**2Η ΕΡΓΑΣΙΑ ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ**

**ΝΙΚΟΛΕΤΑ-ΚΛΕΙΩ ΠΑΤΑΤΣΗ 3180266**

**ΝΤΕΝΙΣ ΝΤΙΑΜΑΝΤΙΣ 3150129**

**ΠΛΑΤΩΝ ΚΑΡΑΓΕΩΡΓΗΣ 3180068**

**ID3**

Ο πρώτος αλγόριθμος που δημιουργήσαμε είναι ο ID3. Αρχικά ψάξαμε τον τρόπο με τον οποίο έπρεπε να διαβάσουμε τα αρχεία έτσι ώστε μελλοντικά να αξιοποιήσουμε τις πληροφορίες τους. Φτιάξαμε μία μέθοδο loadAll() η οποία έχει καθήκον να φορτώνει τα αρχεία ανάλογα με το όρισμα που της δίνουμε στη main. Η loadAll() αφού διαβάσει το κάθε ένα παράδειγμα ξεχωριστά καλεί τη μέθοδο words() που σπάει το κάθε παράδειγμα σε λέξεις και σημεία στίξης. Αυτά, αποθηκεύονται σε 2 πίνακες HashMap masterneg, masterpos όπου ως κλειδί έχουν τη λέξη-ιδιότητα και σαν τιμή τον αριθμό που εμφανίζεται η κάθε λέξη στα θετικά και στα αρνητικά παραδείγματα αντίστοιχα. Έπειτα, καταχωρούμε τα παραδείγματα στο HashMap example και διαχωρίζουμε μέσω των τιμών 0,1 αν είναι κάθε παράδειγμα θετικό ή αρνητικό. Ύστερα, φτιάχνουμε μία μέθοδο getAttributes() η οποία συγχωνεύει τους πίνακες masterneg, masterpos σε έναν πίνακα master τον οποίο τον μετατρέπουμε μέσω της μεθόδου getHashMapKey() προσωρινά σε set με στόχο να τον ταξινομήσουμε αλλά και να αποθηκεύσουμε τα κλειδιά του σε ένα ArrayList. Στο τέλος αυτής της μεθόδου έχουμε ένα arraylist attributes το οποίο έχει αποθηκευμένες όλες τις λέξεις στη σειρά ανάλογα με τη συχνότητα που εμφανίστηκαν στους πίνακες. Στη συνέχεια, καλούμε τη μέθοδο ID3 που είναι αναδρομική, με ορίσματα που βασίζονται στις διαφάνειες του μαθήματος. Δίνουμε σαν όρισμα τα αρχεία-παραδείγματα χωρίς επεξεργασία, το arraylist attributes που κατασκευάσαμε προηγουμένως και μία προεπιλεγμένη τιμή 0 για να ξεκινήσει ο αλγόριθμος. Αυτή η τιμή στην πορεία θα καθορίζεται αν είναι 0 ή 1 μέσω της μεθόδου findMajority() η οποία κρίνει αν εμφανίζεται περισσότερες φορές στα θετικά ή στα αρνητικά παραδείγματα και παίρνει τιμή ανάλογα. Μέσα στον ID3, φτιάχνουμε τα if statements που ελέγχουν αν τελείωσαν τα attributes τα οποία μειώνονται προφανώς κατά 1 σε κάθε loop και την περίπτωση που το αρχείο που διαβάζουμε έχει μόνο λέξεις που ανήκουν σε μία από τις δύο κατηγορίες, δηλαδή μόνο θετικές ή μόνο αρνητικές. Έπειτα πάμε στο else που γίνεται και όλη η δουλειά του αλγορίθμου. Αρχικά μέσω της calculateEntropy() η οποία καλείται μέσα στο else, υπολογίζουμε το καλύτερο attribute στο arraylist που έχει όλα τα attributes μέσα. Δηλαδή τη λέξη ή σημείο στίξης που εμφανίζεται τις περισσότερες φορές. Για την εντροπία χρησιμοποιούμε τους τύπους που υπάρχουν στις διαφάνειες του μαθήματος. Από την calculateEntropy καλούμε και κάποιες βοηθητικές μεθόδους. Αρχικά, η calculateHasWord με όρισμα ένα HashMap, μία λέξη και ένα boolean flag, εάν είναι θετικό το flag υπολογίζει την πιθανότητα το μήνυμα να είναι θετικό εάν περιέχει τη λέξη. Αντίθετα, εάν το flag είναι αρνητικό υπολογίζει την πιθανότητα το μήνυμα να είναι θετικό εάν δεν περιέχει τη λέξη. Επιπλέον, η separate που παίρνει ένα HashMap και μία λέξη, υπολογίζει την πιθανότητα να υπάρχει αυτή η λέξη σε κάποιο παράδειγμα. Τέλος, την calculatePossibility που δέχεται ένα HashMap και υπολογίζει την πιθανότητα των μηνυμάτων που είναι θετικά. Μόλις τη βρούμε τη λέξη, την αποθηκεύουμε σε ένα String best και χωρίζουμε τα αρχεία-παραδείγματα σε 2 κατηγορίες, σε αυτά που την περιέχουν και σε αυτά που δεν την περιέχουν. Αντίστοιχα φτιάχνουμε ένα treemap το οποίο έχει σαν κλειδί τη λέξη συν 0 αν την περιέχει, αλλιώς συν 1 και σαν τιμή μία λίστα με τα παραδείγματα που την περιέχουν ή όχι αντίστοιχα. Ταυτόχρονα έχουμε ένα παράλληλο treemapBETA το οποίο αποθηκεύει την πιθανότητα των συνόλων σε κάθε αντίστοιχο κόμβο να είναι θετικά. Μετά χωρίζουμε τα παραδείγματα σε αυτά που περιέχουν τη λέξη και σε αυτά που δεν την περιέχουν και ξανακαλούμε τον ID3 με αυτά. Όταν τελειώσει η αναδρομική κλήση του ID3 έχουμε στο treemapBETA την πιθανότητα του κάθε κόμβου να περιέχει θετικά παραδείγματα ανάλογα με το αν περιείχε τη λέξη ή όχι. Μετά αφού τελείωσε το training, φορτώνουμε τα test files. Έχοντας τα best attributes κατανεμημένα ελέγχουμε εάν υπάρχουν στο κάθε παράδειγμα και όταν φτάσει σε έναν κόμβο του δέντρου με συγκεκριμένη πιθανότητα τότε αποφασίζουμε αν είναι θετικό ή αρνητικό και κόβουμε τα υπόλοιπα κλαδιά. Έπειτα, συγκρίνουμε την απόφαση του αλγορίθμου με την πραγματική τιμή και αν κατατάξαμε το αρχείο σωστά, αυξάνεται ο μετρητής που φτιάχνουμε και από αυτόν μετέπειτα θα προκύψει το accuracy. Επίσης, το threshold που χρειαζόμασταν για να φτιάξουμε τα σχήματα ήταν η πιθανότητα με την οποία κατατάσσαμε τα παραδείγματα.

**RandomForest**

Ο δεύτερος αλγόριθμος που υλοποιήσαμε είναι ο RandomForest που βασίζεται πάνω στον ID3 που φτιάξαμε παραπάνω, με τη διαφορά ότι η testID3 επιστρέφει ένα πίνακα με τις αποφάσεις του, δηλαδή 0 ή 1. Ουσιαστικά η γενική ιδέα είναι ότι θα φτιάξουμε 5 ID3 που αργότερα θα «συναντηθούν» στην κλάση Forest και θα συγκεντρώσουμε τα αποτελέσματα τους. Με βάση την πλειοψηφία, θα κρίνουμε αν ένα παράδειγμα είναι θετικό ή όχι. Στη main καλούμε την trainForest και έπειτα την testForest. Η trainForest καλεί τον ID3 και διαβάζει τα trainFiles τα οποία τα αποθηκεύει σε ένα HashMap, από αυτό παίρνουμε τα κλειδιά και τα αποθηκεύουμε σε ένα ArrayList μέσω της getHashMapKey. Ύστερα, αποθηκεύουμε τα attributes του αρχικού συνόλου σε ένα ArrayList. Αυτά τα κάνουμε, γιατί στην Forest πιο κάτω θα τα χρησιμοποιήσουμε, για να κάνουμε shuffle τα ArrayLists που κατασκευάσαμε, για τη δημιουργία των τυχαίων υποσυνόλων. Μέσα στην επανάληψη δημιουργούμε το temphash που είναι HashMap και το tempattr που είναι ArrayList τα οποία αρχικοποιούνται με τα αποτελέσματα των shuffle και είναι υποσύνολα των αρχικών πινάκων. Αυτά, είναι τα ορίσματα που δίνουμε στον κάθε ID3 τον οποίο αποθηκεύουμε σε ένα ArrayList<ID3> Forest, αφού φορτώσουμε τα test files σε κάθε ID3. Στη συνέχεια, προχωράμε στη μέθοδο testForest όπου συγκρίνουμε τις αποφάσεις του κάθε ID3 για κάθε παράδειγμα και αν η πλειοψηφία συμφωνεί με την πραγματική τιμή του παραδείγματος, αυξάνουμε τον μετρητή που μας καθορίζει το accuracy. Την υπερπαράμετρο για το πλήθος των δέντρων ID3 την ορίζουμε 5. Δοκιμάσαμε επίσης τα 3 δέντρα, αλλά τα 5 είχαν καλύτερη απόδοση. Όταν βάλαμε 7 δέντρα είδαμε ότι τα 5 είχαν ελάχιστα καλύτερη απόδοση σε κάποιες περιπτώσεις, ενώ και ο χρόνος εκτέλεσης ξέφευγε. Επομένως αποφασίσαμε να κρατήσουμε τα 5. Όσο για το threshold, αλλάζουμε τον αριθμό των δέντρων που απαιτούνται για να καταταχθεί μία λέξη.

**Bayes**

Ο τρίτος και τελευταίος αλγόριθμος που υλοποιήσαμε είναι ο Bayes. Αρχικά του δίνουμε να διαβάσει τα αρχεία όπως και στους προηγούμενους αλγόριθμους. Καλεί την μέθοδο words στην οποία δίνει σαν όρισμα το κάθε αρχείο και 0 ή 1 ανάλογα με το αν το αρχείο είναι θετικό ή αρνητικό. Η words επιστρέφει ένα HashMap το οποίο περιέχει τις λέξεις που έχει το κάθε αρχείο που διαβάζουμε και πόσες φορές περιέχει την κάθε μία. Η κύρια δουλειά σε αυτόν τον αλγόριθμο γίνεται στην μέθοδο possibilityBayes. Σε αυτήν, βάζουμε αρχικά στο HashMap master όλες τις λέξεις των παραδειγμάτων και φτιάχνουμε και ένα ArrayList temp στο οποίο θα αποθηκεύσουμε τις λέξεις και τα σημεία στίξης του κάθε παραδείγματος. Έπειτα αρχικοποιούμε τα pneg, ppos που είναι η πιθανότητα το αρχείο να είναι θετικό και η πιθανότητα να είναι αρνητικό αντίστοιχα. Την ppos την αρχικοποιούμε με την τιμή που θα επιστρέψει η calculatePossibility η οποία είναι η μέθοδος που χρησιμοποιήσαμε και στον ID3. Σαν όρισμα της δίνουμε το example που είναι ένα HashMap που περιέχει όλα τα παραδείγματα. Την pneg την αρχικοποιούμε με 1-ppos. Ακολούθως, ελέγχουμε την πιθανότητα κάθε λέξης. Δημιουργούμε ένα for loop και μέσα σε αυτό κάνουμε ορισμένους ελέγχους τους οποίους θα εξηγήσουμε αναλυτικά παρακάτω. Μέσα στους ελέγχους γίνεται η χρήση 2 ακόμα HashMap, του masterneg και του masterpos. Το masterpos περιέχει όλα τα attributes που υπάρχουν σε θετικά παραδείγματα και σαν value έχει τον αριθμό των θετικών παραδειγμάτων στα οποία εμφανίστηκαν. Το masterneg έχει την ίδια λειτουργία για τα αρνητικά παραδείγματα. Θα ξεκινήσουμε από τον τελευταίο έλεγχο δηλαδή το «else». Αν φτάσουμε σε αυτόν τον έλεγχο, φτιάχνουμε κάποιες τοπικές μεταβλητές. Πρώτα, οι hasWordPos και hasWordNeg που είναι η πιθανότητα το μήνυμα να είναι θετικό και να περιέχεται η λέξη ή αντίστοιχα να είναι αρνητικό και να περιέχεται η λέξη. Έπειτα οι notWordPos και notWordNeg που είναι η πιθανότητα το μήνυμα να είναι θετικό και να μην περιέχεται η λέξη ή αντίστοιχα να είναι αρνητικό και να μην περιέχεται η λέξη. Αυτές τις μεταβλητές τις χρησιμοποιούμε για να υπολογίσουμε την πιθανότητα και τις πολλαπλασιάζουμε μαζί με τα προηγούμενα ppos, pneg. Προηγουμένως είναι πιθανό να μην εμφανίζεται κάποια λέξη είτε στα θετικά είτε στα αρνητικά και γι’αυτό κάνουμε την παραδοχή ότι υπάρχει ένα μήνυμα που την περιέχει και ένα μήνυμα που δεν την περιέχει, για την δημιουργία της εκάστοτε πιθανότητας, όπως ορίζει η εκτιμήτρια Laplace. Τέλος, συγκρίνουμε τα ppos-pneg και ανάλογα με το αποτέλεσμα κατατάσσουμε το αρχείο είτε στα θετικά είτε στα αρνητικά. Όταν μετά φορτώσουμε τα testFiles χρησιμοποιείται η μέθοδος testBayes η οποία απλά καλεί την παραπάνω μέθοδο possibilityBayes για κάθε αρχείο και διαλέγει με βάση το αποτέλεσμα αν το κάθε testFile είναι θετικό ή αρνητικό. Επιπρόσθετα, σαν threshold αλλάζουμε την πιθανότητα pneg διαιρώντας τη με δυνάμεις του 100, δίνοντας έτσι κάποιες φορές πλεονέκτημα στα θετικά και κάποιες στα αρνητικά ή και καθόλου φυσικά.