1η εργαστηριακή άσκηση στο μάθημα "Επεξεργασία φωνής και Φυσικής γλώσσας"

Πέτρος Τζάθας 03115050
Παναγιώτης Παπαντωνάκης 03115012

1ο Μέρος

Βήμα 10: Εξαγωγή στατιστικών

Οι πιθανότητες εφμάνισης μιας λέξης ή ενός χαρακτήρα υπολογίζονται από του παρακάτω τύπους:

$$P(word) = \frac{count(word)}{number\ of\ words \in corpus} \qquad P(letter) = \frac{count(letter)}{number\ of\ letters \in corpus}$$

Τα 2 λεξικά τα κατασκευάζει η συνάρτηση:

create_probability_dictionary_alphabet(tokenizedList)

του αρχείου preprocess.py, η οποία παίρνει στην είσοδό της μια λίστα με tokens. (Τα values των λεξικών που επιστρέφει είναι ο αρνητικός λογάριθμος των προηγουμένως υπολογισμένων σχετικών συχνοτήτων – πιθανοτήτων)

Βήμα 11: Κατασκευή μετατροπέων FST

Η δημιουργία των μετατροπέων που υλοποιούν την απόσταση Levenshtein έγινε όπως και στην προπαρασκευή του εργαστηρίου με την μόνο διαφορά ότι τα βάρη στις ακμές δίνονται από το μέσο όρο των values( ) του κάθε λεξικού. Η συνάρτηση που δημιουργεί τα αρχεία που δέχεται ως είσοδο η fstcompile είναι η:

create_levenshtein_fst (alphabet, delW = 1, insW = 1, subW = 1, name = 'fst')

alphabet: iterable με τους χαρακτήρες του corpus
delW, insW, subW: delition, insertion, substitution weight
name: χρησιμοποιείται για το αρχείο που δημιουργείται

του αρχείου create_fst.py

Θα μπορούσαμε να υπολογίσουμε τα βάρη του Levenshtein FST με πιο έξυπνο τρόπο αν είχαμε στη διαθεσή μας ζεύγη λέξεων στα οποία η μία λέξη είναι η λέξη που έχει γραφεί λανθασμένα και η άλλη είναι η λέξη που θα έπρεπε να είχε γραφεί. Υπολογίζοντας το minimum edit distance μεταξύ των δύο λέξεων για κάθε ζεύγος μπορούμε να υπολογίζουμε τις πιθανότητες διαγραφής, εισαγωγής ενός χαρακτήρα ή της αντικατάστασης του με έναν άλλο και από αυτές τα βάρη. Έτσι τα βάρη θα ήταν διαφορετικά για κάθε χαρακτήρα. (π.χ. είναι πιο πιθανό να ο χαρακτήρας *η* να έχει λανθασμένα γραφεί ως *ι* παρά ως *μ*). Βέβαια και σε αυτήν την περίπτωση θα έπρεπε να μεριμνήσουμε ώστε να υπάρχει ισορροπία μεταξύ των βαρών στο Levenshtein FST και στον αποδοχέα του λεξικού, ωστέ να μην επισκιάζεται η συμβολή του ενός από το άλλο.

Βήμα 12: Κατασκευή γλωσσικών μοντέλων

Για την κατασκευή των λεξικών (σε μορφή που θα ληφθεί ως είσοδος από την fstcompile) χρησιμοποιείται η συνάρτηση:

create_dictionary_fst (dictionary, name = 'fst')

dictionary: λεξικό με keys τις λέξεις και values τα βάρη
name: χρησιμοποιείται για το αρχείο που δημιουργείται

του αρχείου create_fst.py

Η συνάρτηση δημιουργεί έναν acceptor που έχει μια αρχική κατάσταση στην οποία συνδέονται αλυσίδες από καταστάσεις, μία για κάθε λέξη, με την τελευταία κατάσταση στην αλυσίδα να είναι τελική. Από όλες τις ακμές, μόνο η πρώτη έχει βάρος το οποίο δίνεται από το λεξικό εισόδου.

Η συνάρτηση χρησιμοποιείται για την παραγωγή και των δύο λεξικών. Για το word level μοντέλο τα βάρη είναι αυτά που υπολογίστηκαν στο βήμα 10. Για το character level μοντέλο το βάρος μια λέξης είναι το άθροισμα των βαρών των χαρακτήρων που την συναποτελούν. Εφόσον η αλυσίδες καταστάσεων που ξεκινούν από την αρχική δεν διακλαδώνονται στο ενδιάμεσο δεν έχει σημασία αν το βάρος κατανέμεται σε όλες τις ακμές ή αν βρίσκεται συγκεντρωμένο σε μία.

Η κατασκευή των απαραίτητων αρχείων (Levenshtein FSTs και dictionary acceptors) γίνειται με την εντολή:

./compile_fst.sh -p FILENAME

FILENAME: το αρχείο από το οποίο θα κατασκευαστούν τα μοντέλα
(Η επιλογή -p είναι προαιρετική, σε περίπτωση που υπάρχουν ήδη τα αρχεία που

παράγονται από το dict_alph_fst.py υπάρχουν ήδη)

Βήμα 13: Κατασκευή ορθογράφων

Η διόρθωση μιας λέξης γίνεται με την εντολή:

./spell_check.sh [-w || -l] WORD

WORD: η λέξη που πρόκειται να διορθωθεί
Ανάλογα με την επιλογή (-w ή -l) χρησιμοποιείται ο word level ή character level ορθογράφος.

Το αρχείο spell_check.sh χρησιμοποιεί το input_fst.py για την παραγωγή του acceptor της λέξης εισόδου.

Οι ορθογράφοι βρίσκουν την λέξη του λεξικού που βρίσκεται πιο κοντά στην λέξη-είσοδο αλλά είναι και πιθανότερη βάσει του γλωσσικού μοντέλου. Για το word level μοντέλο πιθανότερη είναι η λέξη που εμφανίζεται περισσότερες φορές στο corpus. Για το character level μοντέλο πιθανότερη είναι μια λέξη που αποτελείται από συχνότερους χαρακτήρες. Ο τρόπος που υπολογίζεται το βάρος μιας λέξης σε αυτό το μοντέλο (άθροισμα των βαρών για κάθε χαρακτήρα) έχει ως "παρενέργεια" οι μικρότερες λέξεις να θεωρούνται γενικά πιθανοτερες.

Παραδείγματα διορθώσεων:

./spell_check.sh -w cit
it

./spell_check.sh -l cit
it

./spell_check.sh -w failes
failed

./spell_check.sh -l failes
files

Βήμα 14: Αξιολόγηση των ορθογράφων

Η αξιολόγηση των ορθογράφων γίνεται με το αρχείο evaluator.py το οποίο διαβάζει από το αρχείο spell_checker_test_set.txt τις ανορθόγραφες λέξεις και τις διορθώνει και με τα δύο μοντέλα.

Για το word level:

170 / 270 = 62.96%

Για το character level:

152 / 270 = 56.30%

Το word level μοντέλο φαίνεται να τα πηγαίνει καλύτερα, ίσως επειδή λειτουργεί βάσει πληροφορίας που αφορά την λέξη στο σύνολό της, σε αντίθεση με το character level μοντέλο που λειτουργεί σε επίπεδο χαρακτήρα.

2ο Μέρος

Βήμα 17: Κατασκευή BOW αναπαραστάσεων και ταξινόμηση

Τα βάρος TF αφορά την συχνότητα χρήσης μια λέξης σε ένα κείμενο και άρα πόσο σημαντική είναι για το εν λόγω κείμενο. Το βάρος IDF (αντίστροφη συχνότητα εμφάνισης σε κείμενο) είναι δείκτης της "εξιδείκευσης" αυτής της λέξης, δηλαδή του πόσο πληροφορία μας δίνει η χρήση της σε ένα κείμενο. Για παράδειγμα "λειτουργικές" λέξεις, όπως τα άρθρα, είναι αναμενόμενο να χρησιμοποιούνται συχνά σε ένα κείμενο (συνεπώς το βάρος TF θα είναι μεγάλο) , αλλά δεν μας παρέχουν κάποια πληροφορία. Το βάρος IDF τέτοιων λέξεων θα είναι μικρό, διότι θα εμφανίζονται σε πολλά από τα κείμενα του corpus, οπότε θα αντισταθμιστεί η συμβολή του TF. Σε αντίθεση κάποιο επίθετο που μας προσφέρει πολλή πληροφορία σχετικά με την άποψη του συγγραφέα (το οποίο μας ενδιαφέρει στην άσκηση), αν και ενδέχεται να χρησιμοποιείται μόνο μια φορά στο κείμενο (χαμηλό TF), θα είναι και πιο σπάνια στο corpus (υψηλό IDF).

Βήμα 18: Χρήση Word2Vec αναπαραστάσεων για ταξινόμηση

Η δημιουργία του μοντέλου word2vec έγινε με το αρχείο word2vec.py

Ακολουθούν αποτελέσματα για διάφορες τιμές του αριθμού δειγμάτων. Ο αριθμός αναφέρεται στον αριθμό σχολίων κάθε είδους (positive training, negative training, positive testing, negative testing), δηλαδή ο συνολικό αριθμός σχολίων που διαβάζονται είναι τετραπλάσιο τού αριθμού αυτού.

Για 2000:

```
CountVectorizer:
        training error: 0.0
        test error: 0.15900000000000003

TfidfVectorizer:
        training error: 0.03974999999999995
        test error: 0.14349999999999996


Out of vocabulary words percentage for our word2vec: 0.8253273754111586

Our word2vec:
        training error: 0.25525
        test error: 0.269


Out of vocabulary words percentage for GoogleNes word2vec: 0.21150623719977657

GoogleNews word2vec:
        training error: 0.17049999999999998
        test error: 0.16749999999999998

GoogleNews word2vec with tfidf:
        training error: 0.12649999999999995
        test error: 0.15600000000000003
```

Για 2500:

```
CountVectorizer:
        training error: 0.00019999999999997797
        test error: 0.15680000000000005

TfidfVectorizer:
        training error: 0.04279999999999995
        test error: 0.13939999999999997


Out of vocabulary words percentage for our word2vec: 0.8404187198975472

Our word2vec:
        training error: 0.25680000000000003
        test error: 0.27480000000000004


Out of vocabulary words percentage for GoogleNes word2vec: 0.22831927392187978

GoogleNews word2vec:
        training error: 0.17020000000000002
        test error: 0.1684

GoogleNews word2vec with tfidf:
        training error: 0.134
        test error: 0.15359999999999996
```

Για 3000:

```
CountVectorizer:
        training error: 0.0001666666666666483
        test error: 0.15549999999999997

TfidfVectorizer:
        training error: 0.04366666666666663
        test error: 0.139


Out of vocabulary words percentage for our word2vec: 0.8518310721207794

Our word2vec:
        training error: 0.262
        test error: 0.27849999999999997


Out of vocabulary words percentage for GoogleNes word2vec: 0.24441497500765072

GoogleNews word2vec:
        training error: 0.16500000000000004
        test error: 0.16616666666666668

GoogleNews word2vec with tfidf:
        training error: 0.1343333333333333
        test error: 0.15633333333333332
```

Για 3500:

```
CountVectorizer:
        training error: 0.00028571428571433355
        test error: 0.15314285714285714

TfidfVectorizer:
        training error: 0.04585714285714282
        test error: 0.14


Out of vocabulary words percentage for our word2vec: 0.8610033128253668

Our word2vec:
        training error: 0.2577142857142857
        test error: 0.27714285714285714


Out of vocabulary words percentage for GoogleNes word2vec: 0.2604827259820161

GoogleNews word2vec:
        training error: 0.16400000000000003
        test error: 0.16714285714285715

GoogleNews word2vec with tfidf:
        training error: 0.13671428571428568
        test error: 0.15400000000000003
```

Για 4000:

```
CountVectorizer:
        training error: 0.00024999999999997247
        test error: 0.148625

TfidfVectorizer:
        training error: 0.04812499999999997
        test error: 0.13587499999999997


Out of vocabulary words percentage for our word2vec: 0.8673727580107033

Our word2vec:
        training error: 0.259625
        test error: 0.27449999999999997


Out of vocabulary words percentage for GoogleNes word2vec: 0.2716585682617166

GoogleNews word2vec:
        training error: 0.16137500000000005
        test error: 0.16525

GoogleNews word2vec with tfidf:
        training error: 0.13549999999999995
        test error: 0.15275000000000005
```

Για 4500:

```
CountVectorizer:
        training error: 0.00022222222222223476
        test error: 0.14288888888888884

TfidfVectorizer:
        training error: 0.04988888888888887
        test error: 0.13277777777777777


Out of vocabulary words percentage for our word2vec: 0.8733044180805375

Our word2vec:
        training error: 0.2604444444444445
        test error: 0.2664444444444445


Out of vocabulary words percentage for GoogleNes word2vec: 0.28273164094059616

GoogleNews word2vec:
        training error: 0.15822222222222226
        test error: 0.1617777777777778

GoogleNews word2vec with tfidf:
        training error: 0.13722222222222225
        test error: 0.14988888888888885
```

Για 5000:

```
CountVectorizer:
        training error: 0.00029999999999996696
        test error: 0.14329999999999998

TfidfVectorizer:
        training error: 0.05379999999999996
        test error: 0.1281


Out of vocabulary words percentage for our word2vec: 0.877780503761858

Our word2vec:
        training error: 0.25839999999999996
        test error: 0.26649999999999996


Out of vocabulary words percentage for GoogleNes word2vec: 0.29093065096499837

GoogleNews word2vec:
        training error: 0.15810000000000002
        test error: 0.15900000000000003

GoogleNews word2vec with tfidf:
        training error: 0.13829999999999998
        test error: 0.1502
```

Σχολιασμός αποτελεσμάτων:

Όσον αφορά τον CountVectorizer και τον TfidfVectorizer:

Το σφάλμα στα δεδομένα εκπαίδευσης είναι και για τα δύο πολύ μικρό. Αυτό μάλλον οφείλεται στον πλεονασμό χαρακτηριστικών που καθιστά τις κλάσεις γραμμικά διαχωρίσιμες. Μάλιστα για τον CountVectorizer το σφάλμα είναι ακόμα πιο μικρό διότι η συμβολή των “λειτουργικών” λέξεων στο διάνυσμα χαρακτηριστικών είναι πιο μεγάλη, ενώ στον TfidfVectorizer τέτοια χαρακτηριστικά δεν έχουν μεγάλη συμβολή και έτσι γίνεται ενός τύπου feature selection. Λόγω των βαρών tfidf και του πλεονασμού χαρακτηριστικών ο TfidfVectorizer έχει συνολικά την καλύτερη επίδοση στο test set.

Όσον αφορά τα word2vec:

Τα διανύσματα που εκπαιδεύσαμε εμείς έχουν την χειρότερη απόδοση συνολικά, το οποίο είναι λογικό, διότι το ποσοστό λέξεων εκτός λεξικού είναι περίπου 85%.

Τα προεκπαιδευμένα διανύσματα της Google τα πηγαίνουν πολύ καλύτερα και ιδαιτέρα όταν βεβαρυνθούν με tfidf. Μάλιστα η συμπεριφορά τους φαίνεται να ακολουθεί την επιθυμητή συμπεριφορά των validation curves που παρουσιάστηκε

στο εισαγωγικό εργαστήριο, δηλαδή το training error και το test error φαίνεται να συγκλίνουν, το μεν προς τα πάνω, το δε προς τα κάτω, σε μια τιμή περίπου 15% και συνεπώς φαίνεται το πιο αξιόπιστο. Αν βέβαια είχαμε ένα μηχάνημα που μπορούσε να χειριστεί περισσότερα δεδομένα, ίσως και ο TfidfVectorizer να φερόταν ανάλογα, άλλωστε στα παραπάνω φαίνεται ότι το training error αυξάνεται και το test error μειώνεται σταδιακά. Το πρόβλημα με την αναπράσταση που δίνει ο TfidfVectorizer είναι ότι αυξάνεται σε μέγεθος όταν αυξάνονται τα δεδομένα και συνεπώς η εκπαίδευση των μοντέλων γίνεται υπολογιστικά συνθετότερη, σε αντίθεση με τα διανύσματα word2vec που έχουν σταθερή διάσταση. Ο ρυθμός αύξησης της διάστασης του διανύσματος των χαρατηριστικών συναρτήσει του μεγέθους του corpus δίνεται από τον Heaps' Law, αφού ισούται με το μέγεθος του λεξικού.