ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΕΡΡΩΝ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

Τμήμα Πληροφορικής και Επικοινωνιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κατασκευή ταξινομητών weighted kNN με metric ball trees για εφαρμογές ανακάλυψης γνώσης από βάσεις δεδομένων Oracle

Γεροθανάσης Εμμανουήλ

ΑΕΜ 1723

Μπέκος Ευάγγελος

ΑΕΜ 1986

Επιβλέπων: Κόκκινος Ιωάννης

Σέρρες, 2012

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1 ΕΞΟΡΥΞΗ ΓΝΩΣΗΣ ΚΑΙ ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ORACLE

Εισαγωγή στην εξόρυξη γνώσης

Η εξόρυξη δεδομένων συνίσταται στην ανακάλυψη ενδιαφερόντων τάσεων ή προτύπων σχημάτων μέσα σε μεγάλα σύνολα δεδομένων, με σκοπό να καθοδηγήσει αποφάσεις σχετικές με μελλοντικές δραστηριότητες. Υπάρχει μια γενική προσδοκία ότι τα εργαλεία εξόρυξης δεδομένων θα πρέπει να είναι ικανά να αναγνωρίζουν τέτοια πρότυπα σχήματα μέσα στα δεδομένα με την ελάχιστη διαλογική συμμετοχή του χρήστη. Τα πρότυπα σχήματα που αναγνωρίζονται από τέτοια εργαλεία μπορούν να παρέχουν σε έναν αναλυτή δεδομένων χρήσιμες και απροσδόκητες ερμηνείες, οι οποίες μπορούν να διερευνηθούν περισσότερο προσεκτικά στη συνέχεια, ενδεχομένως με τη χρήση άλλων εργαλείων υποστήριξης αποφάσεων.

Η εξόρυξη δεδομένων σχετίζεται με μια περιοχή της στατιστικής που ονομάζεται διερευνητική ανάλυση δεδομένων, η οποία έχει τους ίδιους στόχους και βασίζεται στα στατιστικά μέτρα. Είναι επίσης στενά συνδεδεμένη με περιοχές της τεχνητής νοημοσύνης, όπως η ανακάλυψη γνώσης και μηχανοποιημένη εκμάθηση. Το σημαντικό χαρακτηριστικό που ξεχωρίζει στην εξόρυξη δεδομένων είναι ο πολύ μεγάλος όγκος των δεδομένων. Παρόλο που έννοιες από τις περιοχές αυτές είναι εφαρμόσιμες σε προβλήματα εξόρυξης δεδομένων, η δυνατότητα κλιμάκωσης σε σχέση με το μέγεθος των δεδομένων αποτελεί ένα νέο σημαντικό κριτήριο. Ένας αλγόριθμος μπορεί και κλιμακώνεται εάν ο χρόνος εκτέλεσης του αυξάνεται ανάλογα με το μέγεθος του συνόλου δεδομένων στα οποία επενεργεί (δηλ. γραμμικά), για δεδομένα μεγέθη των διαθέσιμων πόρων του συστήματος (π.χ. μέγεθος της κύριας μνήμης και το μέγεθος του δίσκου). Οι ήδη υπάρχοντες αλγόριθμοι θα πρέπει να προσαρμοστούν ή νέοι αλγόριθμοι θα πρέπει να αναπτυχθούν, έτσι ώστε να εξασφαλίζεται η δυνατότητα της κλιμάκωσης.

Η ανακάλυψη χρήσιμων τάσεων σε σύνολα δεδομένων αποτελεί έναν αρκετά χαλαρό ορισμό της εξόρυξης δεδομένων. Υπό μία ορισμένη έννοια, όλα τα αιτήματα σε βάσεις δεδομένων, μπορεί να θεωρηθούν ότι κάνουν ακριβώς αυτό. Πράγματι, έχουμε μία συνεχή σειρά εργαλείων ανάλυσης και διερεύνησης, με αιτήματα SQL στο ένα άκρο, αιτήματα OLAP στη μέση και τεχνικές εξόρυξης δεδομένων στο άλλο άκρο. Τα αιτήματα SQL διατυπώνονται χρησιμοποιώντας τη σχεσιακή άλγεβρα (με κάποιες επεκτάσεις), ενώ τα αιτήματα OLAP συνιστούν υψηλότερου επιπέδου σύνταξη που βασίζεται στη χρήση πολυδιάστατων μοντέλων δεδομένων. Η λειτουργικότητα της εξόρυξης δεδομένων εντοπίζεται να ανήκει σε έναν περισσότερο αφηρημένο χώρο ανάλυσης. Μπορούμε να θεωρήσουμε τις διάφορες ενέργειες εξόρυξης δεδομένων ως σύνθετα «αιτήματα», τα οποία καθορίζονται σε υψηλό

επίπεδο με μερικές παραμέτρους που προσδιορίζονται από τον χρήστη και τα οποία υλοποιούνται με τη χρήση περισσότερο εξειδικευμένων αλγορίθμων.

Στην πράξη, η εξόρυξη δεδομένων δεν συνίσταται απλά στην εφαρμογή ενός από αυτούς τους αλγόριθμους. Τα δεδομένα συχνά περιέχουν θόρυβο ή είναι ημιτελή και αν αυτό δε γίνει αντιληπτό και δε διορθωθεί, τότε είναι πιθανό πολλά ενδιαφέροντα πρότυπα σχήματα να μην ανιχνευθούν ενώ η αξιοπιστία των εντοπισμένων προτύπων να είναι χαμηλή. Επιπλέον, ο αναλυτής πρέπει να αποφασίσει ποιους αλγόριθμους εξόρυξης θα χρησιμοποιήσεις, να τους εφαρμόσει σ΄ ένα ορθά επιλεγμένο υποσύνολο από δείγματα δεδομένων και μεταβλητών (δηλαδή πλειάδων και γνωρισμάτων), να συνοψίσει τα αποτελέσματα, να εφαρμόσει άλλα εργαλεία υποστήριξης αποφάσεων και εξόρυξης και να επαναλάβει τη διεργασία.

Η διεργασία της ανακάλυψης γνώσης σε βάσεις δεδομένων ή εν συντομία διεργασία KDD, μπορεί πρόχειρα να χωριστεί σε τέσσερα στάδια. Τα ακατέργαστα δεδομένα περνούν αρχικά από το πρώτο στάδιο, που είναι η επιλογή δεδομένων, κατά το οποίο προσδιορίζουμε το σύνολο δεδομένων και τα γνωρίσματα εκείνα που μας ενδιαφέρουν σε σχέση. Κατά το δεύτερο στάδιο, τον καθαρισμό των δεδομένων, απομακρύνουμε το θόρυβο και τις προς εξαίρεση τιμές, μετασχηματίζουμε τις τιμές των πεδίων σε κοινές μονάδες μέτρησης, δημιουργούμε νέα πεδία συνδυάζοντας τα ήδη υπάρχοντα και φέρνουμε τα δεδομένα στο σχεσιακό σχήμα το οποίο θα χρησιμοποιηθεί στην είσοδο της επεξεργασίας της εξόρυξης δεδομένων. Το στάδιο του καθαρισμού των δεδομένων, μπορεί να περιλαμβάνει και την αποκανονικοποίηση των σχετικών πινάκων. Στο στάδιο της εξόρυξης δεδομένων, εξάγουμε τα πραγματικά πρότυπα σχήματα. Στο τελικό στάδιο, το στάδιο της αξιολόγησης, τα παρουσιάζουμε σε μια κατανοητή μορφή για το τελικό χρήστη, για παράδειγμα με παραστατικό, οπτικό τρόπο. Τα αποτελέσματα οποιουδήποτε βήματος μπορεί να μας οδηγήσουν πίσω σε κάποιο προηγούμενο στάδιο με σκοπό να επαναληφθεί η διεργασία χρησιμοποιώντας τη νέα γνώση που έχει αποκτηθεί.

ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΕΞΟΡΥΞΗΣ ΓΝΩΣΗΣ

Εύρεση Κατηγοριών (Classes)

Τα αποθηκευμένα δεδομένα χρησιμοποιούνται για να εντοπίσουν πληροφορίες για προκαθορισμένες κατηγορίες/κλάσεις. Παραδείγματος χάριν, μια αλυσίδα εστιατορίων θα μπορούσε να εξαγάγει τις καταναλωτικές συνήθειες των πελατών, που καθορίζονται από τις επισκέψεις αυτών και να τις αναλύσει, λειτουργώντας με βάση το τι πραγματικά η ανάλυση αυτή επιτάσσει. Αυτές οι πληροφορίες θα μπορούσαν να ενδεχομένως να χρησιμοποιηθούν για να αυξήσουν την κατανάλωση των σπεσιαλιτέ της ημέρας.

Εύρεση Ομάδων/συστάδων (Clusters)

Τα δεδομένα ομαδοποιούνται σύμφωνα με λογικές σχέσεις ή καταναλωτικές προτιμήσεις. Παραδείγματος χάριν, τα δεδομένα μπορούν να εξαχθούν για να προσδιορίσουν τους τομείς της αγοράς ή τις καταναλωτικές συγγένειες.

Εύρεση κανόνων συσχετίσεων (Association rules)

Τα δεδομένα μπορούν να εξαχθούν για να προσδιορίσουν τις σχέσεις. Το παράδειγμα μπύρα-πάνες είναι ένα παράδειγμα του συνειρμικού-σχεσιακού data mining.

Εύρεση Σειριακών μοτίβων (Sequential Patterns)

Τα δεδομένα εξάγονται ώστε να προβλεφθούν τα μοτίβα και οι τάσεις συμπεριφοράς. Παραδείγματοςχάριν,έναςπωλητήςεξοπλισμούειδώνεξοχής,θαμπορούσεναπροβλέψε ιτηνπιθανότηταπώλησηςενόςσακιδίουπλάτης,βασισμένοςστηναγοράαπόέναν πελάτη, υπνόσακουκαιπαπουτσιώνπεζοπορίας.

ΕΡΓΑΛΕΙΑ ΕΞΟΡΥΞΗΣ ΓΝΩΣΗΣ

Τα διαφορετικά εργαλεία ανάλυσης είναι πολλά. Μερικά είναι τα:

- Δέντρα απόφασης (Decision Trees): Δέντρο-διαμορφωμένες δομές που αντιπροσωπεύουν τα σύνολα αποφάσεων. Αυτές οι αποφάσεις παράγουν τους κανόνες για την ταξινόμηση ενός συνόλου δεδομένων. Μια απλή δομή όπου οι μη τερματικοί κόμβοι αντιπροσωπεύουν τα αποτελέσματα των αποφάσεων. Τα δέντρα αποφάσεων έχουν διάφορα πλεονεκτήματα, όπως το

ότι είναι εύκολο να τα καταλάβουμε, μπορούν να μετασχηματιστούν σε κανόνες και πειραματικά έχει αποδειχθεί ότι λειτουργούν πολύ καλά.

- Μέθοδος κοντινότερων γειτόνων (Nearest neighbor method): Μια τεχνική που ταξινομεί κάθε εγγραφή σε ένα σύνολο δεδομένων βασισμένο σε έναν συνδυασμό των ταξινομήσεων των εγγραφών K και του πιο κοντινού "συγγενή" με το K σε ένα ιστορικό σύνολο δεδομένων.
- Επαγωγή κανόνα (Rule induction): Η εξαγωγή των χρήσιμων if-then κανόνων από τα δεδομένα, βασιζόμενα στη στατιστική σημασία.
- Απεικόνιση στοιχείων (Data visualization): Η οπτική ερμηνεία των σύνθετων σχέσεων στα πολυδιάστατα δεδομένα. Τα εργαλεία γραφικής αναπαράστασης χρησιμοποιούνται για να επεξηγήσουν τις σχέσεις των δεδομένων.
- Τεχνητά νευρωνικά δίκτυα (Artificial neural networks): Μη γραμμικά προβλεπτικά μοντέλα που μαθαίνουν μέσω της εκπαίδευσης και μοιάζουν στη δομή με τα βιολογικά νευρικά δίκτυα.

ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΤΟ ΠΑΚΕΤΟ ORACLE 9i

Με τον όρο βάση δεδομένων εννοείται μία συλλογή από συστηματικά οργανωμένα (formatted) σχετιζόμενα δεδομένα. Ένας τηλεφωνικός κατάλογος, για παράδειγμα, θεωρείται βάση δεδομένων, καθώς αποθηκεύει και οργανώνει σχετιζόμενα τμήματα πληροφορίας, όπως είναι το όνομα και ο αριθμός τηλεφώνου. Ωστόσο, στον κόσμο των υπολογιστών, με τον όρο βάση δεδομένων αναφερόμαστε σε μια συλλογή σχετιζόμενων δεδομένων τμημάτων πληροφορίας ηλεκτρονικά αποθηκευμένων.

Πέρα από την εγγενή της ικανότητα να αποθηκεύει δεδομένα, η βάση δεδομένων παρέχει βάσει του σχεδιασμού και του τρόπου ιεράρχησης των δεδομένων της σε προγράμματα ή συλλογές προγραμμάτων, τα αποκαλούμενα συστήματα διαχείρισης περιεχομένου, τη δυνατότητα γρήγορης άντλησης και ανανέωσης των δεδομένων. Η ηλεκτρονική βάση δεδομένων χρησιμοποιεί ιδιαίτερου τύπου λογισμικό προκειμένου να οργανώσει την αποθήκευση των δεδομένων της. Το διακριτό αυτό λογισμικό είναι γνωστό ως Σύστημα διαχείρισης βάσης δεδομένων συντομευμένα (DBMS). Τα κατά κόρον χρησιμοποιούμενα πακέτα για βάσεις δεδομένων είναι τα παρακάτω MySQL, oracle, mssql, sql server και access ενώ ως γλώσσες εφαρμογών χρησιμοποιούντε οι: php, asp, perl.

Η Oracle ανήκει στις Σχεσιακές Βάσεις Δεδομένων (Relational DataBases), δηλ. στηρίζεται σε σχέσεις (relations) που δηλώνονται με βάση τα κοινά πεδία διαφορετικών πινάκων (tables).

Η Oracle διαθέτει την γλώσσα αναζήτησης ή ερωτημάτων (query language) SQL*Plus, με την βοήθεια της οποίας μπορούμε να διαχειριστούμε τις πληροφορίες μιας βάσης δεδομένων της Oracle. Με την SQL*Plus μπορούμε να δημιουργήσουμε πίνακες, εγγραφές, πεδία και σχέσεις και στην συνέχεια να κάνουμε εργασίες ανεύρεσης και ενημέρωσης (τροποποίησης) των αποθηκευμένων δεδομένων, παρέχοντας έτσι ένα δυναμικό εργαλείο διαχείρισης ενός συστήματος πληροφόρησης. Πριν κάνουμε ο,τιδήποτε, πρέπει να έχουμε υπόψη μας

ότι παίζει πολύ μεγάλο ρόλο η σωστή οργάνωση της βάσης δεδομένων και η δημιουργία των σωστών σχέσεων μεταξύ των αρχείων που την αποτελούν

(DataBase design).

Αν δουλεύουμε σαν χρήστες της Oracle σ' ένα μεγάλο σύστημα, ο Διαχειριστής της Βάσης Δεδομένων, DataBase Administrator (DBA), θα πρέπει να μας παραχωρήσει ένα όνομα χρήστη (user name) της Oracle για όσο χρόνο χρησιμοποιούμε την Oracle.

Πλεονεκτήματα της ORACLE:

☐ Ευκολία διαχείρισης και εγκατάστασης.

☐ Παροχή μηχανισμών ασφαλείας και ανάκτησης δεδομένων

☐ Μηχανισμοί που την καθιστούν ταχύτατη σε βαριά και απαιτητικά περιβάλλοντα.

☐ Εξαιρετικά μεγάλη και καλή τεκμηρίωση.

Μειονεκτήματα της ORACLE:

☐ Υψηλότατο κόστος.

☐ Πολύ υψηλές απαιτήσεις σε πόρους συστήματος.

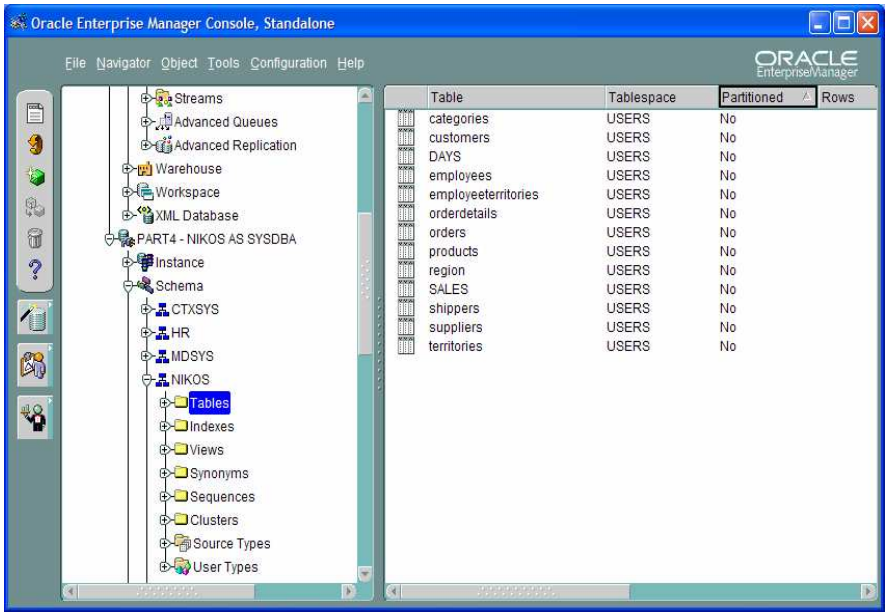
☐ Μηχανισμοί που την καθιστούν ταχύτατη σε βαριά και απαιτητικά περιβάλλοντα.

☐ Απαιτεί καλή εμπειρία στη διαχείριση και στη συντήρησή της.

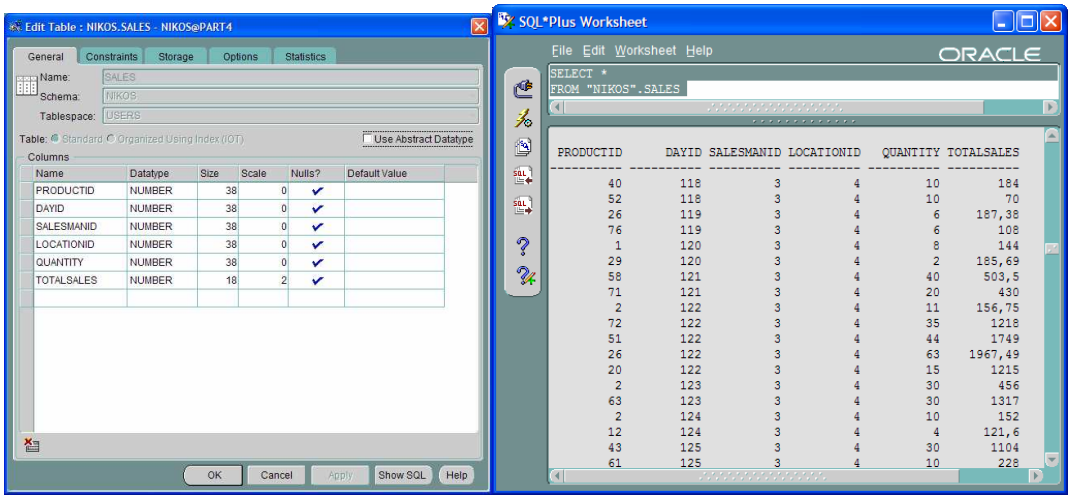
Το σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων Oracle σχεδιάστηκε για να επιτρέψει την ταυτόχρονη πρόσβαση σε μεγάλες κατανεμημένες βάσεις δεδομένων.

Η βάση δεδομένων διαιρείται σε ένα ή περισσότερα λογικά κομμάτια που είναι γνωστά ως tablespaces .Ένα tablespace χρησιμοποιείται για να συγκεντρώσει τα δεδομένα. Το μέγιστο μέγεθος ενός datafile είναι 32GB (gigabytes). Ο μέγιστος αριθμός datafiles ανά tablespace είναι 1,022. Το μέγιστο μέγεθος ενός tablespace είναι 32TB (terabyte)[B3]. Η ονοματολογία των πινάκων και των πεδίων αυτών ακολουθεί τους κανόνες του σχεσιακού μοντέλου.

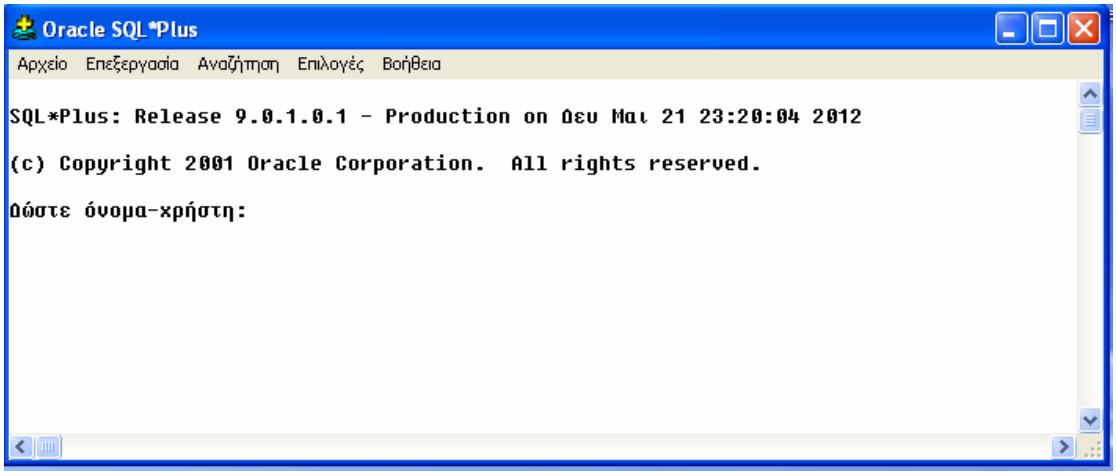
Ο Oracle Enterprise Manager είναι το πρωτεύον εργαλείο διαχείρισης των βάσεών της.



Το περιβάλλον SQL * Plus Worksheet χρησιμοποιείται για την άμεση εκτέλεση δηλώσεων SQL και εμφάνιση αποτελεσμάτων.



Ένα πολύ δυνατό εργαλείο που προσφέρει το πακέτο oracle είναι το oracle SQL *Plus που μας επιτρέπει να χειριστούμε τα δεδομένα των βάσεων μας με online χρήση μέσω εντολών SQL. Επίσης επιτρέπει την απομακρυσμένη διαχείριση



Κεφάλαιο 2 ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ

Βασικές Έννοιες Κατηγοριοποίησης

Η κατηγοριοποίηση (classification) είναι η πιο γνωστή και πιο δημοφιλής τεχνική εξόρυξης γνώσης(data mining). Πολλές εταιρίες του ιδιωτικού και του δημοσίου τομέα χρησιμοποιούν σε καθημερινή βάση συστήματα κατηγοριοποίησης. Παραδείγματα τέτοιου είδους συστημάτων είναι τα συστήματα ιατρικών διαγνώσεων, συστήματα έγκρισης δανείων και πιστωτικών καρτών, συστήματα ανίχνευσης λαθών σε τεχνολογικές εφαρμογές, συστήματα κατηγοριοποίησης των τάσεων στην οικονομία κ.α. Για παράδειγμα όταν κάποιος προβλέπει μια ηλικία, στην ουσία επιλύει ένα πρόβλημα κατηγοριοποίησης. Ένα άλλο, πιο καλά ορισμένο, παράδειγμα παρουσιάζεται παρακάτω:

Παράδειγμα 1: Οι δάσκαλοι κατηγοριοποιούν τους μαθητές τους ως A,B,C,D ή F με βάση τους βαθμούς τους. Χρησιμοποιώντας απλά όρια(60,70,80,90) μπορούμε να έχουμε τον παρακάτω διαχωρισμό των μαθητών σε κλάσεις:

$$90 \leq \text{βαθμός} \rightarrow \text{A},$$
$$80 \leq \text{βαθμός} < 90 \rightarrow \text{B},$$
$$70 \leq \text{βαθμός} < 80 \rightarrow \text{C},$$
$$60 \leq \text{βαθμός} < 70 \rightarrow \text{D},$$
$$\text{Βαθμός} < 60 \rightarrow \text{F}$$

Όλες οι προσεγγίσεις στην εκτέλεση της κατηγοριοποίησης προϋποθέτουν γνώση των δεδομένων. Συνήθως χρησιμοποιούμε ένα σύνολο εκπαίδευσης για να καθορίσει τις συγκεκριμένες παραμέτρους που απαιτούνται από την τεχνική. Τα δεδομένα εκπαίδευσης(training data) αποτελούνται από ένα δείγμα δεδομένων εισόδου καθώς επίσης και από την κατηγοριοποίηση που έχει δοθεί σε αυτά τα δεδομένα.

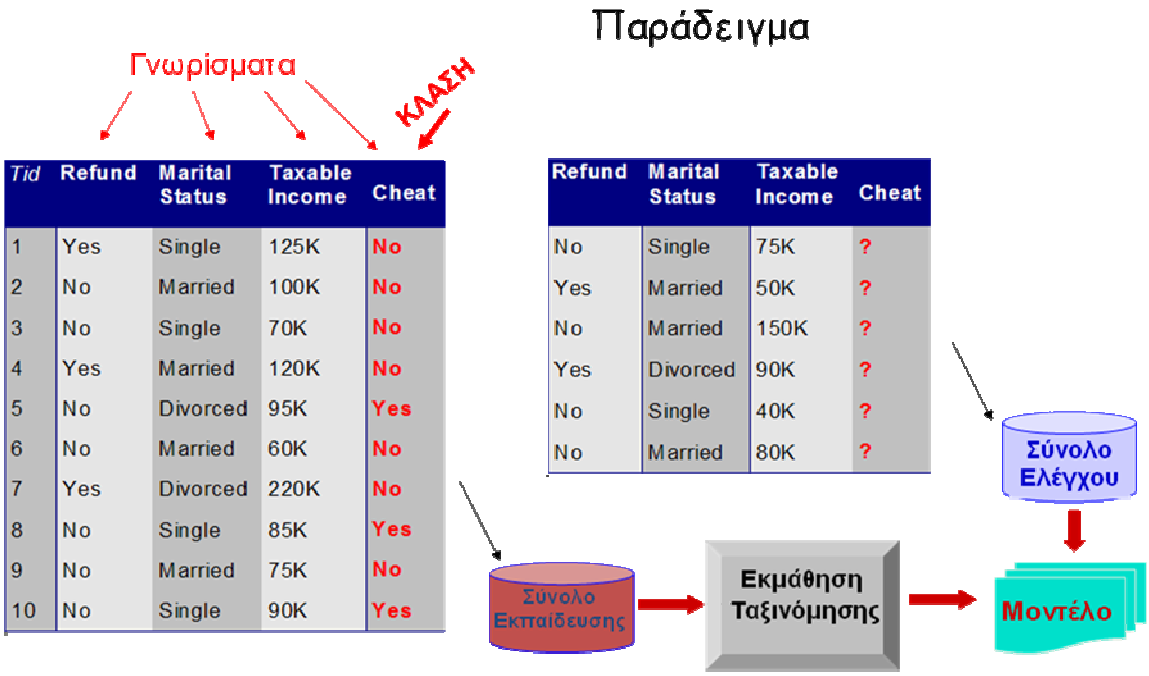
Πρέπει να υπογραμμίσουμε ότι οι κατηγορίες είναι προκαθορισμένες, δεν επικαλύπτονται και διαμερίζουν ολόκληρη τη Βάση Δεδομένων. Κάθε στοιχείο της βάσης δεδομένων τοποθετείται σε ακριβώς μία κατηγορία.

Η επίλυση των προβλημάτων κατηγοριοποίησης περιλαμβάνει δύο βασικά στάδια:

Πρώτο στάδιο: Δημιουργούμε ένα μοντέλο από την αξιολόγηση και την ανάλυση των δεδομένων εκπαίδευσης. Αυτό το βήμα έχει σαν είσοδο τα δεδομένα εκπαίδευσης και σαν έξοδο έναν ορισμό του μοντέλου που αναπτύχθηκε. Το μοντέλο που δημιουργείται από αυτό το στάδιο είναι σε θέση να κατηγοριοποιεί τα δεδομένα εκπαίδευσης με όσο το δυνατόν μεγαλύτερη ακρίβεια. Όταν είναι ήδη γνωστές οι κατηγορίες του συνόλου των δεδομένων εκπαίδευσης, δηλαδή το σύνολο των

δεδομένων εκπαίδευσης περιλαμβάνει ένα χαρακτηριστικό το οποίο δείχνει την κλάση στην οποία κατηγοριοποιείται η κάθε πλειάδα, τότε το βήμα αυτό καλείται *εποπτευόμενη μάθηση* (supervised learning), σε αντίθετη περίπτωση, δηλαδή αν δεν είναι γνωστές οι κατηγορίες του συνόλου των δεδομένων εκπαίδευσης, τότε το βήμα αυτό καλείται *μη-εποπτευόμενη μάθηση* (unsupervised learning).

Δεύτερο Στάδιο: Εφαρμόζουμε το μοντέλο που αναπτύχθηκε στο προηγούμενο βήμα κατηγοριοποιώντας τις πλειάδες της υπό εξέταση Βάσης Δεδομένων (μελλοντικές περιπτώσεις).



Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Refund	Marital Status	Taxable Income	Cheat
No	Single	75K	?
Yes	Married	50K	?
No	Married	150K	?
Yes	Divorced	90K	?
No	Single	40K	?
No	Married	80K	?

Προεπεξεργασία

Είναι πολύ συχνό το φαινόμενο κατά το οποίο τιμές χαρακτηριστικών (attributes) των πλειάδων του συνόλου δεδομένων είναι λανθασμένες, ασυνεπείς και ελλιπείς. Το φαινόμενο των λανθασμένων τιμών προέρχεται από ανθρώπινα λάθη ή λάθη του υπολογιστή. Αντίστοιχα το φαινόμενο των ασυνεπών δεδομένων προέρχεται από ενοποιήσεις δεδομένων όπου ένα χαρακτηριστικό έχει διαφορετικό όνομα στις διαφορετικές βάσεις δεδομένων. Επίσης το φαινόμενο των ελλιπών δεδομένων προέρχεται από μη εισαγωγή στοιχείων για κάποιες συγκεκριμένες πλειάδες την ώρα της εισαγωγής αφού αυτά την συγκεκριμένη στιγμή δεν είχαν αξία.

Έτσι πριν ξεκινήσει η διαδικασία εκπαίδευσης του αλγορίθμου κατηγοριοποίησης και της δοκιμής του θα πρέπει να γίνει το λεγόμενο καθάρισμα των δεδομένων (data cleaning).

Το φαινόμενο των ελλιπών δεδομένων μπορεί να αντιμετωπιστεί με έναν από τους παρακάτω τρόπους:

- ✓ Αγνόησε το παράδειγμα: Αυτό γίνεται όταν λείπει η τιμή του χαρακτηριστικού της κλάσης.
- ✓ Γέμισε τις τιμές που λείπουν με το χέρι: χρονοβόρα μέθοδος. Δεν είναι εφικτή αν είναι πάρα πολλές οι πλειάδες.
- ✓ Χρησιμοποίησε μια σταθερά για το γέμισμα των τιμών που λείπουν (π.χ.«unknown»). Το σύστημα θα χρησιμοποιήσει λανθασμένα αυτού του είδους τις τιμές. Αν και είναι απλή μέθοδος δεν προτείνεται.
- ✓ Χρήση του μέσου όρου των τιμών του χαρακτηριστικού που λείπει για την συμπλήρωση των ελλιπών τιμών.
- ✓ Χρήση του μέσου όρου των τιμών του χαρακτηριστικού για όλες τις πλειάδες που ανήκουν στην ίδια κλάση. π.χ. συμπλήρωση των τιμών που λείπουν με τον μέσο όρο του εισοδήματος για τους πελάτες με το ίδιο credit_risk.
- ✓ Χρήση της πιο πιθανής τιμής

Απόδοση Της Κατηγοριοποίησης

Οι αλγόριθμοι κατηγοριοποίησης είναι αρκετοί. Η ερώτηση που γεννιέται τώρα σχετίζεται με το ποιος είναι ο καλύτερος. Η επίδοση των αλγορίθμων εξετάζεται με την εκτίμηση της **ακρίβειας(accuracy)** της κατηγοριοποίησης, δηλαδή την ικανότητα του μοντέλου να προβλέπει την κατηγορία μιας νέας περίπτωσης. Η εκτίμηση της ακρίβειας είναι ένα πολύ σημαντικό ζήτημα στο χώρο της κατηγοριοποίησης αφού κάτι τέτοιο μας δείχνει πόσο καλά ανταποκρίνεται ο αλγόριθμος μας για δεδομένα με τα οποία δεν έχει εκπαιδευτεί. Η εκτίμηση της ακρίβειας είναι επίσης θεμιτή αφού μας επιτρέπει την σύγκριση των διαφόρων αλγόριθμων κατηγοριοποίησης.

Αν και η ακρίβεια είναι το πιο σημαντικό μέτρο αποτίμησης της απόδοσης της απόδοσης του αλγορίθμου κατηγοριοποίησης που χρησιμοποιούμε, υπάρχουν και άλλα μέτρα σύγκρισης:

- **Ταχύτητα:** Κόστος υπολογισμού(συμπεριλαμβανομένου την παραγωγή και την χρήση του μοντέλου)
- **Rebustness:** Σωστή πρόβλεψη με ελλιπή δεδομένα ή δεδομένα με θόρυβο
- **Scalability:** Αποδοτική κατασκευή του μοντέλου δοθέντος μεγάλη ποσότητα δεδομένων (μπορεί να εκτιμηθεί μετρώντας τις λειτουργίες I/O που απαιτεί ο αλγόριθμος)
- **Interpretability:** Επίπεδο κατανόησης και γνώση που παρέχεται από το μοντέλο. (Μπορεί να εκτιμηθεί μετρώντας πόσο πολύπλοκο είναι το μοντέλο π.χ. αριθμός κόμβων στα δέντρα απόφασης, αριθμός επιπέδων στα νευρωνικά δίκτυα κ.α.)

Τώρα ας επιστρέψουμε στο σημαντικότερο μέτρο μέτρησης απόδοσης, δηλαδή την ακρίβεια στην πρόβλεψη της κλάσης

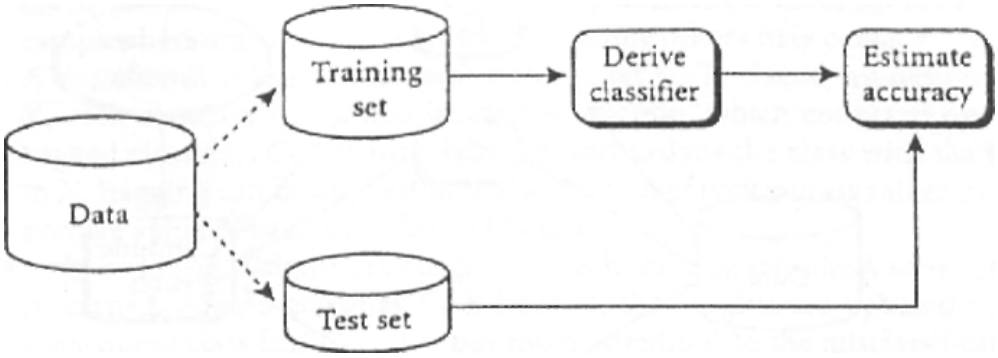
Εκτίμηση απόδοσης κατηγοριοποίησης

Holdout with random split

Με βάση την απόδοση του συστήματος στα διαθέσιμα δεδομένα προκύπτει ότι εάν ένα μοντέλο καλύπτει καλά ένα «ικανό» σύνολο δεδομένων τότε καλύπτει καλά και το σύνολο των δεδομένων λειτουργίας.

Ας εξετάσουμε τώρα το τι πρέπει να κάνουμε όταν το ποσό των δεδομένων για training και testing είναι περιορισμένο. Η μέθοδος **holdout** (μέθοδος κατακράτησης) κρατά ένα καθορισμένο ποσό των δεδομένων για testing και χρησιμοποιεί το υπόλοιπο για training. Στην πράξη συνηθίζεται να κρατάμε το 1/3 για testing και να χρησιμοποιούμε τα 2/3 για training.

Φυσικά, μπορεί να είμαστε άτυχοι και το δείγμα που χρησιμοποιούμε για training (ή testing) να μην είναι αντιπροσωπευτικό. Γενικά, δεν μπορούμε να καθορίσουμε αν το δείγμα είναι αντιπροσωπευτικό ή όχι. Αλλά υπάρχει ένας απλός έλεγχος που μπορεί να αποβεί χρήσιμος: κάθε μια από τις κλάσεις στο πλήρης σύνολο δεδομένων πρέπει να αναπαρίσταται με περίπου την σωστή αναλογία στα σύνολα training και testing.



Εκτίμηση ακρίβειας χρησιμοποιώντας την μέθοδο holdout

Πράγματι, πρέπει να εξασφαλίσουμε ότι η τυχαία δειγματοληψία πραγματοποιείται με τέτοιο τρόπο ώστε να εγγυάται ότι κάθε κλάση αναπαρίσταται κατάλληλα και στα σύνολα εκπαίδευσης και ελέγχου. Η διαδικασία αυτή καλείται stratification (στρωμάτωση), και μπορούμε να μιλήσουμε για stratified holdout. Το stratification παρέχει κάποιο βαθμό προστασίας από μια μη-ομαλή εκπροσώπηση κλάσεων στα σύνολα εκπαίδευσης και ελέγχου, αλλά μπορεί να μην επαρκεί.

Ένας γενικότερος τρόπος για την μετρίαση της πόλωσης (bias) που οφείλεται σε ένα συγκεκριμένο δείγμα που κρατείται για holdout, είναι η επανάληψη της συνολικής διαδικασίας, εκπαίδευσης και ελέγχου, αρκετές φορές για διαφορετικά τυχαία δείγματα. Σε κάθε επανάληψη μια συγκεκριμένη αναλογία - π.χ. 2/3 - των δεδομένων επιλέγεται τυχαία για εκπαίδευση, πιθανώς με stratification, και το υπόλοιπο

χρησιμοποιείται για έλεγχο. Ο μέσος όρος των ρυθμών λαθών σε διαφορετικές επαναλήψεις παρέχει μια καλύτερη εκτίμηση του συνολικού ρυθμού λαθών. Η μέθοδος αυτή της εκτίμησης του ρυθμού σφαλμάτων καλείται repeated holdout.

Holdout with stratified random split

Στην εργασία χρησιμοποιούμε ***stratification***,δηλαδή μέθοδο ***stratified random split*** που γίνεται για κάθε κλάση. Αν δηλαδή επιλέξουμε να χωρίσουμε το σύνολο δεδομένων εκπαίδευσης με ποσοστό 50 % σε train set κ test set θα το κάνει ξεχωριστά για κάθε κλάση. Αν η πρώτη κλάση έχει 1000 σημεία, τα 500 θα πάνε για train set και τα υπόλοιπα 500 για test set. Αν η δεύτερη κλάση έχει 80 σημεία τα 40 θα πάνε για train set και τα υπόλοιπα για test set.Δηλαδή επιλέγουμε ίσο ποσοστό από κάθε κλάση.

Cross-validation

Σε μια απλή διαδικασία holdout, μπορεί να θεωρήσουμε την ανταλλαγή των ρόλων των συνόλων εκπαίδευσης και ελέγχου - δηλαδή, την εκπαίδευση του συστήματος στα δεδομένα ελέγχου και τον έλεγχό του στα δεδομένα εκπαίδευσης - και την παραγωγή του μέσου όρου των δύο αποτελεσμάτων, με την συνακόλουθη μείωση του αποτελέσματος της ακανόνιστης κατανομής των κλάσεων στα σύνολα εκπαίδευσης και ελέγχου. Αυτό είναι προφανές μόνο για 50:50 διαχωρισμό μεταξύ δεδομένων εκπαίδευσης και ελέγχου, ο οποίος γενικά δεν είναι ιδεατός, γιατί είναι καλύτερη η χρησιμοποίηση της μεγαλύτερης αναλογίας δεδομένων για εκπαίδευση, ακόμη και σε βάρος των δεδομένων ελέγχου. Όμως, μια απλή τροποποίηση αποτελεί την βάση μιας σημαντικής στατιστικής τεχνικής που καλείται **cross-validation**. Στην cross validation αποφασίζουμε για ένα σταθερό αριθμό από folds, ή διαιρέσεις (partitions) των δεδομένων. Υποθέτουμε ότι χρησιμοποιούμε τρεις. Τότε τα δεδομένα θα διαχωριστούν σε τρεις προσεγγιστικά ίσες partitions, και κάθε μία στην συνέχεια θα χρησιμοποιηθεί για testing ενώ το υπόλοιπο για training. Δηλαδή, χρησιμοποιούμε τα 2/3 για εκπαίδευση και το 1/3 για έλεγχο, και επαναλαμβάνουμε την διαδικασία τρεις φορές έτσι ώστε στο τέλος κάθε instance (στιγμιότυπο) να έχει χρησιμοποιηθεί ακριβώς μια φορά για testing. Η τεχνική αυτή καλείται 3-fold cross-validation, και εάν συνδυάζεται με stratification (που είναι κοινή πρακτική), τότε αναφέρεται ως stratified 3-fold cross-validation.

Με την προσέγγιση της k-πλής σταυρωτής επικύρωσης ικανοποιείται το αίτημα της ανεξαρτησίας μεταξύ των παραδειγμάτων εκπαίδευσης και επικύρωσης και αμβλύνεται η διάσταση μεταξύ των τιμών των μέτρων αποτελεσματικότητας για διαφορετικά σύνολα επικύρωσης, καθώς η έξοδος της μεθόδους είναι ο μέσος όρος τους.

Ένας άλλος καθιερωμένος τρόπος για την πρόβλεψη του ρυθμού λαθών μιας τεχνικής μάθησης δεδομένου ενός απλού, σταθερού δείγματος δεδομένων είναι η χρησιμοποίηση stratified tenfold cross-validation. Τα δεδομένα διαιρούνται τυχαία σε δέκα τμήματα, σε κάθε ένα από τα οποία η κλάση αναπαρίσταται σε προσεγγιστικά ίδιες αναλογίες με ότι στο πλήρες σύνολο δεδομένων. Κάθε τμήμα (δηλ. 1/10) κρατείται για σύνολο ελέγχου (αναφέρεται ως holdout set) και το σχήμα μάθησης εκπαιδεύεται στο υπόλοιπα 9/10. έπειτα υπολογίζεται ο ρυθμός λαθών στο holdout set. Έτσι η διαδικασία μάθησης εκτελείται συνολικά δέκα φορές, σε διαφορετικά σύνολα εκπαίδευσης. Τελικά, ο συνολικός ρυθμός λαθών προκύπτει ως μέσος όρος των δέκα επιμέρους εκτιμήσεων.

Γιατί όμως διάσπαση σε δέκα τμήματα; Εκτενείς έλεγχοι σε πλήθος διαφορετικών συνόλων δεδομένων, με διαφορετικές τεχνικές μάθησης, έχουν δείξει ότι το δέκα είναι ο καταλληλότερος αριθμός των folds για να πάρουμε την καλύτερη εκτίμηση του λάθους, και υπάρχει και θεωρητική υποστήριξη για την επιλογή δέκα folds. Αν και αυτά τα επιχειρήματα δεν είναι πλήρως αποδεκτά από όλη την επιστημονική κοινότητα, η μέθοδος tenfold cross-validation έχει γίνει κοινή πρακτική. Έλεγχοι επίσης έχουν δείξει ότι το stratification βελτιώνει επιπλέον τα αποτελέσματα.

Το cross-validation μπορεί επίσης να χρησιμοποιηθεί για την επιλογή ενός υποσύνολου του πίνακα εκπαίδευσης έτσι ώστε η συνολική απόδοση του ταξινομητή να βελτιωθεί. Ένας μικρός αριθμός δειγμάτων που οδηγούν σε μεγάλη πιθανότητα λανθασμένης πρόβλεψης μπορεί να αφαιρεθεί από τον πίνακα εκπαίδευσης με n-fold cross validation ή με την μέθοδο leave-one-out cross validation.

Είδη Αλγορίθμων Κατηγοριοποίησης

Μπορούμε να διακρίνουμε πέντε είδη κατηγοριών αλγορίθμων κατηγοριοποίησης. Συγκεκριμένα υπάρχουν οι:

- ☞ Αλγόριθμοι κατηγοριοποίησης βασισμένοι στην απόσταση (KNN)
- ☞ Αλγόριθμοι κατηγοριοποίησης βασισμένοι στα δέντρα απόφασης
- ☞ Αλγόριθμοι κατηγοριοποίησης βασισμένοι στα Νευρωνικά Δίκτυα
- ☞ Αλγόριθμοι κατηγοριοποίησης βασισμένοι σε κανόνες
- ☞ Στατιστικοί αλγόριθμοι κατηγοριοποίησης.

Κεφάλαιο 3 KNN CLASSIFIER

Εισαγωγή

Ο Αλγόριθμος Κ κοντινότεροι γείτονες (K Nearest Neighbors - KNN) είναι μία πολύ γνωστή και ευρεία χρησιμοποιούμενη τεχνική κατηγοριοποίησης που στηρίζεται στη χρήση μέτρων βασισμένων στην απόσταση.

Η κεντρική ιδέα είναι πως η τιμή της συνάρτησης-στόχου για ένα νέο στιγμιότυπο βασίζεται αποκλειστικά και μόνο στις αντίστοιχες τιμές των k πιο «κοντινών» στιγμιότυπων εκπαίδευσης, τα οποία αποτελούν τους «γείτονες» του. Δύο ζητήματα πρέπει να αποφασιστούν προκειμένου να καθοριστεί πλήρως ο αλγόριθμος:

1. Ο ορισμός της απόστασης μεταξύ δύο στιγμιότυπων, δηλαδή μιας τιμής πάνω στο χώρο των στιγμιότυπων, που θα εκφράζει την εγγύτητα, ή αλλιώς την «ομοιότητα» μεταξύ των στιγμιότυπων.
2. Η τιμή του k.

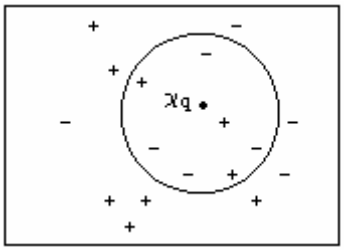
Για το πρώτο ζήτημα, υπάρχουν πολλές εναλλακτικές επιλογές. Η απόφαση εξαρτάται από τα ειδικά χαρακτηριστικά του χώρου στιγμιότυπων του προβλήματος. Ιδιαίτερη σημασία έχει αν στην αναπαράσταση των στιγμιότυπων περιλαμβάνονται αριθμητικά ή συμβολικά χαρακτηριστικά. Στον «παραδοσιακό» k-NN αλγόριθμο, στον οποίο τα στιγμιότυπα θεωρούνται πως ανήκουν στον n-διάστατο χώρο $\mathbb{R}^n$, μια μέτρηση που υιοθετείται συχνά είναι η γνωστή Ευκλείδεια απόσταση. Πιο συγκεκριμένα, αν τα στιγμιότυπα αναπαρίστανται ως διανύσματα από χαρακτηριστικά που παίρνουν τιμές πραγματικούς αριθμούς, δηλαδή το στιγμιότυπο x αναπαρίσταται από το διάνυσμα:

$$<a_1(x), a_2(x),\ldots,a_n(x)> ,$$

Όπου $a_r(x)$ δηλώνει την τιμή του r-οστού χαρακτηριστικού του x, τότε η απόσταση $d(x_i,x_j)$ μεταξύ δύο στιγμιότυπων x_i και x_j ορίζεται ως:

$$d(x_i,x_j)=\sqrt{\sum_{r=1}^{n}(a_r(x_i)-a_r(x_j))^2}$$

Φυσικά, είναι δυνατόν να επιλεχθούν και άλλα μέτρα ομοιότητας-ανομοιότητας, αντί της Ευκλείδειας.



Στην εικόνα φαίνεται η λειτουργία του k-NN. Τα “+” και τα “-“ δείχνουν τα στιγμιότυπτα εκπαίδευσης της κάθε κλάσης και το x_q ένα στιγμιότυπο προς κατάταξη. Φαίνεται πως ο 1-NN κατατάσσει το x_q ως “+” ενώ ο 7-NN το κατατάσσει ως “-“.

Ένα μειονέκτημα που παρουσιάζεται στο προηγούμενο παράδειγμα είναι πως όλα τα χαρακτηριστικά θεωρούνται ισοδύναμα κατά τον υπολογισμό της απόστασης. Αυτό είναι ιδιαίτερα προβληματικό αν δεν είναι όλα τα χαρακτηριστικά σχετικά με τη συγκεκριμένη συνάρτηση-στόχο που επιδιώκεται να προσεγγιστεί, αλλά και γενικότερα, οποτεδήποτε υπάρχουν σημαντικές διαφορές μεταξύ των χαρακτηριστικών ως προς την αξία τους στον προσδιορισμό της συνάρτησης. Σε μία τέτοια περίπτωση, οι παραπάνω μετρήσεις είναι παραπλανητικές, από την άποψη πως μερικά στιγμιότυπα που σχετίζονται πραγματικά μεταξύ τους, είναι δυνατόν να θεωρούνται απομακρυσμένα λόγω των διαφορών τους σε άσχετα ή ασήμαντα χαρακτηριστικά.

Μια λύση σε αυτό το πρόβλημα είναι κάθε χαρακτηριστικό να αποτιμάται διαφορετικά στον υπολογισμό της απόστασης, ανάλογα με την αξία του. Η μέθοδος αυτή λέγεται αποτίμηση των χαρακτηριστικών (feature weighting). Με βάση αυτήν, ο τύπος της Ευκλείδειας θα μπορούσε να γίνει:

$$d(x_i,x_j)=\sqrt{\sum_{r=1}^{n} w_r(a_r(x_i)-a_r(x_j))^2}$$

όπου w_r είναι το βάρος τους χαρακτηριστικού α_r.

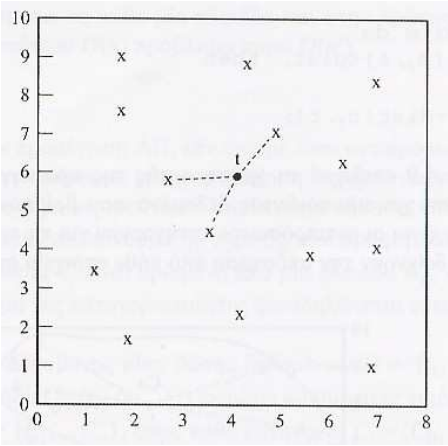
Ο αλγόριθμος KNN

Στα παρακάτω σχήματα παρουσιάζεται η διαδικασία που χρησιμοποιείται από τον αλγόριθμο KNN. Στο πρώτο σχήμα φαίνονται τα σημεία του συνόλου εκπαίδευσης. Παρουσιάζονται τα τρία κοντινότερα στοιχεία στο σύνολο εκπαίδευσης. Το t θα τοποθετηθεί στην κατηγορία στην οποία ανήκουν τα περισσότερα από αυτά τα Ê στοιχεία.

Ο Αλγόριθμος του παρακάτω σχήματος περιγράφει τη χρήση του KNN.

Πρέπει να υπογραμμιστεί ότι η τεχνική KNN είναι υπερβολικά ευαίσθητη στην τιμή του Ê, δηλαδή πόσοι κοντινότεροι γείτονες χρησιμοποιούνται για την κατηγοριοποίηση. Σύμφωνα με μια εμπειρική μέθοδο πρέπει να ισχύει ότι

$$\hat{E}=\leq\sqrt{\text{αριθμός\_στοιχείων\_εκπαίδευσης}}$$



Σχήμα. Κατηγοριοποίηση με χρήση KNN

Είσοδος: T // Σύνολο δεδομένων εκπαίδευσης
K // Αριθμός κοντινότερων γειτόνων
t // πλειάδα προς κατηγοριοποίηση
Έξοδος: c // Κλάση όπου θα κατηγοριοποιηθεί η t
Αλγόριθμος _ K _ Κοντινότερων _ Γειτόνων
$N = \emptyset$
Για κάθε $d \in T$ επανέλαβε
Αν $|N| \leq K$ τότε
$N = N \cup \{d\}$;
Αλλιώς
Αν $\exists$ u $\in N$ τέτοιο ώστε dist(t,u) $\leq$ dist (t, d), τότε
$N = N - \{u\}$;
$N = N \cup \{d\}$;
Τέλος _ αν
Τέλος _ επανάληψης
c = κλάση όπου τα περισσότερα $u \in N$ κατηγοριοποιούνται
Τέλος αλγορίθμου

Σχήμα __ Αλγόριθμος KNN

Παραδείγματα KNN

Στο παρακάτω παράδειγμα βλέπουμε πέντε στοιχεία (πελάτες) που αποτελούνται από τα εξής χαρακτηριστικά: Ηλικία, Εισόδημα, Αριθμός πιστωτικών καρτών και απάντηση.

Ψάχνουμε να βρούμε ποια θα είναι η απάντηση του έκτου πελάτη David.

Καταρχήν βρίσκουμε τις (ευκλείδειες) αποστάσεις των πελατών και του David.

Κατόπιν βρίσκουμε του τρεις κοντινότερους γείτονες (John, Rachel, Nellie).

Βλέπουμε ότι η πλειοψηφία των απαντήσεων των κοντινότερων γειτόνων είναι ΝΑΙ. Άρα η απάντηση του David θα είναι ΝΑΙ.

Παράδειγμα 3 NN

Πελάτης	Ηλικία	Εισόδημα	Αριθ. Πιστωτικών Καρτών	Απάντηση
John	35	35K	3	NAI
Rachel	22	50K	2	OXI
Hannah	63	200K	1	OXI
Tom	59	170K	1	OXI
Nellie	25	40K	4	NAI
David	**37**	**50K**	**2**	**?**

Πελάτης	Ηλικία	Εισόδημα	Αριθ. Πιστωτικών Καρτών	Απάντηση	Απόσταση από τον David
John	35	35K	3	NAI	sqrt [(35-37)2+(35-50)2 +(3-2)2]=15.16
Rachel	22	50K	2	OXI	sqrt [(22-37)2+(50-50)2 +(2-2)2]=15
Hannah	**63**	**200K**	**1**	**OXI**	**sqrt [(63-37)2+(200-50)2 +(1-2)2]=152.23**
Tom	**59**	**170K**	**1**	**OXI**	**sqrt [(59-37)2+(170-50)2 +(1-2)2]=122**
Nellie	25	40K	4	NAI	sqrt [(25-37)2+(40-50)2 +(4-2)2]=15.74
David	37	50K	2	**NAI**	

Στο παραπάνω παράδειγμα αν αναζητούσαμε μόνο τον πρώτο κοντινότερο γείτονα θα παίρναμε τελείως διαφορετικό αποτέλεσμα. Αυτό θα συμβεί γιατί ο κοντινότερος γείτονας του David είναι η Rachel που έχει απαντήσει OXI. Έτσι η πρόβλεψη της απάντησης του David θα είναι OXI. Αυτό το βλέπουμε στους παρακάτω πίνακες:

Σύγκριση 1NN με τον kNN

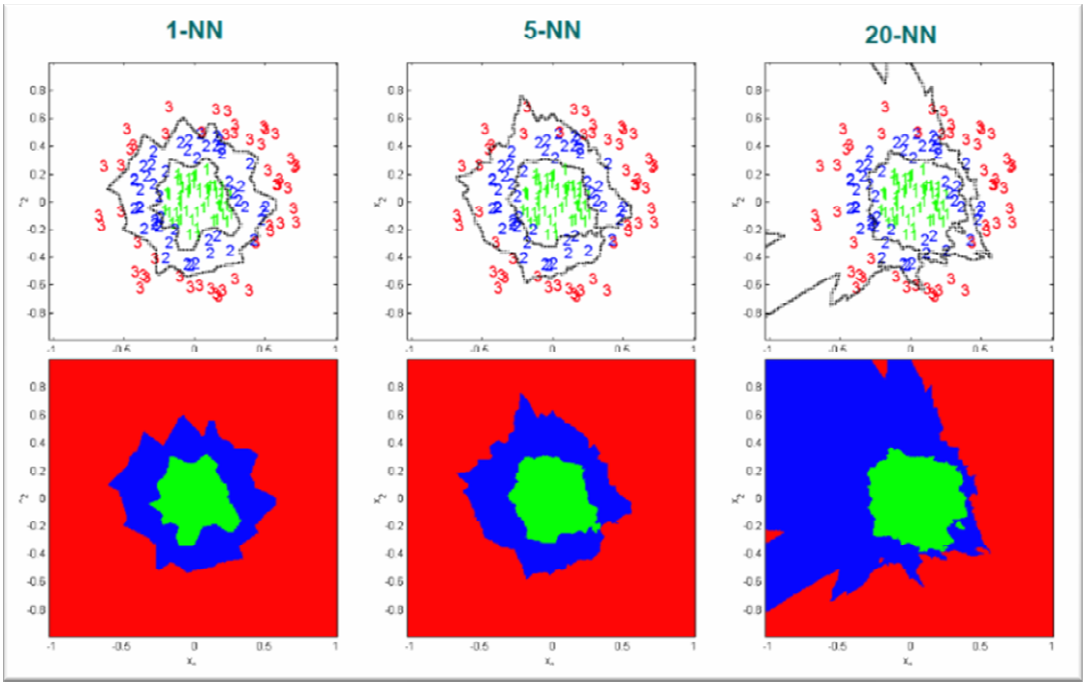
Χρησιμοποιώντας μεγαλύτερο αριθμό k γειτόνων παίρνουμε ομαλότερες περιοχές απόφασης.

Παρόλα αυτά αν θέσουμε πολύ μεγάλο αριθμό k γειτόνων παρατηρούμε τις εξής ανεπιθύμητες συμπεριφορές.

Καταστρέφεται η σωστή εκτίμηση των δεδομένων καθώς λαμβάνονται υπόψη παραδείγματα που έχουν μεγάλη απόσταση.

Αυξάνεται το υπολογιστικό κόστος.

Χρησιμοποιώντας το προηγούμενο παράδειγμα και αλλάζοντας διαδοχικά τον αριθμό k σε 1,5 και 20 παίρνουμε τα επόμενα αποτελέσματα.



Για να επιταχύνουν τη διαδικασία της εύρεσης του κοντινότερου γείτονα έχουν προταθεί διάφορες προσεγγίσεις. Οι Fukunaka και Nerendra έφτιαξαν έναν τέτοιο αλγόριθμο. Η ιδέα ήταν να διαιρέσουν το χώρο σε μικρότερες περιοχές και κατόπιν να εξερευνούν μια περιοχή μόνο αν υπάρχουν πιθανότητες να βρουν εκεί κάποιο κοντινότερο γείτονα. Οι περιοχές αυτές χωρίζονταν ιεραρχικά σε υποσύνολα, μετά σε υπο-υποσύνολα και ούτω καθεξής.

Μερικές, ακόμα, μέθοδοι που χρησιμοποιήθηκαν στο παρελθόν και είχαν τον ίδιο σκοπό της επιτάχυνσης της διαδικασίας, ήταν οι condensed-nearest-neghbour του Hart, reduced-nearest-neighbour του Gates και ο edited-nearest-neighboure των Hand και Batchelor. Όλες αυτές οι μέθοδοι προσπαθούσαν να μειώσουν το σύνολο των δεδομένων, κρατώντας μόνο αυτά που θα μπορούσαν να δώσουν έναν κοντινότερο γείτονα. Έτσι έχοντας λιγότερα δεδομένα η ταχύτητα ήταν μεγαλύτερη.

Η επιλογή του αριθμού k μπορεί να γίνει με την μέθοδο της διασταυρωμένης επικύρωσης. Αυτή η μέθοδος, όμως, μπορεί να είναι απαγορευτική σε μεγάλα σύνολα δεδομένων εξαιτίας των πολλών πράξεων που απαιτούνται. Οι πολλές πράξεις έχουν και αποτέλεσμα τους τον μεγάλο χρόνο. Παρατηρείται ότι για k μεγαλύτερο από ένα (k>1) η διάρκεια εκτέλεσης της μεθόδου δεν είναι ικανοποιητική αφού κάθε δεδομένο πρέπει να αποθηκευθεί και μετά να εξεταστεί για την ταξινόμησή του.

Επιδόσεις του k-NN σε σύνολα δεδομένων

Σύνολο δεδομένων: Χειρόγραφα ψηφία (Dig44)

Αυτό το σύνολο δεδομένων αποτελείται από 18.000 παραδείγματα των ψηφίων 0 έως 9 τα οποία συλλέχθηκαν από ταχυδρομικούς κωδικούς σε επιστολές στη Γερμανία. Τα χειρόγραφα παραδείγματα ψηφιοποιήθηκαν σε εικόνες των 16x16 pixels και 256 grey levels. Μετά διαβάστηκαν από αυτόματους αναγνώστες διευθύνσεων που φτιάχτηκε από μία γερμανική επιχείρηση. Το μέγεθός τους άλλαξε αναλογικά. Έτσι δεν λέπτυναν ούτε περιστράφηκαν. Ένα παράδειγμα των ψηφίων φαίνεται στην παρακάτω εικόνα

Το dataset χωρίστηκε σε 900 παραδείγματα για κάθε ψηφίο. Λόγω των μεγάλων απαιτήσεων λίγοι αλγόριθμοι κατάφεραν να ανταποκριθούν. Έτσι για να εξαχθούν συγκρίσιμα αποτελέσματα των αλγορίθμων, χρησιμοποιήθηκαν 16 ιδιότητες {οι οποίες προετοιμάστηκαν με τον μέσο όρο των 4x4 γειτονιών των κανονικών εικόνων BOOK Machine learning 143}. Όπως θα δούμε στον παρακάτω πίνακα ο k-NN βγήκε πρώτος σε μία σειρά αλγορίθμων. Το γεγονός αυτό μπορεί να εξηγείται από τις λίγες υποθέσεις που κάνει ο αλγόριθμος για τα δεδομένα.

Αποτελέσματα για 4x4 digit dataset (10 classes, 16 attributes, (train, test)=(9000,9000) observations).

Algorithm	Max. Storage	Time (sec.) Train	Time (sec.) Test	Error Rate Train	Error Rate Test	Rank
Discrim	252	65.3	30.2	0.111	0.114	12
Quadisc	324	194.4	152.0	0.052	0.054	2
Logdisc	1369	5110.2	138.2	0.079	0.086	10
SMART	337	19490.6	33.0	0.096	0.104	11
ALLOC80	393	1624.0	7041.0	0.066	0.068	5
k-NN	497	2230.7	2039.2	0.016	0.047	1
CASTLE	116	252.6	4096.8	0.180	0.170	20
CART	240	251.6	40.8	0.180	0.160	19
IndCART	884	3614.5	50.6	0.011	0.154	17
NewID	532	500.7	112.5	0.080	0.150	16
AC^2	770	10596.0	22415.0	*	0.155	18
Baytree	186	1117.0	59.8	0.015	0.140	14
NaiveBay	129	42.7	61.8	0.220	0.233	23
CN2	1926	3325.9	119.9	0.000	0.134	13
$C4.5^2$	248	778.1	60.6	0.041	0.149	15
ITrule	504	1800.1	9000	*	0.222	22
Cal5	1159	571.0	55.2	0.118	0.220	21
Kohonen	646	67176.0	2075.1	0.051	0.075	7
DIPOL92	110	191.2	43.6	0.065	0.072	6
Backprop	884	28910.0	110.0	0.072	0.080	8
RBF	268	1400.0	250.0	0.080	0.083	9
LVQ	249	1342.6	123.0	0.040	0.061	3
Cascade	2442	19171.0	1.0	0.064	0.065	4
Default	*	*	*	0.900	0.900	24

Σύνολο δεδομένων: Cut

Αυτό το σύνολο δεδομένων συλλέχθηκε από έναν συνεργάτη της StatLog. Το σύνολο των δεδομένων κατασκευάστηκε κατά τη διάρκεια μιας έρευνας σχετικά με το πρόβλημα της απομόνωσης των χαρακτήρων από ένα ενιαίο κείμενο. Το επόμενο σχήμα εμφανίζει ένα παράδειγμα της λέξης Eins (ένα στα Γερμανικά). Κάθε παράδειγμα αποτελείται από διάφορες μετρήσεις που γίνονται στο κείμενο σχετικά με ένα πιθανό σημείο αποκοπής μαζί με μια απόφαση για το αν θα κοπεί το κείμενο σε αυτό το σημείο ή όχι. Όπως παρέχεται το δείγμα περιλαμβάνει παραδείγματα 50 πραγματικών ιδιοτήτων (cut 50). Σε μια προσπάθεια να αξιολογηθεί η απόδοση των αλγορίθμων σχετικά με τη διάσταση του προβλήματος, δημιουργήθηκε ένα νέο σύνολο δεδομένων, όπου χρησιμοποιήθηκαν οι 20 “καλύτερες” ιδιότητες (cut 20).

Αν και τα μεμονωμένα αποτελέσματα διαφέρουν μεταξύ τους οι βαθμολογίες των μεθόδων είναι σχεδόν οι ίδιες.

Είναι ενδιαφέρον ότι σχεδόν όλοι αλγόριθμοι επιτυγχάνουν καλύτερο αποτέλεσμα για το Cut50 από το Cut20.

Αποτελέσματα για Cut20 dataset (2 classes, 20 attributes, (train, test)=(11.220, 7480) observations).

Algorithm	Max. Storage	Time (sec.)		Error Rate		Rank
		Train	Test	Train	Test	
Discrim	71	115.5	22.7	0.052	0.050	15
Quadisc	75	394.8	214.2	0.090	0.088	22
Logdisc	1547	587.0	101.2	0.046	0.046	13
SMART	743	21100.5	21.8	0.047	0.047	14
ALLOC80	302	32552.2	*	0.033	0.037	4
k-NN	190	54810.7	6052.0	0.031	0.036	2
CASTLE	175	1006.0	368.5	0.060	0.061	17
CART	FD	FD	FD	FD	FD	
IndCART	1884	*	*	0.002	0.040	6
NewID	1166	1445.0	3.0	0.000	0.039	5
AC^2	915	917.0	48.0	0.000	0.063	19
Baytree	1676	145.3	25.9	0.002	0.034	1
NaiveBay	1352	83.6	27.6	0.074	0.077	20
CN2	9740	5390.0	470.0	0.000	0.042	8
C4.5	2436	293.0	28.0	0.010	0.036	2
ITrule	630	11011.0	50.9	0.083	0.082	21
Cal5	188	455.5	23.4	0.043	0.045	11
Kohonen	1046	*	*	0.046	0.050	15
DIPOL92	379	506.0	36.1	0.043	0.045	11
Backprop	144	88532.0	7.0	0.037	0.043	9
RBF	901	6041.0	400.0	0.042	0.044	10
LVQ	291	1379.0	86.9	0.029	0.041	7
Default	*	*	*	0.059	0.061	17

ΑΝΑΣΚΟΠΗΣΗ ΤΕΧΝΙΚΩΝ KNN

Μία πολύ καλή ανασκόπηση τεχνικών KNN βρήκαμε στην αναφορά [Bhatia and Vandana 2010]. Ο κανόνας του κοντινότερου γείτονα(NN) καθορίζει την κατηγορία ενός άγνωστου σημείου δεδομένων σε σχέση με τον κοντινότερο του γείτονα του οποίου η κλάση είναι ήδη γνωστή. Έτσι το άγνωστο δεδομένο ταξινομείται ανάλογα με την κατηγορία στην οποία ανήκει ο γείτονας του. Η nearest neighbor (NN) technique είναι πολύ απλή, αποδοτική και αποτελεσματική στο πεδίο της εξόρυξης γνώσης, αναγνώρισης προτύπων, κατηγοριοποίησης κειμένου, αναγνώρισης αντικειμένων και συμβάντων. Η απλότητα της είναι το κύριο της πλεονέκτημα, αλλά παρουσιάζει και σημαντικά μεονεκτήματα.

Η απαίτηση σε μνήμη και η υπολογιστική πολυπλοκότητα αποτελούν περιοριστικούς παράγοντες για τον knn.Πολλές τεχνικές έχουν αναπτυχθεί για να ξεπεράσουν αυτούς τους περιορισμούς. Αυτές οι NN τεχνικές κατηγοριοποιούνται σε δομημένες και μη τεχνικές. Οι δομημένες τεχνικές χρησιμοποιούν μια δομή δεδομένων, συνήθως δενδροειδή, για να λειτουργήσουν. Οι μη δομημένες από την άλλη μεριά λειτουργούν δίχως μια συγκεκριμένη δομή δεδομένων.

Οι Weighted kNN, Model based kNN, Condensed NN, Reduced NN, Generalized NN είναι μη δομημένες τεχνικές ενώ k-d tree, ball tree, Principal Axis Tree, Nearest Feature Line, Tunable NN, Orthogonal Search Tree είναι δομημένοι αλγόριθμοι ανεπτυγμένοι πάνω στη βάση του KNN. Η μη δομημένη μέθοδος περιορίζει τις απαιτήσεις σε μνήμη, καθώς χρειάζονται λιγότερα σημεία δεδομένων, και οι δομημένες τεχνικές μειώνουν την υπολογιστική πολυπλοκότητα, βρίσκοντας τους κοντινότερους γείτονες πιο γρήγορα

Οι T. M. Cover και P. E. Hart σχεδίασαν τον αλγόριθμο του k κοντινότερου γείτονα, στον οποίο ο κοντινότερος γείτονας υπολογίζεται πάνω στη βάση μιας τιμής k, η οποία προσδιορίζει πόσοι κοντινότεροι γείτονες θα ληφθούν υπ' όψιν για να καθοριστεί η κλάση ενός σημείου [1].

Οι T. Bailey και A. K. Jain οι βελτίωσαν τον KNN προσθέτοντας βάρη [2]. Στα σημεία εκπαίδευσης αποδίδονται βάρη σύμφωνα με τις απόστασεις τους από το σημείο δεδομένο που έχει ληφθεί ως δείγμα .Αλλά και πάλι , η υπολογιστική πολυπλοκότητα και οι απαίτησεις σε μνήμη εξακολουθούν να παραμένουν περιοριστικοί παράγοντες.

Γία να ξεπεράσουμε τους περιορισμούς σε μνήμη, το μέγεθος της λίστας δεδομένων μειώνεται. Για αυτό το σκοπό τα επαναλαμβανόμενα πρότυπα, που δεν προσθέτουν καινούρια πληροφορία, αποκλείονται από τα δείγματα εκπαίδευσης.[3-5]. Για μεγαλύτερη βελτίωση τα σημεία δεδομένων που δεν επηρεάζουν το αποτέλεσμα, αποκλείονται επίσης από την σειρά δεδομένων εκπαίδευσης[6].

Εκτός του περιορισμού σε χρόνο και μνήμη , πρέπει να ληφθεί υπ' όψιν η τιμή k πάνω στην βάση της οποίας θα καθοριστεί η κατηγορία του άγνωστου δείγματος.

Ο Gongde Guo επιλέγει το k χρισιμοποιώντας προσέγγιση βασισμένη σε μοντέλο [7]. Το μοντέλο που προτείνεται, αυτόματα επιλέγει την τιμή του k. Παρομοίως, πολλές βελτίωσεις προτείνονται για τη βελτίωση της ταχύτητας του κλασικού kNN χρησιμοποιώντας την έννοια της κατάταξης[8], της λανθασμένης πληροφορίας γείτονα [9], της ομαδοποίησης [10].

Α. ΜΗ ΔΟΜΗΜΕΝΕΣ NN ΤΕΧΝΙΚΕΣ

Στην πρώτη κατηγορία ανήκει η τεχνική k κοντινού γείτονα , στην οποία όλα τα δεδομένα ταξινομούνται σε δεδομένα εκπαίδευσης και δειγματοληπτικά σημεία δεδομένων. Έτσι υπολογίζεται η απόσταση κάθε σημείου δείγματος από όλα τα σημεία εκπαίδευσης και το σημείο με την μικρότερη απόσταση καλείτε κοντινότερος γείτονας.

Η τεχνική αυτή είναι εύκολα υλοποιήσημη αλλά, σε κάποιες περιπτώσεις, η τιμή του k επηρεάζει το αποτέλεσμα. Ο Bailey χρησιμοποιεί βάρη μαζί με τον κλασικό kNN και δημιουργεί τον αλγόριθμο weighted kNN (WkNN) [2]. Ο ***weighted kNN*** αναθέτει σε κάθε γείτονα ένα βάρος της τάξης του 1/d, όπου d η απόσταση του σημείου από τον γείτονα, και με βάση αυτό επιλέγεται ο κοντινότερος γείτονας καθώς και η κλάση του δείγματος. Στην εργασία χρησιμοποιούμε το βάρος exp(– d) που λειτουργεί καλά για όλες τις κατανομές αποστάσεων.

Ο Condensed Nearest Neighbor (CNN), αποθηκεύει τα πρότυπα ένα προς ένα και αποκλείει τα διπλά (διπλοεγγραφές). Στόχος του είναι να ελαττώσει το πλήθος των δεδομένων εκπαίδευσης .Ως εκ τούτου ο CNN αποκλείει τα σημεία δεδομένων που δεν προσθέτουν άλλη πληροφορία και παρουσιάζουν ομοιότητα με άλλη σειρά εκπαίδευσης δεδομένων.Κρατά μόνα τα δείγματα που απαιτούνται για τον διαχωρισμό των κλάσεων.

Ο Reduced Nearest Neighbor (RNN), είναι βελτίωση του Condensed Nearest Neighbor. Περιλαμβάνει ένα ακόμα βήμα που είναι ο αποκλεισμός των προτύπων που δεν επηρεάζουν το αποτέλεσμα της σειράς εκπαίδευσης δεδομένων. Δηλαδή από τη λίστα δεδομένων αφαιρούμε ένα πρότυπο. Αν τα πρότυπα μετά την αφαίρεση ταξινομούνται σωστά συνεχίζουμε, αλλιώς επιστρέφουμε το πρότυπο στη λίστα. Μία άλλη τεχνική η βασισμένη σε μοντέλο kNN, επιλέγει μέτρα ομοιότητας και δημιουργεί μια μήτρα (πίνακα ομοιότητας) από δοσμένη λίστα εκπαίδευσης.

Μετά στην ίδια κατηγορία , βρίσκεται ο μεγαλύτερος τοπικός γείτονας που καλύπτει μεγάλο αριθμό γειτόνων και μία πλειάδα τοποθετείται στην μεγαλύτερη γενική γειτονία. Αυτά τα βήματα επαναλαμβάνονται μέχρι όλες οι πλειάδες να ομαδοποιηθούν. Μόλις τα δεδομένα διαμορφωθούν χρησιμοποιώντας το μοντέλο, ο kNN εκτελείται για να καθορίσει την κατηγορία του άγνωστου δείγματος

Οι Subash C.Bagui και Sikha Bagui [8] βελτίωσαν τον kNN εισάγοντας την έννοια των τάξεων (ranks). Η μέθοδος αυτή συγκεντρώνει όλες της παρατηρήσεις διαφόρων κατηγοριών και θέτει βαθμό (τάξη) σε κάθε δείγμα σε αύξουσα σειρά. Έπειτα οι παρατηρήσεις μετρώνται και πάνω στη βάση των βαθμών επιλέγεται η κλάση ενός τυχαίου δείγματος.

Είναι πολύ χρήσιμος στην περίπτωση δεδομένων με πολλές ιδιότητες. Στον Modified kNN, που είναι και μια μετατροπή του WkNN η εγκυρότητα όλων των δειγμάτων στη σειρά εκπαίδευσης υπολογίζεται , ανάλογα ανατίθενται βάρη και έπειτα εγκυρότητα και πλήθος μαζί αποτελούν τη βάση κατηγοριοποίησης της κλάσης κάθε δείγματος σημείου.

Οι Yong zeng, Yupu Zeng και Liang Zhou ορίζουν μια νέα έννοια κατηγοριοποιήσηςς δεδομένων δείγματος. Η μέθοδος εισάγει τον ψευδο-γείτονα , που δεν είναι ο πραγματικός κοντινότερος γείτονας. Ουσιαστικά ένας νέος κοντινότερος γείτονας επιλέγεται στη βάση της τιμής ενός σταθμισμένου συνόλου “μέσου όρου” των αποστάσεων του kNN από μη κατηγοριοποιημένα πρότυπα σε κάθε κλάση.

Μετά υπολογίζεσαι η ευκλείδεια απόσταση και ο ψευδο-γείτονας με το μεγαλύτερο βάρος βρίσκεται και κατηγοριοποιείται για άγνωστο δείγμα. Στην τεχνική της Zhou Yong [11], η ομαδοποίηση χρησιμοποιείται για να υπολογιστεί ο κοντινότερος γείτονας.

Τα βήματα περιλαμβάνουν, πρώτα απ’ όλα αφαίρεση των δειγμάτων που βρίσκονται κοντα στα όρια της σειράς εκπαίδευσης. Ομαδοποιείται κάθε σειρά εκπαίδευσης σύμφωνα με μια τιμή k και όλα τα κέντρα των ομάδων δημιουργούν ένα νέο σύνολο εκπαίδευσης. Ανατίθενται βάρη σε κάθε ομάδα σύμφωνα με τον αριθμό των δειγμάτων εκπαίδευσης που κάθε ομάδα έχει.

Β. ΔΟΜΗΜΕΝΕΣ ΤΕΧΝΙΚΕΣ ΝΝ

Η δεύτερη κατηγορία τεχνικών κοντινότερου γείτονα βασίζεται πάνω σε δομές δεδομένων όπως τα Ball Tree, k-d Tree, principal axis Tree (PAT), orthogonal structure Tree (OST), Nearest feature line (NFL), Center Line (CL) κ.τ.λ. Ο Ting Liu εισάγει την έννοια του Ball Tree.

Το ball tree είναι ένα δυαδικό δέντρο και κατασκευάζεται χρησιμοποιώντας την από πάνω προς τα κάτω προσέγγιση . Αυτή η τεχνική είναι βέλτιστη του kNN όσον αφορά την ταχύτητα. Τα φύλλα του δέντρου περιέχουν σχετική πληροφορία και οι εσωτερικοί κόμβοι χρησιμοποιούνται για την αποτελεσματική αναζήτηση ανάμεσα στα φύλλα. Τα δέντρα k-διαστάσεων χωρίζουν τα δεδομένα εκπαίδευσης σε δύο μέρη, τον δεξιό και τον αριστερό κόμβο. Η αριστερή ή η δεξιά μεριά του δέντρου θα εξεταστεί ανάλογα με αρχεία με ερωτήματα Αφού φτάσουμε στον αρχικό κόμβο, τα αρχεία του τελικού κόμβου εξετάζονται για να βρεθεί ο κοντινότερος κόμβος ως προς το αρχικό ερώτημα

Η έννοια του NFL που δόθηκε απο τον Stan Z.Li και τονChan K.L. [24] διαχωρίζει τα δεδομένα εκπαίδευσης σε ένα επίπεδο. Μια γραμμή χαρακτηριστικών (feature

line)χρησιμοποιείται για να βρεθεί ο κοντινότερος γείτονας.Αυτή είναι η ευθεία γραμμή που περνά από δύο δείγματα εκπαίδευσης της ίδιας κλάσης. Η αποσταση FL ανάμεσα στο σημείο ερωτήμα και σε κάθε ζευγάρι της γραμμής χαρακτηριστικών, δηλαδή η απόσταση του σημείου ερωτήματος και της προβολής του πάνω στην FL, υπολογίζεται για κάθε κλάση,για την αναζήτηση κοντινότερων γειτόνων.

Το αποτέλεσμα είναι μία σειρά αποστάσεων. Οι υπολογισμένες αποστάσεις ταξινομούνται με αύξουσα σειρά και η απόσταση NFL μπαίνει στην πρώτη θέση. Μία βελτίωση του NFL είναι Local nearest neighbour που υπολογίζει τη γραμμή χαρακτηριστικών ή το σημείο χαρακτηριστικού κάθε κλάσης, για τα σημεία εκείνα μόνο , που τα πρωτότυπα τους είναι γείτονες του σημείου ερωτήματος.

Οι Yongli Zhou και Changshui Zhang παρουσίασαν μία νέα μετρική μέθοδο που υπολογίζει τις αποστάσεις για τον NFL, και όχι την feature line. Αυτή καλείται συντονισμένη ή προσαρμοσμένη μετρική μέθοδος. Ακολουθεί την ίδια διαδικασία με την NFL αλλά στο πρώτο στάδιο χρησιμοποιεί αυτή τη μετρική μέθοδο για να υπολογίσει την απόσταση και μετά υλοποιεί τα βήματα του NFL.

Ο κεντρικά βασιζόμενος κοντινότερος γείτονας είναι βελτιώση του NFL και του προσαρμοσμένου κοντινότερου γείτονα. Χρησιμοποιεί την center based line(CL),τη γραμμή δηλαδή που συνδέει ένα σημείο δείγμα με γνωστά σημεία με ετικέτα που έχουν ταξινομηθεί. Πρώτα απ' ολα υπολογίζεται η CL, η οποία είναι μία ευθεία γραμμή που συνδέει ένα σημείο δείγμα και το κέντρο μιας γνωστής κλάσης,αντί για δύο σημεία μιας κλάσης όπως συμβαίνει στην NFL. Μετά υπολογίζεται η απόσταση από το σημείο ερώτημα και τη CL, και βρίσκεται ο κοντινότερος γείτονας.

Μια άλλη μέθοδος είναι η PAT. Αυτή επιτρέπει τη διαίρεση των δεδομένων εκπαίδευσης με αποδοτικό τρόπο ,όσον αφορά την ταχύτητα ,για αξιολόγηση κοντινότερου γείτονα. Αποτελείται από δύο φάσεις 1) κατασκευή του PAT 2) αναζήτηση κοντινότερων γειτόνων.

Η PAT χρησιμοποιεί την μέθοδο (PCA) και χωρίζει τα δεδομένα σε περιοχές που περιέχουν τον ίδιο αριθμό σημείων. Όταν δημιουργηθεί το δέντρο ο kNN χρησιμοποιείται για την αναζήτηση του κοντινότερου γείτονα στο PAT. Οι περιοχές μπορούν να καθοριστούν για δοσμένο σημείο χρησιμοποιώντας δυαδική αναζήτηση .

Βελτίωση του PAT αποτελεί η μέθοδος OST, που χρησιμοποιεί ορθογώνιο άνυσμα. Επιτυγχάνει επιτάχυνση της διαδικασίας αναζήτησης κοντινότερου γείτονα. Χρησιμοποιεί την έννοια του κανόνα του μήκους, που υπολογίζεται στο πρώτο στάδιο. Μετά δημιουργείται το δέντρο ορθογώνιας αναζήτησης, δημιουργώντας έναν κόμβο ρίζα και τοποθετώντας όλα τα σημεία σ' αυτόν τον κόμβο. Μετά οι αριστεροί και οι δεξιοί κόμβοι δημιουργούνται χρησιμοποιώντας την pop διαδικασία .

TABLE I. COMPARISON OF NEAREST NEIGHBOR TECHNIQUES

Sr No	Technique	Key Idea	Advantages	Disadvantages	Target Data
1.	k Nearest Neighbor (kNN) [1]	Uses nearest neighbor rule	1. training is very fast 2. Simple and easy to learn 3. Robust to noisy training data 4.Effective if training data is large	1. Biased by value of k 2.Computation Complexity 3.Memory limitation 4. Being a supervised learning lazy algorithm i.e. runs slowly 5. Easily fooled by irrelevant attributes	large data samples
2.	Weighted k nearest neighbor (WkNN) [2]	Assign weights to neighbors as per distance calculated	1. Overcomes limitations of kNN of assigning equal weight to k neighbors implicitly. 2. Use all training samples not just k. 3. Makes the algorithm global one	1. Computation complexity increases in calculating weights 2. Algorithm runs slow	Large sample data
3.	Condensed nearest neighbor (CNN) [3,4,5]	Eliminate data sets which show similarity and do not add extra information	1. Reduce size of training data 2. Improve query time and memory requirements 3.Reduce the recognition rate	1. CNN is order dependent; it is unlikely to pick up points on boundary. 2. Computation Complexity	Data set where memory requirement is main concern
4.	Reduced Nearest Neigh (RNN) [6]	Remove patterns which do not affect the training data set results	1. Reduce size of training data and eliminate templates 2. Improve query time and memory requirements 3.Reduce the recognition rate	1.Computational Complexity 2.Cost is high 3.Time Consuming	Large data set
5.	Model based k nearest neighbor (MkNN) [7]	Model is constructed from data and classify new data using model	1. More classification accuracy 2.Value of k is selected automatically 3.High efficiency as reduce number of data points	1.Do not consider marginal data outside the region	Dynamic web mining for large repository
6.	Rank nearest neighbor (kRNN) [8]	Assign ranks to training data for each category	1.Performs better when there are too much variations between features 2.Robust as based on rank	1.Multivariate kRNN depends on distribution of the data	Class distribution of Gaussian nature

Vol. 8, No. 2, 2010

			3.Less computation complexity as compare to kNN		
7.	Modified k nearest neighbor (MkNN) [10]	Uses weights and validity of data point to classify nearest neighbor	1.Partially overcome low accuracy of WkNN 2.Stable and robust	1.Computation Complexity	Methods facing outlets
8.	Pseudo/Generalized Nearest Neighbor (GNN) [9]	Utilizes information of n-1 neighbors also instead of only nearest neighbor	1.uses n-1 classes which consider the whole training data set	1.does not hold good for small data 2.Computationa;l complexity	Large data set
9.	Clustered k nearest neighbor [11]	Clusters are formed to select nearest neighbor	1.Overcome defect of uneven distributions of training samples 2.Robust in nature	1.Selection of threshold parameter is difficult before running algorithm 2.Biased by value of k for clustering	Text Classification
10.	Ball Tree k nearest neighbor (KNS1) [21,22]	Uses ball tree structure to improve kNN speed	1.Tune well to structure of represented data 2.Deal well with high dimensional entities 3.Easy to implement	1.Costly insertion algorithms 2.As distance increases KNS1 degrades	Geometric Learning tasks like robotic, vision, speech, graphics
11.	k-d tree nearest neighbor (kdNN) [23]	divide the training data exactly into half plane	1.Produce perfectly balanced tree 2.Fast and simple	1.More computation 2.Require intensive search 3.Blindly slice points into half which may miss data structure	organization of multi dimensional points
12.	Nearest feature Line Neighbor (NFL) [24]	take advantage of multiple templates per class	1.Improve classification accuracy 2.Highly effective for small size 3.utilises information ignored in nearest neighbor i.e. templates per class	1.Fail when prototype in NFL is far away from query point 2.Computations Complexity 3.To describe features points by straight line is hard task	Face Recognition problems
13.	Local Nearest Neighbor [25]	Focus on nearest neighbor prototype of query point	1.Cover limitations of NFL	1.Number of Computations	Face Recognition
14.	Tunable Nearest Neighbor (TNN) [26]	A tunable metric is used	1.Effective for small data sets	1.Large number of computations	Discrimination problems
15.	Center based Nearest Neighbor (CNN) [27]	A Center Line is calculated	1.Highly efficient for small data sets	1. Large number of computations	Pattern Recognition
16.	Principal Axis Tree Nearest Neighbor (PAT) [28]	Uses PAT	1.Good performance 2.Fast Search	1.Computation Time	Pattern Recognition
17.	Orthogonal Search Tree Nearest Neighbor [29]	Uses Orthogonal Trees	1.Less Computation time 2.Effective for large data sets	1.Query time is more	Pattern Recognition

Εικόνα (από [Bhatia and Vandana 2010]). Σύγκριση μεθόδων KNN

Κεφάλαιο 4 ΜΕΘΟΔΟΙ ΑΝΑΖΗΤΗΣΗΣ ΚΟΝΤΙΝΟΤΕΡΩΝ ΓΕΙΤΟΝΩΝ

Κοντινότεροι γείτονες και instance based learning

Οι μέθοδοι προσπέλασης για πολυδιάστατα δεδομένα (Multidimensional access methods) εκτελούν τις αναζητήσεις κοντινότερων γειτόνων σε δεδομένα σημείων. Η αναζήτηση κοντινότερων γειτόνων [Bhatia and Vandana 2010] (nearest neighbor search), γνωστή και ως αναζήτηση εγγύτητας (proximity search), αναζήτηση ομοιότητας (similarity search) ή αναζήτηση κοντινότερου σημείου (closest point search) είναι ένα πρόβλημα βελτιστοποίησης για την εύρεση σημείων σε d-διάστατους ευκλείδειους και μη χώρους, σύμφωνα με κάποια συνάρτηση απόστασης. Η άπληστη αναζήτηση όλων των σημείων N με όλα είναι πολυπλοκότητας O(N^2), και η αναζήτηση ομάδων πυκνοτήτων μπορεί να γίνει O(N^3), ή και O(N^4) για αναζήτηση χώρων πλέγματος μέγιστης πυκνότητας. Πολλοί μέθοδοι διαμέρισης του χώρου αναπτύχθηκαν με ένα από τα απλούστερα και αποτελεσματικότερα δένδρα που εξετάζουμε εδώ. Ο χρόνος πολυπλοκότητας για τις αναζητήσεις σε λίγες διαστάσεις είναι γενικά O(log N).

Πολλοί αλγόριθμοι αναζήτησης κοντινότερων γειτονικών σημείων περιλαμβάνουν τα quadtrees, kD-trees, kd-B-tree αλλά και τα αποτελεσματικά για πολλές διαστάσεις Balltrees. Στην περίπτωση γενικών μετρικών χώρων η προσέγγιση είναι γνωστή κάτω από το όνομα Metric trees που περιλαμβάνουν τα VP-tree και Bk-tree, ή την μέθοδο Locality-sensitive hashing (LSH) που είναι τεχνική για το πρόβλημα των πολλών διαστάσεων, που προσπαθεί να ομαδοποιήσει σημεία στο χώρο μέσα σε κάδους (buckets), βασιζόμενη σε κάποια μετρική απόσταση, όπου σημεία κοντινά μεταξύ τους αντιστοιχίζονται στον ίδιο κάδο με μεγάλη πιθανότητα, ή τα spill trees για προσεγγιστική αναζήτηση γειτόνων.

Η μέθοδος του κοντινότερου γείτονα δημιουργήθηκε πριν από αρκετές δεκαετίες, και στατιστικολόγοι ανέλυαν τη διάταξη k-κοντινότερων- γειτόνων στις αρχές της δεκαετίας του 1950. Αν ο αριθμός των περιπτώσεων εκπαίδευσης είναι μεγάλος, δημιουργούν ενστικτώδη αίσθηση για τη χρησιμοποίηση περισσότερων του ενός κοντινότερων γειτόνων, αλλά σαφώς αυτό είναι επικίνδυνο εάν υπάρχουν λίγες περιπτώσεις. Μπορεί να δειχθεί [Witten and Frank 2005] ότι , όταν k και ο αριθμό n και των δύο περιπτώσεων γίνουν άπειρα με τέτοιο τρόπο ώστε $k/n \to 0$, η πιθανότητα λάθους πλησιάζει το θεωρητικό ελάχιστο για το σύνολο δεδομένων. Η μέθοδος του κοντινότερου γείτονα, υιοθετήθηκε ως μια μέθοδος ταξινόμησης στις αρχές της δεκαετίας του 1960 και έχει χρησιμοποιηθεί ευρέως στον τομέα της αναγνώρισης προτύπων για περισσότερο από τρεις δεκαετίες.

Όταν η μάθηση βασίζεται σε παραδείγματα εκπαίδευσης [Witten and Frank 2005] όπως οι αλγόριθμοι One Nearest Neighbor, K-Nearest Neighbor, και άλλοι τότε μία

συνάρτηση απόστασης θα χρησιμοποιείται για να προσδιορίσει ποιο μέλος του συνόλου εκπαίδευσης είναι πλησιέστερο σε ένα άγνωστο παράδειγμα. Μόλις βρεθεί το πλησιέστερο παράδειγμα εκπαίδευσης η κλάση του θεωρείται ότι είναι ίδια με αυτή του αγνώστου παραδείγματος. Το μόνο πρόβλημα που παραμένει είναι να καθοριστεί η συνάρτηση απόστασης και αυτό δεν είναι πολύ δύσκολο, ιδιαίτερα αν τα χαρακτηριστικά γνωρίσματα είναι αριθμητικά.

Η συνάρτηση απόστασης (distance function)

Παρόλο που υπάρχουν και άλλες πιθανές επιλογές, οι περισσότεροι instance-based learners χρησιμοποιούν την Ευκλείδεια απόσταση [Witten and Frank 2005]. Η απόσταση μεταξύ ενός παραδείγματος με τιμές των χαρακτηριστικών $a_1^{(1)}, a_2^{(1)}, \ldots, a_k^{(1)}$ (όπου k είναι ο αριθμός των χαρακτηριστικών) και ενός με τιμές $a_1^{(2)}, a_2^{(2)}, \ldots, a_k^{(2)}$ ορίζεται ως $\sqrt{(a_1^{(1)}-a_1^{(2)})^2+(a_2^{(1)}-a_2^{(2)})^2+\ldots+(a_k^{(1)}-a_k^{(2)})^2}$. Δεν είναι απαραίτητο να βρούμε τη λειτουργία της τετραγωνικής ρίζας; τα αθροίσματα των τετραγώνων μπορούν να συγκριθούν άμεσα. Μια εναλλακτική της Ευκλείδεια απόσταση είναι η Manhattan ή η city-block μετρική, όπου η διαφορά μεταξύ των χαρακτηριστικών τιμών δεν είναι τετράγωνο, αλλά μόλις προστεθούν (μετά τη λήψη της απόλυτης τιμής). Άλλοι τρόποι προκύπτουν με τη λήψη δυνάμεων μεγαλύτερων του τετραγώνου. Μεγαλύτερες δυνάμεις αυξάνουν την επιρροή από μεγάλες διαφορές σε βάρος των μικρών διαφορών. Γενικά, η Ευκλείδεια απόσταση αντιπροσωπεύει έναν καλό συμβιβασμό. Άλλες μετρικές αποστάσεις μπορεί να είναι περισσότερο κατάλληλες σε ειδικές περιπτώσεις. Το κλειδί είναι να σκεφτόμαστε τα πραγματικά δεδομένα και τι σημαίνει γι'αυτά να χωριστούν από μια απόσταση.

Διαφορετικά χαρακτηριστικά γνωρίσματα μετρώνται σε διαφορετικές κλίμακες, έτσι ώστε αν ο τύπος της Ευκλείδειας απόστασης χρησιμοποιήθηκε κατευθείαν, τα αποτελέσματα από μερικά χαρακτηριστικά ίσως είναι εντελώς επισκιασμένα από άλλα που είχαν μεγαλύτερες κλίμακες μέτρησης. Άρα είναι σύνηθες να ομαλοποιούμε όλες τις τιμές των χαρακτηριστικών να απλώνονται μεταξύ 0 και 1, από τον υπολογισμό

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i}$$

όπου vi είναι η πραγματική τιμή του χαρακτηριστικού i, και το μέγιστο και ελάχιστο υπολογίζονται από όλες τις περιπτώσεις στο σύνολο εκπαίδευσης.

Αυτές οι φόρμουλες παίρνουν αριθμητικά χαρακτηριστικά. Εδώ, η διαφορά ανάμεσα σε δύο τιμές είναι ακριβώς η αριθμητική διαφορά μεταξύ τους, και αυτή είναι η διαφορά που τετραγωνίζετε και προστίθεται η απόδοση της συνάρτησης. Για ονομαστικά χαρακτηριστικά που λαμβάνουν τιμές που σύμβολα παρά αριθμοί, η διαφορά μεταξύ δύο τιμών που δεν είναι η ίδια συχνά θεωρείται ότι είναι ένα, ενώ αν οι τιμές είναι οι ίδιες, η διαφορά είναι μηδέν. Δεν απαιτείται κανονικοποίηση σε αυτή την περίπτωση, επειδή χρησιμοποιούνται μόνο οι τιμές 0 και 1.

Μερικές τιμές μπορεί να λείπουν (missing values), ή να είναι κενές. Μια κοινή πολιτική για τη διαχείριση των κενών τιμών (missing values) έχει ως εξής. Για τις ονομαστικές ιδιότητες, θεωρούμε ότι ένα χαρακτηριστικό που λείπει είναι μέγιστα διαφορετικό από οποιαδήποτε άλλη χαρακτηριστική τιμή. Έτσι, αν μία ή και οι δύο τιμές λείπουν, ή αν οι τιμές είναι διαφορετικές, η διαφορά μεταξύ τους λαμβάνεται ως ένα. Η διαφορά είναι μηδέν μόνο αν δεν λείπουν και είναι και οι δύο το ίδιο.

Για αριθμητικά χαρακτηριστικά, η διαφορά μεταξύ δύο τιμών που λείπουν λαμβάνεται επίσης ως ένα. Ωστόσο, αν λείπει μία μόνο τιμή, η διαφορά λαμβάνεται συχνά είτε ως το (ομαλοποιημένο) μέγεθος της άλλης τιμής ή ένα μικρότερο από αυτό μέγεθος, οποιοδήποτε είναι μεγαλύτερο. Αυτό σημαίνει ότι αν οι δύο τιμές λείπουν, η διαφορά τους είναι τόσο μεγάλη όσο μπορεί πιθανώς να είναι.

Παρόλο που η μάθηση από παραδείγματα είναι απλή και αποτελεσματική, είναι συχνά αργή [Witten and Frank 2005]. Ο προφανής τρόπος για να βρούμε ποιο μέλος του συνόλου εκπαίδευσης είναι το κοντινότερο σε ένα άγνωστο παράδειγμα δοκιμής είναι να υπολογίσουμε την απόσταση από κάθε μέλος του συνόλου εκπαίδευσης και να επιλέξουμε το μικρότερο.

QUAD TREES

Τα Quad-Trees είναι για δεδομένα δύο διαστάσεων. Τα Quad-Trees προτάθηκαν από τους R.Finkel και J.L.Bentley το 1974. Πρόκειται για μια από τις πιο έντονα μελετημένες δομές δεδομένων η οποία στηρίζεται στις αρχές της περιοδικά επαναλαμβανόμενης διάσπασης *divide and conquer* methods) Στην απλούστερη του μορφή το Quad –Tree είναι όμοιο με το δυαδικό δένδρο, με εξαίρεση ότι κάθε κόμβος-γονέας έχει τέσσερις απογόνους (κάποιοι από τους οποίους μπορεί να είναι κενοί), για αυτό και ονομάζεται τετραδικό (Quad) δένδρο. Το Quad – Tree είναι μια μη ισορροπημένη (unbalanced) χωρική δομή δεδομένων η οποία διασπάει περιοδικά τον χώρο (δυσδιάστατο χώρο) σε τέσσερα ίσου μεγέθους ασυνάρτητα τεταρτημόρια, μέχρι κάθε κόμβος φύλλο να περιέχει ένα μόνο σημείο.

Τα Quad Trees είναι μια οικογένεια δέντρων με πολλές παραλλαγές. Το κοινό τους στοιχείο είναι ότι αποτελούν ιεραρχικές δομές δεδομένων, που βασίζουν τη λειτουργία τους στην αναδρομική αποσύνθεση (decomposition) του χώρου. Οι διάφορες παραλλαγές quadtrees διαφοροποιούνται με βάση το είδος των δεδομένων που αναπαριστούν, τη μέθοδο (αρχή) με την οποία γίνεται η αποσύνθεση του χώρου (από μεγαλύτερες σε μικρότερες περιοχές) και το αν η ευκρίνεια (resolution) της δομής καθορίζεται στατικά ή μπορεί να μεταβάλλεται.

Σε ένα quad tree κάθε εσωτερικός κόμβος του έχει 4 κόμβους - παιδιά. Κάθε κόμβος αντιστοιχεί σε μια περιοχή του χώρου και κάθε παιδί του αντιστοιχεί σε ένα υποσύνολο του πατρικού χώρου. Εδώ ασχολούμαστε με περιπτώσεις στις οποίες ο χωρισμός του χώρου γίνεται κανονικά δηλαδή κάθε κόμβος είναι τετραγωνικού σχήματος και τα

παιδιά του είναι ισομεγέθη. Έτσι κάθε κόμβος αντιστοιχεί σε μια τετραγωνική περιοχή του χώρου και κάθε παιδί του αντιστοιχεί στο ¼ του χώρου του.

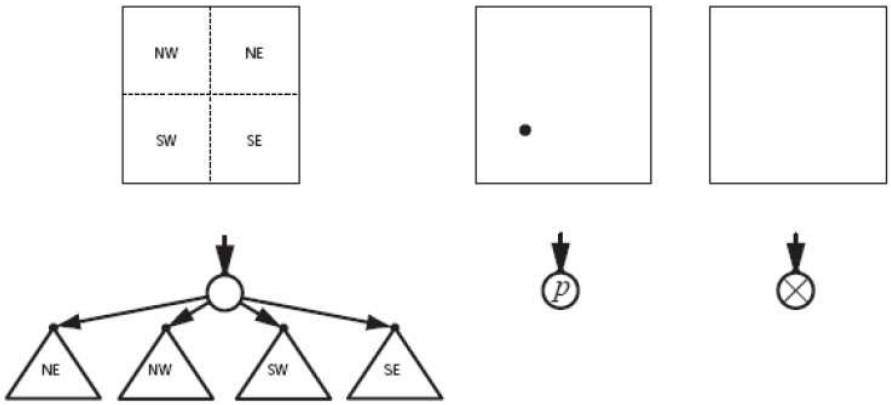
Συγκεκριμένα ο γονικός κόμβος διαμερίζεται στα τέσσερα τεταρτημόριά (quadrants) του, δηλαδή το κάθε παιδί του αντιστοιχεί σε ένα τεταρτημόριό του και για το λόγο αυτό μπορούμε να συμβολίσουμε τα παιδιά του με βάση τη θέση τους ως προς το κέντρο του πατρικού τετραγώνου δηλ. NW (north-west), NE (north-east), SW (south-west), SE (south-east). Φύλλα του δέντρου είναι οι κόμβοι εκείνοι για τους οποίους δεν απαιτείται περεταίρω διαμέριση. Τα δέντρα του τύπου αυτού ονομάζονται region quadtrees.

Σε περιπτώσεις που θέλουμε να κάνουμε για παράδειγμα indexing μια εικόνα (που είναι μια πολύ συχνή χρήση των quadtrees) κάθε φύλλο απλά κρατά την τιμή του χρώματος για την περιοχή αυτή(ο χώρος αποσυντίθεται σε περιοχές με το ίδιο χρώμα). Αντίθετα εδώ μας ενδιαφέρει για κάθε περιοχή του χώρου να γνωρίζουμε ποια σημειακά αντικείμενα οριοθετούνται από αυτή. Γι' αυτό το λόγο σε κάθε φύλλο αποθηκεύουμε τα σημειακά αντικείμενα που περιέχει. Αν θεωρήσουμε ότι κάθε φύλλο του δέντρου χωράει μόνο ένα σημειακό αντικείμενο (το ίδιο ισχύει και για περισσότερα σημεία), τότε μπορεί να έχουμε τρία είδη κόμβων:

• Εσωτερικούς κόμβους (δηλαδή κόμβοι που δεν είναι φύλλα)

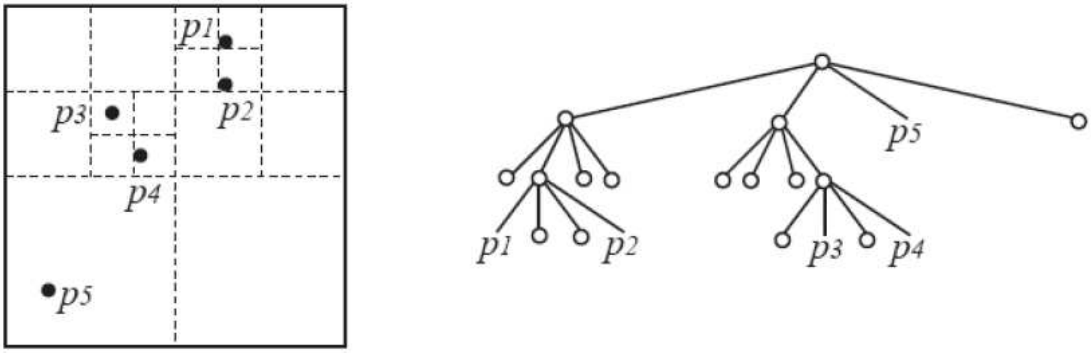
• Κόμβους – φύλλα που περιέχουν ένα σημείο

• Κόμβους – φύλλα που είναι άδειοι.



Εικόνα. Είδη κόμβων Quad tree

Το είδος αυτό του δέντρου, που συσχετίζει τεταρτημόρια με σημεία ονομάζεται PR Quadtree (Point – Region Quadtree). Παρακάτω φαίνεται ο αναδρομικός χωρισμός του αρχικού χώρου μετά την προσθήκη πέντε σημείων, καθώς και το αντίστοιχο δέντρο που προκύπτει.



Εικόνα. Παράδειγμα αναδρομικού χωρισμού του χώρου

Σε ένα PR quadtree κάθε κόμβος είναι μια εγγραφή που περιέχει τις ακόλουθες καταχωρήσεις:

• Τέσσερις δείκτες προς τα παιδιά του κόμβου, καθένας από τους οποίους δηλαδή αντιστοιχεί σε ένα τεταρτημόριο. Αν ο P είναι δείκτης προς ένα κόμβο και το I είναι ένα τεταρτημόριο, οι καταχωρήσεις αυτές αναφέρονται ως SON(P, I).

• Το πέμπτο πεδίο, NODETYPE, έχει την τιμή BLACK αν πρόκειται για φύλλο που περιέχει σημείο, WHITE αν πρόκειται για κενό φύλλο και GRAY αν πρόκειται για κόμβο που δεν είναι φύλλο.

• Το πεδίο NAME, δηλαδή το όνομα του σημειακού αντικειμένου που είναι αποθηκευμένο στον κόμβο

• Τα πεδία XCOORD και YCOORD, δηλαδή τις συντεταγμένες του σημειακού αντικειμένου.

Τα Quad tree είναι για δύο διαστάσεις. Για δεδομένα περισσότερων από δύο διαστάσεων τα k-d δένδρα παρουσιάζονται παρακάτω.

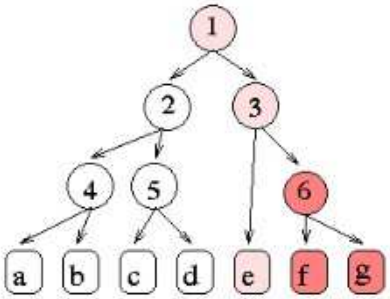
KD TREE

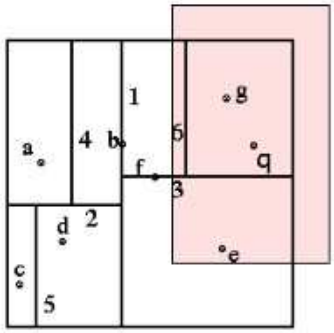
Τα δέντρα kd (k-d trees) εφαρμόζονται στην αναζήτηση πληροφοριών εδώ και αρκετές δεκαετίες [Freidman et al., 1977], [Bentley, 1975]. Πρόκειται για δυαδικά δέντρα και είναι μία δομή δεδομένων που αποθηκεύει έναν πεπερασμένο αριθμό σημείων διάστασης k (το kd είναι συντομογραφία του k-dimentional) και έχουν μεγάλο εύρος εφαρμογών στον τομέα της μάθησης [Moore, a] και των νευρωνικών δικτύων [Omohundro, 1987].

Το δέντρο k-d κατασκευάζεται διαιρώντας τον πολυδιάστατο χώρο σε ημιεπίπεδα διαδοχικά ως προς τις διάφορες διαστάσεις. Ως είσοδο για την κατασκευή του παίρνει n στοιχεία διάστασης k. Διαδοχικά σε κάθε μία από τις k διαστάσεις κυκλικά, ο χώρος διαιρείται στη μέση σε δύο ημιεπίπεδα και καθέ ένα από αυτά έπειτα διαιρούνται ξανά ως προς την επόμενη διάσταση. Η διαδικασία αυτή τερματίζεται όταν κάθε ένα από τα σημεία της εισόδου έχει μόνο του τη δικιά του περιοχή.

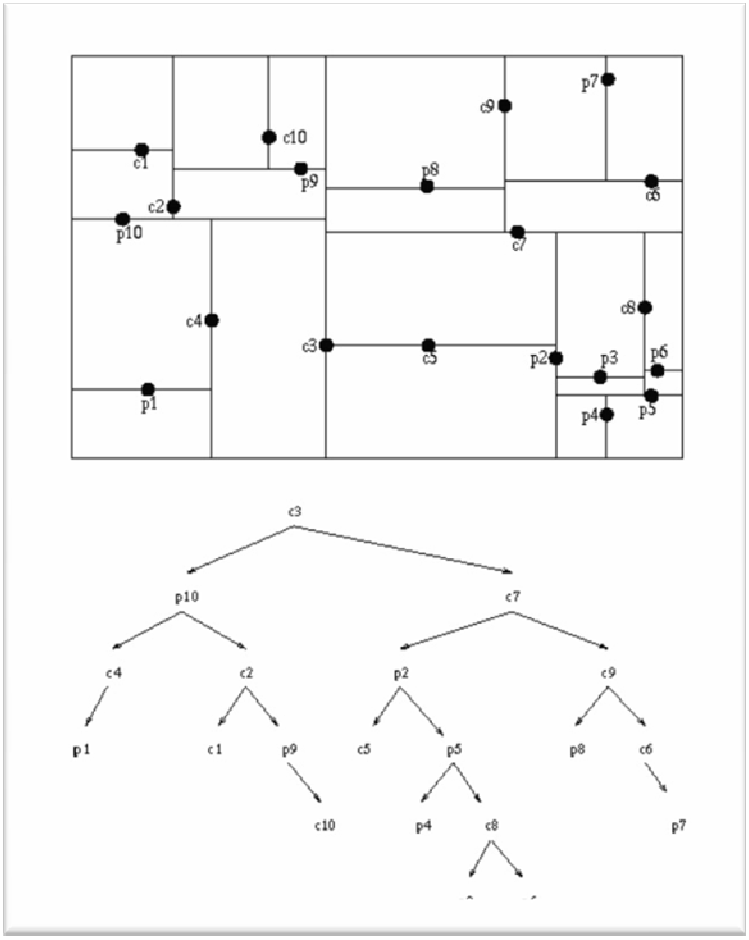
Δημιουργείται έτσι ένα δέντρο, το οποίο μας παρέχει μια πολύ γρήγορη αναζήτηση σε όλα τα σημεία ως προς τη θέση τους. Το δέντρο αυτό έχει ύψος log(n)

Ένα δέντρο για σημεία διάστασης 2 φαίνεται στα επόμενο σχήμα στην αριστερή μεριά. Στη δεξιά έχουμε τον δισδιάστατο χώρο των σημείων και τις περιοχές του καθενός. Σε μία περίπτωση ερωτήματος για τον πλησιέστερο γείτονα ενός σημείου q εκτός δέντρου αρκεί να προσπελαστούν οι χρωματισμοί, στο αριστερό τμήμα, κόμβοι του δέντρου.





Στο επόμενο σχήμα βλέπουμε ένα KD δέντρο που θα το χρησιμοποιήσουμε για να αναφέρουμε ένα παράδειγμα λειτουργίας του kd tree.

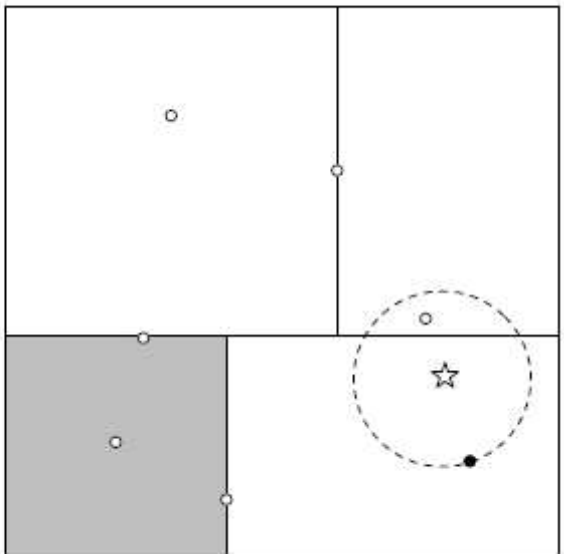


Ξεκινάμε από το c3. Φέρνουμε γραμμή κάθετη που περνάει από αυτό και μετά κάνουμε το ίδιο στους δύο χώρους που προκύπτουν με οριζόντιες γραμμές αυτή τη φορά στα p10 και c7 κ.ο.κ. Να σημειώσουμε ότι είναι υποχρεωτικό να έχουμε εναλλάξ οριζόντιες και κάθετες γραμμές.

Το βασικό πρόβλημα του παραπάνω δέντρου είναι ότι εξαρτάται έντονα από τη σειρά με την οποία θα έρθουν τα σημεία, μιας και, αν αυτή αλλάξει, αλλάζει και το δέντρο. Το προσαρμοζόμενο kd δέντρο είναι μία παραλλαγή και λύνει αυτό ακριβώς το πρόβλημα. Το δέντρο χωρίζει, τώρα, κάθε φορά τους υποχώρους με τέτοιο τρόπο ώστε να βρίσκει κανείς σχεδόν τον ίδιο αριθμό σημείων σε κάθε προκύπτον υποχώρο. Οι γραμμές είναι και πάλι παράλληλες στους άξονες του επιπέδου όμως τώρα δε χρειάζεται να περιλαμβάνουν ένα τουλάχιστον σημείο. Οι εσωτερικοί κόμβοι κρατάνε την τετμημένη ή τεταγμένη των γραμμών και τα σημεία-δεδομένα αποθηκεύονται στα φύλλα του δέντρου.

Εντοπισμός κοντινότερων γειτόνων με k-d δένδρα

Για να εντοπίσετε τον κοντινότερο γείτονα από ένα σημείο στόχο που δόθηκε, ακολουθήστε το δέντρο από κάτω από τη ρίζα του για να εντοπίσετε την περιοχή που περιέχει το στόχο. Η επόμενη εικόνα δείχνει ένα παράδειγμα.



Εικόνα. Χρήση του k-d δένδρου για τον εντοπισμό γειτόνων του στόχου (αστέρι).

Ο στόχος, ο οποίος δεν είναι ένα από τα σημεία στο δέντρο, είναι σημειωμένος από ένα αστέρι. Ο κόμβος φύλλου που περιέχει την περιοχή του στόχου είναι βαμμένος μαύρος. Αυτό δεν είναι απαραίτητα ο στόχος του κοντινότερου γείτονα, όπως αυτό το παράδειγμα απεικονίζει, αλλά είναι μια καλή πρώτη προσέγγιση. Ειδικότερα, κάθε κοντινότερος γείτονας πρέπει να βρίσκεται πιο κοντά- εντός του διακεκομμένου κύκλου. Για να καθοριστεί αν υπάρχει, πρώτα ελέγχουμε αν είναι δυνατό για ένα πιο κοντινό γείτονα να βρίσκεται εντός του αμφιθαλή κόμβου. Στη συνέχεια, επιστρέφουμε στον

πατρικό κόμβο και ελέγχουμε τους αμφιθαλής-όπου εδώ καλύπτουν τα πάντα πάνω από την οριζόντια γραμμή. Σε αυτή την περίπτωση, πρέπει να διερευνηθεί, επειδή η περιοχή που καλύπτει τέμνετε με τον καλύτερο κύκλο μέχρι τώρα.

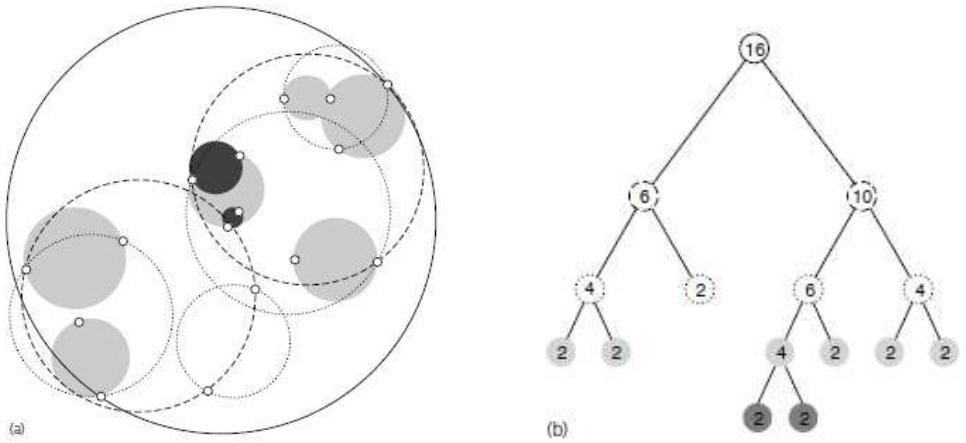
Σε μια τυπική περίπτωση, αυτός ο αλγόριθμος είναι πολύ πιο γρήγορος από την εξέταση όλων των σημείων για να βρει τον πλησιέστερο γείτονα. Οι εργασίες που εμπλέκονται για την εύρεση του αρχικού κατά προσέγγιση πλησιέστερου γείτονα, το μαύρο σημείο στο σχήμα -εξαρτάται από το βάθος του δέντρου, που δίνεται από το λογάριθμο των αριθμών των κόμβων, log2n. Ο φόρτος εργασίας στους υπολογισμούς για την επαναχώρηση κατά τον έλεγχο αν αυτό το σημείο είναι πραγματικά ο πλησιέστερος γείτονας εξαρτάται από το δέντρο, και από το πόσο καλή είναι η αρχική προσέγγιση. Αλλά για ένα καλά δομημένο δέντρο του οποίου οι κόμβοι είναι περίπου τετράγωνοι, παρά μεγάλα λεπτά ορθογώνια, μπορεί επίσης να αποδειχθεί ότι είναι λογαριθμικό του αριθμού των κόμβων.

ΑΠΟ ΤΑ K-D TREES ΣΤΑ BALL TREES

Όπως είδαμε, τα KD-δένδρα είναι καλές δομές δεδομένων για την εύρεση του κοντινότερου γείτονα αποτελεσματικά. Ωστόσο, δεν είναι τέλεια. Λοξά (skewed) σύνολα δεδομένων παρουσιάζουν μια βασική σύγκρουση μεταξύ της επιθυμίας για το δέντρο να είναι απόλυτα ισορροπημένο και της επιθυμίας για τις περιοχές να είναι τετράγωνες. Τα πιο σημαντικά, ορθογώνια, ακόμη και τετράγωνα-δεν είναι τα καλύτερα σχήματα για να χρησιμοποιούνται οπουδήποτε, εξαιτίας των γωνιών τους. Εάν ο διακεκομμένος κύκλος στην προηγούμενη εικόνα ήταν μεγαλύτερος, το οποίο θα συνέβαινε αν το μαύρο παράδειγμα ήταν λίγο μακρύτερα από τον στόχο, τότε θα έτεμνε την χαμηλότερη δεξιά γωνία του ορθογωνίου στην αριστερή κορυφή και μετά αυτό το ορθογώνιο θα έπρεπε να εξεταστεί επίσης, παρά το γεγονός ότι οι περιπτώσεις εκπαίδευσης που την καθορίζουν είναι πολύ μακριά από τη γωνία στην ερώτηση. Οι γωνίες στις περιοχές του ορθογωνίου είναι άβολες.

Η λύση, είναι να χρησιμοποιούμε υπερσφαίρες, όχι υπερορθογώνια. Γειτονικές σφαίρες ίσως υπερκαλύπτονται ενώ τα ορθογώνια που μπορούν να εφάπτονται-συνορεύουν, αλλά αυτό δεν είναι το πρόβλημα, επειδή ο αλγόριθμος του κοντινότερου-γείτονα για KD-tree που περιγράφηκε προηγουμένως δεν εξαρτάται από διαχωρισμένες περιοχές. Μια δομή δεδομένων που ονομάζεται ball tree ορίζει k-διαστάσεις υπερσφαιρών (”balls”), που καλύπτουν τα σημεία δεδομένων, και τους οργανώνει σε ένα δέντρο.

Η επόμενη εικόνα (a) δείχνει 16 περιπτώσεις εκπαίδευσης σε δισδιάστατο χώρο, και σε υπέρθεση από ένα μοτίβο επικάλυπτόμενων κύκλων και η (b) δείχνει ένα δέντρο που σχηματίζεται από αυτούς τους κύκλους.



Εικόνα. Ball tree για 16 περιπτώσεις κατάρτισης: (a) περιπτώσεις και μπάλες και (b) το δέντρο.Οι κύκλοι στην εικόνα σε διαφορετικά επίπεδα του δέντρου υποδηλώνονται από διάφορες μορφές παύλας, και οι μικρότεροι κύκλοι ζωγραφίζονται σε αποχρώσεις του γκρι.

Κάθε κόμβος του δέντρου αντιπροσωπεύει μια μπάλα, και ο κόμβος είναι διακεκομμένος ή σκιασμένος σύμφωνα με την ίδια συνήθεια, έτσι ώστε να μπορείτε να αναγνωρίσετε σε ποιό επίπεδο είναι οι σφαίρες. Για να σας βοηθήσει να καταλάβετε το δέντρο, οι αριθμοί τοποθετούνται στους κόμβους για να δείξουν πόσα σημεία δεδομένων του θεωρούνται ότι είναι μέσα σε αυτή την σφαίρα. Αλλά: αυτό δεν είναι απαραίτητα το ίδιο με τον αριθμό των σημείων που εμπίπτουν μέσα στην χωρική περιοχή που η μπάλα αντιπροσωπεύει. Οι περιοχές σε κάθε επίπεδο, μερικές φορές συμπίπτουν, αλλά τα σημεία που εμπίπτουν μέσα στην περιοχή επικάλυψης ορίζονται σε μία μόνο από τις επικαλύψασες μπάλες (το διάγραμμα δεν δείχνει σε ποια). Στο σχήμα (b) οι κόμβοι των πραγματικών ball-trees αποθηκεύουν το κέντρο και την ακτίνα της σφαίρας τους; οι κόμβοι των φύλλων καταγράφουν τα σημεία που περιέχουν επίσης.

Για να χρησιμοποιήσουμε ένα ball tree για να βρούμε τον κοντινότερο γείτονα στο στόχο που μας δόθηκε, ξεκινάμε να διασχίζουμε το δέντρο από πάνω προς τα κάτω για να εντοπίσουμε το φύλλο που περιέχει το στόχο και να βρούμε το κοντινότερο σημείο στο στόχο σε αυτή τη σφαίρα. Αυτό δίνει ένα ανώτερο όριο για την απόσταση του στόχου από το κοντινότερο γείτονά του. Στη συνέχεια, όπως και για τα-kD tree , εξετάστε τον αμφιθαλή κόμβο (sibling node). Αν η απόσταση από το στόχο για το αμφιθαλή κέντρο υπερβαίνει την ακτίνα του συν τον τρέχον άνω όριο, δεν μπορεί να περιέχει ένα κοντινότερο σημείο. Διαφορετικά πρέπει να εξετάσουμε περαιτέρω τον αμφιθαλή κατερχόμενοι το δέντρο.

Κεφάλαιο 5 METRIC BALL TREES

ΜΕΘΟΔΟΙ ΚΑΤΑΣΚΕΥΗΣ

Τα Metric Trees όπως και τα KDTrees είναι επίσης δυαδικά δέντρα. Αυτά αποσυνθέτουν ιεραρχικά το σημείο του χώρου σε περιοχές υπερσφαιρών. Κάθε κόμβος του δέντρου συνδέεται με μια μόνο σφαίρα που αντιπροσωπεύει μια περιοχή υπερσφαίρας του σημείου χώρου, και αποθηκεύει το κέντρο της σφαίρας και την ακτίνα. Κάθε κόμβος επίσης αποθηκεύει τα σημεία που βρίσκονται στο εσωτερικό της σφαίρας. Ένας εσωτερικός κόμβος, επιπρόσθετα σ'αυτό, αποθηκεύει επίσης τα δύο παιδιά κόμβους του που αποσυνθέτουν τη σφαίρα σε δύο (συνήθως) μικρότερες μπάλες. Οι σφαίρες επιτρέπεται να αλληλοκαλύπτονται. Ωστόσο, τα σημεία μπορούν να ανήκουν σε μια μόνο σφαίρα κι μια μόνο υπο-σφαίρα. Τα δέντρα χρειάζονται να γνωρίζουν μόνο τα ζεύγη αποστάσεων των σημείων μεταξύ τους, και επειδή αυτά τα μέτρα αποστάσεων λέγονται μετρικές , τα δέντρα λέγονται metric trees.

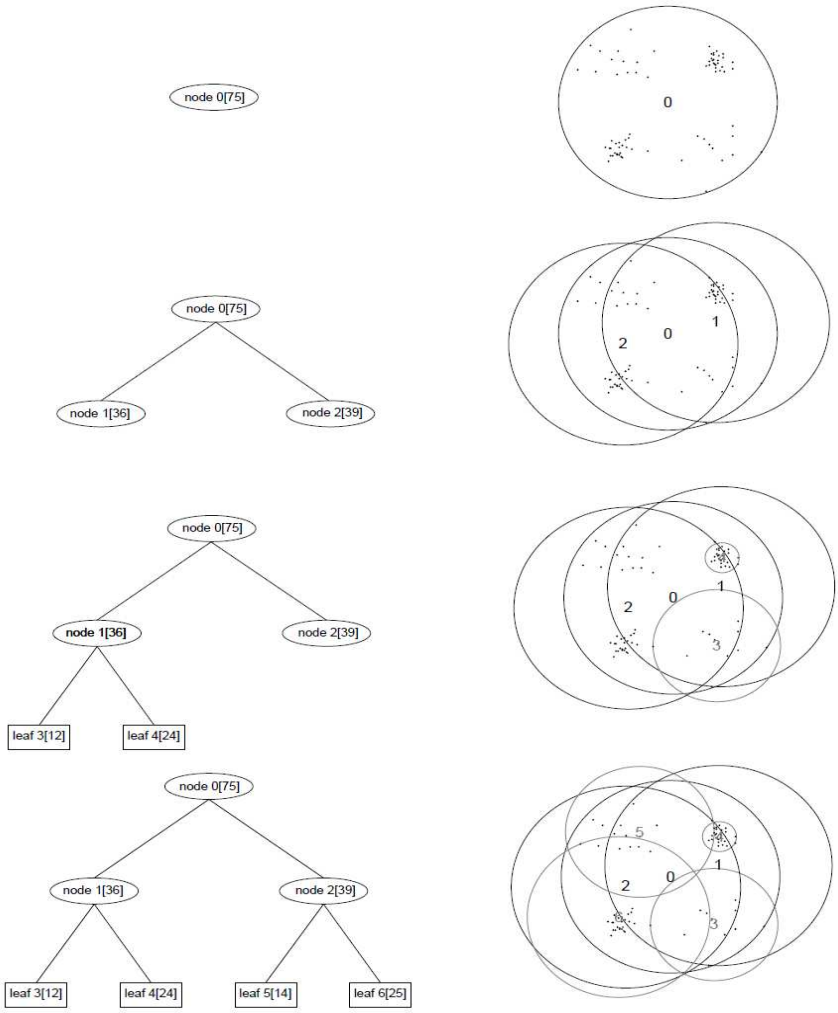
Τα Metric Trees, σε αντίθεση με άλλες μεθόδους NN δεν έχουν μία μόνο μέθοδο κατασκευής. Στα KDTrees, για παράδειγμα, η κατασκευή περιλαμβάνει την επιλογή ενός splitdim και splitval για μια περιοχή και μετά τον διαχωρισμό της περιοχής σύμφωνα με αυτές τις επιλεγμένες τιμές. Όλες οι παραλλαγές των μεθόδων κατασκευής KDTrees έχουν να κάνουν ακριβώς με την επιλογή των splitdim και splitval. Στα Metric Trees, όμως, σε αντίθεση με τα KDTrees και άλλα, δεν υπάρχει καμιά τέτοια ενιαία θεμελιώδης μέθοδος κατασκευής. Τα Metric Trees έχουν τρείς βασικές μεθόδους κατασκευής. Αυτές είναι οι πάνω προς τα κάτω, από κάτω προς τα πάνω και από τη μέση προς τα έξω μέθοδοι κατασκευής. Η περιγραφή των μεθόδων αυτών ακολουθεί παρακάτω.

1. Top Down Construction: η μέθοδος κατασκευής από πάνω προς τα κάτω, όπως το όνομα υπονοεί, χτίζει το δέντρο αρχίζοντας από τον κόμβο στην κορυφή και προς το κάτω μέρος. Αυτή η μέθοδος είναι κάπως παρόμοια με τη μέθοδο κατασκευής του KDTrees. Ξεκινάμε αναθέτοντας στο κόμβο ρίζας μια σφαίρα οριοθέτησης σε ολόκληρο το χώρο σημείου. Η σφαίρα ρίζας στη συνέχεια χωρίζεται σε δύο (συνήθως) μικρότερες σφαίρες οι οποίες στη συνέχεια ανατίθενται στα δύο παιδιά του κόμβου ρίζας. Η διαδικασία της διάσπασης εφαρμόζεται τότε αναδρομικά σε κάθε μία από τις σφαίρες παιδιά, όπως και στα KDTrees, η διάσπαση σταματάει όταν ο αριθμός των σημείων σε μια σφαίρα πέφτει κάτω από ένα συγκεκριμένο όριο (παρόμοιο με το μέγεθος του κάδου b στα KDTrees). Η διαδικασία απεικονίζεται γραφικά στο σχήμα. Στη σφαίρα αποσύνθεσης οι σφαίρες του φύλλου κόμβου αντιπροσωπεύονται με (ελαφρύτερες-πιο απαλές) σφαίρες, ενώ οι πιο σκούρες από αυτές είναι για τους εσωτερικούς κόμβους. Οι αριθμοί στο εσωτερικό των σφαιρών αντιπροσωπεύουν το κέντρο της σφαίρας και επίσης δείχνουν τον κόμβο/φύλλο στον

οποίο μια σφαίρα ανήκει στην αντίστοιχη (εικόνα-παράδειγμα) δέντρου στην αριστερή πλευρά του σχήματος.

2. Bottom Up Construction: Η μέθοδος κατασκευής από κάτω προς τα πάνω χτίζει το δέντρο με την οικοδόμηση του πιο (χαμηλού-κάτω) κόμβου πρώτα και στη συνέχεια επαναλαμβάνοντας την οικοδόμηση ανέρχεται στον υψηλότερο κόμβο. Η μέθοδος ξεκινάει με το να βρεί από όλα τα σημεία ένα ζευγάρι που έχει τη μικρότερη σφαίρα οριοθέτησης, και στη συνέχεια τη δημιουργία ενός κόμβου γι'αυτό το ζεύγος σημείων. Η μέθοδος τότε επαναληπτικά βρίσκει και χτίζει κόμβους για ένα ζευγάρι από σημεία, ή κόμβους, ή ένα ζευγάρι που αποτελείται από ένα κόμβο και ένα σημείο, οποιοδήποτε έχει τη σφαίρα με το ελάχιστο όριο από όλα τα άλλα ζευγάρια. Αυτή η επαναλαμβανόμενη διαδικασία για την εύρεση και την κατασκευή των κόμβων για ζευγάρια με ελάχιστη σφαίρα οριοθέτησης συνεχίζεται έως ότου όλα τα σημεία συγχωνευθούν σε κόμβους, και όλοι οι κόμβοι συγχωνευθούν σε ένα ανώτερο κόμβο.

3. Middle Out Construction: Αυτή η μέθοδος λειτουργεί με την εύρεση ομάδων μεταξύ των σημείων και στη συνέχεια κατασκευάζοντας κόμβους από αυτές τις ομάδες χρησιμοποιώντας μια μέθοδο που ονομάζεται Anchors Hierarchy. Αυτοί οι κόμβοι των σημείων ομάδων στη συνέχεια συγχωνεύθηκαν χρησιμοποιώντας την από κάτω προς τα πάνω κατασκευή μέσα σε ένα ανώτερο κόμβο. Η διαδικασία για την εύρεση ομάδων, την κατασκευή κόμβων, και στη συνέχεια την συγχώνευση τους σε ένα ανώτερο κόμβο στη συνέχεια εφαρμόζεται αναδρομικά σε κάθε ένα από τους κόμβους των ομάδων, και η αναδρομή σταματά αν για κάποιο κόμβο ο αριθμός των σημείων πέσει κάτω από το όριο που δίνεται.

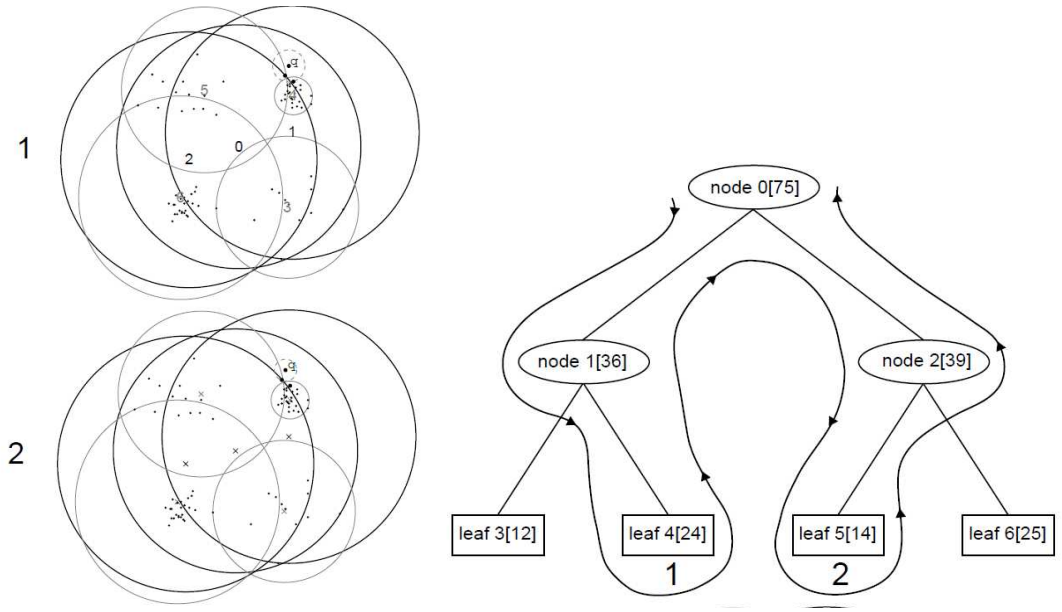


Εικόνα. Παράδειγμα κατασκευής top down για τα metric ball trees

ΒΑΣΙΚΑ ΕΡΩΤΗΜΑΤΑ

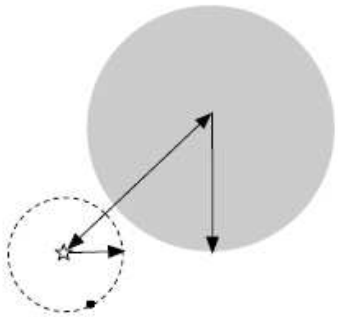
Στα Metric Trees, οι υπερσφαίρες (μπάλες) που αποσυνθέτουν τον χώρο, σε αντίθεση με τις υπερορθογώνιες περιοχές των KDTrees, είναι τελείως κλειστά. Οι σφαίρες που αντιστοιχούν στους κόμβους φύλλων είναι συχνά συγκεντρωμένες κοντά σε ομάδες των σημείων στο χώρο, και ως εκ τούτου είναι πιθανό ότι ένα συγκεκριμένο ερώτημα q δεν βρίσκεται σε κάποια περιοχή φύλλου. Έτσι, για ένα συγκεκριμένο ερώτημα q, η kNN έρευνα εκτελείται από τον πρώτο έλεγχο του κόμβου φύλλου του οποίου η σφαίρα είναι πλησιέστερα στο ερώτημα (ή του οποίου το κέντρο της σφαίρας είναι πλησιέστερα στο ερώτημα, αν και οι δύο σφαίρες φύλλων περιέχουν το ερώτημα). Σε κάθε εσωτερικό κόμβο, η διαδικασία αναζήτησης μετράει την απόσταση του q στα κέντρα των σφαιρών από κάθε κόμβο παιδί, και αναδρομικά ελέγχει τον κόμβο παιδί του οποίου η σφαίρα (ή, στην περίπτωση και των δύο σφαιρών που περιέχουν το q, των οποίων το κέντρο σφαιρών) είναι πιο κοντά στο q. Αφού φθάσει στον επιθυμητό κόμβο φύλλου, οι kNNs του q

αναζητούνται ανάμεσα στα σημεία του κόμβου φύλλου χρησιμοποιώντας απλή γραμμική αναζήτηση, και αποθηκεύονται. Όπως και τα KDTrees, κάνοντας αυτό είναι ισοδύναμο με τον υπολογισμό του ερωτήματος σφαίρας, η οποία επικεντρώνεται στο q και έχει ακτίνα ίση με τον καλύτερο Κοστο κοντινότερο γείτονα που βρέθηκε Κατά τη διάρκεια της επιστροφής, σε κάθε εσωτερικό κόμβο, η διαδικασία μόνο ελέγχει το άλλο παιδί (της οποίας το κέντρο σφαίρας-σφαιρών ήταν πιο μακριά από το ερώτημα) αν το ερώτημα σφαίρα τέμνεται με αυτό. Αν υπάρχει μια διασταύρωση, τότε ολόκληρη η διαδικασία εφαρμόζεται αναδρομικά για το άλλο παιδί. Το σχήμα δίνει ένα γραφικό παράδειγμα της διαδικασίας.



Εικόνα. Διαδικασία ερωτήματος

Στο επόμενο σχήμα ο στόχος είναι σημειωμένος με αστέρι και η μαύρη κουκκίδα είναι του κοντινότερου αυτή την στιγμή γνωστού γείτονα. Ολόκληρο το περιεχόμενο της γκρί σφαίρας μπορεί να αποκλειστεί: δεν μπορεί να περιέχει ένα κοντινότερο σημείο επειδή το κέντρο του είναι πολύ μακριά. Προχώρησε αναδρομικά πίσω το δέντρο μέχρι τη ρίζα του, εξετάζοντας κάθε σφαίρα που μπορεί να περιέχει ένα σημείο πιο κοντά στο τρέχον ανώτερο όριο.



Εικόνα. Αποκλείοντας μια ολόκληρη σφαίρα (γκρι) που βασίζεται σε ένα σημείο στόχο (αστέρι), καθώς και του τωρινού κοντινότερου γείτονα

Τα ball trees είναι χτισμένα από πάνω προς τα κάτω, και όπως με τα KD-trees το βασικό πρόβλημα είναι να βρούμε έναν καλό τρόπο διάσπασης της σφαίρας που περιέχει ένα σύνολο σημείων δεδομένων σε δύο. Στην πράξη, δεν χρειάζεται να συνεχιστεί μέχρι οι σφαίρες των φύλλων να περιέχουν μόνο δύο σημεία: μπορείτε να σταματήσετε νωρίτερα, μόλις ο προκαθορισμένος μικρότερος αριθμός επιτευχτεί, και το ίδιο ισχύει για τα KD-trees. Μια πιθανή μέθοδος διαχωρισμού δίνεται παρακάτω:

Η μέθοδος κατασκευής που χρησιμοποιήσαμε στην πτυχιακή εργασία είναι:

Επιλέγουμε το σημείο της σφαίρας που είναι το πλέον απομακρυσμένο από το κέντρο της, και στη συνέχεια ένα δεύτερο σημείο που είναι το πιο μακρινό από το πρώτο. Έτσι έχουμε δύο κέντρα cluster.

Αναθέτουμε το κάθε σημεία δεδομένων στην σφαίρα στο κοντινότερο από αυτά τα δύο κέντρα των cluster.

Έτσι δημιουργούνται δύο νέα cluster που θα γίνουν οι δύο υπο-σφαίρες.

Στη συνέχεια υπολογίζουμε το κέντρο βάρους της κάθε υπο-σφαίρας και την ακτίνα που απαιτείται για να περιβάλλει όλα τα σημεία δεδομένων που αντιπροσωπεύει.

Αυτή η μέθοδος έχει το προτέρημα ότι το κόστος διαχωρισμού μιας σφαίρας που περιέχει n σημεία είναι μόνο γραμμική ως προς N.

Υπάρχουν πιο περίπλοκοι αλγόριθμοι που δημιουργούν πιο σφικτές σφαίρες, αλλά αυτοί απαιτούν περισσότερους υπολογισμούς.

ΛΕΠΤΟΜΕΡΕΙΕΣ ΜΕΘΟΔΩΝ ΚΑΤΑΣΚΕΥΗΣ

Ας εξετάσουμε τώρα την κατασκευή των δέντρων με περισσότερες λεπτομέρειες. Τα Metric Trees από την αρχική έναρξη τους, έχουν έναν αριθμό μεθόδων κατασκευής που προτείνονται για αυτά. Οι περισσότερες από τις μεθόδους είναι από τον Omohundro, ο οποίος είναι ένας από τους συγγραφείς που πρότεινε αρχικά τις δομές τους, ενώ πιο πρόσφατα μερικές έχουν προταθεί από τον Moore. Ο Omohundro, όταν παρουσίασε τα Metric Trees (τα ονόμαζε Ball Trees) στο (Omohundro,1989), έδωσε πέντε διαφορετικές μεθόδους κατασκευής των δομών. Έδωσε επίσης μια πειραματική σύγκριση των μεθόδων του που παρουσίασε. Αυτός, ωστόσο, σύγκρινε τις μεθόδους μόνο από την άποψη του χρόνου κατασκευής τους και του συνολικού όγκου των σφαιρών που δημιούργησε, για ένα συγκεκριμένο αριθμό σημείων δεδομένων. Δεν υπολογίζει τις μεθόδους του από την άποψη του χρόνου ερωτήματος τους. Ενώ πρόσφατα, οι μέθοδοι που παρουσίασε ο Moore στο (Moore,2000) δεν έχουν εκτιμηθεί κατά τις μεθόδους Omohundro ή οποιασδήποτε άλλης, είτε από το Moore είτε της ομάδας του. Ο Uhlmann, ο άλλος παρουσιαστής των δομών, έδωσε επίσης μια μέθοδο κατασκευής όταν παρουσίασε τα δέντρα στο (Uhlmann,1991a,b). Η μέθοδος του δεν έχει λάβει πολλή προσοχή από τους άλλους,

και καμιά μελέτη δεν είναι γνωστή από τον ίδιο ή οποιονδήποτε άλλο που δίνει εκτίμηση της μεθόδου του έναντι των άλλων προτεινόμενων μεθόδων.

Οι διάφορες μέθοδοι κατασκευής για δέντρα που περιγράφονται λεπτομερώς παρακάτω, ομαδοποιούνται σε μεθόδους Omohundro, Uhlmann, και Moore.

Μέθοδοι Omohundro

1. Median of widest dimension: Αυτή η μέθοδος είναι παρόμοια με την καθιερωμένη μέθοδο κατασκευής του KDTree (που ονομάζεται από τον Omohundro k-d μέθοδος κατασκευής) και κατασκευάζει το δέντρο από την κορυφή προς τα κάτω. Χωρίζει μια περιοχή σύμφωνα με τη διάμεση τιμή των σημείων στη διάσταση στην οποία είναι μέγιστα απλωμένη. Μετά την εύρεση της splitdim διάστασης στην οποία τα σημεία ενός κόμβου είναι πάρα πολύ απλωμένα και τη διάμεση τιμή splitval κατά μήκος της splitdim, παρόμοια με το KDTrees, αυτή η μέθοδος κάνει την αριστερή σφαίρα με σημεία των οποίων η splitdim είναι μικρότερη ή ίση με την splitval και κάνει τη (σωστή-κατάλληλη-καλύτερη) σφαίρα από τα σημεία των οποίων η splitdim είναι μεγαλύτερη από την splitval. Το Metric Tree που κατασκευάστηκε με αυτή την μέθοδο είναι πάντα απόλυτα ισσοροπημένο, έχει 0(logn) βάθος, και μπορεί να κατασκευαστεί σε 0(nlogn) χρόνο.

2. Online μέθοδος εισαγωγής: Αυτή η μέθοδος κατασκευάζει το δέντρο από την κορυφή προς τα κάτω και σε απευθείας σύνδεση με την εισαγωγή σημείου. Θα εισάγει ένα σημείο μέσα στον κόμβο που θα είχε ως αποτέλεσμα την ελάχιστη αύξηση του όγκου του δέντρου. Για ένα συγκεκριμένο σημείο p, περνά από άκρη σε άκρη τους κόμβους του δέντρου, υπολογίζει την αύξηση του όγκου τους και την αύξηση του όγκου των προγόνων τους, που θα προκύψουν ως αποτέλεσμα της εισαγωγής p σε αυτές. Η μέθοδος διατηρεί τους υποψήφιους κόμβους για την εισαγωγή του p σε μια ουρά προτεραιότητας και εισάγει το p στον καλύτερο κόμβο που θα βρεί ότι δίνει την ελάχιστη αύξηση του όγκου σε ολόκληρο το δέντρο.

3. Φθηνότερη online μέθοδος εισαγωγής: Αυτή η μέθοδος είναι παρόμοια με την Online Insertion Method. Η μόνη διαφορά είναι ότι αντί για την αποθήκευση των σημείων στο PQ(Priority Quene) του δέντρου σε κάθε εσωτερικό κόμβο όταν ψάχνουν για τον καλύτερο κόμβο για την εισαγωγή κοιτάζει μόνο στο κλαδί του παιδιού η οποία δίνει μικρότερη αύξηση του όγκου και δεν διατηρεί το PQ των υποψήφιων κόμβων. Ο καλύτερος κόμβος βρίσκεται με αυτό τον ευρετικό τρόπο και το σημείο θα εισαχθεί στον καλύτερο κόμβο. Η μέθοδος είναι φθηνότερη σε κόστος υπολογισμού και αποθήκευσης καθώς λιγότεροι κόμβοι επιθεωρούνται και δεν χρειάζονται κόμβο να αποθηκευτούν κατά τη διάρκεια της διαδικασίας.

4. Minimum Volume Increase Heuristic: Αυτή η μέθοδος παρόμοια με την Online Insertion Method προσπαθεί να παράγει τα δέντρα με το ελάχιστο άθροισμα όγκων των σφαιρών. Η μέθοδος παρόμοια με την Online Insertion δουλεύει από την κορυφή

πρός τα κάτω, ωστόσο, αυτό είναι off-line και χρειάζεται όλα τα σημεία να είναι γνωστά εκ των προτέρων. Για μια συγκεκριμένη περιοχή του κόμβου, αυτή η μέθοδος ταξινομεί τα σημεία της περιοχής κατά μήκος κάθε διάστασης, και στη συνέχεια για κάθε διάσταση μετά την ταξινόμηση περνά μέσα από την ταξινομημένη λίστα των σημείων δύο φορές, πρώτα από τα αριστερά προς τα δεξιά και στη συνέχεια από δεξιά προς τα αριστερά. Όταν πηγαίνει από αριστερά προς τα δεξιά μέσα από μια λίστα, η μέθοδος υπολογίζει τον όγκο της σφαίρας οριοθέτησης των σημείων, καθώς σαρώνεται (ή με άλλα λόγια, κάθε νέο σημείο εισέρχεται μέσα στη σφαίρα και ο όγκος της υπολογίζεται). Το ίδιο γίνεται όταν κάνουμε το αντίστροφο πέρασμα από τα δεξιά προς τα αριστερά. Αυτά τα δύο περάσματα για μια διάσταση στη συνέχεια δίνουν τον όγκο της σφαίρας οριοθέτησης στις δύο πλευρές της κάθε διαχωρισμένης τοποθεσίας από τα σημεία κατά μήκος αυτής της διάστασης. Η μέθοδος στη συνέχεια χρησιμοποιεί τις δύο σφαίρες που είχαν το ελάχιστο άθροισμα των όγκων, για κάποια διάσταση και την αντίστοιχη τοποθεσία διάσπασης, για να χωρίσει το συγκεκριμένο κόμβο

5. Bottom Up method (Μέθοδος από κάτω προς τα πάνω): Η Omohundro μέθοδος από κάτω προς τα πάνω λειτουργεί παρόμοια με αυτό που έχει περιγραφή προηγουμένως . Διατηρεί μια λίστα που αποτελείται από τα σημεία και τους κόμβους, και επαναληπτικά βρίσκει από τη λίστα, ένα ζευγάρι σημείων, ή κόμβους, ή ένα ζευγάρι που αποτελείται από ένα σημείο και ένα κόμβο, οποιοδήποτε έχει την μικρότερη σφαίρα οριοθέτησης, και δημιουργεί ένα κόμβο για αυτό το ζευγάρι με τη μικρότερη σφαίρα οριοθέτησης. Η μέθοδος στη συνέχεια αντικαθιστά το ζευγάρι για το οποίο ο κόμβος δημιουργήθηκε, με τον κόμβο που δημιουργήθηκε στη λίστα, και επαναλαμβάνεται πάλι για να δημιουργήσει ένα κόμβο με τη μικρότερη σφαίρα οριοθέτησης. Το μέγεθος της σφαίρας μετράτε χρησιμοποιώντας τον όγκο. Ωστόσο, δεδομένου ότι ο όγκος και η ακτίνα μιας σφαίρας αυξάνουν μονοτονικά το ένα με το άλλο, η ακτίνα είναι επαρκής για τη μέτρηση του μεγέθους μιας σφαίρας όπως έχει γίνει από τον Moore στο (Moore, 2000). Μπορεί να παρατηρηθεί ότι η διαδικασία μπορεί να απαιτήσει μέχρι n(n-1)/2 υπολογισμούς απόστασης(για τον υπολογισμό της ακτίνας των σφαιρών οριοθέτησης) για ένα πέρασμα για την κατασκευή ενός κόμβου από ένα ζευγάρι. Ως εκ τούτου, στην χειρότερη περίπτωση, εάν δημιουργούνται 0(n) κόμβοι, η μέθοδος μπορεί να πάρει μέχρι 0(n^3) χρόνο κατασκευής.

Ο Omohundro, όταν παρουσίασε τις παραπάνω μεθόδους κατασκευής στο (Omohundro, 1989), πρότεινε ότι η διάμεσος της ευρύτερης διάστασης παράγει τα καλύτερα δέντρα ποιότητας (όσον αφορά fitting με τη δομή των δεδομένων, το οποίο μετράτε από τον Omohundro με το συνολικό όγκο των σφαιρών στο δέντρο) όταν τα δεδομένα είναι ομοιόμορφα κατανεμημένα. Ωστόσο όταν τα δεδομένα είναι

ομαδοποιημένα ή ανομοιόμορφα, τότε σύμφωνα με τον Omohundro, η από κάτω προς τα πάνω μέθοδος παράγει την καλύτερη ποιότητα δέντρων ακολουθούμενο από τη μέθοδο της ελάχιστης ευρετικής αύξησης όγκου. Διαισθητικά αν μια δομή υιοθετεί καλά την τοπική δομή των δεδομένων, είναι πιο πιθανό να περιορίσουν τη σωστή περιοχή ενός ερωτήματος σημείων και πιο πιθανό να περικόπτουν τις μακρινές περιοχές που είναι απίθανο να περιέχουν ένα Nearest Neighbor.

Ο Omohundro στο (Omohundro, 1989) χρησιμοποίησε σφαίρες αντί για σημεία για την κατασκευή των δέντρων. Έτσι, αντί για ένα σύνολο σημείων, αυτός θεώρησε ένα σύνολο από σφαίρες και κατασκεύασε το δέντρο για αυτό το σύνολο των σφαιρών. Αμέσως αν εφαρμοστεί η ελάχιστη ευρετική αύξησης όγκου του Omohundro αυτή θα παράγει υπερβολικά σκεβρωμένα (skewred) δέντρα. Κάθε κόμβος του δέντρου τεμαχιζόταν με μια σφαίρα που αποτελούνταν από ένα μόνο σημείο και το υπόλοιπο αποτελούνταν από τα υπόλοιπα σημεία. Αυτό συμβαίνει επειδή σε κάθε περίπτωση το ελάχιστο άθροισμα όγκου στις σφαίρες σε (σχεδόν) κάθε περίπτωση εμφανίζοντας μόνο αν μία σφαίρα αποτελούνταν από ένα μόνο σημείο (και ως αποτέλεσμα είχε μηδενικό όγκο) και η υπόλοιπη από τα άλλα σημεία.. Παρόλο που ο Omohudro στην εργασία του, το 1989 χρησιμοποίησε σφαίρες και όχι σημεία, χρησιμοποίησε όμως τις εκφυλισμένες περιπτώσεις όπου οι σφαίρες αποτελούνταν από ένα μόνο σημείο. Η αρχική εκτίμηση της μεθόδου δείχνει ότι η μέθοδος δεν είναι αρκετά αποτελεσματική σε σχέση με άλλες.

Μέθοδος Uhlmann

1.Μέση Απόσταση από τυχαίο σημείο: Αυτή είναι μια μέθοδος κατασκευής από την κορυφή προς τα κάτω. Διαχωρίζει τη σφαίρα του όγκου σε δύο σφαίρες βάσει της μέσης απόστασης ενός τυχαία επιλεγμένου σημείου σε σχέση με όλα τα άλλα σημεία μέσα στη σφαίρα. Ως αποτέλεσμα, αφού επιλέξουμε τυχαία ένα σημείο p από το εσωτερικό της σφαίρας, η μέθοδος υπολογίζει την απόσταση του, από όλα τα άλλα σημεία μέσα στην σφαίρα και έπειτα υπολογίζει τη μέση τιμή m αυτών των αποστάσεων. Η μέθοδος μετά δημιουργεί μία μπάλα για την αριστερή θυγατρική(child) σφαίρα από όλα τα σημεία pi|d(pi,p)<_m, και δημιουργεί μια σφαίρα για την αριστερή θυγατρική σφαίρα από όλα τα σημεία pi|d(pi,p)>m . Το δέντρο μπορεί λοιπόν να κατασκευαστεί με 0(nlogn) υπολογισμούς απόστασης σε 0(dnlogn) χρόνο, κάτι το οποίο είναι απολύτως ισορροπημένο και έχει 0(logn) βάθος.

Οι μέθοδοι του Moore

1. **Σημεία κοντινότερα στο πιο απομακρυσμένο ζεύγος** (Points Closest to Furthest Pair): *Αυτή τη μέθοδο χρησιμοποιήσαμε στην εργασία.* Αυτή είναι επίσης μια μέθοδος κατασκευής Top Down που παρουσιάστηκε από τον Moore το 2000. Διαχωρίζει μια σφαίρα βασισμένη στο ζεύγος σημείων μέσα στην σφαίρα τα οποία έχουν την μεγαλύτερη απόσταση μεταξύ τους. Η εύρεση του πιο απομακρυσμένου ζεύγους σημείων σε μία σφαίρα με n σημεία είναι μια 0(n^2) διαδικασία και ως εκ τούτου η μέθοδος χρησιμοποιεί μια γραμμικού χρόνου **ευρετική** για να βρει κατά προσέγγιση το ζεύγος σημείων με τη μεγαλύτερη απόσταση. Η μέθοδος λειτουργεί με την εύρεση ενός σημείου p1 το οποίο είναι το πιο απομακρυσμένο από το κέντρο της σφαίρας ενός δεδομένου κόμβου. Έπειτα υπολογίζει την απόσταση του σημείου p1 από όλα τα σημεία μέσα στην σφαίρα ώστε να βρει το σημείο p2 που είναι το πιο απομακρυσμένο από το p1. Η μέθοδος έπειτα δημιουργεί μια σφαίρα για την αριστερή θυγατρική σφαίρα από τα σημεία που είναι πιο κοντινά στο p1 (σε σχέση με το p2) και δημιουργεί μια σφαίρα για τη δεξιά θυγατρική σφαίρα από σημεία που είναι πιο κοντινά στο p2. Η μέθοδος έχει το χαρακτηριστικό ότι ταιριάζει πολύ καλά με τη δομή των στοιχείων δεδομένων. Από όλες τις μεθόδους, μόνο η από κάτω προς τα πάνω κατασκευή μερικές φορές ξεπερνά αυτή τη μέθοδο σε αυτό το χαρακτηριστικό.

2. Middle Out μέθοδος: Επίσης παρουσιάζεται στο Moore(2000). Η μέθοδος αυτή επινοήθηκε ειδικά για να επιταχύνει τον αλγόριθμο ομαδοποίησης KMeans με τη χρήση Μετρικών Δέντρων. Η μέθοδος λειτουργεί με τη χρήση της ιεραρχικής μεθόδου Anchors Hierarchy (που παρουσιάζεται από το Moore,2000)) ώστε πρώτα να βρεί sqrt(n) ομάδες και να δημιουργήσει σφαίρες από αυτά. Η διαδικασία είναι όμοια με τη μέθοδο farthest point clustering [Gonraler 1985]. Έπειτα χρησιμοποιεί τη μέθοδο από κάτω προς τα πάνω για μα συγχωνεύσει αυτές τις σφαίρες σε μία κορυφαία σφαίρα και μετά αναδρομικά εφαρμόζει την ιεραρχία Anchors και την από κάτω προς τα πάνω μέθοδο για κάθε μία από τις sqrt(n) σφαίρες στις συστάδες που αρχικά δημιουργήθηκαν.

Η μέθοδος Anchors Hierarchy λειτουργεί με την επιλογή ενός τυχαίου σημείου που ονομάζεται anchor a1, από τα σημεία και με τη μετέπειτα δημιουργία μιας σφαίρας με επίκεντρο το a1 που να περιέχει όλα αυτά τα σημεία. Η μέθοδος προχωρά έπειτα βρίσκοντας το επόμενο anchor a2 που είναι το πιο απομακρυσμένο από το a1 και δημιουργεί άλλη μια σφαίρα για το a2 προσδιορίζοντας την με όλα εκείνα τα σημεία που είναι πιο κοντινά στο a2 σε σχέση με το a1. Η μέθοδος μετά επαναληπτικά επιλέγει το επόμενο ai που είναι το πιο απομακρυσμένο σημείο στην μεγαλύτερη σφαίρα και δημιουργεί μια σφαίρα προσδιορίζοντας την με όλα εκείνα τα σημεία που είναι πιο κοντινά στο ai σε σχέση με οποιοδήποτε άλλο anchor. Η επανάληψη συνεχίζεται μέχρι να δημιουργηθούν οι sqrt(n) μπάλες για ένα δεδομένο σύνολο n σημείων.

Λεπτομέρειες των δομών

Για τα δέντρα το κέντρο μιας σφαίρας θεωρείται συνήθως η κεντροειδής των σημείων, που είναι επίσης το κέντρο των σημείων γεωμετρικά και που χρησιμοποιήθηκε από το Moore (Moore,2000). Άρα αν c είναι το κέντρο μιας μπάλας που περιέχει n σημεία και αν το cj είναι το jth κομμάτι του κέντρου και το xij το jth κομμάτι ενός σημείου xi μέσα στην σφαίρα τότε το κάθε cj υπολογίζεται ως εξής:

Η ακτίνα κάθε σφαίρας τότε θεωρείται η απόσταση του πιο απομακρυσμένου σημείου από το κέντρο c.

Επιπλέον, τόσο ο Omohudro όσο και ο Uhlmann στις παρουσιάσεις τους των Μετρικών Δέντρων, σε αντίθεση με τον Moore, δεν χρησιμοποίησαν την έννοια του μέγιστου μεγέθους φύλλου, με άλλα λόγια δεν σταμάτησαν την επαναληπτική κατασκευή δέντρου όταν ο αριθμός των σημείων σε μια σφαίρα ήταν μικρότερος από κάποιο δεδομένο επίπεδο. Παρόλα αυτά, αφού ο Omohudro χρησιμοποίησε την έννοια των σφαιρών και όχι των σημείων υπονόησε ότι υπάρχει μέγιστο μέγεθος φύλλου 1. Συνήθως, για κάθε μέθοδο η επαναληπτική κατασκευή σταματά αν ο αριθμός των σημείων σε μία σφαίρα είναι μικρότερος από (ή ίσος) ένα προαποφασισμένο από το χρήστη επίπεδο.

ΛΕΠΤΟΜΕΡΕΙΕΣ ΕΦΑΡΜΟΓΗΣ

Τα Μετρικά Δέντρα, υλοποιούνται συνήθως είτε όπως τα binary trees είτε με τη χρήση πινάκων (Arrays) από indices για τα σημεία. Αναδιατάσσεται σε συμφωνία με τη διαδικασία δόμησης του δέντρου και κάθε κόμβος αποθηκεύει το index αρχής και τέλους από τον πίνακα για το τμήμα που εκχωρείται στην περιοχή του. Τα σημεία μέσα σε ένα κόμβο περιλαμβάνονται στο τμήμα της παράταξης που ορίζεται από την ένδειξη αρχής και τέλους που αποθηκεύεται μέσα στον κόμβο. Κάθε φορά που ένας κόμβος τεμαχίζεται, τα σημεία που ανήκουν στην αριστερή θυγατρική σφαίρα μετακινούνται στα αριστερά του τμήματος του κόμβου του πίνακα και τα σημεία που ανήκουν στη δεξιά θυγατρική σφαίρα μετακινούνται στα δεξιά του τμήματος του κόμβου τον πίνακα. Η ενδείξεις αρχής και τέλους για την αριστερή και δεξιά θυγατρική σφαίρα του κόμβου τίθενται αντίστοιχα ώστε να δείχνουν τα αντίστοιχα υπο-τμήματα στο τμήμα κόμβου της πίνακα.

Όπως τα KDTrees, τα kNNs κατά τη διάρκεια KNN αναζήτησης στα Μετρικά Δέντρα αποθηκεύονται στη ουρά προτεραιότητας (PQ). Όπως και στα KDTrees, ουρά προτεραιότητας αρχικοποιείται με k κενά αντικείμενα και +00 αποστάσεις, ώστε η σφαίρα αναζήτησης διασταυρώνεται με όλες τις περιοχές (καθώς η ακτίνα της σφαίρας αναζήτησης θεωρείται η Kth μεγαλύτερη απόσταση στην κορυφή της σειράς προτεραιότητας) και σίγουρα θα βρούμε k γείτονες σε περίπτωση που η παράμετρος μέγιστου μεγέθους φύλλου είναι μικρότερη από k. Αφού βάλουμε στην ουρά

προτεραιότητας τους πρώτους k γείτονες που βρέθηκαν στο πρώτο φύλο της αναζήτησης (ή τους πρώτους λίγους αν το μέγιστο μέγεθος φύλου > k), ουρά προτεραιότητας τότε ενημερώνεται μόνο αν ένας καλύτερος kth NN συναντηθεί σε κάποια άλλη περιοχή φύλου κατά τη διάρκεια της αναζήτησης.

Όμοια με τα KDTrees, συνήθως δίνεται προσοχή στη βελτιστοποίηση των metric trees όσο είναι δυνατόν. Μόνο τετράγωνες αποστάσεις υπολογίζονται D=|x-y|^2 κατά τη διάρκεια της kNN αναζήτησης για να αποφεύγουμε την τετραγωνική ρίζα sqrt(D) που κοστίζει. Η sqrt(D) αποστάσεων χρησιμοποιείται μόνο για τα kNNs που παραμένουν στο τέλος της έρευνας. Όμως αντίθετα με τα KDTrees, για κλάδεμα κόμβων, οι κανονικές Ευκλείδειες αποστάσεις sqrt(D)υπολογίζονται. Αυτό επειδή τα κριτήρια κλαδέματος στα Μετρικά Δέντρα περιλαμβάνουν αποστάσεις μετρημένες από 2 διαφορετικά σημεία (r και rc), και ως εκ τούτου μονότονη σχέση μεταξύ τετραγώνων και μη τετραγώνων αποστάσεων δεν υπάρχει ακριβώς.

ΑΝΑΖΗΤΗΣΗ ΚΟΝΤΙΝΩΝ ΓΕΙΤΟΝΩΝ ΣΤΟ BALL TREE

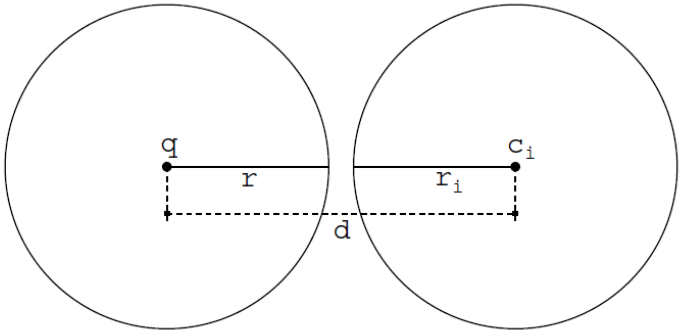
Η περιγραφή των Μετρικών Δέντρων έλαβε μόνο υπόψη του αναζητήσεις περιοχής (σημεία μέσα σε συγκεκριμένη απόσταση r από το σημείο αναζήτηση) και δεν παρείχε καμιά διαδικασία για την KNN αναζήτηση. Ο Uhlmann και ο Moore όμως έλαβαν και οι δύο υπόψη την KNN αναζήτηση και παρείχαν ικανές διαδικασίες οι οποίες ήταν ουσιαστικά ίδιες.

Ο Moore και η ομάδα του παρείχαν τη διαδικασία τους για KNN αναζήτηση για Μετρικά Δέντρα. Η διαδικασία τους λειτουργεί ακριβώς με τον ίδιο τρόπο που περιγράφεται στο εισαγωγικό τμήμα για τα βασικά ερωτήματα. Το κριτήριο για την ανίχνευση της διασταύρωσης (intersection) της σφαίρας αναζήτησης με τη σφαίρα κόμβου, που η διαδικασία τους χρησιμοποιεί για να κλαδευτούν κόμβοι που δεν διασταυρώνονται με τη σφαίρα αναζήτησης, έχει ως εξής:

Ας οριστεί ci το κέντρο της σφαίρας κάποιου κόμβου i σε σχέση με τον οποίο πρέπει να ανιχνεύσουμε την διασταύρωση και ας οριστεί ri η ακτίνα του. Ας είναι r η ακτίνα της σφαίρας αναζήτησης, ίση με την απόσταση των καλύτερων KthNN που συναντάμε. Η σφαίρα ερωτήματος δε διασταυρώνεται με τη σφαίρα κόμβου αν

r < |q-ci| - ri,

και ο κόμβος μπορεί έτσι να κλαδευτεί(prune). Παρόλα αυτά αν r>_|q – ci| - ri, τότε υπάρχει μια διασταύρωση και θα χρειαζόταν να ψάξουμε μέσα στον κόμβο για να είμαστε σίγουροι ότι βρήκαμε τους ακριβείς kNNs(k Nearest Neighbors). Αυτό το κριτήριο κλαδέματος παριστάνεται γραφικά με την εικόνα



Εικόνα: κριτήριο κλαδέματος Μετρικών Δέντρων. Ο κόμβος που αντιστοιχεί στη σφαίρα ci κλαδεύεται μόνο αν r < d – ri .

ΑΛΓΟΡΙΘΜΟΙ ΑΝΑΖΗΤΗΣΗΣ ΤΟΥ OMOHUNDRO

Στην εργασία (Omohundro,1989) περιγράφεται αναλυτικά ότι υπάρχουν αντικείμενα για balls, balltrees και κόμβους balltree. Τα αντικείμενα “BALL” αποτελούνται από ένα διάνυσμα “ctr” το οποίο συγκρατεί το κέντρο της σφαίρας και από μια πραγματική τιμή “r” που περιλαμβάνει την ακτίνα. Τα αντικείμενα “BLT_ND” αποτελούνται από μια σφαίρα “bl”, και δείκτες “par, lt,rt” προς τους γονείς και τα παιδιά των κόμβων. Τα αντικείμενα “BALLTREE” έχουν έναν δείκτη “tree” προς το δέντρο που βρίσκεται από κάτω και μια ποικιλία άλλων μεταβλητών για επαύξηση ανακτήσεων (όπως τοπικές ουρές προτεραιοτήτων).

Ερωτήματα που χρησιμοποιούν Simple Pruning

Υπάρχουν δύο ευρύτερες τάξεις ερωτημάτων οι οποίες υποστηρίζονται αποτελεσματικά από τη δομή ball tree. Η πρώτη τάξη χρησιμοποιεί μια έρευνα με απλό κλάδεμα των μη-σχετικών κλαδιών Αυτή τη μέθοδο χρησιμοποιήσαμε στην εργασία.

Η δεύτερη κατηγορία απαιτεί την χρήση branch and bound κατά τη διάρκεια της έρευνας. Αυτός ο τομέας παρουσιάζει κάποια απλά ερωτήματα κλαδέματος και κατόπιν δίνει παραδείγματα της πιο σύνθετης ποικιλίας.

Με δοσμένη μια σφαίρα ερωτήματος, ίσως απαιτήσουμε μια λίστα όλων των σφαιρών φύλλων οι οποίες περιέχουν τη σφαίρα της ερώτησης. Αυτό το εφαρμόζουμε ως αναδρομική αναζήτηση προς τα κάτω του balltree στο οποίο κλαδεύουμε και αποκρίνουμε τις επαναλαμβανόμενες calls από τους εσωτερικούς κόμβους των οποίων οι σφαίρες δεν εμπεριέχουν τη σφαίρα ερώτησης. Εάν η σφαίρα ενός εσωτερικού κόμβου δεν εμπεριέχει τη σφαίρα ερώτησης, είναι αδύνατο για οποιαδήποτε σφαίρα φύλλων από κάτω του να εμπεριέχει τη σφαίρα ερώτησης επίσης. Στην τάξη κόμβων του balltree “BLT_ND” μπορούμε να ορίσουμε:

```
push_leaves_containing_ball(b:BALL,l:LLIST[BLT_ND]) is
      -- Add the leaves under Current which contain b to l.
   do
      if leaf then
         if bl.contains_ball(b) then l.push(Current) end
      else
         if bl.contains_ball(b) then
            lt.push_leaves_containing_ball(b,l);
            rt.push_leaves_containing_ball(b,l);
         end; -- if
      end; -- if
   end;
```

Ομοίως, μπορούμε να ζητήσουμε όλα τα φύλλα σφαιρών τα οποία τέμνουν μια σφαίρα ερωτήματος. Κόβουμε τις αναδρομικές κλησείς από τους εσωτερικούς κόμβους των οποίων η σφαίρα δεν τέμνει τη σφαίρα ερωτήματος.

```
push_leaves_intersecting_ball(b:BALL,l:LLIST[BLT_ND]) is
      -- Add the leaves under Current which intersect b to l.
   do
      if leaf then
         if bl.intersects_ball(b) then l.push(Current) end
      else
         if bl.intersects_ball(b) then
            lt.push_leaves_intersecting_ball(b,l);
            rt.push_leaves_intersecting_ball(b,l);
         end; -- if
      end; -- if
   end;
```

Τελικά, μπορούμε να ζητήσουμε όλες τις σφαίρες φύλλα που περιέχονται στη σφαίρα ερώτησης. Εδώ πρέπει να συνεχίσουμε να ερευνούμε κάτω από τον εσωτερικό κόμβο του οποίου η σφαίρα τέμνει τη σφαίρα ερωτήματος διότι ίσως κάποια σφαίρα να εμπεριέχεται σαυτόν.

```
push_leaves_contained_in_ball(b:BALL,l:LLIST[BLT_ND]) is
      -- Add the leaves under Current which are contained in b to l.
   do
      if leaf then
         if b.contains_ball(bl) then l.push(Current) end
      else
         if bl.intersects_ball(b) then
            lt.push_leaves_contained_in_ball(b,l);

         rt.push_leaves_contained_in_ball(b,l);
      end; -- if
   end; -- if
end;
```

Ένα σημείο είναι απλά μια εκφυλισμένη σφαίρα. Δύο σημαντικές ιδιαίτερες περιπτώσεις αυτών των ερωτημάτων στις οποίες κάποια από τις σφαίρες είναι σημεία είναι η εργασία της επιστροφής όλων των σφαιρών φύλλων οι οποίες περιέχουν ένα δοσμένο σημείο και η απόδοση όλων τα σημείων φύλλων τα οποία εμπεριέχονται σε μια δοθείσα σφαίρα.

Ερωτήματα που χρησιμοποιούν Branch and Bound

Τα πιο σύνθετα ερωτήματα απαιτούν αναζήτηση Branch and Bound. Ένα σημαντικό παράδειγμα για τα balltrees των οποίων τα φύλλα συγκρατούν σημεία είναι να επανακτηθούν τα m κοντινότερα σημεία ενός σημείου ερώτησης. Χρησιμοποιώντας balltrees μπορούμε να χρησιμοποιήσουμε μια παρόμοια προσέγγιση με εκείνη που διαπραγματεύεται η εργασία [Friedman, et. al., 1977] για τα k-d trees. Ερευνούμε πάλι αναδρομικά το δέντρο. Καθ'όλη τη διάρκεια της έρευνας διατηρούμε την πιο μικρή σφαίρα, στο κέντρο του σημείου ερώτησης το οποίο περιέχει τα m κοντινότερα σημεία φύλλων μέχρι τώρα στην έρευνα. Σε αυτόν τον αλγόριθμο κόβουμε αναδρομικές έρευνες οι οποίες αρχίζουν σε εσωτερικούς κόμβους των οποίων η σφαίρα δεν τέμνει την bball. Αυτό το κλάδεμα είναι πιθανό να γίνει πιο αποτελεσματικά εάν σε κάθε εσωτερικό κόμβο αναζητούμε πρώτα το παιδί το οποίο βρίσκεται κοντινότερα στο σημείο ερώτησης και κατόπιν το άλλο παιδί. Εξαιτίας του ότι οι περιοχές των κόμβων είναι πιο σφιχτές γύρω από τα σημεία δειγμάτων, ωστόσο, τα balltree ενδέχεται να μπορούν να κόβουν κόμβους σε περιπτώσεις όπου ένα k-d tree δεν μπορεί.

Για να επανακτήσουμε τους κοντινότερους γείτονες του m, τηρούμε μια ουρά προτεραιότητας απο τα καλύτερα φύλλα που δεί ταξινομημένα κατά την απόσταση από το σημείο ερωτήματος. Θα δείξουμε μόνο τη συνάρτηση για την εύρεση του κοντινότερου γείτονα, αλλά η επέκταση προς τους m κοντινότερους γείτονες θα πρέπει να είναι ξεκάθαρη.

Σε αυτή την περίπτωση δεχόμαστε ότι η αναζήτηση nn ορίζεται σε μια κλάση η οποία έχει "bball" σαν χαρακτηριστικό γνώρισμα και ότι το κέντρο της "bball.r" έχει τεθεί στο σημείο ερώτησης και η ακτίνα του "bball.r" σε μια αρκετά μεγάλη τιμή η οποία περιέχει τη σφαίρα ρίζας. Το "nn" είναι ακόμα ένα χαρακτηριστικό γνώρισμα το οποίο θα κρατά το αποτέλεσμα όταν επιστρέφει η διαδικασία. "near_dist_to_vector" είναι η διαδικασία στις τάξεις BALL η οποία ανταποδίδει την απόσταση από το κοντινότερο σημείο στη σφαίρα προς ένα δεδομένο διάνυσμα.

```
nn_search(n:T) is
    -- Replace nn by closest leaf under n to bball.ctr, if closer than bball.r
  local d,ld,rd:REAL;
  do
    if n.leaf then
      d:=bball.ctr.dist_to_vector(n.bl.ctr);
      if d < bball.r then bball.set_r(d); nn:=n end; -- reset best
    else -- at internal node
      ld:=n.lt.bl.near_dist_to_vector(bball.ctr);
      rd:=n.rt.bl.near_dist_to_vector(bball.ctr);
      if ld > bball.r and rd > bball.r then -- no sense looking here
      else
```

```
        if ld<=rd then -- search nearer node first
            nn_search(n.lt);
            if rd < bball.r then nn_search(n.rt) end; -- check if still worth searching
        else
            nn_search(n.rt);
            if ld < bball.r then nn_search(n.lt) end;  -- check if still worth searching
        end; -- if
    end; -- if
  end; -- if
end;
```

Υπάρχουν αρκετές φυσικές γενικεύσεις αυτής αυτού του ερωτήματος προς εκείνα που εμμπλέκουν σφαίρες. Η απόσταση μεταξύ σημείων μπορεί να αντικατασταθεί με απόσταση μεταξύ των κέντρων των σφαιρών, ελάχιστη απόσταση μεταξύ των σφαιρών, ή μέγιστη απόσταση μεταξύ των σφαιρών. Σε κάθε περίπτωση η ελάχιστη απόσταση προς τον πρόγονο κόμβο είναι ένα κατώτερο όριο επί της απόστασης προς το φύλλο και έτσι μπορεί να είναι ακριβώς όπως επάνω για να αποκόψει την έρευνα.

Κεφάλαιο 6 ΠΕΡΙΓΡΑΦΗ ΠΕΙΡΑΜΑΤΩΝ ΚΑΙ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Στα παρακάτω πειράματα μελετήθηκε ο αλγόριθμός κ κοντινότερων γειτόνων(k-nn) , σε διάφορα δεδομένα από το UCI Repository of Machine Lerning Databases. Το σημαντικότερο στοιχείο για την λειτουργία του k-nn είναι οι διαφορετικές τημες του k, δηλαδή, πόσοι κοντινότεροι γείτονες επηρεάζουν την κατηγοριοποίηση του νέου στοιχειού. Τα πειράματα θα γίνουν με k = [1, 2, 3,...,19, 20].

Diabetes Pima Indians

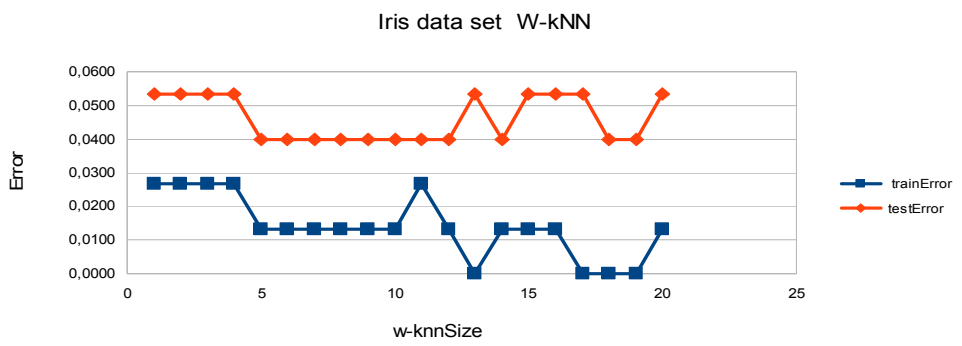
Το σύνολο δεδομένων Pima Indians Diabetes Database από το National Institute of Diabetes and Digestive Diseases και το Applied Physics Laboratory του πανεπιστημίου Johns Hopkins, για τη διάγνωση διαβήτη σε ασθενείς σύμφωνα με τα κριτήρια του Παγκόσμιου Οργανισμού Υγείας. Ο πληθυσμός των ασθενών αφορά άνδρες και γυναίκες τουλάχιστον 21 ετών με καταγωγή από τους Pima Indian, που ζουν κοντά στο Phoenix της Arizona. Περιέχει 768 παραδείγματα με 8 χαρακτηριστικά γνωρίσματα και 268 θετικές και 500 αρνητικές διαγνώσεις/ κλάσεις. Στον πίνακα καθώς που ακολουθεί παρατηρούμε το train error, το test error του σύνολο δεδομένων Diabetes, τις διάφορες τιμές που παίρνει το k, αρχίζοντας από την τιμή 1 μέχρι την τιμή 20 και βλέπουμετα βέλτιστα k για τον απλό KNN και τον σταθμισμένο (weighted) KNN.

	Simple KNN			Weighted KNN	
knnSize	trainError	testError	w-knnSize	trainError	testError
1	0,2995	0,3021	1	0,3073	0,2891
2	0,2943	0,2917	2	0,3073	0,2891
3	0,2734	0,2969	3	0,2604	0,2630
4	0,2891	0,2917	4	0,2734	0,2630
5	0,2656	0,3099	5	0,2500	0,2682
6	0,2891	0,2891	6	0,2500	0,2630
7	0,2708	0,2839	7	0,2734	0,2734
8	0,276	0,2865	8	0,2656	0,2630
9	0,2839	0,2839	9	0,2943	0,2682
10	0,2839	0,2812	10	0,2813	0,2708
11	0,2812	0,2865	11	0,2734	0,2760
12	0,2995	0,2943	12	0,2604	0,2708
13	0,2865	0,2760	13	0,2630	0,2656
14	0,2995	0,2734	14	0,2526	0,2630
15	0,2995	0,2760	**15**	**0,2578**	**0,2604**
16	0,3021	0,2656	16	0,2396	0,2604
17	0,2865	0,2708	17	0,2552	0,2656
18	**0,2891**	**0,2630**	18	0,2500	0,2630
19	0,2943	0,2930	19	0,2578	0,2656
20	0,2917	0,2552	20	0,2500	0,2604

Iris data set

Το σύνολο δεδομένων Iris περιέχει 3 κατηγορίες 50 παραδειγμάτων, όπου κάθε κατηγορία αναφέρεται σε έναν τύπο λουλουδιού iris (κρίνος). Μια κατηγορία είναι γραμμικά διαχωρίσιμη από τις άλλες 2. Οι 2 τελευταίες δεν είναι γραμμικά διαχωρίσιμες η μία από την άλλη. Στον πίνακα καθώς και στην γραφική παράσταση που ακολουθεί παρατηρούμε το train error, το test error του σύνολο δεδομένων Iris, στις διάφορες τιμές που παίρνει το k, αρχίζοντας από την τιμή 1 μέχρι την τιμή 20 και παρατηρούμε πως το βέλτιστο k είναι k = 17 για τον απλό KNN με αντίστοιχο test error : 0.0533, και για τον σταθμισμένο (weighted) KNN είναι k=5, που δίνει και το ελάχιστο σφάλμα.

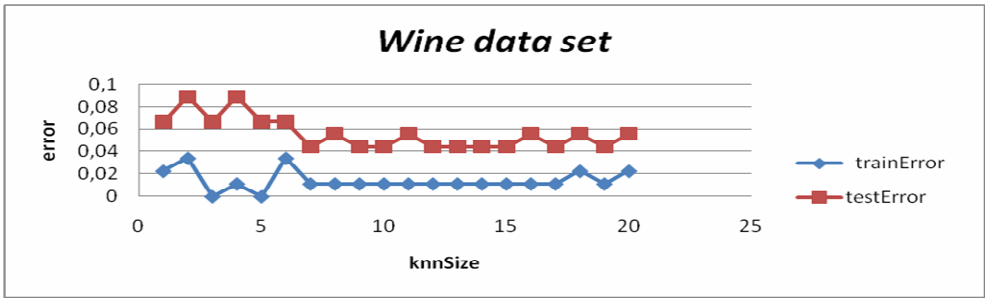
	Simple KNN			Weighted KNN	
knnSize	trainError	testError	w-knnSize	trainError	testError
1	0,0133	0,0667	1	0,0267	0,0533
2	0,0133	0,0667	2	0,0267	0,0533
3	0,0267	0,0667	3	0,0267	0,0533
4	0,0133	0,0667	4	0,0267	0,0533
5	0,0267	0,0800	**5**	**0,0133**	**0,0400**
6	0,0133	0,0667	6	0,0133	0,0400
7	0,0133	0,0800	7	0,0133	0,0400
8	0,0133	0,0800	8	0,0133	0,0400
9	0,0133	0,0667	9	0,0133	0,0400
10	0,0133	0,0667	10	0,0133	0,0400
11	0,0133	0,0667	11	0,0267	0,0400
12	0,0133	0,0667	12	0,0133	0,0400
13	0,0133	0,0667	13	0,0000	0,0533
14	0,0133	0,0667	14	0,0133	0,0400
15	0,0133	0,0667	15	0,0133	0,0533
16	0,0133	0,0800	16	0,0133	0,0533
17	**0,0133**	**0,0533**	17	0,0000	0,0533
18	0,0133	0,0667	18	0,0000	0,0400
19	0,0133	0,0667	19	0,0000	0,0400
20	0,0267	0,0533	20	0,0133	0,0533



Wine data set

Το σύνολο δεδομένων Wine περιέχει τη χημική ανάλυση 178 κρασιών που καλλιεργούνται στην ίδια περιοχή στην Ιταλία αλλά που προέρχονται από τρεις διαφορετικές ποικιλίες. Το πρόβλημα είναι να διακριθούν οι τρεις τύποι βασισμένοι σε 13 συνεχείς ιδιότητες που προέρχονται από τη χημική ανάλυση. Στον πίνακα καθώς και στην γραφική παράσταση που ακολουθεί παρατηρούμε το train error, το test error του σύνολο δεδομένων Wine, τις διάφορες τιμές που παίρνει το k, αρχίζοντας από την τιμή 1 μέχρι την τιμή 20 με ενδιάμεσο βήμα 1 και παρατηρούμε πως το βέλτιστο k δηλαδή η τιμή για την ελαχιστοποίηση του σφάλματος ταξινόμησης είναι k = 7 με αντίστοιχο test error : 0.0444.

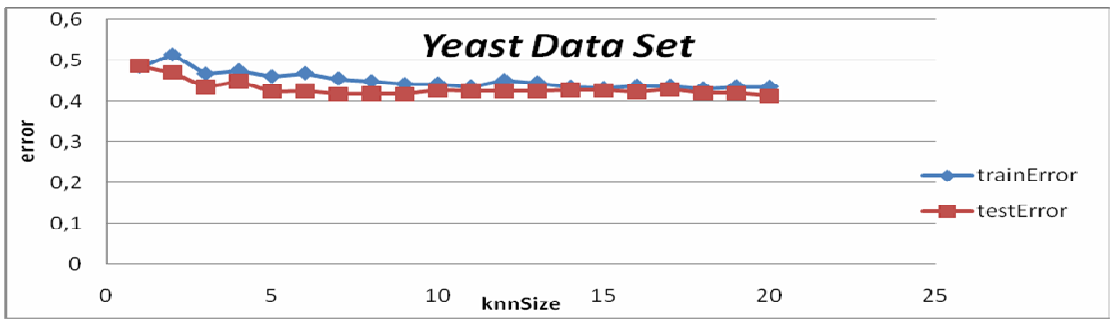
	Simple KNN			Weighted KNN	
knnSize	trainError	testError	w-knnSize	trainError	testError
1	0,0227	0,0667	1	0,0227	0,0778
2	0,0341	0,0889	2	0,0227	0,0778
3	0	0,0667	3	0,0455	0,0889
4	0,0114	0,0889	4	0,0227	0,0889
5	0	0,0667	5	0,0341	0,1000
6	0,0341	0,0667	6	0,0227	0,1000
7	**0,0114**	**0,0444**	7	0,0568	0,1111
8	0,0114	0,0556	8	0,0341	0,1000
9	0,0114	0,0444	9	0,0568	0,0889
10	0,0114	0,0444	10	0,0568	0,0778
11	0,0114	0,0556	11	0,0568	0,0889
12	0,0114	0,0444	**12**	**0,0568**	**0,0667**
13	0,0114	0,0444	13	0,0455	0,0667
14	0,0114	0,0444	14	0,0568	0,0667
15	0,0114	0,0444	15	0,0568	0,0778
16	0,0114	0,0556	16	0,0568	0,0667
17	0,0114	0,0444	17	0,0455	0,0667
18	0,0227	0,0556	18	0,0455	0,0667
19	0,0114	0,0444	19	0,0455	0,0667
20	0,0227	0,0556	20	0,0455	0,0667



Yeast Data Set

Το σύνολο Yeast αποτελείται από 1484 παραδείγματα με 8 χαρακτηριστικά γνωρίσματα μετρήσεων και αναλύσεων σημάτων για περιοχές εντοπισμού πρωτεϊνών. Στον πίνακα καθώς και στην γραφική παράσταση που ακολουθεί παρατηρούμε το train error, το test error του σύνολο δεδομένων Yeast, τις διάφορες τιμές που παίρνει το k, αρχίζοντας από την τιμή 1 μέχρι την τιμή 20 με ενδιάμεσο βήμα 1 και παρατηρούμε πως το βέλτιστο k δηλαδή η τιμή για την ελαχιστοποίηση του σφάλματος ταξινόμησης είναι k = 20 με αντίστοιχο test error : 0.4121.

	Simple KNN			Weighted KNN	
knnSize	trainError	testError	w-knnSize	trainError	testError
1	0,4844	0,4859	1	0,4844	0,4859
2	0,5142	0,4698	2	0,4844	0,4859
3	0,4668	0,4336	3	0,4709	0,4416
4	0,475	0,4483	4	0,4614	0,4362
5	0,4601	0,4228	5	0,4520	0,4094
6	0,4682	0,4242	6	0,4493	0,4228
7	0,4547	0,4161	7	0,4384	0,4107
8	0,4479	0,4174	8	0,4344	0,4121
9	0,4411	0,4161	9	0,4303	0,4148
10	0,4425	0,4255	10	0,4222	0,4134
11	0,4357	0,4242	11	0,4208	0,4188
12	0,4506	0,4242	12	0,4398	0,4174
13	0,4452	0,4242	13	0,4330	0,4148
14	0,4344	0,4268	14	0,4195	0,4188
15	0,4317	0,4255	15	0,4263	0,4174
16	0,4371	0,4215	16	0,4208	0,4242
17	0,4371	0,4282	17	0,4195	0,4148
18	0,4303	0,4188	18	0,4249	0,4094
19	0,4357	0,4188	19	0,4276	0,4081
20	**0,4344**	**0,4121**	**20**	**0,4249**	**0,4067**

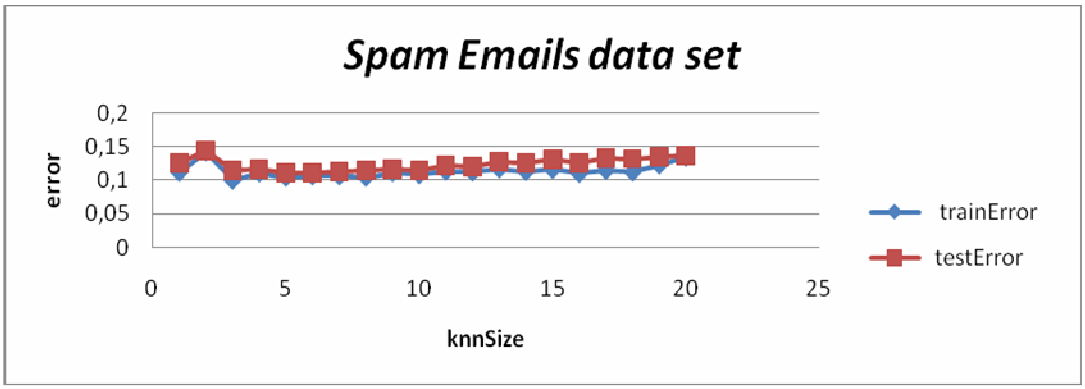


Spam Emails data set

Το σύνολο δεδομένων Spam Emails αποτελείται από 4601 παραδείγματα με 57 χαρακτηριστικά γνωρίσματα μετρήσεων και αναλύσεων Email. Ο στόχος είναι να διακρίνει τις κλάσεις μεταξύ των Spam Emails και των Emails.

Στον πίνακα καθώς και στην γραφική παράσταση που ακολουθεί παρατηρούμε το train error, το test error του σύνολο δεδομένων Spam Emails, τις διάφορες τιμές που παίρνει το k, αρχίζοντας από την τιμή 1 μέχρι την τιμή 20 και παρατηρούμε πως το βέλτιστο k δηλαδή η τιμή για την ελαχιστοποίηση του σφάλματος ταξινόμησης για τον απλό KNN είναι k = 5 με αντίστοιχο test error : 0.1113, και για τον Weighted KNN είναι k=6 με test error = 0.1091.

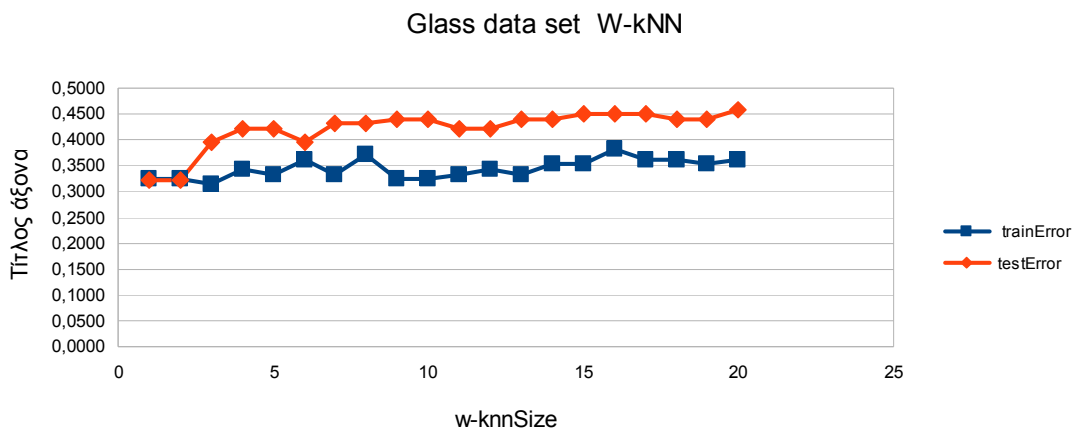
	Simple KNN			Weighted KNN	
knnSize	trainError	TestError	w-knnSize	trainError	testError
1	0,1122	0,1256	1	0,1122	0,1256
2	0,1422	0,1439	2	0,1122	0,1256
3	0,1004	0,1156	3	0,0996	0,1156
4	0,1113	0,1169	4	0,0952	0,1121
5	**0,1048**	**0,1113**	5	0,1048	0,1113
6	0,107	0,1117	**6**	**0,1004**	**0,1091**
7	0,107	0,113	7	0,1074	0,1130
8	0,1048	0,1143	8	0,0987	0,1100
9	0,1117	0,1169	9	0,1113	0,1169
10	0,1087	0,1143	10	0,1043	0,1126
11	0,1139	0,1221	11	0,1139	0,1221
12	0,1126	0,1199	12	0,1065	0,1178
13	0,1174	0,1282	13	0,1148	0,1273
14	0,1135	0,1256	14	0,1096	0,1226
15	0,1165	0,1312	15	0,1139	0,1291
16	0,1104	0,1265	16	0,1052	0,1252
17	0,1152	0,1334	17	0,1126	0,1312
18	0,1126	0,1321	18	0,1083	0,1304
19	0,123	0,1352	19	0,1226	0,1330
20	0,1342	0,1373	20	0,1170	0,1260



Glass data set

Το σύνολο δεδομένων Glass αποτελείται από 214 παραδείγματα με 9 χαρακτηριστικά γνωρίσματα μετρήσεων Στον πίνακα καθώς και στην γραφική παράσταση που ακολουθεί παρατηρούμε το train error, το test error του σύνολο δεδομένων Glass, τις διάφορες τιμές που παίρνει το k, αρχίζοντας από την τιμή 1 μέχρι την τιμή 20 με ενδιάμεσο βήμα 1 και παρατηρούμε πως το βέλτιστο k δηλαδή η τιμή για την ελαχιστοποίηση του σφάλματος ταξινόμησης είναι k = 1, και για τον απλό KNN και για τον σταθμισμένο (weighted) KNN

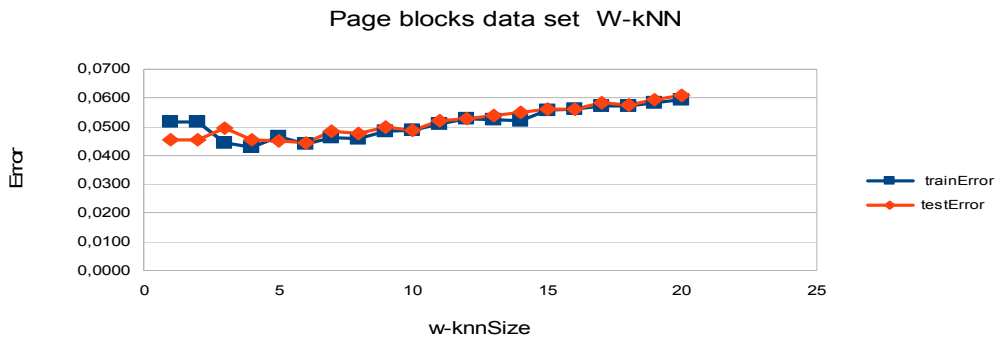
	Simple KNN			Weighted KNN	
knnSize	trainError	testError	w-knnSize	trainError	testError
1	**0,3238**	**0,3211**	**1**	**0,3238**	**0,3211**
2	0,3619	0,3853	2	0,3238	0,3211
3	0,3714	0,4312	3	0,3143	0,3945
4	0,3524	0,4279	4	0,3429	0,4220
5	0,3619	0,4404	5	0,3333	0,4220
6	0,3714	0,4679	6	0,3619	0,3945
7	0,3714	0,4312	7	0,3333	0,4312
8	0,3524	0,4587	8	0,3714	0,4312
9	0,3333	0,4587	9	0,3238	0,4404
10	0,3143	0,4587	10	0,3238	0,4404
11	0,3143	0,4495	11	0,3333	0,4220
12	0,3429	0,4587	12	0,3429	0,4220
13	0,3429	0,4404	13	0,3333	0,4404
14	0,3333	0,4771	14	0,3524	0,4404
15	0,3619	0,4771	15	0,3524	0,4495
16	0,3619	0,4679	16	0,3810	0,4495
17	0,3614	0,4879	17	0,3619	0,4495
18	0,3524	0,4862	18	0,3619	0,4404
19	0,3619	0,4679	19	0,3524	0,4404
20	0,3619	0,4862	20	0,3619	0,4587



Page blocks data set

Το σύνολο δεδομένων Page blocks αποτελείται από 4601 παραδείγματα με 57 χαρακτηριστικά γνωρίσματα Στον πίνακα καθώς και στην γραφική παράσταση που ακολουθεί παρατηρούμε το train error, το test error του σύνολο δεδομένων Page blocks, τις διάφορες τιμές που παίρνει το k, αρχίζοντας από την τιμή 1 μέχρι την τιμή 20 με ενδιάμεσο βήμα 1 και παρατηρούμε πως το βέλτιστο k δηλαδή η τιμή για την ελαχιστοποίηση του σφάλματος ταξινόμησης είναι k = 5 με αντίστοιχο test error : 0.0446, για τον απλό KNN και k=6 για τον σταθμισμένο (weighted) KNN.

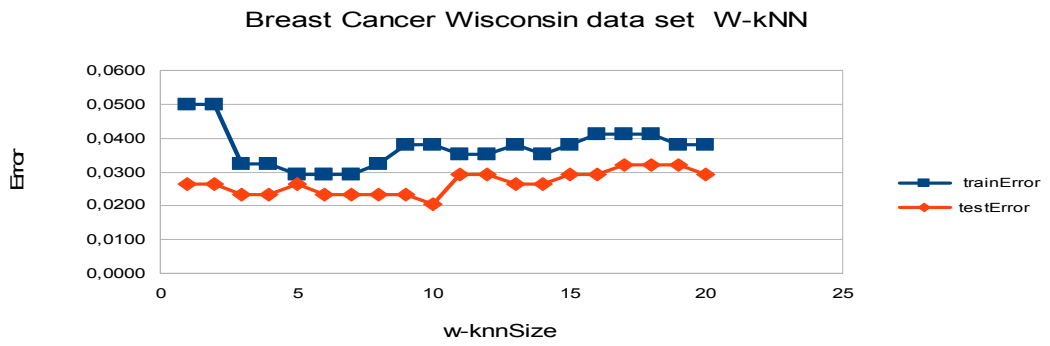
	Simple KNN			Weighted KNN	
knnSize	trainError	testError	w-knnSize	trainError	testError
1	0,0516	0,0453	1	0,0516	0,0453
2	0,0486	0,0603	2	0,0516	0,0453
3	0,0450	0,0489	3	0,0442	0,0493
4	0,0468	0,0500	4	0,0428	0,0453
5	**0,0457**	**0,0446**	5	0,0464	0,0449
6	0,0494	0,0497	**6**	**0,0439**	**0,0442**
7	0,0479	0,0497	7	0,0460	0,0482
8	0,0512	0,0522	8	0,0457	0,0475
9	0,0497	0,0500	9	0,0483	0,0497
10	0,0545	0,0541	10	0,0486	0,0489
11	0,0534	0,0537	11	0,0508	0,0522
12	0,0552	0,0551	12	0,0527	0,0526
13	0,0534	0,0559	13	0,0523	0,0537
14	0,0548	0,0570	14	0,0519	0,0551
15	0,0570	0,0570	15	0,0556	0,0559
16	0,0578	0,0570	16	0,0559	0,0562
17	0,0585	0,0581	17	0,0570	0,0581
18	0,0600	0,0592	18	0,0570	0,0577
19	0,0630	0,0610	19	0,0581	0,0592
20	0,0618	0,0610	20	0,0592	0,0610



Breast Cancer Wisconsin data set

Το σύνολο δεδομένων καρκίνου του μαστού του πανεπιστημίου του Wisconsin χρησιμοποιείται ευρέως για να εξετάσει την αποτελεσματικότητα των αλγορίθμων κατηγοριοποίησης/ταξινόμησης. Ο στόχος είναι να διακρίνει τις κλάσεις μεταξύ των καλοήθων και κακοήθων καρκίνων βασισμένων στις διαθέσιμες εννέα μετρήσεις. Οι μετρήσεις έχουν τιμές ακέραιων από ένα έως δέκα [1… 10]. Η αρχική βάση δεδομένων περιέχει 699 παραδείγματα εντούτοις 16 από αυτά παραλείπονται επειδή είναι ελλιπείς, το οποίο είναι κοινό και με άλλες μελέτες. Η κατανομή κλάσεων είναι 65,5% καλοήθες και 34,5% κακοήθες, αντίστοιχα. Στον πίνακα καθώς και στην γραφική παράσταση που ακολουθεί παρατηρούμε το train error, το test error του σύνολο δεδομένων Breast Cancer Wisconsin, τις διάφορες τιμές που παίρνει το k. Το βέλτιστο k για τον απλό KNN είναι k = 3 με αντίστοιχο test error : 0.0351, και για τον σταθμισμένο KNN είναι k= 10 με error 0.0205.

	Simple KNN			Weighted KNN	
knnSize	trainError	testError	w-knnSize	trainError	testError
1	0,044	0,0497	1	0,0499	0,0263
2	0,0499	0,0526	2	0,0499	0,0263
3	**0,0323**	**0,0351**	3	0,0323	0,0234
4	0,0323	0,0439	4	0,0323	0,0234
5	0,0268	0,0409	5	0,0293	0,0263
6	0,0323	0,0468	6	0,0293	0,0234
7	0,0205	0,0439	7	0,0293	0,0234
8	0,0264	0,0409	8	0,0323	0,0234
9	0,0235	0,0409	9	0,0381	0,0234
10	0,0264	0,0439	**10**	**0,0381**	**0,0205**
11	0,0264	0,0409	11	0,0352	0,0292
12	0,0264	0,0439	12	0,0352	0,0292
13	0,0235	0,0439	13	0,0381	0,0263
14	0,0264	0,0439	14	0,0352	0,0263
15	0,0264	0,0439	15	0,0381	0,0292
16	0,0323	0,0439	16	0,0411	0,0292
17	0,0293	0,0439	17	0,0411	0,0322
18	0,0323	0,0439	18	0,0411	0,0322
19	0,0293	0,0439	19	0,0381	0,0322
20	0,0352	0,0439	20	0,0381	0,0292



Dermatology data set

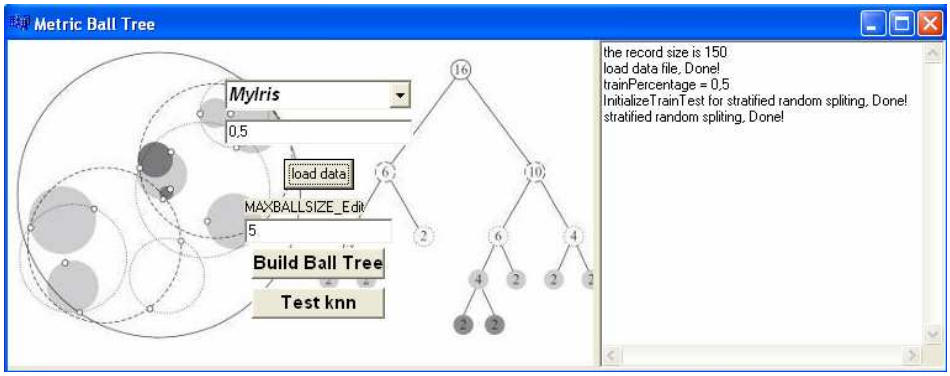
Η διάγνωση των Ερύθημα- επιθήλιο (erythemato-squamous) ασθενειών είναι ένα πραγματικό πρόβλημα στο κλάδο της δερματολογία. Οι ασθένειες σε αυτήν την ομάδα είναι ψωρίαση, Σμηγματορροϊκή δερματίτιδα , Ομαλός λειχήνας, Πιτυρίαση rosea, Πιτυρίαση pilaris rubra. Συνήθως μια βιοψία είναι απαραίτητη για τη διάγνωση αλλά δυστυχώς αυτές οι ασθένειες μοιράζονται πολλά ιστοπαθολογικά χαρακτηριστικά γνωρίσματα. Μια άλλη δυσκολία για τη διάγνωση είναι ότι μια ασθένεια μπορεί αρχικά να παρουσιάσει τα χαρακτηριστικά γνωρίσματα μιας άλλης ασθένειας και μπορεί να έχει τα χαρακτηριστικά γνωρίσματα στα ακόλουθα στάδια. Το σύνολο δεδομένων περιλαμβάνει 34 κατηγορίες με 358 καταχωρήσεις για την κάθε μια. Οι τιμές των ιστοπαθολογικών χαρακτηριστικών γνωρισμάτων καθορίζονται από μια ανάλυση των δειγμάτων κάτω από ένα μικροσκόπιο. Το χαρακτηριστικό γνώρισμα οικογενειακής ιστορίας έχει την αξία 1 εάν οποιαδήποτε από αυτές τις ασθένειες έχει παρατηρηθεί στην οικογένεια, και 0 ειδάλλως. Το χαρακτηριστικό γνώρισμα ηλικίας αντιπροσωπεύει απλά την ηλικία του ασθενή. Σε κάθε άλλο χαρακτηριστικό γνώρισμα (κλινικό και ιστοπαθολογικό) δόθηκε ένας βαθμός από το 0 έως 3. Εδώ, 0 δείχνουν ότι το χαρακτηριστικό γνώρισμα δεν ήταν παρόν, 3 δείχνουν το μεγαλύτερο ποσό πιθανό, και 1 ..2 δείχνει τις σχετικές ενδιάμεσες τιμές. Στον πίνακα που ακολουθεί παρατηρούμε το train error, το test error. Παρατηρούμε πως το βέλτιστο k για τον απλό KNN είναι k = 1 με αντίστοιχο test error : 0.0543, και για τον σταθμισμένο KNN είναι k=5 με αντίστοιχο error = 0.0333.

	Simple KNN			Weighted KNN	
knnSize	trainError	testError	w-knnSize	trainError	testError
1	**0,0275**	**0,0543**	1	0,0618	0,0611
2	0,011	0,0598	2	0,0618	0,0611
3	0,022	0,0598	3	0,0562	0,0444
4	0,0165	0,0707	4	0,0562	0,0444
5	0,022	0,0543	**5**	**0,0506**	**0,0333**
6	0,0165	0,0598	6	0,0562	0,0333
7	0,0165	0,0707	7	0,0506	0,0389
8	0,0165	0,0597	8	0,0449	0,0556
9	0,0165	0,0761	9	0,0449	0,0444
10	0,0165	0,0761	10	0,0618	0,0444
11	0,0385	0,0815	11	0,0618	0,0389
12	0,033	0,0761	12	0,0618	0,0389
13	0,0275	0,0815	13	0,0618	0,0444
14	0,0275	0,087	14	0,0618	0,0444
15	0,033	0,0815	15	0,0674	0,0444
16	0,033	0,0815	16	0,0674	0,0389
17	0,033	0,087	17	0,0674	0,0444
18	0,033	0,0761	18	0,0674	0,0333
19	0,044	0,0924	19	0,0618	0,0444
20	0,044	0,087	20	0,0674	0,0444

Κεφάλαιο 7 ΠΕΡΙΓΡΑΦΗ ΚΑΙ ΚΩΔΙΚΑΣ ΕΦΑΡΜΟΓΗΣ

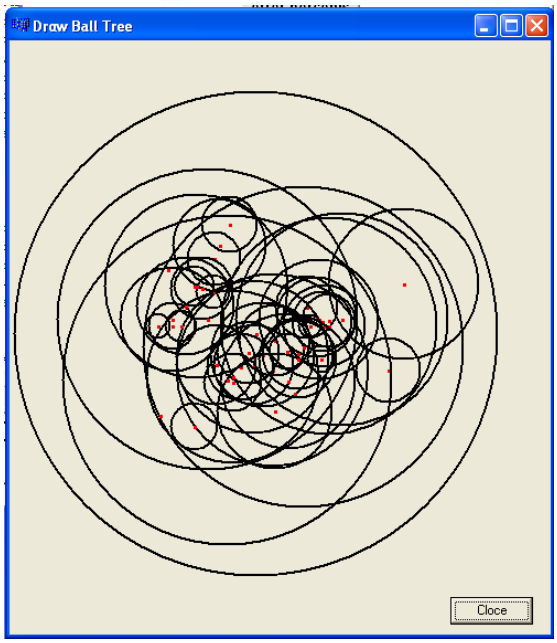
Η εφαρμογή Metric Ball Tree είναι μία εφαρμογή με οπτικό περιβάλλον (GUI), αυτό επιλέχτηκε ώστε να μπορούμε να δούμε σε ένα γραφικό περιβάλλων την δημιουργία του ball tree. Αυτή είναι και η βασική εφαρμογή της πτυχιακής μας, κατά την οποία τα δεδομένα που θέλουμε να επεξεργαστούμε προέρχονται από την σύνδεση της εφαρμογής με βάση δεδομένων Oracle 9. Τα δεδομένα αρχικοποιούνται και χωρίζονται με χρήση της μεθόδου stratified random split . Στην συνέχεα δημιουργείται το δέντρο και εμφανίζεται σε μία δεύτερη φόρμα και στο τέλος δοκιμάζουμε τα test δεδομένα για να βρούμε τους κοντινότερες γείτονες του καθενός με αναζήτηση των κοντινών γειτόνων μέσω του δέντρου.

Όταν ξεκινάει η εφαρμογή εμφανίζετε η αρχική φόρμα επικοινωνίας με τον χρήστη. Σε αυτήν ο χρήστης μπορεί να επιλέξει τον πίνακα στον οποίο θα ήθελε να δημιουργήσει το δέντρο και αργότερα να δοκιμάσει τα testdata σε αυτό. Επίσης μπορεί να επιλέξει και το ποσοστό του stratified random split, ώστε τα δεδομένα να διαχωριστούν ανάλογα σε train set και test set. Ο προκαθορισμένος πίνακας είναι ο MyIris και ότι το αντίστοιχο ποσοστό για train set είναι το 0,5 δηλαδή το μισό των δεδομένων της βάσης. Πατώντας το κουμπί “load data” φορτώνονται όλα τα δεδομένα, γίνετε αρχικοποίηση των τιμών των πινάκων και χωρίζονται τα δεδομένα στους απαραίτητους πίνακες, και εμφανίζονται τα απαραίτητα μηνύματα στο memo, που βρίσκεται δεξιά επί της αρχικής φόρμας.



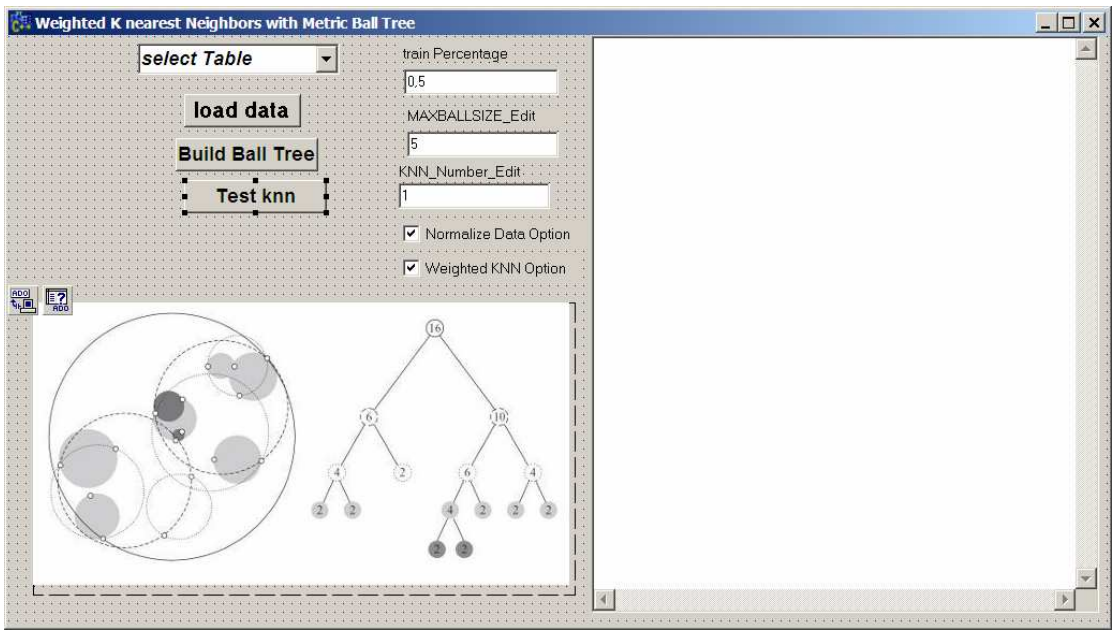
Βασική φόρμα εφαρμογής Metric Ball Tree v4.1

Όταν φορτωθούν τα δεδομένα εμφανίζετε στην φόρμα άλλο ένα κουμπί το “Build Ball Tree” όπου. Το κουμπί αυτό φτιάχνει το ball tree και μας εμφανίζει μία καινούρια φόρμα στην οποία σχεδιάζετε το δέντρο. Το κουμπί αυτό παίρνει υπόψη του την τιμή του MaxBallSize Editbox.



Το δέντρο όπως εμφανίζετε στην φόρμα Draw Ball Tree

Τέλος το κουμπί Test knn τρέχει όλα τα δεδομένα του testset, και ξεχωριστά για το καθένα βρίσκει την μπάλα με την κοντινότερη απόσταση από το συγκεκριμένο σημείο, και μια μία διαδικασία βρίσκει τις γειτονικές μπάλες ώστε να τρέξει ο αλγόριθμος knn για να βρούμε τους k κοντινότερους γείτονες, οι οποίοι εισάγονται από τον χρήστη από το αντίστοιχο Editbox. Όταν γίνει και αυτό τρέχει ο αλγόριθμός εύρεσης 1κοντινότερου γείτονα. Στην συνέχεια ο αλγόριθμος k κοντινότερων γειτόνων με k=1, ώστε να ελέγξουμε κατά πόσο είναι σωστός και έπειτα τρέχουμε ξανά τον knn με k = 1 μέχρι k = 20.



Unit1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <assert.h>
#include <conio.h>

#include <string.h>    // for strcpy
#include <ctype.h>
#include <time.h>
#include <functional>
#include <iostream>


#pragma hdrstop

#include "Unit1.h"
#include "UnitFunctions.h"
#include "Unit2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
/*********************************global variables ******************************/


//**********************for loading all data and labels*************************
   long dataSize;              // data size (number of rows)
   int dataColumns ;                  // Number of dimensions or variables in x vector
   int dataClasses ;                  // Number of Classes   (0 to numOfClasses-1)
   double **allData  = NULL ;   // allData
   int *allY = NULL ;            // all_Y class labels, used double for compatibility across PNN, RBF, GRNN
   long int *numSamplesInClass ;// size of numOfClasses vector contain number
                              // of samples in each class population

   int NormalizeOption ;
   int weightedKNNOption ;

//*********for split (stratified) to train and test*****************************
   double **trainX = NULL ;          // trainData[trainSize][dataColumns]
   double **testX = NULL ;          // testData[testSize][dataColumns]
   int *trainY = NULL ;             // trainLabels[trainSize]
   int *testY = NULL ;            // testLabels[testSize]
   long *trainSamplesInClass = NULL ;// number of train samples in each class
   float trainPercentage ;
   long trainSize , testSize ;       // trainSize and testSize
   double bestval , trainError, testError;
```

```
//-----------------------------------------------------------------------------
/************************** ball tree Types **********************************/
struct   ballNode
{//domh gia kathe kombo
        double* center ; // center of node
        double radius; //radius of node (distance of center from furthest point)
        long int sizeOfPoints;     // points contained if is leaf , and points bellow if is internal node
        struct ballNode *leftBall; //
        struct ballNode *rightBall; //
        struct ballNode *parentBall; //
        double ** pointArray ; //pointer se pinaka domwn pou krataei ta shmeia
        long int *pointIndexes ; // krataei  indexes
        int isleaf;    // 1 is leaf, 0 not leaf
        long firstFurthest;
        long secondFurthest;

} ;
typedef struct ballNode ballNode1 ;

struct ballNode * rootNode = NULL ;
int MAXBALLSIZE ;
         //ball Arrays
long ballCounter = 0 ;
long ballArraySize = 0 ;
struct ballNode ** ballArray = NULL ;
       //leaves Array
long leavesCounter = 0 ;
long  leavesArraySize = 0 ;
struct ballNode ** leavesArray = NULL ;

struct ballNode * ballFound ;
struct ballNode * KNNqueryBall  ;
double ** knnCandidatePoints = NULL ;
long CandidatePointsSize  ;
long * knnCandidatePointsIndexes = NULL ;

 long knnSize = 1 ;


 //------------------for ball tree------------------------
void initializeNode(struct ballNode *bb, long int pointsize,int dimension);
void setBallCenter(struct ballNode *bb, int dimension );
void setBallRadius(struct ballNode *bb, int dimension );
void setBallFurthestPair(struct ballNode *bb, int dimension );
void splitBallNode(struct ballNode* AX, int dim);
void findNearestBall(struct ballNode *bb , int dim, struct ballNode *nearestBall, double * newpoint);
void findNearestKNNqueryBall(struct ballNode *bb , int K);
int ballOneContainsBallTwo(struct ballNode *One, struct ballNode * Two, int dimension);
int ballOneIntersectsBallTwo(struct ballNode *One, struct ballNode * Two, int dimension);
void pushLeavesContainedInBall(struct ballNode *currentBall , int dim , struct ballNode * queryBall);
void pushLeavesIntersectingBall(struct ballNode *currentBall , int dim , struct ballNode * queryBall);
void pushLeavesContainingBall(struct ballNode *currentBall , int dim , struct ballNode * queryBall);
```

```
void findNearest2(struct ballNode *bb , int dimension, double * newpoint) ;
void FindLeavesOfNode(struct ballNode *AX);

void findNearestUsingBallTree( int dataColumns ,long * knnRealIndexes , double * newpoint ,long
knnSize , struct ballNode * rootNode);
void PostOrderDeleteBalls(struct ballNode* BB);

void PostOrderDrawBalls(TPaintBox *PaintBox1, struct ballNode* AX);
void drawBall(TPaintBox *PaintBox1, struct ballNode* AX);


/*******************************************************************************
 fuction :  initializeNode
*******************************************************************************/
void initializeNode(struct ballNode *bb, long int pointsize,int dimension)
{//arxikopoihsh enws kombou me ta stoixeia p theloume

   bb->center = (double*)malloc(dimension*sizeof(double));  // center of node
   bb->radius = 0 ; //radius of node (distance of center from furthest point)
   bb->leftBall = NULL ; //
   bb->rightBall = NULL ; //
   bb->parentBall = NULL ; //
   bb->pointArray = NULL; //pointer se pinaka domwn p krataei ta shmeia
   bb->isleaf = 0 ;    // 1 is leaf, 0 not leaf
   bb->sizeOfPoints = pointsize ;
   bb->pointArray = (double**) malloc(pointsize*sizeof(double*)) ;
   bb->pointIndexes =  (long*) malloc(pointsize*sizeof(double)) ;
   bb->firstFurthest = 0;
   bb->secondFurthest = 0;
   //Form1->Memo1->Lines->Add("initialize Node, Done!");
   ballCounter ++;

}
/*******************************************************************************
set the cender of the ball
*******************************************************************************/
void setBallCenter(struct ballNode *bb, int dimension )
{
   long int   i, d;

  bb->center = (double*)calloc(dimension, sizeof(double));

   for (i=0; i<bb->sizeOfPoints; i++) {
      for (d=0; d<dimension; d++) {
          bb->center[d] += bb->pointArray[i][d];
      }
   }
   for (d=0; d<dimension; d++) {
       if ( bb->sizeOfPoints ==1 ) bb->center[d] = bb->pointArray[0][d] ;
       else    bb->center[d] /=  (double) bb->sizeOfPoints ;
   }
```

```
}
/*******************************************************************************
Function : setBallRadius
*******************************************************************************/
void setBallRadius(struct ballNode *bb, int dimension )
{
   long int   i;
   double dist, max_dist;
   /* find the point id that has max distance to center */
   max_dist = EuclideanDistance(dimension, bb->center, bb->pointArray[0] );
   for (i=1; i<bb->sizeOfPoints; i++) {
      dist = EuclideanDistance(dimension, bb->center, bb->pointArray[i]);
      if (dist > max_dist) { max_dist = dist; }
   }

   bb->radius = max_dist ;
}

//---------------------------------------------------------------------------

void drawBall(TPaintBox *PaintBox1, struct ballNode* AX)
{

int  Xpos, Ypos , Rleft, Rtop, Rright, Rbottom;

 double left, top, right, bottom ;

 Xpos = ( AX->center[0]*(PaintBox1->Width/2) ) / 1  + PaintBox1->Width/4;
 Ypos = ( (1 - AX->center[1])*(PaintBox1->Height/2 ) ) / 1 + PaintBox1->Width/4;
 PaintBox1->Canvas->Pen->Color = clRed;
 PaintBox1->Canvas->Pen->Width = 4;
 PaintBox1->Canvas->MoveTo(Xpos,Ypos);
 PaintBox1->Canvas->Rectangle(Xpos,Ypos,Xpos+1,Ypos+1);


  PaintBox1->Canvas->Pen->Width = 2;
  PaintBox1->Canvas->Pen->Color = clBlack;
 Rleft = Xpos  - ( AX->radius*(PaintBox1->Width/2) ) / 1  ;
 Rright = Xpos + ( AX->radius*(PaintBox1->Width/2) ) / 1 ;
 Rtop = Ypos - ( AX->radius*(PaintBox1->Width/2) ) / 1 ;
 Rbottom = Ypos + ( AX->radius*(PaintBox1->Width/2) ) / 1 ;


 PaintBox1->Canvas->Arc(Rleft, Rtop, Rright, Rbottom,0,0,0,0);

   Form1->Memo1->Lines->Add(AnsiString(AX->radius) + " , " + AnsiString(AX->sizeOfPoints)+ " , " +
AnsiString(AX->center[0]));
  // Form1->Memo1->Lines->Add(AX->sizeOfPoints  );


}
```

```
//---------------------------------------------------------------------------


__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
ButtonTest->Visible =false;
ButtonTrain->Visible=false;
ADOConnection1->GetTableNames(ComboTables->Items, false);

}
//---------------------------------------------------------------------------

void resetLoadData(void){
   // FREE ALL IN REVERSE ORDER


  if (testY != NULL) free (testY);
  if (trainY != NULL) free (trainY);
  if (testX != NULL)  {free(testX[0]); free(testX);} // by allocate2DArray
  if (trainX != NULL) {free(trainX[0]); free(trainX);} // by allocate2DArray
  if (trainSamplesInClass != NULL) free (trainSamplesInClass) ;

  if (numSamplesInClass != NULL) free (numSamplesInClass);
  if (allY != NULL) free (allY);
  if (allData != NULL) {free(allData[0]); free(allData);} // by allocate2DArray
  if (knnCandidatePoints!=NULL) free(knnCandidatePoints);
  if (knnCandidatePointsIndexes!=NULL) free(knnCandidatePointsIndexes);
}

//---------------------------------------------------------------------------
void __fastcall TForm1::ButtonLoadClick(TObject *Sender)
{
AnsiString table;

    resetLoadData();

   Memo1->Lines->Clear();
   if (ComboTables->Text == "select Table")  table="MyIris";
   else   table=Form1->ComboTables->Text ;

   ButtonTest->Visible=true;
   ButtonTrain->Visible=true;
   ADOConnection1->Connected = false;
   ADOConnection1->Connected = true;
```

```
 NormalizeOption = Form1->CheckBox1->Checked ;
 /*real loads input data file*/
   allData = loadDataTable( &dataColumns,  &dataSize,  &dataClasses, &allY,
                  &numSamplesInClass, table, NormalizeOption) ;


  if (Edit1->Text.ToDouble() > 0.0 &&  Edit1->Text.ToDouble() <=  0.9 ) trainPercentage = Edit1-
>Text.ToDouble();
  else {
       Memo1->Lines->Add("error, the trainPercentage is now default, '0.5' " ) ;
       trainPercentage = 0.5 ;
      }

  Memo1->Lines->Add("trainPercentage = "+AnsiString( trainPercentage));

 InitializeTrainTest(dataColumns, dataSize, dataClasses, numSamplesInClass, allData, allY,
    &trainX, &testX, &trainY, &testY, &trainSamplesInClass, &trainSize, &testSize, trainPercentage);

 StratifiedRandomSplitTT(dataColumns, dataClasses, numSamplesInClass,  allData, allY,
         trainX, testX, trainY, testY, trainSamplesInClass, trainPercentage)  ;


 knnCandidatePoints = (double**)malloc(trainSize *sizeof(double*)) ;
 knnCandidatePointsIndexes = (long*)malloc(trainSize * sizeof(long)) ;


}
//---------------------------------------------------------------------------
void ballTreeReset(void)
{
  int i ;

   if  (rootNode != NULL ) {
       //delete ball tree
      ballCounter = 0 ;
      PostOrderDeleteBalls( rootNode ) ;
      for (i=0; i<ballCounter ; i++) free(ballArray[i]);       //itane se sxolio
      rootNode = NULL ;
    }

   if (ballArray != NULL)  free(ballArray) ;           //delete old ball array
   if (leavesArray != NULL)  free(leavesArray) ;

    ballCounter = 0;
    leavesCounter = 0 ;
    ballArraySize = 0 ;
    leavesArraySize = 0 ;
}
 //-------------------------------------------------------------
```

```
void __fastcall TForm1::ButtonTrainClick(TObject *Sender)
{
    int i;

    Memo1->Lines->Clear() ;

    ballTreeReset() ;


    rootNode = ( struct ballNode * ) malloc( sizeof( ballNode1 ) ) ;
    initializeNode( rootNode, trainSize, dataColumns ) ;
    for (i = 0; i < trainSize ; i++) {
        rootNode->pointArray[i] = trainX[i] ;
        rootNode->pointIndexes[i] = i ;
    }

    setBallCenter( rootNode,  dataColumns) ;

    setBallRadius( rootNode,  dataColumns ) ;
    setBallFurthestPair(rootNode,  dataColumns ) ;

    MAXBALLSIZE = MAXBALLSIZE_Edit->Text.ToInt() ;

    splitBallNode( rootNode, dataColumns ) ;          //create ball tree
    // has finished now
   //Memo1->Lines->Add("splitBallNode , Done!");
 /*
     Form2->PaintBox1->Enabled =false;
     Form2->PaintBox1->Enabled=true;
     Application->ProcessMessages() ;
     Form2->Show();
     PostOrderDrawBalls(Form2->PaintBox1, rootNode);  //Draw ball tree
 */

    ballArraySize = ballCounter ;
    leavesArraySize = leavesCounter ;

    Form1->Memo1->Lines->Add( "ballCounter = "+ AnsiString(ballCounter));
    // a list of ball pointers to be used for many things
     //create ball array
    ballArray = (struct ballNode** ) malloc(ballArraySize*sizeof(ballNode1) ) ;
      //create leaves Array
    leavesArray = (struct ballNode** ) malloc(leavesArraySize*sizeof(ballNode1) ) ;



}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
  resetLoadData();
```

```
  ballTreeReset();
}
/*****************************************************************************
 given a dataPoint this function finds the closest to it cluster center
 For N points and k centers has O(Nk) complexity
*****************************************************************************/
int findNearestCluster(
       int numOfClusters,     // Number of clusters
       int   dataColumns,     // Number of dimensions or variables or features
       double  *dataPoint,    // one data point[dataColumns]
       double **clusters      // [numOfClusters][dataColumns]
        )
{
    int   index, i;
    double dist, min_dist;

    /* find the cluster id that has min distance to object */
    index    = 0;
    min_dist = EuclideanDistance(dataColumns, dataPoint, clusters[0]);

    for (i=1; i<numOfClusters; i++) {
        dist = EuclideanDistance(dataColumns, dataPoint, clusters[i]);
        /* no need square root */
        if (dist < min_dist) { /* find the min and its array index */
            min_dist = dist;
            index    = i;
        }
    }
    return(index);
}

/*******************************************************************
autocall function for spliting the ballNode  make the ball tree
*******************************************************************/
void splitBallNode(struct ballNode* AX, int dimension)
{
long int i, BallDataSize, leftsize=0, rightsize=0;
long int index , L=0, R=0 ;

    int *membership = NULL ;         // [BallDataSize] corresponding cluster of each point
    double **clusterCenters = NULL ; // [numOfClusters][dataColumns]
    long clustersSize[2] ;      // count of dataPoints assigned in each cluster
    double threshold ;              // when to stop, percentage% of dataPoints change cluster
    int numOfClusters ;             // number of desired clusters
    int d ;


    BallDataSize = AX->sizeOfPoints ;

     if ((BallDataSize <= MAXBALLSIZE)|| (AX==NULL)) {
         if(AX!=NULL){
          setBallRadius(AX , dimension );
```

```
        setBallCenter(AX ,  dimension );
        // POY EINAI TO SET BALL CENTER ??  EXEI HDH GINEI SET
        AX->isleaf = 1 ;
        leavesCounter++ ;
       }
        return;
   }


   //FIND STATISTICS AND THE SPLITING PIVOT
   //CALL 2-MEANS CLUSTERING
   numOfClusters = 2 ;

   // ------------initialize k-means for clustering the train set ---------
  membership =(int*) malloc(BallDataSize*sizeof(int));

  // allocate a 2D space for clusterCenters[numOfClusters][dataColumns]

  clusterCenters = (double**) malloc(numOfClusters*sizeof(double*));
  clusterCenters[0]= (double*) calloc(dimension,sizeof(double));
  clusterCenters[1]= (double*) calloc(dimension,sizeof(double));

 for (d=0;d<dimension;d++) {
      clusterCenters[0][d]= AX->pointArray[AX->firstFurthest][d] ;
      clusterCenters[1][d]= AX->pointArray[AX->secondFurthest][d] ;
 }
  clustersSize[0]=clustersSize[1]=0;

 for (i=0; i<BallDataSize; i++) {
     // find the array index of nestest cluster center
     index = findNearestCluster(numOfClusters, dimension, AX->pointArray[i], clusterCenters);
      // assign the membership to object i
      membership[i] = index;
     // count of dataPoints located within cluster
    clustersSize[index]++;
}

  //FIND LEFT AND RIGHT SIZES
  leftsize = clustersSize[0] ;
  rightsize = clustersSize[1] ;
  //Form1->Memo1->Lines->Add("right size = "+rightsize) ;

   //SEND POINTS (OR POINT INDEXES) TO LEFT AND RIGHT
  //  ...............................................
  L=R=0;
   if (leftsize>=1)   {
        AX->leftBall = (struct ballNode *) malloc(sizeof(ballNode1));
        initializeNode(AX->leftBall, leftsize ,dimension) ;
        AX->leftBall->parentBall = AX ;      //SET PARENT
      //leftball
        for (i=0; i<BallDataSize; i++)
             if (membership[i]==0) {
```

```
                AX->leftBall->pointArray[L]  = AX->pointArray[i];
                AX->leftBall->pointIndexes[L]= AX->pointIndexes[i] ;
                L++;
              }
        setBallCenter(AX->leftBall,  dimension );
        setBallRadius(AX->leftBall,dimension );
        setBallFurthestPair(AX->leftBall,dimension);
        splitBallNode(AX->leftBall,dimension);
    }

    if (rightsize>=1) {
        AX->rightBall = (struct ballNode*) malloc(sizeof(ballNode1));
        initializeNode(AX->rightBall,rightsize,dimension) ;
        AX->rightBall->parentBall = AX ;      //SET PARENT
        //rightball
        for (i=0; i<BallDataSize; i++)
              if (membership[i]==1) {
               AX->rightBall->pointArray[R]  = AX->pointArray[i];
               AX->rightBall->pointIndexes[R]= AX->pointIndexes[i] ;
               R++;
               }
       setBallCenter(AX->rightBall,  dimension );
       setBallRadius(AX->rightBall,dimension );
       setBallFurthestPair(AX->rightBall,dimension );
       splitBallNode(AX->rightBall,dimension);
    }

       //FREE INPUT ARRAY AX->pointarray TO SAVE MEMORY SPACE
    if (AX->pointArray != NULL) { free(AX->pointArray); AX->pointArray=NULL;}
    if (AX->pointIndexes != NULL) { free(AX->pointIndexes); AX->pointIndexes=NULL;}
    if (membership != NULL) {free(membership);}
        free(clusterCenters[0]);free(clusterCenters[1]);free(clusterCenters);
}

 /*****************************************************************************/
void setBallFurthestPair(struct ballNode *bb, int dimension )
{
   long int   i, index1, index2;
   double dist, max_dist;
   /* find the point id that has max distance to center */
   index1 = 0;
   max_dist = EuclideanDistance(dimension, bb->center, bb->pointArray[0] );
   for (i=1; i<bb->sizeOfPoints; i++) {
      dist = EuclideanDistance(dimension, bb->center, bb->pointArray[i]);
      if (dist > max_dist) { max_dist = dist;  index1    = i;}
   }

   bb->firstFurthest = index1 ;

   index2 = 0 ;
   max_dist = EuclideanDistance(dimension, bb->pointArray[index1], bb->pointArray[0] );
   for (i=1; i<bb->sizeOfPoints; i++) {
```

```
        dist = EuclideanDistance(dimension, bb->pointArray[index1], bb->pointArray[i]);
        if (dist > max_dist) { max_dist = dist;  index2    = i;}
    }

    bb->secondFurthest  = index2 ;

}

/*****************************************************************************/

 void PostOrderDrawBalls(TPaintBox *PaintBox1, struct ballNode* AX)
{

   if (AX==NULL){return;}
   if (AX->leftBall != NULL)  PostOrderDrawBalls(PaintBox1, AX->leftBall);
   if (AX->rightBall != NULL) {
        //Form1->Memo1->Lines->Add("RIGHT BALL") ;
        PostOrderDrawBalls(PaintBox1, AX->rightBall);
        }
   drawBall(PaintBox1, AX);

}
 /*****************************************************************************/

 void PostOrderTraversal( struct ballNode* AX)
{

   if (AX==NULL){return;}
   if (AX->leftBall != NULL)  PostOrderTraversal( AX->leftBall);
   if (AX->rightBall != NULL) PostOrderTraversal( AX->rightBall);

   //drawBall(PaintBox1, AX);

}

/*****************************************************************************/
void PostOrderDeleteBalls(struct ballNode* BB)
{
        if (BB==NULL){return;}
        if (BB->leftBall != NULL) {
                ballArray[ballCounter] = BB->leftBall ;
                ballCounter++ ;
                PostOrderDeleteBalls(BB->leftBall) ;


                //free(BB->leftBall);
                //BB->leftBall==NULL;

                }
        if (BB->rightBall != NULL) {
                ballArray[ballCounter] = BB->rightBall ;
                 ballCounter++ ;
```

```
            PostOrderDeleteBalls(BB->rightBall) ;

            //free(BB->rightBall);
            // BB->rightBall==NULL;
            }
     //freeBallContaints
      if (BB->center != NULL) free(BB->center);
      if (BB->pointArray != NULL) free(BB->pointArray);
      if (BB->pointIndexes != NULL) free(BB->pointIndexes);

}
/*****************************************************************************
find the nearest ball of the ball tree from a new point
*****************************************************************************/
void  findNearestBall(
      struct ballNode *bb ,          //input : the current ball
      int dimension ,                //input : the dimension of  the bata
      struct ballNode *nearestBall , //output: the nearest ball (center) to the newpoint
      double * newpoint              //input : the point to which we bll
      )
{
      int i ;
      double distance1 , distance2 ;

      if ( ( bb->isleaf ) )
      {
         nearestBall->center = (double*)malloc(dimension*sizeof(double));  // center of node
         for (i=0; i<dimension ; i++) nearestBall->center[i] = bb->center[i];

         nearestBall->radius = bb->radius;
         nearestBall->sizeOfPoints = bb->sizeOfPoints ;
         nearestBall->pointArray = (double**) malloc(bb->sizeOfPoints *sizeof(double*));
         nearestBall->pointIndexes = (long*) malloc(bb->sizeOfPoints *sizeof(long)) ;

         for (i=0; i<bb->sizeOfPoints ; i++)  {
               nearestBall->pointArray[i] = bb->pointArray[i] ;
               nearestBall->pointIndexes[i] = bb->pointIndexes[i];
          }

         return ;
      }

      distance1 =  EuclideanDistance ( dimension , bb->leftBall->center ,  newpoint );
   //     Form1->Memo1->Lines->Add(AnsiString(distance1));
      distance2 =  EuclideanDistance ( dimension , bb->rightBall->center , newpoint );
     // Form1->Memo1->Lines->Add(AnsiString(distance1) + " " + AnsiString(distance2) ) ;

      if ( distance1 > distance2 )
          findNearestBall (bb->rightBall , dimension , nearestBall, newpoint );
      else findNearestBall (bb->leftBall , dimension , nearestBall, newpoint );

}
```

```
//-----THIS FUNCTION RETURNS ALL LEAF BALLS OF A NODE---------
void FindLeavesOfNode(struct ballNode *AX)
{   //pre order traversal
   if (AX->isleaf){
       leavesArray[leavesCounter] = AX ;    // global, must be initialized
       leavesCounter++ ;                    // global, must be initialized
        return ;
   }
   if (AX->leftBall != NULL)  FindLeavesOfNode(AX->leftBall) ;
   if (AX->rightBall!= NULL)  FindLeavesOfNode(AX->rightBall) ;

}
//-------------------------------------------------------------
void  findNearest2(struct ballNode *bb , int dimension, double * newpoint)
{
      int i ;      // ballFound is global variable
      double distance1 , distance2 ;
      distance1 =  EuclideanDistance ( dimension , bb->leftBall->center ,  newpoint );
      distance2 =  EuclideanDistance ( dimension , bb->rightBall->center , newpoint );

      if ( distance1 > distance2 )
           if (bb->rightBall->isleaf ) {ballFound =  bb->rightBall ;  return ; }
           else findNearest2 (bb->rightBall , dimension , newpoint ) ;
      else if (bb->leftBall->isleaf ) { ballFound = bb->leftBall ;  return ; }
           else findNearest2 (bb->leftBall , dimension ,  newpoint ) ;

}
//-------------------------------------------------------------
void  findNearestKNNqueryBall(struct ballNode *bb , int K, struct ballNode *queryBall)
{
     // find largest parrent
      if (bb->sizeOfPoints < K ) findNearestKNNqueryBall(bb->parentBall ,K, queryBall) ;
      else { queryBall = bb ; return ; }

}
//-------------------------------------------------------------
int ballOneContainsBallTwo(struct ballNode *One, struct ballNode * Two, int dimension)
{
     double distance1 ;
     distance1 =  EuclideanDistance ( dimension , One->center ,  Two->center ) ;
     if (  ( distance1 - Two->radius ) <= One->radius ) return 1 ;
     else return 0 ;

}

//-----------------------------------------------------------------------------
void  pushLeavesContainingBall(struct ballNode *currentBall , int dimension , struct ballNode *
queryBall)
{      // add leaves under current ball which contains query ball
      int i ;      // ballFound is global variable

       if  ( currentBall->isleaf )  {
```

```
            if ( ballOneContainsBallTwo(currentBall, queryBall, dimension) ) {
                leavesArray[leavesCounter] = currentBall ;
                leavesCounter++ ;  }     // must be initialized to 0
            return ;
        }
        else  {
            if ( ballOneContainsBallTwo(currentBall, queryBall, dimension) ) {
                pushLeavesContainingBall(currentBall->leftBall ,  dimension , queryBall) ;
                pushLeavesContainingBall(currentBall->rightBall , dimension , queryBall) ;
            }
        }

}
//-------------------------------------------------------------
int ballOneIntersectsBallTwo(struct ballNode *One, struct ballNode * Two, int dimension)
{
     double distance1 ;
     distance1 =  EuclideanDistance ( dimension , One->center ,  Two->center ) ;
     if (  distance1 <= ( One->radius + Two->radius ) ) return 1 ;
     else return 0 ;

}
//-------------------------------------------------------------------------------
void  pushLeavesIntersectingBall(struct ballNode *currentBall , int dimension , struct ballNode *
queryBall)
{
      int i ;      // ballFound is global variable

        if  ( currentBall->isleaf )  {
            if ( ballOneIntersectsBallTwo(currentBall, queryBall, dimension) &&
                !ballOneContainsBallTwo(currentBall, queryBall, dimension) ) {
                leavesArray[leavesCounter] = currentBall ;
                leavesCounter++ ;  }     // must be initialized to 0
            return ;
        }
        else  {
            if ( ballOneIntersectsBallTwo(currentBall, queryBall, dimension) &&
                 !ballOneContainsBallTwo(currentBall, queryBall, dimension) ) {
                pushLeavesIntersectingBall(currentBall->leftBall ,  dimension , queryBall) ;
                pushLeavesIntersectingBall(currentBall->rightBall , dimension , queryBall) ;
            }
        }

}
//-------------------------------------------------------------------------------
void  pushLeavesContainedInBall(struct ballNode *currentBall , int dimension , struct ballNode *
queryBall)
{      // add leaves under current ball which contains query ball
      int i ;      // ballFound is global variable

        if  ( currentBall->isleaf )  {
            if ( ballOneContainsBallTwo(queryBall, currentBall, dimension) ) {
```

```
            leavesArray[leavesCounter] = currentBall ;
            leavesCounter++ ;  }     // must be initialized to 0
          return ;
        }
        else  {
            if ( ballOneIntersectsBallTwo(currentBall, queryBall, dimension) &&
                !ballOneContainsBallTwo(currentBall, queryBall, dimension) ) {
            pushLeavesContainedInBall(currentBall->leftBall ,  dimension , queryBall) ;
            pushLeavesContainedInBall(currentBall->rightBall , dimension , queryBall) ;
            }
        }

}

/*****************************************************************************/
// linear Search for KNNs Of a query Point
 //  attention  :for the trainData the first nn is tha same intex ;-)
/*****************************************************************************/
void linearSearchforKNNsOfPoint(
        int dataColumns ,              //input: data dimension
        long * knnIndexes ,            //output:the indexes of the k nearest neighbors
                                  //       in the list of condidates
        double * newpoint ,            //input: the newpoint
        long knnSize ,                 //input: the number of K neighbors
        double ** KnnCandidatePoints ,  //input: the list of candidate neighbors
        long  sizeOfKnnCandidatePoints  //input: the size of the candidates list
  )
{
    long   index, i , j , k , m ;
    double dist, min_dist ,  huge = 9999999.9;
    int foundInKindex;

            for (k=0; k < knnSize ; k++ ){
        // index[k]    = 0;
        min_dist = huge;
                for (j=0; j<sizeOfKnnCandidatePoints; j++) {
                                foundInKindex = 0 ;
                 // an einai knnSize = 1 (k=0) it skips the following line
                                for (m=0; m < k; m++ ){ if (j == knnIndexes[m] )   foundInKindex=1;  }
                                if(foundInKindex==1)continue;
                    dist = EuclideanDistance(dataColumns, newpoint, KnnCandidatePoints[j]);
                                /* no need square root */

                                if (dist < min_dist) { /* find the min and its array index */
                                min_dist = dist;
                                knnIndexes[k]    = j;
                    }
            }  // end for j
            } // end for k
}
/*****************************************************************************/
void findNearestUsingBallTree(
```

```
        int dataColumns ,              //input: data dimension
        long * knnRealIndexes ,        //output: the indexes of the kNN in the TrainData
        double * newpoint ,            //input: the newpoint
        long knnSize ,                 //input: the number of K neighbors
        struct ballNode * rootNode
        )
{
   long i, w , current ;
   long *   knnLocalIndexes  ;     // the indexes of the kNN in the local list of condidates
   struct ballNode *nearestBall ;
   struct ballNode *queryBall ;

   nearestBall = (struct ballNode *) malloc(sizeof(ballNode1));
   knnLocalIndexes  = (long *) malloc(knnSize* sizeof(long));

   findNearestBall( rootNode , dataColumns, nearestBall, newpoint);

  queryBall = nearestBall ;

  //FTAIEI OTI DEN EXEI GINEI INITIALIZED TO queryBall ???

 //  findNearestKNNqueryBall(nearestBall , knnSize, queryBall);


   leavesCounter = 0 ;
 //ShowMessage("OK 1");
   pushLeavesContainingBall(rootNode , dataColumns , queryBall) ;   //

  pushLeavesIntersectingBall(rootNode , dataColumns , queryBall) ; ///insert to
leavesArray[leavesCounter]

  pushLeavesContainedInBall(rootNode , dataColumns , queryBall) ;  //
     // OLA EINAI TORA STH LISTA LEAVESARRAY

   // LINEAR SEARCH OF K-NEAREST NEIGHBORS IN LIST

    CandidatePointsSize = 0 ;
    for ( w=0 ; w < leavesCounter ; w++  ) CandidatePointsSize += leavesArray[w]->sizeOfPoints ;

    current = 0 ;
    for ( w=0 ; w < leavesCounter ; w++  )
           for ( i=0 ; i < leavesArray[w]->sizeOfPoints  ; i++  ){

           knnCandidatePoints[current] = leavesArray[w]->pointArray[i] ;
           knnCandidatePointsIndexes[current] = leavesArray[w]->pointIndexes[i] ;
           current ++ ;
           }



   linearSearchforKNNsOfPoint( dataColumns, knnLocalIndexes , newpoint, knnSize,
                              knnCandidatePoints ,  CandidatePointsSize) ;
```

```
  for ( i=0 ; i < knnSize  ; i++  )
          knnRealIndexes[i] = knnCandidatePointsIndexes[knnLocalIndexes[i]] ;


  free(knnLocalIndexes);
  free(nearestBall->center);
  free(nearestBall->pointArray);
  free(nearestBall->pointIndexes);
  free(nearestBall);

}
/*************************************************************************/
void oneNearestNeighbourUsingBallTree(
   int dataColumns ,                // Number of dimensions or variables in x vector
   long trainSize ,          // train set size
   double **trainData ,      // trainData[trainSize][dataColumns]
   int *trainY ,             // trainLabels[trainSize]
   double **testData ,               // vectorXSize vector to be classified
   int *testY ,             // trainLabels[trainSize]
   long testSize ,          // train set size
   double *trainError ,      // vector of all the summation units
   double *testError ,
   struct ballNode * rootNode
     )
{
   long   index, i , j;
   double dist, min_dist;
    long secondIndex[2];

   (*trainError) = (*testError) = 0.0 ;

   // FIND TEST ERROR
  for (i=0 ; i<testSize; i++) {

      index    = 0;
      min_dist = EuclideanDistance(dataColumns, testData[i], trainData[0]);
      for (j=1; j<trainSize; j++) {
         dist = EuclideanDistance(dataColumns, testData[i], trainData[j]);
         /* no need square root */
         if (dist < min_dist) { /* find the min and its array index */
            min_dist = dist;
            index    = j;
          }
       }
      findNearestUsingBallTree(  dataColumns , &index , testData[i],  1  , rootNode );
      if (trainY[index]!=testY[i])  (*testError)++ ;
  }


      // FIND TRAIN ERROR WITH HOLDOUT
   for (i=0 ; i<trainSize; i++) {
```

```
        findNearestUsingBallTree(  dataColumns , secondIndex , trainData[i],  2  , rootNode );
      // the secondIndex[0] is the same point, the secondIndex[1] is the nearest
      if (trainY[secondIndex[1]]!=trainY[i])  (*trainError)++ ;

    }


}
/*****************************************************************************/
void __fastcall TForm1::ButtonTestClick(TObject *Sender)
{
     long i, d, w , current ;
  struct ballNode *nearestBall ;

   struct ballNode *KNNqueryBall ;
  long * knnIndexes = NULL ;
  double* newpoint ;
  AnsiString s;

   nearestBall = (struct ballNode *) malloc(sizeof(ballNode1));

   KNNqueryBall = (struct ballNode *) malloc(sizeof(ballNode1));

   findNearestBall( rootNode ,dataColumns,nearestBall, testX[0]);
    Form1->Memo1->Lines->Add("findNearestBall");
  Form1->Memo1->Lines->Add(AnsiString(nearestBall->sizeOfPoints ) );
  Form1->Memo1->Lines->Add(AnsiString(nearestBall->radius  ) );
  Form1->Memo1->Lines->Add(AnsiString(nearestBall->center[0]  ) );

 // findNearest2 (rootNode ,dataColumns, trainX[0]);

 // Form1->Memo1->Lines->Add("findNearest2");

  leavesCounter = 0 ;

                                        //query ball
  pushLeavesContainingBall(rootNode , dataColumns , nearestBall) ;   //

  pushLeavesIntersectingBall(rootNode , dataColumns , nearestBall) ; ///insert to
leavesArray[leavesCounter]

  pushLeavesContainedInBall(rootNode , dataColumns , nearestBall) ;  //

   // OLA EINAI TORA STH LISTA LEAVESARRAY

   // LINEAR SEARCH OF K-NEAREST NEIGHBORS IN LIST

    CandidatePointsSize = 0 ;
    for ( w=0 ; w < leavesCounter ; w++  ) CandidatePointsSize += leavesArray[w]->sizeOfPoints ;

    current = 0 ;
    for ( w=0 ; w < leavesCounter ; w++  )
```

```
        for ( i=0 ; i < leavesArray[w]->sizeOfPoints  ; i++  ){

        knnCandidatePoints[current] = leavesArray[w]->pointArray[i] ;
        knnCandidatePointsIndexes[current] = leavesArray[w]->pointIndexes[i] ;
        current ++ ;
        }


  Form1->Memo1->Lines->Add("leavesArray");
 for ( w=0 ; w < leavesCounter ; w++  ){
   Form1->Memo1->Lines->Add("sizeOfPoints " + AnsiString(leavesArray[w]->sizeOfPoints ) );
   Form1->Memo1->Lines->Add("radius " + AnsiString(leavesArray[w]->radius  ) );
   Form1->Memo1->Lines->Add("center " + AnsiString(leavesArray[w]->center[0]  ) );
 }
 Form1->Memo1->Lines->Add( "leavesCounter is "  +AnsiString(leavesCounter)) ;
 Form1->Memo1->Lines->Add( "leavesArraySize is " +AnsiString(leavesArraySize));

 knnSize = 1 ;
  newpoint =  testX[0] ;
  knnIndexes  = (long *) malloc(knnSize* sizeof(long));

  linearSearchforKNNsOfPoint( dataColumns,knnIndexes , newpoint, knnSize,
                        knnCandidatePoints ,  CandidatePointsSize) ;

 for(w=0; w<knnSize; w++ ) {
       s=" id is ";
       s += AnsiString(knnCandidatePointsIndexes[knnIndexes[w]]) + ", data are " ;
       for(d=0; d<dataColumns; d++ ) s += AnsiString(knnCandidatePoints[knnIndexes[w]][d]) + " " ;
       Form1->Memo1->Lines->Add("nearest " + s) ;
 }


   s="";
   for(d=0; d<dataColumns; d++ ) s += AnsiString(newpoint[d]) +" " ;
   Form1->Memo1->Lines->Add("query  data are " + s) ;

 Memo1->Lines->Add(" ");
  Memo1->Lines->Add("trainSize =" + AnsiString(trainSize) + ", testSize =" +AnsiString(testSize) );

oneNearestNeighbour(dataColumns, trainSize, trainX, trainY, testX, testY, testSize,
              &trainError,  &testError) ;

Memo1->Lines->Add("oneNearestNeighbour "+
                  AnsiString(trainError/(double)trainSize) + "\t " +
                  AnsiString(testError/(double)testSize)  );

 oneNearestNeighbourUsingBallTree(dataColumns, trainSize, trainX, trainY, testX, testY, testSize,
              &trainError,  &testError, rootNode) ;

 Memo1->Lines->Add("oneNearestNeighbourUsingBallTree "+
                  AnsiString(trainError/(double)trainSize) + "\t " +
                  AnsiString(testError/(double)testSize)  );
```

```
 findNearestUsingBallTree(  dataColumns , knnIndexes , testX[0],  knnSize  , rootNode );


printf("from 1-NEAREST NEIGHBOOR ---------> trainError = %lf  testError = %lf\n",
                trainError / (double)trainSize,testError / (double)testSize);

trainError = testError = 0.0 ;

knnSize = Form1->KNN_Number_edit->Text.ToInt() ;
//knnSize =1;

kNearestNeighbours(dataColumns, trainSize, trainX, trainY, testX, testY, testSize,
                 &trainError,  &testError ,knnSize , dataClasses , 0 ) ;

printf("from k-NEAREST NEIGHBOOR ---------> trainError = %lf  testError = %lf\n",
                trainError / (double)trainSize,testError / (double)testSize);

 Memo1->Lines->Add("knnSize,    trainError,   \t testError");

 weightedKNNOption = Form1->CheckBox2->Checked ;

  for (knnSize = 1; knnSize<= 20 ; knnSize++) {

      kNearestNeighbours(dataColumns, trainSize, trainX, trainY, testX, testY, testSize,
                 &trainError,  &testError ,knnSize , dataClasses , weightedKNNOption ) ;

   //    printf("%i\t%3.4lf\t%3.4lf\n",
   //       knnSize, trainError / (double)trainSize,testError / (double)testSize);

       Memo1->Lines->Add(AnsiString(knnSize)+"\t " +
                    AnsiString(trainError/(double)trainSize) + "\t " +
                    AnsiString(testError/(double)testSize)  );


   } // end for knnSize


}

//---------------------------------------------------------------------------
```

UnitFunctions.h

```
//---------------------------------------------------------------------------

#ifndef UnitFunctionsH
#define UnitFunctionsH

//---------------------------------------------------------------------------

/*****************************************************************************
 This function loads data from file specified by 'filename'
 counts the number of samples in each class
 normalizes all colums to (Xi-Xmin)/(Xmax-Xmin)
 and produce all outputs needed for initialization
*****************************************************************************/

double** loadDataTable(
        int *dataColumns ,        // output: data dimension, except last class column
        long int *dataRows ,      // output: data size
        int *dataClasses ,        // output: number of data classes
        int **Yis ,               // output: pointer to *class labels 1D array for memory allocation
        long **numSamplesInClass, // output: pointer to *numSamplesInClass for memory allocation
        AnsiString table ,        // the table name to read
        char normalizedYesNo      // for 1 normalize each column
        ) ;

/*****************************************************************************
 this common function used for allocate and return a 2D array[rows][columns]
*****************************************************************************/
double** allocate2DArray(long rows, long columns);


//---------------------------------------------------------------------------
/*****************************************************************************
  this function allocate memory for train test sets initializes
  InitializeTrainTest(vectorXSize, numOfClasses, numSamplesInClass, allData, allY,
          &trainX, &testX, &trainY, &testY, & trainSamplesInClass, percentage)
*****************************************************************************/


void InitializeTrainTest(
    int vectorXSize ,                   // Number of dimensions or variables in x vector
    long dataSetSize ,          // data size
    int numOfClasses ,                  // Number of Classes   (0 to numOfClasses-1)
    long int *numSamplesInClass ,// size of numOfClasses vector contain number
                                // of samples in each class population
    double **allData ,          // allData
    int *allY ,                 // all_Y     class labels
    double ***trainData ,       // output: pointer to **trainData [trainSize][vectorXSize] for memory
allocation
    double ***testData ,        // output: pointer to **testData [testSize][vectorXSize] for memory allocation
    int **trainY ,              // output: pointer to *train_Y [trainSize]for memory allocation
```

```
   int **testY,                // output: pointer to *test_Y [testSize] for memory allocation
   long **trainSamplesInClass , // output: pointer to *trainSamplesInClass for memory allocation
   long *trainSize ,           // output: pointer to trainSize
   long *testSize ,            // output: pointer to testSize
   float percentage            // percentage % of train set portions
   );

/*******************************************************************************
 SPLIT_TO_TRAIN_TEST WITH STRATIFIED RANDOM SAMPLING
 this function randomly choose a record for train set backets or test set backets
 until both become full with equall percentage for every class so as to
 implement stratification   (equal percentage for every class)
*******************************************************************************/

void StratifiedRandomSplitTT (
   int vectorXSize ,                // Number of dimensions or variables in x vector
   int numOfClasses ,               // Number of Classes   (0 to numOfClasses-1)
   long *numSamplesInClass ,   // size of numOfClasses vector contain number
                               // of samples in each class population
   double **allData ,          // allData
   int *allY ,                 // all_Y
   double **trainData ,        // trainData
   double **testData ,         // testData
   int *trainY ,               // train_Y
   int *testY,                 // test_Y
   long *trainSamplesInClass , // ((float)num_Samples[i]*percentage)/1   /*float to int truncate*/
   float trainPercentage       // percentage % of train set portions
   );

//-------------------------------------------------------------------------------

/*******************************************************************************
square of Euclidean distance between two multi-dimensional points
*******************************************************************************/

double EuclideanDistance ( int size, double *vect1, double *vect2  );

//-------------------------------------------------------------------------------



/*******************************************************************************
 this function is the One Nearest Neighbour classifier
 used here only for comparizon
*******************************************************************************/
void oneNearestNeighbour(
   int dataColumns ,                // Number of dimensions or variables in x vector
   long trainSize ,            // train set size
   double **trainData ,        // trainData[trainSize][dataColumns]
   int *trainY ,               // trainLabels[trainSize]
   double **testData ,              // vectorXSize vector to be classified
   int *testY ,                // trainLabels[trainSize]
```

```
    long testSize ,          // train set size
    double *trainError ,       // vector of all the summation units
    double *testError
      );

 /*******************************************************************************
 this function is the (Weighted) Nearest Neighbours classifier used here
*******************************************************************************/
void kNearestNeighbours(
    int dataColumns ,                // Number of dimensions or variables in x vector
    long trainSize ,          // train set size
    double **trainData ,       // trainData[trainSize][dataColumns]
    int *trainY ,             // trainLabels[trainSize]
    double **testData ,               // vectorXSize vector to be classified
    int *testY ,             // testLabels[trainSize]
    long testSize ,          // test set size
    double *trainError ,       // vector of all the summation units
    double *testError ,
    long knnSize ,
    int dataClasses ,
    char weightedYes
     ) ;



#endif
```

UnitFunctions.cpp

```
//---------------------------------------------------------------------------
 #include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <conio.h>
#include <math.h>
#include <string.h>    // for strcpy
#include <ctype.h>
#include <time.h>
#include <functional>
#include <iostream>

#pragma hdrstop

#include "UnitFunctions.h"
#include "Unit1.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)
/*****************************************************************************************
 This function loads data from file specified by 'filename'
 counts the number of samples in each class
 normalizes all colums to (Xi-Xmin)/(Xmax-Xmin)
 and produce all outputs needed for initialization
*****************************************************************************************/
double** loadDataTable(
        int *dataColumns ,        // output: data dimension, except last class column
        long int *dataRows ,      // output: data size
        int *dataClasses ,        // output: number of data classes
        int **Yis ,               // output: pointer to *class labels 1D array for memory allocation
        long **numSamplesInClass, // output: pointer to *numSamplesInClass for memory allocation
        AnsiString table ,        // the table name to read
        char normalizedYesNo      // for 1 normalize each column
        )
{
    // 1) loads data points from user specified TABLE
    // 2) NORMALIZE THEM
    // 3) PREPARE LABELS FOR CLASSIFICATION
    int j, Columns, Classes, classIndex, firstClass, FieldCount;
    long int i, Rows;

    double** temp, b , *Xmin=NULL, *Xmax=NULL;

   AnsiString sql;
   // find Columns, Classes, Rows, FirstClass.
   Form1->ADOQuery1->SQL->Text = "Select category from " + table + " group by category";
  // ShowMessage("OK 2");
```

```
Form1->ADOQuery1->Open();

 Classes = Form1->ADOQuery1->RecordCount;
 Form1->ADOQuery1->Close() ;

 sql = "Select * from " + table + " order by category";
 Form1->ADOQuery1->SQL->Text = sql;
 Form1->ADOQuery1->Open();

 Rows = Form1->ADOQuery1->RecordCount;
 FieldCount = Form1->ADOQuery1->FieldCount ;
 Columns = FieldCount - 2 ;   //first is recordID, last is categoryID
 Form1->Memo1->Lines->Add("the record size is " + AnsiString(Rows));


 firstClass = Form1->ADOQuery1->Fields->Fields[FieldCount-1]->Value ;


   // get memory for  arrays
  (*numSamplesInClass) = (long*)malloc(Classes* sizeof(long));
  for(j=0; j<Classes; j++)  (*numSamplesInClass)[j]=0;

  (*Yis) = (int*)malloc(Rows*sizeof(int));   // allocate Yis[iRows]

  temp = allocate2DArray(Rows, Columns) ;    // allocate temp[Rows][Columns]

  Xmin=(double*)malloc(Columns* sizeof(double));
  Xmax=(double*)malloc(Columns* sizeof(double));

    // read the first line
     for (j=0; j<Columns; j++){
         temp[0][j] = Form1->ADOQuery1->Fields->Fields[j+1]->Value;
         Xmin[j] = temp[0][j] ;
         Xmax[j] = temp[0][j] ;
      }
      b = Form1->ADOQuery1->Fields->Fields[FieldCount-1]->Value ;
      classIndex = b / 1 ;      // double to int truncate
      classIndex -= firstClass ;  //if firstClass=0 then is ok
      (*Yis)[0] = classIndex ;
      (*numSamplesInClass)[classIndex]++;

      //move to next line
      Form1->ADOQuery1->Next();


  for (i=1; i<Rows; i++) {
        for (j=0; j<Columns; j++) {
            temp[i][j] = Form1->ADOQuery1->Fields->Fields[j+1]->Value;
            if ( temp[i][j] < Xmin[j] ){ Xmin[j] = temp[i][j] ;}
            if ( temp[i][j] > Xmax[j] ){ Xmax[j] = temp[i][j] ;  }
        }
```

```
            // scan a 'double' category label
            b = Form1->ADOQuery1->Fields->Fields[FieldCount-1]->Value;
            classIndex = b / 1 ;      // double to int truncate
            classIndex -= firstClass ;  //if firstClass=0 then is ok
            (*Yis)[i] = classIndex ;
            (*numSamplesInClass)[classIndex]++;

            //move to next line
            Form1->ADOQuery1->Next();
        }

 //    for(j=0; j<Classes; j++) printf("numSamplesInClass[%i]=%i, ", j, (*numSamplesInClass)[j]);

      Form1->Memo1->Lines->Add("load data file, Done!");
        Form1->ADOQuery1->Close() ;


    if (normalizedYesNo) {
        for (i=0; i<Rows; i++)
            for (j=0; j<Columns; j++)
                temp[i][j] = (temp[i][j] - Xmin[j])/(Xmax[j]-Xmin[j]);
      }

    if (Xmin != NULL) free (Xmin);
    if (Xmax != NULL) free (Xmax);

     *dataColumns = Columns ;
     *dataRows = Rows ;
     *dataClasses = Classes ;

      return temp;

}



////////////////////////////////////////////////////////////////////////////////////////

/*****************************************************************************
 this common function used for allocate and return a 2D array[rows][columns]
*****************************************************************************/
double** allocate2DArray(long rows, long columns)
{
   long i;        //to cover both short and long arrays
   double** temp;       //temp points at first [][]
   temp = (double**)malloc(rows*sizeof(double*));   //allocate temp[][]
   assert(temp != NULL);
      // temp[0] points at first [0][], but allocates memory space for all
   temp[0] = (double*)calloc(rows*columns,sizeof(double));  //calloc initialize to zeros
   assert(temp[0] != NULL);
   //assign temp pointers [] to each 'row' []  ( = prev pointer + column number)
   for (i=1; i<rows; i++) temp[i]= temp[i-1] + columns ;
```

```
  return temp;
  //  free(temp[0]);
  //  free(temp);
}

/*****************************************************************************
  this function allocate memory for train test sets initializes
  InitializeTrainTest(vectorXSize, numOfClasses, numSamplesInClass, allData, allY,
          &trainX, &testX, &trainY, &testY, & trainSamplesInClass, percentage)
*****************************************************************************/


void InitializeTrainTest(
   int vectorXSize ,                    // Number of dimensions or variables in x vector
   long dataSetSize ,          // data size
   int numOfClasses ,                   // Number of Classes   (0 to numOfClasses-1)
   long int *numSamplesInClass ,// size of numOfClasses vector contain number
                              // of samples in each class population
   double **allData ,          // allData
   int *allY ,                 // all_Y     class labels
   double ***trainData ,       // output: pointer to **trainData [trainSize][vectorXSize] for memory
allocation
   double ***testData ,        // output: pointer to **testData [testSize][vectorXSize] for memory allocation
   int **trainY ,              // output: pointer to *train_Y [trainSize]for memory allocation
   int **testY,                // output: pointer to *test_Y [testSize] for memory allocation
   long **trainSamplesInClass , // output: pointer to *trainSamplesInClass for memory allocation
   long *trainSize ,           // output: pointer to trainSize
   long *testSize ,            // output: pointer to testSize
   float percentage            // percentage % of train set portions
   )
{
    int c, numberOfColumns ;
    long i, testSetSize, trainSetSize = 0 ;
    numberOfColumns = vectorXSize ;

  //printf("InitializeTrainTest for stratified random spliting\n");

    (*trainSamplesInClass) = (long*) malloc(numOfClasses*sizeof(long));

      for (c=0 ; c<numOfClasses ; c++){  // Evaluate for each class
       (*trainSamplesInClass)[c] = ( (float)numSamplesInClass[c]*percentage ) /1 ;   /*float to int
truncate*/
       //printf("trainSamplesInClass[%d]= %i \n", c,
         //      (*trainSamplesInClass)[c]) ;   //trainSamplesInClass[0][i]

     //   Form1->Memo1->Lines->Add("trainSamplesInClass[" + *trainSamplesInClass)[c]+ "]");
       }
       // find total TrainSet size and total data set
      for (c=0 ; c<numOfClasses ; c++){
         trainSetSize += (*trainSamplesInClass)[c] ;
       }
```

```
        //  testSet size
         testSetSize = dataSetSize - trainSetSize;

        // allocate memory (continous) for trainData , trainY, testData, testY
           (*trainData) = allocate2DArray(trainSetSize, numberOfColumns) ;
           (*testData) =  allocate2DArray(testSetSize, numberOfColumns) ;

           (*trainY) = (int*) malloc(trainSetSize*sizeof(int));
           (*testY) = (int*) malloc(testSetSize*sizeof(int));

      *trainSize = trainSetSize ;
      *testSize = testSetSize ;
       Form1->Memo1->Lines->Add("InitializeTrainTest for stratified random spliting, Done!");
}


//-----------------------------------------------------------------------------
/*****************************************************************************
 SPLIT_TO_TRAIN_TEST WITH STRATIFIED RANDOM SAMPLING
 this function randomly choose a record for train set backets or test set backets
 until both become full with equall percentage for every class so as to
 implement stratification   (equal percentage for every class)
*****************************************************************************/

void StratifiedRandomSplitTT (
    int vectorXSize ,                  // Number of dimensions or variables in x vector
    int numOfClasses ,                 // Number of Classes   (0 to numOfClasses-1)
    long *numSamplesInClass ,   // size of numOfClasses vector contain number
                                // of samples in each class population
    double **allData ,          // allData
    int *allY ,                 // all_Y
    double **trainData ,        // trainData
    double **testData ,         // testData
    int *trainY ,               // train_Y
    int *testY,                 // test_Y
    long *trainSamplesInClass , // ((float)num_Samples[i]*percentage)/1   /*float to int truncate*/
    float trainPercentage       // percentage % of train set portions
    )
{
   long numOfcases, icase;
   int d,  iclass  ;
   double choise;
   long maxTrain, maxTest, currentAll = 0, currentTrain , currentTest;
   long lastTrain = 0,  lastTest =0;

   srand(11001);  // srand ( time(NULL));
   //printf("stratified random spliting\n");

     for (iclass=0 ; iclass<numOfClasses ; iclass++) { // Evaluate for each class

      numOfcases = numSamplesInClass[iclass] ;// Number of data samples of this class
      maxTrain = trainSamplesInClass[iclass];
```

```
    maxTest= numOfcases - maxTrain;

    currentTrain =0;  //limit
    currentTest=0;   //limit
    for (icase=0 ; icase<numOfcases ; icase++) {     //Do all pattern cases in this class
        choise = (double)rand()/(double)RAND_MAX;  //rixnei to zari apo 0.0 eos 1.0

        if (choise<=trainPercentage && currentTrain<maxTrain ) {
        for (d=0 ; d<vectorXSize ; d++) { // for every dimension d
          trainData[lastTrain+currentTrain][d] = allData[currentAll][d] ;
           }
        trainY[lastTrain+currentTrain]= allY[currentAll];
        currentTrain++;
        }
       else  {
            if (currentTest<maxTest) {
                for (d=0 ; d<vectorXSize ; d++) { // for every dimension d
                 testData[lastTest+currentTest][d] = allData[currentAll][d] ;
                 }
                  testY[lastTest+currentTest]= allY[currentAll];
                currentTest++;
            }
           else {
               for (d=0 ; d<vectorXSize ; d++) { // for every dimension d
               trainData[lastTrain+currentTrain][d] = allData[currentAll][d] ;
                }
                trainY[lastTrain+currentTrain]= allY[currentAll];
                currentTrain++;
           }
         }
         currentAll++;

        }//end for cases
      lastTrain +=currentTrain ;
      lastTest +=currentTest ;


     }//end for classes

  Form1->Memo1->Lines->Add("stratified random spliting, Done!");
}
//---------------------------------------------------------------------------
/*****************************************************************************
square of Euclidean distance between two multi-dimensional points
*****************************************************************************/
double EuclideanDistance ( int size, double *vect1, double *vect2  )
{
   int i; double ans=0.0;
   for (i=0; i<size; i++)
      ans += (vect1[i]-vect2[i]) * (vect1[i]-vect2[i]);
   return sqrt(ans);
}
```

```
//-----------------------------------------------------------------------

/*******************************************************************************
 this function is the One Nearest Neighbour classifier
 used here only for comparizon
*******************************************************************************/
void oneNearestNeighbour(
   int dataColumns ,                // Number of dimensions or variables in x vector
   long trainSize ,          // train set size
   double **trainData ,      // trainData[trainSize][dataColumns]
   int *trainY ,             // trainLabels[trainSize]
   double **testData ,               // vectorXSize vector to be classified
   int *testY ,             // trainLabels[trainSize]
   long testSize ,          // train set size
   double *trainError ,       // vector of all the summation units
   double *testError
     )
{
   long   index, i , j;
   double dist, min_dist;

   (*trainError) = (*testError) = 0.0 ;

   // FIND TEST ERROR
  for (i=0 ; i<testSize; i++) {

      index    = 0;
      min_dist = EuclideanDistance(dataColumns, testData[i], trainData[0]);
      for (j=1; j<trainSize; j++) {
         dist = EuclideanDistance(dataColumns, testData[i], trainData[j]);
         /* no need square root */
         if (dist < min_dist) { /* find the min and its array index */
            min_dist = dist;
            index    = j;
          }
       }

     if (trainY[index]!=testY[i])  (*testError)++ ;
  }


      // FIND TRAIN ERROR WITH HOLDOUT
   for (i=0 ; i<trainSize; i++) {
      index    = 0;
      min_dist = EuclideanDistance(dataColumns, trainData[i], trainData[0]);
      for (j=1; j<trainSize; j++) {
        if(i==j) continue ;    //holdout
         dist = EuclideanDistance(dataColumns, trainData[i], trainData[j]);
         /* no need square root */
         if (dist < min_dist) { /* find the min and its array index */
            min_dist = dist;
```

```
            index    = j;
          }
        }
      if (trainY[index]!=trainY[i])  (*trainError)++ ;

   }


}
 /*******************************************************************************
 this function is the (Weighted) Nearest Neighbours classifier
 used here
*******************************************************************************/
void kNearestNeighbours(
   int dataColumns ,                // Number of dimensions or variables in x vector
   long trainSize ,          // train set size
   double **trainData ,       // trainData[trainSize][dataColumns]
   int *trainY ,             // trainLabels[trainSize]
   double **testData ,               // vectorXSize vector to be classified
   int *testY ,             // testLabels[trainSize]
   long testSize ,          // test set size
   double *trainError ,       // vector of all the summation units
   double *testError ,
   long knnSize ,
   int dataClasses ,
   char weightedYes
    )
{
   long   index, i , j , k , m ;
   double dist, min_dist ,  huge = 9999999.9;
   long * knnIndexes = NULL ;
   double  label  ,  maxVoteClass  ;
   int foundInKindex;

   double* labelVotes = NULL ;

   (*trainError) = (*testError) = 0.0 ;
   knnIndexes = (long *)calloc (knnSize,sizeof(long));
   labelVotes =  (double *)calloc (dataClasses,sizeof(double));

   // FIND TEST ERROR
  for (i=0 ; i<testSize; i++) {

      // find k-nearest neighbors of i
           for (k=0; k < knnSize ; k++ ){
     // index[k]    = 0;
      min_dist = huge;
                 for (j=0; j<trainSize; j++) {
                                foundInKindex = 0 ;
                  // an einai knnSize = 1 (k=0) it skips the following line
                                for (m=0; m < k; m++ ){ if (j == knnIndexes[m] )   foundInKindex=1;  }
                                if(foundInKindex==1)continue;
```

```
                    dist = EuclideanDistance(dataColumns, testData[i], trainData[j]);
                                /* no need square root */

                                if (dist < min_dist) { /* find the min and its array index */
                                min_dist = dist;
                                knnIndexes[k]    = j;
                    }
          }  // end for j
          } // end for k

     // use k-nearest neighbors to classify data sample i

          // edw tha broume to label
     for (m=0; m < dataClasses ; m++ ) labelVotes[m] = 0.0 ;

     if (weightedYes == 1)
        for (k=0; k < knnSize ; k++ ) {
           dist = EuclideanDistance(dataColumns, testData[i], trainData[knnIndexes[k]]);
           //  labelVotes[ trainY[ knnIndexes[k] ] ] += 1.0/dist ;
           labelVotes[ trainY[ knnIndexes[k] ] ] +=  exp ( - dist )  ;
        }
          else  for (k=0; k < knnSize ; k++ )
            labelVotes[ trainY[ knnIndexes[k] ] ]++ ; //brikame ti psifise o kathe gitonas
                                    // += 1/distance  geia weighted knn
     // find maximum votes
          maxVoteClass = 0.0 ;
          for (m=0; m < dataClasses; m++ )
                    if (labelVotes[m] > maxVoteClass ) { maxVoteClass = labelVotes[m]; label = m ; }

    if (label!=testY[i])  (*testError)++ ;

  } // end for i


     // FIND TRAIN ERROR WITH HOLDOUT
    // * trainError =0 ;
  for (i=0 ; i<trainSize; i++) {

      // find k-nearest neighbors of i
          for (k=0; k < knnSize ; k++ ){
    // index[k]    = 0;
     min_dist = huge;
                for (j=0; j<trainSize; j++) {  if (i == j)  continue ;
                                foundInKindex = 0 ;
                 // an einai knnSize = 1 (k=0) it skips the following line
                                for (m=0; m < k; m++ ){ if (j == knnIndexes[m] )   foundInKindex=1;  }
                                if(foundInKindex==1)continue;
                    dist = EuclideanDistance(dataColumns, trainData[i], trainData[j]);
                                /* no need square root */

                                if (dist < min_dist) { /* find the min and its array index */
                                min_dist = dist;
```

```
                                knnIndexes[k]    = j;
                        }
            }  // end for j
            } // end for k

        // use k-nearest neighbors to classify data

            // edw tha broume to label
        for (m=0; m < dataClasses ; m++ ) labelVotes[m] = 0.0 ;


        if (weightedYes == 1)
          for (k=0; k < knnSize ; k++ )  {
              dist = EuclideanDistance(dataColumns, trainData[i], trainData[knnIndexes[k]]);
              //labelVotes[ trainY[ knnIndexes[k] ] ] += 1.0/dist ;
              labelVotes[ trainY[ knnIndexes[k] ] ] += exp ( - dist )  ;
           }
            else  for (k=0; k < knnSize ; k++ )
              labelVotes[ trainY[ knnIndexes[k] ] ]++ ; //brikame ti psifise o kathe gitonas
                                          // += 1/distance  geia weighted knn
            maxVoteClass = 0.0 ;
            for (m=0; m < dataClasses; m++ )
                        if (labelVotes[m] > maxVoteClass ) { maxVoteClass = labelVotes[m]; label = m ;}

       if (label!=trainY[i])  (*trainError)++ ;

   } // end for i


    free(knnIndexes);
    free(labelVotes);
}

/*****************************************************************************
 given a dataPoint this function finds the closest to it cluster center
 For N points and k centers has O(Nk) complexity
*****************************************************************************/
int findNearestCluster(
       int numOfClusters,     // Number of clusters
       int   dataColumns,     // Number of dimensions or variables or features
       double  *dataPoint,    // one data point[dataColumns]
       double **clusters      // [numOfClusters][dataColumns]
        )
{
    int   index, i;
    double dist, min_dist;

    /* find the cluster id that has min distance to object */
    index    = 0;
    min_dist = EuclideanDistance(dataColumns, dataPoint, clusters[0]);

    for (i=1; i<numOfClusters; i++) {
```

```
        dist = EuclideanDistance(dataColumns, dataPoint, clusters[i]);
        /* no need square root */
        if (dist < min_dist) { /* find the min and its array index */
            min_dist = dist;
            index    = i;
        }
    }
    return(index);
}
//-----------------------------------------------------------------------
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[Witten and Frank 2005] Witten I.H., and Frank E., “Data Mining: Practical Machine Learning Tools and Techniques”, Second Edition, Morgan Kaufmann Publishers, pp. 162-169, 2005.

[Bhatia and Vandana 2010] Bhatia N., and Vandana A., (2010) “Survey of Nearest Neighbor Techniques”, International Journal of Computer Science and Information Security, Vol. 8, No. 2, pp. 302-305.

[Cayton 2008] Cayton Lawrence. Fast nearest neighbor retrieval for bregman divergences. In Proceedings of the International Conference on Machine Learning, Helsinki, Finland, pp. 112–119, 2008.

[Cayton 2009] Cayton Lawrence. Bregman Proximity Search. PhD thesis, University of California, San Diego, 2009.

[Datar 2004] Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S., (2004). Locality-sensitive hashing scheme based on p-stable distributions. SCG 2004.

[Liu 2004] Liu, T., Moore, A. W., Gray, A., & Yang, K. (2004), “An investigation of practical approximate neighbor algorithms. NIPS.

[Banerjee et al., 2005] Banerjee, A., Merugu, S., Dhillon, I. S., & Ghosh, J. (2005). Clustering with bregman divergences. JMLR.

[Gray and Moore 2000] Gray Alexander and Moore Andrew. ‘N-body’ problems in statistical learning. In Advances in Neural Information Processing Systems, 2000.

[Gray and Moore 2003] Gray Alexander and Moore Andrew. Nonparametric density estimation: Toward computational tractability. In SIAM International Conference on Data Mining, 2003.

[Liu 2006] Liu Ting, Moore Andrew, and Gray Alexander. “New algorithms for efficient high-dimensional nonparametric classification”, Journal of Machine Learning Research, pp. 1135-1158, 2006.

M.H. Dunham, “Data Mining introductory and advanced topics” Prentice Hall 2004.

S. Russell , P. Norvig,“Τεχνητή Νοημοσύνη, μια σύγχρονη προσέγγιση” 2005.

Jiawei Han and Micheline Kamber “Data Mining: Concepts and Techniques”, Morgan Kaufmann Publishers, August 2000.

S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 26:65-74, 1997.

Osmar R. Zaïane “Machine Learning and Data Mining” Dunham University, Lecture notes 2005.

Ricardo Gutierrez-Osuna “Selected Topics in Computer Science” Winter 2002

http://courses.cs.tamu.edu/rgutier/cs790_w02.

A. Silberschatz, H. F. Korth and S. Sudrshan, “Database System Concepts” 4th ed., McGraw Hill, 2002.

R. Elmasri and S.B Navathe. “Θεμελιώδεις αρχές συστημάτων Βάσεων Δεδομένων, Τόμος Α’, 3η Έκδοση Αναθεωρημένη”. Εκδόσεις Δίαυλος 2001.

Jorrod Hollingworth, Bob Swart, Mark Cashman and Paul Gustavson. “Borland C++ Builder 6”. Εκδόσεις Μ. Γκιούρδας

D. Petkovic, “Οδηγός του SQL Server 2000”, Μ. Γκιούρδας, 2000

R. Ramakrishnan, J. Gehrke “Database Management Systems” 2nd ed., McGraw Hill 2002

P.-N. Tan, M.Steinbach, V. Kumar, «Introduction to Data Mining», Addison Wesley, 2006

“Introduction to Pattern Analysis” Ricardo Gutierrez-Osuna

Machine Learning, Neural and Statistical Classification - Editors: D. Michie, D.J. Spiegelhalter, C.C. Taylor - February 17, 1994

http://el.wikipedia.org/wiki/Oracle

http://conf.ellak.gr/2010/wp-content/uploads/2010/05/ellak_parousiasi.B.pdf
Πανεπιστήμιο Κρήτης –Ομάδα Εκπαίδευσης Υ.Κ.

https://forums.oracle.com/forums/thread.jspa?threadID=1083307 δικτιακος τοπος της ετερίας ORACLE