Επεξεργασία Φωνής και Φυσικής Γλώσσας

1η εργαστηριακή άσκηση
Εισαγωγή στις γλωσσικές αναπαραστάσεις

Ηλίας Κουμουκέλης, el18157
Νικόλαος Παγώνας, el18175

Μέρος 1: Κατασκευή ορθογράφου

Στο script `part1.py`, που βρίσκεται στον φάκελο `lab1/`, περιέχεται η υλοποίηση των βημάτων 1-10. Σε αυτό χρησιμοποιούνται ορισμένα από τα αρχεία του φακέλου `scripts/`. Κάποια από τα αρχεία αυτά τα χρησιμοποιήσαμε ως έχουν, ενώ σε κάποια κάναμε μικρές αλλαγές/συμπληρώσεις.

Να σημειωθούν τα εξής:

- Το `part1.py` υλοποιεί κάποιες χρονοβόρες διαδικασίες, όπως η αξιολόγηση των ορθογράφων και η δημιουργία αρχείου με όλα τα edits. Γι' αυτόν το λόγο, στην πρώτη γραμμή του script ορίζεται η παράμετρος `I_AM_PATIENT`. Όταν έχει την τιμή `False`, οι χρονοβόρες διαδικασίες παραλείπονται, και ο χρόνος εκτέλεσης είναι μερικά δευτερόλεπτα. Αντίθετα, όταν έχει την τιμή `True`, τότε ο χρόνος εκτέλεσης είναι ~50 λεπτά (στα δικά μας μηχανήματα). Έτσι μπορεί κάποιος να επιλέξει αν θα εκτελεστούν οι χρονοβόρες διαδικασίες που αναφέρθηκαν.
- Το `part1.py` πρέπει να εκτελεστεί μέσα από τον φάκελο `lab1/`, δηλαδή η εντολή `pwd` πρέπει να τυπώνει:

  ```
  $ pwd

  /path/to/lab1/
  ```

  ώστε να μην υπάρξουν θέματα κατά την εύρεση των αρχείων που διαχειρίζεται το `part1.py`.

Βήμα 1: Κατασκευή corpus

α)

Αρχικά αφού τρέξαμε το script που κατέβαζει το gutenberg corpus παρατηρήσαμε ότι κάνει μια προεπεξεργασία στα δεδομένα. Συγκεκριμένα:

- Απομονώνει κάθε γραμμή του αρχείου ξεχωριστά.

- Αφαιρεί τα επιπλέον κενά, κάνει όλους τους χαρακτήρες lowercase και μετατρέπει φράσεις όπως ”I'm” στην πλήρη μορφή τους (`"I am"`).
- Διαγράφει τους χαρακτήρες που δεν είναι γράμματα ή κενά.
- Δημιουργεί έναν πίνακα με στοιχεία τις γραμμές του αρχείου, όπου κάθε γραμμή είναι ένα string
- Αγνοεί τις κενές συμβολοσειρές
- Τέλος γράφει όλα τα στοιχεία του πίνακα που έμειναν από την παραπάνω διαδικασία στο stdout.

Μία περίπτωση στην οποία θα θέλαμε να διατηρήσουμε τα σημεία στίξης είναι αν θέλουμε να διορθώσουμε αρκτικόλεξα, οπότε εκεί δεν θα επιθυμούσαμε τόσο επιθετικό preprocessing.

β)

Με την επέκταση του corpus:

- Εμπλουτίζουμε το data set μας με περισσότερα στοιχεία (λέξεις, ιδιωματισμούς, ορολογίες κλπ.) από άλλες περιοχές. Έτσι για παράδειγμα, το corpus μπορεί να συμπεριλάβει και λέξεις που συναντώνται συχνά σε συγκεκριμένα γνωστικά αντικείμενα, αλλά είναι σπάνιες εκτός αυτών.
- Μεγαλώνει ο δειγματικός χώρος και επομένως αυξάνεται η δυνατότητα κατάταξης της συχνότητας εμφάνισης των λέξεων, διορθώνοντας πιθανές ανωμαλίες. Για παράδειγμα, ένα βιβλίο όπως ο Tom Sawyer θα περιλαμβάνει πολλές φορές την λέξη ”Tom”, όμως αν συμπεριλάβουμε μεγάλο αριθμό βιβλίων, η συχνότητα εμφάνισης της λέξης ”Tom” σταματά να είναι τόσο έντονη σε σχέση με άλλες λέξεις που είναι συχνές σε όλα τα βιβλία, όπως η λέξη ”the”.

Βήμα 2: Κατασκευή λεξικού

α)

Δημιουργούμε ένα dictionary της μορφής `{token: number of occurrences}`

β)

Φιλτράρουμε το λεξικό, απορρίπτοντας τις λέξεις που έχουν εμφανιστεί κάτω από 5 φορές. Αυτό το βήμα χρειάζεται, διότι αλλιώς το FST που θα φτιάχναμε θα λάμβανε υπόψιν του πάρα πολλές λέξεις με αποτέλεσμα τόσο ο αποθηκευτικός χώρος που χρειάζεται το FST, όσο και ο χρόνος που απαιτούν πράξεις όπως η βελτιστοποίηση του FST και η διόρθωση μιας λέξης θα αύξαναν απαγορευτικά, ειδικά αν αναλογιστούμε το πόσο μικρό όφελος προσφέρουν τόσο σπάνιες λέξεις που κατά πάσα πιθανότητα δεν θα προτιμηθούν ποτέ από τον ορθογράφο μας.

γ)

Αποθηκεύουμε το λεξικό που φτιάξαμε στο αρχείο `vocab/words.vocab.txt` σε δύο tab separated στήλες, όπου η πρώτη στήλη περιέχει τα tokens και η δεύτερη στήλη τους αντίστοιχους αριθμούς εμφανίσεων.

Βήμα 3: Δημιουργία συμβόλων εισόδου/εξόδου

α)

Δημιουργούμε το αρχείο `vocab/chars.syms` στο οποίο αποθηκεύουμε στην πρώτη στήλη το κάθε γράμμα και στην δεύτερη έναν αύξοντα ακέραιο αριθμό-index (το `<eps>` αντιστοιχεί στο 0 και οι υπόλοιποι χαρακτήρες αντιστοιχούν στον ASCII κωδικό τους), με τη βοήθεια των συναρτήσεων `lowercase_to_index()` και `create_lowercase_to_index_file()`.

β)

Αντίστοιχα κάνουμε το ίδιο για τα tokens με τη βοήθεια της συνάρτησης `create_word_to_index_file()`, και αποθηκεύουμε το αποτέλεσμα στο αρχείο `vocab/words.syms`. Να σημειωθεί ότι και εδώ χρειάζεται το `<eps>` να αντιστοιχεί στο 0.

Βήμα 4: Κατασκευή μετατροπέα edit distance

α)

Φτιάχνουμε έναν μετατροπέα L βασισμένο στην απόσταση Levenshtein με τη βοήθεια της συνάρτησης `create_L_transducer_file()`. Κάθε μετατροπή χαρακτήρα έχει το αντίστοιχο κόστος:

1) Κάθε χαρακτήρας αντιστοιχίζεται στον εαυτό του με βάρος 0 (**καμία αλλαγή**).

2) Κάθε χαρακτήρας αντιστοιχίζεται στο `<eps>` με βάρος 1 (**διαγραφή**).

3) Το `<eps>` αντιστοιχίζεται σε κάθε χαρακτήρα με βάρος 1 (**εισαγωγή**).

4) Κάθε χαρακτήρας αντιστοιχίζεται σε κάθε άλλο χαρακτήρα με βάρος 1.

Αν πάρουμε το shortest path, αυτός ο μετατροπέας θα παράγει τις λέξεις στις οποίες μετασχηματίζεται η λέξη εισόδου, με το μονοπάτι μετασχηματισμών να έχει ελάχιστο κόστος, δηλαδή θα αφήσει την λέξη απαράλλαχτη με κόστος 0.

β)

Η συνάρτηση `create_L_transducer_file()` αναλαμβάνει και την αποθήκευση του αρχείου `fsts/L.fst`.

γ)

Με την βοήθεια της `fstcompile` δημιουργούμε το αρχείο `fsts/L.binfst`.

δ)

Πιθανά edits που θα μπορούσαμε να συμπεριλάβουμε είναι:

- Να λάβουμε υπόψιν λάθη που κάνουν συχνά άτομα με δυσλεξία (πχ $3 \to \varepsilon$). Αυτό φυσικά προϋποθέτει να έχουμε συμπεριλάβει χαρακτήρες όπως το 3 και το ε.
- Να λάβουμε υπόψιν edits της μορφής `r` $\to$ `rr` (πχ στη λέξη occurrence)

ε)

Επειδή ορισμένα πλήκτρα στο πληκτρολόγιο είναι πιο κοντά μεταξύ τους, μπορούμε να τροποποιήσουμε τα βάρη έτσι ώστε τα edits που αφορούν κοντινά πλήκτρα να προτιμώνται έναντι αυτών που τα πλήκτρα είναι μακρύτερα. (πχ το q και το w είναι πολύ κοντύτερα απ' ό,τι το q και το p, επομένως είναι λογικό το edit q → w να κοστίζει λιγότερο απ' ό,τι το edit q → p)

ζ)

Χρησιμοποιούμε την `fstdraw` για να σχεδιάσουμε τον L. Χρησιμοποιούμε ένα πολύ μικρό υποσύνολο των χαρακτήρων για εύκολη οπτικοποίηση.

Βήμα 5: Κατασκευή αποδοχέα λεξικού

α)

Με τη χρήση της συνάρτησης `create_V_acceptor_file()` κατασκευάζουμε έναν αποδοχέα V με μία αρχική κατάσταση και μηδενικά βάρη στις ακμές του. Αυτός ο αποδοχέας απλά αποδέχεται μια λέξη αν αυτή ανήκει στο λεξικό.

β)

Καλούμε τις `fstrmepsilon`, `fstdeterminize` και `fstminimize` ώστε να βελτιστοποιήσουμε το μοντέλο. Οι εντολές αυτές εκτελούν τις ακόλουθες πράξεις (κάθε φορά το FST που προκύπτει είναι ισοδύναμο):

- `fstrmepsilon`: Αφαιρεί τις ε-μεταβάσεις (όπου είσοδος και έξοδος είναι ε).
- `fstdeterminize`: Κάνει ντετερμινιστικό τον transducer, δηλαδή καμία κατάσταση δεν έχει δύο μεταβάσεις με την ίδια είσοδο.
- `fstminimize`: Ελαχιστοποιεί τον transducer, δηλαδή φτιάχνει ένα ισοδύναμο transducer με τον ελάχιστο δυνατό αριθμό καταστάσεων.

γ)

Αποθηκεύουμε το αρχείο σε Openfst text format με όνομα `fsts/V.fst`.

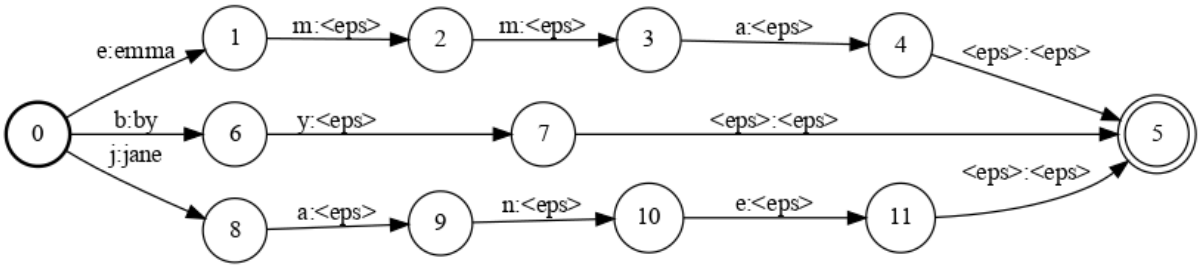
δ)

Με την `fstcompile` κάνουμε compile τον αποδοχέα και δημιουργούμε το αρχείο `fsts/V.binfst`.

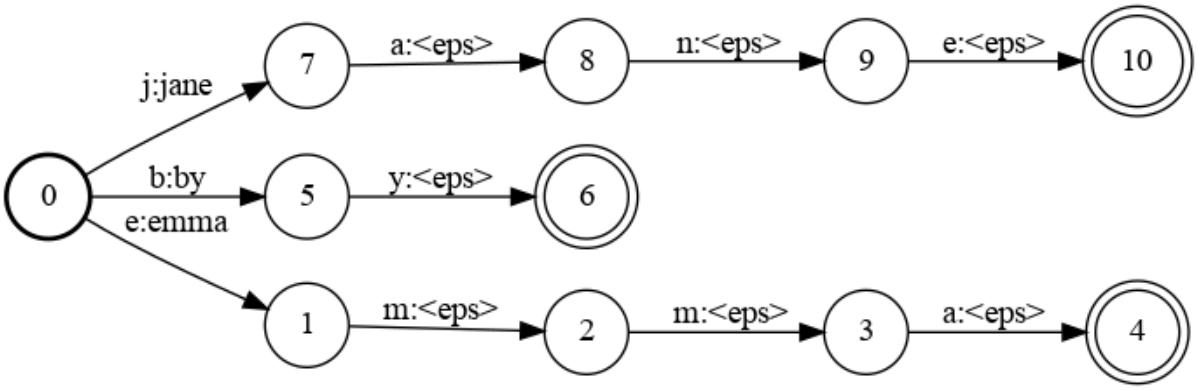
ε)

Χρησιμοποιούμε την `fstdraw` για να σχεδιάσουμε τον V, χρησιμοποιώντας ένα μικρό υποσύνολο των λέξεων για εύκολη οπτικοποίηση. Συγκεκριμένα, σχεδιάζουμε:

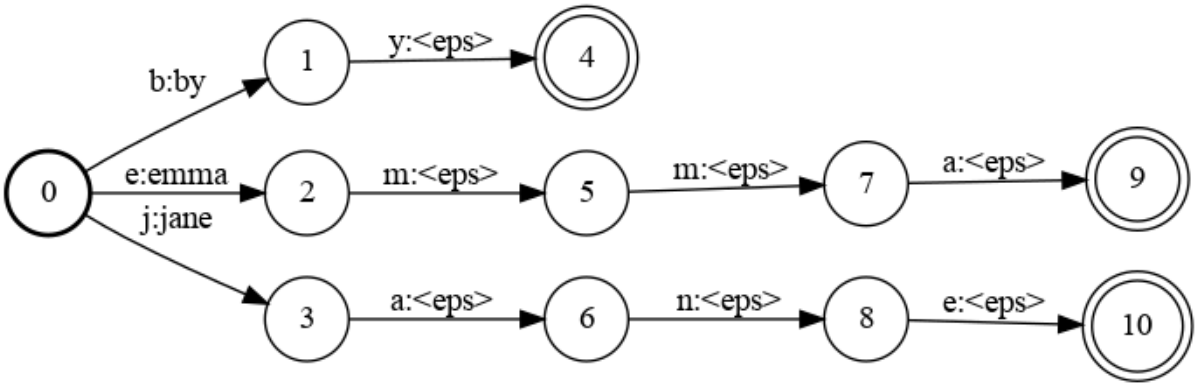
- Τον V πριν εφαρμοστεί οποιαδήποτε βελτιστοποίηση:



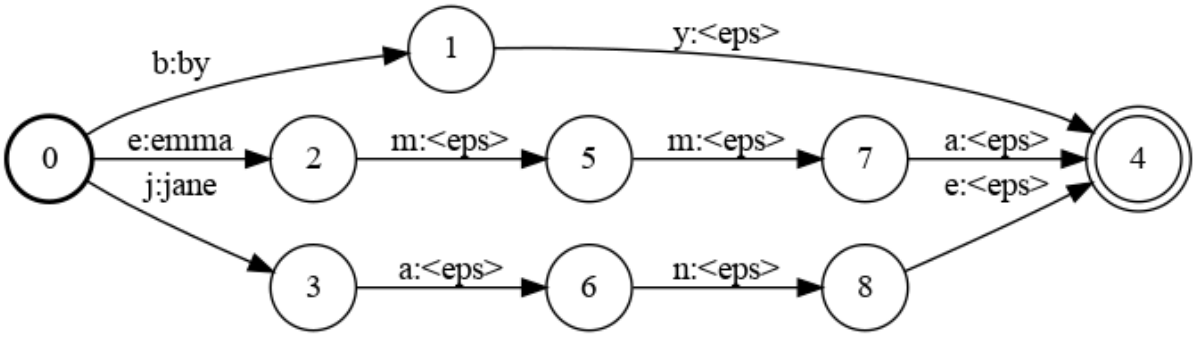
- Ύστερα από την `fstrmepsilon`:



- Ύστερα και από την `fstdeterminize` (δεν υπάρχει κάποια αλλαγή):



- Τέλος, ύστερα και από την `fstminimize`:



Βήμα 6: Κατασκευή ορθογράφου

α)

Συνθέτουμε τον L με τον V, παράγοντας έναν edit distance spell checker S, ο οποίος λειτουργεί με μοναδικό κριτήριο να ελαχιστοποιήσει το κόστος των edits που χρειάζονται ώστε μία λανθασμένη λέξη να μετατραπεί σε μία σωστή (που ανήκει δηλαδή στο λεξικό). Φροντίζουμε να καλέσουμε την `fstarcsort` στις εξόδους του L και στις εισόδους του V ώστε να μην υπάρξουν προβλήματα κατά τη σύνθεση.

1) Στην περίπτωση που τα edits είναι ισοβαρή, ουσιαστικά ο transducer έχει ως σκοπό να κάνει το μικρότερο πλήθος edits ώστε να φτάσει σε μία έγκυρη λέξη.

2) Στην περίπτωση που τα edits έχουν διαφορετικά βάρη, μπορούμε να εισάγουμε περισσότερη ”ευφυΐα” στον μετατροπέα, δίνοντας μικρότερο βάρος σε πιθανότερες διορθώσεις, ώστε να προτιμηθούν.

β)

Πιθανές προβλέψεις του min edit spell checker:

- `cit` → `cat` (1 αλλαγή), `it` (1 διαγραφή), `cite` (1 προσθήκη)
- `cwt` → `cut`, `cat` (1 αλλαγή)

Βήμα 7: Δοκιμή ορθογράφου

α)

Χρησιμοποιούμε το `lab1/data/spell_test.txt` για το evaluation.

β)

Χρησιμοποιούμε τη συνάρτηση `test_spell_checker()` η οποία καλεί το `lab1/scripts/predict.sh` για τις 20 πρώτες λέξεις του test set. Ενδεικτικά:

Wrong word	Corrected word	Actual word
contenpted	contented	contented
contende	contend	contented
begining	beginning	beginning
problam	problem	problem
proble	problem	problem
dirven	driven	driven
exstacy	exactly	ecstasy
ecstacy	ecstasy	ecstasy
guic	guil	juice
juce	juice	juice
localy	local	locally
compair	compare	compare
pronounciation	pronouncing	pronunciation
transportibility	respectability	transportability
miniscule	mince	minuscule

Όσον αφορά το `predict.sh`, αυτό δημιουργεί με τη χρήση του `mkfstinput.py` ένα FST που αποδέχεται τη λανθασμένη λέξη εισόδου και το συνθέτει με τον spell checker. Βρίσκει το shortest path στη σύνθεση αυτή και αφαιρεί τις ε-μεταβάσεις. Η fstprint τυπώνει τις καταστάσεις που περιλαμβάνει το shortest path μαζί με την έξοδό τους, οπότε επιλέγοντας μόνο τις εξόδους (με την εντολή cut) και ύστερα φιλτράροντας ε-εξόδους και την γραμμή της τελικής κατάστασης, μένει μόνο η λέξη εξόδου, που είναι και η διορθωμένη.

Βήμα 8: Υπολογισμός κόστους των edits

Σε αυτό το βήμα χρησιμοποιούμε το corpus `lab1/data/wiki.txt` που περιέχει συχνά ορθογραφικά λάθη της Wikipedia, προκειμένου να δώσουμε διαφορετικό κόστος σε κάθε edit operation. Να σημειωθεί ότι διαγράψαμε 3 γραμμές από το `wiki.txt` οι οποίες περιείχαν χαρακτήρες που δεν ήταν πεζά αγγλικά γράμματα (π.χ. `a.k.a.`). Τρέχουμε το `scripts/word_edits.sh` που εκτελεί τα βήματα (α), (β) και (γ):

α)

Αρχικά χρησιμοποιούμε ένα μόνο παράδειγμα (abandonned → abandoned). Δημιουργούμε έναν αποδοχέα M για τη λανθασμένη λέξη (abandonned) και ένα μετατροπέα N για τη διορθωμένη (abandoned).

β)

Δημιουργούμε τη σύνθεση MLN.

γ)

Εκτελούμε την `fstshortestpath` ακολουθούμενη από την `fstprint` με την σύνταξη που ζητείται, παράγοντας ένα μονοπάτι ελάχιστου κόστους για τη διόρθωση της λανθασμένης λέξης. Με την `grep` αγνοούμε τις γραμμές με μηδενικό κόστος και με την `cut` επιλέγουμε μόνο τα edits.

Εν συντομία, το `word_edits.sh` (με τη βοήθεια του `mkfstinput.py`):

- Δημιουργεί ένα FST M που αποδέχεται τη λανθασμένη λέξη
- Δημιουργεί ένα FST N που αποδέχεται τη σωστή λέξη
- Συνθέτει το M με τον L και στη συνέχεια με το N, παράγοντας το MLN
- Βρίσκει το shortest path πάνω στο MLN
- Φιλτράρει κατάλληλα (με τις εντολές grep και cut) την έξοδο της `fstprint` πάνω στο MLN ώστε να μείνουν μόνο τα edits μη μηδενικού κόστους.

Παραδείγματα εκτέλεσης:

```
$ bash scripts/word_edits.sh helo hello
<eps>     l

$ bash scripts/word_edits.sh mrlow mellow
r         l
<eps>     e

$ bash scripts/word_edits.sh crase increase
<eps>     i
<eps>     e
<eps>     n
```

δ)

Γράφουμε τη συνάρτηση `save_all_edits_to_file()` η οποία εκτελεί την παραπάνω διαδικασία για κάθε ζεύγος λέξεων στο `wiki.txt` και αποθηκεύει όλα τα edits στο αρχείο `lab1/data/edits.txt`.

ε)

Εξάγουμε τη συχνότητα κάθε edit με την συνάρτηση `extract_frequency_to_dict()` και την αποθηκεύουμε σε ένα λεξικό στη μορφή:

```
{(source, target): frequency}
```

στ)

Επαναλαμβάνουμε το βήμα 4, όπου κατασκευάζουμε τον μετατροπέα edit distance, αλλά αυτή τη φορά τα edits έχουν διαφορετικό κόστος μεταξύ τους. Συγκεκριμένα, κάθε edit έχει βάρος ίσο με:

$$-\log(\text{frequency}) = -\log\left(\frac{\text{number of occurrences for the particular edit}}{\text{total number of edits}}\right)$$

Αν κάποιο edit δεν έχει εμφανιστεί στο corpus βάζουμε έναν πολύ μεγάλο αριθμό ως βάρος (σταθερά `INFINITY`). Έτσι προκύπτει ο transducer E.

ζ)

Επαναλαμβάνουμε τα βήματα 6, 7 χρησιμοποιώντας τον transducer E αντί για τον L, δηλαδή κατασκευάζουμε και δοκιμάζουμε τον ορθογράφο EV.
Ενδεικτικά:

Wrong word	Corrected word	Actual word
contenpted	contented	contented
contende	contend	contented
begining	beginning	beginning
problam	problem	problem
proble	problem	problem
dirven	driven	driven
exstacy	exactly	ecstasy
ecstacy	ecstasy	ecstasy
guic	guil	juice
juce	juice	juice
localy	local	locally
compair	compare	compare
pronounciation	pronouncing	pronunciation
transportibility	respectability	transportability
miniscule	mince	minuscule

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

Σε αυτό το βήμα σκοπός είναι να εκμεταλλευτούμε την συχνότητα εμφάνισης των λέξεων, ώστε ο ορθογράφος να προτιμά πιο συχνές λέξεις ως πιο πιθανές.

α)

Χρησιμοποιούμε το λεξικό συχνοτήτων που φτιάξαμε στο Βήμα 2.

β)

Κατασκευάζουμε τον αποδοχέα W που έχει μία κατάσταση και αντιστοιχίζει κάθε λέξη στον εαυτό της με βάρος ίσο με $-\log(\text{frequency})$, όπου frequency είναι η (σχετική) συχνότητα εμφάνισης της αντίστοιχης λέξης.

γ)

Κατασκευάζουμε τον ορθογράφο LVW συνθέτοντας τον L με τον V και ύστερα με το W. Κάνουμε optimize τον W (οι L και V είναι ήδη optimized) με χρήση των `fstrmepsilon`, `fstdeterminize` και `fstminimize`. Για την σύνθεση χρησιμοποιούμε την `fstarcsort` στις εξόδους του αριστερού μέρους και στις εισόδους του δεξιού.

δ)

Με παρόμοιο τρόπο κατασκευάζουμε τον ορθογράφο EVW.

ε)

Αξιολογούμε τον ορθογράφο LVW με την διαδικασία του ερωτήματος 8, και αναφέρουμε ενδεικτικά κάποιες διορθώσεις:

Wrong word	Corrected word	Actual word
contenpted	the	contented
contende	the	contented
begining	beginning	beginning
problam	of	problem
proble	the	problem
dirven	the	driven
exstacy	the	ecstasy
ecstacy	the	ecstasy
guic	the	juice
juce	the	juice
localy	and	locally
compair	and	compare
pronounciation	unto	pronunciation
transportibility	and	transportability
miniscule	the	minuscule

Βλέπουμε ότι σχεδόν όλες οι λέξεις έχουν μετατραπεί σε ”the” και ”and”, ακριβώς επειδή αυτές οι λέξεις είναι πολύ συχνές και υπερνικούν όλα τα άλλα πιθανά μονοπάτια, οδηγώντας σε πολύ χειρότερη επίδοση του ορθογράφου.

στ)

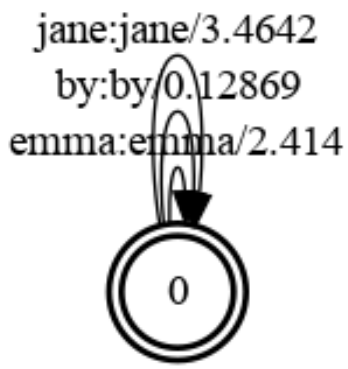
Συγκρίνουμε τα αποτελέσματα του LVW με τον LV. Η έξοδος των μετατροπέων σε κάθε περίπτωση είναι:

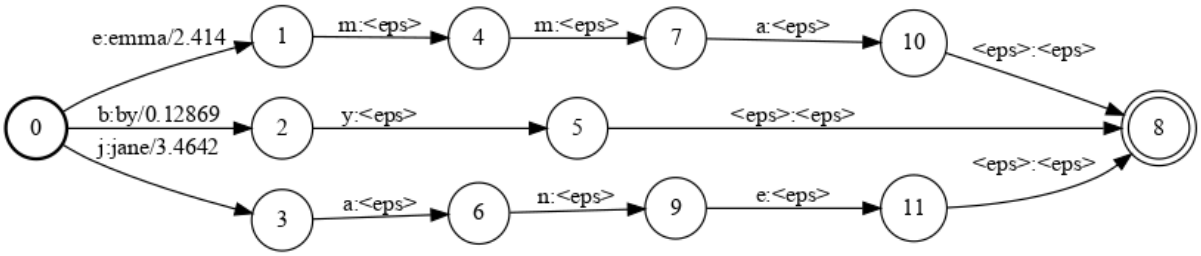
- LVW:
  - `cwt` $\rightarrow$ `the`
  - `cit` $\rightarrow$ `it`
- LV:
  - `cwt` $\rightarrow$ `cut`
  - `cit` $\rightarrow$ `city`

Βλέπουμε ότι ο LVW δείχνει ισχυρή προτίμηση σε συχνά εμφανιζόμενες λέξεις, ακόμα κι αν δεν έχουν καμία σχέση με την είσοδο (π.χ. `cwt` $\rightarrow$ `the`). Αντίθετα, ο LV απλώς προσπαθεί να φτάσει σε μία έγκυρη λέξη με τον μικρότερο αριθμό edits.

ζ)

Χρησιμοποιούμε την `fstdraw` για να σχεδιάσουμε τον W και τον VW. Χρησιμοποιούμε ένα μικρό υποσύνολο των λέξεων για πιο εύκολη οπτικοποίηση.





Βήμα 10: Αξιολόγηση των ορθογράφων

Χρησιμοποιούμε το `run_evaluation.py` για να συγκρίνουμε τα αποτελέσματα των ορθογράφων LV, LVW, EV, EVW. Ως test set χρησιμοποιούμε το αρχείο `data/spell_test.txt`. Έχουμε:

- LV accuracy: ~0.596 = 59.6%
- LVW accuracy: ~0.044 = 4.4%
- EV accuracy: ~0.689 = 68.9%
- EVW accuracy: ~0.641 = 64.1%

Και στις δύο περιπτώσεις η προσθήκη του W ρίχνει την ακρίβεια των ορθογράφων. Ωστόσο, στην πρώτη περίπτωση, όπου όλα τα edits έχουν το ίδιο βάρος, η πτώση της ακρίβειας είναι πολύ σημαντική, σε σημείο που ο ορθογράφος πρακτικά αρχηστεύεται, αφού προτιμώνται κατά πολύ οι συχνές λέξεις. Αντίθετα στη δεύτερη περίπτωση, η πτώση δεν είναι τόσο σημαντική αφού η επίδραση του W έχει μετριαστεί από την στοιχειώδη "ευφυΐα" που προσθέσαμε στα edits, ώστε να έχουν κόστος ανάλογα της συχνότητάς τους.

Μέρος 2: Εξοικείωση με το W2V

Ο κώδικας που υλοποιεί το Μέρος 2 βρίσκεται στο αρχείο `lab1/steps12-14.py` και χρησιμοποιεί το αρχείο `scripts/w2v_sentiment_analysis.py`. Να σημειωθεί ότι πρέπει να εκτελεστεί από τον φάκελο `lab1/` για να μην υπάρξουν προβλήματα με τα ονόματα των αρχείων που επεξεργάζεται.

Βήμα 12: Εξαγωγή αναπαραστάσεων word2vec

α)

Αρχικά επεξεργαζόμαστε το corpus για κάθε πρόταση ξεχωριστά και χρησιμοποιούμε την συνάρτηση `word_tokenize()` έτσι ώστε να έχουμε για κάθε πρόταση tokenized λέξεις.

β)

Έπειτα εκπαιδεύουμε το μοντέλο ωστε να αναπαραστήσουμε κάθε λέξη με ένα διανύσμα 100 διαστάσεων με window=5 για 1000 εποχές.

γ)

Με την βοήθεια της συνάρτησης `most_similar` (που χρησιμοποιεί cosine similarity) βρίσκουμε τις 5 πιο κοινές λέξεις για “bible”, “book”, “bank”, “water”.

1) Τα αποτελέσματα είναι ικανοποιητικά και οι ”συγγενείς” λέξεις που βγαίνουν είναι κοντά εννοιολογικά.

2) Αλλάζουμε τον αριθμό των εποχών σε 10, 100 και 1000.

3) Παρατήρουμε ότι όσο αύξανονται οι εποχές τόσο βελτιώνονται τα αποτελέσματα:

```
#model 10 epoch
similarity(model_10)
```

```
top 5 similar words for: bible [('salem', 0.5701761245727539), ('monarchy', 0.5499080419540405), ('x', 0.534144401
550293), ('chronicles', 0.5331150889396667), ('medes', 0.531923770904541)]

top 5 similar words for: book [('written', 0.6517412662506104), ('chronicles', 0.6446224451065063), ('note', 0.573
5135078430176), ('letter', 0.5544382333755493), ('prophecy', 0.5454708337783813)]

top 5 similar words for: bank [('floor', 0.8167724609375), ('beach', 0.7894853353500366), ('lawn', 0.7636570930480
957), ('shelf', 0.7516579627990723), ('top', 0.7483644485473633)]

top 5 similar words for: water [('waters', 0.6417844891548157), ('butter', 0.5764515399932861), ('brook', 0.570820
8084106445), ('wood', 0.5589873194694519), ('smoke', 0.5568504333496094)]
```

```
#model 100 epoch
similarity(model_100)
```

```
top 5 similar words for: bible [('grecian', 0.5747950077056885), ('salem', 0.5374585390090942), ('dwells', 0.53632
33089447021), ('admah', 0.5149062871932983), ('renowned', 0.5117109417915344)]

top 5 similar words for: book [('chronicles', 0.6792780756950378), ('written', 0.6492640972137451), ('prophecy',
0.5814546942710876), ('volume', 0.5579489469528198), ('letter', 0.5521500706672668)]

top 5 similar words for: bank [('floor', 0.815850019454956), ('lawn', 0.7951639294624329), ('beach', 0.79146766662
59766), ('ridge', 0.7521273493766785), ('hill', 0.7439787983894348)]

top 5 similar words for: water [('waters', 0.6199480891227722), ('wood', 0.6116408109664917), ('ashes', 0.60293328
76205444), ('springs', 0.5738457441329956), ('streams', 0.5656701326370239)]
```

```
#model 1000 epoch
similarity(model)
```

```
top 5 similar words for: bible [('propitious', 0.3643248975276947), ('copyright', 0.3560762405395508), ('absolut
e', 0.3544497489929199), ('preacher', 0.3475169539451599), ('sets', 0.3396170735359192)]

top 5 similar words for: book [('written', 0.5506107807159424), ('chronicles', 0.5242385268211365), ('temple', 0.4
4809606671333313), ('papers', 0.4410388767719269), ('letter', 0.4409632980823517)]

top 5 similar words for: bank [('pool', 0.5303764343261719), ('ridge', 0.4930790960788727), ('floor', 0.4843547344
2077637), ('top', 0.4795888662338257), ('table', 0.4602685272693634)]

top 5 similar words for: water [('waters', 0.6073228120803833), ('wine', 0.5268726348876953), ('river', 0.51152980
3276062), ('liquid', 0.5035266876220703), ('rivers', 0.4922991991043091)]
```

4) Η αύξηση των εποχών είναι μια καλή ιδέα αλλά έχει ένα τέλμα γιατί μπορεί να οδηγήσει σε over fitting. Το καλύτερο είναι να αυξήσουμε τα δεδομένα (πχ. ερώτημα ε)

δ)

Ομοίως με το παραπάνω ερώτημα κάναμε το ίδιο για τις τριπλέτες λέξεων που μας δόθηκαν:

```
similarity_vectors(model)
```

```
girls queen kings
top 5 similar words for  girls - queen + kings [('men', 0.48008933663368225), ('cities', 0.4657207131385803), ('pa
rts', 0.46389126777648926), ('nations', 0.41672438383102417), ('creatures', 0.40624964237213135), ('wives', 0.3935
786783695221), ('people', 0.3871099352836609), ('notions', 0.38323408365249634), ('disasters', 0.382081270217895
5), ('countries', 0.3811590075492859)]

good tall taller
top 5 similar words for  good - tall + taller [('better', 0.41280120611190796), ('best', 0.39290037751197815), ('d
ear', 0.3863082528114319), ('spared', 0.369650661945343), ('please', 0.3676787316799164), ('used', 0.3601299524307
251), ('pleased', 0.357248455286026), ('comfort', 0.35277992486953735), ('anathoth', 0.3519642651081085), ('merr
y', 0.3460538387298584)]

france paris london
top 5 similar words for  france - paris + london [('england', 0.4456145167350769), ('sussex', 0.3989991545677185),
('spot', 0.3951275050640106), ('uppercross', 0.39154985547065735), ('court', 0.38648563623428345), ('inn', 0.38351
306319236755), ('neighbourhood', 0.38284409046173096), ('chamber', 0.3799216151237488), ('atmospheres', 0.37882909
178733826), ('town', 0.3760843873023987)]
```

ε)

Κατεβάζουμε το αρχείο και φορτώνουμε το μοντελο στην μνήμη με limit = 250000

στ, ζ)

Έδω ακολουθούμε την ίδια διαδικασία και παρατηρούμε παρεμφερή, αλλά καλύτερα αποτελέσματα από αυτά των 1000 εποχών. Αυτό είναι λογικό γιατί περιλαμβάνει περισσότερα βιβλία και η κάθε λέξη αποτελείται από 300διάστατα διανύσματα:

```
similarity(model2)#google model similarity for words
```

```
top 5 similar words for: bible [('Bible', 0.736778199672699), ('bibles', 0.6052597761154175), ('Holy_Bible', 0.598
9600419998169), ('scriptures', 0.574568510055542), ('scripture', 0.569790244102478)]

top 5 similar words for: book [('tome', 0.7485830783843994), ('books', 0.7379177808761597), ('memoir', 0.730292797
088623), ('paperback_edition', 0.6868364810943604), ('autobiography', 0.6741527318954468)]

top 5 similar words for: bank [('banks', 0.7440758943557739), ('banking', 0.690161406993866), ('Bank', 0.669869840
1451111), ('lender', 0.6342284679412842), ('banker', 0.6092953681945801)]

top 5 similar words for: water [('potable_water', 0.6799106597900391), ('Water', 0.6706870794296265), ('sewage',
0.6619377136230469), ('groundwater', 0.6588346362113953), ('freshwater', 0.6307883262634277)]
```

```
similarity_vectors(model2)#google model similarity for vectors
```

```
girls queen kings
top 5 similar words for  girls - queen + kings [('boys', 0.7183727025985718), ('men', 0.5000520348548889), ('teena
gers', 0.49677684903144836), ('schoolboys', 0.46822962164878845), ('kids', 0.45648258924484253), ('youngsters', 0.
44871407747268677), ('wrestlers', 0.44103771448135376), ('Boys', 0.4328978657722473), ('youths', 0.431538403034210
2), ('fathers', 0.4184094965457916)]

good tall taller
top 5 similar words for  good - tall + taller [('better', 0.7245720624923706), ('nicer', 0.5474837422370911), ('gr
eat', 0.5431876182556152), ('quicker', 0.5431677103042603), ('bad', 0.5423746109008789), ('stronger', 0.5373172760
009766), ('decent', 0.5306718349456787), ('happier', 0.5232836008071899), ('nice', 0.5104230642318726), ('terrifi
c', 0.5000960826873779)]

france paris london
top 5 similar words for  france - paris + london [('england', 0.5836853981018066), ('europe', 0.5529575347900391),
('european', 0.5125302076339722), ('africa', 0.4936657249927521), ('america', 0.48779499530792236), ('boston', 0.4
788355529308319), ('liverpool', 0.47737205028533936), ('germany', 0.4766533374786377), ('nigeria', 0.4715959429740
906), ('chelsea', 0.46420517563819885)]
```

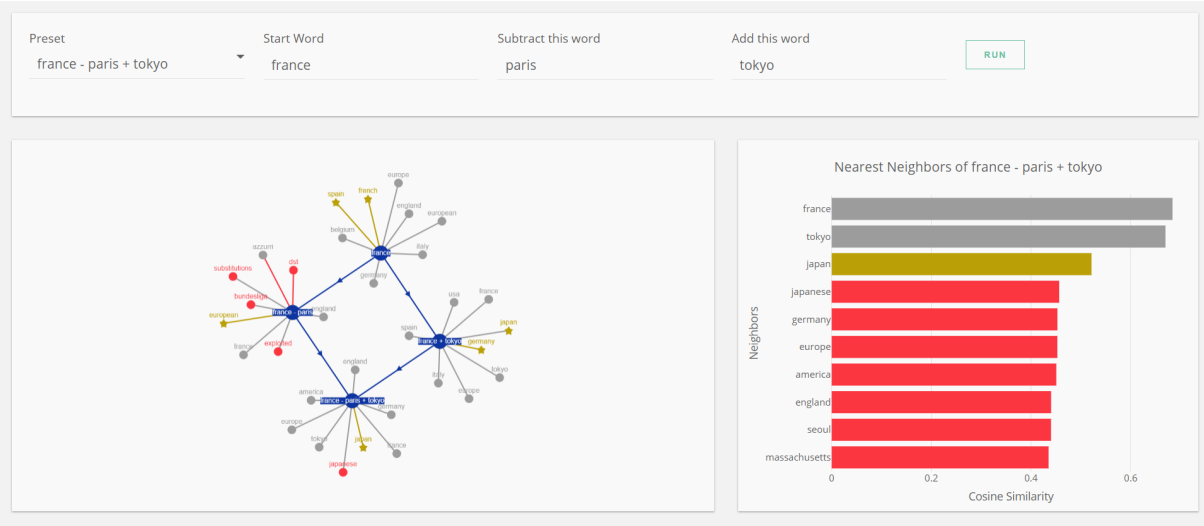
Βήμα 13: Οπτικοποίηση των word embeddings

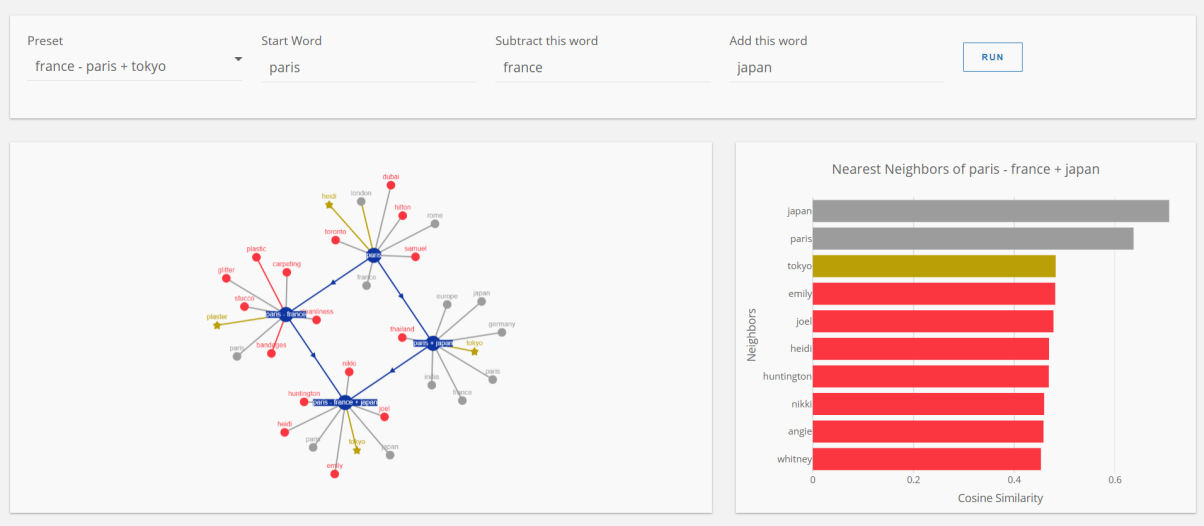
α)

Εδώ πειραματιστήκαμε με το online πρόγραμμα αυτό και πραγματοποιήσαμε τις πράξεις:

`France - Paris + Tokyo`
`Paris - France + Japan`

Παρατηρούμε ότι ο κοντινότερος γείτονας της πράξης `France - Paris + Tokyo` είναι `Japan` και της `Paris - France + Japan` είναι `Tokyo`.





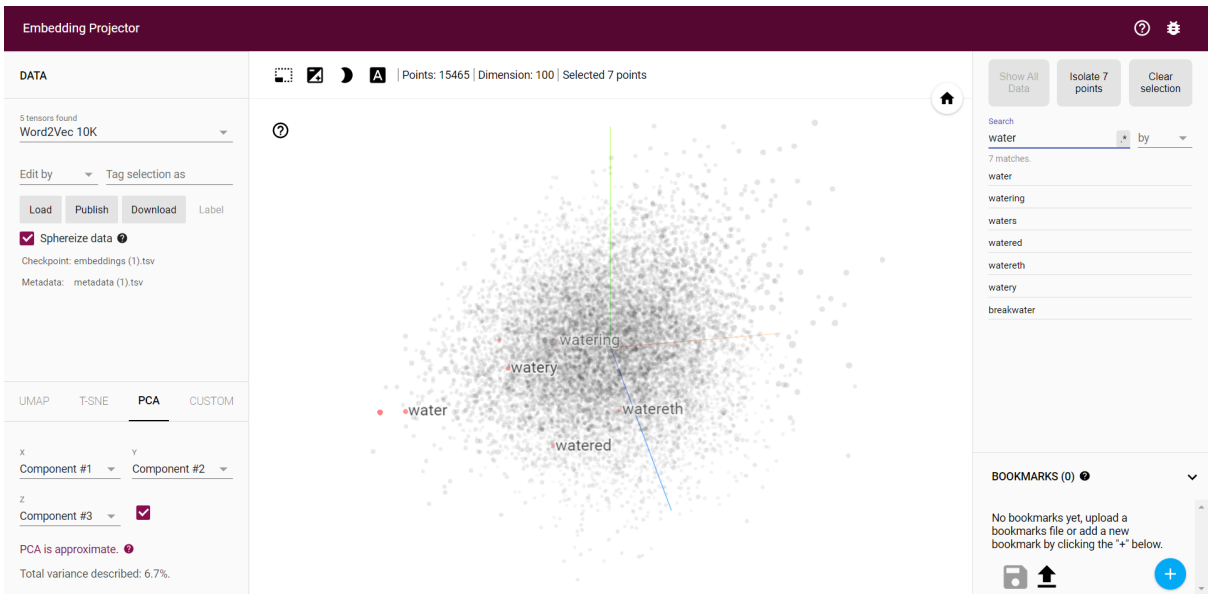
β)

Εξάγουμε το `embeddings.tsv` και το `metadata.tsv`.

γ)

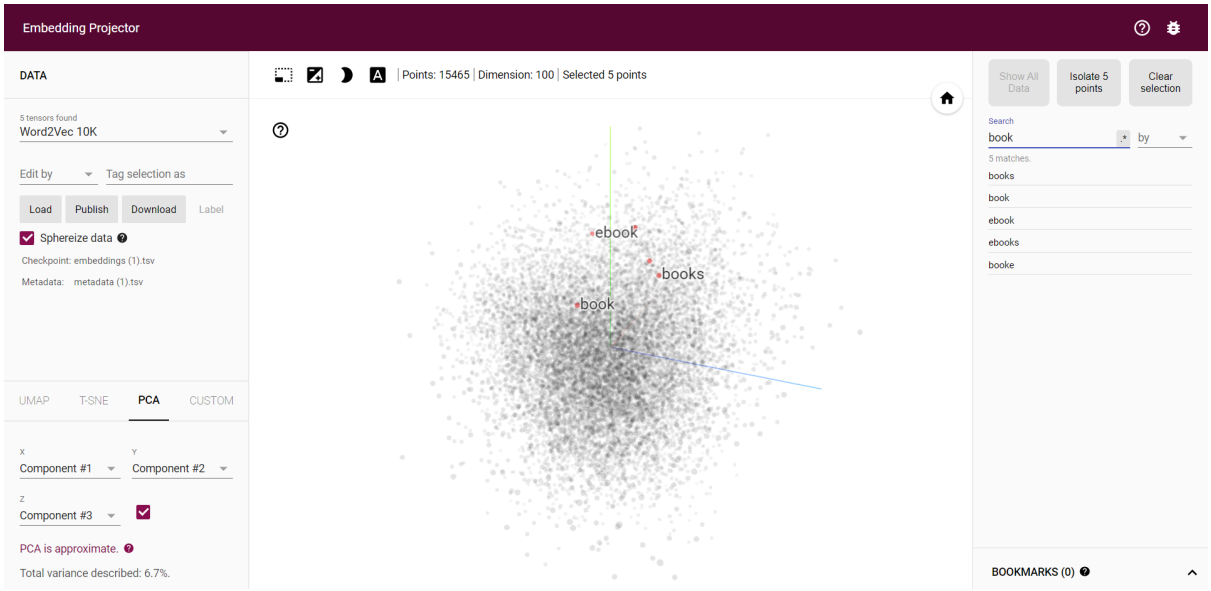
Βάζουμε τα παραπάνω αρχεία στον Embedding Projector και τα οπτικοποιούμε σε 3διάστατο χώρο με τις τεχνικές T-SNE (είναι μια non-linear μέθοδος μείωσης διαστάσεων και χρησιμοποιεί ένα t-distributed variant) και PCA (που είναι μια linear μέθοδος). Παίρνουμε τα εξής αποτελέσματα:

- Για τον PCA:
  - Η λέξη water:



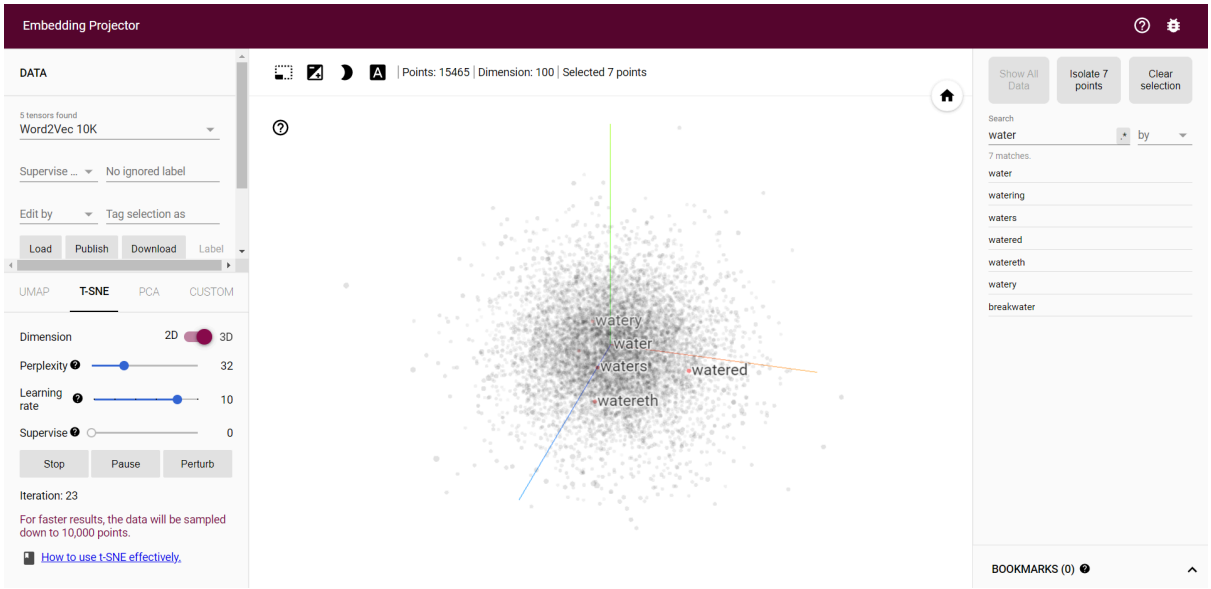
παρατηρούμε ότι είναι κοντά στην λέξη watery, watering

  - Η λέξη book:

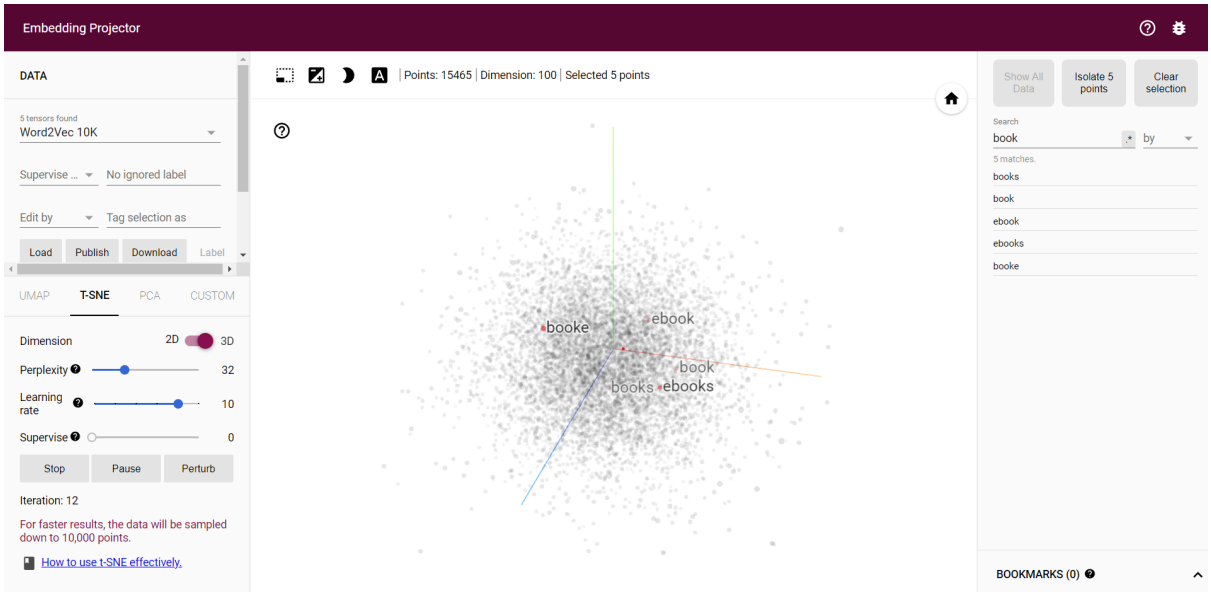


παρατηρούμε ότι είναι κοντά στη λέξη ebook, books.

- Για τον T-SNE:
  - Η λέξη water:



  - Η λέξη book:



Βήμα 14: Ανάλυση συναισθήματος με word2vec embeddings

α)

Αρχικά κατεβάσαμε τα δεδομένα που ζητούνται.

β)

Υλοποιήσαμε τον κώδικα `w2v_sentiment_analysis.py` και χρησιμοποιήσαμε το μοντέλο που εκπαιδεύσαμε για την εξαγωγή τον embeddings των θετικών και αρνητικών σχολίων από το dataset. Μέσω της μεθόδου Neural Bag of Words υπολογίσαμε τα embeddings των προτάσεων ως μέσο όρο των word embeddings των λέξεων που τις αποτελούν ως 100διάστατα διανύσματα για κάθε πρόταση.

γ)

Στην συνέχεια εκπαιδεύσαμε το Logistic Regression Model ώστε να προβλέπει αν μια πρόταση είναι θετική η αρνητική με ποσοστό ακρίβειας 74.14%

δ)

Επαναλάβαμε το παραπάνω βήμα χρησιμοποιώντας τα embeddings από το μοντέλο της Google με ποσοστό 83.2%.