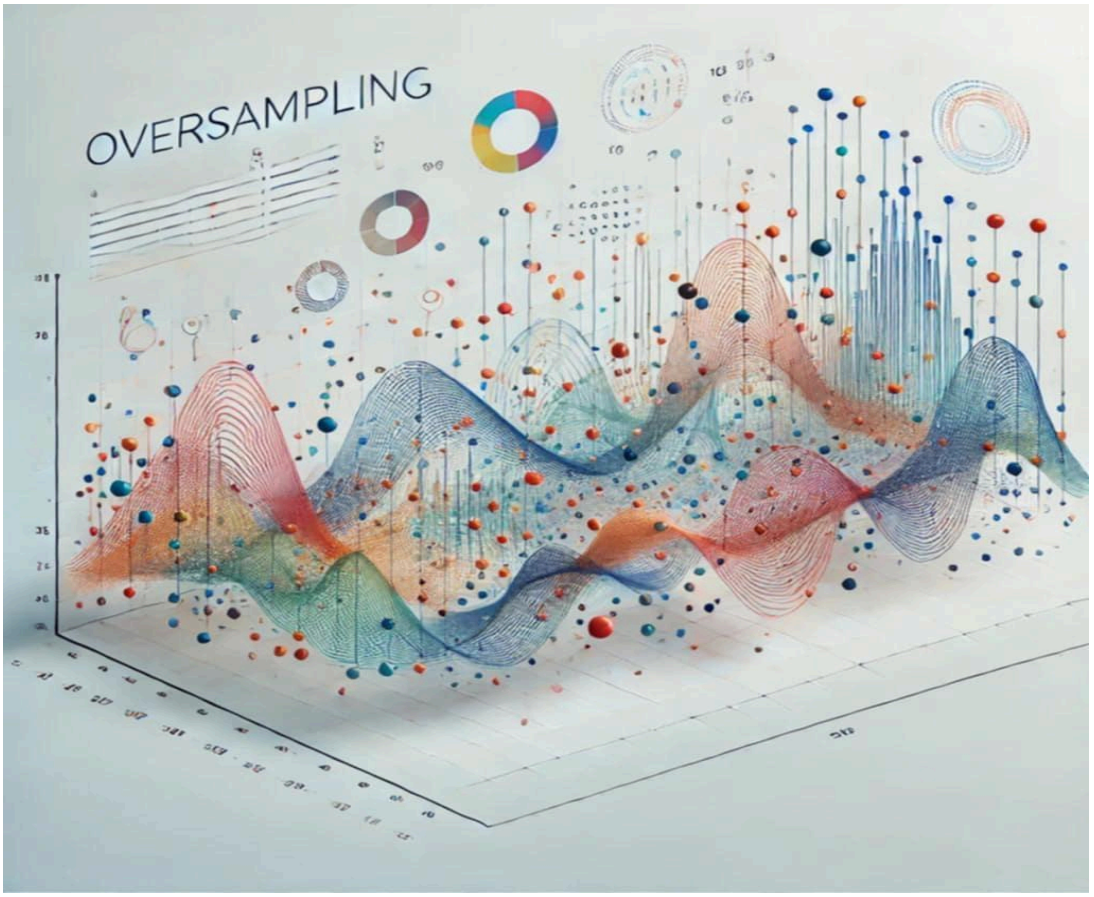
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Αντιμετώπιση της ανισοκατανομής των κλάσεων σε προβλήματα κατηγοριοποίησης μέσω υπερδειγματοληψίας: Ανασκόπηση της βιβλιογραφίας και πειραματική μελέτη»



Της φοιτήτριας
Χωλίδου Αφροδίτης
Αρ. Μητρώου: 144329

Επιβλέπων
Ονοματεπώνυμο: Ουγιάρογλου Στέφανος
Βαθμίδα ……

Ημερομηνία 13/02/2025

Αντιμετώπιση της ανισοκατανομής των κλάσεων σε προβλήματα κατηγοριοποίησης μέσω υπερδειγματοληψίας: Ανασκόπηση της βιβλιογραφίας και πειραματική μελέτη

24113
Χωλίδου Αφροδίτη
Ουγιάρογλου Στέφανος

26/01/2024

Ημερομηνία περάτωσης Δ.Ε. …

Βεβαιώνω ότι είμαι η συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Η επιλογή του θέματος αυτής της εργασίας προέρχεται από το έντονο ενδιαφέρον μας για τις τεχνικές διαχείρισης μη ισορροπημένων συνόλων δεδομένων, έναν κρίσιμο τομέα στη μηχανική μάθηση. Η αυξανόμενη σημασία της ανάλυσης δεδομένων καθιστά απαραίτητη τη βαθύτερη κατανόηση και εφαρμογή μεθόδων που βελτιώνουν την απόδοση των αλγορίθμων κατηγοριοποίησης, ιδιαίτερα σε περιπτώσεις όπου οι κατηγορίες δεν είναι ισόρροπα κατανεμημένες. Οι τεχνικές oversampling, όπως το SMOTE και οι παραλλαγές του, έχουν αποδείξει την αξία τους στη βελτίωση της ταξινόμησης μειονοτικών κλάσεων, γεγονός που μας ώθησε να εστιάσουμε στη σύγκριση διαφορετικών προσεγγίσεων. Η μελέτη της επίδρασης αυτών των αλγορίθμων στην ακρίβεια και τη γενίκευση των μοντέλων κατηγοριοποίησης αποτελεί έναν ιδιαίτερα ενδιαφέροντα τομέα, καθώς συνδυάζει τη θεωρητική γνώση με την πρακτική εφαρμογή. Μέσα από αυτή την έρευνα, στοχεύουμε να αποκτήσουμε βαθύτερη κατανόηση των χαρακτηριστικών των τεχνικών oversampling και να ενισχύσουμε τις γνώσεις μας στη μηχανική μάθηση, δημιουργώντας ένα ισχυρό θεμέλιο για μελλοντικές ερευνητικές ή επαγγελματικές εφαρμογές.

Περίληψη

Η ανισοκατανομή των κλάσεων σε προβλήματα κατηγοριοποίησης αποτελεί ένα συνηθισμένο ζήτημα που μπορεί να επηρεάσει την απόδοση των αλγορίθμων μηχανικής μάθησης. Στην παρούσα εργασία, μελετώνται οι τεχνικές αντιμετώπισης αυτού του προβλήματος μέσω υπερδειγματοληψίας. Ιδιαίτερη έμφαση δίνεται σε μεθόδους όπως το SMOTE (Synthetic Minority Over-sampling Technique), το SMOTEN, το ADASYN (Adaptive Synthetic Sampling), το KMeansSMOTE και το SVM SMOTE, οι οποίες εφαρμόζονται σε πειραματική μελέτη σε διάφορα σύνολα δεδομένων.

Η βιβλιογραφική ανασκόπηση περιλαμβάνει την ανάλυση των θεωρητικών αρχών των παραπάνω αλγορίθμων και εξετάζει τις διαφορές τους σε επίπεδο απόδοσης και πολυπλοκότητας. Στο πλαίσιο της πειραματικής μελέτης, αξιολογείται η αποτελεσματικότητα κάθε μεθόδου σε δεδομένα με έντονη ανισοκατανομή, λαμβάνοντας υπόψη μετρικές όπως η ακρίβεια, η ευαισθησία και η στατιστική αξιοπιστία. Τα αποτελέσματα της μελέτης αναδεικνύουν ότι τεχνικές όπως το SMOTE και οι παραλλαγές του μπορούν να βελτιώσουν την απόδοση των μοντέλων όσον αφορά τις κλάσεις μειοψηφίας. Επιπλέον, μέθοδοι όπως το ADASYN, το KMeansSMOTE και το SVM SMOTE επιδεικνύουν ιδιαίτερη αποτελεσματικότητα σε πιο σύνθετα και διαφορετικά σύνολα δεδομένων.

Η σύγκριση των αλγορίθμων υποδεικνύει ότι η επιλογή της κατάλληλης μεθόδου εξαρτάται από τα χαρακτηριστικά των δεδομένων και τη φύση του προβλήματος κατηγοριοποίησης. Μέσω αυτής της ανάλυσης, προτείνονται κατευθύνσεις για την επιλογή της βέλτιστης προσέγγισης, συμβάλλοντας στη βελτίωση της απόδοσης των μοντέλων μηχανικής μάθησης σε περιβάλλοντα με ανισοκατανομή κλάσεων.

Addressing class imbalance in classification problems through oversampling:

A literature review and experimental study

Cholidou Afroditi

Abstract

Uneven distribution of classes in classification problems is a common issue that can affect the performance of machine learning algorithms. In the present work, the techniques for dealing with this problem through oversampling and undersampling are studied. Special emphasis is given to methods such as SMOTE (Synthetic Minority Over-sampling Technique), SMOTEN, ADASYN (Adaptive Synthetic Sampling), KMeansSMOTE and SVM SMOTE, which are applied in an experimental study on various datasets.

The literature review includes the analysis of the theoretical principles of the above algorithms and examines their differences in terms of performance and complexity. In the context of the experimental study, the effectiveness of each method on highly skewed data is evaluated, taking into account metrics such as accuracy, sensitivity and statistical reliability. The results of the study highlight that techniques such as SMOTE and its variants can improve the performance of models with respect to minority classes. Furthermore, methods such as ADASYN, KMeansSMOTE, and SVM SMOTE demonstrate particular effectiveness on more complex and diverse datasets.

The comparison of algorithms indicates that the choice of the appropriate method depends on the characteristics of the data and the nature of the classification problem. Through this analysis, directions are suggested for choosing the optimal approach, helping to improve the performance of machine learning models in environments with unequal distribution of classes.

Ευχαριστίες

Σε αυτήν την ενότητα θα ήθελα να ευχαριστήσω τον διδάσκοντα καθηγητή μου, κ. Ουγιάρογλου Στέφανο για την αμέριστη προσοχή και σημαντική βοήθειά του κατά την εκπόνηση αυτής της εργασίας. Επιπλέον, θα ήθελα να ευχαριστήσω τους συναδέλφους μου για την ψυχολογική και γνωστική υποστήριξη που μου παρείχαν ανιδιοτελώς.

Περιεχόμενα

ΒΙΒΛΙΟΓΡΑΦΙΑ

Κατάλογος Εικόνων / Πινάκων

Σχήμα 1: A Novel Multi-class Imbalanced EEG Signals Classification Based on the Adaptive Synthetic Sampling (ADASYN) approach - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-working-of-ADASYN-in-a-dataset-A-presents-the-original-dataset-class-distribution_fig2_351588991 [accessed 11 Feb 2025]

Σχήμα 2: Don’t Get Caught in the Trap of Imbalanced Data When Building a Model - Brian Roepke

Σχήμα 3: Random Forest Algorithm Explained - Anas Brital

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

Κεφάλαιο 1ο: Εισαγωγή στη μηχανική μάθηση

1.1 Κατηγοριοποίηση δεδομένων

Η κατηγοριοποίηση δεδομένων είναι μια διαδικασία κατά την οποία ένα σύστημα προσπαθεί να αναγνωρίσει και να τοποθετήσει ένα σύνολο δεδομένων σε προκαθορισμένες κατηγορίες. Στόχος είναι να δημιουργηθεί ένα μοντέλο που μπορεί να προβλέψει σε ποια κατηγορία ανήκουν νέα δεδομένα, χρησιμοποιώντας κάποια χαρακτηριστικά που περιγράφουν τις εγγραφές. Η διαδικασία αυτή αποτελεί βασικό μέρος της μηχανικής μάθησης και χρησιμοποιείται σε πολλές εφαρμογές, όπως η αναγνώριση εικόνων, η επεξεργασία κειμένου και η πρόβλεψη συμπεριφοράς χρηστών.

Η κατηγοριοποίηση των δεδομένων περιλαμβάνει δύο βασικά στάδια: πρώτα το σύστημα «εκπαιδεύεται» χρησιμοποιώντας ένα σύνολο δεδομένων όπου είναι γνωστές οι κατηγορίες στις οποίες ανήκουν οι εγγραφές. Στη συνέχεια, ελέγχεται η απόδοσή του χρησιμοποιώντας νέα δεδομένα για να διαπιστωθεί κατά πόσο οι προβλέψεις του είναι σωστές. Η αποτελεσματικότητα ενός τέτοιου συστήματος αξιολογείται μέσω μετρικών όπως η ακρίβεια, η ικανότητα σωστής ανίχνευσης των διαφορετικών κατηγοριών και η συνολική απόδοση του μοντέλου.

Υπάρχουν διάφοροι αλγόριθμοι που χρησιμοποιούνται για την κατηγοριοποίηση δεδομένων. Ορισμένοι από αυτούς είναι τα δέντρα απόφασης, τα οποία βασίζονται στη λήψη αποφάσεων μέσω μιας ιεραρχικής δομής κανόνων. Ένας άλλος διαδεδομένος αλγόριθμος είναι το τυχαίο δάσος, το οποίο αποτελείται από πολλά δέντρα απόφασης που συνεργάζονται για πιο αξιόπιστα αποτελέσματα. Οι μηχανές διανυσμάτων υποστήριξης είναι μια άλλη τεχνική που προσπαθεί να βρει το καλύτερο όριο που διαχωρίζει τις κατηγορίες, ενώ η παλινδρόμηση logistic χρησιμοποιείται για να υπολογίσει πιθανότητες που δείχνουν σε ποια κατηγορία ανήκει ένα δεδομένο. Μια πιο απλή μέθοδος είναι οι εγγύτεροι γείτονες, όπου η κατηγορία ενός νέου δεδομένου καθορίζεται από τα πιο κοντινά σημεία που του μοιάζουν. Τέλος, ο αλγόριθμος Naive Bayes χρησιμοποιεί πιθανότητες για να προβλέψει την κατηγορία στην οποία ανήκει ένα δεδομένο, υποθέτοντας ότι όλα τα χαρακτηριστικά είναι ανεξάρτητα μεταξύ τους.

Η επιλογή του κατάλληλου αλγορίθμου εξαρτάται από διάφορους παράγοντες, όπως το είδος και η ποσότητα των δεδομένων, η πολυπλοκότητα του προβλήματος και η ανάγκη για ερμηνευσιμότητα των αποτελεσμάτων. Ωστόσο, ένα σημαντικό πρόβλημα που προκύπτει συχνά στην κατηγοριοποίηση είναι η ανισοκατανομή των δεδομένων, όπου ο αριθμός των δειγμάτων σε ορισμένες κατηγορίες είναι πολύ μικρότερος από άλλες. Αυτό μπορεί να οδηγήσει σε μεροληψία του μοντέλου υπέρ των κυρίαρχων κατηγοριών και να μειώσει την ικανότητά του να αναγνωρίζει σωστά τις λιγότερο συχνές περιπτώσεις. Στην παρούσα εργασία, η ανισοκατανομή των κλάσεων αποτελεί το κύριο πρόβλημα που διερευνάται και προτείνονται τεχνικές αντιμετώπισης μέσω της διαδικασίας της υπερδειγματοληψίας.

1.2 Κατηγοριοποίηση δεδομένων με ανισοκατανομή κλάσεων

Η κατηγοριοποίηση δεδομένων γίνεται πιο δύσκολη όταν υπάρχει ανισοκατανομή στις κλάσεις. Αυτό σημαίνει ότι κάποιες κατηγορίες έχουν πολύ περισσότερα δεδομένα από άλλες. Για παράδειγμα, σε ένα σύστημα ανίχνευσης απάτης στις συναλλαγές, οι περισσότερες συναλλαγές είναι φυσιολογικές,

ενώ οι περιπτώσεις απάτης είναι πολύ λιγότερες. Αυτό δημιουργεί προβλήματα γιατί το μοντέλο μπορεί να μάθει να αγνοεί τις σπάνιες κατηγορίες και να δίνει έμφαση μόνο στις πιο συχνές, με αποτέλεσμα να μην μπορεί να εντοπίσει σωστά τις σπάνιες περιπτώσεις.

Η ανισοκατανομή των κλάσεων επηρεάζει αρνητικά την απόδοση των μοντέλων μηχανικής μάθησης. Τα περισσότερα μοντέλα λειτουργούν καλύτερα όταν οι κατηγορίες έχουν περίπου τον ίδιο αριθμό δεδομένων. Όταν υπάρχει μεγάλη διαφορά, τα μοντέλα τείνουν να προβλέπουν πάντα τη συχνότερη κατηγορία, καθώς έτσι φαίνεται ότι έχουν υψηλή ακρίβεια. Ωστόσο, στην πραγματικότητα μπορεί να αποτυγχάνουν να αναγνωρίσουν τις λιγότερο συχνές περιπτώσεις, κάτι που είναι πολύ σημαντικό σε εφαρμογές όπως η ιατρική διάγνωση ή η ανίχνευση κυβερνοεπιθέσεων.

Για την αντιμετώπιση του προβλήματος της ανισοκατανομής, υπάρχουν διάφορες τεχνικές που μπορούν να χρησιμοποιηθούν. Μία από τις πιο κοινές λύσεις είναι η υπερδειγματοληψία, η οποία αυξάνει τον αριθμό των δεδομένων στις κατηγορίες που έχουν λίγα δείγματα, δημιουργώντας νέα δεδομένα που είναι παρόμοια με τα υπάρχοντα. Με αυτόν τον τρόπο, το μοντέλο εκπαιδεύεται καλύτερα και έχει περισσότερες πιθανότητες να προβλέψει σωστά τις λιγότερο συχνές κλάσεις. Μια άλλη προσέγγιση είναι η υποδειγματοληψία, όπου μειώνεται ο αριθμός των δειγμάτων στις συχνές κατηγορίες, ώστε να υπάρχει καλύτερη ισορροπία.

Εκτός από τις τεχνικές διαχείρισης των δεδομένων, η επιλογή κατάλληλων αλγορίθμων κατηγοριοποίησης παίζει σημαντικό ρόλο. Ορισμένοι αλγόριθμοι, όπως οι δέντρα απόφασης και τα τυχαία δάση, έχουν τη δυνατότητα να προσαρμόζονται σε δεδομένα με ανισοκατανομή, δίνοντας μεγαλύτερη βαρύτητα στις λιγότερο συχνές περιπτώσεις. Επίσης, η χρήση ειδικών μετρικών αξιολόγησης, όπως η ανάκληση και η μέτρηση F1, μπορεί να προσφέρει μια πιο ρεαλιστική εικόνα της απόδοσης του μοντέλου στις σπάνιες κλάσεις.

Συνοψίζοντας, η ανισοκατανομή των κλάσεων είναι ένα σοβαρό πρόβλημα στην κατηγοριοποίηση δεδομένων, καθώς μπορεί να οδηγήσει σε λανθασμένα συμπεράσματα και ανεπαρκή απόδοση των μοντέλων. Οι τεχνικές υπερδειγματοληψίας και υποδειγματοληψίας, καθώς και η σωστή επιλογή αλγορίθμων και μετρικών αξιολόγησης, μπορούν να βοηθήσουν στη βελτίωση των αποτελεσμάτων και στη δημιουργία πιο αξιόπιστων μοντέλων.

1.3 Κίνητρο και συνεισφορά

Η ανισοκατανομή των κλάσεων αποτελεί μια από τις πιο συχνές προκλήσεις στη μηχανική μάθηση, καθώς μπορεί να επηρεάσει αρνητικά την απόδοση των αλγορίθμων κατηγοριοποίησης. Σε πολλά προβλήματα, όπως η ανίχνευση απάτης και η διάγνωση ασθενειών, η μειοψηφική κλάση είναι αυτή που έχει τη μεγαλύτερη σημασία. Ωστόσο, τα περισσότερα μοντέλα μηχανικής μάθησης τείνουν να ευνοούν τις πλειοψηφικές κλάσεις, οδηγώντας σε αναξιόπιστες προβλέψεις για τις σπανιότερες κατηγορίες. Το γεγονός αυτό δημιουργεί την ανάγκη για την εφαρμογή κατάλληλων τεχνικών που να αντιμετωπίζουν το πρόβλημα αποτελεσματικά.

Στην εργασία αυτή, επικεντρωνόμαστε στην εφαρμογή και την ανάλυση τεχνικών υπερδειγματοληψίας, όπως οι Adasyn, SMOTE και Random Oversampler, με στόχο τη βελτίωση της απόδοσης των αλγορίθμων ταξινόμησης σε δεδομένα με ανισοκατανομή. Η συνεισφορά της εργασίας έγκειται στην αξιολόγηση των επιδόσεων αυτών των τεχνικών σε πραγματικά σύνολα δεδομένων,

προκειμένου να διαπιστωθεί η αποτελεσματικότητά τους σε διαφορετικές περιπτώσεις. Πιο συγκεκριμένα, θα πραγματοποιηθεί μια βιβλιογραφική ανασκόπηση των βασικών μεθόδων υπερδειγματοληψίας, καθώς και μια πειραματική μελέτη όπου θα συγκριθούν οι αλγόριθμοι αυτοί σε διάφορα σύνολα δεδομένων.

Η εργασία στοχεύει στην παροχή μιας ολοκληρωμένης επισκόπησης των διαθέσιμων τεχνικών και στην κατανόηση των παραμέτρων που επηρεάζουν την απόδοσή τους. Επιπλέον, μέσα από τη χρήση της γλώσσας Python και δημοφιλών βιβλιοθηκών όπως η Scikit-learn και η Imbalanced-learn, επιδιώκεται η παρουσίαση πρακτικών εφαρμογών που μπορούν να χρησιμοποιηθούν από ερευνητές και επαγγελματίες.

Συνολικά, η εργασία φιλοδοξεί να συνδυάσει τη θεωρητική γνώση με την πρακτική εφαρμογή, προσφέροντας έναν χρήσιμο οδηγό για την αντιμετώπιση του προβλήματος της ανισοκατανομής των κλάσεων σε προβλήματα κατηγοριοποίησης.

1.4 Οργάνωση της εργασίας

Η παρούσα εργασία είναι οργανωμένη σε έξι κεφάλαια, καλύπτοντας τόσο τη θεωρητική ανασκόπηση όσο και την πειραματική αξιολόγηση των τεχνικών υπερδειγματοληψίας. Στο πρώτο κεφάλαιο γίνεται μια εισαγωγή στο πρόβλημα της ανισοκατανομής των κλάσεων, παρουσιάζοντας τη σημασία της κατηγοριοποίησης δεδομένων και τους λόγους για τους οποίους η ανισοκατανομή αποτελεί πρόκληση στη μηχανική μάθηση. Επίσης, αναφέρονται το κίνητρο και η συνεισφορά της εργασίας.

Στο δεύτερο κεφάλαιο αναλύονται οι πιο σημαντικές τεχνικές υπερδειγματοληψίας που χρησιμοποιούνται για την αντιμετώπιση της ανισοκατανομής των κλάσεων. Παρουσιάζονται μέθοδοι όπως το SMOTE, το ADASYN και ο Random Oversampler, ενώ γίνεται αναφορά στις βασικές αρχές λειτουργίας τους και στις διαφορές που παρουσιάζουν μεταξύ τους.

Το τρίτο κεφάλαιο επικεντρώνεται στους αλγόριθμους κατηγοριοποίησης (classification algorithms) που χρησιμοποιούνται στην εργασία. Συγκεκριμένα, εξετάζονται αλγόριθμοι όπως τα δέντρα απόφασης (decision trees), το τυχαίο δάσος (random forest), οι μηχανές διανυσμάτων υποστήριξης (support vector machines - SVM), η λογιστική παλινδρόμηση (logistic regression), οι K εγγύτεροι γείτονες (k-nearest neighbors - K-NN) και ο Naive Bayes. Περιγράφονται τα βασικά χαρακτηριστικά τους, καθώς και οι παράγοντες που επηρεάζουν την απόδοσή τους σε δεδομένα με ανισοκατανομή.

Στο τέταρτο κεφάλαιο παρουσιάζεται η υλοποίηση των τεχνικών υπερδειγματοληψίας και των αλγορίθμων κατηγοριοποίησης χρησιμοποιώντας τη γλώσσα προγραμματισμού Python. Γίνεται αναφορά στις βιβλιοθήκες που χρησιμοποιούνται, όπως η Scikit-learn, καθώς και στον τρόπο με τον οποίο εφαρμόζονται οι τεχνικές στο πλαίσιο της παρούσας εργασίας.

Το πέμπτο κεφάλαιο περιλαμβάνει την πειραματική μελέτη, όπου περιγράφονται τα σύνολα δεδομένων που χρησιμοποιήθηκαν, η διαδικασία πειραματισμού και τα αποτελέσματα που προέκυψαν. Εξετάζεται η απόδοση των τεχνικών υπερδειγματοληψίας σε συνδυασμό με διαφορετικούς αλγόριθμους κατηγοριοποίησης, ώστε να διαπιστωθεί ποιοι συνδυασμοί προσφέρουν την καλύτερη επίδοση.

Τέλος, το έκτο κεφάλαιο συνοψίζει τα συμπεράσματα της εργασίας, παρουσιάζοντας τα κύρια ευρήματα και προτείνοντας πιθανές κατευθύνσεις για μελλοντική έρευνα. Η εργασία ολοκληρώνεται με τη βιβλιογραφία, η οποία περιλαμβάνει όλες τις πηγές που χρησιμοποιήθηκαν.

Κεφάλαιο 2ο: Τεχνικές Υπερδειγματοληψίας

Η υπερδειγματοληψία (oversampling) είναι μία τεχνική που χρησιμοποιείται σε προβλήματα μηχανικής μάθησης, δηλαδή όταν οι κατηγορίες της εξαρτημένης μεταβλητής δεν έχουν ίση κατανομή. Η επιλογή της κατάλληλης τεχνικής εξαρτάται από το είδος των δεδομένων, το μέγεθος του συνόλου των δεδομένων και την πολυπλοκότητα του προβλήματος.

2.1 ADASYN

Ο αλγόριθμος **ADASYN (Adaptive Synthetic Sampling)** είναι μια τεχνική που χρησιμοποιείται για να αντιμετωπίσει το πρόβλημα των μη ισορροπημένων δεδομένων σε προβλήματα ταξινόμησης. Ο σκοπός του είναι να αυξήσει τον αριθμό των δειγμάτων της μειοψηφικής κατηγορίας, δημιουργώντας νέα συνθετικά δεδομένα, δίνοντας παράλληλα μεγαλύτερη προσοχή στις περιπτώσεις που είναι πιο δύσκολο να ταξινομηθούν. Η λειτουργία του βασίζεται σε τρία κύρια στάδια. Πρώτα, υπολογίζεται ένας συντελεστής δυσκολίας για κάθε δείγμα της μειοψηφίας, ανάλογα με το πόσο κοντά βρίσκεται σε δείγματα της πλειονότητας. Όσο πιο κοντά είναι τα δείγματα της πλειονότητας, τόσο μεγαλύτερη είναι η δυσκολία. Στη συνέχεια, με βάση τη δυσκολία αυτή, καθορίζεται πόσα νέα δείγματα θα δημιουργηθούν για κάθε σημείο της μειοψηφίας, δίνοντας μεγαλύτερη έμφαση στις πιο δύσκολες περιπτώσεις. Τέλος, τα νέα συνθετικά δείγματα δημιουργούνται με γραμμική παρεμβολή μεταξύ ενός δείγματος της μειοψηφίας και των γειτόνων του, με τη θέση και την ποσότητά τους να καθορίζονται από τον βαθμό δυσκολίας κάθε δείγματος.

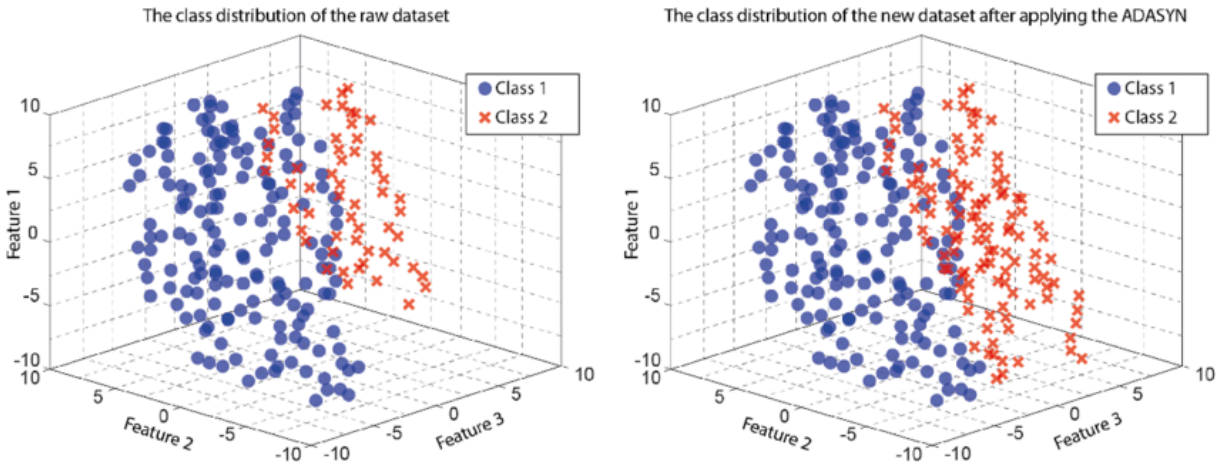
Ο ADASYN προσφέρει αρκετά πλεονεκτήματα. Δημιουργεί συνθετικά δεδομένα εστιάζοντας στις πιο δύσκολες περιοχές, δηλαδή στα σημεία που βρίσκονται κοντά στην πλειονότητα, ενισχύοντας έτσι την ικανότητα του μοντέλου να αναγνωρίζει τις πιο σημαντικές περιπτώσεις. Επίσης, αποφεύγει την υπερβολική παραγωγή δεδομένων σε περιοχές όπου η μειονότητα είναι ήδη καλά κατανεμημένη και επικεντρώνεται σε περιοχές όπου υπάρχουν ελλείψεις. Αυτό βοηθά στη βελτίωση της απόδοσης του ταξινομητή, ειδικά όσον αφορά την ακρίβεια και την ανάκληση, καθώς το μοντέλο εκπαιδεύεται καλύτερα στα πιο δύσκολα παραδείγματα.

Ωστόσο, ο αλγόριθμος έχει και ορισμένα μειονεκτήματα. Μπορεί να δημιουργήσει συνθετικά δείγματα που βρίσκονται κοντά σε σημεία με θόρυβο ή ακραίες τιμές, κάτι που μπορεί να προκαλέσει σύγχυση στο μοντέλο και να μειώσει την απόδοσή του. Επίσης, η υπερβολική έμφαση στις δύσκολες περιοχές μπορεί να οδηγήσει σε υπερπροσαρμογή, δηλαδή το μοντέλο να εστιάσει υπερβολικά σε συγκεκριμένα σημεία και να μην αποδίδει καλά σε νέα δεδομένα. Επιπλέον, η διαδικασία δημιουργίας των νέων δειγμάτων απαιτεί αρκετό υπολογιστικό χρόνο, ειδικά όταν το σύνολο δεδομένων είναι πολύ μεγάλο. Ένας ακόμα περιορισμός είναι ότι τα νέα δείγματα που δημιουργούνται μπορεί να μην αντικατοπτρίζουν πάντα σωστά τη συνολική δομή των δεδομένων, με αποτέλεσμα το μοντέλο να οδηγείται σε λανθασμένα συμπεράσματα.

Στο παρακάτω παράδειγμα δημιουργείται ένα σύνολο δεδομένων με ανισοκατανεμημένες κατηγορίες. Στη συνέχεια, γίνεται διαχωρισμός των δεδομένων σε σύνολα εκπαίδευσης και δοκιμής. Εφαρμόζεται η τεχνική **ADASYN**, η οποία δημιουργεί νέα συνθετικά δείγματα για τη μειονοτική κατηγορία με έμφαση στις περιοχές που είναι πιο δύσκολο να ταξινομηθούν. Ακολουθεί η εκπαίδευση ενός μοντέλου **Random Forest**, η πρόβλεψη των κατηγοριών για τα δεδομένα δοκιμής και η εκτύπωση της ακρίβειας του μοντέλου.

```python
from sklearn.datasets import make_classification
from imblearn.over_sampling import ADASYN
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from collections import Counter

# Δημιουργία ανισοκατανεμημένων δεδομένων
X, y = make_classification(n_samples=1000, n_features=10, weights=[0.9,
0.1], random_state=42)

# Εκτύπωση της κατανομής των κλάσεων πριν την υπερδειγματοληψία
print("Κατανομή πριν την υπερδειγματοληψία:", Counter(y))

# Διαχωρισμός δεδομένων
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εφαρμογή της τεχνικής ADASYN
adasyn = ADASYN(random_state=42)
X_resampled, y_resampled = adasyn.fit_resample(X_train, y_train)

# Εκτύπωση της κατανομής μετά την υπερδειγματοληψία
print("Κατανομή μετά την υπερδειγματοληψία:", Counter(y_resampled))

# Εκπαίδευση και αξιολόγηση μοντέλου
model = RandomForestClassifier(random_state=42)
model.fit(X_resampled, y_resampled)
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Συνοψίζοντας, ο ADASYN είναι μια χρήσιμη τεχνική για την αντιμετώπιση της ανισοκατανομής δεδομένων, καθώς εστιάζει σε περιοχές που είναι δύσκολο να ταξινομηθούν, προσφέροντας καλύτερη ισορροπία μεταξύ ακρίβειας και ανάκλησης και αποφεύγοντας την υπερβολική αναπαραγωγή δεδομένων. Παρόλα αυτά, η χρήση του απαιτεί προσεκτική ρύθμιση και αξιολόγηση, ώστε να αποφευχθούν πιθανά προβλήματα, όπως η υπερπροσαρμογή και η εισαγωγή θορύβου στα δεδομένα. [1], [2]



[1]

2.2 SMOTE

Ο αλγόριθμος **SMOTE (Synthetic Minority Over-sampling Technique)** είναι μια τεχνική που χρησιμοποιείται για να αντιμετωπίσει το πρόβλημα της ανισοκατανομής των δεδομένων στη μηχανική μάθηση. Σε αντίθεση με άλλες μεθόδους που απλώς αντιγράφουν τα υπάρχοντα δεδομένα της μειοψηφικής κατηγορίας, ο SMOTE δημιουργεί νέα, συνθετικά δεδομένα που βασίζονται σε υπάρχοντα σημεία. Ο τρόπος λειτουργίας του περιλαμβάνει τρία βασικά βήματα. Αρχικά, επιλέγεται ένα δείγμα από τη μειοψηφική κατηγορία. Στη συνέχεια, ο αλγόριθμος εντοπίζει τους πλησιέστερους γείτονές του, δηλαδή τα δείγματα που βρίσκονται πιο κοντά σε αυτό. Με βάση αυτούς, δημιουργεί νέα συνθετικά δεδομένα χρησιμοποιώντας έναν απλό μαθηματικό υπολογισμό που υπολογίζει ένα ενδιάμεσο σημείο μεταξύ του αρχικού δείγματος και ενός από τους γείτονές του. Αυτή η διαδικασία επαναλαμβάνεται για όλα τα δείγματα της μειοψηφίας, αυξάνοντας έτσι την παρουσία τους στο σύνολο δεδομένων.

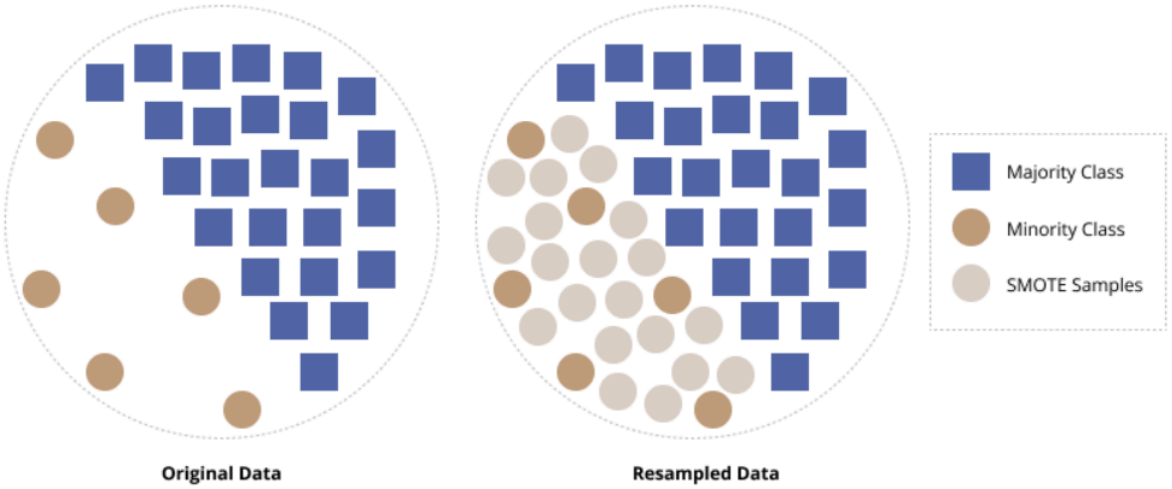
Ένα από τα σημαντικότερα πλεονεκτήματα του SMOTE είναι ότι επεκτείνει τις περιοχές απόφασης του μοντέλου για τη μειοψηφική κατηγορία, καθιστώντας το πιο ικανό να αναγνωρίζει σωστά αυτές τις περιπτώσεις. Επιπλέον, συνδυάζοντας την υπερδειγματοληψία της μειονοτικής κατηγορίας με υποδειγματοληψία από την πλειονότητα, μπορεί να βελτιώσει την απόδοση του μοντέλου, εξισορροπώντας το σύνολο δεδομένων και αποφεύγοντας την προκατάληψη υπέρ της συχνότερης κατηγορίας.

Παρόλα αυτά, υπάρχουν και ορισμένα μειονεκτήματα. Ο SMOTE μπορεί να δημιουργήσει συνθετικά δείγματα κοντά στα όρια των δύο κατηγοριών, γεγονός που ενδέχεται να προκαλέσει θόρυβο και να επηρεάσει αρνητικά την εκπαίδευση του μοντέλου, ειδικά όταν οι κατηγορίες είναι πολύ κοντά μεταξύ τους. Επίσης, η διαδικασία δημιουργίας γενικευμένων περιοχών απόφασης μπορεί να οδηγήσει σε απώλεια ευαισθησίας, καθώς τα συνθετικά δεδομένα μπορεί να μην καλύπτουν πλήρως την ποικιλία των πραγματικών δεδομένων. Ένα ακόμα ζήτημα είναι ότι η τεχνική αυτή έχει σχεδιαστεί κυρίως για προβλήματα δυαδικής κατηγοριοποίησης, και παρόλο που υπάρχουν προσαρμογές για προβλήματα με περισσότερες από δύο κατηγορίες, η απόδοσή της μπορεί να μην είναι εξίσου αποτελεσματική. Τέλος, η επιλογή του αριθμού των γειτόνων που χρησιμοποιούνται στη δημιουργία των συνθετικών δεδομένων είναι κρίσιμη, καθώς εάν ο αριθμός είναι πολύ μεγάλος ή πολύ μικρός, μπορεί να οδηγήσει σε ανακριβή αποτελέσματα ή υπερβολική γενίκευση.

Στο παρακάτω παράδειγμα δημιουργείται ένα σύνολο δεδομένων με ανισοκατανομή κατηγοριών. Τα δεδομένα χωρίζονται σε σύνολα εκπαίδευσης και δοκιμής και στη συνέχεια εφαρμόζεται η τεχνική **SMOTE**, η οποία δημιουργεί νέα δείγματα μέσω παρεμβολής μεταξύ των υπαρχόντων δειγμάτων της μειονοτικής κατηγορίας. Το μοντέλο **Random Forest** εκπαιδεύεται στα εξισορροπημένα δεδομένα και στη συνέχεια αξιολογείται η απόδοσή του μέσω ακρίβειας.

```
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from collections import Counter

# Δημιουργία δεδομένων με ανισοκατανομή κλάσεων
X, y = make_classification(n_samples=1000, n_features=10, weights=[0.9,
0.1], random_state=42)

# Εκτύπωση της κατανομής πριν την υπερδειγματοληψία
print("Κατανομή πριν την υπερδειγματοληψία:", Counter(y))

# Διαχωρισμός δεδομένων
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εφαρμογή της τεχνικής SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Εκτύπωση της κατανομής μετά την υπερδειγματοληψία
print("Κατανομή μετά την υπερδειγματοληψία:", Counter(y_resampled))

# Εκπαίδευση και αξιολόγηση του μοντέλου
model = RandomForestClassifier(random_state=42)
model.fit(X_resampled, y_resampled)
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Συνοψίζοντας, ο SMOTE είναι μια ισχυρή τεχνική για την αντιμετώπιση της ανισοκατανομής των κλάσεων, καθώς βοηθά στη δημιουργία ενός πιο ισορροπημένου συνόλου δεδομένων και βελτιώνει την απόδοση των μοντέλων. Πειραματικές μελέτες έχουν δείξει ότι υπερέχει έναντι άλλων μεθόδων, καθώς αυξάνει τις πιθανότητες δημιουργίας ενός πιο αποδοτικού και σταθερού ταξινομητή. [3]



[2]

2.3 RandomOversampler

Ο αλγόριθμος **RandomOverSampler** είναι μια τεχνική που χρησιμοποιείται για την εξισορρόπηση δεδομένων όταν υπάρχει άνιση κατανομή μεταξύ των κατηγοριών. Ο κύριος στόχος του είναι να αυξήσει τον αριθμό των δειγμάτων της μειοψηφικής κατηγορίας έτσι ώστε να επιτευχθεί ισορροπία μεταξύ των κατηγοριών. Αυτό γίνεται με τη μέθοδο της τυχαίας δειγματοληψίας με αντικατάσταση, που σημαίνει ότι τα υπάρχοντα δείγματα της μειοψηφίας επιλέγονται τυχαία και προστίθενται στο σύνολο δεδομένων, δημιουργώντας περισσότερα αντίγραφα αυτών. Η τεχνική αυτή είναι ιδιαίτερα απλή στην εφαρμογή και κατάλληλη για βασικές ανάγκες εξισορρόπησης, καθώς συνεργάζεται εύκολα με δημοφιλείς βιβλιοθήκες όπως η Scikit-learn και επιτρέπει στον χρήστη να καθορίσει το επίπεδο ισορροπίας μέσω παραμέτρων.

Μια βασική παράμετρος του αλγορίθμου είναι η **sampling_strategy**, η οποία ελέγχει τον αριθμό των δειγμάτων της μειοψηφικής κατηγορίας που θα δημιουργηθούν. Ο χρήστης μπορεί να καθορίσει αυτήν την παράμετρο με διάφορους τρόπους, όπως με τη μορφή αριθμητικής τιμής που δηλώνει την επιθυμητή αναλογία μεταξύ μειοψηφίας και πλειοψηφίας, ως προκαθορισμένο όρισμα που στοχεύει αποκλειστικά στην εξισορρόπηση της μειοψηφίας, ή μέσω προσαρμοσμένων κανόνων που επιτρέπουν λεπτομερή έλεγχο της διαδικασίας. Επιπλέον, υπάρχει η παράμετρος **random_state**, η οποία χρησιμοποιείται για να εξασφαλιστεί η αναπαραγωγιμότητα των αποτελεσμάτων, ενώ στις πιο πρόσφατες εκδόσεις του αλγορίθμου προστέθηκε και ο **shrinkage_factor**, ο οποίος βοηθά στη μείωση της υπερβολικής αύξησης των δεδομένων της μειοψηφίας.

Η τεχνική RandomOverSampler παρουσιάζει αρκετά πλεονεκτήματα. Είναι εύκολη στην εφαρμογή, καθιστώντας την μια γρήγορη λύση για την αντιμετώπιση της ανισοκατανομής. Εξασφαλίζει ότι η μειοψηφική κατηγορία δεν θα παραμείνει υποεκπροσωπούμενη, παρέχοντας περισσότερες ευκαιρίες στο μοντέλο να μάθει από αυτήν. Επίσης, μπορεί να συνδυαστεί εύκολα με άλλες τεχνικές επεξεργασίας δεδομένων και μοντέλα μηχανικής μάθησης, καθιστώντας την ευέλικτη επιλογή για διάφορες εφαρμογές.

Ωστόσο, υπάρχουν και ορισμένα μειονεκτήματα που πρέπει να ληφθούν υπόψη. Επειδή η μέθοδος δημιουργεί αντίγραφα των υπαρχόντων δειγμάτων, υπάρχει αυξημένος κίνδυνος υπερπροσαρμογής,

καθώς το μοντέλο μπορεί να μάθει να εξαρτάται υπερβολικά από τα επαναλαμβανόμενα δεδομένα. Επιπλέον, καθώς αυξάνεται ο αριθμός των δειγμάτων, ο χρόνος εκπαίδευσης του μοντέλου μπορεί να γίνει σημαντικά μεγαλύτερος, κάτι που μπορεί να επηρεάσει την απόδοση του συστήματος σε μεγάλα σύνολα δεδομένων.

Στο παρακάτω παράδειγμα δημιουργείται ένα σύνολο δεδομένων με ανισοκατανομή κατηγοριών. Τα δεδομένα χωρίζονται σε σύνολα εκπαίδευσης και δοκιμής και στη συνέχεια εφαρμόζεται η τεχνική **SMOTE**, η οποία δημιουργεί νέα δείγματα μέσω παρεμβολής μεταξύ των υπαρχόντων δειγμάτων της μειονοτικής κατηγορίας. Το μοντέλο **Random Forest** εκπαιδεύεται στα εξισορροπημένα δεδομένα και στη συνέχεια αξιολογείται η απόδοσή του μέσω ακρίβειας.

```
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from collections import Counter

# Δημιουργία δεδομένων με ανισοκατανομή κλάσεων
X, y = make_classification(n_samples=1000, n_features=10, weights=[0.9,
0.1], random_state=42)

# Εκτύπωση της κατανομής πριν την υπερδειγματοληψία
print("Κατανομή πριν την υπερδειγματοληψία:", Counter(y))

# Διαχωρισμός δεδομένων
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εφαρμογή της τεχνικής SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Εκτύπωση της κατανομής μετά την υπερδειγματοληψία
print("Κατανομή μετά την υπερδειγματοληψία:", Counter(y_resampled))

# Εκπαίδευση και αξιολόγηση του μοντέλου
model = RandomForestClassifier(random_state=42)
model.fit(X_resampled, y_resampled)
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Συνοψίζοντας, ο RandomOverSampler είναι μια απλή και αποτελεσματική τεχνική για την αντιμετώπιση της ανισοκατανομής των κλάσεων, ειδικά όταν απαιτείται γρήγορη και εύκολη εφαρμογή. Παρόλο που μπορεί να προσφέρει σημαντικά οφέλη, η χρήση του πρέπει να γίνεται με

προσοχή ώστε να αποφεύγονται προβλήματα υπερπροσαρμογής και αυξημένης πολυπλοκότητας στην εκπαίδευση του μοντέλου. [4]

Καταλήγοντας, οι τεχνικές **Random Oversampler**, **SMOTE** και **ADASYN** αποτελούν αποτελεσματικά εργαλεία για την αντιμετώπιση της ανισορροπίας δεδομένων, αλλά διαφοροποιούνται σημαντικά στον τρόπο εφαρμογής τους και στα αποτελέσματα που προσφέρουν. Ο **Random Oversampler** προσφέρει μια απλή και εύκολη λύση, ωστόσο δεν εισάγει νέα πληροφορία, περιορίζοντας τη δυνατότητα του μοντέλου να γενικεύσει καλύτερα. Αντίθετα, το **SMOTE** δημιουργεί συνθετικά δείγματα που βασίζονται στη γραμμική παρεμβολή, παρέχοντας μεγαλύτερη ποικιλομορφία στα δεδομένα της μειοψηφικής κατηγορίας, γεγονός που βελτιώνει τη μάθηση του μοντέλου. Τέλος, το **ADASYN**, επεκτείνοντας το SMOTE, εστιάζει στις περιοχές με μεγαλύτερη δυσκολία ταξινόμησης, εξασφαλίζοντας καλύτερη προσαρμογή και πιθανώς υψηλότερη ακρίβεια στις προβλέψεις. Η επιλογή της κατάλληλης τεχνικής εξαρτάται από τις ιδιαιτερότητες του κάθε προβλήματος και τη φύση των δεδομένων.

Κεφάλαιο 3ο: Αλγόριθμοι κατηγοριοποίησης

3.1 Logistic Regression

Ο αλγόριθμος **Logistic Regression** είναι ένα στατιστικό μοντέλο που χρησιμοποιείται για την πρόβλεψη της πιθανότητας ενός δυαδικού αποτελέσματος, δηλαδή όταν ένα δεδομένο μπορεί να ανήκει σε μία από δύο κατηγορίες, όπως "ναι" ή "όχι", "αληθές" ή "ψευδές". Είναι μια από τις πιο δημοφιλείς τεχνικές ταξινόμησης και εφαρμόζεται σε διάφορα προβλήματα, όπως η ανίχνευση απάτης, η διάγνωση ασθενειών και η πρόβλεψη συμπεριφοράς χρηστών. Ο αλγόριθμος υπολογίζει την πιθανότητα ενός δείγματος να ανήκει σε μια συγκεκριμένη κατηγορία, χρησιμοποιώντας τη **λογιστική συνάρτηση (sigmoid function)**, η οποία μετατρέπει οποιαδήποτε αριθμητική τιμή σε μια τιμή μεταξύ 0 και 1, που αντιπροσωπεύει την πιθανότητα της θετικής κλάσης.

Η βασική μαθηματική εξίσωση του Logistic Regression είναι:

$$p = \frac{1}{1 + e^{-z}} \, p = \frac{1}{1 + e^{-z}}$$

Όπου το p είναι η πιθανότητα ότι το δείγμα ανήκει στην κατηγορία "1" και το z είναι ένας γραμμικός συνδυασμός των χαρακτηριστικών εισόδου, ο οποίος υπολογίζεται ως εξής:

$$z = b + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Στην παραπάνω εξίσωση, το b είναι η σταθερά (bias), οι συντελεστές $w_1, w_2, \ldots, w_n$ εκφράζουν τη βαρύτητα των χαρακτηριστικών $x_1, x_2, \ldots, x_n$, και το e είναι η βάση των φυσικών λογαρίθμων.

Η Logistic Regression έχει αρκετά πλεονεκτήματα. Είναι απλή στην κατανόηση και στην υλοποίηση, καθιστώντας την ιδανική επιλογή για προβλήματα ταξινόμησης όπου υπάρχει ανάγκη για ερμηνεύσιμα αποτελέσματα. Επιπλέον, παρέχει εκτιμήσεις πιθανοτήτων που μπορούν να χρησιμοποιηθούν σε διάφορες εφαρμογές, όπως στη λήψη αποφάσεων και στην αξιολόγηση ρίσκου. Είναι επίσης ένα αποδοτικό μοντέλο όταν τα δεδομένα έχουν δυαδικά αποτελέσματα και δεν απαιτεί μεγάλη υπολογιστική ισχύ.

Ωστόσο, η Logistic Regression παρουσιάζει και κάποιους περιορισμούς. Υποθέτει ότι υπάρχει γραμμική σχέση μεταξύ των χαρακτηριστικών και της πιθανότητας του αποτελέσματος, κάτι που μπορεί να μην ισχύει πάντα. Αν τα δεδομένα είναι πολύπλοκα ή περιέχουν μη γραμμικές σχέσεις, το μοντέλο μπορεί να έχει μειωμένη απόδοση. Επίσης, δεν είναι ιδιαίτερα αποτελεσματικό σε προβλήματα με πολλές κλάσεις, καθώς είναι σχεδιασμένο κυρίως για δυαδικά προβλήματα. Αν και υπάρχουν επεκτάσεις όπως η **πολυωνυμική λογιστική παλινδρόμηση (multinomial logistic regression)**, αυτές απαιτούν μεγαλύτερη υπολογιστική ισχύ και περισσότερη πολυπλοκότητα στη ρύθμιση των παραμέτρων. Επιπλέον, η Logistic Regression είναι ευαίσθητη σε ακραίες τιμές και μπορεί να επηρεαστεί αρνητικά από δεδομένα με μεγάλη ανισορροπία μεταξύ των κλάσεων.

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Δημιουργία δεδομένων
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)

# Διαχωρισμός των δεδομένων σε εκπαίδευση και δοκιμή
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εκπαίδευση του μοντέλου
model = LogisticRegression()
model.fit(X_train, y_train)

# Πρόβλεψη και αξιολόγηση
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Ο παραπάνω κώδικας υλοποιεί τη λογιστική παλινδρόμηση χρησιμοποιώντας τη βιβλιοθήκη Sklearn. Αρχικά, δημιουργούνται δεδομένα με δύο κατηγορίες, τα οποία χωρίζονται σε σύνολα εκπαίδευσης και δοκιμής. Στη συνέχεια, το μοντέλο εκπαιδεύεται με τα δεδομένα εκπαίδευσης και γίνεται πρόβλεψη στα δεδομένα δοκιμής. Τέλος, υπολογίζεται η ακρίβεια του μοντέλου.

Παρά τις αδυναμίες της, η Logistic Regression παραμένει μια από τις πιο αξιόπιστες και ευρέως χρησιμοποιούμενες τεχνικές ταξινόμησης, ειδικά όταν η ερμηνευσιμότητα και η απλότητα είναι σημαντικοί παράγοντες. Χρησιμοποιείται ευρέως σε τομείς όπως η χρηματοοικονομική ανάλυση, η ιατρική διάγνωση και το μάρκετινγκ, όπου η δυνατότητα ερμηνείας των αποτελεσμάτων έχει ιδιαίτερη σημασία. [5],[6],[7],[8]

3.2 Decision Trees

Ο αλγόριθμος **Decision Trees** είναι μία από τις πιο βασικές και ευρέως χρησιμοποιούμενες μεθόδους στη μηχανική μάθηση για την επίλυση προβλημάτων ταξινόμησης και παλινδρόμησης. Πρόκειται για

μια διαδικασία λήψης αποφάσεων που απεικονίζεται γραφικά ως ένα διακλαδωμένο δέντρο, όπου κάθε κόμβος αναπαριστά μια συνθήκη βασισμένη σε συγκεκριμένα χαρακτηριστικά, και κάθε διακλάδωση αντιστοιχεί σε μια πιθανή απόφαση. Το δέντρο αποτελείται από διαφορετικούς τύπους κόμβων, με τον **κόμβο ρίζας** να περιέχει το πλήρες σύνολο δεδομένων και να αποτελεί το σημείο εκκίνησης. Καθώς προχωράει η διαδικασία, το σύνολο δεδομένων διασπάται σε μικρότερα υποσύνολα μέσω συνθηκών διαχωρισμού, όπως για παράδειγμα εάν η τιμή ενός χαρακτηριστικού, όπως η ηλικία, είναι μεγαλύτερη ή μικρότερη από μια συγκεκριμένη τιμή. Οι εσωτερικοί κόμβοι αντιπροσωπεύουν επιπλέον διαχωρισμούς, ενώ οι τελικοί κόμβοι, γνωστοί ως **φύλλα**, αντιστοιχούν στην τελική απόφαση ή πρόβλεψη του αλγορίθμου.

Για να προβλέψει το αποτέλεσμα ενός νέου δείγματος, το μοντέλο ακολουθεί μια πορεία από τον κόμβο ρίζας προς ένα φύλλο, λαμβάνοντας αποφάσεις σε κάθε κόμβο με βάση τις τιμές των χαρακτηριστικών. Ο διαχωρισμός των δεδομένων πραγματοποιείται συνήθως χρησιμοποιώντας μετρικές όπως η εντροπία (entropy) και ο δείκτης Gini (Gini Index). Η εντροπία μετρά την αβεβαιότητα των δεδομένων και δίνεται από τον τύπο:

$$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i \, H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

όπου (p_i) είναι η πιθανότητα εμφάνισης μιας κατηγορίας στα δεδομένα. Ο δείκτης Gini, που επίσης χρησιμοποιείται συχνά για τη διαίρεση των δεδομένων, υπολογίζεται ως:(p_i) είναι η πιθανότητα εμφάνισης μιας κατηγορίας στα δεδομένα. Ο δείκτης Gini, που επίσης χρησιμοποιείται συχνά για τη διαίρεση των δεδομένων, υπολογίζεται ως:

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2 \, Gini(S) = 1 - \sum_{i=1}^{n} p_i^2$$

Ο αλγόριθμος Decision Trees εφαρμόζεται σε πολλά διαφορετικά πεδία, όπως στη λήψη αποφάσεων σε χρηματοοικονομικά συστήματα για την έγκριση δανείων, στην ταξινόμηση πελατών και προϊόντων, στην αναγνώριση μοτίβων και στη διάγνωση ασθενειών στον τομέα της ιατρικής.

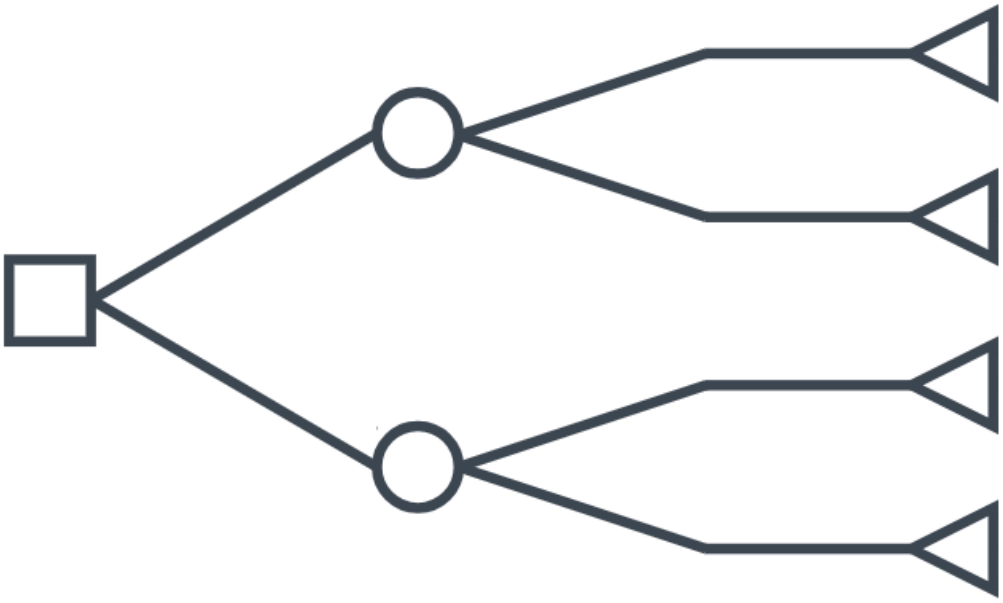
Ένα σημαντικό πλεονέκτημα του αλγορίθμου είναι η απλότητά του, καθώς είναι εύκολα κατανοητός και ερμηνεύσιμος ακόμη και από άτομα χωρίς εξειδικευμένες γνώσεις στατιστικής ή μηχανικής μάθησης. Παρέχει μια ξεκάθαρη οπτική αναπαράσταση της διαδικασίας λήψης αποφάσεων, καθιστώντας τον ιδιαίτερα χρήσιμο για επιχειρηματικές και επιστημονικές εφαρμογές. Επιπλέον, μπορεί να εφαρμοστεί τόσο σε αριθμητικά όσο και σε κατηγορικά δεδομένα, προσφέροντας ευελιξία στη χρήση του.

Ωστόσο, υπάρχουν και ορισμένα μειονεκτήματα. Ένα από τα βασικά προβλήματα των δέντρων απόφασης είναι ότι αν το δέντρο μεγαλώσει πολύ, υπάρχει κίνδυνος υπερπροσαρμογής στα δεδομένα εκπαίδευσης, με αποτέλεσμα να μην μπορεί να γενικεύσει καλά σε νέα δεδομένα. Επιπλέον, η παραμικρή αλλαγή στα δεδομένα μπορεί να οδηγήσει σε σημαντικές διαφοροποιήσεις στη δομή του δέντρου, καθιστώντας το λιγότερο σταθερό.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Δημιουργία δεδομένων
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)

# Διαχωρισμός των δεδομένων σε εκπαίδευση και δοκιμή
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εκπαίδευση του μοντέλου
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Πρόβλεψη και αξιολόγηση
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Στον κώδικα δημιουργούνται τυχαία δεδομένα, τα οποία χωρίζονται σε σύνολα εκπαίδευσης και δοκιμής. Ο ταξινομητής δέντρου απόφασης εκπαιδεύεται με τα δεδομένα εκπαίδευσης και στη συνέχεια πραγματοποιεί προβλέψεις. Η ακρίβεια του μοντέλου αξιολογείται συγκρίνοντας τις προβλέψεις με τις πραγματικές τιμές των δεδομένων δοκιμής.

Συμπερασματικά, τα Decision Trees είναι ένα ισχυρό και ευέλικτο εργαλείο στη μηχανική μάθηση, το οποίο διακρίνεται για την απλότητά του και τη δυνατότητα ερμηνείας των αποτελεσμάτων του. Παρά τα μειονεκτήματα που παρουσιάζει, όπως ο κίνδυνος υπερπροσαρμογής, η σωστή διαχείριση του μοντέλου, σε συνδυασμό με τεχνικές όπως το κλάδεμα (pruning), μπορεί να οδηγήσει σε υψηλής απόδοσης ταξινομητές, καθιστώντας τον αλγόριθμο έναν από τους βασικούς πυλώνες των συστημάτων ανάλυσης δεδομένων. [9]



3.3 Random Forest

Ο αλγόριθμος **Random Forest** είναι μία από τις πιο ισχυρές τεχνικές μηχανικής μάθησης που χρησιμοποιείται τόσο για προβλήματα ταξινόμησης όσο και για προβλήματα πρόβλεψης. Πρόκειται για έναν αλγόριθμο εποπτευόμενης μάθησης που βασίζεται στη δημιουργία πολλαπλών δέντρων απόφασης. Ο κύριος στόχος του είναι να συνδυάσει τις προβλέψεις από πολλά δέντρα απόφασης προκειμένου να βελτιώσει την ακρίβεια και τη σταθερότητα του μοντέλου. Το τελικό αποτέλεσμα προκύπτει μέσω της πλειοψηφικής ψήφου στην περίπτωση ταξινόμησης ή του μέσου όρου των προβλέψεων των δέντρων στην περίπτωση πρόβλεψης.

Η διαδικασία εκπαίδευσης του αλγορίθμου περιλαμβάνει τη δημιουργία πολλαπλών υποσυνόλων δεδομένων από το αρχικό σύνολο εκπαίδευσης μέσω μιας μεθόδου που ονομάζεται **bootstrapping**, όπου επιλέγονται τυχαία δείγματα με αντικατάσταση. Κάθε υποσύνολο χρησιμοποιείται για την εκπαίδευση ενός δέντρου απόφασης, ενώ σε κάθε διαχωρισμό μέσα στο δέντρο επιλέγεται τυχαία ένα υποσύνολο χαρακτηριστικών. Αυτή η τυχαιοποίηση μειώνει τη συσχέτιση μεταξύ των δέντρων και αυξάνει τη συνολική ποικιλομορφία του μοντέλου, καθιστώντας το πιο ανθεκτικό σε προβλήματα υπερπροσαρμογής.

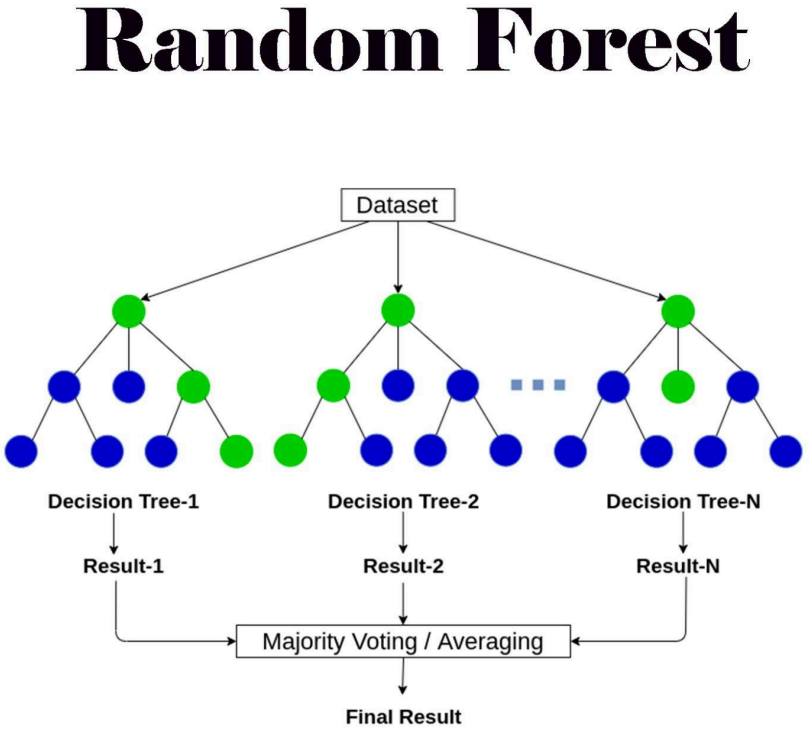
Ένα από τα βασικά πλεονεκτήματα του Random Forest είναι η ικανότητά του να διαχειρίζεται δεδομένα με μεγάλο αριθμό χαρακτηριστικών, ακόμη και όταν οι σχέσεις μεταξύ τους δεν είναι γραμμικές. Επιπλέον, είναι ανθεκτικός σε προβλήματα υπερπροσαρμογής, ειδικά όταν υπάρχει μεγάλος όγκος δεδομένων, καθώς η διαδικασία κατασκευής πολλαπλών δέντρων προσδίδει στο μοντέλο μεγαλύτερη γενίκευση. Ένα άλλο σημαντικό χαρακτηριστικό είναι η δυνατότητα του να παρέχει πληροφορίες σχετικά με τη σημασία των χαρακτηριστικών, κάτι που βοηθά στην κατανόηση του ποια δεδομένα είναι πιο σημαντικά για τις προβλέψεις του μοντέλου.

Ωστόσο, ο αλγόριθμος παρουσιάζει και κάποιους περιορισμούς. Ένα βασικό μειονέκτημα είναι οι υψηλές υπολογιστικές απαιτήσεις του, καθώς απαιτεί περισσότερο χρόνο και πόρους σε σχέση με άλλους απλούστερους αλγορίθμους, ιδιαίτερα όταν εφαρμόζεται σε μεγάλα σύνολα δεδομένων. Επιπλέον, λόγω της πολυπλοκότητάς του, είναι λιγότερο ερμηνεύσιμος σε σύγκριση με τα απλά δέντρα απόφασης, γεγονός που μπορεί να καταστήσει δύσκολη την ανάλυση και κατανόηση των αποτελεσμάτων του.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Δημιουργία δεδομένων
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)

# Διαχωρισμός των δεδομένων σε εκπαίδευση και δοκιμή
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εκπαίδευση του μοντέλου
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Πρόβλεψη και αξιολόγηση
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Το παραπάνω παράδειγμα δείχνει την εφαρμογή του αλγορίθμου Random Forest. Χρησιμοποιείται ένα σύνολο δεδομένων ταξινόμησης, το οποίο χωρίζεται σε εκπαίδευση και δοκιμή. Το μοντέλο εκπαιδεύεται με 100 δέντρα και αξιολογείται μετρώντας την ακρίβεια του στις προβλέψεις.

Συνοψίζοντας, ο Random Forest είναι ένας εξαιρετικά αποδοτικός και ευέλικτος αλγόριθμος που συνδυάζει την απλότητα των δέντρων απόφασης με τη δύναμη των μεθόδων ensemble learning. Παρέχει αξιόπιστες προβλέψεις, είναι ανθεκτικός σε θόρυβο και υπερπροσαρμογή και εφαρμόζεται σε ένα ευρύ φάσμα προβλημάτων μηχανικής μάθησης. Παρά τις υπολογιστικές απαιτήσεις του, τα πλεονεκτήματά του τον καθιστούν ιδιαίτερα δημοφιλή τόσο στην έρευνα όσο και στη βιομηχανία, όπου απαιτούνται ακριβή και αξιόπιστα μοντέλα πρόβλεψης. [10],[11]



[3]

3.4 Support Vector Machine (SVM)

Ο αλγόριθμος **Support Vector Machines (SVM)** είναι μια μέθοδος μηχανικής μάθησης που χρησιμοποιείται κυρίως για την ταξινόμηση δεδομένων. Ο στόχος του είναι να διαχωρίσει διαφορετικές κατηγορίες δεδομένων στο χώρο των χαρακτηριστικών με τον καλύτερο δυνατό τρόπο, αφήνοντας τη μεγαλύτερη δυνατή απόσταση ανάμεσα τους. Αυτή η απόσταση ονομάζεται **περιθώριο (margin)** και βοηθά το μοντέλο να κάνει σωστές προβλέψεις σε νέα δεδομένα.

Το SVM λειτουργεί αναζητώντας ένα **υπερεπίπεδο (hyperplane)** που χωρίζει τα δεδομένα σε δύο κατηγορίες. Ανάλογα με τις διαστάσεις του χώρου των χαρακτηριστικών, το υπερεπίπεδο μπορεί να είναι μια γραμμή, ένα επίπεδο ή ένα πιο πολύπλοκο σχήμα. Τα σημεία που βρίσκονται πιο κοντά στο υπερεπίπεδο ονομάζονται **διανύσματα υποστήριξης (support vectors)** και παίζουν σημαντικό ρόλο στον καθορισμό του διαχωρισμού των κατηγοριών.

Το περιθώριο, που είναι η απόσταση μεταξύ του υπερεπιπέδου και των πλησιέστερων σημείων, υπολογίζεται με τον τύπο:

$$\text{Margin} = \frac{1}{\|w\|} \text{Margin} = \frac{1}{\|w\|}$$

όπου το w είναι οι συντελεστές του υπερεπιπέδου. w είναι οι συντελεστές του υπερεπιπέδου.

Όταν τα δεδομένα δεν μπορούν να διαχωριστούν με μια απλή ευθεία γραμμή, το SVM χρησιμοποιεί μια τεχνική που ονομάζεται **πυρήνας (kernel)**, η οποία μετατρέπει τα δεδομένα σε έναν χώρο υψηλότερων διαστάσεων όπου είναι ευκολότερο να βρεθεί ένας διαχωρισμός. Οι πιο γνωστοί τύποι πυρήνων είναι ο **γραμμικός (linear kernel)**, ο **πολυωνυμικός (polynomial kernel)** και ο **RBF (Radial Basis Function)**, που χρησιμοποιείται συχνά για πιο πολύπλοκα σύνολα δεδομένων.

Ο αλγόριθμος SVM ακολουθεί μερικά βασικά βήματα: αρχικά, τα δεδομένα προετοιμάζονται και κανονικοποιούνται ώστε να έχουν παρόμοιο εύρος τιμών. Στη συνέχεια, το μοντέλο προσπαθεί να βρει το καλύτερο υπερεπίπεδο που διαχωρίζει τις κατηγορίες, και αν χρειαστεί, χρησιμοποιεί πυρήνες για να επιτύχει καλύτερο διαχωρισμό. Στην εκπαίδευση, το μοντέλο βασίζεται στα διανύσματα υποστήριξης για να διαμορφώσει τον τελικό διαχωρισμό.

Ο SVM έχει αρκετά πλεονεκτήματα. Είναι πολύ καλός για προβλήματα με πολλά χαρακτηριστικά και μπορεί να διαχειριστεί ακόμα και περιπτώσεις όπου τα χαρακτηριστικά είναι περισσότερα από τα δεδομένα. Επίσης, μπορεί να χειριστεί μη γραμμικούς διαχωρισμούς, χάρη στη χρήση πυρήνων.

Ωστόσο, έχει και κάποια μειονεκτήματα. Είναι αργός όταν τα δεδομένα είναι πολλά, επειδή οι υπολογισμοί που απαιτούνται είναι χρονοβόροι. Επίσης, για να λειτουργήσει σωστά, χρειάζεται προσεκτική ρύθμιση των παραμέτρων του, όπως η επιλογή του κατάλληλου πυρήνα και των συντελεστών που καθορίζουν την πολυπλοκότητα του μοντέλου.

```
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Δημιουργία δεδομένων
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)

# Διαχωρισμός των δεδομένων σε εκπαίδευση και δοκιμή
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εκπαίδευση του μοντέλου
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Πρόβλεψη και αξιολόγηση
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Η υλοποίηση του αλγορίθμου SVM περιλαμβάνει τη χρήση γραμμικού πυρήνα. Τα δεδομένα δημιουργούνται και χωρίζονται σε εκπαίδευση και δοκιμή. Το μοντέλο SVM εκπαιδεύεται στα

δεδομένα εκπαίδευσης και στη συνέχεια γίνεται αξιολόγηση της απόδοσής του μέσω της ακρίβειας στις προβλέψεις του συνόλου δοκιμής.

Συμπερασματικά, ο αλγόριθμος SVM είναι μια πολύ χρήσιμη μέθοδος για προβλήματα ταξινόμησης, καθώς παρέχει υψηλή ακρίβεια και είναι κατάλληλος τόσο για απλά όσο και για πιο περίπλοκα προβλήματα. Παρόλο που απαιτεί αρκετό χρόνο για την εκπαίδευση, είναι μια από τις πιο αξιόπιστες επιλογές για εφαρμογές που χρειάζονται ακριβή πρόβλεψη και σωστό διαχωρισμό των δεδομένων. [12], [13], [14].

3.5 K-Nearest Neighbors (K-NN)

Ο αλγόριθμος **K-Nearest Neighbors (K-NN)** είναι μια απλή αλλά ισχυρή μέθοδος μηχανικής μάθησης που χρησιμοποιείται κυρίως για ταξινόμηση και, σε ορισμένες περιπτώσεις, για πρόβλεψη αριθμητικών τιμών. Βασίζεται στην αρχή της ομοιότητας, δηλαδή ταξινομεί ένα νέο δεδομένο συγκρίνοντάς το με τα πιο κοντινά σημεία από ένα γνωστό σύνολο δεδομένων. Ένα από τα βασικά χαρακτηριστικά του είναι ότι δεν απαιτεί εκπαίδευση, καθώς αποθηκεύει τα δεδομένα και λαμβάνει αποφάσεις όταν έρθει ένα νέο δείγμα, γι' αυτό και συχνά χαρακτηρίζεται ως αλγόριθμος **"training-free"**.

Η διαδικασία λειτουργίας του K-NN είναι απλή. Όταν δοθεί ένα νέο δείγμα, ο αλγόριθμος υπολογίζει την απόστασή του από όλα τα υπάρχοντα δεδομένα στο σύνολο εκπαίδευσης. Στη συνέχεια, ταξινομεί αυτά τα δεδομένα με βάση την απόστασή τους από το νέο δείγμα και επιλέγει τα **k πιο κοντινά γειτονικά σημεία**. Η τελική ταξινόμηση του νέου δείγματος καθορίζεται από την κατηγορία στην οποία ανήκει η πλειοψηφία των γειτόνων. Αν ο στόχος είναι η πρόβλεψη αριθμητικών τιμών (παλινδρόμηση), τότε ο K-NN επιστρέφει τον μέσο όρο των τιμών των πλησιέστερων γειτόνων.

Η απόσταση μεταξύ των σημείων μπορεί να υπολογιστεί με διάφορους τρόπους, με τις πιο κοινές μεθόδους να είναι η **Ευκλείδεια απόσταση (Euclidean distance)**, η οποία υπολογίζεται ως:

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2 \, d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2$$

η **Μανχάταν απόσταση (Manhattan distance)** και η πιο γενικευμένη **Minkowski απόσταση**, η οποία εξαρτάται από έναν παράγοντα $p : p$:

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Η επιλογή του αριθμού των γειτόνων k είναι πολύ σημαντική, καθώς επηρεάζει την ακρίβεια του αλγορίθμου. Αν η τιμή του k είναι πολύ μικρή, το μοντέλο μπορεί να επηρεαστεί από θόρυβο στα δεδομένα και να οδηγήσει σε υπερπροσαρμογή, ενώ αν το kkk είναι πολύ μεγάλο, μπορεί να αγνοήσει σημαντικά μοτίβα και να δώσει λανθασμένες προβλέψεις.k είναι πολύ σημαντική, καθώς επηρεάζει την ακρίβεια του αλγορίθμου. Αν η τιμή του k είναι πολύ μικρή, το μοντέλο μπορεί να επηρεαστεί από θόρυβο στα δεδομένα και να οδηγήσει σε υπερπροσαρμογή, ενώ αν το kkk είναι πολύ μεγάλο, μπορεί να αγνοήσει σημαντικά μοτίβα και να δώσει λανθασμένες προβλέψεις.

Ο K-NN έχει αρκετά πλεονεκτήματα. Είναι εύκολος στην κατανόηση και στην υλοποίηση, γεγονός που τον καθιστά ιδανικό για αρχάριους στη μηχανική μάθηση. Επειδή δεν απαιτεί διαδικασία εκπαίδευσης, μπορεί να εφαρμοστεί γρήγορα σε μικρά σύνολα δεδομένων. Επιπλέον, είναι

αποτελεσματικός για σύνολα δεδομένων που δεν παρουσιάζουν γραμμική σχέση μεταξύ των χαρακτηριστικών και των κατηγοριών.

Παρόλα αυτά, έχει και ορισμένα μειονεκτήματα. Ένα βασικό πρόβλημα είναι ότι για μεγάλα σύνολα δεδομένων, ο υπολογισμός της απόστασης από όλα τα σημεία μπορεί να είναι πολύ αργός και να απαιτεί σημαντικούς υπολογιστικούς πόρους. Επίσης, είναι ευαίσθητος στην κλίμακα των χαρακτηριστικών, γεγονός που σημαίνει ότι τα χαρακτηριστικά με μεγαλύτερες τιμές μπορεί να επηρεάσουν δυσανάλογα την απόσταση. Για το λόγο αυτό, απαιτείται συχνά **κανονικοποίηση (normalization)** των δεδομένων, ώστε όλα τα χαρακτηριστικά να έχουν την ίδια βαρύτητα.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Δημιουργία δεδομένων
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)

# Διαχωρισμός των δεδομένων σε εκπαίδευση και δοκιμή
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εκπαίδευση του μοντέλου
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

# Πρόβλεψη και αξιολόγηση
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Στο παραπάνω παράδειγμα, χρησιμοποιείται ο αλγόριθμος KNN για την ταξινόμηση δεδομένων. Το μοντέλο εξετάζει τα 5 πιο κοντινά σημεία για κάθε νέο δείγμα και επιστρέφει την πιο κοινή κατηγορία. Το σύνολο δεδομένων χωρίζεται σε εκπαίδευση και δοκιμή, και η απόδοση αξιολογείται μέσω της ακρίβειας.

Συμπερασματικά, ο αλγόριθμος K-NN είναι μια απλή και αποδοτική μέθοδος για την ταξινόμηση και την πρόβλεψη, που βασίζεται στην ομοιότητα μεταξύ των δεδομένων. Αν και είναι υπολογιστικά απαιτητικός για μεγάλα σύνολα δεδομένων, η απλότητά του τον καθιστά ιδανική επιλογή για εισαγωγικές εφαρμογές στη μηχανική μάθηση και για προβλήματα όπου η ερμηνευσιμότητα και η ευκολία εφαρμογής είναι σημαντικοί παράγοντες.[15], [16], [17].

3.6 Gaussian naive Bayes

Ο αλγόριθμος **Gaussian Naive Bayes (GNB)** είναι μια απλή και αποτελεσματική μέθοδος που χρησιμοποιείται κυρίως για την ταξινόμηση δεδομένων. Βασίζεται στο **Θεώρημα του Bayes**, το οποίο χρησιμοποιεί πιθανότητες για να προβλέψει σε ποια κατηγορία ανήκει ένα νέο δείγμα. Ο

αλγόριθμος υπολογίζει την πιθανότητα ενός δείγματος να ανήκει σε κάθε διαθέσιμη κατηγορία και επιλέγει αυτήν με τη μεγαλύτερη πιθανότητα. Ονομάζεται "Naive" επειδή κάνει την υπόθεση ότι τα χαρακτηριστικά των δεδομένων είναι ανεξάρτητα μεταξύ τους, κάτι που συνήθως δεν ισχύει στην πραγματικότητα, αλλά στην πράξη λειτουργεί αρκετά καλά.

Η βασική αρχή του αλγορίθμου είναι το **Θεώρημα του Bayes**, το οποίο υπολογίζει την πιθανότητα ενός γεγονότος A, δεδομένου ότι έχει συμβεί ένα άλλο γεγονός B, σύμφωνα με τον τύπο:A, δεδομένου ότι έχει συμβεί ένα άλλο γεγονός B, σύμφωνα με τον τύπο:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Εδώ, $P(A|B)$ είναι η πιθανότητα του γεγονότος A υπό την προϋπόθεση ότι έχει συμβεί το B, $P(B|A)$ είναι η πιθανότητα του γεγονότος B όταν ισχύει το A, $P(A)$ είναι η αρχική πιθανότητα του A, και $P(B)$ είναι η πιθανότητα του B.$P(A|B)$ είναι η πιθανότητα του γεγονότος A υπό την προϋπόθεση ότι έχει συμβεί το B, $P(B|A)$ είναι η πιθανότητα του γεγονότος B όταν ισχύει το A, $P(A)$ είναι η αρχική πιθανότητα του A, και $P(B)$ είναι η πιθανότητα του B.

Ο Gaussian Naive Bayes χρησιμοποιείται όταν τα δεδομένα ακολουθούν μια **κανονική κατανομή**, δηλαδή τα χαρακτηριστικά τους έχουν τιμές που συγκεντρώνονται γύρω από έναν μέσο όρο. Η πιθανότητα εμφάνισης μιας τιμής xxx σε μια κανονική κατανομή υπολογίζεται από την εξίσωση:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

όπου το $$ μ $$ είναι η μέση τιμή των δεδομένων και το $$ \sigma^2 $$ η διασπορά τους.

Για να ταξινομήσει ένα νέο δείγμα, ο αλγόριθμος υπολογίζει πρώτα τη μέση τιμή και τη διασπορά για κάθε κατηγορία χρησιμοποιώντας τα δεδομένα εκπαίδευσης. Στη συνέχεια, εφαρμόζει την εξίσωση της κανονικής κατανομής για να υπολογίσει την πιθανότητα κάθε χαρακτηριστικού. Με βάση το Θεώρημα του Bayes, υπολογίζει τη συνολική πιθανότητα για κάθε κατηγορία και τελικά το δείγμα ταξινομείται στην κατηγορία με τη μεγαλύτερη πιθανότητα.

Ο Gaussian Naive Bayes έχει αρκετά πλεονεκτήματα. Είναι πολύ γρήγορος και μπορεί να εφαρμοστεί εύκολα ακόμα και σε μεγάλα σύνολα δεδομένων. Λειτουργεί καλά σε προβλήματα με πολλά χαρακτηριστικά, όπως η ταξινόμηση κειμένων και η ανάλυση συναισθήματος. Χρειάζεται ελάχιστο χρόνο εκπαίδευσης, επειδή χρησιμοποιεί απλές πιθανότητες, γεγονός που τον καθιστά ιδανικό για εφαρμογές όπου η ταχύτητα είναι κρίσιμη.

Παρόλα αυτά, υπάρχουν και μειονεκτήματα. Η υπόθεση ότι τα χαρακτηριστικά είναι ανεξάρτητα μεταξύ τους μπορεί να οδηγήσει σε λάθη αν τα δεδομένα είναι συσχετισμένα. Επίσης, αν τα δεδομένα δεν ακολουθούν την κανονική κατανομή, η ακρίβεια του αλγορίθμου μπορεί να μειωθεί. Είναι επίσης ευαίσθητος σε ακραίες τιμές, οι οποίες μπορούν να επηρεάσουν τα αποτελέσματα.

Ο Gaussian Naive Bayes έχει πολλές εφαρμογές σε διάφορους τομείς. Χρησιμοποιείται στο φιλτράρισμα email για τον εντοπισμό ανεπιθύμητων μηνυμάτων, στην ανάλυση κειμένου για τον εντοπισμό θετικών ή αρνητικών σχολίων και στην ιατρική για τη διάγνωση ασθενειών με βάση τα συμπτώματα των ασθενών. Είναι επίσης χρήσιμος σε συστήματα σύστασης, όπου προτείνει προϊόντα με βάση τις προτιμήσεις του χρήστη.

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Δημιουργία δεδομένων
X, y = make_classification(n_samples=1000, n_features=10,
random_state=42)

# Διαχωρισμός των δεδομένων σε εκπαίδευση και δοκιμή
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εκπαίδευση του μοντέλου
model = GaussianNB()
model.fit(X_train, y_train)

# Πρόβλεψη και αξιολόγηση
y_pred = model.predict(X_test)
print(f'Ακρίβεια: {accuracy_score(y_test, y_pred):.2f}')
```

Ο αλγόριθμος Gaussian Naive Bayes εφαρμόζεται σε σύνολο δεδομένων ταξινόμησης. Το μοντέλο υποθέτει ότι τα δεδομένα ακολουθούν κανονική κατανομή και χρησιμοποιεί το Θεώρημα του Bayes για να υπολογίσει τις πιθανότητες των κατηγοριών. Η αξιολόγηση του μοντέλου γίνεται μέσω της ακρίβειας των προβλέψεων.

Συνοψίζοντας, ο Gaussian Naive Bayes είναι ένας απλός και αποτελεσματικός αλγόριθμος ταξινόμησης, που μπορεί να εφαρμοστεί γρήγορα σε πολλά προβλήματα. Παρά το γεγονός ότι κάνει την υπόθεση της ανεξαρτησίας των χαρακτηριστικών, η απόδοσή του είναι ικανοποιητική σε πολλές περιπτώσεις. Είναι ιδανικός για εφαρμογές που απαιτούν ταχύτητα και ευκολία, καθιστώντας τον μια από τις πιο δημοφιλείς επιλογές στη μηχανική μάθηση.

Τέλος, η επιλογή του κατάλληλου αλγορίθμου μηχανικής μάθησης εξαρτάται από τα χαρακτηριστικά των δεδομένων και τις απαιτήσεις του προβλήματος. Για απλά, γραμμικά προβλήματα, αλγόριθμοι όπως ο **Logistic Regression** και ο **Gaussian Naive Bayes** είναι κατάλληλοι. Αντίθετα, για πιο περίπλοκα προβλήματα με μη γραμμικά σύνολα δεδομένων, αλγόριθμοι όπως ο **Random Forest** και το **SVM** προσφέρουν καλύτερη ακρίβεια και ανθεκτικότητα. Τα **Decision Trees**, από την άλλη, παρέχουν διαφάνεια και ευκολία στην ερμηνεία των αποτελεσμάτων.

Κεφάλαιο 4ο: Υπερδειγματοληψία με χρήση της Python

4.1 Sklearn (Scikit-learn)

Η **Scikit-learn (sklearn)** είναι μια βιβλιοθήκη της Python που χρησιμοποιείται στη μηχανική μάθηση. Παρέχει πολλά εργαλεία για την ανάλυση δεδομένων, την εκπαίδευση μοντέλων και την αξιολόγηση των αποτελεσμάτων. Είναι εύκολη στη χρήση και πολύ αποδοτική, καθιστώντας την ιδανική τόσο για αρχάριους όσο και για πιο έμπειρους χρήστες. Με τη βοήθειά της, μπορούμε να εφαρμόσουμε τεχνικές υπερδειγματοληψίας για να αντιμετωπίσουμε την ανισοκατανομή των δεδομένων, δηλαδή όταν μία κατηγορία έχει πολύ λιγότερα δείγματα από τις άλλες.

Η βιβλιοθήκη sklearn περιλαμβάνει αρκετές τεχνικές υπερδειγματοληψίας, όπως το **SMOTE (Synthetic Minority Over-sampling Technique), ADASYN (Adaptive Synthetic Sampling)** και ο **Random Oversampler**. Αυτές οι τεχνικές δημιουργούν νέα συνθετικά δεδομένα, ώστε να εξισορροπήσουν το σύνολο δεδομένων και να βοηθήσουν τους αλγορίθμους ταξινόμησης να εκπαιδευτούν καλύτερα.

Ας δούμε ένα απλό παράδειγμα χρήσης της **sklearn** και της βιβλιοθήκης **imbalanced-learn**, που χρησιμοποιείται συχνά για την υπερδειγματοληψία.

```
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Δημιουργία ενός συνόλου δεδομένων με ανισοκατανομή κλάσεων
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2,
                           weights=[0.9, 0.1], random_state=42)

# Διαχωρισμός των δεδομένων σε σύνολα εκπαίδευσης και δοκιμής
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Εφαρμογή της τεχνικής SMOTE για εξισορρόπηση των κατηγοριών
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Εκπαίδευση μοντέλου Random Forest
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_resampled, y_resampled)

# Πρόβλεψη στο σύνολο δοκιμής
y_pred = classifier.predict(X_test)

# Υπολογισμός της ακρίβειας
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Ακρίβεια του μοντέλου: {accuracy:.2f}')
```

Στο παραπάνω παράδειγμα, αρχικά δημιουργούμε ένα ψεύτικο σύνολο δεδομένων, όπου μία κατηγορία είναι πολύ λιγότερη από την άλλη. Χωρίζουμε τα δεδομένα σε εκπαίδευση και δοκιμή. Στη συνέχεια, εφαρμόζουμε την τεχνική **SMOTE**, η οποία δημιουργεί νέα συνθετικά δείγματα για την κατηγορία με τα λιγότερα δεδομένα. Μετά την εξισορρόπηση των κατηγοριών, εκπαιδεύουμε ένα μοντέλο **Random Forest** και κάνουμε προβλέψεις. Τέλος, υπολογίζουμε την ακρίβεια του μοντέλου.

Αυτό το παράδειγμα δείχνει πόσο εύκολο είναι να χρησιμοποιήσουμε την **sklearn** και την **imbalanced-learn** για να βελτιώσουμε τα αποτελέσματα σε ανισοκατανεμημένα δεδομένα. Η εφαρμογή τεχνικών όπως το SMOTE μπορεί να βοηθήσει σημαντικά τους αλγορίθμους να αναγνωρίσουν καλύτερα τις κατηγορίες και να αυξήσουν την ακρίβεια των προβλέψεων.

Η **sklearn** είναι πολύ χρήσιμη γιατί επιτρέπει την εύκολη ενσωμάτωση αυτών των τεχνικών με άλλους αλγόριθμους ταξινόμησης, όπως η **λογιστική παλινδρόμηση (Logistic Regression), τα Δέντρα Απόφασης (Decision Trees),** και οι **Μηχανές Διανυσμάτων Υποστήριξης (SVMs).** Έτσι, μπορούμε να δοκιμάσουμε διαφορετικές τεχνικές και να συγκρίνουμε τα αποτελέσματα.

Συμπερασματικά, η sklearn αποτελεί ένα ισχυρό εργαλείο που επιτρέπει την εφαρμογή τεχνικών εξισορρόπησης και τη δημιουργία αξιόπιστων μοντέλων με ελάχιστη προσπάθεια, ακόμα και όταν έχουμε δεδομένα με άνιση κατανομή κατηγοριών.

4.2 Τεχνικές υπερδειγματοληψίας στην Python

Στο κεφάλαιο 2 παρουσιάστηκαν οι πιο διαδεδομένες τεχνικές υπερδειγματοληψίας, οι οποίες χρησιμοποιούνται για την αντιμετώπιση της ανισοκατανομής των κατηγοριών στα δεδομένα ταξινόμησης. Οι τεχνικές αυτές περιλαμβάνουν το **ADASYN (Adaptive Synthetic Sampling), το SMOTE (Synthetic Minority Over-sampling Technique)** και τον **Random Oversampler**, οι οποίες δημιουργούν συνθετικά δείγματα για τη μειονοτική κατηγορία, βελτιώνοντας έτσι την απόδοση των αλγορίθμων ταξινόμησης.

Στην Python, οι τεχνικές αυτές υλοποιούνται μέσω της βιβλιοθήκης **imbalanced-learn**, η οποία παρέχει εξειδικευμένες συναρτήσεις για την εξισορρόπηση των δεδομένων. Στη συνέχεια, θα δούμε τον κώδικα που αναπτύξαμε για τις πειραματικές δοκιμές, όπου εφαρμόζονται οι παραπάνω τεχνικές σε πραγματικά δεδομένα. Ο κώδικας περιλαμβάνει τη δημιουργία και επεξεργασία των δεδομένων, την εφαρμογή των τεχνικών υπερδειγματοληψίας και την αξιολόγηση των αποτελεσμάτων μετά την εξισορρόπηση. Μέσα από αυτά τα πειράματα, εξετάζεται η αποτελεσματικότητα των τεχνικών αυτών και συγκρίνονται οι επιδόσεις των αλγορίθμων ταξινόμησης σε διαφορετικά σενάρια.

```
oversampling_algorithms = [
      ("Baseline", None),
```

```
      ("Random Oversampler", RandomOverSampler(random_state=42)),
      ("SMOTE Overampler", SMOTE(random_state=42)),
      ("SMOTEN Oversampler", SMOTEN(random_state=42)),
      ("KMEANSSMOTE Oversampler", KMeansSMOTE(random_state=42)),
      ("SVM SMOTE Oversampler", SVMSMOTE(random_state=42)),
      ("ADASYN Oversampler", ADASYN(random_state=42)),
]
```

Η λίστα `oversampling_algorithms` περιέχει ζεύγη που αντιστοιχούν στις ονομασίες και τις αντίστοιχες υλοποιήσεις διαφόρων τεχνικών υπερδειγματοληψίας (oversampling) στη βιβλιοθήκη **imbalanced-learn**. Κάθε στοιχείο περιλαμβάνει το όνομα της τεχνικής και το αντίστοιχο αντικείμενο κλάσης της, επιτρέποντας τη δοκιμή διαφορετικών μεθόδων για την εξισορρόπηση των δεδομένων. Το στοιχείο "Baseline" χρησιμοποιείται ως σημείο αναφοράς χωρίς εφαρμογή υπερδειγματοληψίας.

```
def oversample(classification_algorithm_name, directory,
oversampling_algorithm_name, oversampling_impl, x_train,
            y_train):
      if oversampling_algorithm_name != "Baseline":
      try:
            x_res, y_res = oversampling_impl.fit_resample(x_train,
y_train)
      except Exception as e:
            print(
                  f"An exception occurred for Oversampling alg:
{oversampling_algorithm_name}\nClassification alg:
{classification_algorithm_name}\nDataset name: {directory}\n\n")
            print(e)
            print("========================================")
      else:
      x_res, y_res = x_train, y_train
      return x_res, y_res
```

Η συνάρτηση `oversample` χρησιμοποιείται για την εφαρμογή τεχνικών υπερδειγματοληψίας σε δεδομένα εκπαίδευσης με σκοπό την εξισορρόπηση των κλάσεων στα ανισοκατανεμημένα σύνολα δεδομένων. Όταν ένα σύνολο δεδομένων περιέχει πολύ περισσότερα δείγματα από μία κατηγορία σε σύγκριση με τις υπόλοιπες, οι αλγόριθμοι κατηγοριοποίησης τείνουν να έχουν μειωμένη απόδοση στην αναγνώριση της μειονοτικής κατηγορίας. Η εφαρμογή τεχνικών υπερδειγματοληψίας βοηθά στη δημιουργία νέων, συνθετικών δεδομένων της μειονοτικής κατηγορίας, καθιστώντας το σύνολο δεδομένων πιο ισορροπημένο και διευκολύνοντας τη διαδικασία εκπαίδευσης του ταξινομητή.

Η συνάρτηση δέχεται ως είσοδο το όνομα του αλγορίθμου ταξινόμησης, το όνομα του συνόλου δεδομένων που χρησιμοποιείται, το όνομα της τεχνικής υπερδειγματοληψίας που εφαρμόζεται, την αντίστοιχη υλοποίησή της, καθώς και τα δεδομένα εκπαίδευσης που περιλαμβάνουν τα χαρακτηριστικά και τις ετικέτες κατηγοριοποίησης. Το όνομα του αλγορίθμου ταξινόμησης χρησιμοποιείται για λόγους καταγραφής και αναφοράς, ώστε να μπορεί να προσδιοριστεί ποιος

αλγόριθμος εφαρμόστηκε σε ποιο σύνολο δεδομένων. Το όνομα του συνόλου δεδομένων υποδεικνύει από ποιο αρχείο προέρχονται τα δεδομένα, επιτρέποντας την καταγραφή των αποτελεσμάτων της διαδικασίας υπερδειγματοληψίας για μελλοντική αναφορά. Το όνομα της τεχνικής υπερδειγματοληψίας υποδεικνύει ποια μέθοδος χρησιμοποιείται, ενώ η παράμετρος `oversampling_impl` περιέχει το αντίστοιχο αντικείμενο υλοποίησης της μεθόδου, το οποίο θα κληθεί για να εκτελέσει την υπερδειγματοληψία στα δεδομένα.

Η συνάρτηση αρχικά ελέγχει αν η τεχνική υπερδειγματοληψίας που επιλέχθηκε είναι η βασική προσέγγιση "Baseline". Εάν το όνομα της τεχνικής είναι διαφορετικό από το "Baseline", η συνάρτηση προσπαθεί να εφαρμόσει την τεχνική υπερδειγματοληψίας χρησιμοποιώντας τη μέθοδο `fit_resample` στο σύνολο δεδομένων εκπαίδευσης, που αποτελείται από τα χαρακτηριστικά και τις ετικέτες. Η μέθοδος αυτή λαμβάνει τα αρχικά δεδομένα ως είσοδο και επιστρέφει ένα νέο σύνολο δεδομένων όπου η μειονοτική κατηγορία έχει εμπλουτιστεί με συνθετικά δεδομένα, προκειμένου να επιτευχθεί ισορροπία με την πλειοψηφική κατηγορία. Η διαδικασία αυτή βασίζεται σε τεχνικές όπως η δημιουργία συνθετικών δεδομένων μέσω παρεμβολής μεταξύ υπαρχόντων σημείων ή η τυχαία αντιγραφή των δεδομένων της μειονοτικής κατηγορίας.

Εάν παρουσιαστεί οποιοδήποτε σφάλμα κατά τη διαδικασία εφαρμογής της τεχνικής υπερδειγματοληψίας, η συνάρτηση το χειρίζεται μέσω ενός μηχανισμού εξαίρεσης (exception handling). Σε περίπτωση σφάλματος, εμφανίζεται στην οθόνη ένα μήνυμα που περιλαμβάνει πληροφορίες σχετικά με την τεχνική υπερδειγματοληψίας που χρησιμοποιήθηκε, τον αλγόριθμο ταξινόμησης που εκτελείται και το όνομα του συνόλου δεδομένων. Επίσης, εκτυπώνεται το μήνυμα του σφάλματος, παρέχοντας πληροφορίες για την αιτία του προβλήματος, και προστίθεται μια διαχωριστική γραμμή για τη βελτίωση της αναγνωσιμότητας.

Εάν η επιλεγμένη τεχνική υπερδειγματοληψίας είναι η "Baseline", που σημαίνει ότι δεν εφαρμόζεται κάποια συγκεκριμένη μέθοδος εξισορρόπησης, η συνάρτηση επιστρέφει τα αρχικά δεδομένα εκπαίδευσης χωρίς καμία τροποποίηση. Με τον τρόπο αυτό, διατηρείται η δυνατότητα σύγκρισης των επιδόσεων των αλγορίθμων κατηγοριοποίησης με και χωρίς υπερδειγματοληψία, ώστε να αξιολογηθεί η πραγματική επίδραση των τεχνικών εξισορρόπησης στις επιδόσεις των ταξινομητών.

Η συνάρτηση επιστρέφει τα δεδομένα εκπαίδευσης μετά την εφαρμογή της υπερδειγματοληψίας, εάν αυτή πραγματοποιήθηκε επιτυχώς, ή τα αρχικά δεδομένα σε περίπτωση που επιλέχθηκε η βασική προσέγγιση. Αυτή η διαδικασία είναι χρήσιμη καθώς επιτρέπει την αυτοματοποίηση της διαδικασίας εξισορρόπησης των δεδομένων σε πειραματικές διαδικασίες, επιτρέποντας τη δοκιμή και σύγκριση διαφορετικών τεχνικών σε διάφορα σύνολα δεδομένων. Εξασφαλίζει επίσης ότι η διαδικασία συνεχίζεται χωρίς διακοπή, ακόμα και αν μια συγκεκριμένη τεχνική υπερδειγματοληψίας δεν είναι εφαρμόσιμη για ένα συγκεκριμένο σύνολο δεδομένων.

4.3 Αλγόριθμοι κατηγοριοποίησης στην Python

Η Python προσφέρει μια πληθώρα αλγορίθμων κατηγοριοποίησης μέσω της βιβλιοθήκης **Scikit-learn (sklearn)**, η οποία παρέχει εύχρηστα εργαλεία για την εκπαίδευση και αξιολόγηση μοντέλων. Οι αλγόριθμοι κατηγοριοποίησης χρησιμοποιούνται για να προβλέψουν σε ποια κατηγορία ανήκει ένα

δείγμα, βασιζόμενοι σε χαρακτηριστικά που τους δίνονται ως είσοδος. Ανάλογα με τη φύση των δεδομένων, διαφορετικοί αλγόριθμοι μπορούν να αποδώσουν καλύτερα.

Στην παρούσα ενότητα, παρουσιάζουμε τον κώδικα που αναπτύχθηκε για την ανάγνωση των δεδομένων και την κατηγοριοποίησή τους, χρησιμοποιώντας τη βιβλιοθήκη **sklearn**. Η διαδικασία περιλαμβάνει τη φόρτωση των δεδομένων, την προεπεξεργασία τους, καθώς και την εφαρμογή διαφορετικών αλγορίθμων κατηγοριοποίησης για την εξαγωγή αποτελεσμάτων. Ο κώδικας έχει σχεδιαστεί ώστε να επιτρέπει την εύκολη εναλλαγή μεταξύ διαφορετικών αλγορίθμων, παρέχοντας ένα συνεκτικό και αποδοτικό πλαίσιο για την ανάλυση και αξιολόγηση της απόδοσής τους. Στη συνέχεια, ακολουθεί η αναλυτική παρουσίαση του κώδικα που χρησιμοποιήθηκε στα πειράματα.

```
def get_column_names(file_as_string: str):
      # Regular expression pattern to extract attribute names
      pattern = r'@attribute\s+(\S+)'

      matches = re.findall(pattern, file_as_string)
      column_names = []
      for match in matches:
      column_names.append(match)
      return column_names
```

Η συνάρτηση `get_column_names` είναι υπεύθυνη για την εξαγωγή των ονομάτων των στηλών από ένα αρχείο δεδομένων σε μορφή ARFF. Τα αρχεία ARFF χρησιμοποιούνται συχνά στη μηχανική μάθηση και περιέχουν πληροφορίες σχετικά με τα χαρακτηριστικά των δεδομένων μέσω γραμμών που ξεκινούν με τη λέξη-κλειδί `@attribute`. Η συνάρτηση δέχεται ως είσοδο το περιεχόμενο του αρχείου ως συμβολοσειρά και χρησιμοποιεί έναν μηχανισμό κανονικών εκφράσεων (regular expressions) για την ανάλυσή του.

Αρχικά, ορίζεται ένα πρότυπο αναζήτησης με τη μορφή της κανονικής έκφρασης `r'@attribute\s+(\S+)'`. Η συγκεκριμένη έκφραση λειτουργεί ως εξής:

- Το `@attribute` αναζητά τις γραμμές που περιέχουν τον συγκεκριμένο όρο, ο οποίος υποδηλώνει ότι ακολουθεί το όνομα μιας στήλης.
- Το `\s+` αντιπροσωπεύει ένα ή περισσότερα κενά διαστήματα που χωρίζουν τη λέξη-κλειδί από το όνομα της στήλης.
- Το `(\S+)` συλλαμβάνει το πρώτο μη κενό σύμβολο που ακολουθεί, το οποίο αντιστοιχεί στο όνομα του χαρακτηριστικού.

Αφού εφαρμοστεί η αναζήτηση μέσω της συνάρτησης `re.findall(pattern, file_as_string)`, όλα τα αποτελέσματα αποθηκεύονται στη λίστα `matches`, η οποία περιέχει τα ονόματα των στηλών που βρέθηκαν στο αρχείο. Στη συνέχεια, δημιουργείται μια κενή λίστα με το όνομα `column_names`, στην οποία αποθηκεύονται διαδοχικά τα ονόματα των στηλών που έχουν ανιχνευθεί μέσω ενός βρόχου `for`. Κάθε στοιχείο της λίστας `matches` προστίθεται στη λίστα `column_names`, η οποία επιστρέφεται ως τελικό αποτέλεσμα της συνάρτησης.

Με την εκτέλεση αυτής της συνάρτησης, μπορούμε να ανακτήσουμε τα ονόματα των χαρακτηριστικών από το αρχείο και να τα χρησιμοποιήσουμε αργότερα κατά τη διαδικασία φόρτωσης και ανάλυσης των δεδομένων.

```
def read_data(file_path: str) -> pd.DataFrame:
      with open(file_path, 'r') as file:
      file_content = file.read()
      data_start_index = file_content.find('@data') + len('@data') + 1
      column_names = get_column_names(file_content)
      data_content = file_content[data_start_index:]
      return pd.read_csv(StringIO(data_content), header=None,
names=column_names)
```

Η συνάρτηση `read_data` είναι υπεύθυνη για την ανάγνωση δεδομένων από ένα αρχείο ARFF και την επιστροφή τους σε μορφή **pandas DataFrame**. Τα αρχεία ARFF χρησιμοποιούνται ευρέως σε προβλήματα μηχανικής μάθησης και περιέχουν τόσο πληροφορίες για τα χαρακτηριστικά των δεδομένων όσο και τα ίδια τα δεδομένα. Η συνάρτηση δέχεται ως είσοδο το μονοπάτι προς το αρχείο και ακολουθεί τα παρακάτω βήματα για την επεξεργασία του.

Αρχικά, ανοίγει το αρχείο σε λειτουργία ανάγνωσης (`'r'`) και διαβάζει ολόκληρο το περιεχόμενό του στη μεταβλητή `file_content`. Στη συνέχεια, εντοπίζει τη θέση έναρξης των δεδομένων αναζητώντας τη λέξη-κλειδί `@data` μέσα στο περιεχόμενο του αρχείου. Η θέση αποθηκεύεται στη μεταβλητή `data_start_index`, η οποία περιλαμβάνει το μήκος της λέξης-κλειδί ώστε να παραληφθεί από το κείμενο και να παραμείνουν μόνο τα δεδομένα.

Έπειτα, η συνάρτηση καλεί τη βοηθητική συνάρτηση `get_column_names`, η οποία εξάγει τα ονόματα των χαρακτηριστικών από το περιεχόμενο του αρχείου. Αυτή η διαδικασία επιτρέπει τη δημιουργία του συνόλου δεδομένων με τις κατάλληλες επικεφαλίδες στηλών.

Το πραγματικό περιεχόμενο των δεδομένων αποθηκεύεται στη μεταβλητή `data_content`, το οποίο αποτελεί το τμήμα του αρχείου μετά την ετικέτα `@data`. Στη συνέχεια, χρησιμοποιείται η συνάρτηση `pd.read_csv` της βιβλιοθήκης **pandas**, η οποία διαβάζει τα δεδομένα από το κείμενο και τα μετατρέπει σε **DataFrame**. Οι παράμετροι `header=None` και `names=column_names` διασφαλίζουν ότι τα δεδομένα φορτώνονται χωρίς προϋπάρχουσα κεφαλίδα, και τα ονόματα των στηλών ορίζονται από τη λίστα που εξήχθη προηγουμένως.

Η συνάρτηση επιστρέφει το τελικό DataFrame, το οποίο μπορεί να χρησιμοποιηθεί για περαιτέρω ανάλυση και επεξεργασία. Συνολικά, η συνάρτηση αυτή διευκολύνει τη φόρτωση αρχείων ARFF στην Python και εξασφαλίζει ότι τα δεδομένα είναι έτοιμα για ανάλυση σε μορφή κατάλληλη για εφαρμογές μηχανικής μάθησης.

```
datasets = ["data/ecoli1/ecoli1.dat",
            "data/glass0/glass0.dat",
```

```
            "data/page-blocks0/page-blocks0.dat",
            "data/iris0/iris0.dat",
            "data/new-thyroid1/new-thyroid1.dat",
            "data/pima/pima.dat",
            "data/haberman/haberman.dat",
            "data/segment0/segment0.dat",
            "data/vehicle0/vehicle0.dat",
            "data/wisconsin/wisconsin.dat",
            "data/yeast1/yeast1.dat"
            ]
```

Η λίστα `datasets` περιέχει τα ονόματα των αρχείων δεδομένων που θα χρησιμοποιηθούν στα πειράματα. Κάθε στοιχείο της λίστας αντιστοιχεί στη διαδρομή ενός αρχείου που περιέχει δεδομένα σε μορφή ARFF, τα οποία θα φορτωθούν και θα επεξεργαστούν.

```
def scale_column(column_data):
      data_reshaped = np.array(column_data).reshape(-1, 1)
      # Fit and transform the data
      return scaler.fit_transform(data_reshaped)
```

Η συνάρτηση `scale_column` είναι υπεύθυνη για την κανονικοποίηση μιας στήλης δεδομένων χρησιμοποιώντας τον **scaler** της βιβλιοθήκης `sklearn`. Αρχικά, τα δεδομένα της στήλης μετατρέπονται σε έναν πίνακα δύο διαστάσεων με χρήση της συνάρτησης `reshape(-1,1)`, ώστε να μπορούν να γίνουν αποδεκτά από τον κανονικοποιητή. Στη συνέχεια, εφαρμόζεται η διαδικασία κλιμάκωσης, η οποία προσαρμόζει και μετασχηματίζει τα δεδομένα ώστε να βρίσκονται σε μια συγκεκριμένη κλίμακα τιμών, βελτιώνοντας την απόδοση των αλγορίθμων ταξινόμησης.

```
def scale_table(data):
      scaling_start_time = time.time()
      for column in list(data):
      if column == "Class":
            continue
      data[column] = scale_column(data[column])
      scaling_completion_time = time.time() - scaling_start_time
      print(f"--- Took {scaling_completion_time} seconds to scale the
---")
      return data, scaling_completion_time
```

Η συνάρτηση `read_and_save_folds` είναι υπεύθυνη για την ανάγνωση των δεδομένων από ένα αρχείο, την κανονικοποίησή τους και τη δημιουργία πολλαπλών πτυχών (folds) για χρήση σε τεχνικές διασταυρούμενης επικύρωσης. Αρχικά, εξάγεται το όνομα του συνόλου δεδομένων από το όνομα του αρχείου, το οποίο διαβάζεται μέσω της συνάρτησης `read_data`. Μετά την ανάγνωση, τα δεδομένα κλιμακώνονται χρησιμοποιώντας τη συνάρτηση `scale_table`, ενώ ο χρόνος κανονικοποίησης αποθηκεύεται σε ένα αρχείο με κατάληξη `scaling_time.txt`. Στη συνέχεια, διαχωρίζονται τα χαρακτηριστικά από την ετικέτα κατηγορίας και εφαρμόζεται η μέθοδος **Stratified K-Fold**, η οποία

δημιουργεί πολλαπλές πτυχές διατηρώντας την αναλογία των κλάσεων στα δεδομένα εκπαίδευσης και δοκιμής. Τα σύνολα εκπαίδευσης και δοκιμής αποθηκεύονται ως αρχεία CSV, ενώ ο χρόνος που απαιτήθηκε για τη δημιουργία των πτυχών καταγράφεται σε αρχείο `fold_creation_time.txt`. Η διαδικασία αυτή επιτρέπει την καλύτερη αξιολόγηση των αλγορίθμων μέσω της διασταυρούμενης επικύρωσης.

```
def read_and_save_folds(relative_file_path: str):
      fold_counter = 0
      root_path = "/".join(relative_file_path.split("/")[:-1]) + "/"
      dataset_name = relative_file_path.split("/")[-1].split(".")[0]

      initial_data = read_data(relative_file_path)
      scaled_data, scaling_completion_time = scale_table(initial_data)

      scaling_time_file_path = relative_file_path + "scaling_time.txt"
      fold_creation_time_file_path = relative_file_path +
"fold_creation_time.txt"
      with open(scaling_time_file_path, 'w') as file_to_write:
      file_to_write.write(f'Time to scale {dataset_name} table took
{scaling_completion_time} seconds')

      x_data = scaled_data.loc[:, scaled_data.columns != "Class"]

      fold_creation_start_time = time.time()
      for train_indices, test_indices in skf.split(x_data,
scaled_data["Class"]):
      x_train = x_data.iloc[train_indices]
      y_train = scaled_data["Class"].iloc[train_indices].apply(lambda x:
x.strip())
      x_y_data_train_data = x_train.join(y_train)
      new_train_file_name = f"{root_path +
dataset_name}_train_{fold_counter}.csv"
      # if not os.path.isfile(new_train_file_name):
      x_y_data_train_data.to_csv(new_train_file_name, index=False)

      x_test = x_data.iloc[test_indices]
      y_test = scaled_data["Class"].iloc[test_indices].apply(lambda x:
x.strip())
      new_test_file_name = f"{root_path +
dataset_name}_test_{fold_counter}.csv"
      x_y_data_test_data = x_test.join(y_test)
      # if not os.path.isfile(new_test_file_name):
      x_y_data_test_data.to_csv(new_test_file_name, index=False)
      fold_counter += 1
      fold_creation_end_time = time.time()
      with open(fold_creation_time_file_path, 'w') as file_to_write:
```

```
        file_to_write.write(f'Time to create folds for {dataset_name} table
took {fold_creation_end_time} seconds')
```

Η συνάρτηση `read_and_save_folds` είναι υπεύθυνη για την ανάγνωση των δεδομένων από ένα αρχείο, την κανονικοποίησή τους και τη δημιουργία πολλαπλών πτυχών (folds) για χρήση σε τεχνικές διασταυρούμενης επικύρωσης. Αρχικά, εξάγεται το όνομα του συνόλου δεδομένων από το όνομα του αρχείου, το οποίο διαβάζεται μέσω της συνάρτησης `read_data`. Μετά την ανάγνωση, τα δεδομένα κλιμακώνονται χρησιμοποιώντας τη συνάρτηση `scale_table`, ενώ ο χρόνος κανονικοποίησης αποθηκεύεται σε ένα αρχείο με κατάληξη `scaling_time.txt`. Στη συνέχεια, διαχωρίζονται τα χαρακτηριστικά από την ετικέτα κατηγορίας και εφαρμόζεται η μέθοδος **Stratified K-Fold**, η οποία δημιουργεί πολλαπλές πτυχές διατηρώντας την αναλογία των κλάσεων στα δεδομένα εκπαίδευσης και δοκιμής. Τα σύνολα εκπαίδευσης και δοκιμής αποθηκεύονται ως αρχεία CSV, ενώ ο χρόνος που απαιτήθηκε για τη δημιουργία των πτυχών καταγράφεται σε αρχείο `fold_creation_time.txt`. Η διαδικασία αυτή επιτρέπει την καλύτερη αξιολόγηση των αλγορίθμων μέσω της διασταυρούμενης επικύρωσης.

```
for d in datasets:
        read_and_save_folds(d)
```

Η επανάληψη `for d in datasets` καλεί τη συνάρτηση `read_and_save_folds` για κάθε αρχείο δεδομένων στη λίστα `datasets`. Κάθε στοιχείο της λίστας περιέχει τη διαδρομή ενός συνόλου δεδομένων, το οποίο επεξεργάζεται, κανονικοποιείται και διαχωρίζεται σε πτυχές με σκοπό τη χρήση τους στα πειράματα. Αυτό το βήμα διασφαλίζει ότι όλα τα σύνολα δεδομένων προετοιμάζονται με τον ίδιο τρόπο, επιτρέποντας δίκαιες συγκρίσεις μεταξύ των διαφορετικών τεχνικών και αλγορίθμων που θα εφαρμοστούν στη συνέχεια.

```
classification_algorithms = [
        ("Logistic regression", LogisticRegression()),
        ("Decision trees", DecisionTreeClassifier()),
        ("Random forest", RandomForestClassifier()),
        ("SVMs", SVC()),
        ("KNN", KNeighborsClassifier()),
        ("Naive Bayes", GaussianNB()),
]
```

Η λίστα `classification_algorithms` περιέχει ζεύγη που αντιστοιχούν στα ονόματα και τις αντίστοιχες υλοποιήσεις των πιο δημοφιλών αλγορίθμων κατηγοριοποίησης στη βιβλιοθήκη **Scikit-learn (sklearn)**. Κάθε στοιχείο της λίστας αποτελείται από το όνομα του αλγορίθμου και το αντίστοιχο αντικείμενο κλάσης του ταξινομητή, επιτρέποντας την εύκολη εναλλαγή και δοκιμή διαφορετικών αλγορίθμων κατά τη διαδικασία πειραματισμού.

```
def train_and_eval(directory: str):
```

```
      for classification_algorithm_name, classification_impl in
classification_algorithms:
      for oversampling_algorithm_name, oversampling_impl in
oversampling_algorithms:
            start = time.time()
            y_tests = []
            y_preds = []
            file_tuples = get_file_tuples(directory)
            for file_tuple in file_tuples:
                  train_csv = pd.read_csv(file_tuple[0])
                  test_csv = pd.read_csv(file_tuple[1])
                  x_train = train_csv.loc[:, train_csv.columns !=
'Class']
                  y_train = train_csv.loc[:, train_csv.columns ==
'Class']

                  x_test = test_csv.loc[:, test_csv.columns != 'Class']
                  y_test = test_csv.loc[:, test_csv.columns == 'Class']

                  x_res, y_res =
oversample(classification_algorithm_name, directory,
oversampling_algorithm_name, oversampling_impl, x_train, y_train)

                  classification_impl.fit(x_res, y_res.squeeze())
                  training_end_time = time.time() - start
                  # Predict the labels for the test set
                  y_pred = classification_impl.predict(x_test)
                  # Evaluate the classifier
                  for v in y_test.values.flatten().tolist():
                  y_tests.append(v)
                  for v in y_pred:
                  y_preds.append(v)
            accuracy = accuracy_score(y_tests, y_preds)
            report = classification_report(y_tests, y_preds)
            tn, fp, fn, tp = confusion_matrix(y_tests, y_preds).ravel()

            oversampling_algorithm_print_text = f"Oversampling algorithm:
{oversampling_algorithm_name}\n"
            classification_algorithm_print_text = f"Classification
algorithm: {classification_algorithm_name}\n"
            confusion_matrix_print_text = f"TP: {tp}\nTN: {tn}\nFP:
{fp}\nFN: {fn}\n"
            accuracy_print_text = f"Accuracy: \n{accuracy}\n"
            classification_report_print_text = f"Classification  report:
\n{report}\n"
```

```
            end = time.time()

            reports_dir = f"{directory}/reports"
            if not os.path.exists(reports_dir):
                  os.makedirs(reports_dir)
            with
open(f"{reports_dir}/{oversampling_algorithm_name}_{classification_algori
thm_name}_report.txt",
                        "w") as f:
                  f.write(oversampling_algorithm_print_text)
                  f.write("BEFORE: \n" +
train_csv['Class'].value_counts().to_string() + "\n")
                  f.write("AFTER: \n" + y_res.value_counts().to_string()
+ "\n\n")
                  f.write(classification_algorithm_print_text)
                  f.write(confusion_matrix_print_text)
                  f.write(accuracy_print_text)
                  f.write(classification_report_print_text)
                  f.write(f"Seconds to run training:
{training_end_time}s\n")
                  f.write(f"Seconds to run evaluation: {end - start -
training_end_time}s")
            print(f"Finished with: {directory} -
{classification_algorithm_name} - {oversampling_algorithm_name} @ Seconds
to run: {end - start}")
```

Η συνάρτηση `train_and_eval` είναι υπεύθυνη για την εκπαίδευση και την αξιολόγηση διαφορετικών αλγορίθμων ταξινόμησης σε σύνολα δεδομένων που αποθηκεύονται στον καθορισμένο κατάλογο (directory). Χρησιμοποιεί διάφορες τεχνικές υπερδειγματοληψίας για να βελτιώσει την ισορροπία των κατηγοριών στα δεδομένα και αξιολογεί την απόδοση των ταξινομητών σε όρους ακρίβειας και άλλων μετρικών. Ο κώδικας ακολουθεί μια επαναληπτική διαδικασία που καλύπτει όλα τα διαθέσιμα σύνολα δεδομένων, τις τεχνικές υπερδειγματοληψίας και τους αλγορίθμους ταξινόμησης, καταγράφοντας τα αποτελέσματα σε αρχεία αναφοράς.

Η διαδικασία ξεκινά με δύο εμφωλευμένους βρόχους `for`, όπου για κάθε αλγόριθμο ταξινόμησης που περιέχεται στη λίστα `classification_algorithms`, εφαρμόζονται διαδοχικά όλες οι τεχνικές υπερδειγματοληψίας από τη λίστα `oversampling_algorithms`. Για κάθε συνδυασμό, καταγράφεται ο χρόνος έναρξης της διαδικασίας εκπαίδευσης και αξιολόγησης. Οι μεταβλητές `y_tests` και `y_preds` δημιουργούνται για να αποθηκεύσουν τις πραγματικές και τις προβλεπόμενες ετικέτες αντίστοιχα, οι οποίες θα χρησιμοποιηθούν αργότερα για την αξιολόγηση της απόδοσης του ταξινομητή.

Στη συνέχεια, η συνάρτηση `get_file_tuples` λαμβάνει τα αντίστοιχα αρχεία εκπαίδευσης και δοκιμής από τον συγκεκριμένο κατάλογο. Για κάθε ζεύγος αρχείων, τα δεδομένα φορτώνονται ως `DataFrame` μέσω της βιβλιοθήκης `pandas`. Τα χαρακτηριστικά των δεδομένων εκπαίδευσης και

δοκιμής απομονώνονται από τη στήλη "Class", η οποία περιέχει τις ετικέτες των κατηγοριών, και αποθηκεύονται στις μεταβλητές `x_train`, `y_train`, `x_test` και `y_test`.

Η επόμενη σημαντική διαδικασία αφορά την υπερδειγματοληψία, όπου καλείται η συνάρτηση `oversample` για να εφαρμοστεί η επιλεγμένη τεχνική υπερδειγματοληψίας στα δεδομένα εκπαίδευσης. Εάν η τεχνική δεν είναι η "Baseline", τα δεδομένα εξισορροπούνται και επιστρέφονται τα νέα σύνολα εκπαίδευσης `x_res` και `y_res`. Διαφορετικά, επιστρέφονται τα αρχικά δεδομένα εκπαίδευσης χωρίς αλλαγές.

Αφού ολοκληρωθεί η διαδικασία της υπερδειγματοληψίας, το ταξινομητικό μοντέλο εκπαιδεύεται με βάση τα δεδομένα εκπαίδευσης χρησιμοποιώντας τη μέθοδο `fit`. Ο χρόνος εκπαίδευσης καταγράφεται και αποθηκεύεται στη μεταβλητή `training_end_time`. Στη συνέχεια, το μοντέλο χρησιμοποιείται για την πρόβλεψη των ετικετών της κλάσης στα δεδομένα δοκιμής και οι προβλέψεις αποθηκεύονται στη λίστα `y_preds`, ενώ οι πραγματικές τιμές προστίθενται στη λίστα `y_tests`.

Αφού ολοκληρωθεί η διαδικασία πρόβλεψης, υπολογίζεται η ακρίβεια του μοντέλου μέσω της συνάρτησης `accuracy_score`, ενώ δημιουργείται μια αναλυτική αναφορά απόδοσης χρησιμοποιώντας τη `classification_report`. Επιπλέον, υπολογίζεται ο πίνακας σύγχυσης (`confusion_matrix`), ο οποίος παρέχει πληροφορίες σχετικά με τις σωστές και λανθασμένες ταξινομήσεις, δίνοντας τις τιμές των αληθώς θετικών (TP), αληθώς αρνητικών (TN), ψευδώς θετικών (FP) και ψευδώς αρνητικών (FN) προβλέψεων.

Στη συνέχεια, δημιουργείται το κείμενο των αποτελεσμάτων που περιλαμβάνει το όνομα της τεχνικής υπερδειγματοληψίας, τον αλγόριθμο ταξινόμησης, τις μετρήσεις από τον πίνακα σύγχυσης, την ακρίβεια του ταξινομητή και την πλήρη αναφορά κατηγοριοποίησης. Τα αποτελέσματα αποθηκεύονται σε ένα αρχείο αναφοράς που δημιουργείται στον κατάλογο `/reports` εντός του directory. Εάν ο κατάλογος δεν υπάρχει, δημιουργείται αυτόματα. Το αρχείο περιλαμβάνει επίσης πληροφορίες σχετικά με τη διανομή των κατηγοριών πριν και μετά την εφαρμογή της τεχνικής υπερδειγματοληψίας, καθώς και τους χρόνους που απαιτήθηκαν για την εκπαίδευση και την αξιολόγηση του μοντέλου.

Τέλος, εμφανίζεται μήνυμα στην κονσόλα που ενημερώνει για την ολοκλήρωση της διαδικασίας εκπαίδευσης και αξιολόγησης για τον συγκεκριμένο συνδυασμό δεδομένων, τεχνικής υπερδειγματοληψίας και ταξινομητή, μαζί με τον συνολικό χρόνο εκτέλεσης.

```
def get_file_tuples(directory):
      dataset_name = directory.split("/")[-2]
      data_tuples = []
      for i in range(4):
      pattern_train = f"{dataset_name}_train_{i}.csv"
      pattern_test = f"{dataset_name}_test_{i}.csv"
      train_file_path = directory + pattern_train
      test_file_path = directory + pattern_test
      data_tuples.append((train_file_path, test_file_path))
      return data_tuples
```

Η συνάρτηση `get_file_tuples` χρησιμοποιείται για τη δημιουργία μιας λίστας με τα αρχεία εκπαίδευσης και δοκιμής που σχετίζονται με ένα συγκεκριμένο σύνολο δεδομένων. Η συνάρτηση λαμβάνει ως είσοδο τη διαδρομή του φακέλου όπου βρίσκονται αποθηκευμένα τα αρχεία και επιστρέφει μια λίστα από πλειάδες (tuples), κάθε μία από τις οποίες περιέχει το όνομα του αρχείου εκπαίδευσης και του αντίστοιχου αρχείου δοκιμής.

Η συνάρτηση ξεκινά εξάγοντας το όνομα του συνόλου δεδομένων από τη διαδρομή που παρέχεται ως όρισμα. Αυτό γίνεται μέσω της συνάρτησης `split("/")[-2]`, η οποία διαχωρίζει τη διαδρομή με βάση τον χαρακτήρα `/` και λαμβάνει τον προτελευταίο φάκελο, ο οποίος αντιπροσωπεύει το όνομα του dataset. Στη συνέχεια, δημιουργείται μια κενή λίστα με το όνομα `data_tuples`, όπου θα αποθηκευτούν οι πλειάδες των αρχείων εκπαίδευσης και δοκιμής.

Ο κύριος κορμός της συνάρτησης αποτελείται από έναν βρόχο επανάληψης `for i in range(4)`, ο οποίος εκτελείται τέσσερις φορές, προκειμένου να δημιουργηθούν οι διαδρομές για τέσσερα διαφορετικά αρχεία εκπαίδευσης και δοκιμής. Κατά τη διάρκεια της κάθε επανάληψης, δημιουργούνται οι διαδρομές των αρχείων εκπαίδευσης και δοκιμής μέσω της μορφοποίησης συμβολοσειράς (`f-string`). Χρησιμοποιούνται τα μοτίβα `"{dataset_name}_train_{i}.csv"` και `"{dataset_name}_test_{i}.csv"`, όπου το όνομα του dataset και ο αριθμός της επανάληψης ενσωματώνονται δυναμικά. Αυτές οι διαδρομές συνδυάζονται με τον αρχικό φάκελο που παρέχεται στη μεταβλητή `directory` για να προκύψουν οι πλήρεις διαδρομές των αρχείων.

Τα αρχεία προστίθενται ως πλειάδα στη λίστα `data_tuples`, όπου κάθε στοιχείο περιλαμβάνει τη διαδρομή του αντίστοιχου αρχείου εκπαίδευσης και δοκιμής. Τελικά, η συνάρτηση επιστρέφει τη λίστα με τις πλειάδες αρχείων, παρέχοντας έναν οργανωμένο τρόπο ανάκτησης των δεδομένων που θα χρησιμοποιηθούν στη διαδικασία εκπαίδευσης και αξιολόγησης.

Η συγκεκριμένη συνάρτηση επιτρέπει την εύκολη πρόσβαση σε σύνολα δεδομένων που έχουν ήδη χωριστεί σε πολλαπλές πτυχές (folds) για χρήση σε διαδικασίες διασταυρούμενης επικύρωσης (cross-validation). Ο αριθμός των επαναλήψεων στη συγκεκριμένη υλοποίηση είναι τέσσερις, κάτι που υποδηλώνει ότι κάθε dataset έχει διαχωριστεί σε τέσσερις διαφορετικές πτυχές εκπαίδευσης και δοκιμής.

```
dataset_dirs = [
      "data/ecoli1/",
      "data/glass0/",
      "data/page-blocks0/",
      "data/new-thyroid1/",
      "data/pima/",
      "data/haberman/",
      "data/segment0/",
      "data/vehicle0/",
      "data/wisconsin/",
      "data/yeast1/"
]
```

Η λίστα `dataset_dirs` περιέχει τις διαδρομές των φακέλων που περιέχουν τα σύνολα δεδομένων που θα χρησιμοποιηθούν στα πειράματα. Κάθε στοιχείο της λίστας είναι μια συμβολοσειρά που αντιπροσωπεύει το μονοπάτι προς έναν συγκεκριμένο φάκελο δεδομένων. Οι φάκελοι αυτοί περιέχουν αρχεία που περιλαμβάνουν τα δεδομένα που πρόκειται να φορτωθούν, να επεξεργαστούν και να χρησιμοποιηθούν για την εκπαίδευση και αξιολόγηση αλγορίθμων ταξινόμησης.

Η λίστα αυτή επιτρέπει τη διαχείριση πολλαπλών datasets σε μια ενιαία δομή, διευκολύνοντας την αυτοματοποίηση της διαδικασίας πειραμάτων. Μέσα σε κάθε φάκελο υπάρχουν αρχεία που προέρχονται από συγκεκριμένα προβλήματα ταξινόμησης, όπως το `ecoli1`, το `glass0` και άλλα, που χρησιμοποιούνται ευρέως σε έρευνες μηχανικής μάθησης και κατηγοριοποίησης δεδομένων. Αυτή η προσέγγιση επιτρέπει την εύκολη επανάληψη της διαδικασίας πειραματισμού σε διαφορετικά datasets χωρίς την ανάγκη χειροκίνητης εισαγωγής των διαδρομών αρχείων κάθε φορά.

```
for d in dataset_dirs:
      train_and_eval(d)
```

Η εντολή `for d in dataset_dirs: train_and_eval(d)` εκτελεί τη συνάρτηση `train_and_eval` για κάθε φάκελο δεδομένων στη λίστα `dataset_dirs`. Με αυτόν τον τρόπο, αυτοματοποιείται η διαδικασία εκπαίδευσης και αξιολόγησης των αλγορίθμων κατηγοριοποίησης σε όλα τα διαθέσιμα σύνολα δεδομένων. Κάθε dataset φορτώνεται, υποβάλλεται σε υπερδειγματοληψία και χρησιμοποιείται για την εκπαίδευση των ταξινομητών, ενώ τα αποτελέσματα καταγράφονται για περαιτέρω ανάλυση. Αυτή η προσέγγιση επιτρέπει τη σύγκριση της απόδοσης των διαφορετικών τεχνικών σε πολλαπλά datasets με ελάχιστη χειροκίνητη παρέμβαση.

Κεφάλαιο 5ο: Πειραματική μελέτη

5.1 Σύνολα δεδομένων

Στην παρούσα πειραματική μελέτη χρησιμοποιούνται δέκα διαφορετικά σύνολα δεδομένων, τα οποία προέρχονται από διάφορους τομείς και έχουν διαφορετικά χαρακτηριστικά όσον αφορά τον αριθμό των παραδειγμάτων, των χαρακτηριστικών και την ανισοκατανομή των κλάσεων. Τα datasets αυτά είναι ευρέως χρησιμοποιούμενα στη βιβλιογραφία για την αξιολόγηση τεχνικών μηχανικής μάθησης, ιδιαίτερα σε περιπτώσεις όπου υπάρχει ανισοκατανομή στις κλάσεις.

Το **ecoli1** dataset αφορά δεδομένα από το πεδίο της βιολογίας και περιέχει πληροφορίες σχετικά με την τοποθέτηση πρωτεϊνών στις κυτταρικές μεμβράνες του βακτηρίου E. coli. Χρησιμοποιείται συχνά για προβλήματα κατηγοριοποίησης βιολογικών δεδομένων και έχει έντονη ανισοκατανομή κλάσεων.

Το **glass0** dataset προέρχεται από εφαρμογές εγκληματολογικής ανάλυσης και αφορά την ταξινόμηση διαφορετικών τύπων γυαλιού με βάση τις χημικές τους ιδιότητες. Περιλαμβάνει χαρακτηριστικά όπως η περιεκτικότητα σε οξείδια και χρησιμοποιείται για προβλήματα ταξινόμησης σε εγκληματολογικές μελέτες.

Το **haberman** dataset αφορά ιατρικά δεδομένα από χειρουργικές επεμβάσεις μαστού και περιλαμβάνει πληροφορίες σχετικά με την επιβίωση των ασθενών μετά από τη θεραπεία. Το σύνολο δεδομένων έχει έντονη ανισοκατανομή κλάσεων, καθώς η επιβίωση σε βάθος χρόνου είναι πιο σπάνια.

Το **new-thyroid1** dataset προέρχεται από ιατρικές εφαρμογές και χρησιμοποιείται για τη διάγνωση διαταραχών του θυρεοειδούς αδένα. Περιέχει χαρακτηριστικά από αιματολογικές εξετάσεις και παρουσιάζει προκλήσεις λόγω της ανισοκατανομής των κλάσεων.

Το **page-blocks0** dataset αφορά δεδομένα από συστήματα επεξεργασίας κειμένου και περιέχει πληροφορίες σχετικά με την ταξινόμηση τμημάτων εγγράφων σε διαφορετικές κατηγορίες, όπως κείμενο και γραφικά. Χρησιμοποιείται ευρέως σε εφαρμογές αναγνώρισης προτύπων και κατηγοριοποίησης κειμένων.

Το **pima** dataset προέρχεται από τον ιατρικό τομέα και περιλαμβάνει δεδομένα σχετικά με την εμφάνιση διαβήτη σε γυναίκες ινδιάνικης καταγωγής της φυλής Pima. Το σύνολο δεδομένων είναι γνωστό για την ανισοκατανομή των κλάσεων και την πρόκληση που θέτει στους αλγορίθμους ταξινόμησης.

Το **segment0** dataset αφορά δεδομένα από εφαρμογές επεξεργασίας εικόνας και περιλαμβάνει χαρακτηριστικά που χρησιμοποιούνται για την ταξινόμηση τμημάτων μιας εικόνας σε διαφορετικές κατηγορίες. Το σύνολο δεδομένων έχει μεγάλη ποικιλομορφία και είναι χρήσιμο για την αξιολόγηση αλγορίθμων κατηγοριοποίησης εικόνων.

Το **vehicle0** dataset σχετίζεται με εφαρμογές αναγνώρισης τύπων οχημάτων μέσω χαρακτηριστικών που εξάγονται από εικόνες. Χρησιμοποιείται σε προβλήματα μηχανικής όρασης και κατηγοριοποίησης αντικειμένων, όπου υπάρχει ανισοκατανομή μεταξύ των κλάσεων.

Το **wisconsin** dataset προέρχεται από τον ιατρικό τομέα και χρησιμοποιείται για τη διάγνωση του καρκίνου του μαστού. Περιέχει χαρακτηριστικά που προέρχονται από βιοψίες και είναι ευρέως διαδεδομένο στην αξιολόγηση αλγορίθμων μηχανικής μάθησης σε ιατρικά δεδομένα.

Το **yeast1** dataset περιλαμβάνει δεδομένα που αφορούν την ταξινόμηση πρωτεϊνών ζυμομυκήτων σύμφωνα με τις κυτταρικές τους λειτουργίες. Χρησιμοποιείται σε προβλήματα βιοπληροφορικής και χαρακτηρίζεται από ανισοκατανομή των κλάσεων, γεγονός που το καθιστά κατάλληλο για την αξιολόγηση τεχνικών υπερδειγματοληψίας.

Αυτά τα σύνολα δεδομένων επιλέχθηκαν για την πειραματική μελέτη, καθώς καλύπτουν ένα ευρύ φάσμα εφαρμογών και προκλήσεων, επιτρέποντας την εξαγωγή χρήσιμων συμπερασμάτων για την αποτελεσματικότητα των τεχνικών υπερδειγματοληψίας και των αλγορίθμων ταξινόμησης.

5.2 Εγκαθίδρυση πειραμάτων (Experimental setup)

Σε αυτή την ενότητα περιγράφουμε τα βήματα που ακολουθήσαμε για να πραγματοποιήσουμε τα πειράματα, από την προετοιμασία των δεδομένων μέχρι την εκπαίδευση και αξιολόγηση των αλγορίθμων ταξινόμησης. Στόχος είναι να συγκρίνουμε τις διαφορετικές μεθόδους υπερδειγματοληψίας και να αξιολογήσουμε την απόδοσή τους, χρησιμοποιώντας την ακρίβεια ως βασικό μέτρο. Παράλληλα, καταγράφουμε τον χρόνο εκτέλεσης κάθε αλγορίθμου ώστε να έχουμε μια εικόνα για το πόσο γρήγορα λειτουργεί η κάθε μέθοδος.

Αρχικά, κάθε σύνολο δεδομένων φορτώθηκε από τους αντίστοιχους φακέλους, όπου τα δεδομένα είχαν ήδη χωριστεί σε τέσσερα μέρη με τη μέθοδο της **διασταυρούμενης επικύρωσης (cross-validation)**. Αυτό σημαίνει ότι κάθε σύνολο δεδομένων χωρίστηκε σε τέσσερις πτυχές, όπου κάθε φορά μία από αυτές χρησιμοποιείται για δοκιμή, ενώ οι υπόλοιπες τρεις για εκπαίδευση. Με αυτόν τον τρόπο, εξασφαλίζεται ότι όλα τα δεδομένα χρησιμοποιούνται τόσο για εκπαίδευση όσο και για δοκιμή, μειώνοντας το ενδεχόμενο μεροληψίας.

Πριν την εκπαίδευση των αλγορίθμων, εφαρμόστηκε κανονικοποίηση των δεδομένων χρησιμοποιώντας τη μέθοδο **MinMax Scaling**, η οποία μετατρέπει τις τιμές των χαρακτηριστικών σε μια κοινή κλίμακα, συνήθως μεταξύ 0 και 1. Αυτή η διαδικασία βοηθά ώστε όλα τα χαρακτηριστικά να έχουν ίσο βάρος στη διαδικασία εκμάθησης. Ακολούθως, εφαρμόστηκαν οι **ADASYN, SMOTE, Random Oversampler, SMOTEN, KMeansSMOTE και SVM SMOTE** όπως είδαμε και στο πρηγούμενο κεφάλαιο, για να αντιμετωπιστεί η ανισοκατανομή των κλάσεων στα δεδομένα. Για λόγους σύγκρισης, χρησιμοποιήθηκαν επίσης τα αρχικά δεδομένα χωρίς καμία παρέμβαση, ως βασική γραμμή αναφοράς (baseline).

Στη συνέχεια, χρησιμοποιήθηκαν έξι διαφορετικοί αλγόριθμοι ταξινόμησης από τη βιβλιοθήκη **Scikit-learn**, συγκεκριμένα οι **Logistic Regression, Decision Trees, Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (KNN) και Gaussian Naive Bayes.** Κάθε αλγόριθμος εκπαιδεύτηκε σε κάθε σύνολο δεδομένων και κάθε μέθοδο υπερδειγματοληψίας, ώστε να έχουμε πλήρη εικόνα της απόδοσής τους.

Η αξιολόγηση των μοντέλων έγινε με βάση την **ακρίβεια (accuracy)**, δηλαδή το ποσοστό των σωστών προβλέψεων σε σχέση με το σύνολο των δεδομένων. Εκτός από την ακρίβεια, καταγράψαμε

και τον **χρόνο εκτέλεσης (CPU time)** που χρειάστηκε τόσο για την εκπαίδευση όσο και για την πρόβλεψη των δεδομένων. Η μέτρηση του χρόνου είναι σημαντική καθώς μας δίνει μια ιδέα για το πόσο αποδοτικός είναι κάθε αλγόριθμος και αν είναι κατάλληλος για εφαρμογές που απαιτούν γρήγορη ανταπόκριση.

Όλα τα αποτελέσματα αποθηκεύτηκαν σε αναφορές που περιλαμβάνουν πληροφορίες όπως η ακρίβεια των μοντέλων, η κατανομή των δεδομένων πριν και μετά την υπερδειγματοληψία και ο συνολικός χρόνος εκτέλεσης. Οι αναφορές αυτές αποθηκεύτηκαν σε φακέλους για κάθε πείραμα, ώστε να μπορούν να χρησιμοποιηθούν για περαιτέρω ανάλυση και σύγκριση.

Η διαδικασία αυτή υλοποιήθηκε πλήρως μέσω της γλώσσας προγραμματισμού Python, επιτρέποντας την αυτοματοποίηση των πειραμάτων. Έτσι, διασφαλίστηκε ότι όλα τα δεδομένα επεξεργάστηκαν με τον ίδιο τρόπο, διατηρώντας τη συνέπεια των αποτελεσμάτων και διευκολύνοντας την επανάληψη της διαδικασίας αν χρειαστεί.

5.3 Πειραματικά αποτελέσματα (σύγκριση τεχνικών υπερδειγματοληψίας)

Αποτελέσματα για Dataset ecoli1

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 57 (minority class)

Negative: 195 (majority class)

ecoli1	**Sample counts**		**Decision Trees**					
			Accuracy	**CPU Train Time (second)**	**Precision**		**Recall**	
					Positive Class	**Negative Class**	**Positive Class**	**Negative Class**
Baseline	**P=57**	**N=195**	0.880	0.011	0.78	0.93	0.78	0.93
Adasyn	**P=183**	**N=195**	0.872	0.0313	0.75	0.94	0.81	0.92
Kmeans Smote	**P=195**	**N=195**	0.892	0.037	0.78	0.94	0.79	0.93
Random Oversampler	**P=195**	**N=195**	0.880	0.022	0.77	0.93	0.75	0.93
Smote Oversampler	**P=195**	**N=195**	0.886	0.023	0.74	0.95	0.83	0.91
SVM Smote	**P=195**	**N=195**	0.877	0.030	0.68	0.92	0.74	0.90

ecoli1	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=57**	**N=195**	0.645	0.008	0.39	0.99	0.97	0.55
Adasyn	**P=183**	**N=195**	0.607	0.045	0.37	0.98	0.97	0.50
Kmeans Smote	**P=195**	**N=195**	0.755	0.031	0.48	0.97	0.94	0.70
Random Oversampler	**P=195**	**N=195**	0.607	0.019	0.37	0.99	0.99	0.49
Smote Oversampler	**P=195**	**N=195**	0.702	0.019	0.43	0.98	0.96	0.63
SVM Smote	**P=195**	**N=195**	0.607	0.033	0.37	0.99	0.99	0.49

ecoli1	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=57**	**N=195**	0.916	0.013	0.86	0.93	0.77	0.96
Adasyn	**P=183**	**N=195**	0.866	0.034	0.65	0.97	0.90	0.86
Kmeans Smote	**P=195**	**N=195**	0.910	0.039	0.81	0.94	0.79	0.95
Random Oversampler	**P=195**	**N=195**	0.875	0.037	0.67	0.96	0.88	0.87
Smote Oversampler	**P=195**	**N=195**	0.892	0.026	0.73	0.96	0.86	0.90
SVM Smote	**P=195**	**N=195**	0.886	0.035	0.70	0.96	0.88	0.89

ecoli1	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=57**	**N=195**	0.875	0.026	0.81	0.89	0.60	0.96
Adasyn	**P=183**	**N=195**	0.860	0.025	0.63	0.98	0.94	0.84
Kmeans Smote	**P=195**	**N=195**	0.869	0.043	0.66	0.96	0.88	0.86
Random Oversampler	**P=195**	**N=195**	0.866	0.027	0.65	0.97	0.91	0.85
Smote Oversampler	**P=195**	**N=195**	0.869	0.022	0.65	0.97	0.91	0.86
SVM Smote	**P=195**	**N=195**	0.866	0.032	0.64	0.98	0.94	0.85

ecoli1	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=57**	**N=195**	0.898	0.304	0.79	0.93	0.77	0.94
Adasyn	**P=183**	**N=195**	0.907	0.289	0.75	0.97	0.90	0.91
Kmeans Smote	**P=195**	**N=195**	0.898	0.335	0.79	0.93	0.75	0.94
Random Oversampler	**P=195**	**N=195**	0.898	0.321	0.77	0.94	0.81	0.93
Smote Oversampler	**P=195**	**N=195**	0.901	0.356	0.75	0.96	0.86	0.92
SVM Smote	**P=195**	**N=195**	0.904	0.309	0.74	0.97	0.90	0.91

ecoli1	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=57**	**N=195**	0.910	0.016	0.85	0.93	0.74	0.96
Adasyn	**P=183**	**N=195**	0.869	0.032	0.64	0.99	0.96	0.84
Kmeans Smote	**P=195**	**N=195**	0.898	0.048	0.77	0.94	0.81	0.93
Random Oversampler	**P=195**	**N=195**	0.869	0.020	0.65	0.97	0.91	0.86
Smote Oversampler	**P=195**	**N=195**	0.872	0.024	0.66	0.97	0.92	0.86
SVM Smote	**P=195**	**N=195**	0.866	0.055	0.65	0.97	0.91	0.85

Αποτελέσματα για Dataset glass0

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 53 (minority class)

Negative: 108 (majority class)

glass0	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=53	N=108	0.813	0.010	0.72	0.86	0.70	0.87
Adasyn	P=104	N=108	0.761	0.033	0.61	0.85	0.73	0.78
Kmeans Smote	P=108	N=108	0.794	0.039	0.69	0.84	0.67	0.85
Random Oversampler	P=108	N=108	0.827	0.019	0.74	0.87	0.73	0.88
Smote Oversampler	P=108	N=108	0.794	0.024	0.67	0.87	0.74	0.82
SVM Smote	P=108	N=108	0.780	0.028	0.65	0.86	0.73	0.81

glass0	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=53	N=108	0.640	0.015	0.47	0.92	0.91	0.51
Adasyn	P=104	N=108	0.626	0.034	0.46	0.90	0.89	0.50
Kmeans Smote	P=108	N=108	0.644	0.045	0.48	0.90	0.87	0.53
Random Oversampler	P=108	N=108	0.640	0.015	0.48	0.96	0.96	0.49
Smote Oversampler	P=108	N=108	0.626	0.027	0.46	0.90	0.89	0.50
SVM Smote	P=108	N=108	0.630	0.029	0.47	0.91	0.90	0.50

glass0	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=53	N=108	0.780	0.014	0.64	0.86	0.74	0.80
Adasyn	P=104	N=108	0.761	0.032	0.59	0.93	0.90	0.69
Kmeans Smote	P=108	N=108	0.775	0.034	0.63	0.88	0.77	0.78
Random Oversampler	P=108	N=108	0.761	0.025	0.60	0.90	0.83	0.73
Smote Oversampler	P=108	N=108	0.771	0.025	0.61	0.91	0.86	0.73
SVM Smote	P=108	N=108	0.780	0.031	0.61	0.95	0.93	0.71

glass0	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=53	N=108	0.705	0.023	0.68	0.71	0.19	0.96
Adasyn	P=104	N=108	0.644	0.024	0.48	0.96	0.96	0.49
Kmeans Smote	P=108	N=108	0.738	0.041	0.58	0.84	0.71	0.75
Random Oversampler	P=108	N=108	0.654	0.020	0.49	0.95	0.94	0.51
Smote Oversampler	P=108	N=108	0.649	0.024	0.48	0.93	0.91	0.52
SVM Smote	P=108	N=108	0.654	0.036	0.49	0.96	0.96	0.51

glass0	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=53	N=108	0.859	0.269	0.79	0.90	0.79	0.90
Adasyn	P=104	N=108	0.841	0.267	0.71	0.93	0.87	0.83
Kmeans Smote	P=108	N=108	0.864	0.280	0.79	0.90	0.80	0.90
Random Oversampler	P=108	N=108	0.864	0.393	0.77	0.92	0.84	0.88
Smote Oversampler	P=108	N=108	0.855	0.445	0.75	0.92	0.84	0.86
SVM Smote	P=108	N=108	0.859	0.320	0.75	0.93	0.86	0.86

glass0	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=53	N=108	0.705	0.012	0.61	0.72	0.29	0.91
Adasyn	P=104	N=108	0.644	0.023	0.48	0.99	0.99	0.48
Kmeans Smote	P=108	N=108	0.724	0.047	0.60	0.77	0.47	0.85
Random Oversampler	P=108	N=108	0.654	0.018	0.49	0.95	0.94	0.51
Smote Oversampler	P=108	N=108	0.649	0.022	0.48	0.96	0.96	0.50
SVM Smote	P=108	N=108	0.658	0.038	0.49	0.99	0.99	0.50

Αποτελέσματα για Dataset haberman

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 61 (minority class)

Negative: 169 (majority class)

haberman	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=61	N=169	0.692	0.010	0.42	0.80	0.44	0.78
Adasyn	P=184	N=169	0.673	0.023	0.38	0.77	0.36	0.79
Kmeans Smote	P=169	N=169	0.676	0.038	0.38	0.77	0.33	0.80
Random Oversampler	P=169	N=169	0.653	0.018	0.33	0.76	0.31	0.78
Smote Oversampler	P=169	N=169	0.647	0.021	0.36	0.78	0.44	0.72
SVM Smote	P=169	N=169	0.627	0.029	0.33	0.77	0.40	0.71

haberman	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=61	N=169	0.741	0.005	0.53	0.76	0.20	0.94
Adasyn	P=184	N=169	0.722	0.019	0.38	0.77	0.36	0.79
Kmeans Smote	P=169	N=169	0.751	0.031	0.56	0.78	0.30	0.92
Random Oversampler	P=169	N=169	0.751	0.013	0.55	0.79	0.36	0.89
Smote Oversampler	P=169	N=169	0.751	0.014	0.55	0.80	0.37	0.89
SVM Smote	P=169	N=169	0.761	0.022	0.59	0.79	0.32	0.92

haberman	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=61	N=169	0.692	0.015	0.32	0.15	0.74	0.89
Adasyn	P=184	N=169	0.594	0.022	0.32	0.77	0.48	0.64
Kmeans Smote	P=169	N=169	0.702	0.033	0.39	0.76	0.22	0.88
Random Oversampler	P=169	N=169	0.591	0.026	0.32	0.77	0.47	0.64
Smote Oversampler	P=169	N=169	0.637	0.022	0.37	0.79	0.51	0.68
SVM Smote	P=169	N=169	0.647	0.028	0.35	0.77	0.41	0.73

haberman	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=61	N=169	0.735	0.015	0.50	0.74	0.52	0.99
Adasyn	P=184	N=169	0.663	0.034	0.40	0.82	0.57	0.70
Kmeans Smote	P=169	N=169	0.751	0.033	0.56	0.79	0.31	0.91
Random Oversampler	P=169	N=169	0.722	0.024	0.48	0.82	0.52	0.80
Smote Oversampler	P=169	N=169	0.692	0.023	0.43	0.80	0.48	0.77
SVM Smote	P=169	N=169	0.732	0.028	0.49	0.80	0.41	0.85

haberman	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=61	N=169	0.689	0.219	0.38	0.76	0.27	0.84
Adasyn	P=184	N=169	0.676	0.250	0.39	0.78	0.40	0.78
Kmeans Smote	P=169	N=169	0.709	0.245	0.43	0.77	0.31	0.85
Random Oversampler	P=169	N=169	0.666	0.265	0.37	0.78	0.38	0.77
Smote Oversampler	P=169	N=169	0.676	0.255	0.39	0.78	0.38	0.78
SVM Smote	P=169	N=169	0.699	0.267	0.42	0.78	0.37	0.82

haberman	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=61	N=169	0.725	0.017	0.40	0.74	0.47	0.96
Adasyn	P=184	N=169	0.656	0.033	0.39	0.80	0.52	0.71
Kmeans Smote	P=169	N=169	0.741	0.036	0.53	0.77	0.22	0.93
Random Oversampler	P=169	N=169	0.673	0.023	0.41	0.81	0.54	0.72
Smote Oversampler	P=169	N=169	0.643	0.026	0.37	0.79	0.48	0.70
SVM Smote	P=169	N=169	0.702	0.032	0.44	0.80	0.44	0.80

Αποτελέσματα για Dataset new-thyroid1

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 27 (minority class)

Negative: 135 (majority class)

new-thyroid1	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=27	N=135	0.953	0.009	0.84	0.98	0.89	0.97
Adasyn	P=133	N=135	0.986	0.022	0.94	0.99	0.97	0.99
Kmeans Smote	P=136	N=135	0.972	0.045	0.89	0.99	0.94	0.98
Random Oversampler	P=135	N=135	0.967	0.020	0.89	0.98	0.91	0.98
Smote Oversampler	P=135	N=135	0.981	0.021	0.92	0.99	0.97	0.98
SVM Smote	P=135	N=135	0.981	0.025	0.94	0.99	0.94	0.99

new-thyroid1	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=27	N=135	0.981	0.009	0.90	1.00	1.00	0.98
Adasyn	P=133	N=135	0.948	0.022	0.76	1.00	1.00	0.94
Kmeans Smote	P=136	N=135	0.990	0.033	0.97	0.99	0.97	0.99
Random Oversampler	P=135	N=135	0.948	0.017	0.76	1.00	1.00	0.94
Smote Oversampler	P=135	N=135	0.962	0.021	0.81	1.00	1.00	0.96
SVM Smote	P=135	N=135	0.944	0.044	0.74	1.00	1.00	0.93

new-thyroid 1	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=27	N=135	0.962	0.013	0.97	0.96	0.80	0.99
Adasyn	P=133	N=135	0.967	0.029	0.83	1.00	1.00	0.96
Kmeans Smote	P=136	N=135	0.976	0.035	0.94	0.98	0.91	0.99
Random Oversampler	P=135	N=135	0.972	0.024	0.85	1.00	1.00	0.97
Smote Oversampler	P=135	N=135	0.981	0.028	0.90	1.00	1.00	0.98
SVM Smote	P=135	N=135	0.972	0.030	0.85	1.00	1.00	0.97

new-thyroid 1	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=27	N=135	0.902	0.020	1.00	0.90	0.40	1.00
Adasyn	P=133	N=135	0.967	0.025	0.85	0.99	0.97	0.97
Kmeans Smote	P=136	N=135	0.967	0.033	1.00	0.96	0.80	1.00
Random Oversampler	P=135	N=135	0.981	0.016	0.94	0.99	0.94	0.99
Smote Oversampler	P=135	N=135	0.981	0.021	0.97	0.98	0.91	0.99
SVM Smote	P=135	N=135	0.976	0.031	0.92	0.99	0.94	0.98

new-thyroid1	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=27	N=135	0.990	0.222	1.00	0.99	1.00	0.94
Adasyn	P=133	N=135	0.981	0.265	0.94	0.99	0.94	0.99
Kmeans Smote	P=136	N=135	0.990	0.288	1.00	0.99	1.00	0.94
Random Oversampler	P=135	N=135	0.986	0.219	0.97	0.99	0.94	0.99
Smote Oversampler	P=135	N=135	0.986	0.282	0.97	0.99	0.94	0.99
SVM Smote	P=135	N=135	0.990	0.229	0.95	1.00	1.00	0.99

new-thyroid1	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=27	N=135	0.976	0.011	1.00	0.97	0.86	1.00
Adasyn	P=133	N=135	0.976	0.042	0.88	1.00	1.00	0.97
Kmeans Smote	P=136	N=135	0.976	0.045	0.94	0.98	0.91	0.99
Random Oversampler	P=135	N=135	0.986	0.020	0.92	1.00	1.00	0.98
Smote Oversampler	P=135	N=135	0.986	0.020	0.92	1.00	1.00	0.98
SVM Smote	P=135	N=135	0.986	0.033	0.92	1.00	1.00	0.98

Αποτελέσματα για Dataset page-blocks0

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 419 (minority class)

Negative: 3685 (majority class)

pageblocks	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=419**	**N=3685**	0.966	0.152	0.83	0.98	0.84	0.98
Adasyn	**P=3744**	**N=3685**	0.954	0.315	0.74	0.98	0.86	0.96
Kmeans Smote	**P=3685**	**N=3685**	0.967	0.413	0.84	0.98	0.84	0.98
Random Oversampler	**P=3685**	**N=3685**	0.966	0.170	0.84	0.98	0.83	0.98
Smote Oversampler	**P=3685**	**N=3685**	0.957	0.261	0.76	0.98	0.87	0.97
SVM Smote	**P=3685**	**N=3685**	0.960	0.569	0.78	0.98	0.86	0.97

pageblocks	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=419**	**N=3685**	0.902	0.053	0.55	0.96	0.64	0.94
Adasyn	**P=3744**	**N=3685**	0.910	0.101	0.52	0.94	0.47	0.95
Kmeans Smote	**P=3685**	**N=3685**	0.921	0.162	0.65	0.95	0.52	0.97
Random Oversampler	**P=3685**	**N=3685**	0.896	0.087	0.49	0.94	0.52	0.94
Smote Oversampler	**P=3685**	**N=3685**	0.897	0.088	0.48	0.98	0.83	0.98
SVM Smote	**P=3685**	**N=3685**	0.906	0.326	0.43	0.98	0.58	0.87

pageblocks	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=419**	**N=3685**	0.959	0.150	0.52	0.94	0.47	0.95
Adasyn	**P=3744**	**N=3685**	0.928	0.164	0.60	0.99	0.72	0.93
Kmeans Smote	**P=3685**	**N=3685**	0.959	0.187	0.71	0.95	0.56	0.96
Random Oversampler	**P=3685**	**N=3685**	0.946	0.180	0.49	0.98	0.83	0.90
Smote Oversampler	**P=3685**	**N=3685**	0.940	0.168	0.66	0.99	0.87	0.95
SVM Smote	P=3685	N=3685	0.945	0.477	0.69	0.98	0.76	0.96

pageblocks	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=419**	**N=3685**	0.932	0.082	0.80	0.94	0.45	0.99
Adasyn	**P=3744**	**N=3685**	0.832	0.153	0.37	0.99	0.90	0.83
Kmeans Smote	**P=3685**	**N=3685**	0.931	0.462	0.71	0.95	0.56	0.97
Random Oversampler	**P=3685**	**N=3685**	0.894	0.215	0.49	0.98	0.83	0.90
Smote Oversampler	**P=3685**	**N=3685**	0.890	0.191	0.48	0.98	0.83	0.90
SVM Smote	**P=3685**	**N=3685**	0.871	0.722	0.43	0.98	0.86	0.87

pageblocks	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=419**	**N=3685**	0.975	2.687	0.90	0.98	0.86	0.99
Adasyn	**P=3744**	**N=3685**	0.965	4.259	0.78	0.99	0.92	0.97
Kmeans Smote	**P=3685**	**N=3685**	0.974	3.974	0.89	0.98	0.86	0.99
Random Oversampler	**P=3685**	**N=3685**	0.974	2.691	0.87	0.99	0.88	0.98
Smote Oversampler	**P=3685**	**N=3685**	0.969	3.782	0.81	0.99	0.92	0.98
SVM Smote	**P=3685**	**N=3685**	0.969	5.773	0.82	0.99	0.91	0.98

pageblocks	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=419**	**N=3685**	0.947	0.544	0.90	0.95	0.54	0.99
Adasyn	**P=3744**	**N=3685**	0.908	2.893	0.53	0.99	0.94	0.90
Kmeans Smote	**P=3685**	**N=3685**	0.943	0.827	0.79	0.96	0.61	0.98
Random Oversampler	**P=3685**	**N=3685**	0.927	2.321	0.60	0.99	0.90	0.93
Smote Oversampler	**P=3685**	**N=3685**	0.926	1.736	0.59	0.99	0.91	0.93
SVM Smote	**P=3685**	**N=3685**	0.923	1.837	0.58	0.92	0.92	0.99

Αποτελέσματα για Dataset pima

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 201 (minority class)

Negative: 375 (majority class)

pima	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=201	N=375	0.727	0.021	0.61	0.79	0.60	0.80
Adasyn	P=367	N=375	0.678	0.066	0.54	0.77	0.59	0.72
Kmeans Smote	P=376	N=375	0.665	0.082	0.52	0.75	0.54	0.73
Random Oversampler	P=375	N=375	0.700	0.243	0.57	0.78	0.60	0.75
Smote Oversampler	P=375	N=375	0.697	0.078	0.56	0.77	0.59	0.76
SVM Smote	P=	N=375	0.696	0.064	0.56	0.79	0.63	0.73

pima	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=201	N=375	0.75	0.054	0.66	0.79	0.60	0.83
Adasyn	P=367	N=375	0.739	0.036	0.61	0.83	0.71	0.76
Kmeans Smote	P=376	N=375	0.746	0.046	0.65	0.79	0.60	0.83
Random Oversampler	P=375	N=375	0.740	0.028	0.62	0.82	0.68	0.78
Smote Oversampler	P=375	N=375	0.740	0.037	0.62	0.82	0.69	0.77
SVM Smote	P=375	N=375	0.735	0.052	0.61	0.82	0.69	0.76

pima	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=201	N=375	0.730	0.027	0.63	0.77	0.55	0.83
Adasyn	P=367	N=375	0.683	0.044	0.53	0.82	0.73	0.66
Kmeans Smote	P=376	N=375	0.738	0.059	0.63	0.80	0.62	0.80
Random Oversampler	P=375	N=375	0.696	0.046	0.55	0.80	0.67	0.71
Smote Oversampler	P=375	N=375	0.695	0.038	0.55	0.81	0.71	0.69
SVM Smote	P=375	N=375	0.701	0.091	0.56	0.81	0.69	0.71

pima	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=201	N=375	0.765	0.021	0.74	0.77	0.51	0.90
Adasyn	P=367	N=375	0.753	0.044	0.62	0.85	0.75	0.76
Kmeans Smote	P=376	N=375	0.748	0.053	0.64	0.80	0.63	0.81
Random Oversampler	P=375	N=375	0.753	0.030	0.63	0.84	0.72	0.77
Smote Oversampler	P=375	N=375	0.753	0.075	0.63	0.84	0.73	0.77
SVM Smote	P=375	N=375	0.751	0.060	0.62	0.84	0.72	0.77

pima	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=201	N=375	0.756	0.369	0.68	0.79	0.57	0.85
Adasyn	P=367	N=375	0.752	0.427	0.64	0.82	0.68	0.79
Kmeans Smote	P=376	N=375	0.759	0.432	0.67	0.80	0.61	0.84
Random Oversampler	P=375	N=375	0.766	0.411	0.67	0.82	0.66	0.82
Smote Oversampler	P=375	N=375	0.755	0.422	0.64	0.82	0.67	0.80
SVM Smote	P=375	N=375	0.760	0.486	0.65	0.83	0.69	0.80

pima	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=201	N=375	0.774	0.034	0.72	0.79	0.57	0.88
Adasyn	P=367	N=375	0.733	0.066	0.59	0.86	0.78	0.71
Kmeans Smote	P=376	N=375	0.765	0.100	0.67	0.81	0.65	0.83
Random Oversampler	P=375	N=375	0.740	0.057	0.61	0.83	0.72	0.75
Smote Oversampler	P=375	N=375	0.743	0.056	0.61	0.85	0.76	0.74
SVM Smote	P=375	N=375	0.743	0.110	0.60	0.85	0.76	0.73

Αποτελέσματα για Dataset segment0

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 246 (minority class)

Negative: 1485 (majority class)

segment0	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=246	N=1485	0.989	0.054	0.96	0.99	0.96	0.99
Adasyn	P=1486	N=1485	0.992	0.121	0.96	0.98	1.00	0.99
Kmeans Smote	P=1485	N=1485	0.996	0.132	0.99	1.00	0.99	1.00
Random Oversampler	P=1485	N=1485	0.991	0.080	0.97	1.00	0.97	0.99
Smote Oversampler	P=1485	N=1485	0.993	0.120	0.97	1.00	0.98	0.99
SVM Smote	P=1485	N=1485	0.992	0.153	0.96	1.00	0.99	0.99

segment0	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=246	N=1485	0.832	0.045	0.46	1.00	0.98	0.81
Adasyn	P=1486	N=1485	0.785	0.157	0.39	0.98	0.92	0.76
Kmeans Smote	P=1485	N=1485	0.861	0.100	0.51	0.97	0.85	0.86
Random Oversampler	P=1485	N=1485	0.812	0.097	0.43	1.00	0.98	0.78
Smote Oversampler	P=1485	N=1485	0.821	0.058	0.44	1.00	0.98	0.79
SVM Smote	P=1485	N=1485	0.779	0.105	0.39	0.99	0.96	0.75

segment0	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=246	N=1485	0.990	0.066	0.96	1.00	0.98	0.99
Adasyn	P=1486	N=1485	0.989	0.098	0.93	1.00	0.99	0.99
Kmeans Smote	P=1485	N=1485	0.995	0.116	0.97	1.00	0.99	1.00
Random Oversampler	P=1485	N=1485	0.987	0.089	0.93	1.00	0.99	0.99
Smote Oversampler	P=1485	N=1485	0.988	0.103	0.93	1.00	0.99	0.99
SVM Smote	P=1485	N=1485	0.986	0.185	0.92	1.00	0.99	0.99

segment0	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=246	N=1485	0.975	0.049	0.98	0.97	1.00	0.84
Adasyn	P=1486	N=1485	0.917	0.092	0.63	1.00	1.00	0.90
Kmeans Smote	P=1485	N=1485	0.955	0.118	0.79	0.99	0.94	0.96
Random Oversampler	P=1485	N=1485	0.978	0.100	0.88	1.00	0.98	0.98
Smote Oversampler	P=1485	N=1485	0.978	0.085	0.88	1.00	0.98	0.98
SVM Smote	P=1485	N=1485	0.913	0.137	0.62	1.00	1.00	0.90

segment0	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=246	N=1485	0.997	0.575	1.00	1.00	0.98	1.00
Adasyn	P=1486	N=1485	0.996	1.421	0.99	1.00	0.99	1.00
Kmeans Smote	P=1485	N=1485	0.999	1.311	1.00	1.00	1.00	1.00
Random Oversampler	P=1485	N=1485	0.995	1.113	0.98	1.00	0.98	1.00
Smote Oversampler	P=1485	N=1485	0.996	1.617	0.99	1.00	0.98	1.00
SVM Smote	P=1485	N=1485	0.996	1.430	0.98	1.00	0.99	1.00

segment0	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=246	N=1485	0.996	0.094	1.00	1.00	0.98	1.00
Adasyn	P=1486	N=1485	0.978	0.186	0.87	1.00	1.00	0.98
Kmeans Smote	P=1485	N=1485	0.991	0.242	0.96	1.00	0.98	0.99
Random Oversampler	P=1485	N=1485	0.996	0.129	1.00	1.00	0.98	1.00
Smote Oversampler	P=1485	N=1485	0.996	0.148	1.00	1.00	0.98	1.00
SVM Smote	P=1485	N=1485	0.976	0.237	0.86	1.00	1.00	0.97

Αποτελέσματα για Dataset vehicle0

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 150 (minority class)

Negative: 485 (majority class)

vehicle0	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=150	N=485	0.936	0.033	0.88	0.95	0.84	0.96
Adasyn	P=469	N=485	0.926	0.060	0.84	0.95	0.84	0.95
Kmeans Smote	P=487	N=486	0.955	0.063	0.90	0.97	0.90	0.97
Random Oversampler	P=485	N=485	0.942	0.040	0.89	0.96	0.86	0.97
Smote Oversampler	P=485	N=485	0.939	0.053	0.87	0.96	0.87	0.96
SVM Smote	P=485	N=485	0.950	0.077	0.89	0.97	0.90	0.97

vehicle0	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=150	N=485	0.658	0.021	0.40	0.94	0.87	0.59
Adasyn	P=469	N=485	0.641	0.038	0.38	0.90	0.79	0.60
Kmeans Smote	P=487	N=486	0.691	0.052	0.42	0.94	0.87	0.64
Random Oversampler	P=485	N=485	0.657	0.033	0.40	0.97	0.95	0.57
Smote Oversampler	P=485	N=485	0.663	0.043	0.41	0.97	0.95	0.57
SVM Smote	P=485	N=485	0.651	0.056	0.39	0.93	0.86	0.59

vehicle0	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=150	N=485	0.945	0.032	0.90	0.96	0.87	0.97
Adasyn	P=469	N=485	0.907	0.046	0.73	0.99	0.97	0.89
Kmeans Smote	P=487	N=486	0.938	0.060	0.84	0.97	0.91	0.95
Random Oversampler	P=485	N=485	0.924	0.053	0.77	0.99	0.96	0.91
Smote Oversampler	P=485	N=485	0.914	0.054	0.75	0.99	0.96	0.90
SVM Smote	P=485	N=485	0.911	0.074	0.74	0.98	0.95	0.90

vehicle0	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=150	N=485	0.908	0.037	0.89	0.91	0.70	0.97
Adasyn	P=469	N=485	0.919	0.086	0.75	1.00	0.99	0.90
Kmeans Smote	P=487	N=486	0.942	0.084	0.82	0.99	0.96	0.94
Random Oversampler	P=485	N=485	0.929	0.048	0.77	1.00	0.99	0.91
Smote Oversampler	P=485	N=485	0.929	0.046	0.77	0.99	0.98	0.91
SVM Smote	P=485	N=485	0.920	0.101	0.75	1.00	0.99	0.90

vehicle0	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=150	N=485	0.964	0.360	0.92	0.98	0.92	0.98
Adasyn	P=469	N=485	0.970	0.617	0.91	0.99	0.97	0.97
Kmeans Smote	P=487	N=486	0.981	0.614	0.94	0.99	0.98	0.98
Random Oversampler	P=485	N=485	0.969	0.543	0.90	0.99	0.97	0.97
Smote Oversampler	P=485	N=485	0.966	0.777	0.91	0.99	0.96	0.97
SVM Smote	P=485	N=485	0.968	0.790	0.90	0.99	0.97	0.97

vehicle0	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=150	N=485	0.968	0.037	0.91	0.99	0.96	0.97
Adasyn	P=469	N=485	0.957	0.067	0.85	1.00	1.00	0.94
Kmeans Smote	P=487	N=486	0.971	0.087	0.91	0.99	0.97	0.97
Random Oversampler	P=485	N=485	0.968	0.059	0.88	1.00	1.00	0.96
Smote Oversampler	P=485	N=485	0.965	0.063	0.87	1.00	1.00	0.96
SVM Smote	P=485	N=485	0.959	0.088	0.85	1.00	1.00	0.95

Αποτελέσματα για Dataset wisconsin

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 333 (majority class)

Negative: 180 (minority class)

wisconsin	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=333	N=180	0.951	0.010	0.95	0.95	0.91	0.97
Adasyn	P=328	N=328	0.941	0.033	0.92	0.95	0.91	0.96
Kmeans Smote	P=336	N=333	0.950	0.050	0.95	0.96	0.90	0.98
Random Oversampler	P=333	N=333	0.939	0.023	0.93	0.94	0.89	0.97
Smote Oversampler	P=333	N=333	0.942	0.027	0.93	0.95	0.90	0.97
SVM Smote	P=333	N=333	0.942	0.040	0.92	0.96	0.92	0.96

wisconsin	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=333	N=180	0.961	0.011	0.92	0.99	0.97	0.95
Adasyn	P=328	N=328	0.963	0.242	0.91	0.99	0.99	0.95
Kmeans Smote	P=336	N=333	0.960	0.110	0.92	0.98	0.97	0.95
Random Oversampler	P=333	N=333	0.963	0.025	0.92	0.99	0.98	0.95
Smote Oversampler	P=333	N=333	0.963	0.025	0.92	0.99	0.98	0.95
SVM Smote	P=333	N=333	0.964	0.056	0.92	0.99	0.99	0.95

wisconsin	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=333	N=180	0.970	0.019	0.95	0.98	0.96	0.98
Adasyn	P=328	N=328	0.972	0.043	0.93	1.00	0.96	1.00
Kmeans Smote	P=336	N=333	0.966	0.057	0.94	0.98	0.96	0.97
Random Oversampler	P=333	N=333	0.972	0.035	0.93	1.00	0.99	0.96
Smote Oversampler	P=333	N=333	0.970	0.035	0.94	0.99	0.97	0.97
SVM Smote	P=333	N=333	0.970	0.053	0.93	1.00	0.99	0.96

wisconsin	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	**P=333**	**N=180**	0.967	0.018	0.96	0.97	0.95	0.98
Adasyn	**P=328**	**N=328**	0.976	0.037	0.95	0.99	0.99	0.97
Kmeans Smote	**P=336**	**N=333**	0.966	0.049	0.95	0.98	0.95	0.97
Random Oversampler	**P=333**	**N=333**	0.969	0.029	0.95	0.96	0.96	0.97
Smote Oversampler	**P=333**	**N=333**	0.970	0.027	0.95	0.98	0.97	0.97
SVM Smote	**P=333**	**N=333**	0.972	0.036	0.94	0.99	0.99	0.96

wisconsin	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=333	N=180	0.967	0.245	0.95	0.98	0.96	0.97
Adasyn	P=328	N=328	0.973	0.447	0.95	0.97	0.97	0.97
Kmeans Smote	P=336	N=333	0.967	0.354	0.95	0.98	0.96	0.97
Random Oversampler	P=333	N=333	0.970	0.272	0.95	0.98	0.97	0.97
Smote Oversampler	P=333	N=333	0.970	0.293	0.95	0.98	0.97	0.97
SVM Smote	P=333	N=333	0.970	0.301	0.94	0.99	0.97	0.97

wisconsin	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=333	N=180	0.972	0.034	0.94	0.99	0.98	0.97
Adasyn	P=328	N=328	0.969	0.034	0.92	1.00	1.00	0.95
Kmeans Smote	P=336	N=333	0.972	0.049	0.94	0.99	0.98	0.97
Random Oversampler	P=333	N=333	0.972	0.029	0.94	0.99	0.99	0.96
Smote Oversampler	P=333	N=333	0.973	0.042	0.94	0.99	0.98	0.97
SVM Smote	P=333	N=333	0.969	0.046	0.92	1.00	1.00	0.95

Αποτελέσματα για Dataset yeast1

Αρχικός αριθμός δειγμάτων ανά κλάση:

Positive: 321 (minority class)

Negative: 792 (majority class)

yeast1	Sample counts		Decision Trees					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=321	N=792	0.710	0.028	0.50	0.80	0.51	0.79
Adasyn	P=816	N=792	0.710	0.075	0.50	0.82	0.60	0.76
Kmeans Smote	P=792	N=792	0.702	0.080	0.49	0.79	0.49	0.79
Random Oversampler	P=792	N=792	0.702	0.044	0.48	0.78	0.44	0.79
Smote Oversampler	P=792	N=792	0.686	0.054	0.46	0.80	0.53	0.75
SVM Smote	P=792	N=792	0.712	0.139	0.50	0.81	0.54	0.78

yeast1	Sample counts		Gaussian Naive Bayes					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=321	N=792	0.317	0.015	0.30	0.90	0.99	0.04
Adasyn	P=816	N=792	0.319	0.044	0.30	0.02	0.05	0.99
Kmeans Smote	P=792	N=792	0.354	0.055	0.31	0.93	0.98	0.10
Random Oversampler	P=792	N=792	0.314	0.045	0.30	0.91	0.04	0.99
Smote Oversampler	P=792	N=792	0.321	0.070	0.30	0.93	0.99	0.05
SVM Smote	P=792	N=792	0.322	0.108	0.30	0.93	0.99	0.05

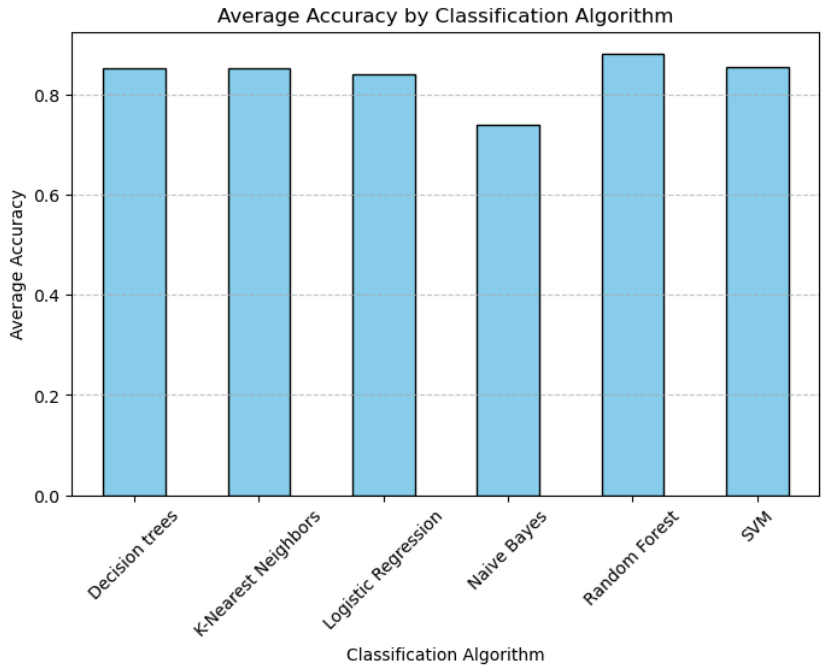
yeast1	Sample counts		KNN					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=321	N=792	0.754	0.038	0.60	0.80	0.45	0.88
Adasyn	P=816	N=792	0.673	0.144	0.46	0.85	0.72	0.66
Kmeans Smote	P=792	N=792	0.755	0.102	0.59	0.81	0.50	0.86
Random Oversampler	P=792	N=792	0.681	0.087	0.46	0.83	0.66	0.69
Smote Oversampler	P=792	N=792	0.683	0.068	0.47	0.83	0.66	0.69
SVM Smote	P=792	N=792	0.690	0.162	0.47	0.83	0.64	0.71

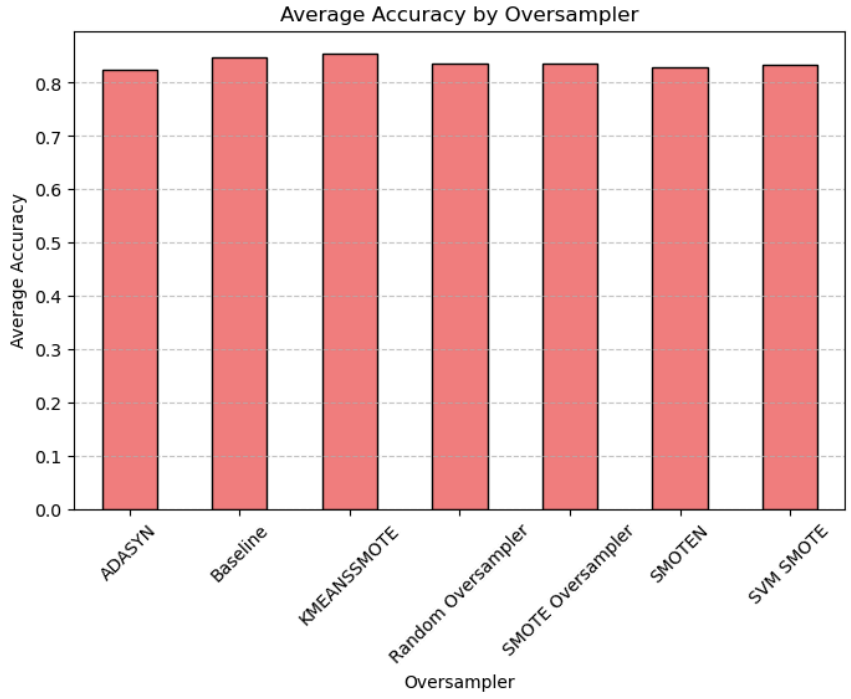
yeast1	Sample counts		Logistic Regression					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=321	N=792	0.742	0.049	0.65	0.75	0.24	0.95
Adasyn	P=816	N=792	0.682	0.058	0.47	0.77	0.77	0.65
Kmeans Smote	P=792	N=792	0.765	0.071	0.62	0.81	0.49	0.88
Random Oversampler	P=792	N=792	0.716	0.046	0.51	0.86	0.70	0.72
Smote Oversampler	P=792	N=792	0.709	0.087	0.50	0.85	0.70	0.71
SVM Smote	P=792	N=792	0.723	0.121	0.47	0.84	0.66	0.70

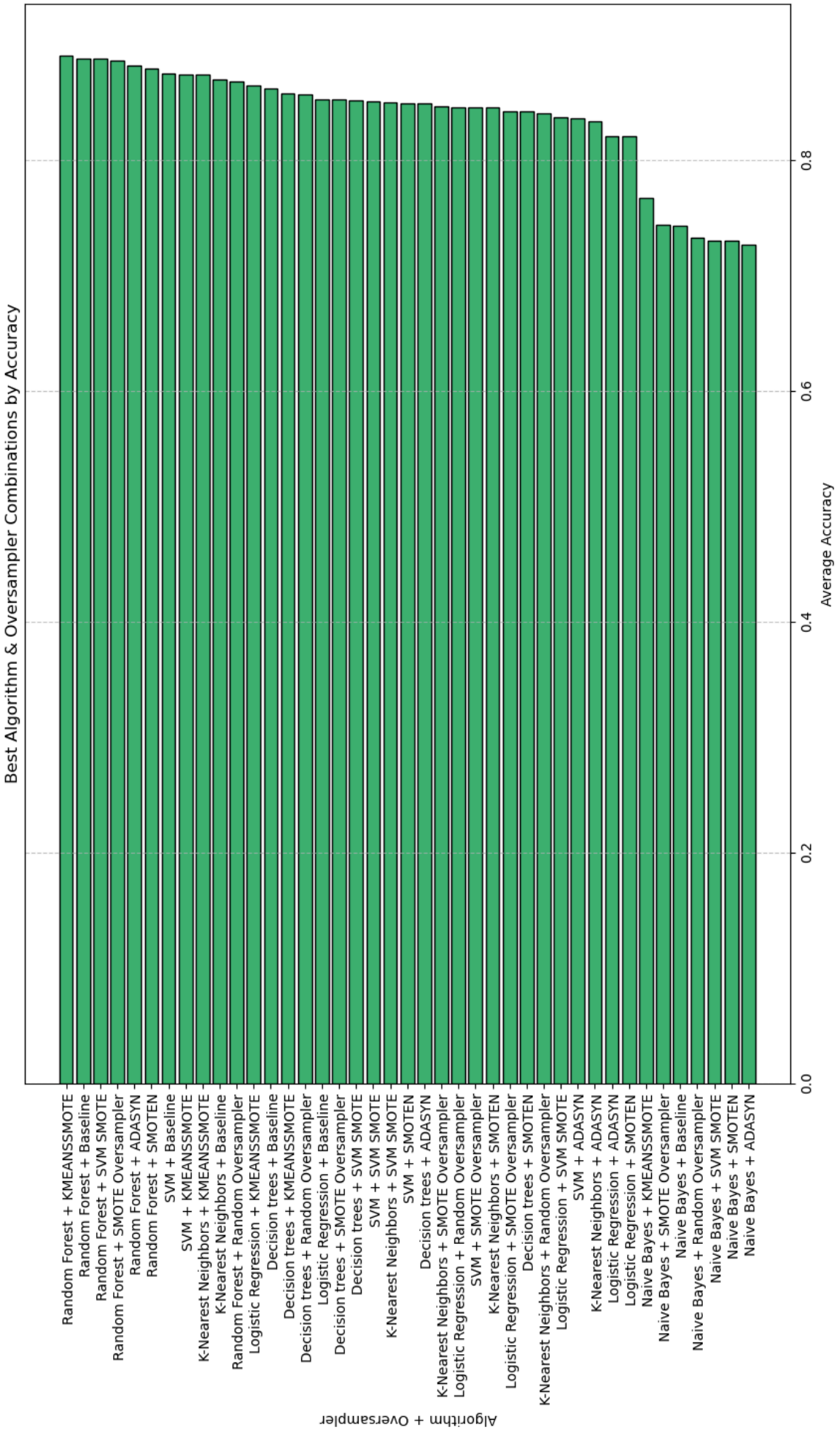
yeast1	Sample counts		Random Forest					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=321	N=792	0.768	0.569	0.64	0.80	0.45	0.90
Adasyn	P=816	N=792	0.754	0.694	0.57	0.84	0.61	0.81
Kmeans Smote	P=792	N=792	0.772	0.608	0.65	0.81	0.47	0.90
Random Oversampler	P=792	N=792	0.770	0.651	0.61	0.83	0.56	0.85
Smote Oversampler	P=792	N=792	0.762	0.867	0.59	0.84	0.60	0.83
SVM Smote	P=792	N=792	0.762	0.690	0.59	0.84	0.60	0.83

yeast1	Sample counts		SVMs					
			Accuracy	CPU Train Time (second)	Precision		Recall	
					Positive Class	Negative Class	Positive Class	Negative Class
Baseline	P=321	N=792	0.758	0.137	0.71	0.76	0.28	0.95
Adasyn	P=816	N=792	0.663	0.346	0.46	0.90	0.84	0.59
Kmeans Smote	P=792	N=792	0.759	0.205	0.64	0.79	0.39	0.91
Random Oversampler	P=792	N=792	0.696	0.179	0.48	0.88	0.77	0.67
Smote Oversampler	P=792	N=792	0.696	0.175	0.48	0.88	0.77	0.67
SVM Smote	P=792	N=792	0.718	0.287	0.51	0.87	0.73	0.71

5.4 Συγκεντρωτικοί πίνακες αποτελεσμάτων






Best Algorithm & Oversampler Combinations by Accuracy
Algorithm + Oversampler
Average Accuracy
0.0
0.2
0.4
0.6
0.8
Random Forest + KMEANSSMOTE
Random Forest + Baseline
Random Forest + SVM SMOTE
Random Forest + SMOTE Oversampler
Random Forest + ADASYN
Random Forest + SMOTEN
SVM + Baseline
SVM + KMEANSSMOTE
K-Nearest Neighbors + KMEANSSMOTE
K-Nearest Neighbors + Baseline
Random Forest + Random Oversampler
Logistic Regression + KMEANSSMOTE
Decision trees + Baseline
Decision trees + KMEANSSMOTE
Decision trees + Random Oversampler
Logistic Regression + Baseline
Decision trees + SMOTE Oversampler
Decision trees + SVM SMOTE
SVM + SVM SMOTE
K-Nearest Neighbors + SVM SMOTE
SVM + SMOTEN
Decision trees + ADASYN
K-Nearest Neighbors + SMOTE Oversampler
Logistic Regression + Random Oversampler
SVM + SMOTE Oversampler
K-Nearest Neighbors + SMOTEN
Logistic Regression + SMOTE Oversampler
Decision trees + SMOTEN
K-Nearest Neighbors + Random Oversampler
Logistic Regression + SVM SMOTE
SVM + ADASYN
K-Nearest Neighbors + ADASYN
Logistic Regression + ADASYN
Logistic Regression + SMOTEN
Naive Bayes + KMEANSSMOTE
Naive Bayes + SMOTE Oversampler
Naive Bayes + Baseline
Naive Bayes + Random Oversampler
Naive Bayes + SVM SMOTE
Naive Bayes + SMOTEN
Naive Bayes + ADASYN

Κεφάλαιο 6ο: Συμπεράσματα

Ο πρώτος πίνακας συγκρίνει τη μέση ακρίβεια που επιτυγχάνεται από διαφορετικές μεθόδους υπερδειγματοληψίας, συμπεριλαμβανομένων των ADASYN, KMEANS SMOTE, Random Oversampler, SMOTE, SMOTEN και SVM SMOTE, μαζί με τον Baseline (χωρίς υπερδειγματοληψία). Τα αποτελέσματα δείχνουν ότι οι περισσότερες τεχνικές υπερδειγματοληψίας προσφέρουν μόνο οριακές βελτιώσεις σε σχέση με το Baseline. Το KMEANS SMOTE και το Baseline επιτυγχάνουν την υψηλότερη ακρίβεια, υποδηλώνοντας ότι η υπερδειγματοληψία μπορεί να μην είναι πάντα απαραίτητη όταν χρησιμοποιείται ένας καλά βελτιστοποιημένος ταξινομητής.

Αυτό το εύρημα είναι σημαντικό, καθώς αμφισβητεί την υπόθεση ότι η υπερδειγματοληψία θα οδηγεί πάντα σε βελτιωμένη απόδοση του μοντέλου. Σε ορισμένες περιπτώσεις, ιδιαίτερα όταν ο ταξινομητής είναι ήδη ισχυρός, η υπερδειγματοληψία μπορεί να εισάγει θόρυβο ή περιττές πληροφορίες αντί να βελτιώνει την ακρίβεια της ταξινόμησης.

Ο δεύτερος πίνακας αξιολογεί διάφορους αλγορίθμους ταξινόμησης χωρίς να λαμβάνει υπόψη την υπερδειγματοληψία. Τα αποτελέσματα δείχνουν ότι ο Random Forest επιτυγχάνει τη μεγαλύτερη ακρίβεια, καθιστώντας τον τον πιο αποτελεσματικό ταξινομητή για αυτό το σύνολο δεδομένων. Οι K-Nearest Neighbors (KNN) και Support Vector Machines (SVM) αποδίδουν επίσης καλά, υποδηλώνοντας ότι είναι κατάλληλες επιλογές για αυτήν την ταξινόμηση. Η Logistic Regression ακολουθεί, με ελαφρώς χαμηλότερη απόδοση από το KNN και το SVM, αλλά διατηρεί υψηλή ακρίβεια. Ο Naïve Bayes αποδίδει χειρότερα, με σημαντικά χαμηλότερη ακρίβεια από τα υπόλοιπα μοντέλα. Αυτά τα αποτελέσματα υπογραμμίζουν ότι η επιλογή του μοντέλου είναι εξίσου σημαντική με την υπερδειγματοληψία. Αν ένας ταξινομητής, όπως ο Random Forest, ήδη αποδίδει καλά, η προσθήκη μιας τεχνικής υπερδειγματοληψίας μπορεί να προσφέρει μόνο μια μικρή ή ακόμα και αμελητέα βελτίωση στην ακρίβεια.

Ο τρίτος πίνακας κατατάσσει τους πιο αποδοτικούς συνδυασμούς ταξινομητών και τεχνικών υπερδειγματοληψίας. Οι κορυφαίοι συνδυασμοί περιλαμβάνουν τον Random Forest με KMEANS SMOTE, που επιτυγχάνει την υψηλότερη ακρίβεια, τον Random Forest δίχως Oversampler, SVM SMOTE και SMOTE Oversampler, οι οποίοι επίσης κατατάσσονται υψηλά, τα SVM και ο KNN σε συνδυασμό με KMEANS SMOTE oversampling, που αποδίδουν καλά, ενισχύοντας την αποτελεσματικότητα αυτών των ταξινομητών. Αξιοσημείωτο είναι ότι οι συνδυασμοί με τη **χαμηλότερη απόδοση περιλαμβάνουν σταθερά τον Naïve Bayes**, ανεξάρτητα από την τεχνική υπερδειγματοληψίας που χρησιμοποιείται. Αυτό υποδηλώνει ότι ο Naïve Bayes αντιμετωπίζει δυσκολίες με αυτό το σύνολο δεδομένων, πιθανότατα λόγω των απλοποιητικών υποθέσεών του σχετικά με την ανεξαρτησία των χαρακτηριστικών, οι οποίες μπορεί να μην ισχύουν στην πράξη.

Συμπερασματικά, μπορούμε να δούμε πως ο Random Forest είναι ο πιο αποτελεσματικός ταξινομητής, ξεπερνώντας τους άλλους ακόμα και με ελάχιστη ή και καθόλου υπερδειγματοληψία. Οι μέθοδοι υπερδειγματοληψίας δεν βελτιώνουν σημαντικά την ακρίβεια συνολικά, γεγονός που υποδηλώνει ότι η επιλογή του μοντέλου έχει μεγαλύτερο αντίκτυπο. Το KMEANS SMOTE είναι η πιο αποτελεσματική τεχνική υπερδειγματοληψίας, αλλά η χρησιμότητά της εξαρτάται από τον ταξινομητή με τον οποίο συνδυάζεται. Ο Naïve Bayes αποδίδει σταθερά χαμηλότερα, δείχνοντας ότι ορισμένοι ταξινομητές είναι εγγενώς λιγότερο κατάλληλοι για αυτό το σύνολο δεδομένων, ανεξάρτητα από την υπερδειγματοληψία. Τελικώς, φαίνεται, πως αν και η υπερδειγματοληψία μπορεί να είναι χρήσιμη σε ορισμένες περιπτώσεις, η επιλογή του κατάλληλου μοντέλου ταξινόμησης παίζει πολύ πιο καθοριστικό ρόλο στην επίτευξη υψηλής ακρίβειας. Ο Random Forest, ειδικότερα, αποδεικνύεται ένας ισχυρός ταξινομητής, καθιστώντας τον την καλύτερη επιλογή για την ταξινόμηση σε αυτό το σενάριο.

ΒΙΒΛΙΟΓΡΑΦΙΑ:

[1] He, H., & Bai, Y. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. IEEE International Joint Conference on Neural Networks, 1322-1328. https://sci2s.ugr.es/keel/pdf/algorithm/congreso/2008-He-ieee.pdf

[2] He, H., & Bai, Y. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. IEEE International Joint Conference on Neural Networks, 1322-1328.

[3] SMOTE: Synthetic Minority Over-sampling Technique. N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer

[4] Lemaître, G., Nogueira, F., & Aridas, C. K. (2024). *imblearn.over_sampling.RandomOverSampler* (Version stable). Imbalanced-learn. Retrieved February 20, 2025, from https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html

[5] IBM. (2025). *Logistic Regression*. IBM, from https://www.ibm.com/topics/logistic-regression

[6] Kanade, V. (2023). *What Is Linear Regression? Types, Equation, Examples, and Best Practices for 2022*. Spiceworks. from https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-linear-regression

[7] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, Applied Logistic Regression. Wiley Series in Probability and Statistics, 2013.

[8] Techniques and algorithms for computer aided diagnosis of pigmented skin lesions—A review

Sameena Pathan, ... P.C. Siddalingaswamy, in Biomedical Signal Processing and Control, 2018

[9] GeeksforGeeks, "Decision tree algorithms," 2023. Accessed: 2024-12-21.

[10] A. Liaw and M. Wiener, "Classification and regression by randomforest," R news, vol. 2, no. 3, pp. 18–22, 2002.

[11] L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, pp. 5–32, 2001.

[12] C. M. Bishop, "Pattern recognition and machine learning," Springer, 2006.

[13] Evgeniou, Theodoros & Pontil, Massimiliano. (2001). Support Vector Machines: Theory and Applications. 2049. 249-257. 10.1007/3-540-44673-7_12.

[14] Huibing Wang, Jinbo Xiong, Zhiqiang Yao, Mingwei Lin, and Jun Ren. 2017. Research Survey on Support Vector Machine. In Proceedings of EAI International Conference on Mobile Multimedia Communications, Chongqing, People's Republic of China, July 2017 (10th), 9 pages. DOI: 10.475/123 4

[15] I. P. Sari, A. N. Hidayanto, and E. Yuniarno, "Implementation of K-Nearest Neighbor (K-NN) Algorithm to Predict the Level of Damage to Road Infrastructure," *IOP Conference Series: Materials Science and Engineering*, vol. 719, no. 1, p. 012072, 2020. DOI:10.1088/1757-899X/719/1/012072

[16] M. A. Jabbar, B. L. Deekshatulu, and P. Chandra, "Heart Disease Classification Using Nearest Neighbor Classifier with Feature Subset Selection," *Procedia Computer Science*, vol. 85, pp. 125–132, 2016. DOI:10.1016/j.procs.2016.05.180

[17] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," IEEE transactions on information theory, vol. 13, no. 1, pp. 21–27, 1967.

[18] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," in AAAI-98 workshop on learning for text categorization, pp. 41–48, 1998.

[19] Scikit-learn Development Team, “Naive bayes documentation - scikit-learn,” 2024. Accessed: 22 December 2024.