ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Επεξεργασία Φωνής και Φυσικής Γλώσσας

1η Εργαστηριακή Άσκηση: Εισαγωγή στις γλωσσικές αναπαραστάσεις

ΑΝΑΦΟΡΑ

Μαρκος Γιαννόπουλος 03118103

Νικόλαος Καραφύλλης 03119890

ΜΕΡΟΣ 1: ΚΑΤΑΣΚΕΥΗ ΟΡΘΟΓΡΑΦΟΥ

Βήμα 1: Κατασκευή corpus

α)

Εκτελούμε το script fetch_gutenberg.py και αποθηκευουμε το αποτέλεσμα του στο αρχείο corpus.txt :

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/fetch_gutenberg.py > data/
corpus.txt
[nltk_data] Downloading package gutenberg to
[nltk_data]     /home/nickarafyllis/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
```

Βήματα προεπεξεργασίας:
1.αφαίρεση των leading / trailing spaces (τα κενά πριν από τον πρώτο και μετά από τον τελευταίο χαρακτήρα κάθε γραμμής)
2. μετατροπή σε πεζά (lowercase)
3. επέκταση συνενώσεων/συντομογραφιών (contractions expanding)
4. αφαίρεση πολλαπλών κενών
5. διατήρηση μόνο πεζών γραμμάτων και κενών (άρα χωρίς σημεία στίξης, ειδικούς χαρακτήρες και αριθμούς)

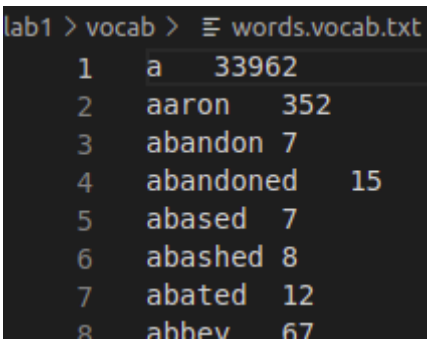
Θα θέλαμε να διατηρήσουμε τα σημεία στίξης σε περιπτώσεις όπως:
1. δυσκολία προσδιορισμού κατάφασης ή ερώτησης ,
2. εντοπισμός ενδιάμεσων προτάσεων
2. συναισθηματική ανάλυση του κειμένου, εντοπισμός ειρωνείας κλπ (“!”, “...”)+/”
?

β) Μπορούμε να συνενώσουμε περισσότερα βιβλία για να επεκτείνουμε το corpus, με αυτόν τον τρόπο θα έχουμε τα πλεονεκτήματα :

1. καλύτερη γενίκευση του γλωσσικού μοντέλου, με το να έχουμε περισσότερα βιβλία και κείμενα από διαφορετικούς συγγραφείς, πηγές και θεματολογία αποφεύγουμε την υπερπροσαρμογή
2. έχουμε μεγαλύτερο όγκο δεδομένων και άρα οι στατιστικές μέθοδοι θα έχουν καλύτερες προσεγγίσεις.

Βήμα 2: Κατασκευή λεξικού

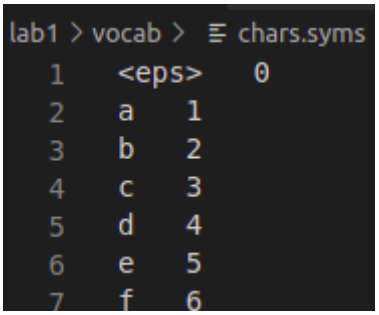
α,β,γ) Εκτελούμε το bash scripts/make_dict.sh, οπότε έχουμε:



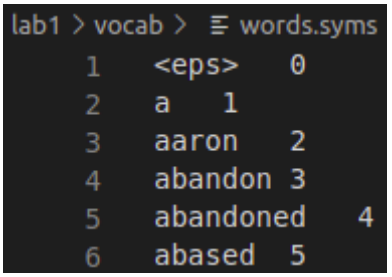
Το βήμα του φιλτραρίσματος των tokens βάσει του αριθμού εμφάνισης χρειάζεται για να αφαιρέσουμε λέξεις που εμφανίζονται σπάνια και ανεβάζουν την πολυπλοκότητα του μοντέλου μας, αποφεύγοντας ταυτόχρονα και την υπερπροσαρμογή.

Βήμα 3: Δημιουργία συμβόλων εισόδου/εξόδου

α) Εκτελούμε το python scripts/create_chars_syms.py, οπότε έχουμε:



β) Εκτελούμε το script bash scripts/create_word_syms.sh vocab/words.vocab.txt vocab/words.syms:



Βήμα 4: Κατασκευή μετατροπέα edit distance

Εκτελούμε το bash scripts/4.sh, που χρησιμοποιεί την συνάρτηση make_L.py για το βήμα α :

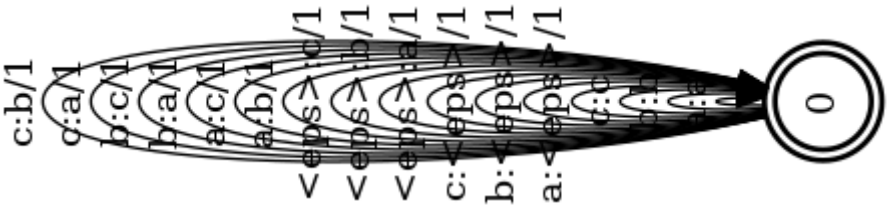
```
#!/bin/bash

# 4a,b
# write the fst file
python scripts/make_L.py > ./fsts/L.fst

# 4c
# compile the fst to bin
fstcompile --isymbols=./vocab/chars.syms --osymbols=./vocab/chars.syms ./fsts/L.fst ./fsts/L.binfst

# draw the FST and save as png
fstdraw --isymbols=./vocab/chars.syms --osymbols=./vocab/chars.syms ./fsts/L.binfst | dot -Tpng > ./fsts/L.png
```

Αν πάρουμε το shortest path σε μια λέξη εισόδου, προφανώς θα μας επιστρέψει την ίδια ακριβώς λέξη, αφού θα ακολουθήσει τις ακμές όπου τα βάρη είναι 0.

Θα μπορούσαμε να συμπεριλάβουμε και ένα 4ο edit, της μετάθεσης 2 γειτονικών χαρακτήρων (transposition) -όπως άλλωστε χρησιμοποιείται στην Damerau–Levenshtein distance- λαμβάνοντας υπόψη ότι κατα την πληκτρολόγηση είναι συχνό φαινόμενο να πατηθούν πλήκτρα με λάθος σειρά(π.χ. λόγω έλλειψης συντονισμού αριστερού-δεξιού χεριού ή δακτύλων).

Δεδομένου ότι σε ένα πληκτρολόγιο οι θέσεις των πλήκτρων είναι συγκεκριμένες θα μπορούσαμε να έχουμε διαφορετικά βάρη αντικατάστασης για κάθε ζευγάρι χαρακτήρων ανάλογα με την απόσταση των χαρακτήρων στο πληκτρολόγιο(π.χ. 0.5 βάρος στην αντικατάσταση r με t, αλλά 1.5 στην αντικατάσταση a με p). Κάτι αντίστοιχο θα μπορούσε να γίνει και για την διαγραφή.

Αν στην make_L.py ορίσουμε: `alphabet = 'abc'`, τότε το L γίνεται:



Βήμα 5: Κατασκευή αποδοχέα λεξικού

Η υλοποίηση των ερωτημάτων α,β,γ,δ γίνεται στο script 5.sh :
Εκτελείται με την εντολή bash scripts/5.sh vocab/words.syms

Η fstrmepsilon αφαιρεί τις ε μεταθέσεις(από ε σε ε).

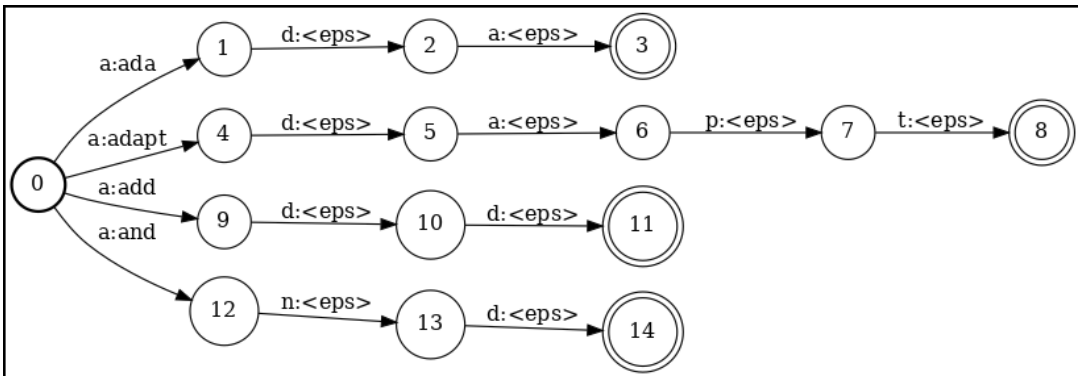
Η fstdeterminize κάνει το fst ντετερμινιστικό, ώστε καμία κατάσταση να μην έχει 2 ακμές με την ίδια είσοδο.

Η fstminimize ελαχιστοποιεί τις καταστάσεις και ακμές του αυτόματου.

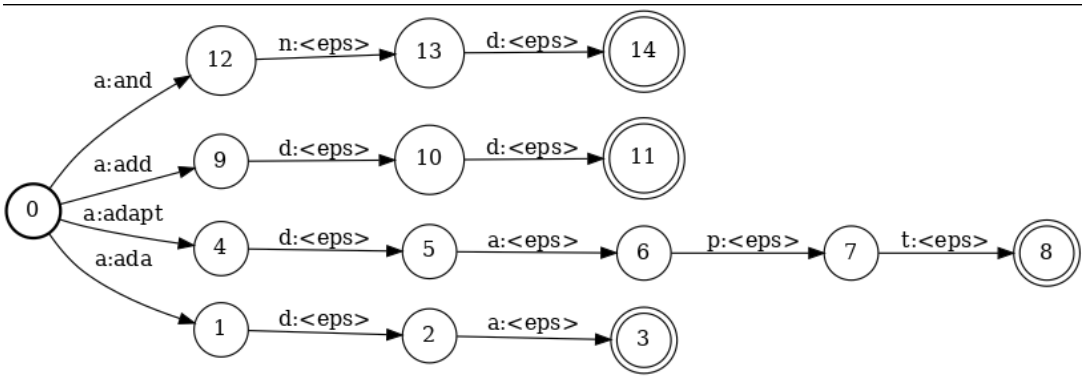
Το όφελος αυτών των βελτιστοποιήσεων είναι ότι θα έχουμε λιγότερες καταστάσεις και ακμές, οπότε θα έχει μικρότερη πολυπλοκότητα το μοντέλο(και επομένως μεγαλύτερη ταχύτητα και μικρότερη ανάγκη υπολογιστικού χώρου).
Άλλωστε οι ακμές και η πολυπλοκότητα διάβασης(traversal) σε ένα ντετερμινιστικό αυτόματο είναι πολύ λιγότερες από ότι σε ένα μη-ντετερμινιστικό, όπως θα δούμε παρακάτω.

Η υλοποίηση του ερωτήματος ε γίνεται στο script 5e.sh, στο οποίο χρησιμοποιούμε το υποσύνολο λέξεων testwords.syms οπότε προκύπτουν οι ακόλουθες εικόνες:

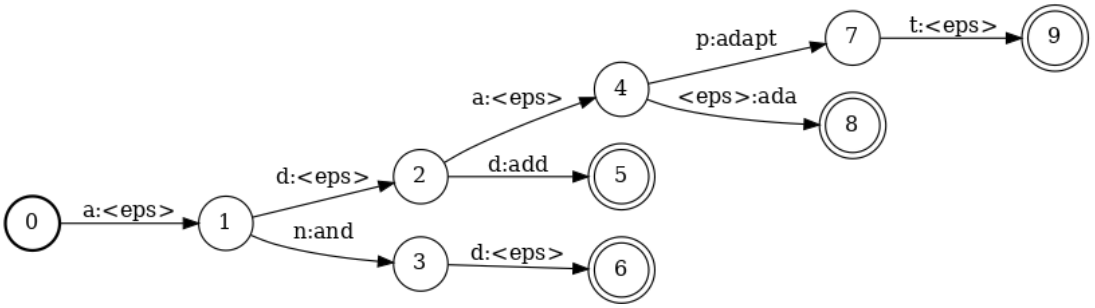
1. πριν τις βελτιστοποιήσεις



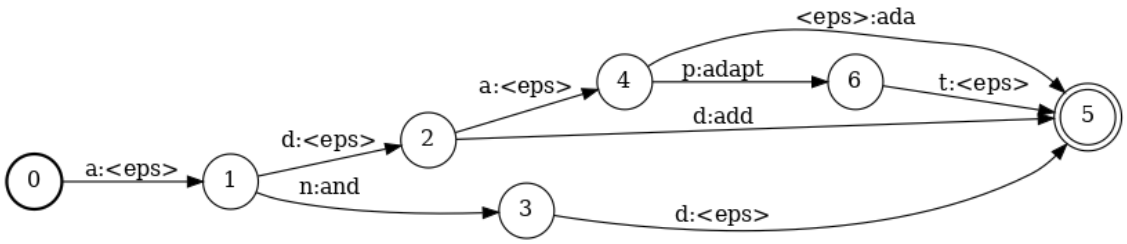
2. fstrmepsilon(δεν υπάρχουν ε μεταθέσεις, οπότε δεν τροποποιείται το διάγραμμα)



3. fstdeterminize(π.χ. όλες οι λέξεις έχουν α-μετάβαση στην αρχή οπότε ενοποιούνται)



4. fstminimize (όλες οι τελικές καταστάσεις ενοποιήθηκαν σε μία)



Παρατηρούμε ότι μετά τις βελτιστοποιήσεις μειώθηκε σημαντικά το μέγεθος και η πολυπλοκότητα του αυτομάτου.

Βήμα 6: Κατασκευή ορθογράφου

α) Εκτελούμε το script composer.sh :

```
#!/bin/bash

# 6a
# sort outputs of transducer
fstarcsort --sort_type=olabel fsts/L.binfst fsts/L_sort.binfst

# sort inputs of acceptor
fstarcsort --sort_type=ilabel fsts/V_opt.binfst fsts/V_sorted.binfst

# compose transducer and acceptor to create min edit distance spell checker
fstcompose fsts/L_sort.binfst fsts/V_sorted.binfst fsts/S.binfst
```

Η συμπεριφορά αυτού του μετατροπέα είναι πολύ naive καθώς δεν λαμβάνει υπόψη του γλωσσικά χαρακτηριστικά αλλά και άλλη πρότερη γνώση όπως αυτά που προτείναμε στο 4δ,ε.
Στην περίπτωση όμως που δεν ήταν ισοβαρής ο μετατροπέας, θα μπορούσε να λάβει υπόψη αυτή την πρότερη γνώση(γλωσσική, φωνολογική ανάλογα με την πηγή των δεδομένων) ώστε να είναι πολύ πιο αποτελεσματικός.

β) Κάποιες πιθανές προβλέψεις του min distance spell checker για τις λέξεις cit, cwt:
με μία αλλαγή(κόστος 1)
αντικατάσταση: fit, hit, lit, mit, cat, cut
αφαίρεση: it
προσθήκη: city,

Δοκιμάζουμε τον spell checker για τις λέξεις cit, cwt με χρήση της συνάρτησης predict.sh :
bash scripts/predict.sh fsts/S.binfst cwt

```
(slplab) nickaraf:~/slplab/lab1$ bash scripts/predict.sh fsts/S.binfst cit
clit(slplab) nickaraf:~/slplab/lab1$ bash scripts/predict.sh fsts/S.binfst cwt
cut(slplab) nickaraf:~/slplab/lab1$
```

Αν φτιάξω τον S με unoptimized V προκύπτουν διαφορετικά αποτελέσματα, τα οποία απέχουν όμως την ίδια uniform Levenstein απόσταση:

```
(slplab) nickaraf:~/slplab/lab1$ bash scripts/predict.sh fsts/S.binfst cit
wit(slplab) nickaraf:~/slplab/lab1$ bash scripts/predict.sh fsts/S.binfst cwt
cat(slplab) nickaraf:~/slplab/lab1$
```

Βήμα 7: Δοκιμή ορθογράφου

Εκτελούμε το script 7.sh για να αξιολογήσουμε τον S για τις πρώτες 20 λέξεις:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/7.sh fsts/S.binfst
contented: contented contended contended contented
beginning: beginning
problem: problem problem problem people
driven: given
ecstasy: ecstasy ecstasy
juice: guil june june guise june
locally: local
compare: company
pronunciation: provocation
transportability: respectability
minuscule: ridicule
independent: independent independent
arranged: ranged arranged
poetry: party poetry poetry poets poetry
level: legal
basically: sickly
triangular: triangular
unexpected: unexpected unexpected unexpected
standardizing: standing
variable: parable
Accuracy: .46
```

Παρατηρούμε ότι ο spell checker έχει αρκετά μεγάλη ακρίβεια σε σχέση με την απλότητα υλοποίησης του.

Το script predict.sh είναι ένα pipeline που διορθώνει την λανθασμένη λέξη με βάση έναν spell checker:
Φτιάχνει ένα fst για την (λανθασμένη) λέξη που δέχεται ως είσοδο, το μεταγλωτίζει και το συνθέτει με τον spell checker. Στη συνέχεια βρίσκει το μικρότερο μονοπάτι, αφαιρεί τις έψιλον μεταθέσεις, σορτάρει τα arcs τοπολογικά, το τυπώνει και απομονώνει την λέξη αποτέλεσμα(την διορθωμένη).

ΤΕΛΟΣ ΠΡΟΠΑΡΑΣΚΕΥΗΣ

ΜΕΡΟΣ 1

Βήμα 8: Υπολογισμός κόστους των edits

Το αρχείο wiki.txt περιέχει συχνά ορθογραφικά λάθη από την Wikipedia.

Για τα βήματα α)-γ), τρέχουμε το script word_edits.sh που μας δίνεται με είσοδο μόνο το ζευγάρι ανορθόγραφης-διορθωμένης λέξης abandonned->abandoned.

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/word_edits.sh abandonned abandoned
n       <eps>
```

Δημιουργεί έναν αποδοχέα M για την ανορθόγραφη λέξη και έναν αποδοχέα N για την διορθωμένη.
Τους συνθέτει διαδοχικά με τον Levenshtein transducer, δημιουργώντας τον MLN.
Στη συνέχεια βρίσκει το συντομότερο μονοπάτι από την ανορθόγραφη στη σωστή λέξη με την εντολή fstshortestpath, το τυπώνει με την εντολή fstprint, κρατάει μόνο τις γραμμές με μη μηδενικό βάρος με την εντολή grep -v "0$" και κρατάει μόνο τις στήλες 3,4 που έχουν το source και destination symbol με την εντολή cut.
Το αποτέλεσμα που δίνει στο παραπάνω παράδειγμα είναι η μετάβαση από το n στο <eps>
Άλλα παραδείγματα εκτέλεσης:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/word_edits.sh abundacies abundances
i       c
c       n
```

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/word_edits.sh specif specific
<eps>   c
<eps>   i
```

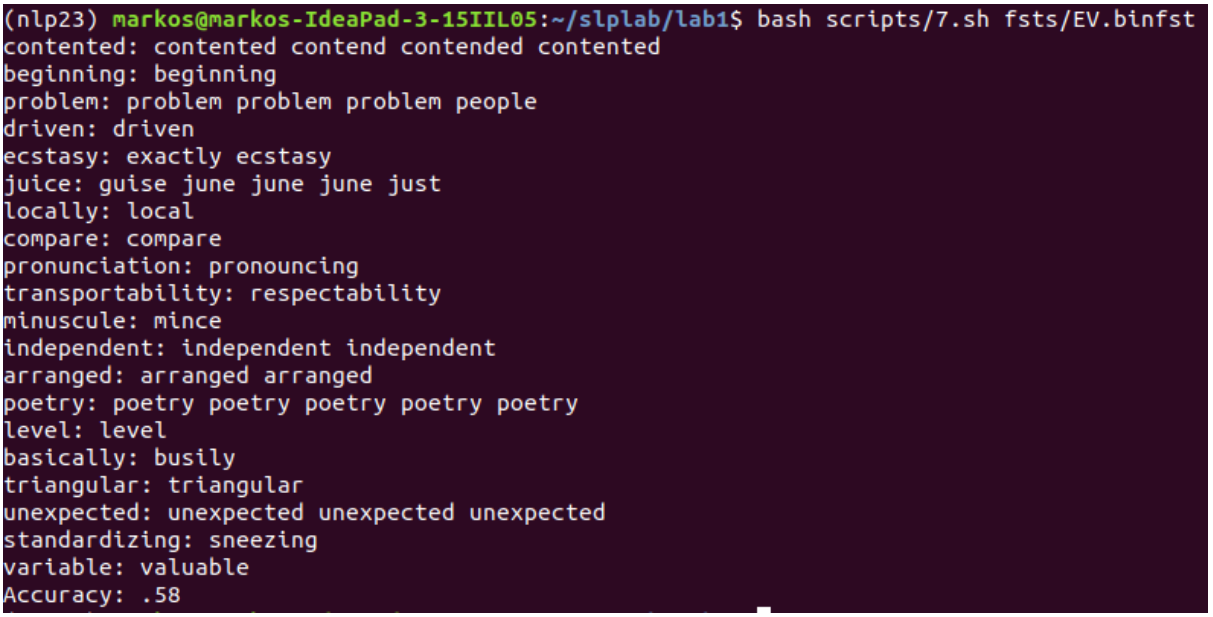
```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/word_edits.sh soveits soviets
i       e
e       i
```

Τρέχουμε το 4_for_8.sh, το οποίο κάνει τα εξής:

- Εκτελεί την προηγούμενη διαδικασία για κάθε γραμμή του wiki.txt, ώστε να παράξει όλα τα δυνατά edits και τα γράφει στο αρχείο data/files.txt. Εξαιρούνται κάποιες γραμμές στις οποίες υπάρχουν διαφορετικοί χαρακτήρες (π.χ. “.”).
- Μετράει το πλήθος των εμφανίσεων κάθε edit και αποθηκεύει το αποτέλεσμα σε ένα λεξικό με keys τις τούπλες (source,destination).
- Κατασκευάζει έναν μετατροπέα edit distance E όπως στο βήμα 4, μόνο που τώρα τα βάρη των edits δεν είναι ομοιόμορφα αλλά ισούνται με τον αρνητικό λογάριθμο της συχνότητας εμφάνισης τους στο λεξικό. Αν δεν εμφανίζονται καθόλου βάζουμε ένα πολύ μεγάλο βάρος.

Για να επαναλάβουμε το βήμα 6, τρέχουμε το composer_for_8.sh και κατασκευάζουμε τον spell checker EV συνθέτοντας τον transducer E με τον αποδοχέα V.

Για το βήμα 7, δηλαδή το evaluation του νέου spell checker για τις 20 πρώτες λέξεις, τρέχουμε το script 7.sh με είσοδο τον EV:



Δηλαδή αυξήθηκε σε σχέση με τον naive spell checker.

Βήμα 9: Εισαγωγή της συχνότητας εμφάνισης λέξεων (Unigram word model)
Για όλο το βήμα 9, τρέχουμε το script 9.sh :
bash scripts/9.sh vocab/words.vocab.txt vocab/words.syms
β) Εκτελεί το make_acceptor_word_freq.py, το οποίο κατασκευάζει 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, όπως αυτή είχε υπολογιστεί στο βήμα 2. Στη συνέχεια τον μεταγλωττίζει και τον βελτιστοποιεί, φτιάχνοντας το W_opt.binfst.

γ) Συνθέτουμε τον μετατροπέα LVW, ο οποίος είναι ένας ορθογράφος που λαμβάνει υπόψιν του την συχνότητα των λέξεων.

δ) Συνθέτουμε τον μετατροπέα EVW, ο οποίος είναι ένας ορθογράφος που λαμβάνει υπόψιν του τόσο την συχνότητα των λέξεων, όσο και την συχνότητα των edits.

ε) Αξιολογούμε τον LVW, όπως στο βήμα 7:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/7.sh fsts/LVW.binfst
contented: the the the and
beginning: beginning
problem: of the the people
driven: the
ecstasy: the the
juice: the the the the the
locally: and
compare: and
pronunciation: unto
transportability: and
minuscule: the
independent: and and
arranged: and and
poetry: the the the the the
level: the
basically: and
triangular: and
unexpected: and and and
standardizing: and
variable: the
Accuracy: .02
```

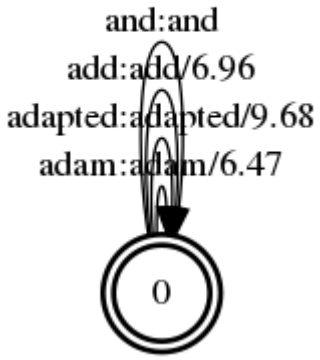
Παρατηρούμε ότι για τις περισσότερες λέξεις προβλέπει λανθασμένα “and” και “the”, αφού είναι οι πιο συχνές λέξεις στο corpus. Επομένως η αποτελεσματικότητα του είναι πολύ χειρότερη από τον απλό spell checker.

στ) Συγκρίνουμε τις προβλέψεις του LVW και του LV (S) για τις λέξεις “cwt” και “cit”:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/predict.sh fsts/LVW.binfst cwt
the(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/predict.sh fsts/LVW.binfst cit
it(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/predict.sh fsts/S.binfst cwt
cut(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/predict.sh fsts/S.binfst cit
clit(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$
```

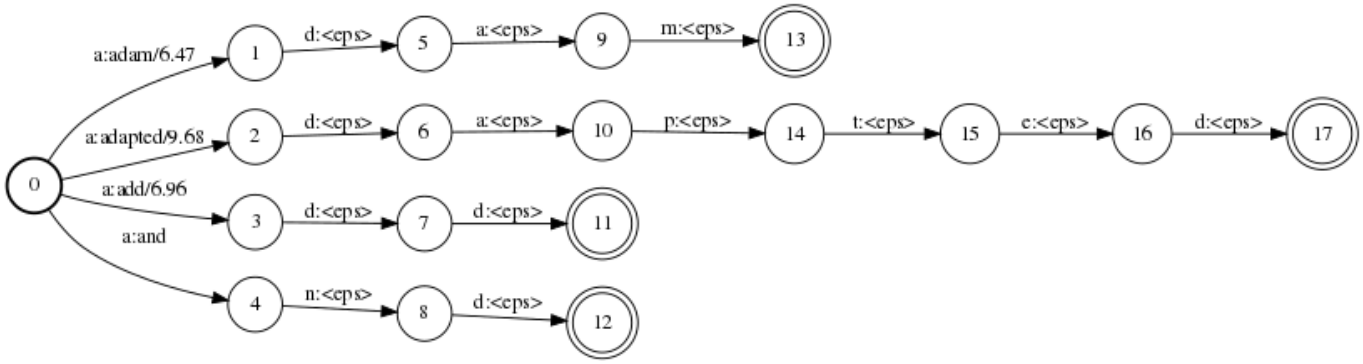
Πάλι φαίνεται ότι ο LVW επιλέγει πιο συχνές λέξεις (“the”,”it”) ακόμα και αν απέχουν αρκετά edits. Ο LV επιλέγει πάντα κάποια λέξη που προκύπτει με ελάχιστα edits.

ζ) Σχεδιάζουμε με την fstdraw τον W και τον VW για ένα μικρό υποσύνολο λέξεων.
W:



Το and έχει βάρος 0 γιατί η συχνότητα εμφάνισης του σε σχέση με τις υπόλοιπες 3 λέξεις είναι τεράστια.

VW:



Σημειώνεται ότι στο παράδειγμα για την οπτικοποίηση το fst δεν είναι optimal.

Βήμα 10: Αξιολόγηση των ορθογράφων

Για να αξιολογήσουμε τον ορθογράφο LV (δηλαδή τον S), τρέχουμε την εντολή python3 scripts/run_evaluation.py fsts/S.binfst.

Το accuracy που μας επιστρέφει για όλο το spell_test.txt είναι:

```
Accuracy: 0.5851851851851851
```

Όμοια αξιολογούμε τον LVW:

```
Accuracy: 0.044444444444444446
```

Λόγω της προτίμησης του στις συχνότερες λέξεις, τα αποτελέσματα του είναι πάρα πολύ κακά.

Αξιολογούμε τον EV:

```
Accuracy: 0.674074074074074
```

Παρατηρούμε ότι με την προσθήκη πληροφορίας για την συχνότητα των edits, ο ορθογράφος βελτιώθηκε.

Αξιολογούμε τον EVW:

```
Accuracy: 0.6370370370370371
```

Πάλι σε σχέση με τον απλό EV, το accuracy έπεσε με την προσθήκη της πληροφορίας για την συχνότητα των λέξεων. Όμως η μείωση είναι μικρή και σε κάποιες λέξεις στις οποίες ο EV έκανε λάθος, ο EVW βρίσκει σωστό αποτέλεσμα.

Βήμα 11: (Bonus) Βελτιώσεις του ορθογράφου

α) Στο βήμα 8 χρησιμοποιήσαμε πολύ μεγάλα βάρη στα edits με μηδενική συχνότητα εμφάνισης.

Τώρα για να έχουμε πιο ομαλά βάρη σε αυτά τα edits, χρησιμοποιούμε Add-1 smoothing σύμφωνα με το βιβλίο των Jurafsky-Martin. (https://web.stanford.edu/~jurafsky/slp3/3.pdf).

Το νέο πλήθος του κάθε edit δίνεται από τον τύπο:

$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

όπου ci το προηγούμενο πλήθος, N το συνολικό πλήθος των edits και V το πλήθος των unique edits.
Δημιουργούμε και αξιολογούμε το fst EV11.binfst τρέχοντας τις παρακάτω εντολές:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/4_for_11.sh
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/composer_for_11.sh
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ python3 scripts/run_evaluation.py fsts/EV11.binfst
```

Το αποτέλεσμα της αξιολόγησης είναι:

```
Accuracy: 0.6851851851851852
```

Δηλαδή υπήρξε μια πολύ μικρή βελτίωση σε σχέση με τον EV χωρίς add-1 smoothing.

β) Κατεβάζουμε διαφορετικά λεξικά από άλλες πηγές για να δούμε αν υπάρχουν διαφορές στα αποτελέσματα.
curl
https://raw.githubusercontent.com/hermitdave/FrequencyWords/master/content/2016/en/en_50k.txt > vocab/en_50k.txt
Με αυτή την εντολή κατεβάσαμε ένα λεξικό που προέκυψε από υπότιτλους από το opensubtitles.
curl
https://raw.githubusercontent.com/IlyaSemenov/wikipedia-word-frequency/master/results/enwiki-2022-08-29.txt > vocab/enwiki.txt
Με αυτή την εντολή κατεβάσαμε ένα λεξικό που προέκυψε από την αγγλική Wikipedia.

Φτιάχνουμε τα αντίστοιχα αρχεία του βήματος 3 για τα νέα λεξικά:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/create_word_syms.sh vocab/en_50k.txt vocab/words_en_50k.syms
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/create_word_syms.sh vocab/enwiki.txt vocab/words_enwiki.syms
```

Φτιάχνουμε το νέο V.fst για το πρώτο από τα νέα λεξικά με την εντολή:
bash scripts/5.sh vocab/words_en_50k.syms
Με τις παρακάτω εντολές φτιάχνουμε νέο EV και το αξιολογούμε:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/composer_for_11.sh
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ python3 scripts/run_evaluation.py fsts/EV.binfst
```

```
Accuracy: 0.6851851851851852
```

Όπως αναμενόταν δεν άλλαξε κάτι γιατί δεν χρησιμοποιήσαμε την πληροφορία για την συχνότητα των λέξεων στο λεξικό.
Δοκιμάζουμε το EVW:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/9.sh vocab/en_50k.txt vocab/words_en_50k.syms
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ python3 scripts/run_evaluation.py fsts/EVW.binfst
```

```
Accuracy: 0.6703703703703704
```

Το αποτέλεσμα είναι καλύτερο από το προηγούμενο EVW αλλά χειρότερο από το EV.

Κάνουμε τα ίδια για το άλλο, μεγαλύτερο λεξικό:

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/5.sh vocab/words_enwiki.syms
```

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/composer_for_11.sh
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ python3 scripts/run_evaluation.py fsts/EV.binfst
```

```
Accuracy: 0.6851851851851852
```

```
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ bash scripts/9.sh vocab/enwiki.txt vocab/words_enwiki.syms
(nlp23) markos@markos-IdeaPad-3-15IIL05:~/slplab/lab1$ python3 scripts/run_evaluation.py fsts/EVW.binfst
```

```
Accuracy: 0.6703703703703704
```

Δηλαδή τα αποτελέσματα ήταν τα ίδια και για αυτό το λεξικό.

ΜΕΡΟΣ 2: Εξοικείωση με το W2V

Βήμα 12: Εξαγωγή αναπαραστάσεων word2vec

α,β) Τροποποιούμε την συνάρτηση w2v_train που μας δίνεται ώστε να δημιουργήσει μια λίστα από tokenized προτάσεις και στη συνέχεια να εκπαιδεύσει το μοντέλο Word2Vec πάνω σε αυτή τη λίστα, και να το αποθηκεύσει ως "gutenberg_w2v.1000e.5w.model".

```
(slplab) nickaraf:~/slplab$ python scripts/w2v_train.py
```

γ) Εκτελούμε το script test_model.py :

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/test_model.py models/gutenberg_w2v.1000e.5w.model
bible
gaming 0.3984750211238861
respects 0.39616912603378296
relaxed 0.37580406665802
propitious 0.37397709488868713
wrist 0.36933085322380066
book
letter 0.527445912361145
written 0.5126458406448364
note 0.4692584276199341
temple 0.4617908298969269
pen 0.45216208696365356
bank
floor 0.5262984037399292
table 0.5223300457000732
top 0.5171929597854614
side 0.49434855580329895
wall 0.49354124069213867
water
waters 0.6004678606987
blood 0.5125412940979004
river 0.4998939037322998
wave 0.4801769256591797
oil 0.4715138077735901
```

1. Παρατηρούμε ότι τα αποτελέσματα είναι αποδοτικά, αλλά σίγουρα δεν είναι αρκετά ικανοποιητικά και χρειάζονται μεγάλες βελτιώσεις ώστε να μπορούν να χρησιμοποιηθούν.

2. Αν μεγαλώσουμε το παράθυρο, τα αποτελέσματα γίνονται πιο ‘out of context’:

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/test_model.py models/gutenberg_w2v.1000e.10w.model
bible
ebony 0.3965035378932953
chattering 0.3801114559173584
intermission 0.36909785866737366
venerable 0.35540539026260376
impossibility 0.3552953898906708
book
letter 0.4999278783798218
temple 0.47804027795791626
pen 0.4538153409957886
mouth 0.4333064556121826
written 0.4282667636871338
bank
table 0.5081517100334167
edge 0.49056053161621094
floor 0.4756288230419159
top 0.475234717130661
wall 0.469192236661911
water
waters 0.6110764741897583
blood 0.5119513869285583
sea 0.4656442701816559
wood 0.46501579880714417
wine 0.45411404967308044
```

Αν μεγαλώσουμε τον αριθμό των εποχών, τα αποτελέσματα είναι παρόμοια με τα αρχικά:

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/test_model.py models/gutenberg_w2v.2000e.5w.model
bible
bondman 0.38629674911499023
convict 0.3853670060634613
intermission 0.3579338490962982
respects 0.3515641689300537
nevertheless 0.3491015136241913
book
written 0.5178571343421936
letter 0.5113521814346313
temple 0.47532138228416443
papers 0.43626493215560913
mouth 0.4356008470058441
bank
side 0.5331422090530396
top 0.5186131596565247
table 0.5167802572250366
wall 0.5119251608848572
floor 0.4952813684940338
water
waters 0.5847479104995728
wine 0.5085524320602417
blood 0.48105117678642273
wood 0.4729791283607483
river 0.4717206656932831
```

3. Το corpus εκπαίδευσης είναι περιορισμένο οπότε δεν έχουμε μεγάλες βελτιώσεις + οι διαστάσεις των embeddings παραμένει 100.
4. Το πρώτο που θα κάναμε για να βελτιώσουμε τα αποτελέσματα θα ήταν να αποκτήσουμε ένα πολύ μεγαλύτερο corpus.
   Θα μπορούσαμε επίσης να μεγαλώσουμε και τις διαστάσεις των embeddings.

δ)

Εκτελούμε την συνάρτηση analogies.py για τα τριπλέτες λέξεων που μας υποδυκνείονται:

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/analogies.py girls queen kings
parts
(slplab) nickaraf:~/slplab/lab1$ python scripts/analogies.py good taller tall
handsome
(slplab) nickaraf:~/slplab/lab1$ python scripts/analogies.py france paris london
england
```

και έχουμε το σωστό αποτελέσμα μόνο στην δοκιμή “france-paris+london”

ε,στ)
Η υλοποίηση βρίσκεται στο script google_test.py

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/google_test.py
bible
Bible 0.736778199672699
bibles 0.6052597761154175
Holy_Bible 0.5989601612091064
scriptures 0.574568510055542
scripture 0.5697901844978333
book
tome 0.7485831379890442
books 0.7379177808761597
memoir 0.7302927374839783
paperback_edition 0.6868364214897156
autobiography 0.6741527915000916
bank
banks 0.7440759539604187
banking 0.6901614069938666
Bank 0.6698698401451111
lender 0.6342284679412842
banker 0.6092953085899353
water
potable_water 0.6799106597900391
Water 0.6706871390342712
sewage 0.6619377136230469
groundwater 0.6588346362113953
Floridan aquifer 0.6422533988952637
```

Προφανώς τα αποτελέσματα του μοντέλου της google είναι πολύ πιο ποιοτικά λόγω του όγκου των δεδομένων στα οποία το έχουν εκπαιδεύσει.

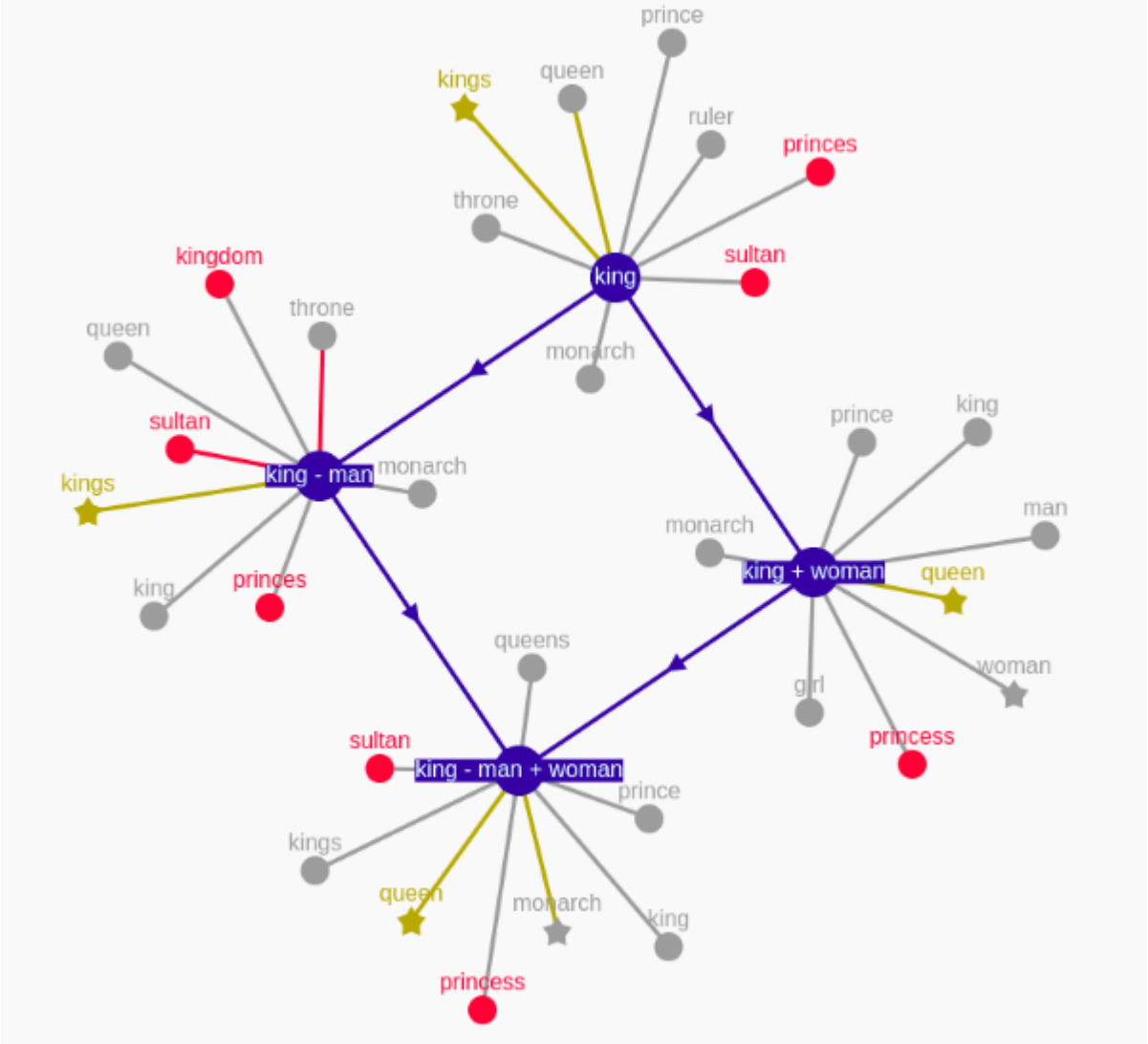
ζ)

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/analogies_google.py girls queen kings
boys
(slplab) nickaraf:~/slplab/lab1$ python scripts/analogies_google.py good taller tall
great
(slplab) nickaraf:~/slplab/lab1$ python scripts/analogies_google.py france paris london
england
```

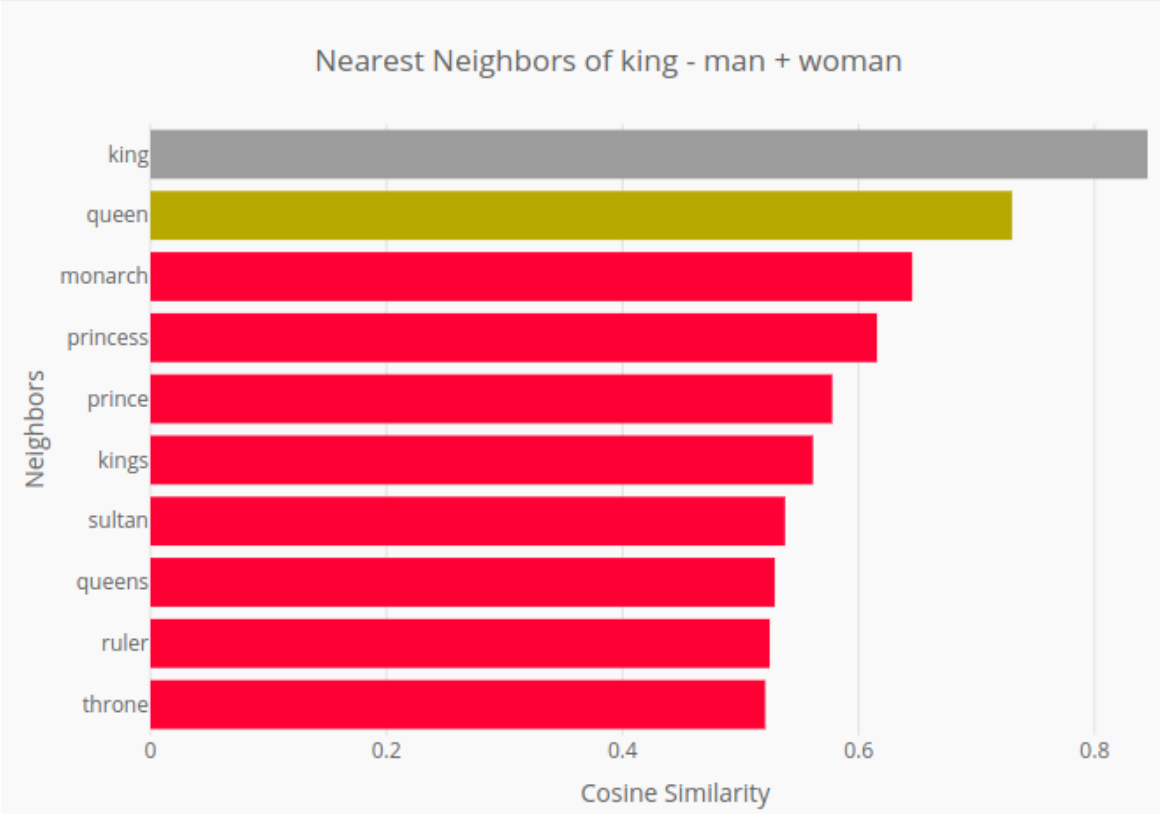
και εδώ παρατηρούμε ότι το μοντέλο της google φέρνει πολύ καλύτερα αποτελέσματα στις πρώτες δύο δοκιμές, σε σχέση με το προηγούμενο μοντέλο.

Βήμα 13: Οπτικοποίηση των word embeddings

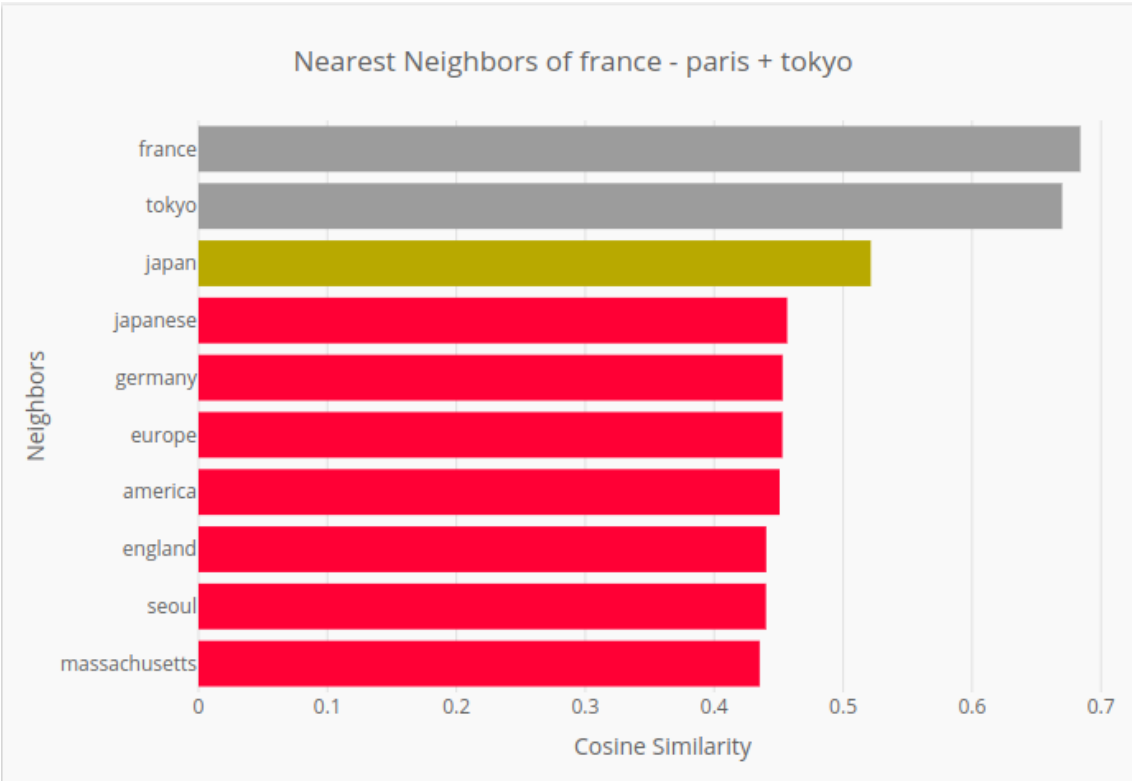
α)
Πειραματιζόμαστε με το εργαλείο οπτικοποίησης https://dash.gallery/dash-word-arithmetic/



Εκτός από τις αναλογίες, παρουσιάζουν ενδιαφέρον οι πράξεις king-man και king+woman που έχουν κοντινότερες λέξεις τις kings και queen αντίστοιχα.



Παρατηρούμε ότι τη μεγαλύτερη ομοιότητα με το king-man+woman το έχει η λέξη king, κάτι που μας κάνει να αμφιβάλλουμε για την διαισθητικότητα της κωδικοποίησης word2vec. Εκτός βέβαια αν παραλείψουμε τα αρχικές λέξεις, οπότε βγαίνει queen.
Ομοίως και στο παράδειγμα france-paris+tokyo όπου εμφανίζονται πρώτες οι λέξεις france και tokyo πριν το αναμενόμενο japan, όπως φαίνεται παρακάτω.

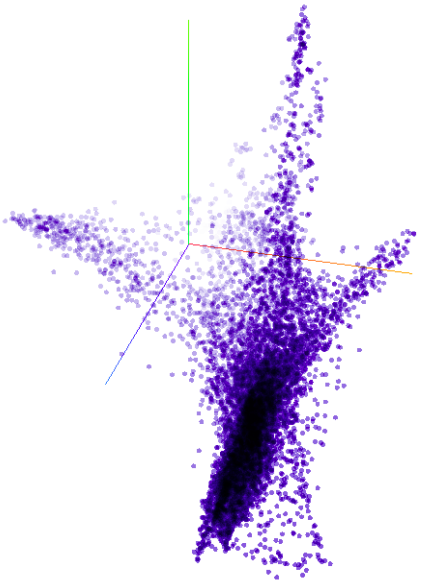


β) Η υλοποίηση του ερωτήματος γίνεται στο script 13.py

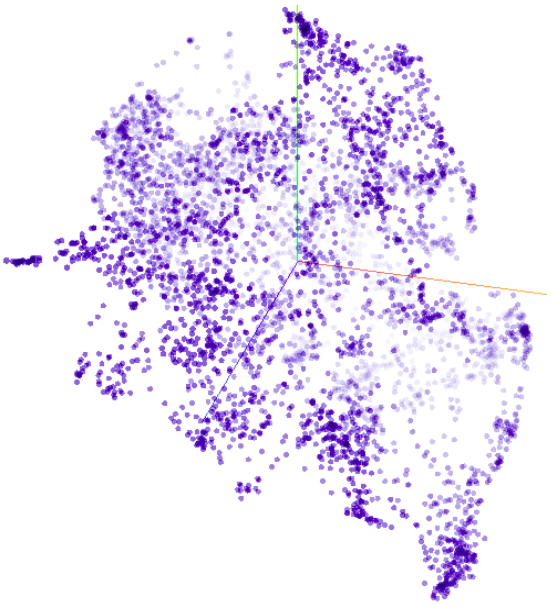
γ) Φορτώνουμε τα .tsv και παρατηρούμε ότι:

1. οι λέξεις που είναι φαινομενικά λάθος είναι απομακρυσμένες από τις άλλες και δεν ανήκουν σε κάποιο cluster λέξεων όπως γίνεται στις φαινομενικά σωστές λέξεις.
2. Παρατηρούμε ότι η μέθοδος μείωσης διαστατικότητας t-SNE προσφέρει μια πολύ καλύτερη οπτικοποίηση των δεδομένων σε σχέση με την PCA, καθώς εστιάζει περισσότερο στις τοπικές συσχετίσεις. Η UMAP φαίνεται να έχει μια ενδιάμεση οπτικοποίηση(πιο συμπιεσμένη από την t-SNE, αλλά όχι τόσο ομοιόμορφη όσο η PCA).

   t-SNE:



   UMAP:



PCA:

Βήμα 14: Ανάλυση συναισθήματος με word2vec embeddings

α,β)
Τροποποιούμε την συνάρτηση w2v_sentiment_analysis.py κατάλληλα

γ)

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/w2v_sentiment_analysis.py gutenberg
Accuracy score:  0.7344
```

Για binary classification το 0.73 θεωρείται πολύ μέτριο σκορ.
Βελτίωση της επίδοσης του μοντέλου θα μπορούσαμε να πετύχουμε με:

1. καλύτερη προεπεξεργασία των δεδομένων εκπαίδευσης και με χρηση μεθόδων mixup για να έχουμε καλύτερη γενίκευση.
2. περισσότερα δεδομένα για την εκπαίδευση των word embeddings
3. αλλαγή αλγορίθμου ταξινόμησης
4. βελτιστοποίηση υπερπαραμέτρων αλγορίθμου ταξινόμησης

δ)

```
(slplab) nickaraf:~/slplab/lab1$ python scripts/w2v_sentiment_analysis.py google
Accuracy score:  0.8288
```

Παρατηρούμε ότι τα αποτελέσματα βελτιώθηκαν, αλλά όχι πάρα πολύ σε σχέση με τα embeddings που υπολογίσαμε εμείς, κάτι που μας οδηγεί στο να συμπεράνουμε ότι η ανάλυση συναισθήματος είναι ένα σύνθετο task στο οποίο είναι δύσκολο να βγάλεις ξεκάθαρα αποτελέσματα με μεγάλη ακρίβεια, ακόμα και αν έχεις πρόσβαση σε τεράστιο όγκο δεδομένων εκπαίδευσης. Τα δεδομένα δεν αρκούν.