ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Επεξεργασία Φωνής και Φυσικής Γλώσσας

1η Εργαστηριακή Άσκηση: Εισαγωγή στις γλωσσικές αναπαραστάσεις

ΔΙΑΔΙΚΑΣΤΙΚΑ ΕΡΓΑΣΤΗΡΙΟΥ

Εξηγήστε περιεκτικά και επαρκώς την εργασία σας. Κώδικας χωρίς σχόλια δεν θα βαθμολογηθεί. Επιτρέπεται η συνεργασία εντός ομάδων των 2 ατόμων εφόσον φοιτούν στο ίδιο πρόγραμμα σπουδών (είτε ομάδες προπτυχιακών, είτε ομάδες μεταπτυχιακών). Κάθε ομάδα 2 ατόμων υποβάλλει μια κοινή αναφορά που αντιπροσωπεύει μόνο την προσωπική εργασία των μελών της. Αν χρησιμοποιήσετε κάποια άλλη πηγή εκτός των βιβλίων και του εκπαιδευτικού υλικού του μαθήματος, πρέπει να το αναφέρετε. Η παράδοση της αναφοράς και του κώδικα της εργασίας θα γίνει ηλεκτρονικά στο helios. **Επισημαίνεται ότι απαγορεύεται η ανάρτηση των λύσεων των εργαστηριακών ασκήσεων στο github, ή σε άλλες ιστοσελίδες.**

ΒΟΗΘΗΤΙΚΟ ΥΛΙΚΟ

Στον ακόλουθο σύνδεσμο βρίσκονται τα παραδείγματα που είδαμε στο φροντιστήριο του εργαστηρίου. Στην ακόλουθη σελίδα μπορείτε να βρείτε βοηθητικό κώδικα σχετικά με τα εργαστήρια. Στη σελίδα αυτή μπορείτε επίσης να υποβάλετε απορίες και ερωτήσεις προς τους βοηθούς του μαθήματος με μορφή issues. **Ερωτήσεις αναφορικά με το εργαστήριο που θα γίνονται μέσω mail δεν θα λαμβάνουν απάντηση**.

ΠΕΡΙΓΡΑΦΗ

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

Η άσκηση χωρίζεται σε δύο μέρη.

Στο πρώτο μέρος θα χρησιμοποιήσουμε απλά γλωσσικά μοντέλα και μετασχηματισμούς για τη δημιουργία ενός ορθογράφου βασισμένο σε μηχανές πεπερασμένης κατάστασης. Θα γίνει χρήση της βιβλιοθήκης openfst.

Μπορείτε να εγκαταστήσετε το Openfst με αυτό το script:
https://github.com/slp-ntua/slp-labs/blob/master/lab1/install_openfst.sh

Το documentation βρίσκεται εδώ: http://www.openfst.org/ μαζί με παραδείγματα
http://www.openfst.org/twiki/bin/view/FST/FstExamples.

Αυτή η άσκηση θα κάνουμε χρήση Python και Bash scripting.
Για την εξοικείωση με την Python μπορείτε να ανατρέξετε στα εισαγωγικά εργαστήρια:
https://github.com/slp-ntua/prep-lab
Για Bash scripting μπορείτε να δείτε μια εισαγωγή σε αυτό το Blog:
http://matt.might.net/articles/bash-by-example/

ΠΡΟΣΟΧΗ: **Χρησιμοποιείστε την έκδοση 1.6.1 του Openfst. Επόμενες εκδόσεις έχουν σημαντικό bug στη σύνθεση FST**

ΕΚΤΕΛΕΣΗ

ΜΕΡΟΣ 1: ΚΑΤΑΣΚΕΥΗ ΟΡΘΟΓΡΑΦΟΥ

Βήμα 1: Κατασκευή corpus

Σε αυτό το βήμα θα δημιουργήσουμε ένα μικρό corpus από διαθέσιμες πηγές στο διαδίκτυο.
Το project Gutenberg είναι μια πηγή για βιβλία που βρίσκονται στο public domain και μία καλή πηγή για γρήγορη συλλογή δεδομένων για κατασκευή γλωσσικών μοντέλων.
α) Σας παρέχεται το script https://github.com/slp-ntua/slp-labs/blob/master/lab1/scripts/fetch_gutenberg.py το οποίο κατεβάζει το gutenberg coprus που παρέχεται από τη βιβλιοθήκη NLTK. Αυτό το corpus θα χρησιμοποιηθεί για την εξαγωγή στατιστικών κατά την κατασκευή γλωσσικών μοντέλων.
Επίσης πραγματοποιούμε μια προεπεξεργασία για τον καθαρισμό του corpus. Διαβάστε και εκτελέστε το script. Σχολιάστε εν συντομία τα βήματα της προεπεξεργασίας του κειμένου. Σχολιάστε περιπτώσεις όπου θα θέλαμε να διατηρήσουμε τα σημεία στίξης και να μην πραγματοποιήσουμε τόσο επιθετικό preprocessing.
β) Μπορείτε επίσης να επεκτείνετε αυτό το corpus με περισσότερα βιβλία ή κείμενα από άλλες πηγές για την κατασκευή ενός μεγαλύτερου corpus. Ποια είναι το πλεονεκτήματα αυτής της τεχνικής (εκτός από το μέγεθος των δεδομένων) Αναφέρετε τουλάχιστον 2.

Βήμα 2: Κατασκευή λεξικού

Σε αυτό το βήμα θα κατασκευάσουμε ένα λεξικό θα χρησιμεύσει στην κατασκευή του ορθογράφου.
α) Δημιουργήστε ένα dictionary που περιέχει σαν keys όλα τα μοναδικά tokens που βρίσκονται στο corpus που κατασκευάσαμε στο Βήμα 1. Τα values θα πρέπει να είναι ο αριθμός εμφανίσεων του token στο κείμενο.
β) Φιλτράρετε όλα τα tokens που εμφανίζονται λιγότερες από 5 φορές. Γιατί χρειάζεται αυτό το βήμα;
γ) Γράψτε στο αρχείο vocab/words.vocab.txt το λεξικό σε δύο tab separated στήλες, όπου η πρώτη στήλη περιέχει τα tokens και η δεύτερη στήλη τους αντίστοιχους αριθμούς εμφανίσεων.

Βήμα 3: Δημιουργία συμβόλων εισόδου/εξόδου

α) Για την κατασκευή των FSTs χρειάζονται αρχεία που να αντιστοιχίζουν τα σύμβολα εισόδου (ή εξόδου) σε μοναδικά indices. Γράψτε μια συνάρτηση Python που αντιστοιχίζει κάθε lowercase χαρακτήρα της Αγγλικής γλώσσας σε ένα αύξοντα ακέραιο index. Το πρώτο σύμβολο με index 0 είναι το ε (<eps>). Το αποτέλεσμα θα γραφτεί στο αρχείο vocab/chars.syms με αυτή τη μορφή
http://www.openfst.org/twiki/pub/FST/FstExamples/ascii.syms
β) Κατά αντιστοιχία δημιουργείστε το αρχείο words.syms που αντιστοιχίζει κάθε λέξη (token) από το λεξιλόγιο που κατασκευάσατε στο Βήμα 2 σε ένα μοναδικό ακέραιο index. Το αποτέλεσμα πρέπει να γραφτεί στο vocab/words.syms.

Βήμα 4: Κατασκευή μετατροπέα edit distance

Για τη δημιουργία του ορθογράφου θα χρησιμοποιήσουμε ένα μετατροπέα **L** βασισμένο στην απόσταση Levenshtein (https://en.wikipedia.org/wiki/Levenshtein_distance).
Θα χρησιμοποιήσουμε 3 τύπους από edits: εισαγωγές χαρακτήρων, διαγραφές χαρακτήρων και αντικαταστάσεις χαρακτήρων. Κάθε ένα από αυτά τα edits χαρακτηρίζεται από ένα κόστος. Σε αυτό το στάδιο θα θεωρήσουμε ότι όλα τα edits έχουν κόστος 1.

α) Κατασκευάστε ένα μετατροπέα **L** με μία κατάσταση που υλοποιεί την απόσταση Levenshtein αντιστοιχίζοντας:

1) κάθε χαρακτήρα στον εαυτό του με βάρος 0 (no edit)
2) κάθε χαρακτήρα στο *ε* με βάρος 1 (deletion)
3) το *ε* σε κάθε χαρακτήρα με βάρος 1 (insertion)
4) κάθε χαρακτήρα σε κάθε άλλο χαρακτήρα με βάρος 1.

Τι κάνει αυτός ο μετατροπέας σε μια λέξη εισόδου αν πάρουμε το shortest path;
β) Αποθηκεύστε το αρχείο fsts/L.fst που περιέχει την περιγραφή του μετατροπέα σε Openfst text format.
γ) Χρησιμοποιείστε την fstcompile για να κάνετε compile τον L και αποθηκεύστε το αποτέλεσμα στο fsts/L.binfst.
δ) Δώστε ένα παράδειγμα από άλλα πιθανά edits που θα μπορούσαμε να συμπεριλάβουμε.
ε) Δώστε παραδείγματα για το πώς θα μπορούσαμε να βελτιώσουμε τα βάρη των edits εισάγοντας πρότερη γνώση.
ζ) Χρησιμοποιείστε την fstdraw για να σχεδιάσετε τον **L**. Μπορείτε να χρησιμοποιήσετε ένα μικρό υποσύνολο των χαρακτήρων για πιο εύκολη οπτικοποίηση.
Hint: Για τα δ) και ε) σκεφτείτε συχνά λάθη ορθογραφίας / πληκτρολόγησης.

Βήμα 5: Κατασκευή αποδοχέα λεξικού

α) Κατασκευάστε έναν αποδοχέα **V** με μία αρχική κατάσταση που αποδέχεται κάθε λέξη του λεξικού από το Βήμα 2. Τα βάρη όλων των ακμών είναι 0. Αυτό είναι ένας αποδοχέας ο οποίος απλά αποδέχεται μια λέξη αν ανήκει στο λεξικό.
β) Καλέστε τις fstrmepsilon, fstdeterminize και fstminimize για να βελτιστοποιήσετε το μοντέλο. Τι κάνει κάθε συνάρτηση και ποιο είναι το όφελος; Σχολιάστε την πολυπλοκότητα traversal και τον αριθμό των ακμών σε ένα ντετερμινιστικό και σε ένα μη ντετερμινιστικό αυτόματο.
γ) Αποθηκεύστε το αρχείο περιγραφής του αποδοχέα στο fsts/V.fst σε Openfst text format.
δ) Χρησιμοποιείστε την fstcompile για να κάνετε compile τον αποδοχέα στο fsts/V.binfst
ε) Χρησιμοποιήστε την fstdraw για να σχεδιάσετε τον **V**. Μπορείτε να χρησιμοποιήσετε ένα μικρό υποσύνολο των λέξεων για εύκολη οπτικοποίηση. Συγκεκριμένα σχεδιάστε τον **V** πριν τις πράξεις fstrmepsilon. fstdeterminize, fstminimize καθώς και μετά από κάθε μια από αυτές. Συγκρίνετε τα FSTs που προκύπτουν.
Hint: Χρησιμοποιείστε τα chars.syms για σύμβολα εισόδου και τα words.syms για σύμβολα εξόδου

Βήμα 6: Κατασκευή ορθογράφου

α) **Συνθέστε** τον Levenshtein transducer **L** με τον αποδοχέα **V** του ερωτήματος 5α παράγοντας τον min edit distance spell checker **S**. Αυτός ο transducer διορθώνει τις λέξεις χωρίς να λαμβάνει υπόψιν του κάποια γλωσσική πληροφορία, με κριτήριο να κάνει τις ελάχιστες δυνατές μετατροπές στην λέξη εισόδου. Αναλύστε τη συμπεριφορά αυτού του μετατροπέα

1) στην περίπτωση που τα edits είναι ισοβαρή,
2) για διαφορετικά βάρη των edits (πχ cost(insertion)=cost(deletion)=1, cost(substitution)=1.5)

Hint χρησιμοποιήστε την fstcompose
β) Ποιες είναι οι πιθανές προβλέψεις του min edit spell checker αν η είσοδος είναι η λέξη cit και η λέξη cwt?
(Hint: Μπορεί να χρειαστεί να καλέσετε την fstarcsort στις εξόδους του transducer ή/και στις εισόδους του acceptor)

Βήμα 7: Δοκιμή ορθογράφου

α) Κατεβάστε αυτό το σύνολο δεδομένων για το evaluation
https://github.com/slp-ntua/slp-labs/blob/master/lab1/data/spell_test.txt
β) Η βέλτιστη διόρθωση προβλέπεται με βάση τον αλγόριθμο ελαχίστων μονοπατιών στο γράφο του μετατροπέα του Βήματος 6.

Χρησιμοποιείστε το μετατροπέα για να διορθώσετε κάποιες από τις λέξεις του test set. Σε αυτό το σημείο επιλέξτε τις 20 πρώτες λέξεις και σχολιάστε το αποτέλεσμα. Μπορείτε να χρησιμοποιήσετε σαν βάση αυτόν τον κώδικα: https://github.com/slp-ntua/slp-labs/blob/master/lab1/scripts/predict.sh
Τι κάνει αυτό το script; Σχολιάστε τις πράξεις που πραγματοποιούνται στα fsts, τις ενδιάμεσες εξόδους και πώς αυτή η αλληλουχία πράξεων μας παράγει τη διορθωμένη λέξη.

ΤΕΛΟΣ ΠΡΟΠΑΡΑΣΚΕΥΗΣ

ΜΕΡΟΣ 1

Σε αυτό το μέρος της άσκησης θα προσπαθήσουμε να βελτιώσουμε τον ορθογράφο που δημιουργήσαμε στην προπαρασκευή εισάγοντας γλωσσική πληροφορία.

Βήμα 8: Υπολογισμός κόστους των edits

Σε αυτό το βήμα θα αναθέσουμε διαφορετικό κόστος για κάθε edit operation του Levenshtein transducer. Για να υπολογίσουμε όμως τα διαφορετικά αυτά κόστη θα χρειαστούμε κάποια επιπρόσθετη πληροφορία. Για αυτό το λόγο σας δίνεται το *corpus* *https://github.com/slp-ntua/slp-labs/blob/master/lab1/data/wiki.txt* που περιέχει συχνά ορθογραφικά λάθη από άρθρα της Wikipedia. Το corpus έχει την παρακάτω μορφή, όπου κάθε γραμμή περιέχει μια misspelled λέξη και τη διόρθωσή της.

```
abandonned      abandoned
aberation       aberration
abilityes       abilities
abilties        abilities
abilty          ability
```

α) Χρησιμοποιήστε ένα μόνο παράδειγμα (abandonned->abandoned). Δημιουργείστε ένα αποδοχέα **M** για την ανορθόγραφη λέξη (abandonned) και ένα μετατροπέα **N** για τη διόρθωμένη (abandoned).
β) Δημιουργείστε τη σύνθεση **MLN** συνθέτοντας το **M** με τον transducer **L** και το αποτέλεσμα με το **N.**
γ) Εκτελέστε την fstshortestpath ακολουθούμενη από την fstprint όπως φαίνεται παρακάτω

```
fstshortestpath MLN.fst |
   fstprint --isymbols=chars.syms --osymbols=chars.syms  --show_weight_one
```

Το αποτέλεσμα σας δείχνει το μονοπάτι (μετασχηματισμούς) που ακολουθούνται για να μετατραπεί η ανορθόγραφη λέξη στη σωστή.

```
11      10      a       a       0
0       0
1       0       <eps>   <eps>   0
2       1       d       d       0
3       2       e       e       0
4       3       n       n       0
5       4       n       <eps>   3
6       5       o       o       0
7       6       d       d       0
8       7       n       n       0
9       8       a       a       0
10      9       b       b       0
```

Η τελευταία στήλη δείχνει το κόστος κάθε edit. Μπορείτε να αγνοήσετε τις γραμμές που έχουν μηδενικό κόστος και να αποθηκεύσετε μονο το edit με χρήση των εντολών grep και cut.
Σας δίνεται το scripts/word_edits.sh που πραγματοποιεί την παραπάνω διαδικασία για ένα ζευγάρι λέξεων.

```
❯ bash scripts/word_edits.sh abandonned abandoned
n       <eps>
# source target
```

Διαβάστε και σχολιάστε τη λειτουργία αυτού του script και εκτελέστε το για μερικά ζευγάρια εισόδου.
δ) Γράψτε ένα python script που θα εκτελεί τη διαδικασία του 8γ για κάθε γραμμή του wiki.txt ώστε να παράξετε όλα τα edits. Αποθηκεύστε τα edits σε ένα αρχείο.

ε) Εξάγετε την συχνότητα κάθε edit και αποθηκεύστε τη σε ένα λεξικό με key την διπλέτα (source, target) και value την συχνότητα εμφάνισής του.
στ) Επαναλάβετε το Βήμα 4, χρησιμοποιώντας σαν βάρη των ακμών τον αρνητικό λογάριθμο της συχνότητας του αντίστοιχου edit. Αν αυτό το edit δεν έχει εμφανιστεί στο corpus μπορείτε να βάλετε ένα πολύ μεγάλο αριθμό ως βάρος (άπειρο κόστος). Αυτός είναι ο transducer **E.**
ζ) Επαναλάβετε τα Βήματα 6 και 7 με χρήση του **E** αντί για τον **L** για να κατασκευάσετε και να δοκιμάσετε τον ορθογράφο **EV.**

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

Σε αυτό το βήμα θα εισάγουμε στο μοντέλο τις συχνότητες εμφάνισης λέξεων, ώστε ο ορθογράφος να προτείνει πιο πιθανές λέξεις στις διορθώσεις του.
α) Χρησιμοποιείστε το λεξιλόγιο με τις συχνότητες των λέξεων που δημιουργήσατε στο Βήμα 2.
β) Κατασκευάστε 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.
γ) Κατασκευάστε τον ορθογράφο **LVW** πραγματοποιώντας τη σύνθεση του Levenshtein transducer **L** με τον acceptor του λεξιλογίου **V** και το γλωσσικό μοντέλο **W.** Μην ξεχάσετε να χρησιμοποιήσετε τις fstrmepsilon, fstdeterminize, fstminimize και την fstarcsort όπου χρειάζεται.
δ) Κατασκευάστε τον ορθογράφο **EVW** πραγματοποιώντας τη σύνθεση του Levenshtein transducer **E** με τον acceptor του λεξιλογίου **V** και το γλωσσικό μοντέλο **W.**
ε) Αξιολογείστε τον ορθογράφο **LVW** με τη διαδικασία του Βήματος 8.
στ) Συγκρίνετε τα αποτελέσματα του **LVW** με τον **LV.** Ποια είναι η διόρθωση που προτείνει ο κάθε μετατροπέας αν στην είσοδο εισάγουμε τα tokens “cwt” και “cit”; Γιατί;
ζ) Χρησιμοποιήστε την fstdraw για να σχεδιάσετε τον **W** και τον **VW.** Μπορείτε να χρησιμοποιήσετε ένα μικρό υποσύνολο των λέξεων για εύκολη οπτικοποίηση.

Βήμα 10: Αξιολόγηση των ορθογράφων

Σας δίνεται αυτό το σύνολο δεδομένων για το evaluation https://github.com/slp-ntua/slp-labs/blob/master/lab1/data/spell_test.txt, καθώς και το script https://github.com/slp-ntua/slp-labs/blob/master/lab1/scripts/run_evaluation.py.
Χρησιμοποιείστε το run_evaluation.py για να συγκρίνετε τα αποτελέσματα των ορθογράφων **LV, LVW, EV, EVW** και να υπολογίσετε την ακρίβεια κάθε ορθογράφου στο test set. Σχολιάστε τα αποτελέσματα και συγκρίνετε τη συμπεριφορά των διαφορετικών ορθογράφων.

Βήμα 11: (Bonus) Βελτιώσεις του ορθογράφου

α) Στο Βήμα 8στ χρησιμοποιήσαμε άπειρο κόστος για τα edits τα οποία δεν εμφανίζονται στο Wikipedia corpus. Μια πιθανή βελτίωση του μοντέλου είναι να πραγματοποιήσουμε smoothing των πιθανοτήτων για τα edits που δεν εμφανίζονται. Σε αυτό το Βήμα καλείστε να υλοποιήσετε ένα απλό Add-1 smoothing (Jurafsky 3.4.1). Την μάζα πιθανότητας που περισσεύει μπορείτε να την ισομοιράσετε στα edits με μηδενική συχνότητα εμφάνισης.
Κατασκευάστε και αξιολογείστε τον ορθογράφο **EV** χρησιμοποιώντας additive smoothing για τα edits με μηδενική πιθανότητα εμφάνισης. Για την αξιολόγηση χρησιμοποιείστε όλο το corpus όπως στο Βήμα 10.
β) Το μέγεθος και η ποιότητα του λεξικού, καθώς και το domain από το οποίο προκύπτει, παίζουν μεγάλο ρόλο στην απόδοση του ορθογράφου. Δοκιμάστε λεξικά διαφορετικού μεγέθους από διαφορετικά / μεγαλύτερα corpora. Μπορείτε να χρησιμοποιήσετε εξωτερικές πηγές για την απόκτηση αυτών των λεξικών όπως τα παρακάτω λεξικά που προκύπτει από υποτίτλους στο opensubtitles (https://github.com/hermitdave/FrequencyWords/blob/master/content/2016/en/) ή τα λεξικά που παρέχονται από τη wikipedia https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists. Σχολιάστε την επίδραση του μεγέθους, της ποιότητας και του domain του λεξικού στον ορθογράφο.
γ) Αναφέρετε και υλοποιείστε άλλες πιθανές βελτιώσεις για τον ορθογράφο.

ΜΕΡΟΣ 2: Εξοικείωση με το W2V

Στο πρώτο μέρος της άσκησης ασχοληθήκαμε κυρίως με συντακτικά μοντέλα για την κατασκευή ενός ορθογράφου. Εδώ θα ασχοληθούμε με τη χρήση λεξικών αναπαραστάσεων και συγκεκριμένα με το Word2Vec μοντέλο. Αρχικά θα εκπαιδεύσουμε ένα δικό μας μοντέλο πάνω στα Gutenberg κείμενα. Στη συνέχεια θα το συγκρίνουμε με ένα προεκπαιδευμένο μοντέλο της Google. Η σύγκριση θα γίνει σε πρώτη φάση διαισθητικά με βάση τη δική σας (ανθρώπινη) αντίληψη της γλώσσας.
Στη συνέχεια θα κατασκευάσουμε έναν ταξινομητή συναισθήματος. Για την κατασκευή ενός τέτοιου μοντέλου θα χρησιμοποιήσουμε δεδομένα από σχόλια για ταινίες στην ιστοσελίδα IMDB. Σκοπός είναι να ταξινομήσουμε τις κριτικές σε θετικές και αρνητικές ως προς το συναίσθημα. Τα μοντέλα που θα κατασκευάσετε θα δέχονται σαν είσοδο τις γλωσσικές αναπαραστάσεις είτε του δικού σας μοντέλου είτε του προεκπαιδευμένου. Το πείραμα αυτό αποτελεί μια εμπειρική σύγκριση των δύο αναπαραστάσεων.

Βήμα 12: Εξαγωγή αναπαραστάσεων word2vec

Τα word embeddings μπορούν να χρησιμοποιηθούν για να αναπαραστήσουμε λέξεις και προτάσεις σε machine learning pipelines. Για αυτό το σκοπό γίνεται χρήση προεκπαιδευμένων embeddings. Τα word embeddings είναι πυκνές (dense) αναπαραστάσεις που κωδικοποιούν σημασιολογικά χαρακτηριστικά μιας λέξης με βάση την υπόθεση ότι λέξεις με παρόμοιο νόημα εμφανίζονται σε παρόμοια συγκείμενα (contexts).
Σε αυτό το βήμα θα εστιάσουμε στα word2vec embeddings. Αυτά τα embeddings προκύπτουν από ένα νευρωνικό δίκτυο με ένα layer το οποίο καλείται να προβλέψει μια λέξη με βάση το context της (παράθυρο 3-5 λέξεων γύρω από αυτή). Αυτό ονομάζεται CBOW μοντέλο. Εναλλακτικά το δίκτυο καλείται να προβλέψει το context με βάση τη λέξη (skip-gram μοντέλο).
Μπορείτε να διαβάσετε το paper και το παρακάτω άρθρο.

Μπορείτε να βασιστείτε σε αυτόν τον κώδικα για την εκπαίδευση του word2vec μοντελου
https://github.com/slp-ntua/slp-labs/blob/master/lab1/scripts/w2v_train.py

α) Διαβάστε το corpus που φτιάξατε στο Βήμα 1 σε μια λίστα από tokenized προτάσεις.
β) Χρησιμοποιείστε την κλάση Word2Vec του gensim για να εκπαιδεύσετε 100 διαστατα word2vec embeddings με βάση τις προτάσεις του βηματος 9α. Χρησιμοποιείστε window=5 και 1000 εποχές. (παραδείγματα: https://radimrehurek.com/gensim/models/word2vec.html)
γ) Επιλέξτε τις λέξεις “bible”, “book”, “bank”, “water” από το λεξικό και βρείτε τις σημασιολογικά κοντινότερες τους (χρησιμοποιώντας cosine similarity).

1) Είναι τα αποτελέσματα τόσο ποιοτικά όσο περιμένατε;
2) Βελτιώνονται αν αλλάξετε το μέγεθος του παραθύρου context / τον αριθμό εποχών;
3) Γιατί;
4) Τι θα κάνατε για να βελτιώσετε τα αποτελέσματα;

δ) 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*», Πχ «Η Αθήνα είναι για την Ελλάδα ότι το Παρίσι για τη Γαλλία».
Στα word2vec embeddings μπορούμε να το εκφράσουμε αυτό σαν μια αλγεβρική πράξη διανυσμάτων
v = w2v(“Ελλάδα”) – w2v(“Αθήνα”) + w2v(“Παρίσι”)
Με αυτή την πράξη περιμένουμε η κοντινότερη λέξη για το διάνυσμα **v** να είναι η λέξη Παρίσι.
Σε αυτό το ερώτημα καλείστε να βρείτε τη λέξη που προκύπτει χρησιμοποιώντας τις τριπλέτες λέξεων (“grils”, “queen”, “kings”), (“good”, “taller”, “tall”) και (“france”, “paris”, “london”) χρησιμοποιώντας την αναλογία στα word2vec embeddings που εκπαιδεύσατε.
ε) Κατεβάστε και φορτώστε τα προεκπαιδευμένα GoogleNews vectors.
https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
Hint:

```
model = KeyedVectors.load_word2vec_format('./GoogleNewsvectorsnegative300.bin', binary=True, limit=NUM_W2V_TO_LOAD)
```

Χρησιμοποιείστε την παράμετρο limit για να μη γεμίσετε τη μνήμη.
στ) Επαναλάβετε το Βήμα 12γ για τα GoogleNews vectors. Συγκρίνετε τα αποτελέσματα.
ζ) Επαναλάβετε το Βήμα 12δ για τα GoogleNews vectors. Συγκρίνετε τα τα αποτελέσματα.

Βήμα 13: Οπτικοποίηση των word embeddings

Οι αναπαραστάσεις word2vec κωδικοποιούν τα σημασιολογικά χαρακτηριστικά των λέξεων σε ένα πολυδιάστατο διανυσματικό χώρο. Συχνά η οπτικοποίηση αυτών των χώρων αποτελεί ένα χρήσιμο εργαλείο και παρέχει διαίσθηση για τον τρόπο με τον οποίο κωδικοποιούνται οι έννοιες. Για την οπτικοποίηση είναι απαραίτητη η μείωση της διάστασης του σημασιολογικού χώρου σε δύο η τρεις διαστάσεις με χρήση κάποιου αλγορίθμου μείωσης διαστατικότητας. Ένας αλγόριθμος που δουλεύει καλά στην πράξη για αυτό το σκοπό είναι η T-SNE (περισσότερα εδώ).
α) Πειραματιστείτε με το ακόλουθο εργαλείο. Παρουσιάστε ορισμένα παραδείγματα και σχολιάστε τα visualizations
β) Αποθηκεύστε τα word embeddings από το μοντέλο που εκπαιδεύσατε στο Βήμα 12 σε ένα tab separated αρχείο με όνομα embeddings.tsv και τις αντίστοιχες λέξεις σε ένα αρχείο με όνομα metadata.tsv, σε αναλογία με το παρακάτω παράδειγμα:

```
# embeddings.tsv
0       0       1
0       1       0
1       0       0


# metadata.tsv
word1
word2
word3
```

γ) Φορτώστε τα .tsv αρχεία του ερωτήματος β) στο Embedding Projector και πειραματιστείτε τόσο με τις μεθόδους μείωσης διαστατικότητας όσο και με παραδείγματα τα οποία είναι διαισθητικά σωστά/λάθος. Παρουσιάστε τα σχετικά αποτελέσματα.

Βήμα 14: Ανάλυση συναισθήματος με word2vec embeddings

Μια πρόταση μπορεί να αναπαρασταθεί ως ο μέσος όρος των w2v διανυσμάτων κάθε λέξης που περιέχει (Neural Bag of Words). Το NBOW είναι μια διανυσματική αναπαράσταση μιας πρότασης που προκύπτει από το μέσο όρο των embeddings των λέξεων που την αποτελούν. Σε αυτό το Βήμα θα κάνουμε χρήση αυτών των αναπαραστάσεων για την ανάλυση συναισθήματος σε movie reviews.

α) Κατεβάστε τα δεδομένα από το παρακάτω λινκ
http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
β) Διαβάστε και προεπεξεργαστείτε τα δεδομένα. Σας δίνεται για διευκόλυνση ο κώδικας ανάγνωσης και κάποιες απλές συναρτήσεις προεπεξεργασίας
https://github.com/slp-ntua/slp-labs/blob/master/lab1/scripts/w2v_sentiment_analysis.py
γ) Χρησιμοποιήστε τα embeddings που εκπαιδεύσατε για την κατασκευή Neural Bag of Words αναπαραστάσεων για κάθε σχόλιο στο corpus και εκπαιδεύστε ένα Logistic Regression μοντέλο για ταξινόμηση. Σχολιάστε τα αποτελέσματα. Εξηγείστε πώς θα μπορούσατε να βελτιώσετε την επίδοση του μοντέλου.
Hint: Για τα παρακάτω ερωτήματα θεωρείστε την αναπαράσταση μιας OOV λέξης το μηδενικό διάνυσμα
δ) Επαναλάβετε το Βήμα 14γ με τα Google News. Συγκρίνετε και σχολιάστε τα αποτελέσματα.

ΠΑΡΑΔΟΤΕΑ

Συγκεντρώστε **σε ένα .zip αρχείο** το οποίο θα περιέχει 1) το φάκελο lab1 από το github με συμπληρωμένες τις υλοποιήσεις σας 2) ένα αρχείο .pdf με την αναφορά σας. Μην υποβάλετε αρχεία με fsts μέσα στο .zip. Εαν θέλετε να ελέγξουμε κάποιο μοντέλο σας στείλτε το μέσω drive link. Το .zip να αποσταλεί εντός της προκαθορισμένης ημερομηνίας στο helios.