Επεξεργασία Φωνής και Φυσικής Γλώσσας
Εργαστήριο σε *FSTs* και *word embeddings*

Ονοματεπώνυμο: Άρης Μαρκογιαννάκης
ΑΜ: 03120085

Βήμα 1: Κατασκευή corpus

(α) Τα βήματα της προεπεξεργασίας του κειμένου τα οποία κάνει το **step1.py**, το οποίο είναι στην ουσία το **fetch_gutenberg.py** που δίνεται, είναι τα εξής:

- Κατεβάζει το επιλεγμένο *corpus* (προεπιλογή είναι το *'gutenberg'*) από τη βιβλιοθήκη *NLTK* και επιστρέφει το περιεχόμενό του σε μορφή *raw text*
- Αφαίρεί τα *leading* και *trailing spaces*
- Μετατρέπει σε *lowercase* όλα τα γράμματα
- Αντικαθιστά τα *contractions* (π.χ. *'don't'* γίνεται *'do not'*)
- Κάνει *strip* τα πολλαπλά *whitespaces*
- Αφαίρεί όλους τους χαρακτήρες εκτός από τα πεζά γράμματα και τα κενά (δηλ. σημεία στίξης, αριθμούς κ.λπ.)
- Κάνει *tokenization* διαχωρίζοντας το κείμενο σε λέξεις, αγνοώντας τις κενές λέξεις

Η προεπεξεργασία του κειμένου που πραγματοποιείται είναι αρκετά επιθετική. Αυτό μπορεί να είναι επωφελές για διάφορους λόγους, όπως την αφαίρεση του θορύβου καθώς, αφαιρώντας σημεία στίξης και μετατρέποντας τα κείμενα σε πεζά γράμματα, μειώνουμε τον θόρυβο στο κείμενο, που μπορεί να βοηθήσει στη βελτίωση της ακρίβειας των γλωσσικών μοντέλων. Επίσης μπορεί να βοηθήσει και την στην ομοιομορφία των δεδομένων, αφού με την συγκεκριμένη αντιμετώπιση των *contractions* και την ομοιόμορφη μορφοποίηση του κειμένου, διασφαλίζουμε ότι η ανάλυση θα γίνει σε μια πιο ομοιόμορφη μορφή των δεδομένων.
Ωστόσο, υπάρχουν περιπτώσεις όπου μπορεί να θέλουμε να διατηρήσουμε τα σημεία στίξης και να μην πραγματοποιήσουμε τόσο επιθετικό *preprocessing*:

- **Ανάλυση συναισθήματος**: Τα σημεία στίξης, όπως τα ερωτηματικά και τα θαυμαστικά, μπορούν να παρέχουν σημαντικές πληροφορίες για το συναίσθημα του κειμένου.
- **Σημασιολογική ανάλυση**: Η θέση και η χρήση των σημείων στίξης μπορεί να επηρεάσουν τη σημασία των προτάσεων.

(β) Η επέκταση του *corpus* με περισσότερα βιβλία ή κείμενα από άλλες πηγές έχει πολλά πλεονεκτήματα πέρα από την αύξηση του μεγέθους των δεδομένων. Δύο σημαντικά πλεονεκτήματα είναι:

- **Ποικιλία Λεξιλογίου**: Η εισαγωγή περισσότερων και διαφορετικών κειμένων εμπλουτίζει το λεξιλόγιο του *corpus*, καθώς περιλαμβάνει λέξεις και φράσεις που μπορεί να μην υπήρχαν στα αρχικά κείμενα καθώς και μεγαλύτερο εύρος από θεματικές ενότητες
- **Λιγότερο Overfitting**: Με περισσότερα δεδομένα, το μοντέλο είναι λιγότερο πιθανό να υπερπροσαρμοστεί στα δεδομένα εκπαίδευσης, δηλαδή να μαθαίνει τις ιδιαιτερότητες του *corpus* αντί για τις γενικές αρχές της γλώσσας

Βήμα 2: Κατασκευή λεξικού

Σε αυτό το βήμα χρησιμοποιούμε το **step2.py**

α) Δημιουργούμε ένα *dictionary* που περιέχει σαν *keys* όλα τα μοναδικά *tokens* που βρίσκονται στο *corpus* και *values* τον αριθμό των εμφανίσεων του κάθε *token* στο *corpus*. Παρακάτω φαίνονται κάποια $key - value\ pairs$ του λεξικού που παράγεται:

```
tigris 3
weathering 1
headland 5
diaz 1
via 1
magnification 1
turks 4
devoutly 2
mosque 4
pitchpoling 6
axles 1
unctuousness 4
barges 4
actium 1
fleetness 3
unintermitted 4
imperative 1
dexterities 1
sleights 2
```

β)Έπειτα φιλτράρουμε όλα τα *tokens* που εμφανίζονται λιγότερες από 5 φορές. Αυτό γίνεται για να μειώσουμε το μέγεθος του λεξικού και να αποφύγουμε σπάνιες λέξεις που μπορεί να μην είναι χρήσιμες για τον ορθογράφο. Επίσης κάποιες από τις λέξεις που εμφανίζονται πολύ σπάνια μπορεί να έχουν οριγραφικό λάθος.

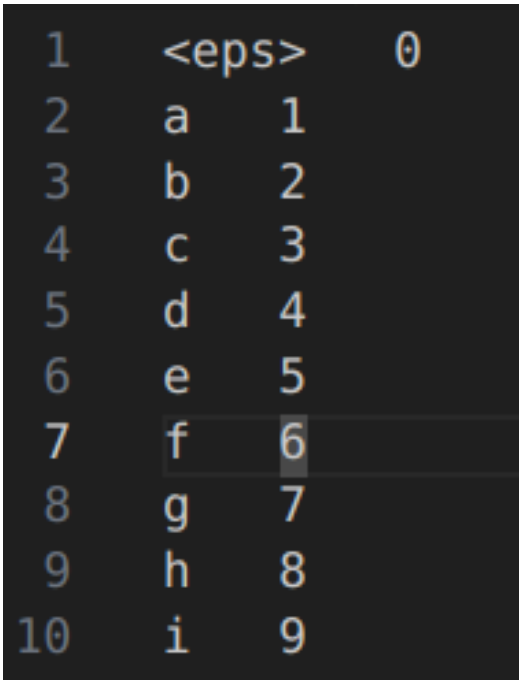
γ) Έπειτα γράφουμε στο αρχείο *vocab/words.vocab.txt* το λεξικό σε δύο *tab separated* στήλες, όπου η πρώτη στήλη περιέχει τα *tokens* και η δεύτερη στήλη τους αντίστοιχους αριθμούς εμφανίσεων. Παρακάτω φαίνονται οι πρώτες γραμμές αυτού του αρχείου:

```
1   emma    866
2   by  8512
3   jane    303
4   volume  32
5   i   30300
6   chapter 342
7   woodhouse   314
8   handsome    132
9   clever  78
10  and 95444
11  rich    240
12  with    17600
13  a   33962
14  comfortable 108
15  home    683
16  happy   569
17  disposition 73
18  seemed  1086
```

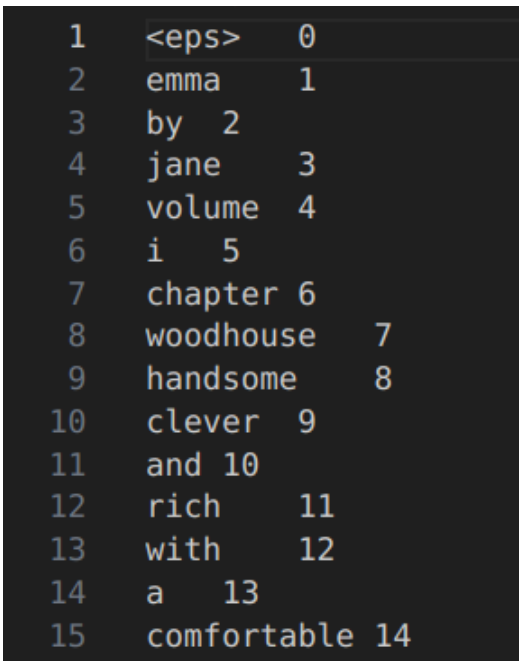
Βήμα 3: Δημιουργία συμβόλων εισόδου/εξόδου

Σε αυτό το βήμα χρησιμοποιούμε το **step3.py**

α) Γράφουμε μια συνάρτηση *Python* που αντιστοιχίζει κάθε *lowercase* χαρακτήρα της Αγγλικής γλώσσας σε ένα αύξοντα ακέραιο *index*. Το πρώτο σύμβολο με *index* 0 είναι το ϵ ($< eps >$). Το αποτέλεσμα γράφεται στο αρχείο *vocab/chars.syms* με αυτή τη μορφή. Παρακάτω φαίνονται οι πρώτες γραμμές αυτού του αρχείου:



β) Δημιουργούμε το αρχείο *words.syms* που αντιστοιχίζει κάθε λέξη (*token*) από το λεξιλόγιο που κατασκευάσαμε στο Βήμα 2 σε ένα μοναδικό ακέραιο *index* και γράφουμε το αποτέλεσμα στο *vocab/words.syms*. Παρακάτω φαίνονται οι πρώτες γραμμές αυτού του αρχείου:



Βήμα 4: Κατασκευή μετατροπέα edit distance

Σε αυτό το βήμα χρησιμοποιούμε το **step4.py**

α) Θα κατασκευάσουμε έναν μετατροπέα *Levenshtein* που υλοποιεί την απόσταση *Levenshtein* με κόστος 1 για κάθε τύπο επεξεργασίας (*insertions*, *deletions*, ; *substitutions*) και κόστος 0 για τις ταυτίσεις χαρακτήρων.

- Κάθε χαρακτήρας αντιστοιχεί στον εαυτό του με βάρος 0 (*no edit*)
- Κάθε χαρακτήρας αντιστοιχεί στο ε (κενό) με βάρος 1 (*deletion*)
- Το ε (κενό) αντιστοιχεί σε κάθε χαρακτήρα με βάρος 1 (*isnetion*)
- Κάθε χαρακτήρα σε κάθε άλλο χαρακτήρα με βάρος 1 (*substitutions*)

Ο μετατροπέας αυτός, όταν εφαρμόζεται σε μια λέξη εισόδου, υπολογίζει την απόσταση *Levenshtein* μεταξύ της εισόδου και των πιθανών λέξεων εξόδου, επιλέγοντας το μονοπάτι με το μικρότερο κόστος το οποίο είναι το *shortest path*.

β) Αποθηκεύουμε το αρχείο $fsts/L.fst$ που περιέχει την περιγραφή του μετατροπέα σε *Openfst text format*. Παρακάτω φαίνονται οι πρώτες γραμμές αυτού του αρχείου:

```
1   0 0 <eps> <eps> 0
2   0 0 a a 0
3   0 0 <eps> a 1
4   0 0 a <eps> 1
5   0 0 a b 1
6   0 0 a c 1
7   0 0 a d 1
8   0 0 a e 1
tensions (Ctrl+Shift+X)
10  0 0 a g 1
11  0 0 a h 1
12  0 0 a i 1
13  0 0 a j 1
14  0 0 a k 1
15  0 0 a l 1
```

γ) Χρησιμοποιόντας την *fstcompile* κάνουμε *compile* τον L και αποθηκεύουμε το αποτέλεσμα στο $fsts/L.binfst$. Παρακάτω φαίνονται οι πρώτες γραμμές αυτού του αρχείου:

```
1   d6fd b27e 0600 0000 7665 6374 6f72 0800
2   0000 7374 616e 6461 7264 0200 0000 0000
3   0000 0300 42a5 8100 0000 0000 0000 0000
4   0000 0100 0000 0000 0000 0000 0000 0000
5   0000 0000 807f d902 0000 0000 0000 0000
6   0000 0000 0000 0000 0000 0000 0000 0100
7   0000 0100 0000 0000 0000 0000 0000 0000
8   0000 0100 0000 0000 803f 0000 0000 0100
9   0000 0000 0000 0000 803f 0000 0000 0100
10  0000 0200 0000 0000 803f 0000 0000 0100
11  0000 0300 0000 0000 803f 0000 0000 0100
12  0000 0400 0000 0000 803f 0000 0000 0100
13  0000 0500 0000 0000 803f 0000 0000 0100
14  0000 0600 0000 0000 803f 0000 0000 0100
15  0000 0700 0000 0000 803f 0000 0000 0100
16  0000 0800 0000 0000 803f 0000 0000 0100
17  0000 0900 0000 0000 803f 0000 0000 0100
18  0000 0a00 0000 0000 803f 0000 0000 0100
19  0000 0b00 0000 0000 803f 0000 0000 0100
20  0000 0c00 0000 0000 803f 0000 0000 0100
```

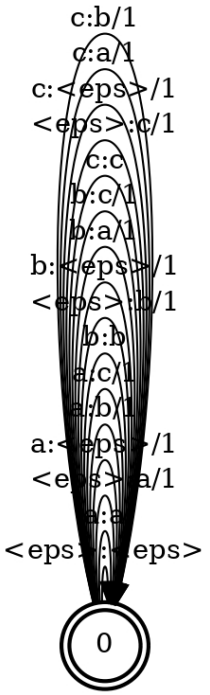
δ) Άλλα πιθανά *edits* που θα μπορούσαμε να συμπεριλάβουμε περιλαμβάνουν:

- Αντιστοίχιση συχνών λαθών πληκτρολόγησης (π.χ., 'q' αντί για 'a' λόγω εγγύτητας στο πληκτρολόγιο)
- Αντιμετώπιση των *transpositions* (π.χ., 'hte' αντί 'the")

ε) Για να βελτιώσουμε τα βάρη των *edits* εισάγοντας πρότερη γνώση, μπορούμε:

- Στα συχνά λάθη πληκτρολόγησης να βάλουμε κόστος για το *substitution* μικρότερο από 1, π.χ. 0.5
- Στα *transportations* για συχνά εμφανιζόμενα *strings* να βάλουμε κόστος μικρότερο από 2 (που αντιστοιχεί δε δύο *substitutions*), π.χ. 1.5

ζ) Χρησιμοποιώντας την $fstdraw$ σχεδιάζουμε τον L ένα μικρό υποσύνολο των χαρακτήρων, $\{a, b, c\}$. Παρακάτω φαίνεται το αποτέλεσμα:



Βήμα 5: Κατασκευή αποδοχέα λεξικού

Σε αυτό το βήμα χρησιμοποιούμε το **step5.py**

α) Κατασκευάζουμε τον αποδοχέα στο αρχείο $fsts/V.fst$ σύμφωνα με την εκφώνηση. β),γ),δ) Έπειτα, με τη χρήση της $fstcompile$ κάνουμε $compile$ τον αποδοχέα και τον αποθηκεύουμε ως $fsts/V.binfst$. Στην συνέχεια εκτελούμε τις $fstrmepsilon, fstdeterminize, fstminimize$.

- **fstrmepsilon** : Μετατρέπει το FST σε ένα ισοδύναμο, το οποίο δεν περιέχει ϵ-μεταβάσεις.
- **fstdeterminize** : Μετατρέπει το FST σε ένα ισοδύναμο ντετερμινιστικό. Δηλαδή σε κάθε κατάσταση, για κάθε σύμβολο εισόδου υπάρχει μοναδική μετάβαση.
- **fstminimize**: Μετατρέπει το FST σε ένα ισοδύναμό του, το οποίο έχει τον ελάχιστο αριθμό καταστάσεων και μεταβάσεων.

Traversal Complexity:

Ένα DFA έχει μια καθορισμένη μετάβαση για κάθε συνδυασμό κατάστασης και συμβόλου εισόδου, επομένως η διάσχιση ενός DFA έχει πολυπλοκότητα $O(n)$, όπου n είναι το μήκος της ακολουθίας εισόδου, επειδή το DFA επεξεργάζεται κάθε σύμβολο της ακολουθίας μία φορά.

Ένα NFA μπορεί να έχει πολλές μεταβάσεις για κάθε συνδυασμό κατάστασης και συμβόλου εισόδου, και μπορεί να χρησιμοποιήσει ε-μεταβάσεις. Η πολυπλοκότητα της διάσχισης ενός NFA είναι συνήθως $O(2^n)$, όπου n είναι το μήκος της ακολουθίας εισόδου, λόγω της πιθανότητας εξερεύνησης πολλών διαδρομών ταυτόχρονα. Στην πράξη, αυτό σημαίνει ότι η διάσχιση μπορεί να είναι εκθετική στο χειρότερο σενάριο.

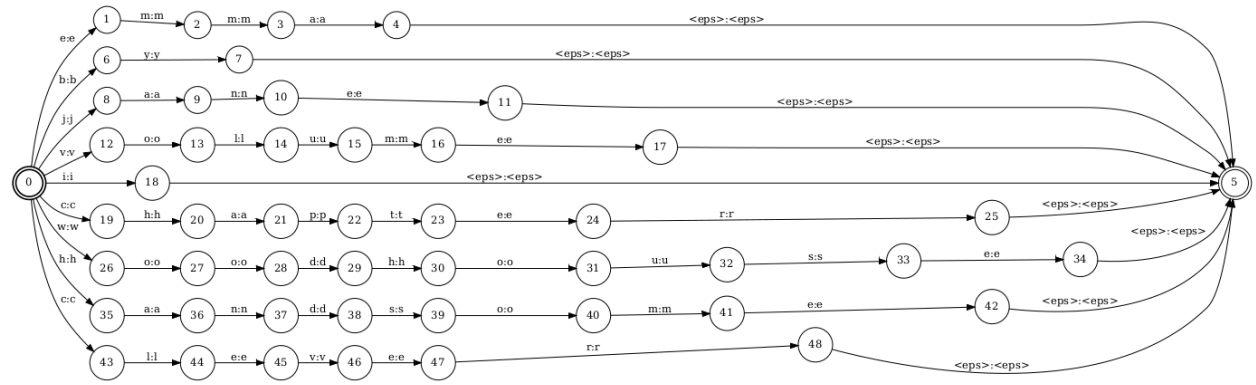
Αριθμός Ακμών:

Σε ένα DFA, κάθε κατάσταση έχει ακριβώς μία μετάβαση για κάθε σύμβολο του αλφαβήτου. Έτσι, αν το αλφάβητο έχει μέγεθος $|\Sigma|$ και το αυτόματο έχει Q καταστάσεις, τότε ο συνολικός αριθμός των ακμών είναι το πολύ $Q \times |\Sigma|$.

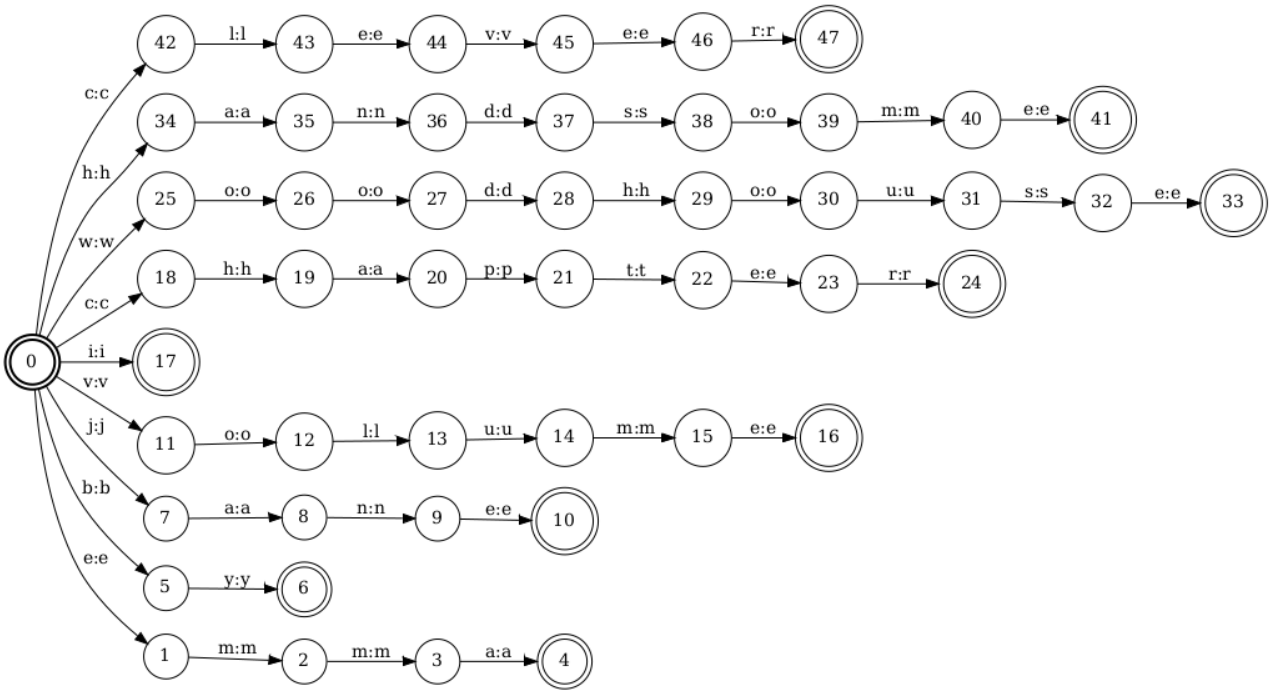
Σε ένα NFA, κάθε κατάσταση μπορεί να έχει μηδέν, μία ή πολλές μεταβάσεις για κάθε σύμβολο του αλφαβήτου, καθώς και ε-μεταβάσεις. Έτσι, αν το αλφάβητο έχει μέγεθος $|\Sigma|$ και το αυτόματο έχει Q καταστάσεις, τότε ο συνολικός αριθμός των ακμών είναι το πολύ
$Q \times Q \times |\Sigma|$ (για κάθε κατάσταση, προς κάθε άλλη με κάθε σύμβολο) $+ Q \times Q$ (για τις ε-μεταβάσεις) $= Q^2(|\Sigma| + 1)$

ε) Στην οπτικοποίηση των FST με την χρήση της $fstdraw$ χρησιμοποιήσαμε μόνο τις λέξεις
$emma$, by, και $jane$

Αρχικό FST

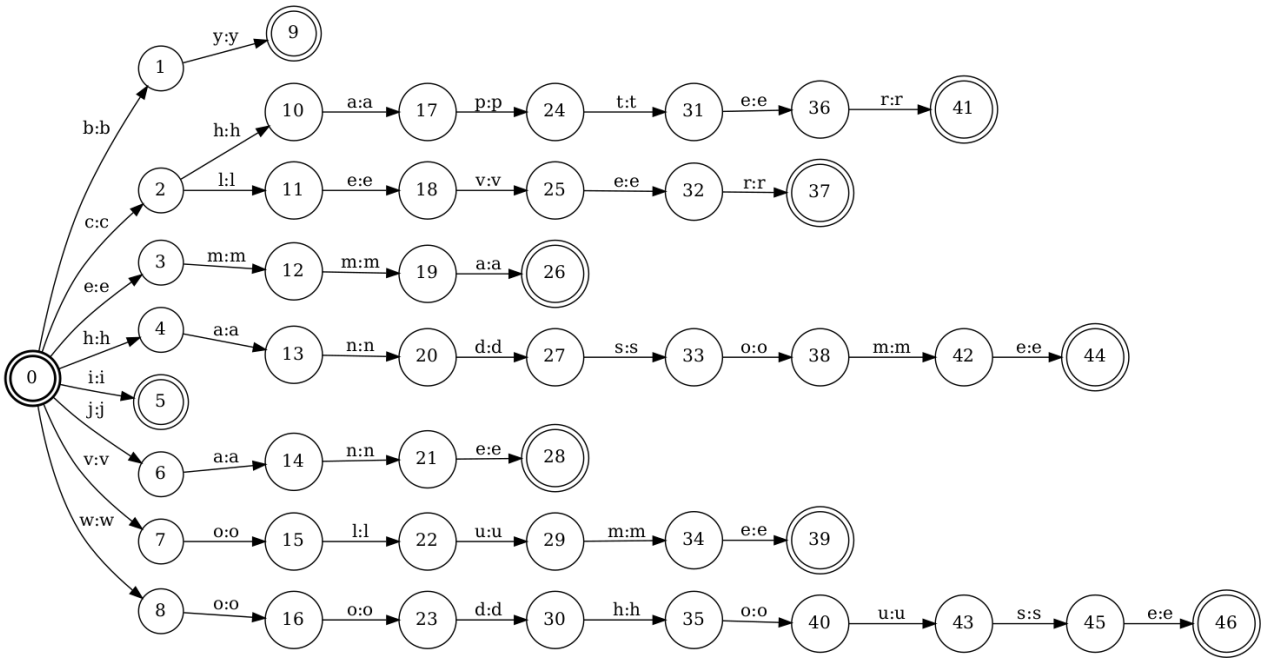


Μετά την εφαρμογή της $fstrmepsilon$



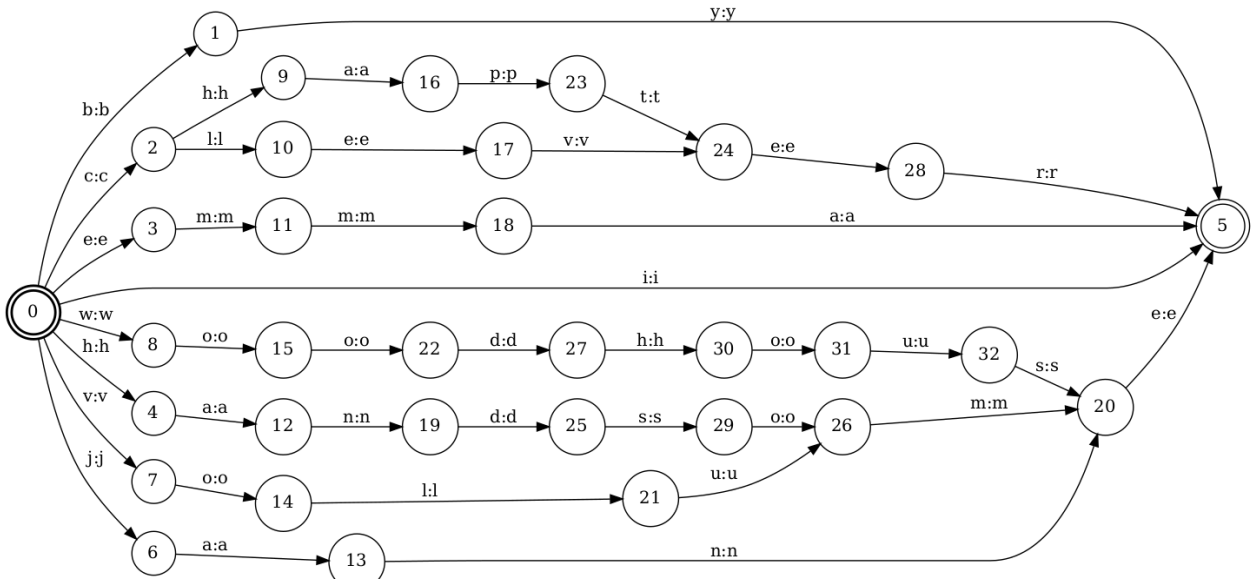
Παρατηρούμε ότι δεν υπάρχουν πια μεταβάσεις που έχουν το $< eps >$ για σύμβολο εισόδου.

Μετά την εφαρμογή της $fstdeterminize$



Παρατηρούμε ότι το FST έγινε ντετερμινιστικό, δηλαδή σε κάθε κατάσταση, για κάθε σύμβολο εισόδου υπάρχει μοναδική μετάβαση από αυτήν.

Μετά την εφαρμογή της $fstminimize$



Παρατηρούμε ότι το FST έχει μικρότερο μέγεθος από το προηγούμενο κάτι που σημαίνει ότι η ελαχιστοποίηση δούλεψε.

Βήμα 6: Κατασκευή ορθογράφου

Σε αυτό το βήμα χρησιμοποιούμε το **step6.py**

α) Αφού πρώτα χρησιμοποιήσουμε την $fstarcsort$ στους $fsts/V.binfst$ και $fsts/L.binfst$, χρησιμοποιούμε την $fstcompose$ για να φτιάξουμε τον ορθογράφο $fsts/S.binfst$.

Ισοβαρή edits:

Όταν όλες οι αλλαγές (εισαγωγή, διαγραφή, αντικατάσταση) έχουν το ίδιο βάρος, ο μετατροπέας $fsts/S.binfst$ θα δώσει προτεραιότητα στις διορθώσεις που απαιτούν τον μικρότερο αριθμό αλλαγών στην είσοδο, ανεξάρτητα από τον τύπο της αλλαγής.

Edits με διαφορετικά βάρη:

Όταν όλες οι αλλαγές (εισαγωγή, διαγραφή, αντικατάσταση) δεν έχουν το ίδιο βάρος, ο μετατροπέας $fsts/S.binfst$ θα δώσει προτεραιότητα στις διορθώσεις που περιέχουν *deletions* και *insertions* και όχι *substitutions*, αν ισχύει π.χ. $cost(insertion) = cost(deletion) = 1$, $cost(substitution) = 1.5$.

β) Για τις λέξεις *cit* και *cwt*, ο *min edit spell checker* με ισοβαρή *edits* θα δώσει σαν προβλέψη λέξεις που απέχουν το πολύ 1 *insertion*, *deletion* ή *substitution* από αυτές που δώσαμε για είσοδο.

Για την λέξη *cit* κάποιες πιθανές προβλέψεις είναι :

- Από *insertion*: *city*
- Από *deletion*: *it*
- Από *substitution*: *cut*, *cat*, *hit*, *sit*

Για την λέξη *cwt* κάποιες πιθανές προβλέψεις είναι :

- Από *insertion*: -
- Από *deletion*: -
- Από *substitution*: *cut*, *cat*

Βήμα 7: Δοκιμή ορθογράφου

Σε αυτό το βήμα χρησιμοποιούμε το **step7.py**

α) Κατεβάζουμε το συνολο δεδομένων που αναφέρεται στην εκφώνηση.
β) Αρχικά κατεβάζουμε τα $mkfstinput.py$ και $util.py$ και συμπληρώνουμε των κώδικα που λείπει.
Το *script predict.sh* διορθώνει ορθογραφικά λάθη σε μια λέξη χρησιμοποιώντας τον *transducer* που φτιάξαμε στα προηγούμενα βήματα, επιτελώντας κατά σειρά τις παρακάτω πράξεις:

- Χρησιμοποιώντας το $mkfstinput.py$ φτιάχνει ένα fst για την (λανθασμένη) λέξη που του δίνουμε για είσοδο
- Με την $fstcompile$ κάνουμε *compile* το FST της λέξης εισόδου και στην συνέχεια με την $fstcompose$ το συνθέτει με τον *spell checker* που φτιάξαμε στο βήμα 6. Ο παραγόμενος *transducer* περιέχει διαδρομές που αντιπροσωπεύουν πιθανές διορθώσεις της λανθασμένης λέξης, με βάση το μοντέλο.
- Με την $fstshortestpath$ βρίσκει τη συντομότερη διαδρομή στο παραγόμενο FST, η οποία αντιστοιχεί στην πιο πιθανή διόρθωση της λέξης.
- Με την $fstrmepsilon$ αφαιρεί τις ϵ μεταβάσεις.
- Με την $fsttopsort$ κάνει τοπολογική ταξινόμηση του FST
- Με την $fstprint$ τελικό FST μετατρέπεται σε αναγνώσιμη μορφή χρησιμοποιώντας το αρχείο συμβόλων $wrods.syms$
- Μέ την $cut -f4$ εξάγει την τέταρτη στήλη της εξόδου της $fstprint$, η οποία περιέχει τις διορθωμένες λέξεις.

- Με την $grep\ -v\ "<eps>"$ αφαιρεί τις κενές εξόδους.
- Με την $head\ -n\ -1$ αφαιρεί την τελευταία γραμμή, η οποία είναι η τελική κατάσταση του FST.
- Με την $tr\ -d\ "\backslash n"$ αφαιρεί τυχόν επιπλέον χαρακτήρες $newline$.

Τα αποτελέσματα που πήραμε:

```
Input: contenpted, Correct: contented, Predicted: contented
Input: contende, Correct: contented, Predicted: contend
Input: contended, Correct: contented, Predicted: contended
Input: contentid, Correct: contented, Predicted: contented
Input: begining, Correct: beginning, Predicted: beginning
Input: problam, Correct: problem, Predicted: problem
Input: proble, Correct: problem, Predicted: problem
Input: promblem, Correct: problem, Predicted: problem
Input: proplen, Correct: problem, Predicted: prophet
Input: dirven, Correct: driven, Predicted: dive
Input: exstacy, Correct: ecstasy, Predicted: ecstasy
Input: ecstacy, Correct: ecstasy, Predicted: ecstasy
Input: guic, Correct: juice, Predicted: guil
Input: juce, Correct: juice, Predicted: june
Input: jucie, Correct: juice, Predicted: lucil
Input: juise, Correct: juice, Predicted: juice
Input: juse, Correct: juice, Predicted: use
Input: localy, Correct: locally, Predicted: local
Input: compair, Correct: compare, Predicted: complain
Input: pronounciation, Correct: pronunciation, Predicted: provocation
Input: transportibility, Correct: transportability, Predicted: respectability
Input: miniscule, Correct: minuscule, Predicted: ridicule
Input: independant, Correct: independent, Predicted: independent
Input: independant, Correct: independent, Predicted: independent
Input: aranged, Correct: arranged, Predicted: arranged
Input: arrainged, Correct: arranged, Predicted: arranged
Input: poartry, Correct: poetry, Predicted: party
Input: poertry, Correct: poetry, Predicted: poetry
Input: poetre, Correct: poetry, Predicted: poetry
Input: poety, Correct: poetry, Predicted: poet
Input: powetry, Correct: poetry, Predicted: poetry
Input: leval, Correct: level, Predicted: level
Input: basicaly, Correct: basically, Predicted: scaly
Input: triangulaur, Correct: triangular, Predicted: triangular
Input: unexpcted, Correct: unexpected, Predicted: unexpected
Input: unexpeted, Correct: unexpected, Predicted: unexpected
Input: unexspected, Correct: unexpected, Predicted: unexpected
Input: stanerdizing, Correct: standardizing, Predicted: standing
Input: varable, Correct: variable, Predicted: parable
```

Παρατηρούμε ότι αυτός ο αρκετά αφελής *spell checket* καταφέρνει να διορθώσει σωστά τις 21 από τις 37 εισόδους, δηλαδή έχει *accuracy* σχεδόν 57%.

Βήμα 8: Υπολογισμός κόστους των edits

Σε αυτό το βήμα χρησιμοποιούμε το **step8.py**

α), β), γ) Τα βήματα που αναφέρονται στην εκφώνηση εκτελούνται από το *script word_edits.sh* που έχει δοθεί. Τρέχουμε το *word_edits.sh* για μερικά παραδείγματα:

```
Wrong word: tst, Correct: test, Edit to check: insertion
<eps>   e
Wrong word: applle, Correct: apple, Edit to check: deletion
l       <eps>
Wrong word: workong, Correct: working, Edit to check: substitution
o       i
```

Φαίνεται ότι το *word_edits.sh* λειτουργεί σωστά.

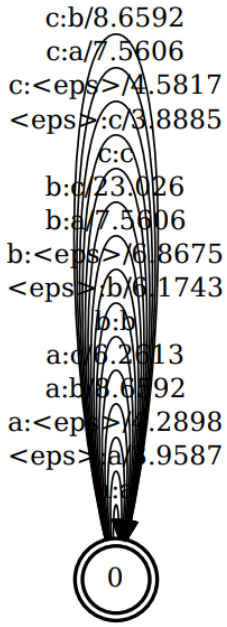
Το *word_edits.sh* χρησιμοποιείται για να υπολογίσει και να παρουσιάσει τον ελάχιστο αριθμό *edits* που απαιτούνται για να διορθωθεί μία ανορθόγραφη λέξη. Το *script* επιτελεί τις παρακάτω λειτουργίες:

- Αρχικά παίρνει τις λέξεις $WRONG$ και $CORRECT$ από τα *commnand line arguments* και καθορίζει τα *paths* στα οποία βρίσκεται ο $Vanilla\ Levinstein\ FST$, το αρχείο *chars.syms* και που θα αποθηκευτεί ο ενδιάμεσος $ML.fst$
- Έπειτα χρησιμοποιώντας το $mkfstinput.py$ φτιάχνει ένα FST για την ανορθόγραφη λέξη, το κάνει *compile* και το συνθέτει με τον $Levinstein\ FST$ φτιάχνοντας τον $ML.fst$
- το αρχείο *chars.syms* και που θα αποθηκευτεί ο ενδιάμεσος $ML.fst$
- Έπειτα χρησιμοποιώντας το $mkfstinput.py$ φτιάχνει ένα FST για την σωστή λέξη, το κάνει *compile* και το συνθέτει με τον $ML.FST$ φτιάχνοντας τον $MLN.fst$. Ο τελευταίος FST περιέχει μονοπάτια από την ανορθόγραξη προς την σωστή λέξη με βάρη στις ακμές του τα βάρη των *edits*
- Με την $fstshortestpath$ βρίσκει το συντομότερο μονοπάτι, δηλαδή αυτό με την μικρότερη *edit distance*, από την ανορθόγραφη λέξη προς την σωστή και με την $fstprint$ το τυπώνει. Δεν τυπώνεται όλο το μονοπατι αλλά μόνο τα *edits* που πρεπει να γίνουν λόγω των εντολών *grep* και *cut*.

δ) Για κάθε γραμμή του *data/wiki.txt* παράγουμε όλα όλα τα *edits* και τα αποθηκεύουμε στο *data/wiki_edits.txt*. Οι πρώτες γραμμές του αρχείου *data/wiki_edits.txt*:

```
1  n       <eps>
2  <eps>   r
3  y       i
4  <eps>   i
5  <eps>   i
6  o       a
7  b       <eps>
8  u       t
9  t       u
10 c       <eps>
11 o       a
12 o       a
13 o       a
14 e       i
15 <eps>   s
16 a       e
```

ε), στ) Αρχικά εξάγουμε τις συχνότητες του κάθε *edit* και τις αποθηκεύουμε σε ένα λεξικό και στην συνέχεια φτιάχνουμε των $E.fst$ όπου το βάρος της κάθε ακμής είναι το κόστος του αντίστοιχου *edit*. Παρακάτω φαίνεται ο $E.fst$ για ένα υποσύνολο:



ζ) Επαναλαμβάνουμε τα βήματα 6 και 7 και τώρα κατασκευάζουμε τον $EV.fst$. Παρακάτω φαίνονται Τα αποτελέσματα που πήραμε:

```
Input: contenpted, Correct: contented, Predicted: contented
Input: contende, Correct: contented, Predicted: contend
Input: contended, Correct: contented, Predicted: contended
Input: contentid, Correct: contented, Predicted: contented
Input: begining, Correct: beginning, Predicted: beginning
Input: problam, Correct: problem, Predicted: problem
Input: proble, Correct: problem, Predicted: problem
Input: promblem, Correct: problem, Predicted: problem
Input: proplen, Correct: problem, Predicted: people
Input: dirven, Correct: driven, Predicted: driven
Input: exstacy, Correct: ecstasy, Predicted: exactly
Input: ecstacy, Correct: ecstasy, Predicted: ecstasy
Input: guic, Correct: juice, Predicted: guil
Input: juce, Correct: juice, Predicted: juice
Input: jucie, Correct: juice, Predicted: juice
Input: juise, Correct: juice, Predicted: juice
Input: juse, Correct: juice, Predicted: just
Input: localy, Correct: locally, Predicted: local
Input: compair, Correct: compare, Predicted: compare
Input: pronounciation, Correct: pronunciation, Predicted: pronouncing
Input: transportibility, Correct: transportability, Predicted: respectability
Input: miniscule, Correct: minuscule, Predicted: mince
Input: independant, Correct: independent, Predicted: independent
Input: independant, Correct: independent, Predicted: independent
Input: aranged, Correct: arranged, Predicted: arranged
Input: arrainged, Correct: arranged, Predicted: arranged
Input: poartry, Correct: poetry, Predicted: poetry
Input: poertry, Correct: poetry, Predicted: poetry
Input: poetre, Correct: poetry, Predicted: poetry
Input: poety, Correct: poetry, Predicted: poetry
Input: powetry, Correct: poetry, Predicted: poetry
Input: leval, Correct: level, Predicted: level
Input: basicaly, Correct: basically, Predicted: busily
Input: triangulaur, Correct: triangular, Predicted: triangular
Input: unexpcted, Correct: unexpected, Predicted: unexpected
Input: unexpeted, Correct: unexpected, Predicted: unexpected
Input: unexspected, Correct: unexpected, Predicted: unexpected
Input: stanerdizing, Correct: standardizing, Predicted: sneezing
Input: varable, Correct: variable, Predicted: invariable
```

Παρατηρούμε ότι αυτός ο $EV.fst$ καταφέρνει να διορθώσει σωστά τις 27 από τις 37 εισόδους, δηλαδή έχει $accuracy$ σχεδόν 73%, το οποίο είναι αρκετά βελτιωμένο σε σχέση με το 57% που πετύχαμε στο Βήμα 7.

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

Σε αυτό το βήμα χρησιμοποιούμε το **step9.py**

α),β) Κατασκευάζουμε τον αποδοχέα W ο οποίος αποτελείται από μια κατάσταση και αντιστοιχίζει κάθε λέξη στον εαυτό της με βάρος τον αρνητικό λογάριθμο της συχνότητας εμφάνισης της λέξης. Για τις συχνότητες της κάθε λέξης χρησιμοποιούμε το *vocab/words.vocab.txt* που κατασκευάσαμε στο Βήμα 2.

γ),δ) Σύμφωνα με την διαδικασία που αναφέρεται στην εκφώνηση κατασκευάζουμε τους **LV**, **VW**, **LVW**, **EVW**

ε) Τα αποτελέσματα που πήραμε για τον **LVW**:

```
Input: contenpted, Correct: contented, Predicted: contented
Input: contende, Correct: contented, Predicted: contend
Input: contended, Correct: contented, Predicted: contended
Input: contentid, Correct: contented, Predicted: contented
Input: begining, Correct: beginning, Predicted: beginning
Input: problam, Correct: problem, Predicted: problem
Input: proble, Correct: problem, Predicted: people
Input: promblem, Correct: problem, Predicted: problem
Input: proplen, Correct: problem, Predicted: people
Input: dirven, Correct: driven, Predicted: given
Input: exstacy, Correct: ecstasy, Predicted: stay
Input: ecstacy, Correct: ecstasy, Predicted: ecstasy
Input: guic, Correct: juice, Predicted: in
Input: juce, Correct: juice, Predicted: the
Input: jucie, Correct: juice, Predicted: the
Input: juise, Correct: juice, Predicted: his
Input: juse, Correct: juice, Predicted: just
Input: localy, Correct: locally, Predicted: only
Input: compair, Correct: compare, Predicted: company
Input: pronounciation, Correct: pronunciation, Predicted: provocation
Input: transportibility, Correct: transportability, Predicted: respectability
Input: miniscule, Correct: minuscule, Predicted: minute
Input: independant, Correct: independent, Predicted: independent
Input: independant, Correct: independent, Predicted: independent
Input: aranged, Correct: arranged, Predicted: and
Input: arrainged, Correct: arranged, Predicted: arranged
Input: poartry, Correct: poetry, Predicted: party
Input: poertry, Correct: poetry, Predicted: poetry
Input: poetre, Correct: poetry, Predicted: the
Input: poety, Correct: poetry, Predicted: not
Input: powetry, Correct: poetry, Predicted: power
Input: leval, Correct: level, Predicted: evil
Input: basicaly, Correct: basically, Predicted: easily
Input: triangulaur, Correct: triangular, Predicted: triangular
Input: unexpcted, Correct: unexpected, Predicted: unexpected
Input: unexpeted, Correct: unexpected, Predicted: unexpected
Input: unexspected, Correct: unexpected, Predicted: unexpected
Input: stanerdizing, Correct: standardizing, Predicted: standing
Input: varable, Correct: variable, Predicted: parable
```

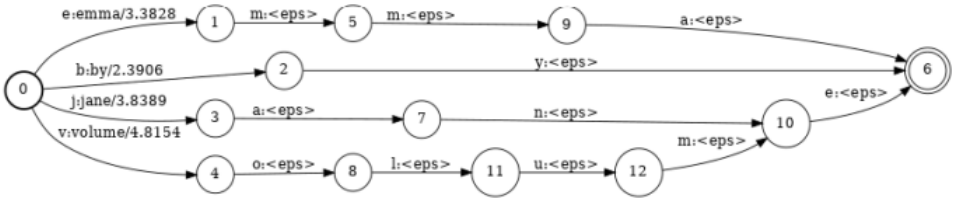
Παρατηρούμε ότι ο **LVW** καταφέρνει να διορθώσει σωστά τις 15 από τις 37 εισόδους, δηλαδή πετυχαίνει *accuracy* σχεδόν 41%. Η επίδοση αυτή είναι χειρότερη από τον απλό **LV**, χωρίς την σύνθεση με τον **W**, τον οποίο είχαμε κατασκευάσει στο Βήμα 6.
στ)

	LV	**LVW**
cwt	*cat*	*it*
cit	*wit*	*it*

Παρατηρούμε ότι ο **LVW** δίνει και στις δύο περιπτώσεις σαν διόρθωση την λέξη *it* κάτι που συμβαίνει μάλλον επειδή η λέξη *'it'* έχει αρκετά μεγάλη συνχότητα εμφάνισης, ενώ ο **LV** δίνει σαν διόρθωση τις λέξεις που απέχουν τον μικρότερο αριθμό *edits*.

ζ) **W_small.binfst**:

VW_small.binfst:



Βήμα 10: Αξιολόγηση των ορθογράφων

Σε αυτό το βήμα χρησιμοποιούμε το **run_evaluation.py**

Κατεβάζουμε τα δεδομένα που ζητούνται.
Τρέχοντας το παραπάνω **run_evaluation.py** για τους 4 $FSTs$ που ζητούνται πήραμε τα παρακάτω αποτελέσματα:

Model	**Accuracy**
LV	0.59
EV	0.71
LVW	0.42
EVW	0.62

Παρατηρούμε ότι την καλύτερη επίδοση την πετυχαίνει ο **EV**, ενώ η σύνθεση με το *unigram* μοντέλο **W** φαίνεται να ρίχνει την απόδοση των μοντέλων. Αυτό συμβαίνει ίσως επειδή το *unigram* είναι μία αρκετά απλοϊκή κατασκευή ή το *corpus* που χρησιμοποιήσαμε δεν ήταν αρκετά ποικιλόμορφο και έδινε μεγάλες συχνότητες σε λέξεις που δεν θα έπρεπε.

Βήμα 11: (*Bonus*) Βελτιώσεις του ορθογράφου

Σε αυτό το βήμα χρησιμοποιούμε το **step11.py**

α) Υλοποιούμε το $add-one\ smoothing$ και τρέχουμε το **run_evaluation.py** με είσοδο τον **EV_new.binfst** αυτή την φορά. Το μοντέλο πετυχαίνει **accuracy 0.76**. Παρατηρούμε λοιπόν ότι η τεχνική $add-one-smoothing$ που χρησιμοποιήσαμε βελτιώνει την επίδοση του μοντέλου.

β) Αρχικά κάνουμε ένα *merge* κατά κάποιο τρόπο ενώνοντας το $vocab/words.vocab.txt$ με το $data/en\_50k.txt$, στην συνέχεια φτιάχνουμε τους **V_new** και **W_new** οι οποίοι έχουν προκύψει από το καινούριο $merged\_vocab.txt$ και τέλος φτιάχνουμνε το **EVW_new** το οποίο ενσωματώνει και τις καινούριες συχνότητες. Τώρα τρέχουμε το **run_evaluation.py** με είσοδο τον **EVW_new.binfst** αυτή την φορά. Το μοντέλο πετυχαίνει **accuracy 0.68**. Παρατηρούμε λοιπόν ότι εμπλουτίζοντας το λεξικό μας και τις συχνότητες των λέξεων παίρνουμε καλύτερα αποτελέσματα στον **EVW**.

γ) Άλλες πιθανές βελτιώσεις που θα μπορούσαμε να κάνουμε είναι:

- Να ενσωματώσουμε πιο ρεαλιστικά μοντέλα σφαλμάτων (π.χ. απόσταση πληκτρολογίου) στο $E.fst$.
- Να χρησιμοποιήσουμε μοντέλα $n-gram$ υψηλότερης τάξης, αντί για $unigram$ που χρησιμοποιούμε με τον **W**, για να καταγράψουμε περισσότερα συμφραζόμενα.

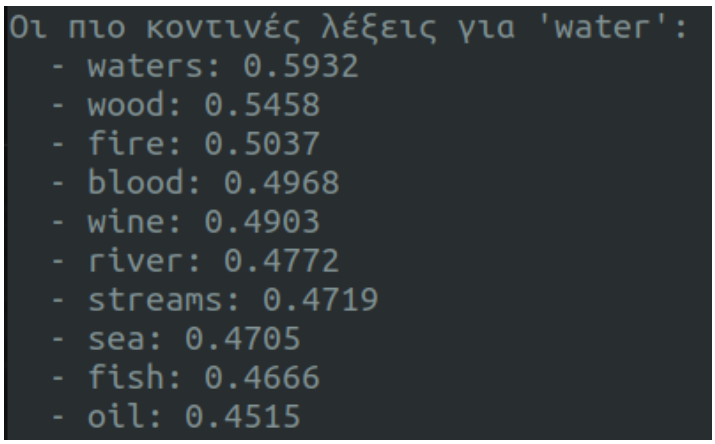
Βήμα 12: Εξαγωγή αναπαραστάσεων word2vec

Σε αυτό το βήμα χρησιμοποιούμε το **step12.py** και το **w2v_train.py**

α), β), γ) Συμπληρώνουμε τον κώδικα στο $w2v\_train.py$ που λείπει για να κατασκευάσουμε τα $embeddings$. Παρακάτω φαίνονται τα αποτελέσματα για τις λέξεις που ζητούνται:

```
Οι πιο κοντινές λέξεις για 'bible':
  - island: 0.3512
  - respects: 0.3468
  - convenience: 0.3467
  - syrian: 0.3235
  - poz: 0.3210
  - heroism: 0.3206
  - coffin: 0.3146
  - drinke: 0.3121
  - horn: 0.3042
  - heap: 0.3012
```

```
Οι πιο κοντινές λέξεις για 'book':
  - written: 0.5043
  - note: 0.4899
  - letter: 0.4851
  - temple: 0.4705
  - mouth: 0.4469
  - pen: 0.4429
  - history: 0.4377
  - chapter: 0.4347
  - epistle: 0.4318
  - seal: 0.4218
```



1) Για την λέξη:

- *bible*: Τα αποτελέσματα φαίνονται προβληματικά. Οι λέξεις '*island*', '*respects*', '*convenience*' κ.λπ., δεν έχουν προφανή σημασιολογική σχέση με τη λέξη '*bible*'. Αυτό υποδηλώνει ότι τα *embeddings* δεν έχουν συλλάβει τις θρησκευτικές ή πολιτισμικές έννοιες που σχετίζονται με τη Βίβλο.

- *book*: Λέξεις όπως *written*, *note*, *letter* και *chapter* σχετίζονται έντονα με την έννοια του βιβλίου, υποδεικνύοντας ότι το μοντέλο έχει μάθει κάποιες σχετικές σχέσεις. Ωστόσο, οι λέξεις *temple* και *mouth* είναι λιγότερο διαισθητικά σωστές υποδηλώνοντας κάποιο $overfitting$ ίσως.

- *bank*: Τα αποτελέσματα είναι και πάλι προβληματικά. Οι λέξεις '*top*' '*wall*' '$floor$' κ.λπ., σχετίζονται με την έννοια της όχθης (*river bank*) παρά με την έννοια της τράπεζας . Αυτό υποδηλώνει ότι τα *embeddings* δεν διακρίνουν μεταξύ των πολλαπλών σημασιών της λέξης '*bank*'.

- *water*: Τα αποτελέσματα είναι αρκετά καλά. Οι λέξεις '*waters*', '$fire$' '*blood*' κ.λπ., σχετίζονται με το νερό με διάφορους τρόπους (στοιχείο, αντίθεση, ουσίες που περιέχουν νερό).

2),3) Βελτίωση με Αλλαγή Παραμέτρων:

- Μέγεθος παραθύρου (*window*): Ένα μεγαλύτερο παράθυρο μπορεί να βοηθήσει στην καλύτερη κατανόηση του πλαισίου των λέξεων. Ωστόσο, ένα πολύ μεγάλο παράθυρο μπορεί να εισάγει θόρυβο και να κάνει τα *embeddings* λιγότερο διακριτικά.

- Αριθμός εποχών: Περισσότερες εποχές μπορούν να βελτιώσουν την εκπαίδευση, αλλά υπάρχει κίνδυνος $overfitting$.

4) Πιθανές Βελτιώσεις:

- Μεγαλύτερο *Corpus*: Ένα μεγαλύτερο και πιο ποικίλο *corpus* θα βοηθήσει το μοντέλο να μάθει περισσότερες και πιο ακριβείς σχέσεις μεταξύ των λέξεων.

- Προεπεξεργασία κειμένου: Τεχνικές προεπεξεργασίας κειμένου, όπως το *stemming*, το *lemmatization*, και η αφαίρεση *stop words*, μπορεί να βοηθήσουν στη μείωση του θορύβου και στη βελτίωση της ποιότητας των *embeddings*.

δ) Τα αποτελέσματα που πήραμε:

```
Top 5 similar words for the analogy 'queen : girl :: king :':
  - girl: 0.5908
  - king: 0.5899
  - woman: 0.4496
  - lad: 0.4301
  - man: 0.4259
Top 5 similar words for the analogy 'tall : taller :: good :':
  - taller: 0.6302
  - good: 0.5448
  - thankful: 0.3848
  - useful: 0.3846
  - gentleness: 0.3731
Top 5 similar words for the analogy 'paris : france :: london :':
  - france: 0.7016
  - school: 0.4709
  - london: 0.4097
  - highbury: 0.3791
  - country: 0.3750
```

Αναλογία *queen* : *girl* :: *king* :

Η λέξη '*boy*' που περιμέναμε να δούμε δεν εμφανίστηκε. Όμως οι λέξεις '*lad*' και '*man*' έχουν αρκετά κοντινή σημασία σε αυτό που περιμέναμε.

Αναλογία *tall* : *taller* :: *good* :

Η λέξη *better* που περιμέναμε να δούμε δεν εμφανίζεται και δεν εμφανίζεται κανένα επίθετο σε συγκριτικό βαθμό. Το $useful$ και το *gentle* έχουν κατά κάποιο τρόπο ομοιότητα με την λέξη καλός.

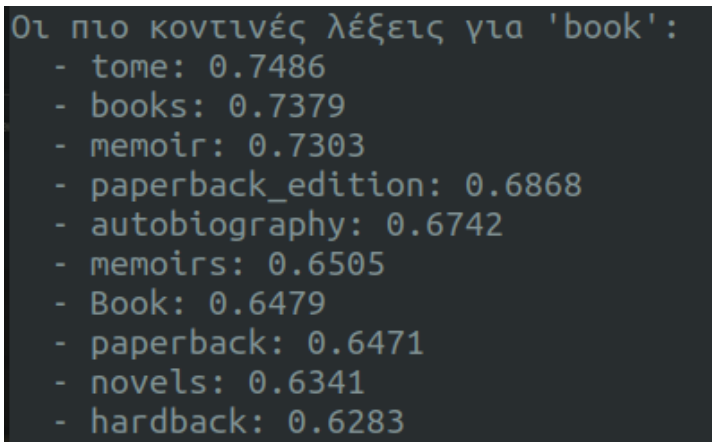
Αναλογία *paris* : $france$ :: *london* :

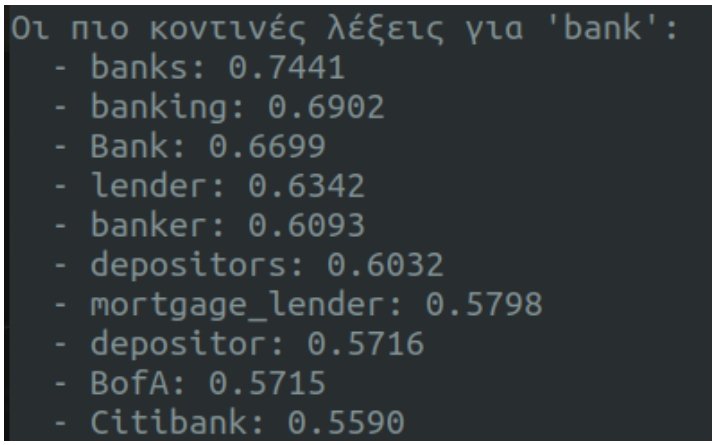
Θα περιμέναμε να δούμε την λέξη '*england*' κάτι που δεν συμβαίνει. Ισως δεν υπάρχει στο *corpus* που χρησιμοποιήθηκε για την εκπαίδευση.

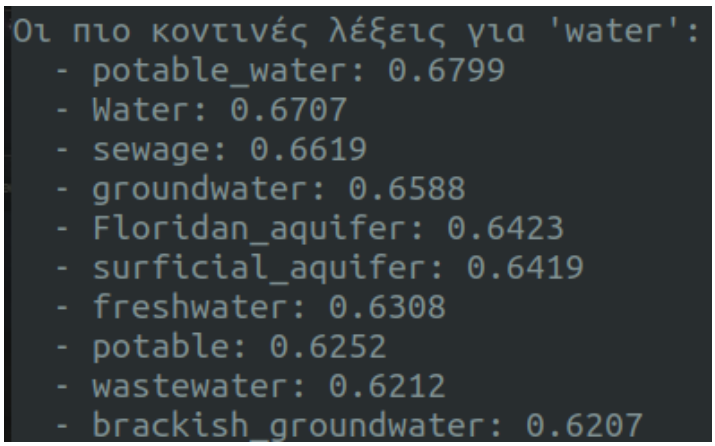
ε) Κατεβάζουμε τα *embeddings* που ζητούνται.

στ) Τα αποτελέσματα που πήραμε:

```
Οι πιο κοντινές λέξεις για 'bible':
  - Bible: 0.7368
  - bibles: 0.6053
  - Holy_Bible: 0.5990
  - scriptures: 0.5746
  - scripture: 0.5698
  - New_Testament: 0.5639
  - Scripture: 0.5503
  - Didache: 0.5502
  - Scriptures: 0.5412
  - Oxford_Annotated: 0.5391
```







Για την λέξη:

- *bible*: Τα αποτελέσματα εδώ είναι εξαιρετικά. Οι κοντινότερες λέξεις είναι όλες άμεσα σχετικές με τη Βίβλο, είτε ως παραλλαγές της λέξης (’$bibles$¨, ’$Holy\_Bible$¨, ’$Scripture$¨), είτε ως αναφορές σε συγκεκριμένα μέρη της (’$New\_Testament$¨), είτε ως γενικότεροι όροι για ιερές γραφές (’$scriptures$¨). Αυτό δείχνει ότι τα *Google News vectors* έχουν συλλάβει πολύ καλά τη θρησκευτική σημασία της λέξης.

- *book*: Τα αποτελέσματα είναι επίσης πολύ καλά. Οι κοντινότερες λέξεις περιλαμβάνουν τον πληθυντικό της λέξης (’$books$¨), συνώνυμα (’$tome$¨), είδη βιβλίων (’$memoir$,’ ’$autobiography$¨), και μορφές βιβλίων (’$paperback$¨, ’$hardback$¨). Αυτό δείχνει ότι τα *Google News vectors* κατανοούν την έννοια του βιβλίου σε διάφορα επίπεδα.

- *bank*: Και πάλι, τα αποτελέσματα είναι εξαιρετικά. Οι κοντινότερες λέξεις σχετίζονται με τον χρηματοπιστωτικό τομέα, όπως ο πληθυντικός της λέξης (’$banks$¨), η έννοια της τραπεζικής (’$banking$¨),

και συγκεκριμένες τράπεζες (*'BofA"*, *'Citibank"*). Επίσης, περιλαμβάνονται λέξεις που σχετίζονται με τις λειτουργίες των τραπεζών (*'lender"*, *'depositor"*). Αυτό δείχνει ότι τα *Google News vectors* έχουν συλλάβει την οικονομική έννοια της λέξης *'bank"*.

- *water*: Τα αποτελέσματα είναι πολύ καλά και εδώ. Οι κοντινότερες λέξεις περιλαμβάνουν τον πληθυντικό της λέξης (*'waters"*), τύπους νερού (*'potable water"*, *'groundwater"*, *'freshwater"*), και σχετικές έννοιες (*'sewage"*, *'wastewater"*). Επιπλέον, περιλαμβάνονται και πιο εξειδικευμένοι όροι όπως "Φλοριδαν αχυιφερ' και *'brackish groundwater"*, που δείχνουν την ικανότητα των *Google News vectors* να κατανοούν και πιο τεχνικές πτυχές της έννοιας.

ζ) Τα αποτελέσματα που πήραμε:

```
Top 5 similar words for the analogy 'queen : girl :: king :':
  - boy: 0.7688
  - girl: 0.6964
  - man: 0.5588
  - teenage_girl: 0.5509
  - teenager: 0.5473
Top 5 similar words for the analogy 'tall : taller :: good :':
  - better: 0.7095
  - good: 0.6328
  - taller: 0.5863
  - quicker: 0.5862
  - stronger: 0.5661
Top 5 similar words for the analogy 'paris : france :: london :':
  - london: 0.7542
  - france: 0.7367
  - england: 0.6008
  - europe: 0.5708
  - birmingham: 0.5392
```

Αναλογία $queen : girl :: king :$

Η λέξη *'boy'* που αναμέναμε εμφανίζεται ως η καταλληλότερη και με σχετικά μεγάλο βαθμό ομοιότητας.

Αναλογία $tall : taller :: good :$

Η λέξη *'better'* που αναμέναμε εμφανίζεται ως η καταλληλότερη, ενώ εμφανίζονται και άλλα επίθετα σε συγκριτικό βαθμό.

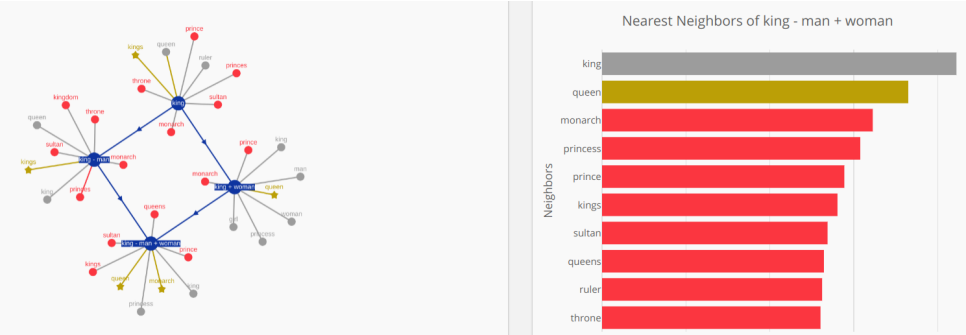
Αναλογία $paris : france :: london :$

Η λέξη *'london'* που αναμέναμε, εμφανίζεται ως 3η καταλληλότερη. Με βάση την αποτελεσματικότητα, ως τώρα, των $GoogleNewsEmbediings$ ίσως την περιμέναμε πιο ψηλά. Πάντως πάνω από το *'london'* εμφανίζονται μόνο οι λέξεις $france$, $london$

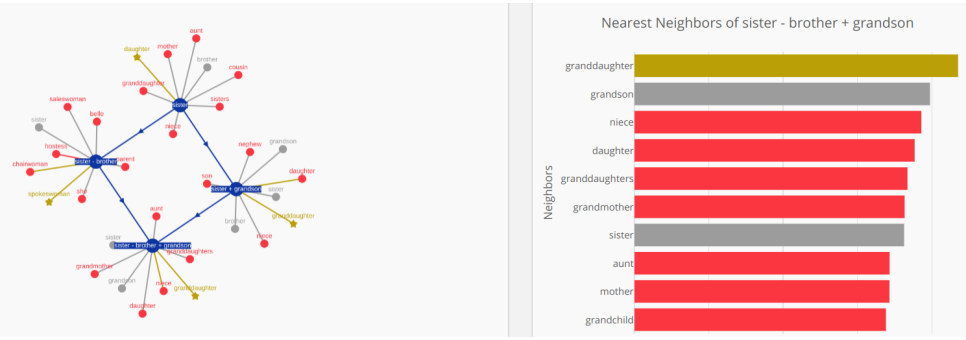
Βήμα 13: Οπτικοποίηση των word embeddings

Σε αυτό το βήμα χρησιμοποιούμε το **step13.py**

α) $king - man + woman :$



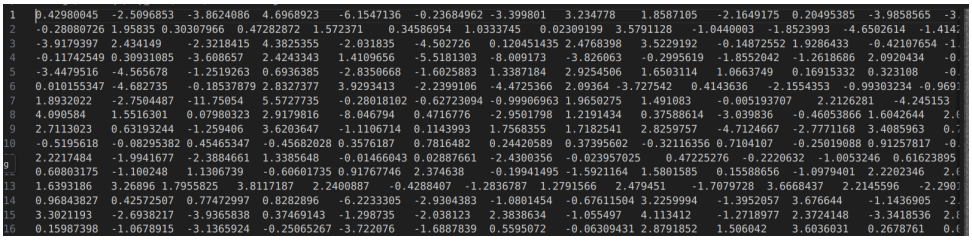
Παρατηρούμε ότι η λέξη '*queen*' που είναι διαισθητικά η καταλληλότερη είναι αυτή που βγαίνει και πρώτη, ενώ όλες οι λέξεις με μεγάλη ομοιότητα έχουν σχέση με εξουσία-βασιλεία. Επίσης σε όλες τις οπτικοποιήσεις όλων των επιμέρους αποτελεσμάτων όλες οι απαντήσεις φαίνονται να είναι αρκετά συναφείς. *sister − brother + grandson* :



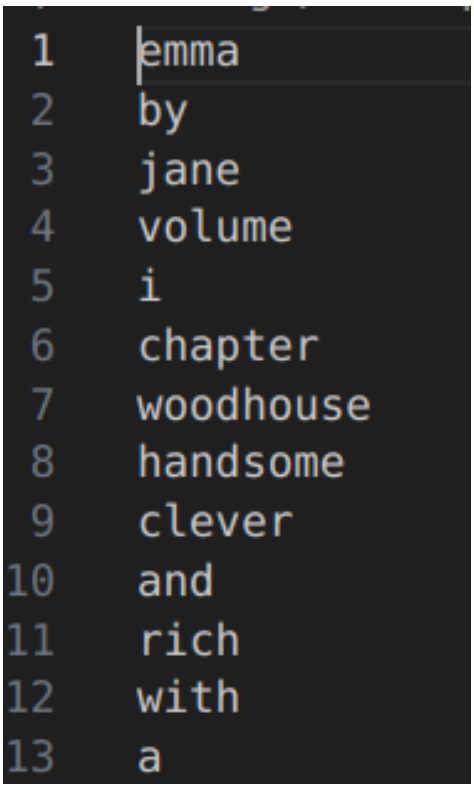
Παρατηρούμε ότι η λέξη '*grandaughter*' που είναι διαισθητικά η καταλληλότερη είναι αυτή που βγαίνει και πρώτη, ενώ όλες οι υπόλοιπες πιθανές λέξεις είναι συναφείς με σχέσεις μεταξύ μελών μιας οικογένειας και συνήθως αναφέρονται σε θηλυκά πρόσωπα. Επίσης σε όλες τις οπτικοποιήσεις όλων των επιμέρους αποτελεσμάτων όλες οι απαντήσεις φαίνονται να είναι αρκετά συναφείς.

β) Κατασκευάζουμε τα *embeddings.tsv* και *metadata.tsv* σύμφωνα με την εκφώνηση.

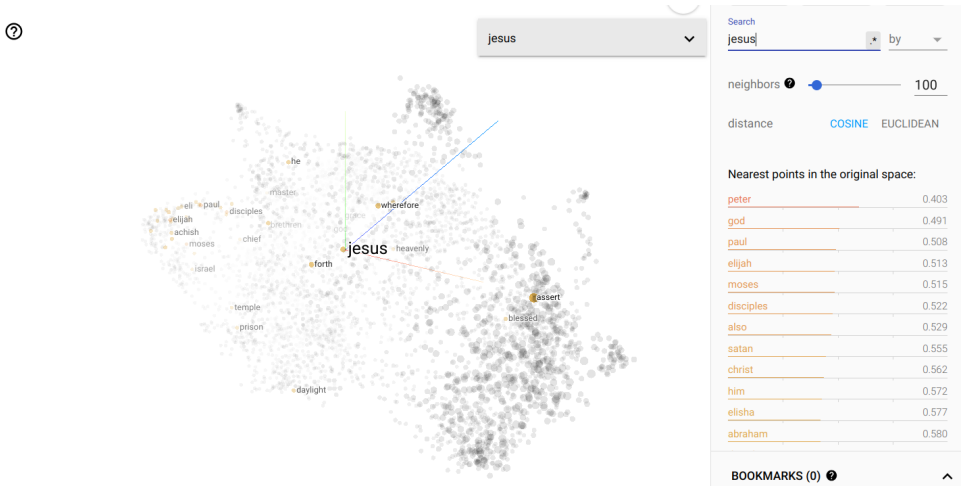
embeddings.tsv :



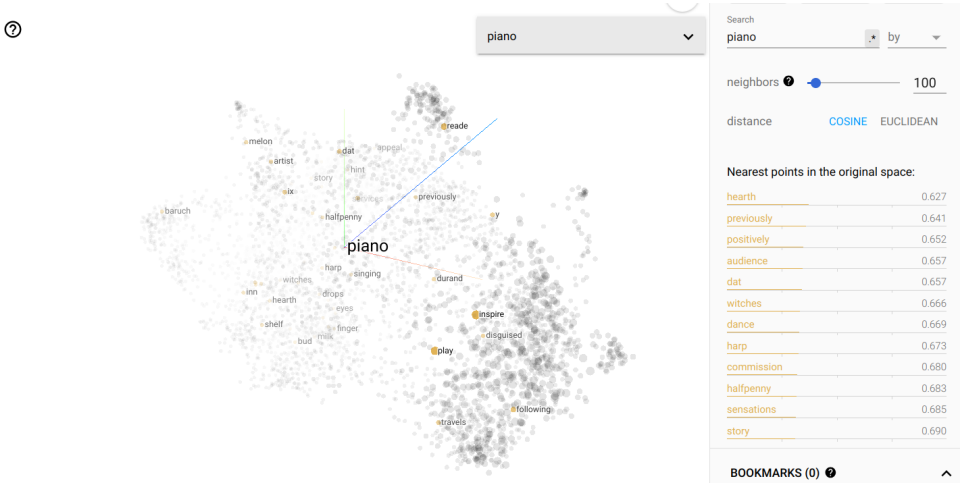
metadata.tsv :



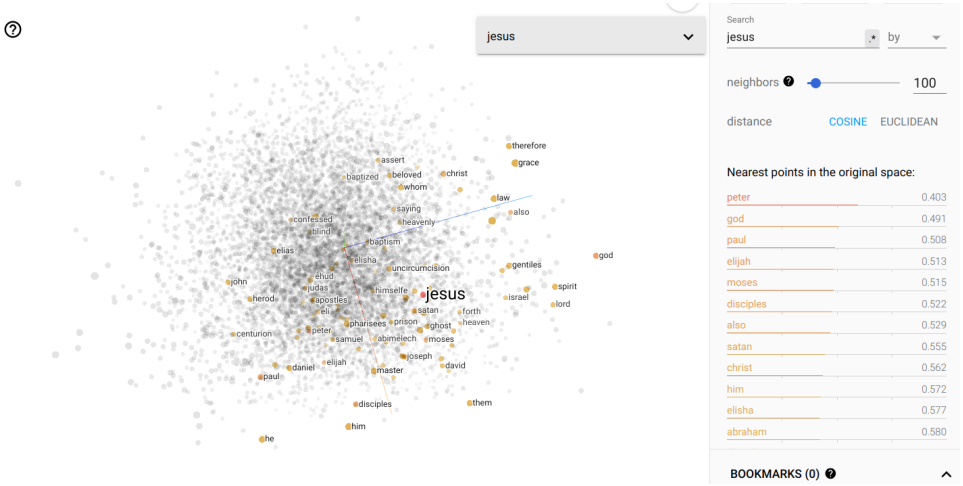
γ) λέξη: *'jesus'*: , με $UMAP$



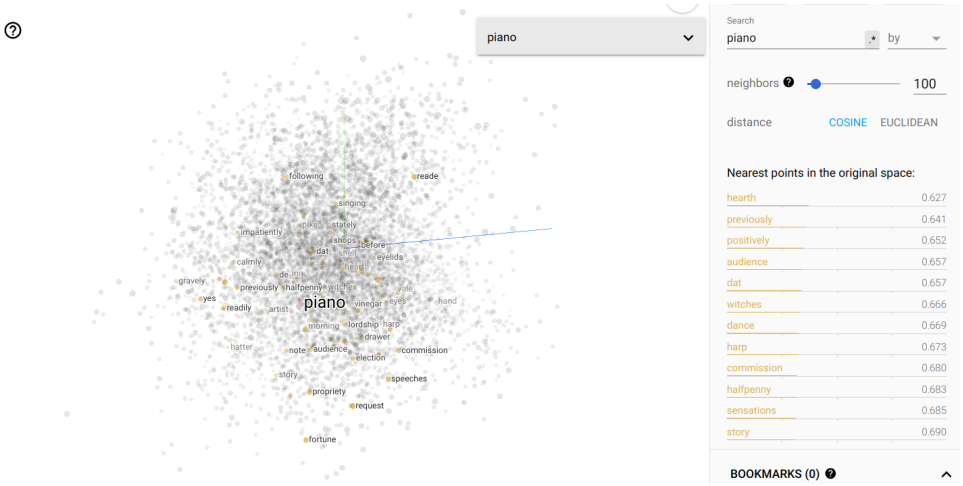
λέξη: *'piano'* , με $UMAP$



λέξη: *'jesus¨*, με PCA



λέξη: *'piano'* , με PCA



Για την λέξη *jesus* φαίνεται να είναι καλύτερη η οπτικοποίηση κάνοντας μείωση διαστατικότητας με PCA καθώς φαίνεται να βρίσκεται κοντά στο χώρο με λέξεις που έχουν σχέση με την θρησκεία και συνεπώς έχει μεγάλη ομοιότητα. Αυτο δεν συμβαίνει όταν χρησιμοποιούμε $UMAP$ για μείωση διαστατικότητας καθώς οι λέξεις σχετικές με την λέξη *jesus* βρίσκονται αρκετά μακριά από αυτή.

Για την λέξη *piano* φαίνεται αρχικά το *corpus* που χρησιμοποιήθηκε για την εκπαίδευση των *embeddings*

να μην είχε πολύ πληροφορία για αυτήν και έτσι οι οπτικοποιήσιες και με τις δύο μεθόδους να μην είναι επιτυχημένες αφού δεν εμφανίζεται κοντά σε λέξεις που έχουν σχέση με μουσική.

Βήμα 14: Ανάλυση συναισθήματος με word2vec embeddings

Σε αυτό το βήμα χρησιμοποιούμε το **w2v_sentiment_analysis.py**

α) Κατεβάζουμε τα δεδομένα που αναφέρονται στην εκφώνηση

β) Συμπληρώνουμε τον κώδικα που λείπει στο **w2v_sentiment_analysis.py**

γ) Τα αποτελέσματα για τα *embeddings* που φτιάξαμε στο Βήμα 12:

	precision	recall	f1-score	support
0	0.7248	0.7509	0.7376	2501
1	0.7414	0.7147	0.7278	2499
accuracy			0.7328	5000
macro avg	0.7331	0.7328	0.7327	5000
weighted avg	0.7331	0.7328	0.7327	5000

δ) Τα αποτελέσματα για τα $GoogleNewsVectors$:

	precision	recall	f1-score	support
0	0.8331	0.8392	0.8361	2456
1	0.8436	0.8377	0.8406	2544
accuracy			0.8384	5000
macro avg	0.8383	0.8384	0.8384	5000
weighted avg	0.8384	0.8384	0.8384	5000

Το μοντέλο που χρησιμοποιεί τα *embeddings* από το *Google News* αποδίδει σαφώς καλύτερα από το μοντέλο που χρησιμοποιεί *embeddings* που κατασκευάστηκαν από το *Gutenberg corpus*, αφού σε όλες τις μετρικές έχει υωψηλότερη τιμή. Επίσης τόσο για τα *embeddings* που κατασκευάσαμε εμείς όσο και για τα *embeddings* του *Google News* παρατηρείται ότι όλες οι μετρικές κυμαίνονται γύρω από τις ίδιες τιμές (0.72-0.75 για τα δικά μας, 0.83-0.85 για τα *Google News*) κάτι που σημαίνει πως το μοντέλο δεν είναι *biased* προς τα αρνητικά ή τα θετικά.