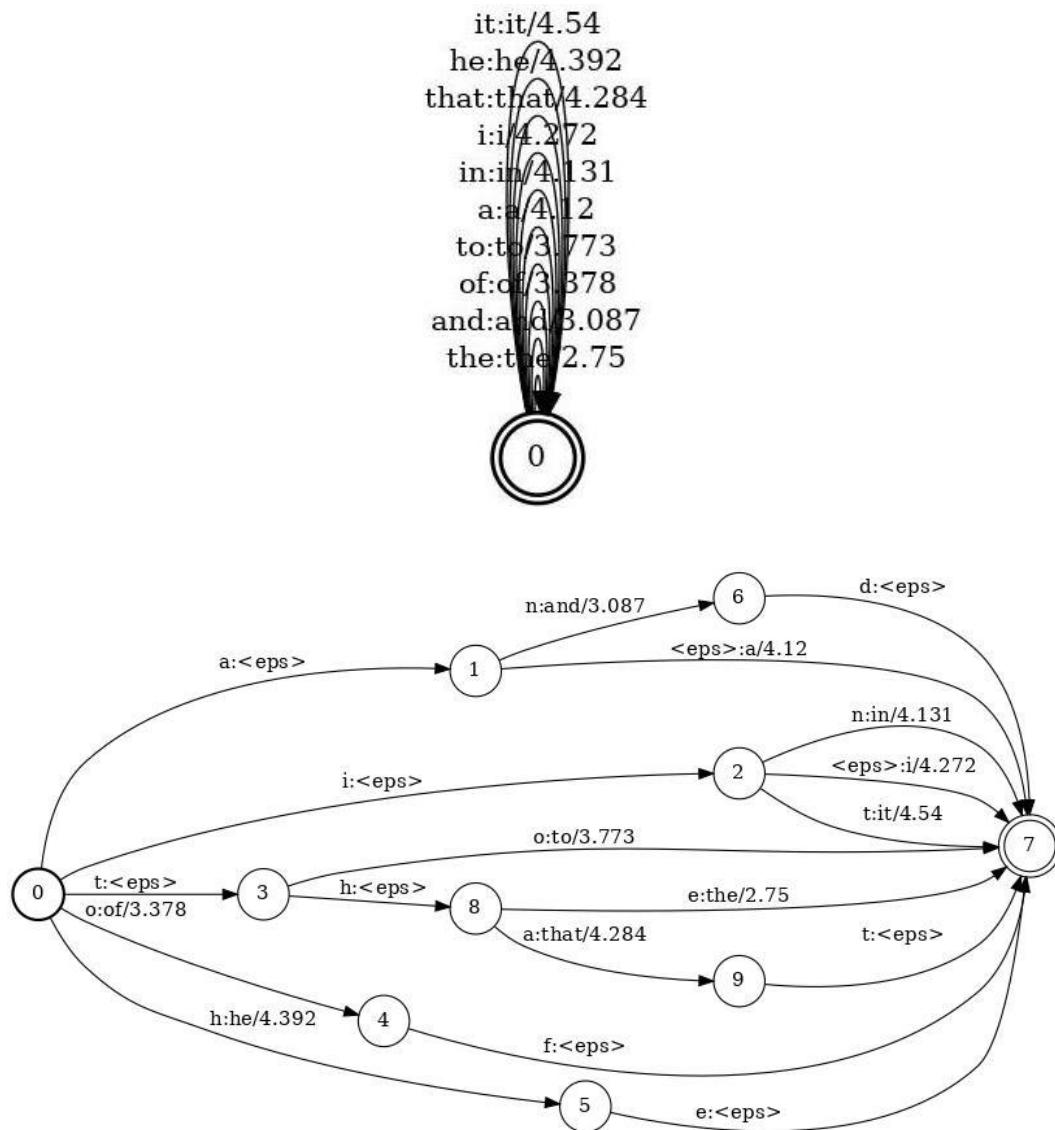
Για παράδειγμα, ο $L \circ V$ διορθώνει τις λέξεις cwt και cit, ως εξής:

- cit → city
- cat → cat

Ενώ ο $L \circ V \circ W$ ως εξής:

- cit → it
- cat → and

Πριν προχωρήσουμε στην αξιολόγηση των ορθογράφων, σχεδιάζουμε ένα υποσύνολο (10 λέξεων) των γλωσσικών μοντέλων W και $V \circ W$.



1.10 Βήμα 10: Αξιολόγηση των ορθογράφων

Εν τέλει, αξιολογούμε τους ορθογράφους $L \circ V$, $L \circ V \circ W$, $E \circ V$ & $E \circ V \circ W$, με χρήση του script "run_evaluation.py" και ολοκληρώνουμε το Μέρος 1 της Εργαστηριακής Άσκησης.

- Accuracy $L \circ V$: (161/270 hits) → 59.63 %
- Accuracy $L \circ V \circ W$: (12/270 hits) → 4.44 %
- Accuracy $E \circ V$: (186/270 hits) → 68.89 %
- Accuracy $E \circ V \circ W$: (173/270 hits) → 64.07 %

Παρατηρούμε οτι το ποσοστό ευστοχίας του $L \circ V$ transducer μειώνεται σημαντικά μετά τη σύνθεσή του με τον W . Αυτό συμβαίνει, διότι ο W δίνει βάρος στις λέξεις που αποδέχεται ανάλογα με τη συχνότητα εμφάνισής τους στο gutenber corpus, γεγονός που συνεπάγεται πως πολλές λέξεις που είναι οι πρακτικά σωστές διορθώσεις των λέξεων εισόδου δεν παράγονται από τον $L \circ V \circ W$, γιατί είναι πολύ σπάνιες και δεν προτιμώνται έναντι άλλων πολύ συχνότερων (όπως η "and" και η "the").

Στη συνέχεια, ο E "προτείνει" διορθώσεις που είναι πιο συχνές (πρότερη γνώση από το αρχείο wiki.txt που περιέχει συχνά λάθη χρηστών) και, ως εκ τούτου, ο $E \circ V$ έχει μεγαλύτερο accuracy από τον $L \circ V$.

Τέλος, η σύνθεση του $E \circ V$ με τον W , ρίχνει το accuracy του $E \circ V$, για τον ίδιο λόγο που η σύνθεση του $L \circ V$ με τον W , ρίχνει το accuracy του $L \circ V$, σε μικρότερο, ωστόσο, βαθμό, καθώς στην πρώτη περίπτωση οι ακμές των μεταβάσεων - διορθώσεων του E δεν είναι ισοβαρείς, όπως του L (που έχουν όλες βάρος 1 πλην της no edit).

2 Μέρος 2: Εξοικείωση με το W2V

Στο δεύτερο μέρος ασχολούμαστε με το Word2Vec μοντέλο. Σκοπός είναι να εκπαιδεύσουμε ένα μοντέλο της δημιουργίας μας πάνω στα Gutenberg κείμενα, το οποίο και θα συγχρίνουμε με ένα προεκπαιδευμένο μοντέλο της Google, σε πρώτη φάση διαισθητικά, σύμφωνα με την προσωπική μας αντίληψη. Επιπλέον, επιθυμούμε να κατασκευάσουμε έναν ταξινομητή συναισθήματος, αξιοποιώντας δεδομένα από IMDB σχόλια ταινιών, προκειμένου να ταξινομήσουμε τις κριτικές σε θετικές και αρνητικές ως προς το συναίσθημα. Το πείραμα αυτό πρόκειται για μια εμπειρική σύγκριση των δύο αναπαραστάσεων.

2.1 Βήμα 12: Εξαγωγή αναπαραστάσεων word2vec

Σε αυτό το βήμα, εστιάζουμε στα word2vec embeddings, δηλαδή σε αναπαραστάσεις που κωδικοποιούν σημασιολογικά χαρακτηριστικά μιας λέξης με βάση την υπόθεση ότι “λέξεις με παρόμοιο νόημα εμφανίζονται σε παρόμοια συγκείμενα”. Για τη δημιουργία των embeddings, χρησιμοποιούμε ένα νευρωνικό δίκτυο με ένα layer, το οποίο καλείται να προβλέψει μια λέξη με βάση το context της (CBOW Model).

Αφού διαβάσουμε πάλι το gutenberg corpus σε μια λίστα από tokenized προτάσεις, χρησιμοποιούμε το python script “w2v_train.py” για την εκπαίδευση του νευρωνικού δικτύου. Ειδικότερα, εκπαιδεύουμε word2vec embeddings διάστασης 100, χρησιμοποιώντας window = 5, epochs = 1000 και min_word_count = 10. Αποθηκεύουμε το μοντέλο που εκπαιδεύσαμε στο αρχείο ”gutenberg_w2v.100d.model”.

Στη συνέχεια, χρησιμοποιώντας cosine similarity στο words embeddings model, βρίσκουμε για τις παρακάτω λέξεις του λεξικού τις 3 σημασιολογικά κοντινότερες. Σημειώνεται ότι όσο πιο κοντά στη μονάδα είναι η επιστρεφόμενη τιμή, τόσο περισσότερο σημασιολογικά κοντινότερη είναι η λέξη:

- | | |
|--|---|
| <ul style="list-style-type: none">• book<ul style="list-style-type: none">[('chronicles', 0.5085127353668213), ('prophet', 0.4747866988182068), ('epistle', 0.47408944368362427)]• bible<ul style="list-style-type: none">[('scoresby', 0.5652458667755127), ('turk', 0.5601562857627869), ('theme', 0.5524604320526123)] | <ul style="list-style-type: none">• water<ul style="list-style-type: none">[('wood', 0.6571595072746277), ('ashes', 0.6310352683067322), ('smoke', 0.5910261273384094)]• bank<ul style="list-style-type: none">[('shelf', 0.8117345571517944), ('floor', 0.8082176446914673), ('top', 0.7994749546051025)] |
|--|---|

Παρατηρούμε ότι τα αποτελέσματα δεν είναι τόσο ποιοτικά όσο θα θέλαμε. Ενδεικτικά, η σημασιολογικά κοντινότερες λέξεις των περισσότερων λέξεων που εξετάσαμε δεν είναι νοηματικά παρεμφερείς (πχ water – wood), με μοναδική ίσως εξαίρεση τη λέξη book (σημασιολογικά κοντινότερη λέξη: chronicles). Αλλάζοντας το μέγεθος του παραθύρου/εποχών δεν φαίνεται να αλλάζει αισθητά το αποτέλεσμα, καθώς το cosine similarity σχετίζεται περισσότερο με orientation/angle και όχι με το length των vectors. To context μέσα στο οποίο εμφανίζονται οι λέξεις, δηλαδή, παραμένει ίδιο.

Προκειμένου, λοιπόν, να εξάγουμε ποιοτικότερα αποτελέσματα, θα χρειαστεί να εκπαιδεύσουμε το μοντέλο μας σε ένα μεγαλύτερο corpus, ώστε να έχουμε και περισσότερες προτάσεις για την εξαγωγή συμπερασμάτων.

Ακολούθως, χρησιμοποιούμε τα word2vec embeddings που εκπαιδεύσαμε για να εκφράσουμε σημασιολογικές αναλογίες μεταξύ λέξεων. Ειδικότερα, παίρνουμε τα ακόλουθα αποτελέσματα:

- $v1 = \text{girls} - \text{queen} + \text{kings} = \text{nations}$
- $v2 = \text{taller} - \text{tall} + \text{good} = \text{worse}$
- $v3 = \text{france} - \text{paris} + \text{london} = \text{england}$

Η μόνη σημασιολογική αναλογία που φαίνεται να είναι επιτυχής είναι η τρίτη, ενώ οι δύο πρώτες είναι λιγότερο πετυχημένες.

Προκειμένου να κάνουμε τη σύγκριση με το δικό μας μοντέλο, φορτώνουμε τα προεκπαίδευμένα GoogleNews vectors με χρήση του script google.py και επαναλαμβάνουμε τα ερωτήματα με cosine similarity και σημασιολογικές αναλογίες. Παίρνουμε τα εξής αποτελέσματα:

- bible
 - [('Bible', 0.7367782592773438), ('scripture', 0.5697901844978333), ('New_Testament', 0.5638793706893921)]
- book
 - [('tome', 0.7485830783843994), ('books', 0.7379177808761597), ('memoir', 0.730292797088623)]
- bank
 - [('banks', 0.7440758943557739), ('banking', 0.690161406993866), ('Bank', 0.6698697805404663)]
- water
 - [('potable_water', 0.6799106597900391), ('Water', 0.6706870794296265), ('sewage', 0.6619377136230469)]

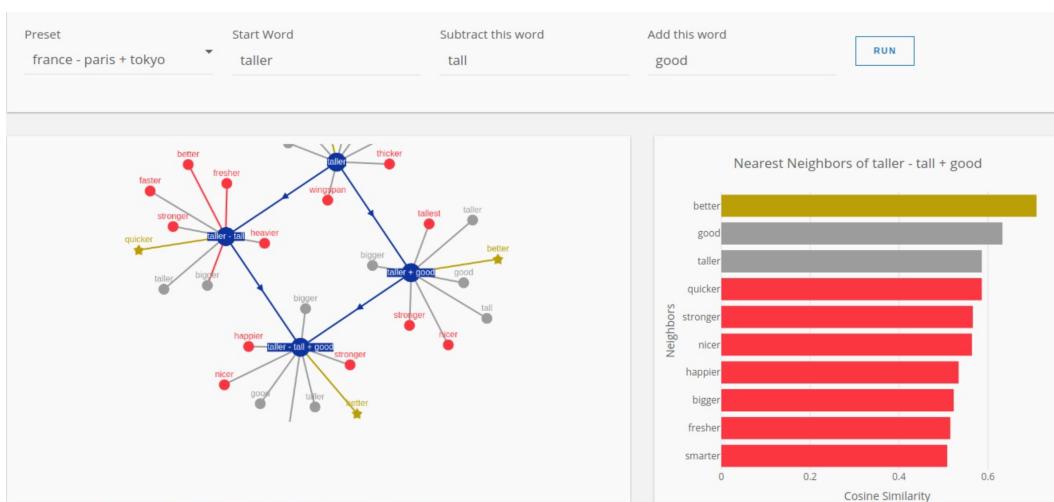
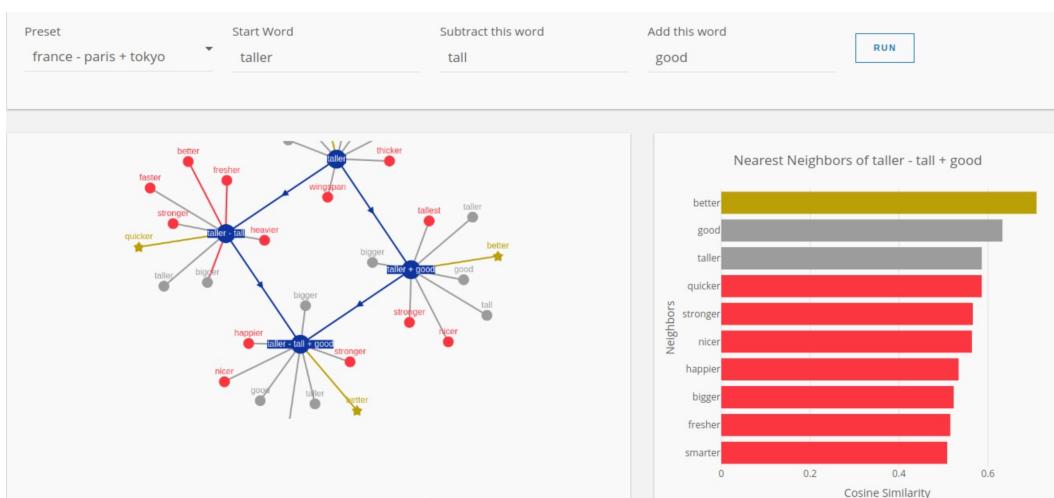
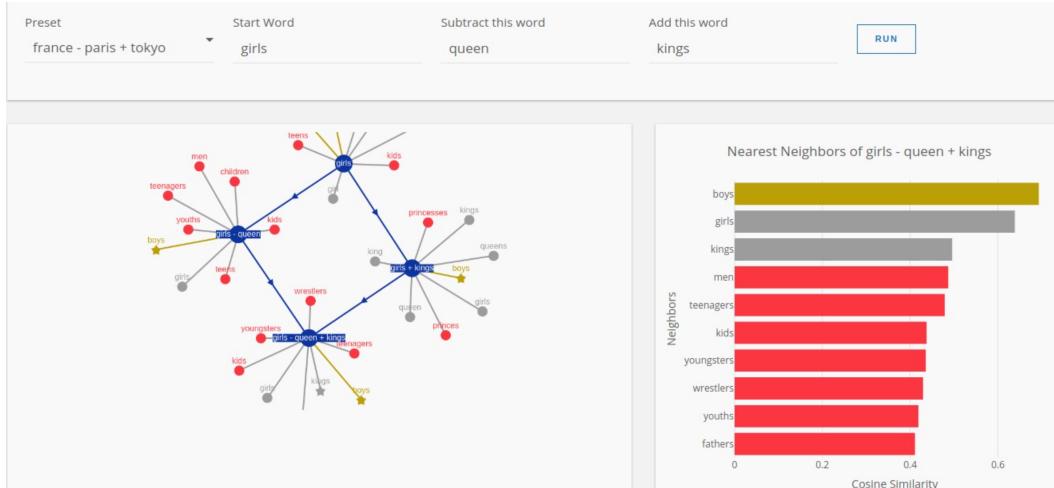
Σημασιολογική Αναλογία:

- $v1 = \text{girls} - \text{queen} + \text{kings} = \text{boys}$
- $v2 = \text{taller} - \text{tall} + \text{good} = \text{better}$
- $v3 = \text{France} - \text{Paris} + \text{London} = \text{Britain}$

Προφανώς, τα προεκπαίδευμένα μοντέλα της Google δίνουν ποιοτικότερα αποτελέσματα από τα δικά μας embedding vectors, σημειώνοντας επιτυχία και στο cosine similarity, αλλά και στις σημασιολογικές αναλογίες.

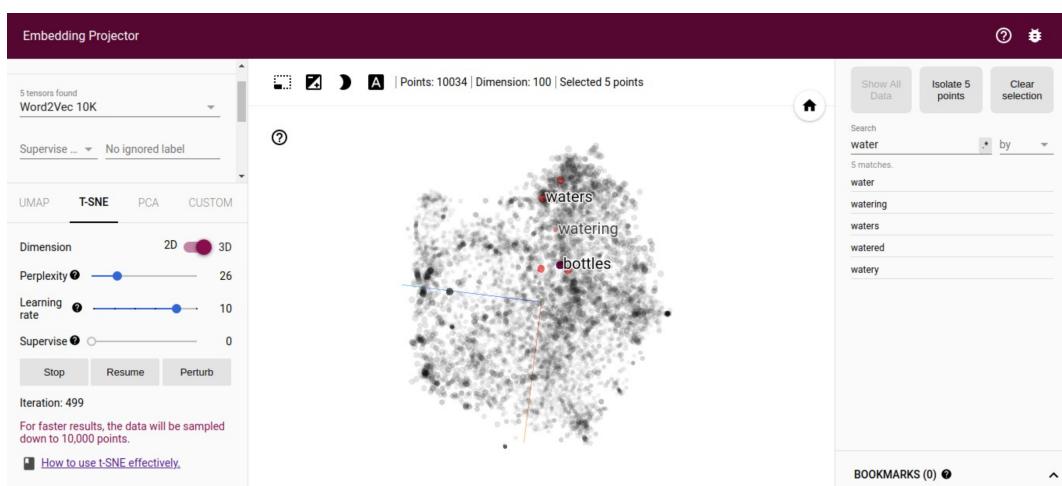
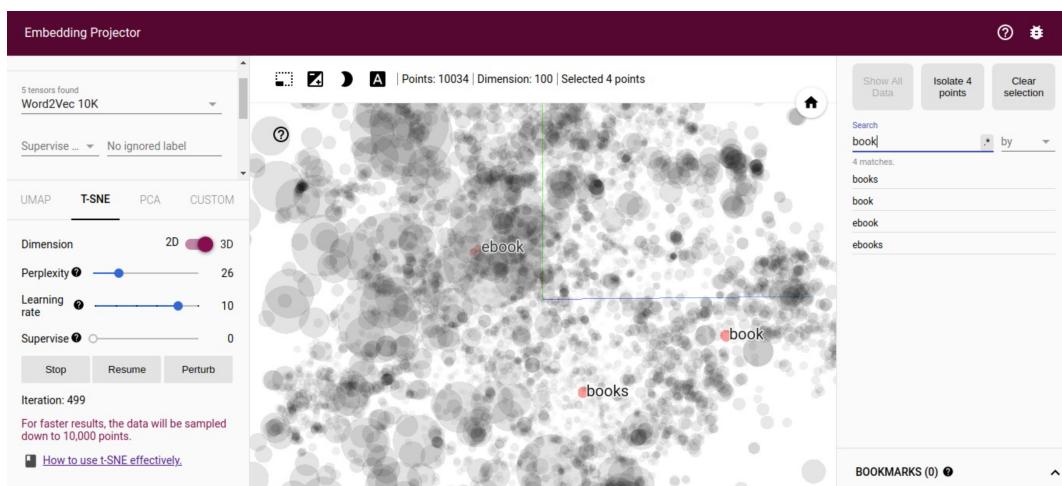
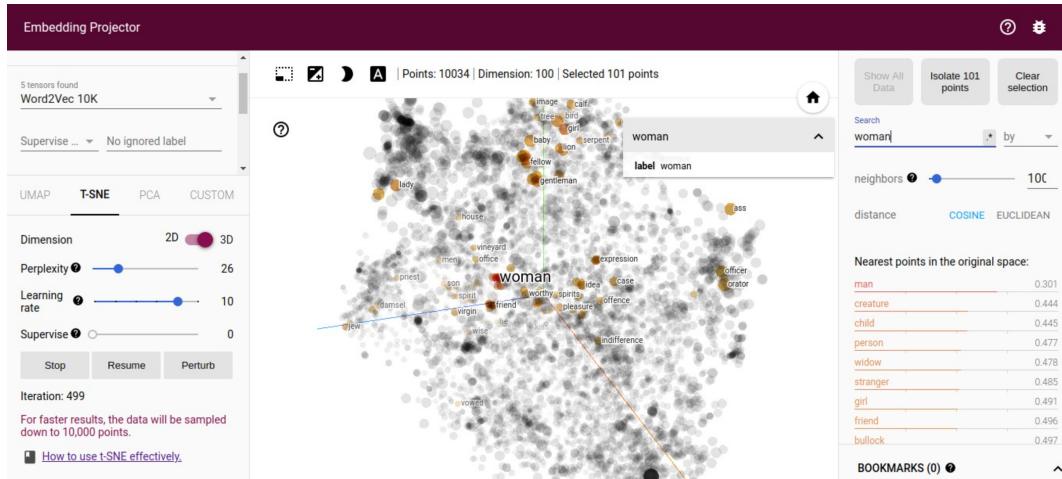
2.2 Βήμα 13: Οπτικοποίηση των word embeddings

Πειραματίζόμαστε με το εργαλείο που δίνεται για visualizations πάνω στις σημασιολογικές ερμηνείες του προηγούμενου ερωτήματος. Βλέπουμε πως μέσω διανυσματικών πράξεων περιηγούμαστε στον σημασιολογικό χώρο και βρίσκουμε τα αναμενόμενα αποτελέσματα:



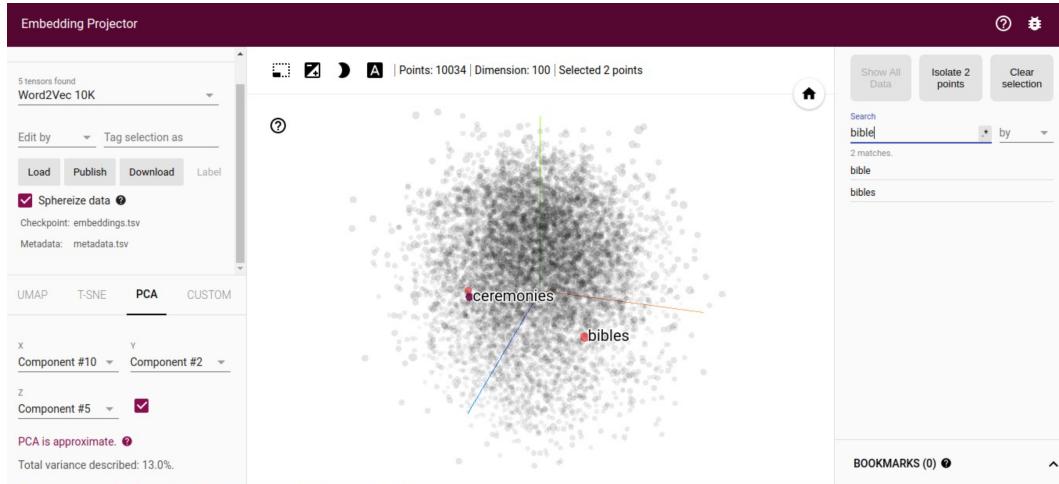
Στη συνέχεια, αποθηκεύουμε τα word embeddings από το μοντέλο που εκπαιδεύσατε στο Βήμα 12 σε ένα tab separated αρχείο με όνομα embeddings.tsv και τις αντίστοιχες λέξεις σε ένα αρχείο με όνομα metadata.tsv. Φορτώνουμε τα .tsv αρχεία στο Embedding Projector.

T - SNE:



Παραπάνω φαίνεται ο σημασιολογικός χώρος που προκύπτει έπειτα από εφαρμογή T-SNE στα δεδομένα μας με στόχο τη μείωση της διαστατικότητας τους. Παρατηρούμε ότι σημασιολογικά κοντινές λέξεις δεν φαίνεται να βρίσκονται τόσο κοντά μεταξύ τους (πχ Book – ebook), ενώ κοντά σε μια λέξη φαίνεται να εμφανίζονται λέξεις που είναι πιθανότερο να εμφανιστούν σε κοινό context (πχ bottles – water).

PCA:



Αντίστοιχα αποτελέσματα βλέπουμε και για τη χρήση PCA για τη μείωση διαστατικότητας. (Για παράδειγμα, λέξη Bible έχει κοντά της τη λέξη ceremonies που είναι νοηματικά παρεμφερής, ενώ απέχει περισσότερο από τη λέξη bibles που είναι σημασιολογικά ισοδύναμη). Σημειώνεται ότι ο σημασιολογικός χώρος αλλάζει, καθώς τροποποιούμε τη διασπορά των τριών components.

2.3 Βήμα 14: Ανάλυση συναισθήματος με word2vec embeddings

Στο τελευταίο βήμα της άσκησης καλούμαστε να αναλύσουμε συναισθήματα με word2vec embeddings, χρησιμοποιώντας δεδομένα από σχόλια για ταινίες στην ιστοσελίδα IMDB.

Αρχικά κατεβάζουμε τα δεδομένα από τον φάκελο train (5000 positive & 5000 negative κριτικές), προβαίνουμε στην προεπεξεργασία τους με τις έτοιμες συναρτήσεις που δίνονται και δημιουργούμε το train_corpus με τα αντίστοιχα train_labels για τη δημιουργία του classifier. Ομοίως, διαβάζουμε από τον φάκελο test και ακολουθώντας την ίδια διαδικασία δημιουργούμε τα test_corpus και test_labels.

Στη συνέχεια, χρησιμοποιούμε τα embeddings που εκπαιδεύσαμε για τη δημιουργία neural bag of words, δηλαδή διανυσματικές αναπαραστάσεις μιας πρότασης που προκύπτουν από το μέσο όρο των embeddings των λέξεων που την αποτελούν. Αυτό επιτυγχάνεται με τη συνάρτηση extract_nbows(corpus, model) την οποία εφαρμόζουμε τόσο στο train_corpus, όσο και στο test_corpus.

Στη συνέχεια, αφού κανονικοποιήσουμε τα δύο corpora, ζεκινάμε την εκπαίδευση του Logistic Regression Classifier εισάγοντας το train_corpus στη συνάρτηση train_sentiment_analysis(). Στο τέλος, αξιολογούμε τον classifier μας με χρήση του test_corpus στην είσοδο της συνάρτησης evaluate_sentiment_analysis. Το ποσοστό ευστοχίας που λαμβάνουμε είναι:

Accuracy score: 73.54 %

Το ποσοστό ευστοχίας είναι αρκετά ικανοποιητικό για έναν Logistic Regression Classifier, καθώς περιμέναμε ευστοχία της τάξης 70-80

Προκειμένου να βελτιώσουμε το ποσοστό ευστοχίας του ταξινομητή, θα μπορούσαμε να κανονικοποιήσουμε τα δεδομένα εισόδου μας (όπως και κάναμε), να πραγματοποιήσουμε regularization ή ακόμα να επιλέξουμε κάποιο άλλο μοντέλο ταξινομητή (πχ MLPClassifier ή GaussianMixture).

Αντιστοίχως, αν εφαρμόσουμε την ίδια διαδικασία στα προεκπαιδευμένα GoogleNews vectors αντί για τα δικά μας embeddings, λαμβάνουμε το εξής αποτέλεσμα ευστοχίας:

Accuracy score: 85.23 %

το οποίο είναι κάπως καλύτερο (τάξης 80 %) από αυτό του δικού μας μοντέλου, γεγονός που οφείλεται στο ότι το μοντέλο της Google έχει εκπαιδευτεί με χρήση ενός αρκετά μεγαλύτερου corpus.

References

- [1] [J&M] D. Jurafski, J. H. Martin. *"Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics."* Prentice-Hall.
- [2] Wikipedia, Levenshtein Distance
https://en.wikipedia.org/wiki/Levenshtein_distance.
- [3] Wikipedia, Frequency Lists
https://en.wiktionary.org/wiki/Wiktionsary:Frequency_lists.
- [4] Jay Alammar. *"The Illustrated Word2vec".*
- [5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffrey Dean. *"Distributed Representations of Words and Phrases and their Compositionality".*