ΔΠΜΣ Επιστήμη Δεδομένων και Μηχανική Μάθηση

Επεξεργασία Φωνής και Φυσικής Γλώσσας

1η Εργαστηριακή Άσκηση

Ονοματεπώνυμο: Ευάγγελος Τσόγκας

Αριθμός Μητρώου: 03400120

ΜΕΡΟΣ 1: Κατασκευή Ορθογράφου

1. Corpus

Προκειμένου να φτιαχτεί ένα λεξικό για τον ορθογράφο είναι απαραίτητο να γίνει πρώτα ένα σωστό preprocessing και tokenization του corpus απ' όπου θα εξάγουμε τις πληροφορίες μας. Στα πλαίσια της προ επεξεργασίας του κειμένου διαγράφουμε περιττά κενά διαστήματα για τον σωστό διαχωρισμό σε tokens, μετατρέπουμε όλα τα κεφαλαία γράμματα σε πεζά και αγνοούμε οτιδήποτε δεν είναι πεζός χαρακτήρας προκειμένου να κατασκευάσουμε ένα καλό λεξικό. Επίσης, για να είναι σωστό το λεξικό διορθώνουμε και τα contractions, ώστε να πάρουμε τις επιμέρους λέξεις από τις οποίες αποτελούνται. Στο πρόβλημά μας είναι πολύ σημαντική η αφαίρεση σημείων στίξης καθώς θέλουμε να κατασκευάσουμε ένα σωστό και καθαρό λεξικό. Παρ' όλα αυτά, υπάρχουν περιπτώσεις που δεν θέλουμε να κάνουμε τόσο επιθετική προεπεξεργασία και χρειαζόμαστε ειδικούς χαρακτήρες, όπως για παράδειγμα όταν θέλουμε να διατυπώσουμε ερωτήματα πάνω σε κείμενα που αφορούν αριθμητικές τιμές (πχ. ποσό χρημάτων $, ποσοστά %) ή όπου ακρωνύμια λέξεων μπορούν να δώσουν σημαντική πληροφορία, όπως επίσης αν θέλουμε να μοντελοποιήσουμε περίπλοκες συντακτικές δομές, πχ. μαθηματικές παραστάσεις.

Θα μπορούσαμε επιπλέον να επεκτείνουμε το corpus με κείμενα από άλλες πηγές. Ένα από τα πλεονεκτήματα αυτής της τεχνικής είναι ότι θα έχουμε περισσότερες λέξεις για ένα πιο πλούσιο λεξικό, εννοώντας περισσότερες διαφορετικές λέξεις με βάση το λήμμα τους και όχι απλά περισσότερα tokens. Επιπλέον, θα έχουμε περισσότερες μορφές μιας λέξης με το ίδιο λήμμα, δηλαδή ποικιλία ως προς τη μορφολογία (πχ. παράγωγα) που θα οδηγήσει σε έναν πιο αποδοτικό ορθογράφο. Τέλος, με βάση τη συχνότητα των λέξεων μπορούμε να είμαστε πιο σίγουροι για τη σωστή ορθογραφία τους, καθώς λέξεις που εμφανίζονται ελάχιστες φορές μπορεί να είναι απλά ανορθόγραφες.

2. Λεξικό και σύμβολα εισόδου/εξόδου

Η κατασκευή του λεξικού έγινε με τη χρήση του script create_dictionary.py. Tokens που εμφανίζονται λιγότερες από 5 φορές έχουν φιλτραριστεί, καθώς κατά πάσα πιθανότητα είναι ανορθόγραφες λέξεις (ή απλά πολύ σπάνιες), επομένως θα έχουν αρνητική επίδραση στον ορθογράφο.

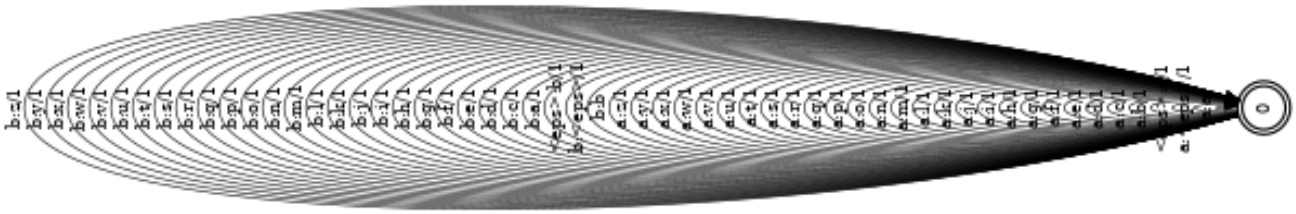
Η δημιουργία των αρχείων που αντιστοιχίζουν χαρακτήρες και λέξεις σε indices έγινε με το script create_symbols.py. Οι συναρτήσεις για την αντιστοίχιση δημιουργούν ένα λεξικό με κλειδιά τους χαρακτήρες (ή τις λέξεις αντίστοιχα) και με τιμές τα μοναδικά αύξοντα indices.

3. Μετατροπέας edit distance L

Ο μετατροπέας που υλοποιεί την απόσταση Levenshtein δημιουργήθηκε με το script mkfstLevenshtein.py και τα αρχεία περιγραφής του με το script savefstLevenshtein.sh. Αυτός ο μετατροπέας για κάποια λέξη εισόδου μας δίνει ακριβώς την ίδια λέξη αν πάρουμε το shortest path, καθώς η αντιστοίχιση ενός χαρακτήρα με τον εαυτό του έχει μηδενικό κόστος.

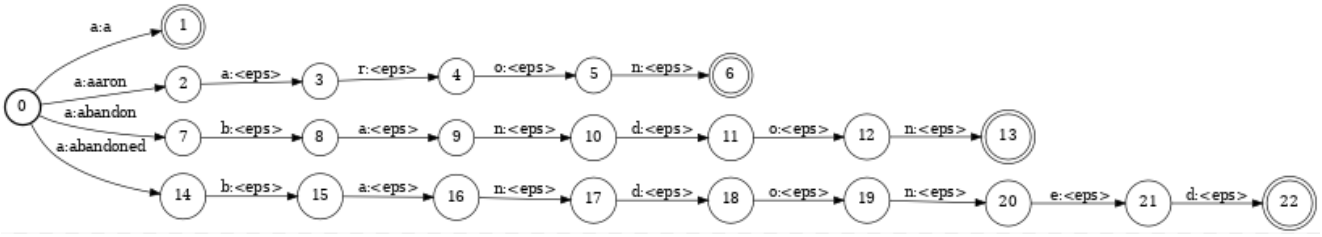
Ένα άλλο edit που θα μπορούσαμε να συμπεριλάβουμε είναι το swap δύο χαρακτήρων της λέξης. Για παράδειγμα από βιασύνη μπορεί κάποιος να πληκτρολογήσει ‘καρέλκα’ αντί για ‘καρέκλα’. Επίσης, θα μπορούσαμε να βελτιώσουμε τα βάρη του substitution έτσι ώστε η αντικατάσταση χαρακτήρων που βρίσκονται κοντά στο πληκτρολόγιο να έχει μικρότερο κόστος (πχ. ο -> p ή y -> u).

Στην παρακάτω εικόνα φαίνεται το fst μόνο για τους χαρακτήρες a και b.

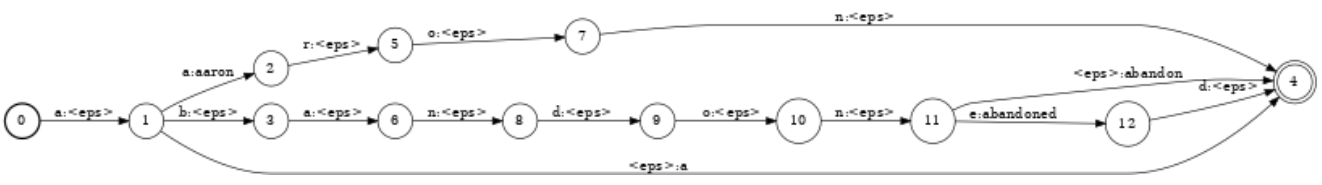


4. Αποδοχέας λεξικού V

Για την κατασκευή του αποδοχέα λεξικού χρησιμοποιήθηκαν τα scripts mkfstLexicon.py και savefstLexicon.sh. Ο αποδοχέας έχει μια μοναδική αρχική κατάσταση και μια τελική κατάσταση για κάθε λέξη που αποδέχεται. Στην παρακάτω εικόνα φαίνεται ένα παράδειγμα του αποδοχέα για τις 4 πρώτες λέξεις του λεξικού.



Επίσης, το ίδιο αποδοχέας αφού τον βελτιστοποιήσουμε είναι ο εξής:



Παρατηρούμε ότι πλέον έχει γίνει ντετερμινιστικός αφού δεν υπάρχουν τέσσερις μεταβάσεις από την κατάσταση 0 με το ίδιο σύμβολο. Επίσης, κάποιες μεταβάσεις έχουν φτιαχτεί πιο έξυπνα με αποτέλεσμα να μειωθεί ο συνολικός αριθμός τους από 22 σε 15 και ο συνολικός αριθμός των καταστάσεων από 22 σε 12.

Η βελτιστοποίηση του αποδοχέα έγινε χρησιμοποιώντας τις εντολές fstrmepsilon, fstdeterminize και fstminimize. Η fstrmepsilon αφαιρεί τις epsilon μεταβάσεις ώστε να γίνει στη συνέχεια ντετερμινιστικό το αυτόματο με την fstdeterminize. Στο συγκεκριμένο πρόβλημα δεν υπήρχαν epsilon μεταβάσεις, αλλά όπως είδαμε η μη ντετερμινιστικότητα του αυτομάτου από την κατάσταση 0 διορθώθηκε. Η fstminimize όπως παρατηρήσαμε δημιουργεί ένα ισοδύναμο αυτόματο με λιγότερες μεταβάσεις και καταστάσεις.

Το traversal ενός ντετερμινιστικού αυτομάτου γίνεται σε γραμμικό χρόνο, αφού με κάποιο σύμβολο πάμε σε μοναδική κατάσταση. Αντίθετα, η πολυπλοκότητα του traversal ενός μη ντετερμινιστικού αυτομάτου είναι μεγαλύτερη, αφού πρέπει να εξετάσουμε όλες τις δυνατές καταστάσεις στις οποίες οδηγεί ένα σύμβολο. Το γεγονός όμως πως ένα μη ντετερμινιστικό αυτόματο μας επιτρέπει να εκφράσουμε πολλαπλές μεταβάσεις από μια κατάσταση σε άλλες χρησιμοποιώντας το ίδιο σύμβολο έχει ως αποτέλεσμα να μπορεί να φτιαχτεί με λιγότερες καταστάσεις και ακμές σε σύγκριση με ένα ντετερμινιστικό αυτόματο.

5. Ορθογράφος LV

Ο ορθογράφος δημιουργήθηκε συνθέτοντας τον μετατροπέα Levenshtein με τον αποδοχέα λεξικού χρησιμοποιώντας το script savefstSpellchecker.sh. Για το compose χρησιμοποιήθηκε η fstarcsort στο output του μετατροπέα.

Στην περίπτωση που τα edits είναι ισοβαρή τότε η διόρθωση μια λέξης γίνεται με τον μικρότερο αριθμό από edits ανεξαρτήτως του ποια είναι αυτά. Αν κάποιο edit έχει μεγαλύτερο κόστος από τα άλλα τότε πιο εύκολα θα επιλέγεται μια διόρθωση χωρίς αυτό το edit. Για παράδειγμα αν το κόστος αντικατάστασης είναι 1.5 αντί για 1, τότε η λέξη ‘cot’ θα μετατραπεί σε ‘cost’ αντί για ‘cut’, δεδομένου ότι η εισαγωγή έχει κόστος 1.

Αν έχουμε σαν είσοδο τη λέξη cit πιθανές προβλέψεις είναι οι εξής με κόστος 1:

- Με εισαγωγή: city, cite
- Με διαγραφή: it
- Με αντικατάσταση: cat, cut, wit, pit, sit, fit, hit, kit, lit, bit

Αν έχουμε σαν είσοδο τη λέξη cwt πιθανές προβλέψεις είναι οι εξής με κόστος 1:

- Με αντικατάσταση: cut, cat

Παρατηρούμε πως στην πρώτη περίπτωση υπάρχουν πολλές πιθανές διορθώσεις σε σύγκριση με τη δεύτερη. Αν όμως η αντικατάσταση είχε μεγαλύτερο κόστος τότε και στην πρώτη θα είχαμε λιγότερες επιλογές.

6. Δοκιμή ορθογράφου LV

Με χρήση του script test_spellchecker.sh το οποίο καλεί εσωτερικά το script predict.sh για κάθε μια από τις πρώτες 20 λέξεις του spell_test.txt παίρνουμε τα αποτελέσματα του παρακάτω πίνακα. Παρατηρούμε πως αν και δεν μας έδωσε το αποτέλεσμα που θα θέλαμε σε όλες τις λέξεις ο αλγόριθμος του min edit distance λειτουργεί όπως θα έπρεπε. Για παράδειγμα αν και με τη λέξη ‘juse’ εννοούσαμε ‘juice’ είναι απόλυτα λογικό με το min edit distance να διορθωθεί σε ‘use’ με μια απλή διαγραφή αντί μιας εισαγωγής και μιας αντικατάστασης.

Λανθασμένη Λέξη	Διόρθωση Spell Checker LV	Σωστή Λέξη
contenpted	contented	contented
contende	contend	contented
contended	contended	contented
contentid	contented	contented
begining	beginning	beginning
problam	problem	problem
proble	problem	problem
promblem	problem	problem
proplen	prophet	problem
dirven	dive	driven
exstacy	ecstasy	ecstasy
ecstacy	ecstasy	ecstasy
guic	guil	juice
juce	june	juice
jucie	lucil	juice
juise	juice	juice
juse	use	juice
localy	local	locally
compair	complain	compare
pronounciation	provocation	pronunciation

Με το script predict.sh ουσιαστικά δίνουμε μια λέξη στον spell checker ώστε να τη διορθώσει. Προκειμένου όμως αυτή η λέξη να μπορεί να δοθεί σαν είσοδος στον spell checker πρώτα φτιάχνουμε ένα αυτόματο που αποδέχεται αυτή τη λέξη χαρακτήρα προς χαρακτήρα και ύστερα το συνθέτουμε με τον spell checker. Ο spell checker αποτελεί τη σύνθεση του Levenshtein μετατροπέα και του αποδοχέα λεξικού, επομένως γίνονται όλες οι δυνατές αλλαγές στην λέξη εισόδου (edits) από τον μετατροπέα, ώστε να παραχθούν λέξεις του λεξικού που αποδέχεται ο αποδοχέας και διαλέγοντας το shortest path του spell checker οδηγούμαστε στη λέξη του λεξικού για την οποία χρειάστηκε ο μικρότερος αριθμός από edits (με το μικρότερο κόστος).

7. Υπολογισμός κόστους των edits – Ορθογράφος EV

Τα edits που χρειάζονται για να μετατραπεί μια λάθος λέξη σε μια σωστή βρίσκονται με το script word_edits.sh. Το script αυτό στην ουσία δουλεύει παρόμοια με τον spellchecker, αλλά αντί για λεξικό με σωστές λέξεις έχουμε μια μοναδική σωστή λέξη που αντιστοιχεί στη λανθασμένη και το output είναι χαρακτήρες και όχι λέξεις. Επομένως συνθέτουμε τρία fst: τον αποδοχέα της λάθος λέξης, τον μετατροπέα Levenshtein και τον αποδοχέα της σωστής λέξης. Ως αποτέλεσμα ο μετατροπέας Levenshtein κάνει αλλαγές στη λέξη εισόδου (τη λάθος λέξη) και καταλήγει στη λέξη εξόδου (τη σωστή λέξη) όπου παίρνοντας το shortest path βρίσκουμε τα edits με το μικρότερο κόστος για την διαδικασία διόρθωσης.

Αφού βρούμε όλα τα edits με το scripts save_edits.py, ξαναφτιάχνουμε τον ορθογράφο, αλλά αυτή τη φορά τα βάρη των edits στον μετατροπέα είναι ίσα με τον αρνητικό λογάριθμο της συχνότητάς τους. Για το σκοπό αυτό χρησιμοποιούνται τα scripts mkfst_E.py, savefst_E.sh, savefst_EV.sh.

Με το script test_EV.sh δοκιμάζουμε τον καινούριο ορθογράφο στις 20 πρώτες λέξεις του spell_test.txt και τα αποτελέσματα φαίνονται στον παρακάτω πίνακα.

Λανθασμένη Λέξη	Διόρθωση Spell Checker EV	Σωστή Λέξη
contenpted	contented	contented
contende	contend	contented
contended	contended	contented
contentid	contented	contented
begining	beginning	beginning
problam	problem	problem
proble	problem	problem
promblem	problem	problem
proplen	people	problem
dirven	driven	driven
exstacy	exactly	ecstasy
ecstacy	ecstasy	ecstasy
guic	guil	juice
juce	juice	juice
jucie	juice	juice
juise	juice	juice
juse	just	juice
localy	local	locally
compair	compare	compare
pronounciation	pronouncing	pronunciation

Παρατηρούμε πως σε σύγκριση με τον ορθογράφο LV, ο EV έχει καταφέρει να διορθώσει λίγες περισσότερες λέξεις, αλλά έχει κάνει και κάποιες διαφορετικές διορθώσεις.

8. Εισαγωγή της συχνότητας εμφάνισης λέξεων – Ορθογράφοι LVW, EVW

Αφού κατασκευάσουμε το γλωσσικό μοντέλο W, το συνθέτουμε με τους ορθογράφους LV και EV, ώστε πλέον να λαμβάνουμε υπόψιν και πόσο πιθανή είναι μια λέξη. Τα scripts που χρησιμοποιήθηκαν είναι τα mkfst_W.py, savefst_W.sh, savefstLVW.sh, και savefstEVW.sh. Αξιολογώντας τον ορθογράφο LVW στις 20 πρώτες λέξεις του spell_test.txt παρατηρούμε πως τείνει να διορθώνει τις λέξεις σε κάποιες όπως τις ‘and’ και ‘the’ οι οποίες εμφανίζονται πολύ συχνά και για αυτό έχουν μικρό βάρος στο γλωσσικό μοντέλο με αποτέλεσμα να είναι καλύτερες διορθώσεις από λέξεις που θα χρειάζονταν λιγότερα edits αλλά έχουν αρκετά μεγαλύτερο βάρος και έτσι δεν τα πηγαίνει καθόλου καλά στις διορθώσεις.

Για παράδειγμα, ο LV διορθώνει τις λέξεις cit και cwt ως εξής:

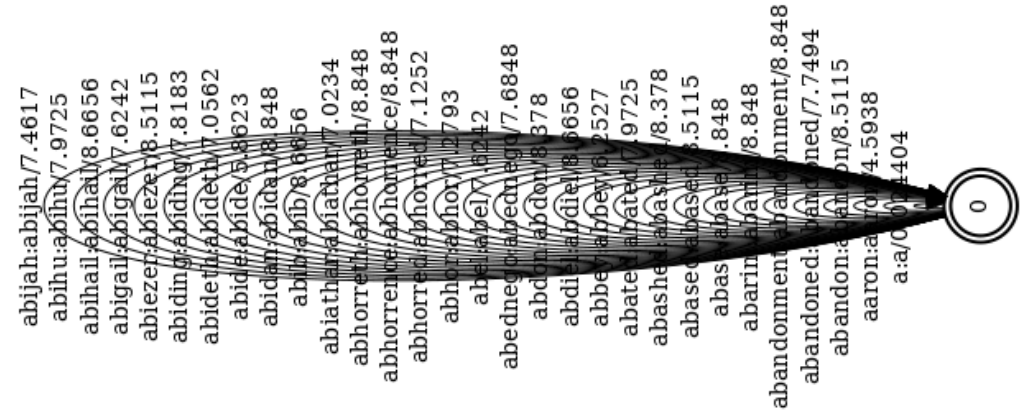
- cit -> wit
- cwt -> cut

ενώ ο LVW:

- cit -> it
- cwt -> the

Η διόρθωση σε ‘it’ είναι καλύτερη από το ‘wit’, καθώς το ‘it’ είναι σίγουρα πιο συχνή λέξη, παρόλα αυτά ο LVW διόρθωσε το ‘cwt’ σε ‘the’ για τον λόγο που προαναφέρθηκε.

Παρακάτω φαίνονται τα fst W και VW αντίστοιχα, σχεδιασμένα χρησιμοποιώντας λίγες λέξεις:



9. Αξιολόγηση των ορθογράφων

Τα αποτελέσματα της αξιολόγησης των ορθογράφων LV, LVW, EV και EVW στο spell_test.txt φαίνονται στον παρακάτω πίνακα:

Spellchecker	Accuracy
LV	0.5889
LVW	0.0444
EV	**0.6889**
EVW	0.6407

Παρατηρούμε πως ο καλύτερος ορθογράφος είναι ο EV, δηλαδή με χρήση διαφορετικών βαρών ανάλογα τη συχνότητα των edits. Η προσθήκη του γλωσσικού μοντέλου στον EV από ότι φαίνεται δεν βελτιώνει τα αποτελέσματα. Αυτό μπορεί να οφείλεται στο γεγονός ότι δεν παράγεται αρκετά καλό γλωσσικό μοντέλο από τα βιβλία που χρησιμοποιήσαμε ή στο ότι δεν έχουμε αρκετά δεδομένα. Στην περίπτωση της απλής Levenshtein απόστασης βλέπουμε πως η προσθήκη των βαρών με το γλωσσικό μοντέλο καθιστά τον ορθογράφο μη αποδοτικό. Αυτό οφείλεται στο γεγονός, ότι το κόστος των edits είναι μονάδα, ενώ το κόστος της αποδοχής λέξεων είναι αρκετά μεγαλύτερο λόγω του αρνητικού λογαρίθμου των πιθανοτήτων, επομένως αυτή η ανομοιότητα στα βάρη οδηγεί τον ορθογράφο LVW, απλά να διορθώνει στις πιο συχνές λέξεις που βρίσκει στο λεξικό (όπως ‘and’ και ‘the’), ενώ τα edits που χρησιμοποιεί δεν παίζουν τόσο σημαντικό ρόλο.

10. Βελτιώσεις του ορθογράφου EV

Προκειμένου να βελτιώσουμε τον ορθογράφο EV χρησιμοποιούμε την τεχνική του add-1 smoothing για να αντιμετωπίσουμε το πρόβλημα των edits που δεν εμφανίζονται καθόλου στο corpus. Δημιουργούμε λοιπόν με παρόμοιο τρόπο τον ορθογράφο EV_v2 και τον τεστάρουμε χρησιμοποιώντας 3 διαφορετικά λεξικά: αυτό που παράχθηκε από τη συλλογή Gutenberg που χρησιμοποιήσαμε και προηγουμένως, το λεξικό του Opensubtitles που περιέχει λέξεις από υποτίτλους ταινιών και το λεξικό Simpsons (https://pastebin.com/anKcMdvk) που περιέχει τις λέξεις από όλα τα επεισόδια της σειράς Simpsons. Για αυτή τη διαδικασία χρησιμοποιήθηκαν παρόμοια scripts με τα προηγούμενα βήματα για τη δημιουργία των συμβόλων εισόδου/εξόδου και των fst. Το

evaluation έγινε και πάλι με τη χρήση του scripts predict.sh, αλλάζοντας κατάλληλα το αρχείο των συμβόλων για τις λέξεις.

Η επίδοση του ορθογράφου χρησιμοποιώντας αυτά τα τρία λεξικά, αφού υπέστησαν μια μικρή προ-επεξεργασία για αφαίρεση μη αγγλικών χαρακτήρων, φαίνεται στον παρακάτω πίνακα:

Λεξικό	Accuracy EV_v2
Gutenberg (15466 λέξεις)	0.6926
Simpsons (4764 λέξεις)	0.4519
Opensubtitles (49644 λέξεις)	**0.7556**

Παρατηρούμε πως με το λεξικό του Opensubtitles ο ορθογράφος είχε τα καλύτερα αποτελέσματα και αυτό σίγουρα οφείλεται στο μεγάλο μέγεθός του. Αντίθετα, τα αποτελέσματα του ορθογράφου με το λεξικό Simpsons που αποτελείται από πολύ λίγες λέξεις δεν ήταν τόσο καλά. Βλέπουμε όμως πως παρόλο που το λεξικό του Opensubtitles έχει πάνω από τις τριπλάσιες λέξεις απ' ότι του Gutenberg το αποτέλεσμα δεν ήταν τόσο πολύ καλύτερο. Αυτό μπορεί να οφείλεται στο ότι το λεξικό που φτιάχτηκε από μια συλλογή βιβλίων είναι πιο ποιοτικό σε σύγκριση με το λεξικό που φτιάχτηκε από υπότιτλους ταινιών.

Μια άλλη βελτίωση που μπορούμε να κάνουμε στον ορθογράφο είναι αντί για το κλασσικό add-1 smoothing στα edits να χρησιμοποιήσουμε το πιο γενικό add-k smoothing με k<1. Επίσης μπορούμε να εισάγουμε το γλωσσικό μοντέλο στον ορθογράφο που χρησιμοποιεί το λεξικό του Opensubtitles.

Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα του ορθογράφου EV_v3 με add-k smoothing όπου k=0.5 για το λεξικό Gutenberg και επίσης του ορθογράφου EVW_v3 όπου έχουμε προσθέσει επιπλέον το γλωσσικό μοντέλο για το λεξικό Opensubtitles.

Spellchecker	Accuracy
EV_v3 (Gutenberg)	0.6963
EVW_v3 (Opensubtitles)	**0.7926**

Όπως παρατηρούμε το αποτέλεσμα με k=0.5 είναι λίγο καλύτερο και αυτό επειδή τα edits που δεν είχαν εμφανιστεί καθόλου ήταν αρκετά και με add-1 smoothing τους δίναμε μεγάλη μάζα πιθανότητας. Επίσης, βλέπουμε πως από τα frequencies του λεξικού Opensubtitles παράχθηκε ένα γλωσσικό μοντέλο που βελτίωσε αρκετά τα αποτελέσματα σε σύγκριση με το γλωσσικό μοντέλο του Gutenberg που είχε εξεταστεί προηγουμένως. Αυτό οφείλεται στο ότι τα frequencies των λέξεων του Opensubtitles έχουν εξαχθεί από εκατομμύρια αρχεία κειμένων και επομένως το γλωσσικό μοντέλο είναι πιο ποιοτικό.

ΜΕΡΟΣ 2: Το Μοντέλο Word2Vec

1. Εξαγωγή αναπαραστάσεων word2vec

Αφού εκπαιδεύσουμε το μοντέλο word2vec με window=5 για 1000 εποχές, δοκιμάζουμε να βρούμε μερικές σημασιολογικά κοντινές λέξεις. Στους παρακάτω πίνακες φαίνονται οι τρεις πιο κοντινές λέξεις για κάθε μια από τις λέξεις: “bible”, “book”, “bank”, “water” χρησιμοποιώντας το μοντέλο που εκπαιδεύσαμε στο Gutenberg corpus, καθώς και το προ-εκπαιδευμένο μοντέλο στα google news.

Gutenberg vectors			
Επιλεγμένες λέξεις	**Σημασιολογικά κοντινότερες λέξεις**		
bible	official	coffin	velvet
book	written	letter	papers
bank	top	river	wall
water	waters	river	wine

GoogleNews vectors			
Επιλεγμένες λέξεις	**Σημασιολογικά κοντινότερες λέξεις**		
bible	Bible	bibles	Holy_Bible
book	tome	books	memoir
bank	banks	banking	Bank
water	portable_water	Water	sewage

Στην περίπτωση του μοντέλου που εκπαιδεύσαμε στο Gutenberg corpus βλέπουμε πως τα αποτελέσματα για τις λέξεις book και water είναι καλά, αλλά για τις λέξεις bible και bank δεν είναι τόσο ποιοτικά, σε αντίθεση με τα αποτελέσματα που πήραμε από τα GoogleNews vectors τα οποία δίνουν πολύ όμοιες λέξεις. Αυξάνοντας το μέγεθος του window σε 10 και τον αριθμό των εποχών σε 1500 τα αποτελέσματα αν και λίγο διαφορετικά, δεν βελτιώνονται όπως ούτε και αν τα μειώσουμε σε 3 και 300 αντίστοιχα. Από ότι φαίνεται μάλλον το λεξιλόγιο απλά δεν είναι αρκετά μεγάλο ώστε να βρούμε αρκετά ποιοτικές σημασιολογικά κοντινές λέξεις. Για να βελτιωθούν τα αποτελέσματα θα μπορούσαμε να εκπαιδεύσουμε το μοντέλο σε περισσότερα δεδομένα ώστε να βρεθούν πιο ποιοτικές συσχετίσεις λέξεων και επίσης να αυξήσουμε το dimension των embeddings σε 300, ώστε να έχουμε πιο ακριβείς αναπαραστάσεις των λέξεων.

Επιπλέον, δοκιμάζουμε τα μοντέλα ως προς τις σημασιολογικές αναλογίες. Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα.

Τριπλέτα λέξεων	**Αποτέλεσμα Gutenberg**	**Αποτέλεσμα GoogleNews**
(“girls”, “queen”, “kings”)	carriages	boys
(“good”, “taller”, “tall”)	handsome	great
(“france”, “paris”, “london”)	school	england

Παρατηρούμε πως τα αποτελέσματα του μοντέλου GoogleNews είναι άριστα αφού πράγματι περιμέναμε τις αναλογίες:

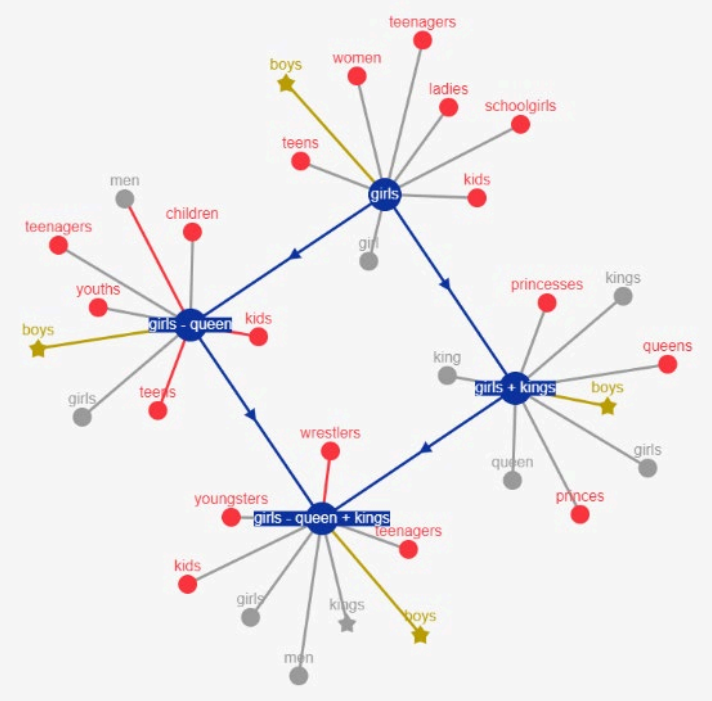
- girls – queen + kings = boys
- good – taller + tall = great
- france – paris + london = england

Δυστυχώς όμως τα αποτελέσματα του μοντέλου Gutenberg δεν είναι ικανοποιητικά.

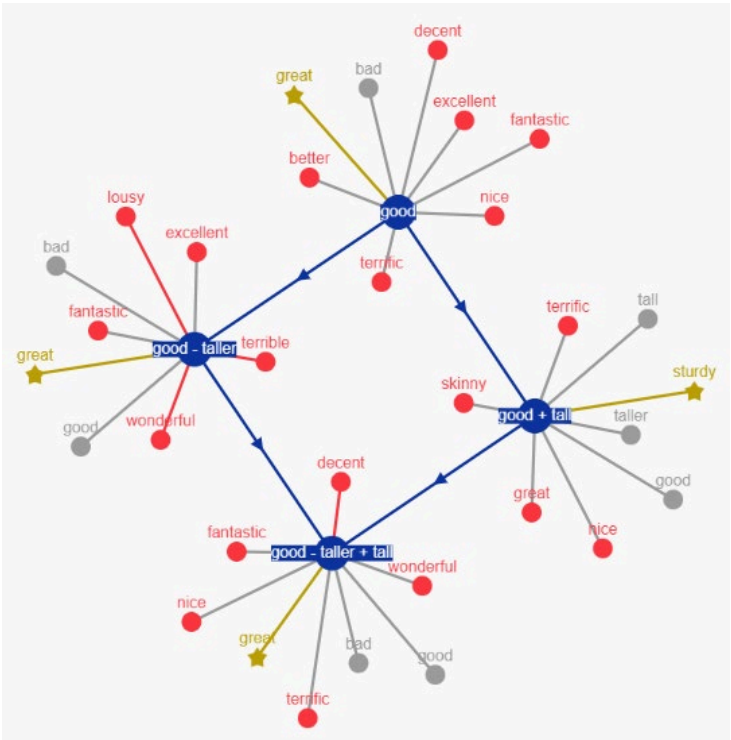
2. Οπτικοποίηση των word embeddings

Στις παρακάτω εικόνες έχουμε κάνει visualize τις αναλογίες που δοκιμάσαμε προηγουμένως.

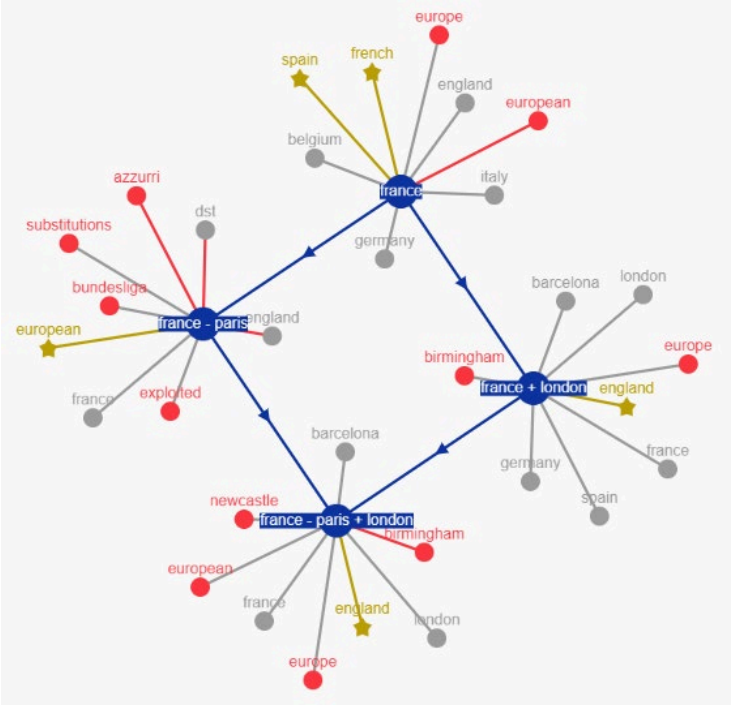
- **girls – queen + kings = boys**



- **good – taller + tall = great**

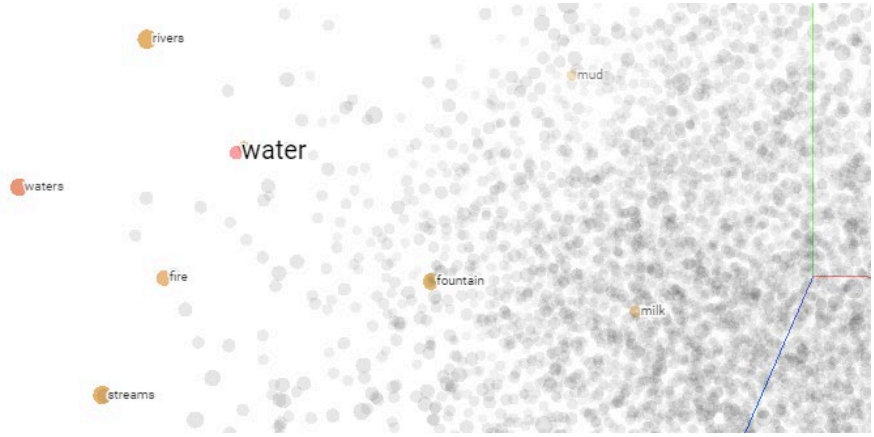


- **france – paris + london = england**

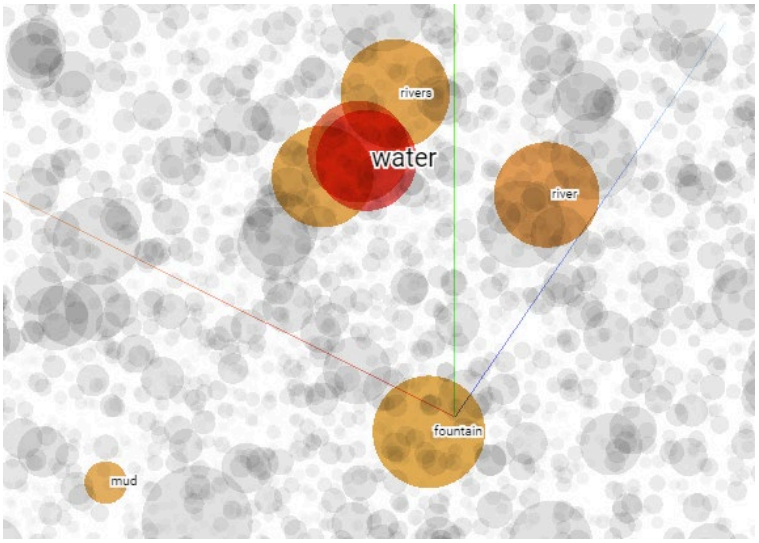


Βλέπουμε πως με κίτρινο χρώμα φαίνεται η πιο κοντινή λέξη όταν εφαρμόζουμε τις αντίστοιχες αλγεβρικές πράξεις, αλλά φαίνονται και άλλες σημασιολογικά κοντινές λέξεις. Επίσης, αναπαρίστανται και επιμέρους πράξεις με βάση την πρώτη λέξη από την τριπλέτα που έχουμε χρησιμοποιήσει. Τα αποτελέσματα είναι αντίστοιχα με αυτά που βρήκαμε όταν εφαρμόσαμε τις πράξεις χρησιμοποιώντας τα GoogleNews vectors.

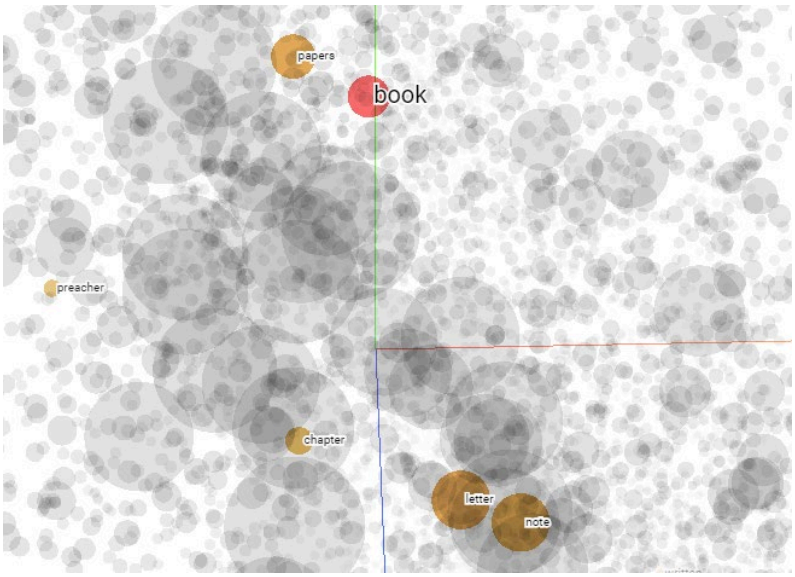
Στη συνέχεια φορτώνουμε τα embeddings που εκπαιδεύσαμε στο Embedding Projector και βλέπουμε τις σημασιολογικά κοντινές λέξεις που είδαμε προηγουμένως. Αν για παράδειγμα επιλέξουμε τη λέξη “water” βλέπουμε τις σημασιολογικά κοντινές λέξεις με χρήση PCA ως εξής:



Αν και οι λέξεις βρίσκονται στην ίδια μεριά τον αξόνων, στην αναπαράσταση με PCA φαίνονται διασκορπισμένες και όχι πραγματικά κοντά η μια στην άλλη. Αν όμως χρησιμοποιήσουμε t-SNE για τη μείωση διάστασης τότε βλέπουμε καλύτερα τις κοντινές αποστάσεις, όπως φαίνεται στην παρακάτω εικόνα.



Ακόμα και σε αυτή την περίπτωση όμως τα αποτελέσματα δεν είναι πάντα τέλεια. Αν πάρουμε για παράδειγμα την λέξη book βλέπουμε πως η πιο κοντινή της λέξη που είναι η “written” βρίσκεται αρκετά μακριά της.



3. Ανάλυση συναισθήματος

Αφού δημιουργήσουμε τις NBOW αναπαραστάσεις των προτάσεων εκπαιδεύουμε έναν sentiment analysis Logistic Regression classifier στα δεδομένα του Imdb. Η επίδοση των classifiers χρησιμοποιώντας τα Gutenberg και GoogleNews word vectors φαίνεται στον παρακάτω πίνακα:

Word Embeddings	Test Accuracy
Gutenberg	0.72984
GoogleNews	**0.82784**

Παρατηρούμε ότι με τα GoogleNews vectors το μοντέλο τα πήγε πολύ καλύτερα. Αυτό είναι λογικό, καθώς όπως εξετάσαμε και προηγουμένως τα Gutenberg vectors δεν είναι τόσο ποιοτικά λόγω λίγων δεδομένων και μικρού λεξιλογίου.

Υπάρχουν διάφοροι τρόποι να βελτιώσουμε την επίδοση του sentiment analysis. Θα μπορούσαμε για παράδειγμα να χρησιμοποιήσουμε deep learning τεχνικές όπως CNN και RNN και state of the art word embeddings όπως το μοντέλο Bert. Αν δεν θέλαμε να χρησιμοποιήσουμε νευρωνικά δίκτυα θα μπορούσαμε να εκπαιδεύσουμε έναν απλό classifier όπως το Logistic Regression, αλλά χρησιμοποιώντας sentence embeddings όπως τον Universal Sentence Encoder, αντί για NBOW αναπαραστάσεις.