Τεχνητή Νοημοσύνη

Εργασία 2

Επιβλέπων Καθηγητής: Ίων Ανδρουτσόπουλος

Φοιτητές: Κοσμίδη Έλλη 3220086, Μουστακάτου Ελένη 3220130, Πρωικάκης Σπύρος 3220167

ΜΕΡΟΣ Α

Για την προεπεξεργασία του κειμένου χρησιμοποιούμε αρχικά δύο συναρτήσεις την load_imdb_data για την φόρτωση των δεδομένων και την clean_text η οποία κάνει κάποιες χρήσιμες μετατροπές στην μορφή των κειμένων. Αρχικά η load_imdb_data δέχεται ως παράμετρο το path στο οποίο βρίσκονται τα folder pos και neg που περιέχουν τα θετικά και τα αρνητικά reviews. Διατρέχει κάθε αρχείο σε αυτούς τους φακέλους(δηλαδή τα txt) και αποθηκεύει τα κείμενα αυτά στη λίστα texts. Επίσης αποθηκεύει στη λίστα labels τις αντίστοιχες ετικέτες των κειμένων με τους αριθμούς 1 και 0 για τις θετικές και τις αρνητικές κριτικές αντίστοιχα. Τέλος επιστρέφει τη λίστα texts και τη λίστα labels(σε μορφή numpy array για λόγους απόδοσης και συμβατότητας με τις βιβλιοθήκες που χρησιμοποιούμε). Η μέθοδος clean_text δέχεται ως παράμετρο ένα κείμενο και μετατρέπει τις λέξεις του σε πεζά γράμματα ώστε να μην μετράμε ίδιες λέξεις δύο φορές λόγω κεφαλαίων γραμμάτων. Επίσης αφαιρεί τα σημεία στίξης ώστε να μην μετράμε δύο φορές λέξεις οι οποίες έχουν ένα σημείο στίξης δίπλα τους(αν δε το κάναμε αυτό ο CountVectorizer θα μετρούσε για παράδειγμα ως διαφορετικές λέξεις τις hello και hello!). Μετά δίνουμε το κατάλληλο path και καλούμε την load_imdb_data. Για κάθε κείμενο στην λίστα texts που έχει επιστρέψει καλούμε την clean_text και έτσι όλα τα κείμενα έχουν μετατραπεί στη μορφή που θέλουμε. Προχωρώντας παρακάτω στην επεξεργασία των κειμένων χρησιμοποιούμε τον CountVectorizer από τη βιβλιοθήκη scikit-learn ώστε να μετατρέψουμε τα κείμενα σε διανύσματα 0 και 1 ανάλογα με το αν περιέχουν κάθε λέξη ή όχι(1 αν τη περιέχουν 0 αλλιώς). Δημιουργούμε με αυτό τον τρόπο έναν πίνακα ο οποίος έχει για γραμμές τα κείμενα και για στήλες τις λέξεις. Έπειτα υπολογίζουμε την συχνότητα κάθε λέξης ως το άθροισμα κάθε στήλης του πίνακα που δείχνει πόσα κείμενα περιέχουν τη συγκεκριμένη λέξη. Ακόμη, αποθηκεύουμε σε ένα λεξικό κάθε λέξη και δίπλα τη συχνότητα της. Αφού το κάνουμε αυτό μετά τις ταξινομούμε με φθίνουσα σειρά από τη λέξη με τη μεγαλύτερη συχνότητα

προς τη μικρότερη. Τυπώνουμε τις 70 πιο συχνές λέξεις για να δούμε τι είδους λέξεις είναι και παρατηρούμε ότι οι περισσότερες είναι λέξεις όπως the, and,this και γενικά stop-words τα οποία δεν έχουν κάποια άμεση σχέση με θετικές και αρνητικές κριτικές ταινιών αλλά απλά χρησιμοποιούνται πολύ λόγω της γλώσσας. Τυπώνοντας επίσης τις 200 πιο σπάνιες βλέπουμε ότι οι λέξεις αυτές είναι αρκετά ειδικές και δε χρησιμεύουν τόσο για το κέρδος πληροφορίας που θέλουμε να υπολογίσουμε. Χρησιμοποιώντας την έτοιμη υλοποίηση για το κέρδος πληροφορίας της scikit-learn υπολογίζουμε το κέρδος για κάθε λέξη από αυτές που έμειναν μετά την αφαίρεση των 70 πιο συχνών και 200 πιο σπάνιων. Από αυτές που έμειναν κρατάμε τις 10.000 πιο πολύτιμες και φτιάχνουμε τον τελικό πίνακα ο οποίος αποτελείται από τα κείμενα ως διανύσματα(οι γραμμές του πίνακα) και τις 10.000 πιο πολύτιμες λέξεις(οι στήλες του πίνακα). Τυπώνουμε τις εκατό πιο πολύτιμες και παρατηρούμε ότι όλες σχεδόν είναι επίθετα όπως bad,worst,excellent etc τα οποία σχετίζονται άμεσα με ένα θετικό ή αρντικό review για μια ταινία άρα και έχουν σημαντικό κέρδος πληροφορίας. Η διαλογή των k και n προήλθε μετά από δοκιμές με αριθμούς και prints ώστε να βλέπουμε ποιες λέξεις κατατάσσονται στις n πιο συχνές και k πιο σπάνιες χωρίς να αφαιρέσουμε λέξεις με σημαντικό πληροφοριακό κέρδος. Η επιλογή του m ως 10.000 έγινε με σκοπό να δημιουργηθεί ένα αποδοτικό λεξιλόγιο το οποίο θα χρησιμοποιείται για γρήγορη εκπαίδευση, θα απορρίπτει το θόρυβο μειώνοντας τον κίνδυνο υπερπροσαρμογής και θα καταναλώνει μικρό χώρο μνήμης.

1. ΑΛΓΟΡΙΘΜΟΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΝΑΪVE-BAYES

Ο αλγόριθμος Naïve-Bayes αποτελείται από δύο βασικά στάδια: **Εκπαίδευση**(υλοποιείται μέσω της μεθόδου fit) και **Πρόβλεψη**(υλοποιείται μέσω της μεθόδου predict). Στην κλάση Naïve-Bayes έχουμε ορίσει μια μέθοδο init η οποία δέχεται ως παράμετρο την εκτιμήτρια laplace (για την αποφυγή μηδενικών πιθανοτήτων) και αρχικοποιεί το λεξικό(dictionary) class_probs το οποίο χρησιμοποείται για την αποθήκευση της πιθανότητας κάθε κατηγορίας (συχνότητα εμφάνισης της κατηγορίας στο σύνολο εκπαίδευσης) και το λεξικό word_probs το οποίο αποθηκεύει την πιθανότητα κάθε λέξης δεδομένης μιας κατηγορίας. Έπειτα, έχουμε ορίσει τη μέθοδο fit η οποία δέχεται ως παραμέτρους ένα array(παράμετρος X) το οποίο έχει ως στήλες τις λέξεις του πίνακα(features) και ως γραμμές τα κείμενα που είναι επεξεργασμένα με τη μορφή διανυσμάτων ώστε να έχουν 0 ή 1 ανάλογα με το αν περιέχουν κάθε λέξη ή όχι. Επίσης δέχεται την παράμετρο y η οποία αντιπροσωπεύει τα labels(pos->1 και neg->0). Μετά με ένα for loop για κάθε μοναδική κατηγορία του y υπολογίζουμε την πιθανότητα κατηγορίας P(y) ως αριθμός κειμένων στην κατηγορία y προς συνολικό αριθμό κειμένων. Φτιάχνουμε ένα ξεχωριστό array για τα κείμενα κάθε κατηγορίας και έπειτα υπολογίζουμε την πιθανότητα κάθε λέξης δεδομένης μιας κατηγορίας δηλαδή το P(x|y) χρησιμοποιώντας και την εκτιμήτρια laplace. Η μέθοδος predict δέχεται ως παραμέτρους ένα array(παράμετρος X) και υπολογίζει για κάθε

κείμενο(γραμμές του X) την λογαριθμική πιθανότητα P(y|X) (χρησιμοποιούμε τη λογαριθμική ώστε να αποφύγουμε το underflow λόγω μικρών πιθανοτήτων) για κάθε κατηγορία. Τελικά επιλέγεται η κατηγορία με τη μεγαλύτερη πιθανότητα για κάθε κείμενο και προσθέτουμε την πρόβλεψη σε ένα array predictions στο οποίο αποθηκεύεται η πρόβλεψη για κάθε κείμενο. Για την χρήση του αφελή ταξινομητή Bayes στα δεδομένα μας, τα χωρίζουμε σε train,test,development (70%-20%-10%) και τρέχουμε επαναληπτικά το fit και το predict για το 10%,20%,30%...100% των δεδομένων του train . Σε κάθε επανάληψη υπολογίζουμε τα precision,recall,f1 και τα αποθηκεύουμε σε αντίστοιχους πίνακες. Σχεδιάζουμε με matplot τους παραπάνω πίνακες. Από το διάγραμμα του Precision βλέπουμε το ποσοστό σε κάθε στάδιο εκπαίδευσης από τα δείγματα που προέβλεψε ότι ανήκουν στην θετική κατηγορία και άνηκε πραγματικά σε αυτήν την κατηγορία. Καταλήγει να είναι ~91% για το training set και ~90.5% για το development set. Από το διάγραμμα του Recall βλέπουμε το ποσοστό σε κάθε στάδιο εκπαίδευσης των θετικών δειγμάτων που κατέταξε σωστά. Καταλήγει να είναι ~87% για το training set και ~86% για το development set. Στο διάγραμμα του F1 φαίνεται η απόδοση του μοντέλου (εξαρτάται από τα precision και recall). Καταλήγει να είναι ~89% για το training set και ~88% για το development set.

2. ΛΟΓΙΣΤΙΚΗ ΠΑΛΙΝΔΡΟΜΙΣΗ

Η Λογιστική Παλινδρόμηση χρησιμοποιείται για την ταξινόμηση των δεδομένων σε δύο κατηγορίες (0 ή 1) . Αυτό το πετυχαίνουμε υπολογίζοντας αρχικά τον γραμμικό συνδυασμό z= w1x1 + w2x2+..wnxn +b των χαρακτηριστικών μια των βαρών του μοντέλου όπου στο παράδειγμά μας τα x είναι οι διάφορες λέξεις που εμφανίζονται σε μια κριτική και στην κάθε λέξη ανάλογα με την σημαντικότητα της εκχωρούμε ένα βάρος. Σαν συνάρτηση ενεργοποίησης χρησιμοποιούμε την σιγμοειδή (σ(z)) . Αν σ(z)>=0.5 τότε κατατάσσουμε το μοντέλο στην κατηγορία 1 αλλιώς στην 0. Την ενημέρωση των βαρών την υλοποιούμε με στοχαστική ανάβαση κλίσης, η οποία προσπαθεί να βρει τα βάρη που ελαχιστοποιούν το κόστος και βελτιώνουν την ακρίβεια των προβλέψεων. Στα παραπάνω εφαρμόζουμε και ομαλοποίηση λ= 0.001 ώστε να αποφεύγεται το overfitting (επιλέξαμε το λ σύμφωνα με τα validation loss).

Τα παραπάνω υλοποιούνται με κώδικα ως εξής:

Υλοποιούμε κλάση LogisticRegrationSGA η οποία κωδικοποιεί την Λογιστική Παλινδρόμηση με Στοχαστική Ανάβαση Κλίσης. Αρχικοποιούμε ρυθμό μάθησης (lr), τις εποχές (epochs) και το λ (l) σύμφωνα με αυτά που έδωσε ο χρήστης. Η μέθοδος **sigmoid** δέχεται το z και υπολογίζει την πιθανότητα σύμφωνα με την σιγμοειδή συνάρτηση. Η μέθοδος **cross_entropy_loss** δέχεται τα δεδομένα (X) και τις κατηγορίες (y) τους και παράγει το κόστος κάθε εποχής σύμφωνα με τους τύπους των διαλέξεων. Εφαρμόζει και ομαλοποίηση. Η μέθοδος **fit** καλείται για την εκπαίδευση του μοντέλου πάνω στα δεδομένα. Δέχεται σαν παραμέτρους τα δεδομένα (X) και τις κατηγορίες τους

(y) και σε κάθε εποχή αλλάζει ένα προς ένα τα βάρη κάθε δείγματος σύμφωνα με τον τύπο της στοχαστικής ανάβασης κλίσης και της ομαλοποίησης. Ουσιαστικά η **fit** σύμφωνα με τα δεδομένα εκπαίδευσης φτιάχνει τα βάρη ώστε να προβλέπουν όσο το δυνατόν καλύτερα τις κατηγορίες νέων δειγμάτων. Στο τέλος των εποχών ανά δύο εκτυπώνουμε το κόστος το οποίο σταδιακά μειώνεται. Αυτό σημαίνει ότι το μοντέλο βελτιώνεται και μαθαίνει τα δεδομένα. Η μέθοδος **predict** δέχεται δείγματα (X) και μέσω της σιγμοειδούς συνάρτησης επιστρέφει την κατηγορία κάθε δείγματος.

Για την χρήση της Λογιστικής Παλινδρόμησης στα δεδομένα μας, τα χωρίζουμε σε train,test,development (70%-20%-10%) και τρέχουμε επαναληπτικά το fit και το predict για το 10%,20%,30%...100% των δεδομένων του train . Σε κάθε επανάληψη υπολογίζουμε τα precision,recall,f1 και τα αποθηκεύουμε σε αντίστοιχους πίνακες. Σχεδιάζουμε με matplot τους παραπάνω πίνακες. Από το διάγραμμα του Precision βλέπουμε το ποσοστό σε κάθε στάδιο εκπαίδευσης από τα δείγματα που προέβλεψε ότι ανήκουν στην θετική κατηγορία και άνηκε πραγματικά σε αυτήν την κατηγορία. Καταλήγει να είναι ~88% για το training set και ~84% για το development set. Από το διάγραμμα του Recall βλέπουμε το ποσοστό σε κάθε στάδιο εκπαίδευσης των θετικών δειγμάτων που κατέταξε σωστά. Καταλήγει να είναι ~91% για το training set και ~87.5% για το development set. Στο διάγραμμα του F1 φαίνεται η απόδοση του μοντέλου (εξαρτάται από τα precision και recall). Καταλήγει να είναι ~89.5% για το training set και ~86% για το development set.

3. ΤΥΧΑΙΟ ΔΑΣΟΣ

Το Τυχαίο Δάσος είναι μια μέθοδος ταξινόμησης δεδομένων σε κατηγορίες. Χρησιμοποιεί πολλά δέντρα απόφασης ID3 , τα οποία τρέχουν σε παραλλαγές του αρχικού σετ δεδομένων, αποφασίζουν σε ποια κατηγορία θα καταταγούν τα νέα δείγματα και στο τέλος συνδυάζουμε τις προβλέψεις και κατατάσσουμε τα κάθε νέο δεδομένο στην κατηγορία που λέει η πλειοψηφία. Για να αποφασίσει το κάθε δέντρο την κατηγορία, στο στάδιο της εκπαίδευσης τα δέντρα αποφασίζουν σύμφωνα με ποια χαρακτηριστικά θα κατατάσσουν τα δεδομένα (η σειρά επιλογής των χαρακτηριστικών γίνεται σύμφωνα με το κέρδος πληροφορίας, στην περίπτωσή μας επιλέγουμε πρώτα τις λέξεις με το μεγαλύτερο κέρδος πληροφορίας) .

Τα παραπάνω υλοποιούνται με κώδικα ως εξής:

Η κλάση RandomForest υλοποιεί το Τυχαίο Δάσος. Αρχικοποιούμε πόσα ID3 δέντρα θα χρησιμοποιεί (n_estimators) και το μέγιστο βάθος των ID3 δέντρων σύμφωνα με αυτά που δίνει ο χρήστης. Η μέθοδος **fit** δέχεται τα δεδομένα X και τις κατηγορίες τους y και φτιάχνει στο στάδιο της εκπαίδευσης τα ID3 δέντρα με βάση τα χαρακτηριστικά των δεδομένων, καλώντας την fit της έτοιμης κλάσης DecisionTreeClassifier με παράμετρο criterion="entropy" ώστε να λειτουργεί ως ID3. Κάθε δέντρο το φτιάχνει πάνω σε

διαφορετικά τυχαιοποιημένα δεδομένα που προκύπτουν από τα αρχικά δεδομένα. Η μέθοδος **predict** δέχεται τα δείγματα που θέλει να κατατάξει, καλεί την predict για κάθε δέντρο και στο τέλος συνδυάζει τις προβλέψεις τους για κάθε δείγμα και διαλέγει την κατηγορία που εμφανίζεται πιο πολλές φορές.

Για την χρήση του Random Forest στα δεδομένα μας, τα χωρίζουμε σε train,test,development (70%-20%-10%) , αρχικοποιούμε τον αριθμό των δέντρων και το βάθος και τρέχουμε επαναληπτικά το fit και το predict για το 10%,20%,30%...100% των δεδομένων του train . Σε κάθε επανάληψη υπολογίζουμε τα precision,recall,f1 και τα αποθηκεύουμε σε αντίστοιχους πίνακες. Σχεδιάζουμε με matplot τους παραπάνω πίνακες.

ΜΕΡΟΣ Β

1. Αφελής ταξινομητής Bayes

Η υλοποίηση μας:

	precision	recall	f1-score	support
0	0.860983	0.902213	0.881116	2485.000
1	0.898581	0.856064	0.876807	2515.000
accuracy	0.879000	0.879000	0.879000	0.879
macro avg	0.879782	0.879138	0.878962	5000.000
weighted avg	0.879895	0.879000	0.878949	5000.000
micro avg	0.879000	0.879000	0.879000	5000.000

Διαθέσιμη υλοποίηση του Scikit-learn:

	precision	recall	f1-score	support
0	0.867670	0.905030	0.885956	2485.0000
1	0.901993	0.863618	0.882389	2515.0000
accuracy	0.884200	0.884200	0.884200	0.8842
macro avg	0.884832	0.884324	0.884173	5000.0000
weighted avg	0.884935	0.884200	0.884162	5000.0000
micro avg	0.884200	0.884200	0.884200	5000.0000

Precision: Η διαθέσιμη υλοποίηση του Scikit-learn υπερτερεί ελαφρώς σε όλες τις κατηγορίες, με διαφορά 0.5%-1% στο precision. Η διαφορά είναι μικρή, αλλά δείχνει ότι το Scikit-learn έχει βελτιστοποιημένους υπολογισμούς πιθανοτήτων.

Recall: Το recall της Scikit-learn υλοποίησης είναι ελαφρώς καλύτερο και στις δύο κλάσεις. Η δική μας υλοποίηση παρουσιάζει χαμηλότερο recall στην κλάση 1 (-0.7%), δείχνοντας ότι μπορεί να χάνει κάποια θετικά δείγματα

F1-Score: Το F1-score είναι υψηλότερο στο Scikit-learn και στις δύο κλάσεις, αλλά η διαφορά είναι μικρή. Η δική μας υλοποίηση είναι πολύ κοντά στην απόδοση της Scikit-learn, με διαφορά μόνο 0.5%-1%.

Accuracy: Η Scikit-learn υλοποίηση έχει 0.5% υψηλότερη ακρίβεια. Η διαφορά είναι μικρή, αλλά δείχνει ότι η βελτιστοποίηση των πιθανοτήτων στο Scikit-learn είναι πιο αποδοτική.

Συμπέρασμα: Η Scikit-learn υλοποίηση είναι ελαφρώς ανώτερη σε Precision, Recall, F1-score και Accuracy, αλλά η διαφορά είναι πολύ μικρή (0.5% - 1%). Η δική μας υλοποίηση προσεγγίζει αρκετά την απόδοση του Scikit-learn, γεγονός που δείχνει ότι η πιθανότητα υπερβολικών σφαλμάτων είναι μικρή. Εν ολίγοις, η υλοποίησή μας αποδίδει πολύ καλά σε σχέση με το Scikit-learn.

2. Λογιστική Παλινδρόμηση

Η υλοποίηση μας:

	precision	recall	f1-score	support
0	0.883299	0.861972	0.872505	2485.0000
1	0.866796	0.887475	0.877014	2515.0000
accuracy	0.874800	0.874800	0.874800	0.8748
macro avg	0.875048	0.874723	0.874759	5000.0000
weighted avg	0.874998	0.874800	0.874773	5000.0000
micro avg	0.874800	0.874800	0.874800	5000.0000

Διαθέσιμη υλοποίηση του Scikit-learn:

	precision	recall	f1-score	support
0	0.893176	0.884909	0.889024	2485.0000
1	0.887313	0.895427	0.891352	2515.0000
accuracy	0.890200	0.890200	0.890200	0.8902
macro avg	0.890245	0.890168	0.890188	5000.0000
weighted avg	0.890227	0.890200	0.890195	5000.0000
micro avg	0.890200	0.890200	0.890200	5000.0000

Precision: Η διαθέσιμη υλοποίηση του Scikit-learn έχει υψηλότερο Precision σε όλες τις κατηγορίες, με διαφορά περίπου 1%. Η δική μας υλοποίηση εμφανίζει ελαφρώς χαμηλότερη ακρίβεια στον εντοπισμό των σωστών θετικών προβλέψεων.

Recall: Το recall της Scikit-learn υλοποίησης είναι ελαφρώς καλύτερο σε όλες τις κατηγορίες. Η διαφορά είναι μικρή αλλά σταθερή, δείχνοντας ότι το Scikit-learn μπορεί να αναγνωρίσει περισσότερα σωστά θετικά δείγματα.

F1-Score: Το F1-score είναι υψηλότερο στην υλοποίηση Scikit-learn, υποδεικνύοντας ότι επιτυγχάνει καλύτερη ισορροπία μεταξύ Precision & Recall. Η δική μας υλοποίηση υπολείπεται κατά περίπου 1-1.5%, αλλά βρίσκεται κοντά στο επιθυμητό επίπεδο.

Accuracy: Η Scikit-learn υλοποίηση έχει 1.5% υψηλότερη ακρίβεια. Η διαφορά δείχνει ότι η εκπαίδευση της Scikit-learn εκτιμά καλύτερα τις παραμέτρους του μοντέλου, οδηγώντας σε πιο ακριβείς προβλέψεις.

Συμπέρασμα: Η Scikit-learn υλοποίηση είναι πιο αποδοτική, επιτυγχάνοντας καλύτερο Precision, Recall, F1-score και Accuracy. Η διαφορά κυμαίνεται μεταξύ 1% - 1.5%, γεγονός που δείχνει ότι η δική μας υλοποίηση είναι αρκετά κοντά αλλά δεν φτάνει την ακρίβεια του Scikit-learn. Εν ολίγοις, η υλοποίηση του Scikit-learn αποδίδει καλύτερα, αλλά η δική μας βρίσκεται σε ικανοποιητικό επίπεδο

3. Τυχαίο Δάσος

Η υλοποίηση μας:

	precision	recall	f1-score	support
0	0.788180	0.606439	0.685467	2485.0000
1	0.683290	0.838966	0.753168	2515.0000
accuracy	0.723400	0.723400	0.723400	0.7234
macro avg	0.735735	0.722702	0.719318	5000.0000
weighted avg	0.735420	0.723400	0.719521	5000.0000
micro avg	0.723400	0.723400	0.723400	5000.0000

Διαθέσιμη υλοποίηση του Scikit-learn:

	precision	recall	f1-score	support
0	0.852766	0.806439	0.828956	2485.0000
1	0.818491	0.862425	0.839884	2515.0000
accuracy	0.834600	0.834600	0.834600	0.8346
macro avg	0.835628	0.834432	0.834420	5000.0000
weighted avg	0.835525	0.834600	0.834452	5000.0000
micro avg	0.834600	0.834600	0.834600	5000.0000

Precision: Η υλοποίηση του Scikit-learn έχει υψηλότερο Precision σε όλες τις κατηγορίες. Η δική μας υλοποίηση εμφανίζει ιδιαίτερα χαμηλό Precision για την κλάση 1 (0.6833 vs 0.8185), κάτι που δείχνει περισσότερες λανθασμένες θετικές προβλέψεις.

Recall: Το recall της Scikit-learn υλοποίησης είναι υψηλότερο, ειδικά για την κλάση 0 (0.6064 vs 0.8064). Η δική μας υλοποίηση δυσκολεύεται να εντοπίσει σωστά τα δείγματα της κλάσης 0, γεγονός που υποδηλώνει χαμηλή γενίκευση του μοντέλου.

F1-Score: Το F1-score είναι υψηλότερο στην υλοποίηση Scikit-learn, που σημαίνει ότι ισορροπεί καλύτερα Precision & Recall. Η δική μας υλοποίηση υπολείπεται σημαντικά, ιδιαίτερα για την κλάση 0 όπου η διαφορά φτάνει περίπου 14%.

Accuracy: Η Scikit-learn υλοποίηση έχει 11% υψηλότερη ακρίβεια. Η διαφορά αυτή είναι πολύ μεγάλη και δείχνει ότι η Scikit-learn προσφέρει μια πολύ πιο αποτελεσματική εκδοχή του Random Forest, ενώ η δική μας υλοποίηση δεν αποδίδει το ίδιο καλά.

Συμπέρασμα: Η Scikit-learn υλοποίηση είναι ανώτερη, καθώς εμφανίζει σημαντικά υψηλότερο Precision, Recall, F1-score και Accuracy. Η διαφορά στις μετρικές, υποδηλώνει ότι η δική μας υλοποίηση έχει αδυναμίες, ιδιαίτερα στην ταξινόμηση της κλάσης 0.

ΜΕΡΟΣ Γ

Για την υλοποίηση του στοιβαγμένου διπλής κατεύθυνσης RNN (stacked bidirectional RNN) με κελιά LSTM ή GRU και global max pooling σε PyTorch με τον Adam optimizer, χρησιμοποιήσαμε τον κώδικα του φροντηστηρίου κάνοντας κάποιες μικρές αλλαγές. Συγκεκριμένα προσθέσαμε αριθμό από Layers (num_layers) ώστε να λειτουργεί ως στοιβαγμένο και ορίσαμε default τιμή num_layers=2 μετά από δοκιμές στα δεδομένα εκπαίδευσης. Επίσης ορίσαμε default τιμή στο use_pooling=True , ώστε να λειτουργεί εξαρχής με max pooling. Σύμφωνα με τον κώδικα του φροντιστηρίου στην αρχή μετασχηματίζουμε το λεξιλόγιο, ώστε σε κάθε μια να αντιστοιχίσουμε έναν αριθμό. Πριν από κάθε λέξη αφήνουμε 2 αριθμούς κενούς, ένα για το PAD (padding: το χρησιμοποιούμε για να φτιάξουμε τις προτάσεις να έχουν ίδιο μήκος) και το UNK (unknown: επειδή χρησιμοποιούμε τις m πιο συνήθεις λέξεις, κάποιες λέξεις της πρότασης μπορεί να μην ξέρει να τις διαχειριστεί το RNN, οπότε τις αντικαθιστούμε με το UNK και εκχωρούμε το ίδιο embedding στα UNK). Η μέθοδος **tokenize** της κλάσης TextData μετατρέπει κάθε πρόταση σε λίστα με αριθμούς για κάθε λέξη (εφαρμόζοντας PAD και UNK όπου χρειάζεται). Η μέθοδος **forward** μετατρέπει τις εισόδους σε embeddings και παράγει την έξοδο σύμφωνα με το τι τύπου RNN έχουμε. Τέλος εφαρμόζει pooling αν χρειάζεται. Η μέθοδος **train_model** εκπαιδεύει το μοντέλο (μηδενίζει κλίσεις, περνάει δεδομένα σε GPU, εκτελεί την forward και υπολογίζει κόστη και απώλειες).

ΣΥΓΚΡΙΣΗ ΕΠΙΔΟΣΕΩΝ ΜΕΡΟΥΣ Γ ΜΕ ΜΕΡΗ Α ΚΑΙ Β

ΜΕ ΜΕΡΟΣ Α

ΜΕ ΝΑΪVE **BAYES**

Accuracy

- Naïve Bayes: 0.8790
- GRU: 0.8724
- LSTM: 0.8768
- **Νικητής:** Ο Naïve Bayes έχει ελαφρώς καλύτερη συνολική ακρίβεια από τα RNNs.

Macro Precision / Recall / F1

- Naïve Bayes: ~0.879
- GRU: ~0.872
- LSTM**:** ~0.877

Παρατήρηση: Ο Naïve Bayes έχει ελαφρώς καλύτερα macro metrics από τα RNNs.

Recall ανά κλάση

- Στην κλάση **0**, ο Naïve Bayes έχει υψηλότερο recall (0.902**)** από GRU & LSTM, άρα ανιχνεύει περισσότερα αληθινά δείγματα της κλάσης αυτής. Στην κλάση **1**, τα RNNs έχουν λίγο υψηλότερο recall από τον Naïve Bayes.

ΜΕ ΛΟΛΙΣΤΙΚΗ ΠΑΛΙΝΔΡΟΜΗΣΗ

Η λογιστική παλινδρόμηση (Logistic Regression) έχει **accuracy = 87.48%**, το οποίο είναι: Καλύτερο από το GRU (87.24%) και χειρότερο από το LSTM (87.68%).

Precision & Recall

- Για την κατηγορία **0 (αρνητικές κριτικές)**:
  - **Precision = 88.33%**, **Recall = 86.19%**
  - Για την κατηγορία **1 (θετικές κριτικές)**:
  - **Precision = 86.68%**, **Recall = 88.75%**

Οι τιμές είναι αρκετά ισορροπημένες, δείχνοντας ότι το μοντέλο δεν ευνοεί κάποια κατηγορία. Σ σύγκριση με το GRU έχει μεγαλύτερο precision και μικρότερο recall στην κατηγορία 0 ενώ στην κατηγορία 1 εμφανίζει μικρότερο precision και μεγαλύτερο recall. Σε σύγκριση με το LSTM εμφανίζει μικρότερο precision και μεγαλύτερο recall στην κατηγορία 0 ενώ στην κατηγορία 1 εμφανίζει μεγαλύτερο precision και μικρότερ recall.

Συμπέρασμα: Οι συγκεκριμένες υλοποιήσεις έχουν πολύ μικρές διαφορές στις μετρικές τους με αποτέλεσμα καμία να μην υπερέχει συντριπτικά της άλλης. Η υλοποίηση που θα διαλέξει κάποιος εξαρτάται από τη δική του προτίμηση(π.χ χρησιμοποιούμε λογιστική παλινδρόμηση αν θέλουμε χαμηλό υπολογιστικό κόστος).

ΜΕ ΤΥΧΑΙΟ ΔΑΣΟΣ

Τα GRU & LSTM υπερέχουν σημαντικά σε όλα τα metrics (Accuracy, Precision, Recall, F1-score). Ο Random Forest εμφανίζει σημαντικά χαμηλότερη ακρίβεια (–15% από τα RNNs), γεγονός που δείχνει ότι η υλοποίηση αυτή δεν είναι ιδανική για τη συγκεκριμένη περίπτωση.

ΜΕ ΜΕΡΟΣ Β

Με έτοιμη υλοποίηση naïve-bayes:

Ο Naïve Bayes υπερέχει ελαφρώς από τα RNNs όσον αφορά το **Accuracy**, έχοντας **+1% περίπου** καλύτερη επίδοση. Οι διαφορές στις υπόλοιπες μετρικές είναι επίσης κοντά με μια μικρή υπεροχή του Naïve-Bayes ο οποίος φαίνεται να επωφελείται από τη φύση του προβλήματος το οποίο έχει σχετικά απλές δομές που μπορούν να αναλυθούν με μια στατιστική προσέγγιση και δε χρειάζεται απαραίτητα κάτι πιο περίπλοκο όπως RNNs.

Με έτοιμη υλοποίηση λογιστκής παλινδρόμησης με στοχαστική ανάβαση

Η Λογιστική Παλινδρόμηση (Logistic Regression) εμφανίζει την καλύτερη ακρίβεια (Accuracy) με 89.02%, ξεπερνώντας τόσο το GRU όσο και το LSTM κατά 1.3-1.8%. Η **Λογιστική Παλινδρόμηση πιθανώς επωφελείται από τη γραμμικότητα του προβλήματος**, δηλαδή τα χαρακτηριστικά που χρησιμοποιήθηκαν(λέξεις με υψηλό πληροφοριακό κέρδος) μάλλον έχουν υψηλή διακριτική ικανότητα, επιτρέποντας στη Logistic Regression να αποδώσει εξαιρετικά.

Με έτοιμη υλοποίηση Random Forest

Τα RNN μοντέλα (GRU & LSTM) έχουν σαφώς καλύτερη απόδοση από το Random Forest (+3.7% έως +4.2% στο accuracy) . Επίσης έχουν καλύτερες επιδόσεις στα precision και recall για τις δύο κατηγορίες από την υλοποίηση του τυχαίου δάσους.