ΗΜΜΥ ΕΜΠ

Επεξεργασία Φωνής και Φυσικής Γλώσσας (2022-2023)

1η Εργαστηριακή Άσκηση

Ηλιόπουλος Γεώργιος:	03118815	giliopoulos301@gmail.com
Νικόλαος Σπανός:	03118822	nickspanos23@gmail.com

Σημείωση: Για να τρέξει το .py αρχείο πρέπει πρώτα να οριστεί το path όπου βρίσκονται τα βιβλία από το Gutenberg Corpus και το script directory στο 2ο μέρος

Στον ακόλουθο σύνδεσμο είναι ανεβασμένο το μοντέλο μας.

Θέμα: Εισαγωγή στις γλωσσικές αναπαραστάσεις

Σκοπός της 1ης εργαστηριακής άσκησης είναι η εξοικείωση με τη χρήση κλασικών και ευρέως χρησιμοποιούμενων γλωσσικών αναπαραστάσεων για την επεξεργασία φυσικής γλώσσας.

Μέρος 1: Κατασκευή Ορθογράφου

Βήμα 1: Κατασκευή corpus

α) Σε αυτό το βήμα δημιουργούμε ένα σώμα κειμένων (corpus) από το project Gutenberg που θα χρησιμοποιήσουμε στην άσκηση. Αυτό το corpus θα χρησιμοποιηθεί για την εξαγωγή στατιστικών κατά την κατασκευή γλωσσικών μοντέλων. Τα βήματα προεπεξεργασίας που ακολουθούμε είναι τα εξής:

- Αφαίρεση ειδικών χαρακτήρων και αριθμών από τα κείμενα,
- Μετατροπή των κεφαλαίων σε πεζά γράμματα,
- Tokenize βάση του χαρακτήρα «space».

Σε ορισμένες περιπτώσεις, όπως για παράδειγμα όταν θέλουμε να διατηρήσουμε το context για sentimental analysis, τα σημεία στίξης μπορεί να αλλάξουν το νόημα της πρότασης υπονοώντας έκπληξη (!), στεναχώρια, απόγνωση (…) κ.α. Για αυτό είναι προτιμότερο να διατηρούνται τα σημεία στίξης καθώς παρέχουν χρήσιμη πληροφορία.

β) Ένας τρόπος για την εξαγωγή καλύτερων στατιστικών αποτελεσμάτων είναι με τον εμπλουτισμό του corpus με περισσότερα βιβλία. Η εκπαίδευση πάνω σε ένα εμπλουτισμένο corpus μπορεί να οδηγήσει σε:

- πιο αντιπροσωπευτικά αποτελέσματα,
- δημιουργία εφαρμογών πάνω σε μεγαλύτερη ποικιλία λέξεων, εξειδικευμένου λεξιλογίου και γλωσσών,
- μείωση προκατάληψης (bias) καθώς τα δεδομένα μπορεί να αντιπροσωπεύουν διαφορετικές χρονικές περιόδους ή/και πολιτισμούς.

Βήμα 4: Κατασκευή μετατροπέα edit distance

Ο μετατροπέας **L** βασίστηκε πάνω στην απόσταση Levenshtein. Η απόσταση Levenshtein υπολογίζει την απόσταση μεταξύ δύο string, δηλαδή πόσες αλλαγές (insertion, deletion,

substitution) χρειάζεται να γίνουν για να είναι ίδια τα strings. Αυτές οι αλλαγές θα υλοποιηθούν και στο transducer της άσκησης.

$$lev(a,b) = \begin{cases} |a| & if\ |b| = 0, \\ |b| & if\ |a| = 0, \\ lev(tail(a), b) & if\ |b| = |a|, \\ 1 + min \begin{cases} lev(tail(a), b) \\ len(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} & otheriwise \end{cases}$$

α) Ο μετατροπέας **L** αντιστοιχίζει:

- κάθε χαρακτήρα στον εαυτό του με βάρος 0 (no edit),
- κάθε χαρακτήρα στο ε βάρος 1 (deletion),
- το ε κάθε χαρακτήρα με βάρος 1 (insertion),
- κάθε χαρακτήρα σε κάθε άλλο χαρακτήρα με βάρος 1 (substitution).

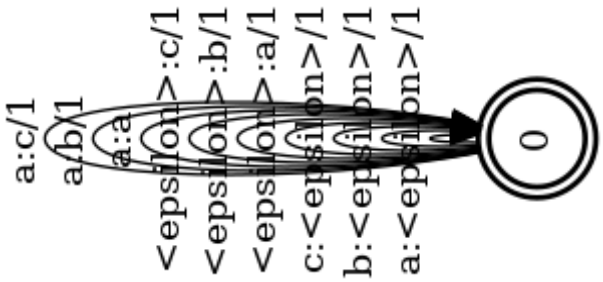
Εάν λάβουμε το shortest path αυτού του transducer προκύπτει η λέξη εισόδου χωρίς καμία αλλαγή καθώς κάθε χαρακτήρας αντιστοιχίζεται στον εαυτό του με βάρος 0.

δ) Ως επιπρόσθετα edits θα μπορούσαμε α προσθέσουμε:

- αλλαγή θέσης κοντινών χαρακτήρων (transposition),
- αντικαταστάσεις βάσει του context της λέξης.

ε) Σε περίπτωση που είχαμε στη διάθεση μας δεδομένα για τις συχνότητες εμφάνισης συγκεκριμένων λαθών πληκτρολόγησης, όπως για παράδειγμα η πληκτρολόγηση του χαρακτήρα «,» αντί για «M», ή για ορθογραφικά λάθη, όπως «e» αντί για «i» θα μπορούσαμε να εισάγουμε την πληροφορία αλλάζοντας τα βάρη, μειώνοντας για τα πιο συχνά και αυξάνοντας για τα πιο σπάνια λάθη.

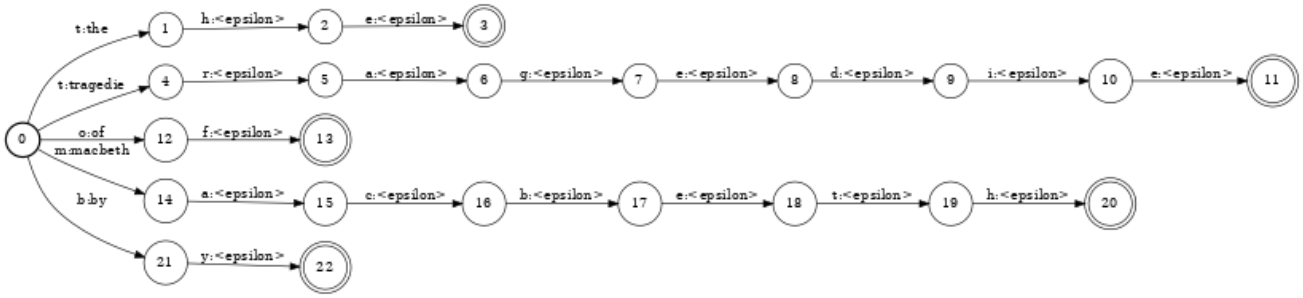
ζ) Χρησιμοποιώντας την εντολή fstdraw μπορούμε να οπτικοποιήσουμε τον **L**. Στην εικόνα 1 φαίνεται ο μετατροπέας για ένα υποσύνολο χαρακτήρων.



Εικόνα 1: Μετατροπέας L για χαρακτήρες a,b και c

Βήμα 5: Κατασκευή αποδοχέα λεξικού

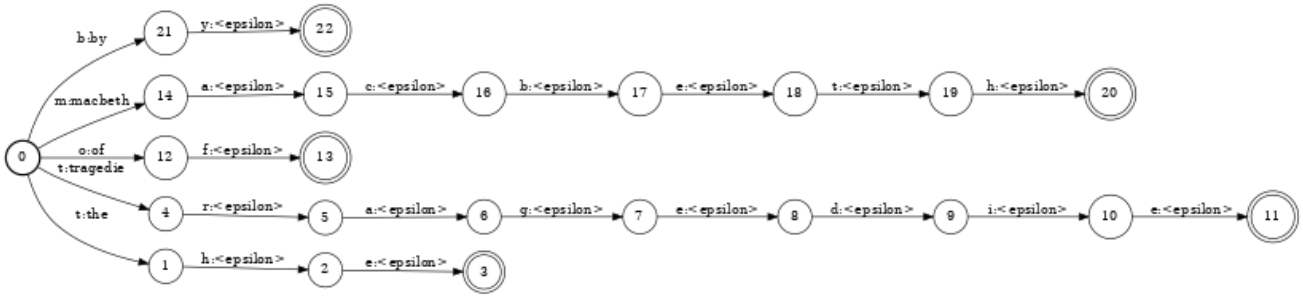
Ο αποδοχέας V αποδέχεται οποιαδήποτε λέξη που ανήκει στο λεξικό δηλαδή αποτελεί την ένωση (union) όλων των αυτομάτων που αποδέχονται τις λέξεις του λεξιλογίου. Στην εικόνα 2 φαίνεται η οπτικοποίηση ενός αποδοχέα μόνο για 5 λέξεις



Εικόνα 2: Αποδοχέας για τις έξεις the, tragedie, of, macbeth και by

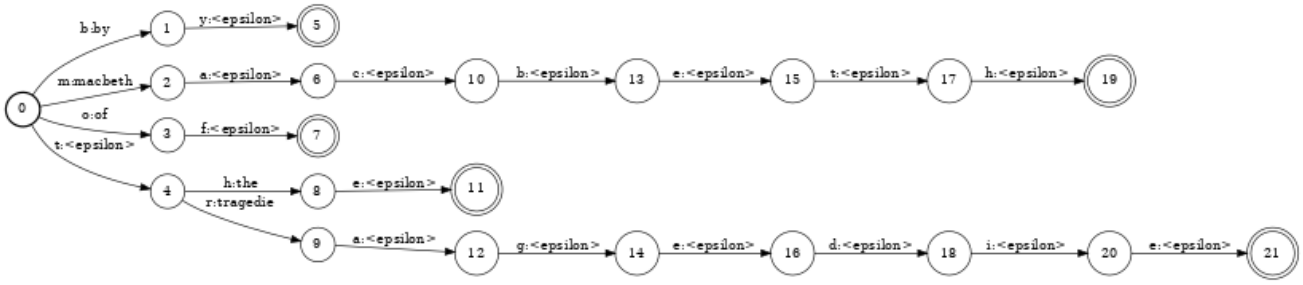
Για τη βελτιστοποίηση του μοντέλου καλούμε τις εξής συνασρτήσεις:

`fstrmepsilon`: Αυτή η συνάρτηση αφαιρεί τις μεταβάσεις όπου το input και output label είναι *ε* από το FST. Η αφαίρεσή τους απλοποιεί το FST και μειώνει το μέγεθός του. Στην εικόνα 3 δεν βλεπουμε κάποια διαφορά από την εικόνα 2 αφού δεν είχαμε καμία πράξη `<epsilon>:<epsilon>`.



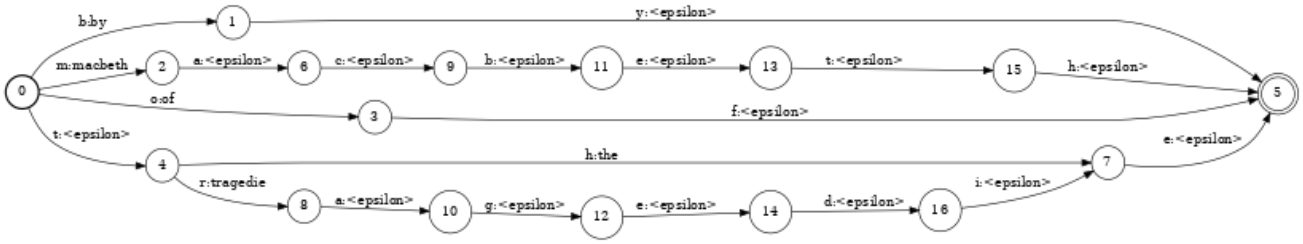
Εικόνα 3: Αποδοχέας εικόνας 2 μετά από `fstrmepsilon`

`fstdeterminize`: Αυτή η συνάρτηση μετατρέπει το FST από ένα μη-ντετερμινιστικό αυτόματο (NFA) σε ένα ντετερμινιστικό αυτόματο (DFA). Ουσιαστικά μετατρέπει το αυτόματο ώστε από κάθε κόμβο να φεύγει μοναδική ακμή για ένα input label. Για παράδειγμα αν έχουμε τις λέξεις «the» και «tragedie» οι δύο λέξεις θα έχουν κοινή τη πρώτη καταστάση και θα διακλαδίζονται μετά ανάλογα με το input (εικόνα 4). Έτσι εξασφαλίζει πως θα παράγεται κάθε φορά η ίδια έξοδος για συγκεκριμένη είσοδο.



Εικόνα 4: Αποδοχέας εικόνας 3 μετά από `fstdeterminize`

`fstminimize`: Αυτή η συνάρτηση ελαχιστοποιεί τον αριθμό των καταστάσεων στο FST. Αυτό μπορεί να μειώσει σημαντικά το μέγεθος της FST, καθιστώντας την ταχύτερη στη διέλευση και τη χρήση της. Το αποτέλεσμα της συνάρτησης φαίνεται στην εικόνα 5.



Εικόνα 5: Αποδοχέας εικόνας 4 μετά από `fstminimize`

Η πολυπλοκότητα traversal ενός DFA είναι $O(n)$, όπου n είναι το μήκος της συμβολοσειράς εισόδου. Αυτό συμβαίνει επειδή ένα DFA έχει πάντα μια μοναδική μετάβαση για κάθε σύμβολο εισόδου και κατάσταση, οπότε μπορεί να επεξεργαστεί τη συμβολοσειρά εισόδου με γραμμικό τρόπο. Αντίθετα, η πολυπλοκότητα διέλευσης ενός μη ντετερμινιστικού αυτομάτου είναι $O(2^n)$, όπου n είναι ο αριθμός των καταστάσεων, επειδή μπορεί να χρειαστεί να εξερευνήσει πολλαπλά μονοπάτια για να βρει τη σωστή έξοδο.

Ο αριθμός των ακμών σε ένα ντετερμινιστικό αυτόματο είναι συνήθως μεγαλύτερος από ό,τι σε ένα μη ντετερμινιστικό αυτόματο, επειδή κάθε κατάσταση πρέπει να έχει μια μοναδική μετάβαση για κάθε σύμβολο εισόδου.

Βήμα 6: Κατασκευή ορθογράφου

Εκτελούμε την συνάρτηση `fstarcsort` η οποία ταξινομεί τις ακμές των L και V ανάλογα με τα output και input labels αντίστοιχα και εκτελούμε `fstcompose` για να συνθέσουμε τον `min edit distance spell checker S`. Ο transducer διορθώνει τις λέξεις χωρίς να λαμβάνει υπόψιν του κάποια γλωσσική πληροφορία, με κριτήριο να κάνει τις ελάχιστες δυνατές μετατροπές στην λέξη εισόδου. Η συμπεριφορά του μετατροπέα διαφέρει ανάλογα με τα βάρη των edits:

- αν τα edits είναι **ισοβαρή** κάθε πιθανή λέξη με ίδιο αριθμό edits έχει την ίδια πιθανότητα εμφάνισης, αφού δεν υπάρχει προτίμηση σε κάποιο edit,
- για **διαφορετικά βάρη** για κάθε edit ορισμένες λέξεις έχουν μεγαλύτερη πιθανότητα εμφάνισης. Για παράδειγμα για τον εξής συνδυασμό $cost(insertion) = cost(deletion) = 1, cost(substitution) = 1.5$ είναι πιο πιθανό να εμφανιστεί μία μικρότερη λέξη που εμπεριέχεται στην λανθασμένη παρά να γίνει αντικατάσταση χρακτήρων.

Συγκεκριμένα αποτελέσματα φαίνονται στον πίνακα 1.

Λέξη προς διόρθωση	**Πιθανές λέξεις**
cit	cut, cin, sit, hit, fit

cwt	lot, cas, cap, cut, cat

Πίνακας 1: Αποτελέσματα

Παρά το γεγονός ότι οι δύο λέξεις απέχουν την ίδια απόσταση από τη λέξη «cut» τα αποτελέσματα διαφέρουν αρκετά καθώς η λέξη «cit» είναι πιο κοντά σε διαφορετικές λέξεις του λεξιλογίου σε σχέση με τη λέξη «cwt».

Βήμα 7: Δοκιμή ορθογράφου

Στη συνέχεια θα προσπαθήσουμε να διορθώσουμε κάποιες λέξεις από το αρχείο spell_test.txt. Στον πίνακα 2 φαίνονται τα αποτελέσματα για ορισμένες λέξεις.

Λέξη προς διόρθωση	**Σωστή λέξη**	**Πιθανές λέξεις**
contentid	contented	content, contented, contend
begining	beginning	begging, beginning
proble	problem	prone, problem
dirven	driven	dirge, given, driven
poety	poetry	poet, piety, poetry

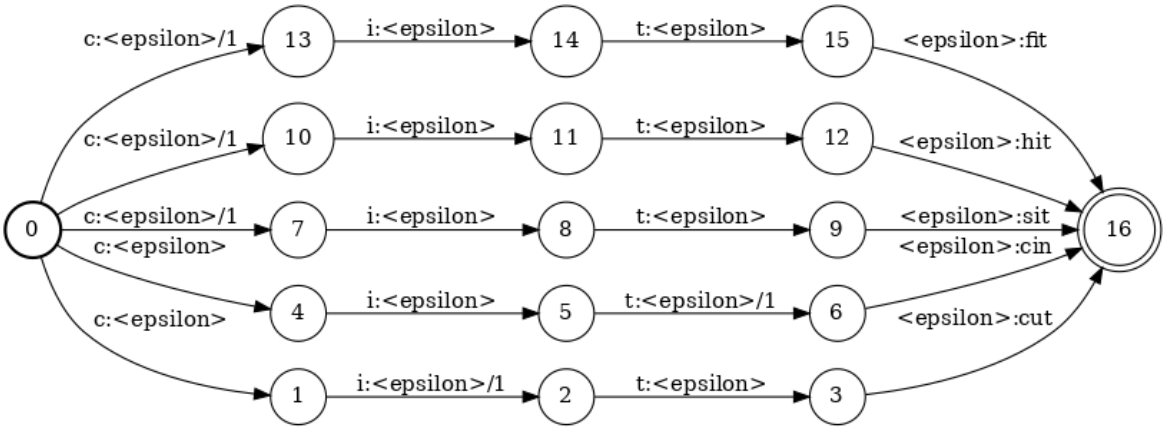
Πίνακας 2: Οι τρεις πιο πιθανές λέξεις κατά τη διόρθωση

Υπολογίζοντας το accuracy για την πρώτη πιθανή λέξη παίρνουμε αποτέλεσμα **46**. **15**%.

Σε λανθασμένες διορθώσεις η αιτία μπορεί να είναι είτε πως η λέξη δεν υπάρχει στο λεξικό οπότε δεν την βρίσκει (transportability respectability) είτε επειδή βρίσκει ισοβαρές μονοπάτι προς άλλη λέξη (localy -> local).

Η πρόβλεψη έγινε με τη συνάρτηση **`predict_word(word, mode, transducer, verbose)`** που υλοποιήσαμε η οποία δημιουργεί ένα txt αρχείο με τις μεταβάσεις για τον acceptor της λανθασμένης λέξης και ύστερα τρέχει σε bash το `cpredict.sh` το οποίο εκτελεί τα εξής:

- compile το wrong_word.bin.fst δηλαδή το binary της λανθασμένης λέξης,
- compose με τον ορθογράφο,
- shortest_path στον composed γράφο,
- fstrmepsilon για αφαίρεση των epsilon μεταβάσεων,
- fsttopshort που αλλάζει τα ids για να μεταβαίνουν από υψηλότερο σε χαμηλότερο id,
- fstprint για να πάρουμε τα αποτελέσματα των outputs,
- fstdraw για οπτικοποίηση του γράφου της ελεγμένης λέξης.



Εικόνα 6: Αποτέλεσμα για τη λέξη «cit»

Τέλος, ελέγχουμε από το αποτέλεσμα της fstprint προβλεπόμενες λέξεις και επιστρέφουμε έναν πίνακα που τις περιέχει. Οι παράμετροι της συνάρτησης είναι οι εξής:

- **`word`**: η λέξη προς διόρθωση,
- **`mode`**: 0 για λήψη των 5 πρώτων λέξεων, 1 για λήψη της πρώτης λέξης,
- **`transducer`**: ορθογράφος που θα χρησιμοποιηθεί για τη πρόβλεψη,
- **`verbose`**: 1 αν θέλουμε να εμφανιστούν τα αποτελέσματα.

Βήμα 8: Υπολογισμός κόστους των edits

Σε αυτό το βήμα θα διορθώσουμε τον vanilla Levenshtein εξάγοντας πληροφορία από το wiki.txt για την συχνότητα εμφάνισης ορισμένων edits. Αφού εξάγουμε τη συχνότητα τη μετατρέπουμε σε βάρος παίρνοντας τον αρνητικό δεκαδικό λογάριθμο της πιθανότητας.

Λέξη προς διόρθωση	Πιθανές λέξεις
cit	city, sit, it, cut, clit
cwt	cow, caw, wet, wit

Πίνακας 3: Αποτελέσματα για τον νέο levenshtein

Λέξη προς διόρθωση	Σωστή λέξη	Πιθανές λέξεις
contentid	contented	content, contented, contend, contented
begining	beginning	beginning, beginnings
proble	problem	probable, robe, parable, problems, problem
dirven	driven	dived, divine, dive, driven
poety	poetry	poets, piety, poetry, poet

Πίνακας 4: Αποτελέσματα για τον νέο levenshtein

Υπολογίζοντας το accuracy για την πρώτη πιθανή λέξη παίρνουμε αποτέλεσμα **58.97**%. Παρατηρούμε με την πρόσθετη πληροφορία της συχνότητας εμφάνισης των edits το αποτέλεσμα βελτιώθηκε κατά **10**%. Επίσης παρατηρούμε ότι οι προβλεπόμενε λέξεις είναι αρκετά διαφορετικές από τον vanilla Levenshtein, πράγμα λογικό αφού κάποιες μεταβάσεις είναι προτιμότερες από άλλες.

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

Σε αυτό το σημείο θα βελτιώσουμε τον αποδοχέα **V** εισάγοντας με τον ίδιο τρόπο την πληροφορία για τη συχνότητα εμφάνισης λέξεων από το corpus που διαθέτουμε. Δημιουργούμε έναν acceptor **W** ο οποίος αποτελείται από μια κατάσταση και αντιστοιχίζει κάθε λέξη στον εαυτό της με βάρος τον αρνητικό λογάριθμο της συχνότητας εμφάνισης της λέξης και δημιουργούμε τον ορθογράφο **LVW** πραγματοποιώντας τη σύνθεση του Levenshtein transducer L με τον acceptor του λεξιλογίου **V** και το γλωσσικό μοντέλο **W**. Στους πίνακες 5 και 6 φαίνονται τα αποτελέσματα για τον ορθογράφο **LVW**.

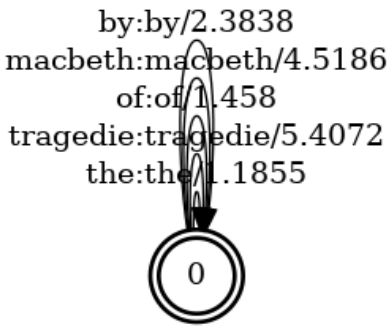
Λέξη προς διόρθωση	**Πιθανές λέξεις LVW**
cit	his, in, it
cwt	the, but, not, it

Πίνακας 5: Αποτελέσματα για ορθογράφο LVW

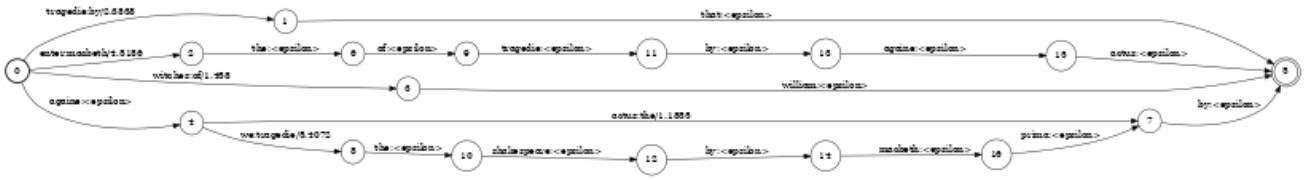
Λέξη προς διόρθωση	**Σωστή λέξη**	**Πιθανές λέξεις**
contentid	contented	continued, contented, contend, content, contented
begining	beginning	being, beginning
proble	problem	people
dirven	driven	even, in, given
poety	poetry	of, poet, they, the, not

Πίνακας 6: Αποτελέσματα για ορθογράφο LVW

Υπολογίζοντας το accuracy για την πρώτη πιθανή λέξη παίρνουμε αποτέλεσμα **35.9**%. Όπως βλέπουμε η απόδοση πέφτει κατά πολύ καθώς έχουμε εισάγει μεγάλο bias για το λεξικό και ορισμένα μονοπάτια προς λάθος λέξεις είναι προτιμότερα ακόμα και αν διαθέτουν πολλά edits.



Εικόνα 7: Οπτικοποίηση W



Εικόνα 8: Οπτικοποίηση VW

Παρατηρούμε πως ο γράφος **VW** είναι ίδιος με τον γράφο **V** με τα βάρη που ορίζει ο **W**.

Βήμα 10: Αξιολόγηση των ορθογράφων

Στον πίνακα 7 φαίνονται τα αποτελέσματα συγκριτικά για τους ορθογράφους **LV**, **LVW**, **EV** και **EVW**.

Ορθογράφος	Accuracy
LV	58.15%
LVW	43.49%
EV	67.78%
EVW	66.30%

Πίνακας 7: Συγκριτικά αποτελέσματα ορθογράφων

Παρατηρούμε πως η πληροφορία για τη συχνότητα των αλλαγών βελτίωσε την απόδοση του ορθογράφου παρ' όλα αυτά η πληροφορία συχνότητα εμφάνισης λέξεων εισήγαγε μεγάλο bias και μείωσε την απόδοση του ορθογράφου.

Μέρος 2: Εξοικείωση με το W2V

Βήμα 12: Εξαγωγή αναπαραστάσεων word2vec

Μία πιο σύγχρονη μέθοδος αναπαράστασης λέξεων είναι με τη χρήση του word2vec μοντέλου, δηλαδή τη δημιουργία word embeddings που προκύπτουν από την εκπαίδευση ενός νευρωνικού δικτύου που προβλέπει τη λέξη βάσει του context σε ένα παράθυρο γύρω της. Αυτό μας επιτρέπει να κάνουμε feature extraction στις λέξεις και να τις αναπαραστήσουμε ως αριθμητικά διανύσματα. Για τη συγκεκριμένη εφαρμογή θα εφαρμόζουμε τη βιβλιοθήκη genism πάνω στο Gutenberg corpus για να εξάγουμε δικά μας word embeddings. Το νευρωνικό δίκτυο αποτελείται από ένα layer (CBOW model) και το χρησιμοποιούμε για να παράξουμε εκατονταδιάστατα word embeddings σε παράθυρο 5 λέξεων σε 1000 εποχές. Η εκπαίδευση έγινε πάνω στο tokenized Corpus, δηλαδή δημιουργήσαμε μία λίστα λιστών που αποτελείται από τις tokenized προτάσεις. Επίσης χρησιμοποιήθηκαν και προεκπαιδευμένα GoogleNews vectors για σύγκριση των αποτελεσμάτων.

γ, στ) Στον πίνακα 8 φαίνονται οι σημασιολογικά κοντινές λέξεις που προέκυψαν από την παραπάνω διαδικασία για ορισμένα παραδείγματα.

Αρχική λέξη	**Σημασιολογικά κοντινές λέξεις**	
	Gutenberg vectors	**GoogleNews vectors**
bible	surplice, incitement, headship	Bible, scriptures, scripture
book	written, chronicles, jasher	tome, books, memoir
bank	pool, ridge, table	banks, banking, Bank
water	waters, river, rivers	potable_water, Water, sewage

Πίνακας 8: Σημασιολογικά κοντινές λέξεις

Παρατηρούμε πως τα αποτελέσματα για τα Vectors του Gutenberg έχουν πράγματι σημασιολογικά κοντινές λέξεις ωστόσο τα αποτελέσματα δεν είναι βέλτιστα όπως για τις λέξεις bank και bible, δεδομένου ότι υπάρχει μεγάλη εξάρτηση από το training set. Δεδομένου πως τα GoogleNews vectors έχουν παραχθεί από την εκπαίδευση ενός w2v σε ένα τεράστιο σύνολο δεδομένων είναι λογικό να παρουσιάζει καλύτερα αποτελέσματα παρ' όλα αυτά εμφανίζονται και λέξεις όπως Bible, Bank, potable_water και Water λόγω διαφορετικού tokenization.

Αλλάζοντας το παράθυρο

δ, ζ) Μια άλλη ιδιότητα των word2vec είναι η δυνατότητα τους να εκφράζουν αλγεβρικά σημασιολογικές αναλογίες για έννοιες. Με τον όρο σημασιολογική αναλογία εννοούμε το εξής: αν έχουμε δύο ζευγάρια εννοιών (a,b) και (c,d) τότε υπάρχει σημασιολογική αναλογία αν ικανοποιείται η πρόταση «το a είναι για το b ότι το c για το d».

$$v = w2v(a) - w2v(b) + w2v(d)$$

Στον πίνακα 9 φαίνονται τα αποτελέσματα για ορισμένες τριπλέτες.

Τριπλέτα	**Λέξη με μέγιστο similarity**	
	Gutenberg vectors	**GoogleNews vectors**
girls, queen, kings	girls	Boys
good, taller, tall	taller	Better
france, paris, london	france	Britain

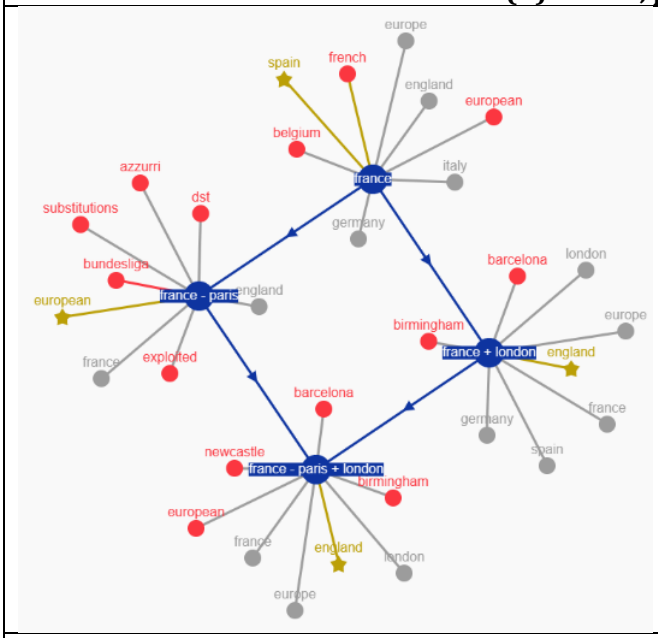
Πίνακας 9: Λέξεις που προκύπτουν από vector analogy

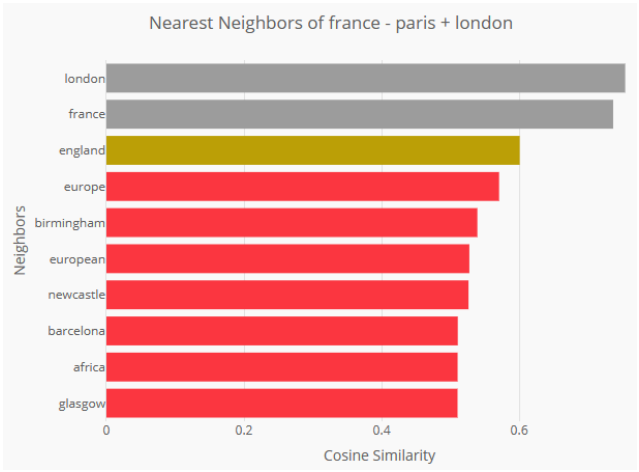
Παρατηρούμε πως τα custom word embeddings δεν είναι εκπαιδευμένα σε αρκετά μεγάλο dataset για να κατανοήσουν την εννοιολογική σχέση της πράξεις και δεν επιστρέφει το σωστό αποτέλεσμα. Από την άλλη τα GoogleNews vectors έχουν την δυνατότητα να κατανοήσουν το context «το a είναι για το b ότι το c για το d» και επιστρέφουν σωστά αποτελέσματα.

Βήμα 13: Οπτικοποίηση των word embeddings

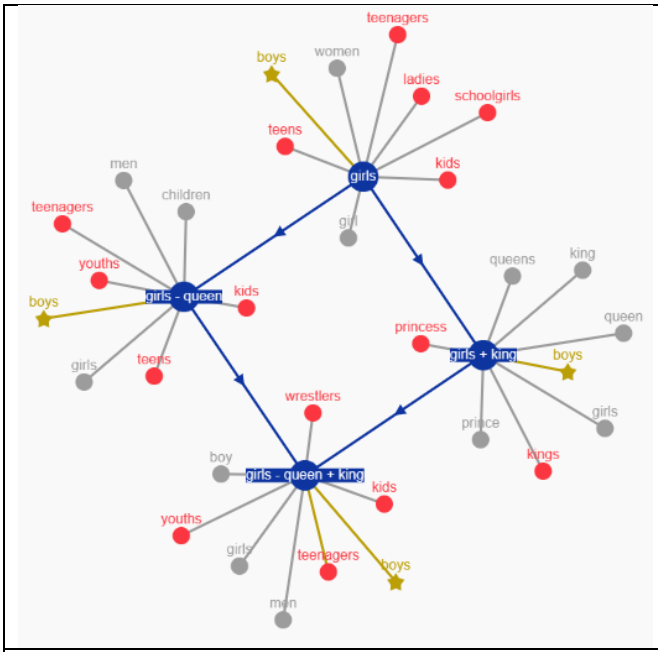
α) Χρησιμοποιούμε το εργαλείο dash gallery για να πειραματιστούμε με τις πράξεις μεταξύ λέξεων. Τα αποτελέσματα φαίνονται στην εικόνα 9.

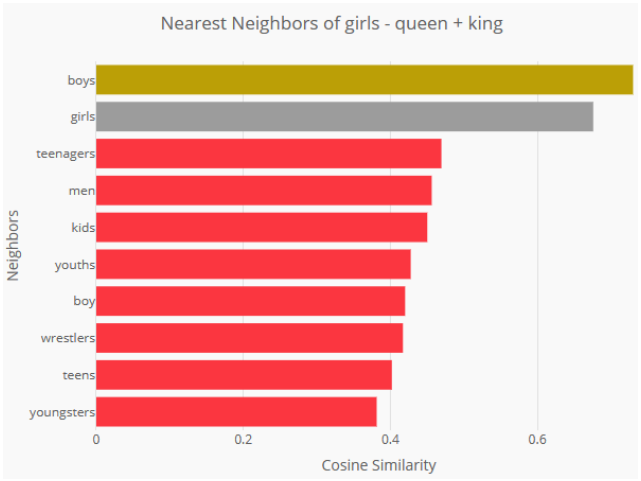
(α) france, paris, london



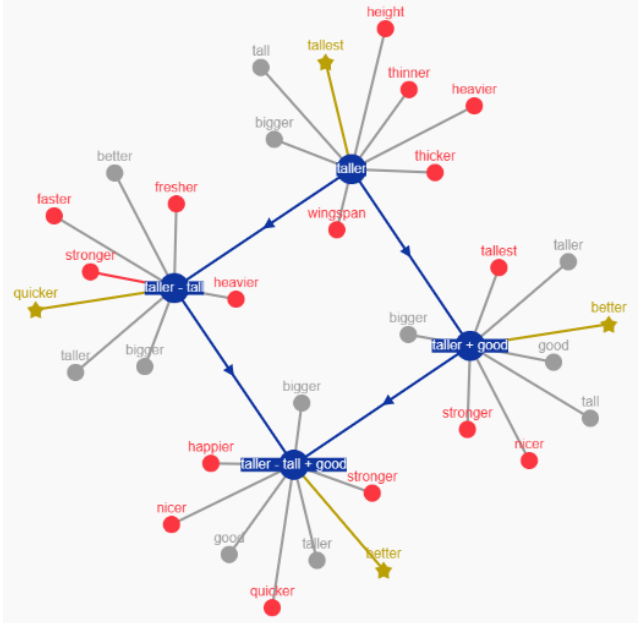


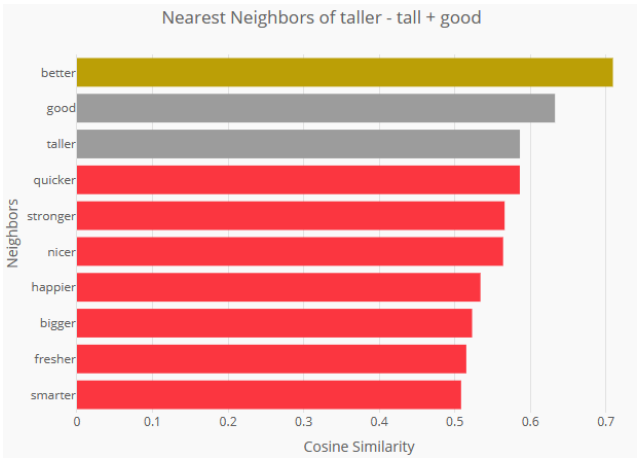
(β) girls, queen, kings





(γ) good, taller, tall

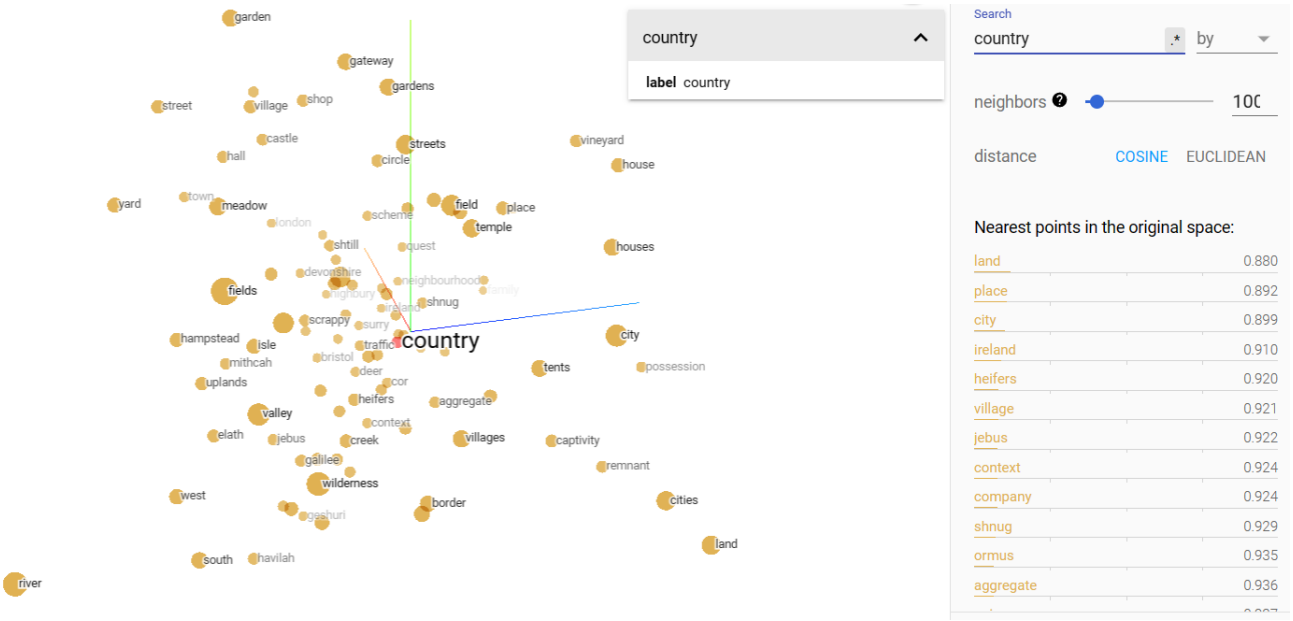




Πίνακας 10: Αναπαράσταση στο χώρο (αριστερά) και γράφημα με cosines similarity (δεξιά) για τις τριπλέτες

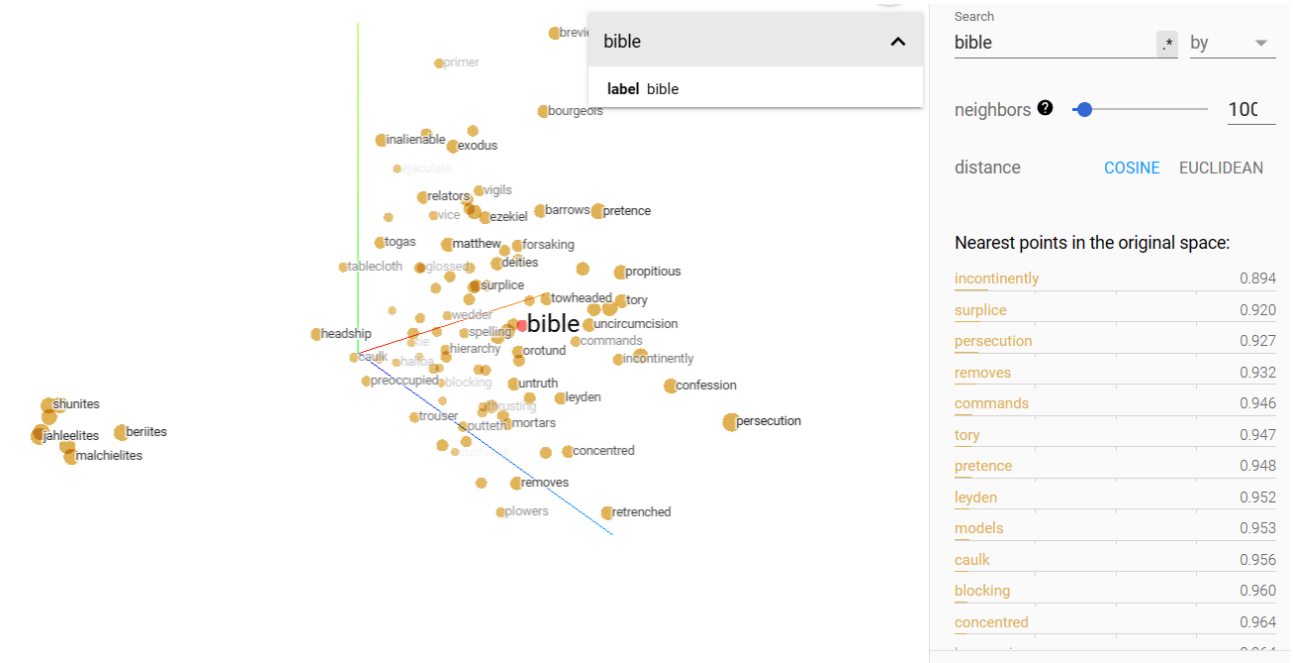
Παρατηρούμε πως το εργαλείο αναγνωρίζει τη σημασιολογική σχέση της πρότασης και ότι οι λέξεις με τις οποίες γίνεται η πράξη έχουν μεγάλο cosine similarity.

γ) Στο σημείο που βρισκόμαστε κάθε word embedding έχει 100 διαστάσεις οπότε για να το οπτικοποιήσουμε και να παρατηρήσουμε σχέσεις μεταξύ λέξεων χρειαζόμαστε κάποια μέθοδο μείωσης διαστατικότητας. Για το σκοπό αυτό θα χρησιμοποιήσουμε το online εργαλείο της Tensorflow για να οπτικοποιήσουμε τα δεδομένα στις 3 διαστάσεις χρησιμοποιώντας PCA. Παράλληλα θα εκτελέσουμε T-SNE αλγόριθμο ο οποίος μετατρέπει ομοιότητες μεταξύ σημείων σε από κοινού κατανομές πιθανότητας και προσπαθεί να ελαχιστοποιήσει την απόκλιση μεταξύ αυτών των πιθανοτήτων των χαμηλών διαστάσεων embeddings και των δεδομένων υψηλών διαστάσεων.



Εικόνα 9: Οπτικοποίηση word embeddings για τη λέξη country (αριστερά) και οι κοντινότερες σημασιολογικά λέξεις (δεξιά) με PCA

Παρατηρούμε πως οι πιο κοντινές λέξεις στη λέξη country είναι οι land, place και city, οι οποίες πράγματι είναι κοντά σημασιολογικά.



Εικόνα 10: : Οπτικοποίηση word embeddings για τη λέξη bible (αριστερά) και οι κοντινότερες σημασιολογικά λέξεις (δεξιά) με PCA

Στην εικόνα 10 βλέπουμε πως τα αντίστοιχα αποτελέσματα για τη λέξη bible δεν είναι ικανοποιητικά.

Στις εικόνες 9 και 10 έχει χρησιμοποιηθεί μέθοδος μείωσης PCA. Χρησιμοποιώντας T-SNE τα αποτελέσματα είναι παρόμοια.

Βήμα 14: Ανάλυση συναισθήματος με word2vec embeddings

Ο μέσος όρος των w2v διανυσμάτων κάθε λέξης που περιέχονται σε μία πρόταση μπορεί να δώσει μία αναπαράσταση της πρότασης (Neural Bag of Words). Σε αυτό το βήμα θα κάνουμε χρήση αυτών των αναπαραστάσεων για την ανάλυση συναισθήματος σε movie reviews.

Κατασκευάζουμε δύο Neural Bag of Words ένα από τα custom emdeddings που δημιουργήσαμε από το Gutenberg corpus και ένα από τα GoogleNews vectors. Τα NBOWs τα φτιάχνουμε υπολογίζοντας τον μέσο όρο των embeddings των λέξεων μίας πρότασης και κάνοντας concatenate σε ένα array. Για τις OOV (Out Of Vocabulary) λέξεις προσθέτουμε μηδενικό διάνυσμα. Η ταξινόμηση έγινε χρησιμοποιώντας Logistic Regression με solver = LBFGS για 200 iterations.

Τα αποτελέσματα για τα δύο NBOWs φαίνονται στον πίνακα 11.

Custom embeddings	**GoogleNews vectors**
74.8%	82.53%

Πίνακας 11: Αποτελέσματα για τα δύο NBOWs

Το μοντέλο με τα GoogleNews embeddings παρουσίασε καλύτερη απόδοση. Γενικότερα για τη βελτίωση της απόδοσης του μοντέλου θα ήταν ιδανικό να διαθέταμε περισσότερα δεδομένα και με μεγάλη ποικιλία σε context καθώς πολλές φορές ίδιες λέξεις είναι διαφορετικά συναισθηματικά φορτισμένες. Η πρόσθεση σημείων στίξης, όπως θαυμαστικό, θα βοηθούσε επίσης στην απόδοση του μοντέλου.