ΤΕΙ ΔΥΤΙΚΗΣ ΕΛΛΑΔΟΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Των

Σκιπητάρης Δανιήλ και Αλέξανδρος Σακιώτης

Μοντελοποίηση Αλγορίθμων Συσταδοποίησης και Εφαρμογές στη Μηχανική Μάθηση

Επιβλέπων Καθηγητής: Ιωάννης Κούγιας

Αντίρριο 2019

1.

Περίληψη

Μηχανική μάθηση ονομάζεται η διαδικασία δημιουργίας μοντέλων ή προτύπων από ένα σύνολο δεδομένων από ένα υπολογιστικό σύστημα. Η μάθηση μπορεί να πραγματοποιηθεί με ή χωρίς επίβλεψη. Στη μάθηση χωρίς επίβλεψη το σύστημα ανακαλύπτει από μόνο του συσχετίσεις ή ομάδες σε ένα σύνολο δεδομένων, δημιουργώντας πρότυπο χωρίς να γνωρίζει αν υπάρχουν, ποια είναι και το πλήθος τους. Οι αλγόριθμοι συσταδοποίησης ανήκουν στην κατηγορία αυτή καθώς οι συστάδες που πρέπει να προκύψουν δεν είναι γνωστές εκ των προτέρων.

Στα πλαίσια της παρούσας εργασίας αυτής έγινε μία βιβλιογραφική μελέτη της μαθηματικής μοντελοποίησης και ανάλυσης αλγορίθμων. Επίσης, διερευνήθηκαν τα μαθηματικά μοντέλα που εφαρμόζονται στο συγκεκριμένο πεδίο και μελετήθηκαν διάφοροι αλγόριθμοι συσταδοποίησης. Τέλος, υλοποιήθηκε ο αλγόριθμος συσταδοποίησης DBSCAN και παρουσιάστηκαν εφαρμογές των αλγορίθμων συσταδοποίησης στη μηχανική μάθηση.

Ευχαριστίες

Αρχικά, θα θέλαμε να ευχαριστήσουμε τον καθηγητή μας Κύριο Ιωάννη Κούγια ο οποίος μας εμπιστεύτηκε το συγκεκριμένο θέμα και μας καθοδηγούσε καθ' όλη τη διάρκεια της εκπόνησής της.

Στη συνέχεια θα θέλαμε να ευχαριστήσουμε τις οικογένειες μας, η οποίες μας στήριξε και ήταν δίπλα μας σε όλα μας τα βήματα.

Ακόμη, δεν θα μπορούσαμε να μην ευχαριστήσουμε τους φίλους μας, οι οποίοι μας υποστήριξαν, μας βοήθησαν και μας χάρισαν ωραίες στιγμές στην μέχρι τώρα πορεία μας.

Περιεχόμενα

Περίληψη 3
Ευχαριστίες 4
Περιεχόμενα 5
Κατάλογος Πινάκων 7
Κατάλογος Εικόνων 8
Κεφάλαιο 1 9
Εισαγωγή 9
1.1 Αντικείμενο διπλωματικής εργασίας 9
1.2 Σύνοψη διπλωματικής εργασίας 9
Κεφάλαιο 2 11
Θεωρητικό υπόβαθρο 11
2.1 Μαθηματική μοντελοποίηση 11
2.2 Μηχανική μάθηση 13
Κεφάλαιο 3 14
Αλγόριθμοι Συσταδοποίησης 14
3.1 Ιεραρχικοί 15
3.2 Διαμεριστικοί 16
3.3 Μοντέλων 17
3.4 Πλέγματος 18
3.5 Πυκνότητας 19
Κεφάλαιο 4 23
Υλοποίηση αλγορίθμου DBSCAN 23
Κεφάλαιο 5 26
Πειράματα και Εφαρμογές 26
5.1 Δεδομένα με διακριτές συστάδες 26
5.2 Δεδομένα με κανονική κατανομή 29
5.3 Δεδομένα με ποικίλη πυκνότητα 31
5.4 Χρόνος εκτέλεσης 39

5.5 Εφαρμογές στη Μηχανική Μάθηση 41
Κεφάλαιο 6 42
Συμπεράσματα 42
Βιβλιογραφία 43
Παράρτημα 44
Αρχείο dbscan.c 44
Αρχείο plot_data.m 48
Αρχείο create_data_clear_clusters.m 50
Αρχείο create_data_normal_random.m 51

Κατάλογος Πινάκων

Πίνακας 1: Κύρια ρουτίνα 23
Πίνακας 2: Αλγόριθμος DBSCAN 25
Πίνακας 3: Συνάρτηση range_query 25

Κατάλογος Εικόνων

Εικόνα 1: Διάγραμμα ροής μοντελοποίησης κι ανάπτυξης αλγορίθμου. 12
Εικόνα 2: Παράδειγμα συσταδοποίησης. 14
Εικόνα 3: Ιεραρχικοί αλγόριθμοι. 16
Εικόνα 4: Παράδειγμα συσταδοποίησης k-means 17
Εικόνα 5: Παράδειγμα συσταδοποίησης EM. 18
Εικόνα 6: Αλγόριθμος STING 19
Εικόνα 7: Παράδειγμα συσταδοποίησης DBSCAN. 21
Εικόνα 8: 1^{ο} σετ, EPS=1, MINPTS=5 27
Εικόνα 9: 1ο σετ, EPS=3, MINPTS=5 27
Εικόνα 10: 1ο σετ, EPS=10, MINPTS=5 28
Εικόνα 11: 1ο σετ, EPS=50, MINPTS=5 28
Εικόνα 12: 2ο σετ, EPS=1, MINPTS=5 30
Εικόνα 13: 2ο σετ, EPS=3, MINPTS=5 30
Εικόνα 14: 2ο σετ, EPS=10, MINPTS=5 31
Εικόνα 15: 3ο σετ 32
Εικόνα 16: 3ο σετ, EPS=50, MINPTS=5 33
Εικόνα 17: 3ο σετ, EPS=5, MINPTS=5 34
Εικόνα 18: 3ο σετ, EPS=10, MINPTS=3 35
Εικόνα 19: 3ο σετ, EPS=60, MINPTS=5 36
Εικόνα 20: 3ο σετ, EPS=60, MINPTS=10 37
Εικόνα 21: 3ο σετ, k-means, k=2 38
Εικόνα 22: 3ο σετ, k-means, k=4 39
Εικόνα 23: Χαρακτηριστικά επεξεργαστή 40
Εικόνα 24: Χρόνος εκτέλεσης 40

Κεφάλαιο 1

Εισαγωγή

Στο παρόν κεφάλαιο γίνεται μια μικρή παρουσίαση του αντικειμένου της εργασίας, της δομής αυτού και των σκοπών του. Ακόμη, παρουσιάζεται η σύνοψη των επόμενων κεφαλαίων ώστε να μπορέσει ο αναγνώστης να κατατοπιστεί επαρκώς σχετικά με το τι πρόκειται να διαβάσει στις επόμενες σελίδες.

1.1 Αντικείμενο διπλωματικής εργασίας

Αντικείμενο της εργασίας αυτής αποτελεί η βιβλιογραφική μελέτη της μαθηματικής μοντελοποίησης και ανάλυσης αλγορίθμων, δίνοντας έμφαση στους αλγόριθμους συσταδοποίησης. Επιπλέον, θα διερευνηθούν τα μαθηματικά μοντέλα που εφαρμόζονται στο συγκεκριμένο πεδίο και θα μελετηθούν διάφοροι αλγόριθμοι συσταδοποίησης. Επιπροσθέτως, θα υλοποιηθεί ο αλγόριθμος συσταδοποίησης DBSCAN και θα παρουσιαστούν πρακτικές εφαρμογές των αλγορίθμων συσταδοποίησης στη μηχανική μάθηση.

1.2 Σύνοψη διπλωματικής εργασίας

Η παρούσα εργασία αποτελείται από έξι κεφάλαια. Στο πρώτο κεφάλαιο περιγράφεται το αντικείμενο της εργασίας. Στο δεύτερο κεφάλαιο πραγματοποιείται ανασκόπηση της βιβλιογραφίας όσον αφορά τη μοντελοποίηση κι ανάλυση αλγορίθμων, καθώς και της μηχανικής μάθησης. Στο τρίτο κεφάλαιο παρουσιάζονται διάφοροι αλγόριθμοι συσταδοποίησης. Στο τέταρτο κεφάλαιο παρουσιάζεται η υλοποίηση του αλγορίθμου συσταδοποίησης DBSCAN. Στο πέμπτο κεφάλαιο

παρουσιάζονται τα πειραματικά αποτελέσματα που προέκυψαν από την υλοποίηση, καθώς και πρακτικές εφαρμογές των αλγορίθμων συσταδοποίησης στη μηχανική μάθηση. Στο έκτο και τελευταίο κεφάλαιο, καταγράφονται τα συμπεράσματα της παρούσας εργασίας.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

Στο παρόν κεφάλαιο γίνεται μια μικρή παρουσίαση του απαιτούμενου θεωρητικού υπόβαθρου που στόχο έχει την καλύτερη κατανόηση του αναγνώστη για τα επόμενα κεφάλαια.

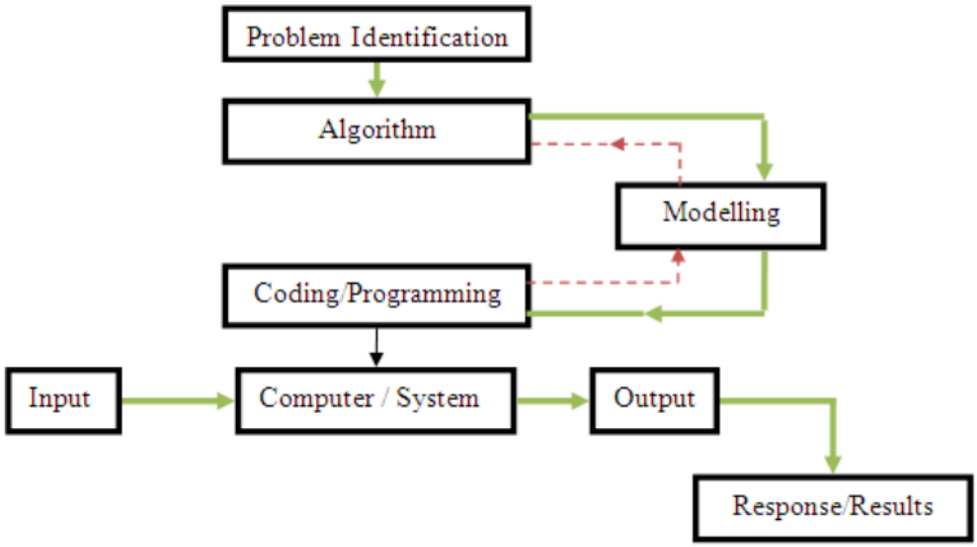
2.1 Μαθηματική μοντελοποίηση

Η μαθηματική μοντελοποίηση είναι η έκφραση των βασικών ιδιοτήτων/νόμων που διέπουν διεργασίες της φύσης με κατάλληλες εξισώσεις και η μελέτη/επίλυση τους υπό αρχικές συνθήκες με κατάλληλες μαθηματικές μεθόδους. Η αριθμητική μοντελοποίηση είναι η χρήση αριθμητικών/υπολογιστικών τεχνικών με την εισαγωγή κατάλληλων οριακών συνθηκών για την αριθμητική επίλυση προβλημάτων χρησιμοποιώντας ηλεκτρονικό υπολογιστή. Η μαθηματική μοντελοποίηση παρουσιάζει εφαρμογές σε πολλές επιστήμες όπως η μηχανολογία, γενετική, φυσική, ιατρική κ.α. Η μοντελοποίηση ακολουθείται κατά κανόνα από εξομοιώσεις για την αξιολόγηση και τυχόν επαναπροσδιορισμό του μοντέλου, εξασφαλίζοντας την ποιότητας ενός μοντέλου.

Η μαθηματική μοντελοποίηση είναι σημαντική για τους ακόλουθους λόγους:

- Δυνατότητα αναπαράστασης της απόδοσης ενός συστήματος ή μιας διεργασίας.
- Δημιουργία χρήσιμων ψηφιακών δεδομένων.

- Αναπαράσταση πραγματικών φαινομένων με μοναδική συμπεριφορά όπως:
  - Φυσικά συστήματα.
  - Οικονομικά συστήματα
  - Κοινωνικά συστήματα.
- Πιθανοτική πρόβλεψη αποτελεσμάτων.
- Η οικονομική μοντελοποίηση είναι πιο αποδοτική από τα πειράματα στην πραγματική ζωή.
- Η μοντελοποίηση στη Μηχανική επιτρέπει την αξιολόγηση μετασχηματισμών που πρόκειται να εφαρμοστούν.
- Τα μοντέλα είναι κρίσιμα στην ανάλυση και πραγματοποίηση αποφάσεων.



Εικόνα 1: Διάγραμμα ροής μοντελοποίησης κι ανάπτυξης αλγορίθμου.

Στην Εικόνα 1 (Mapoka, 2013) παρουσιάζεται το διάγραμμα ροής στην ανάπτυξη ενός αλγορίθμου σε σχέση με τη μοντελοποίηση ενός προβλήματος και την υλοποίηση μιας λύσης σε υπολογιστικό περιβάλλον που οδηγεί στα τελικά αποτελέσματα.

2.2 Μηχανική μάθηση

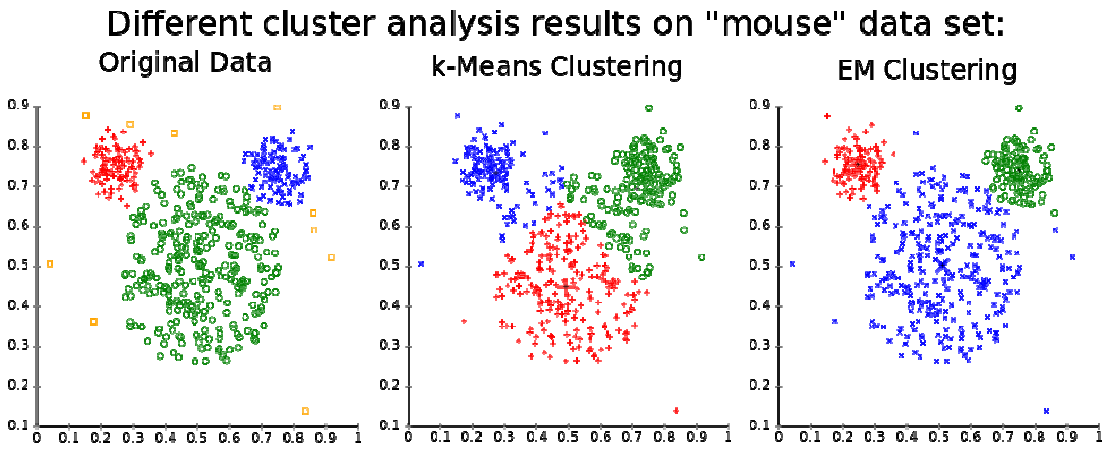
Ο άνθρωπος από την αρχή της ύπαρξης του προσπαθεί να κατανοήσει το περιβάλλον στο οποίο ζει παρατηρώντας το και δημιουργώντας μια αφαιρετική εκδοχή του που ονομάζεται μοντέλο. Η δημιουργία ενός τέτοιου μοντέλου ονομάζεται επαγωγική μάθηση, ενώ η γενικότερη διαδικασία ονομάζεται επαγωγή. Επίσης, ο άνθρωπος έχει τη δυνατότητα να οργανώνει και να συσχετίζει τις εμπειρίες και τις παραστάσεις του δημιουργώντας νέες δομές που ονομάζονται πρότυπα. Η δημιουργία μοντέλων ή προτύπων από ένα σύνολο δεδομένων από ένα υπολογιστικό σύστημα ονομάζεται μηχανική μάθηση (machine learning).

Οι δύο κύριες κατηγορίες μηχανικής μάθησης είναι η μάθηση με επίβλεψη ή με παραδείγματα και η μάθηση χωρίς επίβλεψη ή από παρατήρηση. Στην πρώτη περίπτωση το σύστημα μαθαίνει μία έννοια ή συνάρτηση από ένα σύνολο δεδομένων που αποτελεί περιγραφή ενός μοντέλου. Στη μάθηση χωρίς επίβλεψη το σύστημα ανακαλύπτει από μόνο του συσχετίσεις ή ομάδες σε ένα σύνολο δεδομένων, δημιουργώντας πρότυπο χωρίς να γνωρίζει αν υπάρχουν, ποια είναι και το πλήθος τους. Οι αλγόριθμοι συσταδοποίησης που παρουσιάζονται στο επόμενο κεφάλαιο ανήκουν στην κατηγορία της μάθησης χωρίς επίβλεψης καθώς οι συστάδες που πρέπει να προκύψουν δεν είναι γνωστές εκ των προτέρων.

Κεφάλαιο 3

Αλγόριθμοι Συσταδοποίησης

Μία συστάδα (cluster) είναι μία ομάδα αντικειμένων δεδομένων, τα οποία έχουν παρόμοια χαρακτηριστικά με τα υπόλοιπα της ίδιας συστάδας και διαφορετικά των αντικειμένων που ανήκουν σε διαφορετικές συστάδες. Η συσταδοποίηση είναι η διαδικασία δημιουργίας αυτών των ομάδων από ένα σύνολο αντικειμένων κι έχει εφαρμογή σε πολλά πεδία εφαρμογών, όπως βιολογία, ιατρική, μηχανική μάθηση κ.α. Στη βιβλιογραφία μπορούν να βρεθούν διάφορες προσεγγίσεις στο ζήτημα, καθώς η συσταδοποίηση μπορεί να γίνει θέτοντας διάφορα κριτήρια. Στην Εικόνα 6 παρουσιάζεται ένα παράδειγμα συσταδοποίησης που προκύπτει από δύο διαφορετικούς αλγόριθμους. Στο συγκεκριμένο παράδειγμα φαίνεται πως ο αλγόριθμος EM λειτουργεί καλύτερα έναντι του k-means. Οι δύο αλγόριθμοι θα αναφερθούν μεταξύ άλλων στις επόμενες ενότητες, όπου θα παρουσιαστούν οι σημαντικότερες κατηγορίες αλγορίθμων συσταδοποίησης.



Εικόνα 2: Παράδειγμα συσταδοποίησης.

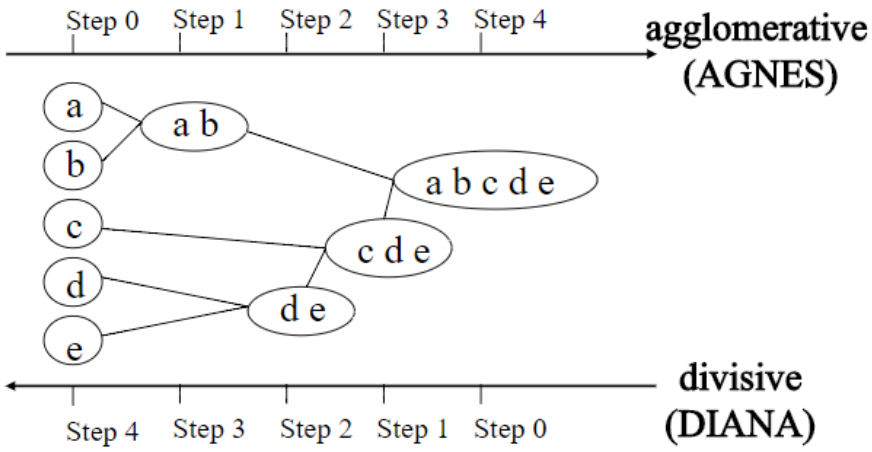
3.1 Ιεραρχικοί

Οι ιεραρχικοί (hierarchical) αλγόριθμοι βασίζονται στη συνδεσιμότητα και κατά την εκτέλεση τους παράγεται ένα δενδρόγραμμα που αναπαριστά μια ιεραρχική ομαδοποίηση των αντικειμένων του συνόλου με βάση κάποιο κριτήριο. Η ρίζα του δέντρου περιέχει όλα τα αντικείμενα, τα φύλλα του αποτελούνται από ένα μόνο αντικείμενο ενώ οι ενδιάμεσοι κόμβοι αναπαριστούν τις συστάδες που δημιουργούνται από την ένωση πολλών φύλλων. Οι αλγόριθμοι αυτής της κατηγορίας δεν χρειάζονται εκ των προτέρων το πλήθος των συστάδων ως είσοδο. Κάθε επίπεδο παριστάνει ένα κατώφλι απόστασης και αν δύο συστάδες έχουν απόσταση μικρότερη από αυτό το κατώφλι, τότε συνενώνονται σε μία συστάδα.

Στους ιεραρχικούς αλγόριθμους υπάρχει η συσσωρευτική προσέγγιση (bottom-up) και η διαιρετική προσέγγιση (top-down). Στην πρώτη περίπτωση κάθε αντικείμενο τοποθετείται αρχικά στη δική του συστάδα, στη συνέχεια οι ατομικές συστάδες ενώνονται επαναληπτικά σε όλο και μεγαλύτερες μέχρι όλα τα αντικείμενα να βρεθούν σε μία συστάδα. Αντίθετα, στη διαιρετική μέθοδο ακολουθείται η αντίστροφη διαδικασία με τα αντικείμενα να ξεκινάνε από μία κοινή συστάδα, η οποία στη συνέχεια διασπάται σε όλο και μικρότερες συστάδες. Οι διαιρετικοί αλγόριθμοι χρησιμοποιούνται πιο σπάνια και χρειάζονται κριτήριο τερματισμού.

Σημαντικοί ιεραρχικοί αλγόριθμοι είναι οι AGNES, DIANA, BIRCH, CURE και ROCK με τους δύο πρώτους να είναι οι πιο διαδεδομένοι. Ο συσσωρευτικός ιεραρχικός αλγόριθμος AGNES (Agglomerative Nesting) προτάθηκε το 1990 κι έχει εφαρμογή σε προγράμματα στατιστικής ανάλυσης. Στον AGNES η απόσταση μεταξύ δύο συστάδων είναι η μικρότερη απόσταση από κάθε αντικείμενο της μίας συστάδας με κάθε αντικείμενο της άλλης. Η συγκεκριμένη μέθοδος υπολογισμού της απόστασης μεταξύ συστάδων ονομάζεται απλού συνδέσμου (single link), ενώ χρησιμοποιείται ο πίνακας ανομοιότητας σημείων (dissimilarity matrix), για την ένωση αντικειμένων που έχουν τη μικρότερη ανομοιότητα. Ο διαιρετικός αλγόριθμος DIANA (Divisive Analysis), ο οποίος προτάθηκε από τους δημιουργούς του AGNES το 1990, λειτουργεί με τον αντίστροφο τρόπο από τον AGNES. Ο τρόπος λειτουργίας των δύο αλγορίθμων απεικονίζεται στην Εικόνα 3. Οι ατομικές συστάδες του

γραφήματος αποτελούν ταυτόχρονα το πρώτο βήμα του AGNES και το τελευταίο βήμα του DIANA.



Εικόνα 3: Ιεραρχικοί αλγόριθμοι.

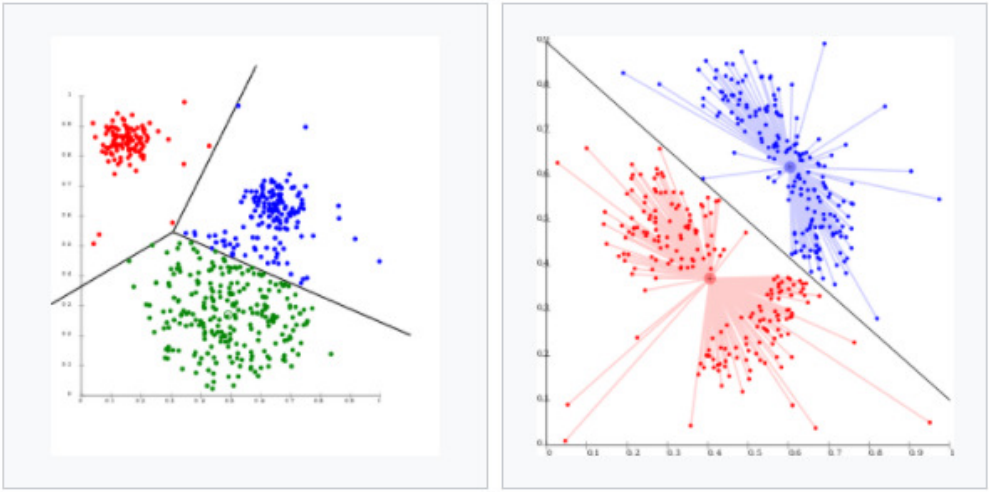
3.2 Διαμεριστικοί

Στους διαμεριστικούς (partitioning) αλγορίθμους κατασκευάζονται διάφορες διαμερίσεις (partitions) με βάση την απόστασή τους, οι οποίες αξιολογούνται με βάση κάποιο κριτήριο. Το επιθυμητό πλήθος *k* των διαμερίσεων δίνεται ως είσοδος στον αλγόριθμο σε αντίθεση με τους ιεραρχικούς αλγόριθμους. Η βέλτιστη λύση θα ήταν η εξαντλητική αναζήτηση όλων των διαμερίσεων, αλλά προτιμώνται ευρετικές (heuristic) μέθοδοι για εξοικονόμηση πολύτιμου υπολογιστικού χρόνου. Οι δύο πιο γνωστοί διαμεριστικοί αλγόριθμοι είναι k-means και ο k-medoids.

Ο αλγόριθμος k-means προτάθηκε το 1967 (MacQueen, 1967) και κάθε συστάδα αντιπροσωπεύεται από το κέντρο της. Ο αλγόριθμος k-means εκτελεί τα εξής βήματα:

1. Διαχωρισμός των αντικειμένων σε k τυχαία μη κενά υποσύνολα.
2. Υπολογισμός κέντρου κάθε συστάδας (το μέσο σημείο).
3. Ανάθεση κάθε αντικειμένου στη συστάδα με το πιο κοντινό κέντρο.
4. Αν δεν υπάρχει δυνατότητα νέας ανάθεσης, τερματισμός. Αλλιώς πήγαινε στο βήμα 2.

Ο αλγόριθμος k-medoids λειτουργεί όπως και ο k-means μόνο που για το κέντρο της κάθε συστάδας επιλέγεται ένα από τα αντικείμενα που ανήκουν σε αυτή. Στην Εικόνα 4 φαίνονται τα αποτελέσματα του αλγορίθμου k-means σε δύο διαφορετικά σετ δεδομένων. Σε συστάδες που δημιουργούνται λόγω πυκνότητας (δεξιά γραφική) ο αλγόριθμος παράγει λάθος αποτελέσματα, ενώ αριστερά τα αποτελέσματα είναι καλύτερα, αλλά όχι άριστα καθώς κάποια μπλε σημεία θα μπορούσαν να ανήκουν στην πράσινη συστάδα.



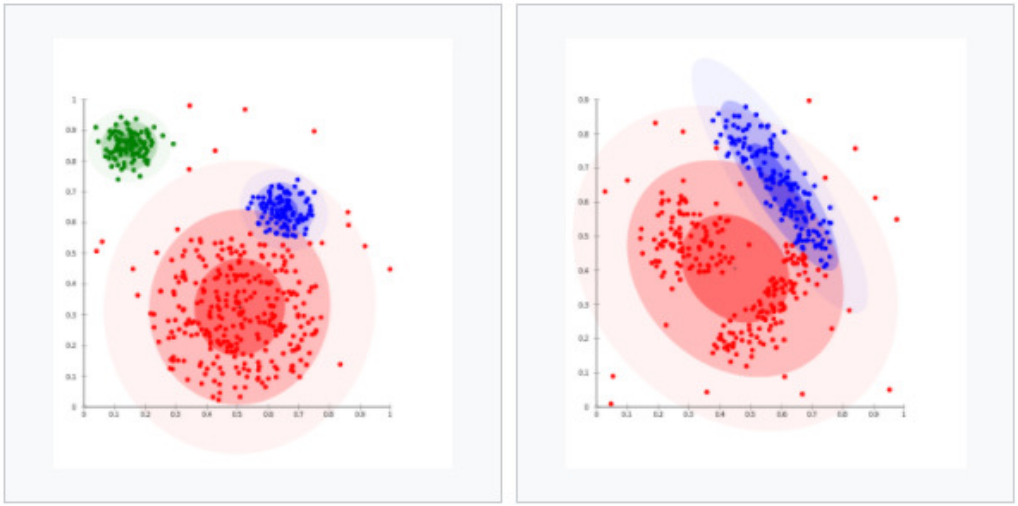
Εικόνα 4: Παράδειγμα συσταδοποίησης k-means

3.3 Μοντέλων

Στους αλγορίθμους μοντέλων (model-based) θεωρείται ένα μοντέλο για κάθε συστάδα και σκοπός είναι να βρεθούν τα αντικείμενα που ταιριάζουν καλύτερα (best fit) στο μοντέλο αυτό. Οι αλγόριθμοι μοντέλων βασίζονται στην υπόθεση ότι τα δεδομένα παράγονται από ένα μείγμα υποκείμενων πιθανοτικών κατανομών.

Οι πιο γνωστοί αλγόριθμοι μοντέλων είναι ο αλγόριθμος EM (Expectation Maximization) και ο HMBC (Hierarchical Model-Based Clustering). Ο αλγόριθμος EM δημιουργεί τις συστάδες τοποθετώντας επαναληπτικά κάθε αντικείμενο στη συστάδα στην οποία αναμένεται να ανήκει με βάση τις πιθανότητες, ενώ ο HBMC ενώνει ζεύγη συστάδων που αναλογούν στην ελάχιστη μείωση της μεταξύ τους πιθανοφάνειας. Στην Εικόνα 5 παρουσιάζεται ένα παράδειγμα εκτέλεσης του αλγόριθμου EM σε δύο διαφορετικά σετ δεδομένων. Στα αριστερά τα δεδομένα

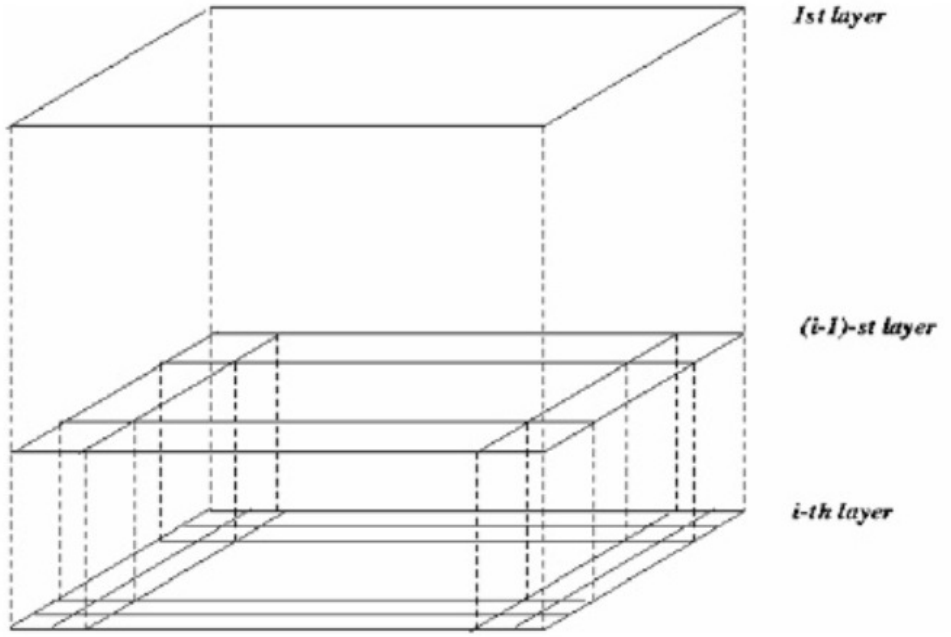
ακολουθούν τη γκαουσιανή κατανομή και τα αποτελέσματα είναι ικανοποιητικά, ενώ στα δεξιά που οι συστάδες δημιουργούνται με βάση την πυκνότητα ο τρόπος λειτουργίας του EM οδηγεί σε λανθασμένα αποτελέσματα.



Εικόνα 5: Παράδειγμα συσταδοποίησης EM.

3.4 Πλέγματος

Οι αλγόριθμοι πλέγματος (grid-based) βασίζονται σε μία δομή πλέγματος πολλαπλών επιπέδων. Αρχικά ο χώρος συσταδοποίησης διαιρείται σε κελιά κι έπειτα εφαρμόζεται η μέθοδο συσταδοποίησης στα κελιά που έχουν ήδη δημιουργηθεί. Χαρακτηριστικό παράδειγμα της κατηγορίας αυτής είναι ο αλγόριθμος STING (A Statistical Information Grid Approach) ο οποίος προτάθηκε το 1997 (Wei Wang, 1997). Ο χώρος στην περίπτωση αυτή διαιρείται σε ορθογώνια κελιά, όπως φαίνεται στην Εικόνα 6.



Εικόνα 6: Αλγόριθμος STING

Κάθε επίπεδο κελιών αντιστοιχεί σε διαφορετικά επίπεδα ανάλυσης. Κάθε κελί χωρίζεται σε μικρότερα κελιά στο αμέσως χαμηλότερο επίπεδο, ενώ υπολογίζεται και αποθηκεύεται εκ των προτέρων στατιστική πληροφορία για κάθε κελί ώστε να χρησιμοποιηθεί αργότερα. Ο υπολογισμός των παραμέτρων (μέσος όρος, μέγιστη κι ελάχιστη τιμή, τύπος κατανομής) των κελιών στο υψηλότερο επίπεδο είναι πιο εύκολος από αυτές των κελιών στα χαμηλότερα επίπεδα. Στον αλγόριθμο πλέγματος STING χρησιμοποιείται μία προσέγγιση top-down για να απαντηθούν τα ερωτήματα σχετικά με τα χωρικά δεδομένα.

3.5 Πυκνότητας

Οι αλγόριθμοι πυκνότητας (density-based) βασίζονται στη συνδεσιμότητα και στις συναρτήσεις πυκνότητας. Ως πυκνότητα ορίζεται το ελάχιστο πλήθος αντικειμένων που απέχουν συγκεκριμένη απόσταση μεταξύ τους. Οι αλγόριθμοι πυκνότητας το μόνο που χρειάζονται ως είσοδο είναι οι ακόλουθες δύο παραμέτρους:

- MinPts: Ελάχιστο πλήθος αντικειμένων σε μία συστάδα.

- Eps: Μέγιστη ακτίνα της γειτονιάς, όπου η γειτονιά είναι η συστάδα που έχει σαν κέντρο το συγκεκριμένο αντικείμενο.

Οι αλγόριθμοι αυτής της κατηγορίας έχουν διάφορα πλεονεκτήματα, όπως η δυνατότητα ανακάλυψης συστάδων με αυθαίρετες μορφές, η δυνατότητα αναγνώρισης των ακραίων αντικειμένων, το γεγονός ότι δεν επηρεάζονται από το θόρυβο και δεν χρειάζονται το πλήθος των συστάδων εξ αρχής ως είσοδο.

Ένα αντικείμενο *p* είναι άμεσα προσβάσιμο με βάση την πυκνότητα (directly density-reachable) από ένα αντικείμενο *q* αν:

(α) Το αντικείμενο *p* είναι εντός της ακτίνας που ξεκινάει από το *q*.
(β) Η γειτονιά του *q* έχει τουλάχιστον MinPts αντικείμενα.

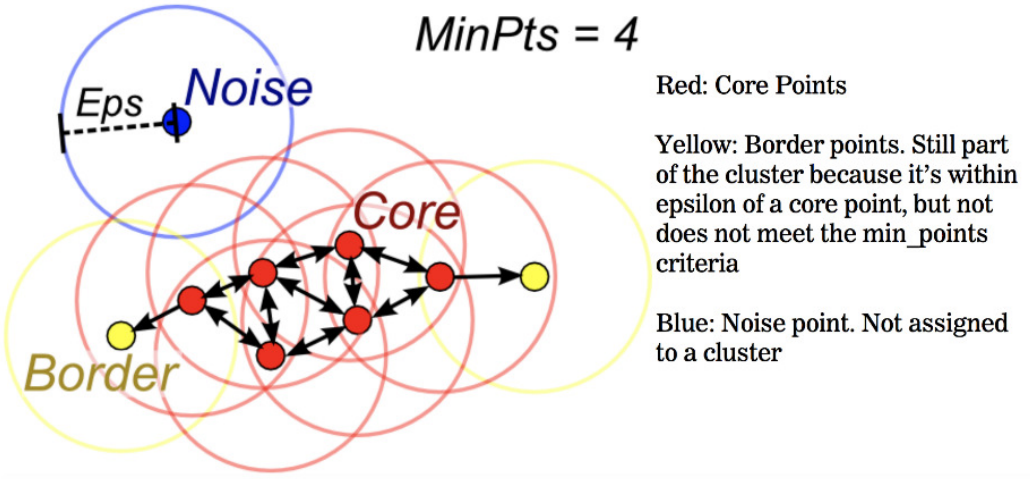
Ένα αντικείμενο *p* είναι προσβάσιμο με βάση την πυκνότητα (density-reachable) από ένα αντικείμενο *q* αν υπάρχει μία αλυσίδα αντικειμένων στην οποία κάθε αντικείμενο είναι άμεσα προσβάσιμο με βάση την πυκνότητα από το επόμενο του.

Ένα αντικείμενο *p* είναι συνδεδεμένο με βάση την πυκνότητα (density-connected) με ένα αντικείμενο *q* αν υπάρχει ένα αντικείμενο *r* από το οποίο τα αντικείμενα *p* και *q* είναι προσβάσιμα με βάση την πυκνότητα.

Ο αλγόριθμος πυκνότητας DBSCAN (Density Based Spatial Clustering of Applications with Noise) προτάθηκε το 1996 (Martin Ester, 1996). Ο DBSCAN διαλέγει στην τύχη ως σημείο εκκίνησης ένα αντικείμενο και δημιουργεί τη γειτονιά του με βάση τα Eps, MinPts. Αν η γειτονιά περιέχει αρκετά αντικείμενα, τότε μία συστάδα δημιουργείται. Σε αντίθετη περίπτωση, το αντικείμενο χαρακτηρίζεται προσωρινά ως θόρυβος, αλλά μπορεί στη συνέχεια να συμπεριληφθεί σε μία έγκυρη γειτονιά που ξεκινά από ένα διαφορετικό αντικείμενο. Σε μία έγκυρη συστάδα οι γειτονιές που ξεκινάνε από τα αντικείμενα της, ενώνονται με την αρχική συστάδα εφόσον τηρούν τις προϋποθέσεις μέχρι να μη μπορούν να προστεθούν νέα αντικείμενα. Τότε, ο DBSCAN ξεκινά μία νέα συστάδα από ένα νέο τυχαίο αντικείμενο που δεν έχει επισκεφτεί μέχρι εκείνο το σημείο κι επαναλαμβάνει τη διαδικασία μέχρι να μην μπορεί να σχηματίσει νέες συστάδες.

Ο DBSCAN μπορεί με αυτό τον τρόπο να ανακαλύπτει συστάδες τυχαίου σχήματος σε χωρικές βάσεις δεδομένων με θόρυβο, όμως όταν η πυκνότητα των αντικειμένων ποικίλλει κατά πολύ, τότε δεν πετυχαίνει αρκετά ικανοποιητικά αποτελέσματα. Σε κάθε περίπτωση χρειάζεται προσοχή στην επιλογή των κατάλληλων παραμέτρων ανάλογα με τα χαρακτηριστικά των αντικειμένων προς συσταδοποίηση.

Στην Εικόνα 7 (Πηγή: https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818) παρουσιάζεται ένα παράδειγμα συσταδοποίησης από τον DBSCAN. Τα κόκκινα και κίτρινα αντικείμενα ανήκουν σε μία συστάδα, με τα κόκκινα να είναι αντικείμενα πυρήνα, επειδή ξεκινώντας από το καθένα από αυτά σε απόσταση Eps υπάρχουν MinPts=4 αντικείμενα (μαζί με το ίδιο). Το μπλε σημείο χαρακτηρίζεται ως θόρυβος από τον DBSCAN δίνοντας τις συγκεκριμένες παραμέτρους. Αν το Eps είχε μεγαλύτερη τιμή, τότε ακόμα και το μπλε θα συμπεριλαμβανόταν στη συστάδα.



Εικόνα 7: Παράδειγμα συσταδοποίησης DBSCAN.

Ο αλγόριθμος OPTICS (Ordering Points To Identify the Clustering Structure) προτάθηκε το 1999 (M. Ankerst, 1999) και αντιμετωπίζει την αδυναμία του DBSCAN να πετύχει ικανοποιητικά αποτελέσματα όταν η πυκνότητα των αντικειμένων ποικίλλει κατά πολύ. Το σκεπτικό του αλγορίθμου είναι παρόμοιο με του DBSCAN, αλλά αντί να χρησιμοποιεί μία προκαθορισμένη ακτίνα Eps, ο

OPTICS ταξινομεί έτσι τα αντικείμενα ώστε τα αντικείμενα που είναι χωρικά πιο κοντά να γίνονται γείτονες κατά τη διάρκεια της ταξινόμησης.

Κεφάλαιο 4

Υλοποίηση αλγορίθμου DBSCAN

Στο τρέχον κεφάλαιο παρουσιάζουμε την υλοποίηση του αλγορίθμου DBSCAN που πραγματοποιήθηκε στην παρούσα εργασία. Στο Παράρτημα βρίσκονται ολοκληρωμένος ο κώδικας C που γράφτηκε στο περιβάλλον Eclipse Luna.

Η κύρια ρουτίνα παρατίθεται στον Πίνακας 1 όπου αρχικά καλείται η συνάρτηση read_data() που διαβάζει τα δεδομένα εισόδου, έπειτα καλείται ο αλγόριθμος DBSCAN και τέλος καταγράφονται τα αποτελέσματα σε ένα αρχείο με τη βοήθεια της συνάρτησης write_data().

```
int main(){
	read_data();
	dbscan();
	write_data();
	return 0;
}
```

Πίνακας 1: Κύρια ρουτίνα

Βασικό εργαλείο πριν την υλοποίηση του αλγορίθμου είναι η δημιουργία μίας δομής (struct) που περιγράφει το σημείο και όλες τις απαραίτητες πληροφορίες. Η δομή του σημείου έχεις τις ακόλουθες ιδιότητες:

- (x,y): συντεταγμένες του σημείου.
- label: Ο αριθμός της συστάδας στην οποία ανήκει το σημείο. Αν είναι -1, δεν έχει οριστεί ακόμα, ενώ αν είναι 0 πρόκειται για θόρυβο.

- neighbors: Τα IDs των γειτόνων του σημείου. (Το ID ταυτίζεται με τη θέση του αρχικού πίνακα την οποία καταλαμβάνει κάθε σημείο).
- neighbors_number: Το πλήθος των γειτόνων του σημείου.
- neighbors_distance: Η απόσταση του κάθε γείτονα από το σημείο.

Στον Πίνακας 2 παρατίθεται το κύριο σώμα του αλγορίθμου DBSCAN. Ο αλγόριθμος διατρέχει ένα ένα τα σημεία εισόδου και εκτελείται η ακόλουθη διαδικασία:

1. Αν το label έχει ήδη οριστεί, πήγαινε στο επόμενο σημείο.
2. Βρες τους γείτονες του σημείου (range_query).
3. Αν οι γείτονες του σημείου είναι λιγότεροι από MINPTS, τότε χαρακτήρισε το σημείο ως θόρυβο και πήγαινε στο επόμενο σημείο.
4. Αύξησε το πλήθος της συστάδας κατά 1 και θέσε τον αριθμό αυτό στο τρέχων σημείο ως τη συστάδα στην οποία ανήκει.
5. Για κάθε ένα από τα γειτονικά σημεία του τρέχοντος σημείου:
    a. Αν δεν έχει οριστεί το label, δώσε του το label του τρέχοντος σημείου.
    b. Βρες τους γείτονες του γειτονικού σημείου (range_query).
    c. Αν το γειτονικό σημείο έχει πάνω από MINPTS γείτονες, πρόσθεσε στον πίνακα data_seed τους γείτονες του γειτονικού σημείου.

```
Void dbscan(){
      int cluster_counter = 0;
      int i = 0;
      int j = 0;
      int k = 0;

      // For each data point
      for ( i=0; i<NUMBER_OF_POINTS; i++ ){
            if ( data[i].label != -1 )
                  continue;
            range_query(i);
            if ( data[i].neighbors_number < MINPTS ){
                  data[i].label = 0;
                  continue;
            }
            cluster_counter++;
            data[i].label = cluster_counter;
```

```
            // Expand neighbors
            for ( k=0; k<NUMBER_OF_POINTS; k++ ){
                  data_seed[k] = data[i].neighbors[k];
            }

            for ( j=0; j<data[i].neighbors_number; j++ ){
                  int neighbor_index = data[i].neighbors[j];
                  if ( data[neighbor_index].label == 0 )
                        data[neighbor_index].label = cluster_counter;
                  if ( data[neighbor_index].label != -1 )
                        continue;
                  data[neighbor_index].label = cluster_counter;
                  range_query(neighbor_index);
                  if ( data[neighbor_index].neighbors_number >=
MINPTS ){

                        for ( k=0; k<NUMBER_OF_POINTS; k++ ){
                              if ( data_seed[k] != -1 )
                                    continue;
                              data_seed[k] =
data[neighbor_index].neighbors[k];
                        }
                  }
            }
      }
}
```

Πίνακας 2: Αλγόριθμος DBSCAN

Στον Πίνακας 3 παρατίθεται η συνάρτηση range_query που αναφέρθηκε προηγουμένως. Στόχος της είναι να βρει τους γείτονες του σημείου και την απόστασή τους από το σημείο που λαμβάνει ως είσοδο.

```
void range_query(int point){
      int i = 0;
      int neighbors_counter = 0;

      for ( i = 0; i < NUMBER_OF_POINTS; i++ ){
            // Skip input point
            if ( i == point )
                  continue;
            double distance =
calculate_distance(data[point],data[i]);
            if ( distance <= EPS ){
                  data[point].neighbors[neighbors_counter] = i;
                  data[point].neighbors_distance[neighbors_counter] =
distance;
                  neighbors_counter++;
            }
      }

      data[point].neighbors_number = neighbors_counter;
}
```

Πίνακας 3: Συνάρτηση range_query

Κεφάλαιο 5

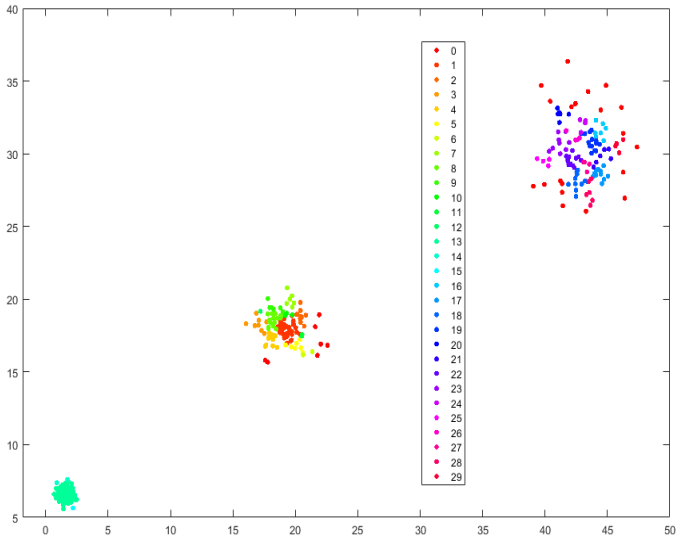
Πειράματα και Εφαρμογές

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα των πειραμάτων που πραγματοποιήθηκαν στην παρούσα εργασία. Τα αποτελέσματα αφορούν την ποιότητα των αποτελεσμάτων ανάλογα με το σετ δεδομένων και τις παραμέτρους EPS, MINPTS του αλγορίθμου DBCAN. Επίσης, πραγματοποιήθηκε μία καταγραφή του υπολογιστικού χρόνου του αλγορίθμου και το πώς αυτός αλλάζει σε σχέση με το πλήθος των σημείων που διαθέτει το κάθε σετ δεδομένων. Για την εξαγωγή συμπερασμάτων χρησιμοποιήθηκαν τρία διαφορετικά σετ δεδομένων. Τέλος γίνεται μια αναφορά σε εφαρμογές των αλγορίθμων συσταδοποίησης στη Μηχανική Μάθηση.

5.1 Δεδομένα με διακριτές συστάδες

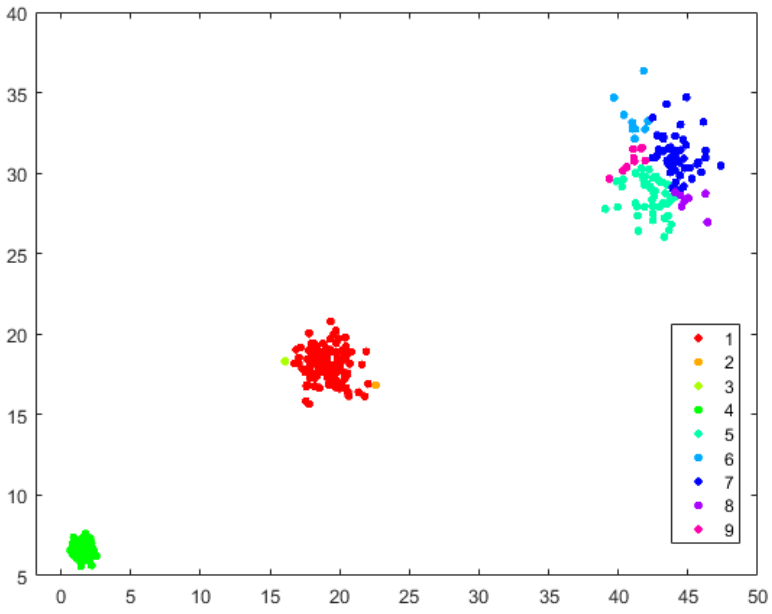
Το πρώτο σετ δεδομένων δημιουργήθηκε με το Matlab script “create_data_clear_clusters.m” που παρατίθεται στο Παράρτημα και αποτελείται από 500 σημεία, τα οποία έχουν δημιουργηθεί με τέτοιο τρόπο ώστε να είναι σαφώς ορισμένες ποιες θα είναι οι συστάδες που πρέπει να προκύψουν.

Στην Εικόνα 8 παρουσιάζονται τα αποτελέσματα για το πρώτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=1 και MINPTS=5.



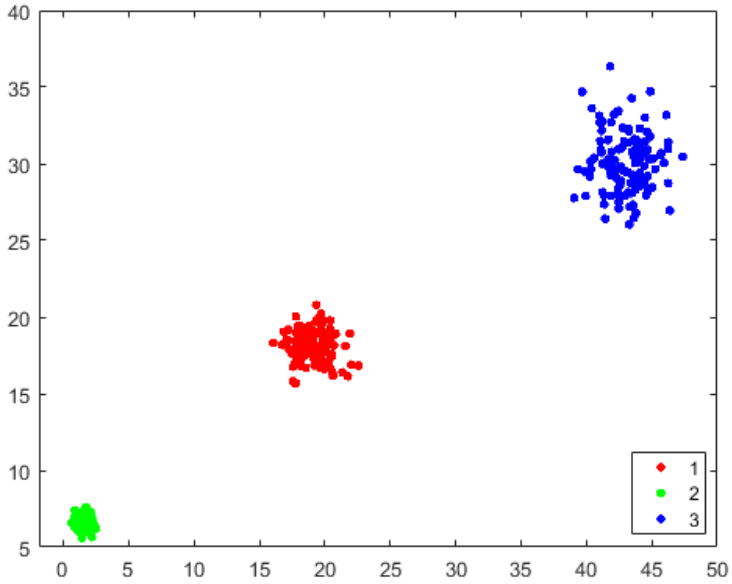
Εικόνα 8: 1^{ο} σετ, EPS=1, MINPTS=5

Στην Εικόνα 9 παρουσιάζονται τα αποτελέσματα για το πρώτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=3 και MINPTS=5.



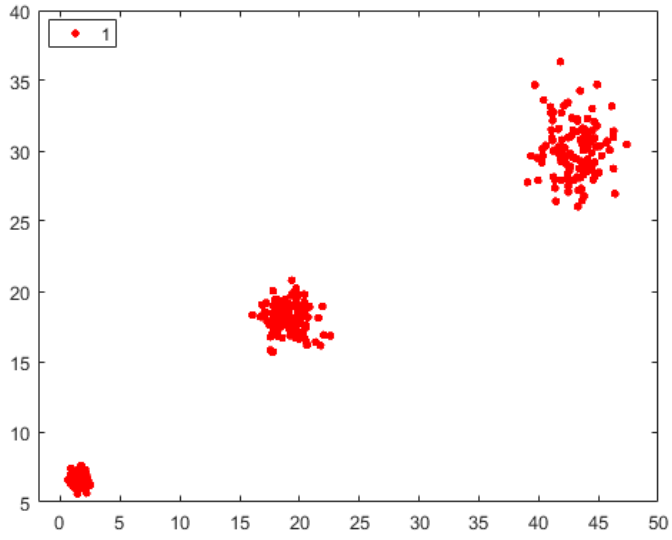
Εικόνα 9: 1ο σετ, EPS=3, MINPTS=5

Στην Εικόνα 10 παρουσιάζονται τα αποτελέσματα για το πρώτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=10 και MINPTS=5.



Εικόνα 10: 1ο σετ, EPS=10, MINPTS=5

Στην Εικόνα 11 παρουσιάζονται τα αποτελέσματα για το πρώτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=50 και MINPTS=5.



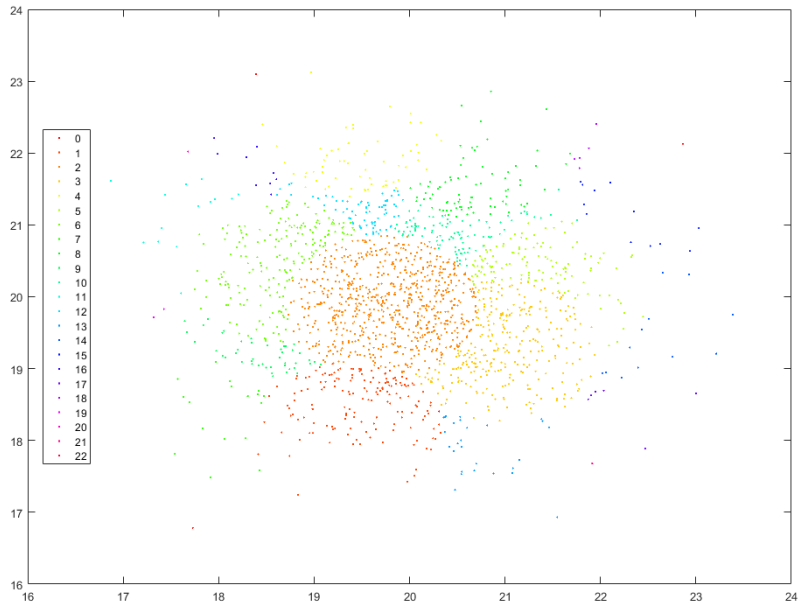
Εικόνα 11: 1ο σετ, EPS=50, MINPTS=5

Είναι σαφές από τις εικόνες πως το $1^{ο}$ σετ δεδομένων περιέχει τρεις διακριτές συστάδες Η επιλογή της παραμέτρου EPS είναι κρίσιμη για την απόδοση του αλγορίθμου DBSCAN. Στο συγκεκριμένο σετ δεδομένων, αν οριστεί μια πολύ μικρή τιμή, τότε οι δύο πιο «αραιές» συστάδες χωρίζονται σε μικρότερες, με τον αλγόριθμο να εντοπίζει σωστά μόνο τη πιο συμπαγή κάτω δεξιά. Τα αποτελέσματα βελτιώνονται ελαφρώς όταν αυξάνεται λίγο η παράμετρος EPS, αλλά η πάνω δεξιά «αραιή» συστάδα εξακολουθεί να χωρίζεται σε περισσότερες συστάδες. Αν αυξήσουμε υπερβολικά την παράμετρο EPS, τότε όλα τα σημεία ανιχνεύονται ότι ανήκουν σε μια κοινή συστάδα. Στο συγκεκριμένο σετ δεδομένων φαίνεται πως η πιο κατάλληλη τιμή είναι EPS=10 λόγων των αποστάσεων μεταξύ των σημείων.

5.2 Δεδομένα με κανονική κατανομή

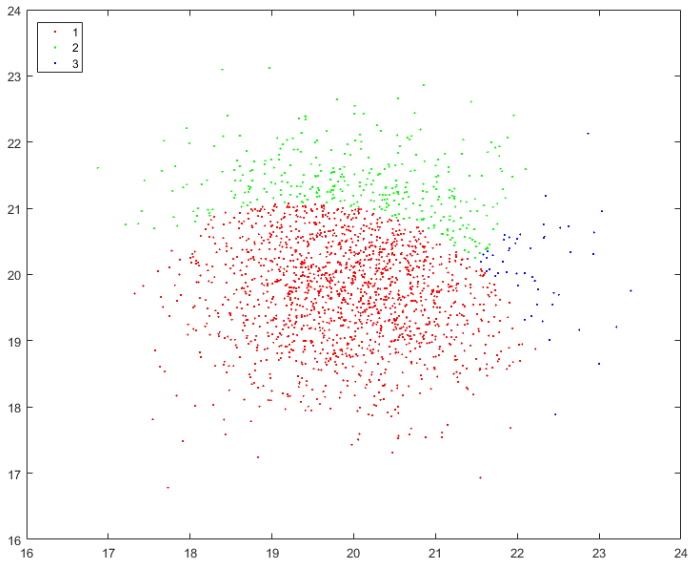
Το πρώτο σετ δεδομένων δημιουργήθηκε με το Matlab script "create_data_normal_random.m" που παρατίθεται στο Παράρτημα και αποτελείται από 2000 σημεία, τα οποία έχουν δημιουργηθεί με την κανονική τυχαία κατανομή με τη χρήση της εντολής normrnd με μέση τιμή 20 και απόκλιση 1.

Στην Εικόνα 12 παρουσιάζονται τα αποτελέσματα για το δεύτερο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=1 και MINPTS=5.



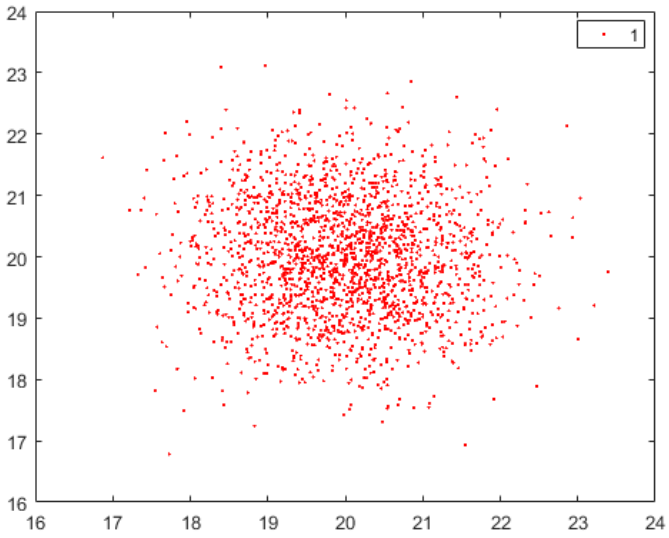
Εικόνα 12: 2ο σετ, EPS=1, MINPTS=5

Στην Εικόνα 13 παρουσιάζονται τα αποτελέσματα για το δεύτερο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=3 και MINPTS=5.



Εικόνα 13: 2ο σετ, EPS=3, MINPTS=5

Στην Εικόνα 14Εικόνα 11 παρουσιάζονται τα αποτελέσματα για το δεύτερο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=10 και MINPTS=5.



Εικόνα 14: 2ο σετ, EPS=10, MINPTS=5

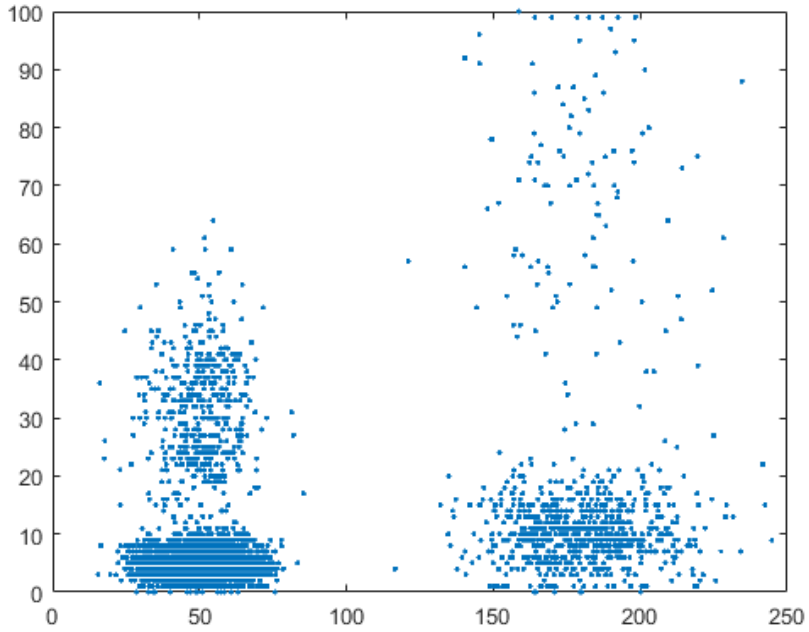
Από τις παραπάνω εικόνες είναι φανερό πως το δεύτερο σετ είναι αρκετά διαφορετικό σε σχέση με το πρώτο, με τα αντικείμενα να είναι πολύ κοντά μεταξύ τους και με την πρώτη ματιά, οι περισσότεροι θα έλεγαν ότι τα αντικείμενα αποτελούν μία μοναδική συστάδα.

Όντως, αν τεθεί μια αρκετά μεγάλη τιμή στην παράμετρο EPS, όπως στην Εικόνα 14, τότε ο αλγόριθμος καταλήγει σε μία ενιαία συστάδα. Όμως, αν επιλέξει μία πιο μικρή τιμή, τότε οι συστάδες που εξάγονται τελικά είναι περισσότερες, όπως φαίνεται από τα αποτελέσματα στις Εικόνα 12 και Εικόνα 13. Η τιμή EPS=50 που χρησιμοποιήθηκε στο πρώτο σετ δεδομένων δίνει προφανώς το ίδιο αποτέλεσμα και δεν παρατίθεται εδώ.

5.3 Δεδομένα με ποικίλη πυκνότητα

Το τρίτο σετ δεδομένων αποτελείται από 4000 σημεία των οποίων η πυκνότητα διαφέρει ανά περιοχή, παρουσιάζεται στην Εικόνα 15 και προέρχεται από την πηγή (Trevino, 2018). Στο άρθρο αυτό παρουσιάζεται ο αλγόριθμος συσταδοποίησης k-

means και χρησιμοποιείται το προαναφερθέν σετ δεδομένων που αναπαριστά τη συμπεριφορά οδηγών που κάνουν διανομές. Ο άξονας x αντιστοιχεί στη μέση απόσταση που διένυσε κάθε μέρα και ο άξονας y το μέσο όρο του χρόνου κατά τον οποίο ο διανομέας οδηγούσε με τουλάχιστον πέντε μίλια/ώρα πάνω από το όριο ταχύτητας. Τα δεδομένα αυτά μας δίνουν ένα παράδειγμα εφαρμογής των αλγορίθμων συσταδοποίησης στην πραγματική ζωή.

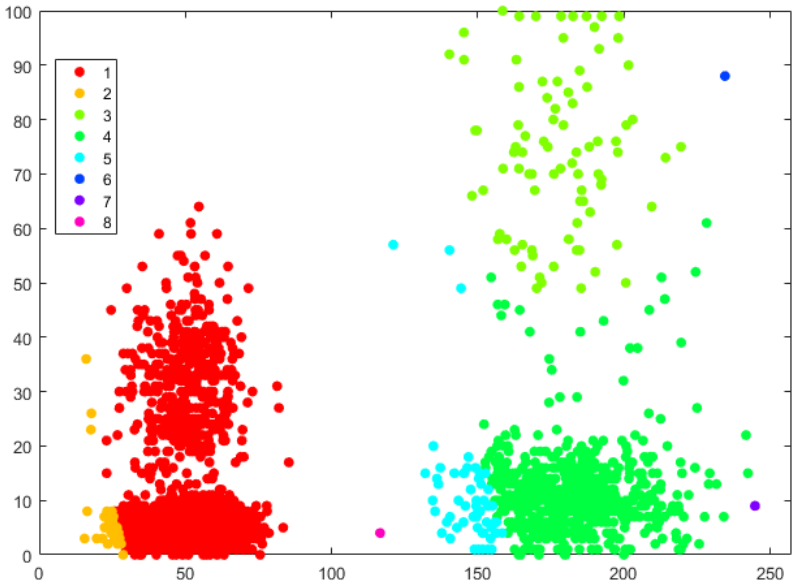


Εικόνα 15: 3ο σετ

Στις ακόλουθες εικόνες επιλέχτηκε, λόγω του μεγαλύτερου πλήθους δεδομένων, μεγαλύτερο μέγεθος για τις τελείες που αναπαριστούν τα δεδομένα ώστε να υπάρχει μία πιο σαφή εικόνα των συστάδων, χάνοντας σε λεπτομέρεια για το κάθε σημείο ξεχωριστά.

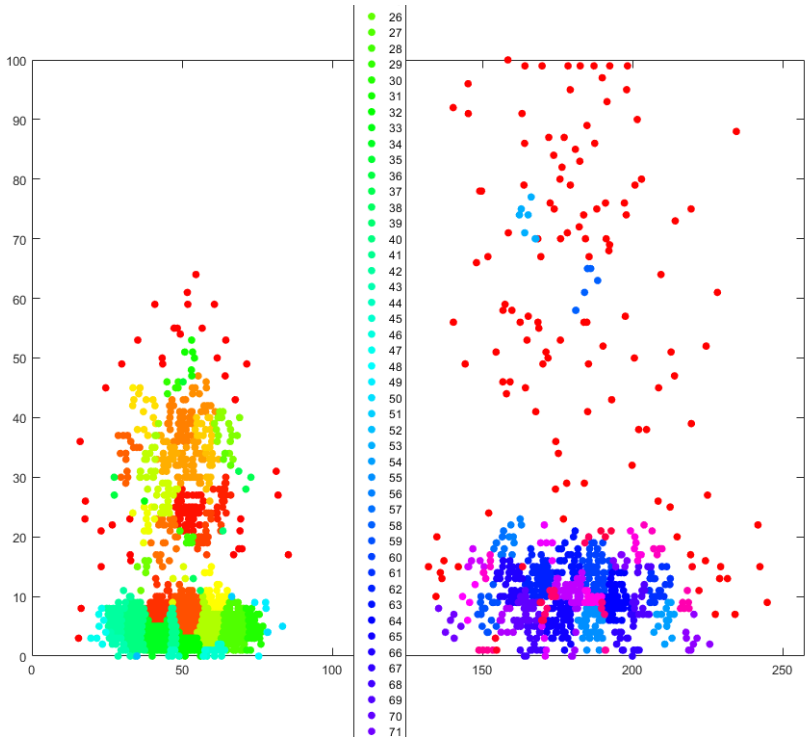
Στην Εικόνα 16 παρουσιάζονται τα αποτελέσματα για το τρίτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=50 και MINPTS=5. Οι συστάδες 6, 7, 8 αποτελούνται από ένα σημείο και χαρακτηρίστηκαν ως θόρυβος από τον αλγόριθμο. Η μεγάλη τιμή που επιλέχτηκε για την παράμετρο EPS επιτρέπει στον αλγόριθμο DBSCAN να χωρίσει ορθώς τον κύριο όγκο των σημείων στην πρώτη και

στην τρίτη συστάδα, αλλά η δεύτερη και η πέμπτη συστάδα θα μπορούσαν να είχαν ενσωματωθεί στην πρώτη και τρίτη αντίστοιχα.



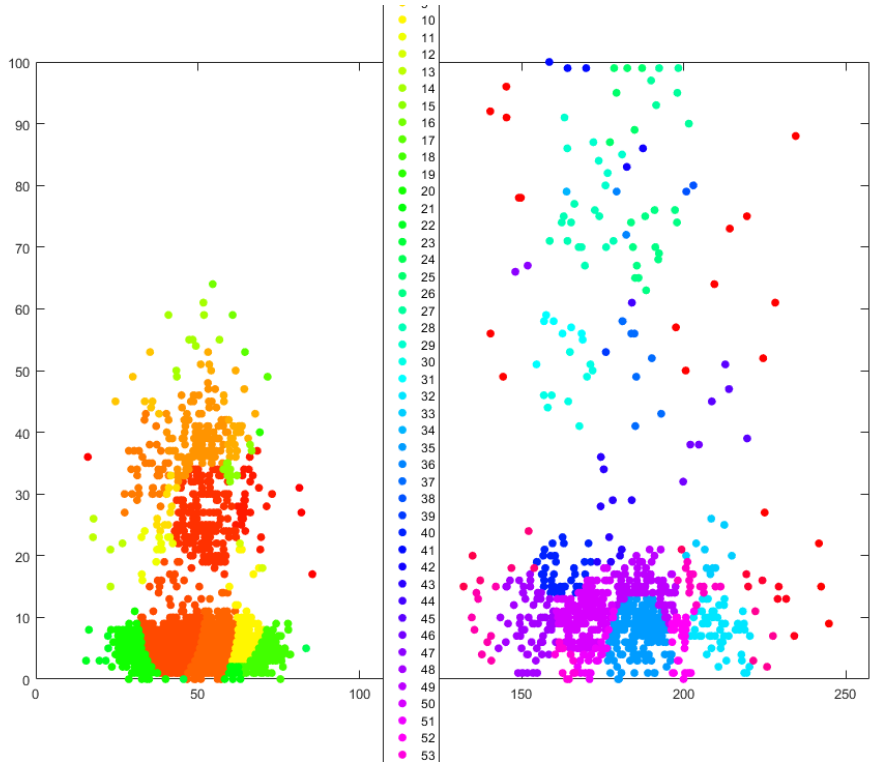
Εικόνα 16: 3ο σετ, EPS=50, MINPTS=5

Στην Εικόνα 17 παρουσιάζονται τα αποτελέσματα για το τρίτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=50 και MINPTS=5. Η παράμετρος EPS έχει μειωθεί υπερβολικά σε σημείο μάλιστα που ο αλγόριθμος DBSCAN αναγνωρίζει 95 διαφορετικές συστάδες. Το γεγονός αυτό δεν μας επιτρέπει να εξάγουμε κανένα συμπέρασμα για τους διανομείς που αποτελούν την κάθε συστάδα.



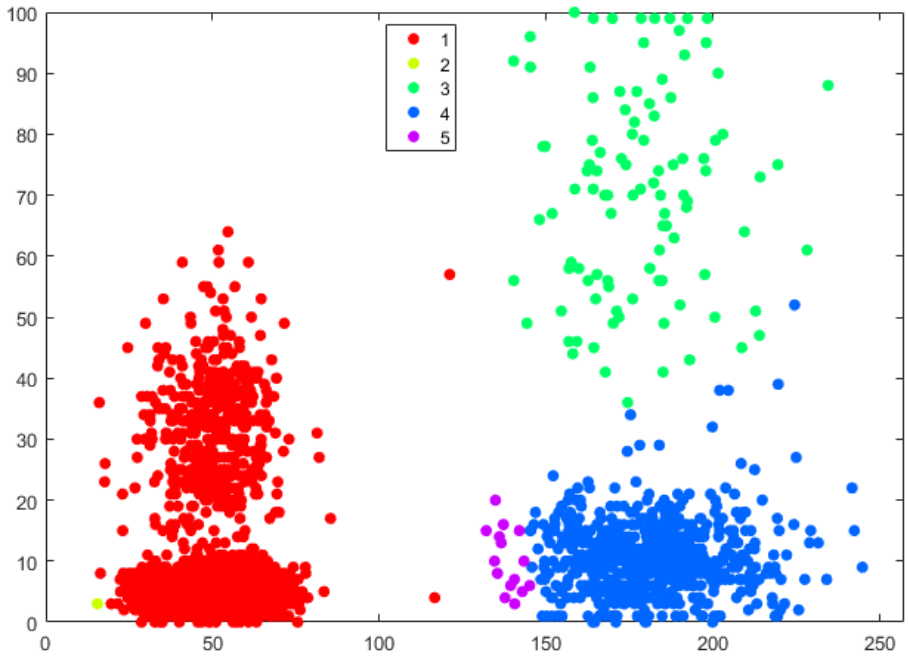
Εικόνα 17: 3ο σετ, EPS=5, MINPTS=5

Στην Εικόνα 18 παρουσιάζονται τα αποτελέσματα για το τρίτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=10 και MINPTS=3. Στο πείραμα αυτό επιλέχτηκε μια ελαφρώς μεγαλύτερη τιμή για την παράμετρο EPS και επιλέχτηκε να μειωθεί και η παράμετρος MINPTS στη ελάχιστη δυνατή τιμή της ώστε να βελτιωθούν εκ νέου τα αποτελέσματα. Πράγματι οι συστάδες που ανιχνεύονται είναι λιγότερες, αλλά και πάλι τα αποτελέσματα δεν είναι ικανοποιητικά.



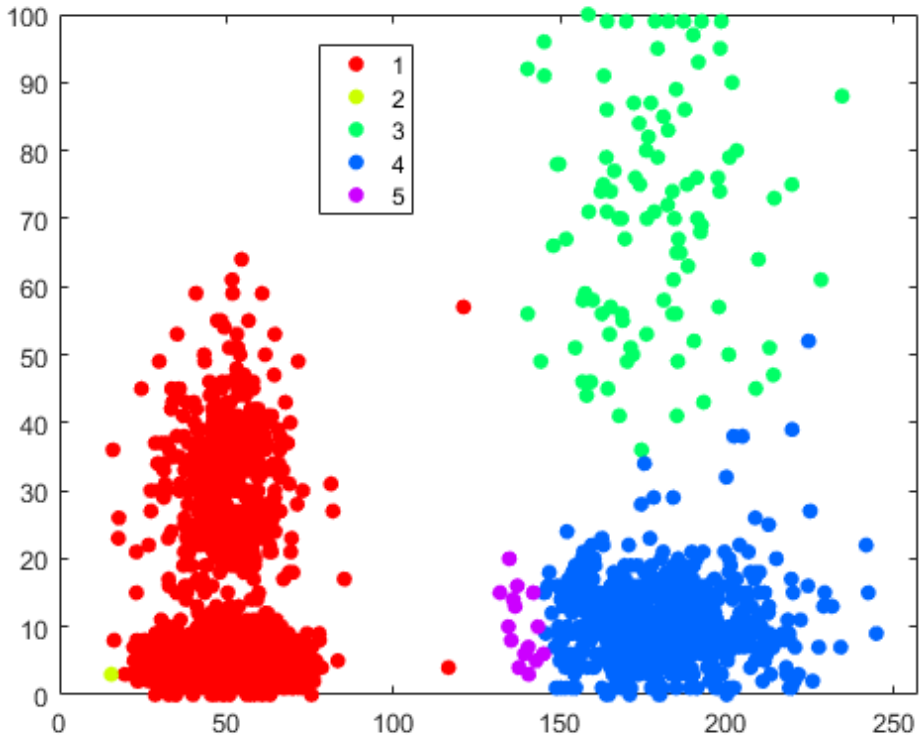
Εικόνα 18: 3ο σετ, EPS=10, MINPTS=3

Στην Εικόνα 19Εικόνα 18 παρουσιάζονται τα αποτελέσματα για το τρίτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=60 και MINPTS=5. Παρατηρώντας την απόδοση του αλγορίθμου στα προηγούμενα πειράματα αποφασίστηκε η επιστροφή στην τιμή 5 για την παράμετρο MINPTS καθώς η μείωση της δεν έφερε βελτίωση, ενώ στην παράμετρο EPS δόθηκε μία μεγαλύτερη τιμή από αυτή στο πείραμα της Εικόνα 16. Οι κινήσεις αυτές έφεραν βελτίωση των αποτελεσμάτων με το πλήθος των συστάδων να πέφτει στις 5 και την πρώτη, τρίτη και τέταρτη συστάδα είναι σαφώς ορισμένες.



Εικόνα 19: 3ο σετ, EPS=60, MINPTS=5

Στην Εικόνα 20 παρουσιάζονται τα αποτελέσματα για το τρίτο σετ δεδομένων, επιλέγοντας τις παραμέτρους του αλγορίθμου EPS=60 και MINPTS=10. Στο τελευταίο αυτό πείραμα αποφασίστηκε να διατηρηθεί η τιμή EPS=60 και να αυξηθεί η παράμετρος MINPTS στην τιμή 10. Οι αλλαγές αυτές δεν προξένησαν καμία αλλαγή με τα αποτελέσματα να μένουν ακριβώς τα ίδια με την προηγούμενη περίπτωση.

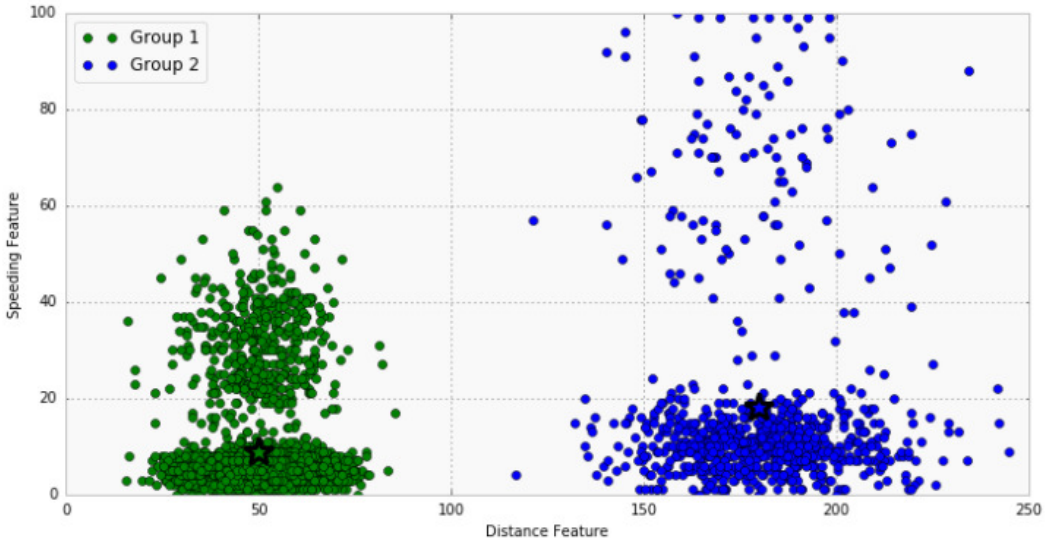


Εικόνα 20: 3ο σετ, EPS=60, MINPTS=10

Είναι σαφές από τα παραπάνω πειράματα ότι ο αλγόριθμος DBSCAN δυσκολεύεται να αποδώσει στο συγκεκριμένο σετ δεδομένων αποτυγχάνοντας να δώσει σωστές συστάδες. Είναι γνωστό από τη βιβλιογραφία ότι ο αλγόριθμος DBSCAN δεν δίνει καλά αποτελέσματα για σετ δεδομένων που η πυκνότητα τους ποικίλει και τα αποτελέσματα των πειραμάτων καταδεικνύουν το γεγονός αυτό.

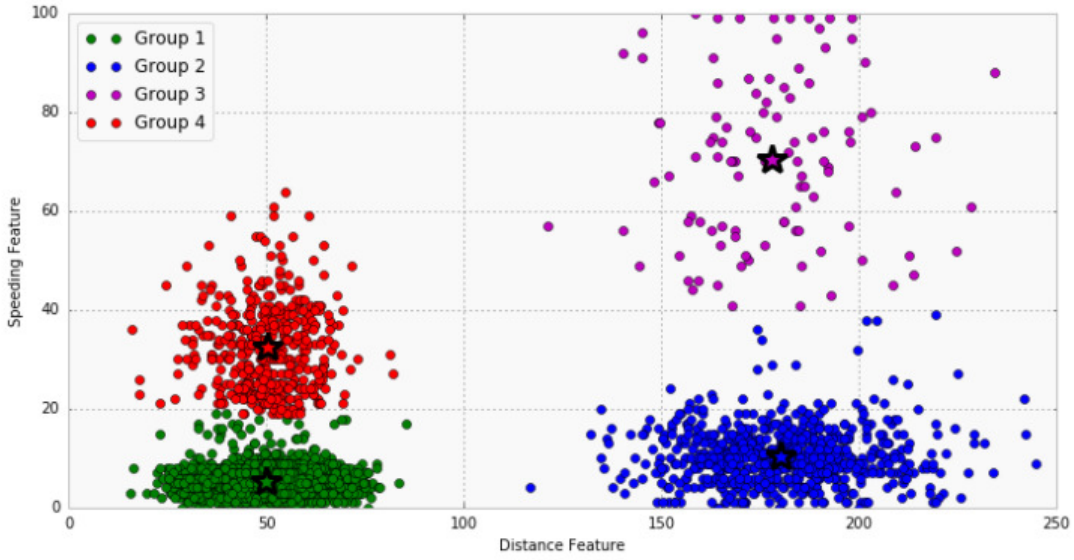
Στο σετ αυτό πιο κατάλληλος αλγόριθμος για εφαρμογή θα ήταν ο k-means και για να πραγματοποιηθεί η εν λόγω σύγκριση παρατίθενται τα αποτελέσματα από την (Trevino, 2018).

Στην Εικόνα 21 είναι τα αποτελέσματα του k-means για k=2. Παίρνοντας υπόψιν τα χαρακτηριστικά κάθε άξονα που αφορούν την απόσταση που διανύει και την ταχύτητα του κάθε διανομέα, μπορεί να εξαχθεί το συμπέρασμα ότι η πρώτη συστάδα αποτελείται από διανομείς που εργάζονται στην πόλη, ενώ η δεύτερη συστάδα από διανομείς που οδηγούν κυρίως σε επαρχιακούς ή εθνικούς οδούς.



Εικόνα 21: 3ο σετ, k-means, k=2

Στην Εικόνα 22 είναι τα αποτελέσματα του k-means για k=4. Πλέον ο αλγόριθμος αναγνωρίζει τέσσερις συστάδες αντί για δύο και οι διανομείς διαχωρίζονται επιπλέον με βάση την ταχύτητα τους. Η πρώτη συστάδα αποτελείται από διανομείς που εργάζονται στην πόλη και τηρούν τα όρια ταχύτητας, ενώ η τέταρτη συστάδα αποτελείται από διανομείς που εργάζονται στην πόλη, αλλά δεν τηρούν τα όρια ταχύτητας. Αντίστοιχα, η δεύτερη συστάδα αποτελείται από διανομείς που οδηγούν σε επαρχιακές ή εθνικές οδούς και τηρούν τα όρια ταχύτητας, ενώ η τρίτη συστάδα αποτελείται από διανομείς που οδηγούν σε επαρχιακές ή εθνικές οδούς και δεν τηρούν τα όρια ταχύτητας.



Εικόνα 22: 3ο σετ, k-means, k=4

Η συσταδοποίηση που φαίνεται στην Εικόνα 22 δεν έγινε δυνατό να επιτευχθεί με τον αλγόριθμο DBSCAN στα πειράματα που δοκιμάσαμε. Το συγκεκριμένο σετ δεδομένων απέδειξε τη χαμηλή απόδοση του αλγορίθμου όταν η πυκνότητα των δεδομένων ποικίλει.

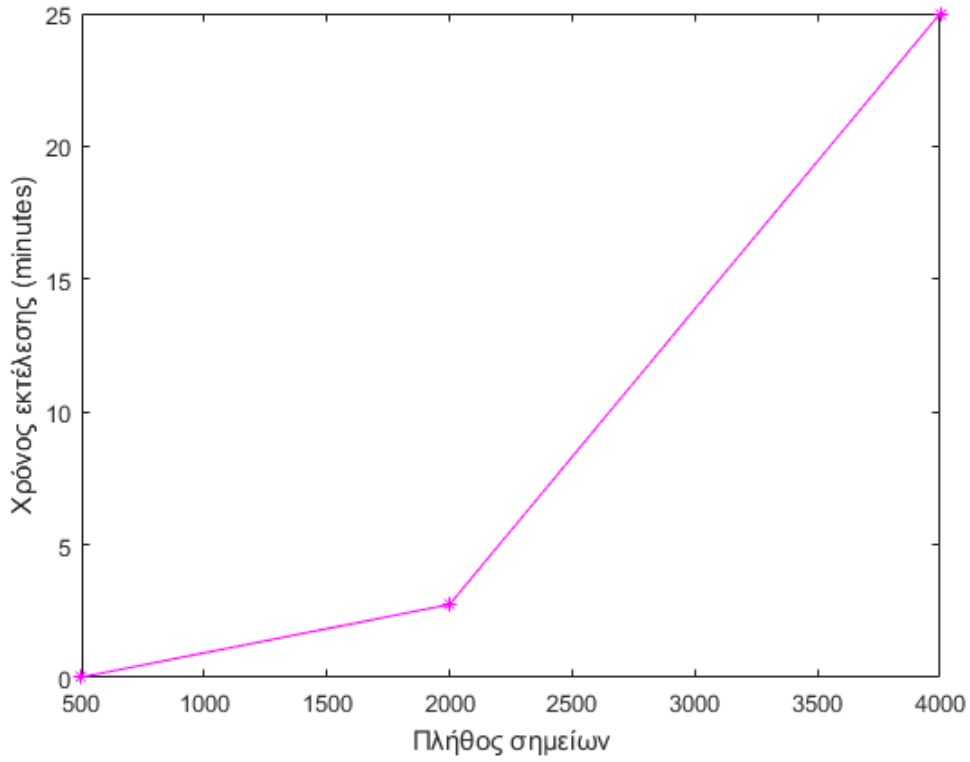
5.4 Χρόνος εκτέλεσης

Μία τελευταία πτυχή του αλγορίθμου που εξετάσαμε ήταν ο χρόνος εκτέλεσης του αλγορίθμου DBSCAN σε σχέση με το μέγεθος των σετ δεδομένων που χρησιμοποιήσαμε. Όλα τα πειράματα εκτελέστηκαν σε περιβάλλον Windows 10 (64 bit) χρησιμοποιώντας το IDE Eclipse Luna σε υπολογιστή που διαθέτει επεξεργαστή με 2 πυρήνες με τα χαρακτηριστικά που παρουσιάζονται στην Εικόνα 23

PROCESSOR / CHIPSET /	
CPU	Intel Core i7 (4th Gen) 4510U / 2 GHz
Max Turbo Speed	3.1 GHz
Number of Cores	Dual-Core
Cache	4 MB
64-bit Computing	Yes
Features	Hyper-Threading Technology, Intel Turbo Boost Technology 2.0, integrated memory controller

Εικόνα 23: Χαρακτηριστικά επεξεργαστή

Στην Εικόνα 24 φαίνεται ο χρόνος εκτέλεσης (σε λεπτά) σε σχέση με το πλήθος των σημείων που είχαν τα σετ δεδομένων. Είναι σαφές πως ο χρόνος υπολογισμού παρουσιάζει μια εκθετική αύξηση, καθώς αυξάνεται το πλήθος των σημείων.



Εικόνα 24: Χρόνος εκτέλεσης

5.5 Εφαρμογές στη Μηχανική Μάθηση

Έχοντας δει πλέον πως λειτουργεί και στην πράξη ο αλγόριθμος DBSCAN σημειώνονται στο σημείο αυτό κάποιες πρακτικές εφαρμογές που έχει. Ο αλγόριθμος DBSCAN μπορεί να χρησιμοποιηθεί για την αναγνώριση θορύβου ώστε να γίνεται καλύτερη αναπαράσταση πραγματικών δεδομένων. Επίσης, μπορεί να χρησιμοποιηθεί για την ομαδοποίηση αντικειμένων, καθώς και δημιουργία διαφορετικών επιπέδων από λεπτομέρειες. Επιπλέον, μπορεί να χρησιμοποιηθεί για τον έλεγχο ποιότητας του νερού και για αυτόματη ανίχνευσή του με άλλες ουσίες, ενώ χρησιμοποιείται ακόμα και στην ανάλυση εικόνων στη νευροχειρουργική (Riedel, 2016). Στην εργασία (Wang, Liang, & Delahaye, 2017) παρουσιάστηκε ένα σύστημα πρόβλεψης κίνησης αεροπλάνων χρησιμοποιώντας και τον αλγόριθμο DBSCAN.

Κεφάλαιο 6

Συμπεράσματα

Στα πλαίσια της παρούσας εργασίας υλοποιήθηκε ο αλγόριθμος DBSCAN χρησιμοποιώντας τη γλώσσα προγραμματισμού C, ενώ η δημιουργία των δεδομένων εισόδου και η απεικόνιση των αποτελεσμάτων έγινε σε Matlab.

Από τα αποτελέσματα που παρουσιάστηκαν, αποδεικνύεται πως ο αλγόριθμος DBSCAN μπορεί να λειτουργήσει ικανοποιητικά για διάφορα σετ δεδομένων αναγνωρίζοντας σωστά τις υπάρχουσες συστάδες, εφόσον επιλεχθεί κατάλληλη τιμή για την παράμετρο EPS. Το μειονέκτημα του αλγορίθμου DBSCAN είναι πως δεν παράγει ικανοποιητικά αποτελέσματα όταν η πυκνότητα των δεδομένων ποικίλει. Η πυκνότητα της συστάδας και η απόσταση μεταξύ των συστάδων καθορίζουν την τιμή που πρέπει να πάρει η παράμετρος EPS και την απόδοση του αλγορίθμου. Η παράμετρος MINPTS δεν φάνηκε να παίζει σημαντικό ρόλο στα πειράματα που τρέξαμε, αλλά προτείνεται να παίρνει μεγάλη τιμή σε σετ δεδομένων με πολύ θόρυβο.

Τέλος, ένας αποδοτικός αλγόριθμος συσταδοποίησης όπως ο DBSCAN, αν και έχει μειονεκτήματα, όπως και κάθε άλλος αλγόριθμος, έχει τη δυνατότητα να εφαρμοστεί σε ένα ευρύ φάσμα πεδίων που σχετίζονται με τη Τεχνητή Νοημοσύνη και τη μηχανική μάθηση.

Βιβλιογραφία

M. Ankerst, M. B. (1999). Optics: Ordering Points to Identify the Clustering Structure. *SIGMOD*, (σσ. 49-60).

MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, (σσ. 281-297).

Mapoka, K. (2013). Mathematical Models and Algorithms Challenges. *International Journal of Control Theory and Computer Modeling* , 21-28.

Martin Ester, H.-P. K. (1996). A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, (σσ. 226-231). Portland.

Riedel, I. M. (2016). *University of Iceland*. Ανάκτηση από https://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/jurecapt16/jurecapt16-sbda.pdf?__blob=publicationFile

Trevino, A. (2018). *DataScience*. Ανάκτηση από https://www.datascience.com/blog/k-means-clustering

Wang, Z., Liang, M., & Delahaye, D. (2017). Short-term 4D Trajectory Prediction Using Machine Learning Methods: a case study on Beijing BCIA airport. *SESAR Innovation Days.*

Wei Wang, J. Y. (1997). STING: A statistical information grid approach to spatial data mining. *roceedings of the 23rd International Conference on Very Large Data Bases*, (σσ. 186-195).

Παράρτημα

Αρχείο dbscan.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>

#define EPS 3           // Maximum radius
#define MINPTS 5  // Number of points required to form a cluster

#define NUMBER_OF_POINTS 500        // Number of points to cluster

// Point p is a core point if at least MINPTS points are found within
its EPS

// ----- Structs -------- //
typedef struct{
      double x;                                          // Latitude
      double y;                                          // Longitude
      int label;                                         // -1:
Undefined  - 0: Noise  -  >0: Cluster Label
      int neighbors[NUMBER_OF_POINTS];    // Neighbors' IDs of each
point
      int neighbors_number;                        // Number of
neighbors of each point
      double neighbors_distance[NUMBER_OF_POINTS];    // The distance
of every neighbor of each point

} point;


// Global Variables
point data[NUMBER_OF_POINTS];
int data_seed[NUMBER_OF_POINTS];


// Calculate a distance between 2 points
double calculate_distance(point a, point b){
      return sqrt(pow((b.x-a.x),2)+pow((b.y-a.y),2));
}
```

```
// Read data points
void read_data(){
	FILE *data_file;
	int lines_counter = 0;
	char line[256];
	char input_file[] = "input.txt";
	int i = 0;

	data_file = fopen(input_file, "r");

	if ( data_file == NULL ) {
		printf("File doesn't exit.\n");
	}
	else{
		printf("Reading data points.\n");
		while ( fgets(line, sizeof(line), data_file) ){
			lines_counter++;

			if ( lines_counter%2 == 0 ){
				data[(lines_counter - 1)/2].y = atof(line);
				data[(lines_counter - 1)/2].label = -1;
				data[(lines_counter - 1)/2].neighbors_number
= -1;

				for ( i = 0; i < NUMBER_OF_POINTS; i++){
					data[(lines_counter -
1)/2].neighbors[i] = -1;
					data[(lines_counter -
1)/2].neighbors_distance[i] = -1.0;
				}
			}
			else{
				data[(lines_counter - 1)/2].x = atof(line);
			}
		}
		fclose(data_file);
	}
}


// DBSCAN Algorithm
void dbscan(){
	int cluster_counter = 0;
	int i = 0;
	int j = 0;
	int k = 0;

	// For each data point
	for ( i=0; i<NUMBER_OF_POINTS; i++ ){
		if ( data[i].label != -1 )
			continue;
		range_query(i);
		if ( data[i].neighbors_number < MINPTS ){
			data[i].label = 0;
			continue;
		}
		cluster_counter++;
		data[i].label = cluster_counter;
```

```
            // Expand neighbors
            for ( k=0; k<NUMBER_OF_POINTS; k++ ){
                  data_seed[k] = data[i].neighbors[k];
            }

            for ( j=0; j<data[i].neighbors_number; j++ ){
                  int neighbor_index = data[i].neighbors[j];
                  if ( data[neighbor_index].label == 0 )
                        data[neighbor_index].label = cluster_counter;
                  if ( data[neighbor_index].label != -1 )
                        continue;
                  data[neighbor_index].label = cluster_counter;
                  range_query(neighbor_index);
                  if ( data[neighbor_index].neighbors_number >=
MINPTS ){

                        for ( k=0; k<NUMBER_OF_POINTS; k++ ){
                              if ( data_seed[k] != -1 )
                                    continue;
                              data_seed[k] =
data[neighbor_index].neighbors[k];
                        }
                  }
            }
      }
}



// Find neighbors of input point
void range_query(int point){
      int i = 0;
      int neighbors_counter = 0;

      for ( i = 0; i < NUMBER_OF_POINTS; i++ ){
            // Skip input point
            if ( i == point )
                  continue;
            double distance =
calculate_distance(data[point],data[i]);
            if ( distance <= EPS ){
                  data[point].neighbors[neighbors_counter] = i;
                  data[point].neighbors_distance[neighbors_counter] =
distance;
                  neighbors_counter++;
            }
      }

      data[point].neighbors_number = neighbors_counter;
}


void write_data(){
      int i = 0;
      FILE *data_file;

      char output_file[] = "clustersCreated.txt";
```

```
	data_file = fopen(output_file, "w");

	for ( i = 0; i < NUMBER_OF_POINTS; i++ ){
		fprintf(data_file, "%.8g \n", data[i].x);
		fprintf(data_file, "%.8g \n", data[i].y);
		fprintf(data_file, "%d \n", data[i].label);
	}
	fclose(data_file);
}


int main(){
	read_data();
	dbscan();
	write_data();
	return 0;
}
```

Αρχείο plot_data.m

```
clear all;
close all;

NumberOfPoints = 500;

inputPoints = zeros(NumberOfPoints,2);
clusterPoints = zeros(NumberOfPoints,2);
clusterID = zeros(NumberOfPoints,1);

fID = fopen('input.txt');
lineCounter = 1;
line = fgetl(fID);

while ischar(line)
    if mod(lineCounter,2)==1
        inputPoints(ceil(lineCounter/2),1) = str2double(line);
    else
        inputPoints(ceil(lineCounter/2),2) = str2double(line);
    end
    lineCounter=lineCounter+1;
    line = fgetl(fID);
end
fclose(fID);
plot(inputPoints(:,1),inputPoints(:,2),'Marker','.','LineStyle','none
','MarkerSize',7)



fID = fopen('clustersCreated.txt');
clustersCounter = 0;
pointsCounter = 1;
line = fgetl(fID);
tagCounter = 0;
while ischar(line)
    if (tagCounter == 0)
        clusterPoints(pointsCounter,1) = str2double(line);
        tagCounter = 1;
    elseif (tagCounter == 1)
        clusterPoints(pointsCounter,2) = str2double(line);
        tagCounter = 2;
    elseif (tagCounter == 2)
        clusterID(pointsCounter) = str2double(line);
        tagCounter = 0;
        pointsCounter = pointsCounter + 1;
    end
    line = fgetl(fID);
end
fclose(fID);

gscatter(clusterPoints(:,1),clusterPoints(:,2),clusterID,'','',20);
```

Αρχείο create_data_clear_clusters.m

```
clear all;
close all;

NumberOfPoints = 500;

P=randn(NumberOfPoints,2)+20;
x=P(:,1)';
y=P(:,2)';

FirstIndex = NumberOfPoints/4;
SecondIndex = FirstIndex + NumberOfPoints/2;

x=[   x(1:FirstIndex)-1     1/3.*x((FirstIndex+1):SecondIndex)-5
1.8.*x((SecondIndex+1):NumberOfPoints)+7   ];
y=[   y(1:FirstIndex)-2     1/3.*y((FirstIndex+1):SecondIndex)
1.8.*y((SecondIndex+1):NumberOfPoints)-6   ];
plot(x,y,'Marker','.','LineStyle','none')
axis('equal')

fileID = fopen('input.txt','w');

for i=1:NumberOfPoints
    fprintf(fileID,'%f\n',x(i));
    fprintf(fileID,'%f\n',y(i));
end
```

Αρχείο create_data_normal_random.m

```
clear all;
close all;

NumberOfPoints = 2000;

x = normrnd(20,1,NumberOfPoints,1);
y = normrnd(20,1,NumberOfPoints,1);

plot(x,y,'Marker','.','LineStyle','none')
axis('equal')

fID = fopen('input.txt','w');
for i=1:NumberOfPoints
    fprintf(fID,'%f\n',x(i));
    fprintf(fID,'%f\n',y(i));
end
```