ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΝΕΩΝ ΑΛΓΟΡΙΘΜΩΝ ΕΞΟΡΥΞΗΣ ΔΕΔΟΜΕΝΩΝ

ΣΜΥΡΛΗΣ ΠΑΝΑΓΙΩΤΗΣ
ΤΣΟΥΡΟΣ ΔΗΜΟΣ

Επιβλέποντες :

ΣΑΡΗΓΙΑΝΝΙΔΗΣ ΠΑΝΑΓΙΩΤΗΣ
ΤΣΙΠΟΥΡΑΣ ΜΑΡΚΟΣ

Κοζάνη, Νοέμβρης 2017

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΝΕΩΝ ΑΛΓΟΡΙΘΜΩΝ ΕΞΟΡΥΞΗΣ ΔΕΔΟΜΕΝΩΝ

ΣΜΥΡΛΗΣ ΠΑΝΑΓΙΩΤΗΣ
ΤΣΟΥΡΟΣ ΔΗΜΟΣ

Επιβλέποντες :

ΣΑΡΗΓΙΑΝΝΙΔΗΣ ΠΑΝΑΓΙΩΤΗΣ
ΤΣΙΠΟΥΡΑΣ ΜΑΡΚΟΣ

Κοζάνη, Νοέμβρης 2017

Δήλωση Πνευματικών Δικαιωμάτων

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν.1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο

“Ανάπτυξη νέων αλγορίθμων εξόρυξης δεδομένων”

καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Σαρηγιαννίδη Παναγιώτη, και του έκτακτου διδάσκοντα κ. Μάρκου Τσίπουρα αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας ήταν η ανάπτυξη δύο νέων αλγορίθμων εξόρυξης δεδομένων, για το πρόβλημα της κατηγοριοποίησης. Ο πρώτος, ο Constrained K-Means Classification, είναι αλγόριθμος κατηγοριοποίησης μέσω ομαδοποίησης ενώ ο δεύτερος, ο Stochastic Forest, είναι αλγόριθμος - κατηγοριοποιητής ομάδας.

Οι δύο νέοι αλγόριθμοι αξιολογήθηκαν σε πλήθος σύνολων δεδομένων από το UCI machine learning repository, μεταξύ των οποίων και τα πιο γνωστά και χρησιμοποιημένα. Έγινε επίσης συγκριτική μελέτη σε σχέση με αρκετούς γνωστούς αλγορίθμους εξόρυξης δεδομένων και φυσικά με τους επικρατέστερους αλγορίθμους που χρησιμοποιούνται για κατηγοριοποίηση μέσω ομαδοποίησης (K-Means) και κατηγοριοποίηση με τη μέθοδο ομάδων (Random Forest).

Ακόμα, ο πρώτος αλγόριθμος αξιολογήθηκε ως προς την ικανότητα εφαρμογής του σε ένα ιατρικό πρόβλημα, για την εκτίμησης της εξέλιξης της ίνωσης του ήπατος, από εικόνες βιοψίας, αλλά και ως προς την καταλληλότητά του για το πρόβλημα συγκριτικά με έναν τους πιο γνωστούς αλγορίθμους κατηγοριοποίησης.

Τα αποτελέσματα από την αξιολόγηση και των δύο αλγορίθμων, είναι ενθαρρυντικά, παρουσιάζουν σημαντικά ποσοστά επιτυχίας, με βάση και τις τρεις μετρικές απόδοσης που χρησιμοποιήθηκαν (Accuracy, Sensitivity, Precision). Επιπλέον, παρουσιάζουν καλύτερα αποτελέσματα από τους επικρατέστερους αλγορίθμους στην κατηγορία τους.

Λέξεις Κλειδιά: Εξόρυξη Δεδομένων, Αλγόριθμος Κ-Μέσων, Περιορισμοί, Ομαδοποίηση, Κατηγοριοποίηση, Κατηγοριοποίηση Μέσω Ομαδοποίησης, Βάρη, Εμβαρυμένη Ευκλείδεια Απόσταση, Κατηγοριοποιητές Ομάδας, Τυχαία Δάση, Στοχαστικά Δάση, Δέντρα Απόφασης

Abstract

The scope of this thesis was the development of two data mining algorithms, specifically on the field of classification. The first algorithm, Constrained K-Means Classification, is a classification via clustering algorithm, while the second, Stochastic Forest, is an ensemble classifier.

Both algorithms were evaluated with numerous datasets from the “UCI machine learning repository”, including the most used and well known datasets. Also, comparative study took place, between our algorithms and several well known classification algorithms and of course the dominant algorithm on their fields, classification via clustering algorithm (K-Means) and ensemble classification (Random Forest).

Furthermore, the first algorithm was evaluated on its ability to be applied to the medical problem of predicting the severity of liver fibrosis out of liver biopsy images. Also, it was evaluated on the problem compared to a well know classification algorithm.

The evaluation of both algorithms, presented significant and encouraging results, according to all metrics used (Accuracy, Sensitivity, Precision). Moreover, they present significant improvement compared to the algorithms mentioned above., on their category respectively.

Keywords: Data Mining, K-Means, Constraints, Clustering, Classification, Classification via Clustering, Weights, Weighted Euclidean Distance, Ensemble Classifiers, Random Forest, Stochastic Forest, Decision Trees

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 2

Κεφάλαιο 6

ΕΠΕΞΗΓΗΣΕΙΣ ΣΥΜΒΟΛΙΣΜΩΝ

N : Πλήθος κατηγοριοποιητών σε κάποιον κατηγοριοποιητή ομάδας

X : Πλήθος εγγραφών του συνόλου δεδομένων

A : Πλήθος χαρακτηριστικών του συνόλου δεδομένων

x_i : Εγγραφή του συνόλου δεδομένων

a_i : Τα χαρακτηριστικά του συνόλου δεδομένων

y_i : ετικέτα κατηγορίας

D : Σύνολο δεδομένων

E : Εντροπία

p_i : η πιθανότητα εμφάνισης της κατηγορίας y_i

c : Το πλήθος των κατηγοριών ενός συνόλου δεδομένων

IG : Κέρδος πληροφορίας

V_a : Το σύνολο των κλαδιών που δημιουργούνται σε ένα δέντρο απόφασης μετά από κάποιον διαχωρισμό

D_v : Υποσύνολο δεδομένων σε κάποιο κλαδί του δέντρου

S : Πληροφορία διαχωρισμού

GAIN : Αναλογία κέρδους

F : Πλήθος χαρακτηριστικών για την εκπαίδευση κάθε δέντρου απόφασης, στον αλγόριθμο των τυχαίων δασών

s : ευαισθησία (Sensitivity)

p : ακρίβεια (Precision)

K : Πλήθος των κέντρων στον αλγόριθμο K-μέσων

b : Περιορισμοί

m : Μέσος όρος

l : Συντελεστής αυστηρότητας περιορισμών

std : Τυπική απόκλιση

w : Συντελεστής βάρους χαρακτηριστικού

P(i) : Πιθανότητα επιλογής του χαρακτηριστικού i, με τον αλγόριθμος Stochastic Forest

MI : Το μικρότερο αποδεκτό πλήθος εγγραφών σε κάθε κλαδί του δέντρου, με βάση των αλγόριθμο C4.5

CPA : Αναλογία κολλαγόνου προς τον ηπατικό ιστό

DT : Decision Table

Bayes : Ο αλγόριθμος Naive Bayes

MLP : Ο αλγόριθμος “Πολυεπίπεδο Perceptron”

J48 : Υλοποίηση του αλγορίθμου C4.5 για κατασκευή δέντρων απόφασης

RF : Random Forest

SF : Stochastic Forest

C. K-Means: Constrained K – Means Classification

1 *Εισαγωγή*

Σε αυτό το Κεφάλαιο γίνεται αναφορά στις βασικές έννοιες και περιγράφεται ο χώρος στον οποίο ανήκει η παρούσα διπλωματική εργασία. Επιπλέον παρουσιάζεται το αντικείμενο της παρούσας διπλωματικής, τα προβλήματα στα οποία έχει εφαρμογή και οι στόχοι της. Τέλος, παρουσιάζεται η δομή της και τα περιεχόμενα των κεφαλαίων.

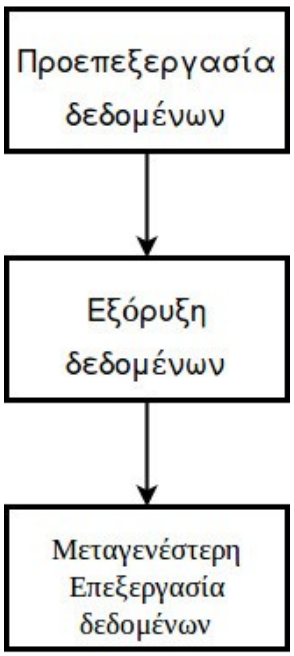
1.1 Εξόρυξη Δεδομένων

Οι ραγδαίες εξελίξεις στην τεχνολογία αποθήκευσης δεδομένων και συνολικά στη συλλογή δεδομένων και τη διακίνηση των πληροφοριών έχουν οδηγήσει στη δημιουργία πολύ μεγάλων σύνολων δεδομένων κάθε είδους, σχεδόν σε κάθε επιστημονικό κλάδο, π.χ. ιατρικά δεδομένα, δεδομένα κατανάλωσης σε ένα ή περισσότερα καταστήματα, δεδομένα που έχουν καταγραφεί από αισθητήρες ή φωτογραφίες από ένα δορυφόρο.

Θεωρώντας αυτές τις μεγάλες ποσότητες δεδομένων ως πληροφορίες, είναι αναγκαία η εξαγωγή της απαραίτητης γνώσης για την αποτελεσματική αξιοποίησή τους. Η ανακάλυψη γνώσης σε μεγάλα σύνολα δεδομένων έχει αποδειχθεί μεγάλη πρόκληση, αφού συχνά, λόγω της ποσότητας των δεδομένων ή της μη συμβατικής φύσης τους, δεν μπορούν να εφαρμοστούν κλασσικές στατιστικές προσεγγίσεις. Το

πρόβλημα που προκύπτει είναι αν υπάρχει τρόπος να διαχειριστούμε τέτοια σύνολα δεδομένων, ώστε να εξάγουμε τα κατάλληλα συμπεράσματα π.χ. για ιατρικά προβλήματα να μπορούμε να καθορίσουμε την σοβαρότητα μιας ασθένειας, ή με τα δεδομένα παραγωγής και κατανάλωσης προϊόντων να εξασφαλιστούν οι προδιαγραφές για την κάλυψη των αναγκών της κοινωνίας με τον κατάλληλο σχεδιασμό.

Εξόρυξη δεδομένων (Data Mining) [1] είναι η διαδικασία της αυτόματης ανακάλυψης χρήσιμων πληροφοριών μέσα από μεγάλα σύνολα δεδομένων. Ο επικρατέστερος ορισμός είναι: «Εξόρυξη Δεδομένων είναι η ανάλυση – συνήθως τεράστιων – παρατηρούμενων (observational) συνόλων δεδομένων, έτσι ώστε να βρεθούν μη παρατηρηθείσες σχέσεις και να συνοψιστούν τα δεδομένα με τρόπους οι οποίοι να είναι κατανοητοί και χρήσιμοι στον κάτοχο των δεδομένων» (Hand et al., 2001) [2]. Ο παραπάνω ορισμός με τη σύνοψη των δεδομένων και την εύρεση σχέσεων εννοεί την περιγραφή τους ως μοντέλο – πρότυπο.



Εικόνα 1.1 - Διαδικασία ανακάλυψης γνώσης

Η εξόρυξη δεδομένων είναι αναπόσπαστο κομμάτι της Ανακάλυψης Γνώσης από Βάσεις Δεδομένων (Knowledge Discovery in Databases – KDD). Αυτή η διαδικασία

αποτελεί τη συνολική διεργασία μετατροπής ακατέργαστων δεδομένων σε αξιοποιήσιμες πληροφορίες. Αποτελείται από: Την προεπεξεργασία δεδομένων (preprocessing), την Εξόρυξη Δεδομένων και τη μεταγενέστερη επεξεργασία των δεδομένων (postprocessing) όπως φαίνεται στην Εικόνα 1.1.

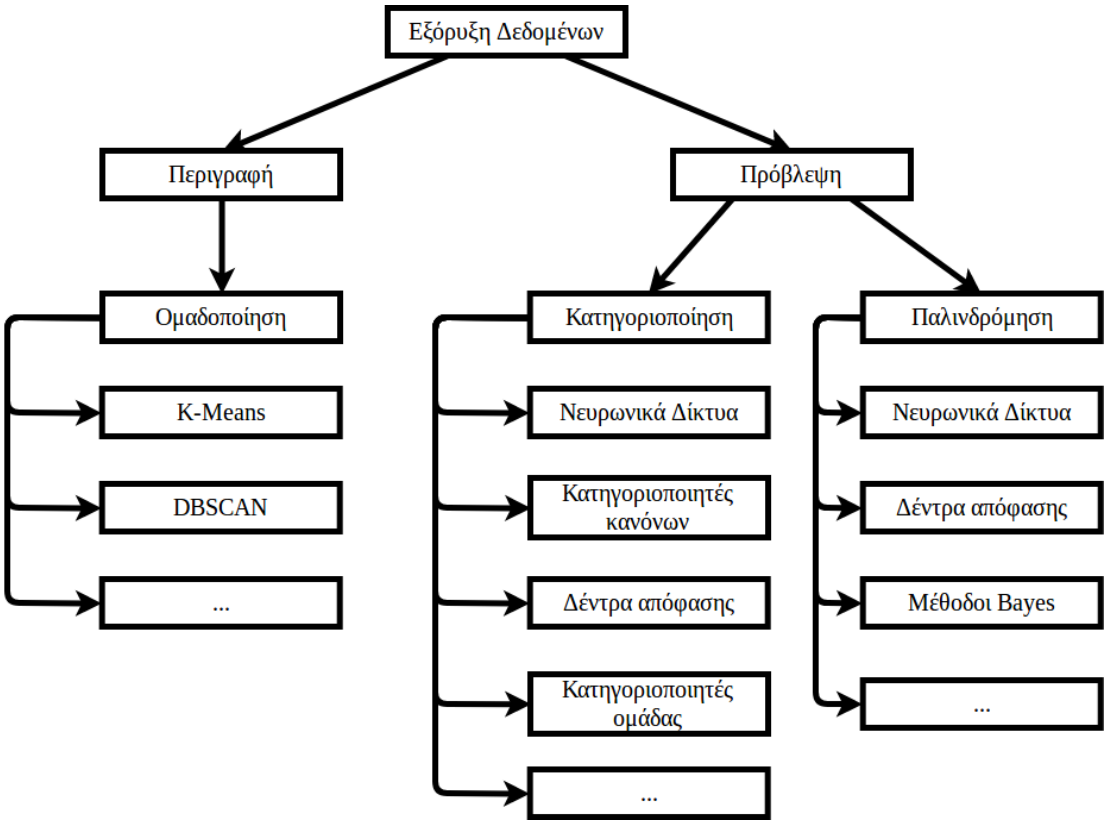
Με την επεξεργασία μεγάλου σύνολου δεδομένων, στόχος είναι να ανακαλύψουμε την ύπαρξη της λεγόμενης «κρυμμένης γνώσης». Δηλαδή να εντοπίσουμε συσχετίσεις, αλληλεξάρτηση, ομαδοποιήσεις ή κανόνες μεταξύ των δεδομένων, τα οποία μπορεί να μην είναι άμεσα εμφανή, αλλά να είναι χρήσιμα για την αναγνώριση των προτύπων που επικρατούν ή/και την δημιουργία προβλέψεων όσον αφορά τη μελλοντική αξία ή συμπεριφορά κάποιων μεταβλητών/χαρακτηριστικών. Με βάση τα όσα αναφέρονται παραπάνω μπορεί να ειπωθεί πως βασική δουλειά και στόχος της Εξόρυξης Δεδομένων είναι η *περιγραφή* και η *πρόβλεψη*.

Οι *περιγραφικές εργασίες* έχουν ως στόχο την ερμηνεία των δεδομένων και επικεντρώνονται στην κατανόηση του τρόπου που σχετίζονται τα δεδομένα, εξάγοντας υποδείγματα (συσχετίσεις, συστάδες – ομάδες, τάσεις κλπ.) για να συνοψίσουν τις βασικές σχέσεις που υπάρχουν μεταξύ των δεδομένων. Η βασική εφαρμογή περιγραφικών εργασιών είναι η συσταδοποίηση – ομαδοποίηση. Η συσταδοποίηση – ομαδοποίηση (clustering) [3] έχει ως στόχο να βρει ομάδες δεδομένων με κοινά στοιχεία ως προς τα χαρακτηριστικά τους. Πιο αναλυτικά για την ομαδοποίηση θα αναφερθούμε σε επόμενη ενότητα.

Οι *προγνωστικές εργασίες* έχουν ως στόχο την πρόβλεψη της τιμής ενός χαρακτηριστικού βασιζόμενες στις τιμές των υπόλοιπων χαρακτηριστικών του σύνολου δεδομένων. Το χαρακτηριστικό του οποίου την τιμή θέλουμε να προβλέψουμε είναι γνωστό και ως εξαρτημένη μεταβλητή ενώ τα υπόλοιπα χαρακτηριστικά ως ανεξάρτητες μεταβλητές.

Η προγνωστική μοντελοποίηση αναφέρεται στην προσπάθεια δημιουργίας ενός μοντέλου για την εξαρτημένη μεταβλητή ως συνάρτηση των ανεξάρτητων μεταβλητών. Υπάρχουν 2 τύποι τέτοιων εργασιών: η κατηγοριοποίηση (classification), η οποία χρησιμοποιείται όταν η μεταβλητή στόχος είναι διακριτή – κατηγορική, και η παλινδρόμηση (regression) η οποία χρησιμοποιείται όταν η μεταβλητή στόχος είναι συνεχής. Για παράδειγμα, η προσπάθεια διάγνωσης στην ιατρική, αν θεωρήσουμε το χαρακτηριστικό *Άρρωστος*, που παίρνει τιμές *ΝΑΙ - ΟΧΙ ,*

είναι εργασία κατηγοριοποίησης βασιζόμενη σε μια σειρά ιατρικών εξετάσεων - δεδομένων. Αντίθετα, η προσπάθεια πρόβλεψης της κατανάλωσης ρεύματος σε ένα μήνα, συνάρτηση διάφορων παραγόντων όπως η εποχή, η θερμοκρασία κλπ, είναι εργασία παλινδρόμησης, αφού η εξαρτημένη μεταβλητή παίρνει συνεχείς τιμές. Στην Εικόνα 1.2 παρουσιάζεται η ταξινόμηση των βασικών εργασιών τη Εξόρυξης Δεδομένων.



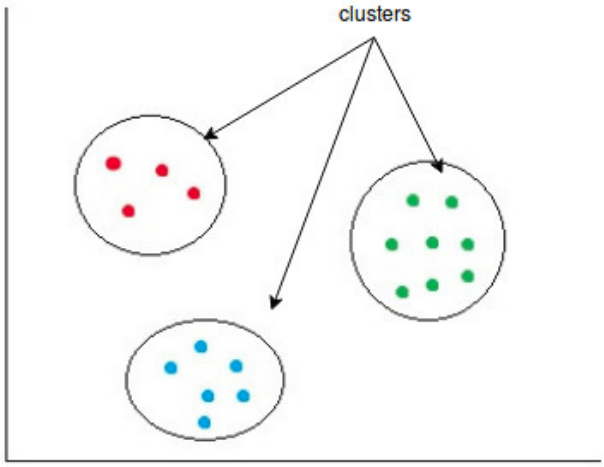
Εικόνα 1.2 – Βασικές εργασίες της εξόρυξης δεδομένων

Αξίζει να αναφερθεί πως μια άλλη ορολογία για τις μεθόδους πρόβλεψης είναι η μάθηση με επίβλεψη (supervised learning), αφού χρειάζεται να εκπαιδευτεί το μοντέλο με βάση κάποια προϋπάρχουσα γνώση. Για κάποιο ιατρικό πρόβλημα π.χ., χρειάζεται ένα σύνολο δεδομένων για την εκπαίδευση (training set) που να υπάρχει ο χαρακτηρισμός *Άρρωστος ή Όχι άρρωστος* ώστε ο αλγόριθμος να φτιάξει ένα μοντέλο με βάση αυτό. Αντιθέτως, οι περιγραφικές εργασίες χαρακτηρίζονται και ως μάθηση χωρίς επίβλεψη (unsupervised learning), καθώς δεν υπάρχει προηγούμενη γνώση ή

πληροφορία και η περιγραφή που γίνεται στηρίζεται στο συγκεκριμένο σύνολο δεδομένων. Στη συνέχεια θα γίνει πιο αναλυτική περιγραφή του προβλήματος της ομαδοποίησης (clustering) και της κατηγοριοποίησης (classification) που χρησιμοποιούνται στην παρούσα διπλωματική.

1.2 Ομαδοποίηση (clustering)

Η ομαδοποίηση είναι η διαδικασία διαχωρισμού ενός συνόλου δεδομένων σε υποσύνολα που τα στοιχεία τους ομοιάζουν με βάση ένα προεπιλεγμένο κριτήριο. Αποτελεί μία από τις κύριες μεθόδους της Εξόρυξης Δεδομένων και ανήκει στην κατηγορία μάθησης χωρίς επίβλεψη, καθώς κατά τη διαδικασία δε χρησιμοποιείται κάποια γνώση για τα δεδομένα από πριν. Η διαδικασία, δεδομένου ενός κριτηρίου, ανακαλύπτει τις ομάδες των οποίων τα στοιχεία έχουν όμοια συμπεριφορά (Εικόνα 1.3), πετυχαίνοντας να εξάγει συμπεράσματα/πρότυπα για τα δεδομένα που δεν ήταν εκ των προτέρων προφανή και γνωστά και γι' αυτό χαρακτηρίζεται ως μία περιγραφική μέθοδος. Σε μια εφαρμογή, τα στοιχεία μιας ομάδας αναμένεται να έχουν ίδια αξία για το πρόβλημα.



Εικόνα 1.3 – Παράδειγμα ομαδοποίησης

Ένας αλγόριθμος ομαδοποίησης δέχεται ως είσοδο ένα σύνολο από στιγμιότυπα δεδομένων που κάθε ένα έχει τα δικά του γνωρίσματα, και το αναθέτει στην

κατάλληλη ομάδα. Όλες μαζί οι ομάδες που σχηματίζονται συνιστούν μια ομαδοποίηση. Οι ομαδοποιήσεις κατατάσσονται σε κατηγορίες με κριτήριο το πως αναθέτουν τα δεδομένα στις ομάδες. Επιπρόσθετα, διάκριση μπορεί να γίνει σύμφωνα με τον τύπο των ομάδων. Η διάκριση αυτή απαντά στο ερώτημα "Τι συνιστά μια ομάδα", χρησιμοποιώντας διάφορα κριτήρια μέτρησης της ομοιότητας των δεδομένων. Η επιλογή του τύπου της ομαδοποίησης και των ομάδων, έχει οδηγήσει σε ποικιλία αλγορίθμων στην ανάλυση ομάδων, επιτρέποντας τους αναλυτές να επιλέξουν την κατάλληλη ανάλογα με το πρόβλημα που καλούνται να επιλύσουν.

Μερικοί από τους πιο γνωστούς αλγόριθμους ομαδοποίησης είναι ο K-Means [4], ο DBSCAN [5] και η Συσσωρευτική Ιεραρχική Συσταδοποίηση [6].

1.2.1 Τύποι ομαδοποίησης

Σε ό,τι αφορά τον τύπο ομαδοποίησης, μια πρώτη διάκριση μπορεί να γίνει σε ανάθεση σημείων (hard clustering) και ιεραρχική (soft clustering). Στην πρώτη κατηγορία ένα δεδομένο είτε ανατίθεται σε μία ομάδα είτε όχι. Στη δεύτερη τα δεδομένα ανήκουν ποσοστιαία στις κατηγορίες. Συσχετίζονται δηλαδή με όλες τις κατηγορίες κατά έναν υπολογιζόμενο βαθμό. Ο δεύτερος αυτός τρόπος ονομάζεται και ασαφής ομαδοποίηση (fuzzy clustering) [7].

Ακόμα μια διάκριση διαχωρίζει τις ομαδοποιήσεις σε 5 είδη:

- ➔ Αυστηρή διαμέριση, όπου κάθε δεδομένο μπορεί να ανήκει αποκλειστικά σε μία κατηγορία,
- ➔ Αυστηρή διαμέριση με ακραίες τιμές, όπου τα δεδομένα μπορούν να ανήκουν αυστηρά σε μία κατηγορία, είτε να μην ανήκουν σε καμία, ως ακραίες τιμές,
- ➔ Εναλλακτική ή Επικαλυπτόμενη, όπου ένα δεδομένο μπορεί να ανήκει σε περισσότερες από μία κατηγορίες,
- ➔ Ιεραρχική, όπου ένα δεδομένο ανατίθεται σε μία ομάδα και ταυτόχρονα ανήκει και στη γονική της ομάδα,
- ➔ Υποχώρου, όπου κάθε δεδομένο ανήκει αποκλειστικά σε μία ομάδα και οι ομάδες ομαδοποιούνται σε υποχώρους.

1.2.2 Τύποι ομάδων

Σε επόμενη φάση, διακρίνουμε τις ομάδες που σχηματίζονται ως εξής:

- Καλά διαχωρισμένες, όπου σύμφωνα με ένα επιλεγόμενο κριτήριο ομοιότητας, τα δεδομένα που ανήκουν σε μια ομάδα είναι πιο όμοια με όλα τα άλλα δεδομένα της ομάδας από ότι με τα δεδομένα που ανήκουν στις άλλες ομάδες. Η μέτρηση της ομοιότητας γίνεται μέσω ενός κριτιρίου απόστασης σύμφωνα με το οποίο τα δεδομένα μιας ομάδας απέχουν λιγότερο μεταξύ τους από οτι με κάθε δεδομένο μιας άλλης ομάδας.
- Βασισμένες σε πρότυπο, όπου τα δεδομένα μιας ομάδας βρίσκονται πιο κοντά στο πρότυπο αυτής της ομάδας, παρά στο πρότυπο οποιασδήποτε άλλης ομάδας. Ένα παράδειγμα τέτοιου προτύπου είναι τα κέντρα των ομάδων, όπου ανάλογα με το ποιο κέντρο βρίσκεται πιο κοντά στο δεδομένο, με κριτήριο ένα μέτρο απόστασης, η ανάθεση γίνεται στην αντίστοιχη ομάδα.
- Βασισμένες σε γράφο. Εδώ προϋποτίθεται ότι τα δεδομένα είναι στη μορφή γράφου ως κόμβοι που είτε συνδέονται είτε όχι μεταξύ τους με ακμές. Μια ομάδα μπορεί να θεωρηθεί ως συνιστώσα που είναι συνδεδεμένη και δε συνδέεται με τους κόμβους άλλης ομάδας. Άλλες προσεγγίσεις ομάδας σε αυτή την κατηγορία είναι δυνατές με άλλα κριτήρια, όπως η ομάδα βασισμένη στη γειτνίαση, όπου ως ομάδα θεωρούνται κόμβοι που βρίσκονται σε μια ορισμένη απόσταση μεταξύ τους.
- Βασισμένες στην πυκνότητα, όπου κριτήριο για την κατάταξη δεδομένων σε μία ομάδα είναι η πυκνότητα της περιοχής που βρίσκεται. Π.χ. δεδομένα που βρίσκονται σε μια περιοχή υψηλής πυκνότητας βρίσκονται στην ίδια ομάδα, ενώ άλλα που βρίσκονται σε περιοχή χαμηλότερης πυκνότητας σε άλλη.
- Ομάδες με κοινές ιδιότητες (Εννοιολογική ομαδοποίηση). Σε αυτή την κατηγορία μια ομάδα μπορεί να οριστεί όπως ορίστηκε στις παραπάνω κατηγορίες, εμπλουτίζοντας το κριτήριο κατάταξης στην ομάδα με κάποιες κοινές ιδιότητες (πρότυπα), προκειμένου να ανακαλυφθούν ομάδες με πιο σύνθετους σχηματισμούς στο χώρο .

1.3 Κατηγοριοποίηση

Ένα από τα προβλήματα στο πεδίο της εξόρυξης δεδομένων, είναι η αντιστοίχηση πρωτοεμφανιζόμενων παρατηρήσεων σε ετικέτες (κατηγορίες), από ένα προκαθορισμένο σύνολο ετικετών. Είναι ένα πρόβλημα που ανακύπτει στην προσπάθεια δημιουργίας νοημόνων οντοτήτων με δυνατότητα όχι απλά να ξεχωρίζουν, αλλά να αναγνωρίζουν φαινόμενα, καταστάσεις, πράγματα κ.λ.π. και να περιγράφουν τη δομή ενός είδους-κατηγορίας, στα πλαίσια ανάπτυξης συστημάτων που μπορούν αποτελεσματικά να υποστηρίζουν αποφάσεις αλλά και γενικότερα στα πλαίσια της επεξήγησης φαινομένων και οντοτήτων.

Η μέθοδος - εργαλείο στην προσπάθεια επίλυσης του προβλήματος είναι η κατηγοριοποίηση. Φυσικά, για να αναγνωριστεί μία οντότητα σε μία κατηγορία ή να περιγραφεί μία κατηγορία, προϋποτίθεται γνώση για αυτήν και συγκεκριμένα για το πως προκύπτει ότι κάποιο αντικείμενο ανήκει σε μία κατηγορία. Μόλις αποκτηθεί, έχουμε μια περιγραφή και μπορούμε πλέον να κάνουμε προβλέψεις.

Ας δούμε όμως τι νοείται γνώση στην κατηγοριοποίηση. Η διαδικασία είναι αυτοδύναμη, αντλώντας όλη τη γνώση από τα δεδομένα. Δεν υπάρχει κάποιος ειδικός, ο οποίος τροφοδοτεί τη διαδικασία με τις θεωρήσεις που είναι απαραίτητες για τη διάκριση των κατηγοριών. Η γνώση χτίζεται - εξορύσσεται απευθείας και αποκλειστικά από τα δεδομένα που τροφοδοτούνται σε γενικευμένες συστηματικές διαδικασίες μάθησης.

Η βάση της γνώσης είναι ένα σύνολο δεδομένων, το οποίο περιλαμβάνει εγγραφές με γνωστές ετικέτες. Ένας, επιλεγμένος κάθε φορά, κατάλληλος αλγόριθμος, δουλεύει με είσοδο αυτά τα δεδομένα αποδίδοντας ένα σύστημα (μοντέλο) που είναι ικανό να κατηγοριοποιεί καινούρια δεδομένα του τύπου που περιέχονται στο σύνολο δεδομένων για το εκάστοτε πρόβλημα κατηγοριοποίησης.

Το σύστημα που προκύπτει ονομάζεται μοντέλο κατηγοριοποίησης και η διαδικασία δημιουργίας του εκπαίδευση. Ο αλγόριθμος που επιδρά στο σύνολο δεδομένων ονομάζεται αλγόριθμος μάθησης και υπάρχει μια μεγάλη ποικιλία τέτοιων. Είναι στην κρίση του προγραμματιστή να επιλέξει το σωστό αλγόριθμο, που θα του δώσει ένα αποτελεσματικό μοντέλο, αναλόγως του εκάστοτε προβλήματος.

Όπως προαναφέρθηκε, βασικό και αναπόσπαστο συστατικό της εκπαίδευσης είναι το σύνολο δεδομένων. Η κάθε εγγραφή που περιλαμβάνει είναι ένα στιγμιότυπο

χαρακτηριστικής περίπτωσης στο πρόβλημα. Είναι επίσης ένα σύνολο διακριτών χαρακτηριστικών που θεωρείται ότι πρέπει να παίξουν ρόλο στην κατηγοριοποίηση. Κάθε χαρακτηριστικό λοιπόν είναι μια παράμετρος που παίζει κάποιο ρόλο στη διαδικασία απόφασης, ενώ όλα τα χαρακτηριστικά είναι ό,τι πρέπει να ληφθεί υπόψιν. Για παράδειγμα, μια εγγραφή ενός συνόλου δεδομένων για μία ασθένεια, θα μπορούσε να έχει ως χαρακτηριστικά τα αποτελέσματα μιας εξέτασης αίματος κάποιου, τα οποία είναι δείκτες της ασθένειας, ενώ ένα παράδειγμα χαρακτηριστικού θα μπορούσε να είναι ο δείκτης της χοληστερίνης.

Επίσης, για τη διαδικασία μάθησης, οι εγγραφές του συνόλου περιλαμβάνουν ακόμη ένα χαρακτηριστικό που είναι η ετικέτα. Είναι η κατηγορία στην οποία ανήκει το δεδομένο. Το σύνολο δεδομένων είναι λοιπόν μια συλλογή γνωστών και χαρακτηριστικών περιπτώσεων του εκάστοτε προβλήματος. Σε συνέχεια του προηγούμενου παραδείγματος, η ετικέτα κάθε εγγραφής θα περιελάμβανε την ασθένεια που διαγιγνώσκεται, δεδομένων των συγκεκριμένων τιμών δεικτών των εξετάσεων αίματος.

Είναι καθοριστικό το σύνολο αυτό να είναι η κατάλληλη είσοδος για τον αλγόριθμο μάθησης. Αυτό μεταφράζεται σε πληρότητα παρατηρήσεων σε ό,τι αφορά τις περιπτώσεις που θέλουμε να διακρίνουμε. Σύμφωνα με το προηγούμενο παράδειγμα, η πληρότητα δε μεταφράζεται στην ύπαρξη δεδομένων με κάθε μία παρατηρούμενη τιμή δείκτη. Αυτό είναι τόσο αδύνατο, όσο και δεν ταιριάζει στη διαδικασία μάθησης της κατηγοριοποίησης. Η μάθηση αυτή δεν είναι φωτογραφική των περιπτώσεων που υπάρχουν στα δεδομένα, αλλά είναι μια διαδικασία που εξάγει γενικευμένα συμπεράσματα από τα δεδομένα για τις περιπτώσεις ενδιαφέροντος. Αυτό σημαίνει ότι πρέπει να υπάρχουν δεδομένα που θα επιτρέψουν στον αλγόριθμο να μάθει τις περιπτώσεις ενδιαφέροντος και όχι ότι πρέπει να υπάρχουν δεδομένα τα οποία αναμένεται να παρατηρηθούν καθεαυτά στο μέλλον.

Η πεμπτουσία της δημιουργίας ενός μοντέλου είναι η ικανότητα πρόβλεψης κατηγορίας σε νέες παρατηρήσεις που δεν έχουν ιδωθεί ακόμα. Απαιτείται γνώση και εκπαίδευση από πριν για μία κατηγορία, αλλιώς ένα μοντέλο πολύ απλά δεν μπορεί να κάνει αυτή την πρόβλεψη. Γι’ αυτό και η κατηγοριοποίηση συγκαταλέγεται στις τεχνικές μάθησης με επίβλεψη.

Όπως σε όλα τα υπολογιστικά συστήματα, ένα σύστημα δε θα κάνει ποτέ αυτό που δεν ξέρει. Έτσι και ένα μοντέλο, ξέρει πως να ξεχωρίζει μόνο τις κατηγορίες που

ξέρει. Αν δεν υπάρχει γνώση για κάποια κατηγορία, τα δεδομένα αυτής δε μπορούν να προβλεφθούν σωστά. Μπορούν όμως να προβλεφθούν σωστά τα νέα δεδομένα γνωστών κατηγοριών πολύ απλά γιατί στο τέλος ο αλγόριθμος δε γνωρίζει ένα-ένα τα δεδομένα, αλλά διαθέτει γενικευμένη γνώση από αυτά για τις κατηγορίες, πράγμα που οδηγεί στο ότι τελικά το μοντέλο διαθέτει ευφυΐα για να το κάνει.

Προκειμένου να επιτευχθεί γενικευμένη γνώση για μια κατηγορία, είναι σημαντικό να υπάρχουν στο σύνολο δεδομένων στιγμιότυπα όλων των διαφορετικών πτυχών της κατηγορίας. Αυτά θα επιτρέψουν στον αλγόριθμο εκπαίδευσης να εξάγει το εύρος της κατηγορίας. Έστω ότι έχουμε ένα σετ δεδομένων για μία ασθένεια που χαρακτηρίζεται από υψηλό πυρετό. Αν για τους αρρώστους έχουμε δεδομένα μόνο με θερμοκρασία σώματος 41 και για τους υγιείς θερμοκρασία 37, εσφαλμένα θα είχαμε ένα μοντέλο που περιγράφει τη θερμοκρασία σώματος των ασθενών στο 41.

Άρα το σετ δεδομένων πρέπει να περιλαμβάνει περιπτώσεις για ένα εύρος θερμοκρασιών σώματος ασθενών και υγιών, προκειμένου το μοντέλο να ξεχωρίζει με γενικευμένο τρόπο ασθενείς και υγιείς, αλλιώς υπάρχει σίγουρα αβεβαιότητα, αν όχι σίγουρο σφάλμα της σωστής πρόγνωσης. Σίγουρα όμως η περιγραφή της κατηγορίας από το μοντέλο είναι ελλιπής. Το φαινόμενο αυτό ονομάζεται υποπροσαρμογή μοντέλου και οδηγεί σε μοντέλα χωρίς ικανότητα γενίκευσης.

Ένα άλλο φαινόμενο, είναι αυτό της υπερπροσαρμογής μοντέλου. Σύμφωνα με αυτό, κατά τη διαδικασία εκπαίδευσης, αναπτύσσεται ένα μοντέλο που είναι υπερβολικά προσαρμοσμένο στα δεδομένα, λειτουργεί δηλαδή πιο “φωτογραφικά” των περιπτώσεων του συνόλου εκπαίδευσης. Πρώτον, αυτό κάνει το μοντέλο επιρρεπές στο λάθος λόγω θορύβου, στην περίπτωση που κάποιο δεδομένο από αυτά που προσαρμόστηκε είναι λανθασμένο. Έστω ότι στο σετ δεδομένων που προαναφέρθηκε, υπάρχει ένα δεδομένο ασθενούς με θερμοκρασία 36.6, λόγω θορύβου. Αν το μοντέλο μας προσαρμοστεί τέλεια στα δεδομένα, αυτό θα οδηγήσει σε υγιείς ανθρώπους που θα προβλέπονται ως ασθενείς, καθώς το μοντέλο θα έχει προσαρμοστεί στο 36.6 ως θερμοκρασία ασθενών. Δεύτερον, στο ίδιο παράδειγμα, ένα τέλεια προσαρμοσμένο μοντέλο στα δεδομένα δεν έχει ικανότητα γενίκευσης, καθώς θα λειτουργεί εξειδικευμένα για τις τιμές που εμφανίστηκαν στο σύνολο εκπαίδευσης, λαθεύοντας στις πραγματικές περιπτώσεις ασθενών.

Ακόμα όμως και αν έχουμε εκπαιδεύσει ένα μοντέλο, αυτό δεν είναι αρκετό για να προχωρήσουμε σε προβλέψεις τις οποίες θα εμπιστευόμαστε. Πρέπει να ξέρουμε

σίγουρα ότι αυτό το μοντέλο έχει εκπαιδευτεί και είναι ικανό να προβλέπει σωστά τις κατηγορίες ενδιαφέροντος. Για το σκοπό αυτό, τη φάση της εκπαίδευσης, ακολουθεί η φάση ελέγχου του μοντέλου, κατά την οποία γίνονται προβλέψεις από το μοντέλο οι οποίες ακολουθούνται από διασταύρωση και εξάγονται μετρήσεις αποτελεσματικότητας.

Συνοπτικά, στην αρχή της κατηγοριοποίησης, ένα σύνολο δεδομένων χωρίζεται σε δύο κύρια υποσύνολα. Το σύνολο εκπαίδευσης (training set) και το σύνολο ελέγχου (test set). Αυτά χρησιμοποιούνται αντίστοιχα για τη φάση εκπαίδευσης και για τη φάση ελέγχου. Εάν το μοντέλο που προέκυψε από τη φάση εκπαίδευσης αξιολογείται αξιόπιστο σύμφωνα με τις μετρήσεις που εξάγονται από τη φάση ελέγχου, τότε μπορεί να χρησιμοποιηθεί για κατηγοριοποίηση νέων δεδομένων (Εικόνα 1.4). Η διαδικασία εκπαίδευσης-ελέγχου παρουσιάζεται στο Εικόνα 1.5.

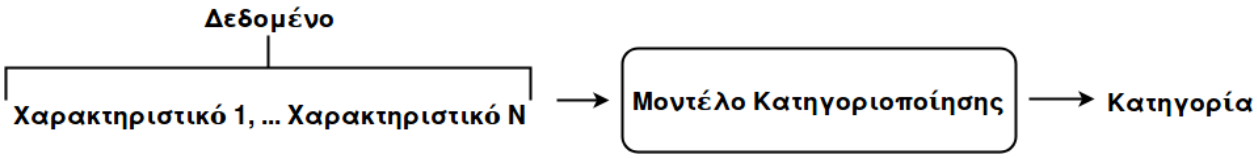
1.3.1 Αλγόριθμοι κατηγοριοποίησης

Στη βιβλιογραφία έχουν παρουσιαστεί διάφορες προσεγγίσεις που αφορούν αλγορίθμους εξόρυξης δεδομένων. Αυτοί διαφέρουν τόσο στη διαδικασία εκπαίδευσης, όσο και στο μοντέλο που παράγουν. Το δεύτερο αφορά κυρίως την περιγραφική χρήση των μοντέλων, καθώς στην πρόγνωση, το μοντέλο λειτουργεί ως “μαύρο κουτί”. Οι κύριες προσεγγίσεις είναι οι ακόλουθες:

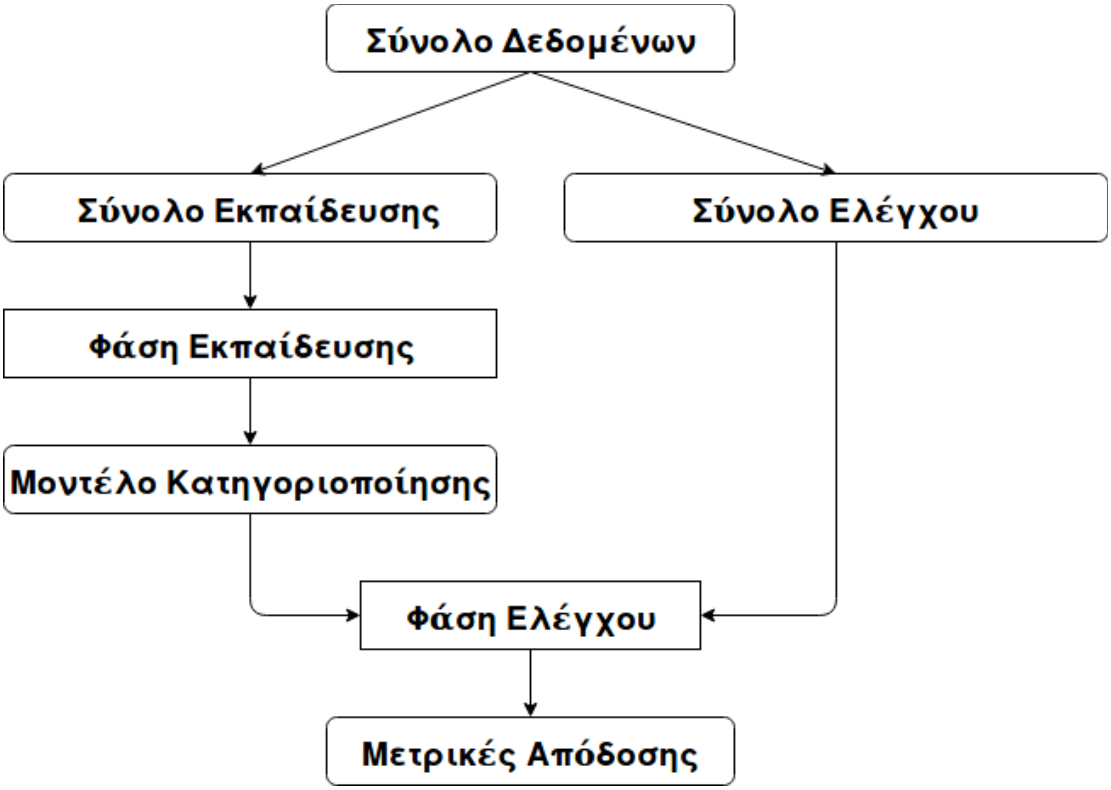
1.3.1.1 Κατηγοριοποιητές του Bayes

Οι κατηγοριοποιητές του Bayes [8] είναι από τις πιο δημοφιλείς μεθόδους κατηγοριοποίησης. Υπολογίζουν για κάθε δεδομένο την πιθανότητα να βρίσκεται σε κάποια κατηγορία, δεδομένων των τιμών των χαρακτηριστικών του. Η κατηγορία με τη μεγαλύτερη πιθανότητα είναι εκείνη που ανατίθεται στο δεδομένο. Κατά τη διαδικασία μάθησης, υπολογίζονται οι πιθανότητες για κάθε κατηγορία και αυτές κάθε τιμής των χαρακτηριστικών (για συνεχείς τιμές χαρακτηριστικών γίνεται διακριτοποίηση ή υπόθεση μιας κατανομής).

Κατά την ταξινόμηση των νέων δεδομένων υπολογίζονται οι πιθανότητες της κάθε κατηγορίας δεδομένων των εκάστοτε τιμών των χαρακτηριστικών και η μεγαλύτερη από αυτές δείχνει την κατηγορία του νέου δεδομένου. Η όλη διαδικασία υποθέτει τη στατιστική ανεξαρτησία των χαρακτηριστικών και των κατηγοριών.



Εικόνα 1.4 – Διαδικασία κατηγοριοποίησης



Εικόνα 1.5 – Διαδικασία εκπαίδευσης και ελέγχου

1.3.1.2 Δέντρα Απόφασης

Οι κατηγοριοποιητές με δέντρα απόφασης [9] χτίζουν ένα μοντέλο αποφάσεων σύμφωνα με τα διαθέσιμα δεδομένα. Κάθε κόμβος του δέντρου είναι μία συνθήκη για την τιμή κάποιου χαρακτηριστικού των δεδομένων και κάθε φύλλο του δέντρου είναι κάποια από τις κατηγορίες. Στόχος είναι μέσω ενός αριθμού διαδοχικών αποφάσεων

να εξάγεται η απόφαση για την κατηγορία κάθε νέου δεδομένου. Στις περιπτώσεις συνεχών χαρακτηριστικών, εφαρμόζεται διαχωρισμός τους με κάποιο κριτήριο.

Στην αρχή, όπως και για κάθε δημιουργία νέου ενδιάμεσου κόμβου, τα χαρακτηριστικά ιεραρχούνται σύμφωνα με κάποιο κριτήριο και το καλύτερο επιλέγεται για να δημιουργήσει τη νέα διακλάδωση. Η διαδικασία πρόβλεψης γίνεται με διάσχιση του δέντρου στο δρόμο που δείχνουν τα χαρακτηριστικά του νέου δεδομένου, καταλήγοντας στην κατηγορία. Οι κατηγοριοποιητές αυτοί αναλύονται εκτενώς στο κεφάλαιο 2.

1.3.1.3 Κατηγοριοποίηση Μέσω Ομαδοποίησης

Είναι μια χρήση της τεχνικής της ομαδοποίησης για την κατηγοριοποίηση. Η λογική είναι η αντιστοίχηση των κατηγοριών σε ομάδες, των οποίων κατά την εκπαίδευση αναζητείται η μορφή. Δοθέντος ενός αριθμού κατηγοριών, γίνεται μια αρχικοποίηση ίδιου ή μεγαλύτερου αριθμού προτύπων. Κατόπιν, διαδοχικά ανατίθενται τα δεδομένα στις κατηγορίες σύμφωνα με ένα κριτήριο ομοιότητας με το πρότυπο και επανυπολογίζονται τα πρότυπα. Κατά τη διαδικασία πρόβλεψης, υπάρχουν ως δεδομένα τα υπολογισμένα πρότυπα και τα δεδομένα ανατίθενται στο πρότυπο που είναι πιο όμοια. Το πρότυπο είναι ουσιαστικά ο δείκτης της προβλεπόμενης κατηγορίας.

1.3.1.4 Τεχνητά Νευρωνικά Δίκτυα

Τα τεχνητά νευρωνικά δίκτυα [10] είναι μια απόπειρα προσέγγισης της διαδικασίας της μάθησης του ανθρώπου. Ο απλός νευρώνας Perceptron [11] είναι ο γνωστότερος αλγόριθμος μάθησης που αποτελεί το απλούστερο επίπεδο τεχνητού νευρωνικού δικτύου (ΤΝΔ). Ένα ΤΝΔ δύναται να αποτελείται από πολλά επίπεδα νευρώνων κάποιας διαδικασίας μάθησης [12], ανάλογα με τον επιλεγόμενο αλγόριθμο μάθησης και την επιλεγόμενη τοπολογία.

1.3.1.5 Κατηγοριοποιητές Ομάδων

Στην περίπτωση των κατηγοριοποιητών ομάδων [13] υπάρχει μια συλλογή από κατηγοριοποιητές οι οποίοι αποφασίζουν συμμετοχικά για την τιμή της κατηγορίας ενός νέου δεδομένου. Κατά τη διαδικασία ανάπτυξης των κατηγοριοποιητών, επιλέγεται ο αλγόριθμος μάθησης και αναπτύσσονται τόσα διαφορετικά μοντέλα όσο το επιλεγόμενο μέγεθος της ομάδας. Κατά τη διαδικασία πρόβλεψης, αυτά τα μοντέλα

κατηγοριοποιούν ξεχωριστά το κάθε δεδομένο και η πλειοψηφούσα κατηγορία για αυτό είναι η έξοδος-κατηγορία για αυτό το δεδομένο. Οι μέθοδος των κατηγοριοποιητών ομάδων αναλύεται εκτενώς στο κεφάλαιο 2.

1.3.1.6 Μηχανές διανυσμάτων υποστήριξης

Οι μηχανές διανυσμάτων υποστήριξης (support vector machines) [14] είναι ένας αλγόριθμος που προσπαθεί να ανακαλύψει τον καλύτερο δυνατό διαχωρισμό ανάμεσα στις κατηγορίες, μέσω της εύρεσης ενός υπερ-επιπέδου που μεγιστοποιεί την απόσταση μεταξύ κάποιων κατηγοριών. Τα ακραία σημεία των κατηγοριών χρησιμοποιούνται ως διανύσματα απόστασης. Το πρόβλημα ανάγεται στην εύρεση εκείνου του υπερ-επιπέδου που απέχει περισσότερο από αυτά τα σημεία, το οποίο και χρησιμοποιείται στη συνέχεια για να κατηγοριοποιήσει τα δεδομένα αναλόγως της θέσης τους σε σχέση με το υπερ-επίπεδο.

1.3.1.7 Κατηγοριοποιητές κανόνων

Στους κατηγοριοποιητές κανόνων [15] κατά τη διαδικασία εκπαίδευσης δημιουργείται ένα σύνολο κανόνων που αποτελείται από συνθήκες οι οποίες όταν ικανοποιούνται συνεπάγονται μια κατηγορία. Η εξαγωγή των κανόνων δύναται να γίνει και με διάβασμα ενός άλλου μοντέλου, όπως ένα δέντρο απόφασης. Οι κανόνες συνήθως είναι σύνθετες συνθήκες – κανονικές εκφράσεις – που έχουν χτιστεί βήμα-βήμα δημιουργώντας και συνενώνοντας απλές συνθήκες κατάλληλα.

Στην κατηγοριοποίηση, τα χαρακτηριστικά ενός δεδομένου ελέγχονται και αναλόγως ενεργοποιούν ορισμένους κανόνες-συνθήκες. Αν κάποιο νέο δεδομένο έχει ενεργοποιήσει πολλαπλούς κανόνες, γίνεται ιεράρχηση των κανόνων με κάποια κριτήρια. Αν δεν έχει ενεργοποιηθεί κανένας κανόνας, εφαρμόζεται ένας προεπιλεγμένος κανόνας.

1.4 Αντικείμενο διπλωματικής

Στην παρούσα διπλωματική εργασία αναπτύχθηκαν δύο νέοι αλγόριθμοι εξόρυξης δεδομένων, στο πρόβλημα της κατηγοριοποίησης, δηλαδή με στόχο την επιτυχή ανάπτυξη ενός μοντέλου, μέσω της διαδικασίας της εκπαίδευσης, για την κατηγοριοποίηση δεδομένων με άγνωστη κατηγορία.

Ο πρώτος αλγόριθμος είναι αλγόριθμος κατηγοριοποίησης μέσω ομαδοποίησης (classification via clustering), χρησιμοποιώντας μια γνωστή τεχνική ομαδοποίησης βασισμένης σε πρότυπο (K - Means). Επειδή η ομαδοποίηση είναι μη εποπτευόμενη μάθηση ενώ η κατηγοριοποίηση είναι εποπτευόμενη, δηλαδή υπάρχει ήδη κάποια γνώση για της κατηγορίες, η υπάρχουσα γνώση χρησιμοποιείται στον καθορισμό περιορισμών και βαρών σε κάθε χαρακτηριστικό. Τον αλγόριθμο αυτόν τον ονομάσαμε Constrained K – Means Classification.

Πολλές φορές σε πραγματικές εφαρμογές συναντάμε σύνολα δεδομένων που έχουν μη ισορροπημένη κατανομή των κατηγοριών, δηλαδή η μία κατηγορία να είναι πολύ μικρότερη από κάποια άλλη. Αυτό συμβαίνει π.χ. σε ιατρικά δεδομένα, όπου μια μικρή μειοψηφία μπορεί να διαγνωστούν με κάποια αρρώστια. Μπορούμε να καταλάβουμε πως είναι πολύ σημαντικό ο αλγόριθμος να είναι ακριβής στην μικρή κατηγορία και όχι απλά να έχει καλό ποσοστό ακρίβειας συνολικά. Ο αλγόριθμος κατηγοριοποίησης μέσω ομαδοποίησης αναπτύχθηκε έχοντας στόχο ακριβώς αυτό. Με τον καθορισμό των περιορισμών στο πρότυπο που δημιουργείται, αλλά και των βαρών για κάθε χαρακτηριστικό – διάσταση του προβλήματος, στοχεύει στην καλύτερη ευαισθησία και ακρίβεια κυρίως των μικρών κατηγοριών, όχι γενικά σε καλό ποσοστό επιτυχημένων κατηγοριοποιήσεων.

Γι’ αυτόν το λόγο ο αλγόριθμος εφαρμόστηκε και σε ένα ιατρικό πρόβλημα, αξιολογώντας αυτά ακριβώς τα χαρακτηριστικά του [16]. Το ιατρικό πρόβλημα είχε να κάνει με την διάγνωση και τον καθορισμό της σοβαρότητας της ηπατικής ίνωσης. Το σύνολο δεδομένων είναι ένα σύνολο από 8 εικόνες βιοψίας του ήπατος, παίρνοντας τις RGB τιμές κάθε εικονοστοιχείου (pixel). Στόχος είναι η κατηγοριοποίηση των εικονοστοιχείων σε: φόντο της εικόνας, ηπατικό ιστό, κολλαγόνα. Έτσι, εξάγοντας το ποσοστό των κολλαγόνων έναντι του ηπατικού ιστού μπορεί να γίνει η διάγνωση και η εκτίμηση της σοβαρότητας της ασθένειας.

Ο δεύτερος αλγόριθμος που αναπτύχθηκε, ο Stochastic Forest, είναι αλγόριθμος κατηγοριοποίησης με την μέθοδο των ομάδων. Είναι βασισμένος στα δέντρα απόφασης και την λογική ενός από τους πιο γνωστούς αλγορίθμους – κατηγοριοποιητή ομάδων, του αλγορίθμου τυχαίων δασών Random Forest [17]. Η βασική ιδέα του αλγορίθμου είναι η παραγωγή Ν δέντρων απόφασης με διαφορετικό τρόπο, με βάση το σύνολο εκπαίδευσης. Η κατηγοριοποίηση των μελλοντικών δεδομένων γίνεται ξεχωριστά από το κάθε δέντρο και στο τέλος η κατηγορία στην

οποία το κάθε στιγμιότυπο έχει κατηγοριοποιηθεί από τα περισσότερα δέντρα είναι η τελική του κατηγορία. Η διαφορά με τον Random Forest και άλλους αντίστοιχους αλγορίθμους, βρίσκεται στον τρόπο παραγωγής των Ν διαφορετικών δέντρων.

Η αξιολόγηση των αλγορίθμων έγινε σε αρκετά σύνολα δεδομένων από το UCI machine learning repository [18], που είναι το πιο διαδεδομένο για εμπειρική ανάλυση και αξιολόγηση αλγορίθμων εξόρυξης δεδομένων. Για το σκοπό αυτό χρησιμοποιήθηκαν και τα πιο γνωστά και τα πιο χρησιμοποιημένα σύνολα δεδομένων.

1.5 Οργάνωση κειμένου

Στο Κεφάλαιο 2 θα αναλυθούν βασικοί αλγόριθμοι και τεχνικές τις οποίες χρησιμοποιούμε και είναι αναγκαία η κατανόησή τους από τον αναγνώστη πριν προχωρήσουμε στους αλγόριθμους που αναπτύξαμε. Εργασίες σχετικές με το αντικείμενο της διπλωματικής και με τους αλγορίθμους που αναπτύχθηκαν παρουσιάζονται στο Κεφάλαιο 3. Στο Κεφάλαιο 4 περιγράφεται ο αλγόριθμος κατηγοριοποίησης μέσω ομαδοποίησης Constrained K – Means Classification που αναπτύξαμε. Στο Κεφάλαιο 5 παρουσιάζεται ο αλγόριθμος – κατηγοριοποιητής ομάδας Stochastic Forest που αναπτύξαμε. Στο Κεφάλαιο 6 παρουσιάζονται και αναλύονται η αξιολόγηση και τα αποτελέσματα των αλγορίθμων. Ακόμα παρουσιάζεται και η εφαρμογή σε ιατρικό πρόβλημα του αλγόριθμου Constrained K – Means Classification. Τέλος, στο Κεφάλαιο 7 γίνεται η σύνοψη των αποτελεσμάτων της διπλωματικής και παρουσιάζονται τα συμπεράσματα. Ακόμα, αναφέρονται κάποιες μελλοντικές επεκτάσεις.

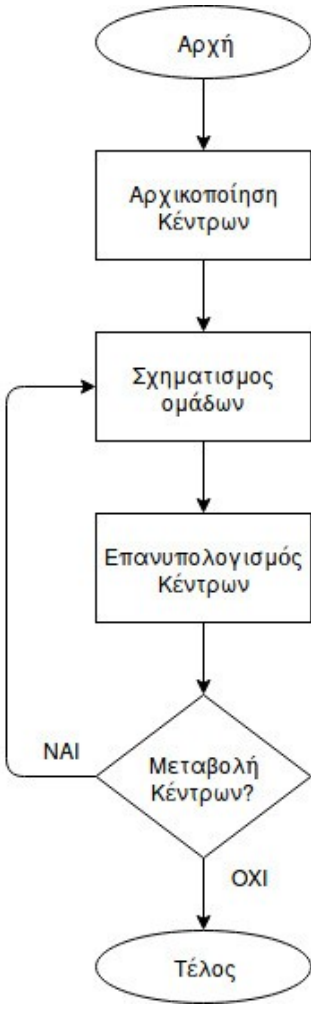
2 *Θεωρητικό υπόβαθρο*

Σε αυτό το κεφάλαιο θα αναφερθούμε πιο αναλυτικά σε κάποιες μεθοδολογίες και αλγορίθμους που χρησιμοποιήθηκαν στην εκπόνηση της διπλωματικής εργασίας και είναι αναγκαία η κατανόησή τους από τον αναγνώστη πριν την παρουσίαση και ανάλυση των αλγορίθμων που αναπτύξαμε και της αξιολόγησής τους.

Στην ενότητα 2.1 θα αναφερθούμε στον αλγόριθμο K – Means, που είναι αλγόριθμος ομαδοποίησης μέσω προτύπου και τον χρησιμοποιούμε στον αλγόριθμο κατηγοριοποίησης μέσω ομαδοποίησης που αναπτύξαμε. Στην συνέχεια, στις ενότητες 2.2 και 2.3 θα αναλύσουμε κάποια βασικά ζητήματα για την κατηγοριοποίηση με δέντρα απόφασης και την κατηγοριοποίηση με τη μέθοδο ομάδων. Σε αυτή την ενότητα θα αναφερθούμε και στα τυχαία δάση (random forests) που χρησιμοποιούν δέντρα απόφασης για τη δημιουργία της ομάδας των κατηγοριοποιητών. Η ενότητα 2.4 επικεντρώνεται στις βασικές τεχνικές αξιολόγησης και επικύρωσης των αλγορίθμων κατηγοριοποίησης [19], όπως το k – fold cross - validation και το leave – one - out cross – validation. Στην ενότητα 2.5 επεξηγείται η μήτρα σύγχυσης και η χρησιμότητά της στην εξαγωγή συμπερασμάτων από τα αποτελέσματα της αξιολόγησης του αλγορίθμου. Τέλος, στην ενότητα 2.6 αναφερόμαστε στα βασικά μέτρα απόδοσης που χρησιμοποιούνται για την αξιολόγηση των αλγορίθμων και χρησιμοποιήθηκαν στην παρούσα διπλωματική.

2.1 Ο αλγόριθμος K - Means

Ο αλγόριθμος Κ-μέσων ανήκει στην κατηγορία της ομαδοποίησης υποχώρου, που σημαίνει ότι κάθε δεδομένο ανήκει αποκλειστικά σε μία ομάδα. Οι ομάδες είναι βασισμένες σε πρότυπο, όπου για κάθε ομάδα υπάρχει ένα κέντρο και τα δεδομένα ανατίθενται στην ομάδα της οποίας το κέντρο είναι πιο κοντά. Υπάρχουν Κ τον αριθμό ομάδες, παράμετρος που καθορίζεται από το χρήστη.



Εικόνα 2.1 – Διάγραμμα ροής αλγορίθμου K-Means

Γενικότερα, η διαδικασία περιλαμβάνει τα εξής βήματα (διάγραμμα ροής στην Εικόνα 2.1):

1. Αρχικοποίηση κέντρων

2. Σχηματισμός ομάδων
3. Επανυπολογισμός κέντρων
4. Επανάληψη των βημάτων 2 και 3 έως ότου ο αλγόριθμος συγκλίνει

2.1.1 Αρχικοποίηση κέντρων

Στη βασική έκδοση του αλγορίθμου, η πιο απλή υλοποίηση του πρώτου αυτού βήματος είναι η τυχαία επιλογή των αρχικών κέντρων βάρους για να ξεκινήσει η διαδικασία. Αυτή η τυχαία επιλογή κάνει τον αλγόριθμο ευαίσθητο στην αρχικοποίηση καθώς ενδέχεται να οδηγήσει σε λάθος σχηματισμό των τελικών ομάδων με κάποιες από τις ομάδες να μοιράζονται κάποιο κέντρο βάρους ενώ κάποιες άλλες να διαιρούνται σε περισσότερα κέντρα βάρους.

Προκειμένου να περιοριστεί αυτή η ευαισθησία, εναλλακτικές μέθοδοι αρχικοποίησης έχουν προταθεί. Μια μέθοδος περιλαμβάνει επαναληπτικές εκτελέσεις του αλγορίθμου, με τα τελικά κέντρα βάρους να επιλέγονται ως τα κέντρα βάρους που έχουν το ελάχιστο υπολογιζόμενο σφάλμα. Μια άλλη μέθοδος είναι η εφαρμογή ιεραρχικής ομαδοποίησης. Γίνεται λήψη ενός δείγματος των δεδομένων, εφαρμόζεται ο αλγόριθμος πάνω σε αυτό και τα κέντρα που προκύπτουν επιλέγονται ως αρχικά στον αλγόριθμο για το συνολικό σετ δεδομένων. Μια τρίτη μέθοδος, προσπαθεί να διασπείρει τα αρχικά κέντρα στο σετ δεδομένων. Επιλέγει τυχαία το πρώτο κέντρο σε κάποιο στιγμιότυπο δεδομένων και κατόπιν επιλέγει κάθε επόμενο με κριτήριο τη μέγιστη απόσταση από τα ήδη επιλεγμένα.

2.1.2 Σχηματισμός Ομάδων

Δεδομένων των κέντρων, το επόμενο βήμα είναι η ανάθεση κάθε ενός στιγμιότυπου από το σετ δεδομένων στις ομάδες με κριτήριο την απόστασή του από τα κέντρα. Κάθε δεδομένο ανατίθεται στην ομάδα της οποίας το κέντρο απέχει λιγότερο. Υπάρχουν διάφορα μέτρα υπολογισμού της απόστασης και για κάθε υλοποίηση, ο προγραμματιστής καλείται, αναλόγως το πρόβλημα, να επιλέξει εκείνο που θεωρεί πως θα του προσφέρει την επιθυμητή ομαδοποίηση. Τα πιο γνωστά μέτρα απόστασης είναι η Ευκλείδεια απόσταση [48], η απόσταση Manhattan [59] και η απόσταση Mahalanobis [49, 50].

2.1.3 Επανυπολογισμός Κέντρων

Τα κέντρα βάρους των ομάδων, σύμφωνα με τον αλγόριθμο, προσδιορίζονται μαθηματικά μέσω των δεδομένων που έχουν ανατεθεί στις ομάδες. Όταν λοιπόν ένα δεδομένο ανατίθεται σε μια ομάδα, το κέντρο δύναται να μετακινηθεί ώστε να αντιστοιχεί στο κέντρο της ομάδας, όπως τροποποιήθηκε με την προσθήκη αυτού του δεδομένου. Το νέο κέντρο της ομάδας είναι ο μέσος όρος των δεδομένων που υπάρχουν στην ομάδα. Η ονομασία του αλγορίθμου οφείλεται σε αυτή τη διαδικασία, καθώς τα κέντρα που προκύπτουν είναι οι μέσοι όροι από τα δεδομένα των ομάδων.

2.1.4 Συνθήκη τερματισμού

Στο τέλος κάθε επανάληψης του αλγορίθμου χρησιμοποιείται το άθροισμα του τετραγωνικού σφάλματος, ή διασπορά, ως μέτρο σύγκλισης του. Υπολογίζεται το τετράγωνο της απόστασης κάθε σημείου από το κέντρο της ομάδας του και το άθροισμα αυτών, δίνει το σφάλμα. Τίθεται επίσης μία τιμή κατωφλίου, που όταν το σφάλμα είναι μικρότερο από αυτή, η επαναληπτική διαδικασία τερματίζεται και έχουμε βρει τα τελικά κέντρα των ομάδων. Έχει αποδειχθεί ότι το κέντρο βάρους που ελαχιστοποιεί το άθροισμα τετραγωνικού σφάλματος μιας ομάδας είναι ο μέσος.

2.2 Δέντρα Απόφασης

Τα δέντρα απόφασης (decision trees) είναι από τα πιο γνωστά και αποτελεσματικά μοντέλα κατηγοριοποίησης. Τα δέντρα απόφασης προκύπτουν από την εφαρμογή τεχνικών «διαίρει και βασίλευε» (divide and conquer) σε ένα σύνολο δεδομένων μέσω μιας επαγωγικής διαδικασίας μάθησης. Ένα δέντρο απόφασης επάγεται από ένα σύνολο μάθησης (train set), που αποτελείται από δεδομένα x_i, καθένα από τα οποία περιγράφεται από ένα σύνολο χαρακτηριστικών a_1 ,a_2 ,...,a_n και από μια κατηγορία y_j. Τα χαρακτηριστικά που περιγράφουν το σύνολο δεδομένων, όπως είπαμε και πιο πριν, μπορεί να είναι είτε συνεχής ή ακέραιοι αριθμοί είτε κατηγορικά χαρακτηριστικά.

Το δέντρο απόφασης ουσιαστικά είναι ένα σύνολο από “ερωτήσεις” που, με βάση κάποιο χαρακτηριστικό, χωρίζουν το σύνολο δεδομένων σε μικρότερα σύνολα, ώσπου όλα τα δεδομένα x_i, ή το μεγαλύτερο μέρος, να ανήκουν σε μια κατηγορία. Η σειρά

των “ερωτήσεων” και οι πιθανές απαντήσεις οργανώνονται στην μορφή ενός δέντρου απόφασης, το οποίο είναι μια ιεραρχική δομή που αποτελείται από κόμβους και κατευθυνόμενες ακμές. Ουσιαστικά το δέντρο είναι μια ιεραρχημένη συλλογή σύνθετων διαζευκτικών προτάσεων, οι οποίες με τη σειρά τους αποτελούνται από ένα σύνολο λογικών συζεύξεων.

Η σημαντικότερη πρόταση σε κάθε σύζευξη είναι η πρώτη, ο πρώτος κόμβος δηλαδή, και ορίζεται ως «ρίζα» του δέντρου. Όλοι οι κόμβοι του δέντρου, συμπεριλαμβανομένης και της ρίζας, προσδιορίζονται από τα ονόματα των χαρακτηριστικών με βάση τα οποία διαχωρίζουν το σύνολο δεδομένων. Τα κλαδιά – ακμές του δέντρου προσδιορίζονται από τις δυνατές διακριτές τιμές ή το εύρος τιμών που μπορεί να λάβει το χαρακτηριστικό ακολουθώντας αυτή τη διαδρομή. Τα φύλλα οδηγούν στις διαφορετικές κατηγορίες.

Ένα δέντρο απόφασης περιέχει 3 τύπους κόμβων:

- Τον **κόμβο ρίζα** που είναι η πρώτη “ερώτηση” που πραγματοποιείται και έτσι δεν έχει καμία εισερχόμενη ακμή και καμία ή περισσότερες εξερχόμενες.
- Τους **εσωτερικούς κόμβους** που ο καθένας έχει ακριβώς μία εισερχόμενη ακμή και 2 ή περισσότερες εξερχόμενες.
- Τα **φύλλα** του δέντρου, που καθένα έχει ακριβώς μία εισερχόμενη ακμή και καμία εξερχόμενη.

Η διαδικασία της κατηγοριοποίησης πραγματοποιείται ακολουθώντας ένα μονοπάτι που οδηγεί προς τα κάτω στο δέντρο, επιλέγοντας τα κλαδιά που αντιστοιχούν στις τιμές των χαρακτηριστικών των δεδομένων.

Η πλέον χρησιμοποιούμενη επαγωγή μάθησης για την κατασκευή δέντρων απόφασης είναι η επαγωγική κατασκευή δέντρων απόφασης από την κορυφή προς τα κάτω (top down). Ο στόχος ενός αλγόριθμου κατασκευής δέντρων απόφασης από ένα σύνολο δεδομένων είναι η ανάπτυξη ενός μοντέλου που αντιστοιχίζει σωστά τα δεδομένα με την κάθε κατηγορία. Ένας γενικός αλγόριθμος για την κατασκευή ενός δέντρου απόφασης είναι ο εξής, θεωρώντας αρχικά έναν κόμβο ρίζα και ένα σύνολο δεδομένων εκπαίδευσης:

- αν όλα τα δεδομένα του κόμβου ανήκουν στην ίδια κατηγορία
- τότε
    - Μετατροπή του κόμβου σε φύλλο του δέντρου και ανάθεση σε αυτόν της κατηγορίας
    - Τερματισμός της διαδικασίας
- αλλιώς
    - Βαθμολόγηση κάθε χαρακτηριστικού χρησιμοποιώντας ένα κριτήριο διαχωρισμού
    - Επιλογή του καλύτερου χαρακτηριστικού και δημιουργία κόμβων-παιδιά όσες οι πιθανές τιμές του χαρακτηριστικού. Αν δεν υπάρχει κατάλληλο χαρακτηριστικό τότε τερματίζει η διαδικασία
    - Κατανομή των δεδομένων εκπαίδευσης στους κόμβους-παιδιά και για καθένα επανάληψη της διαδικασίας από την αρχή αναδρομικά

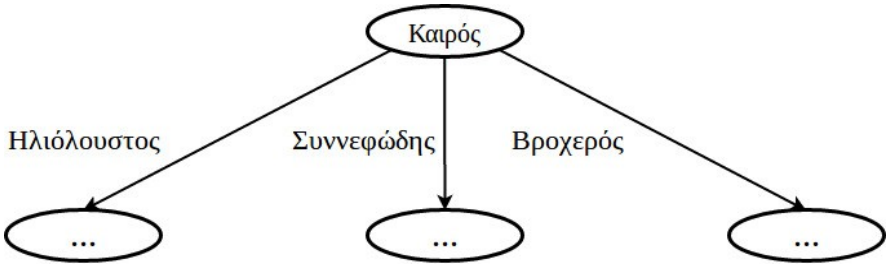
Πίνακας 2.1 – Σύνολο δεδομένων Weather

Καιρός	Θερμοκρασία	Υγρασία	Ανεμώδης	Παχνίδι
Ηλιόλουστος	Ζέστη	Υψηλή	Όχι	Όχι
Ηλιόλουστος	Ζέστη	Υψηλή	Ναι	Όχι
Συννεφώδης	Ζέστη	Υψηλή	Όχι	Ναι
Βροχερός	Ήπια	Υψηλή	Όχι	Ναι
Βροχερός	Δροσιά	Κανονική	Όχι	Ναι
Βροχερός	Δροσιά	Κανονική	Ναι	Όχι
Συννεφώδης	Δροσιά	Κανονική	Ναι	Ναι
Ηλιόλουστος	Ήπια	Υψηλή	Όχι	Όχι
Ηλιόλουστος	Δροσιά	Κανονική	Όχι	Ναι
Βροχερός	Ήπια	Κανονική	Όχι	Ναι
Ηλιόλουστος	Ήπια	Κανονική	Ναι	Ναι
Συννεφώδης	Ήπια	Υψηλή	Ναι	Ναι
Συννεφώδης	Ζέστη	Κανονική	Όχι	Ναι
Βροχερός	Ήπια	Υψηλή	Ναι	Όχι

Για την καλύτερη κατανόηση της κατηγοριοποίησης με δέντρα απόφασης θα παρουσιάσουμε ένα παράδειγμα που χρησιμοποιείται συχνά, με ένα απλό πρόβλημα κατηγοριοποίησης. Ας θεωρήσουμε το σύνολο δεδομένων του Πίνακα 2.1. Τα χαρακτηριστικά του συνόλου δεδομένων είναι : *Καιρός, Θερμοκρασία, Υγρασία,*

Ανεμώδης. Η κατηγορία είναι αν έπαιξαν ποδόσφαιρο ή όχι. Όλα τα χαρακτηριστικά είναι κατηγορικά.

Βλέπουμε πως στο σύνολο εκπαίδευσης έχουμε 9 δεδομένα στην κατηγορία *Ναι* και 5 στην κατηγορία *Όχι*. Προχωράμε στην διαδικασία κατασκευής του δέντρου απόφασης. Τα κριτήριο επιλογής χαρακτηριστικού διαχωρισμού θα τα δούμε στην συνέχεια. Θεωρούμε πως επιλέγουμε το χαρακτηριστικό 'Καιρός' για τον πρώτο διαχωρισμό. Έτσι αρχικά το δέντρο διαμορφώνεται όπως φαίνεται στην Εικόνα 2.2.



Εικόνα 2.2 - Το δέντρο απόφασης μετά τον πρώτο διαχωρισμό

Στους πίνακες 2.2 , 2.3 , 2.4 παρουσιάζονται τα δεδομένα που έχουν καταμεριστεί στους εσωτερικούς κόμβους μετά τον πρώτο διαχωρισμό. Στον 1ο εσωτερικό κόμβο με την τιμή του χαρακτηριστικού *Καιρός* να είναι *Ηλιόλουστος* (Πίνακας 2.2), έχουμε 3 δεδομένα στην κατηγορία *Όχι* και 2 στην κατηγορία *ΝΑΙ*. Παρατηρούμε πως αν επαναλάβουμε την διαδικασία αναδρομικά σε αυτόν τον κόμβο και επιλέξουμε ως χαρακτηριστικό διαχωρισμού την '*Υγρασία*' θα δημιουργηθούν 2 φύλλα. Στο ένα φύλλο που θα έχουμε την *Υψηλή* υγρασία θα έχουμε την ετικέτα κατηγορίας *Όχι* ενώ στο άλλο φύλλο με *Κανονική* υγρασία θα έχουμε την ετικέτα κατηγορίας *Ναι*.

Πίνακας 2.2 - Το σύνολο δεδομένων για τον πρώτο κόμβο - παιδί

Καιρός	Θερμοκρασία	Υγρασία	Ανεμώδης	Παχνίδι
Ηλιόλουστος	Ζέστη	Υψηλή	Όχι	Όχι
Ηλιόλουστος	Ζέστη	Υψηλή	Ναι	Όχι
Ηλιόλουστος	Ήπια	Υψηλή	Όχι	Όχι
Ηλιόλουστος	Δροσιά	Κανονική	Όχι	Ναι
Ηλιόλουστος	Ήπια	Κανονική	Ναι	Ναι

Πίνακας 2.3 - Το σύνολο δεδομένων για τον δεύτερο κόμβο - παιδί

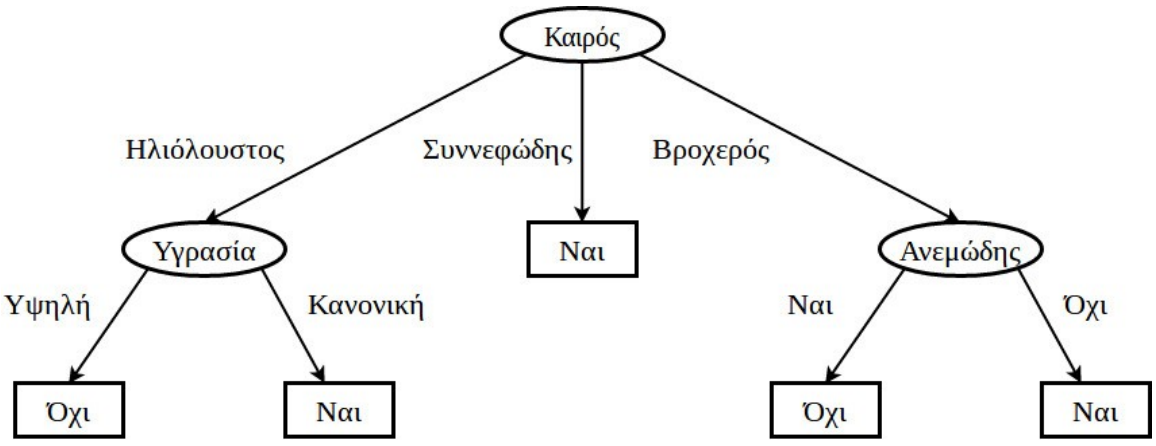
Καιρός	Θερμοκρασία	Υγρασία	Ανεμώδης	Παχνίδι
Συννεφώδης	Ζέστη	Υψηλή	Όχι	Ναι
Συννεφώδης	Δροσιά	Κανονική	Ναι	Ναι
Συννεφώδης	Ήπια	Υψηλή	Ναι	Ναι
Συννεφώδης	Ζέστη	Κανονική	Όχι	Ναι

Πίνακας 2.4 - Το σύνολο δεδομένων για τον τρίτο κόμβο - παιδί

Καιρός	Θερμοκρασία	Υγρασία	Ανεμώδης	Παχνίδι
Βροχερός	Ήπια	Υψηλή	Όχι	Ναι
Βροχερός	Δροσιά	Κανονική	Όχι	Ναι
Βροχερός	Δροσιά	Κανονική	Ναι	Όχι
Βροχερός	Ήπια	Κανονική	Όχι	Ναι
Βροχερός	Ήπια	Υψηλή	Ναι	Όχι

Αντίστοιχα και στον 3ο εσωτερικό κόμβο με την τιμή του χαρακτηριστικού *Καιρός* να είναι *Βροχερός* (Πίνακας 2.4). Σε αυτόν τον κόμβο έχουμε 3 δεδομένα στην κατηγορία *Ναι* και 2 στην κατηγορία *Όχι*. Παρατηρούμε πως αν επαναλάβουμε την διαδικασία αναδρομικά σε αυτόν τον κόμβο και επιλέξουμε ως χαρακτηριστικό διαχωρισμού το χαρακτηριστικό '*Ανεμώδης*' θα δημιουργηθούν και εδώ 2 φύλλα που θα κατηγοριοποιούν σωστά όλα τα δεδομένα.

Στον 2ο εσωτερικό κόμβο παρατηρούμε πως όλα τα δεδομένα είναι στην κατηγορία *Ναι*, άρα ο κόμβος μετατρέπεται σε φύλλο με ετικέτα κατηγορίας *Ναι*. Στην Εικόνα 2.3 παρουσιάζεται το τελικό δέντρο που κατασκευάστηκε.



Εικόνα 2.3 – Τελικό δέντρο απόφασης του παραδείγματος

Κάποια ζητήματα που μπορεί να προκύψουν με την εφαρμογή του παραπάνω γενικού αλγορίθμου κατασκευής δέντρων απόφασης είναι:

1. Αν κανένα δεδομένο στο σύνολο εκπαίδευσης δεν περιέχει τον συνδυασμό τιμών των χαρακτηριστικών που οδηγούν σε έναν κόμβο, τότε ο κόμβος αυτός θα είναι άδειος και δεν θα μπορεί να συνεχίσει την επαγωγική διαδικασία κατασκευής του δέντρου. Σε αυτήν την περίπτωση, ο κόμβος μετατρέπεται σε φύλλο και έχει ως ετικέτα κατηγορίας την κατηγορία στην οποία ανήκει η πλειοψηφία των δεδομένων εκπαίδευσης του κόμβου-γονέα του.
2. Αν όλα τα δεδομένα ενός κόμβου έχουν τις ίδιες τιμές χαρακτηριστικών αλλά διαφορετική ετικέτα κατηγορίας τότε δεν μπορεί να γίνει ο διαχωρισμός και να συνεχίσει η επαγωγική διαδικασία κατασκευής του δέντρου. Σε αυτή τη περίπτωση ο κόμβος μετατρέπεται σε φύλλο με ετικέτα κατηγορίας την κατηγορία την οποία ανήκει η πλειοψηφία των δεδομένων εκπαίδευσης του κόμβου.

Στον παραπάνω γενικό αλγόριθμο κατασκευής δέντρων απόφασης αφήσαμε κάποια ζητήματα ανοιχτά. Πιο συγκεκριμένα, έναν αλγόριθμος μάθησης για την κατασκευή ενός δέντρου απόφασης χρειάζεται να αντιμετωπίζει τα παρακάτω 2 βασικά ζητήματα:

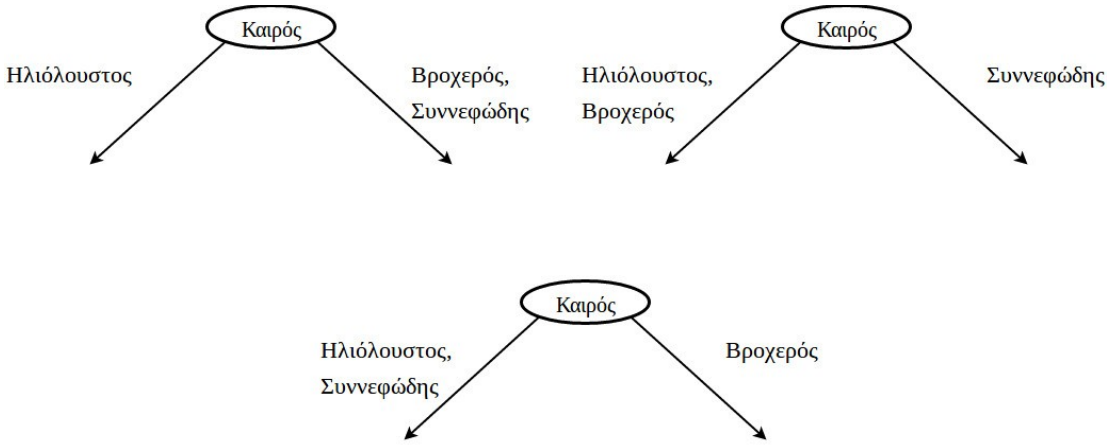
1. Ποιο είναι το κριτήριο επιλογής του χαρακτηριστικού διαχωρισμού;
2. Πότε σταματάει η αναδρομική διαδικασία κατασκευής του δέντρου απόφασης;

2.2.1 Κριτήριο επιλογής του χαρακτηριστικού διαχωρισμού

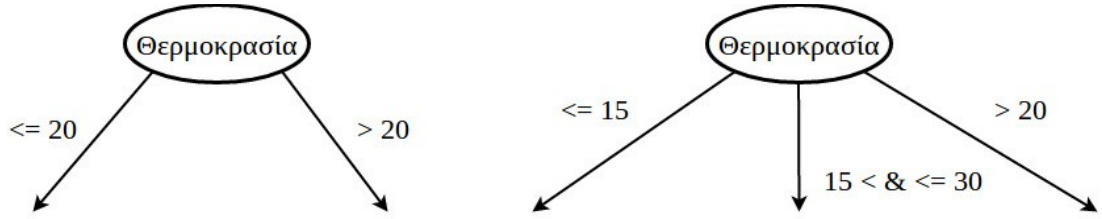
Για να προχωρήσουμε στην ανάλυση των κριτηρίων επιλογής χαρακτηριστικού διαχωρισμού, χρειάζεται πρώτα να δούμε πως εκφράζονται οι συνθήκες ελέγχου. Στο παράδειγμα που είδαμε είχαμε μόνο κατηγορικά χαρακτηριστικά, με το πλήθος των κόμβων παιδιών που δημιουργήθηκε να είναι ίσος με τον αριθμό των διακριτών τιμών του αντίστοιχου χαρακτηριστικού. Αυτό όμως δεν ισχύει πάντα, καθορίζεται ανάλογα με τον αλγόριθμο. Μερικοί αλγόριθμοι δέντρων απόφασης, όπως ο γνωστός αλγόριθμος CART [20], παράγουν μόνο δυαδικούς διαχωρισμούς, επιλέγοντας τον

καλύτερο από όλους τους $2^{κ-1}$ – 1 πιθανούς δυαδικούς διαχωρισμούς k τιμών ενός χαρακτηριστικού. Οι πιθανοί δυαδικοί διαχωρισμοί του παραδείγματος που είδαμε προηγουμένως φαίνονται στην Εικόνα 2.4.

Τι γίνεται όμως στην περίπτωση που έχουμε συνεχή χαρακτηριστικά; Σε αυτή την περίπτωση η συνθήκη ελέγχου εκφράζεται ως ένα ή περισσότερα κριτήρια σύγκρισης με κάποιον αριθμό (π.χ. < u). Αυτός ο αριθμός ονομάζεται σημείο διαχωρισμού. Και στα συνεχή χαρακτηριστικά υπάρχει η δυνατότητα του δυαδικού διαχωρισμού και του διαχωρισμού πολλών κατευθύνσεων. Ο δυαδικός διαχωρισμός απαιτεί ένα σημείο διαχωρισμού, βρίσκοντας το καλύτερο από όλα τα πιθανά. Για τον πολλαπλό διαχωρισμό χρειάζεται να ληφθούν υπόψιν όλες οι πιθανές περιοχές των συνεχών τιμών. Οι δύο προσεγγίσεις φαίνονται στην Εικόνα 2.5.



Εικόνα 2.4 – Πιθανοί δυαδικοί διαχωρισμοί του παραδείγματος



Εικόνα 2.5 – Διαχωρισμοί σε συνεχή χαρακτηριστικά

Υπάρχουν πολλά μέτρα για την επιλογή του κατάλληλου χαρακτηριστικού διαχωρισμού. Όλα τα μέτρα βασίζονται στην κατανομή των κατηγοριών στις οποίες ανήκουν τα δεδομένα πριν και μετά τον διαχωρισμό. Ένα μέτρο του βαθμού ανομοιογένειας είναι η *εντροπία (Shannon's entropy) [21]*. Αν y_i είναι οι ετικέτες κατηγορίας του προβλήματος και p_i είναι η πιθανότητα εμφάνισης της κάθε κατηγορίας, με i = 1, 2, ..., c και c το πλήθος των κατηγοριών, τότε η εντροπία E του συνόλου δεδομένων D ορίζεται ως:

$$E(D) = -\sum_{i=1}^{c} p_i \log_2 p_i . \qquad \text{(Εξ 2.1)}$$

Στην περίπτωση που η πιθανότητα εμφάνισης μιας κατηγορίας είναι 0, υπάρχει η θεώρηση πως $0 * \log_2 0 = 0$. Η εντροπία ουσιαστικά εκφράζει την βεβαιότητα ως προς την εμφάνιση μιας κατηγορίας σε ένα σύνολο δεδομένων. Για παράδειγμα ένας κόμβος που το σύνολο δεδομένων του έχει κατανομή κατηγοριών (0,2) έχει εντροπία E=0*0+1*0=0, ενώ ένας κόμβος που το σύνολο δεδομένων του έχει κατανομή κατηγοριών (1,1) έχει εντροπία E = - (-0.5 – 0.5) = 1.

Όπως παρατηρούμε, μικρές τιμές εντροπίας οδηγούν σε μεγαλύτερη βεβαιότητα, ενώ μεγάλες τιμές εντροπίας δείχνουν μικρή βεβαιότητα για την πρόβλεψη κατηγορίας. Έτσι, για το χτίσιμο του δέντρου απόφασης, χρειάζεται να επιλέξουμε το χαρακτηριστικό που μας οδηγεί σε μεγαλύτερη μείωση της εντροπίας, ή αλλιώς σε μεγαλύτερο *κέρδος πληροφορίας [9]*. Το *κέρδος πληροφορίας (information gain) IG* ενός χαρακτηριστικού a_i, σε ένα σύνολο δεδομένων D ορίζεται ως:

$$IG(D, a_i) = E(D) - \sum_{v \in V_a} \frac{|D_v|}{|D|} E(D_v) \qquad \text{(Εξ 2.2)}$$

όπου V_a είναι το σύνολο των κλαδιών που δημιουργούνται με τον διαχωρισμό στο χαρακτηριστικό a_i. D_v είναι ένα υποσύνολο δεδομένων του D με $v \in V_a$, ενώ $|D_v|$ και $|D|$ είναι το μέγεθος των συνόλων D_v και D , αντίστοιχα. Αυτό το κριτήριο χρησιμοποιείται στο γνωστό αλγόριθμο κατασκευής δέντρων απόφασης ID3.

Η κατασκευή δέντρων απόφασης με κριτήριο διαχωρισμού το *κέρδος πληροφορίας* IG έχει ένα σημαντικό μειονέκτημα. Οδηγείται εύκολα σε

χαρακτηριστικά που έχουν μεγάλο πλήθος διακριτών τιμών, δηλαδή μεγάλο μέγεθος του συνόλου V_a (μεγάλο πλήθος τιμών του χαρακτηριστικού που επιλέγεται σε ένα κόμβο του δέντρου οδηγεί στην παραγωγή ενός αντίστοιχα μεγάλου αριθμού κόμβων παιδιών). Το αποτέλεσμα είναι η επαγωγή δέντρων απόφασης τα οποία είναι πολύπλοκα και έχουν μεγάλο παράγοντα διακλάδωσης, τείνουν να είναι εξειδικευμένα στο σύνολο εκπαίδευσης, δηλαδή να υπερπροσαρμόζονται, και παρουσιάζουν μικρή γενικευτική ικανότητα. Για παράδειγμα σε ένα σύνολο δεδομένων με τα χαρακτηριστικά: *κωδικός πελάτη*, τύπος αυτοκινήτου ... , το χαρακτηριστικό κωδικός πελάτη, αν και πρακτικά άσχετο με το πρόβλημα, θα έχει μεγαλύτερο IG αφού έχει X διακριτές τιμές και άρα θα δημιουργήσει X κλαδιά, με 1 εγγραφή στο καθένα.

Για την αντιμετώπιση του παραπάνω προβλήματος, υπάρχουν 2 κύριες στρατηγικές. Ο πρώτος τρόπος είναι να έχουμε μόνο δυαδικούς διαχωρισμούς. Αυτός ο τρόπος χρησιμοποιείται από τον αλγόριθμο CART. Η δεύτερη στρατηγική είναι να αλλάξει το κριτήριο διαχωρισμού, ώστε ο αλγόριθμος να λαμβάνει υπόψιν και το πλήθος των κλαδιών που δημιουργούνται στο δέντρο μετά τον διαχωρισμό. Ο πιο γνωστός αλγόριθμος επαγωγής δέντρων απόφασης, ο C4.5 [21], χρησιμοποιεί ως κριτήριο διαχωρισμού την *αναλογία κέρδους* (*gain ratio*).

Η αναλογία κέρδους λαμβάνει υπόψιν το κέρδος πληροφορίας κάθε χαρακτηριστικού, σε συνδυασμό όμως και με το πλήθος των αποτελεσμάτων που επιστρέφει ο διαχωρισμός, που μετριέται με την πληροφορία διαχωρισμού (split information). Η *πληροφορία διαχωρισμού* έχει σκοπό να αποθαρρύνει την επιλογή χαρακτηριστικών που έχουν μεγάλο κέρδος πληροφορίας λόγω μεγάλου πλήθους τιμών. Η πληροφορία διαχωρισμού S ενός χαρακτηριστικού a_i, ορίζεται ως:

$$S\left(D,a_i\right)=\sum_{v\in V_a}\frac{\left(D_v\right)}{\left(D\right)}*\log_2\left(\frac{\left(D_v\right)}{\left(D\right)}\right) \qquad \text{(Εξ 2.3)}$$

Έτσι, η αναλογία κέρδους GAIN ορίζεται ως ο λόγος του κέρδους πληροφορίας με την πληροφορία διαχωρισμού:

$$GAIN\left(D,a_i\right)=\frac{IG\left(D,a_i\right)}{S\left(D,a_i\right)} \qquad \text{(Εξ 2.4)}$$

2.2.2 Κλάδεμα (Pruning)

Ένα πρόβλημα που χρειάζεται να λύσουν οι αλγόριθμοι επαγωγής δέντρων απόφασης είναι η υπερπροσαρμοστικότητα (overfitting). Δέντρα απόφασης με μεγάλο μέγεθος μπορεί να είναι απόλυτα προσαρμοσμένα στο σύνολο δεδομένων εκπαίδευσης και να μην αντικατοπτρίζουν πραγματικές συσχετίσεις μεταξύ των χαρακτηριστικών και των κατηγοριών.

Για την αντιμετώπιση του προβλήματος ακολουθούνται οι εξής βασικές προσεγγίσεις:

1. Κλάδεμα κατά την ανάπτυξη του δέντρου (Pre-pruning). Το δέντρο εμποδίζεται να αναπτυχθεί τελείως, χρησιμοποιώντας ένα κριτήριο τερματισμού όπως είναι ο ορισμός ενός κατώτατου ορίου ως προς το πλήθος των δεδομένων που πρέπει να υπάρχουν σε κάθε κόμβο.
2. Κλάδεμα μετά την ανάπτυξη του δέντρου (Post-pruning). Το δέντρο αναπτύσσεται κανονικά και στη συνέχεια τμήματά του περικόπτονται με βάση κάποιο κριτήριο σφάλματος σε κάθε φύλλο, ώστε να προκύψει το τελικό δέντρο απόφασης.

Να τονιστεί πως δεν υπάρχει μέθοδος κλαδέματος που να λειτουργεί αποτελεσματικά για όλα τα προβλήματα.

2.3 Κατηγοριοποιητές ομάδων

Ένας από τους τρόπους για την βελτίωση της ορθότητας της κατηγοριοποίησης και της αποφυγής φαινομένων υπερπροσαρμοστικότητας είναι η συνάθροιση των προβλέψεων πολλών κατηγοριοποιητών. Αυτές οι τεχνικές είναι γνωστές ως μέθοδοι ομάδων (ensemble methods). [13]

Η βασική ιδέα των μεθόδων ομάδων είναι πως αν από τα δεδομένα εκπαίδευσης που είναι διαθέσιμα δημιουργηθούν περισσότεροι από έναν διαφορετικοί κατηγοριοποιητές, η κατηγοριοποίηση των μελλοντικών, άγνωστων, δεδομένων με βάση των συνδυασμό των αποφάσεών τους (π.χ. με ψήφο) θα μειώσει το ποσοστό λάθους των μεμονωμένων κατηγοριοποιητών. Για να επιτευχθεί αυτό χρειάζεται το

μοντέλο που θα δημιουργήσουν οι κατηγοριοποιητές της ομάδας να είναι διαφορετικό ώστε τα σφάλματά τους να μην είναι συσχετιζόμενα. Αν και στην πράξη είναι πολύ δύσκολο οι κατηγοριοποιητές να είναι πλήρως ανεξάρτητοι μεταξύ τους, με εξασφάλιση μικρής συσχέτισης επιτυγχάνεται βελτίωση στην ακρίβεια των αποτελεσμάτων.

2.3.1 Τεχνικές κατασκευής ομάδας κατηγοριοποιητών

Για την δημιουργία μιας ομάδας κατηγοριοποιητών υπάρχουν αρκετές τεχνικές, συνήθως χρησιμοποιώντας τον ίδιο βασικό αλγόριθμο [1]. Οι βασικές ιδέες των τεχνικών αυτών είναι:

1. **Προσαρμογή του συνόλου εκπαίδευσης**. Με αυτήν την προσέγγιση το αρχικό σύνολο δεδομένων σπάει σε πολλά μικρότερα σύνολα εκπαίδευσης με επαναδειγματοληψία των αρχικών δεδομένων. Με αυτά τα διαφορετικά σύνολα εκπαίδευσης εκπαιδεύονται οι κατηγοριοποιητές χρησιμοποιώντας κάποιον συγκεκριμένο αλγόριθμο. Με αυτόν τον τρόπο, αφού οι κατηγοριοποιητές έχουν εκπαιδευθεί με διαφορετικό σύνολο εκπαίδευσης, επιτυγχάνεται αυτό που αναφέρθηκε παραπάνω, δηλαδή να εξασφαλιστεί μικρή συσχέτιση μεταξύ τους. Παραδείγματα δημιουργίας ομάδας κατηγοριοποιητών με αυτή τη μέθοδο είναι η εμφωλίαση (bagging) και η ενίσχυση (boosting).
2. **Διαχείριση των χαρακτηριστικών του συνόλου εκπαίδευσης.** Με αυτήν την τεχνική δημιουργούνται πάλι πολλαπλά σύνολα εκπαίδευσης, 1 για κάθε κατηγοριοποιητή, αλλά χωρίς δειγματοληψία στα δεδομένα. Η διαφορά στα σύνολα εκπαίδευσης βρίσκεται στα χαρακτηριστικά των δεδομένων, αφού για καθένα επιλέγεται διαφορετικό υποσύνολο των χαρακτηριστικών. Η επιλογή του υποσυνόλου των χαρακτηριστικών μπορεί να γίνει είτε με τυχαίο τρόπο, είτε με βάση κάποια πιθανότητα επιλογής κάθε γνωρίσματος με βάση την εντροπία ή κάποιο άλλο μέτρο ανομοιογένειας. Αυτή η τεχνική λειτουργεί πολύ αποτελεσματικά όταν τα σύνολα δεδομένων περιέχουν πλεονάζοντα γνωρίσματα. Τα τυχαία δάση (Random Forests) είναι ένας πολύ γνωστός αλγόριθμος που δημιουργεί τους κατηγοριοποιητές του με αυτό το τρόπο, χρησιμοποιώντας δέντρα απόφασης.

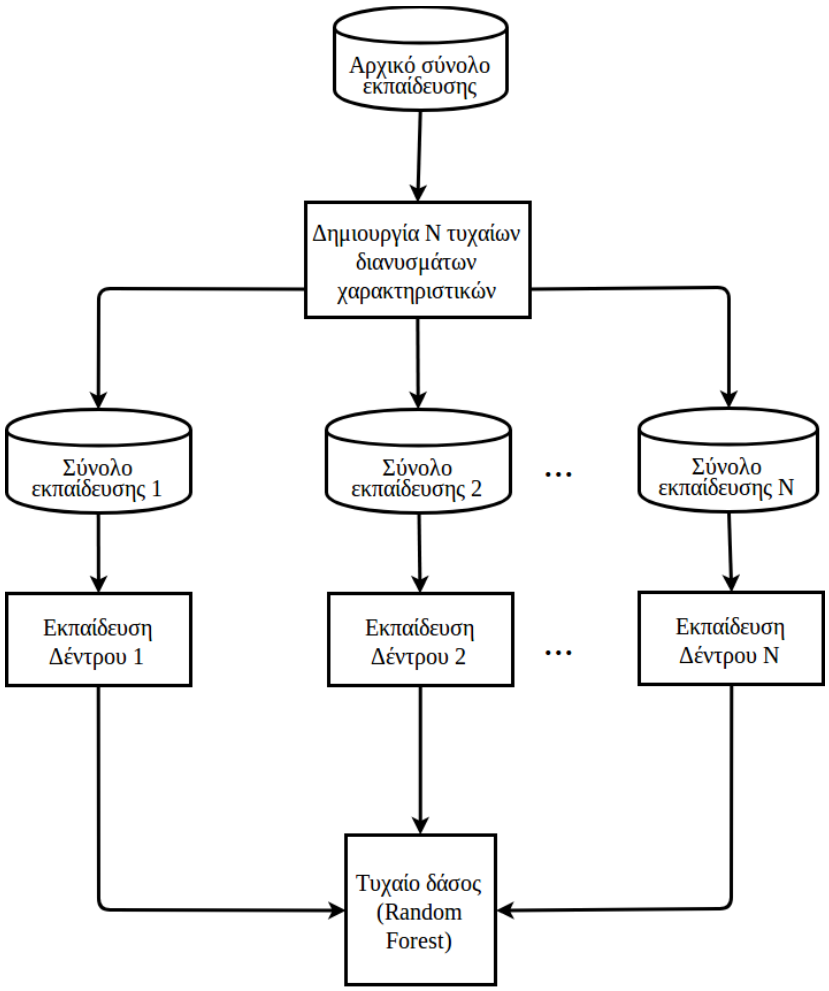
3. **Διαχείριση των ετικετών κατηγορίας του συνόλου εκπαίδευσης**. Αυτή η τεχνική χρησιμοποιείται κυρίως σε περιπτώσεις που οι κατηγορίες είναι σχετικά πολλές. Με αυτή τη μέθοδο, το σύνολο εκπαίδευσης μετασχηματίζεται σε πρόβλημα δυαδικής κατηγοριοποίησης, διαχωρίζοντας τυχαία τις ετικέτες κατηγορίας σε 2 ομάδες, τις K_0 και K_1 . Ο κάθε κατηγοριοποιητής εκπαιδεύεται με το ίδιο σύνολο εκπαίδευσης, με την διαφορά να έγκειται στις ετικέτες κατηγορίας. Η κατηγοριοποίηση των μελλοντικών δεδομένων λειτουργεί ως εξής: Ο κάθε κατηγοριοποιητής "ψηφίζει" μια ομάδα των ετικετών κατηγορίας, και όλες οι κατηγορίες που ανήκουν σε αυτή την ομάδα παίρνουν από μια ψήφο. Στο τέλος προκύπτει η κατηγορία με τις περισσότερες ψήφους.
4. **Διαχείριση του αλγόριθμου εκπαίδευσης.** Οι τρεις προηγούμενες προσεγγίσεις είναι γενικές μέθοδοι που εφαρμόζονται με όλους τους αλγόριθμους εκπαίδευσης, αφού η προσαρμογή που επηρεάζει την δημιουργία των μοντέλων κατηγοριοποίησης γίνεται στο σύνολο εκπαίδευσης. Αυτή η προσέγγιση μπορεί να εφαρμοστεί με συγκεκριμένους αλγορίθμους που μπορούν να χρησιμοποιηθούν με τέτοιο τρόπο ώστε με το ίδιο σύνολο δεδομένων να δημιουργηθούν διαφορετικά μοντέλα. Παραδείγματα τέτοιων αλγορίθμων είναι τα δέντρα απόφασης που με εισαγωγή τυχαιότητας στην επιλογή των χαρακτηριστικών διαχωρισμού, μπορούν να προκύψουν πολλά διαφορετικά μοντέλα. Αντίστοιχα, τα τεχνητά νευρωνικά δίκτυα μπορούν να παράξουν διαφορετικά μοντέλα αλλάζοντας την τοπολογία τους ή τα αρχικά βάρη των συνδέσμων των νευρώνων τους.

2.3.2 Τυχαία δάση (Random Forests)

Όπως αναφέρθηκε και πριν, τα τυχαία δάση είναι ένας αλγόριθμος κατηγοριοποιητών ομάδων, που χρησιμοποιεί δέντρα απόφασης, με κυρίαρχη τεχνική κατασκευής των κατηγοριοποιητών την δημιουργία των μοντέλων του με διαχείριση των χαρακτηριστικών του συνόλου δεδομένων. Υπάρχουν και άλλες προσεγγίσεις, π.χ. με εμφωλίαση και χρήση δέντρων απόφασης, όπου η τυχαιότητα εισάγεται μέσα από την τυχαία επιλογή υποσύνολου δεδομένων στο σύνολο δεδομένων εκπαίδευσης. Τα τυχαία δάση χρησιμοποιούν μεγάλο αριθμό δέντρων απόφασης, ώστε να

περιορίζεται το σφάλμα από την εμφάνιση φαινομένων υπερεκπαίδευσης στα ξεχωριστά δέντρα.

Με βάση την κυρίαρχη προσέγγιση [17], το κάθε δέντρο παράγεται με βάση ένα διαφορετικό, επιλεγμένο τυχαία, διάνυσμα - υποσύνολο χαρακτηριστικών του συνόλου δεδομένων. Ακόμα, χρησιμοποιείται και η τεχνική της εμφωλίασης (bagging), που χρησιμοποιεί δειγματοληψία με επανατοποθέτηση, δημιουργώντας νέα σύνολα εκπαίδευσης, ίδιου μεγέθους, για την κατασκευή κάθε ξεχωριστού κατηγοριοποιητή.



Εικόνα 2.6 – Δημιουργία τυχαίου δάσους

Τα δέντρα απόφασης στην συνέχεια κατασκευάζονται με την διαδικασία που αναφέρθηκε πριν. Κάθε δέντρο αναπτύσσεται στον μεγαλύτερο δυνατό βαθμό, χωρίς

όμως να χρησιμοποιείται κανένα είδος κλαδέματος. Ένα τυχαίο δάσος, για να προχωρήσει στην κατηγοριοποίηση των μελλοντικών δεδομένων, συνδυάζει τις ψήφους – προβλέψεις από όλα τα δέντρα απόφασης που έχουν κατασκευαστεί.

Το κάθε μεμονωμένο δέντρο κατηγοριοποιεί τα στοιχεία ξεχωριστά με τον τρόπο που αναλύθηκε στην ενότητα 2.2. Στην Εικόνα 2.6 φαίνονται τα βασικά βήματα που ακολουθεί ο αλγόριθμος δημιουργίας ενός τυχαίου δάσους.

Το ποσοστό λάθους των τυχαίων δασών στην πρόβλεψη μελλοντικών αποτελεσμάτων, έχει αποδειχθεί ότι εξαρτάται από δύο κύριους παράγοντες: τη συσχέτιση μεταξύ δύο δένδρων (correlation) και τη «δύναμη» κάθε δένδρου (strength).

Για την συσχέτιση μεταξύ δύο δέντρων, δηλαδή δύο ξεχωριστών κατηγοριοποιητών αναφερθήκαμε παραπάνω. Όσον αφορά τη δύναμη κάθε δένδρου, με αυτόν τον όρο εννοείται το πόσο καλός κατηγοριοποιητής είναι το εκάστοτε δέντρο απόφασης. Ένα δένδρο με μικρό ποσοστό λάθους, είναι και ένας καλός - «δυνατός» κατηγοριοποιητής.

Ένα από τα βασικά προβλήματα που αντιμετωπίζουν στην κατασκευή τους τα τυχαία δάση, είναι αυτό του καθορισμού του πλήθους F των χαρακτηριστικών που θα επιλεγούν για κάθε κόμβο. Αυτό γιατί και η συσχέτιση μεταξύ των τυχαίων δέντρων, αλλά και η «δύναμή» τους, εξαρτάται από το μέγεθος του F. Αν το F είναι σχετικά μικρό, τότε τα δέντρα τείνουν να είναι λιγότερο συσχετιζόμενα, ενώ η «δύναμή» ενός δέντρου βελτιώνεται με μεγαλύτερο πλήθος χαρακτηριστικών. Για την επιλογή του πλήθους F των επιλεγμένων χαρακτηριστικών για κάθε δέντρο , από ένα πλήθος A χαρακτηριστικών του συνόλου δεδομένων εκπαίδευσης, έχει επικρατήσει να χρησιμοποιείται ο τύπος:

$$F = \log_2 A + 1 \qquad \text{(Εξ 2.5)}$$

Κάποια βασικά **πλεονεκτήματα** των τυχαίων δασών είναι τα ακόλουθα:

- Είναι ανεκτικά ως προς την παρουσία θορύβου.
- Περιλαμβάνει καλές μεθόδους σε περιπτώσεις άγνωστων τιμών σε δεδομένα.
- Υπάρχει η δυνατότητα παράλληλης κατασκευής των δέντρων.

- Λόγω του πλήθους των δέντρων στο δάσος, το σφάλμα γενίκευσης είναι περιορισμένο. Αυτό έχει ως αποτέλεσμα τη μη εμφάνιση φαινομένων υπερεκπαίδευσης.
- Με βάση αρκετές έρευνες είναι ο πιο ακριβής αλγόριθμος κατηγοριοποίησης.

Μερικά **μειονεκτήματα** των τυχαίων δασών είναι τα ακόλουθα:

- Υψηλό υπολογιστικό κόστος.
- Κάθε μελλοντικό στοιχείο πρέπει να διασχίσει όλα τα δέντρα του δάσους για να κατηγοριοποιηθεί.
- Για την επέκταση ενός μοντέλου με στόχο την εισαγωγή π.χ. μιας ακόμα κατηγορίας, απαιτείται ξανά εκπαίδευση και κατασκευή του μοντέλου από την αρχή.
- Η υψηλή τους ικανότητα γενίκευσης και ακρίβειας συνοδεύεται από το τίμημα της περιορισμένης ερμηνευσιμότητας.

2.4 Διασταυρωμένη επικύρωση (Cross Validation)

Η διασταυρωμένη επικύρωση [19] είναι μία μέθοδος εκτίμησης της απόδοσης ενός κατηγοριοποιητή. Είναι μια διαδικασία που ενισχύει την ορθότητα των αποτελεσμάτων, βοηθώντας στην εξαγωγή γενικευμένων, ασφαλών συμπερασμάτων για τη συμπεριφορά του κατηγοριοποιητή σε ένα σύνολο δεδομένων.

Μέτρα απόδοσης, όπως το ποσοστό των επιτυχημένων κατηγοριοποιήσεων, μας δίνουν μια εικόνα της αποτελεσματικότητας του κατηγοριοποιητή σε ένα συγκεκριμένο σύνολο δεδομένων. Από μόνα τους όμως δεν οδηγούν σε ασφαλείς εκτιμήσεις και δεν μας επιτρέπουν να γενικεύσουμε τα συμπεράσματά μας για την απόδοση του κατηγοριοποιητή στο συγκεκριμένο σύνολο δεδομένων.

Κατά την διαδικασία της κατηγοριοποίησης, όπως αναλύθηκε, το σύνολο δεδομένων χωρίζεται σε σετ εκπαίδευσης και σετ αξιολόγησης. Κατά τη διαδικασία εκπαίδευσης χτίζεται ένα μοντέλο αποκλειστικά από τα δεδομένα του συνόλου εκπαίδευσης, χωρίς να υπεισέρχεται κάποια επιπλέον γνώση για το πρόβλημα από τον

προγραμματιστή ή κάποιον ειδήμων. Πως όμως μπορεί να επηρεάσει τη διαδικασία η κατανομή των δεδομένων στο σύνολο;

Έστω θεωρητικά ένα σύνολο το οποίο διαθέτει δύο κατηγορίες με ίσο αριθμό δεδομένων και στην κατανομή των δεδομένων, τα στοιχεία της μίας είναι πρώτα και ακολουθούν τα στοιχεία της δεύτερης. Αν το σύνολο δεδομένων πάρει το πρώτο μισό, το αποτέλεσμα θα είναι ένα μοντέλο που γνωρίζει να κατηγοριοποιεί την πρώτη κατηγορία, αλλά όχι τη δεύτερη, οδηγώντας τη γενικότερη διαδικασία σε αποτυχία. Επιπρόσθετα, το αποτέλεσμα αυτό δεν θα οφείλεται στην έλλειψη δεδομένων για το πρόβλημα αλλά στη μη αξιοποίησή τους για το πρόβλημα.

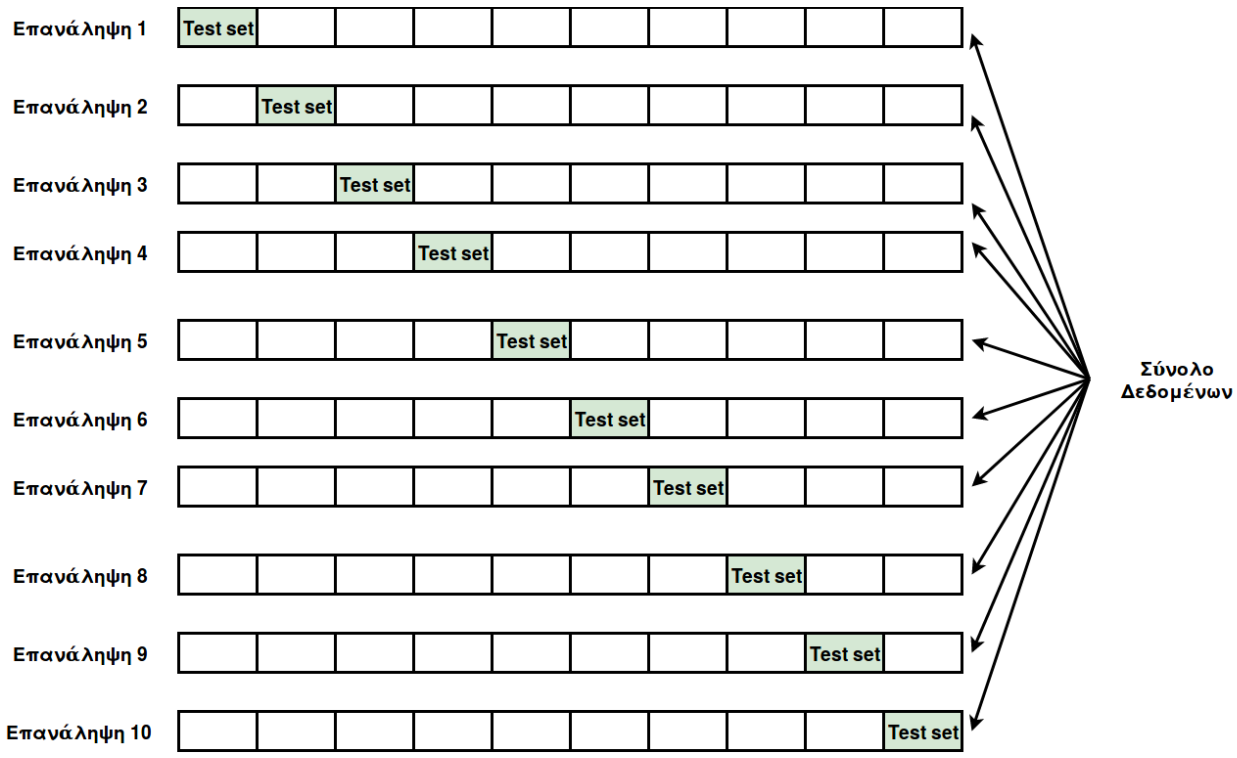
Όπως γίνεται φανερό, οι εκτιμήσεις μας δε μπορούν να είναι βάσιμες και γενικεύσιμες χωρίς περαιτέρω επικύρωση. Είναι αναγκαία λοιπόν η άρση της ευαισθησίας της κατηγοριοποίησης στην κατανομή των δεδομένων. Αυτό επιτυγχάνεται με τη διασταυρωμένη επικύρωση.

Η γενική ιδέα ορίζει ότι το σύνολο δεδομένων χωρίζεται σε ίσα τμήματα και ακολουθεί μια επαναληπτική διαδικασία κατά την οποία κάθε φορά ένα τμήμα χρησιμοποιείται για αξιολόγηση και τα υπόλοιπα για εκπαίδευση του συνόλου, σε κυκλική σειρά. Τα τμήματα είναι διακριτά και μη επικαλυπτόμενα, έτσι κάθε τμήμα του συνόλου δεδομένων χρησιμοποιείται μία ακριβώς φορά για έλεγχο και τις υπόλοιπες για εκπαίδευση. Το συνολικό σφάλμα της κατηγοριοποίησης υπολογίζεται το άθροισμα των σφαλμάτων των επαναλήψεων. Η πλήρης ονομασία της διαδικασίας είναι “k-fold cross validation”, όπου k είναι ο αριθμός των τμημάτων. Συνηθέστερα συναντάται η περίπτωση με k = 10, το “10-fold cross validation” (Εικόνα 2.7).

Μια παραλλαγή του k-fold είναι το “stratified k-fold”, όπου επιπρόσθετα γίνεται προσπάθεια για τήρηση των αναλογιών δεδομένων στα επιμέρους τμήματα. Αναλυτικά, ο χωρισμός σε τμήματα γίνεται με τυχαία επιλογή δεδομένων (χωρίς επανάληψη δεδομένων), τηρείται όμως η αναλογία των κλάσεων του συνόλου και στα υπό-συνολα. Έχει δηλαδή υπολογιστεί εκ των προτέρων τι ποσοστό των δεδομένων καταλαμβάνει κάθε κατηγορία και διατηρείται αναλλοίωτο στα τμήματα.

Μια ακόμη ευρέως χρησιμοποιούμενη παραλλαγή της μεθόδου, περιλαμβάνει τον ορισμό του k σε X, όπου X το πλήθος των δεδομένων. Ακολουθεί μια επαναληπτική διαδικασία N φορών, όπου κάθε φορά το σύνολο αξιολόγησης αποτελείται από ένα δεδομένο, ενώ όλα τα άλλα χρησιμοποιούνται για εκπαίδευση. Αυτό ονομάζεται “leave-one-out cross validation”. Σε γενικές γραμμές η διαδικασία leave-one-out

θεωρείται μια υπολογιστικά ακριβή διαδικασία, όσο τα σύνολα δεδομένων γίνονται μεγαλύτερα σε μέγεθος, καθώς αυτό μεταφράζεται πρωτίστως σε πάρα πολλές επαναλήψεις και μεγάλα σύνολα δεδομένων εκπαίδευσης.



Εικόνα 2.7 – 10-fold cross validation

Για την τελευταία αυτή τεχνική, μπορεί να θεωρηθεί και για περισσότερα δεδομένα, τα οποία όμως αποτελούν μαζί μία ενιαία οντότητα. Παράδειγμα αυτού είναι ένα σύνολο δεδομένων που αποτελείται από ειικονοστοιχεία μερικών εικόνων. Σε τέτοιες περιπτώσεις, leave-one-out μπορεί να θεωρηθεί αν κρατηθεί για αξιολόγηση μια ολόκληρη οντότητα, π.χ. μια ολόκληρη εικόνα.

Τέλος, η διασταυρωμένη επικύρωση μειώνει την πιθανή υπερπροσαρμογή του μοντέλου κατηγοριοποίησης. Αυτό συμβαίνει επι παραδείγματι στην περίπτωση που υπάρχουν δεδομένα στο σύνολο τα οποία δεν είναι έγκυρα (θόρυβος). Η εκπαίδευση και αξιολόγηση σε όλα τα δεδομένα κυκλικά και η συνεκτίμηση των επιμέρους αποτελεσμάτων για την εκτίμηση του συνολικού σφάλματος μειώνει αυτό το φαινόμενο.

2.5 Μήτρα Σύγχυσης (Confusion Matrix)

Η μήτρα σύγχυσης [22] είναι μια συνολική εικόνα των αποτελεσμάτων της κατηγοριοποίησης. Μιλάμε για έναν πίνακα, ο οποίος μας δίνει πληροφορίες για το σε ποια κατηγορία έχουν προβλεφθεί τα δεδομένα σε σύγκριση με το που έπρεπε να έχουν προβλεφθεί και χρησιμοποιείται στη φάση ελέγχου του μοντέλου. Είναι c x c διαστάσεων, όπου c ο αριθμός των κατηγοριών.

Μία γραμμή και μία στήλη αντιστοιχούν σε κάθε κατηγορία. Για κάθε δεδομένο του συνόλου ελέγχου, αυξάνεται κατά ένα η τιμή εκείνου του κελιού που βρίσκεται στη στήλη που αντιστοιχεί στην κατηγορία που έχει προβλεφθεί από το μοντέλο και στη γραμμή που αντιστοιχεί στην κατηγορία που βρίσκεται στην πραγματικότητα. Όπως γίνεται φανερό, τα δεδομένα που έχει προβλέψει σωστά το μοντέλο βρίσκονται πάνω στη διαγώνιο του πίνακα.

Έτσι, βλέποντας τον πίνακα, με μια ματιά διακρίνονται οι κατηγορίες εκείνες που ο αλγόριθμος αποτυγχάνει να προβλέψει σωστά αλλά και εξάγονται συμπεράσματα για τις περιοχές που αποτυγχάνει (π.χ. ο αλγόριθμος προβλέπει μεγάλο μέρος μιας συγκεκριμένης κατηγορίας σε μια άλλη). Επίσης, συγκεντρωτικά οι μετρήσεις που απεικονίζονται στον πίνακα χρησιμοποιούνται ως βάση για την εξαγωγή περαιτέρω μετρικών απόδοσης του αλγορίθμου. Κατά τη διαδικασία της διασταυρωμένης επικύρωσης, τα επιμέρους αποτελέσματα της κάθε επανάληψης συγκεντρώνονται στο τέλος σε έναν πίνακα σύγχυσης δίνοντας μία, συνολική εικόνα για την απόδοση του αλγορίθμου στο σύνολο δεδομένων.

Πίνακας 2.5 - Μήτρα Σύγχυσης

		Προβλεπόμενη Κατηγορία	
		Ασθενείς	**Υγιείς**
Πραγματική Κατηγορία	**Ασθενείς**	**5**	**2**
	Υγιείς	**3**	**4**

Στον Πίνακα 2.5, παρουσιάζεται ένα παράδειγμα μήτρας σύγχυσης μιας υποθετικής κατηγοριοποίησης ενός συνόλου ελέγχου που αποτελείται από 14 δεδομένα, 7 από τα οποία αντιστοιχούν σε ασθενείς και 7 σε υγιείς ανθρώπους. Όπως μας δείχνει ο πίνακας, το υποθετικό μοντέλο έχει προβλέψει σωστά 5 ασθενείς και 4 υγιείς ανθρώπους, ενώ έχει προβλέψει λάθος 2 ασθενείς ως υγιείς και 3 υγιείς ως ασθενείς.

2.6 Μέτρα απόδοσης

Το πιο διαδεδομένο και πιο συχνά χρησιμοποιούμενο μέτρο απόδοσης, όπως είναι φυσικό είναι το **ποσοστό των επιτυχημένων κατηγοριοποιήσεων (accuracy).** Δεδομένου όμως πως αυτό το μέτρο ουσιαστικά θεωρεί όλα τα δεδομένα ως ίδιας σημασίας, σε όποια κατηγορία και να ανήκουν, σε κάποιες περιπτώσεις δεν είναι το πιο κατάλληλο για την ανάλυση των αποτελεσμάτων ενός αλγορίθμου. Αυτό συμβαίνει σε περιπτώσεις όπου οι κατηγορίες δεν είναι ισορροπημένες, δηλαδή το πλήθος εγγραφών μιας κατηγορίας να είναι πολύ περισσότερα από κάποια άλλη.

Τέτοιου είδους σύνολα δεδομένων εμφανίζονται σε πολλές πραγματικές εφαρμογές. Αυτό συμβαίνει π.χ. σε ιατρικά προβλήματα, που το μοντέλο χρειάζεται να εξάγει ως συμπέρασμα αν κάποιος πάσχει από μια ασθένεια ή όχι. Το ποσοστό των εγγραφών που έχουν προέλθει από υγιείς ανθρώπους μπορεί να είναι 99%, ενώ οι ασθενείς να είναι το 1%. Ένας αλγόριθμος που τους κατηγοριοποιεί όλους ως υγιείς, ενώ θα έχει 99% επιτυχία, δεν θα έχει πετύχει στον κύριο σκοπό του, δηλαδή να βρει τους ασθενείς. Κάποιες φορές, η σωστή κατηγοριοποίηση της σπάνιας κατηγορίας, σε τέτοιου είδους προβλήματα, έχει μεγαλύτερη αξία από ότι η σωστή κατηγοριοποίηση της μεγάλης κατηγορίας.

Η **ανάκληση (recall)** ή αλλιώς **ευαισθησία (sensitivity)** και η **ακρίβεια (precision)** [23] είναι δύο διαδεδομένα μέτρα απόδοσης, τα οποία χρησιμοποιούνται σε τέτοιου είδους προβλήματα. Η ευαισθησία είναι η αναλογία των σωστά κατηγοριοποιημένων δειγμάτων μιας κατηγορίας, σε σχέση με το σύνολο των δειγμάτων που ανήκουν σε αυτή τη κατηγορία. Ουσιαστικά είναι το ποσοστό των επιτυχημένων κατηγοριοποιήσεων ανά κατηγορία. Η ακρίβεια είναι το ποσοστό των

σωστά κατηγοριοποιημένων δειγμάτων σε μια κατηγορία, σε σχέση με το σύνολο των δειγμάτων που έχουν κατηγοριοποιηθεί σε αυτή.

Για την καλύτερη κατανόηση τους, ας τα δούμε σε σχέση με την μήτρα σύγχυσης ενός δυαδικού προβλήματος, που φαίνεται στον Πίνακα 2.6.

Πίνακας 2.6 - Μήτρα Σύγχυσης ενός δυαδικού προβλήματος

		Προβλεμόμενη κατηγορία	
		1	2
Πραγματική Κατηγορία	1	X_{11}	X_{12}
	2	X_{21}	X_{22}

Η ευαισθησία της κατηγορίας 1 είναι όσα κατηγοριοποιήθηκαν σωστά σε αυτή (x11) ανάλογα με το σύνολο των δειγμάτων αυτής της κατηγορίας (x11 + x21). Πιο γενικά, αν c είναι ο αριθμός των κατηγοριών του προβλήματος:

$$sensitivity \;\; s = \frac{x_{ii}}{\sum_{j=1}^{c} x_{ji}} \qquad \text{(Εξ 2.6)}$$

Αντίστοιχα, η ακρίβεια της κατηγορίας 1 είναι το ποσοστό των σωστά κατηγοριοποιημένων δειγμάτων σε αυτή τη κατηγορία (x22), σε σχέση με το σύνολο των δειγμάτων που έχουν κατηγοριοποιηθεί σε αυτή (x21 + x22). Πιο γενικά, αν c είναι ο αριθμός των κατηγοριών του προβλήματος:

$$precision \;\; p = \frac{x_{ii}}{\sum_{j=1}^{c} x_{ij}} \qquad \text{(Εξ 2.7)}$$

Για να αναλυθεί η απόδοση ενός κατηγοριοποιητή στο σύνολο του προβλήματος, και όχι απλά σε μια κατηγορία, μαζί με το συνολικό ποσοστό των επιτυχημένων κατηγοριοποιήσεων, παίρνονται υπόψιν και ο μέσος όρος της ευαισθησίας και της ακρίβειας όλων των κατηγοριών.

Η κάθε κατηγορία, ανεξάρτητα του πλήθους των εγγραφών που έχει, προσμετράται στον ίδιο βαθμό στην διαμόρφωση του μέσου όρου. Με αυτόν τον τρόπο, σε προβλήματα που η σπάνια κατηγορία είναι εξίσου σημαντική, η και περισσότερο, από την συχνή κατηγορία, η κατηγοριοποίηση δειγμάτων όλων των κατηγοριών έχει τον ίδιο ρόλο στην διαμόρφωση της ανάλυσης της απόδοσης ενός αλγόριθμου κατηγοριοποίησης.

3 *Σχετικές εργασίες*

Σε αυτό το κεφάλαιο θα αναφερθούμε σύντομα στις σχετικές δημοσιευμένες εργασίες και θα εξηγήσουμε τις διαφορές με τους αλγορίθμους που αναπτύξαμε και προτείνονται στην παρούσα διπλωματική εργασία.

Στην ενότητα 3.1 θα αναφερθούμε στη θεματική περιοχή του πρώτου αλγορίθμου, του Constrained K – Means Classification, δηλαδή στην κατηγοριοποίηση μέσω ομαδοποίησης, αλλά και συνολικά σε προσπάθειες που έχουν γίνει για αξιοποίηση περαιτέρω γνώσης στην χρήση του K – Means για ομαδοποίηση, π.χ. με χρήση περιορισμών κλπ.

Στην ενότητα 3.2 θα αναφερθούμε σε εργασίες σχετικές με τον δεύτερο αλγόριθμο που αναπτύχθηκε, τον Stochastic Forest, δηλαδή σε κατηγοριοποιητές ομάδας που χρησιμοποιούν δέντρα απόφασης, με διαφορετικό τρόπο από τα παραδοσιακά τυχαία δάση (Random Forest).

3.1 Ομαδοποίηση και κατηγοριοποίηση με τον K-Means

Ο όρος “k-means” χρησιμοποιήθικε πρώτα από τον Mc Queen το 1967 [4]. Η αρχική ιδέα του αλγορίθμου, όπως περιγράφηκε στο κεφάλαιο 2, παρουσιάστηκε το

1957 από τον Stuart Lloyd (δημοσιεύτηκε το 1982) [24]. Έκτοτε έχουν προταθεί στη βιβλιογραφία διάφορες βελτιώσεις του, στην προσπάθεια αντιμετώπισης γνωστών αδυναμιών του.

Μία παραλλαγή του K-Means, όπου εισάγονται περιορισμοί που αφορούν το κάθε δεδομένο, είναι ο "COP-KMEANS" [25]. Στον αρχικό αλγόριθμο δίνονται επιπλέον δύο είδη περιορισμών, αυτοί που ορίζουν ότι δύο δεδομένα πρέπει να είναι στην ίδια ομάδα και εκείνοι που ορίζουν ότι κάποια δεδομένα δεν μπορούν να είναι στην ίδια ομάδα. Η ανάθεση του κάθε δεδομένου σε κάποια ομάδα γίνεται δεδομένου ότι κανένας περιορισμός δεν παραβιάζεται.

Στην ίδια λογική αναπτύχθηκε ακόμα μία τεχνική [26], η οποία εισάγει και πάλι περιορισμούς επιπέδου δεδομένου, ορίζοντας νέους τύπους περιορισμών σε συνέχεια του προηγούμενου αλγόριθμου. Στην περίπτωση αυτή, υπάρχουν και πάλι τα δύο είδη περιορισμών που προαναφέρθηκαν, αλλά εξετάζονται και δύο καινούρια τα οποία βασίζονται στην μέγιστη ή ελάχιστη απόσταση που έχουν τα σημεία που βρίσκονται στην ίδια ή διαφορετική ομάδα αντίστοιχα.

Ακόμα ένας αλγόριθμος που εισάγει περιορισμούς αναφέρεται ως "Constrained K-Means" [27]. Εδώ οι περιορισμοί είναι ένας για κάθε ομάδα και χρησιμοποιούν και πάλι γνώση από τα δεδομένα. Αφορούν μία τιμή κατωφλίου για κάθε ομάδα, που καθορίζει τα λιγότερα στοιχεία που μπορεί να περιέχει αυτή. Έτσι, γίνεται προσπάθεια προσδιορισμού των βέλτιστων κέντρων, υπό τους περιορισμούς των ελάχιστων αριθμών δεδομένων σε κάθε ομάδα.

Μια άλλη προσέγγιση, είναι ο αλγόριθμος "global K-Means" [28]. Σύμφωνα με αυτή, ο προσδιορισμός των K κέντρων γίνεται μέσω μιας επαναληπτικής διαδικασίας, όπου ανακαλύπτεται κάθε φορά ένα ακόμα κέντρο. Η διαδικασία ξεκινά λύνοντας το πρόβλημα για ένα κέντρο, και κάθε φορά που εκτελείται προστίθεται στη λύση ένα ακόμα κέντρο. Ουσιαστικά, για να προκύψει η λύση με τα K κέντρα, έχουν προηγουμένως βρεθεί όλες οι λύσεις για K μικρότερο του ζητούμενου.

Έχει προταθεί επίσης μια τροποποίηση του προηγούμενου αλγορίθμου, με στόχο τη μείωση του χρόνου εκτέλεσής του. Αυτός είναι ο "Fast global K-means" [28], ο οποίος αντί να επιλύσει το k-οστό πρόβλημα δεδομένου του k-1 με τον K-Means, εισάγει το νέο κέντρο και υπολογίζει την εγγυημένη μείωση του σφάλματος λόγω της εισαγωγής αυτού του κέντρου. Κατόπιν ορίζει ένα άνω όριο για το σφάλμα που

είναι όσο το προηγούμενο, μείον την εγγυημένη μείωση και εκτελεί τον K-Means με όριο λάθους αυτή την ποσότητα.

Ένας άλλος αλγόριθμος που έχει προταθεί στη βιβλιογραφία είναι ο “Fuzzy K-Means” [29]. Εισάγει ασαφή λογική στην επίλυση του προβλήματος, με αυτό να μεταφράζεται στο ότι κάθε δεδομένο πλέον δεν εντάσσεται αποκλειστικά σε μια ομάδα, αλλά ανήκει σε κάποιο βαθμό σε όλες τις ομάδες. Έτσι, κάθε δεδομένο, ανήκει κατά έναν, υπολογιζόμενο, βαθμό σε κάθε ομάδα και συνεισφέρει ανάλογα με τον κάθε βαθμό στον υπολογισμό όλων των κέντρων λειτουργώντας ως βάρος του σημείου.

Αναφορικά με την κατηγοριοποίηση μέσω ομαδοποίησης, συνήθως παρουσιάζεται χρησιμοποιώντας τους αλγορίθμους “K-Means” και “Fuzzy K-Means” για ομαδοποίηση, αντιστοιχίζοντας τα κέντρα με τις κατηγορίες [30, 31, 32, 33, 34, 35]. Η διαδικασία περιλαμβάνει αυτούσια τη διαδικασία ομαδοποίησης και ακολουθεί μια μέθοδος αντιστοίχησης των ομάδων σε κατηγορίες.

3.2 Κατηγοριοποιητές ομάδων με δέντρα απόφασης

Στην βιβλιογραφία οι βασικοί αλγόριθμοι και μέθοδοι που έχουν προταθεί για κατηγοριοποιητές ομάδων χρησιμοποιώντας δέντρα απόφασης, εκτός των ευρέως γνωστών τυχαίων δασών (Random Forest) [17], είναι οι μέθοδοι Bagging [36] και Boosting [37], ο Random Subspaces [38], ο Rotation Forest [39], και ο Randomized C4.5 ή Random Trees [40]. Ακόμα, έχει προταθεί μια μέθοδος για δημιουργία εντελώς τυχαίων δέντρων (Extremely randomized trees) [41].

Οι μέθοδοι της εμφωλίασης (Bagging) και της ενίσχυσης (Boosting) χρησιμοποιούν διαχείριση του συνόλου εκπαίδευσης για την δημιουργία των N διαφορετικών κατηγοριοποιητών της ομάδας. Ουσιαστικά ο κάθε κατηγοριοποιητής δημιουργείται από διαφορετικό σύνολο εκπαίδευσης.

Η μέθοδος Bagging δειγματοληπτεί επαναληπτικά, με αντικατάσταση, από το αρχικό σύνολο δεδομένων εκπαίδευσης σύμφωνα με μια ομοιόμορφη κατανομή πιθανότητας. Κάθε σύνολο δεδομένων που δημιουργείται έχει το ίδιο μέγεθος με τα αρχικά δεδομένα, αφού η δειγματοληψία πραγματοποιείται με αντικατάσταση. Η κατηγοριοποίηση των εγγραφών γίνεται λαμβάνοντας την ψήφο πλειοψηφίας μεταξύ των προβλέψεων κάθε κατηγοριοποιητή.

Σε αντίθεση με την εμφωλίαση, η μέθοδος της ενίσχυσης (Boosting) αποδίδει κάποιο βάρος σε κάθε δείγμα του συνόλου εκπαίδευσης. Τα βάρη στη μέθοδο της ενίσχυσης μπορούν να αξιοποιηθούν με 2 διαφορετικές προσεγγίσεις. Η πρώτη είναι να χρησιμοποιηθούν ως κατανομή δειγματοληψίας προκειμένου να εξαχθεί ένα σύνολο δεδομένων από τα αρχικά δεδομένα. Η δεύτερη είναι να χρησιμοποιούνται από τον βασικό κατηγοριοποιητή, για την δημιουργία μοντέλου που είναι μεροληπτικό.

Η διαδικασία που ακολουθεί η ενίσχυση είναι επαναληπτική. Αρχικά αποδίδονται ίσα βάρη σε όλα τα δείγματα του συνόλου δεδομένων, ώστε να είναι ίση η πιθανότητα τους να επιλεγούν για εκπαίδευση. Ένας κατηγοριοποιητής επάγεται από έναν από τους παραπάνω τρόπους και χρησιμοποιείται για την κατηγοριοποίηση του συνόλου εκπαίδευσης. Στο τέλος κάθε τέτοιας επανάληψης ανανεώνονται τα βάρη, με τα δείγματα που κατηγοριοποιούνται εσφαλμένα, δηλαδή αυτά που θεωρούνται πιο “δύσκολο” να κατηγοριοποιηθούν, να έχουν αυξημένα βάρη. Η ομάδα κατηγοριοποιητών που δημιουργείται τελικά, είναι η συνάθροιση των κατηγοριοποιητών που κατασκευάστηκαν από κάθε επανάληψη της διαδικασίας της ενίσχυσης. Ο πιο γνωστός αλγόριθμος που χρησιμοποιεί την μέθοδο της ενίσχυσης είναι ο Adaboost [42].

Η προτεινόμενη μέθοδος των Random Subspaces χρησιμοποιεί ένα τυχαίο υποσύνολο των χαρακτηριστικών του συνόλου δεδομένων για την κατασκευή κάθε δέντρου απόφασης. Αυτό το υποσύνολο συνήθως είναι τα μισά χαρακτηριστικά του συνόλου εκπαίδευσης. Για την κατασκευή κάθε κατηγοριοποιητή χρησιμοποιούνται όλες οι εγγραφές του συνόλου εκπαίδευσης, χωρίς να χρησιμοποιείται κάποια μέθοδος εμφωλίασης ή ενίσχυσης. Η επαγωγή του κάθε δέντρου απόφασης στην συνέχεια πραγματοποιείται με τον τρόπο που περιγράφηκε στο Κεφάλαιο 2.

Ο αλγόριθμος Random Forest, που αναλύθηκε στο Κεφάλαιο 2, ουσιαστικά χρησιμοποιεί συνδυασμένα τις μεθόδους bagging και Random Subspaces, για την δημιουργία των N διαφορετικών κατηγοριοποιητών.

Όπως αναφέρει ο Ho [38], οι παραπάνω μέθοδοι που χρησιμοποιούν ένα υποσύνολο των χαρακτηριστικών του συνόλου δεδομένων έχουν καλά αποτελέσματα όταν το σύνολο δεδομένων έχει μεγάλο αριθμό από χαρακτηριστικά, ενώ η απόδοσή τους δεν είναι τόσο καλή όταν ο αριθμός των χαρακτηριστικών δεν είναι μεγάλος,

ενώ ταυτόχρονα είναι μικρός και ο αριθμός των δειγμάτων του συνόλου ή είναι μεγάλος ο αριθμός των ετικετών κατηγορίας.

Ο αλγόριθμος Rotation Forest χρησιμοποιεί και αυτός εξαγωγή χαρακτηριστικών, όπως και ο Random Forest και ο Random Subspaces, επιλογή υποσυνόλου τους, εισάγοντας ταυτόχρονα και διαδικασίες Principal Component Analysis (PCA) [43, 44], εξάγοντας τελικά ένα νέο χώρο χαρακτηριστικών. Ακόμα πραγματοποιείται και περιστροφή των χαρακτηριστικών. Με αυτούς τους τρόπους επιτυγχάνεται η δημιουργία αρκετά διαφορετικών κατηγοριοποιητών, που ταυτόχρονα είναι και “δυνατοί”.

Η τεχνική εισαγωγής τυχαιότητας στην κατασκευή των Ν διαφορετικών δέντρων απόφασης, που προτείνεται με τον Randomized C4.5 ή αλλιώς Random Trees, αφορά την εύρεση των 20 καλύτερων διαχωριστικών με το μέτρο της αναλογίας κέρδους, ανάμεσα στα χαρακτηριστικά που έχουν μη μηδενικό κέρδος πληροφορίας, και στην συνέχεια την τυχαία επιλογή από αυτά, με ομοιόμορφη κατανομή πιθανότητας. Για την εύρεση των 20 καλύτερων διαχωριστικών, σε περίπτωση που υπάρχουν αριθμητικά, συνεχή χαρακτηριστικά, λαμβάνονται υπόψιν όλοι οι πιθανοί διαχωρισμοί στο χαρακτηριστικό που έχουν μη αρνητικό κέρδος πληροφορίας. Έτσι, τα 20 καλύτερα διαχωριστικά δεν είναι ανάγκη να είναι από 20 διαφορετικά χαρακτηριστικά, αλλά μπορεί ένα συνεχές χαρακτηριστικό να συμμετέχει πάνω από μία φορά με διαφορετικά σημεία διαχωρισμού.

Μια συγκριτική μελέτη μεταξύ των Random Subspaces, Random Forest, Random Trees και Adaboost μπορεί να βρεθεί στο [45].

Τέλος, οι Pierre Geurts, Damien Ernst, Louis Wehenkel προτείνουν τα εντελώς τυχαία δέντρα (Extremely randomized trees) όπου η επαγωγή των Ν διαφορετικών δέντρων απόφασης πραγματοποιείται εφαρμόζοντας τυχαιότητα, μερικώς ή εντελώς, στην επιλογή των διαχωριστικών σε κάθε κόμβο, ενώ το ίδιο ισχύει για την επιλογή των σημείων διαχωρισμού για τα συνεχή χαρακτηριστικά. Επιπλέον, η προσέγγιση αυτή χρησιμοποιεί ολόκληρο το σύνολο εκπαίδευσης, χωρίς κάποιου είδους προσαρμογή. Η εντελώς τυχαία επαγωγή των δέντρων απόφασης, όπως υποστηρίζουν, μπορεί να οδηγήσει σε πολύ διαφορετικούς κατηγοριοποιητές μειώνοντας το σφάλμα γενίκευσης. Το κύριο αρνητικό είναι πως δεν εξασφαλίζεται πως οι κατηγοριοποιητές θα είναι υψηλής δύναμης.

Ο αλγόριθμος κατηγοριοποιητής ομάδας που αναπτύξαμε στα πλαίσια αυτής της διπλωματικής, ο Stochastic Forest, δεν χρησιμοποιεί καμία μέθοδο διαχείρισης του συνόλου δεδομένων για την δημιουργία των διαφορετικών κατηγοριοποιητών. Δεν χρησιμοποιείται δηλαδή δειγματοληψία, αντιθέτως χρησιμοποιείται όλο το σύνολο δεδομένων εκπαίδευσης, με όλα του τα χαρακτηριστικά για την δημιουργία κάθε κατηγοριοποιητή. Αυτή είναι και η βασική διαφορά με την πλειοψηφία των παραπάνω μεθόδων και αλγορίθμων που έχουν προταθεί στην βιβλιογραφία.

Ο Stochastic Forest χρησιμοποιεί την τεχνική της διαχείρισης του αλγορίθμου εκπαίδευσης για την δημιουργία των κατηγοριοποιητών του. Από τους αλγορίθμους που παρουσιάστηκαν παραπάνω, μόνο οι Randomized C4.5 και Extremely randomized trees δημιουργούν τους κατηγοριοποιητές τους με αυτόν τον τρόπο. Ο αλγόριθμος που αναπτύξαμε στοχεύει σε δημιουργία πιο “δυνατών” κατηγοριοποιητών, γι’ αυτό αν και εισάγει τυχαιότητα, χρησιμοποιεί μόνο τα καλύτερα χαρακτηριστικά με βάση το κέρδος πληροφορίας, σε αντίθεση με τον Extremely randomized trees, με κάποιο βάρος για την επιλογή του κάθε χαρακτηριστικού από αυτά, σε αντίθεση με τον Randomized C4.5.

4 *Constrained K – Means*

Classification

Η κατηγοριοποίηση μέσω ομαδοποίησης είναι μία τεχνική κατηγοριοποίησης η οποία δύναται να εφαρμόζεται σε σύνολα δεδομένων που αποτελούνται από αριθμητικές τιμές. Βασίζεται στο σχηματισμό ομάδων, μέσω της μεθόδου της ομαδοποίησης, μέσα στην οποία υπάρχουν τα δεδομένα που της αντιστοιχούν. Η διαδικασία της εκπαίδευσης χτίζει ένα πρότυπο για τον διαχωρισμό των ομάδων από τα γνωστά δεδομένα των κατηγοριών, ενώ στη διαδικασία πρόβλεψης το πρότυπο είναι γνωστό. Με αυτόν τον τρόπο ανατίθενται τα δεδομένα στις ομάδες σύμφωνα με κάποιο κριτήριο ομοιότητας.

Ο πιο γνωστός αλγόριθμος ομαδοποίησης που χρησιμοποιείται για αυτήν την τεχνική είναι ο K-means, ο αλγόριθμος Κ-κέντρων. Εδώ για τις ανάγκες τις κατηγοριοποίησης, σύμφωνα με τον κυρίαρχο τρόπο, το Κ είναι ίσο με τον αριθμό των κατηγοριών στο σύνολο δεδομένων και κάθε ομάδα αντιστοιχίζεται με κάποιο τρόπο με κάθε κατηγορία. Τα κέντρα είναι ουσιαστικά το πρότυπο της κάθε ομάδας. Η διαδικασία περιλαμβάνει ομαδοποίηση στο σύνολο των δεδομένων προκειμένου να υπολογιστούν τα Κ κέντρα. Ο αλγόριθμος λειτουργεί επαναληπτικά, αναθέτοντας δεδομένα στις ομάδες και επανυπολογίζοντας τα κέντρα έως ότου τα κέντρα δεν μεταβάλλονται άλλο ή συγκλίνουν. Τότε θεωρείται ότι το πρότυπο της ομάδας έχει πλέον σχηματιστεί. Η αντιστοίχηση στις κατηγορίες γίνεται με καταμέτρηση του αριθμού των στοιχείων κάθε κατηγορίας που περιλαμβάνει κάθε ομάδα. Η πλειοψηφούσα κατηγορία σε μια ομάδα αντιστοιχείται στην ομάδα.

Ο παραπάνω αλγόριθμος, χαρακτηρίζεται από ορισμένες γνωστές αδυναμίες, ευαισθησίες που επηρεάζουν την απόδοσή του αλλά και δημιουργούν σημαντικές αποκλίσεις στο ρυθμό σφάλματος ακόμα και για εκτελέσεις ίδιας εισόδου [46, 47]. Η πρώτη αδυναμία, είναι η τυχαιότητα της αρχικοποίησης. Η όλη διαδικασία του K-means είναι ευάλωτη σε μια κακή αρχικοποίηση των κέντρων. Μια τέτοια κατάσταση, πιθανολογικά δύναται να συμβαίνει πολύ συχνά, με κίνδυνο να δημιουργούνται λάθος ομάδες, καθώς τα κέντρα μπορεί να αρχικοποιηθούν σε σημεία που να τους ανατίθενται δεδομένα από διαφορετικές ομάδες.

Μια ακόμη αδυναμία του αλγορίθμου, είναι κατά τη διαδικασία επανυπολογισμού των κέντρων. Εδώ υπάρχουν δύο καταστάσεις που δημιουργούν ανεπιθύμητες μεταβολές των κέντρων. Η πρώτη είναι στο σύνολο δεδομένων να περιλαμβάνονται έγκυρες τιμές σε μία κατηγορία που χαρακτηρίζουν κάποια εξαιρετική περίπτωση.

Στην περίπτωση αυτή, κάθε τέτοιο δεδομένο “τραβάει” το κέντρο προς μια κατεύθυνση που δεν αντιπροσωπεύει τη χαρακτηριστική περίπτωση για την κατηγορία. Μια τέτοια μετακίνηση είναι ανεπιθύμητη, καθώς στο επόμενο βήμα στην ομάδα αυτή είναι πολύ πιθανό να ανατεθούν δεδομένα άλλων ομάδων. Επίσης μπορεί να οδηγήσει σε ένα μοντέλο που περιγράφει εσφαλμένα κατηγορίες, αδυνατεί να γενικεύσει καλά, με αυτό να μεταφράζεται τόσο σε λανθασμένη εικόνα, όσο και σε λάθη πρόβλεψης.

Η δεύτερη κατάσταση που μπορεί να δημιουργήσει μια όμοια αλυσιδωτή αντίδραση είναι η παρουσία θορύβου στο σύνολο δεδομένων. Η περίπτωση αυτή είναι ακόμη πιο ζημιογόνα καθώς το κέντρο θα μεταβληθεί προς μια κατεύθυνση εντελώς λάθος, που δεν αντιστοιχεί σε πραγματικά δεδομένα της κατηγορίας.

Μια ακόμα αδυναμία που παρατηρήθηκε, είναι στο κριτήριο της ομοιότητας των δεδομένων με την ομάδα. Το συνήθη κριτήρια ομοιότητας, είναι κριτήρια απόστασης, με πιο συχνά χρησιμοποιούμενο αυτό της ευκλείδειας απόστασης [48] και λιγότερο αυτό της απόστασης Mahalanobis [49, 50]. Σε καμία από τις δύο περιπτώσεις, δε λαμβάνεται με κανένα τρόπο υπόψιν το εύρος των τιμών των χαρακτηριστικών. Και τα δύο είναι πρότυπα απόστασης, όπου η απόσταση κλιμακώνεται με συγκεκριμένο τρόπο. Στο πρώτο κριτήριο ένα σημείο βρίσκεται σε ίδια απόσταση από κάποιο άλλο όταν βρίσκεται πάνω στα σημεία ενός κύκλου με κέντρο το πρώτο σημείο, ενώ στο δεύτερο όταν βρίσκεται στην περιφέρεια μιας έλλειψης με κέντρο το πρώτο σημείο.

Έχοντας παρατηρήσει τις παραπάνω αδυναμίες, στα πλαίσια της παρούσας εργασίας γίνεται προσπάθεια περιορισμού τους, μέσω κάποιων τροποποιήσεων σε βήματα του αλγορίθμου και εισαγωγής νέων βημάτων. Αυτά οδήγησαν στη δημιουργία ενός νέου αλγορίθμου, που παρουσιάζει σημαντικά βελτιωμένες επιδόσεις και σταθερότητα. Για να επιτευχθούν αυτά χρειάζεται να αξιοποιηθεί καλύτερα η ήδη υπάρχουσα γνώση. Τα στοιχεία που εισήχθησαν είναι δύο: α) όρια στην αρχικοποίηση και τον επανυπολογισμό των κέντρων και β) εμβαρυμένη απόσταση ως κριτήριο ομοιότητας των ομάδων.

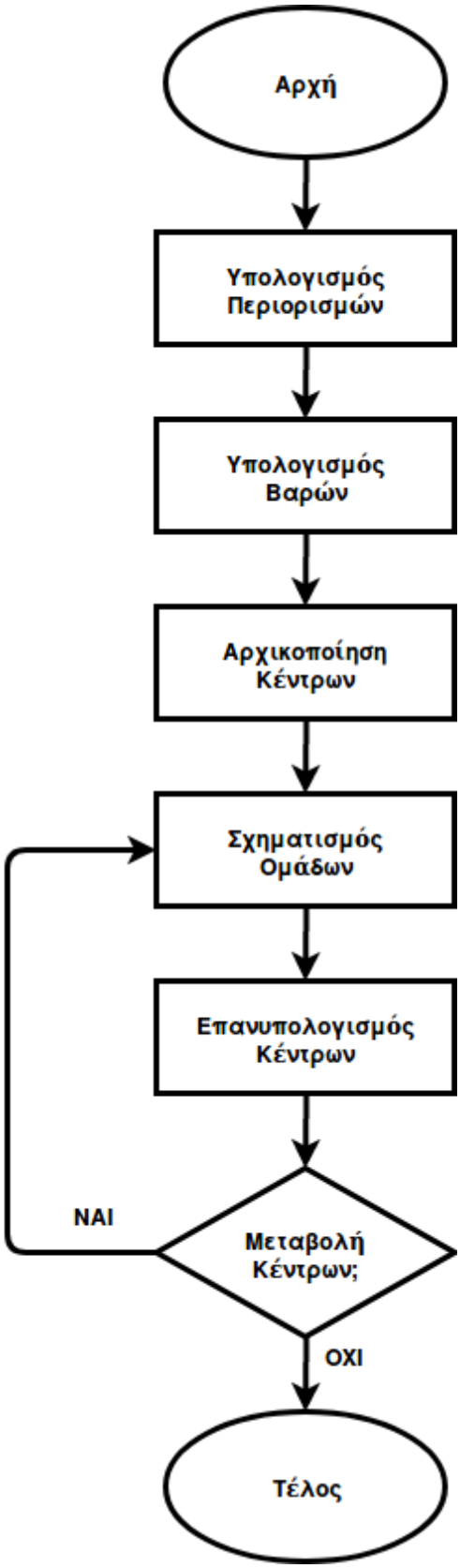
Το παρόν κεφάλαιο αναπτύσσεται ως εξής: Στην ενότητα 4.1 αρχικά δίνεται μια συνοπτική περιγραφή του αλγορίθμου. Κατόπιν, παρουσιάζονται και αιτιολογούνται αναλυτικά τα νέα βήματα που εισήχθησαν και πως αυτά αναμένεται να βελτιστοποιήσουν την αρχική τεχνική. Στην ενότητα 4.2 παρουσιάζονται οι περιορισμοί και στην ενότητα 4.3 τα βάρη. Ακολούθως, στην ενότητα 4.4 παρουσιάζεται η αντιστοίχηση των κατηγοριών στις ομάδες και στην ενότητα 4.5 η κατηγοριοποίηση των νέων εγγραφών.

4.1 Ο αλγόριθμος

Συνοπτικά, ο διαδικασία περιλαμβάνει τα εξής βήματα:

1. Υπολογισμός περιορισμών
2. Υπολογισμός βαρών
3. Αρχικοποίηση κέντρων σύμφωνα με τους περιορισμούς
4. Υπολογισμός της εμβαρυμένης απόστασης του κάθε δεδομένου από κάθε κέντρο
5. Ανάθεση του κάθε δεδομένου στην κατηγορία-ομάδα που απέχει λιγότερο.
6. Επανυπολογισμός Κέντρων
7. Έλεγχος συνθήκης τερματισμού
8. Αν η συνθήκη δεν ικανοποιείται, επιστροφή στο βήμα 4

Στην Εικόνα 4.1 παρουσιάζεται το διάγραμμα ροής του προτεινόμενου αλγόριθμου.



Εικόνα 4.1 – Διάγραμμα ροής του Constrained K – Means Classification

Το παραγόμενο μοντέλο περιλαμβάνει τα εξής χαρακτηριστικά:

- Βάρη χαρακτηριστικών
- Περιορισμοί

4.2 Περιορισμοί

Σε μια προσπάθεια να αντιμετωπιστεί η ευαλωτότητα γενίκευσης του K-means και της ευαισθησίας του στην τυχαιότητα της αρχικοποίησης, προτείνεται η εισαγωγή περιορισμών για τα κέντρα χρησιμοποιώντας κάποια πρότερη γνώση από τα δεδομένα για τις κατηγορίες. Η ιδέα βασίστηκε στην εκτίμηση ότι κάποια γενικά μέτρα (π.χ. μέση τιμή) των χαρακτηριστικών θα μπορούσαν να δώσουν στον αλγόριθμο μια εικόνα της δομής τους, ώστε να λειτουργήσει στοχευμένα και όχι αυθαίρετα. Ταυτόχρονα, η χρήση των περιορισμών έπρεπε να αφήνει αρκετό χώρο στον αλγόριθμο να εκμεταλλευτεί τα δεδομένα για να εξάγει τα κέντρα καλύτερα. Με λίγα λόγια στόχος ήταν να υπάρξει στοχευμένη μάθηση και όχι περιορισμένη.

Έστω μερικά σημεία, των οποίων ζητείται ένα κέντρο. Ένας άνθρωπος, ευφυώς και διαισθητικά, δε θα έβαζε ποτέ το κέντρο σε κάποιο σημείο πολύ μακριά από τα σημεία, πριν ακόμα προβεί σε ενδεχόμενους υπολογισμούς για να το τοποθετήσει ακριβώς. Κατ’ αναλογία συνεπώς, είναι θεμιτό κάτι τέτοιο να κάνει και ο αλγόριθμος. Χρησιμοποιούνται λοιπόν κάποια μέτρα θέσης και διασποράς στη διαδικασία εκπαίδευσης, προκειμένου να οριστεί μια αποδεκτή περιοχή κέντρου.

Αυτό χρησιμοποιήθηκε στη συνέχεια, για να έχει μια “αίσθηση” ο αλγόριθμος ότι αυτό που υπολογίζει είναι ένα αποδεκτό κέντρο, έχοντας μια εκτίμηση του πότε αυτό ξεφεύγει πολύ από το πως θα έπρεπε να είναι. Οι περιορισμοί δεν αφήνουν το κέντρο να μετακινηθεί πάρα πολύ από λίγα δεδομένα της κατηγορίας τα οποία μπορεί να είναι εξαιρετικά απομακρυσμένα, ή από δεδομένα-θόρυβο με τέτοια μορφή. Η αρχική εκτίμηση του αλγορίθμου θα τα συμπεριλάβει και αυτά, επηρεάζοντας κατά ένα περιορισμένο παράγοντα την “αίσθηση” του για το που μπορεί να βρίσκεται το κέντρο, αλλά ο υπολογισμός θα είναι περιορισμένος στα πλαίσια ενός συνολικού κριτηρίου για τα δεδομένα.

Για το σχηματισμό αυτής της εικόνας, επιλέχθηκαν κάποια μέτρα θέσης και διασποράς των χαρακτηριστικών κάθε κατηγορίας, ως μέρη στην εξίσωση των

περιορισμών. Η μέση τιμή είναι το μέτρο για την εκτίμηση των συνιστωσών των κέντρων. Το μέτρο αυτό αντικατοπτρίζει μια αρκετά κοινότυπη-διαισθητική απάντηση στο ερώτημα της τοποθέτησης του κέντρου, που λέει ότι αυτό βρίσκεται στη μέση. Για τις ανάγκες ευελιξίας της επακόλουθης διαδικασίας και τον ορισμό της περιοχής για το αποδεκτό κέντρο, χρησιμοποιήθηκε επίσης η τυπική απόκλιση για να βοηθήσει τον αλγόριθμο να έχει μια εκτίμηση του πόσο μακριά από τη μέση τιμή είναι συνήθως τα χαρακτηριστικά.

Ακολούθως, εισήχθη ένας ακόμη παράγοντας στην εξίσωση, ο οποίος καθορίζει την αυστηρότητα των περιορισμών. Είναι ένας όρος που πολλαπλασιαζόμενος με την τυπική απόκλιση του χαρακτηριστικού δίνει την επιτρεπτή απόσταση μετακίνησης του κέντρου σε αυτή τη διάσταση. Ο παράγοντας αυτός είναι καθοριζόμενος από τον προγραμματιστή ανάλογα με το πόσο αυστηρούς περιορισμούς θέλει να έχει. Συνοψίζοντας, τα παραπάνω οδήγησαν σε δύο εξισώσεις για κάθε χαρακτηριστικό τις μέγιστες και ελάχιστες τιμές του όπως παρακάτω:

$$b_{a,c}^{min} = m_{a,c} - l * std_{a,c} \quad \text{(Εξ 4.1)}$$

$$b_{a,c}^{max} = m_{a,c} + l * std_{a,c} \quad \text{(Εξ 4.2)}$$

όπου b είναι ο περιορισμός, α το χαρακτηριστικό, c η κατηγορία, m η μέση τιμή, s η τυπική απόκλιση και l ο παράγοντας αυστηρότητας.

Όλοι μαζί οι περιορισμοί στις ν διαστάσεις ορίζουν έναν υπερκύβο στο επίπεδο που είναι η περιοχή επιτρεπτών μεταβολών για το κάθε κέντρο. Έτσι κατά τον επανυπολογισμό του κέντρου, εκείνο κινείται στην κατεύθυνση που επιβάλλουν τα δεδομένα, με μια καθορισμένη ελευθερία στο πόσο μακρυά μπορεί να πάει κατά κατεύθυνση, σύμφωνα με τη ν εικόνα που έχει αποκτήσει ο αλγόριθμος για τη δομή των χαρακτηριστικών. Όταν ο επανακαθορισμός ενός κέντρου ξεφεύγει από τα προβλεπόμενα όρια, το κέντρο μετακινείται μόνο μέχρι τα όρια, λαμβάνοντας υπόψιν ότι υπάρχουν δεδομένα προς εκείνη την κατεύθυνση, χωρίς να ξεφύγει από την εκτίμησή του για την περιοχή που βρίσκεται το κέντρο.

4.3 Βάρη

Σε μια προσπάθεια ανεύρεσης ενός πιο αποδοτικού κριτηρίου ομοιότητας δεδομένων με τις ομάδες, διερευνήθηκε η χρήση ενός κριτηρίου το οποίο θα

αντλούσε και αυτό κάποια γνώση από τα δεδομένα προκειμένου να μετρά την ομοιότητα με πιο ευέλικτο τρόπο. Αρχικά, παρατηρήθηκε ότι τόσο στο μέτρο της ευκλείδειας απόστασης, όσο και σε αυτό της Mahalanobis, τα χαρακτηριστικά ως παράγοντες έχουν ένα συγκεκριμένο, καθολικά καθορισμένο βάρος στην εξίσωση από το πρότυπο της εκάστοτε απόστασης.

Η ιδέα ήταν ότι σε ένα σύνολο δεδομένων πολλά χαρακτηριστικά ενδέχεται να είναι σε πολύ διαφορετικές μεταξύ τους τάξεις μεγέθους. Έτσι, αναλογικά μικρές μεταβολές σε κάποιο χαρακτηριστικό μεγάλης τάξης μεγέθους ενδέχεται να μεταφράζονται σε μεγάλες μεταβολές της απόστασης και αντίστροφα. Θεωρήθηκε λοιπόν χρήσιμο να μελετηθεί η εισαγωγή βαρών στα χαρακτηριστικά, τα οποία κατά τον υπολογισμό της απόστασης θα περιορίζουν κατά ένα παράγοντα τα χαρακτηριστικά με ποσοτικά, αλλά όχι αναλογικά μεγάλες μεταβολές, ενώ θα ενισχύουν τα χαρακτηριστικά με τις αναλογικά αλλά όχι ποσοτικά μεγάλες μεταβολές. Στην προσπάθεια αυτή, ερευνήθηκαν δύο κατευθύνσεις. Η πρώτη, χρησιμοποιεί την τυπική απόκλιση ως εκτίμηση της μεταβολής ενός χαρακτηριστικού, ενώ η δεύτερη αξιολογεί τα χαρακτηριστικά χρησιμοποιώντας την εντροπία και το κέρδος πληροφορίας.

Στην πρώτη κατεύθυνση, η επιδίωξη ήταν ποσοτικά και όχι αναλογικά μεγάλες μεταβολές του χαρακτηριστικού να οδηγούν σε ένα μικρό βάρος για αυτό και αντίστροφα. Στη συνέχεια, δύναται να υπάρχει κανονικοποίηση των υπολογισμένων βαρών, προκειμένου το κάθε χαρακτηριστικό να αντικατοπτρίζει ένα μέρος της απόστασης, ανάλογο της τιμής του βάρους του. Αλλιώς το βάρος μπορεί απλά να αυξομειώνει τις τιμές των χαρακτηριστικών ανάλογα με την τιμή του στην εξίσωση της απόστασης.

Η τυπική απόκλιση είναι ένα αξιόπιστο μέτρο της κατά μέσο όρο ποσοτικής μεταβολής των χαρακτηριστικών από τη μέση τιμή. Για το σκοπό που παρουσιάστηκε παραπάνω, η τυπική απόκλιση διαιρείται με τη μέγιστη τιμή που παρουσιάζεται στο χαρακτηριστικό στην εκάστοτε κατηγορία, κάνοντας μια ποσοτικά μεγάλη μεταβολή μικρή, αλλά μια ποσοτικά μικρή απειροελάχιστη. Το βάρος κατόπιν προκύπτει από τον τύπο:

$$w_{a,c} = 1 - std_{a,c} / max_{a,c} \qquad \text{(Εξ 4.1)}$$

όπου α το χαρακτηριστικό, c η κατηγορία, w το βάρος του και s η τυπική απόκλισή του. Με την αφαίρεση από το 1 επιτυγχάνονται μεγάλα βάρη για τις ποσοτικά πολύ μικρές μεταβολές και αρκετά μικρότερα για τις ποσοτικά μικρές. Τελικά, στον αλγόριθμο προκύπτουν α*c τον αριθμό βάρη, που αντιστοιχούν σε κάθε χαρακτηριστικό κάθε κατηγορίας, το οποία χρησιμοποιούνται για τη μετατροπή της ευκλείδειας απόστασης σε εμβαρυμένη ευκλείδεια απόσταση.

Η δεύτερη κατεύθυνση που δοκιμάστηκε ήταν εκείνη των καθορισμένων από το κέρδος πληροφορίας βαρών για κάθε χαρακτηριστικό. Εδώ τα βάρη είναι ένα για κάθε χαρακτηριστικό κάθε κατηγορίας. Αρχικά γίνεται ο υπολογισμός του κέρδους πληροφορίας κάθε χαρακτηριστικού κάθε κατηγορίας, το οποίο αποτελεί το βάρος του στην εξίσωση της απόστασης. Το βάρος αυτό πολλαπλασιάζει την τιμή του χαρακτηριστικού, ενισχύοντας χαρακτηριστικά κατά κατηγορία με μεγάλο κέρδος πληροφορίας. Ο υπολογισμός δηλαδή της απόστασης από κάθε κέντρο είναι μια διαφορετική εξίσωση, αναλόγως των βαρών. Από τις δύο προσεγγίσεις, πειραματικά και εκ του αποτελέσματος προτιμήθηκε η πρώτη, καθώς μόνο αυτή απέφερε βελτίωση στα ποσοστά επιτυχίας του αλγορίθμου.

4.4 Αντιστοίχιση των κατηγοριών στα κέντρα

Στην κλασσική τεχνική κατηγοριοποίησης μέσω ομαδοποίησης, μετά το σχηματισμό των ομάδων ακολουθεί καταμέτρηση των στοιχείων των κατηγοριών σε κάθε ομάδα και η πλειοψηφούσα κατηγορία επικρατεί. Δεδομένης της εισαγωγής των περιορισμών στον αλγόριθμο, το βήμα αυτό απαλείφεται. Η πρότερη γνώση που χρησιμοποιείται για την εξαγωγή των περιορισμών είναι εξειδικευμένη στα χαρακτηριστικά κάθε κατηγορίας. Αυτό συνεπάγεται ότι η αντιστοιχία ομάδων-κατηγοριών είναι από εκείνη τη στιγμή και μετά γνωστή, αφού οι περιορισμοί “κουβαλούν” γνώση που αναφέρεται στην εκάστοτε κατηγορία σε κάποια από τις ομάδες.

4.5 Κατηγοριοποίηση

Όταν πλέον το μοντέλο είναι έτοιμο, περιλαμβάνει τους περιορισμούς και τα βάρη. Για να προχωρήσει ο αλγόριθμος στην κατηγοριοποίηση νέων δεδομένων, ξεκινώντας, τα κέντρα αρχικοποιούνται τυχαία μέσα στον υπερκύβο των

περιορισμών. Η διαδικασία συνεχίζει με επαναληπτική ανάθεση των δεδομένων στις ομάδες και επανυπολογισμό των κέντρων έως ότου υπάρξει σύγκλιση των κέντρων.

Η διαφορά στις αναθέσεις δεδομένων, σε σχέση με τον παραδοσιακό τρόπο για κατηγοριοποίηση μέσω ομαδοποίησης, είναι ότι το κριτήριο ομοιότητας με την ομάδα είναι η εμβαρυμένη ευκλείδεια απόσταση από το κέντρο, με τα βάρη όπως υπολογίστηκαν κατά τη διαδικασία εκπαίδευσης. Ακολούθως, τα νέα κέντρα υπολογίζονται και ελέγχεται η συμβατότητά τους με τους περιορισμούς. Όταν ένα κέντρο έχει υπολογιστεί έξω από τα όρια, τότε παίρνει την τιμή των ορίων, προς την κατεύθυνση του υπολογισμένου κέντρου. Αλλιώς ισχύει η υπολογισμένη τιμή. Η επαναληπτική διαδικασία σταματάει όταν τα κέντρα συγκλίνουν. Η τελική κατηγορία των εγγραφών είναι αυτή της τελευταίας ομάδας στην οποία ανατέθηκαν.

5 *Stochastic Forest*

Ο δεύτερος αλγόριθμος που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας είναι ένας αλγόριθμος κατηγοριοποίησης με την μέθοδο των ομάδων (ensemble method). Ακολουθώντας την λογική των τυχαίων δασών, που αναλύθηκαν παραπάνω, λειτουργεί με δέντρα απόφασης, κατασκευάζοντας έναν πλήθος από Ν δέντρα με βάση το σύνολο εκπαίδευσης, με τέτοιο τρόπο ώστε τα δέντρα να είναι διαφορετικά και το σφάλμα γενίκευσης να είναι περιορισμένο. Στον αλγόριθμο που αναπτύχθηκε, όλα τα δέντρα παράγονται από ολόκληρο το σύνολο εκπαίδευσης, χωρίς να χρησιμοποιείται κάποια τεχνική εμφωλίασης ή ενίσχυσης στο σύνολο δεδομένων.

Η κατηγοριοποίηση των μελλοντικών δειγμάτων γίνεται ξεχωριστά από το κάθε δέντρο και στο τέλος η κατηγορία η οποία έχει λάβει τις περισσότερες “ψήφους” από το σύνολο των δέντρων απόφασης, είναι και η τελική προβλεπόμενη κατηγορία του δείγματος. Η διαφορά με τον Random Forest, αλλά και άλλους αντίστοιχους αλγορίθμους που αναφέρθηκαν, βρίσκεται στον τρόπο παραγωγής των Ν διαφορετικών δέντρων. Στον αλγόριθμο που αναπτύχθηκε γίνεται με στοχαστικό τρόπο, όπως θα αναλυθεί παρακάτω. Γι’ αυτόν τον λόγο τον ονομάσαμε **Stochastic Forest**. Ο αλγόριθμος ουσιαστικά χρησιμοποιεί την τέταρτη τεχνική δημιουργίας κατηγοριοποιητών ομάδων που αναλύθηκε στο κεφάλαιο 2, δηλαδή την διαχείριση του αλγορίθμου εκπαίδευσης και όχι του συνόλου εκπαίδευσης.

Η δομή του κεφαλαίου έχει ως εξής: Στην ενότητα 5.1 θα παρουσιαστούν κάποια βασικά ζητήματα για τον αλγόριθμο που αναπτύχθηκε. Στην ενότητα 5.2 αναλύεται ο τρόπος κατασκευής των δέντρων και κάποια πιο συγκεκριμένα ζητήματα για τον C4.5 [21], τον αλγόριθμο κατασκευής δέντρων απόφασης που βασίζεται η κατασκευή των δέντρων στον αλγόριθμο που αναπτύχθηκε. Στην ενότητα 5.3 παρουσιάζεται ο τρόπος με τον οποίο πραγματοποιείται η κατηγοριοποίηση με τον προτεινόμενο αλγόριθμο, μετά την εκπαίδευσή του.

5.1 Ο αλγόριθμος

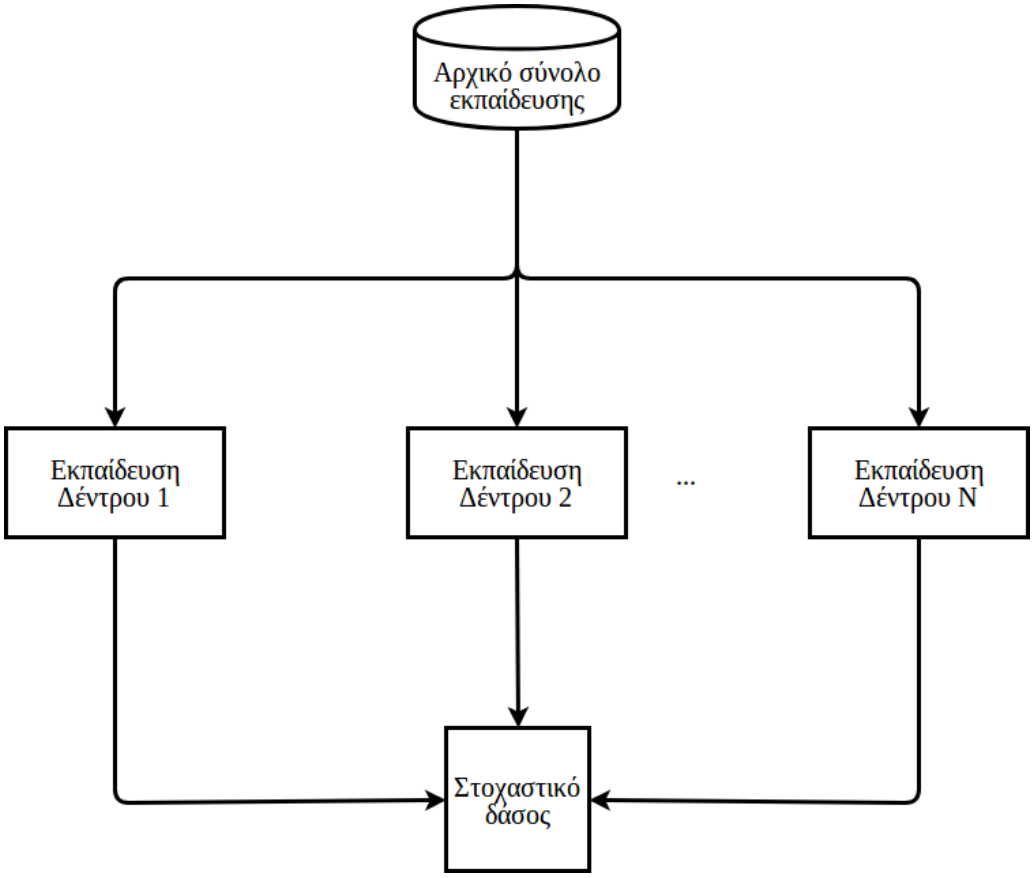
Όπως αναφέρθηκε και πριν ο αλγόριθμος λειτουργεί παρόμοια με τα τυχαία δάση, χωρίς όμως να περιέχει το βήμα της κατασκευής πολλών διαφορετικών συνόλων εκπαίδευσης, με διαχείριση των γνωρισμάτων εισόδου, με τα οποία κατασκευάζονται τα ξεχωριστά δέντρα απόφασης.

Στην αρχή κατασκευάζονται τα δέντρα, με το ίδιο σύνολο εκπαίδευσης, και στην συνέχεια κατηγοριοποιούν τα “άγνωστα δεδομένα” με την μέθοδο των “ψήφων”. Αυτή η διαδικασία φαίνεται στην Εικόνα 5.1.

Στην ανάπτυξη αλγορίθμου κατηγοριοποιητή ομάδων που χρησιμοποιεί δέντρα απόφασης, υπάρχουν αρκετά ζητήματα που χρειάζεται να απαντηθούν. Κάποια από αυτά είναι αν θα αξιοποιηθεί ολόκληρο το σύνολο εκπαίδευσης στην κατασκευή όλων των δέντρων απόφασης, ή θα γίνει διαχείριση των δεδομένων εκπαίδευσης για την ενίσχυση της διαφορετικότητας μεταξύ των δέντρων. Επίσης αν θα αξιοποιηθούν όλα τα γνωρίσματα στην κατασκευή του κάθε δέντρου ή κάποια τυχαία επιλεγμένα, ή διαφορετικά κάποια επιλεγμένα με κάποιο συγκεκριμένο τρόπο. Κάποια ακόμα ζητήματα σχεδίασης είναι στο πως θα κατασκευαστούν τα δέντρα απόφασης, πότε να σταματάνε την ανάπτυξή τους κλπ.

Για την ανάπτυξη του νέου αλγορίθμου, λήφθηκε υπόψιν πως τα δέντρα απόφασης προσφέρουν την δυνατότητα να αναπτύξουν διαφορετικά μοντέλα με εισαγωγή τυχαιότητας ή στοχαστικότητας. Συνεπώς, αποφασίσαμε να χρησιμοποιείται ολόκληρο το σύνολο εκπαίδευσης για να αξιοποιείται όλη η πληροφορία στον μέγιστο βαθμό. Ακόμα, επιλέχθηκε να αξιοποιούνται όλα τα γνωρίσματα, γιατί π.χ. σε κάποιο πρόβλημα με πληθώρα πλεονάζων γνωρισμάτων, αν κάθε δέντρο αξιοποιεί ένα υποσύνολο των συνολικών γνωρισμάτων του συνόλου

δεδομένων, είναι εξαιρετικά πιθανό να μην αξιοποιήσει τα πιο σημαντικά γνωρίσματα.



Εικόνα 5.1 – Δημιουργία του στοχαστικού δάσους

Με αυτόν τον τρόπο ο αλγόριθμος εκμεταλλεύεται σε μεγάλο βαθμό ολόκληρη την υπάρχουσα γνώση και στηρίζεται στα θετικά στοιχεία των δέντρων απόφασης. Αντίστοιχα δίνεται η δυνατότητα μέσω κάποιων πιθανοτήτων, ανάλογα με το κέρδος πληροφορίας (information gain) και την αναλογία κέρδους (gain ratio) του κάθε χαρακτηριστικού, να καταλήξει στην κατασκευή κάποιων δέντρων, αξιοποιώντας χαρακτηριστικά με καλό κέρδος πληροφορίας αλλά όχι αναγκαστικά την καλύτερη αναλογία κέρδους.

Με αυτή τη μέθοδο, εκτιμήθηκε πως μπορεί να οδηγηθεί σε καλύτερα αποτελέσματα στην συνέχεια της κατασκευής του δέντρου. Επιπλέον, επιτυγχάνεται να δημιουργηθούν δέντρα με αρκετούς, διαφορετικούς, “καλούς” τρόπους , ώστε να ψηφίζουν για την κατηγοριοποίηση μόνο δέντρα με ικανοποιητική “δύναμη”.

Ένα μειονέκτημα του αλγόριθμου Stochastic Forest, που έγκειται στον παραπάνω τρόπο διαχείρισης των χαρακτηριστικών του συνόλου δεδομένων, είναι αυτό του υψηλού χρόνου εκπαίδευσης του μεγάλου πλήθους των δέντρων. Αυτό γιατί κάθε φορά χρειάζεται να κάνει τους υπολογισμούς που απαιτούνται παίρνοντας υπόψη όλα τα χαρακτηριστικά του συνόλου δεδομένων, όχι μόνο ένα μικρό μέρος όπως ο αλγόριθμος των τυχαίων δασών.

Σε σύνολα δεδομένων με πολύ μεγάλο πλήθος χαρακτηριστικών, το μειονέκτημα αυτό είναι σημαντικό. Αντιθέτως όμως, σε σύνολα δεδομένων με σχετικά λίγα χαρακτηριστικά, κάτω από 8, το μειονέκτημα του χρόνου εκπαίδευσης, σε σύγκριση με αυτόν των τυχαίων δασών, δεν είναι σημαντικό. Η δύναμη όμως των δέντρων που δημιουργούνται είναι σημαντικά ανώτερη, αφού θα εκπαιδευτούν και με όλα τα χαρακτηριστικά και όχι με 1 – 2 που πολύ πιθανό να μην μπορούν να οδηγήσουν σε σωστές αποφάσεις.

5.2 Κατασκευή των Δέντρων Απόφασης

Για την κατασκευή των δέντρων απόφασης έχουν προταθεί αρκετοί αλγόριθμοι, με πιο γνωστούς τους ID3, CART, C4.5. Στον αλγόριθμο που αναπτύξαμε χρησιμοποιήθηκε ως βάση για την κατασκευή των δέντρων ο C4.5, με κάποιες απαραίτητες διαφοροποιήσεις για την εισαγωγή της στοχαστικότητας, ώστε οι κατηγοριοποιητές που θα δημιουργηθούν να είναι διαφορετικοί μεταξύ τους. Όπως και στον βασικό αλγόριθμο των τυχαίων δασών, δεν πραγματοποιείται κάποιο είδος κλαδέματος στο δέντρο που κατασκευάζεται, αντιθέτως αναπτύσσεται στο μεγαλύτερο δυνατό βαθμό, με βάση τα κριτήρια τερματισμού του αλγορίθμου C4.5, που θα αναφερθούν παρακάτω. Αυτό γίνεται γιατί, αφού θα υπάρχει ένα πλήθος από N διαφορετικά δέντρα, φαινόμενα υπερπροσαρμογής θα εξαλειφθούν.

Για την ανάπτυξη του δέντρου απόφασης αρχικά υπολογίζεται η συνολική εντροπία του συνόλου δεδομένων, όπως περιγράφηκε στο κεφάλαιο 2. Στην συνέχεια υπολογίζεται το κέρδος πληροφορίας του κάθε χαρακτηριστικού του. Σε περίπτωση που το χαρακτηριστικό είναι αριθμητικό, συνεχές, χρειάζεται να βρεθεί το καλύτερο σημείο διαχωρισμού του, δηλαδή το σημείο διαχωρισμού που προσφέρει το μεγαλύτερο κέρδος πληροφορίας.

Για να επιτευχθεί αυτό γίνεται αρχικά ταξινόμηση των τιμών του συγκεκριμένου χαρακτηριστικού. Για την ταξινόμηση στην υλοποίηση του αλγορίθμου επιλέχθηκε ο αλγόριθμος Quicksort. Μετά την ταξινόμηση, υπολογίζεται το κέρδος πληροφορίας για κάθε διαφορετική τιμή που υπάρχει στο χαρακτηριστικό. Ακολούθως, βρίσκει τον αριθμό στον οποίο υπάρχει το μεγαλύτερο κέρδος πληροφορίας για αυτό το χαρακτηριστικό. Με βάση την μέθοδο που χρησιμοποιεί ο C4.5, στην συνέχεια υπολογίζεται το threscost, που είναι ίσο με τον λογάριθμο με βάση το 2 των προσπαθειών που έκανε δια το πλήθος των εγγραφών του συνόλου δεδομένων:

$$Threscost = \frac{log_2(tries)}{instances} \qquad \text{(Εξ 5.1)}$$

Αυτός ο αριθμός αφαιρείται από το μέγιστο κέρδος πληροφορίας που βρέθηκε προηγουμένως και έτσι προκύπτει το κέρδος πληροφορίας του συγκεκριμένου χαρακτηριστικού.

Αφού ολοκληρωθεί ο υπολογισμός του κέρδους πληροφορίας για όλα τα χαρακτηριστικά του συνόλου εκπαίδευσης, υπολογίζεται ο μέσος όρος τους. Για όσα χαρακτηριστικά έχουν κέρδος πληροφορίας μεγαλύτερο ή ίσο με τον μέσο όρο υπολογίζεται η αναλογία κέρδους. Με βάση την αναλογία κέρδους του κάθε χαρακτηριστικού, υπολογίζεται η πιθανότητα P(i) για την επιλογή του ως χαρακτηριστικό διαχωρισμού. Ο υπολογισμός της πιθανότητας P(i) γίνεται με τον τύπο:

$$P(i) = \frac{GAIN(i)}{\sum_{j=1}^{A} GAIN(j)} \qquad \text{(Εξ 5.2)}$$

με το A να είναι το πλήθος των χαρακτηριστικών του συνόλου εκπαίδευσης και P(i) η πιθανότητα επιλογής του χαρακτηριστικού i.

Η πιθανότητα επιλογής των χαρακτηριστικών με κέρδος πληροφορίας χαμηλότερο του μέσου όρου είναι μηδενική. Με αυτό το τρόπο επιτυγχάνεται ο σκοπός για την δημιουργία δέντρου απόφασης μόνο από τα καλύτερα χαρακτηριστικά, ώστε να κατασκευαστεί ένα “δυνατό” δέντρο. Εδώ χρειάζεται να τονιστεί πως στις καλύτερες συνθήκες διαχωρισμού παίρνεται υπόψιν μία φορά το κάθε χαρακτηριστικό. Π.χ. αν

σε κάποιο συνεχές χαρακτηριστικό αρκετά σημεία διαχωρισμού έχουν κέρδος πληροφορίας υψηλότερο από τον μέσο όρο, μόνο το καλύτερο θα ληφθεί υπόψιν.

Με βάση τον παραπάνω τρόπο καθορισμού της πιθανότητας επιλογής για κάθε χαρακτηριστικό, αξιοποιείται και το μέτρο του κέρδους πληροφορίας μαζί με το μέτρο της αναλογίας κέρδους. Ακόμα, χαρακτηριστικά που δεν έχουν σημαντικό ρόλο στο σύνολο δεδομένων, άρα έχουν κέρδος πληροφορίας μικρότερο του μέσου όρου, αποκλείονται από την διαδικασία επιλογής του καλύτερου διαχωριστικού. Αυτό συμβαίνει επειδή, βάση του μέτρου του κέρδους πληροφορίας, δεν μπορούν να οδηγήσουν στην κατασκευή ενός δυνατού δέντρου απόφασης. Αντίστοιχα, χαρακτηριστικά που μπορεί να οδηγήσουν σε δυνατό δέντρο απόφασης, ίσως και πιο δυνατό από αυτό που δημιουργείται με το καλύτερο χαρακτηριστικό, αλλά με βάση τον παραδοσιακό τρόπο κατασκευής των δέντρων απόφασης δεν θα επιλεγόντουσαν, με βάση τον παραπάνω τρόπο έχουν πιθανότητες επιλογής.

Αφού υπολογιστεί η πιθανότητα επιλογής σε κάθε χαρακτηριστικό, η κατασκευή του δέντρου προχωράει στην επιλογή του διαχωρισμού. Το σύνολο εκπαίδευσης διαχωρίζεται αναλόγως, με κάθε υποσύνολο να ανατίθεται στα κλαδιά παιδιά του δέντρου. Στην συνέχεια, η διαδικασία κατασκευής νέων κόμβων του δέντρου με επιλογή του χαρακτηριστικού διαχωρισμού ξεκινάει από την αρχή, μέχρι κάποιο από τα κριτήρια τερματισμού, που θα αναφερθούν παρακάτω, γίνει αληθές. Όταν κάποιο κριτήριο τερματισμού ικανοποιηθεί, ο κόμβος μετατρέπεται σε φύλλο του δέντρου και “δείχνει” σε μια κατηγορία. Η κατηγορία στην οποία ”δείχνει” το φύλλο, είναι αυτή στην οποία ανήκουν τα περισσότερα δεδομένα από το υποσύνολο δεδομένων που έχει ανατεθεί σε αυτό το κόμβο.

Η διαδικασία κατασκευής νέων κόμβων – παιδιών στο δέντρο απόφασης τερματίζει με τα ακόλουθα κριτήρια:

- Σε περίπτωση που το υποσύνολο των δεδομένων εκπαίδευσης που έχει ανατεθεί σε έναν κόμβο έχει εγγραφές που ανήκουν σε μία μόνο κατηγορία, δηλαδή η εντροπία του είναι ίση με 0. Ακόμα, σε περίπτωση που η εντροπία είναι σχετικά μικρή και δεν μπορεί να επιτευχθεί από κανένα χαρακτηριστικό του συνόλου δεδομένων κέρδος πληροφορίας μεγαλύτερο του 0.001, ο κόμβος μετατρέπεται σε φύλλο του δέντρου απόφασης.

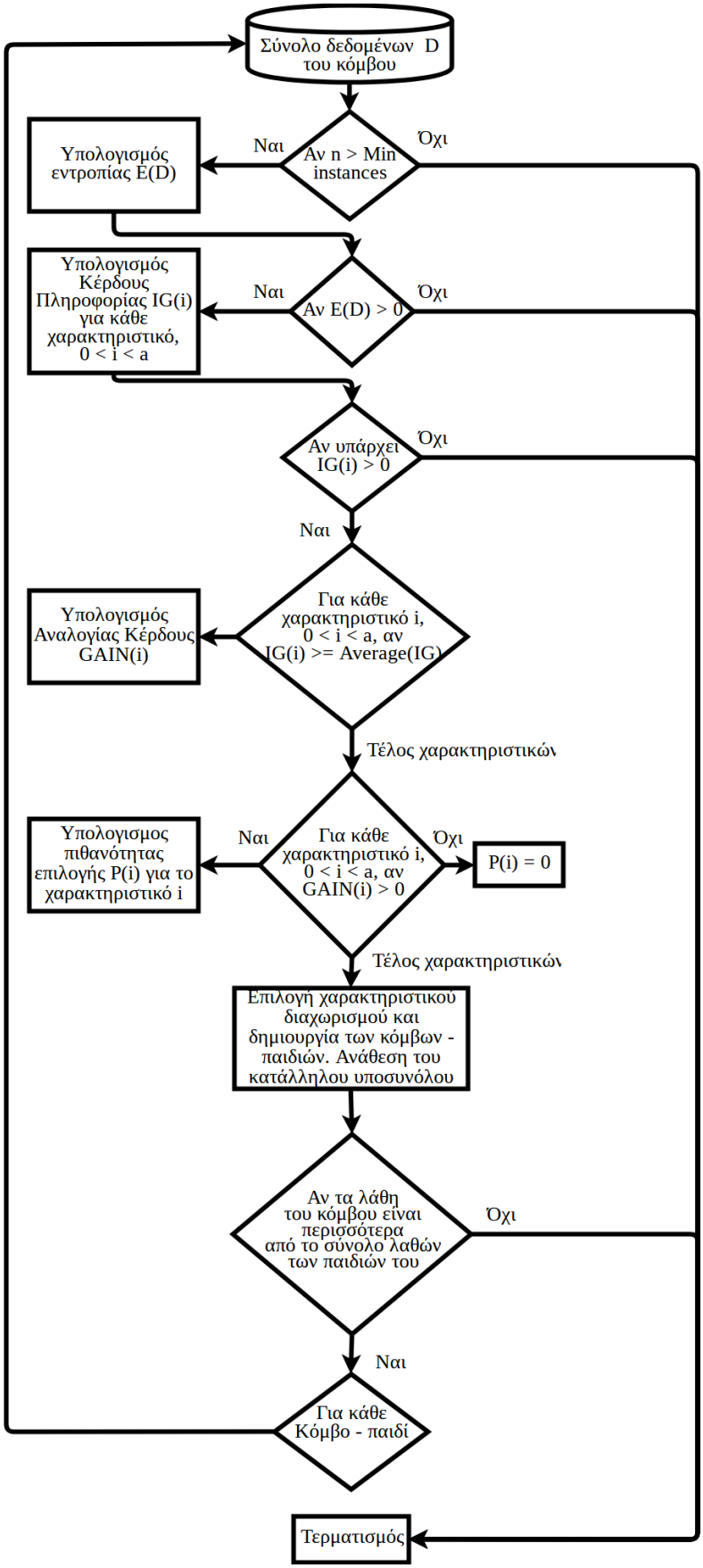
- Για να αποφευχθούν φαινόμενα υπερπροσαρμοστικότητας στα δέντρα, υπολογίζεται στην αρχή της διαδικασίας ένα όριο στο μικρότερο πλήθος εγγραφών (MI) που επιτρέπεται να αντιπροσωπεύει κάθε κόμβος του δέντρου. Αυτό το όριο υπολογίζεται ως εξής:

$$Minimun\ Instances\ \ MI = 0.1 * \frac{X}{c} \qquad (\text{Εξ } 5.3)$$

με το X να είναι το πλήθος των εγγραφών του συνόλου εκπαίδευσης και το c το πλήθος των κατηγοριών του. Ακόμα, με βάση τον C4.5, το όριο MI πρέπει να είναι μεγαλύτερο ή ίσο με το 2 και μικρότερο ή ίσο με το 25.

Με βάση το συγκεκριμένο κριτήριο, αν ένας κόμβος δεν μπορεί να διαχωρίσει τα δεδομένα του σε υποσύνολα που να έχουν αποδεκτό πλήθος εγγραφών, τότε σταματάει την διαδικασία κατασκευής νέων κόμβων, και μετατρέπεται σε φύλλο του δέντρου. Για παράδειγμα, αν το όριο είναι 10 εγγραφές και σε έναν κόμβο έχουν ανατεθεί 15, τότε δεν μπορεί να συνεχίσει το διαχωρισμό. Για να συνεχίσει χρειάζεται τουλάχιστον 20 ώστε να μπορεί να τα σπάσει σε 10 και 10 με δυαδικό διαχωρισμό. Αντίστοιχα στην περίπτωση που θα είχε 20 εγγραφές, δεν μπορεί να επιλεγεί ένα χαρακτηριστικό που δημιουργεί τρεις καινούριους κόμβους – παιδιά. Σε αυτή τη περίπτωση αξιοποιούνται μόνο τα χαρακτηριστικά που οδηγούν σε δυαδικό διαχωρισμό.

- Αναλύοντας το παραπάνω κριτήριο για το μικρότερο πλήθος εγγραφών σε κάθε κόμβο του δέντρου, για κάθε χαρακτηριστικό γίνεται έλεγχος αν ο διαχωρισμός που προκύπτει είναι αποδεκτός. Δηλαδή, μπορεί ένας κόμβος να έχει 100 εγγραφές και το όριο να είναι 10, αλλά ο διαχωρισμός που προκύπτει από ένα χαρακτηριστικό με κατηγορικές τιμές να είναι 95 – 5. Σε αυτή τη περίπτωση το χαρακτηριστικό αυτό δεν παίρνεται υπόψιν. Ακόμα, η αναζήτηση του καλύτερου διαχωρισμού σε ένα συνεχές χαρακτηριστικό δεν λαμβάνει μέρος σε τιμές οι οποίες δεν διαχωρίζουν αποδεκτά το υποσύνολο δεδομένων.



Εικόνα 5.2 - Η διαδικασία κατασκευής των δέντρων απόφασης

- Σε περίπτωση που ο κόμβος πληρεί τις προϋποθέσεις με βάση τα παραπάνω κριτήρια, γίνεται ένας τελευταίος έλεγχος. Με βάση το τελευταίο κριτήριο τερματισμού, υπολογίζονται τα λάθη που θα έκανε ο κόμβος αν μετατρεπόταν σε φύλλο του δέντρου και προχωρούσε στην κατηγοριοποίηση των δεδομένων εκπαίδευσης που του έχουν ανατεθεί. Αντίστοιχα, υπολογίζονται τα λάθη που θα έκαναν οι κόμβοι – παιδιά με βάση το χαρακτηριστικό διαχωρισμού που έχει επιλέξει. Αν ο αριθμός των λαθών του κόμβου είναι μικρότερος ή ίσος με τον αριθμό λαθών των κόμβων – παιδιών του τότε ο διαχωρισμός ακυρώνεται και ο κόμβος μετατρέπεται σε φύλλο το δέντρου.

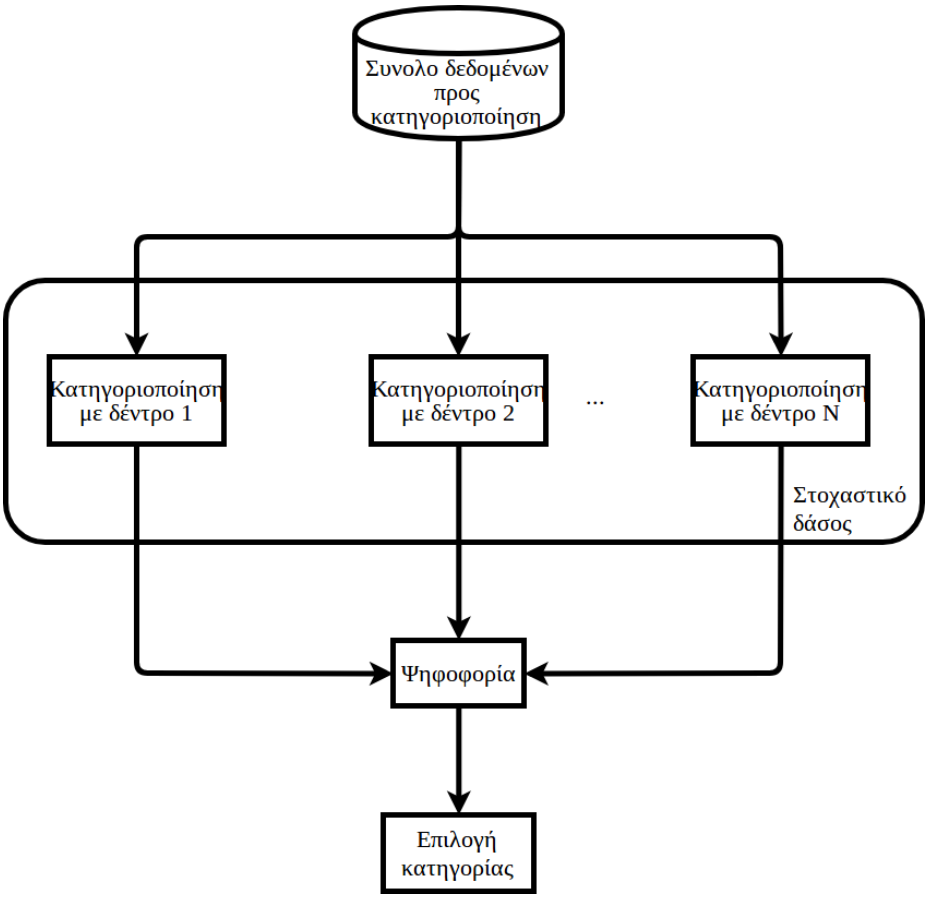
Στην Εικόνα 5.2 αποτυπώνεται η διαδικασία κατασκευής κάθε δέντρου απόφασης για τον αλγόριθμο κατηγοριοποίησης με την μέθοδο ομάδων που αναπτύξαμε. Όπως φαίνεται και από το διάγραμμα η διαδικασία είναι αναδρομική και αφορά την επιλογή αν θα γίνει διαχωρισμός σε αυτόν τον κόμβο ή αν τελικά ο κόμβος θα μετατραπεί σε φύλλο του δέντρου. Ακόμα αναφέρεται στην δημιουργία των παιδιών του κάθε κόμβου. Ο τερματισμός είναι η μετατροπή του κόμβου σε φύλλο και στη συνέχεια ακολουθεί η εύρεση της κατηγορίας στην οποία οδηγεί ο κόμβος – φύλλο.

5.3 Κατηγοριοποίηση με τον Stochastic Forest

Έχοντας εκπαιδεύσει τα δέντρα απόφασης δημιουργώντας έτσι τους διαφορετικούς κατηγοριοποιητές για την ομάδα των κατηγοριοποιητών μας μπορούμε να προχωρήσουμε στην διαδικασία της κατηγοριοποίησης εγγραφών με άγνωστη κατηγορία.

Όπως αναφέρεται και παραπάνω, η κατηγοριοποίηση γίνεται με την μέθοδο των “ψήφων”, ακριβώς όπως και στα τυχαία δάση. Αυτό σημαίνει πως η τελική κατηγορία που επιλέγει ο κατηγοριοποιητής ομάδας είναι αυτή με της περισσότερες ψήφους. Δηλαδή, κάθε νέα εγγραφή χρειάζεται να διασχίσει όλα τα δέντρα του κατηγοριοποιητή και στην συνέχεια να παρθεί η απόφαση. Η διαδικασία αυτή αποτυπώνεται στην Εικόνα 5.3.

Εικόνα 5.3 – Κατηγοριοποίηση με τον Stochastic Forest



6 *Αξιολόγηση και αποτελέσματα.*

Η υλοποίηση και των αλγορίθμων για την αξιολόγησή τους πραγματοποιήθηκε σε C++ [51]. Οι κώδικες της υλοποίησής τους υπάρχουν στο Παράρτημα 1 για τον αλγόριθμο Constrained K – Means Classification, και στο Παράρτημα 2 για τον αλγόριθμο Stochastic Forest.

Η αξιολόγηση των αλγορίθμων που αναπτύχθηκαν στα πλαίσια της παρούσας διπλωματικής πραγματοποιήθηκε σε πολλαπλά σύνολα δεδομένων από το UCI machine learning repository, που είναι το πιο διαδεδομένο αποθετήριο για αξιολόγηση αλγορίθμων εξόρυξης δεδομένων [18].

Στο πλαίσιο αυτό χρησιμοποιήθηκαν, μεταξύ άλλων, τα πιο γνωστά και τα πιο χρησιμοποιημένα σύνολα δεδομένων όπως είναι το iris, τα breast_cancer, το car_evaluation, το Abalone, το heart disease, το ecoli, το ionosphere, το waveform, το wine κ.α. .

Συνολικά χρησιμοποιήθηκαν 40 σύνολα δεδομένων για την αξιολόγηση του κάθε αλγορίθμου ώστε τα αποτελέσματα να είναι πιο αξιόπιστα. Ακόμα, ενισχύοντας τη αξιοπιστία των αποτελεσμάτων, σε κάθε σύνολο δεδομένων εκτελέσθηκε 10 φορές ο κάθε αλγόριθμος και στα αποτελέσματα παρουσιάζεται ο μέσος όρος τους. Αυτό γιατί και οι δύο νέοι αλγόριθμοι παρουσιάζουν μια μικρή απόκλιση. Η διακύμανση αυτή είναι αναπόφευκτη λόγω της ύπαρξης τυχαιότητας, χωρίς αυτό να επηρεάζει αρνητικά την σύγκλιση του αλγορίθμου.

Πίνακας 6.1 - Σύνολα δεδομένων που χρησιμοποιήθηκαν

	Dataset	Attributes	Instances
1	**Abalone**	**8**	**4177**
2	**Balance scale**	**4**	**625**
3	**Banknote**	**4**	**1372**
4	**Blood Transfusion**	**4**	**748**
5	**Breast cancer (wdbc)**	**32**	**569**
6	**Breast cancer (winsconsin)**	**9**	**699**
7	**Breast_tissue**	**9**	**106**
8	**Car evaluation**	**6**	**1728**
9	**Cardiotocography**	**21**	**2126**
10	**Contraceptive**	**9**	**1473**
11	**Dermatology**	**34**	**366**
12	**Ecoli**	**7**	**336**
13	**Fertility Diagnosis**	**9**	**100**
14	**Firm Teacher**	**19**	**10800**
15	**Glass**	**9**	**214**
16	**GPS**	**6**	**163**
17	**Ionosphere**	**34**	**351**
18	**Iris**	**4**	**150**
19	**Messidor features**	**18**	**1151**
20	**Optdigit**	**64**	**1797**
21	**Parkinson disease**	**22**	**195**
22	**Parkinson speech**	**26**	**1239**
23	**Pima indian**	**8**	**768**
24	**Seeds**	**7**	**210**
25	**Semeion**	**255**	**477**
26	**Sonar**	**60**	**208**
27	**Soybean backup large**	**35**	**307**
28	**Spect heart**	**22**	**267**
29	**Spectf heart**	**44**	**267**
30	**Statlog aust credit**	**14**	**690**
31	**Statlog german credit**	**24**	**1000**
32	**Statlog heart**	**13**	**270**
33	**Statlog img seg**	**19**	**810**
34	**Statlog sat image**	**36**	**2000**
35	**Thyroid disease new**	**5**	**215**
36	**Urban**	**147**	**168**
37	**Vertebral column 3C**	**6**	**310**
38	**Waveform**	**21**	**5000**
39	**Wine**	**13**	**178**
40	**Yeast**	**8**	**1484**
41	**Zoo**	**16**	**83**

Όλα τα σύνολα δεδομένων που χρησιμοποιήθηκαν φαίνονται στον Πίνακα 6.1. Στον αριθμό των χαρακτηριστικών (attributes) που παρουσιάζεται, δεν μετριέται η ετικέτα κατηγορίας.

Τα αποτελέσματα των αλγορίθμων σε αυτά τα σύνολα δεδομένων παρουσιάζονται συγκριτικά με τα αποτελέσματα 6 γνωστών αλγορίθμων κατηγοριοποίησης. Αναλυτικά οι αλγόριθμοι αυτοί είναι οι πίνακες απόφασης (Decision Table – DT) [52], ο Naive Bayes (Bayes) [53], ο IBK που είναι υλοποίηση του αλγορίθμου των k κοντινότερων γειτόνων [54], τα νευρωνικά δίκτυα με την αρχιτεκτονική Multi-Layer Perceptron (MLP) [55], τα δέντρα απόφασης με τον J48 που είναι υλοποίηση του αλγόριθμου C4.5 [21] και τέλος τα τυχαία δάση (Random Forest – RF) [17].

Ακόμα, στα πλαίσια της παρούσας διπλωματικής εργασίας, ο αλγόριθμος Constrained K – Means Classification αξιολογήθηκε σε ένα σύνολο δεδομένων με τιμές εικονοστοιχείων οκτώ εικόνων βιοψίας ήπατος και εκτιμήθηκε η δυνατότητα χρήσης του στην εξαγωγή αξιόπιστων συμπερασμάτων στην ιατρική διάγνωση και την εκτίμηση της σοβαρότητας της κίρρωσης του ήπατος μέσα από εικόνες βιοψίας ήπατος.

Τα αποτελέσματα των αλγορίθμων αυτών πάρθηκαν με την χρήση του εργαλείου WEKA [56], που είναι open source. Οι παραπάνω αλγόριθμοι εκτελέσθηκαν στο WEKA με τις προκαθορισμένες τιμές σε όλες τις παραμέτρους που έχουν. Π.χ. ο Random Forest εκτελέστηκε με 100 δέντρα.

Η αξιολόγηση των δύο νέων αλγορίθμων έγινε με την μέθοδο stratified 10 – fold cross – validation, όπως αναλύθηκε στο Κεφάλαιο 2. Η λήψη των αποτελεσμάτων όλων των αλγορίθμων, σε κάθε σύνολο δεδομένων, έγινε με αυτή την τεχνική, για μεγαλύτερη αξιοπιστία. Η διασταυρωμένη επικύρωση είναι μια διαδικασία που αξιοποιείται για την επικύρωση της ορθότητας των αποτελεσμάτων, βοηθώντας στην εξαγωγή ασφαλών συμπερασμάτων για τη συμπεριφορά του κατηγοριοποιητή σε ένα σύνολο δεδομένων.

Η γενική ιδέα ορίζει ότι το σύνολο δεδομένων χωρίζεται σε δέκα ίσα τμήματα και ακολουθεί μια επαναληπτική διαδικασία κατά την οποία κάθε φορά ένα τμήμα χρησιμοποιείται για κατηγοριοποίηση και αξιολόγηση, ενώ τα υπόλοιπα χρησιμοποιούνται για εκπαίδευση του κατηγοριοποιητή.

Τα τμήματα είναι διακριτά και μη επικαλυπτόμενα, έτσι κάθε τμήμα του συνόλου δεδομένων χρησιμοποιείται μία ακριβώς φορά για έλεγχο. Ακόμα, γίνεται προσπάθεια

για τήρηση των αναλογιών των κατηγοριών στα επιμέρους τμήματα. Έχει δηλαδή υπολογιστεί εκ των προτέρων τι ποσοστό των εγγραφών καταλαμβάνει κάθε κατηγορία και διατηρείται στα τμήματα. Αυτό συμβαίνει ώστε τα υποσύνολα που δημιουργούνται να είναι αντιπροσωπευτικές εικόνες του συνόλου δεδομένων.

Στο τέλος της διαδικασίας δημιουργείται η μήτρα σύγχυσης που μας δείχνει για κάθε στοιχείο που κατηγοριοποιήθηκε και που θα έπρεπε να κατηγοριοποιηθεί. Με τη μήτρα σύγχυσης μπορούμε να εξάγουμε όλα τα μέτρα απόδοσης που θέλουμε να χρησιμοποιήσουμε. Τα μέτρα απόδοσης που χρησιμοποιούνται και για τους δύο αλγορίθμους είναι το ποσοστό επιτυχημένων κατηγοριοποιήσεων (accuracy), η ευαισθησία (sensitivity) και η ακρίβεια (precision).

Η δομή του υπόλοιπου Κεφαλαίου έχει ως εξής: Στην ενότητα 6.1 παρουσιάζεται η αξιολόγηση του αλγορίθμου Constrained K – Means Classification. Στην ενότητα 6.2 αναλύεται το ιατρικό πρόβλημα στο οποίο εφαρμόστηκε αυτός ο αλγόριθμος και παρουσιάζει τα αποτελέσματά του σε αυτό. Στην ενότητα 6.3 αναλύεται η αξιολόγηση του αλγορίθμου Stochastic Forest.

6.1 Αξιολόγηση του αλγορίθμου Constrained K – Means Classification

Για την αξιολόγηση του αλγορίθμου ελήφθησαν μετρήσεις του ποσοστού επιτυχημένων κατηγοριοποιήσεων, ευαισθησίας και ακρίβειας, με την πειραματική διαδικασία, όπως περιγράφηκε. Για τον καθορισμό της αυστηρότητας των περιορισμών η παράμετρος l στις εξισώσεις 4.1, 4.2 τέθηκε σε 0.10. Αυτή η τιμή προέκυψε μέσα από την πειραματική διαδικασία. Ακόμα, πριν την εκτέλεση του συγκεκριμένου αλγορίθμου, πραγματοποιείται κανονικοποίηση των δεδομένων στο διάστημα 0 – 100. Τα συγκεντρωτικά αποτελέσματα παρατίθενται σε τρεις πίνακες, 6.2, 6.3, 6.4, έναν για κάθε μετρική στο τέλος του υποκεφαλαίου.

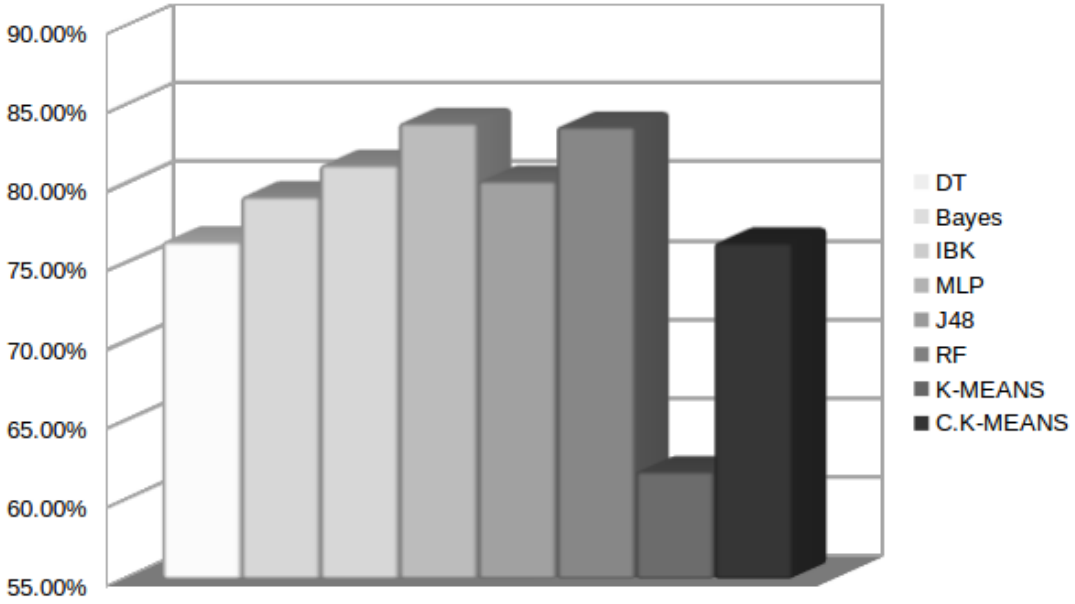
6.1.1 Ποσοστό επιτυχίας

Με μια πρώτη ματιά στον πίνακα του ποσοστού επιτυχίας (Πίνακας 6.2), βλέπουμε ότι η προταθείσα τεχνική επιτυγχάνει μέσο ποσοστό 76.34%. Πρώτος είναι ο αλγόριθμος του πολυεπίπεδου νευρωνικού δικτύου, που επιτυγχάνει ποσοστό

83.91% κυριαρχώντας σε 18 από τα 40 σύνολα δεδομένων. Ο αναπτυχθής αλγόριθμος υπολείπεται του πρώτου κατά 7.58% και σημαντικά λιγότερο από τους υπόλοιπους.

Το υψηλότερο ποσοστό που καταγράφεται για τον αλγόριθμο είναι το 98.78% στο σύνολο δεδομένων “Firm Teacher”, ενώ καταγράφεται και ένα σύνολο όπου ο αλγόριθμος υπερτερεί σε ποσοστό επιτυχίας σε σχέση με όλους τους άλλους (Breast Canser Winsconsin), σημειώνοντας 96.85%.

Ακόμη, ο Constrained K-means παρουσιάζει πολύ σημαντική βελτίωση σε σχέση με το παραδοσιακό classification via clustering με τον απλό K-means. Γενικώς, υπερτερεί του K-means, ο οποίος κυμαίνεται σε ποσοστό 61.2%, κατά 14.52% κατά μέσο όρο. Όπως παρατηρούμε, η προταθείσα τεχνική σημειώνει καλύτερα ποσοστά από την υπάρχουσα σε όλα τα σύνολα δεδομένων, με διαφορά που κυμαίνεται από 1.14% έως και 43,22%. Η Εικόνα 6.1 είναι μια συγκριτική αναπαράσταση του μέσου ποσοστού επιτυχίας όλων των αλγορίθμων.



Εικόνα 6.1 - Μέσο ποσοστό επιτυχίας (Accuracy)

6.1.2 Ευαισθησία

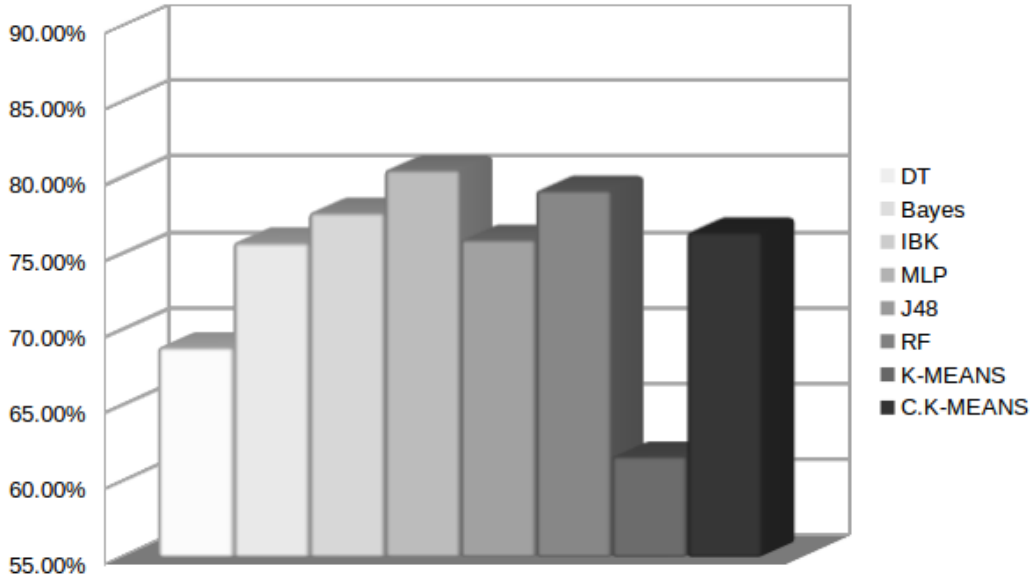
Βλέποντας τον πίνακα 6.3, παρατηρούμε ότι ο αλγόριθμος παρουσιάζει εξίσου υψηλά ποσοστά ευαισθησίας, όπως με το ποσοστό επιτυχίας. Επιπρόσθετα, στη

μετρική αυτή κατατάσσεται στην τέταρτη θέση, καταγράφοντας ανώτερο ποσοστό από κάποιους διαχρονικά καταξιωμένους αλγορίθμους, τον J48 και τον Naive Bayes. Η μέση διαφορά από αυτούς είναι της τάξης του 0.5-0.75%.

Πρώτος στην κατάταξη είναι πάλι ο Multi-Layer Perceptron, υπερτερώντας σε 15 σύνολα δεδομένων, ενώ η προταθείσα τεχνική πρωτεύει σε 6, όντας στην τρίτη θέση με αυτό το μέτρο μετά και από τον Random Forest. Η διαφορά του νέου αλγορίθμου από τον MLP κυμαίνεται σε ποσοστό 4.09%, σημαντικά μικρότερη από την αντίστοιχη του ποσοστού επιτυχίας. Το μεγαλύτερο ποσοστό που καταγράφει ο αλγόριθμος είναι το 97.96% στο σύνολο δεδομένων “Firm Teacher”.

Συγκριτικά με τον απλό K-means, ο αλγόριθμος φαίνεται πάλι βελτιωμένος και σε αυτήν τη μετρική, αποδίδοντας κατά μέσο όρο σχεδόν ισόποσες διαφορές με το ποσοστό επιτυχίας (14.77%). Πιο συγκεκριμένα, οι διαφορές κυμαίνονται ανάμεσα στο 1.88% (σύνολο “Wine”) και 35.25% (σύνολο “Soybean Backup Large”).

Η μέτρηση αυτή κατατάσσει το νέο αλγόριθμο σε σημαντική θέση ανάμεσα σε γνωστούς, αξιόπιστους και πολυχρησιμοποιούμενους αλγορίθμους, στη βάση του τι δεδομένα προβλέπει σωστά κατά μέσο όρο, σε σχέση με αυτά που θα έπρεπε να προβλέπει, ανά κατηγορία. Ακόμη ενισχύει την αρχική υπόθεση ότι οι προταθείσες βελτιώσεις θα απέφεραν σημαντική βελτίωση της απόδοσης του αλγορίθμου. Η Εικόνα 6.2 δείχνει το μέσο ποσοστό ευαισθησίας όλων των αλγορίθμων.



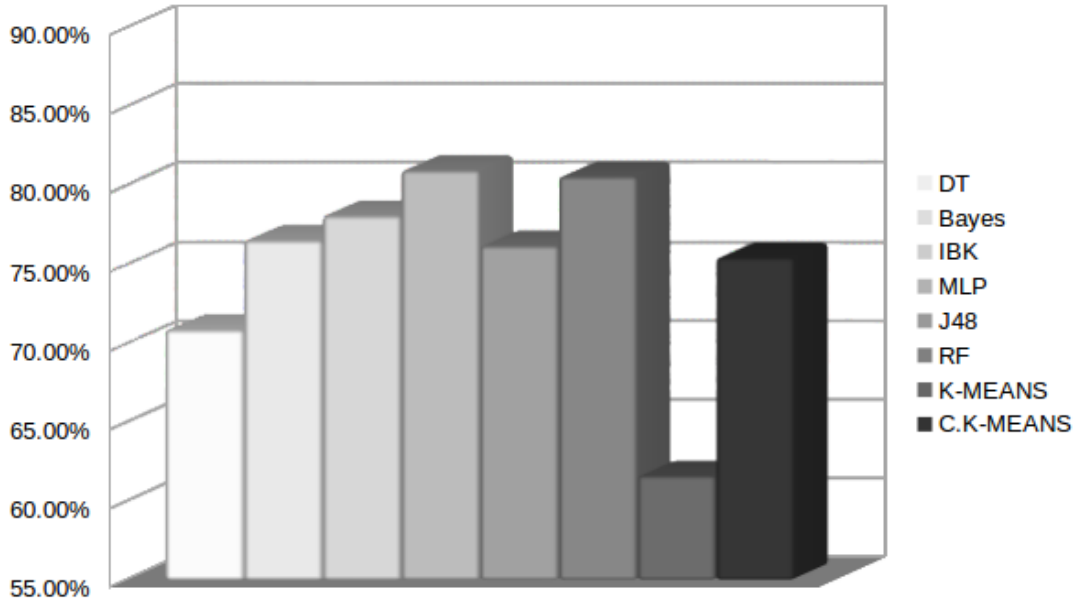
Εικόνα 6.2 - Μέσο ποσοστό ευαισθησίας (Sensitivity)

6.1.3 Ακρίβεια

Η τρίτη μετρική που παρουσιάζεται είναι η ακρίβεια, στον πίνακα 6.4. Το μέσο ποσοστό ακρίβειας κυμαίνεται στα ίδια επίπεδα με τα προηγούμενες δύο μετρικές, με τιμή 75.45%. Τα στοιχεία δείχνουν ότι ο αλγόριθμος υπερισχύει του επιλεγμένου κατηγοριοποιητή κανόνων περίπου κατά 5 ποσοστιαίες μονάδες, ενώ υπολείπεται των άλλων αλγορίθμων κατά μικρότερα ποσοστά με μέγιστο τις 5.5 μονάδες από τον πρώτο.

Πρώτος έρχεται και πάλι ο MLP, με το μέσο ποσοστό του να καταγράφεται στο 80.96%, πρωτεύοντας σε 18 σύνολα δεδομένων. Ο νέος αλγόριθμος καταλαμβάνει την πρώτη θέση σε 3 σύνολα δεδομένων, όσα και ο Naive Bayes και ο κατηγοριοποιητής κανόνων. Το μεγαλύτερο ποσοστό που πετυχαίνει ο Constrained K-means είναι το 97.62% και πάλι στο σύνολο “Firm Teacher”.

Σε αντιπαραβολή με τον απλό K-means, ο νέος αλγόριθμος σημειώνει και σε αυτή τη μετρική ανάλογες διαφορές, με τη μέση τιμή τους να είναι στο 13.84%. Η μεγαλύτερη διαφορά είναι 38.82% στο σύνολο “Firm Teacher” ενώ η μικρότερη είναι 0.38% στο σύνολο “Breast Cancer (wdbc)”. Μόλις ένα σύνολο δεδομένων καταγράφηκε, όπου ο απλός K-means παρουσιάζει καλύτερη ακρίβεια, με διαφορά 1.73% (σύνολο “Abalone”).



Εικόνα 6.3 - Μέσο ποσοστό ακρίβειας (Precision)

Η μέτρηση αυτή με τη σειρά της επιβεβαιώνει τη βελτίωση της κλασσικής τεχνικής κατηγοριοποίησης μέσω ομαδοποίησης από το νέο αλγόριθμο. Εκτός αυτού, καθιστά τον αλγόριθμο αξιόπιστο, στη βάση του αν τα δεδομένα που προβλέπει σε μια κατηγορία, ανήκουν όντως σε αυτή την κατηγορία. Γενικότερα, επισφραγίζει την επίτευξη συγκρίσιμων ποσοστών από την κατηγοριοποίηση μέσω ομαδοποίησης, σε σχέση με υψηλά αξιολογούμενους και καθιερωμένους αλγορίθμους. Η Εικόνα 6.3 παρουσιάζει το μέσο ποσοστό ακρίβειας για όλους τους αλγορίθμους.

Πίνακας 6.2 – Ποσοστό επιτυχίας του αλγορίθμου

Constrained K-Means Classification

	Dataset	DT	Bayes	IBK	MLP	J48	RF	K-MEANS	C.K-MEANS
1	Abalone	52.98%	51.78%	50.35%	**55.97%**	52.81%	53.15%	50.73%	52.21%
2	Balance scale	72.00%	90.40%	84.80%	**90.72%**	76.64%	81.44%	55.52%	68.64%
3	Banknote	95.92%	84.26%	99.85%	**99.93%**	98.54%	99.13%	57.43%	85.57%
4	Blood Transfusion	76.20%	75.40%	70.45%	**78.21%**	77.81%	72.73%	59.09%	66.44%
5	Breast cancer (Winsconsin)	95.28%	95.99%	95.14%	95.28%	94.56%	95.99%	95.71%	**96.85%**
6	Breast cancer (wdbc)	94.02%	92.97%	95.96%	**96.66%**	93.15%	96.49%	92.62%	93.85%
7	Breast_tissue	57.55%	70.75%	**71.70%**	66.98%	66.04%	67.92%	41.51%	60.38%
8	Cardiotocography	67.73%	70.51%	78.65%	83.16%	83.58%	**87.35%**	34.20%	54.99%
9	Contraceptive	**54.99%**	49.29%	43.31%	54.51%	53.22%	51.53%	37.54%	47.39%
10	Dermatology	90.98%	97.54%	95.36%	**98.36%**	95.90%	96.99%	71.86%	97.81%
11	Ecoli	76.79%	85.42%	80.36%	**85.71%**	84.23%	84.82%	52.98%	79.76%
12	Fertility Diagnosis	88.00%	88.00%	83.00%	**90.00%**	85.00%	88.00%	44.00%	68.00%
13	Firm Teacher	83.47%	89.22%	98.29%	99.94%	**99.96%**	99.95%	55.56%	98.78%
14	Glass	70.09%	49.53%	70.56%	67.29%	65.89%	**80.37%**	40.19%	45.33%
15	GPS	80.37%	84.05%	82.21%	**85.28%**	80.37%	80.37%	66.87%	82.82%
16	Ionosphere	89.46%	82.62%	86.32%	91.17%	91.45%	**92.87%**	70.94%	75.21%
17	Iris	92.66%	96.00%	95.33%	**97.33%**	96.00%	94.00%	84.00%	93.33%
18	Messidor Features	62.38%	56.82%	61.34%	**72.02%**	64.38%	65.60%	52.74%	57.41%
19	Optdigit	60.21%	90.82%	**98.83%**	97.94%	87.42%	94.94%	74.80%	90.76%
20	Parkinson Disease	81.03%	69.23%	**96.41%**	90.77%	80.51%	91.79%	58.46%	76.41%
21	Parkinson Speech	65.81%	59.85%	67.96%	**68.96%**	66.89%	73.10%	57.78%	60.02%
22	Pima Indian	71.22%	76.30%	70.18%	**75.39%**	73.83%	73.05%	64.84%	70.96%
23	Seeds	86.19%	91.43%	94.29%	**95.24%**	91.90%	90.95%	88.57%	91.43%
24	Semeion	50.52%	82.81%	86.79%	**90.57%**	63.52%	77.36%	58.07%	82.18%
25	Sonar	69.23%	67.79%	**86.54%**	82.21%	71.15%	**86.54%**	54.33%	73.56%
26	Soybean Backup Large	76.54%	91.53%	89.25%	91.21%	87.95%	**92.51%**	46.91%	86.97%
27	Spect Heart	**74.91%**	68.91%	68.91%	70.04%	70.41%	70.41%	62.55%	67.42%
28	Spectf Heart	79.40%	68.54%	70.41%	77.15%	74.91%	**82.02%**	64.04%	66.29%
29	Statlog Aust Credit	84.64%	77.25%	80.00%	82.90%	85.22%	**87.25%**	76.81%	85.65%
30	Statlog German Credit	71.00%	75.70%	67.80%	70.90%	**73.90%**	**73.90%**	59.40%	70.40%
31	Statlog Heart	**84.81%**	83.70%	75.19%	78.15%	76.67%	82.22%	77.04%	80.37%
32	Statlog Img Seg	84.94%	86.00%	94.69%	94.07%	93.46%	**96.00%**	57.90%	85.06%
33	Statlog Sat Image	78.60%	79.20%	87.85%	87.85%	83.80%	**89.25%**	67.30%	78.15%
34	Thyroid Disease New	91.63%	96.74%	**97.21%**	96.74%	92.09%	94.88%	85.12%	94.42%
35	Urban	63.69%	78.57%	76.79%	78.57%	**79.17%**	85.12%	60.12%	73.21%
36	Vertebral Column 3C	79.35%	83.23%	78.39%	**85.48%**	81.61%	84.19%	44.19%	74.52%
37	Waveform	70.40%	81.02%	76.90%	**82.40%**	75.94%	81.88%	49.94%	81.20%
38	Wine	88.76%	96.63%	94.94%	**97.19%**	93.82%	96.63%	94.38%	96.63%
39	Yeast	54.92%	57.61%	52.29%	59.03%	55.66%	**59.10%**	36.86%	50.27%
40	Zoo	86.75%	**96.39%**	95.18%	95.18%	90.36%	95.18%	69.88%	92.77%
	Average	76.39%	79.25%	81.24%	**83.91%**	80.24%	83.67%	61.82%	76.34%

Πίνακας 6.3 – Ευαισθησία του αλγορίθμου Constrained K-Means Classification

	Dataset	DT	Bayes	IBK	MLP	J48	RF	K-MEANS	C.K-MEANS
1	Abalone	52.53%	53.53%	50.43%	**55.70%**	52.80%	53.03%	50.90%	53.54%
2	Balance scale	52.07%	65.40%	61.37%	**83.13%**	55.43%	58.90%	51.43%	73.93%
3	Banknote	95.70%	83.75%	99.85%	**99.90%**	98.55%	99.10%	57.70%	84.92%
4	Blood Transfusion	50.00%	56.40%	55.70%	61.95%	65.95%	58.35%	49.20%	**67.16%**
5	Breast cancer (Winsconsin)	94.85%	96.35%	94.40%	94.95%	94.05%	95.55%	94.85%	**96.81%**
6	Breast cancer (wdbc)	93.10%	92.30%	95.60%	**96.30%**	93.05%	95.95%	91.05%	93.28%
7	Breast_tissue	55.03%	70.23%	**71.40%**	65.88%	64.43%	67.00%	45.02%	58.62%
8	Cardiotocography	51.21%	71.86%	72.63%	77.50%	77.31%	**81.50%**	38.32%	64.02%
9	Contraceptive	52.27%	50.23%	41.77%	**54.00%**	50.73%	48.80%	37.77%	49.47%
10	Dermatology	87.77%	97.67%	95.25%	**98.22%**	95.53%	96.70%	75.67%	97.55%
11	Ecoli	40.14%	60.75%	59.24%	57.89%	56.64%	59.35%	49.99%	**61.79%**
12	Fertility Diagnosis	50.00%	50.00%	57.95%	**72.75%**	48.30%	60.80%	50.40%	60.23%
13	Firm Teacher	50.00%	67.40%	95.55%	99.90%	**99.95%**	99.90%	65.80%	97.96%
14	Glass	56.90%	53.53%	67.65%	58.75%	68.95%	**76.33%**	40.27%	51.96%
15	GPS	79.80%	83.00%	82.20%	**85.00%**	79.80%	80.05%	65.15%	81.66%
16	Ionosphere	89.80%	83.45%	82.20%	88.25%	89.45%	**91.85%**	71.40%	76.48%
17	Iris	92.67%	96.00%	95.30%	**97.30%**	96.00%	94.00%	87.30%	93.33%
18	Messidor Features	63.25%	59.05%	61.45%	**71.90%**	64.60%	65.90%	51.85%	57.42%
19	Optdigit	60.14%	90.84%	**98.82%**	97.94%	87.39%	94.92%	75.81%	90.76%
20	Parkinson Disease	73.35%	76.80%	**96.20%**	88.95%	73.05%	87.55%	70.50%	77.34%
21	Parkinson Speech	64.85%	57.60%	67.25%	68.60%	65.25%	**71.95%**	57.35%	60.04%
22	Pima Indian	67.00%	**72.80%**	66.20%	72.00%	70.55%	68.05%	60.55%	72.33%
23	Seeds	86.20%	91.43%	94.30%	**95.27%**	91.87%	90.93%	88.57%	91.43%
24	Semeion	49.40%	82.34%	85.73%	**90.04%**	62.02%	76.16%	58.14%	81.95%
25	Sonar	68.95%	68.75%	**86.20%**	82.05%	71.15%	86.15%	54.35%	73.08%
26	Soybean Backup Large	69.03%	89.12%	87.24%	89.34%	82.89%	**89.74%**	53.17%	88.42%
27	Spect Heart	**72.55%**	67.85%	66.60%	68.95%	68.45%	68.70%	60.65%	67.42%
28	Spectf Heart	53.35%	76.80%	59.80%	68.10%	62.00%	62.45%	50.00%	**78.77%**
29	Statlog Aust Credit	84.90%	75.40%	79.85%	82.70%	84.90%	**87.35%**	78.55%	86.30%
30	Statlog German Credit	59.95%	68.45%	61.80%	65.15%	66.70%	62.50%	52.50%	**69.71%**
31	Statlog Heart	**84.35%**	83.25%	75.00%	78.15%	77.15%	81.55%	77.15%	80.83%
32	Statlog Img Seg	84.83%	86.84%	94.87%	94.29%	93.69%	**96.19%**	62.81%	85.79%
33	Statlog Sat Image	75.63%	78.38%	86.67%	86.08%	81.68%	**87.13%**	66.55%	77.59%
34	Thyroid Disease New	86.43%	93.43%	**96.13%**	96.10%	88.87%	91.23%	72.80%	87.62%
35	Urban	61.98%	80.28%	77.42%	78.91%	79.60%	**85.50%**	63.36%	74.44%
36	Vertebral Column 3C	69.87%	79.33%	75.13%	**82.43%**	76.67%	79.90%	41.52%	73.22%
37	Waveform	72.77%	80.97%	76.90%	**83.80%**	75.93%	81.87%	49.97%	81.11%
38	Wine	88.83%	96.90%	95.77%	**97.43%**	93.40%	96.87%	95.30%	97.18%
39	Yeast	37.75%	50.46%	52.26%	47.03%	51.63%	49.64%	37.93%	**57.24%**
40	Zoo	75.07%	**90.71%**	88.67%	88.67%	81.27%	88.67%	65.06%	84.85%
	Average	68.86%	75.74%	77.72%	**80.53%**	75.94%	79.20%	61.67%	76.44%

Πίνακας 6.4– Ακρίβεια του αλγορίθμου Constrained K-Means Classification

	Dataset	DT	Bayes	IBK	MLP	J48	RF	K-MEANS	C.K-MEANS
1	Abalone	52.43%	49.63%	50.57%	**55.80%**	52.67%	53.13%	52.57%	50.84%
2	Balance scale	48.83%	60.27%	59.63%	**80.13%**	52.97%	59.77%	55.03%	71.66%
3	Banknote	96.00%	84.25%	99.85%	**99.95%**	98.50%	99.10%	57.60%	85.85%
4	Blood Transfusion	38.10%	62.50%	56.75%	**69.55%**	68.75%	60.15%	49.35%	62.75%
5	Breast cancer (Winsconsin)	94.75%	95.00%	94.80%	94.65%	93.95%	95.55%	95.60%	**96.28%**
6	Breast cancer (wdbc)	94.05%	92.60%	95.70%	**96.55%**	92.45%	96.50%	93.15%	93.53%
7	Breast_tissue	52.73%	69.30%	**71.33%**	65.03%	66.07%	67.90%	50.40%	61.40%
8	Cardiotocography	64.24%	64.81%	75.36%	78.51%	79.79%	**86.62%**	34.30%	53.60%
9	Contraceptive	53.13%	49.47%	41.63%	**53.57%**	51.33%	49.47%	37.60%	48.41%
10	Dermatology	91.47%	97.37%	94.90%	**98.13%**	95.15%	96.13%	84.45%	97.55%
11	Ecoli	47.78%	64.05%	58.05%	**58.58%**	55.95%	61.31%	52.19%	57.77%
12	Fertility Diagnosis	44.00%	44.00%	58.60%	**76.65%**	43.80%	70.20%	50.20%	54.96%
13	Firm Teacher	41.75%	94.30%	98.20%	**99.95%**	**99.95%**	**99.95%**	58.80%	97.62%
14	Glass	**77.56%**	50.18%	67.41%	58.72%	64.60%	70.48%	29.32%	52.39%
15	GPS	81.10%	**87.80%**	82.10%	85.50%	81.10%	80.50%	71.05%	87.05%
16	Ionosphere	87.10%	81.30%	88.30%	**92.90%**	91.85%	92.60%	69.85%	74.46%
17	Iris	92.67%	96.00%	95.30%	**97.30%**	96.00%	94.00%	88.70%	93.38%
18	Messidor Features	64.20%	68.50%	61.35%	**71.95%**	64.60%	65.95%	52.00%	57.39%
19	Optdigit	62.21%	91.18%	**98.84%**	97.94%	87.39%	94.94%	76.24%	91.13%
20	Parkinson Disease	74.45%	69.90%	**94.40%**	86.95%	73.75%	89.90%	66.25%	71.54%
21	Parkinson Speech	65.05%	58.50%	67.30%	68.45%	66.20%	**72.65%**	57.25%	59.85%
22	Pima Indian	68.10%	**74.00%**	66.95%	72.90%	71.10%	70.35%	60.90%	70.33%
23	Seeds	86.37%	91.43%	94.33%	**95.27%**	91.90%	90.97%	88.90%	91.38%
24	Semeion	51.22%	83.65%	88.08%	**90.94%**	63.05%	78.61%	59.35%	82.61%
25	Sonar	69.05%	69.85%	**86.85%**	82.20%	71.05%	87.00%	54.35%	73.68%
26	Soybean Backup Large	81.54%	88.66%	87.35%	**89.47%**	81.56%	89.39%	49.53%	85.11%
27	Spect Heart	**74.85%**	67.90%	67.90%	69.05%	69.50%	69.45%	61.05%	67.81%
28	Spectf Heart	65.25%	67.60%	58.35%	66.25%	61.85%	**73.95%**	37.85%	68.97%
29	Statlog Aust Credit	84.50%	79.60%	79.75%	82.70%	84.90%	**87.05%**	76.90%	85.91%
30	Statlog German Credit	63.90%	**70.95%**	61.75%	65.25%	68.55%	69.05%	52.45%	67.19%
31	Statlog Heart	**84.85%**	83.65%	74.85%	77.90%	76.35%	82.30%	76.85%	80.46%
32	Statlog Img Seg	86.21%	87.16%	94.93%	94.53%	93.77%	**96.30%**	63.16%	85.72%
33	Statlog Sat Image	78.00%	78.20%	86.70%	86.22%	81.98%	**88.28%**	68.18%	77.36%
34	Thyroid Disease New	90.60%	97.47%	96.33%	95.27%	89.80%	94.90%	86.47%	**97.53%**
35	Urban	71.82%	80.27%	77.94%	79.58%	80.63%	**85.40%**	64.74%	74.47%
36	Vertebral Column 3C	75.03%	79.53%	75.97%	**81.63%**	76.33%	79.97%	43.17%	72.23%
37	Waveform	72.77%	84.13%	76.90%	83.80%	75.87%	81.97%	50.07%	**84.63%**
38	Wine	90.27%	96.63%	94.83%	**97.03%**	94.33%	96.57%	94.20%	96.43%
39	Yeast	46.62%	**55.97%**	51.58%	49.18%	53.97%	55.67%	34.00%	48.65%
40	Zoo	69.97%	**93.21%**	92.10%	92.65%	84.57%	89.13%	60.17%	87.94%
	Average	70.86%	76.52%	78.10%	**80.96%**	76.20%	80.58%	61.60%	75.45%

6.2 Διάγνωση της κίρρωσης του ήπατος από εικόνες βιοψίας

Δεδομένων των καλών ποσοστών της κατηγοριοποίησης μέσω ομαδοποίησης στην αξιολόγηση με ικανό αριθμό συνόλων δεδομένων, επιχειρήθηκε η χρήση του στην επίλυση του προβλήματος της εκτίμησης της εξέλιξης της ίνωσης του ήπατος από εικόνες βιοψίας [16]. Αξιολογήθηκε η ικανότητα του αλγορίθμου να προβλέψει σωστά τα δεδομένα ενδιαφέροντος τόσο με μετρικές απόδοσης, όσο και συγκριτικά με έναν επιλεγμένο, σύγχρονο και υψηλά αξιολογημένο αλγόριθμο. Ακολούθως, ο αλγόριθμος εφαρμόστηκε στο πρόβλημα με επιτυχία.

6.2.1 Το πρόβλημα

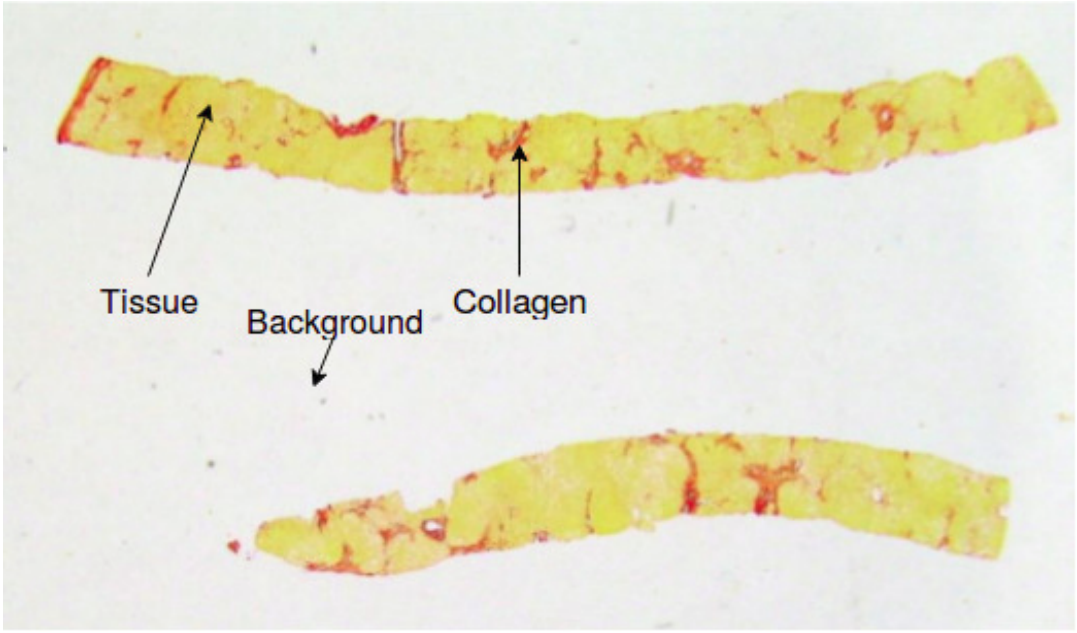
Η ίνωση του ήπατος είναι μια παθολογική κατάσταση κατά την οποία ινώδες κολλαγόνο συγκεντρώνεται στο ήπαρ, προκαλώντας τη δυσλειτουργία του [57]. Το τελευταίο στάδιο της εξέλιξης της ίνωσης είναι η κίρρωση. Όταν παρουσιάζεται, η αντιμετώπισή της είναι δύσκολη, αλλά και μπορεί να επιφέρει μη αναστρέψιμες συνέπειες για τον ασθενή. Είναι λοιπόν καίρια η σημασία της έγκαιρης και ακριβούς διάγνωσης για τη διαμόρφωση της θεραπευτικής στρατηγικής που θα ακολουθηθεί.

Το πρόβλημα ξεκινά από την ύπαρξη ινώδους κολλαγόνου στο συκώτι που επιβαρύνει τη ροή αίματος διαμέσου του, το οποίο προοδευτικά προκαλεί ηπατική δυσλειτουργία. Συγκεντρώνεται στο συκώτι, σχηματίζοντας κοιλότητες στον ιστό, οι οποίες πληθαίνουν και ενώνονται όσο η νόσος εξελίσσεται, αχρηστεύοντας προοδευτικά τον ιστό.

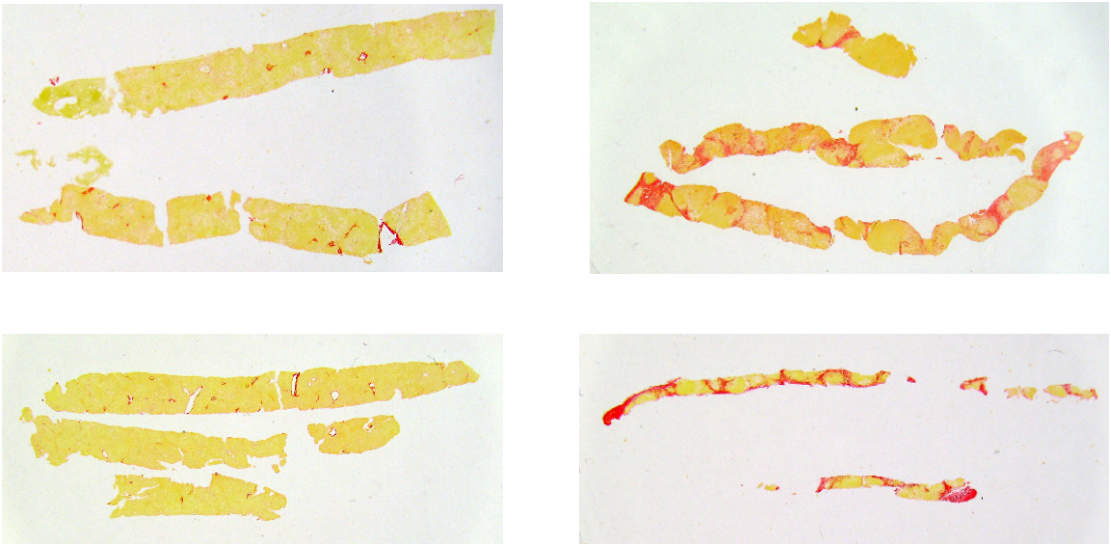
Η διάγνωση της κατάστασης παραδοσιακά πραγματοποιούταν με οπτική εκτίμηση των εικόνων βιοψίας, βαθμολογώντας το πόσο υγιής είναι ένας ιστός ανάλογα με ορισμένες παρατηρήσεις, σύμφωνα με ορισμένες προτυποποιημένες εξετάσεις. Ακόμη, έχει αποδειχθεί πως σημαντικός δείκτης για την εκτίμηση της σοβαρότητας της ίνωσης είναι το ποσοστό του μολυσμένου ιστού σε σχέση με τον υγιή (CPA) [58].

Στην Εικόνα 6.4 παρατίθεται ένα παράδειγμα εικόνας βιοψίας ήπατος. Διακρίνεται ο μολυσμένος ιστός που χαρακτηρίζεται από το κολλαγόνο με κόκκινο χρώμα, ο υγιής ιστός με το κίτρινο και το φόντο. Το πρόβλημα λοιπόν ανάγεται στη διάκριση των τριών ειδών εικονοστοιχείων που αποτελούν την εικόνα και στην ακριβή καταμέτρηση των εικονοστοιχείων ενδιαφέροντος για την εξαγωγή του CPA. Επίσης

στην Εικόνα 6.5 παρατίθενται ενδεικτικά τέσσερις εικόνες βιοψίας ήπατος, από υγιή και ασθενή δείγματα.



Εικόνα 6.4 – Κατηγορίες των pixel των εικόνων βιοψίας



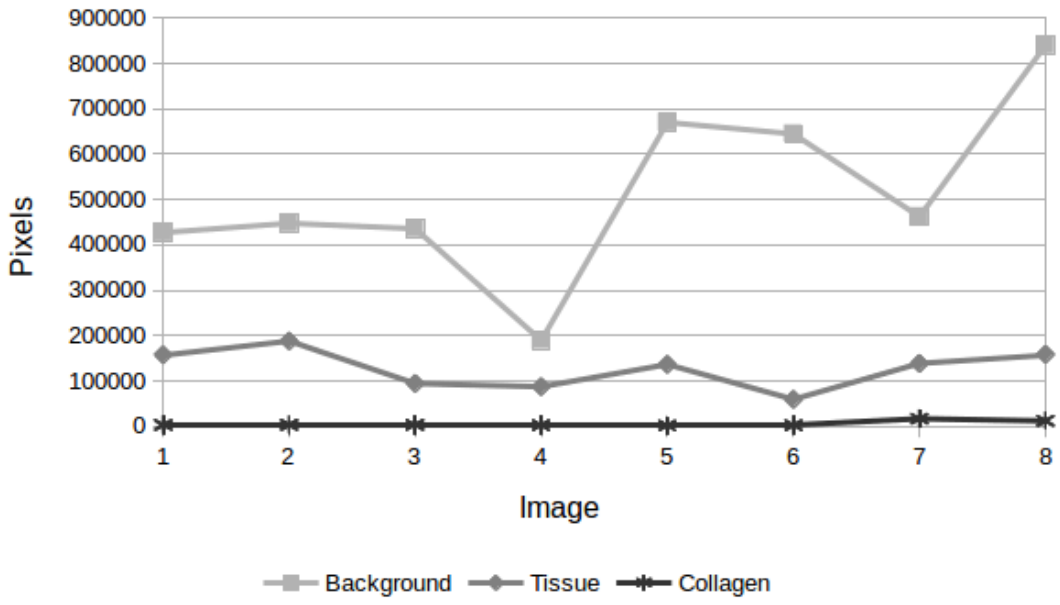
Εικόνα 6.5– Ενδεικτικές εικόνες βιοψίας ήπατος

6.2.2 Το σύνολο δεδομένων και η πειραματική διαδικασία

Αρχικά κατασκευάστηκε ένα σύνολο δεδομένων που αποτελούταν από τις τιμές εικονοστοιχείων οκτώ εικόνων βιοψίας. Για τις εικόνες αυτές είχε προηγηθεί προσημείωση των διακριτών περιοχών από ειδικευμένο γιατρό. Οι εικόνες αντιπροσώπευαν βιοψίες που αντιστοιχούσαν σε διάφορα στάδια της νόσου.

Για κάθε εικονοστοιχείο, υπήρχαν τρία χαρακτηριστικά, για τις τιμές R, G, B αντίστοιχα, ένα χαρακτηριστικό που ήταν το αναγνωριστικό της κάθε εικόνας και το ειδικό χαρακτηριστικό της ετικέτας της κατηγορίας του. Το σύνολο δεδομένων χαρακτηρίζεται ως μη ισορροπημένο, καθώς η κατανομή των εικονοστοιχείων στις κατηγορίες ήταν κατά προσέγγιση: 77.9% φόντο, 21.1% υγιής ιστός και λιγότερο του 1% κολλαγόνο. Ο Πίνακας 6.5 παρουσιάζει τα πραγματικά εικονοστοιχεία κάθε κατηγορίας ανά εικόνα, όπως και το υπολογισμένο CPA. Το Εικόνα 6.6 δείχνει την πραγματική κατανομή των εικονοστοιχείων στις εικόνες.

Για τις ανάγκες του πειράματος η τιμή της παραμέτρου l στις εξισώσεις 4.1, 4.2 – αυστηρότητα των περιορισμών – τέθηκε σε 0.35 (όπως προέκυψε πειραματικά). Επιπλέον, χρησιμοποιήθηκε η διασταυρωμένη επικύρωση leave-one-out. Αυτό επετεύχθη με χρήση του αναγνωριστικού εικόνας, χρησιμοποιώντας σε κάθε επανάληψη 7 εικόνες για εκπαίδευση και 1 για έλεγχο.



Εικόνα 6.6 – Αριθμός pixel των κατηγοριών ανά εικόνα βιοψίας

Πίνακας 6.5 - Αριθμός pixel των κατηγοριών ανά εικόνα βιοψίας

Image	Number of Pixels			CPA (%)
	Background	*Liver Tissue*	*Collagen*	
1	427576	157995	3433	2.13
2	448032	188381	3724	1.94
3	436082	95009	3848	3.89
4	189834	87849	3038	3.34
5	670074	136913	2263	1.63
6	644737	60072	2930	4.65
7	462720	139298	17465	11.14
8	839979	158962	12553	7.32
Total	**4119034**	**1024479**	**49254**	

6.2.3 Αξιολόγηση

Η δομή του συνόλου δεδομένων σε συνδυασμό με τη φύση του προβλήματος, αναδεικνύουν την επιτακτική ανάγκη για επιτυχή κατηγοριοποίηση στις μικρές κατηγορίες, που είναι οι κατηγορίες ενδιαφέροντος για τη διάγνωση. Χωρίς αυτό να υποβαθμίζει την ανάγκη για συνολικά υψηλά ποσοστά επιτυχίας.

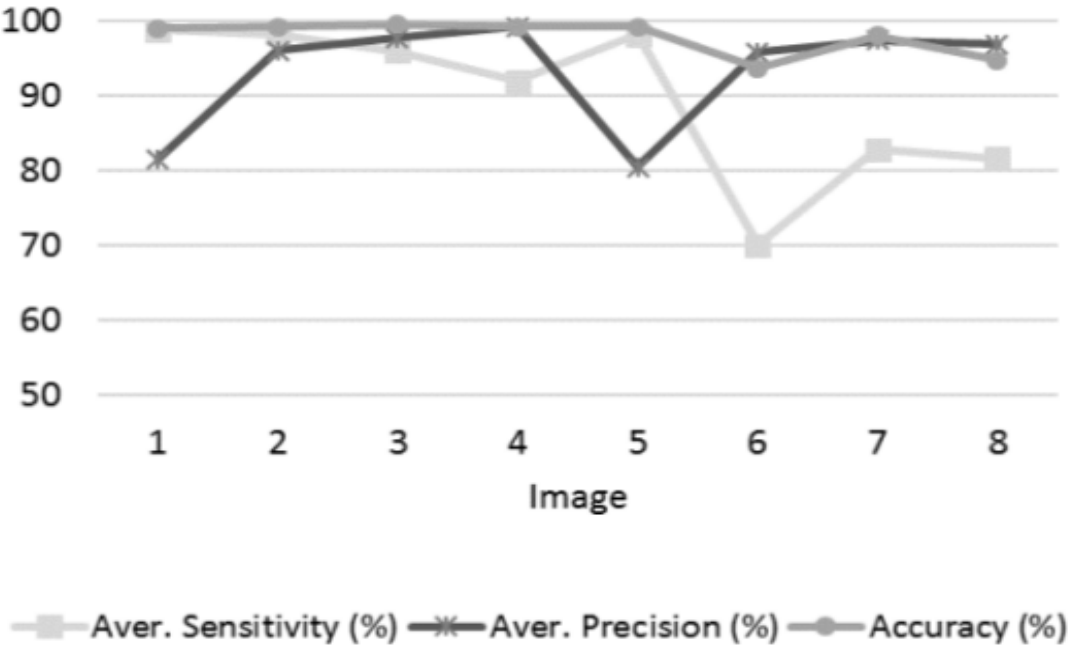
Για την αξιολόγηση του αλγορίθμου, χρησιμοποιήθηκαν οι ίδιες τρεις μετρικές, όπως στην ενότητα 6.1. Το ποσοστό επιτυχημένων κατηγοριοποιήσεων ήταν η πρώτη γενική εικόνα για το αποτέλεσμα της κατηγοριοποίησης, αλλά όχι αντιπροσωπευτική. Αυτό γιατί επηρεάζεται κυρίως από τα αποτελέσματα κατηγοριοποίησης στη μεγάλη

κατηγορία, με αυτό να σημαίνει ότι δεν έχουμε πραγματική εκτίμηση του τι συμβαίνει στις κατηγορίες ενδιαφέροντος.

Βαρύνουσας σημασίας παράμετροι στην αξιολόγηση ήταν η ευαισθησία και η ακρίβεια, καθώς τα μέσα ποσοστά τους επηρεάζονται εξίσου από τα ποσοστά κάθε κατηγορίας. Είναι πρωτεύον να ξέρουμε τόσο ότι προβλέπεται σωστά ένα μεγάλο ποσοστό των εικονοστοιχείων των κατηγοριών (απαίτηση για υψηλό ποσοστό ευαισθησίας), όσο και ότι τα εικονοστοιχεία που προβλέπονται σε κάποια κατηγορία, όντως ανήκουν σε αυτή (απαίτηση για υψηλό ποσοστό ακρίβειας).

Ο Πίνακας 6.6 συγκεντρώνει τα αποτελέσματα της κατηγοριοποίησης για τις οκτώ εικόνες ξεχωριστά, ανά κατηγορία, αλλά και συνολικά. Με μια πρώτη ματιά στον πίνακα, παρατηρούμε τα πολύ υψηλά ποσοστά επιτυχημένων κατηγοριοποιήσεων σε όλες τις εικόνες. Τα συνολικά ποσοστά κυμαίνονται από 94.71% εώς 99.5% με μέση τιμή το 97.79%.

Επίσης, κοιτώντας τις συνολικές μετρήσεις, βλέπουμε επίσης πολύ υψηλά ποσοστά στην ευαισθησία και την ακρίβεια (89.68% και 93.09% αντίστοιχα κατά μέσο όρο). Η ευαισθησία κυμαίνεται από 70.04% ως 98.69% ενώ η ακρίβεια από 80.47% ως 99.2%. Η Εικόνα 6.7 παρουσιάζει τη διακύμανση των τριών μετρικών για όλες τις εικόνες.



Εικόνα 6.7 - Αποτελέσματα ανά εικόνα βιοψίας του προτεινόμενου αλγορίθμου

Πίνακας 6.6 – Αποτελέσματα ανά εικόνα βιοψίας του προτεινόμενου αλγορίθμου

Image	Class	Sensitivity (%)	Precision (%)	Accuracy (%)
1	Background	99.50	99.68	98.89
	Liver Tissue	97.24	99.31	
	Collagen	99.33	45.48	
	Average	***98.69***	***81.49***	
2	Background	99.70	99.20	99.14
	Liver Tissue	97.86	99.22	
	Collagen	97.02	89.48	
	Average	***98.19***	***95.96***	
3	Background	99.91	99.61	99.50
	Liver Tissue	97.98	99.17	
	Collagen	89.79	94.30	
	Average	***95.89***	***97.69***	
4	Background	99.40	99.72	99.16
	Liver Tissue	99.40	97.95	
	Collagen	77.06	99.91	
	Average	***91.95***	***99.20***	
5	Background	99.67	99.80	99.19
	Liver Tissue	96.90	98.31	
	Collagen	97.83	43.30	
	Average	***98.13***	***80.47***	
6	Background	99.98	93.60	93.69
	Liver Tissue	26.67	96.59	
	Collagen	83.48	97.14	
	Average	***70.04***	***95.78***	
7	Background	99.85	99.36	98.02
	Liver Tissue	97.88	93.68	
	Collagen	50.82	98.95	
	Average	***82.85***	***97.33***	
8	Background	99.91	94.39	94.71
	Liver Tissue	68.65	96.81	
	Collagen	76.52	99.24	
	Average	***81.69***	***96.81***	
all	Background	99.74	98.17	97.79
	Liver Tissue	85.32	97.63	
	Collagen	83.98	83.47	
	Average	***89.68***	***93.09***	

6.2.4 Συγκριτική μελέτη και επίλυση του προβλήματος

Τα αποτελέσματα του προτεινόμενου αλγορίθμου ήταν θετικά για την ικανότητα του αλγορίθμου στο πρόβλημα. Ακολούθως, επιλέχθηκε ένας από τους πιο γνωστούς και υψηλά βαθμολογούμενους καθιερωμένους κατηγοριοποιητές (Naive Bayes) και ο πρωτότυπος K-Means, για μια συγκριτική μελέτη απόδοσης πάνω στο πρόβλημα. Αυτό έγινε προκειμένου να διαπιστωθεί αν ο προτεινόμενος αλγόριθμος στο παρόν πρόβλημα συμπεριφέρεται ανάλογα με κάποιον ευρέως γνωστό αλγόριθμο, αλλά και αν παρουσιάζει βελτίωση σε σχέση με το παραδοσιακό classification via clustering.

Παρακάτω παρατίθεται ο Πίνακας 6.7 με όλα τα αποτελέσματα ανά εικόνα αλλά και συγκεντρωτικά για τους τρεις αλγορίθμους. Ακολουθεί η Εικόνα 6.8 που αναπαριστά το ποσοστό επιτυχίας, την ευαισθησία και την ακρίβεια αντίστοιχα και για τους τρεις αλγορίθμους συγκριτικά.

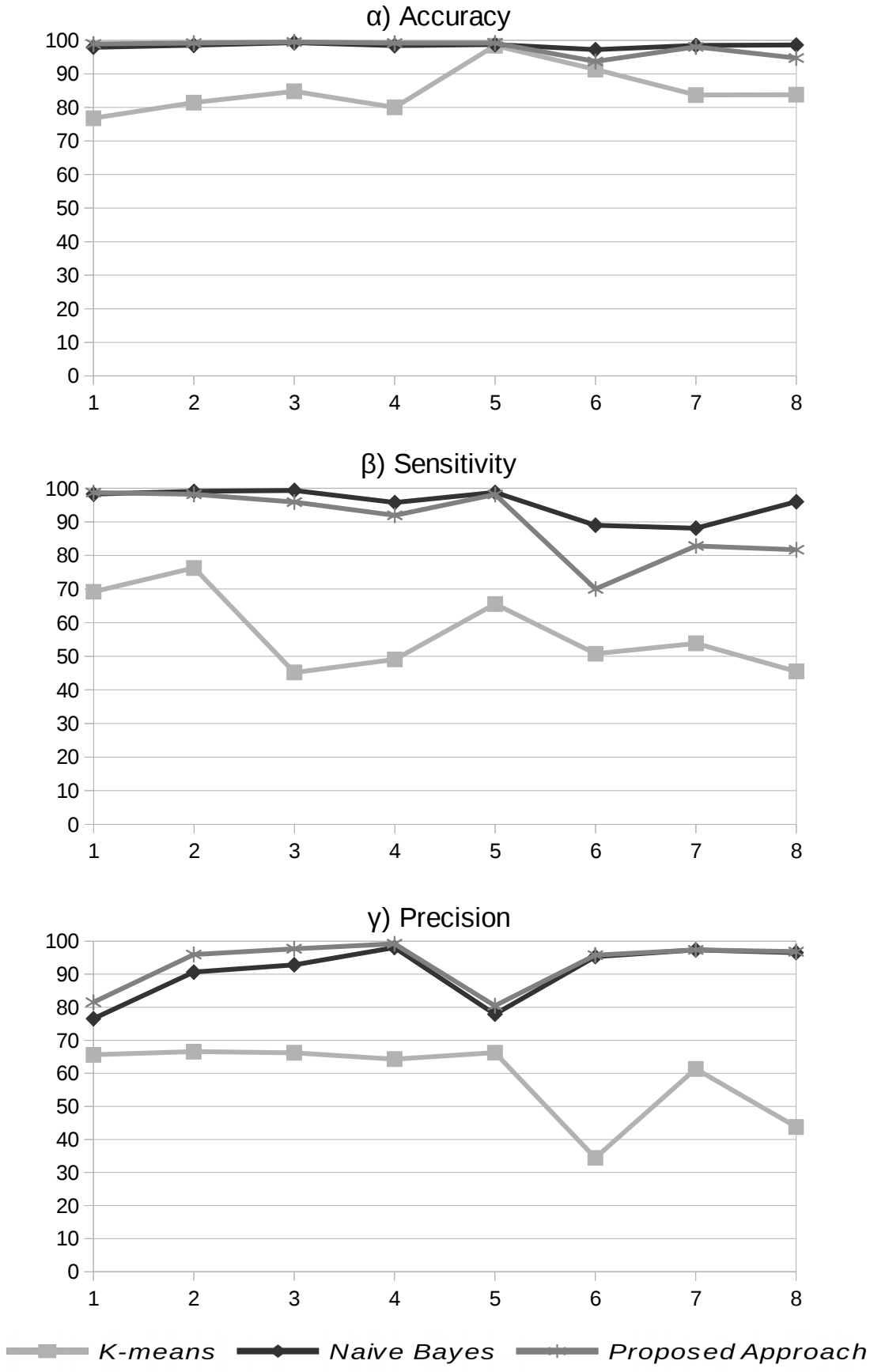
Αρχικά, γίνεται αμέσως αντιληπτές από τα διαγράμματα οι αυξημένες επιδόσεις της προταθείσας τεχνικής σε αντιπαραβολή με τη χρήση του απλού K-means. Επιβεβαιώνεται σε ένα ακόμα πρόβλημα η υπεροχή του νέου αλγορίθμου συγκριτικά με την παλαιότερη τεχνική κατηγοριοποίησης μέσω ομαδοποίησης με διαφορές τάξης μεγέθους άνω του 10% στο μέσο ποσοστό επιτυχίας και άνω του 30% στις άλλες δύο μετρικές.

Η προτεινόμενη τεχνική, όπως φαίνεται από τα διαγράμματα, επιτυγχάνει υψηλότατα ποσοστά, αντίστοιχα του αλγορίθμου Naive Bayes σε όλες τις μετρικές, καταγράφοντας μάλιστα καλύτερες επιδόσεις στη μέση ακρίβεια. Αυτό μεταφράζεται σε καλύτερη απόδοση του νέου αλγορίθμου, στη βάση του αν ο,τι προβλέπει σε μια κατηγορία, όντως ανήκει σε αυτή.

Τα αποτελέσματα της αξιολόγησης από τις μετρικές απόδοσης και τη συγκριτική μελέτη κρίθηκαν αξιόπιστα και ο αλγόριθμος κατάλληλος για εφαρμογή στο πρόβλημα. Στη συνεχεία, εξήχθησαν οι μετρήσεις του CPA για όλες τις εικόνες και συγκρίθηκε η απόδοση των αλγορίθμων και σε αυτή τη μέτρηση. Ο Πίνακας 6.8 συγκεντρώνει τα υπολογισμένα CPA ανά εικόνα για κάθε προσέγγιση και για τα πραγματικά δεδομένα, ενώ το Εικόνα 6.9 απεικονίζει τα αποτελέσματα αυτά για τους δύο αλγορίθμους, Naive Bayes και Constrained K-Means, και για τις πραγματικές τιμές συγκριτικά.

Πίνακας 6.7 – Συγκριτικά αναλυτικά αποτελέσματα ανά εικόνα βιοψίας

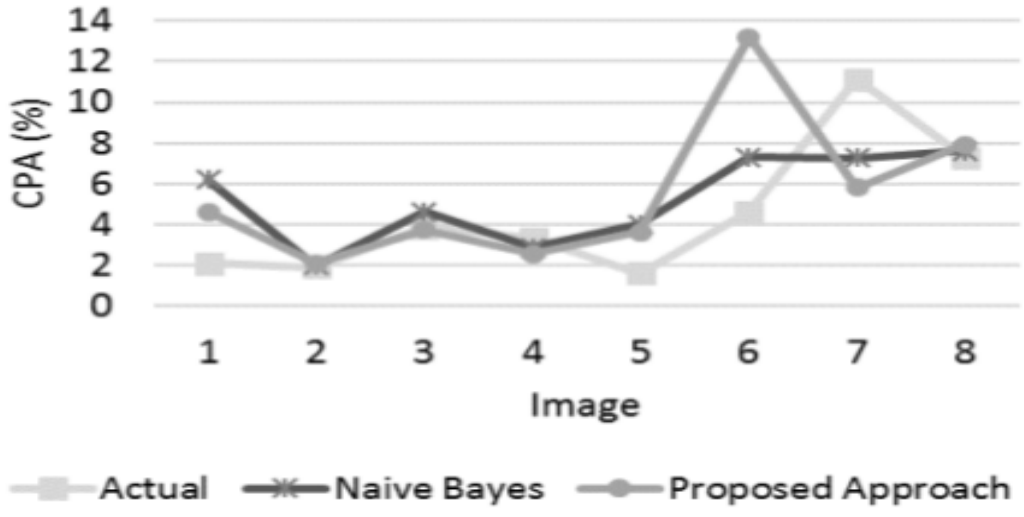
Image		K-means	Naive Bayes	Proposed Approach
1	Average Sensitivity (%)	69.23	98.27	98.69
	Average Precision (%)	65.63	76.53	81.49
	Accuracy (%)	76.76	97.92	98.89
2	Average Sensitivity (%)	76.30	99.07	98.19
	Average Precision (%)	66.57	90.63	95.96
	Accuracy (%)	81.41	98.53	99.14
3	Average Sensitivity (%)	45.20	99.37	95.89
	Average Precision (%)	66.23	92.83	97.69
	Accuracy (%)	84.78	99.27	99.50
4	Average Sensitivity (%)	49.07	95.77	91.95
	Average Precision (%)	64.30	98.07	99.20
	Accuracy (%)	80.02	98.39	99.16
5	Average Sensitivity (%)	65.57	98.77	98.13
	Average Precision (%)	66.27	77.90	80.47
	Accuracy (%)	98.41	98.71	99.19
6	Average Sensitivity (%)	50.77	89.00	70.04
	Average Precision (%)	34.43	95.30	95.78
	Accuracy (%)	91.30	97.20	93.69
7	Average Sensitivity (%)	53.87	88.13	82.85
	Average Precision (%)	61.33	97.33	97.33
	Accuracy (%)	83.69	98.48	98.02
8	Average Sensitivity (%)	45.50	96.00	81.69
	Average Precision (%)	43.80	96.53	96.81
	Accuracy (%)	83.79	98.61	94.71



Εικόνα 6.8 – Συγκριτικά αποτελέσματα ανά εικόνα βιοψίας

Πίνακας 6.8 - CPA ανά εικόνα

Image	Actual	K-means	Naive Bayes	Proposed Approach
1	2.13	84.77	6.21	4.62
2	1.94	61.62	2.01	2.13
3	3.89	79.11	4.62	3.76
4	3.34	57.69	2.89	2.56
5	1.63	6.92	4.02	3.65
6	4.65	93.48	7.31	13.18
7	11.14	57.71	7.27	5.81
8	7.32	90.28	7.63	7.91



Εικόνα 6.9 – CPA ανά εικόνα βιοψίας

Τα στοιχεία επιβεβαιώνουν και πάλι την καταλληλότητα της νέας τεχνικής στο πρόβλημα δείχνοντας ότι το CPA είναι γενικώς κοντά στο πραγματικό αλλά και χωρίς

εξαιρετικές διαφορές από την προσέγγιση με τον Naive Bayes. Ακόμη ο αλγόριθμος έχει πετύχει να υπολογίσει το CPA με απόκλιση μικρότερη του 1%, πράγμα που απεικονίζεται και στο διάγραμμα παρατηρώντας τις εικόνες βιοψίας 2, 3, 4 και 8 όπου οι δύο γραμμές σχεδόν ταυτίζονται. Η απόδοση του απλού K-Means, όπως γίνεται κατανοητό, κρίνεται ανεπαρκής για το πρόβλημα καθώς οι υπολογισμένες τιμές CPA είναι εξαιρετικά μακριά από τις πραγματικές.

Συνοψίζοντας, το πρόβλημα λύθηκε επιτυχώς. Τα παραπάνω αποτελέσματα μας οδηγούν στο συμπέρασμα ότι ο Constrained K-Means Classification είναι κατάλληλος και αποδοτικός αλγόριθμος για χρήση στο συγκεκριμένο πρόβλημα.

6.3 Αξιολόγηση του αλγορίθμου Stochastic Forest

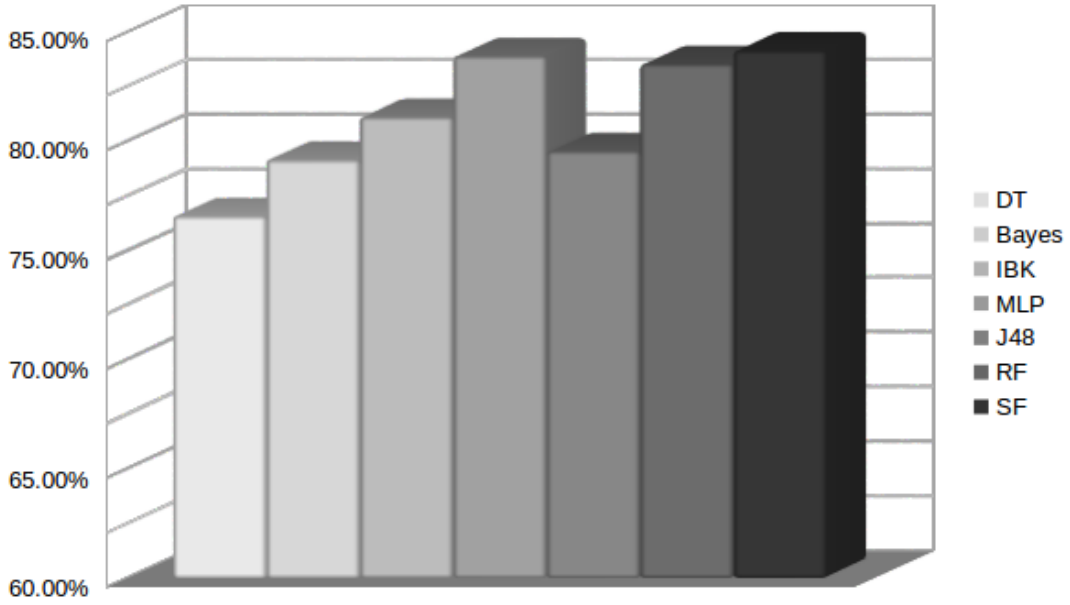
Τα αποτελέσματα του αλγορίθμου Stochastic Forest, συγκρινόμενα με αυτά των υπόλοιπων αλγορίθμων που αναφέρθηκαν παραπάνω παρουσιάζονται στους πίνακες 6.9, 6.10 και 6.11. Το πλήθος των κατηγοριοποιητών της ομάδας για τον Stochastic Forest τέθηκε σε 100, όπως και του Random Forest. Όπως αναφέρθηκε παραπάνω, τα αποτελέσματα παρουσιάζονται στις 3 βασικές μετρικές, το ποσοστό επιτυχημένων κατηγοριοποιήσεων (accuracy), την ευαισθησία (sensitivity) και την ακρίβεια (precision).

Τα αποτελέσματα του αλγορίθμου Stochastic Forest είναι ενθαρρυντικά, με βάση και τα τρία βασικά μέτρα απόδοσης που χρησιμοποιηθήκαν. Ο αλγόριθμος που αναπτύχθηκε πετυχαίνει υψηλά αποτελέσματα στην πλειοψηφία των συνόλων δεδομένων, συγκριτικά με τους υπόλοιπους αλγορίθμους.

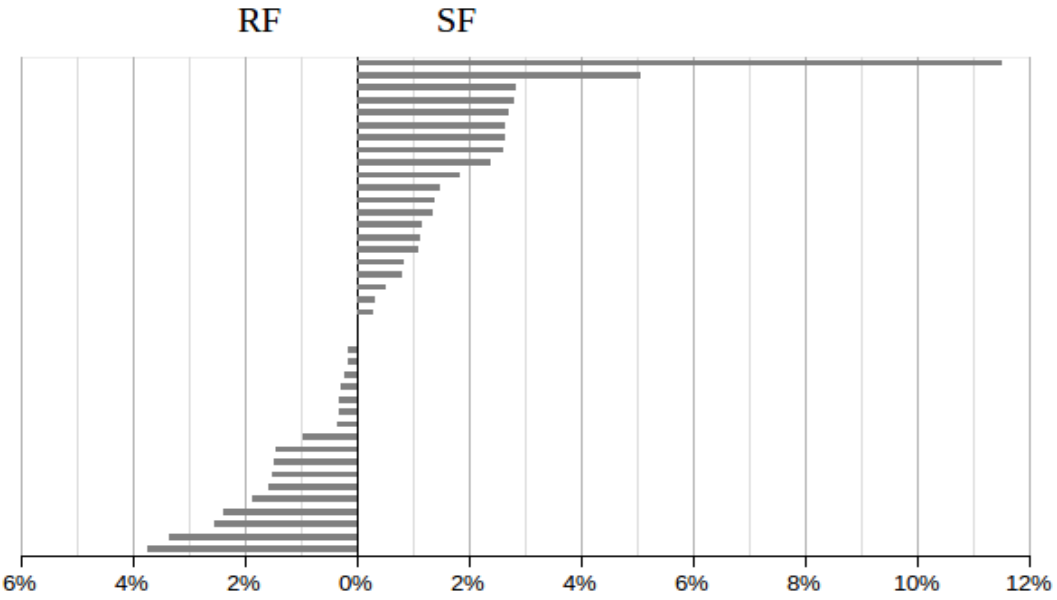
6.3.1 Accuracy

Όπως φαίνεται από τα αποτελέσματα που φαίνονται στον Πίνακα 6.9 και στην Εικόνα 6.10, κατά μέσο όρο, στο **ποσοστό των επιτυχημένων κατηγοριοποιήσεων** (accuracy) ο αλγόριθμος που αναπτύξαμε (Stochastic Forest - SF) έχει την καλύτερη απόδοση (84.17%), το πολυεπίπεδο Perceptron την δεύτερη καλύτερη (83.9%) και τα τυχαία δάση (Random Forest - RF) την τρίτη (83.54%). Όπως βλέπουμε υπάρχει

διαφορά 0.63% ανάμεσα στα τυχαία δάση και τα στοχαστικά δάση, που είναι σημαντική διαφορά, αναλογιζόμενοι το πλήθος των συνόλων δεδομένων.



Εικόνα 6.10 - Μέσο ποσοστό επιτυχίας (Accuracy)



Εικόνα 6.11 - Ποσοστιαία διαφορά σε κάθε σύνολο δεδομένων των αλγορίθμων Random Forest και Stochastic Forest (Accuracy)

Ακόμα παρατηρούμε πως ο Stochastic Forest παρουσιάζει την καλύτερη επίδοση σε 8 σύνολα δεδομένων, το πολυεπίπεδο Perceptron σε 13 και τα τυχαία δάση σε 10. Αυτό δείχνει πως ο αλγόριθμος που αναπτύχθηκε παρουσιάζει σταθερά υψηλά ποσοστά σε όλα τα σύνολα δεδομένων, όπου οι υπόλοιποι αλγόριθμοι μπορεί να έχουν σημαντικές διακυμάνσεις. Λόγω των σταθερά υψηλών ποσοστών του, ο Stochastic Forest τελικώς έχει τον μεγαλύτερο μέσο όρο, ενώ δεν παρουσιάζει το καλύτερο ποσοστό στα περισσότερα σύνολα δεδομένων.

Επιπλέον, συγκριτικά με τον Random Forest, ο αλγόριθμος που αναπτύξαμε παρουσιάζει καλύτερα ποσοστά σε 22 από τα 40 σύνολα δεδομένων, σε 1 έχουν ίδιο ποσοστό, ενώ ο Random Forest έχει καλύτερα ποσοστά σε 17. Τα συγκριτικά αποτελέσματα μεταξύ των δύο αλγορίθμων φαίνονται στην Εικόνα 6.11. Στο διάγραμμα απεικονίζεται η διαφορά των 2 αλγορίθμων σε κάθε σύνολο δεδομένων. Από τα αριστερά του κάθετου άξονα φαίνονται τα σύνολα δεδομένων που έχει καλύτερο ποσοστό Random Forest και από τα δεξιά αντίστοιχα για τον Stochastic Forest.

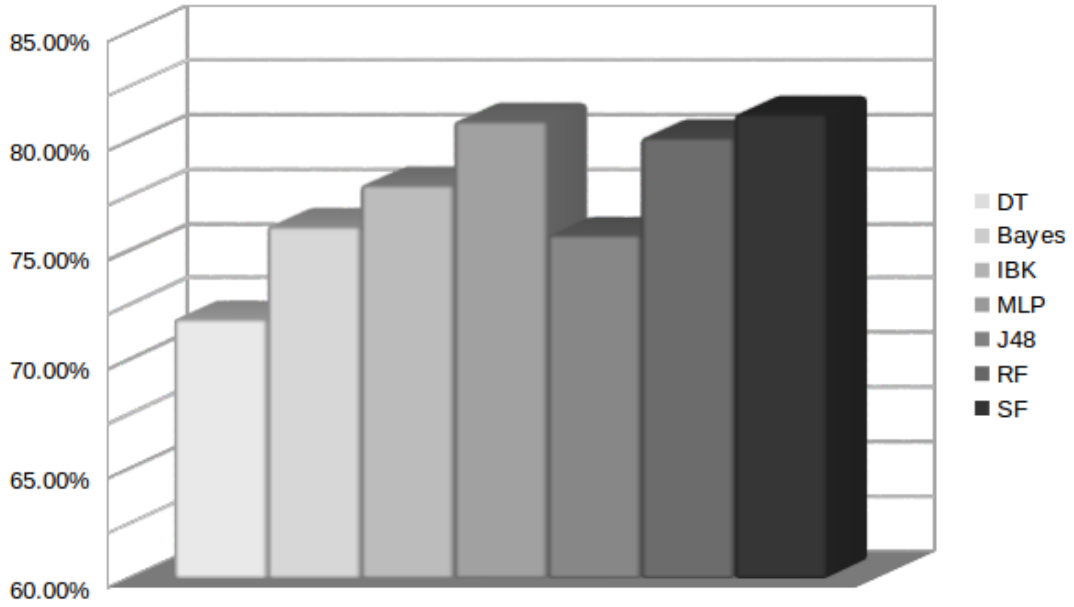
6.3.2 Sensitivity

Στην **ευαισθησία** (sensitivity), που φαίνεται στον Πίνακα 6.10, κατά μέσο όρο την καλύτερη επίδοση την παρουσιάζει ο Multi-Layer Perceptron (80.51%), την δεύτερη καλύτερη ο αλγόριθμος που αναπτύξαμε (79.9%) και ακολουθάνε τα τυχαία δάση (78.84%).

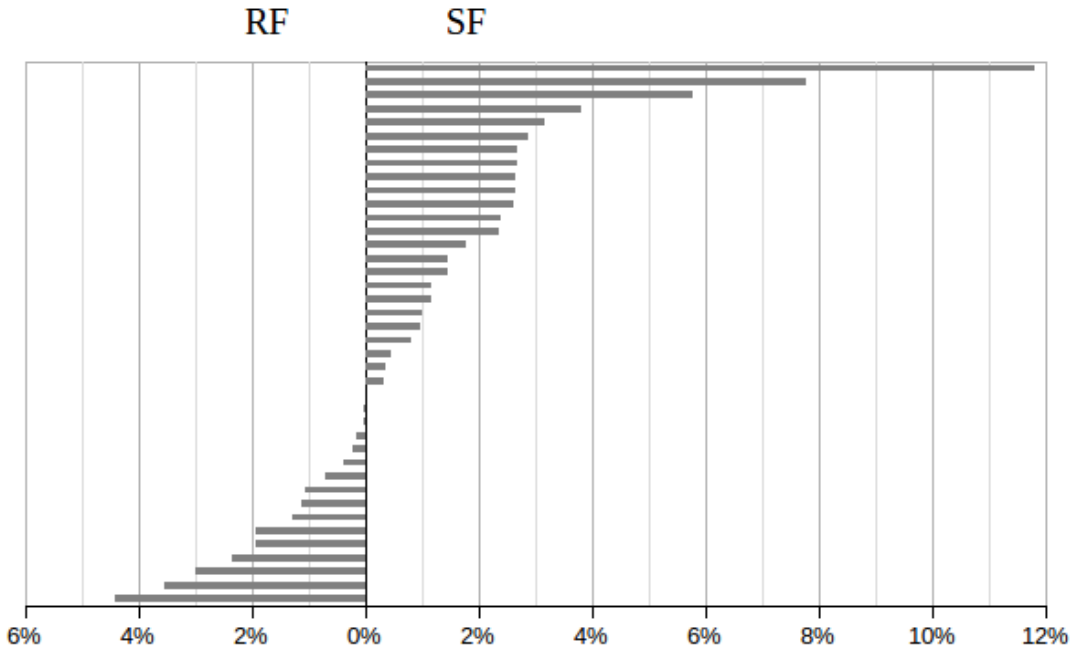
Ακόμα, παρατηρούμε διαφορά 1.06% στον μέσο όρο των αποτελεσμάτων σε όλα τα σύνολα δεδομένων, ανάμεσα στον αλγόριθμο των τυχαίων δασών και τα στοχαστικά δάση. Ο μέσος όρος της επίδοσης κάθε αλγορίθμου με βάση το μέτρο της ευαισθησίας παρουσιάζεται στην Εικόνα 6.12.

Ο αλγόριθμος που αναπτύξαμε, ο Stochastic Forest, έχει το υψηλότερο ποσοστό σε 10 από τα σύνολα δεδομένων, ο Multi-Layer Perceptron σε 13 και ο Random Forest σε 5. Συγκριτικά με τον Random Forest, ο αλγόριθμος που αναπτύξαμε παρουσιάζει καλύτερα ποσοστά σε 25 σύνολα δεδομένων, ενώ ο Random Forest σε 15. Τα συγκριτικά αποτελέσματα μεταξύ των δύο αλγορίθμων με βάση τα ποσοστά

τους στην ευαισθησία φαίνονται στην Εικόνα 6.13. Στο σχήμα απεικονίζεται η διαφορά των 2 αλγορίθμων σε κάθε σύνολο δεδομένων.



Εικόνα 6.12 - Μέσο ποσοστό ευαισθησίας (Sensitivity)

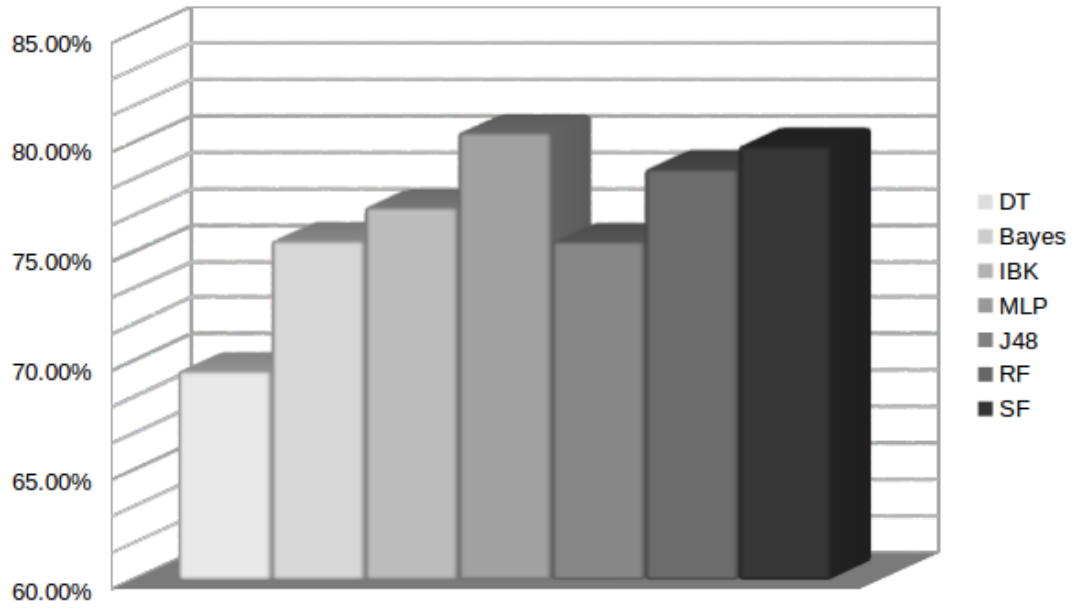


Εικόνα 6.13 - Ποσοστιαία διαφορά σε κάθε σύνολο δεδομένων των αλγορίθμων Random Forest και Stochastic Forest (Sensitivity)

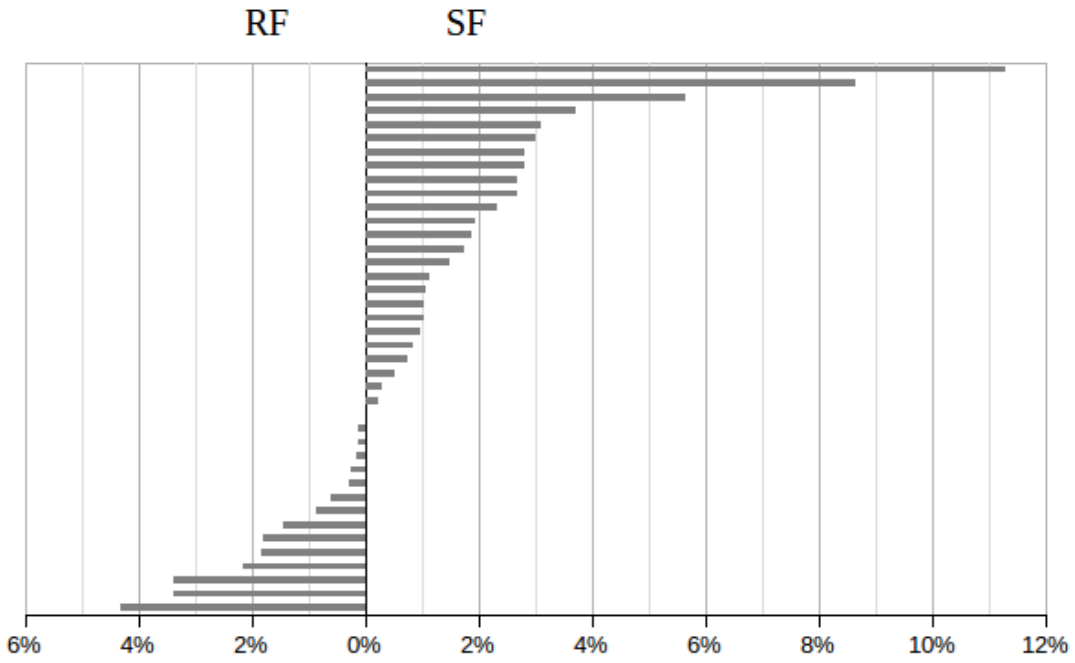
6.3.3 Precision

Στον Πίνακα 6.11 φαίνονται τα αποτελέσματα με βάση το μέτρο της **ακρίβειας** (precision). Παρατηρούμε πως κατά μέσο όρο την καλύτερη επίδοση την παρουσιάζουν τα στοχαστικά δάση (81.27%), ακολουθεί το πολυεπίπεδο Perceptron (80.94%) και την τρίτη καλύτερη επίδοση την έχουν πάλι τα τυχαία δάση (80.19%). Βλέπουμε πως και σε αυτό το μέτρο υπάρχει σημαντική διαφορά, 1.08%, ανάμεσα στα τυχαία δάση και τον αλγόριθμο που αναπτύξαμε, στους μέσους όρους των αποτελεσμάτων τους. Ο μέσος όρος της επίδοσης κάθε αλγορίθμου με βάση το μέτρο της ακρίβειας παρουσιάζεται στην Εικόνα 6.14.

Ο αλγόριθμος που αναπτύχθηκε στα πλαίσια της διπλωματικής παρουσιάζει καλύτερα ποσοστά σε 9 σύνολα δεδομένων, ο Multi-Layer Perceptron σε 13 και ο Random Forest σε 7. Συγκριτικά με τον Random Forest, ο αλγόριθμος των στοχαστικών δασών που αναπτύξαμε έχει πιο υψηλά ποσοστά σε 26 σύνολα δεδομένων, ενώ ο Random Forest σε 14. Τα συγκριτικά αποτελέσματα μεταξύ των δύο αλγορίθμων με βάση τα ποσοστά τους στο μέτρο της ακρίβειας φαίνονται στην Εικόνα 6.15. Στο διάγραμμα απεικονίζεται η διαφορά των 2 αλγορίθμων σε κάθε σύνολο δεδομένων.



Εικόνα 6.14 - Μέσο ποσοστό ακρίβειας (Precision)



Εικόνα 6.15 - Ποσοστιαία διαφορά σε κάθε σύνολο δεδομένων των αλγορίθμων Random Forest και Stochastic Forest (Precision)

6.3.4 Συμπεράσματα

Η βελτίωση που παρατηρείται σε σχέση με τα τυχαία δάση, που είναι και αυτός κατηγοριοποιητής ομάδων και χρησιμοποιεί δέντρα απόφασης, και στα τρία μέτρα απόδοσης είναι σημαντική και ενθαρρυντική. Ακόμα, παρατηρούμε πως ο αλγόριθμος που αναπτύξαμε παρουσιάζει καλύτερα αποτελέσματα από τους υπόλοιπους αλγορίθμους, τον Decision Table, τον Naive Bayes, τον IBK και τα δέντρα απόφασης υλοποιημένα στον J48.

Με βάση τα αποτελέσματα, επιβεβαιώνεται πως με την δημιουργία των N διαφορετικών δέντρων απόφασης από όλα τα χαρακτηριστικά, με στοχαστικό τρόπο και παίρνοντας υπόψιν μόνο τα καλύτερα (με βάση το κέρδος πληροφορίας) δημιουργούνται πιο δυνατά δέντρα απόφασης και ακολούθως πιο δυνατός κατηγοριοποιητής ομάδας. Ακόμα, προκύπτει πως η τυχαιότητα που εισάγεται είναι αρκετή ώστε οι κατηγοριοποιητές που δημιουργούνται να είναι διαφορετικοί και ως ένα βαθμό ανεξάρτητοι μεταξύ τους, ώστε να μειώνεται το σφάλμα.

Πίνακας 6.9 - Ποσοστό επιτυχημένων κατηγοριοποιήσεων (Accuracy)

	Dataset	DT	Bayes	IBK	MLP	J48	RF	SF
1	Abalone	52.98%	51.78%	50.35%	**55.97%**	52.81%	53.15%	54.30%
2	Balance scale	72.00%	90.40%	84.80%	**90.72%**	76.64%	81.44%	80.48%
3	Banknote	95.92%	84.26%	99.85%	**99.93%**	98.54%	99.13%	98.98%
4	Blood Transfusion	76.20%	75.40%	70.45%	**78.21%**	77.81%	72.73%	77.81%
5	Breast cancer (winsconsin)	95.28%	**95.99%**	95.14%	95.28%	94.56%	**95.99%**	95.85%
6	Breast cancer (wdbc)	94.02%	92.97%	95.96%	96.66%	93.15%	96.49%	**96.84%**
7	Breast_tissue	57.55%	70.75%	**71.70%**	66.98%	66.04%	67.92%	70.75%
8	Car evaluation	91.03%	85.53%	93.52%	**99.54%**	92.36%	94.50%	91.96%
9	Cardiotocography	67.73%	70.51%	78.65%	83.16%	83.58%	87.35%	**88.52%**
10	Contraceptive	**54.99%**	49.29%	43.31%	54.51%	53.22%	51.53%	51.19%
11	Dermatology	90.98%	97.54%	95.36%	**98.36%**	95.90%	96.99%	**98.36%**
12	Ecoli	76.79%	85.42%	80.36%	85.71%	84.23%	84.82%	**86.31%**
13	Fertility Diagnosis	88.00%	88.00%	83.00%	**90.00%**	85.00%	88.00%	88.00%
14	Glass	70.09%	49.53%	70.56%	67.29%	65.89%	**80.37%**	76.64%
15	GPS	80.37%	84.05%	82.21%	**85.28%**	80.37%	80.37%	82.21%
16	Ionosphere	89.46%	82.62%	86.32%	91.17%	74.36%	**92.87%**	92.59%
17	Iris	92.66%	96.00%	95.33%	**97.33%**	96.00%	94.00%	96.67%
18	Messidor features	62.38%	56.82%	61.34%	**72.02%**	64.38%	65.60%	66.99%
19	Optdigit	60.21%	90.82%	**98.83%**	97.94%	87.42%	94.94%	97.61%
20	Parkinson disease	81.03%	69.23%	**96.41%**	90.77%	80.51%	91.79%	92.31%
21	Parkinson speech	65.81%	59.85%	67.96%	68.96%	66.89%	**73.10%**	71.52%
22	Pima indian	71.22%	**76.30%**	70.18%	75.39%	73.83%	73.05%	75.78%
23	Seeds	86.19%	91.43%	94.29%	**95.24%**	91.90%	90.95%	93.81%
24	Semeion	50.52%	82.81%	86.79%	**90.57%**	63.52%	77.36%	88.89%
25	Sonar	69.23%	67.79%	**86.54%**	82.21%	71.15%	**86.54%**	85.10%
26	Soybean backup large	76.54%	91.53%	89.25%	91.21%	87.95%	**92.51%**	92.18%
27	Spect heart	**74.91%**	68.91%	68.91%	70.04%	70.41%	70.41%	73.03%
28	Spectf heart	79.40%	68.54%	70.41%	77.15%	74.91%	**82.02%**	80.52%
29	Statlog aust credit	84.64%	77.25%	80.00%	82.90%	85.22%	**87.25%**	83.91%
30	Statlog german credit	71.00%	**75.70%**	67.80%	70.90%	73.90%	73.90%	74.20%
31	Statlog heart	**84.81%**	83.70%	75.19%	78.15%	76.67%	82.22%	80.74%
32	Statlog img seg	84.94%	86.00%	94.69%	94.07%	93.46%	**96.00%**	95.80%
33	Statlog sat image	78.60%	79.20%	87.85%	87.85%	83.80%	89.25%	**90.10%**
34	Thyroid disease new	91.63%	96.74%	**97.21%**	96.74%	92.09%	94.88%	93.02%
35	Urban	63.69%	78.57%	76.79%	78.57%	79.17%	**85.12%**	82.74%
36	Vertebral column 3C	79.35%	83.23%	78.39%	**85.48%**	81.61%	84.19%	83.87%
37	Waveform	70.40%	81.02%	76.90%	82.40%	73.20%	81.88%	**84.28%**
38	Wine	88.76%	96.63%	94.94%	97.19%	93.82%	96.63%	**97.75%**
39	Yeast	54.92%	57.61%	52.29%	59.03%	55.66%	59.10%	**59.91%**
40	Zoo	86.75%	**96.39%**	95.18%	95.18%	90.36%	95.18%	95.18%
	Average	76.57%	79.15%	81.13%	83.90%	79.56%	83.54%	**84.17%**

Πίνακας 6.10 - Ευαισθησία (Sensitivity)

	Dataset	DT	Bayes	IBK	MLP	J48	RF	SF
1	Abalone	52.53%	53.53%	50.43%	**55.70%**	52.80%	53.03%	54.00%
2	Balance scale	52.07%	65.40%	61.37%	**83.13%**	55.43%	58.90%	58.22%
3	Banknote	95.70%	83.75%	99.85%	**99.90%**	98.55%	99.10%	98.97%
4	Blood Transfusion	50.00%	56.40%	55.70%	61.95%	65.95%	58.35%	**66.12%**
5	Breast cancer (winsconsin)	94.85%	**96.35%**	94.40%	94.95%	94.05%	95.55%	95.56%
6	Breast cancer (wdbc)	93.10%	92.30%	95.60%	96.30%	93.05%	95.95%	**96.43%**
7	Breast_tissue	55.03%	70.23%	**71.40%**	65.88%	64.43%	67.00%	69.68%
8	Car evaluation	78.65%	58.90%	70.13%	**98.93%**	82.88%	85.48%	85.85%
9	Cardiotocography	51.21%	71.86%	72.63%	77.50%	77.31%	81.50%	**83.87%**
10	Contraceptive	52.27%	50.23%	41.77%	**54.00%**	50.73%	48.80%	48.43%
11	Dermatology	87.77%	97.67%	95.25%	**98.22%**	95.53%	96.70%	98.16%
12	Ecoli	40.14%	60.75%	59.24%	57.89%	56.64%	59.35%	**61.97%**
13	Fertility Diagnosis	50.00%	50.00%	57.95%	**72.75%**	48.30%	60.80%	60.80%
14	Glass	56.90%	53.53%	67.65%	58.75%	68.95%	**76.33%**	71.93%
15	GPS	79.80%	83.00%	82.20%	**85.00%**	79.80%	80.05%	81.84%
16	Ionosphere	89.80%	83.45%	82.20%	88.25%	89.45%	**91.85%**	90.73%
17	Iris	92.67%	96.00%	95.30%	**97.30%**	96.00%	94.00%	96.67%
18	Messidor features	63.25%	59.05%	61.45%	**71.90%**	64.60%	65.90%	67.35%
19	Optdigit	60.14%	90.84%	**98.82%**	97.94%	87.39%	94.92%	97.60%
20	Parkinson disease	73.35%	76.80%	**96.20%**	88.95%	73.05%	87.55%	87.88%
21	Parkinson speech	64.85%	57.60%	67.25%	68.60%	65.25%	**71.95%**	70.02%
22	Pima indian	67.00%	**72.80%**	66.20%	72.00%	70.55%	68.05%	71.88%
23	Seeds	86.20%	91.43%	94.30%	**95.27%**	91.87%	90.93%	93.81%
24	Semeion	49.40%	82.34%	85.73%	**90.04%**	62.12%	76.16%	87.99%
25	Sonar	68.95%	68.75%	**86.20%**	82.05%	71.15%	86.15%	84.87%
26	Soybean backup large	69.03%	89.12%	87.24%	89.34%	82.89%	89.74%	**90.92%**
27	Spect heart	**72.55%**	67.85%	66.60%	68.95%	68.45%	68.70%	71.36%
28	Spectf heart	53.35%	**76.80%**	59.80%	68.10%	62.00%	62.45%	59.46%
29	Statlog aust credit	84.90%	75.40%	79.85%	82.70%	84.90%	**87.35%**	83.83%
30	Statlog german credit	59.95%	**68.45%**	61.80%	65.15%	66.70%	62.50%	65.67%
31	Statlog heart	**84.35%**	83.25%	75.00%	78.15%	77.15%	81.55%	80.50%
32	Statlog img seg	84.83%	86.84%	94.87%	94.29%	93.69%	**96.19%**	95.96%
33	Statlog sat image	75.63%	78.38%	86.67%	86.08%	81.68%	87.13%	**88.15%**
34	Thyroid disease new	86.43%	93.43%	**96.13%**	96.10%	88.87%	91.23%	89.30%
35	Urban	61.98%	80.28%	77.42%	78.91%	79.60%	**85.50%**	83.14%
36	Vertebral column 3C	69.87%	79.33%	75.13%	**82.43%**	76.67%	79.90%	79.89%
37	Waveform	72.77%	80.97%	76.90%	83.80%	75.93%	81.87%	**84.25%**
38	Wine	88.83%	96.90%	95.77%	97.43%	93.40%	96.87%	**98.03%**
39	Yeast	37.75%	50.46%	52.26%	47.03%	51.63%	49.64%	**55.42%**
40	Zoo	75.07%	90.71%	88.67%	88.67%	81.27%	88.67%	**89.49%**
	Average	69.57%	75.53%	77.08%	**80.51%**	75.52%	78.84%	79.90%

Πίνακας 6.11 - Ακρίβεια (Precision)

	Dataset	DT	Bayes	IBK	MLP	J48	RF	SF
1	Abalone	52.43%	49.63%	50.57%	**55.80%**	52.67%	53.13%	53.99%
2	Balance scale	48.83%	60.27%	59.63%	**80.13%**	52.97%	59.77%	59.65%
3	Banknote	96.00%	84.25%	99.85%	**99.95%**	98.50%	99.10%	98.97%
4	Blood Transfusion	38.10%	62.50%	56.75%	**69.55%**	68.75%	60.15%	68.81%
5	Breast cancer (winsconsin)	94.75%	95.00%	94.80%	94.65%	93.95%	**95.55%**	95.29%
6	Breast cancer (wdbc)	94.05%	92.60%	95.70%	96.55%	92.45%	96.50%	**96.79%**
7	Breast_tissue	52.73%	69.30%	**71.33%**	65.03%	66.07%	67.90%	68.14%
8	Car evaluation	83.00%	78.83%	94.78%	**99.03%**	81.80%	84.48%	85.23%
9	Cardiotocography	64.24%	64.81%	75.36%	78.51%	79.79%	86.62%	**87.77%**
10	Contraceptive	53.13%	49.47%	41.63%	**53.57%**	51.33%	49.47%	48.61%
11	Dermatology	91.47%	97.37%	94.90%	**98.13%**	95.15%	96.13%	97.88%
12	Ecoli	47.78%	64.05%	58.05%	58.58%	55.95%	61.31%	**63.18%**
13	Fertility Diagnosis	44.00%	44.00%	58.60%	**76.65%**	43.80%	70.20%	70.21%
14	Glass	**77.56%**	50.18%	67.41%	58.72%	64.60%	70.48%	76.15%
15	GPS	81.10%	**87.80%**	82.10%	85.50%	81.10%	80.50%	82.45%
16	Ionosphere	87.10%	81.30%	88.30%	92.90%	91.85%	92.60%	**93.14%**
17	Iris	92.67%	96.00%	95.30%	**97.30%**	96.00%	94.00%	96.68%
18	Messidor features	64.20%	68.50%	61.35%	**71.95%**	64.60%	65.95%	67.46%
19	Optdigit	62.21%	91.18%	**98.84%**	97.94%	87.39%	94.94%	97.63%
20	Parkinson disease	74.45%	69.90%	**94.40%**	86.95%	73.75%	89.90%	90.90%
21	Parkinson speech	65.05%	58.50%	67.30%	68.45%	66.20%	**72.65%**	71.20%
22	Pima indian	68.10%	**74.00%**	66.95%	72.90%	71.10%	70.35%	73.46%
23	Seeds	86.37%	91.43%	94.33%	**95.27%**	91.90%	90.97%	93.80%
24	Semeion	51.22%	83.65%	88.08%	**90.94%**	63.05%	78.61%	89.90%
25	Sonar	69.05%	69.85%	86.85%	82.20%	71.05%	**87.00%**	85.18%
26	Soybean backup large	81.54%	88.66%	87.35%	89.47%	81.56%	89.39%	**90.43%**
27	Spect heart	**74.85%**	67.90%	67.90%	69.05%	69.50%	69.45%	72.27%
28	Spectf heart	65.25%	67.60%	58.35%	66.25%	61.85%	**73.95%**	69.65%
29	Statlog aust credit	84.50%	79.60%	79.75%	82.70%	84.90%	**87.05%**	83.69%
30	Statlog german credit	63.90%	**70.95%**	61.75%	65.25%	68.55%	69.05%	68.93%
31	Statlog heart	**84.85%**	83.65%	74.85%	77.90%	76.35%	82.30%	80.50%
32	Statlog img seg	86.21%	87.16%	94.93%	94.53%	93.77%	**96.30%**	96.03%
33	Statlog sat image	78.00%	78.20%	86.70%	86.22%	81.98%	88.28%	**89.35%**
34	Thyroid disease new	90.60%	**97.47%**	96.33%	95.27%	89.80%	94.90%	91.53%
35	Urban	71.82%	80.27%	77.94%	79.58%	80.63%	**85.40%**	83.24%
36	Vertebral column 3C	75.03%	79.53%	75.97%	**81.63%**	76.33%	79.97%	79.37%
37	Waveform	72.77%	84.13%	76.90%	83.80%	75.87%	81.97%	**84.29%**
38	Wine	90.27%	96.63%	94.83%	97.03%	94.33%	96.57%	**97.62%**
39	Yeast	46.62%	55.97%	51.58%	49.18%	53.97%	55.67%	**58.67%**
40	Zoo	69.97%	**93.21%**	92.10%	92.65%	84.57%	89.13%	92.86%
	Average	71.89%	76.13%	78.01%	80.94%	75.74%	80.19%	**81.27%**

7 *Επίλογος*

Σε αυτό το Κεφάλαιο θα γίνει η σύνοψη των αποτελεσμάτων της διπλωματικής και τα συμπεράσματα που προκύπτουν από αυτά. Ακόμα, θα αναφερθούμε σε μελλοντικές επεκτάσεις για τους αλγορίθμους που αναπτύχθηκαν.

Στα πλαίσια της παρούσας διπλωματικής εργασίας αναπτύχθηκαν 2 νέοι αλγόριθμοι εξόρυξης δεδομένων, πιο συγκεκριμένα στο πρόβλημα της κατηγοριοποίησης. Αυτό σημαίνει πως στόχος των αλγορίθμων που αναπτύχθηκαν είναι η επιτυχής ανάπτυξη μοντέλων μέσω της διαδικασίας της εκπαίδευσης με χρήση δεδομένων με γνωστή ετικέτα κατηγορίας, για την κατηγοριοποίηση μελλοντικών δεδομένων με άγνωστη κατηγορία.

Ο πρώτος αλγόριθμος ονομάζεται Constrained K – Means Classification και ανήκει στην κατηγορία αλγορίθμων κατηγοριοποίησης μέσω ομαδοποίησης (classification via clustering). Ο αλγόριθμος αυτός βασίζεται σε μια γνωστή τεχνική ομαδοποίησης βασισμένης σε πρότυπο, τον K - Means. Επειδή η ομαδοποίηση είναι μη εποπτευόμενη μάθηση ενώ η κατηγοριοποίηση είναι εποπτευόμενη, δηλαδή υπάρχει ήδη και χρησιμοποιείται κάποια γνώση για της κατηγορίες, η υπάρχουσα γνώση χρησιμοποιείται μέσω αξιοποίησης μέτρων θέσης και διασποράς, για τον καθορισμό περιορισμών και βαρών σε κάθε χαρακτηριστικό.

Ο δεύτερος αλγόριθμος που αναπτύχθηκε, ο Stochastic Forest, είναι αλγόριθμος κατηγοριοποίησης με την μέθοδο των ομάδων. Είναι βασισμένος στην κατασκευή

δέντρων απόφασης στα πρότυπα του γνωστού αλγορίθμου τυχαίων δασών Random Forest. Ο αλγόριθμος αναπτύσσει Ν διαφορετικούς κατηγοριοποιητές, με κατασκευή δέντρων απόφασης με διαφορετικό τρόπο. Η κατηγοριοποίηση των μελλοντικών δεδομένων με άγνωστη ετικέτα κατηγορίας, γίνεται ξεχωριστά από τον κάθε κατηγοριοποιητή και στο τέλος η πλειοψηφούσα κατηγορία για το κάθε στιγμιότυπο είναι η τελική του κατηγορία. Η διαφορά με τον Random Forest και άλλους αντίστοιχους αλγορίθμους, βρίσκεται στον τρόπο παραγωγής των Ν διαφορετικών δέντρων.

Η αξιολόγηση των αλγορίθμων που αναπτύχθηκαν στα πλαίσια της παρούσας διπλωματικής πραγματοποιήθηκε σε πολλαπλά σύνολα δεδομένων από το UCI machine learning repository. Στο πλαίσιο αυτό χρησιμοποιήθηκαν, μεταξύ άλλων, τα πιο γνωστά και τα πιο χρησιμοποιημένα σύνολα δεδομένων. Συνολικά χρησιμοποιήθηκαν 40 σύνολα δεδομένων για την αξιολόγηση του κάθε αλγορίθμου ώστε τα αποτελέσματα να είναι πιο αξιόπιστα. Επιπλέον, οι αλγόριθμοι αξιολογήθηκαν με τρεις διαφορετικές μετρικές απόδοσης. Οι μετρικές που λήφθηκαν υπόψιν είναι το accuracy, το sensitivity και το precision. Η αξιολόγηση πραγματοποιήθηκε με πολλαπλές εκτελέσεις σε κάθε σύνολο δεδομένων, για μεγαλύτερη αξιοπιστία.

Ακόμα, στα πλαίσια της παρούσας διπλωματικής εργασίας, ο αλγόριθμος Constrained K – Means Classification εφαρμόστηκε σε ένα ιατρικό πρόβλημα, αυτό της εκτίμησης της σοβαρότητας της ίνωσης του ήπατος από εικόνες βιοψίας. Αξιολογήθηκε η ικανότητα επίλυσης του προβλήματος από τον αλγόριθμο, όσο και η καταλληλότητά του συγκριτικά με άλλους αλγορίθμους.

Όσον αφορά τα αποτελέσματα της αξιολόγησης, αυτά είναι ενθαρρυντικά καθώς παρουσιάζεται βελτίωση σε σχέση με τους βασικούς αλγορίθμους στην αντίστοιχη μέθοδο κατηγοριοποίησης, παίρνοντας υπόψιν όλες τις μετρικές απόδοσης που χρησιμοποιήθηκαν.

Ο Constrained K-Means παρουσιάζει ικανοποιητικά αποτελέσματα και σημαντική βελτίωση συγκρινόμενος με το παραδοσιακό classification via clustering με τον απλό K-Means. Στο μέτρο του ποσοστού επιτυχίας κατά μέσο όρο παρουσιάζει βελτίωση14.52% , στην ευαισθησία 14.77% και στην ακρίβεια 13.84%. Με βάση τις μετρήσεις, επιβεβαιώνεται πως η εισαγωγή των περιορισμών και των βαρών βελτιώνει σημαντικά την τεχνική της κατηγοριοποίησης μέσω ομαδοποίησης .

Ακόμα, στην αξιολόγηση στο ιατρικό πρόβλημα ο αλγόριθμος αξιολογήθηκε τόσο ως ικανός όσο και κατάλληλος για την εφαρμογή του σε αυτό. Ο αλγόριθμος πετυχαίνει κατά μέσο όρο 97.79% ποσοστό επιτυχίας, 89.68% ευαισθησία και 93.09% ακρίβεια, ποσοστά που επιβεβαιώνουν τη χρηστικότητα των αποτελεσμάτων για τη διάγνωση. Ακόμα, το υπολογιζόμενο cpa παρουσιάζει ακρίβεια άνω του 99% στις 4 από τις 8 εικόνες βιοψίας.

Όσον αφορά την αξιολόγηση του αλγορίθμου Stochastic Forest, η βελτίωση που παρατηρείται σε σχέση με τον αλγόριθμο των τυχαίων δασών, και στα τρία μέτρα απόδοσης είναι σημαντική και ενθαρρυντική. Η διαφορά με τον Random Forest είναι 0.63% στο ποσοστό επιτυχίας, 1.06% στην ευαισθησία και 1.08% στην ακρίβεια.

Ακόμα, παρατηρούμε πως ο αλγόριθμος που αναπτύξαμε παρουσιάζει καλύτερα αποτελέσματα από τους υπόλοιπους αλγορίθμους, τον Decision Table, τον Naive Bayes, τον IBK και τα δέντρα απόφασης υλοποιημένα στον J48. Επιπλέον, παρατηρούνται αυξημένα ποσοστά σε σχέση και με τον MLP, στο ποσοστό επιτυχίας και στην ακρίβεια, ενώ υπολείπεται οριακά (0.6%) στην ευαισθησία.

Με βάση τα αποτελέσματα, επιβεβαιώνεται πως με την δημιουργία των N διαφορετικών δέντρων απόφασης από όλα τα χαρακτηριστικά, με στοχαστικό τρόπο και παίρνοντας υπόψιν μόνο τα καλύτερα, δημιουργούνται πιο δυνατά δέντρα απόφασης και συνεπώς πιο δυνατός κατηγοριοποιητής ομάδας. Επίσης, προκύπτει πως η τυχαιότητα που εισάγεται είναι αρκετή ώστε οι κατηγοριοποιητές που δημιουργούνται να είναι διαφορετικοί και ως ένα βαθμό ανεξάρτητοι μεταξύ τους, ώστε να μειώνεται το σφάλμα.

Όσον αφορά τις μελλοντικές επεκτάσεις, για τον αλγόριθμο Costrained K-Means Classification υπάρχει η δυνατότητα διερεύνησης πιο ελαστικών περιορισμών για τα κέντρα, όσο η χρήση διαφορετικών μέτρων ομοιότητας με τις ομάδες, αντί για την βεβαρημένη ευκλείδεια απόσταση. Ακόμα, θα εξεταστούν τεχνικές εύρεσης και χειρισμού ακραίων τιμών.

Για τον αλγόριθμο Stochastic Forest, υπάρχουν διάφορες προσεγγίσεις για την εισαγωγή της στοχαστικότητας που μπορούν να εφαρμοστούν και να διερευνηθούν, π.χ. για τα συνεχή χαρακτηριστικά να παίρνονται υπόψιν όλοι οι καλοί διαχωρισμοί και όχι μόνο ο καλύτερος. Ακόμα, θα γίνει εφαρμογή και αξιολόγηση τεχνικών διαχείρισης του συνόλου δεδομένων για την παραγωγή των κατηγοριοποιητών (π.χ. Bagging και Boosting), συνδυασμένων με την προταθείσα τεχνική. Επιπρόσθετα,

στους κατηγοριοποιητές ομάδας υπάρχουν και διαφορετικές προσεγγίσεις για τον τρόπο επιλογής κατηγορίας, π.χ. βεβαρημένες ψήφοι από κάθε κατηγοριοποιητή ανάλογα με την “δύναμή” του, οι οποίες και θα εξεταστούν.

Τέλος, θα γίνουν προσπάθειες επιτάχυνσης και των δύο αλγορίθμων με τεχνικές παραλληλοποίησης. Σε αυτή τη κατεύθυνση, θα διερευνηθούν σχήματα παραλληλοποίησης αλλά και συστήματα διαφορετικών αρχιτεκτονικών.

8 *Βιβλιογραφία*

[1] Tan, Pang-Ning. *Introduction to data mining*. Pearson Education India, 2006.

[2] Hand, David J., Heikki Mannila, and Padhraic Smyth. *Principles of data mining*. MIT press, 2001.

[3] Gan, Guojun, Chaoqun Ma, and Jianhong Wu. *Data clustering: theory, algorithms, and applications*. Society for Industrial and Applied Mathematics, 2007.

[4] MacQueen, James. *Some methods for classification and analysis of multivariate observations.* Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. Vol. 1. No. 14. 1967.

[5] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. . A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd.* Vol. 96. No. 34. pp. 226-231. 1996, August.

[6] Rokach, Lior, and Oded Maimon. *Clustering methods*. *Data mining and knowledge discovery handbook*. Springer US, pp. 321-352. 2005.

[7] Bezdek, James C. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 2013.

[8] Vapnik, Vladimir Naumovich, and Vlamimir Vapnik. *Statistical learning theory*. Vol. 1. New York: Wiley, 1998.

[9] Quinlan, J. Ross. *Induction of decision trees*. *Machine learning* 1.1. pp. 81-106. 1986.

[10] Craven, Mark W., and Jude W. Shavlik. *Using neural networks for data mining*. Future generation computer systems 13.2-3. pp. 211-229. 1997.

[11] Rosenblatt, Frank. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. No. VG-1196-G-8. CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[12] Schmidhuber, Jürgen. *Deep learning in neural networks: An overview*. Neural networks. Vol 61. pp .85-117. .2015.

[13] Rokach, Lior. *Ensemble-based classifiers*. Artificial Intelligence Review 33.1. pp. 1-39. 2010.

[14] Hearst, Marti A., et al. *Support vector machines*. IEEE Intelligent Systems and their applications 13.4. pp. 18-28.. 1998.

[15] Apté, Chidanand, Fred Damerau, and Sholom M. Weiss. *Automated learning of decision rules for text categorization.* ACM Transactions on Information Systems (TOIS) 12.3. pp. 233-251.1994.

[16] Dimosthenis C. Tsouros, Panagiotis N. Smyrlis, Markos G. Tsipouras,

Dimitrios G. Tsalikakis, Nikolaos Giannakeas, Alexandros T. Tzallas, Pinelopi Manousou. *Automated collagen proportional area extraction in liver biopsy images using a novel classification via clustering algorithm*. Computer Based Medical Systems (CBMS) 2017.

[17] Breiman, Leo. *Random forests*. Machine learning. *Vol.* 45.1. pp. 5-32 . 2001.

[18] Lichman, M. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. 2013.

[19] Refaeilzadeh, Payam, Lei Tang, and Huan Liu. *Cross-validation*. Encyclopedia of database systems. Springer US. pp 532-538. 2009.

[20] *Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J.. Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software. 1984.*

[21] Quinlan, J. Ross. *C4.5: Programming for machine learning.* Morgan Kauffmann 38 . 1993.

[22] Fawcett, Tom. *An introduction to ROC analysis*. Pattern recognition letters Vol27.8. pp. 861-874. 2006.

[23] Powers, David Martin. *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation*. 2011.

[24] Lloyd, Stuart. *Least squares quantization in PCM*. IEEE transactions on information theory Vol 28.2. pp. 129-137. 1982.

[25] Wagstaff, Kiri, et al. *Constrained k-means clustering with background knowledge*. ICML. Vol. 1. 2001.

[26] Davidson, Ian, and S. S. Ravi. *Clustering with constraints: Feasibility issues and the k-means algorithm.* SIAM international conference on data mining. Society for Industrial and Applied Mathematics, 2005.

[27] Bradley, P. S., K. P. Bennett, and Ayhan Demiriz. *Constrained k-means clustering*. Microsoft Research, Redmond. pp. 1-8. 2000.

[28] Likas, Aristidis, Nikos Vlassis, and Jakob J. Verbeek. *The global k-means clustering algorithm.* Pattern recognition. Vol. 36.2. pp. 451-461. 2003.

[29] Bezdek, James C., Robert Ehrlich, and William Full. *FCM: The fuzzy c-means clustering algorithm*. Computers & Geosciences. Vol. 10.2-3. pp. 191-203. 1984.

[30] Lopez, Manuel Ignacio *Classification via clustering for predicting final marks based on student participation in forums*. International Educational Data Mining Society. 2012.

[31] Gorsevski, Pece V., Paul E. Gessler, and Piotr Jankowski. *Integrating a fuzzy k-means classification and a Bayesian approach for spatial prediction of landslide hazard*. Journal of geographical systems. Vol. 5.3. pp. 223-251. 2003.

[32] Erman, Jeffrey, Martin Arlitt, and Anirban Mahanti. *Traffic classification using clustering algorithms.* 2006 SIGCOMM workshop on Mining network data. ACM. 2006.

[33] Panda, Mrutyunjaya, and Manas Ranjan Patra. *A novel classification via clustering method for anomaly based network intrusion detection system.* International Journal of Recent Trends in Engineering. Vol. 2.1. pp. 1-6. 2009.

[34] Burrough, Peter A., Pauline FM van Gaans, and R. A. MacMillan. *High-resolution landform classification using fuzzy k-means.* Fuzzy sets and systems. Vol. 113.1 pp. 37-52. 2000.

[35] Evans, Reuben, Bernhard Pfahringer, and Geoffrey Holmes. *Clustering for classification.* Information Technology in Asia (CITA 11), 2011 7th International Conference on. IEEE, 2011.

[36] Breiman, Leo. *Bagging predictors*. Machine learning Vol. 24.2. pp. 123-140. 1996.

[37] Freund, Yoav, Robert Schapire, and Naoki Abe. *A short introduction to boosting*. *Journal-Japanese Society For Artificial Intelligence Vol.* 14.771-780. pp. 1612. 1999.

[38] Ho, Tin Kam. *The random subspace method for constructing decision forests*. IEEE transactions on pattern analysis and machine intelligence Vol. 20.8. pp. 832-844. 1998

[39] Rodriguez, Juan José, Ludmila I. Kuncheva, and Carlos J. Alonso. *Rotation forest: A new classifier ensemble method*. IEEE transactions on pattern analysis and machine intelligence Vol. 28.10. pp. 1619-1630. 2006

[40] Dietterich, Thomas G. *An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization.* Machine learning Vol. 40.2. pp. 139-157. 2000

[41] Geurts, Pierre, Damien Ernst, and Louis Wehenkel. *Extremely randomized trees.* Machine learning. Vol. 63.1. pp. 3-42. 2006

[42] Freund, Yoav, and Robert E. Schapire. *Experiments with a new boosting algorithm.* Icml. Vol. 96. 1996.

[43] Pearson, Karl. *LIII. On lines and planes of closest fit to systems of points in space.* The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. Vol. 2.11. pp. 559-572. 1901

[44] Jolliffe, Ian T. *Principal Component Analysis and Factor Analysis*. Principal component analysis. Springer New York, pp. 115-128. 1986.

[45] Banfield, Robert E., *A comparison of decision tree ensemble creation techniques.* IEEE transactions on pattern analysis and machine intelligence. Vol. 29.1. pp.173-180. 2007.

[46] Singh, Kehar, Dimple Malik, and Naveen Sharma. *Evolving limitations in K-means algorithm in data mining and their removal*. *International Journal of Computational Engineering & Management Vol*12. pp. 105-109. 2011.

[47] Bradley, Paul S., and Usama M. Fayyad. *Refining Initial Points for K-Means Clustering. ICML*. Vol. 98. 1998.

[48] Gower, John Clifford. *Euclidean distance geometry*. Math. Sci Vol. 7.1 pp. 1-14. 1982.

[49] Mahalanobis, Prasanta Chandra. *On the generalised distance in statistics.* Proceedings of the National Institute of Sciences of India, *pp.* 49-55. 1936.

[50] De Maesschalck, Roy, Delphine Jouan-Rimbaud, and Désiré L. Massart. *The mahalanobis distance*. Chemometrics and intelligent laboratory systems. Vol. 50.1 pp. 1-18. 2000.

[51] Stroustrup, Bjarne. *The C++ programming language*. Pearson Education India, 2000.

[52] Kohavi, Ron. *The power of decision tables*. Machine learning: ECML-95. pp. 174-189. 1995.

[53] John, George H., and Pat Langley. *Estimating continuous distributions in Bayesian classifiers.* Proceedings of the Eleventh conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1995.

[54] Aha, David W., Dennis Kibler, and Marc K. Albert. *Instance-based learning algorithms.* Machine learning Vol. 6.1. pp. 37-66. 1991.

[55] Belue, Lisa M. *Multilayer Perceptrons for Classification*. No. AFIT/GOR/ENS/92M-02. AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING, 1992.

[56] Holmes, Geoffrey, Andrew Donkin, and Ian H. Witten. *Weka: A machine learning workbench*. Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on. IEEE, 1994.

[57] Y. Huang, W. B. de Boer, L. A. Adams, G. MacQuillan, M. K. Bulsara and G. P. Jeffrey, *Image analysis of liver biopsy samples measures fibrosis and predicts clinical outcome.* Journal of Hepatology, vol. 61, pp. 22–27,. 2014.

[58] P. Manousou, A. P. Dhillon, G. Isgro, V. Calvaruso, T. V. Luong, E. Tsochatzis, E. Xirouchakis, G. Kalambokis, T. J. Cross, N. Rolando, J. O'Beirne, D. Patch, D. Thornburn, and A. K. Burroughs, *Digital image analysis of liver collagen predicts clinical outcome of recurrent hepatitis C virus 1 year after liver transplantation.* Liver Transplantation, vol. 17, pp. 178–188, 2011.

[59] Black, Paul E. *Manhattan distance.* Dictionary of Algorithms and Data Structures 18. pp. 2012. 2006.

9 Παραρτήματα

9.1 Παράρτημα 1

Αρχεία:

dataset.h
dataset.cpp
ckmeans.h
ckmeans.cpp
main.cpp

9.1.1 dataset.h

```
#include <vector>
#include <string>

using namespace std;

// Classes.....

class Data_point {

    private:
```

```
    vector<double> attribute;   // data point attributes
    string Class;               // data point class
    string assigned_class;      // data point assigned class

    public:

    Data_point();               // Constructor
    string get_class();         // returns class of data point
    void set_class(string c);   // set the class of the data point
    string get_assigned_class();    // returns assigned class of data point
    void assign(string c);      // assign data point
    void push_attribute(double a);  // push attribute to vector
    void erase_attribute(int i);  // extract attribute i from data point
    double get_attribute(int i);  // get attribute i from data point
    void set_attribute(int i, double a); // set value of attribute i to value of a
    vector<double> get_attribute(); // get all attributes from data point

};

class Dataset {

    private:

    string name;                // dataset name
    int instances;              // dataset number of instances
    int attributes;             // dataset number of attributes
    vector<string> classes;     // dataset classes
    vector<Data_point> data;    // dataset data points

    public:

    Dataset();              // Constructor

    // datasets info ------------------------------------------------

    string get_name();      // return the name of dataset
    void set_name(string n);    // set the name of dataset
```

```
    int get_instances();       // get number of instances
    void set_instances(int i);     // set number of instances [ DO NOT USE THIS ]
    int get_attributes();      // get number of attributes
    void set_attributes(int a);    // set number of attributes

    // class ---------------------------------------------------------

    void push_class(string c);   // push a new found class
    string get_class(int i);   // get class i
    vector<string> get_classes();    // get all classes
    void set_classes(vector<string> c); // set all classes
    int num_of_classes();          // get number of classes
    vector<Dataset> classes_to_datasets(); // get classes in separate datasets

    // data ----------------------------------------------------------

    void push_data(Data_point d);  // push new data point
    void erase_data(int i);   // extract data point i from dataset
    void insert_data(vector<Data_point> d); // insert a vector of data points in the end of vector data
    Data_point get_data(int i);  // get data point i
    void set_data(int i, Data_point d);   // set data point i
    vector<Data_point> get_data();      // get all data

    // other ---------------------------------------------------------

    void print_info();         // print information of dataset
    void print_data();         // print data
    void clear();              // clear dataset
    void arff_read(string filename);      // read arrf file, and save dataset
    void preprocess();         // min - max normalization to dataset. min = 0, max = 100

};

// Functions.....

string remove_spaces(string input);// remove all spaces in string input
```

```
vector<Dataset> split_dataset(Dataset dataset);// split dataset
vector<Dataset> split_dataset_by_id(Dataset dataset); // split dataset by id
vector<Dataset> make_datasets(vector<Dataset> datasets, int folds, int i);  //    make    training
and test dataset. i fold index
```

9.1.2 dataset.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>

#include "dataset.h"

using namespace std;

// Class Data_point member functions ------------------------------------------------------

Data_point::Data_point(){}

string Data_point::get_class(){
    return Class;
}

void Data_point::set_class(string c){
    Class = c;
}

string Data_point::get_assigned_class(){
    return assigned_class;
}

void Data_point::assign(string c){
    assigned_class = c;
}
```

```
void Data_point::push_attribute(double a){
    attribute.push_back(a);
}

void Data_point::erase_attribute(int i){
    attribute.erase(attribute.begin() + i);
}

void Data_point::set_attribute(int i, double a){
    attribute[i] = a;
}

double Data_point::get_attribute(int i){
    return attribute[i];
}

vector<double> Data_point::get_attribute(){
    return attribute;
}

// Class Dataset member functions ------------------------------------------------------------

Dataset::Dataset(){
    instances = 0;
    attributes = 0;
}

string Dataset::get_name(){
    return name;
}

void Dataset::set_name(string n){
    name = n;
}

int Dataset::get_instances(){
```

```
    return instances;
}

void Dataset::set_instances(int i){
    instances = i;
}

int Dataset::get_attributes(){
    return attributes;
}

void Dataset::set_attributes(int a){
    attributes = a;
}

void Dataset::push_class(string c){
    classes.push_back(c);
}

string Dataset::get_class(int i){
    return classes[i];
}

vector<string> Dataset::get_classes(){
    return classes;
}

void Dataset::set_classes(vector<string> c){
    classes = c;
}

int Dataset::num_of_classes(){
    return classes.size();
}

vector<Dataset> Dataset::classes_to_datasets(){
```

```
    int i, j, deb;
    vector<Dataset> classes_data( num_of_classes() ); // initialize vector of classes datasets

    for (j = 0; j < num_of_classes(); j++){
        classes_data[j].set_attributes(get_attributes());
    }

    for (i = 0; i < instances; i++){    // for all data points
        for (j = 0; j < num_of_classes(); j++){   // for all classes
            if ( get_class(j).compare( data[i].get_class() ) == 0 ){    // if in this class
                classes_data[j].push_data( data[i] );                   // push data to class's vector
            }
        }
    }

    return classes_data;
}

void Dataset::push_data(Data_point d){
    data.push_back(d);
    instances++;
}

Data_point Dataset::get_data(int i){
    return data[i];
}

void Dataset::set_data(int i, Data_point d){
    data[i] = d;
}

vector<Data_point> Dataset::get_data(){
    return data;
}

void Dataset::erase_data(int i){
    data.erase(data.begin() + i);
```

```
    instances--;
}

void Dataset::insert_data(vector<Data_point> d){
    data.insert(data.end(), d.begin(), d.end());
}

void Dataset::print_info(){

    cout << endl;
    vector<Dataset> data_of_class;
    cout << "Dataset name: " << name << endl;
    cout << "Number of instances: " << instances << endl;
    cout << "Number of attributes: " << attributes << endl;
    cout << "Classes: ";

    for (int i = 0; i < classes.size(); i++) {

        if ( i != 0 ){
            cout << ", ";
        }

        cout << classes[i];
    }

    cout << endl;

    data_of_class = classes_to_datasets();

    // for each class
    for (int j = 0; j < num_of_classes(); j++){
        cout << "Class " << get_class(j) << " instances: " << data_of_class[j].get_instances()
<< endl;
    }
}

void Dataset::print_data(){
```

```
    cout << "Data: " << endl;
    for (int i = 0; i < instances; i++){
        for (int j=0; j<attributes; j++){
        cout << data[i].get_attribute(j) << " ";
        }
        cout << data[i].get_class() << endl;
    }
    cout << endl;
}

void Dataset::clear(){
    name = "";
    instances = 0;
    data.clear();
}

void Dataset::arff_read(string filename){

    ifstream fin(filename.c_str());
    string line;
    int line_counter, **counter_class, classes_num, flag_arff = 0, attr_num = 0;
    bool flag = true;
    float **mean_value;
    double tmp_attr;
    Data_point tmp_data_point;

    cout << endl << "Reading file ..." << endl;

    // find arff relation
    while ( flag && getline(fin,line) ) {

        if ( line.find("@RELATION") != string::npos || line.find("@relation") != string::npos)
{
            if ( line.find(" ") != string::npos ) {
                set_name(line.substr(line.find(" ")+1));
                flag_arff++;
```

```
                flag = false;
            }
        }
    }

    flag = true;

    // find number of attributes
    while ( flag && getline(fin,line) ) {

        if ( line.find("@ATTRIBUTE") != string::npos || line.find("@attribute") != string::npos) {
            // save classes
            if (line.find("class") != string::npos || line.find("Class") != string::npos){

                line = remove_spaces(line);
                line = line.substr(line.find("{")+1, line.find("}") -(line.find("{") + 1));
                string tmp_str;

                while ( line.find(",") != string::npos ) {

                    // each time in tmp_str is saved the next class, and then it is pushed back to the dataset
                    tmp_str = line.substr(0, line.find(","));

                    // remove previous class from the line and also the ','
                    line = line.substr(line.find(",")+1);
                    push_class(tmp_str);
                }

                // now line has only the last class
                push_class(line);
                flag_arff++;

            } else {
                attr_num++;
            }
```

```
    } else if ( line.find("@DATA") != string::npos|| line.find("@data") != string::npos ) {
        flag_arff++;
        flag = false;
    }
}

if ( flag_arff  < 3){
    cout << "File not arff formatted ot does not exists" << endl;
    exit(-1);
}

classes_num = num_of_classes();
mean_value = (float**) malloc(classes_num*sizeof(float*));
counter_class = (int**) malloc(classes_num*sizeof(int*));

for(int i = 0; i<classes_num; i++){
    mean_value[i] = (float*) malloc(attr_num*sizeof(float));
    counter_class[i] = (int*) malloc(attr_num*sizeof(int));
}

set_attributes(attr_num);

// save data instances
while ( getline(fin,line) ) {

    line = remove_spaces(line); // remove spaces
    Data_point d;   // current data point

    if ( line.find(",") != string::npos ){

        while ( line.find(",") != string::npos ) {

            tmp_attr = atof(line.substr(0, line.find(",")).c_str() ); // convert to double

            // remove previous attribute from the line and also the ','
            line = line.substr(line.find(",")+1);
```

```
                d.push_attribute(tmp_attr);
            }

            // now line has only the class
            d.set_class(line);
            push_data(d);
        }
    }
}

void Dataset::preprocess(){

    int i,j;
    double tmp;
    double maximum[attributes], minimum[attributes];

    cout << endl << "Preprocess data..." << endl;   // define the progress state

    // initialize maximum and minimum values to first values
    for (j=0; j<attributes; j++){
        maximum[j] = data[0].get_attribute(j);
        minimum[j] = data[0].get_attribute(j);
    }

    // find max and min value for each attribute
    for (i=0; i<instances; i++){
        for (j=0; j<attributes; j++){
            maximum[j] = max(maximum[j],data[i].get_attribute(j));
            minimum[j] = min(minimum[j],data[i].get_attribute(j));
        }
    }

    // min - max normalization with max = 100 and min = 0
    for (i=0; i<instances; i++){
        for (j=0; j<attributes; j++){
            if ( minimum[j] == maximum[j] ) {
```

```
                data[i].set_attribute(j, 50);
            } else {
                tmp = ( ( data[i].get_attribute(j) - minimum[j] ) / ( maximum[j] -
minimum[j] ) ) * 100;
                data[i].set_attribute(j, tmp);
            }
        }
    }
}

// Other functions ---------------------------------------------------------------------------

string remove_spaces(string input){
    input.erase(remove(input.begin(), input.end(), ' '),input.end());  // remove spaces
    input.erase(remove(input.begin(), input.end(), '\r'),input.end());// remove spaces
    return input;                                    // return new string
}

vector<Dataset> split_dataset(Dataset dataset){

    int i, j;
    Dataset d;
    vector<Dataset> datasets(10);
    vector<Dataset> class_data;

    cout << endl << "Spliting data..." << endl;

    // classes and number of attributes will be the same in each sub-dataset
    d.set_attributes(dataset.get_attributes());
    d.set_classes(dataset.get_classes());

    for (j = 0; j < 10; j++){
        datasets[j] = d;
    }

    class_data = dataset.classes_to_datasets();
```

```
    for (j = 0; j < class_data.size(); j++){
        for (i = 0; i < class_data[j].get_instances(); i++){
            datasets[i%10].push_data(class_data[j].get_data(i));
        }
    }
    return datasets;
}

vector<Dataset> split_dataset_by_id(Dataset dataset){       // by id for leave-one-out cross validation

    int i;
    double id = dataset.get_data(0).get_attribute(0);      // initialize id
    Dataset d;
    vector<Dataset> datasets;   // vector of new datasets

    cout << endl << "Spliting data by id..." << endl;

    // classes and number of attributes will be the same in each sub dataset
    d.set_attributes(dataset.get_attributes()-1);  // erase id
    d.set_classes(dataset.get_classes());

    for (i = 0; i < dataset.get_instances(); i++){

        Data_point data = dataset.get_data(i) ;

        if ( id == data.get_attribute(0) ) {
            data.erase_attribute(0);
            d.push_data(data);              // push back current data point
        } else {
            id = data.get_attribute(0);

            datasets.push_back(d);              // push back prev sub dataset
            d.clear();                      // and create a new one

            data.erase_attribute(0);
            d.push_data(data);              // push back current data point
```

```
        }
    }

    datasets.push_back(d);

    cout << "NUmber of pics: " << datasets.size() << endl;

    for (i = 0; i< datasets.size(); i++){
        cout << "Pic " << i+1 << " instances:" << datasets[i].get_instances() << endl;
    }

    return datasets;
}

vector<Dataset> make_datasets(vector<Dataset> datasets, int folds, int i){ //  make  training
and test dataset. i is the fold index

    int d;          // sub dataset index

    Dataset training_dataset, test_dataset;
    vector<Dataset> new_datasets;

    cout << "Make training and test dataset...." << endl << endl;

    d = (i+folds)%(folds+1);

    // make training dataset with all pics except i
    training_dataset = datasets[d];

    for (int j = 1; j <= (folds - 1); j++){
        d = (i+j)%(folds+1);

        training_dataset.insert_data(datasets[d].get_data());
        training_dataset.set_instances(  training_dataset.get_instances()  +
datasets[d].get_instances() );
    }
```

```
    // make test dataset with pic i
    test_dataset = datasets[i];

    // return training and test dataset
    new_datasets.push_back(training_dataset);
    new_datasets.push_back(test_dataset);

    return new_datasets;
}
```

9.1.3 ckmeans.h

```
#include <iostream>
#include <fstream>
#include <vector>

#include "dataset.h"

// Classes

class Constraints {

    private:

    int attributes;
    double *maximum;
    double *minimum;
    double *weight;

    public:

    Constraints();
    Constraints(int attr);          // constructor. Allocates memory for maximum and minimum
arrays
    double* get_all_maximum();   // get all maximum limits
    double* get_all_minimum();   // get all minimum limits
```

```
    double* get_all_weight();   // get all contribution weights
    double get_maximum(int i);      // get maximum limit of attribute i
    double get_minimum(int i);      // get minimum limit of attribute i
    double get_weight(int i);     // get contribution weight for attribute i
    void set_all_maximum(double *m);      // set all maximum limits
    void set_all_minimum(double *m);      // set all minimum limits
    void set_all_weight(double *w);           // set all contribution weigh
    void set_maximum(int i, double m);      // set i maximum limit to value m
    void set_minimum(int i, double m);      // set i minimum limit to value m
    void set_weight(int i, double w);     // set contribution weight for attribute i

};


// Constrained k-means Classification algorithm functions

Constraints define_means_limits(Dataset dataset);
Dataset ckmeans(Dataset dataset, Constraints constraints[]);
Dataset assign_all(Dataset dataset, Data_point *means, Constraints *constraints);
Data_point* renew_means(Dataset dataset, Data_point *means, Constraints *constraints);

extern double l;
```

9.1.4 ckmeans.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>        // std::pow
#include <algorithm>

#include "ckmeans.h"

#define INF 999

#define ERROR 0.000000005 // Constrained K-Means convergence Error
```

```
double l = 0.1;

// Class Constraints member functions

    Constraints::Constraints(){}

    Constraints::Constraints(int attr){

        attributes = attr;

        maximum = new double[attr];
        minimum = new double[attr];
        weight = new double[attr];
    }

    double* Constraints::get_all_maximum(){
        return maximum;
    }

    double* Constraints::get_all_minimum(){
        return minimum;
    }

    double* Constraints::get_all_weight(){
        return weight;
    }

    double Constraints::get_maximum(int i){
        return maximum[i];
    }

    double Constraints::get_minimum(int i){
        return minimum[i];
    }

    double Constraints::get_weight(int i){
```

```
        return weight[i];
    }

    void Constraints::set_all_maximum(double *m){
        for (int i = 0; i < attributes; i++){
            maximum[i] = m[i];
        }
    }

    void Constraints::set_all_minimum(double *m){
        for (int i = 0; i < attributes; i++){
            minimum[i] = m[i];
        }
    }

    void Constraints::set_all_weight(double *w){
        for (int i = 0; i < attributes; i++){
            weight[i] = w[i];
        }
    }

    void Constraints::set_maximum(int i, double m){
        maximum[i] = m;
    }

    void Constraints::set_minimum(int i, double m){
        minimum[i] = m;
    }

    void Constraints::set_weight(int i, double w){
        weight[i] = w;
    }


// Constrained k-means Classification algorithm functions

Constraints define_means_limits(Dataset dataset){
```

```
int attributes = dataset.get_attributes();
int i, a;
double sum[attributes], mean[attributes], standard_deviation[attributes];
double tmp, sum2, maximum[attributes], minimum[attributes], weight[attributes];
Constraints c(attributes);

// calculate average value of each attribute

fill_n(sum, attributes, 0);

for (i = 0; i < dataset.get_instances(); i++){
    for (a = 0; a < attributes; a++){
        sum[a] += dataset.get_data(i).get_attribute(a);
    }
}

for (a = 0; a < attributes; a++){
    mean[a] = sum[a]/dataset.get_instances();
}

// calculate standard deviation

fill_n(sum, attributes, 0);

for (i = 0; i < dataset.get_instances(); i++){
    for (a = 0; a < attributes; a++){
        tmp = dataset.get_data(i).get_attribute(a) - mean[a];
        sum[a] += pow( tmp, 2);
    }
}

for (a = 0; a < attributes; a++){
    standard_deviation[a] = sqrt(sum[a]/dataset.get_instances());
}

// define hypercube of constraints and weight for attributes
```

```
    sum2 = 0;

    for (a = 0; a < attributes; a++){

        maximum[a] = mean[a] + l * standard_deviation[a];
        minimum[a] = mean[a] - l * standard_deviation[a];

        weight[a] = 1 - (standard_deviation[a]/100);

        sum2 += weight[a];
    }

    // return constraints and weights

    c.set_all_maximum(maximum);
    c.set_all_minimum(minimum);

    c.set_all_weight(weight);

    return c;

}

Dataset ckmeans(Dataset dataset, Constraints *constraints){

    int num_means = dataset.num_of_classes();
    int attributes = dataset.get_attributes();
    Data_point means[num_means], means2[num_means];

    cout << "Ckmeans ...." << endl << endl;

    // initialize means randomly within limits

    for (int i = 0; i < num_means; i++){

        means[i].set_class(dataset.get_class(i));
```

```
        for (int a = 0; a < attributes; a++){

            double tmp_rand = ((double) rand() / RAND_MAX);
            double r = ( constraints[i].get_maximum(a) - constraints[i].get_minimum(a) ) *
tmp_rand + constraints[i].get_minimum(a);

            means[i].push_attribute(r);

        }
        means2[i] = means[i];
    }

    double E = INF;     // to define if means change or not to end the process
    int counter = 0;

    while ( E > ERROR && counter < 100 ){

        Data_point *d;
        E = 0;

        dataset = assign_all(dataset, means, constraints);

        d = renew_means(dataset, means2, constraints);

        for (int i = 0; i < num_means; i++){
            for (int a = 0; a < attributes; a++){
                E += abs(d[i].get_attribute(a) - means[i].get_attribute(a) );
            }
            means[i] = d[i];            // to convert Data_point* to Data_point
        }
        counter++;
    }

    return dataset;
}
```

```
Dataset assign_all(Dataset dataset, Data_point *means, Constraints *constraints){

    int num_data = dataset.get_instances();
    int num_means = dataset.num_of_classes();
    int attributes = dataset.get_attributes();
    double dist[num_means], min_dist, pos_min, tmp;
    int i, j, a;
    Data_point d;

    // calculate the weighted distance from each data point to each mean
    for (i = 0 ; i < num_data; i++){ // for each data point

        min_dist = INF;
        d = dataset.get_data(i);

        for (j = 0; j < num_means; j++){     // for each class

            dist[j] = 0;

            for (a = 0; a < attributes; a++){

                tmp = (d.get_attribute(a) – means[j].get_attribute(a)) * constraints[j].get_weight(a);

                dist[j] += pow( tmp, 2);

            }

            dist[j] = sqrt(dist[j]);     // distance to mean j

            if ( dist[j] < min_dist ){
                min_dist = dist[j];
                pos_min = j;          // the nearest mean so far
            }
        }

        // assign this data point to nearest mean ( class pos_min )
```

```
            d.assign(dataset.get_class(pos_min));

            // save the change to dataset
            dataset.set_data(i,d);
        }
        return dataset;
}

Data_point* renew_means(Dataset dataset, Data_point *means, Constraints *constraints){

        int num_data = dataset.get_instances();
        int num_means = dataset.num_of_classes();
        int attributes = dataset.get_attributes();
        int i, j, a;
        double sum[num_means][attributes], mean[num_means][attributes];

        // calculate mean
        for (i = 0; i < num_means; i++){

            fill_n(sum[i], attributes, 0);

            for (j = 0; j < num_data; j++){
                for (a = 0; a < attributes; a++){
                    if ( dataset.get_class(i).compare( dataset.get_data(j).get_assigned_class() ) ==
0 ){
                        sum[i][a] += dataset.get_data(j).get_attribute(a);
                    }
                }
            }

            for (a = 0; a < attributes; a++){

                mean[i][a] = sum[i][a]/dataset.get_instances();

                if ( constraints[i].get_maximum(a) < mean[i][a] ){
                    mean[i][a] = constraints[i].get_maximum(a);
                }
```

```
            if ( constraints[i].get_minimum(a) > mean[i][a] ){
                mean[i][a] = constraints[i].get_minimum(a);
            }

            means[i].set_attribute(a,mean[i][a]);
        }
    }

    return means;
}
```

9.1.5 main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>

#include "ckmeans.h"

using namespace std;

int main(int argc, char** argv){

    bool id = false;

    if ( argc < 2 ){
        cout << "Error: Expects at least one argument. Example of use ./<executable> filename" << endl;
        return -1;
    } else if ( argc > 3 ){
        cout << "Error: Expects at most two arguments. Example of use ./<executable> filename id" << endl;
        return -1;
    } else if ( argc == 3 ){
```

```
        cout << argv[2] << endl;
        string tmp(argv[2]);

        if ( tmp.compare("id") == 0){
            id = true;          // defines wether leave-one-out by id cross validation will take place
        } else {
            cout << "Error: second argument must by id or nothing. Example of use ./<executable> filename id" << endl;
            return -1;
        }

    } else if ( argc == 2 ){
        id = false;
    }

    // variables and objects --------------------------------------------------------------

    Dataset dataset;            // the dataset
    vector<Dataset> datasets;       // sub datasets by picture
    vector<Dataset> tmp_datasets;       // temp vector of datasets
    Dataset training_dataset, test_dataset;         // training and test dataset for testing
    vector<Dataset> data_of_class;      // vector of datasets. Each dataset haw all data points of a class.

    string filename = argv[1];          // get filename from command line

    // code -------------------------------------------------------------------------------

    dataset.arff_read(filename);            // read arff file and save dataset
    dataset.print_info();                   // print info of dataset

    if ( id ){
        datasets = split_dataset_by_id(dataset);// split to sub datasets by picture
    } else {
        dataset.preprocess();               // min - max normalization
        datasets = split_dataset(dataset);      // split to sub datasets for 10 fold
```

```
}

// initialiaze confusion matrix,
int conf_matrix[dataset.num_of_classes()][dataset.num_of_classes()];

for (int i = 0; i < dataset.num_of_classes(); i++){
    for (int j = 0; j < dataset.num_of_classes(); j++){
        conf_matrix[i][j] = 0;
    }
}

cout << endl << "Starting 10 fold..." << endl;   // 'pic' fold cross validation

int folds = datasets.size()-1;

for (int i = 0; i <= folds; i++){

    cout << endl << "Fold " << i+1 << ":" << endl << endl;

    // Make training and test datasets ------------------------------------------------

    tmp_datasets = make_datasets(datasets, folds, i);
    training_dataset = tmp_datasets[0];
    test_dataset = tmp_datasets[1];

    // Split training dataset to classes
    data_of_class = training_dataset.classes_to_datasets();

    // create an empty set of constraints for each mean
    Constraints constraints[dataset.num_of_classes()](dataset.get_attributes());

    // Define means hypercube of constraints for each class

    cout << "Define means hypercube of constraints for each class" << endl << endl;

    // for each class
    for (int j = 0; j < dataset.num_of_classes(); j++){
```

```
            // define means' hypercube of constraints and contribution weights for attributes
            constraints[j] = define_means_limits(data_of_class[j]);

        }
        test_dataset = ckmeans(test_dataset, constraints);   // assign data using constrained k-means


        // Check results and fill confusion matrix ----------------------------------------

        cout << "Check results ..." << endl;

        for (int j = 0; j < test_dataset.get_instances(); j++){

            string real_class = test_dataset.get_data(j).get_class();
            string assigned_class = test_dataset.get_data(j).get_assigned_class();
            int x,y;

            for ( int c = 0; c < test_dataset.num_of_classes(); c++){

                if ( real_class.compare(test_dataset.get_class(c)) == 0 ){
                    x = c;   // x line of confusion matrix
                }

                if ( assigned_class.compare(test_dataset.get_class(c)) == 0 ){
                    y = c;   // y column of confusion matrix
                }
            }
            conf_matrix[x][y]++;

        }
    }

    // Calculate performance metrics and print results ------------------------------

    double correct = 0;
```

```
    double class_correct[dataset.num_of_classes()];
    fill_n(class_correct, dataset.num_of_classes(), 0);

    double sum_precision[dataset.num_of_classes()], sum_recall[dataset.num_of_classes()];

    fill_n(sum_precision, dataset.num_of_classes(), 0);
    fill_n(sum_recall, dataset.num_of_classes(), 0);

    double avg_recall = 0, avg_precision = 0, accuracy = 0;;

    cout << "Confusion matrix: " << endl << endl;

    for (int i = 0; i < dataset.num_of_classes(); i++){
        for (int j = 0; j < dataset.num_of_classes(); j++){

            sum_precision[j] += conf_matrix[i][j];
            sum_recall[i] += conf_matrix[i][j];

            if ( i == j ){
                correct += conf_matrix[i][j];
                class_correct[i]+= conf_matrix[i][j];
            }
            cout << conf_matrix[i][j] << "\t ";
        }
        cout << endl;
    }


    for (int i = 0; i < dataset.num_of_classes(); i++){

        cout << endl << "Class " << dataset.get_class(i) << ":" << endl << endl;
        cout << "Recall: " << (class_correct[i]/sum_recall[i])*100 << " %" << endl;
        avg_recall += (class_correct[i]/sum_recall[i])*100;

        if ( sum_precision[i] != 0){
            cout << "Precision: " << (class_correct[i]/sum_precision[i])*100 << " %" <<
endl;
```

```
            avg_precision += (class_correct[i]/sum_precision[i])*100;
        }
    }

    cout << endl;

    avg_recall = avg_recall / dataset.num_of_classes();
    avg_precision = avg_precision / dataset.num_of_classes();
    accuracy = (correct/dataset.get_instances())*100;

    cout << endl << "Accuracy: " << accuracy << " %" << endl;
    cout << "Average recall: " << avg_recall << " %" << endl;
    cout << "Average precision: " << avg_precision << " %" << endl;
    cout << endl;

    return 0;

}
```

9.2 Παράρτημα 2

Αρχεία:

dataset.h
dataset.cpp
stoch_forest.h
stoch_forest.cpp
main.cpp

9.2.1 dataset.h:

```
#include <vector>
#include <string>

#define MIN_SPLIT_INSTANCES 2
#define MAX_SPLIT_INSTANCES 25

extern int MIN_INSTANCES;

using namespace std;
```

```
// Classes.....................................................................................

class Attribute {

    private:

    double numeric;                 // Attribute's value if it is numeric
    string nominal;             // Attribute's value if it is nominal

    public:

    Attribute(); // Constructor

    // set - push ---------------------------

    void set_attribute(double n);     // if attribute is numeric, set it's value to n
    void set_attribute(string n);// if attribute is nominal, set it's value to n


    // get ----------------------------------

    double get_attribute_numeric();      // Get numeric attribute value
    string get_attribute_nominal(); // Get nominal attribute value

    // clear --------------------------------

    void clear();      // clear attribute

};

class Attribute_type {

    private:

    bool attr_type;        //  true: nominal
                    //  false:      numeric

    vector<string> nom_values; // If attribute is nominal

    public:


    Attribute_type();     // Constructor

    // set - push ---------------------------
```

```
    void set_attr_type(bool type);   // Set attribute type
    void set_nom_values(vector<string> v);     // Set all nominal values
    void push_nominal_value(string v);      // Push a nominal value

    // get ---------------------------------

    bool get_attr_type();          // Get attribute type
    vector<string> get_nominal_values();   // get all nominal values
    string get_nominal_value(int i);      // get i nominal value

    // clear -------------------------------

    void clear();          // clear all variable values

};

class Data_point {

    private:

    vector<Attribute> attribute;      // data point attributes
    string Class;                // data point class
    string assigned_class;         // data point assigned class

    public:

    Data_point();                // Constructor

    string get_class();         // returns class of data point
    void set_class(string c);      // set the class of the data point
    string get_assigned_class();      // returns assigned class of data point
    void assign(string c);          // assign data point
    void push_attribute(Attribute a);     // push attribute to vector
    void erase_attribute(int i);   // erase attribute i from data point
    Attribute get_attribute(int i);     // get attribute i from data point
    void set_attribute(int i, Attribute a); // set value of attribute i to value of a
    void set_attribute(int i, double a); // set value of attribute i to value of a
    void set_attribute(int i, string a); // set value of attribute i to value of a
    vector<Attribute> get_attributes(); // get all attributes from data point

};

class Dataset {

    private:
```

```
    string name;                // dataset name
    int instances;              // dataset number of instances
    int attributes;             // dataset number of attributes
    vector<Attribute_type> attribute_types;          // Attribute types
    vector<string> classes;     // dataset classes
    vector<Data_point> data;    // dataset data points
    double entropy;

    public:

    Dataset();              // Constructor

    // datasets info ------------------------------------------------

    string get_name();      // return the name of dataset
    void set_name(string n);    // set the name of dataset
    int get_instances();    // get number of instances
    void set_instances(int i);    // set number of instances [ DO NOT USE THIS ]
    int get_attributes();   // get number of attributes
    void set_attributes(int a);   // set number of attributes
    void push_attribute_type(Attribute_type type);    // push an attribute type
    void erase_attribute_type(int i);
    void set_attribute_types(vector<Attribute_type> types);  // set attribute types
    Attribute_type get_attribute_type(int i);         // get attribute type of attribute i
    vector<Attribute_type> get_attribute_types();     // get attribute types

    // class --------------------------------------------------------

    void push_class(string c);  // push a new found class
    string get_class(int i);  // get class i
    vector<string> get_classes();    // get all classes
    void set_classes(vector<string> c); // set all classes
    int num_of_classes();       // get number of classes
    vector<Dataset> classes_to_datasets(); // get classes in separate datasets

    // data ---------------------------------------------------------

    void push_data(Data_point d);  // push new data point
    void erase_data(int i);  // erase data point i from dataset
    void insert_data(vector<Data_point> d); // insert a vector of data points in the end of
vector data

    Data_point get_data(int i);  // get data point i
    void set_data(int i, Data_point d);  // set data point i
    vector<Data_point> get_data();     // get all data
```

```
    void sort_by_attribute(int a);    // sort data by value of attribute a

    // other ------------------------------------------------------------

    void print_info();          // print information of dataset
    void print_data();          // print data
    void clear();               // clear dataset
    void arff_read(string filename);      // read arrf file, and save dataset

    // entropy ----------------------------------------------------------

    void set_entropy(double e);      // set value of dataset entropy
    double get_entropy();         // get value of dataset_entropy
    void find_entropy();          // find dataset entropy

    double find_entropy(int a, int *&counter); // find entropy of attribute i. For nominal attributes

    double find_entropy(int attribute, double split_point, int &counter);    // find entropy of attribute i
                                                //with split point split_point
                                                // For numeric attributes
                                                // counter returns number of instances
                                                // before split point

    double find_inf_gain(int attribute, int *&counter);  // find inmformation gain for attribute i

    double find_inf_gain(int attribute, double split_point, int &counter);    //                find inmformation gain for attribute i

    double find_split_point(int attribute, double &gain_ratio, double &inf_gain);  // find best split point for attribute i


    // gain ratio -------------------------------------------------------

    double find_split_info(int *counter,  int attribute);
    double find_split_info(int counter);
    double find_gain_ratio(int attribute, double &inf_gain);
    double find_gain_ratio(double inf_gain, double split_info);

};
//  Other functions.....................................................................................

string remove_spaces(string input);// remove all spaces in string input
vector<Dataset> split_dataset(Dataset dataset);// split dataset
```

```
vector<Dataset> make_datasets(vector<Dataset> datasets, int folds, int i);  //    make    training
and test dataset. i fold index

void quickSort( Data_point a[], int first, int last, int at ) ;
int pivot(Data_point a[], int first, int last, int at) ;
void swap(Data_point& a, Data_point& b);
```

9.2.2 dataset.cpp:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <string>

#include "dataset.h"

#define INT_SIZE 4

using namespace std;

int MIN_INSTANCES;

// Class Attribute member functions ------------------------------------------------------

    // Constructor

Attribute::Attribute(){}

    // set - push ---------------------------

void Attribute::set_attribute(double n){
    numeric = n;
}

void Attribute::set_attribute(string n){
    nominal = n;
}

    // get ----------------------------------

double Attribute::get_attribute_numeric(){
    return numeric;
```

```
}

string Attribute::get_attribute_nominal(){
    return nominal;
}


    // clear --------------------------------

void Attribute::clear(){
    numeric=0;
    nominal="";
}

// Class Attribute_type member functions ------------------------------------------------

    // Constructor

Attribute_type::Attribute_type(){
    attr_type = 0;
}

    // set - push ---------------------------

void Attribute_type::set_attr_type(bool type){  // Set attribute type
    attr_type = type;
}

void Attribute_type::set_nom_values(vector<string> v){    // Set all nominal values
    nom_values = v;
}

void Attribute_type::push_nominal_value(string v){     // Push a nominal value
    nom_values.push_back(v);
}

    // get ----------------------------------

bool Attribute_type::get_attr_type(){          // Get attribute type
    return attr_type;
}

vector<string> Attribute_type::get_nominal_values(){  // get all nominal values
    return nom_values;
}
```

```
string Attribute_type::get_nominal_value(int i){     // get i nominal value
     return nom_values[i];
}

     // clear --------------------------------

void Attribute_type::clear(){
     attr_type = false;
     nom_values.clear();
}

//                Class                Data_point                member                functions
--------------------------------------------------------------------------------

Data_point::Data_point(){}

string Data_point::get_class(){
     return Class;
}

void Data_point::set_class(string c){
     Class = c;
}

string Data_point::get_assigned_class(){
     return assigned_class;
}

void Data_point::assign(string c){
     assigned_class = c;
}

void Data_point::push_attribute(Attribute a){
     attribute.push_back(a);
}

void Data_point::erase_attribute(int i){
     attribute.erase(attribute.begin() + i);
}

void Data_point::set_attribute(int i, Attribute a){
     attribute[i] = a;
}

void Data_point::set_attribute(int i, double a){
     attribute[i].set_attribute(a);
```

```
}

void Data_point::set_attribute(int i, string a){
    attribute[i].set_attribute(a);
}


Attribute Data_point::get_attribute(int i){
    return attribute[i];
}

vector<Attribute> Data_point::get_attributes(){
    return attribute;
}


// Class Dataset member functions
-------------------------------------------------------------------------

Dataset::Dataset(){
    instances = 0;
    attributes = 0;
}

    // dataset info ---------------------------------------------

string Dataset::get_name(){
    return name;
}

void Dataset::set_name(string n){
    name = n;
}

int Dataset::get_instances(){
    return instances;
}

void Dataset::set_instances(int i){
    instances = i;
}

int Dataset::get_attributes(){
    return attributes;
}
```

```
void Dataset::set_attributes(int a){
    attributes = a;
}

void Dataset::push_attribute_type(Attribute_type type){
    attribute_types.push_back(type);
}

void Dataset::erase_attribute_type(int i){
    attribute_types.erase(attribute_types.begin() + i);
}

void Dataset::set_attribute_types(vector<Attribute_type> types){
    attribute_types=types;
}

Attribute_type Dataset::get_attribute_type(int i){
    return attribute_types[i];
}

vector<Attribute_type> Dataset::get_attribute_types(){
    return attribute_types;
}

    // class ----------------------------------------------------------

void Dataset::push_class(string c){
    classes.push_back(c);
}

string Dataset::get_class(int i){
    return classes[i];
}

vector<string> Dataset::get_classes(){
    return classes;
}

void Dataset::set_classes(vector<string> c){
    classes = c;
}

int Dataset::num_of_classes(){
    return classes.size();
}
```

```
vector<Dataset> Dataset::classes_to_datasets(){

    vector<Dataset> classes_data( num_of_classes() ); // initialize vector of classes datasets

    for (int j = 0; j < num_of_classes(); j++){
        classes_data[j].set_attribute_types(get_attribute_types());
        classes_data[j].set_attributes(get_attributes());
    }

    for (int i = 0; i < instances; i++){    // for all data points
        for (int j = 0; j < num_of_classes(); j++){    // for all classes
            if ( get_class(j).compare( data[i].get_class() ) == 0 ){    // if in this class
                classes_data[j].push_data( data[i] );                // push data to class's vector
            }
        }
    }
    return classes_data;
}

    // data ------------------------------------------------------------

void Dataset::push_data(Data_point d){
    data.push_back(d);
    instances++;
}

Data_point Dataset::get_data(int i){
    return data[i];
}

void Dataset::set_data(int i, Data_point d){
    data[i] = d;
}

vector<Data_point> Dataset::get_data(){
    return data;
}

void Dataset::erase_data(int i){
    data.erase(data.begin() + i);
    instances--;
}

void Dataset::insert_data(vector<Data_point> d){
    data.insert(data.end(), d.begin(), d.end());
}
```

```
    // other ---------------------------------------------------------

void Dataset::print_info(){

    vector<string> nominal_values;
    cout << endl;
    vector<Dataset> data_of_class;

    cout << "Dataset name: " << name << endl;
    cout << "Number of instances: " << instances << endl;
    cout << "Number of attributes: " << attributes << endl;
    cout << endl << "Attribute types:"<< endl << endl;

    for( int i = 0; i < attributes; i++ ){

        cout << "Attribute " << i << ":";
        if ( attribute_types[i].get_attr_type() ){

            cout << " Nominal" << endl;

            nominal_values=attribute_types[i].get_nominal_values();

            for ( int j = 0; j < nominal_values.size(); j++ ) {

                cout << "\t\t" << j+1 << ". " << nominal_values[j] << endl;
            }

        } else {
            cout << " Numeric" << endl;
        }
    }

    cout<< endl;
    cout << "Classes: " ;

    for (int i = 0; i < classes.size(); i++) {

        if ( i != 0 ){
            cout << ", ";
        }
        cout << classes[i];
    }

    cout << endl;
```

```
    data_of_class = classes_to_datasets();

    // for each class
    for (int j = 0; j < num_of_classes(); j++){
        cout << "Class " << get_class(j) << " instances: " << data_of_class[j].get_instances()
<< endl;
    }
    cout << "Dataset entropy: " << entropy << endl;
}

void Dataset::print_data(){

    cout << endl << "Data: " << endl;

    for (int i = 0; i < instances; i++){
        for (int j=0; j<attributes; j++){
            if ( get_attribute_type(j).get_attr_type() ){
                cout << data[i].get_attribute(j).get_attribute_nominal() << "\t\t";
            } else {
                cout << data[i].get_attribute(j).get_attribute_numeric() << "\t\t";
            }
        }
        cout << data[i].get_class() << endl;
    }
    cout << endl;
}

void Dataset::clear(){

    data.clear();

}

// Read arff file with the dataset
void Dataset::arff_read(string filename){

    ifstream fin(filename.c_str());
    string line;
    int line_counter;
    int flag_arff = 0;     // if file is correct formatted arff file
    bool flag = true;      // to stop while without break
    float **mean_value;
    int **counter_class;
    int classes_num;
    Data_point tmp_data_point;
    cout << endl << "Reading file ..." << endl; // define the progress state
```

```
    // find arff relation
    while ( flag && getline(fin,line) ) {

        if ( line.find("@RELATION") != string::npos || line.find("@relation") != string::npos)
{

            if ( line.find(" ") != string::npos ) {

                set_name(line.substr(line.find(" ")+1));

                flag_arff++;
                flag = false;

            }
        }
    }

    flag = true;
    int attr_num = 0;

    Attribute_type type;

    // find number of attributes
    while ( flag && getline(fin,line) ) {

        if ( line.find("@ATTRIBUTE") != string::npos || line.find("@attribute") !=
string::npos) {

            // save classes
            if (line.find("class") != string::npos || line.find("Class") != string::npos){

                line = remove_spaces(line);
                line = line.substr(line.find("{")+1, line.find("}") -(line.find("{") + 1));
                string tmp_str;

                while ( line.find(",") != string::npos ) {

                    // each time in tmp_str is saved the next class, and then it is pushed back
to the dataset
                    tmp_str = line.substr(0, line.find(","));

                    // remove previous class from the line and also the ','
                    line = line.substr(line.find(",")+1);

                    push_class(tmp_str);
```

```
            }

            // now line has only the last class
            push_class(line);

            flag_arff++;

        } else {

            type.clear();

            attr_num++;


            if ( line.find("{") != string::npos ) {

                line = remove_spaces(line);
                line = line.substr(line.find("{")+1, line.find("}") -(line.find("{") + 1));


                string tmp_str;

                type.set_attr_type(true);

                while ( line.find(",") != string::npos ) {

                    // each time in tmp_str is saved the next attribute nominal value, and
then it is pushed back to the dataset
                    tmp_str = line.substr(0, line.find(","));

                    // remove previous attribute nominal value from the line and also the
','
                    line = line.substr(line.find(",") + 1);

                    type.push_nominal_value(tmp_str);

                }

                // now line has only the last attribute nominal value
                type.push_nominal_value(line);


            } else {
                type.set_attr_type(false);
            }
```

```
                push_attribute_type(type);

            }

        } else if ( line.find("@DATA") != string::npos|| line.find("@data") != string::npos ) {

            flag_arff++;
            flag = false;

        }
    }

    if ( flag_arff  < 3){
        cout << "File not arff formatted or does not exists" << endl;
        exit(-1);
    }

    classes_num = num_of_classes();

    set_attributes(attr_num);
    Attribute attr;
    int counter;

    // save data instances
    while ( getline(fin,line) ) {

        counter=0;
        line = remove_spaces(line); // remove spaces
        Data_point d;   // current data point
        string tmp_class;

        if ( line.find(",") != string::npos ){

            while ( line.find(",") != string::npos ) {

                attr.clear();

                if ( attribute_types[counter].get_attr_type() ) {
                    attr.set_attribute(line.substr(0, line.find(",")));

                } else {
                    attr.set_attribute(atof(line.substr(0,  line.find(",")).c_str()  ));  // convert  to
double
                }

                // remove previous attribute from the line and also the ','
```

```
                line = line.substr(line.find(",")+1);

                d.push_attribute(attr);

                counter++;
            }

            // now line has only the class
            d.set_class(line);
            push_data(d);
        }
    }

}

// Other functions ---------------------------------------------------------------------

string remove_spaces(string input){

    input.erase(remove(input.begin(), input.end(), ' '),input.end());  // remove spaces
    input.erase(remove(input.begin(), input.end(), '\r'),input.end());// remove spaces
    return input;                                       // return new string

}

// split dataset for 10 fold
vector<Dataset> split_dataset(Dataset dataset){

    cout << endl << "Spliting data..." << endl;  // define current state

    vector<Dataset> datasets(10);   // vector of new datasets
    Dataset d;
    vector<Dataset> class_data;

    // classes and number of attributes will be the same in each sub dataset
    d.set_attributes( dataset.get_attributes() );
    d.set_attribute_types( dataset.get_attribute_types() );

    d.set_classes( dataset.get_classes() );

    for (int j = 0; j < 10; j++){
        datasets[j] = d;
    }

    class_data = dataset.classes_to_datasets();
```

```
    for (int j = 0; j < class_data.size(); j++){
        for (int i = 0; i < class_data[j].get_instances(); i++){
            datasets[i%10].push_data(class_data[j].get_data(i));
        }
    }

    return datasets;
}

vector<Dataset> make_datasets(vector<Dataset> datasets, int folds, int i){ //    make    training
and test dataset. i fold index


    Dataset training_dataset, test_dataset;        // training and test dataset for testing

    vector<Dataset> new_datasets; // vector of new datasets

    int d;          // sub dataset index

    cout << "Make training and test dataset...." << endl << endl;

    d = (i+folds)%(folds+1);

    // make training dataset with all pics except i

    training_dataset = datasets[d];

    for (int j = 1; j <= (folds - 1); j++){

        d = (i+j)%(folds+1);

        training_dataset.insert_data(datasets[d].get_data());
        training_dataset.set_instances(                training_dataset.get_instances()                +
datasets[d].get_instances() );

    }

    // make test dataset with pic i

    test_dataset = datasets[i];

    // return training and test dataset

    new_datasets.push_back(training_dataset);

    new_datasets.push_back(test_dataset);
```

```
    return new_datasets;
}


// entropy, information gain, gain ratio related methods ----------------------------------

double Dataset::get_entropy(){
    return entropy;
}

void Dataset::find_entropy(){   // find dataset entropy

    vector<Dataset> data_of_class;

    data_of_class = classes_to_datasets();

    double tmp;
    entropy = 0;

    double *tmp2;

    tmp2 = new double[data_of_class.size()];

    // for each class
    for (int j = 0; j < num_of_classes(); j++){

        tmp = (double) data_of_class[j].get_instances() / get_instances();
        tmp2[j] = tmp;
        if ( tmp != 0 )
            entropy += tmp*log2(tmp);

    }

    if ( entropy != 0 ) {

        entropy = -entropy;

    }
}

double Dataset::find_entropy(int a, int *&counter){     // find entropy if splitted to attribute a
    [nominal]
                                // counter: table with number of instances for each node-children

    vector<Dataset> data_of_class;
```

```
double tmp;
double tmp_entropy, entropy = 0;

vector<string> nominal_values = attribute_types[a].get_nominal_values();
int no_of_values, no_of_classes;
int **counter_class;

int j, k;
bool flag;
string attribute_value;

no_of_values = nominal_values.size();
no_of_classes=num_of_classes();

counter_class = new int*[no_of_values];

counter = new int[no_of_values];

for( j = 0; j < no_of_values; j++ ) {

    counter_class[j] = new int[no_of_classes];

    counter[j] = 0;

    for( k = 0; k < no_of_classes; k++ ) {

        counter_class[j][k] = 0;
    }

}


for(int i = 0; i < get_instances(); i++ ){

    j = 0;
    flag=true;

    attribute_value = data[i].get_attribute(a).get_attribute_nominal();

    while( flag && j < no_of_values ){

        if ( attribute_value == attribute_types[a].get_nominal_value(j) ) {
            for ( int k=0; k < no_of_classes; k++ ){
                if( data[i].get_class() == classes[k] ) {
                    counter_class[j][k]++;
```

```
                    }
                }

                counter[j]++;
                flag=false;

            }

            j++;

        }
    }


    // for each nominal value
    for (j = 0; j < no_of_values; j++){

        tmp_entropy=0;

        for ( k = 0; k < no_of_classes; k++){

            if ( counter[j] > 0 ){
                tmp = (double) counter_class[j][k] / counter[j];

                if ( tmp != 0 )
                    tmp_entropy += tmp*log2(tmp);

            }
        }

        entropy += ( (double) counter[j] / get_instances() ) * tmp_entropy;

    }

    return -entropy;
}


double Dataset::find_entropy(int attribute, double split_point, int &counter){   //  find  entropy if splitted to attribute a
                                        // [numeric]
                                // counter: table with number of instances for each node-children
                                // Splitted to split_point


    int num_of_classes = this->num_of_classes();
```

```
int instances = get_instances();
int *class_counter;
int *class_counter_gr;

counter = 0;

class_counter = new int[num_of_classes];
class_counter_gr = new int[num_of_classes];
Data_point instance;
double attr_entropy = 0;
double attr_entropy_gr = 0;
double tmp;
int i, j, l;


for (l = 0; l < num_of_classes; l++){
    class_counter_gr[l] = 0;
    class_counter[l] = 0;
}


for(j = 0; j < instances; j++){

    instance = get_data(j);

    if ( instance.get_attribute(attribute).get_attribute_numeric() <= split_point ){
        counter++;
        for (l = 0; l < num_of_classes; l++){
            if ( get_class(l)==instance.get_class() ){
                class_counter[l]++;
            }
        }
    } else {
        for (l = 0; l < num_of_classes; l++){
            if ( get_class(l)==instance.get_class() ){
                class_counter_gr[l]++;
            }
        }
    }
}

for (l = 0; l < num_of_classes; l++){     //  for each class

    tmp = (double) class_counter[l] / counter;

    if ( tmp != 0 )
```

```
            attr_entropy += tmp*log2(tmp);

        tmp = (double) class_counter_gr[l] / (instances - counter);

        if ( tmp != 0 )
            attr_entropy_gr += tmp*log2(tmp);

    }


    attr_entropy = ( (double) counter/(double) instances )*attr_entropy + ( (double) ( instances - counter) / (double) instances )*attr_entropy_gr;


    if ( attr_entropy != attr_entropy ){

        attr_entropy = entropy;
    } else {
        attr_entropy = -attr_entropy;
    }


    return attr_entropy;
}

double Dataset::find_inf_gain(int attribute, int *&counter){      // for nominal attribute

    double attribute_entropy = find_entropy( attribute, counter );

    double inf_gain = entropy - attribute_entropy;

    if ( inf_gain != inf_gain ){

    }

    return entropy - attribute_entropy;
}

double Dataset::find_inf_gain(int attribute, double split_point, int &counter){  //  for  numeric attribute

    double attr_entropy = find_entropy(attribute, split_point, counter);

    double inf_gain = entropy - attr_entropy;

    return inf_gain;
```

```
}

double Dataset::find_split_point(int attribute, double &gain_ratio, double &inf_gain){ //    find
best split point for numeric                                                    // attribute

    double max_inf_gain = 0;
    double pos_max;
    int counter;

    int counter_max;

    int i;

    sort_by_attribute(attribute);

    double tmp;
    int tries = 0;

    for(i = MIN_INSTANCES - 1; i < instances - MIN_INSTANCES; i++){

        if ( data[i].get_attribute(attribute).get_attribute_numeric() <
data[i+1].get_attribute(attribute).get_attribute_numeric() - 0.00001 ) {

            tries++;

            tmp = data[i].get_attribute(attribute).get_attribute_numeric() +
data[i+1].get_attribute(attribute).get_attribute_numeric();

            tmp = tmp / 2;

            inf_gain = find_inf_gain(attribute, tmp, counter);

            if ( inf_gain > max_inf_gain ) {
                max_inf_gain = inf_gain;
                pos_max = tmp;
                counter_max = counter;
            }

        }


    }

    double threscost = log2(tries) / instances;

    if ( max_inf_gain > threscost ){
```

```
            max_inf_gain -= threscost;

            gain_ratio = find_gain_ratio( max_inf_gain, find_split_info(counter_max) );
            inf_gain = max_inf_gain;
        }

        return pos_max;
}


double Dataset::find_gain_ratio(int attribute, double &inf_gain){    // for nominal attribute

        int *counter;

        double gain_ratio = 0;

        inf_gain = find_inf_gain(attribute, counter);

        double split_info = find_split_info(counter, attribute);

        if ( split_info > 0 ){

            gain_ratio = inf_gain / split_info;
        }

        return gain_ratio;
}

double Dataset::find_gain_ratio(double inf_gain, double split_info){    // for numeric attribute

        double gain_ratio = 0.0;


        if ( split_info > 0 ){
            gain_ratio = inf_gain / split_info;
        }

        return gain_ratio;
}

double Dataset::find_split_info( int counter ){        // for numeric attribute

        double tmp;
        double splin_info = 0;
```

```
    tmp = (double) counter/instances;

    if ( tmp != 0 )
        splin_info += tmp*log2(tmp);

    tmp = 1 - tmp;

    if ( tmp != 0 )
        splin_info += tmp*log2(tmp);

    return -splin_info;
}

double Dataset::find_split_info(int *counter, int attribute ){          // for nominal attribute

    double splitinfo = 0.0;
    int i;

    double tmp;

    for ( i = 0; i < attribute_types[attribute].get_nominal_values().size(); i++){

        tmp = (double) counter[i] / instances ;

        if ( tmp != 0 )
            splitinfo += tmp*log2(tmp);

    }

    return -splitinfo;

}

void Dataset::sort_by_attribute(int a){
    Data_point *d = &data[0];
    quickSort(d, 0, instances-1, a);
    data.assign( d, d + instances );
}

void quickSort( Data_point a[], int first, int last, int at )
{
   int pivotElement;
   if(first < last)
   {
      pivotElement = pivot(a, first, last, at);
      quickSort(a, first, pivotElement-1, at);
```

```
        quickSort(a, pivotElement+1, last, at);
    }
}

/**
 * a - The array.
 * first - The start of the sequence.
 * last - The end of the sequence.
 * the pivot element
*/
int pivot(Data_point a[], int first, int last, int at)
{
    int  p = first;
    Data_point pivotElement = a[first];

    for(int i = first+1 ; i <= last ; i++)
    {

                                        if(a[i].get_attribute(      at      ).get_attribute_numeric()      <=
pivotElement.get_attribute( at ).get_attribute_numeric())
        {
            p++;
            swap(a[i], a[p]);
        }
    }

    swap(a[p], a[first]);

    return p;
}


/**
 * a - The first parameter.
 * b - The second parameter.
*/
void swap(Data_point& a, Data_point& b)
{
    Data_point temp = a;
    a = b;
    b = temp;
}
```

9.2.3 stoch_forest.h

```
#include <iostream>
#include <fstream>
#include <vector>

// rand, time ...
#include <cstdlib>
#include <ctime>

#include "dataset.h"

class Node {     // decision tree node

    private:
        vector<Node *> children;   // pointers to children nodes
        Node *parent;              // pointer to parent node
        Dataset dataset;      // sub-dataset before current condition
        string assigned_class;         // class
        int attribute;             // split attribute
        float split_point;         // split point for numeric attributes
        int errors;              // number of errors
        int children_errors;       // number of errors in children nodes

    public:

        Node();

        // set - add ----------------------------------------------------------------

        void add_child( Node *ch );     // add child to children vector
        void create_children();        // create node's children by splitting the dataset.
        void set_parent( Node *p );// set parent node
        void set_dataset( Dataset d );    // set sub-dataset after condition of previous node
        void set_class( string c );    // set class
        void set_condition( int attr );              // set a decision-condition using attribute attr
                                    // (for nominal attributes)
```

```
    void set_condition( int attr, double sp );      // set a decision-condition using attribute attr
                                    // with split point sp (for numeric attributes)
    void set_errors(int e);         // set number of errors to e
    void set_children_errors(int e);        // set number of children errors to e

    // get ----------------------------------------------------------------------

    vector<Node *> get_children();      // get node's all children
    Node *get_child( int i );       // get the i-th child of node
    Node *get_parent();             // get parent node
    Dataset get_dataset();          // return sub-dataset
    string get_class();         // get class
    int get_attribute();        // return which attribute is to decide on this node
    double get_split_point();   // return split point used for decision on this node (for numeric attributes)
    int get_errors();       // get number of errors
    int get_children_errors();  // get number of children errors

    // other functions ----------------------------------------------------------
    vector<Dataset> split_dataset();      // split dataset by attribute
    bool find_condition();          // find condition to split the dataset
    void create_subtree();          // create subtree
    void make_leaf();           // make tree leaf
    void remove_child( int ch );      // remove a child from node's children
    void clear();               // unset all properties
    void print( int depth );  // print node
};

// Class Tree ---------------------------------------------------------------------
```

```
class Tree {

    private:
        Node *root;

    public:
        Tree();                     // constructor
        Node *get_root();           // return root node
        void initialize(Dataset dataset); // initialize root node
        void set_root(Node *root_addr);     // set root node
        void create_tree();         // build tree from train set( root's dataset )
        void prune();               // prune tree
        string classify(Data_point data);   // classify test set instances
        void clear();

};
```

9.2.4 stoch_forest.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>      // std::pow
#include <algorithm>

#include "stoch_forest.h"
#define INF 999
#define MIN_GAIN 0.0001

Node::Node(){      // constructor
    attribute = -1;
    split_point = -1;
    parent = NULL;
}
```

```
void Node::add_child( Node *ch ){       // add child to children vector
    children.push_back( ch );
}

void Node::create_children(){

    vector<Dataset> datasets = split_dataset();
    Node *n;
    int tmp_errors = 0;

    for ( int i = 0; i < datasets.size(); i++ ){

        n = new Node;
        n->set_dataset(datasets[i]);
        n->set_parent(this);

        add_child( n );

        n->make_leaf();

        tmp_errors += n->get_errors();
    }

    set_children_errors( tmp_errors );

}

void Node::set_parent( Node *p ){  // set parent node
    parent = p;
}

void Node::set_dataset( Dataset d ){     // set sub-dataset after condition of previous node
    dataset = d;
}

void Node::set_class( string c ){     // set class
```

```
    assigned_class = c;
}

void Node::set_condition( int attr ){                    // set a decision-condition using attribute attr (for nominal attributes)
    attribute = attr;
}

void Node::set_condition( int attr, double sp ){ // set a decision-condition using attribute attr with split point sp (for numeric attributes)
    attribute = attr;
    split_point = sp;
}

void Node::set_errors(int e){
    errors = e;
}

void Node::set_children_errors(int e){
    children_errors = e;
}

vector<Node *> Node::get_children(){ // get node's children
    return children;
}

Node *Node::get_child( int i ){// get the i-th child of node
    return children[i];
}

Node *Node::get_parent(){// get parent node
    return parent;
}

string Node::get_class(){    // get class
    return assigned_class;
}
```

```
Dataset Node::get_dataset(){          // return sub-dataset
    return dataset;
}

int Node::get_attribute(){         // return which attribute is to decide on this node
    return attribute;
}

int Node::get_errors(){
    return errors;
}

int Node::get_children_errors(){
    return children_errors;
}

double Node::get_split_point(){     // return split point used for decision on this node (for numeric attributes)
    return split_point;
}

void Node::remove_child( int ch ){// remove a child from node's children
    children.erase( children.begin() + ch );
}

void Node::clear(){            // unset all properties

    for ( int i = 0; i < children.size(); i++ ){
        children[i]->clear();
    }

    children.clear();
    parent = NULL;

    dataset.clear();
```

```
    attribute = -1;
    split_point = -1;
}

void Node::print( int depth ){    // print node

    int i = 0, j = 0;

    if ( attribute != -1 ){

        Attribute_type attr_type=get_dataset().get_attribute_type( attribute );
        vector<string> nom_values;

        if( attr_type.get_attr_type() ) {  // if attribute is nominal

            nom_values=attr_type.get_nominal_values();

            for( i = 0; i < nom_values.size(); i++ ){

                j = 0;
                cout << endl;

                while( j < depth ){
                    cout << "|\t";
                    j++;
                }

                cout << attribute + 1 << " = " << nom_values[i];
                children[i]->print( depth + 1 );

            }
        } else {                  // if attribute is numeric

            // for data lower than split point
            j = 0;
            cout << endl;
```

```
            while( j < depth ){
                cout << "|\t";
                j++;
            }

            cout << attribute + 1 << " <= " << split_point;

            children[0]->print( depth + 1 );

            // for data bigger than split point
            j = 0;

            cout << endl;

            while( j < depth ){
                cout << "|\t";
                j++;
            }
            cout << attribute + 1 << " > " << split_point;

            children[1]->print( depth + 1 );
        }
    } else {
        cout << " : " << assigned_class ;
    }
}

vector<Dataset> Node::split_dataset(){// split dataset to give it to children nodes

    int i, j;

    bool flag;

    vector<Dataset> datasets;

    if ( dataset.get_attribute_type(attribute).get_attr_type() ){   // nominal attribute
```

```
        vector<string>                          nom_values                          =
dataset.get_attribute_type(attribute).get_nominal_values();
// all possible nominal values for this attribute

        datasets.resize( nom_values.size() );     // set the number of datasets to create

        for ( i = 0; i < dataset.get_instances(); i++ ){     // for every instance

            flag = true;

            for ( j = 0; j < nom_values.size() && flag; j++ ){    //  for  each  possible  nominal
value
                if    (    dataset.get_data(i).get_attribute(attribute).get_attribute_nominal()    ==
nom_values[j] ){
                    flag = false;
                    datasets[j].push_data( dataset.get_data(i) );
                }
            }
        }
    } else {                        // numeric attribute

        datasets.resize( 2 );           // set the number of datasets to create

        for ( i = 0; i < dataset.get_instances(); i++ ){     // for every instance
            // if attribute value less or equal than split point
            if     (     dataset.get_data(i).get_attribute(attribute).get_attribute_numeric()     <=
split_point ){
                datasets[0].push_data( dataset.get_data(i) );
            } else {       // if attribute value greater than split point
                datasets[1].push_data( dataset.get_data(i) );
            }
        }
    }

    // initialize the attributes of the datasets

    for ( i = 0; i < datasets.size(); i++ ){
```

```
        datasets[i].set_attributes( dataset.get_attributes() );
        datasets[i].set_attribute_types( dataset.get_attribute_types() );
        datasets[i].set_classes(dataset.get_classes());
    }

    dataset.clear();

    return datasets;
}

bool Node::find_condition() {

    int attributes = dataset.get_attributes(); // number of attributes
    double gain_ratio[attributes];              // gain ratio for each attribute
    double split_point[attributes];             // if attribute numeric, save it's split point
    double p[attributes];
    double sum = 0;
    double sum_p = 0;
    double inf_gain[attributes];
    double avg_inf_gain = 0;
    int counter = 0;
    double r;
    bool flag;
    int pos_max;                    // attribute that has the max gain_ratio

    dataset.find_entropy();             // find dataset entropy

    for ( int i=0; i < attributes; i++ ){         // for each attribute
        if ( !dataset.get_attribute_type(i).get_attr_type() ) {           // if attribute numeric
            // find best split point and gain ratio for attribute i
            split_point[i] = dataset.find_split_point( i, gain_ratio[i], inf_gain[i] );
        } else {                            // if attribute nominal
            // find gain ratio for attribute i
            gain_ratio[i] = dataset.find_gain_ratio(i, inf_gain[i]);
        }
```

```
        if ( inf_gain[i] <= INF && inf_gain[i] == inf_gain[i] && gain_ratio[i] > MIN_GAIN ){

            avg_inf_gain += inf_gain[i];
            counter++;

        }
    }

    // Find average information gain
    avg_inf_gain = avg_inf_gain / counter;

    for ( int i=0; i < attributes; i++ ){          // for each attribute
        if ( inf_gain[i] < avg_inf_gain ) {
            gain_ratio[i] = 0;
        } else if ( gain_ratio[i] == gain_ratio[i] ){
            sum += gain_ratio[i];
        }
    }

    // Calculate propability for each attribute to be selected to split

    for ( int i=0; i < attributes; i++ ){
        p[i] = gain_ratio[i] / sum;
    }

    // Select randomly the attribute to split
    r = (double) rand() / RAND_MAX;

    sum_p = 0;
    flag = true;
    pos_max = - 1;

    for ( int i=0; i < attributes && flag; i++ ){
        if ( r <= (p[i] + sum_p) ){
            pos_max = i;
            flag = false;
```

```
        } else {
            sum_p += p[i];
        }
    }

    if ( pos_max != -1 && gain_ratio[pos_max] > MIN_GAIN ){

        if ( dataset.get_attribute_type(pos_max).get_attr_type() ){  // if attribute nominal

            set_condition(pos_max);

        } else {                                // if attribute numeric
            set_condition(pos_max, split_point[pos_max]);
        }
        return true;
    } else {
        return false;
    }
}

void Node::create_subtree(){

    int tmp;
    bool flag = true;

    dataset.find_entropy();

    if ( parent != NULL &&     dataset.get_instances()                    ==                    parent-
>get_dataset().get_instances() )
        flag = false;

    MIN_INSTANCES = 0.1 * ( dataset.get_instances() / dataset.num_of_classes() );

    if ( MIN_INSTANCES < MIN_SPLIT_INSTANCES ) {
        MIN_INSTANCES = MIN_SPLIT_INSTANCES;
    } else if ( MIN_INSTANCES > MAX_SPLIT_INSTANCES ) {
        MIN_INSTANCES = MAX_SPLIT_INSTANCES;
```

```
    }

    int tmp_errors = 0;

    if ( ( dataset.get_instances() >=  (MIN_INSTANCES * 2) ) && dataset.get_entropy() >
0 ){// if it worth to make children nodes

        flag = find_condition();

        if ( flag ) {

            create_children();

            for(int i = 0; i < children.size(); i++){
                children[i]->create_subtree();
                tmp_errors += children[i]->get_errors();
            }

            set_children_errors( tmp_errors );

            if ( get_errors() <= get_children_errors() ){
                set_condition( -1 );
            } else {
                set_errors( get_children_errors() );
            }
        } else {
            make_leaf();
        }
    } else {
        make_leaf();
    }
}

void Node::make_leaf(){

        bool flag;
        int num_of_classes = dataset.num_of_classes();
```

```
        int max_class;
        int *counter_classes;
        counter_classes = new int[ num_of_classes ];

        for ( int j = 0; j < num_of_classes; j++ ){
            counter_classes[j] = 0;
        }

        for ( int i = 0; i < dataset.get_instances(); i++){

            flag = true;

            for ( int j = 0; j < num_of_classes && flag; j++ ){
                if ( dataset.get_data(i).get_class() == dataset.get_class(j) ) {

                    counter_classes[j]++;
                    flag = false;

                }
            }
        }

        max_class = distance(counter_classes, max_element(counter_classes, counter_classes
+ num_of_classes) );

        set_class( dataset.get_class(max_class) );
        set_errors( dataset.get_instances() - counter_classes[max_class] );

}


//                    Class                    Tree                    methods
-------------------------------------------------------------------------------------------------

Tree::Tree(){
    root = new Node;
}
```

```
void Tree::initialize(Dataset dataset){              // initialize root node
    clear();
    root->set_dataset(dataset);
    root->set_parent(NULL);
    root->make_leaf();
}

Node* Tree::get_root(){              // return root node
    return root;
}

void Tree::set_root(Node *root_addr){       // set root node
    root=root_addr;
}


void Tree::create_tree(){              // build tree from train set( root's dataset )
    root->create_subtree();
}

string Tree::classify(Data_point data){
    Node *n;
    bool flag;
    int attribute;
    n = root;
    attribute = n->get_attribute();

    while ( attribute != -1 ) {     // while it is not a leave

        // nominal attribute
        if ( n->get_dataset().get_attribute_type( attribute ).get_attr_type() ){

            // all possible nominal values for this attribute
            vector<string>                  nom_values                  =                  n-
>get_dataset().get_attribute_type(attribute).get_nominal_values();
```

```
            flag = true;

            for ( int j = 0; j < nom_values.size() && flag; j++ ){    //    for    each    possible
nominal value

                if ( data.get_attribute(attribute).get_attribute_nominal() == nom_values[j] ){

                    flag = false;
                    n = n->get_child(j);
                    attribute = n->get_attribute();

                }
            }


        } else { // numeric attribute

            if ( data.get_attribute(attribute).get_attribute_numeric() <= n->get_split_point() )
            {
                n = n->get_child(0);
                attribute = n->get_attribute();
            } else {
                n = n->get_child(1);
                attribute = n->get_attribute();
            }
        }
    }
    return n->get_class();
}

void Tree::clear(){
    root->clear();
}
```

9.2.5 main.cpp

```
#include <iostream>
```

```
#include <fstream>
#include <vector>
#include <algorithm>

#include "stoch_forest.h"

#define FOREST_TREES 100

using namespace std;

int main(int argc, char** argv){

    if ( argc > 2 ){

        cout << "Error: Expects only one arguments. Example of use ./<executable> filename" << endl;
        return -1;

    }

    // variables and objects ---------------------------------------------------------------------------

    Dataset dataset;            // the dataset
    vector<Dataset> datasets;       // sub datasets
    vector<Dataset> tmp_datasets;       // temp vector of datasets
    Dataset training_dataset, test_dataset;         // training and test dataset for testing

    vector<Dataset> data_of_class;      // vector of datasets. Each dataset haw all data points of a class.

    string filename = argv[1];          // get filename from command line

    Tree tree[FOREST_TREES];

    // code ---------------------------------------------------------------------------
```

```
dataset.arff_read(filename);            // read arff file and save dataset

dataset.print_info();                   // print info of dataset

datasets = split_dataset(dataset);      // split to sub datasets for 10 fold
cout << endl << endl << endl << endl << endl << endl;

// initialiaze confusion matrix,
int conf_matrix[dataset.num_of_classes()][dataset.num_of_classes()];

for (int i = 0; i < dataset.num_of_classes(); i++){
    for (int j = 0; j < dataset.num_of_classes(); j++){
        conf_matrix[i][j] = 0;
    }
}

// ---------------------------------------------------------------------

cout << endl << "Starting stratified 10 fold ... " << endl;
srand(time(0)); // use current time as seed for random generator
cout << endl << endl << endl << endl << endl << endl;
int folds = 10;         // number of folds
int max;      // the class with the max votes


for (int i = 0; i < folds; i++){
    cout << endl << "Fold " << i+1 << ":" << endl << endl;

    // Make training and test datasets ------------------------------------------------

    tmp_datasets = make_datasets(datasets, folds - 1, i);    // make training and test
dataset

    training_dataset = tmp_datasets[0];
    test_dataset = tmp_datasets[1];

    // create forest ------------------------------------------------------------------
```

```
cout << "Create forest ..." << endl;

for ( int j = 0; j < FOREST_TREES; j++){
    // create tree     ---------------------------------------------------------------------

    tree[j].initialize(training_dataset);
    tree[j].create_tree();
}

// Check results ------------------------------------------------------------

cout << "Check results ..." << endl;

string tmp_class;    // where the data point j is clasified by tree t. Temporary saved

short int counter_classes[test_dataset.num_of_classes()];

// Classify -----------------------------------------------------------

for (int j = 0; j < test_dataset.get_instances(); j++){

    fill_n(counter_classes, dataset.num_of_classes(), 0);

    for ( int t = 0; t < FOREST_TREES; t++){

        tmp_class = tree[t].classify( test_dataset.get_data(j) );

        for ( int c = 0; c < test_dataset.num_of_classes(); c++){

            if ( tmp_class == test_dataset.get_class(c) ){

                counter_classes[c]++;   // voting !!!!

            }
        }
    }
```

```
            // Pick the class with the most 'votes'

            max = distance(counter_classes, max_element( counter_classes, counter_classes +
test_dataset.num_of_classes() ) );

            // Validating results ----------------------------------------------------

            string real_class = test_dataset.get_data(j).get_class();  // real class of data point j
            string assigned_class = dataset.get_class(max);            // where data point j is
classified

            int x,y; // x line, y column

            for ( int c = 0; c < test_dataset.num_of_classes(); c++){

                if ( real_class.compare(test_dataset.get_class(c)) == 0 ){
                    x = c;   // x line of conf matrix
                }

                if ( assigned_class.compare(test_dataset.get_class(c)) == 0 ){

                    y = c;   // y column of conf matrix

                }

            }

            conf_matrix[x][y]++;

        }
    }

    // Print results   ------------------------------------------------------------------------

    double correct = 0;
    double recall_tmp, precision_tmp;
```

```
double class_correct[dataset.num_of_classes()];
fill_n(class_correct, dataset.num_of_classes(), 0);
double sum_precision[dataset.num_of_classes()], sum_recall[dataset.num_of_classes()];

fill_n(sum_precision, dataset.num_of_classes(), 0);

fill_n(sum_recall, dataset.num_of_classes(), 0);

double avg_recall = 0, avg_precision = 0;

cout << "Confusion matrix: " << endl << endl;

for (int i = 0; i < dataset.num_of_classes(); i++){
    for (int j = 0; j < dataset.num_of_classes(); j++){

        sum_precision[j] += conf_matrix[i][j];

        sum_recall[i] += conf_matrix[i][j];

        if ( i == j ){
            correct += conf_matrix[i][j];
            class_correct[i]+= conf_matrix[i][j];
        }

        cout << conf_matrix[i][j] << "\t ";
    }

    cout << endl;
}

for (int i = 0; i < dataset.num_of_classes(); i++){

    cout << endl << "Class " << dataset.get_class(i) << ":" << endl << endl;

    if ( sum_recall[i] != 0){

        recall_tmp = (class_correct[i]/sum_recall[i]) * 100;
```

```
            cout << "Recall: " << recall_tmp << " %" << endl;

            avg_recall += recall_tmp;

        }

        if ( sum_precision[i] != 0){

            precision_tmp = (class_correct[i]/sum_precision[i])*100;

            cout << "Precision: " << precision_tmp << " %" << endl;

            avg_precision += precision_tmp;

        }
    }

    cout << endl;

    avg_recall = avg_recall / dataset.num_of_classes();
    avg_precision = avg_precision / dataset.num_of_classes();

    cout << endl << "Accuracy: " << (correct/dataset.get_instances())*100 << " %" << endl;
    cout << "Average recall: " << avg_recall << " %" << endl;
    cout << "Average precision: " << avg_precision << " %" << endl;

    cout << endl;                    // final new line

    return 0;

}
```