Ατομική Διπλωματική Εργασία

ΑΝΑΛΥΣΗ ΕΤΙΚΕΤΩΝ ΠΟΥ ΕΜΠΕΡΙΕΧΟΝΤΑΙ ΣΕ ΑΡΧΕΙΑ ΛΟΓΙΣΜΙΚΟΥ ΤΟΥ GOOGLE CODE

Χαράλαμπος Χαραλάμπους

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2012

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάλυση ετικετών που εμπεριέχονται σε αρχεία λογισμικού του Google Code

Χαράλαμπος Χαραλάμπους

Επιβλέπων Καθηγητής
Δρ. Πάλλης Γιώργος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2012

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή, Δρ. Γιώργο Πάλλη και τον Δρ. Ιωάννη Κατάκη για την συνεχή καθοδήγησή, τις πολύτιμες συμβουλές και υποδείξεις τους.

Σας ευχαριστώ,
Χαράλαμπος Χαραλάμπους

Περίληψη

Το Minersoft[1] είναι μια μηχανή αναζήτησης λογισμικού, η οποία έχει στόχο να βοηθήσει τους χρήστες να συλλέξουν πληροφορίες που αφορούν αρχεία λογισμικού τα οποία είναι εγκατεστημένα σε κόμβους υπολογιστικών υποδομών μεγάλης κλίμακας Grid και Cloud. Για την παροχή περισσότερης πληροφορίας στο Minersoft και την περαιτέρω ταξινόμηση των αρχείων σε θεματικές κατηγορίες οι οποίες καθορίζονται από τους ίδιους τους χρήστες έχει εισαχθεί η έννοια της ετικετοποίησης (tagging) [2]. Σε αυτή τη διπλωματική εργασία εστιαστήκαμε στην ανάλυση ετικετών (tags) τα οποία συλλέξαμε από το Google Code [3], μια αποθήκη αρχείων λογισμικού(software repository) η οποία παρέχει στους χρήστες της μια ποικιλία από προϊόντα και εργαλεία λογισμικού. Συγκεκριμένα πραγματοποιήσαμε ομαδοποίηση (clustering) των ετικετών χρησιμοποιώντας διάφορους αλγόριθμους ομαδοποίησης και αξιολογήσαμε τα αποτελέσματα κάθε ομαδοποίησης. Επιπρόσθετα εντοπίσαμε τα πιο συχνά εμφανιζόμενα υποσύνολα των ετικετών (frequent 2-sets, frequent 3-sets) και στη συνέχεια χρησιμοποιώντας δύο αλγορίθμους παραγωγής κανόνων συσχέτισης (association rules) εξαγάγαμε τους πιο ισχυρούς κανόνες συσχέτισης των ετικετών. Τέλος εντοπίσαμε τι είδους κατηγορίες ετικετών εμπεριέχονται στα αρχεία λογισμικού. Σκοπός αυτής της ανάλυσης των ετικετών και των αποτελεσμάτων που προέκυψαν είναι η εξόρυξη χρήσιμης πληροφορίας η οποία να μπορεί στη συνέχεια να χρησιμοποιηθεί στην υλοποίηση ή βελτίωση διαδικασιών που αφορούν την ετικετοποίηση (π.χ. σύσταση ετικετών).

Περιεχόμενα

Ευρετήριο διαγραμμάτων

Κεφάλαιο 1

1. Εισαγωγή

1.1 Συνεισφορά ετικετοποίησης (tagging) στο Minersoft

Με τον όρο ετικετοποίηση (tagging) στο Διαδίκτυο εννοούμε την διαδικασία κατά την οποία οι χρήστες χαρακτηρίζουν με λέξεις-κλειδιά (ετικέτες) ένα αντικείμενο (όπως εικόνα, άρθρο, αρχείο βίντεο) με σκοπό να το περιγράψουν όσο το δυνατόν καλύτερα.

Στόχος της μηχανής αναζήτησης Minersoft είναι να προσφέρει αποτελέσματα στο χρήστη με όσο το δυνατόν περισσότερη πληροφορία, για λογισμικό το οποίο είναι εγκατεστημένο σε Grid και Cloud υποδομές. Για το λόγο αυτό έχει υλοποιηθεί στο Minersoft η λειτουργία της ετικετοποίησης η οποία παρέχει στους χρήστες την ευκαιρία να εισάγουν ετικέτες στα αρχεία λογισμικού. Μέσω αυτής της λειτουργίας λοιπόν επιστρέφονται στον χρήστη αποτελέσματα τα οποία είναι εμπλουτισμένα με ετικέτες οι οποίες χαρακτηρίζουν ένα λογισμικό και τις οποίες έχουν προσθέσει οι ίδιοι οι χρήστες. Επιπλέον οι χρήστες είναι σε θέση να κάνουν αναζήτηση βασισμένη στις ετικέτες που έχουν ήδη προστεθεί στα αρχεία λογισμικού. Ωστόσο στην όλη διαδικασία παρουσιάζονται τα εξής προβλήματα: Α) Το γεγονός ότι οι χρήστες δεν είναι πάντα πρόθυμοι να προσθέτουν ετικέτες στα αρχεία λογισμικού και ως επακόλουθο αυτή η δραστηριότητα να είναι μειωμένη και να μην υπάρχει ικανοποιητικός αριθμός ετικετών και Β) Οι χρήστες μπορεί να χρησιμοποιήσουν διαφορετικές λέξεις για τον χαρακτηρισμό ενός αντικειμένου ή αντίθετα να χαρακτηρίσουν με την ίδια λέξη διαφορετικά αντικείμενα. Το Minersoft έχει δώσει λύση σε αυτά τα προβλήματα χρησιμοποιώντας μια προσέγγιση μηχανικής μάθησης στην οποία το σύστημα «μαθαίνει» από τις υπάρχουσες ετικέτες που έχουν προσθέσει οι χρήστες και στη συνέχεια αυτόματα εκχωρεί ετικέτες και σε καινούρια αρχεία λογισμικού. [4]

Ως εκ τούτου όμως η όλη διαδικασία μπορεί να βελτιωθεί ή ακόμα και να επεκταθεί. Για το λόγο αυτό απαιτείται η ανάλυση των ετικετών που εμπεριέχονται σε αρχεία λογισμικού.

1.2 Συνεισφορά διπλωματικής εργασίας

Η βασική συνεισφορά αυτής της διπλωματικής εργασίας είναι η εξόρυξη πληροφορίας που αφορά τις συσχετίσεις των ετικετών που συλλέχθηκαν από το Google Code [3]. Συγκεκριμένα:

- Ομαδοποίηση των ετικετών με βάση του σε ποια αρχεία λογισμικού (projects) του Google Code παρουσιάζονται. (Κεφάλαιο 4)
- Σύγκριση και αξιολόγηση των αποτελεσμάτων που προέκυψαν από τους διάφορους αλγορίθμους ομαδοποίησης. (Κεφάλαιο 4)
- Εντοπισμός των πιο συχνά εμφανιζόμενων υποσυνόλων ετικετών (υποσύνολα των δύο ετικετών, υποσύνολα των τριών ετικετών). (Κεφάλαιο 5)
- Εξόρυξη κανόνων συσχέτισης (association rules) με την χρήση δύο αλγορίθμων. (Κεφάλαιο 5)
- Εντοπισμός του τι κατηγορίες ετικετών υπάρχουν και σε τι ποσοστό παρουσιάζεται η κάθε κατηγορία στο σύνολο των αρχείων λογισμικού που συλλέχτηκαν. (Κεφάλαιο 6)

1.3 Κίνητρο

Όπως έχουμε προαναφέρει στο Minersoft έχουν ήδη υλοποιηθεί διεργασίες ετικετοποίησης. Σκοπός αυτών των διεργασιών είναι η οργάνωση των αρχείων λογισμικού σε κατηγορίες που έχουν εισάγει οι ίδιοι οι χρήστες και ο περεταίρω εμπλουτισμός των αρχείων με περισσότερη πληροφορία.

Το κίνητρο λοιπόν αυτής της διπλωματικής εργασίας είναι να γίνει μια ανάλυση των ετικετών που εμπεριέχονται στα αρχεία λογισμικού και να εξαχθεί χρήσιμη πληροφορία η οποία να μπορεί να χρησιμοποιηθεί στην συνέχεια στην βελτίωση ή και επέκταση των διαδικασιών που αφορούν την ετικετοποίηση στο Minersoft. Συγκεκριμένα τα αποτελέσματα από την όλη ανάλυση μπορούν να χρησιμοποιηθούν στην σύσταση ετικετών, στην αυτόματη σύσταση ετικετών και επιπλέον στην αλλαγή του μηχανισμού ταξινόμησης των αρχείων σε κατηγορίες που προέρχονται από τους ίδιους του χρήστες.

Κεφάλαιο 2

2. Προηγούμενες έρευνες

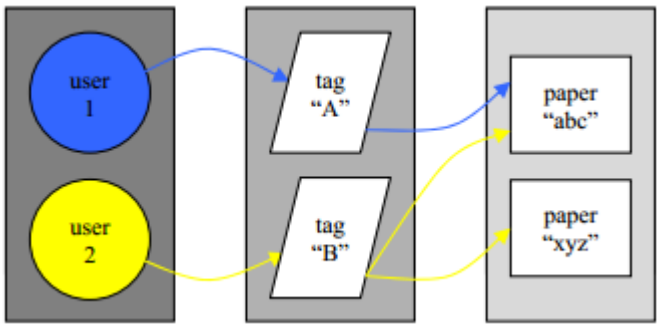
2.1 Αξιολόγηση της συμπεριφοράς των ετικετών σε Social Bookmarking Systems

Ένα Κοινωνικό Σύστημα Ετικετοποίησης(Social Bookmarking System) επιτρέπει στους χρήστες του να καθορίσουν λέξεις-κλειδιά ή ετικέτες σε δεδομένα του διαδικτύου τα οποία τους ενδιαφέρουν. Σκοπός της διαδικασίας αυτής είναι να βοηθήσουν στην οργάνωση των δεδομένων αυτών, αλλά και στο να μοιραστούν αυτά τα δεδομένα και με άλλους χρήστες του συστήματος.

Για την βελτίωση των υπαρχόντων Κοινωνικών Συστημάτων Ετικετοποίησης και την σχεδίαση καινούργιων απαιτείται από τους ειδικούς ερευνητές του τομέα, να καταλάβουν πώς επιτυγχάνεται η διαδικασία αξιολόγησης της συμπεριφοράς των ετικετών.

Στο άρθρο [5] προτείνονται από τους συγγραφείς έξι μετρικές αξιολόγησης των ετικετών με σκοπό την κατανόηση των χαρακτηριστικών ενός Κοινωνικού Συστήματος Ετικετοποίησης. Επεξηγούν αυτές τις μετρικές με δεδομένα τα οποία πάρθηκαν από το CiteULike, ένα σύστημα για ακαδημαϊκά άρθρα τα οποία εμπεριέχει ετικέτες από του χρήστες. Συγκεκριμένα τους παρέχει τη δυνατότητα να αποθηκεύουν, να μοιράζονται και να οργανώνουν πληροφορίες σχετικά με επιστημονικά άρθρα . Επιπλέον χρησιμοποιώντας αυτές τις έξι μετρικές προτείνουν δύο πιθανά ευρετικά σχεδίασης (design heuristics) για την υλοποίηση ενός Κοινωνικού Συστήματος Ετικετοποίησης για το σύστημα CiteSeer, το οποίο είναι μια ευρέως γνωστή, επιστημονική, ψηφιακή βιβλιοθήκη για την επιστήμη της Πληροφορικής.

Ο θεμελιώδης όρος της πληροφορίας σε ένα Κοινωνικό Σύστημα Ετικετοποίησης εμπεριέχει τρία στοιχεία σε μια τριπλέττα(triplet). Συγκεκριμένα η πληροφορία αναπαριστάται με ένα χρήστη, μία πηγή(resource) και μια ετικέτα όπως φαίνεται και στην εικόνα 2.1.1.



Εικόνα 2.1.1.: Απεικόνηση θεμελιώδους όρου της πληροφορίας σε ένα Κοινωνικό Σύστημα Ετικετοποίησης

Στην συνέχεια παρουσιάζονται οι μετρικές αξιολόγησης που προτάθηκαν από τους συγγραφείς του άρθρου [5].

- **Tag growth:** Αυτή η μετρική εξετάζει την αύξηση του λεξιλογίου των ετικετών, δηλαδή κατά πόσο δημιουργούνται καινούριες ετικέτες σε ένα Κοινωνικό Σύστημα Ετικετοποίησης με το πέρας του χρόνου. Η συγκεκριμένη μετρική απαντά ερωτήματα του τύπου πώς εξελίσσεται το πλήθος των ετικετών με το χρόνο, εάν το λεξιλόγιο των ετικετών σταθεροποιείται σε κάποια χρονική στιγμή και επιπρόσθετα πως επηρεάζουν οι νέοι χρήστες στην ανάπτυξη των νέων ετικετών.

- **Tag reuse:** Αυτή η μετρική εξετάζει την επαναχρησιμοποίηση ήδη χρησιμοποιημένων ετικετών. Δηλαδή κατά πόσο ανακυκλώνονται οι παλιές ετικέτες σε ένα Κοινωνικό Σύστημα Ετικετοποίησης και επιπλέον σε συνεργασία με την μετρική tag growth κατά πόσο το λεξιλόγιο των ετικετών εξελίσσεται με το πέρας του χρόνο.

- **Tag non-obviousness:** Αυτή η μετρική εξετάζει το πόσο προφανές είναι μια ετικέτα σύμφωνα με την πηγή την οποία χαρακτηρίζει. Με άλλα λόγια κατά πόσο μια ετικέτα η οποία εμφανίζεται στο κείμενο της πηγής είναι

συσχετιζόμενη με αυτή και τι συμβαίνει στην περίπτωση που μια ετικέτα δεν εμφανίζεται καθόλου στο κείμενο της πηγής που χαρακτηρίζει. Επιπλέον απαντά στο ερώτημα κατά πόσο μη προφανείς ετικέτες μπορούν να φανούν χρήσιμες.

- **Tag discrimination:** Αυτή η μετρική εξετάζει το πόσο καλά μια ετικέτα χαρακτηρίζει και διακρίνει την πηγή στην οποία έχει γίνει tagging. Δηλαδή υποδεικνύει το εύρος της πληροφορίας που εμπεριέχει μια ετικέτα και μπορεί να χρησιμοποιηθεί από τον διαχειριστή ενός συστήματος για να εντοπίσει τις ετικέτες με την λιγότερη πληροφορία και να τις διαγράψει

- **Tag frequency:** Αυτή η μετρική εξετάζει την συχνότητα εμφάνισης των ετικετών και μπορεί να φανεί χρήσιμη στην αξιολόγηση των ετικετών και πώς αυτές χρησιμοποιούνται στο πέρασμα του χρόνου καθώς και σε τι ποσοστό είναι πιθανόν να ξανά-εμφανιστούν.

- **Tag patterns:** Αυτή η μετρική εξετάζει κατά πόσο υπάρχουν πρότυπα (pattern) στην συμπεριφορά των χρηστών και κατά πόσο η συμπεριφορά τους εξαρτάται από παράγοντες όπως τα προσωπικά ενδιαφέρονται και το πεδίου της γνώσης τους. Ο εντοπισμός τέτοιων προτύπων μπορεί να φανεί χρήσιμος στην ανάλυση και υποστήριξη των χρηστών και στην εξαγωγή χρήσιμων συμπερασμάτων.

Τέλος, όπως προαναφέραμε οι συγγραφείς του άρθρου [5] προτείνουν δύο πιθανά ευρετικά σχεδίασης για την υλοποίηση ενός Κοινωνικού Συστήματος Ετικετοποίησης για το σύστημα CiteSeer. Συγκεκριμένα:

- Προτείνουν μια διεπαφή (interface) η οποία να διευκολύνει την επαναχρησιμοποίηση παλιών ετικετών επιτρέποντας στους χρήστες να βλέπουν ήδη χρησιμοποιημένες ετικέτες. Για να το πετύχουν αυτό πιστεύουν ότι πρέπει να παρουσιάζονται στον χρήστη ετικέτες από τρεις κατηγορίες (global tags, personal tags, paper-specific tags). Global tags είναι ετικέτες οι οποίες έχουν χρησιμοποιηθεί παλιότερα από όλους τους χρήστες του συστήματος. Personal tags είναι ετικέτες οι οποίες έχουν χρησιμοποιηθεί παλιότερα από τον ίδιο τον

χρήστη ενώ paper-specific tags είναι ετικέτες που έχουν χρησιμοποιηθεί παλιότερα από του χρήστες για το συγκεκριμένο άρθρο.

- Επιπλέον προτείνουν μια διεπαφή η οποία στηρίζεται στην σύσταση ετικετών οι οποίες θεωρούνται ότι εμπεριέχουν σημαντική πληροφορία σύμφωνα με τις μετρικές tag non-obviousness και tag discrimination. Ωστόσο, τέτοιου είδους ετικέτες μπορεί να έχουν μικρή ή και καθόλου συσχέτιση με το άρθρο στο οποίο θα γίνει η σύσταση των ετικετών. Για το λόγο αυτό οι συγγραφείς προτείνουν να γίνει πρώτα ένας διαχωρισμός των ετικετών με βάση ήδη υπάρχοντα παρόμοια άρθρα τα οποία συσχετίζονται με το υπό εξέταση άρθρο και να παραμείνουν μόνο οι ετικέτες οι οποίες θεωρούνται σχετικές σύμφωνα με αυτή την σύγκριση. Ακολούθως από αυτές τις ετικέτες να προταθούν οι ετικέτες οι οποίες θεωρούνται ότι εμπεριέχουν την σημαντικότερη πληροφορία (informational powerful tags).

2.2 Ετικετοποίηση της ανθρώπινης γνώσης (Tagging Human Knowledge)

Η θεμελιώδης προϋπόθεση σε ένα σύστημα ετικετοποίησης είναι κατά πόσο μπορούν οι χρήστες του να οργανώνουν τα δεδομένα του συστήματος χρησιμοποιώντας ανεξέλεγκτο λεξιλόγιο. Οι συγγραφείς του άρθρου [6] θεωρούν ότι τα συστήματα ετικετοποίησης επιβάλλετε να έχουν τα εξής τρία χαρακτηριστικά συνέπεια, ποιότητα και πληρότητα.

Για να αξιολογήσουν κατά πόσο τέτοια συστήματα διαθέτουν αυτά τα χαρακτηριστικά συνέλεξαν δεδομένα από το LibraryThing και το Goodreads. Το LibraryThing είναι μια εφαρμογή κοινωνικής καταλογογράφησης διαδικτύου (social cataloging web application) για την αποθήκευση και την ανταλλαγή καταλόγων βιβλίων και διαφόρων τύπων μεταδεδομένων βιβλίου. Το Goodreads είναι μια ιστοσελίδα κοινωνικής καταναλογράφησης (social cataloging website) η οποία επιτρέπει στους χρήστες της να καταχωρούν βιβλία και να δημιουργούν τους δικούς τους καταλόγους βιβλίων καθώς και την δημιουργία ομάδων για την σύσταση και τον σχολιασμό βιβλίων.

Στη συνέχεια παρουσιάζονται τα αποτελέσματα της αξιολόγησης που έγινε με σκοπό να προσδιοριστεί κατά πόσο τα συστήματα ετικετοποίησης είναι συνεπείς, ποιοτικά και πλήρη. Οι συγγραφείς του άρθρου [6] έθιξαν τα πιο κάτω θέματα και οδηγήθηκαν στα εξής συμπεράσματα:

- **Συνέπεια:** Διατύπωσαν την ερώτηση κατά πόσο δημιουργείται πρόβλημα με τις συνώνυμες λέξεις όμως κατέληξαν στο συμπέρασμα ότι οι περισσότερες λέξεις έχουν λίγα ή και καθόλου συνώνυμα, επομένως δεν παρουσιάζεται πρόβλημα με τις συνώνυμες λέξεις. Επιπρόσθετα, παρατήρησαν ότι παρόμοια συστήματα ετικετοποίησης έχουν παρόμοιες ετικέτες επομένως υπάρχει συνέπεια. Επιπλέον, στο θέμα του κατά πόσο οι χρήστες χρησιμοποιούν ίδιες ετικέτες για να περιγράψουν ίδια αντικείμενα οδηγήθηκαν στο συμπέρασμα ότι αυτό συμβαίνει για δημοφιλείς ετικέτες όμως παρατηρείται μικρότερη συνέπεια για μη δημοφιλείς ετικέτες. Τέλος, παρατήρησαν ότι οι ετικέτες που προσθέτονται από τους απλούς χρήστες ενός συστήματος επικαλύπτουν τις ετικέτες που προσθέτουν μισθωτοί χρήστες ετικετοποίησης (paid taggers) σε ποσοστό 52 %.

- **Ποιότητα:** Παρατήρησαν ότι οι περισσότερες απλές ετικέτες οι οποίες είναι δημοφιλείς είναι συνδεδεμένες με το αντικείμενο το οποίο χαρακτηρίζουν και συσχετίζονται άμεσα μαζί του. Αυτό συμβαίνει ακόμη και εάν μια ετικέτα χρησιμοποιείται λίγες φορές από ένα χρήστη. Επιπλέον οδηγήθηκαν στο συμπέρασμα ότι οι μισθωτοί χρήστες ετικετοποίησης παράγουν περισσότερες και καλύτερης ποιότητας ετικέτες ειδικά σε παλιά αντικείμενα όπου η συνεισφορά των απλών χρηστών είναι φτωχή.

- **Πληρότητα:** Οδηγήθηκαν στο συμπέρασμα ότι οι συχνότερες ετικέτες που παρουσιάζονται σε μια θεματολογία αντιστοιχούν στους όρους των βιβλιοθηκών από ειδικούς(professional library annotations).

Κεφάλαιο 3

3. Εξόρυξη και Επεξεργασία συνόλου δεδομένων

3.1 Εξόρυξη συνόλου δεδομένων

Για να επιτευχθεί μια ρεαλιστική ανάλυση σε ετικέτες με τις οποίες οι χρήστες συνηθίζουν να χαρακτηρίζουν αρχεία λογισμικού ήταν αναγκαίο να συλλέξουμε πραγματικά δεδομένα. Συγκεκριμένα έπρεπε να συλλέξουμε ετικέτες οι οποίες εμπεριέχονται σε αρχεία λογισμικού. Για το λόγο αυτό υλοποιήσαμε έναν crawler, για την συλλογή αρχείων λογισμικού από το Google Code το οποίο είναι μια αποθήκη αρχείων λογισμικού. Από την διαδικασία του crawling προέκυψε ένα σύνολο από αρχεία λογισμικού (projects) τα οποία εμπεριείχαν και έναν αριθμό από ετικέτες. Στον πίνακα 2.1.1. παρουσιάζονται οι ακριβείς αριθμοί των δεδομένων που συλλέχθηκαν.

	Αριθμός
Projects	1295
Tags	3237

Πίνακας 2.1.1.:Αρχικό σύνολο δεδομένων

Για να πάρουμε μια πρώτη αίσθηση των δεδομένων που συλλέχτηκαν εντοπίσαμε τις ετικέτες με την μεγαλύτερη συχνότητα, καθώς και τα αρχεία λογισμικού που εμπεριέχουν τις περισσότερες ετικέτες. Στον πίνακα 2.1.2. εμφανίζονται τα αρχεία λογισμικού με τις περισσότερες ετικέτες όπου στην περίπτωση του Google Code υπάρχει προφανώς ανώτατο όριο ετικετών 15 για κάθε αρχείο. Στον πίνακα 2.1.3. εμφανίζονται οι ετικέτες με την μεγαλύτερη συχνότητα.

Α/Α	**Όνομα**	**Αριθμός από Tags**
1	Xdelta	15
2	winx	15
3	Web-optimizator	15
4	wadofstuff	15
5	visualfc	15
6	vision-lib	15
7	uim	15
8	txt2tags	15
9	textroom	15
10	tar-cs	15
11	speedcrunch	15
12	specto	15
13	sersync	15
14	scim-unikey	15
15	scim-python	15
16	ruijieclient	15
17	quitesleep	15
18	pyjamas	15
19	protoc-gen-as3	15
20	protobuf-actionscript3	15

Πίνακας 2.1.2.: Αρχεία λογισμικού με τις περισσότερες ετικέτες

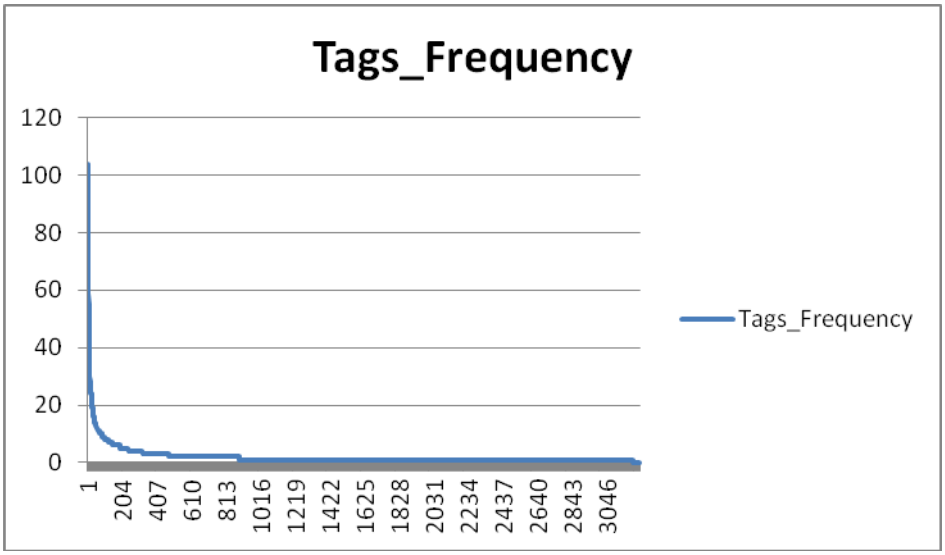
Α/Α	**Όνομα**	**Αριθμός Εμφανίσεων**
1	Python	377
2	Java	104
3	Linux	94
4	Google-summer-of-code	86
5	PHP	82
6	Google	79
7	Django	72
8	JavaScript	59
9	C	55
10	Api	47
11	Library	44
12	CPlusPlus	43
13	Web	38
14	Mysql	32
15	GTK	31
16	Framework	30
17	Xml	28
18	database	27
19	windows	27
20	Erlang	26

Πίνακας 2.1.3.: Ετικέτες με την μεγαλύτερη συχνότητα

3.2 Επεξεργασία συνόλου δεδομένων

Για να επιτευχθεί όσο το δυνατόν καλύτερη ανάλυση των ετικετών και να οδηγηθούμε σε σωστά συμπεράσματα θεωρήθηκε αναγκαίο να γίνει μια διαδικασία επεξεργασίας των αρχικών δεδομένων.

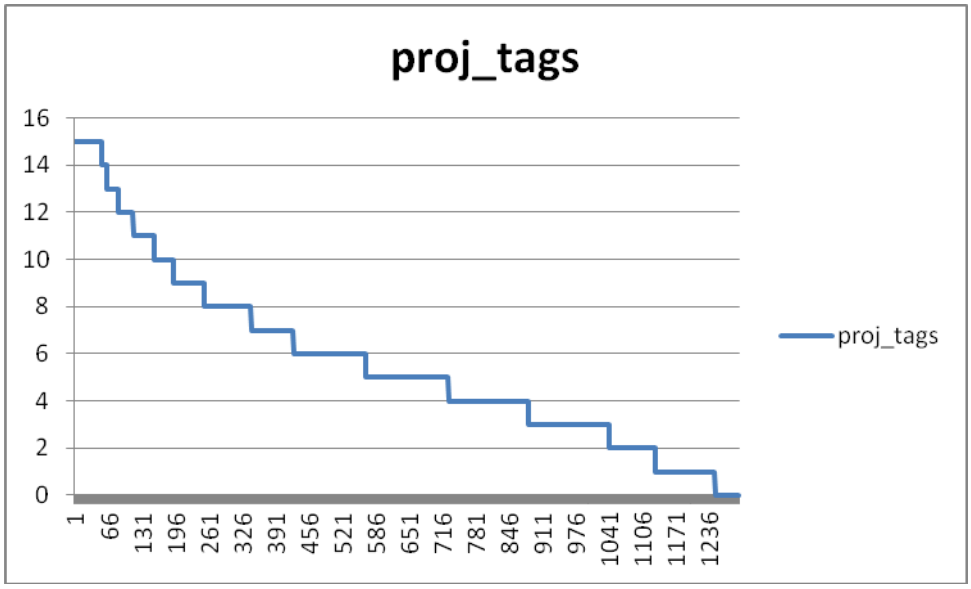
Αυτό που παρατηρήθηκε ήταν ότι οι περισσότερες ετικέτες εμφανίζονταν μόνο σε ένα αρχείο λογισμικού. Επομένως τέτοιου είδους ετικέτες δεν δίνουν ιδιαίτερη πληροφορία και είναι καλύτερο να διαγραφούν από το σύνολο των δεδομένων. Το γράφημα 2.2.1 αναπαριστά των αριθμό των εμφανίσεων των ετικετών στο σύνολο των αρχείων λογισμικού. Για σκοπούς καλύτερης απεικόνισης των δεδομένων έχει γίνει ταξινόμηση των ετικετών σε φθίνουσα σειρά με βάση τη συχνότητα που εμφανίζονται στα αρχεία λογισμικού.



Γράφημα 2.2.1.:Συχνότητα εμφάνισης των ετικετών στα αρχεία λογισμικού

Όπως μπορούμε να παρατηρήσουμε από το γράφημα οι περισσότερα ετικέτες έχουν πολύ χαμηλή συχνότητα κοντά στο 1. Έτσι λοιπόν έγινε μια επεξεργασία στο αρχικό σύνολο των ετικετών και αφαιρέθηκαν όσες ετικέτες εμφανίζονταν σε λιγότερα από δύο αρχεία λογισμικού και καταλήξαμε με ένα σύνολο δεδομένων με μόνο 476 ετικέτες.

Όσον αφορά τα αρχεία λογισμικού αυτό που παρατηρήθηκε ήταν το γεγονός ότι στο αρχικό σύνολο δεδομένων υπήρχαν αρχεία τα οποία δεν περιείχαν κανένα ή περιείχαν μόνο μια ετικέτα. Τέτοιου είδους αρχεία δεν θα συνεισέφεραν καθόλου στην ανάλυση των ετικετών ειδικά για την διαδικασία της ομαδοποίησης αλλά και για την διαδικασία της εξαγωγής κανόνων συσχέτισης. Στο γράφημα 2.2.2. αναπαριστάται ο αριθμός των ετικετών που εμπεριέχονται σε κάθε αρχείο λογισμικού σε φθίνουσα σειρά.



Γράφημα 2.2.2: Αριθμός ετικετών στα αρχεία λογισμικού

Όπως μπορούμε να παρατηρήσουμε υπάρχει ένας αριθμός από αρχεία λογισμικού που έχουν μόνο μία ετικέτα και υπάρχουν επίσης αρχεία χωρίς καμία. Αφαιρώντας λοιπόν τέτοιου είδους projects και αφήνοντας στο σύνολο δεδομένων αρχεία που περιέχουν πέραν τις μιας ετικέτας, καταλήξαμε με ένα σύνολο από 967 αρχεία λογισμικού.

Έτσι λοιπόν μετά την επεξεργασία που επιτεύχθηκε στο αρχικό σύνολο δεδομένων καταλήξαμε με ένα μικρότερο το οποίο παρουσιάζεται στον πίνακα 2.2.3.

	Αριθμός
Projects	967
Tags	476

Πίνακας 2.2.3.:Τελικό σύνολο δεδομένων μετά την επεξεργασία

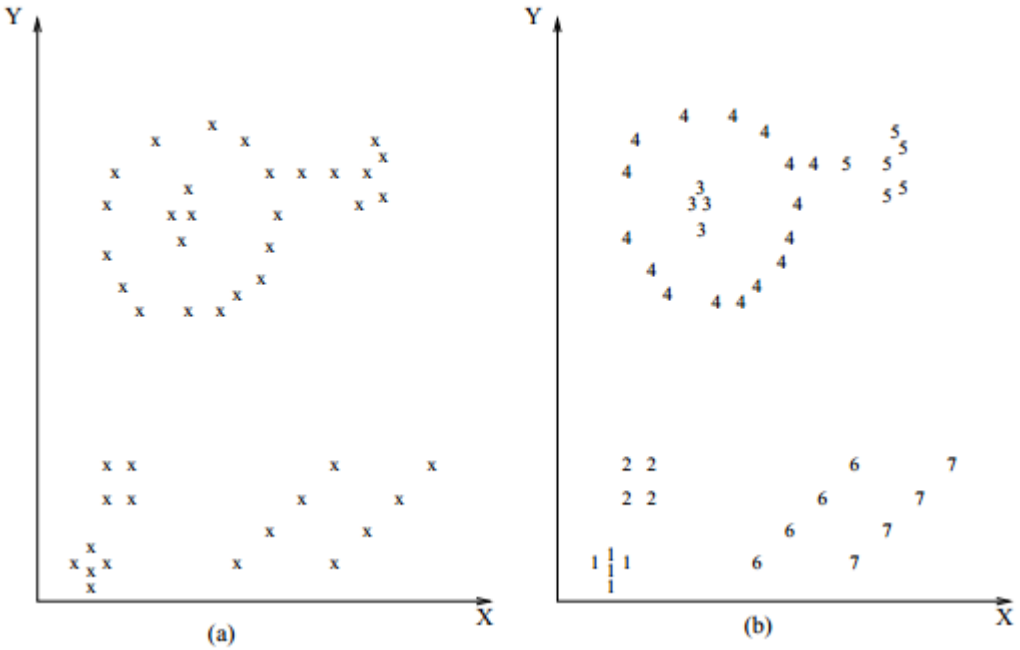
Κεφάλαιο 4

4. Ομαδοποίηση δεδομένων (Data Clustering)

4.1 Θεωρητικό υπόβαθρο γύρω από το cluster analysis

Ομαδοποίηση(clustering): είναι μια μαθηματική έννοια, όπου κύριος της ρόλος είναι η εξαγωγή ομάδων(clusters) με όμοια αντικείμενα βάση μιας μετρικής ομοιότητας. Τα αντικείμενα κάθε ομάδας είναι πιο "όμοια" σε σύγκριση με άλλα αντικείμενα άλλων ομάδων. Είναι μια διαδικασία κατηγοριοποίησης χωρίς επίβλεψη (unsupervised classification) η οποία λαμβάνει χώρα σε πολλές ερευνητικές περιοχές και αναγνωρίζεται ως ένα σημαντικό εργαλείο σε πολλές επιστήμες και ειδικά στο data mining για την εξαγωγή χρήσιμων συμπερασμάτων [7].

Ωστόσο, δεν είναι μια απλή διαδικασία και δεν αποτελείται από έναν αλγόριθμο αλλά μπορεί να επιτευχθεί με διάφορα μοντέλα αλγορίθμων, τα οποία μπορεί να διαφέρουν σημαντικά στην αντίληψη του τι αποτελεί μια ομάδα και πως αυτές οι ομάδες μπορούν να εντοπιστούν αποδοτικά. Ένα παράδειγμα ομαδοποίησης παρουσιάζεται στην εικόνα 4.1.1. όπου τα δεδομένα εισόδου παρουσιάζονται στο γράφημα (α) ενώ η ομαδοποίηση των δεδομένων στο γράφημα (b).



Εικόνα 4.1.1.: Παράδειγμα ομαδοποίησης

Οι πιο συνηθισμένες ερμηνείες του τι είναι ομάδες είναι οι εξής:

- Ομάδες με χαμηλή απόσταση μεταξύ των μελών της κάθε ομάδας.
- Πυκνές περιοχές των δεδομένων ως προς το χώρο.
- Στατιστικές κατανομές των δεδομένων.

Η επιλογή του κατάλληλου αλγορίθμου και των παραμέτρων του (αριθμός των clusters, distance function) για την ομαδοποίηση δεδομένων εξαρτάται από το κάθε διαφορετικό σύνολο δεδομένων και την προβλεπόμενη χρήση των αποτελεσμάτων. Η ομαδοποίηση δεν αποτελεί μια αυτόματη διαδικασία. Τις περισσότερες φορές χρειάζεται μια προ-επεξεργασία του αρχικού συνόλου δεδομένων και επιπρόσθετα μια σειρά από πειραματικές δοκιμές των αλγορίθμων και των παραμέτρων τους μέχρι να εξαχθεί το επιθυμητό αποτέλεσμα.
Με βάση λοιπόν το πρόβλημα πρέπει να επιλεγεί ο κατάλληλος αλγόριθμος. Αν και η ορολογία της έννοιας ομαδοποίηση μπορεί να γίνει γρήγορα κατανοητή εντούτοις με την χρήση διαφορετικών αλγορίθμων υπάρχει πιθανότητα να παρθούν διαφορετικά αποτελέσματα.

Υπάρχουν δύο βασικοί τύποι για αλγορίθμους ομαδοποίησης, οι ιεραρχικοί αλγόριθμοι(hierarchical algorithms) και οι αλγόριθμοι διαχωρισμού (partitioning algorithms).

- **Hierarchical algorithms:** αυτοί οι αλγόριθμοι δημιουργούν μια ιεραρχική αποσύνθεση του συνόλου δεδομένων D. Αυτή η αποσύνθεση παρουσιάζεται με ένα δενδρόγραμμα, το οποίο είναι ένα δένδρο στο οποίο διαιρείται το D επαναληπτικά σε μικρότερα υποσύνολα. Η διαδικασία μπορεί να επιτευχθεί από τα φύλλα προς την ρίζα του δέντρου, από κάτω προς τα πάνω (agglomerative approach) ή από την ρίζα προς τα φύλλα, από πάνω προς τα κάτω (devise approach). Αυτοί οι αλγόριθμοι έχουν το πλεονέκτημα ότι δεν χρειάζεται να οριστεί ο αριθμός των ομάδων ως παράμετρος.

- **Partitioning algorithms:** αυτοί οι αλγόριθμοι διαχωρίζουν το σύνολο δεδομένων D το οποίο αποτελείται από n αντικείμενα σε k ομάδες. Συνήθως ξεκινούν με έναν τυχαίο αρχικό διαχωρισμό του συνόλου και ακολούθως με επαναληπτικές εκτελέσεις βελτιστοποιούν των διαχωρισμό μέχρι να συγκλίνουν. Έχουν το πλεονέκτημα ότι για μεγάλα σύνολα δεδομένων είναι πολύ πιο αποδοτικοί από τους ιεραρχικούς, οι οποίοι απαιτούν την δημιουργία δενδρογράμματος το οποίο στην περίπτωση μεγάλων συνόλων δεδομένων η όλη διαδικασία είναι υπολογιστικά απαγορευτική. Εντούτοις αυτοί οι αλγόριθμοι έχουν το μειονέκτημα ότι πρέπει να προκαθοριστεί ο αριθμός των επιθυμητών ομάδων, δηλαδή να δοθεί ως παράμετρος ο αριθμός των ομάδων που θα δημιουργηθούν κατά την ομαδοποίηση. Αυτό αποτελεί πρόβλημα στην περίπτωση που δεν γνωρίζουμε τον αριθμό των ομάδων που υπάρχουν σε ένα σύνολο δεδομένων [7][8].

4.2 Θεωρητικό υπόβαθρο των βασικών μοντέλων ομαδοποίησης που χρησιμοποιήθηκαν

Στη συνέχεια παρουσιάζονται τα βασικά τυπικά μοντέλα ομαδοποίησης και οι αλγόριθμοι που χρησιμοποιήθηκαν για την διαδικασία ομαδοποίησης των ετικετών.

4.2.1 Centroid-based clustering

Αυτό το τυπικό μοντέλο θεωρείται το πιο δημοφιλές και συχνότερα χρησιμοποιημένο μοντέλο των αλγορίθμων διαχωρισμού. Κάθε ομάδα αναπαριστάται με ένα κεντρικό διάνυσμα(vector) το οποίο μπορεί και να μην είναι μέλος του συνόλου δεδομένων. Ουσιαστικά υποδηλώνει την τάση των μελών της συγκεκριμένης ομάδας και δίνει κατά κάποιο τρόπο τα κοινά χαρακτηριστικά των αντικειμένων που την αποτελούν.

Βασικό χαρακτηριστικό αυτών των αλγορίθμων είναι το γεγονός ότι πρέπει να καθοριστεί εκ των προτέρων ο αριθμός των ομάδων που θα δημιουργηθούν, όπως και κάθε άλλος αλγόριθμος που ανήκει στην κατηγορία των αλγόριθμων διαχωρισμού. Αυτό έχει ως αποτέλεσμα να παρουσιάζεται πρόβλημα βελτιστοποίησης, κατά πόσο δηλαδή μπορούν όλα τα αντικείμενα του συνόλου δεδομένων να καταταγούν στη σωστή ομάδα έτσι ώστε να ελαχιστοποιηθεί η απόσταση των αντικειμένων σε κάθε

ομάδα και να μεγιστοποιηθεί η απόσταση μεταξύ των διαφορετικών ομάδων. Το συγκεκριμένο πρόβλημα βελτιστοποίηση είναι γνωστό ως NP-hard και έτσι η συνήθης προσέγγιση είναι η εύρεση προσεγγιστικής λύσης.

Οι πιο συνήθεις συναρτήσεις απόστασης που χρησιμοποιούνται για τον εντοπισμό της ομοιότητας μεταξύ των αντικειμένων είναι οι εξής:

- Euclidian Distance: η απόσταση μεταξύ δύο διανυσμάτων βρίσκεται ως εξής:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \cdots + (y_n - x_n)^2} = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}.$$

- Manhattan Distance: η απόσταση μεταξύ δύο διανυσμάτων ορίζεται ως το άθροισμα της απόλυτης διαφοράς τους και βρίσκεται ως εξής:

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$$

Ο πιο γνωστός αλγόριθμος της συγκεκριμένη ομάδας αλγορίθμων είναι ο αλγόριθμος k-means [7].

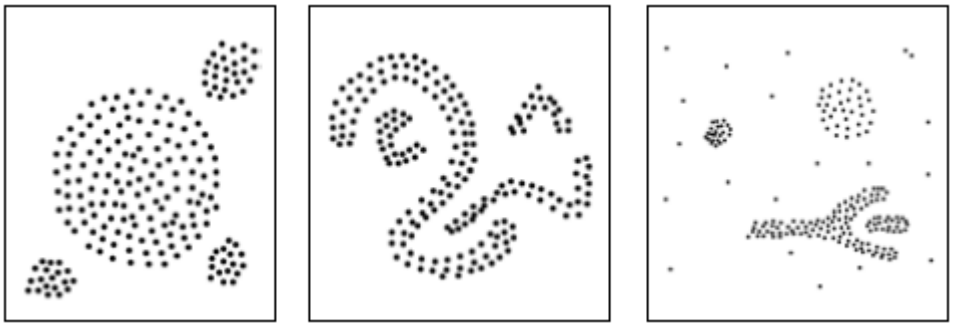
Αλγόριθμος ομαδοποίησης k-means:

1. Επιλέγουμε k τυχαία κέντρα για κάθε ομάδα τα οποία μπορεί να είναι σημεία από το σύνολο δεδομένων ή κάποια τυχαία σημεία.
2. Αναθέτουμε κάθε αντικείμενο του συνόλου δεδομένων στο πιο κοντινό κέντρο ομάδας με βάση μια μετρική απόστασης.
3. Ξανά-υπολογίζουμε τα κέντρα κάθε ομάδας με βάση τα αντικείμενα τα οποία ανατέθηκαν στην κάθε ομάδα.
4. Αν δεν υπάρξει σύγκλιση επιστρέφουμε στο βήμα 2 . Το τυπικό κριτήριο σύγκλισης είναι συνήθως η ελάχιστη μετακίνηση ή καμία μετακίνηση αντικειμένων σε μια άλλη ομάδα.

Ο αλγόριθμος k-means είναι πολύ δημο φλής για το λόγο ότι είναι εύκολο ς στην υλοποίησης. Εντούτοις, παρουσιάζει το βασικό πρόβλημα της σύγκλισης σε τοπικό βέλτιστο λόγω της τυχαίας αρχικοποίησης των κέντρων των ομάδων, η οποία μπορεί να μην είναι η κατάλληλη.

4.2.2 Density-based clustering

Για να γίνει κατανοητή η φιλοσοφία γύρω από αυτό το τυπικό μοντέλο αλγορίθμων αρκεί να παρατηρήσουμε την εικόνα 4.2 2.1 . Μπορούμε εύκολα να εντοπίσουμε τις ομάδες αντικειμένων σε κάθε σύνολο δεδομένων και επιπρόσθετα μπορούμε να εντοπίσουμε τα σημεία τα οποία δεν ανήκουν σε κάποια ομάδα. Ο βασικός λόγος για τον οποίο συμβαίνει αυτό είναι το γεγονός ότι κάθε ομάδα αποτελείται από σημεία υψηλής πυκνότητας σε σχέση με τα σημεία εκτός της ομάδας. Αντίθετα περιοχές με σημεία χαμηλότερης πυκνότητας θεωρούνται θορυβώδεις σημεία και δεν ανήκουν σε κάποια ομάδα.



Εικόνα 4.2.2.1.: Ομάδες αντικειμένων υψηλής πυκνότητας

Σε αυτό το μοντέλο αλγορίθμων ομάδες ορίζονται ως περιοχές με υψηλότερη πυκνότητα αντικειμένων σε σχέση με το υπόλοιπο σύνολο δεδομένων. Αντικείμενα που βρίσκονται σε αραιές περιοχές θεωρούνται θορυβώδεις και γι’ αυτό δεν αποτελούν μέλη κάποιου cluster.

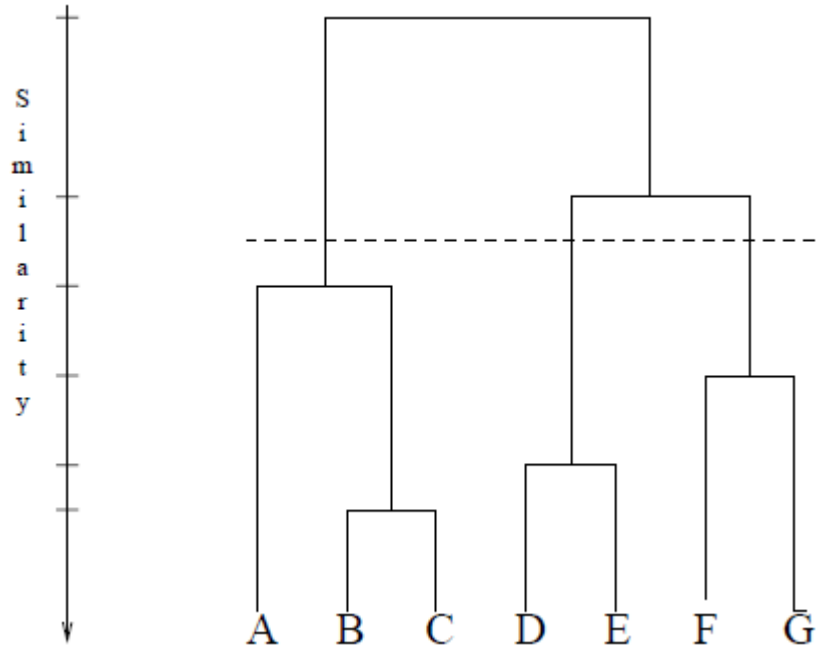
Η πιο γνωστή τέτοια μέθοδος είναι ο αλγόριθμος DBSCAN (Density Based Spatial Clustering of Applications with Noise). Η βασική ιδέα αυτού του αλγορίθμου είναι ότι βασίζεται σε σημεία σύνδεσης μεταξύ των αντικειμένων εντός ενός ορισμένου ορίου απόστασης. Ωστόσο, συνδέει μόνο σημεία τα οποία ικανοποιούν ένα κριτήριο πυκνότητας, το οποίο ορίζεται ως ο ελάχιστος αριθμός άλλων αντικειμένων εντός της ακτίνας αυτής. Ουσιαστικά λοιπόν ένα αντικείμενο **p** μπορεί να θεωρηθεί άμεσα συνδεδεμένο με ένα άλλο αντικείμενο **q,** μόνο αν η απόστασή τους είναι μικρότερη από μια καθορισμένη απόσταση **ε** και ικανοποιούν το κριτήριο πυκνότητας [7].

Σημαντικό πλεονέκτημα αυτών των αλγορίθμων είναι το γεγονός ότι έχουν χαμηλή πολυπλοκότητα (απαιτούν ένα γραμμικό αριθμό ερωτημάτων) και δεν χρειάζονται πολλές επαναλήψεις για την εξαγωγή των αποτελεσμάτων. Από την άλλη όμως μπορεί να θεωρηθεί μειονέκτημα τους το γεγονός ότι αναμένουν κάποια πτώση της πυκνότητας για να εντοπίσουν τα όρια των ομάδων, ειδικά σε ένα σύνολο δεδομένων όπου τα αντικείμενα δεν έχουν μεγάλες αποστάσεις.

4.2.3 Connectivity-based clustering (Hierarchical Clustering)

Βασίζεται στην ιδέα ότι τα αντικείμενα είναι πιο σχετικά με αντικείμενα που βρίσκονται πιο κοντά τους, σε σχέση με πιο μακρινά αντικείμενα. Δηλαδή τέτοιου είδους αλγόριθμοι συνδέουν αντικείμενα για να σχηματίσουν ομάδες με βάση την απόστασή τους.

Αυτές οι διαφορετικές ομάδες αντικειμένων μπορούν να αναπαρασταθούν με ένα δενδρόγραμμα. Σε ένα δενδρόγραμμα ο άξονας των Y σηματοδοτεί την απόσταση στην οποία συγχωνεύονται οι ομάδες, ενώ ο άξονας των X σηματοδοτεί ότι κατά μήκος αυτού του άξονα οι ομάδες δεν συνδέονται [7]. Στο σχήμα 4.2.3.1 παρουσιάζεται ένα παράδειγμα δενδρογράμματος.

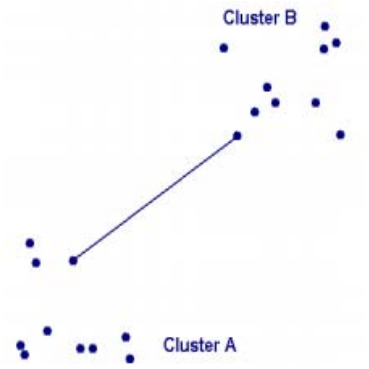


Σχήμα 4.2.3.1.: Παράδειγμα δενδρογράμματος

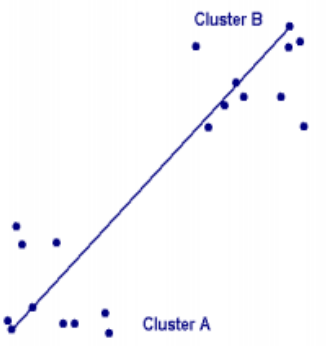
Η ιεραρχική ομαδοποίηση αποτελείται από μια ολόκληρη οικογένεια από μεθόδους οι οποίες διαφέρουν στον τρόπο υπολογισμού της απόστασης. Εκτός από τον υπολογισμό της απόστασης πρέπει να αποφασιστεί και το κριτήριο σύνδεσης, αφού μια ομάδα αποτελείται από πολλαπλά αντικείμενα και τίθεται το θέμα με βάση ποιο από αυτά τα αντικείμενα να υπολογιστεί η απόσταση μεταξύ ενός άλλου αντικειμένου (ή άλλης ομάδας).

Στη συνέχεια παρουσιάζονται οι τρεις βασικές επιλογές για τον υπολογισμό της απόστασης:

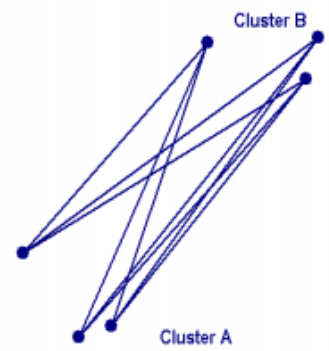
- **Single linkage Distance**: η απόσταση μεταξύ μιας ομάδας C1 και μιας ομάδας C2 είναι η **ελάχιστη** απόσταση μεταξύ ενός αντικειμένου του C1 και ενός αντικειμένου του C2.



- **Complete linkage Distance:** η απόσταση μεταξύ μιας ομάδας C1 και μιας ομάδας C2 είναι η **μέγιστη** απόσταση μεταξύ ενός αντικειμένου του C1 και ενός αντικειμένου του C2.



- **Group average Distance:** η απόσταση μεταξύ μιας ομάδας C1 και μιας ομάδας C2 είναι η **μέση** απόσταση μεταξύ των αντικειμένων του C1 και των αντικειμένων του C2.



Επιπλέον υπάρχουν δύο προσεγγίσεις για το πώς μπορεί να επιτευχθεί μια ιεραρχική ομαδοποίηση. Συγκεκριμένα:

-**Agglomerative:** όπου αρχικά κάθε σημείο αποτελεί μια ομάδα και στη συνέχεια επαναλαμβανόμενα ,συνδέοντας κάθε φορά τις δύο πιο κοντινές ομάδες, δημιουργείται στο τέλος μια ενιαία ομάδα που εμπεριέχει όλα τα σημεία.

-**Divisive:** όπου αρχικά έχουμε μία ομάδα με όλα τα σημεία και στη συνέχεια μέσω μιας επαναληπτικής διαδικασίας διασπούμε τις ομάδες για να πάρουμε στο τέλος μια ομάδα για κάθε σημείο.

Η πολυπλοκότητα τέτοιου είδους αλγορίθμων είναι $O(n^3)$ και είναι πολύ αργοί για μεγάλο αριθμό δεδομένων [7].

4.2.4 Graph-based clustering στο Gephi

4.2.4.1 Πρόβλημα ανίχνευσης κοινοτήτων

Το πρόβλημα της ανίχνευσης κοινοτήτων σε ένα γράφο απαιτεί το χώρισμα του γράφου σε κοινότητες πυκνά συνδεδεμένων κόμβων, με τους κόμβους που ανήκουν σε διαφορετικές κοινότητες να είναι αραιά συνδεδεμένοι μεταξύ τους. Για την επίλυση του συγκεκριμένου προβλήματος και την ανίχνευση κοινοτήτων σε πολύ μεγάλους γράφους, έχουν κατασκευαστεί αρκετοί αλγόριθμοι πολλοί από τους οποίους έχουν πολύ μεγάλη πολυπλοκότητα (minimum-cut method, hierarchical clustering, Girvan-Newman algorithm, Louvain method) .

4.2.4.2 Ανίχνευση κοινοτήτων στο Gephi

Gephi: είναι ένα εργαλείο που χρησιμοποιείται για την ανάλυση και απεικόνιση μεγάλων δικτύων. Υποστηρίζει πολλές λειτουργίες όπως απεικόνιση δικτύων(Visualization), Ranking, ανίχνευση κοινοτήτων(Community Detection),την εξαγωγή χρήσιμων στατιστικών και πολλές άλλα [9].

Χρησιμοποιήσαμε το Gephi για την ανίχνευση κοινοτήτων σε ένα γράφο, όπου κάθε κοινότητα αντιπροσωπεύει μια ομάδα από ετικέτες.
Η μέθοδος που χρησιμοποιεί το Gephi για την διαδικασία της ανίχνευσης κοινοτήτων είναι η Louvain method η οποία περιγράφετε στην συνέχεια.

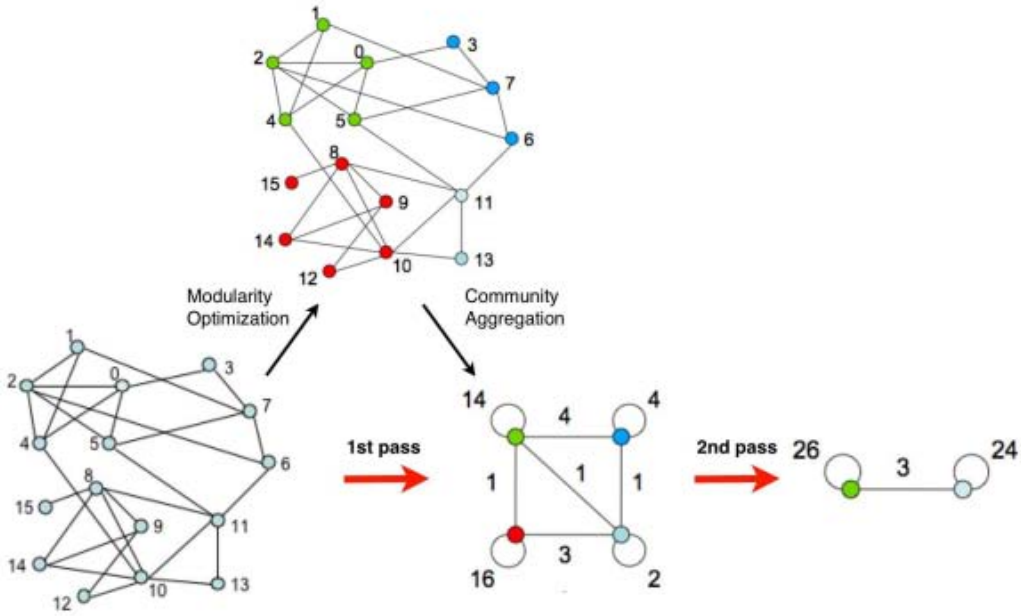
4.2.4.3 Louvain method

Η Louvain είναι μια μέθοδος η οποία ανιχνεύει κοινότητες σε ένα γράφο. Η συγκεκριμένη μέθοδος επιτυγχάνετε σε δύο φάσεις [10] όπως φαίνεται και στο σχήμα 4.2.4.3.1.

- **Πρώτη φάση:** Αρχικά κάθε κόμβος του γράφου αποτελεί μια κοινότητα. Στη συνέχεια για κάθε κόμβο i του γράφου εξετάζεται κατά πόσο η μετακίνηση του στην κοινότητα ενός γειτονικού κόμβου j βελτιώνει το χώρισμα του γράφου. Ο κόμβος i θα μετακινηθεί στην κοινότητα όπου το όφελος του χωρίσματος

μεγιστοποιείται. Αυτή η διαδικασία είναι επαναληπτική και σταματά όταν επιτευχθεί το καλύτερο χώρισμα των κόμβων.

- **Δεύτερη φάση:** Σε αυτή τη φάση κατασκευάζεται ένα νέο δίκτυο, οι κόμβοι του οποίου είναι οι κοινότητες που δημιουργήθηκαν κατά την πρώτη φάση. Τα βάρη των συνδέσεων μεταξύ των δύο κοινοτήτων δίνονται από το άθροισμα των βαρών των κόμβων των δύο κοινοτήτων.



Σχήμα 4.2.4.3.1: Παράδειγμα μεθόδου Louvain

4.2.4.4 Κατασκευή γράφου

Για την επίτευξη του graph based clustering δημιουργήσαμε ένα γράφο. Κάθε κόμβος του γράφου αντιπροσωπεύει μια ετικέτα. Ακμή μεταξύ δύο κόμβων του γράφου υπάρχει μόνα εάν αυτές οι δύο ετικέτες εμφανίζονται έστω και μια φορά στο ίδιο αρχείο λογισμικού. Επιπρόσθετα το βάρος μιας ακμής μεταξύ δύο κόμβων ορίζεται ως ο αριθμός των εμφανίσεων των δύο ετικετών σε κοινά αρχεία λογισμικού δια τον αριθμό όλων των αρχείων λογισμικού.

4.3 Παρουσίαση πειραμάτων ομαδοποίησης

4.3.1 Weka (Waikato Environment for Knowledge Analysis)

Weka: είναι ένα λογισμικό (workbench) για μηχανική μάθηση (machine learning) το οποίο προορίζεται στο να κάνει τις εφαρμογές των τεχνικών μηχανικής μάθησης πιο εύκολες και να βοηθήσει όσους ασχολούνται γύρω από αυτή την περιοχή. Έχει υλοποιηθεί στο πανεπιστήμιο του Waikato στην Νέα Ζηλανδία. Αποτελείται από μια συλλογή από αλγορίθμους μηχανικής μάθησης υλοποιημένους σε γλώσσα Java και συγκεκριμένα εμπεριέχει μαθησιακά σχήματα για κατηγοριοποίηση, ομαδοποίηση, εξαγωγή κανόνων συσχέτισης και πολλά άλλα [11] [13]. Για την πραγματοποίηση της ομαδοποίησης των ετικετών και την εξαγωγή των κανόνων συσχέτισης έχουμε χρησιμοποιήσει το Weka.

4.3.2 Μετατροπή δεδομένων σε μορφή arff

Για να επιτευχθεί η διαδικασία της ομαδοποίησης των ετικετών χρησιμοποιώντας το Weka χρειάζεται να μετατρέψουμε τα δεδομένα εισόδου μας στην μορφή arff (attribute-relation file format) [12]. Ένα arff αρχείο είναι ένα αρχείο κειμένου το οποίο περιγράφει μια λίστα από αντικείμενα τα οποία μοιράζονται κάποια χαρακτηριστικά και έχει την μορφή που φαίνεται στο σχήμα 4.3.2.1.

```
@relation GoogleCodeProjectsAndTags

@attribute 0xbench {yes,no}
@attribute 4bricks {yes,no}
@attribute aam-opencv {yes,no}
@attribute abc2esac {yes,no}
@attribute acre {yes,no}

@data
yes,no,no,no,no
no,yes,yes,no,no
no,yes,no,no,no
no,no,no,yes,yes
```

Σχήμα 4.3.2.1.: Αρχείο μορφής arff

4.3.3 Αποτελέσματα ομαδοποίησης

4.3.3.1 Αλγόριθμος Simple-K-Means

Όπως προαναφέραμε στο κεφάλαιο 4.2.1 για να τρέξουμε τον αλγόριθμο Simple-K-Means χρειάζεται να ορίσουμε τον αριθμό των επιθυμητών ομάδων. Το γεγονός ότι δεν γνωρίζουμε από πριν αυτό τον αριθμό μας οδήγησε στο να πραγματοποιήσουμε μια σειρά από πειράματα αλλάζοντας κάθε φορά τον αριθμό των ομάδων. Η αξιολόγηση και σύγκριση των αποτελεσμάτων παρουσιάζεται στο επόμενο κεφάλαιο. Στον πίνακα 3.3.3.1.2 παρουσιάζονται οι ομάδες που προέκυψαν χρησιμοποιώντας τον αλγόριθμο Simple-K-Means και τις παραμέτρους που φαίνονται στον πίνακα 4.3.3.1.1. Παρουσιάζονται μόνο οι ομάδες που εμπεριέχουν πέραν της μίας ετικέτας. Ο αλγόριθμος δημιούργησε και ομάδες που εμπεριείχαν μόνο μια ετικέτα.

Παράμετροι	
Distance Function:	Euclidian Distance
Number of Clusters:	60

Πίνακας 4.3.3.1.1: Παράμετροι αλγορίθμου

Cluster1	sample, wsdl, soa
Cluster2	robotframework, testautomation
Cluster3	linux, gnome, gtk
Cluster4	pys60, cross-platform
Cluster5	windows, qt, editor
Cluster6	google, adwords, soap
Cluster7	games, concurrency, networking
Cluster8	authentication, saml2.0, shibboleth, simplesamlphp
Cluster9	uml, swf, devtool, flex, scala, actionscript
Cluster10	flash, aac, rtmp, mp4
Cluster11	wii, homebrew
Cluster12	mysql, administration, replication, cluster
Cluster13	jabber, webservices, social, artificialintelligence
Cluster14	java, groovy, datamining, nlp, hadoop, mapreduce, machinelearning
Cluster15	asynchronous, event, libevent
Cluster16	rss, atom
Cluster17	stl, boost, stdext
Cluster18	testing, unittesting, mock
Cluster19	json, django, subversion, form, smf
Cluster20	audio, midi, jack
Cluster21	tracking, map
Cluster22	rest, gdata, webservice
Cluster23	bot, gozerbot, gae, wave, irc
Cluster24	algebra, mathematics, geometry, manipulation, cas, symbolic
Cluster25	osx, objective-c, objc

Πίνακας 4.3.3.1.2: Αποτελέσματα Simple-K-Means

Σχολιασμός αποτελεσμάτων

Όπως μπορούμε να παρατηρήσουμε στον πίνακα 4.3.3.1.2 δεν παρουσιάζονται όλες οι ετικέτες. Ο αλγόριθμος έχει τοποθετήσει τις υπόλοιπες ετικέτες στην ίδια ομάδα. Αυτό συμβαίνει για το λόγο ότι τα διανύσματα που αντιπροσωπεύουν κάθε ετικέτα είναι αραιά (sparse) και ο αλγόριθμος αφού δεν κατάφερε να τα κατατάξει σε κάποιο άλλη ομάδα, τα κατάταξε στην ομάδα η οποία είχε ως κέντρο της το διάνυσμα που παρουσιάζει σε κάθε θέση του "no" (δηλαδή ότι η ετικέτα δεν παρουσιάζεται σε κανένα αρχείο). Οι ετικέτες που ανατέθηκαν σε αυτή την ομάδα είναι ετικέτες που εμφανίζονται σε πολύ λίγα αρχεία γι' αυτό και η απόσταση του από το κέντρο αυτής της ομάδας ήταν η μικρότερη. Επιπρόσθετα ο αλγόριθμος έχει δημιουργήσει 34 ομάδες οι οποίες εμπεριέχουν μόνο μια ετικέτα. Αυτές οι ομάδες δεν παρουσιάζονται στον πίνακα 4.3.3.1.2 και ο λόγος που δημιουργήθηκαν ήταν λόγω της τυχαίας αρχικοποίησης των κέντρων του αλγόριθμου Simple-K-Means.

Από μια εμπειρική αξιολόγηση των αποτελεσμάτων που έγινε παρατηρήθηκε ότι αυτός ο αλγόριθμος παράγει ομάδες ετικετών που συσχετίζονται σε μεγάλο βαθμό μεταξύ τους. Για παράδειγμα η ομάδα 24 που φαίνεται στον πίνακα 4.3.3.1.2 εμπεριέχει τις ετικέτες "algebra", "mathematics", "geometry", "manipulation", "cas" και "symbolic" οι οποίες όπως είναι προφανές είναι σχετικές με λογισμικό το οποίο πολύ πιθανόν να αφορά τα Μαθηματικά.
Όσον αφορά την μεγάλη ομάδα στην οποία εμπεριέχονται οι περισσότερες ετικέτες και η οποία σχολιάστηκε στην προηγούμενη παράγραφο δεν αποτελεί μια ποιοτική ομάδα και οι ετικέτες δεν συσχετίζονται μεταξύ τους.

4.3.3.2 Αλγόριθμος DBSCAN

Όπως προαναφέραμε στο κεφάλαιο 4 2.2 για να τρέξουμε τον αλγόριθμο DBSCAN χρειάζεται να ορίσουμε την παράμετρο e. Το γεγονός ότι δεν γνωρίζουμε από πριν ποια είναι η κατάλληλη τιμή για την παράμετρο e μας οδήγησε στο να πραγματοποιήσουμε μια σειρά από πειράματα αλλάζοντας κάθε φορά την τιμή της. Η αξιολόγηση και σύγκριση τον αποτελεσμάτων παρουσιάζεται στο επόμενο κεφάλαιο. Στον πίνακα 4.3.3.2.2 παρουσιάζονται οι ομάδες που προέκυψαν χρησιμοποιώντας τον αλγόριθμο DBSCAN και τις παραμέτρους που φαίνονται στον πίνακα 4.3.3.2.1. Θέσαμε τον ελάχιστο αριθμό αντικειμένων σε μια ομάδα ίσο με 2, για να πάρουμε όλες τις ομάδες που μπορούν να δημιουργηθούν, ακόμα και αν περιέχουν μόνο δύο ετικέτες.

Παράμετροι	
E(epsilon)	1.8
MinPointsInCluster	2

Πίνακας 4.3.3.2.1: Παράμετροι αλγορίθμου

Cluster 1	datamining, hadoop, mapreduce
Cluster 2	midi, jack, opensocial, gnuplot, genomics, repository, research, iptables, clang, interactive, book, cron, gsoc, l10n, form, imageprocessing, computation, high-performance, epub, sms, sound, recognition, r, line, haskell, docbook, computervision, ocr, tex, social, rst, smf, restructuredtext
Cluster 3	aac, rtmp, mp4
Cluster 4	cocoa, objective-c, objc
Cluster 5	xlib, panel, openbox, tint2
Cluster 6	mercurial, bazaar, git, cvs
Cluster 7	datatypes, regularexpressions
Cluster 8	sip, voip
Cluster 9	oracle, sqlserver
Cluster 10	turbogears, genshi
Cluster 11	saml2.0, shibboleth, simplesamlphp
Cluster 12	unittesting, mock
Cluster 13	spring, guice
Cluster 14	pys60, cross-platform
Cluster 15	memcachedb, berkeleydb
Cluster 16	numpy, scipy
Cluster 17	actionscript3, protocol, as3
Cluster 18	soap, wsdl
Cluster 19	dfp, doubleclick
Cluster 20	stl, boost, stdext
Cluster 21	manipulation, cas, symbolic

Cluster 22	pidgin, libpurple
Cluster 23	serialization, rpc
Cluster 24	input, xim
Cluster 25	m17n, ibus, scim
Cluster 26	tracking, map
Cluster 27	machine-learning, large-scale
Cluster 28	swe, ogc, sensorweb, sensor
Cluster 29	yubico, yubikey
Cluster 30	robotframework, testautomation
Cluster 31	rdfxml, turtle

Πίνακας 4.3.3.2.2: Αποτελέσματα DBSCAN

Σχολιασμός αποτελεσμάτων

Όπως μπορούμε να παρατηρήσουμε στον πίνακα 4.3.3.2.2 ο αλγόριθμος έχει δημιουργήσει 31 ομάδες όμως δεν έχει κατατάξει όλες τις ετικέτες σε κάποια ομάδα. Ο αλγόριθμος DBSCAN έχει θεωρήσει τις υπόλοιπες 369 ετικέτες ως θορυβώδεις δεδομένα, δηλαδή σημεία τα οποία δεν μπορούν να καταταγούν σε κάποια ομάδα για το λόγο ότι δεν ανήκουν εντός του εύρους της epsilon γειτονιάς μια ομάδας και επιπρόσθετα δεν ανήκουν σε κάποια πυκνή περιοχή δεδομένων.

Από μια εμπειρική αξιολόγηση των αποτελεσμάτων που έγινε παρατηρήθηκε ότι αυτός ο αλγόριθμος παράγει ομάδες ετικετών που συσχετίζονται σε μεγάλο βαθμό μεταξύ τους. Για παράδειγμα η ομάδα 9 που φαίνεται στον πίνακα 4.3.3.2.2 εμπεριέχει τις ετικέτες “oracle” και “sqlserver” οι οποίες όπως είναι προφανές είναι σχετικές και αφορούν συστήματα βάσης δεδομένων. Άλλο παράδειγμα είναι η ομάδα 26 η οποία περιέχει τις ετικέτες “map” και “tracking” και οι οποίες επίσης είναι σχετικές.

4.3.3.3 Ιεραρχικός αλγόριθμος

Τρέξαμε τον ιεραρχικό αλγόριθμο πολλές φορές με διαφορετικό τύπο υπολογισμού της απόστασης μεταξύ δύο ομάδων (linkage type). Το γεγονός ότι στα δεδομένα που συλλέξαμε υπάρχει ένα μεγάλος αριθμός ετικετών οι οποίες μπορεί να θεωρηθούν ως θορυβώδεις δεδομένα, είχε ως αποτέλεσμα οι τύποι σύνδεσης single-linkage, complete-linkage και average-linkage να μην εξάγουν καλά αποτελέσματα ομαδοποίησης. Αυτοί οι τύποι σύνδεσης είναι ευαίσθητοι σε θορυβώδεις και ακραία δεδομένα (noise και outliers) και γι' αυτό το λόγο δεν εξήγαγαν ικανοποιητικά αποτελέσματα. Αυτό είχε ως αποτέλεσμα να οδηγηθούμε στην χρησιμοποίηση του τύπου centroid-linkage ο οποίο είναι λιγότερο ευαίσθητος σε θορυβώδεις σημεία.

Στον πίνακα 4.3.3.3.1 παρουσιάζονται οι ομάδες που προέκυψαν χρησιμοποιώντας τον ιεραρχικό αλγόριθμο με centroid-linkage.

Cluster0	android, url, cocoa, iphone, objective-c, objc
Cluster1	performance, benchmark, bioinformatics, genomics, benchmarking, r
Cluster2	Javascript
Cluster3	Php
Cluster4	opencv, ai, nlp, machinelearning, numpy, recognition, scipy, ocr, artificialintelligence
Cluster5	tracking, subversion, mercurial, bazaar, git, cvs, svn, build, pygtk, forms, gps, gis, openlayers, validation, map
Cluster6	Python
Cluster7	music, audio, midi, jack, mp3, aac, tagging, rtmp, sms, gpl, sound, mp4
Cluster8	academic, repository, research, compression, diff, gsoc, google-summer-of-code, algorithm
Cluster9	unix, posix, devtool, getopt, aop
Cluster10	Linux
Cluster11	Google
Cluster12	Framework
Cluster13	twisted, gozerbot, gae, xmpp, jabber, wave
Cluster14	game, chess, engine, book, cpp
Cluster15	sqlite, mysql, database, postgresql, oracle, sql, sqlserver
Cluster16	mvc, jquery, webkit, browser, mobile, gears
Cluster17	dns, server, client, daemon, pidgin, libpurple
Cluster18	gedit, plugin, code, oauth
Cluster19	gnome, bot, mit, irc, chat, robot
Cluster20	adwords, api, sample, soap, wsdl, dfa, dfp, dotnet, soa, doubleclick
Cluster21	Xml
Cluster22	xslt, ajax, xpath, maps, dom, trie, widgets
Cluster23	template, codegeneration, documentation, uml, gnuplot, latex, html, chm, xhtml, tex
Cluster24	parser, middleware, developer, dbus, datatypes, regularexpressions, parsing

Cluster25	web.py, rest, turbogears, genshi, sqlalchemy, webservice, streaming, swe, ogc, sensorweb, sensor
Cluster26	graphics, opengl, 3d, cgi, 2d
Cluster27	pipeline, flash, flex, actionscript, actionscript3, searchappliance, protocol, generator, webserver, protobuf, serialization, rpc, as3, protocolbuffers
Cluster28	Cplusplus
Cluster29	flickr, script, bash, pyglet, cocos2d, shell, ide, debugger
Cluster30	svg, pys60, admin, j2me, logging, interactive, scientific, log, cross-platform, qt4, line
Cluster31	Java
Cluster32	json, appengine, wsgi, network
Cluster33	gtk, keyboard, x11, xlib, panel, gui, openbox, tint2
Cluster34	windows, osx, mac
Cluster35	word, csharp, .net, bsd, freebsd, mono, i18n, l10n, im, chinese, input, xim, m17n, ibus, inputmethod, scim
Cluster36	rdfxml, rdf, semanticweb, turtle, microformats, llvm, clang, semantic, imageprocessing, intel, computervision, schema, development, libraries
Cluster37	commandline, tool, calendar, time, opensource, opened
Cluster38	groovy, ant, statistics, datamining, hadoop, mapreduce, parallel, wordpress, distributed, machine-learning, mpi, large-scale
Cluster39	junit, firefox, maven, testing, chrome, test, tdd, unittesting, spring, python3, mock, guice, scanner
Cluster40	Web
Cluster41	apache, asynchronous, event, libevent, epoll, nginx
Cluster42	visualization, graph, chart, stl, boost, stdext, ctypes, graphs
Cluster43	templates, monitoring, memcached, memcachedb, memcache, plugins, hash, berkeleydb
Cluster44	swf, application, scala, translation, gwt, compiler, desktop
Cluster45	administration, backup, ruby, rsync, sync, ssh, management, synchronization, replication, cluster, configuration
Cluster46	cloudcomputing, http, cloud, proxy, simple, queue, apps
Cluster47	memory, mediawiki, pdf, compress, epub, wiki, wikipedia, haskell, docbook, rst, restructuredtext
Cluster48	embedded, arduino, processing, macosx, apple, supplicant
Cluster49	command-line, autocomplete, automation, youtube, downloader, cli, captcha, upload
Cluster50	qt, nintendo, canvas, html5, wysiwyg, pygame, pyqt
Cluster51	video, media, ffmpeg, cuda, stream, jpeg
Cluster52	C
Cluster53	opensocial, extension, component, enterprise, webservices, social, owasp
Cluster54	utility, kernel, internet, cache, ubuntu, aggregator, news, kml, ldap
Cluster55	Library
Cluster56	mpd, filesystem, fuse, lyrics, last.fm, wrapper
Cluster57	lisp, games, emacs, concurrency, color, scheme, multimedia, inferno, interpreter, top, process
Cluster58	Django
Cluster59	blog, eclipse, cms, saml2.0, shibboleth, drupal, simplesamlphp, symfony, php5, qcodo
Cluster60	text, editor, console, emulator, wii, homebrew, terminal
Cluster61	rss, atom, gdata
Cluster62	amazon, s3, ec2, erlang, mnesia, multithreaded, cherrypy
Cluster63	Security

Cluster64	fuzzing, css, optimization, gstreamer, language, programming, ui, regex, p2p
Cluster65	crossplatform, iptables, cron, lua, utilities, networking, fast
Cluster66	analysis, solr, image, filter, search, interface, widget, form, xapian, smf, tuio
Cluster67	perl, math, computation, high-performance, algebra, mathematics, geometry, manipulation, physics, cas, symbolic
Cluster68	sip, simulation, authentication, module, encryption, cryptography, aes, www, password, otp, voip, yubico, yubikey
Cluster69	jython, twitter, microblogging, virtualmachine, robotframework, testautomatio

Πίνακας 4.3.3.3.1: Αποτελέσματα Ιεραρχικού Αλγόριθμου

Σχολιασμός αποτελεσμάτων

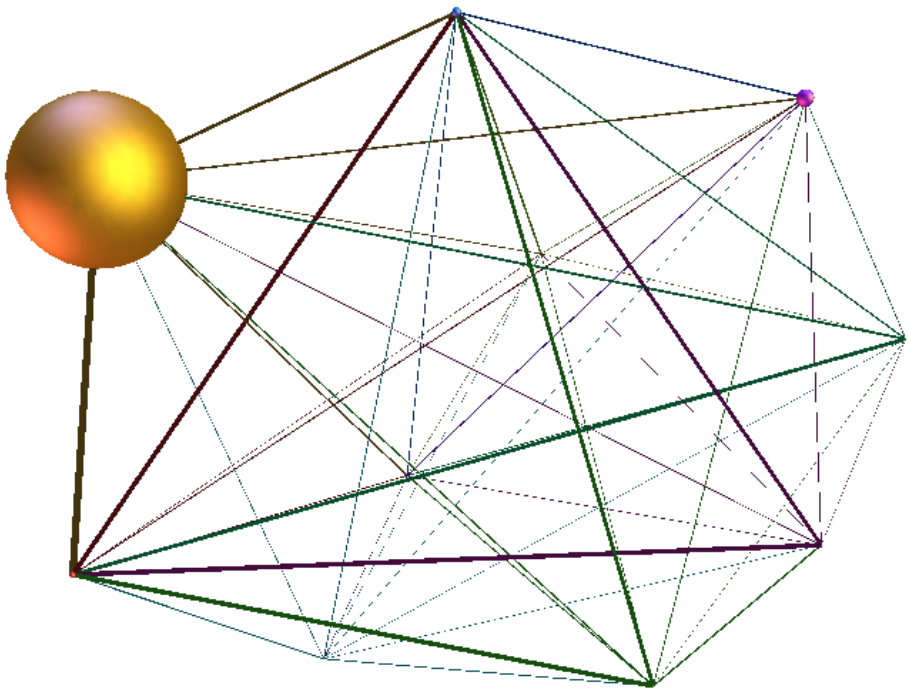
Ο ιεραρχικός αλγόριθμος κατατάσσει όλες τις ετικέτες σε κάποια ομάδα. Όπως έχουμε προαναφέρει δημιουργεί ένα δενδρόγραμμα στο οποίο για να εξαχθούν οι ομάδες χρειάζεται να οριστεί ο αριθμός των ομάδων που επιθυμούμε να δημιουργηθούν. Όπως μπορούμε να παρατηρήσουμε από τον πίνακα 4.3.3.3.1 έχουν δημιουργηθεί μεγαλύτερου μεγέθους ομάδες.

Από μια εμπειρική αξιολόγηση των αποτελεσμάτων που έγινε παρατηρήθηκε ότι αυτός ο αλγόριθμος παράγει από τη μια ομάδες ετικετών που συσχετίζονται σε μεγάλο βαθμό μεταξύ τους, όμως το γεγονός ότι τις κατατάσσει όλες σε κάποια ομάδα έχει ως αποτέλεσμα την δημιουργία και ομάδων χαμηλότερης συσχέτισης.

Για παράδειγμα η ομάδα 15 που φαίνεται στον πίνακα 4.3.3.2.1 εμπεριέχει τις ετικέτες “sqlite”, “mysql”, “database”, “postgresql”, “oracle”, “sql” και “sqlserver” αποτελεί μια πάρα πολύ καλή ομάδα. Σε αντίθεση με τον αλγόριθμο DBSCAN ο ιεραρχικός αλγόριθμος εντόπισε όλες τις ετικέτες που σχετίζονται με συστήματα βάσεις δεδομένων και τις κατέταξε στην ίδια ομάδα. Εντούτοις, όμως δημιούργησε και ομάδες οι οποίες εμπεριέχουν ετικέτες οι οποίες δεν σχετίζονται απόλυτα με όλες τις άλλες. Για παράδειγμα η ομάδα 67 η οποία αποτελείται από τιε ετικέτες “perl”, “math”, “computation”, “high-performance”, “algebra”, “mathematics”, “geometry”, “manipulation”, “physics”, “cas” και “symbolic” είναι μια ομάδα η οποία αποτελείται από ετικέτες που αφορούν κυρίως τα μαθηματικά. Σε αυτή την ομάδα όμως εμπεριέχεται και η ετικέτα “perl” η οποία χαρακτηρίζει μια γλώσσα προγραμματισμού και δεν σχετίζεται απόλυτα με τις άλλες ετικέτες. Κατατάχθηκε σε αυτή την ομάδα για το λόγο ότι είχε την μικρότερη απόσταση σε σχέση με τις άλλες ομάδες όμως δεν συνδέεται απόλυτα με τις άλλες ετικέτες που αποτελούν αυτή την ομάδα.

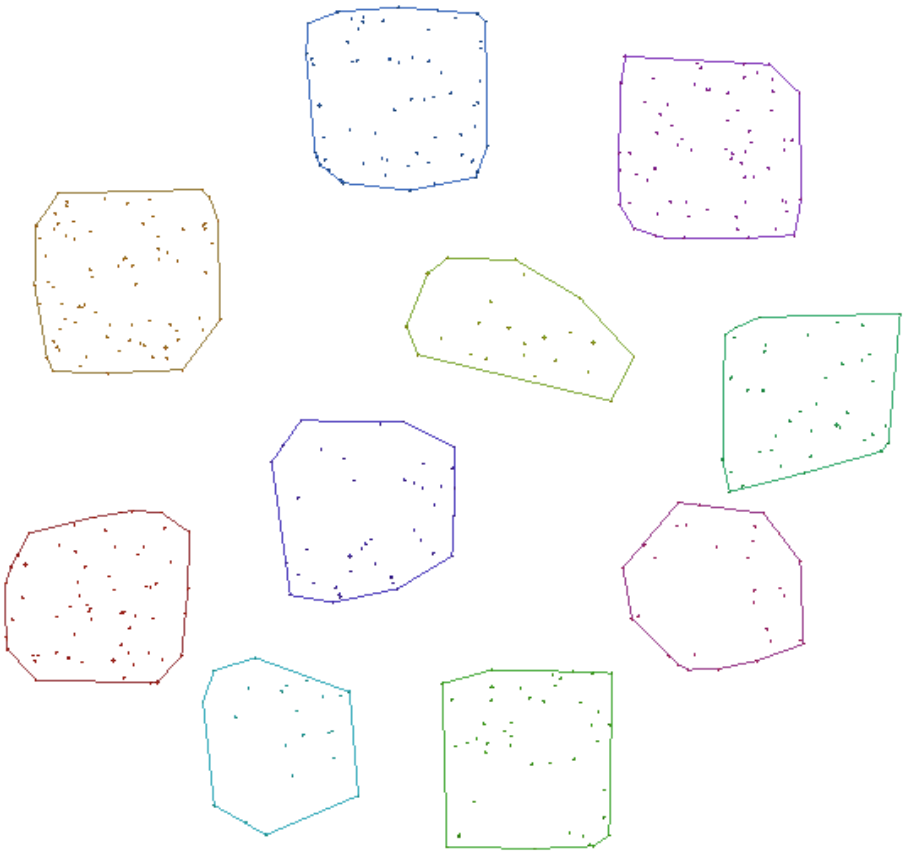
4.3.3.4 Αλγόριθμος Graph-Based

Όπως προαναφέρθηκε, για να πραγματοποιήσουμε Graph-Based ομαδοποίηση των ετικετών χρησιμοποιήσαμε το Gephi [9] και την μέθοδο Louvain [10] η οποία περιγράφεται στο υπό-κεφάλαιο 4.2.4.3. Το Gephi διαθέτει και γραφικό περιβάλλον και υποστηρίζει την απεικόνιση των δεδομένων στο χώρο. Για το λόγο αυτό στη συνέχεια παρουσιάζονται τα αποτελέσματα της Graph-Based ομαδοποίησης όπως αυτά παρουσιάζονται στο Gephi. Χρησιμοποιώντας την μέθοδο Louvain δημιουργήθηκαν οι πιο κάτω 10 ομάδες οι οποίες παρουσιάζονται στο σχήμα 4.3.3.4.1.



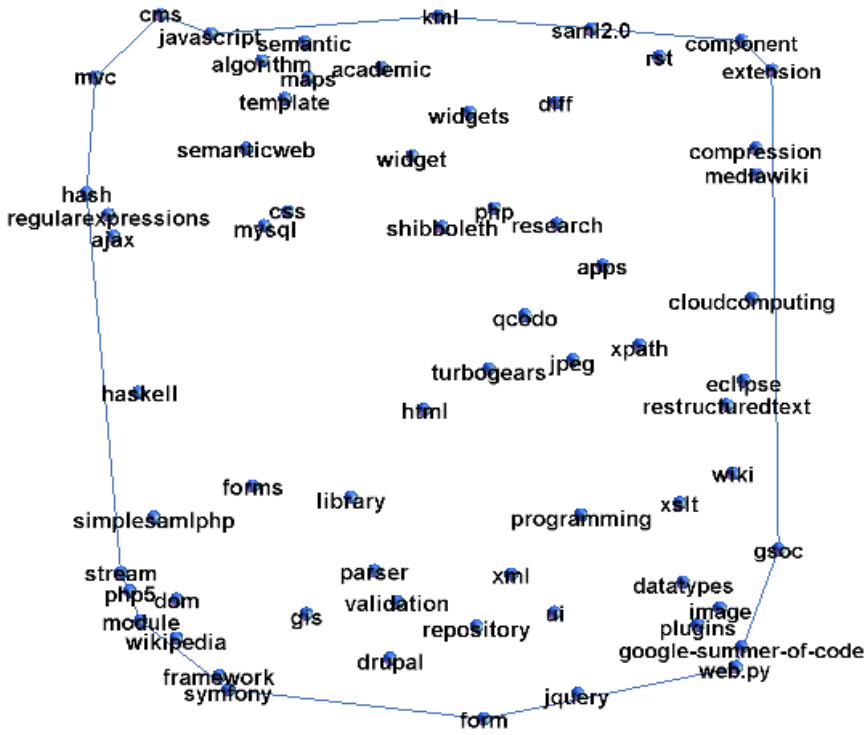
Σχήμα 4.3.3.4.1: Αναπαράσταση 10 ομάδων

Στο σχήμα 4.3.3.4.2 γίνεται μια αναπαράσταση των ομάδων ως κοινότητες του γράφου χωρίς να εμφανίζονται οι ακμές που συνδέουν κάθε κόμβο.

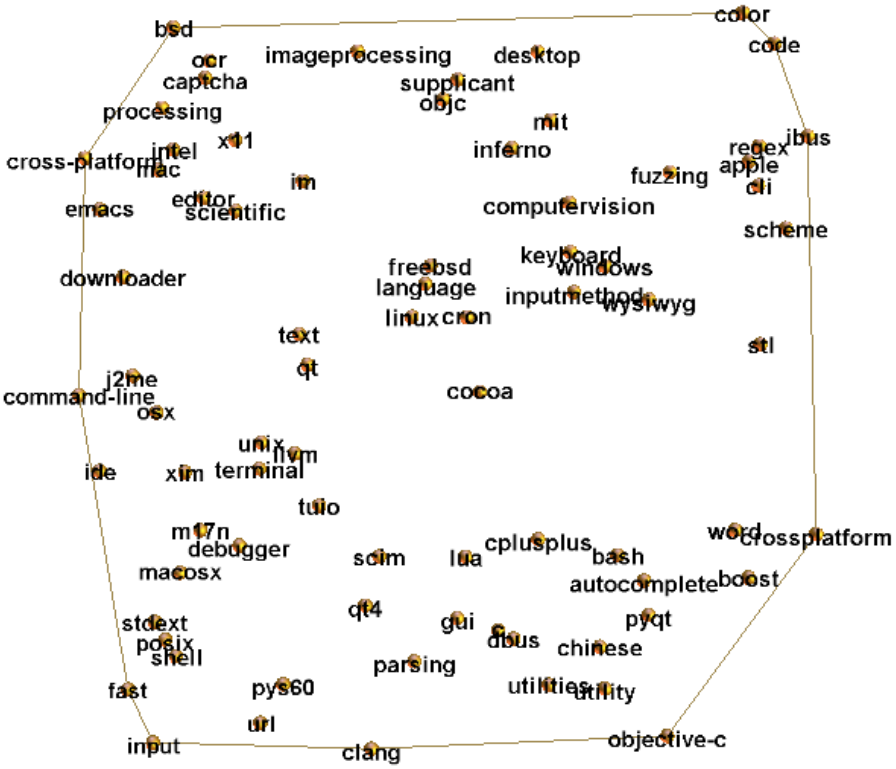


Σχήμα 4.3.3.4.2: Αναπαράσταση κοινοτήτων γράφου

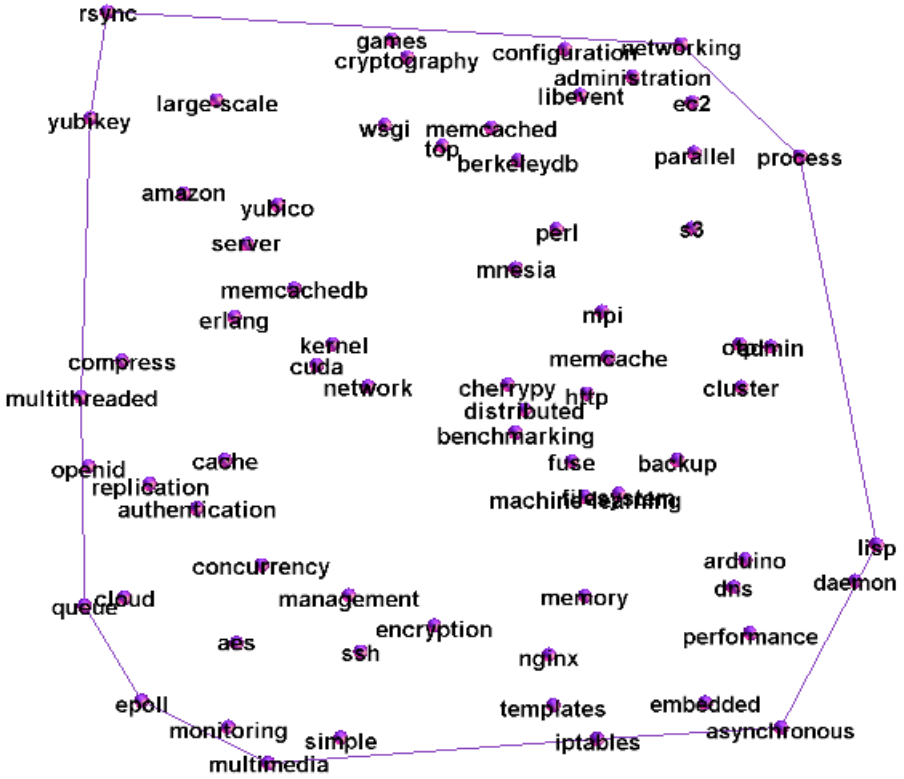
Στη συνέχεια, στα σχήματα 4.3.3.4.3-4.3.3.4.12 που ακολουθούν, θα παρουσιαστούν όλες οι ομάδες ξεχωριστά με όλες τις ετικέτες που τις αποτελούν όπως αυτές απεικονίζονται από το γραφικό περιβάλλον του Gephi.



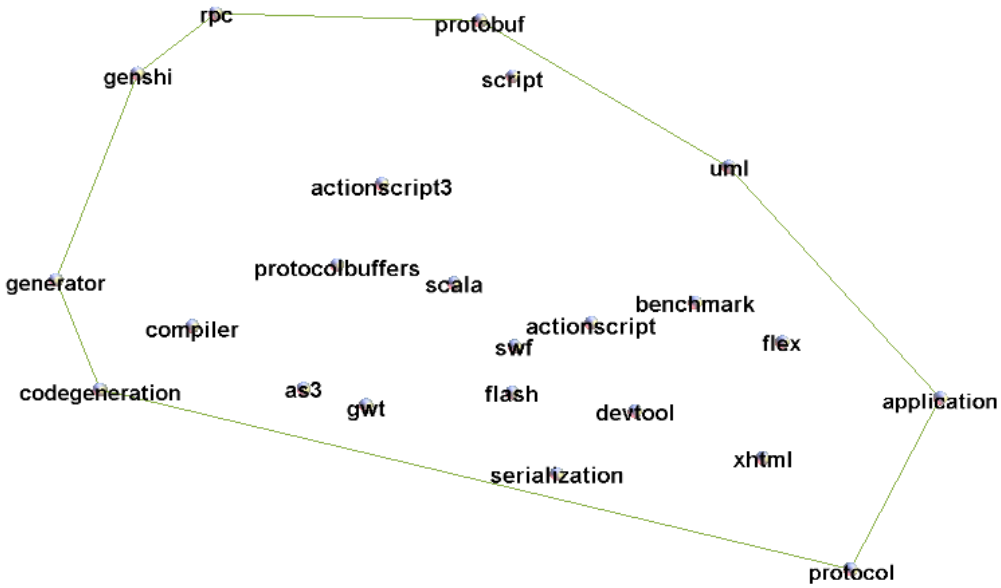
Σχήμα 4.3.3.4.3: Ομάδα 1



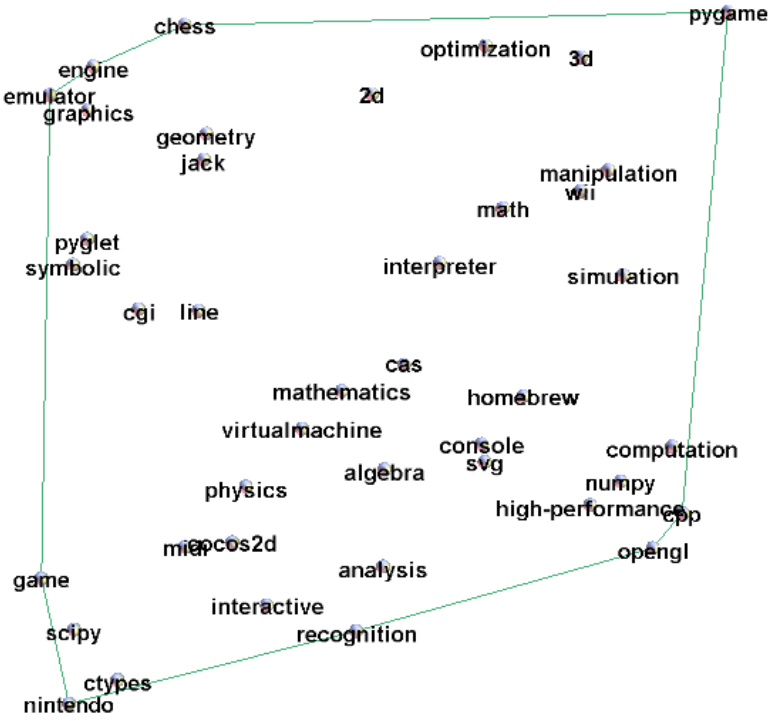
Σχήμα 4.3.3.4.4: Ομάδα 2



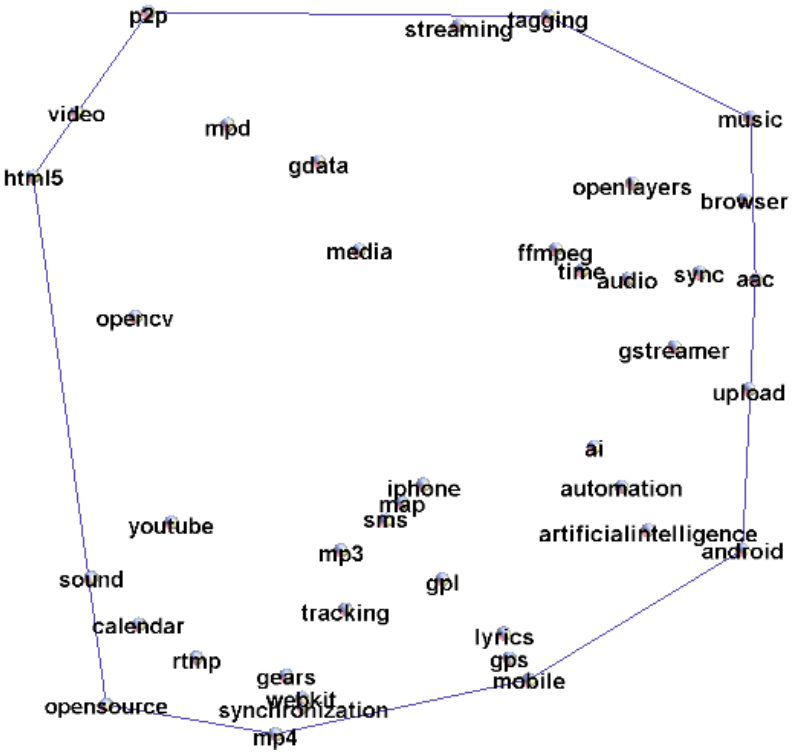
Σχήμα 4.3.3.4.5: Ομάδα 3



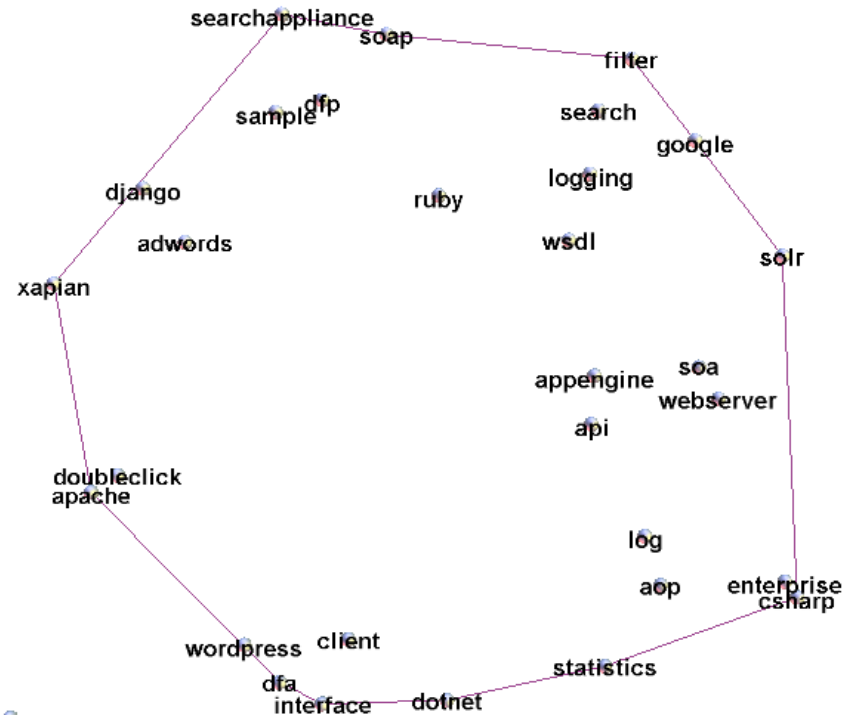
Σχήμα 4.3.3.4.6: Ομάδα 4



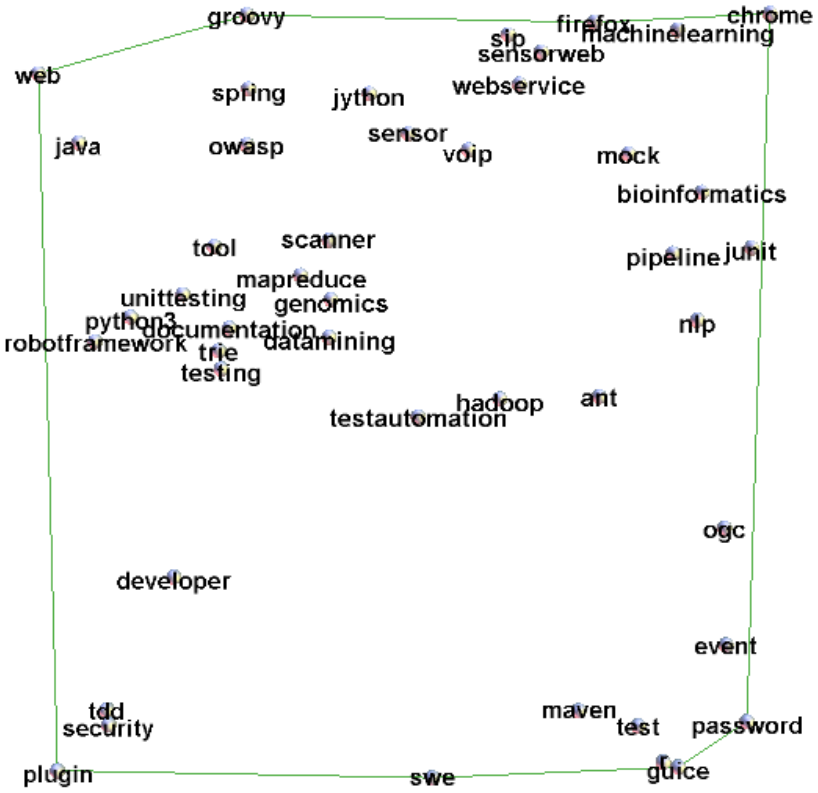
Σχήμα 4.3.3.4.7: Ομάδα 5



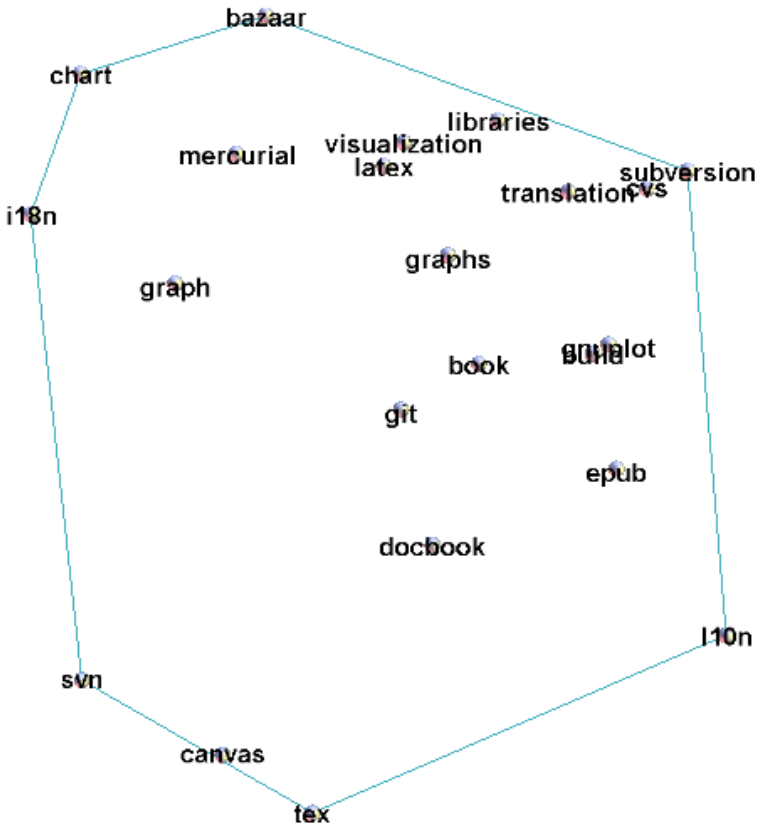
Σχήμα 4.3.3.4.8: Ομάδα 6



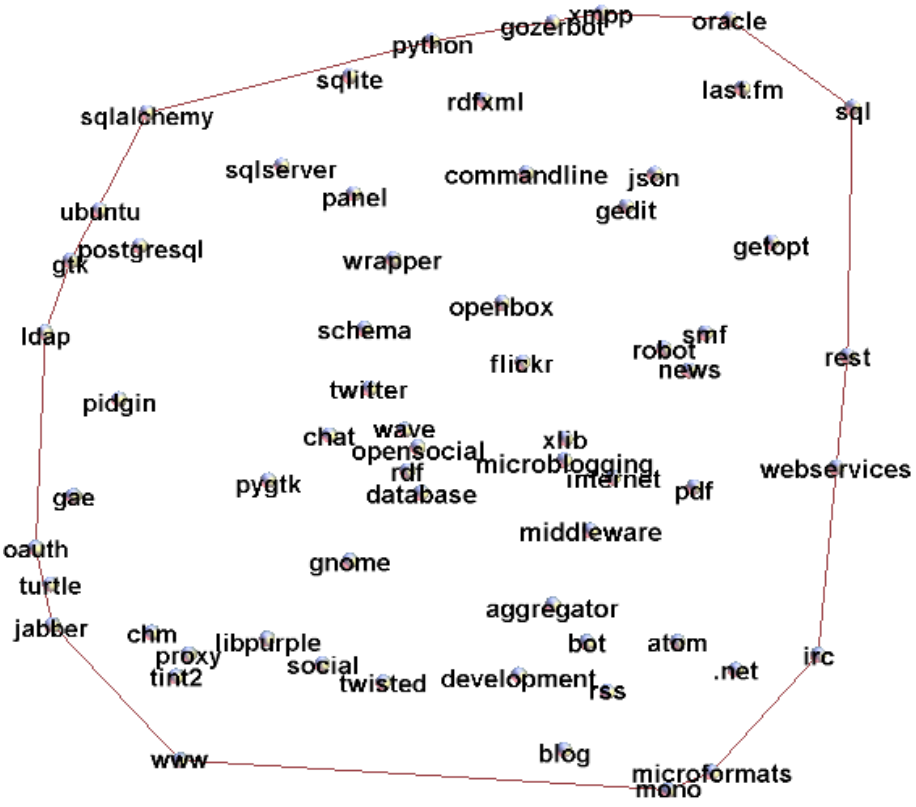
Σχήμα 4.3.3.4.9: Ομάδα 7



Σχήμα 4.3.3.4.10: Ομάδα 8



Σχήμα 4.3.3.4.11: Ομάδα 9



Σχήμα 4.3.3.4.12: Ομάδα 10

Σχολιασμός αποτελεσμάτων

Ο αλγόριθμος Louvain [10] δημιούργησε 10 ομάδες ετικετών. Κάθε ομάδα αποτελείτο από αρκετές ετικέτες και γι' αυτό το λόγο η ποιότητα των ομάδων δεν θεωρείται καλή. Αυτό ήταν φυσικό επακόλουθο για το λόγο ότι ο αλγόριθμος Louvain έχει υλοποιηθεί για την δημιουργία κοινοτήτων σε δίκτυα και στηρίζεται μόνο στις συνδέσεις των κόμβων. Ο γράφος που δημιουργήσαμε για την αναπαράσταση των ετικετών εμπεριείχε κόμβους-ετικέτες οι οποίες είχαν πάρα πολλές ακμές με άλλους κόμβους, δηλαδή ετικέτες που παρουσιάζονταν πολλές φορές στα ίδια αρχεία με άλλες ετικέτες (όπως η ετικέτα Python). Αυτό είχε ως αποτέλεσμα να είναι δύσκολο να εντοπιστούν κοινότητες οι οποίες να εμπεριέχουν ετικέτες πιο μεγάλης σύνδεσης.

4.4 Μετρικές αξιολόγησης ομαδοποίησης (Clustering Evaluation)

4.4.1 Θεωρητικό υπόβαθρο γύρω από το Clustering Evaluation

Η ομαδοποίηση όπως προαναφέραμε είναι μια διαδικασία η οποία πραγματοποιείται χωρίς επίβλεψη (unsupervised process). Υπάρχει ένα μεγάλο πλήθος από διαφορετικούς αλγορίθμους οι οποίοι υλοποιούν την διαδικασία της ομαδοποίησης και σε κάθε περίπτωση εξάγονται διαφορετικά αποτελέσματα. Επιπρόσθετα το γεγονός ότι σε αυτή τη διαδικασία δεν υπάρχουν προκαθορισμένες κλάσεις, έχει ως αποτέλεσμα να γίνεται πολύ δύσκολη η εύρεση μιας κατάλληλης μετρικής η οποία να αξιολογεί κατά πόσο η διαμόρφωση των ομάδων μπορεί να γίνει αποδεκτή. Για το λόγο αυτό έχει αναπτυχθεί μια πληθώρα από τεχνικές και μετρικές για την αξιολόγηση της διαδικασίας ομαδοποίησης αντικειμένων [14] [15].

Η διαδικασία εκτίμησης των αποτελεσμάτων ενός αλγόριθμου ομαδοποίησης ονομάζεται cluster validity assessment και υπάρχουν δύο κριτήρια μέτρησης για την εκτίμηση και την επιλογή του βέλτιστου αλγόριθμου για την πραγματοποίηση της ομαδοποίησης.

- **Compactness**: Κάθε μέλος μιας ομάδας πρέπει να είναι όσο το δυνατόν πιο κοντά με όλα τα υπόλοιπα μέλη της ομάδας.

- **Separation**: Κάθε ομάδα πρέπει να είναι όσο το δυνατόν πιο διαχωρισμένη από όλες τις υπόλοιπες ομάδες. Υπάρχουν τρεις προσεγγίσεις μέτρησης της απόστασης μεταξύ δύο διαφορετικών ομάδων, η απόσταση μεταξύ των πιο κοντινών μελών των δύο ομάδων, η απόσταση μεταξύ των πιο μακρινών μελών των δύο ομάδων, η απόσταση μεταξύ των κέντρων των δύο ομάδων.

Επιπρόσθετα υπάρχουν τρεις διαφορετικές τεχνικές για την εκτίμηση των αποτελεσμάτων που εξάγονται από κάθε αλγόριθμο ομαδοποίησης:

- **External criteria**: Τα αποτελέσματα της ομαδοποίησης αξιολογούνται με βάση κάποια στοιχεία τα οποία δεν χρησιμοποιούνται στην διαδικασία της ομαδοποίησης, για παράδειγμα κάποιες γνωστές ετικέτες κατηγοριών και κάποια εξωτερικά σημεία αναφοράς. Τα εν λόγω κριτήρια αποτελούνται από

ένα σύνολο προ-ταξινομημένων αντικειμένων και αυτά συνήθως δημιουργούνται από εμπειρογνώμονες. Έτσι αυτά τα κριτήρια μπορούν να θεωρηθούν ως ένα χρυσό οδηγό για την αξιολόγηση και με βάση αυτά μπορεί να αξιολογηθεί κατά πόσο τα αποτελέσματα μιας ομαδοποίησης είναι κοντά στις προκαθορισμένες κλάσεις.

- **Internal criteria**: Τα αποτελέσματα της ομαδοποίησης αξιολογούνται με βάση τα στοιχεία τα οποία οδήγησαν ένα αλγόριθμο στην συγκεκριμένη ομαδοποίηση. Αυτές οι μέθοδοι συνήθως αναθέτουν υψηλή βαθμολογία στον αλγόριθμο ο οποίος πέτυχε αποτελέσματα με την μεγαλύτερη ομοιότητα μέσα σε μια ομάδα και την χαμηλότερη ομοιότητα μεταξύ των άλλων περιφερειακών ομάδων.

- **Relative criteria**: Η βασική ιδέα αυτών των κριτηρίων είναι η σύγκριση των πολλών και διαφορετικών σχημάτων ομαδοποίησης. Υπάρχουν αρκετοί αλγόριθμοι ομαδοποίησης οι οποίοι εκτελούνται στο ίδιο σύνολο δεδομένων και παράγουν διαφορετικά αποτελέσματα. Έτσι στόχος αυτών των κριτηρίων είναι να επιλέξουν το καλύτερο σχήμα ομαδοποίηση με βάση τα αποτελέσματά του.

4.4.2 Μετρικές αξιολόγησης των αποτελεσμάτων ομαδοποίησης

4.4.2.1 Dunn and Dunn-like indices

Αυτή η μετρική εγκυρότητας προσπαθεί να προσδιορίσει «συμπαγή και καλά διαχωρισμένες ομάδες» [14] [15].

Ορίζεται από τον πιο κάτω τύπο για ένα συγκεκριμένο αριθμό ομάδων.

$$D_{nc} = \min_{i=1,\dots,nc} \left\{ \min_{j=i+1,\dots,nc} \left(\frac{d(c_i, c_j)}{\max_{k=1,\dots,nc} diam\,(c_k)} \right) \right\}$$

Όπου $d(c_i, c_j)$ είναι η συνάρτηση ανομοιογένειας (dissimilarity function) μεταξύ δύο ομάδων c_i και c_j και ορίζεται ως εξής:

$$d(c_i, c_j) = \min_{x \in c_i, y \in c_j} d(\mathbf{x}, \mathbf{y})$$

Ενώ $diam(c)$ είναι η διάμετρος μιας ομάδας, η οποία μπορεί να θεωρηθεί ως μια μετρική της διασποράς μιας ομάδας και ορίζεται ως εξής:

$$diam(C) = \max_{x,y \in C} d(x, y)$$

Η ιδέα αυτής της μετρικής είναι ότι αν το αποτέλεσμα μιας ομαδοποίησης εμπεριέχει συμπαγείς και καλά διαχωρισμένες ομάδες, τότε η απόσταση μεταξύ των ομάδων αναμένεται να είναι μεγάλη και η διάμετρος των ομάδων αναμένεται να είναι μικρή. Έτσι λοιπόν μπορούμε να συμπεράνουμε ότι μεγάλες τιμές σε αυτή την μετρική υποδεικνύει συμπαγές και καλά διαχωρισμένες ομάδες.

Το διάστημα τιμών για το υπό εξέταση σύνολο δεδομένων των ετικετών παρουσιάζεται στον πίνακα 4.4.2.1.1. Έχει γίνει η υπόθεση ότι δύο ετικέτες δεν αναπαρίστανται με το ίδιο διάνυσμα.

Min Value	Max Value
0,04558	21,81

Σχήμα 4.4.2.1.1: Διάστημα τιμών μετρικής Dunn and Dunn

4.4.2.2 Davies Bouldin index

Αυτή η μετρική βασίζεται σε μια μετρική ομοιότητας των ομάδων (R_{ij}) η οποία με την σειρά της βασίζεται στη διασπορά μιας ομάδας (s_i) και μια μετρική ανομοιογένειας (d_{ij}) [14] [15].

Συνήθως το R_{ij} ορίζεται ως εξής:

$$R_{ij} = \frac{s_i + s_j}{d_{ij}}$$

$$d_{ij} = d(v_i, v_j), \; s_i = \frac{1}{\|c_i\|} \sum_{x \in c_i} d(x, v_i)$$

Έτσι λοιπόν χρησιμοποιώντας την πιο πάνω μετρικής ομοιότητας η μετρική Davies-Bouldin ορίζεται ως εξής:

$$DB = \frac{1}{n_c} \sum_{i=1}^{n_c} R_i, \text{ where}$$

$$R_i = \max_{j=1...nc, i \neq j} (R_{ij}), \; i = 1...n_c$$

Συμπερασματικά λοιπόν όπως διαφαίνεται και από τον ορισμό, η μετρική DB είναι η μέση ομοιότητα μεταξύ κάθε ομάδας και της πιο όμοιας της.
Συνεπώς πολύ μικρό DB συνεπάγεται καλύτερη ομαδοποίηση.

Το διάστημα τιμών για το υπό εξέταση σύνολο δεδομένων των ετικετών παρουσιάζεται στον πίνακα 4.4.2.2.1. Έχει γίνει η υπόθεση ότι δύο ετικέτες δεν αναπαρίστανται με το ίδιο διάνυσμα.

Min Value	Max Value
0.0917	43.62

Πίνακας 4.4.2.2.1: Διάστημα τιμών μετρικής DB

4.4.3 Αξιολόγηση αποτελεσμάτων ομαδοποίησης με βάση τις μετρικές Dunn and Dunn και DB

Όπως προαναφέραμε, η διαδικασία της ομαδοποίησης είναι μια αρκετά πολύπλοκη διαδικασία και δεν μπορεί να αποφασιστεί με ακρίβεια και σιγουριά κατά πόσο μια ομαδοποίηση είναι σωστή ή όχι. Αυτό συμβαίνει για το λόγο ότι είναι μια διαδικασία χωρίς επίβλεψη στην οποία δεν γνωρίζουμε εκ των προτέρων τον αριθμό των επιθυμητών ομάδων. Για το λόγο αυτό έχο **ν** δημιουργηθεί αρκετές μετρικές αξιολόγησης των αποτελεσμάτων μιας ομαδοποίησης, οι οποίες με την σειρά τους δεν αποτελούν αυθεντία και δεν μπορούν να προσδιορίσουν με σιγουριά αν μια ομαδοποίηση είναι καλής ποιότητας η όχι.

Σε αυτό το υποκεφάλαιο θα γίνει μια αξιολόγηση και παράλληλα μια σύγκριση των αποτελεσμάτων που προέκυψαν από κάθε αλγόριθμο ομαδοποίησης με διαφορετικές παραμέτρους με βάση τις μετρικές Dunn and Dunn και DB.

4.4.3.1 Αξιολόγηση αποτελεσμάτων αλγορίθμου Simple-K-Means

Σε αυτό το σημείο γίνεται σύγκριση των αποτελεσμάτων που προέκυψαν από τον αλγόριθμο Simple-K-Means με βάση τις μετρικές DB και Dunn. Όπως προαναφέρθηκε τρέξαμε τον αλγόριθμο πολλές φορές με διαφορετικό επιθυμητό αριθμό των ομάδων. Λαμβάνοντας υπόψη των αριθμό των ετικετών τις οποίες θέλαμε να ομαδοποιήσουμε οι οποίες ήταν 476, θεωρήσαμε ότι ο μέγιστος αριθμός ομάδων που θα επιθυμούσαμε να δημιουργηθούν είναι 120 (δηλαδή 4 ετικέτες σε κάθε ομάδα). Περισσότερες ομάδες θα δημιουργούσαν πολύ μικρές ομάδες οι οποίες δεν θα έδινα κάποια σημαντική πληροφορία. Επιπρόσθετα παρατηρήθηκε ότι καθώς αυξάνονταν οι ομάδες, ο αλγόριθμος δημιουργούσε αρκετές ομάδες οι οποίες εμπεριείχαν μόνο μια ετικέτα. Αυτό ήταν αναμενόμενο αφού αρχικά ο K-Means κάνει μια τυχαία αρχικοποίηση των κέντρων των ομάδων. Από τα πειράματα που πραγματοποιήθηκαν με διαφορετικό αριθμό επιθυμητών ομάδων εξάχθηκαν οι τιμές που παρουσιάζονται στον πίνακα 4.4.3.1.1.

Number of clusters	DB	Dunn
10	1.3196	0.1684
20	1.4969	0.1458
30	1.5938	0.1458
40	1.4918	0.1458
50	1.4391	0.1706
60	1.3887	0.1740
70	1.7090	0.1428
80	1.6423	0.1428
90	1.5699	0.1428
100	1.5429	0.1428
110	1.4146	0.1889
120	1.3907	0.1889

Πίνακας 4.4.3.1.1: Αποτελέσματα Μετρικών για Simple-K-Means

Με βάση λοιπόν τις δύο μετρικές εντοπίστηκαν οι τρεις καλύτερες ομαδοποιήσεις των ετικετών οι οποίες παρουσιάζονται στον πίνακα 4.4.3.1.2.

A/A	Number of Clusters(DB)	Number of clusters(Dunn)
1	10(1.3196)	120(0.1889)
2	60(1.3887)	110(0.1889)
3	120(1.3907)	60(0.1740)

Πίνακας 4.4.3.1.2: Καλύτερα αποτελέσματα με βάση τις δύο μετρικές

Σχολιασμός Αποτελεσμάτων

Βλέποντας τα αποτελέσματα που προέκυψαν με βάση τις δύο μετρικές παρατηρούμε ότι κατέταξαν σε διαφορετική σειρά τα διαφορετικά πειράματα. Σύμφωνα με τον πίνακα 4.4.3.1.2 και οι δύο μετρικές κατάταξαν στην τριάδα των καλύτερων αποτελεσμάτων τα αποτελέσματα για αριθμό ομάδων 60 και 120 όμως σε διαφορετική σειρά. Ο τρόπος λειτουργίας του αλγορίθμου Simple-K-Means ο οποίος αρχικά κάνει μια τυχαία ανάθεση των κέντρων της κάθε ομάδας οδηγεί στην δημιουργία διαφορετικών ομάδων κάθε φορά, οι οποίες όμως έχουν και αρκετές ομοιότητες. Όσο μεγαλύτερος είναι ο αριθμός των ομάδων τόσο πιο μικρές είναι οι ομάδες σε μέγεθος που δημιουργούνται και συνεπώς έχο ιν μικρότερη διάμετρο. Για το λόγο αυτό στα πειράματα με μεγάλο αριθμό ομάδων η μετρική Dunn έδωσε σχετικά καλύτερες τιμές (μεγαλύτερες τιμές).

Λαμβάνοντας υπόψη τα αποτελέσματα των δύο μετρικών μπορούμε να θεωρήσουμε ότι επιτεύχθηκε καλύτερη ομαδοποίηση για αριθμό ομάδων 60 και 120. Λόγω του τρόπου λειτουργίας όμως του αλγόριθμου K-Means δεν μπορούμε να πούμε με σιγουριά κατά πόσο μια ομαδοποίηση είναι καλύτερη από την άλλη. Εξαρτάται από το σύνολο των δεδομένων στο οποίο γίνεται η ομαδοποίηση καθώς και για το που προορίζονται να χρησιμοποιηθούν τα αποτελέσματα.

4.4.3.2 Αξιολόγηση αποτελεσμάτων αλγορίθμου DBSCAN

Σε αυτό το σημείο γίνεται σύγκριση των αποτελεσμάτων που προέκυψαν από τον αλγόριθμο DBSCAN. Όπως προαναφέρθηκε τρέξαμε τον αλγόριθμο πολλές φορές με διαφορετικό e(epsilon). Επιλέχτηκαν τιμές στο epsilon από το διάστημα 1.2-3.0. Μικρότερες τιμές από το 1.2 έδιναν μόνο 3 ομάδες οι οποίες εμπεριείχαν ένα πολύ μικρό αριθμό ετικετών και συνεπώς δεν εμπεριείχαν σημαντική πληροφορία. Αυτό ήταν αναμενόμενο αφού η παράμετρος epsilon, όπως προαναφέραμε καθορίζει της ακτίνα της γειτονιάς στην οποία πρέπει να είναι εντός δύο σημεία για να ανατεθούν στην ίδια ομάδα. Αντίθετα, όπως μπορεί να παρατηρηθεί και στη συνέχεια πολύ μεγάλες τιμές στο epsilon δίνουν επίσης ένα μικρό αριθμό από ομάδες οι οποίες όμως εμπεριέχουν ένα μεγάλο αριθμό από ετικέτες και επομένως δεν γίνεται καλή ομαδοποίηση. Από τα πειράματα που πραγματοποιήθηκαν με διαφορετικό αριθμό επιθυμητών ομάδων εξάχθηκαν οι τιμές που παρουσιάζονται στον πίνακα 4.4.3.2.1.

e(epsilon)	Number of clusters	DB	Dunn
1.2	9	0.3980	1.4142
1.5	20	0.7845	0.7745
1.8	31	1.3388	0.6324
2.1	9	1.2599	0.5590
2.4	7	1.1724	0.5773
2.7	4	1.0957	0.5897
3.0	3	1.1050	0.5773

Πίνακας 4.4.3.2.1: Αποτελέσματα Μετρικών για DBSCAN

Με βάση λοιπόν τις δύο μετρικές εντοπίστηκαν οι τρεις καλύτερες ομαδοποιήσεις των ετικετών οι οποίες παρουσιάζονται στον πίνακα 4.4.3.2.2:

A/A	Number of Clusters(DB)	Number of clusters(Dunn)
1	1.2(0.3980)	1.2(1.4142)
2	1.5(0.7845)	1.5(0.7745)
3	2.7(1.0957)	1.8(0.6324)

Πίνακας 4.4.3.2.2: Καλύτερα αποτελέσματα με βάση τις δύο μετρικές

Σχολιασμός αποτελεσμάτων

Όπως μπορούμε να παρατηρήσουμε οι δύο μετρικές αξιολόγησης των αποτελεσμάτων ομαδοποίησης έδωσαν καλύτερα αποτελέσματα για μικρά epsilon. Αυτό ήταν αναμενόμενο αφού για μικρό epsilon δημιουργούνται ομάδες μεγαλύτερης συσχέτισης λόγω του ότι δύο σημεία μπορούν να μπουν στην ίδια ομάδα μόνο εάν βρίσκονται εντός αυτής της ακτίνας η οποία καθορίζεται από το epsilon.

Αφού εντοπίστηκε με βάση τα αποτελέσματα ότι το ιδανικό epsilon κυμαίνεται στο διάστημα 1 2 με 1 8 (και έγιναν πειράματα μόνο για τις τιμές 1.2, 1.5, 1.8) πραγματοποιήθηκαν επιπλέον πειράματα για να ελεγχθούν και οι υπόλοιπες τιμές της παραμέτρου σε αυτό το διάστημα (1.3, 1.4, 1.6, 1.7). Στον πίνακα 4.4.3.2.3 παρουσιάζονται τα αποτελέσματα με βάση τις δύο μετρικές.

e(epsilon)	Number of clusters	DB	Dunn
1.3	9	0.3980	1.4142
1.4	9	0.3980	1.4142
1.6	20	0.7845	0.7745
1.7	20	0.7845	0.7745

Πίνακας 4.4.3.2.3: Αποτελέσματα Μετρικών

Όπως μπορούμε να παρατηρήσουμε από τα αποτελέσματα του πίνακα 4.4.3.2.3 για epsilon 1.2-1.4 έχει πραγματοποιηθεί η ίδια ομαδοποίηση των ετικετών. Το ίδιο συμβαίνει και για τιμές 1.5-1.7. Στον αλγόριθμο DBSCAN όπως ήταν αναμενόμενο παρουσιάζεται καλύτερη ομαδοποίηση των ετικετών για μικρές τιμές της παραμέτρου epsilon και επιπρόσθετα τα αποτελέσματα παρουσιάζουν μια ομοιομορφία καθώς αλλάζει η συγκεκριμένη παράμετρος.

4.4.3.3 Αξιολόγηση αποτελεσμάτων Ιεραρχικού αλγορίθμου

Όπως έχουμε προαναφέρει, χρησιμοποιώντας τον Ιεραρχικό αλγόριθμο με διαφορετικό linkage type κάθε φορά παρατηρήθηκε ότι χρησιμοποιώντας centroid-based linkage εξάγονταν καλύτερα αποτελέσματα. Το γεγονός αυτό οφείλεται στο ότι πολλά από τα δεδομένα τα οποία θέλουμε να ομαδοποιήσουμε μπορεί να θεωρηθούν θορυβώδεις. Χρησιμοποιώντας centroid-based linkage η οποία είναι λιγότερο ευαίσθητη στο θόρυβο και στους outliers καταφέραμε να πάρουμε καλύτερα αποτελέσματα.

Ο ιεραρχικός αλγόριθμος όπως προαναφέραμε παράγει ένα δενδρόγραμμα. Έτσι πρέπει να καθοριστεί το επίπεδο στο οποίο θα γίνει ο καθορισμό των ομάδων. Το Weka καθορίζει αυτό το επίπεδο με βάση τον αριθμό τον ομάδων που επιθυμούμε να εξαχθούν από την ομαδοποίηση. Έτσι λοιπόν, τρέξαμε τον αλγόριθμο πολλές φορές με διαφορετικό αριθμό επιθυμητών ομάδων και χρησιμοποιώντας τις δύο μετρικές αξιολόγησης των αποτελεσμάτων πήραμε τα αποτελέσματα που παρουσιάζονται στον πίνακα 4.4.3.3.1. Όπως πράξαμε και στον αλγόριθμο Simple-K-Means και λαμβάνοντας υπόψη των αριθμό των ετικετών τις οποίες θέλαμε να ομαδοποιήσουμε οι οποίες ήταν 476, θεωρήσαμε ότι ο μέγιστος αριθμός ομάδων που θα επιθυμούσαμε να δημιουργηθούν είναι 120 (δηλαδή 4 ετικέτες σε κάθε ομάδα). Περισσότερες ομάδες θα δημιουργούσαν πολύ μικρές ομάδες οι οποίες δεν θα έδινα κάποια σημαντική πληροφορία.

Number of clusters	DB	Dunn
10	5.9411	0.1622
20	4.4257	0.1732
30	3.6216	0.1796
40	3.0906	0.2294
50	2.7556	0.2294
60	2.5499	0.2611
70	2.3266	0.2611
80	2.1438	0.2611
90	2.0115	0.3110
100	1.9098	0.3110
110	1.8101	0.3110
120	1.6950	0.3333

Πίνακας 4.4.3.3.1: Αποτελέσματα Μετρικών Ιεραρχικού αλγόριθμου

Με βάση λοιπόν τις δύο μετρικές εντοπίστηκαν οι τρεις καλύτερες ομαδοποιήσεις των ετικετών οι οποίες παρουσιάζονται στον πίνακα 4.4.3.3.2:

A/A	Number of Clusters(DB)	Number of clusters(Dunn)
1	120(1.6950)	110(0.3110)
2	110(1.8101)	100(0.3110)
3	100(1.9098)	90(0.3110)

Πίνακας 4.4.3.3.2: Καλύτερα αποτελέσματα με βάση τις δύο μετρικές

Σχολιασμός αποτελεσμάτων

Όπως ήταν αναμενόμενο αυξάνοντας τον αριθμό των ομάδων σε κάθε πείραμα βελτιώνονταν οι τιμές των δύο μετρικών αξιολόγησης των αποτελεσμάτων. Αυτό συμβαίνει για το λόγο ότι για να παραχθούν περισσότερες ομάδες στον ιεραρχικό αλγόριθμο, πρέπει να μετακινηθούμε προς τα φύλλα του δενδρογράμματος. Όμως όσο πιο κάτω μετακινούμαστε στο δενδρόγραμμα τόσο μικρότερες είναι οι αποστάσεις των κόμβων-ετικετών του δένδρο και συνεπώς παίρνουμε ομάδες ετικετών μεγαλύτερης συσχέτισης. Για το λόγο αυτό παρατηρείται αυτή η ομοιομορφία των τιμών που εξάγονται από τις δύο αυτές μετρικές. Περισσότερες ομάδες συνεπάγεται μικρότερες ομάδες ετικετών μεγαλύτερης συσχέτισης.

4.4.3.4 Αξιολόγηση αποτελεσμάτων Graph-Based ομαδοποίησης

Στον πίνακα 4.4.3.4.1 παρουσιάζονται οι τιμές των μετρικών οι οποίες προέκυψαν από τα αποτελέσματα του Graph Based Clustering χρησιμοποιώντας την μέθοδο Louvain.

Number of clusters	DB	Dunn
10	5.8926	0.0533

Πίνακας 4.4.3.4.1: Αποτελέσματα Μετρικών Graph-Based clustering

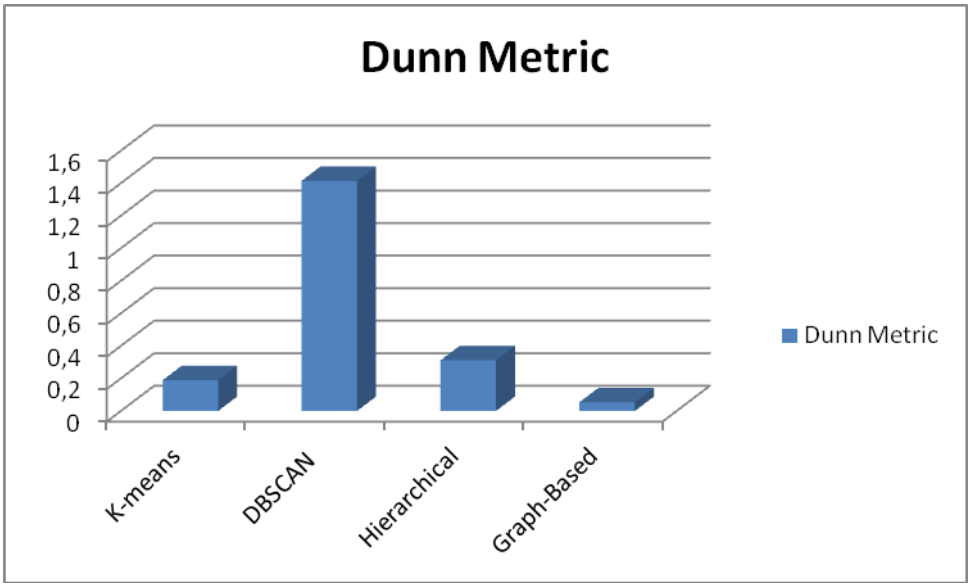
Σχολιασμός αποτελεσμάτων

Όπως μπορούμε να παρατηρήσο με οι τιμές των μετρικών αξιολόγησης των αποτελεσμάτων είναι κατά πολύ χειρότερες από τις τιμές που προέκυψαν από τα αποτελέσματα των άλλων αλγορίθμων ομαδοποίησης. Αυτό μπορεί να εξηγηθεί ξέροντας ότι η μέθοδος Louvain προορίζεται για τον εντοπισμό κοινοτήτων με βάση τις συνδέσεις ενός γράφου. Όπως προαναφέραμε υπάρχει ένας μεγάλος αριθμός ετικετών ο οποίος παρουσιάζεται σε πολλά αρχεία και ως αποτέλεσμα στο γράφο εμπεριέχει ακμές με πολλές άλλες ετικέτες. Αυτό κάνει δύσκολο τον εντοπισμό κοινοτήτων καλά διαχωρισμένων μεταξύ τους και συμπερασματικά η ομαδοποίηση που έγινε χρησιμοποιώντας αυτή την μέθοδο είναι χειρότερης ποιότητας.

4.4.3.5 Σύγκριση αποτελεσμάτων διαφορετικών αλγορίθμων ομαδοποίησης

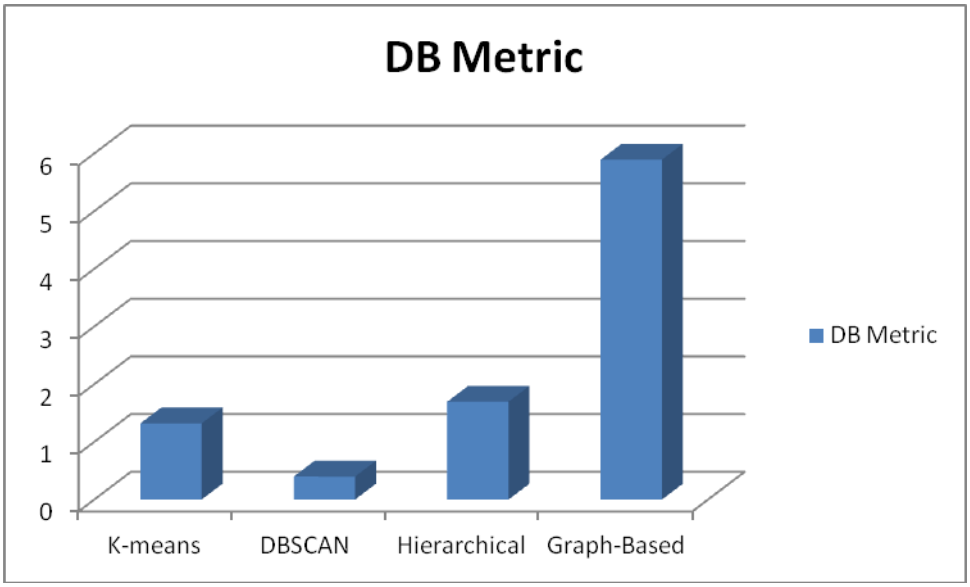
Σε αυτό το υποκεφάλαιο θα γίνει μια σύγκριση των τιμών που προέκυψαν από τις δύο μετρικές αξιολόγησης (Dunn, DB) για κάθε έναν από τους διαφορετικούς αλγορίθμους ομαδοποίησης. Για κάθε έναν αλγόριθμο επιλέγεται η καλύτερη τιμή ομαδοποίησης της κάθε μετρικής που προέκυψε μετά από τις πειραματικές δοκιμές που παρουσιάστηκαν στα προηγούμενα υποκεφάλαια.

Στο γράφημα 4.4.3.5.1 παρουσιάζονται οι τιμές που προέκυψαν για κάθε αλγόριθμο σύμφωνα με την μετρική Dunn and Dunn. Μεγαλύτερες τιμές συνεπάγεται καλύτερη ομαδοποίηση.



Γράφημα 4.4.3.5.1: Τιμές της Dunn μετρικής που προέκυψαν από κάθε αλγόριθμο ομαδοποίησης (μεγαλύτερες τιμές συνεπάγεται καλύτερη ομαδοποίηση)

Στο γράφημα 4.4.3.5.2 παρουσιάζονται οι τιμές που προέκυψαν για κάθε αλγόριθμο σύμφωνα με την μετρική DB. Μικρότερες τιμές συνεπάγεται καλύτερη ομαδοποίηση.



Γράφημα 4.4.3.5.2: Τιμές της DB μετρικής που προέκυψαν από κάθε αλγόριθμο ομαδοποίησης (μικρότερες τιμές συνεπάγεται καλύτερη ομαδοποίηση)

Ερμηνεία Αποτελεσμάτων

Παρατηρώντας το γράφημα 4.4.3.5.1 και το γράφημα 4.4.3.5.2 είναι εμφανές ότι ο αλγόριθμος DBSCAN πέτυχε καλύτερη ομαδοποίηση με βάση τις δύο μετρικές. Αυτό οφείλεται στο γεγονός ότι αυτός ο αλγόριθμος δεν ομαδοποίησε όλα τα δεδομένα. Ετικέτες τις οποίες θεώρησε θορυβώδεις δεδομένα δεν τις κατέταξε σε κάποια ομάδα. Από τους άλλους αλγορίθμους οι οποίοι ομαδοποίησαν όλες τις ετικέτες τα καλύτερα αποτελέσματα παρουσιάζει ο Ιεραρχικός Αλγόριθμος σύμφωνα με την μετρική Dunn, ενώ σύμφωνα με την μετρική DB ο K-means. Και οι δύο αλγόριθμοι όμως παρουσιάζουν παρόμοιες τιμές. Ο ιεραρχικός αλγόριθμος σε κάθε του βήμα βρίσκει την μικρότερη απόσταση μεταξύ δύο αντικειμένων και τα κατατάσσει στην ίδια ομάδα. Έτσι πετυχαίνει καλύτερης ποιότητας ομαδοποίηση. Ο αλγόριθμος K-means βρίσκει μια προσεγγιστική λύση του προβλήματος και αυτός είναι και ο λόγος που έχει μικρότερη χρονική πολυπλοκότητα. Αυτή η προσεγγιστική λύση οφείλεται σε μεγάλο βαθμό στις τυχαίες αρχικοποιήσεις των κέντρων κάθε ομάδας. Τέλος όπως ήταν φυσικό ο Graph Base αλγόριθμος παρουσίασε τα χειρότερα αποτελέσματα, εφόσον ήταν δύσκολο να εντοπιστούν μικρότερες και καλύτερης ποιότητας κοινότητες στο γράφο που δημιουργήσαμε και ως αποτέλεσμα να επιτευχθεί καλύτερη ομαδοποίηση.

Κεφάλαιο 5

5. Κανόνες Συσχέτισης (Association Rules)

5.1 Συχνά εμφανιζόμενα υποσύνολα (Frequent Itemsets)

Μια σημαντική πληροφορία την οποία μπορούμε να εξάγουμε από το σύνολο των ετικετών που συλλέξαμε και η οποία παρουσιάζει ενδιαφέρον είναι η εύρεση των σημαντικότερων ζευγαριών(2-sets) και τριπλέττων (3-sets). Δηλαδή ποιες ετικέτες εμφανίζονται στα ίδια αρχεία λογισμικού με μεγαλύτερη συχνότητα. Στον πίνακα 5.1.1 εμφανίζονται τα 10 πιο δημοφιλή ζευγάρια, ενώ στον πίνακα 5.1.2 οι πιο δημοφιλείς τριπλέττες.

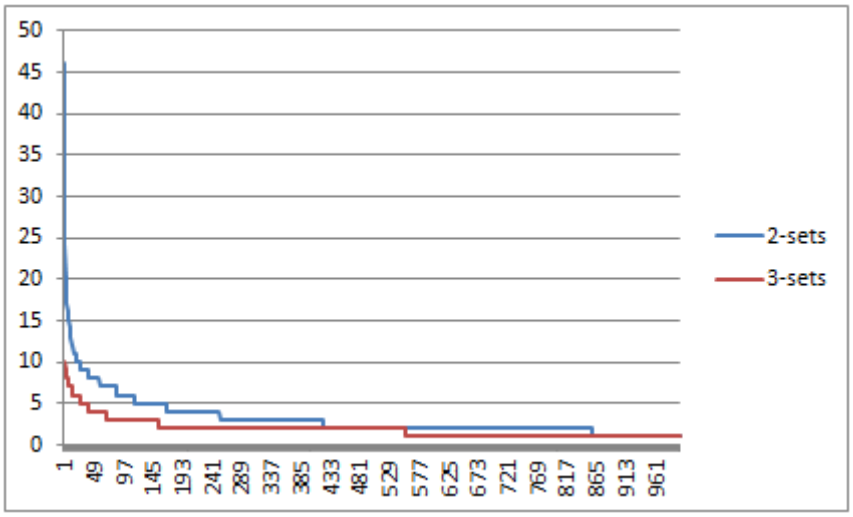
A/A	2-sets	Συχνότητα εμφάνισης
1	python django	38
2	linux windows	16
3	google api	14
4	python web	13
5	python linux	11
6	python java	11
7	api soap	11
8	python google	10
9	google soap	10
10	api wsdl	10

Πίνακας 5.1.1: Τα 10 δημοφιλέστερα ζευγάρια ετικετών

A/A	3-sets	Συχνότητα εμφάνισης
1	linux windows osx	10
2	api soap wsdl	10
3	google api soap	9
4	google api wsdl	8
5	google soap wsdl	8
6	otp yubico yubikey	8
7	python web django	7
8	google adwords api	7
9	google api sample	7
10	api sample soap	7

Πίνακας 5.1.2: Οι 10 δημοφιλέστερες τριπλέττες ετικετών

Στο γράφημα 5.1.3 παρουσιάζονται τα 1000 δημοφιλέστερα ζευγάρια και τριπλέττες ετικετών. Όπως ήταν αναμενόμενο παρατηρείται να υπάρχουν περισσότερα και πιο συχνά εμφανιζόμενα ζευγάρια σε σύγκριση με τις τριπλέττες.



Γράφημα 5.1.3: Τα 1000 δημοφιλέστερα ζευγάρια και τριπλέττες ετικετών ως προς τον αριθμό εμφάνισής τους στα αρχεία λογισμικού

5.2 Θεωρητικό υπόβαθρο γύρω από τους Κανόνες Συσχέτισης

Κανόνας Συσχέτισης(Association rule): είναι ένας κανόνας συμπερασμού της μορφής X → Y, όπου X και Y είναι δύο πλήρως διαχωρισμένα σύνολα αντικειμένων όπου X ⋂ Y = 0 . Η σημαντικότητα ενός κανόνα συσχέτισης μπορεί να μετρηθεί με βάση το support και το confidence του.

Support: ορίζει πόσο συχνά ένας κανόνας εμφανίζεται σε ένα σύνολο δεδομένων. Είναι μια σημαντική μετρική για το λόγο ότι ένας κανόνας με χαμηλό support μπορεί να εμφανίζεται εντελώς τυχαία. Επιπρόσθετα τέτοιοι κανόνες είναι πολύ πιθανόν να μην έχουν σημαντικό ενδιαφέρον λόγω της χαμηλής τους εμφάνισης σε ολόκληρο το σύνολο δεδομένων.

$$\text{Support, } s(X \longrightarrow Y) \quad = \quad \frac{\sigma(X \cup Y)}{N}$$

Confidence: ορίζει πόσο συχνά τα αντικείμενα του συνόλου Y εμφανίζονται στο ίδιο σύνολο με τα αντικείμενα του συνόλου X. Ορίζει ουσιαστικά πόσο αξιόπιστος είναι ένας κανόνας. Δοθέντος ενός κανόνα X → Y το μεγαλύτερο confidence θα παρουσιαστεί στην περίπτωση που το Y εμφανίζεται σε κάθε συναλλαγή στην οποία εμφανίζεται και το σύνολο X. Επιπρόσθετα το confidence παρέχει μια εκτίμησης της πιθανότητας να παρουσιαστεί σε μια συναλλαγή το υποσύνολο Y εάν παρουσιάζεται το υποσύνολο X.

$$\text{Confidence, } c(X \longrightarrow Y) \quad = \quad \frac{\sigma(X \cup Y)}{\sigma(X)}$$

5.3 Θεωρητικό Υπόβαθρο των κύριων αλγορίθμων

Οι περισσότεροι αλγόριθμοι που έχουν δημιουργηθεί για την εξαγωγή κανόνων συσχέτισης βασίζονται σε μια κοινή στρατηγική στην οποία διασπούν το πρόβλημα σε δύο βασικές υπό-διεργασίες [16]:

1. **Frequent Itemset Generation**: που σκοπός της είναι η εξεύρεση όλων των υποσυνόλων αντικειμένων που ικανοποιούν το κατώφλι του ελάχιστου support. Δηλαδή έχουν support μεγαλύτερο από το προκαθορισμένο ελάχιστο support. Αυτά τα σύνολα ονομάζονται συχνά υποσύνολα (frequent itemsets).

2. **Rule Generation:** που σκοπός του είναι η εξαγωγή των κανόνων με υψηλό confidence που προέκυψαν από τα frequent itemsets που εξάχθηκαν από το προηγούμενο βήμα. Αυτοί οι κανόνες ονομάζονται strong rules.

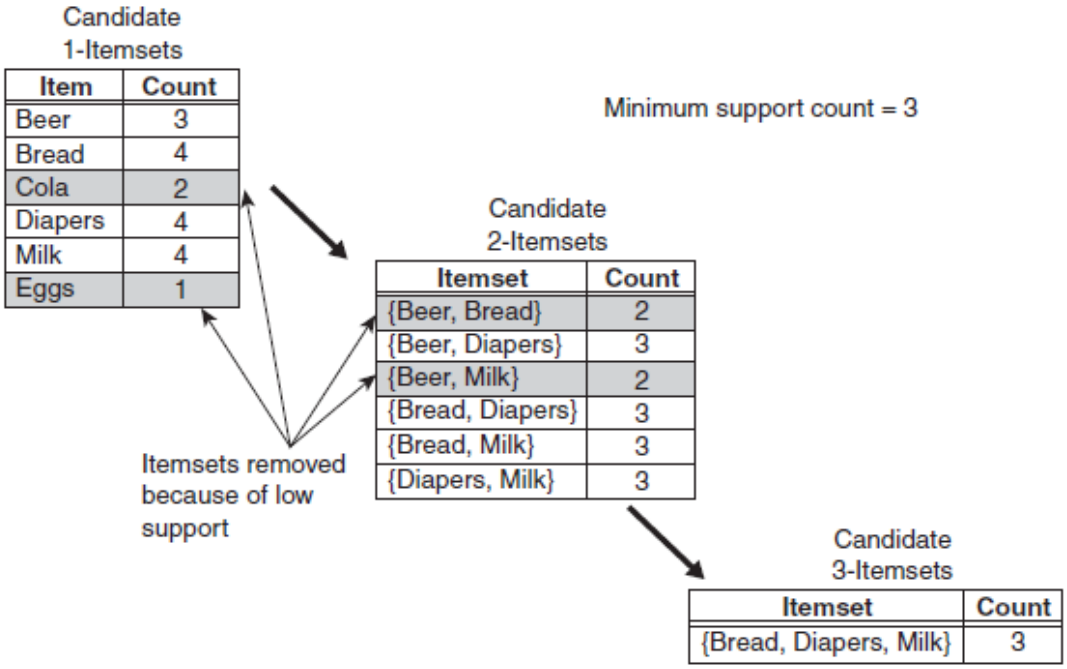
Δύο από τους βασικότερους αλγορίθμους παραγωγής κανόνων συσχέτισης είναι ο Apriori και ο FP-Growth οι οποία παρουσιάζονται στη συνέχεια.

5.3.1 Apriori

Ο Apriori βασίζεται στην ιδέα ότι εάν ένα σύνολο I αντικειμένων έχει support μεγαλύτερο από το κατώφλι του ελάχιστου επιθυμητού support τότε, κάθε υποσύνολό J του I έχει επίσης support μεγαλύτερο από το κατώφλι.
Διαχωρίζει το πρόβλημα σε δύο βασικές διεργασίες:

Παραγωγή συχνά εμφανιζόμενων υποσυνόλων
(Frequent Itemset Generation of the Apriori Algorithm):

Η διαδικασία του εντοπισμού των συχνά εμφανιζόμενων υποσυνόλων(frequent itemsets) ξεκινά υπολογίζοντας το support του κάθε ξεχωριστού αντικειμένου από το σύνολο αντικειμένων. Στη συνέχεια τα αντικείμενα τα οποία έχουν support κάτω από το κατώφλι αποκόπτονται. Στη συνέχεια μέσω μιας επαναληπτικής διαδικασίας και χρησιμοποιώντας τα υποσύνολα μεγέθους k τα οποία θεωρήθηκαν ως frequent itemsets υπολογίζουμε τα itemsets μεγέθους k+1. Ακολούθως υπολογίζουμε το support των νέων συνόλων και αποκόπτουμε αυτά που θεωρήθηκαν non frequent itemsets. Στο σχήμα 5.3.1.1 παρουσιάζεται ένα παράδειγμα της διαδικασία αυτής



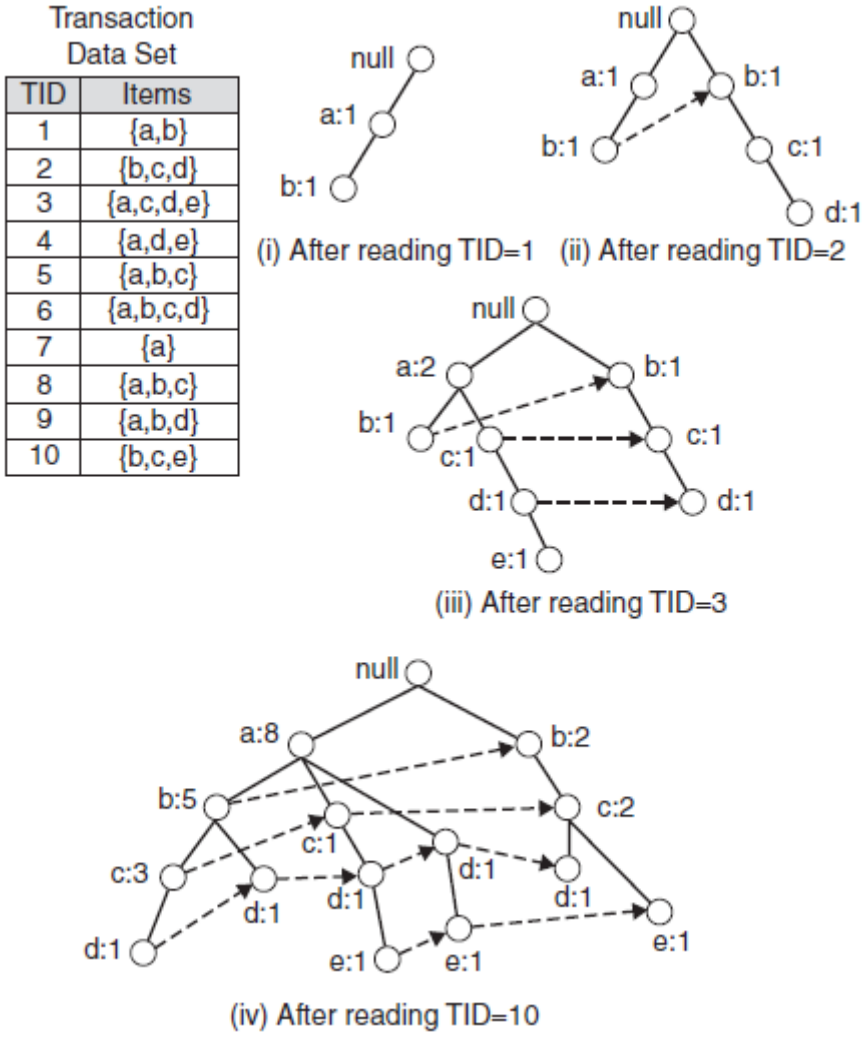
Πίνακας 5.3.1.1: Παράδειγμα εντοπισμού των συχνά εμφανιζόμενων υποσυνόλων με ελάχιστο support =3

Παραγωγή κανόνων (Rule Generation):

Αυτή η διαδικασία είναι υπεύθυνη για τον εντοπισμό των κανόνων συσχέτισης. Για κάθε frequent item-set μεγέθους k, μπορούν να παραχθούν 2^k - 2 κανόνες συσχέτισης. Όλοι οι κανόνες που παράγονται έχουν σίγουρα support μεγαλύτερο του κατωφλίου αφού προέρχονται από frequent item-sets. Γνωρίζοντας το support κάθε item-set, το οποίο έχει υπολογιστεί στην προηγούμενη φάση μπορούμε να υπολογίσουμε το confidence κάθε κανόνα και να εξετάσουμε κατά πόσο αυτός ο κανόνας μπορεί να θεωρηθεί ισχυρός (strong rule) [16].

5.3.2 FP-Growth

Ο αλγόριθμος FP-growth κωδικοποιεί το σύνολο δεδομένων χρησιμοποιώντας μια δομή δεδομένων που ονομάζεται FP-tree και εξάγει τα frequent item-sets από αυτή την δομή. Αρχικά διαβάζεται με ένα πέρασμα το σύνολο δεδομένων και για κάθε συναλλαγή (basket) δημιουργείτε ένα μονοπάτι στο FP-tree. Αν κάποιες συναλλαγές έχουν κοινά αντικείμενα είναι πολύ πιθανόν τα μονοπάτια με τα οποία αναπαριστούνται στο δέντρο να επικαλύπτονται. Κάθε κόμβος αναπαριστά ένα αντικείμενο και επιπλέον εμπεριέχει έναν μετρητή ο οποίος αντιπροσωπεύει τον αριθμό των συναλλαγών που αντιπροσωπεύονται από τον συγκεκριμένο κόμβο. Στο σχήμα 5.3.2.1 παρουσιάζεται ένα παράδειγμα αυτής της διαδικασίας.



Πίνακας 5.3.2.1: Παράδειγμα κωδικοποίησης του συνόλου δεδομένων στο FP-tree

Για την παραγωγή των frequent item-sets γίνεται μια bottom-up εξερεύνηση του FP-tree. Συγκεκριμένα το πρόβλημα διασπάται σε μικρότερα υπό-προβλήματα βρίσκοντας κάθε φορά τα frequent item-sets τα οποία τελειώνουν με ένα συγκεκριμένο πρόθεμα. Έτσι το αρχικό δέντρο διασπάται σε μικρότερα υπό-δέντρα σύμφωνα με ποιο πρόθεμα που έχουν στα φύλλα τους. Σύμφωνα λοιπόν με τον μετρητή του κάθε κόμβου το οποίο υποδηλώνει το support του συγκεκριμένου μονοπατιού εξάγονται τα frequent item-sets. Πριν όμως επιτευχθεί αυτή η διαδικασία αφαιρούνται από το υπό-δέντρο τα μονοπάτια που δεν εμπεριέχουν το υπό-εξέταση πρόθεμα μειώνοντας παράλληλα τον μετρητή των κόμβων που επηρεάζονται από αυτό το κλάδεμα των μονοπατιών (το δέντρο που δημιουργείται από αυτή την διαδικασία ονομάζεται conditional FP-tree) [16].

5.4 Αποτελέσματα κανόνων συσχέτισης

Στη συνέχεια παρουσιάζεται ένα δείγμα των αποτελεσμάτων που προέκυψαν από τους δύο αλγόριθμους παραγωγής κανόνων συσχέτισης.

5.4.1 Αποτελέσματα Apriori

Τρέξαμε τον αλγόριθμο Apriori αρκετές φορές. Αυτός ο αλγόριθμος παράγει και κανόνες οι οποίοι είναι της μορφής tag1=0→ tag2=0. Το 0 για κάθε ετικέτα υποδεικνύει ότι η συγκεκριμένη ετικέτα δεν παρουσιάζεται. Δηλαδή όταν το tag1 δεν παρουσιάζεται, αυτό συνεπάγεται ότι δεν θα παρουσιάζεται ούτε το tag2. Για να αποφύγουμε την πιο πάνω περίπτωση κανόνων μειώσαμε το μέγιστο support το οποίο επιθυμούμε να έχουν οι κανόνες που θα εξαχθούν. Έτσι λοιπόν καταλήξαμε στις παραμέτρους που παρουσιάζει ο πίνακας 5.4.1.1.

Lower Bound Min Support Value	0.01
Upper Bound Min Support Value	0.63
Delta	0.01
Min Confidence Value	0.1

Πίνακας 5.4.1.1: Παράμετροι που χρησιμοποιήθηκαν για τον Apriori

Στον πίνακα 5.4.1.2 παρουσιάζεται ένα πολύ μικρό δείγμα των κανόνων συσχέτισης που εντόπισε ο Apriori με Confidence ίσο με 1. Οι έντονοι αριθμοί αντιπροσωπεύουν το support κάθε υποσυνόλου των ετικετών.

A/A	Κανόνες
1	wsdl=1 api=1 **10** ==> soap=1 **10** conf:(1)
2	soap=1 wsdl=1 **10** ==> api=1 **10** conf:(1)
3	windows=1 osx=1 **10** ==> linux=1 **10** conf:(1)
4	sample=1 **9** ==> api=1 **9** conf:(1)
5	numpy=1 **8** ==> python=1 **8** conf:(1)
6	twisted=1 **8** ==> python=1 **8** conf:(1)
7	wsgi=1 **8** ==> python=1 **8** conf:(1)
8	wsdl=1 google=1 **8** ==> soap=1 **8** conf:(1)
9	yubico=1 otp=1 **8** ==> yubikey=1 **8** conf:(1)
10	wsdl=1 google=1 **8** ==> api=1 **8** conf:(1)

Πίνακας 5.4.1.2: Δείγμα αποτελεσμάτων Apriori

5.4.2 Αποτελέσματα FP-Growth

Τρέξαμε τον αλγόριθμος FPgrowth με τις ίδιες παραμέτρους που είχαμε τρέξει και τον Apriori και οι οποίες παρουσιάζονται στον πίνακα 5.4.2.1.

Lower Bound Min Support Value	0.01
Upper Bound Min Support Value	0.63
Delta	0.01
Min Confidence Value	0.1

Πίνακας 5.4.2.1: Παράμετροι που χρησιμοποιήθηκαν για τον FP-Growth

Στον πίνακα 5.4.2.2 παρουσιάζεται ένα πολύ μικρό δείγμα των κανόνων που προέκυψαν από τον FP-Growth με Confidence ίσο με 1. Οι έντονοι αριθμοί αντιπροσωπεύουν το support κάθε υποσυνόλου των ετικετών. Εκτός από το confidence κάθε κανόνα ο αλγόριθμος FPgrowth υπολογίζει και κάποιες άλλες μετρικές. Συγκεκριμένα:

Lift: Ορίζεται ως η αναλογία του παρατηρούμενου Support ενός κανόνα προς το αναμενόμενο Support του κανόνα και υπολογίζεται ως εξής:

$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(Y) \times \text{supp}(X)}$$

Στην περίπτωση που η τιμή του Lift ενός κανόνα είναι μεγαλύτερη της μονάδας, συνεπάγεται ότι η εμφάνιση του X έχει θετική επίδραση στην εμφάνιση του Y. Στην περίπτωση που η τιμή του Lift ενός κανόνα είναι μικρότερη της μονάδας ισχύει το αντίστροφο.

Conviction: Ορίζεται ως η αναλογία της αναμενόμενης συχνότητας που εμφανίζεται το αντικείμενο X χωρίς το Y. Δηλαδή υπολογίζει την συχνότητα στην οποία ο κανόνας μπορεί να κάνει λάθος πρόβλεψη.

$$\text{conv}(X \Rightarrow Y) = \frac{1 - \text{supp}(Y)}{1 - \text{conf}(X \Rightarrow Y)}$$

A/A	Κανόνες
1	[wsgi=1]: **8** ==> [python=1]: **8** <conf:(1)> lift:(2.9) conv:(5.24)
2	[twisted=1]: **8** ==> [python=1]: **8** <conf:(1)> lift:(2.9) conv:(5.24)
3	[numpy=1]: **8** ==> [python=1]: **8** <conf:(1)> lift:(2.9) conv:(5.24)
4	[scipy=1]: **7** ==> [python=1]: **7** <conf:(1)> lift:(2.9) conv:(4.58)

5	[pyqt=1]: **7** ==> [python=1]: **7** <conf:(1)> lift:(2.9) conv:(4.58)
6	[sqlite=1]: **5** ==> [python=1]: **5** <conf:(1)> lift:(2.9) conv:(3.27)
7	[jython=1]: **4** ==> [python=1]: **4** <conf:(1)> lift:(2.9) conv:(2.62)
8	[bot=1]: **4** ==> [python=1]: **4** <conf:(1)> lift:(2.9) conv:(2.62)
9	[symbolic=1]: **3** ==> [python=1]: **3** <conf:(1)> lift:(2.9) conv:(1.96)
10	[python3=1]: **3** ==> [python=1]: **3** <conf:(1)> lift:(2.9) conv:(1.96)

Πίνακας 5.4.2.2: Δείγμα αποτελεσμάτων FP-Growth

5.5 Σύγκριση αλγορίθμων Apriori και FP-Growth

Ο αλγόριθμος Apriori εντόπισε 4629 κανόνες συσχέτισης. Αντίστοιχα ο αλγόριθμος FPgrowth εντόπισε 4603 κανόνες με τις ίδιες παραμέτρους. Στον πίνακα 5.5.1 παρουσιάζεται ο αριθμός των κανόνων που εντοπίστηκαν για διαφορετικά διαστήματα τιμών της μετρικής Confidence.

Confidence Value	**Αριθμός κανόνων**	
	Apriori	**FPGrowth**
1	1383	1383
0.8-0.99	181	181
0.5-0.79	1413	1413
0.1-0.49	1652	1626

Πίνακας 5.5.1: Κανόνες που εντοπίστηκαν σε διαφορετικά διαστήματα Confidence

Οι δύο αλγόριθμοι πέτυχαν σχεδόν τα ίδια αποτελέσματα, όπως ήταν αναμενόμενο. Εκτός από της σύγκριση που έγινε στα αποτελέσματα των δύο αλγορίθμων έγινε σύγκριση και στο χρόνο ολοκλήρωσης του κάθε αλγορίθμου καθώς και στο ποσοστό της κύριας μνήμης που κατανάλωσε ο κάθε αλγόριθμος για τον υπολογισμό των κανόνων συσχέτισης. Τα αποτελέσματα παρουσιάζονται στον πίνακα 5.5.2.

Αλγόριθμος	**Χρόνος Ολοκλήρωσης (sec)**	**Κατανάλωση Μνήμης (mb)**
Apriori	292.35	269.156
FPGrowth	8.23	30.404

Πίνακας 5.5.2: Χρόνος ολοκλήρωσης και Κατανάλωση μνήμης για κάθε αλγόριθμο

Σχολιασμός αποτελεσμάτων

Όπως μπορούμε να παρατηρήσουμε ο αλγόριθμος FP-Growth παρουσίασε μικρότερες ανάγκες για κύρια μνήμη και ολοκλήρωσε σε μικρότερο χρονικό διάστημα. Όσον αφορά την κύρια μνήμη, ο FP-Growth αναπαριστά κάθε συναλλαγή (basket) σε ένα FP-Tree και αν δύο συναλλαγές αλληλεπικαλύπτονται σε κάποιο σημείο αντικειμένων, τότε αυτό το μέρος θα αναπαρασταθεί μόνο μια φορά στο FP-Tree. Έτσι μειώνονται οι επαναλήψεις και ως αποτέλεσμα μειώνονται οι απαιτήσεις για κύρια μνήμη. Επιπρόσθετα, ο FP-Growth διαβάζει μόνο μια φορά το σύνολο των συναλλαγών για την δημιουργία του FP-Tree. Αντίθετα, ο Apriori κατά το πρώτο πέρασμα διαβάζει όλο το σύνολο των συναλλαγών και υπολογίζει το support του κάθε αντικειμένου και στη συνέχεια κατά το δεύτερο πέρασμα ξαναδιαβάζει της συναλλαγές και κρατά στην κύρια μνήμη μόνο τα ζευγάρια που αποτελούνται από δύο frequent-items. Αυτή η διαδικασία έχει ως αποτέλεσμα να παρουσιάζει χειρότερους χρόνου εκτέλεσης από τον αλγόριθμο FP-Growth.

Κεφάλαιο 6

6. Κατηγορίες ετικετών

6.1 Περιγραφή διαδικασίας

Στόχος του διαδικασίας ήταν να εντοπιστεί τι είδους ετικέτες περιγράφουν τα αρχεία λογισμικού που συλλέχθηκαν από το Google Code. Συγκεκριμένα ποιες είναι οι πιο συνήθεις κατηγορίες ετικετών οι οποίες εμπεριέχονται ως ετικέτες στα αρχεία λογισμικού. Για παράδειγμα η ετικέτα python ανήκει στην κατηγορία "Γλώσσα Προγραμματισμού".

Από τις 3237 ετικέτες οι οποία εντοπίστηκαν στα αρχεία λογισμικού που συλλέχθηκαν από το Google code, επιλέχθηκαν για την διαδικασία οι 476 πιο συχνά εμφανιζόμενες ετικέτες οι οποίες έχουν συχνότητα πέραν των δύο εμφανίσεων. Για κάθε μια από αυτές τις ετικέτες αποφασίστηκε εμπειρικά σε ποια κατηγορία ανήκει. Για την λήψη αυτής της απόφασης και εφόσον αρκετές από τις ετικέτες ήταν άγνωστες χρειάστηκε να γίνει μια προεργασία για να γίνει γνωστό τι αντιπροσωπεύει η κάθε ετικέτα.

Στην συνέχεια και αφού κατηγοριοποιήθηκαν όλα οι ετικέτες υπολογίστηκε το ποσοστό στο οποίο εμφανίζεται κάθε κατηγορία ετικετών σε όλο το σύνολο των αρχείων λογισμικού.

6.2 Κατηγορίες που εντοπίστηκαν

Ανατρέχοντας στο σύνολο των 476 ετικετών εντοπίστηκαν 16 κατηγορίες οι οποίες παρουσιάζονται στον πίνακα 6.2.1. Στον πίνακα παρουσιάζεται επίσης ο αριθμός των διαφορετικών ετικετών που εμπεριέχονται σε κάθε κατηγορία, καθώς και ένα μικρό δείγμα των ετικετών κάθε κατηγορίας.

Όνομα Κατηγορίας	**Αριθμός των ετικετών**	**Δείγμα ετικετών της κατηγορίας**
Operating System	9	Windows Unix Linux Ubuntu Android
Programming Language	40	Python C Java Objective-c Csharp
Library	15	Jquery Libevent XLib Stl M17n
Web Browser	3	Google Chrome Firefox
Database Management System	5	MySQL SQLServer Oracle Sqlite Postgresql
Text Editor	5	Gedit Emacs Latex
Protocols	11	Sip Http Rtmp Xmpp Idap
General Description	109	Music Game Media Time Book

Computing Concepts	151	Api Gui OpenSource Plugin Datatypes
Systems Architecture	4	Mvc Guda EnterPrise
Data Visualization	4	Xml Json Rdf
Way for Data Processing	5	Pipeline Multithreaded Parallel
Software Name/System/Framework	90	Ruby Midi Cocoa Jack Cron
Aspects Of Computer Science	10	DataMining Bioinformatics AI Networking Machine-learning
Hardware/Device	11	Keyboard Scanner Iphone Wii
Graphical Representation of Data	3	Graph Chart Graphs

Πίνακας 6.2.1: Κατηγορίες ετικετών που εντοπίστηκαν

6.3 Σχολιασμός αποτελεσμάτων

Από το πείραμα προέκυψαν τα πιο κάτω αποτελέσματα τα οποία παρουσιάζονται σε φθίνουσα σειρά με βάση το ποσοστό το οποίο εμφανίζονται στο σύνολο των αρχείων λογισμικού στον πίνακα 6.3.1.

Α/Α	Όνομα Κατηγορίας	Συχνότητα εμφάνισης	Ποσοστό Εμφάνισης
1	Computing Concepts	938	24.53
2	Programming Language	892	23.33
3	General Description	670	17.52
4	Software Name/System/Framework	542	14.17
5	Operating System	184	4.81
6	Library	99	2.58
7	Hardware/Device	95	2.48
8	Web Browser	82	2.14
9	Protocol	76	1.98
10	Data Visualization	55	1.43
11	Database Management System	47	1.22
12	Aspect Of Computer Science	46	1.20
13	Text Editor	34	0.88
14	Way for Data processing	28	0.73
15	System Architecture	22	0.57
16	Graphical Representation of Data	13	0.34

Πίνακας 6.3.1: Ποσοστά που συγκέντρωσε κάθε κατηγορία

Όπως μπορούμε να παρατηρήσουμε από τα αποτελέσματα του πειράματος οι ετικέτες ανήκουν κυρίως σε 4 μεγάλες κατηγορίες. Με ποσοστό 24.53% παρουσιάζονται ετικέτες οι οποίες είναι έννοιες των Υπολογιστών και δίνουν κάποια σχετική πληροφορία του τι εμπεριέχει το αρχείο λογισμικού στο οποίο αναφέρονται. Με ποσοστό 23.33% παρουσιάζονται ετικέτες οι οποίες αντιπροσωπεύουν κάποια γλώσσα προγραμματισμού που πολύ πιθανόν να περιγράφει την γλώσσα στην οποία είναι υλοποιημένο ένα λογισμικό ή να περιγράφει σε τι γλώσσα είναι τυχόν κώδικες (source codes) οι οποίοι εμπεριέχονται στο αρχείο . Ακολούθως με 17.52% εμπεριέχονται ετικέτες οι οποίες δίνουν μια πιο γενική περιγραφή στο αρχείο λογισμικού και κατά πάσα πιθανότητα από μόνα τους δεν μπορούν να προσδιορίσουν τα περιεχόμενα του αρχείου στο οποίο αναφέρονται. Επιπρόσθετα με ποσοστό 14.17% εμφανίζονται ετικέτες οι οποίες προσδιορίζουν το όνομα ενός λογισμικού, κάποιου συστήματος ή κάποιου framework και δίνουν μια συγκεκριμένη πληροφορία στο αρχείο λογισμικού.Οι υπόλοιπες κατηγορίες παρουσιάζονται σε μικρότερα ποσοστά με σημαντικότερο το ποσοστό της κατηγορίας Λειτουργικό Σύστημα το οποίο παρουσιάζεται με ποσοστό 4.81% και προσδιορίζει το λειτουργικό σύστημα το οποίο υποστηρίζει το συγκεκριμένο λογισμικό.

Κεφάλαιο 7

7. Συμπεράσματα

7.1 Συμπεράσματα

Όπως προαναφέραμε, στόχος της ατομικής διπλωματικής εργασία ήταν να γίνει μια ανάλυση στο σύνολο των δεδομένων που συλλέχθηκαν από το Google Code και συγκεκριμένα να εξαχθούν κάποιες χρήσιμες πληροφορίες που αφορούν τις ετικέτες των αρχείων με σκοπό την περαιτέρω χρήση αυτών των πληροφοριών .

Από την διαδικασία της ομαδοποίησης των ετικετών, με την χρήση διαφορετικών αλγορίθμων, καταφέραμε να δημιουργήσουμε ο μάδες ετικετών με βάση του σε πια αρχεία λογισμικού εμφανίζονται. Με την αξιολόγηση των ομάδων που έγινε οδηγηθήκαμε στο συμπέρασμα ότι ως επί το πλείστο δημιουργήθηκαν αρκετά καλής ποιότητας ομάδες οι οποίες εμπεριείχαν ετικέτες στενής συσχέτισης. Κάθε αλγόριθμος ομαδοποίησης που χρησιμοποιήθηκε δημιούργησε τις δικές του ομάδες στηριζόμενος στη δική του φιλοσοφία. Τα συμπεράσματα στα οποία οδηγηθήκαμε από τα αποτελέσματα του κάθε αλγορίθμου είναι ότι κάθε αλγόριθμος μπορεί να χρησιμοποιηθεί σε διαφορετικές περιπτώσεις σύμφωνα πάντα με το σκοπό για τον οποίο προορίζονται τα δεδομένα. Ο αλγόριθμος Simple-K-Means δίνει μια προσεγγιστική λύση και προορίζεται για μεγάλου μεγέθους δεδομένα. Αντίθετα, ο ιεραρχικός αλγόριθμος δίνει μιας καλύτερης ποιότητας ομαδοποίηση, όμως έχει μεγαλύτερη χρονική πολυπλοκότητα. Όσον αφορά τον αλγόριθμο DBSCAN κάνει ένα ξεκαθάρισμα των θορυβώδη δεδομένων και ομαδοποιεί μόνο τα αντικείμενα τα οποία θεωρεί ότι μπορούν να καταταγούν σε κάποια ομάδα.

Στη συνέχεια εξάγαμε κανόνες συσχέτισης των ετικετών και πάλι με βάση του σε ποια αρχεία λογισμικού ανήκουν. Αυτή η διαδικασία επιτεύχθηκε με την χρήση δύο αλγορίθμων παραγωγής κανόνων συσχέτισης. Αυτό που παρατηρήσαμε είναι ότι η χρήση του αλγόριθμου FP-Growth επιταχύνει την διαδικασία παραγωγής των κανόνων και επιπρόσθετα μειώνει τις ανάγκες για κύρια μνήμη. Αυτό οφείλεται στον τρόπο με τον οποίο λειτουργεί αυτός ο αλγόριθμος και συγκεκριμένα η αναπαράσταση των δεδομένων με την χρήστη του Fp-tree.

Τέλος, μέσω μιας διαδικασίας προσπαθήσαμε να προσδιορίσουμε τι είδους ετικέτες εμπεριέχονται στα αρχεία λογισμικού. Από τα αποτελέσματα καταλήξαμε στο συμπέρασμα ότι οι ετικέτες ανήκουν σε τέσσερεις κύριες κατηγορίες. Στην πρώτη κατηγορία είναι ετικέτες οι οποίες αντιπροσωπεύουν έννοιες γνωστές στο χώρο της επιστήμης της Πληροφορικής και οι οποίες δίνουν κάποια σχετική πληροφορία του τι εμπεριέχεται στο συγκεκριμένο αρχείο λογισμικού. Στη δεύτερη κατηγορία ανήκουν ετικέτες οι οποίες αντιπροσωπεύουν κάποια γλώσσα προγραμματισμού, ενώ στην τρίτη κατηγορία ετικέτες οι οποίες δεν είναι απαραίτητα όροι της Πληροφορικής και δίνουν μια πιο γενική περιγραφή στο αρχείο στο οποίο εμπεριέχονται. Η τέταρτη κατηγορία αφορά ετικέτες οι οποίες προσδιορίζουν το όνομα ενός λογισμικού ή κάποιου συστήματος.

7.2 Μελλοντική εργασία

Στο Minersoft έχει υλοποιηθεί μηχανισμός σύστασης ετικετών. Τα αποτελέσματα που εξάχθηκαν από την ομαδοποίηση των ετικετών και τους κανόνες συσχέτισης μπορούν να χρησιμοποιηθούν στην δημιουργία ενός νέου και βελτιστοποιημένου συστήματος σύστασης (recommendation system) . Συγκεκριμένα εάν ένα αρχείο λογισμικού εμπεριέχει κάποιες ετικέτες, με βάση τους κανόνες συσχέτισης να προτείνονται στους χρήστες νέες ετικέτες. Επίσης, αυτή η διαδικασία μπορεί να επιτευχθεί και με τις ομάδες που προέκυψαν από την ομαδοποίηση των ετικετών. Συγκεκριμένα, εάν σε ένα αρχείο εμπεριέχονται κάποιες ετικέτες οι οποίες σύμφωνα με την ομαδοποίηση που έγινε ανήκουν στην ίδια κατηγορία τότε μπορεί να προταθούν νέες ετικέτες οι οποίες ανήκουν στην ίδια κατηγορία. Επιπρόσθετα μέσω των αποτελεσμάτων της ομαδοποίησης και των κανόνων συσχέτισης μπορεί να επιτευχθεί και ο εμπλουτισμός των αρχείων λογισμικού με νέες ετικέτες βάση των ήδη υπάρχον ετικετών.

Επιπλέον, από τα αποτελέσματα που προέκυψαν από την κατηγοριοποίηση των ετικετών (Κεφ. 6) , δηλαδή σε ποιες κατηγορίες ανήκουν οι ετικέτες που εμπεριέχονται στα αρχεία λογισμικού (Λειτουργικό Σύστημα, Γλώσσα Προγραμματισμού κτλ) μπορεί να υλοποιηθεί ένα σύστημα βελτιστοποίησης του τρόπου ταξινόμησης των αρχείων λογισμικού με βάση τις ετικέτες των χρηστών. Το Minersoft επιτυγχάνει την ταξινόμηση των αρχείων με βάση τις ετικέτες που επισύναψαν οι χρήστες σε κάποια

αρχεία. Συγκεκριμένα ο ταξινομητής του Minersoft χρησιμοποιεί αυτή την πληροφορία για να κατατάξει όλο τα αρχεία σε κατηγορίες που έχουν ορίσει οι ίδιοι οι χρήστες. Με τα αποτελέσματα που προέκυψαν από την κατηγοριοποίηση των ετικετών που παρουσιάστηκε στο κεφάλαιο 6 μπορεί να τροποποιηθεί η διαδικασία και ο ένας ταξινομητής να αντικατασταθεί με περισσότερους οι οποίοι θα εξειδικεύονται σε διαφορετικές κατηγορίες ετικετών.

Βιβλιογραφία

[1] Marios D. Dikaiakos, Asterios Katsifodimos, George Pallis ,"Minersoft: Software Retrieval in Grid and Cloud Computing Infrastructures." ACM Transactions on Internet Technologies (accepted for publication, April 2012)

[2] Ο. Ονουφρίου, "Αναζήτηση και ανάκτηση δεδομένων από πληροφοριακές υποδομές νεφέλης", University of Cyprus, Nicosia, 2011

[3] "Google Code" Google, Web November 2011. <http://code.googl.com/hosting/>

[4] I. Katakis, G. Pallis, M. D. Dikaiakos and O. Onoufriou, "Automated Tagging for the Retrieval of Software Resources in Grid and Cloud Infrastructures", IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), Ottawa, Canada, 2012

[5] U. Farooq, T. G. Kannampallil, Y. Song, C. H. Ganoe, J. M. Carroll and C. Lee Giles, "Evaluation Tagging Behavior in Social Bookmarking Systems: Metrics and design heuristics", In Proceedings of the 2007 International ACM Conference on Supporting Group Work (New York: ACM Press), 351-360

[6] P. Heymann, A. Paepcke, and H. Garcia-Molina, "Tagging human knowledge," in Proceedings of the Third ACM International Conference on Web Search and Data Mining. New York, NY, USA: ACM, 2010, pp. 51–61

[7] A. K. Jain, M. N. Murty and P. J. Flynn, "Data Clustering: A Review", ACM Computing Surveys, Vol. 31, No. 3, September 1999

[8] Ester M., Kriegel H.-P., Sander J., Xu X.: "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, 1996, pp. 226-231

[9] “Gephi”, Web. February 2012
<http://gephi.org/>

[10] Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast Unfolding of Community Hierarchies in Large Networks. Eprint arXiv:0803.0476v1 at arxiv.org (2008)

[11] R. Dimov, “Weka: Practical machine learning tools and techniques with Java implementations”, AI Tools Seminar University of Saarland 2007

[12] “Attribute-Relation File Format (ARFF).” University of Waikato, 1 November 2008. Web. February 2012
<http://www.cs.waikato.ac.nz/~ml/weka/arff.html>

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and Ian H. Witten (2009), “The Weka Data Mining Software: An Update”, 2nd ed, San Francisco, 2005

[14] F. Kovacs, C. Legany and A. Babos “Cluster Validity Measurement Techniques” 6th Int. Symposium of Hungarian Researchers on Computational Intelligence, November 18-19, Budapest, Hungary, 2005

[15] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, “On Clustering Validation Techniques,” Intelligent Information Systems J., 2001

[16] Pang-Ning Tan, M. Steinbach and V. Kumar, “Introduction to Data Mining ”, ch. 6 “Association Analysis: Basic Concepts and Algorithms” University of Minnesota (2006)

Παράρτημα Α

Στο παράρτημα αυτό παρουσιάζεται η κλάση για την δημιουργία του arff αρχείου, το οποίο χρησιμοποιείται από το Weka για την πραγματοποίηση της ομαδοποίησης.

Παράρτημα Α

Στο παράρτημα αυτό παρουσιάζεται η κλάση για την δημιουργία του arff αρχείου, το οποίο χρησιμοποιείται από το Weka για την πραγματοποίηση της ομαδοποίησης.

```
public class convertInArffFormatForTags {
	static FastVector AllAttributes = new FastVector(968);
	static String[] allProjectNames=new String[967];
	static String[] allTags=new String[476];

	//method for declare attributes
	static void declareAttributes()
	{
		//read all projects' names
		try {
			FileInputStream fstream = new
			FileInputStream("ProjNamesWithMoreThan1Tags.txt");
			DataInputStream in = new DataInputStream(fstream);
			BufferedReader br = new BufferedReader(new
			InputStreamReader(in));
			String strLine;

			int i=0;
			while ((strLine = br.readLine()) != null) {

				// Declare a nominal attribute along with its
				values
				FastVector fvNominalVal = new FastVector(2);
				fvNominalVal.addElement("yes");
				fvNominalVal.addElement("no");

				Attribute Attribute1 = new Attribute(strLine,
				fvNominalVal);

				AllAttributes.addElement(Attribute1);

				allProjectNames[i]=strLine;

				i++;
			}
				AllAttributes.addElement(new
				Attribute("TagName", (FastVector) null));

		}
		catch(Exception e){

		}
	}
```

```
	//method to get all tags' names
	static void getAllTags()
	{
		try {
			FileInputStream fstream = new
			FileInputStream("TagsWithMoreThan2.txt");
			DataInputStream in = new DataInputStream(fstream);
			BufferedReader br = new BufferedReader(new
			InputStreamReader(in));
			String strLine;
			int i=0;

			while ((strLine = br.readLine()) != null) {

				allTags[i]=strLine;

				i++;
			}
		}
		catch(Exception e){
		}
	}

	//method to create instances
	static void createInstances(Instances isTrainingSet)
	{
		int flag=0;

		for(int i=0;i<allTags.length;i++)
		{
			Instance iExample = new Instance(968);
			LinkedList<String> tempTags=new
			LinkedList<String>();
			for(int j=0;j<allProjectNames.length;j++)
			{
				//read tags of a projects
				tempTags.clear();
				try {
					String
location="C:\\Users\\Harris\\workspace\\Weka\\testing\\"+allProjectNam
es[j]+"\\";
FileInputStream fstream = new FileInputStream(location+"tags.txt");
DataInputStream in = new DataInputStream(fstream);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
String strLine;


				while ((strLine = br.readLine()) != null) {
					tempTags.add(strLine);
				}
				}
				catch(Exception e){
				}

				Iterator<String> iter=tempTags.iterator();
				String tempp="";
				flag=0;
				while(iter.hasNext())
				{
					tempp=iter.next();
```

```
        if(tempp.compareToIgnoreCase(allTags[i])==0)
                                {
                                        flag=1;
                                }
                        }
        if(flag==0)
        {
        iExample.setValue((Attribute)AllAttributes.elementAt(j), "no");
        }
        else
        {
        iExample.setValue((Attribute)AllAttributes.elementAt(j), "yes");
        }

        if(j==(allProjectNames.length-1))
        {
        iExample.setValue((Attribute)AllAttributes.elementAt(j+1),
        allTags[i]);
        }
                }
                //add instance in trainingset
                isTrainingSet.add(iExample);

                }
        }

public static void main(String[] args) throws IOException {
                // TODO Auto-generated method stub
                declareAttributes();
                Instances isTrainingSet = new
                Instances("GoogleCodeProjectsAndTags", AllAttributes,968);
                getAllTags();
                createInstances(isTrainingSet);

                //create arff file
                ArffSaver saver = new ArffSaver();
                saver.setInstances(isTrainingSet);
                saver.setFile(new
                File("FinalTagsFormatWithTagNameMoreThan2.arff"));
                saver.writeBatch();
        }
}
```

Παράρτημα Β

Στο παράρτημα αυτό παρουσιάζεται η κλάση για την παραγωγή κανόνων συσχέτισης χρησιμοποιώντας τον αλγόριθμο Apriori.

Παράρτημα Β

Στο παράρτημα αυτό παρουσιάζεται η κλάση για την παραγωγή κανόνων συσχέτισης χρησιμοποιώντας τον αλγόριθμο Apriori.

```
public class AssociationRules {


	private String[]    inputText       = null;
	private String      classString     = null; // the classifier
	private FastVector  attributeInfo   = null;
	private Instances   instances       = null;
	private int[][]     productMatrix   = null;
	private String[]    productNames    = null;
	private Attribute[] attributes      = null;

	private Apriori     apriori         = null;

	// apriori specific parameters
	private double deltaValue                = 0.01;
	private double lowerBoundMinSupportValue =  0.003;
	private double minMetricValue            = 0.1;
	private int numRulesValue                = 10000;
	private double significanceLevelValue    = -1.0;
	private double upperBoundMinSupportValue = 0.63;



	//
	// main, mainly for testing
	//
	public static void main(String args[]) {
		long start=System.currentTimeMillis();
		int thisNumRules = 10000;
		String[] allBuskets=new String[967];
		if (args.length > 0) {
			try {
				thisNumRules = Integer.parseInt(args[0]);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}




		//read all baskets' files
		try {
			FileInputStream fstream = new
			FileInputStream("AllBasketsFile.txt");
			DataInputStream in = new DataInputStream(fstream);
			BufferedReader br = new BufferedReader(new
			InputStreamReader(in));
```

```
            String strLine;
            int i=0;


            while ((strLine = br.readLine()) != null) {

                  allBuskets[i]=strLine;
                  i++;
            }
      }
      catch(Exception e){

      }


      String[] inputText=allBuskets;


      AssociationRules associationRules = new
      AssociationRules(inputText);

      associationRules.setNumRules(thisNumRules);
      System.out.println(associationRules.associate());


      try{
            // Create file with results
            FileWriter fstream = new
            FileWriter("AssociationRulesApriori.txt");
            BufferedWriter out = new BufferedWriter(fstream);


            out.write(associationRules.associate().toString());


            //Close the output stream
            out.close();
      }catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
      }

      long elapsedTime = System.currentTimeMillis() - start;
      System.out.println("***Time:"+elapsedTime/1000F+" sec");
} // end main



//
// constructor
//
AssociationRules(String[] inputText) {

      this.inputText      = inputText;


      //
      // creates this.productMatrix
      //         this.productNames
      //
```

```
            createBasketMatrix(inputText);


            //
            // creates attribute structures
            //
            FastVector occurenceVector = new FastVector(2);
            occurenceVector.addElement("0");
            occurenceVector.addElement("1");
            Attribute occurenceAttribute = new Attribute("occurence",
            occurenceVector);

            // add all attributes
            attributeInfo = new FastVector();
            attributes = new Attribute[productNames.length];
            for (int i = 0; i < productNames.length; i++) {
                  Attribute thisAttribute = new
                  Attribute(productNames[i], occurenceVector);
                  attributes[i] = thisAttribute;
                  attributeInfo.addElement(thisAttribute);
            }


      } // end AssociationRules


      public StringBuffer associate() {
            return(associate(deltaValue, lowerBoundMinSupportValue,
minMetricValue,numRulesValue,significanceLevelValue,upperBoundMinSuppo
rtValue));
      }


      //
      // the main association method
      //
      public StringBuffer associate(double deltaValue, double
      lowerBoundMinSupportValue, double minMetricValue, int
      numRulesValue, double significanceLevelValue, double
      upperBoundMinSupportValue) {

            StringBuffer result = new StringBuffer();

            this.deltaValue = deltaValue;
            this.lowerBoundMinSupportValue =
            lowerBoundMinSupportValue;
            this.numRulesValue             = numRulesValue;
            this.minMetricValue            = minMetricValue;
            this.significanceLevelValue    = significanceLevelValue;
            this.upperBoundMinSupportValue =
            upperBoundMinSupportValue;
```

```
        // creates an empty instances set
        instances = new Instances("data set", attributeInfo, 100);

        //
        // populate instances
        //
        for (int i = 0; i < productMatrix.length; i++) {
              Instance inst = new Instance(attributes.length);
              for (int j = 0; j < attributes.length; j++) {
              inst.setValue(attributes[j],
              productMatrix[i][j]);
              }
              instances.add(inst);
        }


        try {

              result.append("DATA SET:\n" + instances + "\n");


              //
              // Do the Apriori thing!
              //
              apriori = new Apriori();
              apriori.setDelta(deltaValue);

apriori.setLowerBoundMinSupport(lowerBoundMinSupportValue);
apriori.setMinMetric(minMetricValue);
apriori.setNumRules(numRulesValue);
apriori.setUpperBoundMinSupport(upperBoundMinSupportValue);
apriori.buildAssociations(instances);
result.append(apriori.toString() + "\n");


        } catch (Exception e) {
              e.printStackTrace();
              result.append("\nException (sorry!):\n" +
              e.toString());
        }

        return result;

} // end associate
```

```
	/**
	 * converts a string array to a matrix
	 *
	 * returns a string array of the products (product names)
	 *
	 */
	public void createBasketMatrix(String[] basket) {

		int numBaskets = basket.length;

		HashMap productMap = new HashMap();
		HashSet productSet = new HashSet();

		//
		// parse the products id's
		// assumption: product id are strings
		//
		for (int i = 0; i < basket.length; i++) {
			String[] thisBasket = basket[i].split("[\\s,]+");
			List theseProducts = new ArrayList();
			Integer basketIx = new Integer(i);
			for (int j = 0; j < thisBasket.length; j++) {
				try {
					String thisVal = thisBasket[j];
					productSet.add((String)thisVal);
					theseProducts.add(thisVal);
				} catch (Exception e) {
					e.printStackTrace();
				}
				productMap.put(basketIx, theseProducts);
			}
		}

		int numProducts = productSet.size();


		//
		// create the product matrix (numBasket x numProducts)
		//
		String[] products = (String[])productSet.toArray(new
String[0]);

		this.productMatrix = new int[numBaskets][numProducts];

		// for each basket
		for (int i = 0; i < numBaskets; i++) {
			ArrayList theseProds = (ArrayList)productMap.get(new
Integer(i));

			// for each product
			// note: these are not in sorted order
			for (int j = 0; j < products.length; j++) {
				if (theseProds.contains((String)products[j]))
{
					this.productMatrix[i][j] = 1;
				} else {
					this.productMatrix[i][j] = 0;
				}

			}
```

```
            }

            this.productNames = products;

      } // end createBasketMatrix



      //
      // setter for the classifier _string_
      //
      public void setClassifierString(String classString) {
            this.classString = classString;
      }

      public int[][] getProductMatrix() {
            return this.productMatrix;
      }


      public void setNumRules(int numRulesValue) {
            this.numRulesValue = numRulesValue;
      }

      public void setMinMetric(double minMetric) {
            this.minMetricValue = minMetric;
      }


      public void setLowerBoundMinSupport(double
lowerBoundMinSupportValue) {
            this.lowerBoundMinSupportValue =
lowerBoundMinSupportValue;
      }

}
```

Παράρτημα Γ

Αυτό το παράρτημα εμπεριέχει την κύρια κλάση για τον υπολογισμό των ποσοστών που εμφανίζεται κάθε κατηγορία ετικετών στα αρχεία λογισμικού.

Παράρτημα Γ

Αυτό το παράρτημα εμπεριέχει την κύρια κλάση για τον υπολογισμό των ποσοστών που εμφανίζεται κάθε κατηγορία ετικετών στα αρχεία λογισμικού.

```
public class ComputePersentageOfCategories {

	/**
	 * @param args
	 */
	static ArrayList<Category> AllCategories=new
	ArrayList<Category>();
	static String[] AllTags=new String[475];
	static int sum=0;

	//read all tags;
	static void readAllTags()
	{
		try {
			FileInputStream fstream = new
			FileInputStream("TagsforCategories.txt");
			DataInputStream in = new DataInputStream(fstream);
			BufferedReader br = new BufferedReader(new
			InputStreamReader(in));
			String strLine;
			int i=0;


			while ((strLine = br.readLine()) != null) {
				strLine=strLine.toLowerCase();
				strLine=strLine.replace(" ","");
				AllTags[i]=strLine;
				i++;
			}
		}
		catch(Exception e){

		}
	}

	//read categories with their tags
	static void readTagsAndCategories()
	{
		try {
			FileInputStream fstream = new
			FileInputStream("TagsWithCategories.txt");
			DataInputStream in = new DataInputStream(fstream);
			BufferedReader br = new BufferedReader(new
			InputStreamReader(in));
			String strLine;
			int i=0;
```

```
                while ((strLine = br.readLine()) != null) {
                      //every line with *** is a category
                      if(strLine.contains("***"))
                      {
                            strLine=strLine.replace("*","");
                            Category newc=new Category();
                            newc.name=strLine;
                            AllCategories.add(newc);
                            continue;
                      }
                      AllTags[i]=strLine;
                      AllCategories.get(AllCategories.size()-
                      1).tags.add(strLine);
                      i++;
                }
          }
          catch(Exception e){

          }
    }

    //method that set the counter of every tag
    static void setCounter()
    {
          try {
                FileInputStream fstream = new
                FileInputStream("TagsWithCounter.txt");
                DataInputStream in = new DataInputStream(fstream);
                BufferedReader br = new BufferedReader(new
                InputStreamReader(in));
                String strLine;



                while ((strLine = br.readLine()) != null) {
                      String[] temp=strLine.split("[\t]");
                      sum=sum+Integer.parseInt(temp[1]);
                      for(int i=0;i<AllTags.length;i++)
                      {
                            temp[0]=temp[0].replace(" ","");

    if(AllTags[i].compareToIgnoreCase(temp[0])==0)
                            {
                            for(int j=0;j<AllCategories.size();j++)
                                  {
                                        for(int
                                        k=0;k<AllCategories.get(j).t
                                        ags.size();k++)
                                        {

    if(AllCategories.get(j).tags.get(k).compareToIgnoreCase(temp[0])
==0)
                                              {

    AllCategories.get(j).counter=AllCategories.get(j).counter+Intege
r.parseInt(temp[1]);
                                                    break;
                                              }
                                        }
                                  }
```

```
                        }
                    }


                }
            }
            catch(Exception e){

            }
        }



        public static void main(String[] args) {
            // TODO Auto-generated method stub
            //readAllTags();
            readTagsAndCategories();
            setCounter();

            int athroisma=0;
            for(int i=0;i<AllCategories.size();i++)
            {
                athroisma=athroisma+AllCategories.get(i).counter;
            }


            //print results
            for(int i=0;i<AllCategories.size();i++)
            {
            float perc=(float)AllCategories.get(i).counter/athroisma;


        System.out.println(AllCategories.get(i).name+"("+AllCategories.g
et(i).tags.size()+")"+"= "+perc*100+"%"+"
Frequency="+AllCategories.get(i).counter);
            }

        }

}
```