ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Επεξεργασία Φωνής & Φυσικής Γλώσσας
1η Εργαστηριακή Άσκηση: Εισαγωγή στις γλωσσικές αναπαραστάσεις

Μία αναφορά των φοιτητών:

- Σερλή Εμμανουήλ – Αναστάσιου (Α.Μ. 03118125)
- Αβράμη Στέφανου (Α.Μ. 03121724)

Βήμα 1:

α) Αρχικά, πραγματοποιήθηκε download και προεπεξεργασία του gutenberg corpus μέσω του δοθέντος script fetch_gutenberg.py. Το εν λόγω script αποτελείται από τις κάτωθι συναρτήσεις

- download_corpus → η βασική συνάρτηση για να γίνει download το gutenberg corpus
- preprocess_file → η βασική συνάρτηση προεπεξεργασίας του corpus
- preprocess → συνάρτηση η οποία για κάθε μη κενή γραμμή του corpus καλεί τις συναρτήσεις tokenize και clean_text
- clean_text → συνάρτηση η οποία για κάθε string εισόδου, διαγράφει trailing και πολλαπλά κενά, ενώ στην συνέχεια το μετατρέπει σε string με lowercase χαρακτήρες και στην πλήρη του μορφή (π.χ. “ isn’t ” → “is not”)
- tokenize → συνάρτηση η οποία για κάθε μη κενό string το χωρίζει στις επιμέρους λέξεις του, αφού πρώτα πραγματοποιηθεί κλήση της συνάρτησης clean_text

Το τελικό αποτέλεσμα από την κλήση των download_corpus() και preprocess_file() αποθηκεύεται σε txt file (gutenberg.txt) και αποτελείται από όλες τις προεπεξεργασμένες γραμμές του corpus.

Όπως αναφέρθηκε παραπάνω, η clean_txt() απομακρύνει τα σημεία στίξης από τις επιμέρους λέξεις του corpus. Ωστόσο, η εν λόγω τεχνική θα μπορούσε να μην είναι επιθυμητή, ανάλογα με την φύση του προβλήματος επεξεργασίας φυσική γλώσσας το οποίο καλούμαστε να επιλύσουμε (π.χ. διόρθωση και επεξεργασία συντομογραφιών ή/και αρκτικολέξεων)

β) Παρά το γεγονός ότι δεν πραγματοποιήθηκε επέκταση του gutenberg corpus στα πλαίσια της άσκησης, μια τέτοια τεχνική θα ήταν επιθυμητή για τους εξής λόγους:

- Οδηγεί στην δημιουργία dataset, όπου η συχνότητα εμφάνισης των επιμέρους tokens είναι πιο ρεαλιστική. Κάτι τέτοιο πιθανόν να μην είναι εφικτό στην περίπτωση σχηματισμού του corpus από ένα συγκεκριμένο γνωστικό αντικείμενο (π.χ. επιστήμη ή ποίηση) όπου λέξεις του εν λόγω αντικειμένου εμφανίζονται αισθητά πιο συχνά από ό,τι συνήθως.
- Καθιστά ευκολότερη την αποφυγή φαινομένων υπερπροσαρμογής των εκάστοτε NLP μοντέλων στα δεδομένα εισόδου, κάτι που ισχύει ιδιαίτερα στην περίπτωση ανάπτυξης predictive και generative models (π.χ. Word2Vec embeddings).

Βήμα 2:

Στο εν λόγω βήμα, υλοποιήθηκε η συνάρτηση word_count η οποία δέχεται σαν όρισμα το corpus του βήματος 1 και αποθηκεύει (σε μορφή dictionary) κάθε διακριτό token (που εν προκειμένω είναι μία λέξη) μαζί με την συχνότητα εμφάνισής του. Το τελικό dictionary αποθηκεύεται στην κατάλληλη μορφή στο αρχείο *vocab*/words.vocab.txt. Αξίζει να σημειωθεί πως το dictionary δεν περιλαμβάνει tokens με συχνότητα εμφάνισης μικρότερη του 5. Αυτό γίνεται για λόγους περιορισμού του μεγέθους των ορθογράφων - οι οποίοι θα υλοποιηθούν στα επόμενα βήματα - και οι οποίοι αποτελούνται από την ένωση επιμέρους αυτομάτων για κάθε token του corpus.

Βήμα 3:

Στο εν λόγω βήμα, γίνεται η δημιουργία των βασικών αρχείων εισόδου και εξόδου των Finite State Machines(FSMs) των επόμενων ερωτημάτων. Αρχικά, σχηματίζεται το αρχείο vocab/chars.syms το οποίο περιλαμβάνει κάθε lowercase γράμμα του αγγλικού αλφαβήτου με ένα ascii index, ξεκινώντας από την αντιστοίχιση a → 97. Στην συνέχεια, σχηματίζεται το αρχείο vocab/words.syms στο οποίο αντιστοιχίζεται κάθε λέξη του corpus με ένα index. Τέλος , αξίζει να σημειωθεί πως και στα 2 αρχεία περιλαμβάνεται και το κενό σύμβολο ε/<eps> με μηδενικό index.

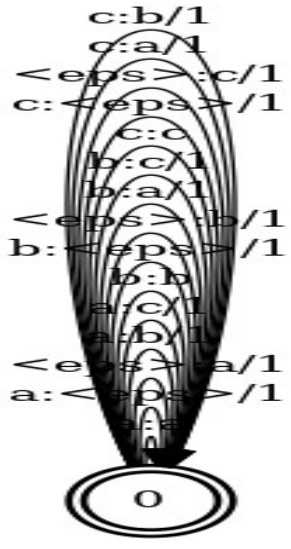
Βήμα 4:

α) Σε αυτό το βήμα, πραγματοποιείται υλοποίηση του μετατροπέα L, ο οποίος βασίζεται στην απόσταση Levenstein. Συγκεκριμένα, η εν λόγω μετρική θέτει στην μετάβαση από τον χαρακτήρα εισόδου στον εαυτό του βάρος ίσο με 0, ενώ σε όλες τις υπόλοιπες μεταβάσεις μοναδιαίο βάρος. Έτσι, αντιλαμβανόμαστε ότι στην περίπτωση χρήσης του μετατροπέα L με shortest path ίσο με 1, τότε αναμένουμε η λέξη εξόδου να ταυτίζεται με την λέξη εισόδου.

β) + γ) Στην συνέχεια, ο μετατροπέας γράφεται στην κατάλληλη μορφή στο txt αρχείο fsts/L.fst μέσω της συνάρτησης create_L() το οποίο γίνεται compile για τον σχηματισμό του executable αυτόματου fsts/L.binfst. Αξίζει να σημειωθεί πως τόσο ως είσοδος όσο και ως έξοδος του L transducer τίθεται το αρχείο vocab/chars.syms.

δ) + ε) Όπως γίνεται εύκολα αντιληπτό, ο εν λόγω transducer επιδέχεται διάφορες βελτιώσεις. Μία εξ αυτών είναι η προσθήκη περισσότερων edits, όπως των μεταβάσεων z→x ή m →n τα οποία αντιστοιχούν σε συχνά λάθη πληκτρολόγησης (λόγω της διπλανής θέσης των αντίστοιχων πλήκτρων). Άλλη μία τεχνική βελτίωσης του transducer με χρήση a-priori γνώσης θα περιλάμβανε αλλαγή των βαρών ανάλογα με την συχνότητα εμφάνισης της εκάστοτε μετάβασης στο corpus.

στ) Με χρήση της fstdraw και για ένα υποσύνολο των χαρακτήρων γίνεται απεικόνιση του transducer L παρακάτω.



Βήμα 5:

α) Σε αυτό το βήμα, πραγματοποιείται η υλοποίηση ενός ορθογράφου V, ο οποίος αποδέχεται την λέξη εισόδου εφόσον αυτή ανήκει στο corpus. Για να επιτευχθεί κάτι τέτοιο, πρέπει το βάρος των μεταβάσεων για κάθε λέξη να τεθεί στο μηδέν.

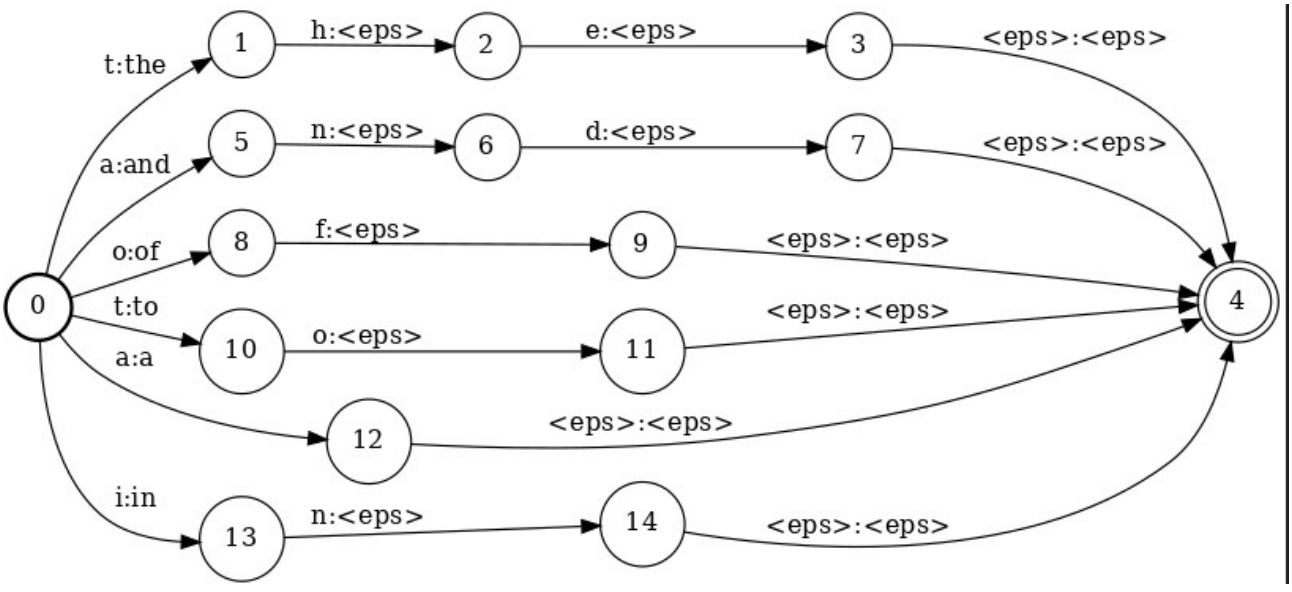
β) Για την βελτιστοποίηση του μοντέλου, πραγματοποιείται κλήση των κάτωθι συναρτήσεων της OpenFST

- fstrmepsilon: Συνάρτηση η οποία απομακρύνει τις ε-μεταβάσεις
- fstdeterminize: Συνάρτηση η οποία καθιστά το FSM ντετερμινιστικό, δηλαδή υπάρχει μόνο μία μετάβαση για κάθε ζεύγος εισόδου-κατάστασης
- fstminimize: Συνάρτηση η οποία ελαχιστοποιεί το συνολικό πλήθος μεταβάσεων του ντετερμινιστικού αυτόματου, μειώνοντας έτσι την πολυπλοκότητά του.

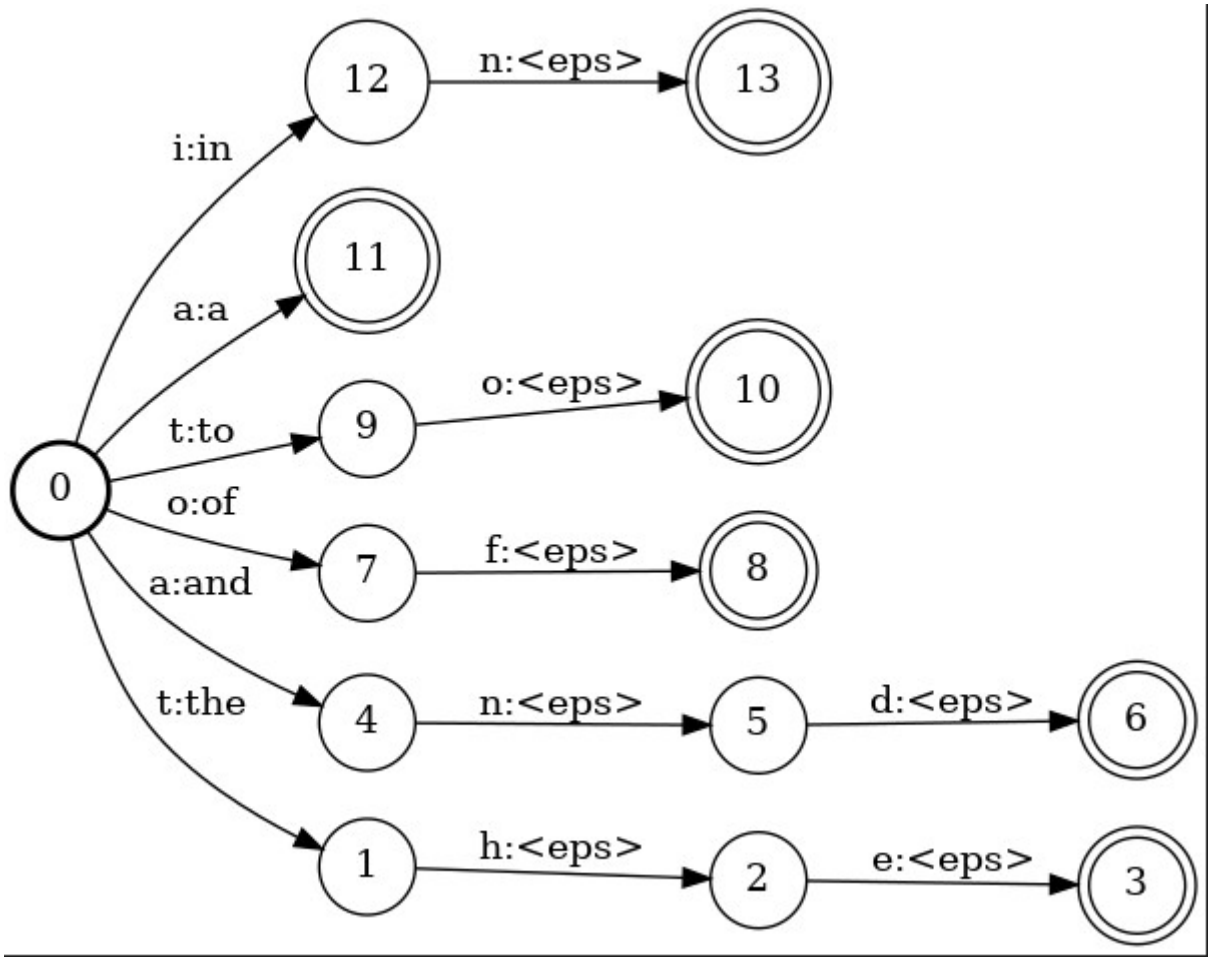
γ) + δ) Σε αντιστοίχιση με το βήμα 4, έγινε υλοποίηση του ορθογράφου μέσω κλήσης της συνάρτησης compile. Το αρχικό .fst file προς compilation δημιουργείται μέσω της συνάρτησης create_V() στο notebook του παραδοτέου.

ε) Παρακάτω ακολουθεί η οπτικοποίηση του αυτόματου του ορθογράφου έπειτα από κάθε διακριτό βήμα:

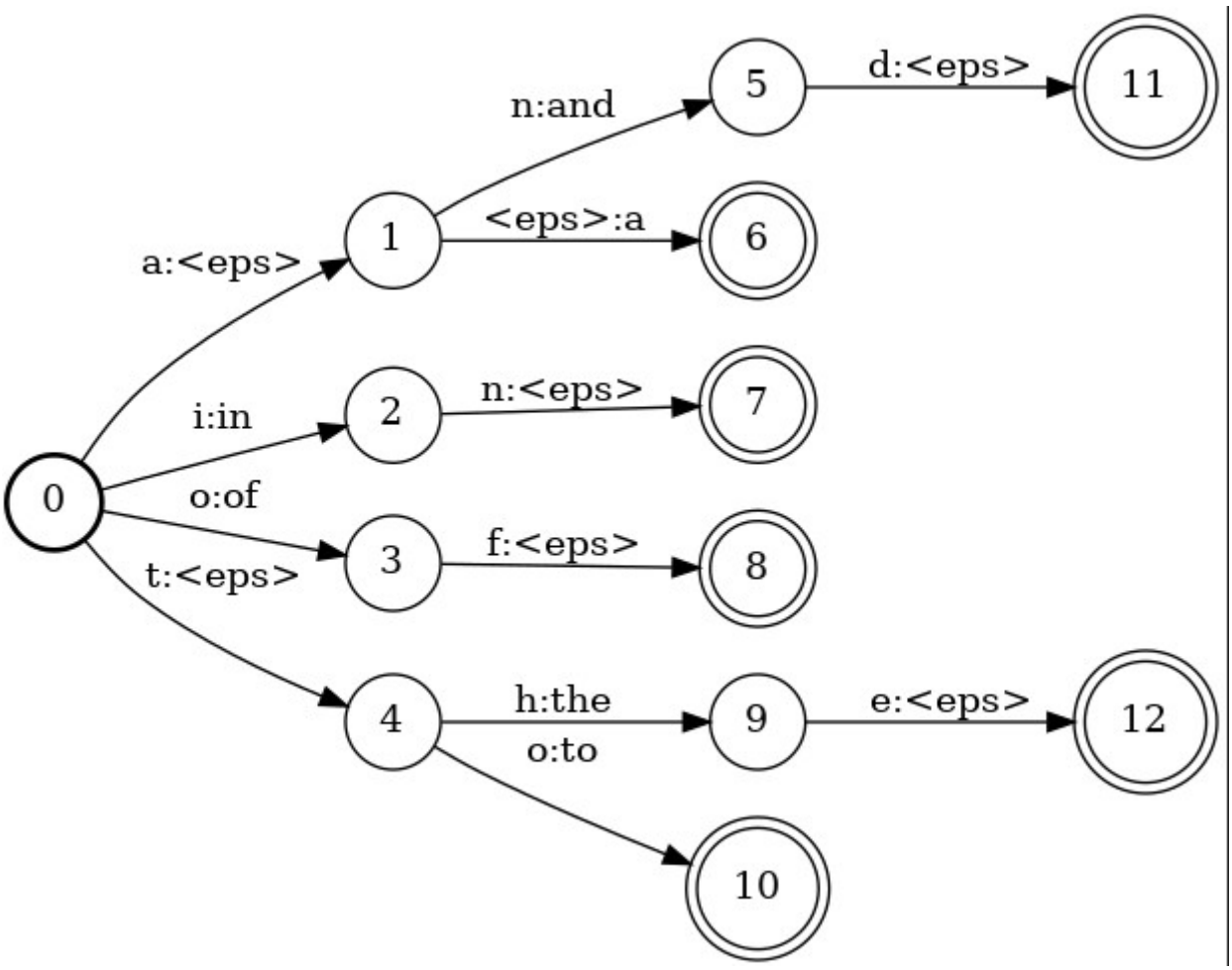
- Αρχικό αυτόματο (πριν οποιαδήποτε βελτιστοποίηση)



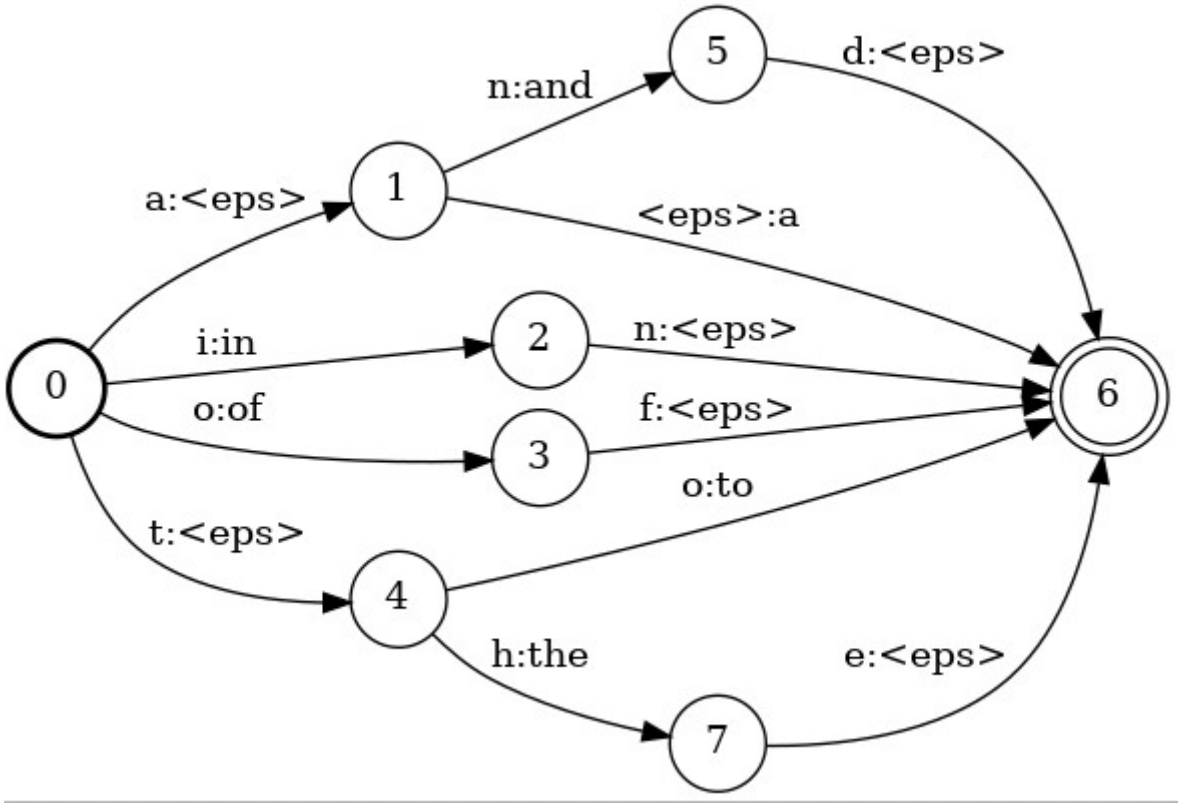
- Αυτόματο μετα την κλήση της fstrmepsilon:



- Αυτόματο μετα την κλήση της fstdeterminize:


0
1
2
3
4
5
6
7
8
9
10
11
12
a:<eps>
i:in
o:of
t:<eps>
n:and
<eps>:a
n:<eps>
f:<eps>
h:the
o:to
d:<eps>
e:<eps>

- Τελικό αυτόματο μετα την κλήση της fstminimize:



Βήμα 6:

α) Στο εν λόγω βήμα έγινε σύνθεση του transducer L του ερωτήματος 4 και του acceptor V του ερωτήματος 5 για τον σχηματισμό του ορθογράφου S. Συγκεκριμένα, αυτό επιτεύχθει μέσω της εντολής fstcompose, με σύμβολα εισόδου χαρακτήρες (vocab/chars.syms) και σύμβολα εξόδου λέξεις (vocab/words.syms). Αξίζει να σημειωθεί ότι προηγήθηκε η κλήση της συνάρτησης fstarcsort για καθένα από τα αυτόματα L και V ώστε να γίνει ορθή αντιστοίχιση των συμβόλων εξόδου του transducer στα σύμβολα εισόδου του acceptor (με βάση το indexing που ακολουθείται στο vocab/chars.syms file).

Όσον αφορά την συμπεριφορά του ορθογράφου:

1) Στην περίπτωση ισοβαρών edits, ο ορθογράφος “δώσει” την έγκυρη λέξη η οποία αντιστοιχεί στο μικρότερο συνολικό πλήθος edits.
2) Στην περίπτωση διαφορετικών βαρών των edits, θα προτιμήσει το edit με το μικρότερο βάρος, το οποίο θα μπορούσε να αντιστοιχεί σε αυξημένη πιθανότητα μετάβασης μέσω a-priori γνώσης.

β) Πιθανά edits του spell checker για καθεμία από τις λέξεις cit και cwt:

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

Βήμα 7:

Στο βήμα αυτό, πραγματοποιήθηκε αξιολόγηση του spell checker του ερωτήματος 6 σε ένα υποσύνολο 20 λέξεων του spell_test.txt. Συγκεκριμένα, υλοποιήθηκε η συνάρτηση test_spell_checker() η οποία δέχεται σαν εισόδους τον ορθογράφο προς αξιολόγηση, το σύνολο δεδομένων καθώς και το επιθυμητό υποσύνολο λέξεων. Στην συνέχεια, απομονώνεται τόσο η σωστή λέξη όσο και η πρώτη λάθος από κάθε γραμμή του set δεδομένων. Η λάθος λέξη μετατρέπεται σε fst και σε binfst μορφή (μέσω της fstcompile) ενώ μετ'έπειτα πραγματοποιείται compose των .binfsts της λανθασμένης λέξης και του ορθογράφου. Το τελικό αυτόματο (έπειτα από βελτιστοποίηση και μετατροπή σε .fst αρχείο με την fstprint) περιλαμβάνει την διορθωμένη λέξη η οποία απομονώνεται και συγκρίνεται με την σωστή για να εξαχθεί ένα συνολικό ποσοστό επιτυχίας στο υποσύνολο δεδομένων.

Παρακάτω ακολουθούν τα αποτελέσματα από το εν λόγω test.

```
correct :  wrong -> corrected

contented : contenpted -> contented
beginning : begining -> beginning
problem : problam -> problem
driven : dirven -> driven
ecstasy : exstacy -> exactly
juice : guic -> guise
locally : localy -> local
compare : compair -> compare
pronunciation : pronounciation -> pronouncing
transportability : transportibility -> respectability
minuscule : miniscule -> mince
independent : independant -> independent
arranged : aranged -> arranged
poetry : poartry -> poetry
level : leval -> level
basically : basicaly -> busily
triangular : triangulaur -> triangular
unexpected : unexpcted -> unexpected
standardizing : stanerdizing -> sneezing
variable : varable -> invariable

final accuracy score is:  55.00000000000001 %  in  20  words
```

Παρατηρούμε ότι τα τελικά ποσοστά επιτυχίας είναι σχετικά χαμηλά, λόγω της απλότητας του spell_checker που υλοποιήθηκε. Αυτό πρόκειται να βελτιωθεί μέσω της προσθήκης a-priori γνώσης από το δοθέν corpus.

Βήμα 8:

Σε αυτό το βήμα θα χρησιμοποιήσουμε το wiki.txt το οποίο περιλαμβάνει συχνά ορθογραφικά λάθη από άρθρα της Wikipedia. Θα χρησιμοποιήσουμε αυτό το αρχείο για να φτιάξουμε αποδοχείς για τις ανορθόγραφες λέξεις αλλά και για τις διορθωμένες και τέλος να τις συνδυάσουμε μαζί με το Levenstein Transducer για να φτάξουμε τον ορθογράφο. Αρχικά θα παραθέσουμε ένα παράδειγμα:

```
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ python3 mkfstinput.py abandonned > ../fsts/M.fst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ python3 mkfstinput.py abandoned > ../fsts/N.fst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ python3 mkfstinput.py abandonned > ../fsts/M.fst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ python3 mkfstinput.py abandoned > ../fsts/N.fst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ python3 mkfstinput.py abandoned > ../fsts/N.fst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ fstcompile --isymbols=../vocab/chars.syms --osymbols=../vocab/chars.syms ../fsts/N.fst > ../fsts/N.binfst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ fstarcsort --sort_type=olabel ../fsts/M.binfst ../fsts/M_sorted.binfst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ fstarcsort --sort_type=ilabel ../fsts/N.binfst ../fsts/N_sorted.binfst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ fstcompose ../fsts/M_sorted.binfst ../fsts/L_sorted.binfst ../fsts/ML.binfst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ fstarcsort --sort_type=ilabel ../fsts/ML.binfst ../fsts/ML_sorted.binfst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ fstcompose ../fsts/ML_sorted.binfst ../fsts/N_sorted.binfst ../fsts/MLN.binfst
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ fstshortestpath ../fsts/MLN.binfst | fstprint --isymbols=../vocab/chars.syms --osymbols=../vocab/chars.syms
--show_weight_one
12      11      a       a       0
0       0
1       0       <eps>   <eps>   0
2       1       <eps>   <eps>   0
3       2       d       d       0
4       3       e       e       0
5       4       n       n       0
6       5       n       <eps>   1
7       6       o       o       0
8       7       d       d       0
9       8       n       n       0
10      9       a       a       0
11      10      b       b       0
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ source word_edits.sh abandonned abandoned
n       <eps>
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$
```

Παραπάνω φαίνεται το παράδειγμα για την πρώτη λέξη (abandonned → abandoned). Το edit που έγινε είναι ότι ένα n αντικαταστάθηκε με τον κενό χαρακτήρα <eps>. Επαναλαμβάνοντας τη διαδικασία για όλες τις λέξεις μπορούμε να πάρουμε μία αρχική εκτίμηση για την συχνότητα εμφάνισης των edits και έτσι να αναθέσουμε διαφορετικό βάρος σε αυτά ανάλογα με τη συχνότητα τους. Αυτό γίνεται με τον παρακάτω κώδικα, και στις περιπτώσεις τις οποίες ένα edit δεν εμφανίζεται βάζουμε βάρος άπειρο.

In [12]:
```python
with open('../data/wiki.txt') as f:
    lines = [line.rstrip().split('\t') for line in f]
edit_dict = {}
with open("../vocab/edits.txt", "w+", newline = '') as out_f:
    for line in lines:
        x = !source word_edits.sh {line[0]} {line[1]}
        out_f.write(x[0] + '\n')
        temp = x[0].split("\t")
        try:
            key = (temp[0], temp[1])
        except:
            print(temp)
            print(line)
            continue
        if key in edit_dict:
            edit_dict[key] = edit_dict[key] + 1
        else:
            edit_dict[key] = 1
```

```
['FATAL: FstCompiler: Symbol "." is not mapped to any integer arc ilabel, symbol table = ../vocab/chars.syms, sour
= standard input, line = 2']
['aka', 'a.k.a.']
['FATAL: FstCompiler: Symbol "á" is not mapped to any integer arc ilabel, symbol table = ../vocab/chars.syms, sour
= standard input, line = 7']
['gauarana', 'guaraná']
['FATAL: FstCompiler: Symbol "1" is not mapped to any integer arc ilabel, symbol table = ../vocab/chars.syms, sour
= standard input, line = 1']
['ninties', '1990s']
```

Οι 3 διορθώσεις που περιλάμβαναν γράμματα που δεν ήταν στο αλφάβητο αγνοήθηκαν.

Βήμα 9:

Σε αυτό το βήμα θα φτιάξουμε τους ορθογράφους LVW και EVW. Ο W είναι ένας αποδοχέας οποίος αποδέχεται όλες τις λέξεις του λεξικού μας με βάρος ίσο με τον αρνητικό λογάριθμο της συχνότητας εμφάνισης της. Θα δοκιμάσουμε αυτούς το δύο ορθογράφους και θα τους συγκρίνουμε με τους EV και LV. Αρχικά ο κώδικας Python που φτιάχνει τον αποδοχέα:

Step 9

```
def create_W(word_dict, filename, num_words=-1):
    total_words = sum(word_dict.values())
    if num_words == -1:
        with open(filename, "w") as f:
            for word in word_dict:
                print(format_arc(0, 0, word, word, weight=-np.log10(word_dict[word]/total_words)), file=f)
            print(0, file=f)  # Accepting state
    else:
        current_word = 0
        with open(filename, "w") as f:
            for word in word_dict:
                if current_word >= num_words:
                    break
                print(format_arc(0, 0, word, word, weight=-np.log10(word_dict[word]/total_words)), file=f)
                current_word = current_word + 1
            print(0, file=f)  # Accepting state
create_W(out_dict, "../fsts/W_arx.fst")
```

Ύστερα θα φτιάξουμε τους ορθογράφους και θα τους κάνουμε έναν πρωταρχικό έλεγχο:

```
! fstcompile -isymbols=../vocab/words.syms -osymbols=../vocab/words.syms ../fsts/W_arx.fst ../fsts/W_arx.binfst
! fstrmepsilon ../fsts/W_arx.binfst | fstdeterminize | fstminimize > ../fsts/W.binfst
! fstprint -isymbols=../vocab/words.syms -osymbols=../vocab/words.syms ../fsts/W.binfst > ../fsts/W.fst
! fstcompile -isymbols=../vocab/words.syms -osymbols=../vocab/words.syms ../fsts/W.fst > ../fsts/W.binfst
! fstarcsort --sort_type=olabel ../fsts/W.binfst ../fsts/W_sorted.binfst

! fstarcsort --sort_type=olabel ../fsts/EV.binfst ../fsts/EV_sorted.binfst
! fstarcsort --sort_type=olabel ../fsts/S.binfst ../fsts/S_sorted.binfst
! fstcompose ../fsts/S_sorted.binfst ../fsts/W_sorted.binfst ../fsts/LVW.binfst
! fstcompose ../fsts/EV_sorted.binfst ../fsts/W_sorted.binfst ../fsts/EVW.binfst
test_spell_checker("../data/spell_test.txt", "LVW",5)
test_spell_checker("../data/spell_test.txt", "EVW",5)
```

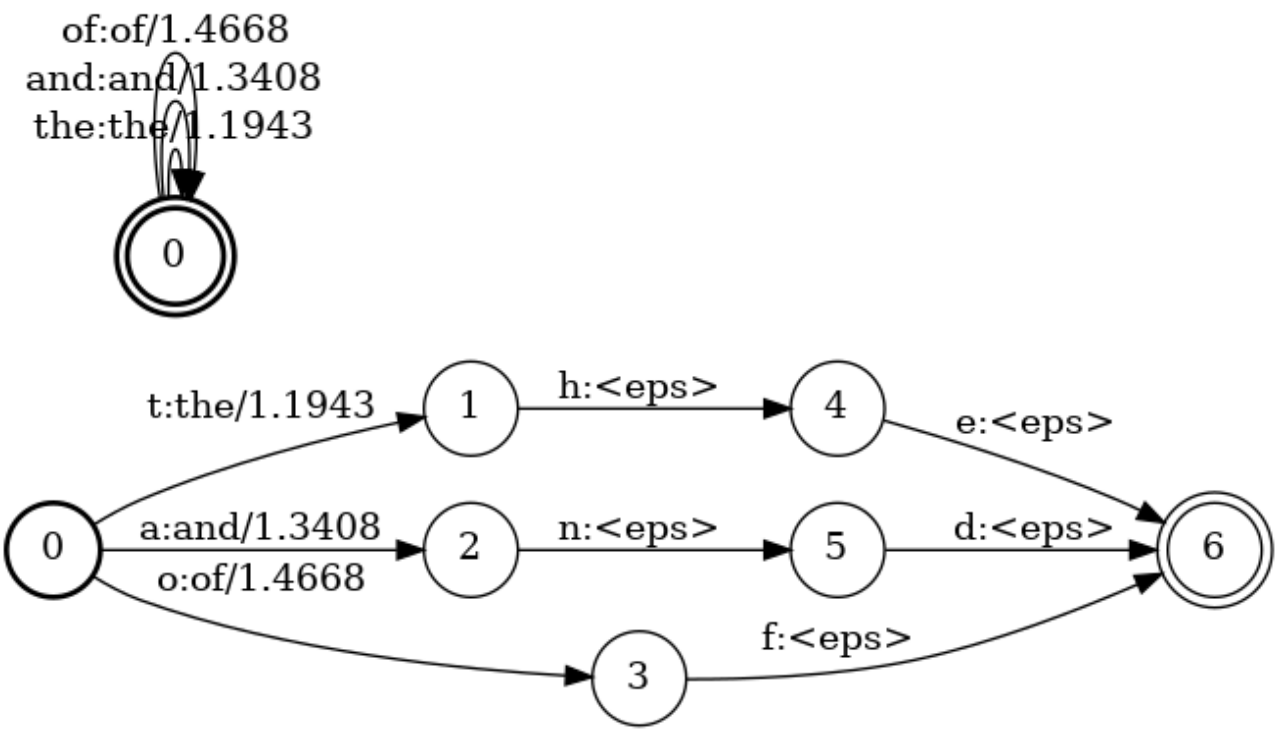
```
correct :  wrong -> corrected

contented : contenpted -> contented
contented : contende -> contend
contented : contended -> contended
contented : contentid -> contented
beginning : begining -> beginning
problem : problam -> problem
problem : proble -> people
problem : promblem -> problem
problem : proplen -> people
driven : dirven -> given
ecstasy : exstacy -> stay
ecstasy : ecstacy -> ecstasy

final accuracy score is:  50.0 %  in  5  words
```

```
contented : contenpted -> contented
contented : contende -> contend
contented : contended -> contended
contented : contentid -> contented
beginning : begining -> beginning
problem : problam -> problem
problem : proble -> problem
problem : promblem -> problem
problem : proplen -> people
driven : dirven -> driven
ecstasy : exstacy -> exactly
ecstasy : ecstacy -> ecstasy

final accuracy score is:  66.66666666666666 %  in  5  words
```

Ύστερα δοκιμάζουμε σαν input τις λέξεις cwt και cit:

```
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ source predict.sh ../fsts/S.binfst cwt
cut
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ source predict.sh ../fsts/S.binfst cit
fit
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ source predict.sh ../fsts/LVW.binfst cwt
it
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$ source predict.sh ../fsts/LVW.binfst cit
it
(NLP_venv) stefanos@stefanos-G3-3779:~/Documents/NLP_Lab/slp_labs_2023-main_NEW/lab1/scripts$
```

Βλέπουμε ότι ο LVW προτείνει τη λέξη it, πιθανότατα επειδή συχνότητα εμφάνισης της λέξης it είναι πολύ μεγαλύτερη σε σχέση με τις υπόλοιπες. Παρακάτω βλέπουμε τμήμα των W και VW:



Βήμα 10:

Χρησιμοποιώντας το run_evaluation.py θα συγκρίνουμε τους ορθογράφους του βήματος 9:
LV:

```
100%|██████████| 270/270 [12:02<00:00,  2.68s/it]
Accuracy: 0.6
```

EV:

```
100%|██████████| 270/270 [11:54<00:00,  2.65s/it]
Accuracy: 0.6962962962962963
```

LVW:

```
100%|██████████| 270/270 [11:35<00:00,  2.57s/it]
Accuracy: 0.43703703703703706
```

EVW:

```
100%|██████████| 270/270 [11:59<00:00,  2.67s/it]
Accuracy: 0.6444444444444445
```

Από τα αποτελέσματα βλέπουμε ότι ο transducer E αποτελεί βελτίωση σε σχέση με τον Levenstein. Ο αποδοχέας W, ενώ θεωρητικά θα έπρεπε να έχει καλύτερα αποτελέσματα, βλέπουμε ότι στην πράξη είναι καλύτερο να μην τον συμπεριλάβουμε στην κατασκευή του ορθογράφου. Αυτό πιθανότατα οφείλεται στο μικρό μέγεθος του corpus που χρησιμοποιήσαμε, και πιθανών ίσως να χρειάζεται κάποιου είδους κανονικοποιήση για τις λέξεις που εμφανίζονται πολύ συχνά αφού κάποιες ενδεικτίκες λάθος διορθώσεις που έκανε είναι bicycal → by, adres → are.

Βήμα 11:

Αρχικά υλοποιούμε τον add one ορθογράφο:

```
def create_E_add_one(charset, frequency_dict, filename):
    total_edits = sum(frequency_dict.values())
    V = 2*len(charset) + len(charset)*(len(charset)-1)
    with open(filename, "w") as f:
        for c in charset:
            print(format_arc(0, 0, c, c, weight=0), file=f)   # 1: Every character to itself with weight 0 (no edit)
            if (c, EPS) in frequency_dict:
                print(format_arc(0, 0, c, EPS, weight=-np.log10((frequency_dict[(c, EPS)]+1)/(total_edits+V))), file=
            else:
                print(format_arc(0, 0, c, EPS, weight=-np.log10((1)/(total_edits+V))), file=f)

            if (EPS, c) in frequency_dict:
                print(format_arc(0, 0, EPS, c, weight=-np.log10((frequency_dict[(EPS, c)]+1)/(total_edits+V))), file=
            else:
                print(format_arc(0, 0, EPS, c, weight=-np.log10((1)/(total_edits+V))), file=f)


            for other_c in charset:                          # 4: Every character to every other character with weig
                if c == other_c:
                    continue # If the characters are the same the weight is 0
                if (c, other_c) in frequency_dict:
                    print(format_arc(0, 0, c, other_c, weight=-np.log10((frequency_dict[(c, other_c)]+1)/(total_edits
                else:
                    print(format_arc(0, 0, c, other_c, weight=-np.log10((1)/(total_edits+V))), file=f)
        print(0, file=f)  # Accepting state

create_E_add_one(CHARS, edit_dict, "../fsts/E_add_one.fst")
```

Και τρέχουμε το run_evaluation:

```
100%|██████████| 270/270 [15:11<00:00,  3.38s/it]
Accuracy: 0.7
```

Ύστερα χρησιμοποιούμε το corpus από το opensubtitles για να εμπλουτίσουμε το λεξικό μας.

```
en_50 = "../data/en_50k.txt"
en_50_words = ""
with open(en_50, 'r') as file:
    en_50_txt = file.read()
en_50_txt = en_50_txt.split('\n')
for wordCount in en_50_txt[:-1]:
    word, count = wordCount.split(' ')
    en_50_words = en_50_words + word + " "
    if word in out_dict:
        out_dict[word] = out_dict[word] + int(count)
    else:
        out_dict[word] = int(count)
create_W(out_dict, "../fsts/W_2_arx.fst")
write_syms(gutenberg_txt + " " + en_50_words)
```

Και ξανατρέχουμε το run_evaluation στους νέους ορθογράφους:

```
100%|| 270/270 [31:30<00:00,  7.00s/it]
Accuracy: 0.5518518518518518
```

```
100%|| 270/270 [31:43<00:00,  7.05s/it]
Accuracy: 0.7888888888888889
```

Βλέπουμε ότι τα ποσοστά και των δύο ορθογράφων βελτιώθηκαν αρκετά. Για να βελτιώσουμε ακόμα περισσότερο τον ορθογράφο θα μπορούσαμε να τροποποιήσουμε τα βάρη των edits σύμφωνα με την κάθε λέξη και ίσως να βάζαμε μεγαλύτερο βάρος στη διαγραφή ή εισαγωγή χαρακτήρα, σε σχέση με την απλή αλλαγή.

Βήμα 12:

α) + β) Πραγματοποιήθηκε ανάγνωση του gutenberg corpus του βήματος 1, το οποίο χρησιμοποιήθηκε για την εκπαίδευση word2vec embedding μοντέλων. Συγκεκριμένα, προπονήθηκαν και αποθηκεύτηκαν 3 μοντέλα που αντιστοιχούν σε προπόνηση 10,100 και 1000 εποχών.

γ) Παρακάτω ακολουθούν τα αποτελέσματα των 3 μοντέλων, τα οποία αντιστοιχούν στις 5 πιο κοντινές λέξεις των λέξεων εισόδου bible,book,bank και water

```
model_10 results
bible  ->  [('title', 0.4902319312095642), ('republic', 0.4884238541126251), ('gentry', 0.48822328448295593), ('grog', 0.48778092861175537), ('hong', 0.4873921275138855)]
book  ->  [('written', 0.6561650037765503), ('chronicles', 0.6195775270462036), ('volume', 0.5716540217399597), ('letter', 0.5286608934402466), ('note', 0.5094970464706421)]
bank  ->  [('beach', 0.699763417243957 5), ('floor', 0.691679835319519), ('wharf', 0.6782059073448181), ('terrace', 0.6756478548049927), ('ridge', 0.673412024974823)]
water  ->  [('waters', 0.6341620683670044), ('bushes', 0.5385381579399109), ('springs', 0.5329419374465942), ('brook', 0.5273190140724182), ('rivers', 0.5260152816772461)]


model_100 results
bible  ->  [('arrum', 0.3908499777317047), ('exodus', 0.38544797897338867), ('untaken', 0.38398146629333496), ('hieroglyphics', 0.37933698296546936), ('preacher', 0.3770232796
book  ->  [('chronicles', 0.5780780911445618), ('written', 0.5374575853347778), ('comedy', 0.4829244315624237), ('papers', 0.45911532640457153), ('letter', 0.45684653520584106
bank  ->  [('ridge', 0.5572410821914673), ('pool', 0.5283218622207642), ('rocks', 0.4880446195602417), ('terrace', 0.4851661622524261 5), ('table', 0.4829841554164886 5)]
water  ->  [('waters', 0.6535888314247131), ('river', 0.597922682762146), ('sea', 0.553991436958313), ('streams', 0.5512164235115051), ('ashes', 0.5112564563751221)]

model_1000 results
bible  ->  [('davenport', 0.39520949125289917), ('proprietary', 0.3803219795227051), ('buckled', 0.3767137825489044), ('scarf', 0.3730965256690979), ('book', 0.371010124683380
book  ->  [('chronicles', 0.5138381719589233), ('written', 0.503126859664917), ('raze', 0.4715029001235962), ('temple', 0.45585402846336365), ('scedule', 0.45351463556289673)]
bank  ->  [('pool', 0.5486129522323608), ('wall', 0.5361111164093018), ('table', 0.49621516466140747), ('foot', 0.461711585521698), ('side', 0.4513593316078186)]
water  ->  [('waters', 0.6290057301521301), ('river', 0.5684349536895752), ('sea', 0.5539792776107788), ('liquid', 0.5368390679359436), ('streams', 0.5314413905143738)]
```

Παρατηρούμε ότι τα αποτελέσματα των μοντέλων βελτιώνονται με την αύξηση των εποχών (π.χ. στην περίπτωση των λέξεων bank και book) ενώ καθίσταται και πιο ομοιόμορφη η κατανομή πιθανότητας μεταξύ των top5 κοντινότερων λέξεων. Παρόλα αυτά, όπως θα φανεί και σε επόμενο ερώτημα, υπάρχουν πολύ πιο σχετικές λέξεις τις οποίες τα άνωθι μοντέλα αδυνατούν να βρουν (κυρίως λόγω χαμηλής τιμής παραθύρου). Μία πιθανή λύση για βελτίωση των μοντέλων θα ήταν είτε η αύξηση του παραθύρου εκπαίδευσης είτε εναλλακτικά η επέκταση του corpus εκπαίδευσης.

δ) Παρακάτω ακολουθούν τα αποτελέσματα για τις ζητούμενες τριπλέτες λέξεων

```
model_10 results
['girls', 'queen', 'kings']  ->  [('men', 0.5662633776664734), ('peasants', 0.5399688482284546), ('creatures', 0.5313206315040588)]
['good', 'tall', 'taller']  ->  [('better', 0.4536677896976471), ('worse', 0.4290015697479248), ('wiser', 0.4288576245307922 4)]
['france', 'paris', 'london']  ->  [('england', 0.533356785774231), ('highbury', 0.5147027969360352), ('parish', 0.47496679425239563)]

model_100 results
['girls', 'queen', 'kings']  ->  [('men', 0.5164756774902344), ('moralists', 0.4826558232307434), ('dwellings', 0.4695836901664734)]
['good', 'tall', 'taller']  ->  [('better', 0.5012955665588379), ('comfort', 0.3999123573303222 7), ('fool', 0.39102932810783386)]
['france', 'paris', 'london']  ->  [('sussex', 0.4982496500015259), ('england', 0.44794729351997375), ('inn', 0.4472661018371582)]

model_1000 results
['girls', 'queen', 'kings']  ->  [('parts', 0.49015381932258606), ('men', 0.46490103006362915), ('cities', 0.4524120092391968)]
['good', 'tall', 'taller']  ->  [('better', 0.4607517719268799), ('pleased', 0.3884280025959015), ('encouragement', 0.3777715563774109)]
['france', 'paris', 'london']  ->  [('england', 0.485245943069458), ('sussex', 0.42492932081222534), ('town', 0.42455434799194336)]
```

Παρατηρούμε ότι στο εν λόγω task, τα w2v μοντέλα ανταποκρίνονται ικανοποιητικά καλά. Μάλιστα, η αύξηση των εποχών προπόνησης, οδηγεί σε επιδύνωση των αποτελεσμάτων λόγω overfitting στα περιορισμένα δεδομένα του gutenberg corpus.

ε) Κατεβάζουμε το ζητούμενο GoogleNews Vector μοντέλο

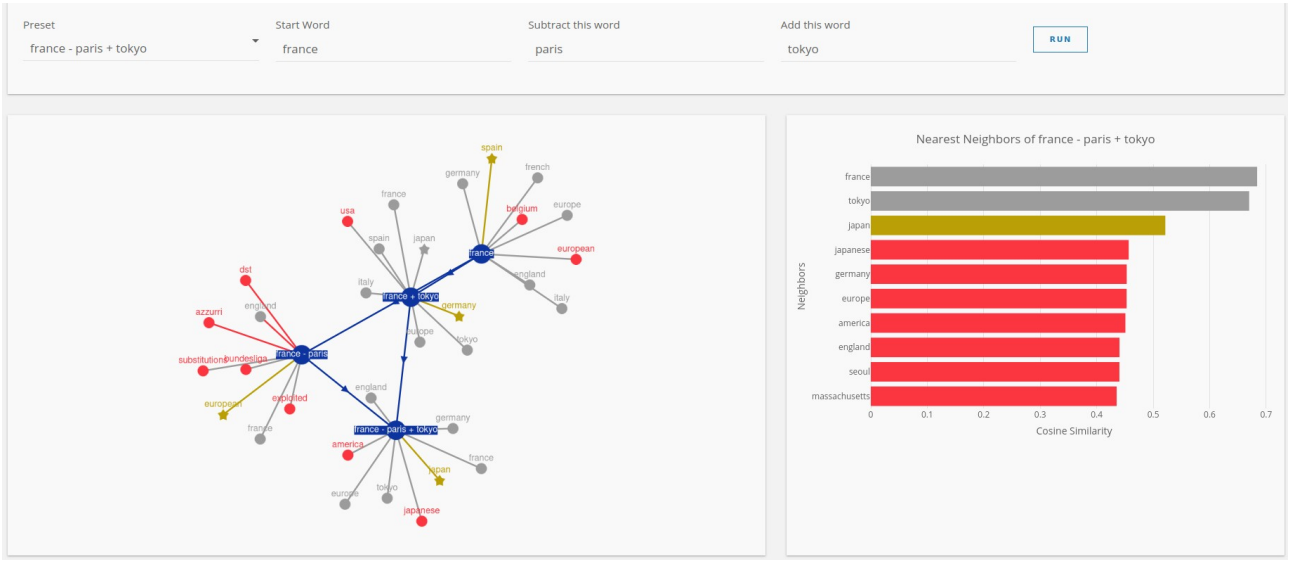
στ) + ζ) Επαναλαμβάνουμε τα 2 άνωθι tests των υποερωτημάτων 12δ και 12ε για το GoogleNews Vector μοντέλο, με τα αποτελέσματα να ακολουθούν παρακάτω:

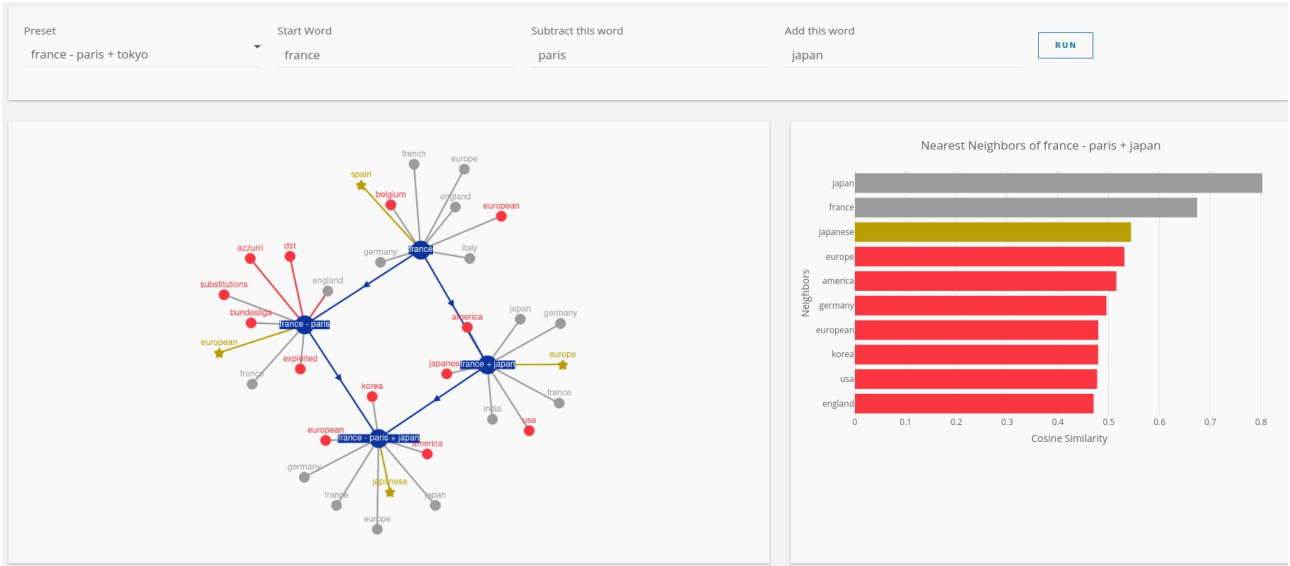
```
model_google results for test1
/tmp/ipykernel_45027/4239279239.py:3: DeprecationWarning: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  voc = model.wv.index2word
/tmp/ipykernel_45027/4239279239.py:6: DeprecationWarning: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  sim = model.wv.most_similar(word,topn=5)
bible  ->  [('Bible', 0.7367781400680542), ('bibles', 0.6052597761154175), ('Holy_Bible', 0.5989601612091064), ('scriptures', 0.574568510055542), ('scripture', 0.5697901248931
book  ->  [('tome', 0.7485830783843994), ('books', 0.7379178404808044), ('memoir', 0.7302927374839783), ('paperback_edition', 0.6868364810943604), ('autobiography', 0.67415279
bank  ->  [('banks', 0.7440759539604187), ('banking', 0.690161406993866), ('Bank', 0.6698698997497559), ('lender', 0.6342284679412842), ('banker', 0.6092953681945801)]
water  ->  [('potable_water', 0.6799106597900391), ('Water', 0.6706870794296265), ('sewage', 0.6619377136230469), ('groundwater', 0.6588345766067505), ('Floridan_aquifer', 0.6

model_google results for test2
['girls', 'queen', 'kings']  ->  [('boys', 0.7183727025985718), ('men', 0.5000519752502441), ('teenagers', 0.49677687883377075)]
['good', 'tall', 'taller']  ->  [('better', 0.7245721220970154), ('nicer', 0.5474838018417358), ('great', 0.5431875586509705)]
['france', 'paris', 'london']  ->  [('england', 0.5836853384971619), ('europe', 0.5529575347900391), ('birmingham', 0.5180004835128784)]
```

Παρατηρούμε ότι το τελευταίο μοντέλο έχει αισθητά βελτιωμένα αποτελέσματα σε σχέση με τα αρχικά w2v μοντέλα. Αυτό έγκειται τόσο στην αύξηση του συνόλου δεδομένων όσο και στην 300 διάστατων παραθύρων (σε σύγκριση με τα 100 διάστατα παράθυρα των προηγούμενων μοντέλων)

Βήμα 13:

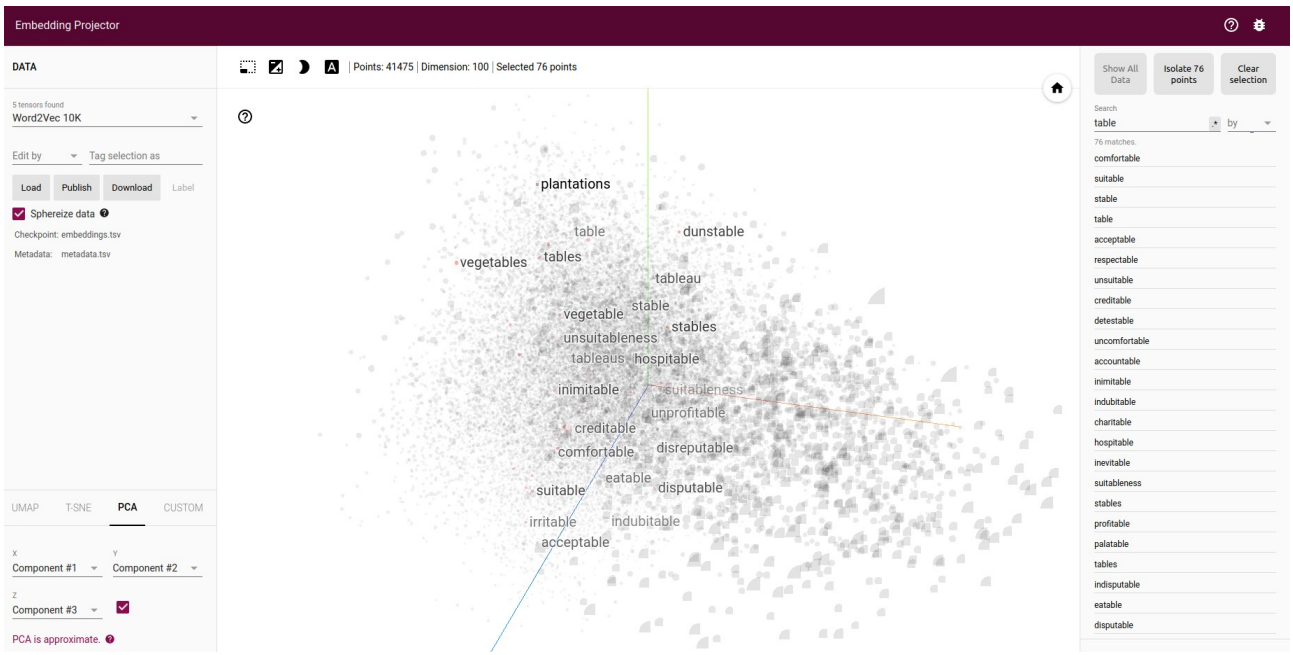
α) Έγινε χρήση του online εργαλείου από το οποίο εκλάβαμε τα κάτωθι αποτελέσματα

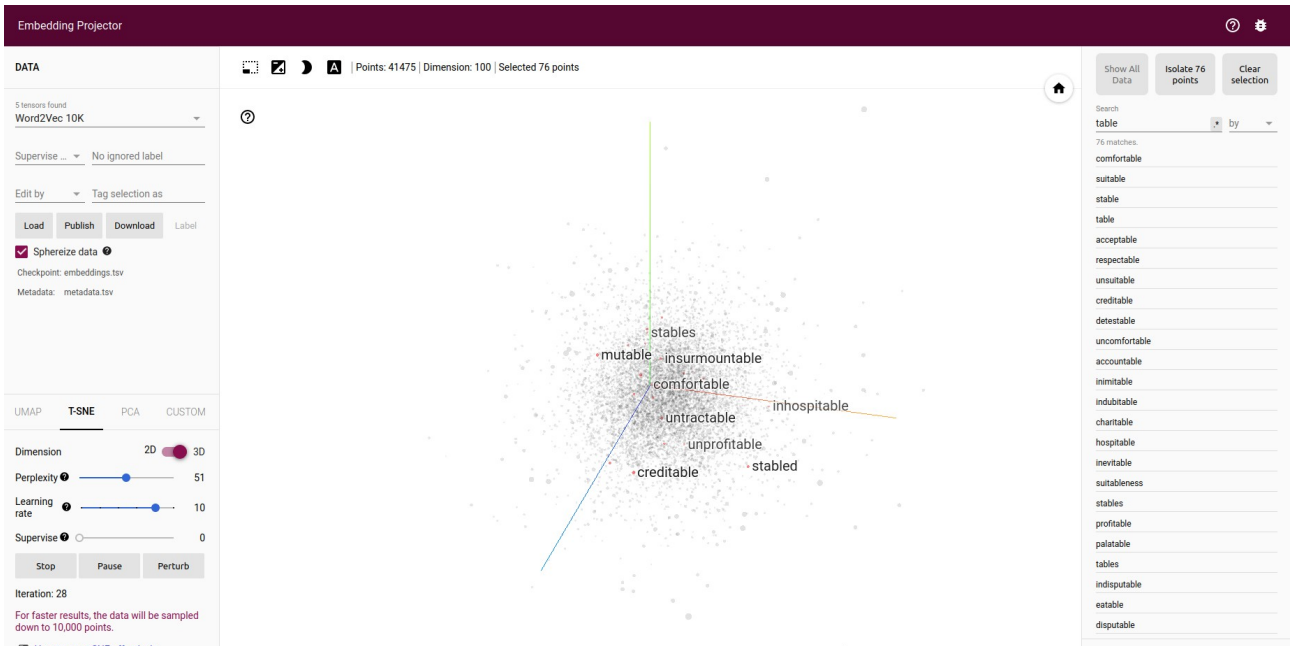




Παρατηρούμε ότι το εργαλείο βρίσκει σε κάθε περίπτωση επιτυχώς τον κοντινότερο γείτονα του δοθέντος preset, με την χρήση cosine similarity (αντίστοιχο test και αποτέλεσμα των ερωτημάτων 12δ και 12ζ)

β) + γ) Αφού δημιουργήθηκαν τα ζητούμενα αρχεία και εισήχθησαν στον embedding projector, έγινε εύρεση λέξεων σχετικών με την λέξη εισόδου μέσω 2 τεχνικών μείωσης διαστατικότητας (PCA & T-SNE). Εξάγαμε τα κάτωθι αποτελέσματα:





Παρατηρούμε ότι και οι 2 τεχνικές οδηγούν σε αρκετά παρεμφερή αποτελέσματα συγγενικών λέξεων (table → comfortable, suitable, stable κ.α.). Ωστόσο, η εν λόγω συσχέτιση είναι περισσότερο συντακτική και γραμματική παρά νοηματική.

Βήμα 14:

α) + β) Μέσω υλοποίησης και συμπλήρωσης του script w2v_sentiment_analysis.py, έγινε λήψη των acImdb δεδομένων και προεπεξεργασία των δεδομένων. Έτσι, σχηματίστηκαν τα Neural BOWs για κάθε πρόταση τόσο στο training όσο και στο test set, που αποτελεί τον μέσος όρο των w2v διανυσμάτων των λέξεων κάθε πρότασης.

γ) + δ) Τα άνωθι NBOWs χρησιμοποιήθηκαν ως training & testing δεδομένα σε ένα Logistic Regression μοντέλο για την αναγνώριση θετικών και αρνητικών συναισθημάτων. Έγινε χρήση των word embeddings του ερωτήματος 12, με τα accuracy scores επί του test set να είναι τα παρακάτω:

LR with initial word embeddings → 74.75%
LR with GoogleNews Vector word embeddings → **83.47%**