Μέρος Α΄

------Κώδικας------

- ***IMDBPreprocessor.py***

Αυτό το αρχείο είναι υπεύθυνο για την **εξαγωγή, επεξεργασία και μετατροπή των δεδομένων κειμένου** σε μορφή κατάλληλη για την εκπαίδευση των αλγορίθμων μηχανικής μάθησης. Δεδομένου ότι οι ταξινομητές δεν μπορούν να διαχειριστούν απευθείας ακατέργαστο κείμενο, απαιτείται προεπεξεργασία, η οποία περιλαμβάνει τα εξής βήματα:

- **Φόρτωση δεδομένων και Τυχαία Αναδιάταξη**: Η φόρτωση των δεδομένων πραγματοποιείται μέσω της μεθόδου *load_dataset().* Τα κείμενα προέρχονται από το **IMDB** Large Movie Review Dataset και ταξινομούνται σε δύο φακέλους: test και train οι οποίοι με την σειρά τους έχουν δύο υποφακέλους: pos (θετικές κριτικές) και neg (αρνητικές κριτικές). Κατά τη φόρτωσή τους, κάθε κείμενο μετατρέπεται σε πεζούς χαρακτήρες και αντιστοιχίζεται σε μία δυαδική ετικέτα (1 για θετικά, 0 για αρνητικά). Για να αποφευχθεί πιθανή σειριακή επίδραση των δεδομένων, η οποία μπορεί να οδηγήσει σε biased εκπαίδευση, τα δεδομένα αναδιατάσσονται τυχαία. Αυτό διασφαλίζει ότι τα παραδείγματα εκπαίδευσης και αξιολόγησης κατανέμονται ομοιόμορφα.
- **Διαχωρισμός Δεδομένων σε Training, Development και Test Sets**: Από τα δεδομένα εκπαίδευσης (train), ένα υποσύνολο 20,000 κειμένων χρησιμοποιείται για την εκπαίδευση, ενώ 5,000 κείμενα διαχωρίζονται ως development set για την επιλογή υπερ-παραμέτρων. Το development set χρησιμοποιείται για τον προσδιορισμό των βέλτιστων παραμέτρων πριν από την τελική αξιολόγηση στο test set. Τα test δεδομένα παραμένουν

ανεξάρτητα και δεν χρησιμοποιούνται μέχρι το τελικό στάδιο.

- **Κατασκευή Λεξιλογίου (Vocabulary)**: Το λεξιλόγιο δημιουργείται αποκλειστικά από τα δεδομένα εκπαίδευσης. Αρχικά, υπολογίζεται η συχνότητα εμφάνισης κάθε λέξης. Οι λέξεις απομονώνονται με κανονικές εκφράσεις (re.sub(r'[^a-zA-Z]', ' ', text)), αφαιρώντας αριθμούς και ειδικούς χαρακτήρες. Χρησιμοποιείται word_tokenize() από το nltk για τον διαχωρισμό των λέξεων. Το Counter() υπολογίζει τη συχνότητα εμφάνισης κάθε λέξης. Το αρχικό λεξιλόγιο μπορεί να περιέχει πάρα πολλές λέξεις, συμπεριλαμβανομένων πολύ κοινών ή σπάνιων λέξεων που δεν συνεισφέρουν στην ταξινόμηση. Για αυτό, εφαρμόζεται φιλτράρισμα. Οι n πιο συχνές λέξεις αφαιρούνται και οι k πιο σπάνιες λέξεις επίσης αφαιρούνται, ώστε να μην επηρεάζουν τα μοντέλα.
- **Μετατροπή Κειμένων σε Feature Matrices.** Κάθε κείμενο αναπαρίσταται πλέον ως δυαδικό διάνυσμα, όπου κάθε θέση δείχνει αν η αντίστοιχη λέξη εμφανίζεται στο κείμενο (1) ή όχι (0). Για το training set, κάθε λέξη αποθηκεύεται μία φορά ανά κείμενο (set(word_tokenize(...))), ώστε να μην επηρεάζει η συχνότητα. Το train_matrix γεμίζει με 1 όπου υπάρχει λέξη στο λεξιλόγιο.
- **Επιλογή των m Χαρακτηριστικών με Information Gain.** Αφού δημιουργηθεί το feature matrix, χρησιμοποιούμε Information Gain για την επιλογή των m σημαντικότερων λέξεων. Το ο *mutual_info_classif* από το *sklearn* υπολογίζει τη σχετική πληροφορία κάθε λέξης. Διατηρούνται τα m χαρακτηριστικά με τη μεγαλύτερη τιμή Information Gain.
- **Μετατροπή των Development και Test Sets.** Τα development και test sets μετατρέπονται με το ίδιο λεξιλόγιο. Τα δεδομένα περνάνε από την ίδια επεξεργασία

όπως και το training set. Τα τελικά feature matrices ενώνονται σε ένα *(self.feature_matrix).*

- ***EvaluationMetrics.py***

Αυτό το αρχείο είναι υπεύθυνο για τον υπολογισμό των μετρικών απόδοσης των αλγορίθμων μηχανικής μάθησης.

- **Υπολογισμός Μετρικών για Κάθε Κλάση**. Η συνάρτηση *calculate_class_metrics(y_true, y_pred)* υπολογίζει τις βασικές μετρικές ξεχωριστά για κάθε κλάση (0 και 1), δηλαδή για τις αρνητικές και θετικές κριτικές αντίστοιχα. **Γίνεται επανάληψη στις δύο κλάσεις (0 και 1)** ώστε να υπολογιστούν ξεχωριστά τα αποτελέσματα. Η *precision_score()* υπολογίζει την ακρίβεια (precision) για κάθε κλάση. Η *recall_score()* υπολογίζει την ανάκληση (recall). Η *f1_score()* υπολογίζει το F1-score. Η έξοδος είναι ένα dictionary (λεξικό), όπου κάθε κλειδί (0 ή 1) έχει μια αντίστοιχη τριάδα τιμών (precision, recall, F1-score).
- **Υπολογισμός Συνολικών Μετρικών**

Η συνάρτηση *calculate_macro_micro_metrics* υπολογίζει τις συνολικές μετρικές απόδοσης, ώστε να έχουμε μια γενική εικόνα της συμπεριφοράς του ταξινομητή. Η συνάρτηση καλεί την *calculate_class_metrics()* για να υπολογίσει τις μετρικές ξεχωριστά για κάθε κλάση και στη συνέχεια επιστρέφει:

- **Macro precision, recall, F1-score**
  Υπολογίζονται ως ο αριθμητικός μέσος των αντίστοιχων τιμών των δύο κλάσεων.
- **Micro precision, recall, F1-score**
  Βασίζονται στον συνολικό αριθμό σωστών και λανθασμένων προβλέψεων ανεξάρτητα από τις κλάσεις.

- ***MetricsCSVWriter.py***

Αυτό το αρχείο είναι υπεύθυνο για την καταγραφή και αποθήκευση των μετρικών απόδοσης των αλγορίθμων ταξινόμησης σε αρχεία CSV.

- **Αποθήκευση μετρικών εκπαίδευσης και ανάπτυξης** μέσω της *write_training_metrics()*. Για κάθε αριθμό παραδειγμάτων εκπαίδευσης (size), καταγράφονται οι μετρικές (Precision, Recall, F1-score) για το training και development set.
- Αποθήκευση μετρικών αξιολόγησης στο test set μέσω της *write_testing_metrics()*. Οι τελικές μετρικές (ανά κλάση και οι macro/micro averages) καταγράφονται σε ξεχωριστό CSV.

- **MetricsVisualizer.py**

Αυτό το αρχείο είναι υπεύθυνο για τη δημιουργία διαγραμμάτων και πινάκων που απεικονίζουν τις επιδόσεις των μοντέλων ταξινόμησης.

- **Δημιουργία Καμπυλών Μάθησης** (*plot_learning_curves()*): Χρησιμοποιώντας δεδομένα από το CSV, δημιουργούνται ξεχωριστές καμπύλες για το Precision, Recall και F1-score, απεικονίζοντας πώς αλλάζει η απόδοση του μοντέλου καθώς αυξάνεται το μέγεθος των δεδομένων εκπαίδευσης.
- **Δημιουργία Πίνακα Αξιολόγησης** (*generate_evaluation_table()*): Τα τελικά αποτελέσματα αποθηκεύονται ως μια εικόνα (PNG), περιλαμβάνοντας τις μετρικές για κάθε κλάση και τους macro/micro μέσους όρους.

- **Γενική Δομή των Main αρχείων**

Τα αρχεία *MainNaiveBayesBernoulli.py*, *MainAdaBoost.py* και *MainRandomForest.py* ακολουθούν **την ίδια βασική δομή**, η οποία περιλαμβάνει τη φόρτωση και προεπεξεργασία των δεδομένων, την εκπαίδευση του ταξινομητή, την αξιολόγηση της απόδοσής του και την αποθήκευση των αποτελεσμάτων. Αυτή η κοινή δομή επιτρέπει τη συγκρίσιμη αξιολόγηση των αλγορίθμων. Όλα τα αρχεία ακολουθούν την παρακάτω δομή

- **Φόρτωση** των Βιβλιοθηκών και Ορισμός Διαδρομών Δεδομένων
- **Ορισμός Υπερπαραμέτρων**. Καθορισμός παραμέτρων για την προεπεξεργασία (π.χ., n, k, m). Ορισμός υπερπαραμέτρων για τον αλγόριθμο (π.χ., num_stumps για AdaBoost, n_estimators για Random Forest).
- **Φόρτωση και Προεπεξεργασία Δεδομένων.** Χρήση του IMDBPreprocessor για την αφαίρεση λέξεων και επιλογή χαρακτηριστικών. Διαχωρισμός των δεδομένων σε training (20,000), development (5,000) και test (25,000) sets.
- **Εκπαίδευση του Μοντέλου με Σταδιακή Αύξηση Δεδομένων.** Εκπαίδευση με διαφορετικά μεγέθη δεδομένων ([5000, 10000, 15000, 20000]).Αξιολόγηση της απόδοσης σε κάθε στάδιο και αποθήκευση των μετρικών στο CSV.
- **Τελική Εκπαίδευση και Αξιολόγηση στο Test Set.** Εκπαίδευση του ταξινομητή με όλα τα διαθέσιμα δεδομένα. Πρόβλεψη στο test set και αποθήκευση των τελικών μετρικών απόδοσης.
- **Οπτικοποίηση των Αποτελεσμάτων**. Δημιουργία καμπυλών μάθησης για Precision, Recall και F1-score. Αποθήκευση των τελικών μετρικών σε αρχεία εικόνας.

- **NaiveBayesBernoulli.py**

Αυτό το αρχείο περιέχει την υλοποίηση του Bernoulli Naïve Bayes ταξινομητή, που χρησιμοποιείται για δυαδική ταξινόμηση. Το μοντέλο βασίζεται στη θεωρία πιθανοτήτων και υπολογίζει την πιθανότητα μιας κλάσης για ένα δεδομένο δείγμα, χρησιμοποιώντας την υπόθεση ότι τα χαρακτηριστικά είναι ανεξάρτητα μεταξύ τους.

- **Αρχικοποίηση του Ταξινομητή *(__init__).*** Ο κατασκευαστής της κλάσης NaiveBayesBernoulli αρχικοποιεί τις παραμέτρους που απαιτούνται για την εκπαίδευση και την πρόβλεψη. Οι εκ των προτέρων πιθανότητες (**prior**

probabilities) για κάθε κλάση (P(y=1), P(y=0)). Οι πιθανοτικές κατανομές των λέξεων ανά κλάση (P(x|y=1) και P(x|y=0)). Η αρχικοποίηση των παραμέτρων σε None σημαίνει ότι θα οριστούν κατά την εκπαίδευση.

- **Εκπαίδευση του Μοντέλου** (**train):** Το μοντέλο λαμβάνει ως είσοδο μια δυαδική μήτρα χαρακτηριστικών (feature_matrix), όπου κάθε γραμμή αντιπροσωπεύει ένα κείμενο και κάθε στήλη αντιστοιχεί σε μια λέξη από το λεξιλόγιο. Παράλληλα, δίνεται ένα διάνυσμα ετικετών (labels), το οποίο περιέχει 0 για αρνητικές και 1 για θετικές κριτικές. Αρχικά, υπολογίζεται ο συνολικός αριθμός των θετικών και αρνητικών κριτικών (positive_count και negative_count αντίστοιχα) και με βάση αυτούς τους αριθμούς προσδιορίζονται οι **εκ των προτέρων πιθανότητες** (priors) για κάθε κλάση, δηλαδή η πιθανότητα εμφάνισης μιας θετικής (P(y=1)) ή αρνητικής (P(y=0)) κριτικής ανεξάρτητα από τα χαρακτηριστικά. Στη συνέχεια, για κάθε λέξη στο λεξιλόγιο, υπολογίζεται το πόσες φορές αυτή εμφανίζεται μέσα στις θετικές (word_count_positive) και αρνητικές (word_count_negative) κριτικές. Με βάση αυτές τις μετρήσεις, υπολογίζονται οι **πιθανότητες εμφάνισης κάθε λέξης μέσα σε θετικές ή αρνητικές κριτικές** (P(x=1|y=1) και P(x=1|y=0)). Για την αποφυγή μηδενικών πιθανοτήτων και για την εξομάλυνση των πιθανοτήτων, εφαρμόζεται **Laplace Smoothing** με την προσθήκη ενός σταθερού όρου (+1) στον αριθμητή και του διπλάσιου (+2) στον παρονομαστή, ώστε να διατηρείται μια ομαλή εκτίμηση ακόμη και για λέξεις που εμφανίζονται σπάνια ή καθόλου.
- **Προβλέψεις του Μοντέλου** (**predict**)**:** Λαμβάνει ως είσοδο ένα νέο δείγμα (feature_vector). Η διαδικασία της πρόβλεψης βασίζεται στον υπολογισμό των πιθανών πιθανοτήτων για κάθε κλάση και την επιλογή της κλάσης με τη μεγαλύτερη πιθανότητα. Αρχικά, υπολογίζονται οι

λογαριθμικές εκ των προτέρων πιθανότητες (log(P(y=1)) και log(P(y=0))), ώστε να διατηρείται αριθμητική σταθερότητα και να αποφεύγονται υπερχειλίσεις (underflow) για πολύ μικρές πιθανότητες. Στη συνέχεια, υπολογίζονται τα λογαριθμικά γινόμενα των πιθανοτήτων των χαρακτηριστικών: για κάθε λέξη που υπάρχει στο δείγμα (x=1), προστίθεται η αντίστοιχη λογαριθμική πιθανότητα εμφάνισης (log(P(x|y))), ενώ για κάθε λέξη που δεν υπάρχει (x=0), προστίθεται η πιθανότητα μη εμφάνισής της (log(1 - P(x|y))). Αυτοί οι υπολογισμοί γίνονται τόσο για την θετική όσο και για την αρνητική κλάση, ώστε να προκύψει η συνολική λογαριθμική πιθανότητα εμφάνισης του δείγματος υπό κάθε κλάση. Τελικά, συγκρίνονται οι δύο λογαριθμικές πιθανότητες (log_prob_positive και log_prob_negative), και το μοντέλο επιστρέφει 1 αν η πιθανότητα μιας θετικής κριτικής είναι μεγαλύτερη, διαφορετικά επιστρέφει 0, δηλώνοντας ότι το κείμενο είναι αρνητικό.

- **RandomForest.py**

Αυτό το αρχείο περιέχει την υλοποίηση του Random Forest, ενός ensemble ταξινομητή που βασίζεται στη δημιουργία πολλαπλών Δέντρων Απόφασης (Decision Trees). Στο Random Forest κάθε δέντρο εκπαιδεύεται σε τυχαίο υποσύνολο των δεδομένων, και η τελική πρόβλεψη προκύπτει μέσω πλειοψηφικής ψήφου (majority voting).

• **Αρχικοποίηση του Ταξινομητή (__init__)**: Ο κατασκευαστής της κλάσης RandomForest αρχικοποιεί τις βασικές παραμέτρους που απαιτούνται για την εκπαίδευση και την πρόβλεψη. Συγκεκριμένα, αποθηκεύονται ο **αριθμός των δέντρων (n_estimators)** που θα περιέχει το δάσος, το **μέγιστο επιτρεπτό βάθος κάθε δέντρου (max_depth)**, καθώς και το **ποσοστό των χαρακτηριστικών που θα εξετάζονται σε κάθε διαχωρισμό (max_features)**. Επιπλέον, δημιουργείται μια κενή λίστα

self.trees, όπου θα αποθηκεύονται τα δέντρα που εκπαιδεύονται.

• **Εκπαίδευση του Μοντέλου (fit_incremental)**: Η εκπαίδευση του Random Forest βασίζεται στην τεχνική του bootstrap sampling, όπου κάθε δέντρο εκπαιδεύεται σε ένα τυχαίο υποσύνολο των δεδομένων. Για να επιτευχθεί αυτό, αρχικά δημιουργούνται νέα δεδομένα εκπαίδευσης μέσω τυχαίας δειγματοληψίας με επανατοποθέτηση (resample(X, y)). Στη συνέχεια, δημιουργείται ένα νέο Δέντρο Απόφασης (Decision Tree) με κριτήριο διαχωρισμού την Entropy (Information Gain), ενώ οι υπερπαράμετροι max_depth και max_features ορίζουν το βάθος και τον αριθμό των χαρακτηριστικών που θα εξεταστούν κατά την ανάπτυξη του δέντρου. Το νέο δέντρο εκπαιδεύεται αποκλειστικά στα δείγματα του bootstrap και αποθηκεύεται στη λίστα self.trees. Ένα βασικό χαρακτηριστικό της υλοποίησης είναι ότι διατηρούνται μόνο τα τελευταία n_estimators δέντρα, διαγράφοντας τα παλαιότερα όταν ο αριθμός τους υπερβαίνει το επιθυμητό όριο. Αυτό επιτρέπει τη σταδιακή εκπαίδευση του μοντέλου, επιτρέποντας στο Random Forest να ενημερώνεται με νέα δεδομένα χωρίς να ξεκινά από την αρχή.

• **Προβλέψεις του Μοντέλου (predict)**: Η διαδικασία πρόβλεψης βασίζεται στη λήψη της **πλειοψηφικής ψήφου** μεταξύ των προβλέψεων όλων των δέντρων. Για κάθε νέο δείγμα, όλα τα δέντρα του δάσους πραγματοποιούν ανεξάρτητες προβλέψεις (tree.predict(X)), οι οποίες αποθηκεύονται σε έναν πίνακα predictions. Στη συνέχεια, για κάθε δείγμα, επιλέγεται η κλάση που εμφανίζεται **συχνότερα** μεταξύ των δέντρων μέσω της συνάρτησης mode(). Το αποτέλεσμα της πλειοψηφικής ψήφου επιστρέφεται ως η τελική πρόβλεψη του μοντέλου.

- **AdaBoost.py**

Αυτό το αρχείο περιέχει την υλοποίηση του αλγορίθμου AdaBoost, ο οποίος είναι ένας ensemble ταξινομητή που

συνδυάζει πολλαπλούς αδύναμους ταξινομητές (decision stumps) ώστε να δημιουργήσει έναν ισχυρό ταξινομητή. Η ιδέα πίσω από τον AdaBoost είναι η διαδοχική εκπαίδευση αδύναμων μοντέλων, όπου κάθε νέο μοντέλο επικεντρώνεται στα δεδομένα που ταξινομήθηκαν λανθασμένα από τα προηγούμενα.

Αρχικοποίηση του Ταξινομητή (__init__): Ο κατασκευαστής της κλάσης AdaBoost αρχικοποιεί τον αριθμό των decision stumps (num_stumps) που θα χρησιμοποιηθούν, έναν κενό κατάλογο (stumps) όπου θα αποθηκεύονται οι decision stumps και έναν κατάλογο (stump_weights) όπου θα αποθηκεύονται οι αντίστοιχες τιμές βαρών (weights) για κάθε stump. Οι decision stumps λειτουργούν ως απλοί δυαδικοί ταξινομητές που χρησιμοποιούν ένα μόνο χαρακτηριστικό για την πρόβλεψη της κλάσης.

Εκπαίδευση του Μοντέλου (train): Η εκπαίδευση πραγματοποιείται μέσω της επαναληπτικής εκπαίδευσης decision stumps σε διαφορετικά βάρη των δεδομένων. Αρχικά, οι τιμές βαρών των δειγμάτων (weights) αρχικοποιούνται ομοιόμορφα. Σε κάθε επανάληψη, ένας decision stump εκπαιδεύεται με βάση τα τρέχοντα βάρη. Υπολογίζεται το ποσοστό των λανθασμένων ταξινομήσεων (error), το οποίο στη συνέχεια χρησιμοποιείται για να καθοριστεί το βάρος (alpha) του stump. Τα δείγματα που ταξινομήθηκαν λανθασμένα λαμβάνουν μεγαλύτερο βάρος στην επόμενη επανάληψη, ώστε οι επόμενοι ταξινομητές να επικεντρωθούν σε αυτά. Η διαδικασία συνεχίζεται μέχρι να δημιουργηθούν num_stumps αδύναμοι ταξινομητές ή μέχρι να μην μπορούν να μειωθούν περαιτέρω τα λάθη.

Προβλέψεις του Μοντέλου (predict): Η τελική πρόβλεψη του AdaBoost γίνεται συνδυάζοντας τις προβλέψεις των decision stumps, όπου κάθε stump συνεισφέρει με το αντίστοιχο βάρος του (alpha). Το αποτέλεσμα είναι ένας σταθμισμένος αθροιστής των επιμέρους προβλέψεων, ο οποίος ταξινομεί κάθε δείγμα

στην κλάση 1 αν το τελικό σκορ είναι θετικό και στην κλάση 0 διαφορετικά.

DecisionStump.py

Αυτό το αρχείο περιέχει την υλοποίηση του decision stump, που είναι ένας απλός ταξινομητής δέντρου απόφασης βάθους 1. Κάθε decision stump επιλέγει ένα μόνο χαρακτηριστικό και ένα κατώφλι (threshold) για να διαχωρίσει τα δεδομένα στις δύο κλάσεις.

Αρχικοποίηση του Decision Stump (__init__): Ο κατασκευαστής της κλάσης DecisionStump αρχικοποιεί τις μεταβλητές selected_feature (το χαρακτηριστικό που θα χρησιμοποιηθεί), threshold (το κατώφλι διαχωρισμού), polarity (η κατεύθυνση του διαχωρισμού) και alpha (το βάρος του stump στην AdaBoost).

Εκπαίδευση του Decision Stump (train): Η εκπαίδευση ενός decision stump πραγματοποιείται με την επιλογή του βέλτιστου χαρακτηριστικού βάσει του Information Gain. Χρησιμοποιείται η συνάρτηση mutual_info_classif της sklearn, ώστε να υπολογιστεί ο Information Gain κάθε χαρακτηριστικού, και επιλέγεται αυτό με τη μεγαλύτερη τιμή. Στη συνέχεια, το stump υπολογίζει το κατώφλι (threshold) και την πολικότητα (polarity) που ελαχιστοποιούν το σφάλμα ταξινόμησης. Οι τιμές των χαρακτηριστικών ταξινομούνται, και δοκιμάζονται διαφορετικά thresholds, επιλέγοντας αυτό που παράγει το μικρότερο σφάλμα σε συνδυασμό με το κατάλληλο polarity (1 ή -1).

Προβλέψεις του Decision Stump (predict): Κατά την πρόβλεψη, το stump εφαρμόζει το threshold στο επιλεγμένο χαρακτηριστικό και κατηγοριοποιεί τα δείγματα στην κλάση 1 ή 0, ανάλογα με την πολικότητα που έχει οριστεί κατά την εκπαίδευση. Αν η τιμή του χαρακτηριστικού είναι μεγαλύτερη από το threshold και η πολικότητα είναι 1, η πρόβλεψη είναι 1, διαφορετικά 0. Αν η πολικότητα είναι -1, το αντίθετο ισχύει.