

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



IONIAN UNIVERSITY

-- Πτυχιακή Εργασία --

Χρήση εργαλείων Μηχανικής Μάθησης (Machine Learning) και
Γλωσσικής Τεχνολογίας (Natural Language Processing) για
την απλοποίηση κειμένου (text simplification)

Αλέξανδρος Πουλής Π2019128

Επιβλέπων: – Κάτια - Λήδα Κερμανίδου –

September 25, 2024

Επιβλέπων(ουσα)

Κάτια - Λήδα Κερμανίδου, Αναπληρώτρια Καθηγήτρια,
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Κάτια - Λήδα Κερμανίδου, Αναπληρώτρια Καθηγήτρια,
Ιόνιο Πανεπιστήμιο

Θεμιστοκλής Έξαρχος, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Καναβός Ανδρέας, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Περίληψη

Το Text Simplification είναι ένας ανερχόμενος τομέας του Natural Language Processing(NLP). Στην εργασία αυτή θα ασχοληθούμε με τον τομέα αυτό και συγκεκριμένα με το Text Classification. Σαν τομέας το Text Classification περιέχει τα εξής βήματα: 1)την επιλογή ενός Corpus, 2) την προεπεξεργασία του Corpus αυτού, 3) την εξαγωγή χαρακτηριστικών από το Corpus, τα οποία δείχνουν την πολυπλοκότητα με διαφορετικούς τρόπους το καθένα, και τέλος 4) την εκπαίδευση Classifier πάνω στο Corpus για την καλύτερη πρόβλεψη της πολυπλοκότητας του. Όλα τα παραπάνω βήματα θα τα αναλύσουμε σε εργασίες οι οποίες ασχολούνται με τον ίδιο χώρο του NLP και θα τα πειραματιστούμε με αυτά. Τέλος, θα συγκρίνουμε τα αποτελέσματα μας με αυτά άλλων εργασιών σε αυτό το πεδίο του NLP.

Πρόλογος και Ευχαριστίες

Η παρούσα Πτυχιακή Εργασία εκπονήθηκε κατά το έτος 2023 - 2024 στα πλαίσια των προπτυχιακών σπουδών μου στο Τμήματος Πληροφορικής του Ιονίου Πανεπιστημίου.

Η εργασία πραγματοποιήθηκε υπό την επίβλεψη της κα. Κάτια - Λήδα Κερμανίδου, Αναπληρώτρια Καθηγήτρια του Τμήματος Πληροφορικής του Ιονίου Πανεπιστημίου, με συνεργάτες τον κ. Θεμιστοκλή Έξαρχο, επίσης Αναπληρωτή Καθηγητή του Τμήματος Πληροφορικής του Ιονίου Πανεπιστημίου και τον κ. Καναβό Ανδρέα, Αναπληρωτή Καθηγητή του Τμήματος Πληροφορικής του Ιονίου Πανεπιστημίου.

Αντικείμενο της εργασίας αποτελεί η "Χρήση εργαλείων Μηχανικής Μάθησης (Machine Learning) και Γλωσσικής Τεχνολογίας (Natural Language Processing) για την απλοποίηση κειμένου (text simplification)". Συγκεκριμένα ασχολούμαστε στο Text Classification, μέρος του τομέα του text classification.

Κέρκυρα, 25/9/2024

Πουλής Αλέξανδρος

Περιεχόμενα

A	Εισαγωγή	1
A.1	ΑΠΛΟΠΟΙΗΣΗ ΚΕΙΜΕΝΟΥ	1
A.2	ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΚΕΙΜΕΝΟΥ	2
A.3	ΠΑΡΑΓΩΓΗ CORPUS ΓΙΑ ΤΗΝ ΠΡΟΒΛΕΨΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑ ΚΕΙΜΕΝΟΥ	2
B	Μηχανική Μάθηση	3
B.1	ΕΦΑΡΜΟΓΗ ΤΗΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΓΙΑ ΤΗΝ ΠΡΟΒΛΕΨΗ ΠΟΛΥ- ΠΛΟΚΟΤΗΤΑ ΠΡΟΤΑΣΕΩΝ	3
C	Παρόμοιες Εργασίες	4
C.1	ONESTOPEGLISH CORPUS: A NEW CORPUS FOR AUTOMATIC READABILITY ASSESSMENT AND TEXT SIMPLIFICATION	4
C.2	TEXT SIMPLIFICATION TO SPECIFIC READABILITY LEVELS . . .	8
C.3	AUTOMATIC CLASSIFICATION OF TEXT COMPLEXITY	9
C.4	TEXT COMPLEXITY CLASSIFICATION BASED ON LINGUISTIC IN- FORMATION: APPLICATION TO INTELLIGENT TUTORING OF ESL	14
C.5	AUTOMATED ASSESSMENT OF LEARNER TEXT COMPLEXITY . . .	18
C.6	USING BROAD LINGUISTIC COMPLEXITY MODELING FOR CROSS- LINGUAL READABILITY ASSESSMENT	19
C.7	WHEN SHALLOW IS GOOD ENOUGH: AUTOMATIC ASSESSMENT OF CONCEPTUAL TEXT COMPLEXITY USING SHALLOW SEMAN- TIC FEATURES	20
D	Τλοποίηση	22

Περιεχόμενα	iv
D.1 ΕΠΙΛΟΓΗ CORPUS	22
D.2 ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ CORPUS	23
D.3 ΕΠΙΛΟΓΗ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ	24
E Αποτελέσματα	39
E.1 ΠΑΡΟΥΣΙΑΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ	39
E.2 ΣΥΓΚΡΙΣΗ ΜΕ ΠΑΡΟΜΟΙΕΣ ΕΡΓΑΣΙΕΣ	42
F Συμπεράσματα	44
F.1 CORPUS	44
F.2 ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ	44
F.3 ΑΛΓΟΡΙΘΜΟΙ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ	45
F.4 ΓΕΝΙΚΑ ΣΥΜΠΕΡΑΣΜΑΤΑ	45
Συντμήσεις	49
Γλωσσάρι Ξενικών Όρων	52
Κώδικας	54

Κατάλογος Πινάκων

<i>C.1 Παράδειγμα ζεύγος προτάσεων του Corpus OneStopEnglish [2].</i>	<i>5</i>
<i>C.2 Πίνακας παρουσίασης αποτελεσμάτων των χαρακτηριστικών που χρησιμοποιούνται στον τομέα του readability classification στην εργασία [2].</i>	<i>6</i>
<i>C.3 Πίνακας παρουσίασης αποτελεσμάτων με την χρήση του SMO Clasifier στην εργασία [2].</i>	<i>7</i>
<i>C.4 Πίνακας παρουσίασης αποτελεσμάτων στην εργασία [10]. Το xval δεν το ανέφεραν το τι είναι στην εργασία και με ψάξιμο υποθέτω το ότι είναι cross-validation.</i>	<i>8</i>
<i>C.5 Πίνακας παρουσίασης αποτελεσμάτων στην εργασία [10]. Το xval δεν το ανέφεραν το τι είναι στην εργασία και με ψάξιμο υποθέτω το ότι είναι cross-validation.</i>	<i>13</i>
<i>C.6 Πίνακας παρουσίασης αποτελεσμάτων στην εργασία [11].</i>	<i>18</i>
<i>C.7 Πίνακας παρουσίασης αποτελεσμάτων στην εργασία [13].</i>	<i>20</i>
<i>E.1 Αποτελέσματα των classifier για το PWKP Corpus.</i>	<i>40</i>
<i>E.2 Confusion Matrix του RF classifier για το PWKP Corpus.</i>	<i>40</i>
<i>E.3 Confusion Matrix του Decision Tree classifier για το PWKP Corpus.</i>	<i>40</i>
<i>E.4 Confusion Matrix του SVM για το PWKP Corpus.</i>	<i>40</i>
<i>E.5 Αποτελέσματα των classifier για το OneStopEnglish Corpus.</i>	<i>41</i>
<i>E.6 Confusion Matrix του RF classifier για το OneStopEnglish Corpus.</i>	<i>41</i>
<i>E.7 Confusion Matrix του Decision Tree classifier για το OneStopEnglish Corpus.</i>	<i>41</i>
<i>E.8 Confusion Matrix του SVM για το OneStopEnglish Corpus.</i>	<i>42</i>

Κεφάλαιο Α

Εισαγωγή

Tο Natural language processing (NLP) είναι κομμάτι ενός ανερχόμενου τομέα, ο οποίος μεγαλώνει με όλο και περισσότερους ερευνητές να τον μελετάνε κάθε μέρα [1]. Ο τομέας του NLP έχει αναπτυχθεί με την πάροδο του χρόνου και οι ερευνητές του αναπτύσσουν εργαλεία για πολλές και διάφορες χρήσεις, με μία από αυτές να είναι το Text Simplification. Ο στόχος του Text Simplification είναι η ανάπτυξη εργαλείων τα οποία απλοποιούνε ένα κείμενο ώστε άτομα με χαμηλότερο επίπεδο γνώσης της γλώσσα του κειμένου να μπορούν να κατανοήσουν το κείμενο με μεγαλύτερη ευκολία. Σε αυτήν την πτυχιακή εργασία μελετάται ένα μέρος του τομέα αυτού, συγκεκριμένα το Text Classification. Σκοπός του Text Classification είναι να βοηθάει με το να καθορίζει το ποιο κείμενο χρειάζεται απλοποίηση και ποιο κείμενο είναι απλοποιημένο αρκετά.

A.1 Απλοποίηση κειμένου

Το Text Simplification έχει ως κύριο στόχο, άτομα τα οποία δε μιλάνε τη γλώσσα του κειμένου σαν μητρική τους γλώσσα, καθώς συνήθως αυτοί είναι οι άνθρωποι που χρειάζονται ένα εργαλείο σαν αυτό περισσότερο. Ένα μεγάλο μέρος των ερευνών γίνεται πάνω στα Αγγλικά και για αυτόν τον λόγο πολλές εργασίες έχουν αναπτύξει έναν όρο για να αναφέρονται σε αυτή την ομάδα ατόμων ως ESL(English as Second Language) Learners [2]. Υπάρχουν και εργασίες οι οποίες ερευνούν εργαλεία και για άλλες γλώσσες όπως για παράδειγμα αυτή η εργασία η οποία ασχολήθηκε με τα Ισπανικά σαν γλώσσα του κειμένου [3], ο γενικός όρος που

αναγνωρίζετε από τους ερευνητές σε αυτή την περίπτωση είναι Second Language (L2) [4]. Το Text Simplification στα λόγια των συγγραφέων της παρακάτω εργασίας [5], μπορεί να περιέχει "τον χωρισμό του κειμένου σε λέξεων και την αναδιάταξη τους" αλλά και αντικατάσταση ή διαγραφή λέξεων για την παραγωγή μιας απλοποιημένης πρότασης.

A.2 Πολυπλοκότητα κειμένου

Η πολυπλοκότητα ενός κειμένου δεν είναι κάτι το οποίο είναι εύκολο να προβλέψουμε με τη χρήση αλγορίθμων Machine Learning (ML). Η πολυπλοκότητά σε ένα κείμενο έρχεται σε πολλούς και διάφορους παράγοντες. Μερικοί από του παράγοντες αυτούς είναι για παράδειγμα η χρήση πολύπλοκων λέξεων, ή το πόσο μεγάλη είναι μία πρόταση, και πολλοί ερευνητές προσπαθούν να βρουν τον καλύτερο συνδυασμό μετρικών για την πρόβλεψη της πολυπλοκότητας αυτής. Αυτό είναι ένα σύνθετο πρόβλημα, το οποίο ακόμα δεν έχει λυθεί με νέους τύπους και νέες μετρικές να ανακαλύπτονται μέρα με τη μέρα και με τις προβλέψεις να γίνονται όλο και πιο ακριβείς.

A.3 Παραγωγή Corpus για την πρόβλεψη πολυπλοκότητα κειμένου

Στον τομέα του Text Classification εκτός από την έρευνα νέων μετρικών για την πρόβλεψη πολυπλοκότητας υπάρχουν και άλλα εξίσου σημαντικά θέματα. Ένα από αυτά είναι η παραγωγή ενός αρκετά μεγάλου Corpus το οποίο είναι σωστά ταξινομημένο. Ένας μεγάλος παράγοντας ο οποίος προσθέτει στη δυσκολία αυτή είναι το ότι κάθε κείμενο πρέπει να ταξινομηθεί με το χέρι από άτομα με ειδική γνώση για την εγγύηση σωστής ταξινόμησής όπως αναφέρετε και εν ολίγοις εδώ [6]. Επίσης, υπάρχει και η δυσκολία συλλογής ενός αρκετά μεγάλου Corpus το οποίο να περιέχει ποιοτικό κείμενο για την εκπαίδευσή του ML κώδικα.

Κεφάλαιο B

Μηχανική Μάθηση

Tο ML σαν ιδέα, σύμφωνα με την παρακάτω εργασία [7], ξεκίνησε με τον Frank Rosenblatt από το Cornell University. Ο πρώτος αλγόριθμος του σύμφωνα με την ίδια εργασία ονομάστηκε "Perceptron", ένας αλγόριθμος ο οποίος αναγνώριζε γράμματα. Η βασική ιδέα του ML, την οποία ανέφερε και ο Simon το 1983 σύμφωνα με την εργασία [8], είναι η εκπαίδευσή ενός αλγόριθμου με δεδομένα τα οποία έχουμε συλλέξει ώστε ο Classifier να "μάθει" από αυτά και να μπορεί να προβλέψει αποτελέσμά [9].

B.1 Εφαρμογή της Μηχανικής Μάθησης για την πρόβλεψη πολυπλοκότητα προτάσεων

Στην εργασία αυτή χρησιμοποιούμε το ML κάνοντας εξαγωγή χαρακτηριστικών, σε μορφή αριθμών, για να "εκπαιδεύσουμε" στον Classifier. Ο Classifier αυτός μετά θα δημιουργήσει ένα στατιστικό μοντέλο και θα το προσαρμόσει ώστε με τα δεδομένα αυτά να έχει το καλύτερο δυνατό ποσοστό σωστών προβλέψεων.

Κεφάλαιο C

Παρόμοιες Εργασίες

Σ την συνέχει θα αναλύσουμε μερικές εργασίες στον τομέα του Text Classification. Στόχος του κεφαλαίου αυτού είναι να δείξουμε τη μεθοδολογία μαζί με τα αποτελέσματα κάθε εργασίας.

C.1 OneStopEnglish corpus: A new corpus for automatic readability assessment and text simplification

Η εργασία αυτή δημοσιεύτηκε το 2018, με συγγραφείς τους Sowmya Vajjala και Ivana Lucic [2]. Κύριος σκοπός της εργασίας αυτής είναι να αναδείξει ένα Corpus το οποίο δημιούργησαν οι συγγραφείς, για τη χρήση του στον τομέα του Text Classification και Simplification.

Το Corpus που δημιούργησαν είναι μία συλλογή από κείμενα δημοσιευμένα στην ιστοσελίδα onestopenglish.com, μία ιστοσελίδα στοχευμένη στους ESL learners που αναφέραμε και παραπάνω. Η ιστοσελίδα αυτή έχει μαθήματα στα οποία έχουν απλοποιηθεί κείμενα δημοσιευμένα σε εφημερίδες, όπως το "The Guardian", από ιδιόχους σε 3 διαφορετικά επίπεδα δυσκολίας, Elementary, Intermediate και Advanced. Η συλλογή περιέχει κείμενα που δημοσιεύτηκαν στην περίοδο 2013–2016 και το κείμενο με τη μεγαλύτερη δυσκολία είναι το κείμενο πιο κοντά στο πρωτότυπο το οποίο πήραν από τις εφημερίδες. Το Corpus περιέχει 2166 ζεύγη προτάσεων. Στη συνέχεια θα δείξουμε ένα παράδειγμα ζεύγη προτάσεων μέσω ενός σχήματος το οποίο πήραμε από αυτή την εργασία [2] μέσω ενός πίνακα C.1.

Πίνακας C.1: Παράδειγμα ζεύγος προτάσεων του *Corpus OneStopEnglish* [2].

Πρόταση	Επίπεδο δυσκολίας πρόταση
Amsterdam still looks liberal to tourists, who were recently assured by the Labour Mayor that the city's marijuana-selling coffee shops would stay open despite a new national law tackling drug tourism. But the Dutch capital may lose its reputation for tolerance over plans to dispatch nuisance neighbours to scum villages made from shipping containers	Advanced
To tourists, Amsterdam still seems very liberal. Recently the city's Mayor assured them that the city's marijuana-selling coffee shops would stay open despite a new national law to prevent drug tourism. But the Dutch capitals plans to send nuisance neighbours to scum villages made from shipping containers may damage its reputation for tolerance.	Intermediate
To tourists, Amsterdam still seems very liberal. Recently the city's Mayor told them that the coffee shops that sell marijuana would stay open, although there is a new national law to stop drug tourism. But the Dutch capital has a plan to send antisocial neighbours to scum villages made from shipping containers, and so maybe now people wont think it is a liberal city any more.	Elementary

Στη συνέχεια μελετούνται τα χαρακτηριστικά που θα χρησιμοποιηθούν για το Text Classification. Οι συγγραφείς αρχίζουν με τα χαρακτηριστικά τα οποία αναφέρουν ως "κοινά χαρακτηριστικά" για τον τομέα. Τα χαρακτηριστικά αυτά περιέχουν τον αριθμό λέξεων, το Flesch-Kincaid Grade Level, το TTR(Type-token ratio) και τον αριθμό ρηματικών, ονομαστικών και προθετικών φράσεων.

Ύστερα καθώς τα παραπάνω χαρακτηριστικά δεν είναι αρκετά μόνα τους για την πρόβλεψη της δυσκολίας χρησιμοποίησαν χαρακτηριστικά που είναι συχνά μεταχειριζόμενα από έρευνες στον τομέα του readability classification, όπως τα n-grams. Τα n-grams είναι μια σειρά από n γειτονικούς χαρακτήρες και συγκεκριμένα χρησιμοποίησαν τα Word n-grams, Pos n-grams(Part of Speech) και character n-grams. Επίσης, έκαναν χρήση ενός εργαλείου

εν ονόματι "Stanford parser" για να αποκτήσουν τους συντακτικούς κανόνες παραγωγής προτάσεων και τις σχέσεις εξάρτησης από το ίδιο εργαλείο.

Τα αποτελέσματα που ανέδειξαν χρησιμοποιώντας μόνο αυτά τα χαρακτηριστικά δείχνουν το ότι το χαρακτηριστικό που προβλέπει καλύτερα τα αποτελέσματα είναι το character n-grams με 77.25% ακρίβεια. Αυτό θα φανεί ποιο αναλυτικά στον παρακάτω πίνακα C.2 που δείχνει ακριβώς τις τιμές που παρουσίασαν στην εργασία τους [2].

Πίνακας C.2: Πίνακας παρουσίασης αποτελεσμάτων των χαρακτηριστικών που χρησιμοποιούνται στον τομέα του *readability classification* στην εργασία [2].

Χαρακτηριστικά	Ακρίβεια
Word n-grams	61.38%
POS n-grams	67.37%
Char n-grams	77.25%
Syntactic Production Rules	54.67%
Dependency Relations	27.16%

Συνεχίζοντας οι ερευνητές αποφάσισαν να κάνουν χρήση και χαρακτηριστικών που χρησιμοποιούνται στον τομέα του ARA(automatic readability assesment). Μερικά τα αναφέραμε και παραπάνω όπως το TTR. Τα χαρακτηριστικά σε αυτό το μέρος της εργασίας αναφέρονται ως προς την κατηγορία τους και όχι ονομαστικά οπότε έτσι θα τα αναφέρουμε και εμείς.

Αρχικά αναφέρονται στη χρήση "παραδοσιακών χαρακτηριστικών και τύπων" που έχουν χρησιμοποιηθεί ιστορικά στον τομέα του ARA. Μετά από αυτά χρησιμοποιούν χαρακτηριστικά που βασίζονται στο PoS tagging μαζί με το TTR και διαφορετικά χαρακτηριστικά λεξιλογικής παραλλαγής. Ύστερα αναφέρουν ότι χρησιμοποίησαν χαρακτηριστικά που βασίζονται σε psycholinguistic databases, constituent parse trees, αλλά και χαρακτηριστικά λόγου.

Τέλος δείχνουν τα αποτελέσματα στον παρακάτω πίνακα C.3. Τα παρακάτω αποτελέσματα έγιναν με τον Sequential Minimal Optimization(SMO) classifier και συγκεκριμένα με linear kernel.

Πίνακας C.3: Πίνακας παρουσίασης αποτελεσμάτων με την χρήση του *SMO Classifier* στην εργασία [2].

Ομάδα χαρακτηρισ- τικά	Αριθμός χαρακτηριστικών της ομάδας	Ακρίβεια
Traditional	10	58.5%
Word	10	67.19%
Psycholinguistic	11	52.02%
LexVar, POS	29	72.48%
Syntactic Features	28	73.89%
Discourse Features	67	63.66%
Total	155	78.13%

Τα αποτελέσματα αυτά σε παρόμοιες εργασίες στον τομέα του ARA είναι κοντά στο 90% αλλά επίσης οι συγγραφείς αναφέρουν και το ότι είναι ποιο δύσκολο να γίνει το classification για κείμενο ίδιου θέματος απλά διαφορετικής δυσκολίας.

Στη συνέχεια αναφέρουν και τα αποτελέσματα τους για το Corpus στον τομέα του Text Simplification. Για την αξιολόγηση του τομέα αυτού χρησιμοποιούνται διαφορετικές μετρικές από την ακρίβεια, συγκεκριμένα 2 φόρμουλες εν ονόματι BLEU και METEOR score. Σύμφωνα με τους συγγραφείς ένα score πάνω από 50 και στα 2 υπονοεί ότι η πρόταση που παράχθηκε από τον κώδικα είναι κατανοητή στον χρήστη. Με αυτό στο νού το BLUE score της εργασίας ήταν στο 54.45 και το METEOR score ήταν 46. Συγκρίνοντας με ήδη υπάρχοντα αποτελέσματα από μία άλλη εργασία που είχαν BLUE score 88.85 υποστηρίζουν ότι τα αποτελέσματα είναι λογικά καθώς η άλλη εργασία χρησιμοποίησε μοντέλο με 300 χιλιάδες ζεύγη προτάσεων, σε σύγκριση με τα 2166 ζεύγη προτάσεων του OneStopEnglish Corpus.

Τέλος, συμπεραίνουν το ότι το Corpus είναι χρήσιμο για τη χρήση του σε άλλες εργασίες στον τομέα του Text Classification και Simplification, κάτι το οποίο έχουν δείξει με τα παραπάνω αποτελέσματα.

C.2 Text Simplification to Specific Readability Levels

Η επόμενη εργασία που θα αναλύσουμε δημοσιεύτηκε το 2023 από τους Wejdan Alkaldi και Diana Inkpen [10] και έχει ως κύριο θέμα το text simplification. Η εργασία αυτή χρησιμοποιεί το Newsela Corpus το οποίο όπως αναφέρουν και οι συγγραφείς "περιέχει 10786 κείμενά σε επίπεδα δυσκολίας από το 0 έως το 4", με το 0 να είναι το πιο δύσκολο ως προς την κατανόησή. Στη συνέχεια οι συγγραφείς έκαναν μία διαδικασία που ονομάζεται Sentence Alignment, στην οποία βρίσκουν τα ζεύγη των προτάσεων από κάθε επίπεδο και τα "ευθυγραμμίζουν", καθώς τα κείμενα δεν ήταν ήδη σε τέτοια μορφή.

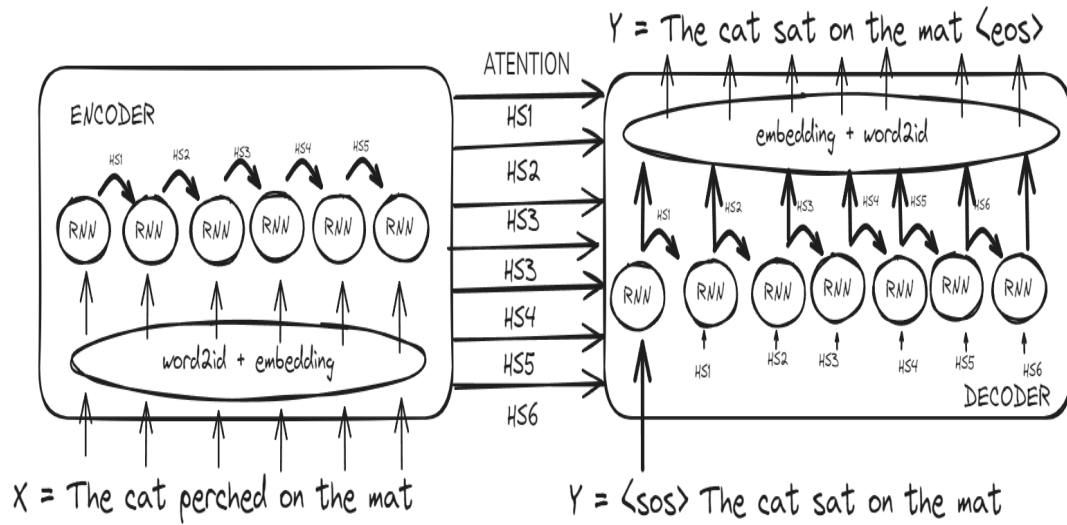
Ύστερα με το ευθυγραμμισμένο Corpus έκαναν classify με 80% train και 20% test split, δηλαδή χρησιμοποίησαν 80% για την εκπαίδευση του αλγορίθμου και 20% για την αξιολόγηση του. Ο αλγόριθμος που χρησιμοποίησαν για αυτό είναι ο CNN(Convolutional Neural Network) classifier, ο SVM(Support vector machine) και τέλος ο Random Forest(RF). Τα αποτελέσματα φαίνονται καλά με 85.69% να είναι το υψηλότερο αποτέλεσμα στον CNN classifier C.4.

Πίνακας C.4: Πίνακας παρουσίασης αποτελεσμάτων στην εργασία [10]. Το xval δεν το ανέφεραν το τι είναι στην εργασία και με ψάξιμο υποθέτω το ότι είναι cross-validation.

Corpus	Classifier	Ακρίβεια
Training (xval)	CNN	85.69%
Training (xval)	SVM	81.02%
Training (xval)	Random Forest	85.64%
Test	CNN	85.52%
Test	SVM	80.68%
Test	Random Forest	85.48%

Στη συνέχεια συνεχίζουν οι συγγραφείς και πάνε στο θέμα του Text Simplification όπου αναφέρουν τον μοντέλο που χρησιμοποίησαν, εν ονόματι "Seq2seq(Sequence to Sequence) with attention". Το μοντέλο όπως αναφέρουν οι συγγραφείς "απαρτίζεται από 2 Recurrent Neural Networks (RNNs) ένα το οποίο το χρησιμοποιούν σαν Encoder και ένα σαν Decoder.

Παρακάτω βλέπουμε ένα σχήμα με το παράδειγμα που έθεσαν και οι συγγραφείς στο οποίο φαίνεται η αρχιτεκτονική του μοντέλου. Καθώς στην εργασία αυτή δε θα ασχοληθούμε με τέτοια μοντέλα θα αναφέρω το ότι το μοντέλο παίρνει σαν είσοδο μία πρόταση και δίνει σαν έξοδο μία άλλη. Η μετατροπή της κάθε λέξης σε μορφή την οποία μπορεί να επεξεργαστεί το μοντέλο είναι ενσωματωμένο μέρος του Encoder και του Decode.



Στη συνέχεια οι συγγραφείς αναλύουν και άλλα μέρη του μοντέλου όπως το Reinforcement Learning, Evaluation Method και τέλος πραγματοποιούν τα πειράματα με train/test split. Σαν αποτελέσματα καθώς οι συγγραφείς δημιουργούν 36 μοντέλα, δε θα μπορέσουμε να τα αναλύσουμε όλα εδώ. Οι μετρικές που χρησιμοποιήθηκαν από τους συγγραφείς είναι το BLUE το οποίο αναφέραμε και παραπάνω και το SARI. Το καλύτερο μοντέλο τους για κάθε επίπεδο δυσκολίας είχε σαν αποτελέσματα SARI 34.08 και BLUE 76.47 για το επίπεδο 1, SARI 32.51 και BLUE 68.69 για το επίπεδο 2, SARI 33.24 και BLUE 61.88 για το επίπεδο 3 και τέλος SARI 33.24 και BLUE 61.38 για το επίπεδο 4.

C.3 Automatic Classification of Text Complexity

Η εργασία [1] δημοσιεύθηκε το 2020 από τους Valentino Santucci, Filippo Santarelli, Luciana Forti and Stefania Spina με στόχο να αντιμετωπίσουν το θέμα του Text Clasifica-

tion αλλά για την Ιταλική γλώσσα. Το Corpus που χρησιμοποίησαν οι συγγραφείς είναι μια συλλογή από 692 κείμενα εξετάσεων για επίπεδα CEFR(Common European Framework of Reference) B1, B2, C1, και C2, τα επίπεδα όπως θα ήταν φυσικό ταξινομήθηκαν από ειδικούς σε αυτά τα επίπεδα.

Το κείμενο υπέστηξε επεξεργασία και συγκεκριμένα tokenization, PoS tagging, lemmatization, analysis of the morphological features, dependency parsing και constituency parsing.

- Tokenization είναι η διαδικασία χωρισμού ενός κειμένου πρόταση προς πρόταση, λέξη προς λέξη, γράμμα προ γράμμα. Γενικά έχει πολλές μορφές, αλλά στην εργασία αυτή όπως αναφέρουν και οι συγγραφείς χωρίζουν αρχικά πρόταση προς πρόταση και μετά λέξη προς λέξη, η διαδικασία αυτή είναι κάτι που πρέπει να γίνει για την εξαγωγή μερικών μετρικών.
- Επόμενο βήμα είναι το PoS tagging, σε αυτό το βήμα βάζουμε σε κάθε token, δηλαδή λέξη, το μέρος του λόγου στο οποίο ανήκει, πχ ρήμα, επίθετο. Είναι και αυτό βήμα που χρειάζεται για την εξαγωγή μερικών μετρικών όπως τον αριθμό των ρημάτων.
- Σαν βήμα 3 έχουμε το lemmatization, όπου στόχος είναι να βρούμε το λήμμα κάθε λέξης.
- Ύστερα οι συγγραφείς αναλύουν τα μορφολογικά χαρακτηριστικά κάθε λέξης πχ κλήση ενός ρήματος ή το γένος ενός ουσιαστικού.
- Μετά κάνουν το dependency parsing για το οποίο δημιουργούν ένα δένδρο όπου φαίνεται το ποιο token επηρεάζεται από ποιο token.
- Και τέλος κάνουν αυτό που αποκαλούν constituency parsing, το οποίο στην ουσία μας δείχνει τις προτάσεις με τη "μητρική πρόταση" να είναι η ρίζα του δένδρου, οι εσωτερικοί κόμβοι είναι κομμάτια τις πρότασης, πχ θα μπορούσε να είναι η ρηματική φράση(verb phrase), και τα φύλλα είναι κάθε token.

Στη συνέχεια οι συγγραφείς αναφέρονται στα Numeric Linguistic Features τα οποία χωρίζουν σε μερικές ομάδες, Raw Text Features, Lexical Features, Morphological Features, Morpho-Syntactic Features, Syntactic Features και Discursive Features.

Τα Raw Text Features είναι χαρακτηριστικά όπως αριθμός προτάσεων σε κάθε κείμενο, αριθμός λέξεων σε κάθε πρόταση, αριθμός χαρακτήρων ανά λέξη και αριθμός ξεχωριστών λέξεων στο κείμενο. Οι περισσότερες εργασίες χρησιμοποιούν έναν συνδυασμό από αυτά τα χαρακτηριστικά, εάν όχι όλα, καθώς είναι τα πιο απλά που θα μπορούσε κάποιος να παράξει από ένα κείμενο.

Τα Lexical Features είναι στα λόγια των συγγραφέων "τα χαρακτηριστικά τα οποία υπολογίζονται χρησιμοποιώντας τα αποτελέσματα των UDPipe, δηλαδή τα lemmas, PoS tags και τα morphological annotations" και "2 χαρακτηριστικά που χρησιμοποιούνται ευρέως για τα Ιταλικά". Τα χαρακτηριστικά αυτά είναι το Basic Italian vocabulary rate, Lexical diversity, Lexical variation, Lexical sophistication και το Nouns abstractness distribution.

- Το Basic Italian vocabulary rate μετράει τα λήμματα που απαρτίζουν κάθε διαφορετική κατηγορία στο "new basic Italian vocabulary" (NVdB). Σε αυτήν τη λίστα είναι χωρισμένα τα λήμματα σε 3 κατηγορίες οι οποίες είναι οι εξής: Fundamentals, High Usage, και High Availability.
- Το Lexical diversity είναι αριθμός των ξεχωριστών tokens προς τον αριθμό των ολικών tokens. Οι συγγραφείς αναφέρουν το πως άμα υπολογιστεί ο τύπος αυτός σε 100 τυχαία tokens το αποτέλεσμα θα είναι πιο δίκαιο για τη σύγκρισή με κείμενο διαφορετικού μεγέθους. Παρακάτω φαίνεται ο τύπος.

$$\text{Lexical Diversity} = \frac{\text{NumberOfTypes(UniqueTokens)}}{\text{NumberOfTokens}}$$

- Το Lexical variation περιέχει χαρακτηριστικά όπως το lexical density το οποίο είναι ο αριθμός των content words προς τον αριθμό των tokens στο κείμενο και την κατανομή καθεμίας από τις κατηγορίες των content words. Τα content words είναι ο αριθμός των tokens που έχουν γίνει PoS tagged σαν verbs, nouns, adjectives, or adverbs,

$$\text{Lexical Density} = \frac{\text{NumberOfContentWords(verbs, nouns, adjectives, or adverbs)}}{\text{NumberOfTokens}}$$

- Το Lexical sophistication είναι ένα χαρακτηριστικό το οποίο υπολογίζεται χρησιμοποιώντας το COLFIS Corpus, το οποίο είναι γραμμένο στα Ιταλικά και περιέχει τη συχνότητα των λέξεων.

Τελευταίο σε αυτήν την ομάδα είναι το Nouns abstractness distribution, το οποίο υπολογίζεται σύμφωνα με το ποσοστό των nouns που έχει θεωρηθεί ως ένα από τα παρακάτω: Abstract, Semiabstract και Concrete.

Η επόμενη ομάδα χαρακτηριστικών είναι τα Μορφολογικά χαρακτηριστικά. Οι συγγραφείς αντιμετωπίζουν την κατηγορία αυτή με τη χρήση του morphological complexity index (MCI).

$$MorphologicalFeatures = \frac{(Within - SetDiv. + Across - Setdiv.)}{2} - 1$$

Συνεχίζοντας στην επόμενη ομάδα έχουμε τα Morpho-Syntactic χαρακτηριστικά. Τα χαρακτηριστικά αυτά υπολογίζονται σύμφωνα με του συγγραφείς με βάση το PoS tagging, morphological analysis και το "parsing performed in the NLP pipeline elaboration". Τα χαρακτηριστικά που περιέχονται στην ομάδα αυτή είναι το Subordinate ratio, POS tags distribution, Verbal moods distribution και Dependency tags distribution.

Ύστερα έχουμε τα Συντακτικά χαρακτηριστικά τα οποία περιέχουν το Depth of the dependency trees, Length of non-verbal chains, Verbal roots, Arity of verbal predicates, Length of the dependency link, Maximal non-verbal phrase, Number of syntactic constituents, Syntactic complexity, Relative subordinate order, Length of clauses και Subordination chains length.

Τέλος είναι και τα Discursive features(χαρακτηριστικά συζήτησης) τα οποία απαρτίζονται από το Referential cohesion και Deep causal cohesion.

- Το Referential cohesion είναι σύμφωνα με τους συγγραφείς "ο μέσος όρος και τυπική απόκλιση του αριθμού των ονομαστικών τύπων, που εμφανίζονται σε περισσότερες παρακείμενες προτάσεις μέχρι το μήκος 3."
- Και το Deep causal cohesion είναι η κατανομή από 8 κατηγορίες συνδετικών και το normalized entropy τους.

Στο επόμενο κεφάλαιο αναλύουν τα classification algorithms που χρησιμοποίησαν τα οποία είναι τα Random Forest classifier(RF), Gradient Boosted Decision Tree classifier(GBDT), Support Vector Machine(SVM) with radial basis function as kernel, Multi-Layer Perceptron with one hidden layer(MLP1), Multi-Layer Perceptron with two hidden layers(MLP2), Multi-Layer Perceptron with three hidden layers(MLP3), Quadratic Discriminant Analysis classifier(QDA), Naive Bayes(NB) classifier which fits Gaussian distri-

butions, K-Nearest Neighbors with uniform weights(kNNU), K-Nearest Neighbors with weights set on the basis of the distance in the features space(kNND).

Κάτι που δεν έχουμε δει απο τις προηγούμενες εργασίες είναι ότι οι συγγραφείς διεξάγουν τα πειράματά για καθένα από τους αλγορίθμους παρουσιάζοντας όχι μόνο το Accuracy του αλγορίθμου αλλά και το Precision, Recall και F1 Score. Και τα 3 θα τα αναλύσουμε σε παρακάτω κεφάλαιο όταν τα χρησιμοποιήσουμε για τα δικά μας πειράματα. Το classifier που είχε καλύτερο αποτέλεσμα ήταν τα Random Forest με accuracy 72,5%, precision 69,6%, recall 69,1% και τέλος f1-score 69%. Όλα τα αποτελέσματα παρουσιάζονται στον παρακάτω πίνακα C.5.

Πίνακας C.5: Πίνακας παρουσίασης αποτελεσμάτων στην εργασία [10]. Το xval δεν το ανέφεραν το τι είναι στην εργασία και με ψάξιμο υποθέτω το ότι είναι cross-validation.

Model	Accuracy	Precision	Recall	F1 Score
Random Forest	72,5%	69,6%	69,1%	69,4%
SVM	71,7%	69,0%	68,2%	68,6%
MLP3	68,4%	66,3%	65,7%	66,0%
GBDT	66,7%	63,4%	63,4%	63,4%
MLP2	67,4%	65,7%	65,3%	65,5%
MLP1	65,1%	62,5%	62,4%	62,4%
NB	59,7%	56,8%	58%	57,4%
kNNU	54,2%	56,0%	54,1%	55%
kNND	53,0%	55,1%	53,7%	54,4%
QDA	41,5%	44,2%	31,8%	36,9%

Τέλος, οι συγγραφείς δοκίμασαν να κάνουν κάτι που λέγεται feature selection. Το feature selection είναι η επιλογή των χαρακτηριστικών τα οποία επηρέασαν περισσότερο το αποτέλεσμα του classification, συγκεκριμένα χρησιμοποίησαν έναν ranking classifier ο οποίος βάζει τα χαρακτηριστικά με τη σειρά από τα πιο "σημαντικά" για τον αλγόριθμο. Τα αποτελέσματα θετικά και σημαντικά καθώς αφαιρώντας από 34.5% μέχρι και 44.6% έκανε βελτίωσή στον

αλγόριθμο από +4% έως και +21,6%, το οποίο δείχνει ότι πολλά από τα παραπάνω χαρακτηριστικά δεν προσθέτουν σημαντικές πληροφορίες και όπως αναφέρουν και οι ίδιοι.

C.4 Text Complexity Classification Based on Linguistic Information: Application to Intelligent Tutoring of ESL

Η εργασία αυτή [11] δημοσιεύτηκε το 2020 από τον M. Zakaria Kurdi. Στην εργασία αυτή ο συγγραφέας χρησιμοποιεί ένα Corpus το οποίο αναφέρει ως ESL Texts in Levels Corpus (ESLTL), το οποίο έχει 6171 κείμενα σε 3 διαφορετικά επίπεδα δυσκολίας τα οποία αντιστοιχούν σε επίπεδα A2, B1 και B2 στο CEFR.

Ο συγγραφέας χρησιμοποίησε διαφορετικές ομάδες χαρακτηριστικών από των οποίων η πρώτη είναι τα φωνολογικά χαρακτηριστικά. Τα χαρακτηριστικά αυτά είναι ο αριθμός συλλαβών ανά token, αριθμός φωνήματα ανα token και τέλος αριθμός γραμμάτων ανά token.

Ύστερα συνεχίζει με τα μορφολογικά χαρακτηριστικά από τα οποία πρώτο και τελευταίο είναι το stem diversity. Το stem diversity υπολογίζεται με τον αριθμό διαφορετικών λημμάτων προς τον αριθμό όλων των λημμάτων το οποίο φαίνεται στον παρακάτω τύπο.

$$StemDiversity = \frac{NumberOfDifferentLemmas}{NumberOfAllLemmas}$$

Στη συνέχεια παρουσιάζει ο συγγραφέας την επόμενη ομάδα η οποία απαρτίζεται από τα προθέματα και επιθήματα ανά λέξη. Μετά αναλύει το μήκος των nouns, adjectives, verbs, adverbs και το verb tenses.

Επόμενη ομάδα χαρακτηριστικών είναι τα lexical features από την οποία χρησιμοποιεί το Lexical Dencity(LEXdens), το Lexical sophistication, το Number of Different Words (NDW), το TTR, το The Measure of Textual Lexical Diversity (MTLD) , και το HD-D,

- Τα Lexical Dencity(LEXdens), το Lexical sophistication και το TTR είναι χαρακτηριστικά που αναφέραμε παραπάνω, αλλά σημαντικό είναι να αναφέρουμε στο TTR ότι ο συγγραφέας μελετάει και μερικά άλλα είδη TTR το GTTR(Guiraud's corrected TTR) και το CTTR(Caroll's corrected TTR).
- Το Number of Different Words (NDW) είναι το Lexical diversity ή variation το οποίο έχουμε αναφέρει επίσης παραπάνω.

Ύστερα ο συγγραφέας συνεχίζει προς τα συντακτικά χαρακτηριστικά. Τα χαρακτηριστικά αυτά είναι πολλά και διάφορα, αλλά αυτά που χρησιμοποίησε ο ίδιος είναι επιπέδου φράσεων, επιπέδου προτάσεων, επιπέδου T-unit, και τέλος βασισμένα στα n-grams.

- Τα χαρακτηριστικά επιπέδου φράσεων είναι χαρακτηριστικά όπως για παράδειγμα το ratio of NP, το οποίο είναι ο αριθμός των NP προς τον αριθμό των XP (total number of phrases)
- Ύστερα έχουμε τα χαρακτηριστικά προτάσεων τα οποία είναι χαρακτηριστικά όπως το μήκος μια πρότασης ή το ύψος του parse tree.
- Τα χαρακτηριστικά που έχουν να κάνουν με τα T-unit είναι 2, το μήκος του T-unit και η αναλογία των αριθμό T-unit προς τον αριθμό των πολύπλοκων t-units. Τα T-units είναι σύμφωνα με τον συγγραφέα "οι μικρότερες γραμματικές προτάσεις"
- Τέλος για την ομάδα αυτή αναφέρει ο συγγραφέας τα n-grams σαν χαρακτηριστικό τα οποία και αυτά έχουμε αναφέρει παραπάνω.

Συνεχίζοντας ο συγγραφέας αναφέρει τα Inter-Sential χαρακτηριστικά. Στην ομάδα αυτή αναλύει το cohesion και το coherence.

- Όσο αναφορά για το cohesion ο συγγραφέας δίνει τους 5 τύπους οι οποίοι είναι για λέξεις και για προτάσεις αντίστοιχα. Για τις λέξεις είναι ο αριθμός των discourse connectors προς τις λέξεις στο κείμενο και για τις προτάσεις είναι ο αριθμός των discourse connectors προς τις προτάσεις.

$$DiscourseConnectors : Words = \frac{NumberOfDiscourseConnectors}{NumberOfwords}$$

$$DiscourseConnectors : Sentances = \frac{NumberOfDiscourseConnectors}{NumberOfSentances}$$

- Τώρα πηγαίνοντας στο Coherence ο συγγραφέας αναφέρει το ότι το Coherence υπολογίζεται συμφωνά με "την απόσταση κάθε λέξης από ζεύγη συνεχόμενων προτάσεων"

Εν συνεχείς βλέπουμε την επόμενη κατηγορία, psycholinguistic features, μία κατηγορία που έχει χρησιμοποιηθεί σε προηγούμενη εργασία που έχουμε αναλύσει, και που περιέχει μετρικές όπως Kucera-Francis Written Frequency (KFWF), Kucera-Francis number of categories (KFNCat), Kucera-Francis Journal of Data Mining and Digital Humanities και πολλές άλλες που αναφέρει ο συγγραφέας.

Συνεχίζοντας πάμε στην τελευταία κατηγορία η οποία είναι η κατηγορία των "readability formulas". Τα readability formulas που χρησιμοποιήθηκαν είναι τα εξής: Gunning's Fog index, The Flesch-Kincaid formula, Coleman-Liau Index, Spache readability formula, The Dale-Chall formula, Automated Readability Index (ARI), The FORCAST Readability Formula και τέλος κάνει συσχέτιση των readability formulas.

- Το πρώτο χαρακτηριστικό στην ομάδα αυτή είναι το Gunning's Fog index. Το χαρακτηριστικό αυτό υπολογίζει τον χρόνο εκπαίδευσης που θα πρέπει να έχει ένας άνθρωπος για να κατανοήσει το κείμενο. Ο τύπος είναι ο εξής:

$$\text{Gunning's Fog Index} = 0.4 * \left(\frac{\text{NumberOfWords}}{\text{NumberOfSentences}} + 100 * \frac{\text{NumberOfComplexWords}}{\text{NumberOfWords}} \right)$$

- Συνεχίζοντας πάμε στο Flesch-Kincaid formula. Ο τύπος αυτό βγάζει σαν αποτέλεσμα αριθμούς από το 0 έως το 100, με το 0 να σημαίνει ότι για να κατανοηθεί το κείμενο πρέπει να είναι ο αναγνώστης σε επίπεδο απόφοιτου πανεπιστημίου και με 100 να σημαίνει ότι χρειάζεται επίπεδο 5η δημοτικού. Ο τύπος φαίνεται από κάτω.

$$\text{FleschKincaid formula} = 206.835 - \left(1.015 * \frac{\text{NumberOfWords}}{\text{NumberOfSentences}} \right) - \left(84.6 * \frac{\text{NumberOfSyllables}}{\text{NumberOfWords}} \right)$$

- Το επόμενο χαρακτηριστικό που αναφέρει ο συγγραφέας είναι το Coleman-Liau Index. Σε παρόμοιο τρόπο με τα προηγούμενα το αποτέλεσμα είναι μία προσέγγιση της τάξης σχολείου που θα πρέπει να είναι ο αναγνώστης, αλλά στο σύστημα σχολείου της US. Ο τύπος βρίσκεται παρακάτω με L να είναι τα γράμματα και S να είναι ο αριθμός προτάσεων.

$$\text{Coleman - Liau Index} = 0.0588 * L - 0.296 * S - 15.8)$$

- Επόμενο στη λίστα είναι το Spache readability formula. Σαν τύπος υπολογίζει το επίπεδο που χρειάζεται για την κατανόησή παιδικών βιβλίων μέχρι τη 4η τάξη δημοτικού. Ο τύπος είναι παρακάτω.

$$SpacheReadabilityFormula = (0.121 * \frac{NumberOfWords}{NumberOfSentences}) - (0.082 * \frac{NumberOfUnfamiliarWords}{NumberOfWords})$$

- Εν συνεχείς το Dale–Chall formula είναι παρόμοιο με το Flesch–Kincaid formula, με τη διαφορά του ότι ο τύπος δίνει περισσότερη βάση σε δύσκολες λέξεις παρά το νούμερο των συλλαβών.

$$TheDaleChallFormula = 0.1579 * (\frac{NumberOfDifficultWords}{NumberOfSentences} * 100) + 0.0496 * (\frac{NumberOfWords}{NumberOfSentences})$$

- Μετά έχουμε το Automated Readability Index (ARI), το οποίο έχει σαν αποτέλεσμα αριθμό ο οποίος αναλόγως με το πόσο μεγάλος είναι τόσο και ποιο περίπλοκο είναι το κείμενο και ο τύπος είναι ο εξής:

$$ARI = 4.71 * (\frac{NumberOfCharacters}{NumberOfWords}) + 0.5 * (\frac{NumberOfWords}{NumberOfSentences}) - 21.43$$

- Για το τελευταίο χαρακτηριστικό που θα μιλήσουμε σε αυτήν την εργασία είναι το FORCAST Readability Formula, το οποίο δημιουργήθηκε για τον Αμερικανικό Στρατό για να εξετάζουν το επίπεδο που χρειάζεται κάποιος για να γίνει στρατιώτης. Ο τύπος βρίσκει ακριβώς από κάτω από την παράγραφο αυτήν και N στον τύπο είναι το νούμερο των μονοσύλλαβων προτάσεων.

$$FORCASTReadabilityFormula = 20 - \frac{N}{10}$$

Τέλος, ο συγγραφέας αναλύει τη συσχέτιση μεταξύ των readability formulas με τη χρήση ενός πίνακα για να δείξει τα αποτελέσματα του.

Ο συγγραφέας μετά τα χαρακτηριστικά συνεχίζει στη διαδικασία επιλογής classifiers. Οι επιλογές του είναι οι εξής: Logistic Regression, Multilayer Perceptron, Adaptive Boosting (AdaBoost), Bagging και Random Forest, μερικούς από τους οποίους έχουμε δει και παραπάνω. Επίσης, η επιλογή τακτικής εκπαίδευσης του συγγραφέα ήταν 10 fold cross-validation. Τέλος, οι μετρικές αξιολόγησης κάθε classifier είναι recall, precision, F-score αλλά και Matthews's Correlation Coefficient¹⁸ (MCC) και το Roc Area.

Ο αλγόριθμος με τα καλύτερα αποτελέσματα ήταν ο Logistic Regression με όλα τα αποτελέσματα να βρίσκονται στον πίνακά C.6.

Πίνακας C.6: Πίνακας παρουσίασης αποτελεσμάτων στην εργασία [11].

Model	Precision	Recall	F1 Score	MCC	Roc
Random Forest	93,5%	93,6%	93,5%	90,3%	99,1%
Perceptron	94,4%	94,4%	94,4%	91,7%	98,9%
Ada-boost	81,2%	82,2%	81,3%	72,3%	93,5%
Bagging	94,1%	94,1%	94,1%	91,1%	99%
Logistic Regression	97,6%	97,6%	97,6%	96,4%	99,8%

Η εργασία αυτή έχει πολύ υψηλά αποτελέσματα σε σχέση με παρόμοιες εργασίες που έχουμε δει παραπάνω με αποτελέσματα που σχεδόν αγγίζουν το 100%.

C.5 Automated assessment of learner text complexity

Η εργασία αυτή [12] δημοσιεύτηκε το 2021 από τους Olga Lyshevskaya, Irina Panteleeva και Olga Vinogradova. Οι συγγραφείς αυτής της εργασίας χρησιμοποίησαν το REALEC corpus, περιέχει 14500 κείμενα εκθέσεων και σύμφωνα με τους συγγραφείς περιέχει περίπου 1.5 εκατομμύριο λέξεις στα Αγγλικά.

Συνεχίζοντας μπαίνουμε και πάλι στα χαρακτηριστικά. Αρχίζουμε και πάλι με γνωστές ομάδες χαρακτηριστικών πλέων, Lexical complexity. Στη συγκεκριμένη εργασία ασχοληθήκαμε με το Diverity, Density και Sophistication.

Στη συνέχεια πάμε στα μορφολογικά χαρακτηριστικά τα οποία απαρτίζονται από Derivational και Inflectional. Το χαρακτηριστικό Derivational το μέτρησαν οι συγγραφείς χωρίζοντας τα επιθέματα σε 6 επίπεδα. Μετά μέτρησαν το νούμερο των επιθεμάτων επιπέδου 3-6 και την αναλογία των επιθεμάτων επιπέδου 3-6 προς τον αριθμό όλων των επιθεμάτων. Όσο αναφορά τα Inflectional features τα μέτρησαν με το MCI που αναφέραμε και παραπάνω.

Όστερα βλέπουμε τα συντακτικά χαρακτηριστικά που χρησιμοποίησαν οι συγγραφείς τα οποία είναι πολλά και κάποια από αυτά έχουμε ξανά αναφέρει παραπάνω οπότε για να μην κουράζουμε δε θα τα αναφέρουμε. Μαζί με τα συντακτικά έχουμε και τα Discourse χαρακτηριστικά τα οποία επίσης αναφέραμε παραπάνω και για τον ίδιο λόγο θα τα προσπεράσουμε.

Τέλος, έχουμε τα Functional N-grams, τα οποία αναφέρουν οι συγγραφείς ως four-grams από λέξεις που χρησιμοποιήθηκαν πάνω από 280 φορές σε όλο το Corpus έχοντας τον αριθμό εμφανίσεων κάθε ενός από των n-grams αλλά και τη συχνότητα της.

Όσο αναφορά τα πειράματα των classifier οι επιλογές των συγγραφών ήταν logistic regression, random forest και k-neighbors. Τα καλύτερα αποτελέσματα που είχαν ήταν με τον random forest classifier με precision 85%, recall 89% και f1-score 85%.

C.6 Using Broad Linguistic Complexity Modeling for Cross-Lingual Readability Assessment

Η συγκεκριμένη εργασία [13] δημοσιεύτηκε το 2021 από τους Weiss Zarah, Chen Xiaobin και Meurers Detmar. Το Corpus το οποίο χρησιμοποίησαν οι συγγραφείς είναι το διαγλωσσικό Spotlight Corpus, το οποίο περιέχει κείμενο από 2 περιοδικά, 1 από περιοδικό στα Αγγλικά και ένα στα Γερμανικά. Το Corpus είναι χωρισμένο σε 2 γλώσσες και 3 επίπεδα δυσκολίας με 3285 κείμενα για το Αγγλικό και 1446 κείμενα για το Γερμανικό Corpus. Επίσης, γίνεται και αναφορά στο OneStopEnglish Corpus το οποίο αναφέραμε και παραπάνω και το οποίο θα χρησιμοποιήσουν οι συγγραφείς της εργασίας αυτής. Στην εργασία αυτή υπολογίστηκαν 312 χαρακτηριστικά, οπότε όπως είναι λογικό πολλά από αυτά θα αναφερθούν ονομαστικά και πολλά θα αναφερθούν ως κατηγορία χαρακτηριστικών.

Οι κατηγορίες των χαρακτηριστικών είναι οι εξής: Surface Length (LEN), Συντακτικά χαρακτηριστικά, Lexical χαρακτηριστικά, μορφολογικά χαρακτηριστικά, Discourse Cohesion, Language Use αλλά και Human Language Processing (HLP).

- Τα Surface Length features είναι χαρακτηριστικά όπως ο αριθμός προτάσεων, συλλαβών και παρομοίων μετρικών που εξερευνήσαμε και αναφέραμε παραπάνω.
- Επίσης έχουμε τα Συντακτικά, Lexical, μορφολογικά και Discourse Cohesion χαρακτηριστικά που έχουμε αναφέρει και προηγουμένως. Το Language Use αν και σαν όρο για χαρακτηριστικά δεν το έχουμε ξαναδεί είναι στην ουσία τα χαρακτηριστικά που έχουν να κάνουν με τη συχνότητα λέξεων τα οποία έχουμε δει παραπάνω.
- Τέλος για τα χαρακτηριστικά έχουμε τα HLP, τα οποία από ότι αναφέρουν οι συγγραφείς είναι "χαρακτηριστικά βασισμένα στη θεωρία τοποθεσίες εξάρτησης". Δεν αναφέρονται

πολλά παραπάνω για τα HLP χαρακτηριστικά.

Αρχικά, γίνετε το πείραμα στο OneStopEnglis Corpus με τους παρακάτω αλγορίθμους: RF, ordinal RF, SVM με radial kernel και SVM με polynomial kernel και τους εκπαιδεύουν με 10folds cross-validation. Τα αποτελέσματα που παρουσιάζουν οι συγγραφείς είναι μόνο για τον SVM με τον polynomial kernel όπως φαίνονται στον παρακάτω πίνακα C.6.

Πίνακας C.7: Πίνακας παρουσίασης αποτελεσμάτων στην εργασία [13].

	Precision	Recall	F1 Score
Elementary	93.2%	94.7%	94.0
Intermediate	87.8%	91.5%	89.6%
Advanced	95.5%	90.0%	92.6%

Στη συνέχεια οι συγγραφείς κάνουν και ένα πείραμα σε κάθε γλώσσα του Spotlight Corpus με τα ίδια χαρακτηριστικά. Για το Αγγλικό μέρος του Corpus έχει αποτελέσματα ο SVM 74.5% ακρίβεια με το corss-validation. Τα αποτελέσματα στο Γερμανικό μέρος του Corpus με τον ίδιο classifier τα αποτελέσματα είναι 88.0% ακρίβεια.

C.7 When Shallow is Good Enough: Automatic Assessment of Conceptual Text Complexity using Shallow Semantic Features

Αυτή η εργασία [14] δημοσιεύτηκε το 2020 από τους Sanja Stajner και Ioana Hulpus, Το Corpus που χρησιμοποιήθηκε από τους συγγραφείς ήταν το Newsela corpus, το οποίο περιέχει 1,911 τα οποία έχουν απλοποιηθεί σε 4 με 5 επίπεδα δυσκολίας.

Το βασικό σημείο για το οποίο μιλάνε οι συγγραφείς είναι ένα τα "Entity-based Conceptual Complexity Features" τα οποία χωρίζονται σε 2 διαφορετικές ομάδες νέων χαρακτηριστικών. Τα χαρακτηριστικά αυτά χωρίζονται σε Surface (text-based) και Shallow Semantics.

Τα Surface Text-based Features είναι χαρακτηριστικά με στόχο να "αποτυπώσουν την πολυπλοκότητά που εκφράζεται στην επιφανειακή δομή του λόγου". Τα χαρακτηριστικά είναι 5

Number of mentions divided by the total number of tokens, Average number of mentions divided by the total number of tokens (in a paragraph), Average number of mentions divided by the total number of tokens (in a sentence), Average distance (in sentences) between consecutive mentions, Average distance (in paragraphs) between consecutive mentions.

Τα Shallow Semantics Features είναι χαρακτηριστικά με στόχο να "αποτυπώσουν το ποσό του υπόβαθρο γνώσης που χρειάζεται για να κατανοηθεί ένα κείμενο". Τα χαρακτηριστικά είναι 14 οπότε θα αναφέρουμε τα πρώτα 5 ενδεικτικά Number of unique entities, Average number of unique entities in a paragraph, Average number of unique entities in a sentence, Entities to mentions ratio in the document, Average entities to mentions ratio in a paragraph.

Καθώς δεν αναλύονται τα χαρακτηριστικά και αναφέρονται μόνο από όνομα, έτσι τα αναφέραμε και εδώ. Επίσης, οι συγγραφείς χρησιμοποίησαν και 4 ψυχογλωσσολογικά χαρακτηριστικά γία όπως το ανάφεραν και οι ίδιοι "ως καλή βάση". Τα χαρακτηριστικά αυτά είναι: familiarity, age of acquisition, concreteness, and imagery.

- Το familiarity είναι στην ουσία η συχνότητα που συναντάμε μία λέξη
- Το age of acquisition είναι η ηλικία στην οποία μία λέξη αποκτάτε.
- Το concreteness είναι το πόσο "χειροπιαστός" είναι το αντικείμενο στο οποίο αναφέρετε η λέξη.
- Και τέλος το imagery είναι η "ένταση" με την οποία μία λέξη "δημιουργεί" μία εικόνα.

Τέλος, έχουμε τα πειράματα. Το καλύτερο αποτέλεσμα σαν ξεχωριστή κατηγορία χαρακτηριστικών ήταν τα shallow semantic feature με 94% F-measure στον Logistic Classifier και με τον συνδυασμό όλων των χαρακτηριστικών να έχει το καλύτερο αποτέλεσμα με 96% F-measure και για τον Logistic αλλά και τον SVM Classifier.

Κεφάλαιο D

Υλοποίηση

Σε αυτό το κεφάλαιο θα εξηγήσουμε τη μεθοδολογία που ακολουθήσαμε για την παραγωγή του κώδικα και τη διεξαγωγή του πειράματος. Τα βήματα που θα κάνουμε είναι βήματα που έχουν γίνει σε κάποιες από τις εργασίες που δείξαμε παραπάνω σαν παράδειγμα και όταν συναντάμε κάτι που χρησιμοποιήσαμε από αυτές θα αναφέρουμε το από πού πήραμε την ιδέα.

D.1 Επιλογή Corpus

Αρχικά έχουμε να επιλέξουμε το Corpus που θα χρησιμοποιήσουμε. Η επιλογή μας θα πρέπει ιδανικά να είναι από ένα Corpus το οποίο έχει sentence aligned προτάσεις για την εύκολη χρήση αλλά και γιατί το να έχουμε ίδιο αριθμό από κάθε "δυσκολία", ή κλάση όπως θα το αναφέρουμε από εδώ και πέρα, σημαίνει το ότι θα χρειαστεί λιγότερη επεξεργασία των αποτελεσμάτων στο τέλος. Η επιλογή μας ήταν κοντινή σε σχέση με το OneStopEnglish[2] και το PWKP Corpus [15]. Το NewSela ήταν και αυτό μία σχέση αλλά καθώς θέλει διαδικασία για τη λήψη του δεν το διαλέξαμε. Η τελική επιλογή ήταν να χρησιμοποιήσουμε και τα 2. Ο αριθμός των προτάσεων που χρησιμοποιούμε στο PWKP είναι οι πρώτες 5000 για κάθε μία από της κλάσεις, οπότε σύνολο 10000 και στον OneStopEnglish πήραμε τα επίπεδα Advanced και Elementary, οι οποίες είναι συγκεκριμένα 2166 ζεύγη. Σημαντικό να αναφερθεί είναι το ότι υπάρχουν κάποια σχόλια στην εργασία [16] σχετικά με το Corpus στην οποία οι συγγραφείς ισχυρίζονται ότι το Corpus δεν είναι αξιόπιστο καθώς ζεύγη προτάσεων είναι paired με λάθος τρόπο και ακόμα και ότι οι προτάσεις είναι σε περιπτώσεις ταξινομημένες λάθος με

την ποιο περίπλοκη μερικές φορές να είναι η ποιο απλή από τις 2. Όσο αναφορά τα δικά μας συμπεράσματα θα τα αναφέρουμε στο δικό τους κεφάλαιο.

D.2 προεπεξεργασία Corpus

Η προεπεξεργασία που έγινε στο Corpus άρχισε με την αφαίρεση των στίξεων, αριθμών και διπλοτύπων κενών ενδιάμεσα από τις λέξεις, καθώς μετά την αφαίρεση των στίξεων δημιουργήθηκαν κενά. Μετά από αυτό κάναμε το lemmatization το οποίο αναφέρθηκε και στην εργασία [1]. Παρακάτω φαίνεται ο κώδικας που χρησιμοποιήθηκε για την προεπεξεργασία του Corpus.

```
1 #function to remove punctuation and digits , as well as covertng all ↵
   characters to lower case and removing extra white spaces
2 def remove_punctuation_and_lowercase(corpus_):
3
4     nlp = spacy.load("en_core_web_sm")#—————
5
6     translation_table_punc = str.maketrans(""," ", string.punctuation↵
       )
7     translation_table_num = str.maketrans(' ',' ', '0123456789')
8     for i in range(len(corpus_)):
9         temp_text = corpus_[i].translate(translation_table_punc).↵
           lower()
10        # a regular expression to replace multiple spaces with a ↵
           singular one
11        temp_text = re.sub(r'\s+', ' ', temp_text)
12
13        # lemmatization
14        doc = nlp(temp_text)#—————
15        lemmatized_text = ' '.join([token.lemma_ for token in doc])
16
17        # temp_text.translate(translation_table_num)
18        corpus_[i] = lemmatized_text.translate(translation_table_num)
19
20    return corpus_
```

D.3 Επιλογή χαρακτηριστικών

Τα χαρακτηριστικά τα οποία χρησιμοποιήσαμε τελικά είναι 29.

Αρχικά έχουμε τα readability formulas, από τα οποία χρησιμοποιήσαμε τα εξής:

- Flesch Reading Ease formula [11],
- Flesch-Kincaid Grade Level [2],
- SMOG Index [17],
- Gunning FOG Formula(Fog Scale) [18],
- ARI [19],
- Dale-Chall Readability Score [11],
- Spache Readability Formula [11],
- και τέλος το Coleman-Liau Index [11].

Όλα τα παραπάνω τα υπολογίζουμε με τη χρήση της βιβλιοθήκης `texstat` όπως φαίνεται και στον παρακάτω κώδικα.

```
1 # function to extract the readability formulas
2 def readability_features_textstat(corpus):
3     flesch_kincaid_features = []
4     flesch_features = []
5     smog_index_features = []
6     gunning_fog_features = []
7     automated_readability_features = []
8     dale_chall_readability_features = []
9     spache_readability_features = []
10    coleman_liau_features = []
11
12    for text in corpus:
```

```

13
14     flesch_kincaid_features.append(txs.flesch_kincaid_grade(text)↵
15         )
16     flesch_features.append(txs.flesch_reading_ease(text))
17     smog_index_features.append(txs.smog_index(text))
18     gunning_fog_features.append(txs.gunning_fog(text))
19     automated_readability_features.append(txs.↵
20         automated_readability_index(text))
21     dale_chall_readability_features.append(txs.↵
22         dale_chall_readability_score(text))
23     spache_readability_features.append(txs.spache_readability(↵
24         text))
25     coleman_liau_features.append(txs.coleman_liau_index(text))# ↵
26     didn't use
27
28     return flesch_kincaid_features, flesch_features, ↵
29         smog_index_features, gunning_fog_features, ↵
30         automated_readability_features, ↵
31         dale_chall_readability_features, spache_readability_features

```

Στη συνέχεια χρησιμοποίησα τα 4 από τα 3 "Raw Text Features" που αναφέρεται στην εργασία [1], τα οποία είναι ο αριθμός των tokens ανά πρόταση, ο αριθμός χαρακτήρων ανά token και ο αριθμός διαφορετικών tokens στην πρόταση [20]. Ο λόγος που δε χρησιμοποίησα και τα 4 είναι ότι ο αριθμός των προτάσεων σε ένα Corpus που απαρτίζεται από 1 πρότασή τη φορά δεν έχει αξία σαν μετρική, θα ήταν για κάθε πρόταση απλά ο αριθμός 1.

Ύστερα υπολόγισα το TTR μαζί και με τα CTTR και GTTR [11]. Το function που βλέπετε παρακάτω παίρνει σαν παραμέτρους το προεπεξεργασμένο corpus και για κάθε πρόταση υπολογίζει και στο τέλος επιστρέφει τα αποτελέσματά. Για θέμα ευκολίας με τη χρήση της βιβλιοθήκης nltk υπολογίζουμε και τα PoS tags κάθε λέξης για τη χρήση τους σε επόμενα functions και τα επιστρέφουμε και αυτά. Ο κώδικας αυτός περιέχετε στο παρακάτω function.

```

1 # function to extract the pos tags s well as some other features
2 # function to extract the pos tags s well as some other features
3 def extract_pos_nltk(corpus):
4     pos_tags= []
5     words = []

```

```

6     sentence_len = []
7     ttr = []
8     gttr = []
9     cttr = []
10    unique_tokens = []
11    char_count = []
12
13    for text in corpus:
14        # tokenize sentence into words
15        words = nltk.word_tokenize(text)
16        sentence_len.append(len(words))
17        char_count.append(len(text.replace(" ", ""))/len(words))
18        # count of the unique words in a sentence
19        unique_words = set(words)
20        unique_tokens.append(len(unique_words))
21        if len(words) == 0:
22            ttr.append(0)
23        else:
24            ttr.append(len(unique_words) / len(words))
25
26        gttr.append(len(unique_words) * math.sqrt(len(words)))
27        cttr.append(len(unique_words) * math.sqrt(len(words)/2))
28        pos_tags.append(nltk.pos_tag(words))
29
30    return sentence_len, pos_tags, ttr, gttr, cttr, unique_tokens, ←
        char_count

```

Στη συνέχεια πήγαμε στα χαρακτηριστικά PoS. Στον παρακάτω κώδικα βλέπετε το function την εξαγωγή των νούμερων των verbs, nouns, adjectives, adverb και pronoun [21] από κάθε πρόταση χρησιμοποιώντας τα PoS tags από το προηγούμενο function. Αρχικά περνάμε τα PoS tags ως παράμετρο στο function, και μετά με ένα loop μετράμε για κάθε πρόταση πόσα από κάθε tags έχουμε και μετά επιστρέφουμε τα αποτελέσματα.

```

1 # function to extract the number of different pos tags per sentence
2 def pos_feats(pos_extracted):
3     verbs = []
4     nouns = []
5     adje = []

```

```

6     advrb = []
7     prn = []
8
9
10    for sentence in pos_extracted:
11        temp_verb = 0
12        temp_noun = 0
13        temp_adje = 0
14        temp_advr = 0
15        temp_prn = 0
16        # find the count of each pos in a sentence
17        for word in sentence:
18            if word[1] in ('VB', 'VBD', 'VBD', 'VBG', 'VBN', 'VBP', '↵
                VBZ'):
19                temp_verb += 1
20
21            if word[1] in ("RB", 'RBR', 'RBS', 'WRB'):
22                temp_advr += 1
23
24            if word[1] in ('NN', 'NNS', 'NNP', 'NNPS'):
25                temp_noun += 1
26
27            if word[1] in ('JJ', 'JJR', 'JJS'):
28                temp_adje += 1
29
30            if word[1] in ('PRP', 'PRP$', 'WP', 'WP$'):
31                temp_prn += 1
32
33        verbs.append(temp_verb)
34        nouns.append(temp_noun)
35        adje.append(temp_adje)
36        advrb.append(temp_advr)
37        prn.append(temp_prn)
38
39    return verbs, nouns, adje, advrb, prn

```

Ύστερα μεταβαίνουμε στο function στο οποίο υπολογίζουμε τα content words incidence και raw content word incidence [1]. Αρχικά θέτουμε μία λίστα με όλα τα content tags, και

αν υπάρχουν τα tags μιας λέξης στη λίστα τα αθροίζουμε. Στη συνέχεια τα διαιρούμε προς το νούμερο των λέξεων κάθε πρότασης. Τέλος, επιστρέφουμε και το μι επεξεργασμένο αριθμό content words.

```

1 # function to extract the number of word incidence
2 def calculate_word_incidence(pos_tags, total_words):
3     content_words = []
4     raw_content_word_incidence = []
5
6     # make a list of the content tags
7     content_tags = {'NN', 'NNS', 'NNP', 'NNPS', # Nouns
8                     'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ', # Verbs
9                     'JJ', 'JJR', 'JJS', # Adjectives
10                    'RB', 'RBR', 'RBS'} # Adverbs
11
12
13     i = 0
14     for sentence in pos_tags:
15         for word in sentence:
16             content_words = 0
17
18             if word[1] in content_tags:
19                 content_words += 1
20
21             if total_words[i] == 0:
22                 content_word_incidence = 0
23
24             else:
25                 content_word_incidence = int(content_words) / int(↵
26                     total_words[i])
27
28                 content_words.append(content_word_incidence)
29                 raw_content_word_incidence.append(content_words)
30
31             i += 1
32
33     return content_words, raw_content_word_incidence

```

Επόμενο function υπολογίζει μόνο το χαρακτηριστικό tf-idf [18]. Το χαρακτηριστικό αυτό το υπολογίζουμε με τη χρήση της sklearn βιβλιοθήκης και συγκεκριμένα με το TfidfVectorizer(). Τέλος, πριν επιστρέψουμε το αποτέλεσμα βεβαιωνόμαστε ότι σε περίπτωση διαίρεσης με το 0 το αποτέλεσμα θα τεθεί σε 0 αντί του να έχουμε error.

```
1 # function to calculate the tf idf feature
2 def calc_tf_idf(corpus):
3
4     tf_idf = []
5
6     # initialize the tf-idf vectorizer
7     vectorizer = TfidfVectorizer()
8
9     # fit and transform the sentence
10    tfidf_matrix = vectorizer.fit_transform(corpus)
11
12    feature_names = vectorizer.get_feature_names_out()
13
14    # convert the matrix to a dense format
15    dense_matrix = tfidf_matrix.todense()
16    dense_matrix_as_array = dense_matrix.tolist()
17
18
19    for doc_index, doc in enumerate(dense_matrix_as_array):
20        temp_val = 0
21        temp_count = 0
22        for word_index, score in enumerate(doc):
23            if score > 0: # append only words with a non-zero TF-IDF↔
24                score
25                temp_val += score
26                temp_count += 1
27            if temp_count > 0:
28                tf_idf.append(temp_val/temp_count)
29
30        else:
31            tf_idf.append(0)
32
33    return tf_idf
```

Επόμενο function που χρησιμοποιούμε είναι για τον υπολογισμό των αριθμών των διαφορετικών φράσεων με τη βοήθεια της βιβλιοθήκης nltk [11]. Θέτουμε μία "γραμματική" και ψάχνουμε τις προτάσεις και της αθροίζουμε ανάλογα με τη δομή της. Για να το κάνουμε αυτό χρησιμοποιούμε και το regex για τη διευκόλυνση της ανίχνευσης τους, το regex είναι ένα εργαλείο για να δημιουργούμε "μάσκες".

```

1 # function to extract number of different phrases
2 def calculate_np_vp_pp(corpus):
3     np_count_list = []
4     vp_count_list = []
5     pp_count_list = []
6
7     grammar = r"""
8         NP: {<DT|JJ|NN.*>+}          # Chunk sequences of DT, ←
9         JJ, NN
10        PP: {<IN><NP>}                # Chunk prepositions ←
11        followed by NP
12
13        VP: {<VB.*><NP|PP|CLAUSE>+$} # Chunk verbs and their ←
14        arguments
15    """
16
17    # Create a chunk parser
18    chunk_parser = nltk.RegexpParser(grammar)
19
20    for sentence in corpus:
21        np_count = 0
22        vp_count = 0
23        pp_count = 0
24
25        # tokenize sentence into words
26        words = nltk.word_tokenize(sentence)
27        # extract POS tags for the words
28        tagged_words = nltk.pos_tag(words)
29        # extract phrases
30        tree = chunk_parser.parse(tagged_words)
31        # count NP, VP, PP
32        for subtree in tree:

```

```

30         if isinstance(subtree, nltk.Tree):
31             if subtree.label() == 'NP':
32                 np_count += 1
33             elif subtree.label() == 'VP':
34                 vp_count += 1
35             elif subtree.label() == 'PP':
36                 pp_count += 1
37
38         np_count_list.append(np_count)
39         vp_count_list.append(vp_count)
40         pp_count_list.append(pp_count)
41
42     return np_count_list, vp_count_list, pp_count_list

```

Επόμενο function χρησιμεύει για τον υπολογισμό του αριθμού n-gram στην πρόταση. Το function αυτό το καλώ για $n = 2, 3$ και 4 , όπως αναφέρεται και στην εργασία [11], και επιστρέφω τον αριθμό τους ανά πρόταση και ανά λέξη.

```

1  # function to extract the number of ngrams per word and per sentence
2  def n_grams_(corpus_smpl, corpus_adv, n, sentence_len_smpl, ←
    sentence_len_adv):
3      bi_grams_per_sentence = []
4      bi_grams_per_word = []
5
6      corpus = corpus_smpl + corpus_adv
7      sentence_len = sentence_len_smpl + sentence_len_adv
8      l = 0
9      for sentence in corpus:
10
11         # tokenize the sentence into words
12         tokenized_sentence = nltk.word_tokenize(sentence)
13
14         # generate n-grams for each sentence and count them
15         sentence_ngrams = ngrams(tokenized_sentence, n)
16         total_ngrams = sum(len(ngram) for ngram in sentence_ngrams)
17         bi_grams_per_sentence.append(total_ngrams)
18         if sentence_len[l] == 0:
19             bi_grams_per_word.append('0')

```

```

20         else:
21             bi_grams_per_word.append(total_ngrams / int(sentence_len[1↵
                ]))
22         l += 1
23
24     return bi_grams_per_sentence, bi_grams_per_word

```

Πηγαίνοντας στο επόμενο function έχουμε τον αριθμό των φωνημάτων [11]. Για τον υπολογισμό τους χρησιμοποιούμε τη βιβλιοθήκη Pronouncing και συγκεκριμένα το count_phonemes() όπως βλέπουμε και παρακάτω.

```

1 # function to count the number of phonemes for an English word
2 def count_phonemes(word):
3
4     # get the ARPAbet phonemes from the CMU Pronouncing Dictionary
5     phoneme_list = pronouncing.phones_for_word(word)
6
7     if phoneme_list:
8         # select the first pronunciation variant (if there are ↵
            multiple)
9         phonemes = phoneme_list[0].split()
10        return len(phonemes), phonemes
11    else:
12        return 0, []

```

Το επόμενο μέρος του κώδικα προετοιμάζει τα χαρακτηριστά σε πίνακα μορφής επεξεργάσιμης από τους classifiers. Αρχικά θέτουμε ένα header για να ξέρουμε το ποιο χαρακτηριστικό είναι η κάθε στήλη. Έπειτα ενώνει τις λίστες ώστε το πρώτο μισό να είναι τα χαρακτηριστικά της κλάσης των απλών προτάσεων και το υπόλοιπο μισό είναι τα χαρακτηριστικά των περίπλοκων προτάσεων. Μετά μετατρέπει τη λίστα αυτή σε δομή pandas η οποία είναι αναγκαία για τη χρήση των classifier. Τέλος, γράφουμε όλα τα αποτελέσματα σε ένα .csv αρχείο σε περίπτωση που τα χρειαστούμε για το κεφάλαιο της ανάλυσης.

```

1 # header list of all the features names for each sentence
2 header = ['flesch_kincaid', 'flesch', 'smog_index', 'gunning_fog',
3          'ARI', 'dale_chall_readability', 'spache_readability',
4          'words_per_sentence', 'TTR', 'GTTR', 'CTTR', 'unique_tokens↵
            ', 'char_count', 'count_phonemes',

```

```

5         'bi-grams-per-sentence', 'bi-grams-per-word', '↵
        tri-grams-per-sentence', 'tri-grams-per-word', '↵
        four-grams-per-sentence', 'four-grams-per-word',
6     'NPs', 'VPs', 'PPs',
7     'verbs', 'nouns', 'adjective', 'adverb', 'pronouns', '↵
        content_words incidences', '↵
        _raw_number_of_content_words']

8
9     length_OSE = len(pp_count_OSE)/2
10    class_smpl_OSE = pd.Series(0, index=range(int(length_OSE)))
11    class_adv_OSE = pd.Series(1, index=range(int(length_OSE)))
12    class_smpl_list_OSE = class_smpl_OSE.values.tolist()
13    class_adv_list_OSE = class_adv_OSE.values.tolist()
14    complexity_level_OSE = (class_smpl_list_OSE + class_adv_list_OSE)
15
16    length_PWKP = len(pp_count_PWKP)/2
17    class_smpl_PWKP = pd.Series(0, index=range(int(length_PWKP)))
18    class_adv_PWKP = pd.Series(1, index=range(int(length_PWKP)))
19    class_smpl_list_PWKP = class_smpl_PWKP.values.tolist()
20    class_adv_list_PWKP = class_adv_PWKP.values.tolist()
21    complexity_level_PWKP = (class_smpl_list_PWKP + class_adv_list_PWKP)
22
23    # list of all the features for each sentence
24    lines_OSE = [flesch_kincaid_OSE, flesch_OSE, smog_index_OSE, ↵
        gunning_fog_OSE,
25        automated_readability_OSE, dale_chall_readability_OSE, ↵
        spache_readability_OSE,
26        words_per_sentence_OSE, TTR_OSE, GTTR_OSE, CTTR_OSE, ↵
        unique_tokens_OSE, char_count_OSE, count_phonemes_OSE,
27        bi_grams_per_sentence_OSE, bi_grams_per_word_OSE, ↵
        tri_grams_per_sentence_OSE, tri_grams_per_word_OSE, ↵
        four_grams_per_sentence_OSE, four_grams_per_word_OSE,
28        verbs_OSE, nouns_OSE, adje_OSE, advrb_OSE, prn_OSE,
29        np_count_OSE, vp_count_OSE, pp_count_OSE, ↵
        content_word_incidence_OSE, ↵
        no_f_raw_content_word_frequency_OSE]

30
31    lines_PWKP = [flesch_kincaid_PWKP, flesch_PWKP, smog_index_PWKP, ↵

```

```

    gunning_fog_PWKP,
32     automated_readability_PWKP, dale_chall_readability_PWKP, ↵
        spache_readability_PWKP,
33     words_per_sentence_PWKP, TTR_PWKP, GTTR_PWKP, CTTR_PWKP, ↵
        unique_tokens_PWKP, char_count_PWKP, count_phonemes_PWKP,
34     bi_grams_per_sentence_PWKP, bi_grams_per_word_PWKP, ↵
        tri_grams_per_sentence_PWKP, tri_grams_per_word_PWKP, ↵
        four_grams_per_sentence_PWKP, four_grams_per_word_PWKP,
35     verbs_PWKP, nouns_PWKP, adje_PWKP, advrb_PWKP, prn_PWKP,
36     np_count_PWKP, vp_count_PWKP, pp_count_PWKP, ↵
        content_word_incidence_PWKP, ↵
        no_f_raw_content_word_frequency_PWKP]
37
38
39 transposed_columns_OSE = list(map(list, zip(*lines_OSE)))
40 transposed_columns_PWKP = list(map(list, zip(*lines_PWKP)))
41
42 lists_df_OSE = pd.DataFrame(transposed_columns_OSE, columns=header).↵
    transpose()
43 lists_df_PWKP = pd.DataFrame(transposed_columns_PWKP, columns=header)↵
    .transpose()
44
45 # extract the features in a csv to possibly use for the paper
46 with open("Features_list_OSE.csv", 'w', newline='') as file:
47
48     writer = csv.writer(file)
49
50     # write each row to the CSV file
51     writer.writerow(header)
52     for row in lists_df_OSE:
53         writer.writerow(lists_df_OSE[row])
54
55 with open("Features_list_PWKP.csv", 'w', newline='') as file:
56
57     writer = csv.writer(file)
58
59     # write each row to the CSV file
60     writer.writerow(header)

```

```

61     for row in lists_df_PWKP:
62         writer.writerow(lists_df_PWKP[row])

```

Μετά θέτουμε τη μέθοδο με την οποία θα εκπαιδεύσουμε τους αλγορίθμους, συγκεκριμένα με train-test split και συγκεκριμένα χωρισμένο σε 70/30 train/test για το PWKP και 90/10 train/test για το OneStopEnglish Corpus των classifier.

```

1  X_OSE = lists_df_OSE.transpose()
2  y_OSE = complexity_level_OSE
3
4  # split the dataset into training and testing sets for the OSE Corpus
5  X_train_OSE, X_test_OSE, y_train_OSE, y_test_OSE = train_test_split(↵
        X_OSE, y_OSE, test_size=0.1, random_state=42)
6
7
8  X_PWKP = lists_df_PWKP.transpose()
9  y_PWKP = complexity_level_PWKP
10
11 # split the dataset into training and testing sets for the PWKP ↵
    Corpus
12 X_train_PWKP, X_test_PWKP, y_train_PWKP, y_test_PWKP = ↵
    train_test_split(X_PWKP, y_PWKP, test_size=0.3, random_state=42)

```

Τα επόμενα 3 block κώδικα είναι οι classifiers, στα οποία φαίνεται η διαδικασία από το πώς τους "ρυθμίζω" μέχρι και το πως τους αξιολογώ. Αρχίζουμε με τον RF classifiers. Οι παράμετροι που αλλάξα είναι το randome state σε 112 το οποίο είναι απλά διαφορετικό seed, αλλά μας φέρνει λίγο καλύτερα αποτελέσματα από το default 42, το min samples split το αλλάξαμε από 2 σε 8, το min samples leaf από 1 σε 2 και το max features το αλλάξα από το sqr(square root) στο log2. Τέλος, τα αποτελέσματα που συλέγω περιέχουν εκτός του accuracy και τα preceision, recall, f1 score και τέλος το consusion matrix. Το ίδιο θα γίνει για όλους τους classifiers.

```

1  # create a Random Forest classifier with various hyperparameters
2  rf_clf = RandomForestClassifier(
3      n_estimators=100,                # number of trees
4      max_depth=10,                   # maximum depth of each tree
5      min_samples_split=8,            # minimum number of samples to split↵
        an internal node

```

```
6     min_samples_leaf=2,          # minimum number of samples at a ↵
    leaf node
7     max_features='log2',        # number of features to consider ↵
    when looking for the best split
8     bootstrap=True,            # use bootstrap samples
9     criterion='gini',          # function to measure the quality of ↵
    a split
10    random_state=112,           # seed for reproducibility
11    n_jobs=-1                   # use all processors for parallel ↵
    processing
12 )
13
14
15 # train the classifier on the training data
16 rf_clf.fit(X_train, y_train)
17
18
19 # make predictions on the testing data
20 y_pred = rf_clf.predict(X_test)
21
22 # calculate and print the accuracy of the model
23 accuracy = accuracy_score(y_test, y_pred)
24 print(f"Accuracy: {accuracy * 100:.2f}%")
25
26 # calculate precision, recall, and F1-score
27 precision = precision_score(y_test, y_pred, average='weighted')
28 recall = recall_score(y_test, y_pred, average='weighted')
29 f1 = f1_score(y_test, y_pred, average='weighted')
30
31 print(f"Precision: {precision * 100:.2f}%")
32 print(f"Recall: {recall * 100:.2f}%")
33 print(f"F1-Score: {f1 * 100:.2f}%")
34
35 # generate a classification report
36 print("\nClassification Report:")
37 print(classification_report(y_test, y_pred))
38
39 # generate a confusion matrix
```

```
40 print("\nConfusion Matrix:")
41 conf_matrix = confusion_matrix(y_test, y_pred)
42 print(conf_matrix)
```

Επόμενος classifier είναι ο DecisionTree. Οι παράμετροι που πειράζαμε είναι το criterion το οποίο έχει ίδιο αποτέλεσμα και για τις 3 επιλογές, gini, entropy και log loss, πειράζαμε το max depth από None σε 4 κάτι που ανέβασε το accuracy από 67,76% σε 75,96%, το random το οποίο όπως αναφέραμε και παραπάνω είναι απλά το seed στο οποίο είδαμε το καλύτερο αποτέλεσμα και τέλος το max features το οποίο αλλάζαμε από None σε log2.

```
1 # initialize the DecisionTreeClassifier with some hyperparameters
2 clf = DecisionTreeClassifier(
3     criterion='entropy',
4     splitter='best',
5     max_depth=4,
6     min_samples_split=2,
7     min_samples_leaf=1,
8     max_features='log2',
9     random_state=163
10 )
11
12 # train the model
13 clf.fit(X_train, y_train)
14
15 # make predictions
16 y_pred = clf.predict(X_test)
17
18 # evaluate the model
19 accuracy = accuracy_score(y_test, y_pred)
20 precision = precision_score(y_test, y_pred, average='weighted')
21 recall = recall_score(y_test, y_pred, average='weighted')
22 f1 = f1_score(y_test, y_pred, average='weighted')
23 cm = confusion_matrix(y_test, y_pred)
24
25 print(f'Accuracy: {accuracy}')
26 print(f'Precision: {precision}')
27 print(f'Recall: {recall}')
28 print(f'F1 Score: {f1}')
```

```
29 print("\nClassification Report:")
30 print(classification_report(y_test, y_pred))
31
32 print(f'Confusion Matrix:\n{cm}')
```

Τέλος, έχουμε τον SVM ο οποίος είναι στην οικογένεια των SVM. Οι παράμετροι που άλλαξαν είναι μόνο ο class weight το οποίο έθεσα σε balanced από None.

```
1 from sklearn.svm import SVC
2 from sklearn.metrics import accuracy_score
3
4 # create a rbf SVM classifier
5 svm = SVC(kernel='rbf', gamma = "scale", probability = False, ←
    class_weight = "balanced")
6
7 # train the classifier
8 svm.fit(X_train, y_train)
9
10 # predict on the test set
11 y_pred = svm.predict(X_test)
12
13 # evaluate the classifier
14 accuracy = accuracy_score(y_test, y_pred)
15 precision = precision_score(y_test, y_pred, average='weighted')
16 recall = recall_score(y_test, y_pred, average='weighted')
17 f1 = f1_score(y_test, y_pred, average='weighted')
18 cm = confusion_matrix(y_test, y_pred)
19
20 print(f'Accuracy: {accuracy}')
21 print(f'Accuracy: {accuracy}')
22 print(f'Precision: {precision}')
23 print(f'Recall: {recall}')
24 print(f'F1 Score: {f1}')
25 print("\nClassification Report:")
26 print(classification_report(y_test, y_pred))
27
28 print(f'Confusion Matrix:\n{cm}')
```

Κεφάλαιο Ε

Αποτελέσματα

Εν συνεχίς μπαίνουμε στο κεφάλαιο των αποτελεσμάτων. Το κεφάλαιο αυτό θα περιέχει την παρουσίαση των αποτελεσμάτων, μετά της οποίας θα γίνει και η αξιολόγηση τους. Τέλος, θα συγκρίνουμε σε σχέση με παρόμοιες εργασίες, για να προετοιμαστούμε για το κεφάλαιο των συμπερασμάτων.

Ε.1 Παρουσίαση αποτελεσμάτων

Αρχικά θα αναφέρω ότι θα ξεκινήσουμε με τα αποτελέσματα του PWKP Corpus. Όπως φαίνεται και στον παρακάτω πίνακα E.1, το πιο υψηλό αποτέλεσμα επιτεύχθηκε με τον SVM, με 76,33% accuracy. Οι άλλοι 2 classifiers δεν ήταν πολύ πίσω σε σχέση με τον RF, ήταν ελάχιστα μακριά, με accuracy 76.17%, μόλις 0,19% πιο πίσω από τον SVM, επίσης το DecisionTree ήταν το χειρότερο και με αυτό να είναι σχετικά πάλι κοντά στα άλλα αποτελέσματα με 74.8%. Στη συνέχεια βλέπουμε το Confusion Matrix για κάθε έναν από τους αλγορίθμους στους πίνακες E.1, E.1 και E.1

Πίνακας Ε.1: Αποτελέσματα των classifier για το PWKP Corpus.

Model	Accuracy	Precision	Recall	F1 Score
RF(RandomForest)	76.17%	76.24%	76.17%	76.17%
DecisionTree	75%	75.28%	75%	74.97%
SVM	76,33%	76,33%	76,33%	76,33%

Πίνακας Ε.2: Confusion Matrix του RF classifier για το PWKP Corpus.

		True class		
		simple	advanced	Total
Classified as	simple	1148	389	1148 + 389
	advanced	326	1137	326 + 1137
Total		1148 + 326	389 + 1137	3000

Πίνακας Ε.3: Confusion Matrix του Decision Tree classifier για το PWKP Corpus.

		True class		
		simple	advanced	Total
Classified as	simple	1092	445	1092 + 445
	advanced	305	1158	305 + 1158
Total		1092 + 305	445 + 1158	3000

Πίνακας Ε.4: Confusion Matrix του SVM για το PWKP Corpus.

		True class		
		simple	advanced	Total
Classified as	simple	1187	350	1187 + 350
	advanced	360	1103	360 + 1103
Total		1187 + 360	350 + 1103	3000

Στη συνέχεια πηγαίνοντας στα αποτελέσματα για το OneStopEnglish Corpus ο καλύτερος

classifier ήταν ο RF, με το καλύτερο accuracy της εργασίας, 78.80%. Σε αυτό το Corpus, όπως φαίνεται και στον παρακάτω πίνακα E.1 οι άλλοι 2 classifier ήταν κατά πολύ χειρότεροι με των Decision Tree να έχει 57.83% και τον SVM 54.60%. Τα 2 αυτά αποτελέσματα είναι κοντά στο 50%, το οποίο σημαίνει ότι οι 2 αυτοί classifier προβλέπουν σχεδόν τυχαία. Και φυσικά έχουμε και για τους classifiers του OneStopEnglish τα Confusion Matrix για κάθε έναν από τους αλγόριθμους στους πίνακες E.1, E.1 και E.1

Πίνακας E.5: Αποτελέσματα των classifier για το OneStopEnglish Corpus.

Model	Accuracy	Precision	Recall	F1 Score
RF(RandomForest)	78.80%	79.15%	78.8%	78.69%
DecisionTree	54.61%	55.18%	54.61%	54.13%
SVM	54.61%	54.79%	54.61%	52.8%

Πίνακας E.6: Confusion Matrix του RF classifier για το OneStopEnglish Corpus.

		True class		
		simple	advanced	Total
Classified as	simple	190	33	190 + 33
	advanced	59	152	59 + 152
Total		190 + 59	33 + 152	434

Πίνακας E.7: Confusion Matrix του Decision Tree classifier για το OneStopEnglish Corpus.

		True class		
		simple	advanced	Total
Classified as	simple	99	124	99 + 124
	advanced	73	138	73 + 138
Total		99 + 73	124 + 138	434

Πίνακας Ε.8: *Confusion Matrix του SVM για το OneStopEnglish Corpus.*

		True class		Total
		simple	advanced	
Classified as	simple	164	59	164 + 59
	advanced	138	73	138 + 73
Total		164 + 138	59 + 73	434

Ε.2 Σύγκριση με παρόμοιες εργασίες

Φυσικά, όπως έχουμε δείξει στο κεφάλαιο με τίτλο "Παρόμοιες Εργασίες" το καλύτερο αποτέλεσμα που έχουμε δει είναι από την εργασία [11] με accuracy 97,6%. Το accuracy αυτό είναι πάρα πολύ καλό, και φυσικά δεν υπάρχει καν νόημα να αρχίσουμε τη σύγκριση αυτή. Αλλά είναι σημαντικό να αναφέρουμε πως στην εργασία αυτή χρησιμοποιήθηκαν 111 διαφορετικά χαρακτηριστικά κάτι το οποίο φυσικά είναι δύσκολο να συλλέξεις από τις προτάσεις καθώς πολλά από τα χαρακτηριστικά αυτά δεν έχουν έτοιμες βιβλιοθήκες και εργαλεία για να σε βοηθήσουν στον υπολογισμό τους. Με μόλις 29 χαρακτηριστικά ήταν λογικό να έχουμε πολύ δρόμο για να φτάσουμε τα αποτελέσματα που βλέπουμε από άλλες εργασίες.

Συνεχίζοντας έχουμε άλλη μία εργασία η οποία χρησιμοποιήσει και αυτή το OneStopEnglish Corpus. Η εργασία αυτή είναι η [13] με σχεδόν το ίδιο καλά αποτελέσματα με το accuracy να φτάνει στο 95.5% για τις Advanced προτάσεις, με τις Elementary να έχουν accuracy 93.2% και τις Intermediate να έχουν accuracy 87.8%. Τα αποτελέσματα αυτά είναι πολύ καλά, και πολύ μακριά από τα δικά μας, αλλά και πάλι οι συγγραφείς είχαν χρησιμοποιήσει 312 διαφορετικά χαρακτηριστικά, πάνω από 10 φορές τον αριθμό αυτών που χρησιμοποιήσαμε. Με αυτό στο νου πάμε στα αποτελέσματα ποιο κοντινά στα δικά μας.

Στη συνέχεια θα συγκρίνουμε τα αποτελέσματά μας από τις εργασίες [2], [10] και [1]. Με την ίδια σειρά τα καλύτερα αποτελέσματα τους είναι με τον SMO 78.13%, με τον CNN 85.69% και με τον RF 72,5%. Κοιτώντας τα καλύτερα αποτελέσματα που δείξαμε και στους πίνακες Ε.1 και Ε.1 βλέπουμε το ότι σαν τιμές είναι σχετικά κοντινές μεταξύ τους.

Τέλος, ίσος η εργασία [16] να είχε δίκιο με τις αμφιβολίες τις στο PWKP Corpus καθώς είχε αποτελέσματα χειρότερα του OneStopEnglish, αλλά δεν έχουμε αρκετά δεδομένα για να το

συμπεράνουμε.

Κεφάλαιο F

Συμπεράσματα

Εν τέλει φτάσαμε στο κεφάλαιο των συμπερασμάτων. Υπάρχουν πολλά πράγματα τα οποία μπορεί να έκανα διαφορετικά όσο αναφορά διάφορους τομείς στην εργασία αυτή.

F.1 Corpus

Αρχικά έχουμε τα Corpus. Ίσως η επιλογή του PWKP σαν corpus να μην ήταν η καλύτερή με αυτά που ξέρω πλέον, καθώς έχει το αρνητικό, το οποίο δε σκεφτήκαμε στην αρχή, του ότι είναι Corpus από κείμενα στο Wiki, δεν είναι ένα Corpus το οποίο δημιουργήθηκε από συλλογή ποιοτικών κειμένων, κάτι το οποίο μπορεί να δυσκόλεψε την έρευνα αυτή. Όσο αναφορά το OneStopEnglish Corpus, παρόλο που είχε το καλύτερο accuracy, είχε και τα 2 χειρότερα accuracy στους άλλους 2 classifiers. Το κύριο θέμα με το Corpus αυτό θεωρούμε ότι είναι το μικρό μέγεθος του. Ενώ για παράδειγμα το PWDK Corpus είχε τα καλύτερα αποτελέσματα για train-test split 70%-30%, το OneStopEnglish τα είχε για train-test split 90%-10%. Ίσως αν συνεχίσει η εμπλούτηση του Corpus θα μπορούσε κατά τη γνώμη μου να έχει κατά πολύ καλύτερα αποτελέσματα.

F.2 Χαρακτηριστικά

Όσο αναφορά τα χαρακτηριστικά, θεωρούμε ότι ο μεγαλύτερος περιορισμός ήταν ο αριθμός ποιητικών χαρακτηριστικών. Από τις εργασίες που αναφέραμε αυτές με τα καλύτερά αποτελέσ-

ματα είχαν πολλές φορές παραπάνω χαρακτηριστικά από τα πόσα χρησιμοποιήσαμε εμείς. Αν ήταν να αλλάξουμε κάτι στον τομέα των χαρακτηριστικών, θεωρούμε ότι αυτό θα ήταν το μόνο που θα αλλάζαμε. Τα χαρακτηριστικά που χρησιμοποιήσαμε φάνηκαν να έκανα μια σχετικά καλή δουλειά σε σύγκριση με τις άλλες εργασίες που αναφέραμε παραπάνω, ή τουλάχιστον αυτό θεωρούμε ότι δείχνουν τα αποτελέσματα των classifiers.

F.3 Αλγόριθμοι Μηχανικής Μάθησης

Στο θέμα των MM classifiers, μία αλλαγή που θα καλυπτεκε τα αποτελέσματα κατά την άποψη μας, θα μπορούσε να ήταν η χρήση παραπάνω αριθμών classifiers. Για παράδειγμα, στο OneStopEnglish Corpus η διαφορά από Classifier σε Classifier είναι γιγάντια (79.26% το καλύτερο και 54.60% το χειρότερο). με την αλλαγή αυτή θα μπορούσαμε να είχαμε βρει αλγόριθμους που είχαν καλύτερα αποτελέσματα.

F.4 Γενικά συμπεράσματα

Τέλος, θα αναφέρουμε το ότι ένα σημαντικό πρόβλημα στον τομέα του text classification θεωρούμε ότι είναι η διαθεσιμότητα διαφορετικών εργαλείων για την εξαγωγή των περίπλοκων χαρακτηριστικών. Με περισσότερα εργαλεία για τον σκοπό αυτό η διαλογή ενός μεγάλου αριθμού χαρακτηριστικών θα ήταν πολύ πιο εύκολη, κάτι που όπως αναφέραμε παραπάνω θεωρούμε ότι είναι ένας από τους πιο σημαντικούς παράγοντες στο θέμα αυτό.

Βιβλιογραφία

- [1] Valentino Santucci et al. “Automatic classification of text complexity”. In: *Applied Sciences* 10.20 (2020), p. 7285.
- [2] Sowmya Vajjala and Ivana Lučić. “OneStopEnglish corpus: A new corpus for automatic readability assessment and text simplification”. In: *Proceedings of the thirteenth workshop on innovative use of NLP for building educational applications*. 2018, pp. 297–304.
- [3] Sanja Štajner and Horacio Saggion. “Readability indices for automatic evaluation of text simplification systems: A feasibility study for spanish”. In: *Proceedings of the Sixth International Joint Conference on Natural Language Processing*. 2013, pp. 374–382.
- [4] Kristopher Kyle and Masaki Eguchi. “Evaluating NLP models with written and spoken L2 samples”. In: *Research Methods in Applied Linguistics* 3.2 (2024), p. 100120.
- [5] Vy Vo, Weiqing Wang, and Wray Buntine. “Unsupervised sentence simplification via dependency parsing”. In: *arXiv preprint arXiv:2206.12261* (2022).
- [6] T Zakzouk and H Mathkour. “Building A Corpus for Text Classification”. In: *The 2010 International Conference on Intelligent Network and Computing*. 2010.
- [7] Alexander L Fradkov. “Early history of machine learning”. In: *IFAC-PapersOnLine* 53.2 (2020), pp. 1385–1390.
- [8] Pat Langley et al. “The changing science of machine learning”. In: *Machine learning* 82.3 (2011), pp. 275–279.
- [9] Koosha Sharifani and Mahyar Amini. “Machine learning and deep learning: A review of methods and applications”. In: *World Information Technology and Engineering Journal* 10.07 (2023), pp. 3897–3904.

- [10] Wejdan Alkaldi and Diana Inkpen. “Text simplification to specific readability levels”. In: *Mathematics* 11.9 (2023), p. 2063.
- [11] M Zakaria Kurdi. “Text complexity classification based on linguistic information: application to intelligent tutoring of ESL”. In: *Journal of Data Mining & Digital Humanities* 2020 (2020).
- [12] Olga Lyashevskaya, Irina Panteleeva, and Olga Vinogradova. “Automated assessment of learner text complexity”. In: *Assessing writing* 49 (2021), p. 100529.
- [13] Zarah Weiss, Xiaobin Chen, and Detmar Meurers. “Using broad linguistic complexity modeling for cross-lingual readability assessment”. In: *Proceedings of the 10th Workshop on NLP for Computer Assisted Language Learning*. 2021, pp. 38–54.
- [14] Sanja Štajner and Ioana Hulpus. *When shallow is good enough: Automatic assessment of conceptual text complexity using shallow semantic features*. European Language Resources Association, ELRA-ELDA, 2020.
- [15] Delphine Bernhard, Zhemin Zhu, and Iryna Gurevych. *Simple-complex Sentence Pairs*. 2010. URL: <https://tudatalib.ulb.tu-darmstadt.de/handle/tudatalib/2447>.
- [16] Reno Kriz et al. “Simplification using paraphrases and context-based lexical substitution”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 2018, pp. 207–217.
- [17] Sanja Štajner et al. “What can readability measures really tell us about text complexity”. In: *Proceedings of workshop on natural language processing for improving textual accessibility*. Citeseer. 2012, pp. 14–22.
- [18] David Kauchak et al. “Text simplification tools: Using machine learning to discover features that identify difficult text”. In: *2014 47th Hawaii international conference on system sciences*. IEEE. 2014, pp. 2616–2625.
- [19] Abdel Karim Al Tamimi et al. “AARI: automatic Arabic readability index.” In: *Int. Arab J. Inf. Technol.* 11.4 (2014), pp. 370–378.
- [20] Sandra Aluisio et al. “Readability assessment for text simplification”. In: *Proceedings of the NAACL HLT 2010 fifth workshop on innovative use of NLP for building educational applications*. 2010, pp. 1–9.

- [21] Andrea Gasparetto et al. “A survey on text classification algorithms: From text to predictions”. In: *Information* 13.2 (2022), p. 83.

Συντμήσεις

AI	Artificial Intelingence
NLP	Natual language processing
ML	Machine Learning
TTR	Type-token ratio
CTTR	Caroll's corrected Type-token ratio
GTTR	Guiraud's corrected Type-token ratio
PoS	Part of Speach
ARA	automatic readability assesment
SMO	Sequential Minimal Optimization
CNN	Convolutional Neural Network
RNNs	Recurrent Neural Networks

RF	Random Forest classifier
GBDT	Gradient Boosted Decision Tree classifier
SVM	Support Vector Machine
MLP	Multi-Layer Perceptron
QDA	Quadratic Discriminant Analysis classifier
ESLTL	ESL Texts in Levels Corpus
LEXdens	Lexical Dencity
MTLD	The Measure of Textual Lexical Diversity
NB	Naive Bayes classifie
kNN	K-Nearest Neighbors
CEFR	Common European Framework of Reference
NDW	Number of Different Words
ARI	Automated Readability Index
HLP	Human Language Processing
sqr	square root

PWKP	parallel Wikipedia Simplification
L2	Second Language
MCC	Matthews's Correlation Coefficient ¹⁸
HLP	Human Language Processing

Γλωσσάρι Ξενικών Όρων

Artificial Intelingence	Τεχνική Νοημοσύνη
Text Simplification	Απλοποίηση Κειμένου
Text Clasification	Ταξινόμηση Κειμένου
Machine Learning	Μηχανική Μάθηση
Corpus	Σώμα κειμένων
Classifier	Αλγόριθμος Ταξινομητής
Part of Speach	Μέρος του λόγου
automatic readability assesment	Αυτόματη αξιολόγηση αναγνωσιμότητας
psycholinguistic databases	Ψυχολinguιστικές βάσεις δεδομένων
Sentence Alignment	Ευθυγράμμιση Προτάσεων
Parser	Αναλυτής

Kernel

Πυρήνας

Reinforcement Learning

Ενισχυτική Μάθηση

Lemmatization

Ληματοποίηση

Κώδικας χωρισμένος σε block

```
1 !pip install textstat
2 !pip install nltk
3 !pip install scikit-learn
4 !pip install Pronouncing
5
6 !pip install spacy
7 !python -m spacy download en_core_web_sm
```

```
8 import spacy
9 import requests
10 import string
11 import textstat as txs
12 import nltk
13 import math
14 from nltk import bigrams, trigrams, ngrams
15 from nltk.tokenize import word_tokenize, sent_tokenize
16 from collections import Counter
17 import re
18 from sklearn.feature_extraction.text import TfidfVectorizer, ←
    CountVectorizer
19 from sklearn.tree import DecisionTreeClassifier
20 from sklearn.svm import SVC
21 from sklearn.metrics import accuracy_score
22 import pandas as pd
23 import csv
24 import pronouncing #for the phonemes
25
26 from sklearn.model_selection import train_test_split, GridSearchCV
```

```

27 from sklearn.ensemble import RandomForestClassifier
28 from sklearn.metrics import accuracy_score, precision_score, ←
    recall_score, f1_score, confusion_matrix, classification_report
29
30 nltk.download('punkt')
31 nltk.download('averaged_perceptron_tagger')
32
33 # Download PWKP Corpus
34 url = "https://raw.githubusercontent.com/AlexandrosPoulis/←
    DownloadableCorpus/main/PWKP-108016"
35 response = requests.get(url)
36 if response.status_code == 200:
37     with open("PWKP.txt", "wb") as file:
38         file.write(response.content)
39         print("File downloaded successfully!")
40 else:
41     print("Failed to download the file.")
42
43
44 # Download OneStop Corpus
45 url = "https://raw.githubusercontent.com/AlexandrosPoulis/←
    DownloadableCorpus/refs/heads/main/ADV-ELE.txt"
46 response = requests.get(url)
47 if response.status_code == 200:
48     with open("Onestop.txt", "wb") as file:
49         file.write(response.content)
50         print("File downloaded successfully!")
51 else:
52     print("Failed to download the file.")

```

```

53 # split the PWKP Corpus in simple and advanced sentences
54 with open("PWKP.txt", "r") as file:
55     lines = file.readlines()
56     i = 0
57     simple_PWKP = []
58     advanced_PWKP = []
59     temp = 0
60

```

```

61     for line in lines:
62         i += 1
63         if i == 1 :
64             temp_adv = line
65
66         elif i == 2 and line.__eq__('\\n') == False:
67             temp_simple = line
68
69
70         if line.__eq__('\\n') :
71
72             advanced_PWKP.append(temp_adv)
73             simple_PWKP.append(temp_simple)
74             i = 0
75
76 # split the Onestop Corpus in simple and advanced sentences
77 with open("Onestop.txt", "r") as file:
78     lines = file.readlines()
79     i = 0
80     simple_Onestop = []
81     advanced_Onestop = []
82     temp = 0
83
84     for line in lines:
85         i += 1
86         if i == 1 :
87             temp_adv = line
88
89         elif i == 2:
90             temp_simple = line
91
92         elif i == 3:
93
94             advanced_Onestop.append(temp_adv)
95             simple_Onestop.append(temp_simple)
96             i = 0

```

```

97 #function to remove punctuation and digits , as well as coverting all ↔

```

```

    characters to lower case and removing extra white spaces
98 def remove_punctuation_and_lowercase(corpus_):
99
100     nlp = spacy.load("en_core_web_sm")#-----
101
102     translation_table_punc = str.maketrans(""," ", string.punctuation↵
        )
103     translation_table_num = str.maketrans(' ', ' ', '0123456789')
104     for i in range(len(corpus_)):
105         temp_text = corpus_[i].translate(translation_table_punc).↵
            lower()
106         # a regular expression to replace multiple spaces with a ↵
            singular one
107         temp_text = re.sub(r'\s+', ' ', temp_text)
108
109         # lemmatization
110         doc = nlp(temp_text)#-----
111         lemmatized_text = ' '.join([token.lemma_ for token in doc])
112
113         # temp_text.translate(translation_table_num)
114         corpus_[i] = lemmatized_text.translate(translation_table_num)
115
116     return corpus_
117
118
119 dict_simpl_ensure_unique = []
120 dict_adv_ensure_unique = []
121 number_of_sentenses = 5000
122
123 corpus_simple_no_punc_OSE = remove_punctuation_and_lowercase(↵
    simple_0nестop[:number_of_sentenses])
124 corpus_adv_no_punc_OSE = remove_punctuation_and_lowercase(↵
    advanced_0nестop[:number_of_sentenses])
125 corpus_no_punc_OSE = corpus_simple_no_punc_OSE + ↵
    corpus_adv_no_punc_OSE
126
127 corpus_simple_no_punc_PWKP = remove_punctuation_and_lowercase(↵
    simple_PWKP[:number_of_sentenses])

```

```

128 corpus_adv_no_punc_PWKP = remove_punctuation_and_lowercase(↵
    advanced_PWKP[:number_of_sentences])
129 corpus_no_punc_PWKP = corpus_simple_no_punc_PWKP + ↵
    corpus_adv_no_punc_PWKP

```

```

130 # function to extract the readability formulas
131 def readability_features_textstat(corpus):
132     flesch_kincaid_features = []
133     flesch_features = []
134     smog_index_features = []
135     gunning_fog_features = []
136     automated_readability_features = []
137     dale_chall_readability_features = []
138     spache_readability_features = []
139     coleman_liau_features = []
140
141     for text in corpus:
142
143         flesch_kincaid_features.append(txs.flesch_kincaid_grade(text)↵
            )
144         flesch_features.append(txs.flesch_reading_ease(text))
145         smog_index_features.append(txs.smog_index(text))
146         gunning_fog_features.append(txs.gunning_fog(text))
147         automated_readability_features.append(txs.↵
            automated_readability_index(text))
148         dale_chall_readability_features.append(txs.↵
            dale_chall_readability_score(text))
149         spache_readability_features.append(txs.spache_readability(↵
            text))
150         coleman_liau_features.append(txs.coleman_liau_index(text))# ↵
            didn't use
151
152
153     return flesch_kincaid_features, flesch_features, ↵
        smog_index_features, gunning_fog_features, ↵
        automated_readability_features, ↵
        dale_chall_readability_features, spache_readability_features
154

```

```

155 # function to extract the pos tags s well as some other features
156 def extract_pos_nltk(corpus):
157     pos_tags= []
158     words = []
159     sentence_len = []
160     ttr = []
161     gttr = []
162     cttr = []
163     unique_tokens = []
164     char_count = []
165
166     for text in corpus:
167         # tokenize sentence into words
168         words = nltk.word_tokenize(text)
169         sentence_len.append(len(words))
170         if len(words) == 0:
171             char_count.append(0)
172         else:
173             char_count.append(len(text.replace(" ", ""))/len(words))
174         # count of the unique words in a sentence
175         unique_words = set(words)
176         unique_tokens.append(len(unique_words))
177         if len(words) == 0:
178             ttr.append(0)
179         else:
180             ttr.append(len(unique_words) / len(words))
181
182             gttr.append(len(unique_words) * math.sqrt(len(words)))
183             cttr.append(len(unique_words) * math.sqrt(len(words)/2))
184             pos_tags.append(nltk.pos_tag(words))
185
186     return sentence_len, pos_tags, ttr, gttr, cttr, unique_tokens, ↵
        char_count
187
188
189 # function to extract the number of different pos tags per sentence
190 def pos_feautres(pos_extracted):
191     verbs = []

```

```

192     nouns = []
193     adje = []
194     advrb = []
195     prn = []
196
197
198     for sentence in pos_extracted:
199         temp_verb = 0
200         temp_noun = 0
201         temp_adje = 0
202         temp_adverb = 0
203         temp_prn = 0
204         # find the count of each pos in a sentence
205         for word in sentence:
206             if word[1] in ('VB', 'VBD', 'VBD', 'VBG', 'VBN', 'VBP', '↵
                VBZ'):
207                 temp_verb += 1
208
209             if word[1] in ("RB", 'RBR', 'RBS', 'WRB'):
210                 temp_adverb += 1
211
212             if word[1] in ('NN', 'NNS', 'NNP', 'NNPS'):
213                 temp_noun += 1
214
215             if word[1] in ('JJ', 'JJR', 'JJS'):
216                 temp_adje += 1
217
218             if word[1] in ('PRP', 'PRP$', 'WP', 'WP$'):
219                 temp_prn += 1
220
221         verbs.append(temp_verb)
222         nouns.append(temp_noun)
223         adje.append(temp_adje)
224         advrb.append(temp_adverb)
225         prn.append(temp_prn)
226
227     return verbs, nouns, adje, advrb, prn
228

```

```

229 # function to extract the number of word incidence
230 def calculate_word_incidence(pos_tags, total_words):
231     content_words = []
232     functional_words = []
233     raw_content_word_incidence = []
234
235
236     # make a list of the content tags
237     content_tags = { 'NN', 'NNS', 'NNP', 'NNPS', # Nouns
238                     'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ', # Verbs
239                     'JJ', 'JJR', 'JJS', # Adjectives
240                     'RB', 'RBR', 'RBS'} # Adverbs
241
242
243     # make a list of the functional tags
244     functional_tags = { 'CC',
245                        'CD',
246                        'DT',
247                        'EX',
248                        'FW',
249                        'IN',
250                        'LS',
251                        'MD',
252                        'PDT',
253                        'POS',
254                        'PRP', 'PRP$',
255                        'RP',
256                        'TO',
257                        'UH',
258                        'WDT', 'WP', 'WP$', 'WRB'}
259
260     i = 0
261     for sentence in pos_tags:
262         for word in sentence:
263             content_words = 0
264             funct_words = 0
265
266             if word[1] in content_tags:

```

```

267         content_words += 1
268
269         if word[1] in functional_tags:
270             funct_words += 1
271
272         if total_words[i] == 0:
273             content_word_incidence = 0
274
275         else:
276             content_word_incidence = int(content_words) / int(↵
                total_words[i])
277             functional_word_incidence = int(funct_words) / int(↵
                total_words[i])
278
279         contenct_words.append(content_word_incidence)
280         functional_words.append(functional_word_incidence)
281         raw_content_word_incidence.append(content_words)
282
283         i += 1
284
285     return contenct_words, functional_words, ↵
        raw_content_word_incidence
286
287 # function to calculate the tf idf feature
288 def calc_tf_idf(corpus):
289
290     tf_idf = []
291
292     # initialize the tf-idf vectorizer
293     vectorizer = TfidfVectorizer()
294
295     # fit and transform the sentence
296     tfidf_matrix = vectorizer.fit_transform(corpus)
297
298     feature_names = vectorizer.get_feature_names_out()
299
300     # convert the matrix to a dense format
301     dense_matrix = tfidf_matrix.todense()

```

```

302     dense_matrix_as_array = dense_matrix.tolist()
303
304
305     for doc_index, doc in enumerate(dense_matrix_as_array):
306         temp_val = 0
307         temp_count = 0
308         for word_index, score in enumerate(doc):
309             if score > 0: # append only words with a non-zero TF-IDF↵
310                 score
311                 temp_val += score
312                 temp_count += 1
313             if temp_count > 0:
314                 tf_idf.append(temp_val/temp_count)
315             else:
316                 tf_idf.append(0)
317
318     return tf_idf
319
320 # function to extract number of different phrases
321 def calculate_np_vp_pp(corpus):
322     np_count_list = []
323     vp_count_list = []
324     pp_count_list = []
325
326     grammar = r"""
327         NP: {<DT| JJ|NN.*>+}           # Chunk sequences of DT,↵
328         JJ, NN
329         PP: {<IN><NP>}                 # Chunk prepositions ↵
330         followed by NP
331         VP: {<VB.*><NP|PP|CLAUSE>+$} # Chunk verbs and their ↵
332         arguments
333         """
334
335     # Create a chunk parser
336     chunk_parser = nltk.RegexpParser(grammar)

```

```

336     for sentence in corpus:
337         np_count = 0
338         vp_count = 0
339         pp_count = 0
340
341         # tokenize sentence into words
342         words = nltk.word_tokenize(sentence)
343         # extract POS tags for the words
344         tagged_words = nltk.pos_tag(words)
345         # extract phrases
346         tree = chunk_parser.parse(tagged_words)
347         # count NP, VP, PP
348         for subtree in tree:
349             if isinstance(subtree, nltk.Tree):
350                 if subtree.label() == 'NP':
351                     np_count += 1
352                 elif subtree.label() == 'VP':
353                     vp_count += 1
354                 elif subtree.label() == 'PP':
355                     pp_count += 1
356
357         np_count_list.append(np_count)
358         vp_count_list.append(vp_count)
359         pp_count_list.append(pp_count)
360
361     return np_count_list, vp_count_list, pp_count_list
362
363 # function to extract the number of ngrams per word and per sentence
364 def n_grams_(corpus, n, sentence_len):
365     bi_grams_per_sentence = []
366     bi_grams_per_word = []
367
368     l = 0
369     for sentence in corpus:
370
371         # tokenize the sentence into words
372         tokenized_sentence = nltk.word_tokenize(sentence)
373

```

```

374     # generate n-grams for each sentence and count them
375     sentence_ngrams = ngrams(tokenized_sentence, n)
376     total_ngrams = sum(len(ngram) for ngram in sentence_ngrams)
377     bi_grams_per_sentence.append(total_ngrams)
378     if sentence_len[1] == 0:
379         bi_grams_per_word.append('0')
380     else:
381         bi_grams_per_word.append(total_ngrams / int(sentence_len[1] ←
382                                     )))
383     l += 1
384     return bi_grams_per_sentence, bi_grams_per_word
385
386
387 # function to count the number of phonemes for an English word
388 def count_phonemes(word):
389
390     # get the ARPAbet phonemes from the CMU Pronouncing Dictionary
391     phoneme_list = pronouncing.phones_for_word(word)
392
393     if phoneme_list:
394         # select the first pronunciation variant (if there are ←
395             multiple)
396         phonemes = phoneme_list[0].split()
397         return len(phonemes), phonemes
398     else:
399         return 0, []
400
401 # function to count the number of phonemes for an English word
402 def count_phonemes_sentence(corpus, words_per_sentence):
403     phonemes_count_list = []
404     l = 0
405     for sentence in corpus:
406
407         # tokenize the sentence into words
408         tokenized_sentence = nltk.word_tokenize(sentence)
409         temp_phonemes_count = 0

```

```

410         # count the phonemes in a sentence
411         for word in tokenized_sentence:
412             num_phonemes, phonemes = count_phonemes(word)
413             temp_phonemes_count += num_phonemes
414
415             phonemes_count_list.append(temp_phonemes_count / ↵
                words_per_sentence[1])
416
417         l += 1
418     return phonemes_count_list

```

```

419 flesch_kincaid_OSE, flesch_OSE, smog_index_OSE, gunning_fog_OSE, ↵
    automated_readability_OSE, dale_chall_readability_OSE, ↵
    spache_readability_OSE = readability_features_textstat(↵
        corpus_no_punc_OSE)
420 flesch_kincaid_PWKP , flesch_PWKP ,  smog_index_PWKP , ↵
    gunning_fog_PWKP , automated_readability_PWKP , ↵
    dale_chall_readability_PWKP , spache_readability_PWKP = ↵
    readability_features_textstat(corpus_no_punc_PWKP)
421
422
423 words_per_sentence_OSE, pos_tags_OSE, TTR_OSE, GTTR_OSE, CTTR_OSE, ↵
    unique_tokens_OSE, char_count_OSE = extract_pos_nltk(↵
        corpus_no_punc_OSE)
424 words_per_sentence_PWKP, pos_tags_PWKP, TTR_PWKP, GTTR_PWKP, ↵
    CTTR_PWKP, unique_tokens_PWKP, char_count_PWKP = extract_pos_nltk↵
    (corpus_no_punc_PWKP)
425
426
427 verbs_OSE, nouns_OSE, adje_OSE, advrb_OSE, prn_OSE = pos_feautres(↵
    pos_tags_OSE)
428 verbs_PWKP, nouns_PWKP, adje_PWKP, advrb_PWKP, prn_PWKP = ↵
    pos_feautres(pos_tags_PWKP)
429
430
431 content_word_incidence_OSE, functional_word_incidence_OSE, ↵
    no_f_raw_content_word_frequency_OSE = calculate_word_incidence(↵
        pos_tags_OSE, words_per_sentence_OSE)

```

```

432 content_word_incidence_PWKP, functional_word_incidence_PWKP, ↵
    no_f_raw_content_word_frequency_PWKP = calculate_word_incidence(↵
        pos_tags_PWKP, words_per_sentence_PWKP)
433
434 tf_idf_OSE = calc_tf_idf(corpus_no_punc_OSE)
435 tf_idf_PWKP = calc_tf_idf(corpus_no_punc_PWKP)
436
437 np_count_OSE, vp_count_OSE, pp_count_OSE = calculate_np_vp_pp(↵
    corpus_no_punc_OSE)
438 np_count_PWKP, vp_count_PWKP, pp_count_PWKP = calculate_np_vp_pp(↵
    corpus_no_punc_PWKP)
439
440 count_phonemes_OSE = count_phonemes_sentence(corpus_no_punc_OSE, ↵
    words_per_sentence_OSE)
441 count_phonemes_PWKP = count_phonemes_sentence(corpus_no_punc_PWKP, ↵
    words_per_sentence_PWKP)
442
443
444 bi_grams_per_sentence_OSE, bi_grams_per_word_OSE = n_grams_(↵
    corpus_no_punc_OSE, 2, words_per_sentence_OSE)
445 tri_grams_per_sentence_OSE, tri_grams_per_word_OSE = n_grams_(↵
    corpus_no_punc_OSE, 3, words_per_sentence_OSE)
446 four_grams_per_sentence_OSE, four_grams_per_word_OSE = n_grams_(↵
    corpus_no_punc_OSE, 4, words_per_sentence_OSE)
447
448
449 bi_grams_per_sentence_PWKP, bi_grams_per_word_PWKP = n_grams_(↵
    corpus_no_punc_PWKP, 2, words_per_sentence_PWKP)
450 tri_grams_per_sentence_PWKP, tri_grams_per_word_PWKP = n_grams_(↵
    corpus_no_punc_PWKP, 3, words_per_sentence_PWKP)
451 four_grams_per_sentence_PWKP, four_grams_per_word_PWKP = n_grams_(↵
    corpus_no_punc_PWKP, 4, words_per_sentence_PWKP)

```

```

452 # header list of all the features names for each sentence
453 header = [ 'flesch_kincaid', 'flesch', 'smog-index', 'gunning-fog',
454            'ARI', 'dale_chall_readability', 'spache_readability',
455            'words_per_sentence', 'TTR', 'GTTR', 'CTTR', 'unique_tokens↵
            ', 'char_count', 'count_phonemes',

```

```

456         'bi-grams-per-sentence', 'bi-grams-per-word', '↵
            tri-grams-per-sentence', 'tri-grams-per-word', '↵
            four-grams-per-sentence', 'four-grams-per-word',
457         'NPs', 'VPs', 'PPs',
458         'verbs', 'nouns', 'adjective', 'adverb', 'pronouns', '↵
            content_wrods_incidences']

459
460 length_OSE = len(pp_count_OSE)/2
461 class_smpl_OSE = pd.Series(0, index=range(int(length_OSE)))
462 class_adv_OSE = pd.Series(1, index=range(int(length_OSE)))
463 class_smpl_list_OSE = class_smpl_OSE.values.tolist()
464 class_adv_list_OSE = class_adv_OSE.values.tolist()
465 complexity_level_OSE = (class_smpl_list_OSE + class_adv_list_OSE)
466
467 length_PWKP = len(pp_count_PWKP)/2
468 class_smpl_PWKP = pd.Series(0, index=range(int(length_PWKP)))
469 class_adv_PWKP = pd.Series(1, index=range(int(length_PWKP)))
470 class_smpl_list_PWKP = class_smpl_PWKP.values.tolist()
471 class_adv_list_PWKP = class_adv_PWKP.values.tolist()
472 complexity_level_PWKP = (class_smpl_list_PWKP + class_adv_list_PWKP)
473
474 # list of all the features for each sentence
475 lines_OSE = [flesch_kincaid_OSE, flesch_OSE, smog_index_OSE, ↵
               gunning_fog_OSE,
476               automated_readability_OSE, dale_chall_readability_OSE, ↵
               spache_readability_OSE,
477               words_per_sentence_OSE, TTR_OSE, GTTR_OSE, CTTR_OSE, ↵
               unique_tokens_OSE, char_count_OSE, count_phonemes_OSE,
478               bi_grams_per_sentence_OSE, bi_grams_per_word_OSE, ↵
               tri_grams_per_sentence_OSE, tri_grams_per_word_OSE, ↵
               four_grams_per_sentence_OSE, four_grams_per_word_OSE,
479               verbs_OSE, nouns_OSE, adje_OSE, advrb_OSE, prn_OSE,
480               np_count_OSE, vp_count_OSE, pp_count_OSE, ↵
               content_word_incidence_OSE]

481
482 lines_PWKP = [flesch_kincaid_PWKP, flesch_PWKP, smog_index_PWKP, ↵
               gunning_fog_PWKP,
483               automated_readability_PWKP, dale_chall_readability_PWKP, ↵

```

```

        spache_readability_PWKP,
484     words_per_sentence_PWKP, TTR_PWKP, GTTR_PWKP, CTTR_PWKP, ↵
        unique_tokens_PWKP, char_count_PWKP, count_phonemes_PWKP,
485     bi_grams_per_sentence_PWKP, bi_grams_per_word_PWKP, ↵
        tri_grams_per_sentence_PWKP, tri_grams_per_word_PWKP, ↵
        four_grams_per_sentence_PWKP, four_grams_per_word_PWKP,
486     verbs_PWKP, nouns_PWKP, adje_PWKP, advrb_PWKP, prn_PWKP,
487     np_count_PWKP, vp_count_PWKP, pp_count_PWKP, ↵
        content_word_incidence_PWKP]

488
489
490
491
492
493 transposed_columns_OSE = list(map(list, zip(*lines_OSE)))
494 transposed_columns_PWKP = list(map(list, zip(*lines_PWKP)))
495
496 lists_df_OSE = pd.DataFrame(transposed_columns_OSE, columns=header).↵
    transpose()
497 lists_df_PWKP = pd.DataFrame(transposed_columns_PWKP, columns=header)↵
    .transpose()

498
499 # extract the features in a csv to possibly use for the paper
500 with open("Features_list_OSE.csv", 'w', newline='') as file:
501
502     writer = csv.writer(file)
503
504     # write each row to the CSV file
505     writer.writerow(header)
506     for row in lists_df_OSE:
507         writer.writerow(lists_df_OSE[row])
508
509 with open("Features_list_PWKP.csv", 'w', newline='') as file:
510
511     writer = csv.writer(file)
512
513     # write each row to the CSV file
514     writer.writerow(header)

```

```

515     for row in lists_df_PWKP:
516         writer.writerow(lists_df_PWKP[row])

```

```

518 X_OSE = lists_df_OSE.transpose()
519 y_OSE = complexity_level_OSE
520
521 # split the dataset into training and testing sets for the OSE Corpus
522 X_train_OSE, X_test_OSE, y_train_OSE, y_test_OSE = train_test_split(↵
    X_OSE, y_OSE, test_size=0.1, random_state=42)
523
524
525 X_PWKP = lists_df_PWKP.transpose()
526 y_PWKP = complexity_level_PWKP
527
528 # split the dataset into training and testing sets for the PWKP ↵
    Corpus
529 X_train_PWKP, X_test_PWKP, y_train_PWKP, y_test_PWKP = ↵
    train_test_split(X_PWKP, y_PWKP, test_size=0.3, random_state=42)

```

```

530 # create a Random Forest classifier with various hyperparameters
531 rf_clf_OSE = RandomForestClassifier(
532     n_estimators=100,                # number of trees
533     max_depth=10,                    # maximum depth of each tree
534     min_samples_split=8,             # minimum number of samples to split↵
        an internal node
535     min_samples_leaf=2,              # minimum number of samples at a ↵
        leaf node
536     max_features='log2',             # number of features to consider ↵
        when looking for the best split
537     bootstrap=True,                  # use bootstrap samples
538     criterion='gini',                # function to measure the quality of↵
        a split
539     random_state=112,                # seed for reproducibility
540     n_jobs=-1                        # use all processors for parallel ↵
        processing
541 )
542
543

```

```

544 # train the classifier on the training data
545 rf_clf_OSE.fit(X_train_OSE, y_train_OSE)
546
547
548 # make predictions on the testing data
549 y_pred_OSE = rf_clf_OSE.predict(X_test_OSE)
550
551 # calculate and print the accuracy of the model
552 accuracy_OSE = accuracy_score(y_test_OSE, y_pred_OSE)
553 print("The results for of the Random Forest Classifier for the OSE ←
      Corpus:")
554 print(f"    Accuracy: {accuracy_OSE * 100:.2f}%")
555
556 # calculate precision, recall, and F1-score
557 precision_OSE = precision_score(y_test_OSE, y_pred_OSE, average='←
      weighted')
558 recall_OSE = recall_score(y_test_OSE, y_pred_OSE, average='weighted')
559 f1_OSE = f1_score(y_test_OSE, y_pred_OSE, average='weighted')
560
561 print(f"    Precision: {precision_OSE * 100:.2f}%")
562 print(f"    Recall: {recall_OSE * 100:.2f}%")
563 print(f"    F1-Score: {f1_OSE * 100:.2f}%")
564
565 # generate a classification report
566 print("\nClassification Report:")
567 print(classification_report(y_test_OSE, y_pred_OSE))
568
569 # generate a confusion matrix
570 print("\nConfusion Matrix:")
571 conf_matrix_OSE = confusion_matrix(y_test_OSE, y_pred_OSE)
572 print(conf_matrix_OSE)

```

```

573 # initialize the DecisionTreeClassifier with some hyperparameters
574 clf_OSE = DecisionTreeClassifier(
575     criterion='log_loss',
576     splitter='best',
577     max_depth=4,
578     min_samples_split=2,

```

```

579     min_samples_leaf=1,
580     max_features='log2',
581     random_state=163
582 )
583
584 # train the model
585 clf_OSE.fit(X_train_OSE, y_train_OSE)
586
587 # make predictions
588 y_pred_OSE = clf_OSE.predict(X_test_OSE)
589
590 # evaluate the model
591 accuracy_OSE = accuracy_score(y_test_OSE, y_pred_OSE)
592 print("The results for of the Decision Tree Classifier for the OSE ↔
      Corpus:")
593 print(f"   Accuracy: {accuracy_OSE * 100:.2f}%")
594
595 # calculate precision, recall, and F1-score
596 precision_OSE = precision_score(y_test_OSE, y_pred_OSE, average='↔
      weighted')
597 recall_OSE = recall_score(y_test_OSE, y_pred_OSE, average='weighted')
598 f1_OSE = f1_score(y_test_OSE, y_pred_OSE, average='weighted')
599
600 print(f"   Precision: {precision_OSE * 100:.2f}%")
601 print(f"   Recall: {recall_OSE * 100:.2f}%")
602 print(f"   F1-Score: {f1_OSE * 100:.2f}%")
603
604 # generate a classification report
605 print("\nClassification Report:")
606 print(classification_report(y_test_OSE, y_pred_OSE))
607
608 # generate a confusion matrix
609 print("\nConfusion Matrix:")
610 conf_matrix_OSE = confusion_matrix(y_test_OSE, y_pred_OSE)
611 print(conf_matrix_OSE)

```

```

612 # create a rbf SVM classifier
613 svm_OSE = SVC(kernel='rbf', gamma = "scale", probability = False, ↔

```

```

        class_weight = "balanced")
614
615 # train the classifier
616 svm_OSE.fit(X_train_OSE, y_train_OSE)
617
618 # predict on the test set
619 y_pred_OSE = svm_OSE.predict(X_test_OSE)
620
621 # calculate and print the accuracy of the model
622 accuracy_OSE = accuracy_score(y_test_OSE, y_pred_OSE)
623 print("The results for of the SVC for the OSE Corpus:")
624 print(f"    Accuracy: {accuracy_OSE * 100:.2f}%")
625
626 # calculate precision, recall, and F1-score
627 precision_OSE = precision_score(y_test_OSE, y_pred_OSE, average='↵
    weighted')
628 recall_OSE = recall_score(y_test_OSE, y_pred_OSE, average='weighted')
629 f1_OSE = f1_score(y_test_OSE, y_pred_OSE, average='weighted')
630
631 print(f"    Precision: {precision_OSE * 100:.2f}%")
632 print(f"    Recall: {recall_OSE * 100:.2f}%")
633 print(f"    F1-Score: {f1_OSE * 100:.2f}%")
634
635 # generate a classification report
636 print("\nClassification Report:")
637 print(classification_report(y_test_OSE, y_pred_OSE))
638
639 # generate a confusion matrix
640 print("\nConfusion Matrix:")
641 conf_matrix_OSE = confusion_matrix(y_test_OSE, y_pred_OSE)
642 print(conf_matrix_OSE)

```

```

643 # create a Random Forest classifier with various hyperparameters
644 rf_clf_PWKP = RandomForestClassifier(
645     n_estimators=100,          # number of trees
646     max_depth=10,              # maximum depth of each tree
647     min_samples_split=8,       # minimum number of samples to split↵
        an internal node

```

```

648     min_samples_leaf=2,          # minimum number of samples at a ↵
        leaf node
649     max_features='log2 ',        # number of features to consider ↵
        when looking for the best split
650     bootstrap=True,              # use bootstrap samples
651     criterion='gini ',           # function to measure the quality of ↵
        a split
652     random_state=112,            # seed for reproducibility
653     n_jobs=-1                    # use all processors for parallel ↵
        processing
654 )
655
656
657 # train the classifier on the training data
658 rf_clf_PWKP.fit(X_train_PWKP, y_train_PWKP)
659
660
661 # make predictions on the testing data
662 y_pred_PWKP = rf_clf_PWKP.predict(X_test_PWKP)
663
664 # calculate and print the accuracy of the model
665 accuracy_PWKP = accuracy_score(y_test_PWKP, y_pred_PWKP)
666 print("The results for of the Random Forest Classifier for the PWKP ↵
        Corpus:")
667 print(f"    Accuracy: {accuracy_PWKP * 100:.2f}%")
668
669 # calculate precision, recall, and F1-score
670 precision_PWKP = precision_score(y_test_PWKP, y_pred_PWKP, average='↵
        weighted')
671 recall_PWKP = recall_score(y_test_PWKP, y_pred_PWKP, average='↵
        weighted')
672 f1_PWKP = f1_score(y_test_PWKP, y_pred_PWKP, average='weighted')
673
674 print(f"    Precision: {precision_PWKP * 100:.2f}%")
675 print(f"    Recall: {recall_PWKP * 100:.2f}%")
676 print(f"    F1-Score: {f1_PWKP * 100:.2f}%")
677
678 # generate a classification report

```

```

679 print("\nClassification Report:")
680 print(classification_report(y_test_PWKP, y_pred_PWKP))
681
682 # generate a confusion matrix
683 print("\nConfusion Matrix:")
684 conf_matrix_PWKP = confusion_matrix(y_test_PWKP, y_pred_PWKP)
685 print(conf_matrix_PWKP)

```

```

686 # initialize the DecisionTreeClassifier with some hyperparameters
687 clf_PWKP = DecisionTreeClassifier(
688     criterion='log_loss ',
689     splitter='best ',
690     max_depth=4,
691     min_samples_split=2,
692     min_samples_leaf=1,
693     max_features='log2 ',
694     random_state=163
695 )
696
697 # train the model
698 clf_PWKP.fit(X_train_PWKP, y_train_PWKP)
699
700 # make predictions
701 y_pred_PWKP = clf_PWKP.predict(X_test_PWKP)
702
703 # calculate and print the accuracy of the model
704 accuracy_PWKP = accuracy_score(y_test_PWKP, y_pred_PWKP)
705 print("The results for of the Decision Tree Classifier for the PWKP ←
      Corpus:")
706 print(f"    Accuracy: {accuracy_PWKP * 100:.2 f}%")
707
708 # calculate precision, recall, and F1-score
709 precision_PWKP = precision_score(y_test_PWKP, y_pred_PWKP, average='←
      weighted ')
710 recall_PWKP = recall_score(y_test_PWKP, y_pred_PWKP, average='←
      weighted ')
711 f1_PWKP = f1_score(y_test_PWKP, y_pred_PWKP, average='weighted ')
712

```

```

713 print(f" Precision: {precision_PWKP * 100:.2f}%")
714 print(f" Recall: {recall_PWKP * 100:.2f}%")
715 print(f" F1-Score: {f1_PWKP * 100:.2f}%")
716
717 # generate a classification report
718 print("\nClassification Report:")
719 print(classification_report(y_test_PWKP, y_pred_PWKP))
720
721 # generate a confusion matrix
722 print("\nConfusion Matrix:")
723 conf_matrix_PWKP = confusion_matrix(y_test_PWKP, y_pred_PWKP)
724 print(conf_matrix_PWKP)

```

```

725 # create a rbf SVM classifier
726 svm_PWKP = SVC(kernel='rbf', gamma = "scale", probability = False, ←
    class_weight = "balanced")
727
728 # train the classifier
729 svm_PWKP.fit(X_train_PWKP, y_train_PWKP)
730
731 # predict on the test set
732 y_pred_PWKP = svm_PWKP.predict(X_test_PWKP)
733
734 # calculate and print the accuracy of the model
735 accuracy_PWKP = accuracy_score(y_test_PWKP, y_pred_PWKP)
736 print("The results for of the Decision Tree Classifier for the PWKP ←
    Corpus:")
737 print(f" Accuracy: {accuracy_PWKP * 100:.2f}%")
738
739 # calculate precision, recall, and F1-score
740 precision_PWKP = precision_score(y_test_PWKP, y_pred_PWKP, average='←
    weighted')
741 recall_PWKP = recall_score(y_test_PWKP, y_pred_PWKP, average='←
    weighted')
742 f1_PWKP = f1_score(y_test_PWKP, y_pred_PWKP, average='weighted')
743
744 print(f" Precision: {precision_PWKP * 100:.2f}%")
745 print(f" Recall: {recall_PWKP * 100:.2f}%")

```

```
746 print(f" F1-Score: {f1_PWKP * 100:.2f}%")
747
748 # generate a classification report
749 print("\nClassification Report:")
750 print(classification_report(y_test_PWKP, y_pred_PWKP))
751
752 # generate a confusion matrix
753 print("\nConfusion Matrix:")
754 conf_matrix_PWKP = confusion_matrix(y_test_PWKP, y_pred_PWKP)
755 print(conf_matrix_PWKP)
```
