Ατομική Διπλωματική Εργασία

ΕΥΦΥΗΣ ΑΝΑΚΤΗΣΗ ΠΟΡΩΝ ΛΟΓΙΣΜΙΚΟΥ ΣΕ ΥΠΟΛΟΓΙΣΤΙΚΕΣ ΝΕΦΕΛΕΣ

Ονησίφορος Ονουφρίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ευφυής Ανάκτηση Πόρων Λογισμικού σε Υπολογιστικές Νεφέλες

Ονησίφορος Ονουφρίου

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

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή, Δρ. Γιώργο Πάλλη, που με εμπιστεύθηκε για την εκπόνηση αυτής της διπλωματικής εργασίας, όπως επίσης και για τη συνεχή καθοδήγησή του.

Ακόμη, θα ήθελα να ευχαριστήσω τον Δρ. Μάριο Δικαιάκο που μου έδωσε την ευκαιρία να εργαστώ στο Εργαστήριο Υπολογιστών Υψηλών Επιδόσεων.

Τέλος, θα ήθελα να ευχαριστήσω τον Δρ. Ιωάννη Κατάκη για τη βοήθεια και τις συμβουλές του, καθώς επίσης και τον Αστέριο Κατσιφοδήμο για τις υποδείξεις του.

Σας ευχαριστώ,
Ονησίφορος Ονουφρίου

Περίληψη

Η αναζήτηση και ανάκτηση πληροφορίας στο Διαδίκτυο είναι μέρος της καθημερινής ζωής εκατομμυρίων ανθρώπων. Κάθε μέρα όλο και περισσότεροι άνθρωποι χρησιμοποιούν μηχανές αναζήτησης για να ικανοποιήσουν τις πληροφοριακές τους ανάγκες ή απλά να πλοηγηθούν στον Παγκόσμιο Ιστό. Μετά από μια αναζήτηση, είναι σημαντικό κάποιος να είναι σε θέση να αποφασίσει αν τα αποτελέσματα που του έχουν επιστραφεί ικανοποιούν τις ανάγκες του. Σε αυτή τη διπλωματική εργασία μελετήσαμε και επεκτείναμε τη μηχανή αναζήτησης Minersoft[1]. Το Minersoft είναι μια μηχανή αναζήτησης η οποία μας επιτρέπει να συλλέξουμε πληροφορίες που αφορούν λογισμικό το οποίο είναι ήδη εγκατεστημένο σε κόμβους υπολογιστικών υποδομών μεγάλης κλίμακας Grid και Cloud. Αφού ανακτήσαμε τις πληροφορίες από 30 τέτοιους κόμβους, 20 από το EC2 της Amazon και 10 από το Rackspace, μελετήσαμε αν είναι δυνατή η χρήση μεθόδων και αλγορίθμων ταξινόμησης από τη Μηχανική Μάθηση με σκοπό την ταξινόμηση των πόρων λογισμικού που ανακτήσαμε σε ανανεώσιμες κατηγορίες μαζί με τη συμβολή των χρηστών μέσω ετικετών. Επιπλέον, υλοποιήσαμε μια διαδικασία μέσω της οποίας κάθε αποτέλεσμα της μηχανής αναζήτησης του Minersoft συσχετίζεται με πληροφορίες που προέκυψαν από αναζήτηση στο διαδίκτυο. Τέλος δημιουργήσαμε μια γραφική διεπαφή μέσω της οποίας οι χρήστες μπορούν να κάνουν αναζήτηση με λέξεις – κλειδιά στη μηχανή αναζήτησης του Minersoft.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Η ανάκτηση πληροφορίας [3] ορίζεται ως η ανεύρεση περιεχομένου, το οποίο βρίσκεται συνήθως σε αδόμητη μορφή και ικανοποιεί τις πληροφοριακές μας ανάγκες. Το υλικό αυτό βρίσκεται σε μεγάλες συλλογές, μέσα σε πολλούς ηλεκτρονικούς υπολογιστές και γρήγορα η αναζήτηση πληροφορίας έγινε μια παγκόσμια ανάγκη μέσω του Παγκόσμιου Ιστού και της εμφάνισης ολοένα και περισσότερων μηχανών αναζήτησης.

Τώρα αυτή η εξέλιξη συνεχίζεται στο Cloud computing, το οποίο αποτελεί ένα διαδικτυακό μοντέλο παροχής υπηρεσιών υπολογιστικής ισχύς, δικτύωσης, αποθηκευτικού χώρου καθώς και ποικιλία εγκατεστημένων λογισμικών. Και ενώ οι εταιρίες που παρέχουν τις υπηρεσίες τους σε Cloud υποδομές αυξάνονται με ραγδαίους ρυθμούς, παρουσιάστηκε ξανά η ανάγκη αναζήτησης και ανάκτησης πληροφορίας πάνω σε αυτές τις υπολογιστικές υποδομές.

Ένας από τους πρωταρχικούς στόχους αυτών των υποδομών, είναι η ύπαρξη εύκολης πρόσβασης στους χρήστες. Επιπλέον, κάποιος χρήστης ο οποίος θα ήθελε να ενοικιάσει χώρο στο Cloud ώστε να χρησιμοποιήσει ένα ήδη εγκατεστημένο λογισμικό θα δυσκολευόταν να το εντοπίσει. Στο πρόβλημα του εντοπισμού λογισμικού, έρχεται να δώσει απάντηση το Minersoft [1] ένα εργαλείο για αναζήτηση λογισμικού πάνω σε Cloud αλλά και Grid υποδομές. Παρόλο όμως που το Minersoft προσφέρει τη δυνατότητα στους χρήστες για μια εύκολη αναζήτηση, δεν παρέχει επαρκείς πληροφορίες σχετικά με το λογισμικό που υπάρχει σε αυτές τις υποδομές, όπου θα διευκόλυναν τους χρήστες στην επιλογή του ορθού λογισμικού.

1.1 Κίνητρο

Σε αυτή την εργασία μελετούμε τρόπους παροχής επιπρόσθετης πληροφορίας στους χρήστες της μηχανής αναζήτησης Minersoft σε Cloud υποδομές. Το Minersoft δίνει τη δυνατότητα στους χρήστες του να αναζητήσουν κάποιο συγκεκριμένο λογισμικό που θέλουν να χρησιμοποιήσουν με τη χρήση λέξεων κλειδιών και παρέχει βασικές πληροφορίες όπως είναι το όνομά κάποιου πόρου λογισμικού, την τοποθεσία του καθώς και το αν το αποτέλεσμα που επιστράφηκε μέσω της αναζήτησης είναι κάποια βιβλιοθήκη, κώδικας κ.τ.λ. Όμως αν κάποιος άπειρος χρήστης χρειαστεί να αναζητήσει

ένα τέτοιο λογισμικό, με την ύπαρξη μόνο αυτών των βασικών πληροφοριών θα δυσκολευτεί πολύ να κάνει μια σωστή επιλογή. Σε αναλογία, αν οι μηχανές αναζήτησης στο διαδίκτυο δεν επέστρεφαν μαζί με κάθε ένα αποτέλεσμα και μια μικρή περιγραφή η οποία προέρχεται από τα περιεχόμενα του ιστιακού πόρου που αναζητήσαμε, θα μας δυσκόλευε πολύ η επιλογή του αποτελέσματος. Επιπλέον υπάρχει η ανάγκη για μια εύχρηστη γραφική διεπαφή χρήστη, μέσω της οποίας ο χρήστης δεδομένης της εμπειρίας του σε μηχανές αναζήτησης του διαδικτύου, θα μπορεί να τη χρησιμοποιήσει χωρίς κανένα εμπόδιο.

1.2 Συνεισφορά

Η βασική συνεισφορά αυτής της διπλωματικής εργασίας είναι:

- Ανάπτυξη μιας εύχρηστης γραφικής διεπαφής χρήστη, μέσω της οποίας οι ενδιαφερόμενοι χρήστες θα μπορούν να εκτελούν αναζήτηση χρησιμοποιώντας τη μηχανή αναζήτησης Minersoft.
- Ανάπτυξη μίας νέας λειτουργίας η οποία επιτρέπει στους χρήστες της μηχανής αναζήτησης Minersoft να τοποθετούν ετικέτες στα αποτελέσματα της αναζήτησης τους έτσι ώστε με τη χρήση ενός ταξινομητή να είναι δυνατός ο εμπλουτισμός των αρχείων λογισμικού με λέξεις κλειδιά καθώς επίσης και η επιμέρους ταξινόμηση των αρχείων αυτών σε κατηγορίες οι οποίες είναι πιο αντιληπτές από τους χρήστες.
- Ανάπτυξη μιας διαδικασίας για τη συσχέτιση αποτελεσμάτων της μηχανής αναζήτησης Minersoft με πληροφορίες από το Διαδίκτυο.
- Πειραματισμός με διάφορους γνωστούς αλγόριθμους ταξινόμησης και η αξιολόγησή τους σε πραγματικά δεδομένα από το Virtual Organization atlas.

Τα υπόλοιπα κεφάλαια αυτής της διπλωματικής εργασίας έχουν ως εξής: Στο Κεφάλαιο 2 οι συσχετιζόμενες εργασίες θα περιγραφούν περιληπτικά. Στο Κεφάλαιο 3 θα γίνει μια γενική ανασκόπηση της αρχιτεκτονικής της μηχανής αναζήτησης του Minersoft καθώς και μια συνοπτική περιγραφή της λειτουργίας του. Επίσης θα περιγραφεί με λεπτομέρεια η γραφική διεπαφή χρήστη που αναπτύξαμε για αυτή τη μηχανή αναζήτησης. Στο Κεφάλαιο 4 θα περιγραφούν διάφοροι μέθοδοι ταξινόμησης δεδομένων καθώς και ο ρόλος που θα διαδραματίσουν στη μηχανή αναζήτησης ενώ στο

Κεφάλαιο 5 θα αναφερθούμε στη διαδικασία που αναπτύξαμε για να εμπλουτίσουμε τα αποτελέσματα και με επιπρόσθετες πληροφορίες από το Διαδίκτυο. Στο Κεφάλαιο 6 θα παρουσιάσουμε τις λεπτομέρειες σχεδίασης και υλοποίησης των διαδικασιών που ως σκοπό έχουν τον εμπλουτισμό των αρχείων λογισμικού με επιπλέον πληροφορίες που προέρχονται από τους χρήστες. Ακολούθως στο Κεφάλαιο 7 θα γίνει πειραματική αξιολόγηση των μεθόδων και διαδικασιών που περιγράφονται στα Κεφάλαια 4 και 5. Τέλος, στο Κεφάλαιο 8 περιλαμβάνονται διάφορα συμπεράσματα.

Κεφάλαιο 2

Προηγούμενες Έρευνες

Οι έρευνες που έγιναν για προηγούμενες εργασίες συμπεριελάμβαναν θέματα από τρείς κύριους τομείς. Πρώτα μελετήθηκαν εργασίες που είχαν ως κύριο θέμα τους την ανάκτηση πληροφορίας και συγκεκριμένα για λογισμικό. Ακολούθως, έγινε επέκταση σε θέματα Μηχανικής Μάθησης και Εξόρυξης Δεδομένων [21, 22] και τέλος μελετήθηκαν πιθανοί τρόποι για ανάκτηση πληροφορίας από το διαδίκτυο.

2.1 Ανάκτηση πληροφορίας λογισμικού (Software Retrieval)

Μέχρι σήμερα έχουν γίνει αρκετές έρευνες και υπάρχουν αρκετές προσεγγίσεις στον τομέα της ανάκτησης λογισμικού (Software Retrieval), οι οποίες περιγράφονται συνοπτικά στον πίνακα 1.1. Όπως φαίνεται, οι πλείστες από αυτές χρησιμοποιούν ως πόρους λογισμικού (Software resources) τα αρχεία κώδικα με εξαίρεση τις μεθόδους GURU [33], Koders [15] και Google Code Search [13] οι οποίες εκτελούν αναζήτηση και σε αρχεία περιγραφής.

Approach	***Corpus***	***Search paradigm***	***Software Resources***			
			Binaries	***Libraries***	***Description Docs***	***Source Codes***
GURU	Software Repositories	Keyword			×	×
SEC	Software Repositories	Keyword				×
Maracatu	Software Repositories	Keyword				×
Extreme Harvesting	Web	Keyword				×
SPARS-J	Web	Keyword				×
Koders	Web	Keyword			×	×
Google Code Search	Web	Keyword			×	×

Sourcerer	Web	Keyword				×
MinerSoft	Cloud/Grid	Keyword	×	×	×	×

Πίνακας 2.1: Προσεγγίσεις στην ανάκτηση λογισμικού

Στο άρθρο [26] οι συγγραφείς παρουσιάζουν το SmartScan, έναν crawler για την ανάκτηση μεταδεδομένων αρχείων και καταλόγων, από το σύστημα αρχείων μηχανών με απώτερο σκοπό την υποβοήθηση των διαχειριστών συστήματος ώστε να εντοπίσουν πληροφορίες όπως «ποια υπόδενδρα καταλόγων χρησιμοποιούν περισσότερο αποθηκευτικό χώρο». Μετά από πειράματα σε δύο μεγάλα συστήματα αρχείων ενός εξυπηρετητή, ο οποίος παρέχει αποθηκευτικό χώρο σε περισσότερα από 100 πανεπιστήμια, και ενός εξυπηρετητή, ο οποίος εξυπηρετεί τις ανάγκες περισσότερων από 3000 φοιτητών από διαφορετικά πανεπιστήμια, έδειξαν κάποια αποτελέσματα. Αυτά ήταν : 1) με τον περιορισμό της διαδικασίας του crawling σε καταλόγους των οποίων τα αρχεία τροποποιήθηκαν πρόσφατα και όχι σε ολόκληρο το σύστημα αρχείων των εξυπηρετητών, το crawling μειώνεται μια με δύο τάξεις μεγέθους, 2) ενώ τα ανανεωμένα δεδομένα τα οποία δεν λαμβάνονται υπόψη στο crawling περιορίστηκαν στο ελάχιστο. Επίσης τα αποτελέσματα σε ερωτήματα μεταδεδομένων δεν διαφέρουν πολύ από τις τιμές που λαμβάνονται όταν έχει προηγηθεί crawl σε όλο το σύστημα αρχείων.

Το MinerSoft [1] αποτελεί μια καινοτομία στις μεθοδολογίες ανάκτησης λογισμικού αφού κάνει χρήση τεσσάρων διαφορετικών πόρων λογισμικού (Binary files, libraries, Description Documents και Source codes). Σκοπός δημιουργίας του Minersoft ήταν η ανάγκη για μια γρήγορη, ακριβής και βασισμένη σε λέξεις κλειδιά (Keyword based) αναζήτηση για την ανάκτηση αρχείων λογισμικού από διάφορες Cloud και Grid υποδομές (π.χ. Amazon EC2, Rackspace, EGEE Grid).

Αρχικά γίνεται ιχνηλασία (crawling) στα διάφορα Grid sites και Cloud servers, κατασκευάζεται το δέντρο συστήματος (file system tree), αποκόπτονται τα αχρείαστα αρχεία και εντοπίζονται οι συσχετίσεις μεταξύ των αρχείων και δημιουργούνται τα ανεστραμμένα ευρετήρια (indexes). Στη συνέχεια τα αρχεία εμπλουτίζονται με περιγραφικά κλειδιά (keyword descriptors) τα οποία βρίσκονται σε αρχεία κειμένου και

ανανεώνονται οι πληροφορίες στα ανεστραμμένα ευρετήρια. Τέλος, με την ολοκλήρωση των παραπάνω διεργασιών το σύστημα είναι σε θέση να δώσει αποτελέσματα στα διάφορα ερωτήματα (queries).

Μέσα από την έρευνα αυτή προέκυψαν αρκετά σημαντικά πειραματικά αποτελέσματα τόσο για τη διαδικασία του crawling και indexing όσο και για το γράφο λογισμικού (Software graph).

Όσον αφορά τη διαδικασία crawling και indexing αφού έγινε επιτυχής ανίχνευση 6.5 εκατομμυρίων αρχείων (συνολικό μέγεθος 380G) το 1/3 των οποίων ανήκε σε διαφορετικά Grid sites προέκυψε ότι η διαδικασία του indexing επηρεάζεται σημαντικά από το υλικό (hardware), τον τύπο των αρχείων καθώς επίσης και από τον φόρτο εργασία των cloud servers. Επιπλέον, από τα αποτελέσματα φαίνεται ότι το 75% αρχείων στου διάφορους cloud servers και Grid sites είναι αρχεία λογισμικού.

Εξίσου σημαντικά ήταν και τα αποτελέσματα της πειραματικής μελέτης του γράφου λογισμικού (Software graph). Τα αποτελέσματα αυτά έδειξαν ότι το MinerSoft μπορεί να βελτιώσει το Precision-10 κατά 160% καθώς επίσης και τα Cumulative gain measures (NDCG, NCG) κατά 173%. Τέλος, η χρήση stemming μπορεί να περιορίσει την απόδοση του συστήματος κατά 4%, όμως μειώνει το μέγεθος των inverted indexes περίπου κατά 10%.

2.2 Tagging και Multi-Label ταξινόμηση

Tagging περιγράφεται ως η διαδικασία κατά την οποία οι χρήστες κάποιας εφαρμογής ή κάποιας ιστοσελίδας αναθέτουν μικρή ποσότητα κειμένου η οποία περιγράφει αντικείμενα με κάποια πληροφορία. Η χρήση των ετικετών (tags) έχει διαδοθεί πάρα πολύ μέσα στα τελευταία χρόνια χάρη στην εξέλιξη του WEB σε WEB 2.0. Στο άρθρο [4] οι συγγραφείς προσπάθησαν να χρησιμοποιήσουν τον αλγόριθμο ταξινόμησης πολλαπλών ετικετών (multi-label) Binary Relevance, με σκοπό την αυτόματη σύσταση ετικετών (tags) για το bibsonomi, ένα σύστημα στο οποίο οι χρήστες αποθηκεύουν κείμενο, βάζουν και μοιράζονται σελιδοδείκτες για βιβλιογραφικές αναφορές και

κάνουν εύκολη την ανάκτηση των πληροφοριών που αποθηκεύουν με την χρήση ετικετών (tags).

Στο άρθρο [32], παρουσιάζεται μια μέθοδος για σύσταση ετικετών (tags). Η μέθοδος αυτή ονομάζεται LATRE και εξάγει κανόνες συσχέτισης από συγκεκριμένα *demand-driven* δεδομένα εκπαίδευσης. Η LATRE ερμηνεύει κάθε κανόνα που εξάγει ως μια ψήφο για μια υποψήφια ετικέτα (tag) και αφού όλες οι ψήφοι έχουν προστεθεί, για κάθε μια ετικέτα υπολογίζεται μια βαθμολογία της. Αργότερα γίνεται ταξινόμηση των ετικετών βάση της βαθμολογίας αυτής και η LATRE βαθμονομεί τις βαθμολογίες αυτές για διόρθωση τυχόν στρεβλώσεων στην τελική κατάταξη των ετικετών.

Στο άρθρο [31], οι συγγραφείς αντιμετωπίζουν το πρόβλημα της σύστασης εγγράφων (π.χ. ιστοσελίδων, ερευνητικών άρθρων) με τη χρήση πληροφορίας που προέρχεται μέσω Tagging των χρηστών, προτείνοντας ένα νέο αλγόριθμο που εξυπηρετεί αυτό το σκοπό. Ο αλγόριθμος αυτός ονομάζεται Multi-type Interrelated Objects Embedding (MIOE) και βρίσκει το σημασιολογικό χώρο τριών οντοτήτων, των χρηστών των ετικετών (tags) και των εγγράφων κρατώντας τα έγγραφα κοντά στο χώρο αυτό. Όταν ο χώρος αυτός υπολογιστεί, τότε έγγραφα τα οποία βρίσκονται κοντά σε αυτόν προτείνονται στους χρήστες.

Η single-label ταξινόμηση αντικειμένων αφορά την εκπαίδευση ενός ταξινομητή με την χρήση μιας ομάδας αντικειμένων τα οποία ανήκουν το κάθε ένα μόνο σε μια κατηγορία, με απώτερο σκοπό να είναι σε θέση να κατατάξει μη κατηγοριοποιημένα αντικείμενα σε διαφορετικές κατηγορίες. Η multi-label ταξινόμηση εφαρμόζει την ίδια ακριβώς ιδέα με μόνη διαφορά ότι τα αντικείμενα μπορεί να ανήκουν σε περισσότερες από μία κατηγορίες. Η multi-label ταξινόμηση μπορεί να χωριστεί σε δύο κατηγορίες μεθόδων. Σε 1) *problem transformation methods* και 2) *algorithm adaptation methods* [6]. Οι μέθοδοι που ανήκουν στην πρώτη κατηγορία εφαρμόζουν κάποιους μετασχηματισμούς έτσι ώστε να μετατρέψουν ένα πρόβλημα multi-label ταξινόμησης σε πολλά προβλήματα single-label ταξινόμησης. Οι μέθοδοι που ανήκουν στην δεύτερη κατηγορία είναι αλγόριθμοι που ακολουθούν το παράδειγμα των δέντρων αποφάσεων, πιθανολογικές μέθοδοι, νευρωνικών δικτύων και συσχετιστικών μεθόδων.

Στο άρθρο [7] οι συγγραφείς προτείνουν τη μέθοδο ταξινόμηση πολλαπλών ετικετών (multi-label) RAKEL (RAndom k-labELsets) η οποία χρησιμοποιεί ένα σύνολο από Label Powerset ταξινομητές, όπου κάθε ένας εκπαιδεύεται σε ένα υποσύνολο από ετικέτες. Η μέθοδος αυτή αξιολογήθηκε σε δεδομένα που αφορούν πρωτεΐνες, εικόνες και αρχεία κειμένου και συγκρίθηκε με τους αλγορίθμους Binary Relevance και Label Powerset έχοντας καταφέρει πιο αποτελεσματική ταξινόμηση.

Στο άρθρο [5] η αυτοματοποιημένη ανίχνευση συναισθημάτων στη μουσική μοντελοποιείται ως ένα πρόβλημα ταξινόμησης πολλαπλών ετικετών (multi-label) ταξινόμησης, προσφέροντας μια αξιολόγηση πάνω σε 4 γνωστούς αλγόριθμους ταξινόμησης από τους οποίους ο RAKEL αναδείχθηκε ως ο πιο αποτελεσματικός και με γενικό συμπέρασμα ότι αλγόριθμοι σαν και αυτόν μπορούν να χρησιμοποιηθούν για ταξινόμηση μεγάλων όγκων δεδομένων.

Στο άρθρο [10] οι συγγραφείς προτείνουν τον αλγόριθμο MLkNN (MultiLabel-kNN). Ο αλγόριθμος αυτός είναι η εξέλιξη του γνωστού αλγόριθμου οκνηρής εκμάθησης και ταξινόμησης μονής ετικέτας kNN (single-label), στην ταξινόμηση πολλαπλών ετικετών (multi-label). Βασίζεται σε στατιστικές πληροφορίες οι οποίες προέρχονται από τους γείτονες ενός στιγμιότυπου. Μετά από πειραματική αξιολόγηση ο αλγόριθμος αυτός ξεπέρασε σε απόδοση τους υπόλοιπους αλγορίθμους με τους οποίους συγκρίθηκε.

Στο άρθρο [11] παρουσιάζεται ο αλγόριθμος BP-MLL για ταξινόμηση πολλαπλών ετικετών (multi-label) ο οποίος εκτελεί ταξινόμηση με τη χρήση νευρωνικών δικτύων. Ο αλγόριθμος αυτός αποτελεί την multi-label έκδοση του αλγορίθμου Back propagation. Εφαρμογές σε προβλήματα που συσχετίζονται με text categorization έδειξαν ότι ο αλγόριθμος αυτός επιφέρει καλύτερα αποτελέσματα από άλλους γνωστούς αλγόριθμους. Ωστόσο η υπολογιστική πολυπλοκότητα και το κόστος εκμάθηση είναι μεγάλο αφού αυτό είναι ένα κοινό χαρακτηριστικό των αλγορίθμων που χρησιμοποιούν νευρωνικά δίκτυα όπως αυτός.

2.3 Υπάρχον λογισμικό για Multi-Label ταξινόμηση

Μελετώντας το πρόβλημα της ταξινόμησης αρχείων λογισμικού διερευνήσαμε αρχικά την περίπτωση χρήσης του Weka [12] σαν λύση. Το Weka είναι μια συλλογή αλγορίθμων μηχανικής μάθησης για εξόρυξη δεδομένων. Οι αλγόριθμοι που υποστηρίζει μπορούν είτε να εφαρμοστούν άμεσα σε ένα σύνολο δεδομένων, μέσα από τη γραφική διεπαφή χρήστη του Weka, ή ενσωματώνοντας τις κατάλληλες βιβλιοθήκες της Java στον κώδικά μας. Παρόλο που το Weka είναι κατάλληλο για την ανάπτυξη νέων συστημάτων μηχανικής μάθησης, επιλύει το πρόβλημα της απλής ταξινόμησης και όχι της πολλαπλής. Ωστόσο, ιδιαίτερα χρήσιμη μας φάνηκε η βιβλιοθήκη σε γλώσσα προγραμματισμού Java Mulan [8, 9]. Η βιβλιοθήκη αυτή παρέχει πληθώρα υλοποιημένων multi-label αλγορίθμων ταξινόμησης και θα αξιολογήσουμε 5 από τους αλγόριθμους αυτούς τους οποίους ξεχωρίσαμε από τις πιο πάνω προηγούμενες εργασίες για το λόγο ότι αντιμετωπίζουν το πρόβλημα της multi-label ταξινόμησης σε ταξινόμηση κειμένου. Οι αλγόριθμοι αυτοί είναι οι BR, LP, RAkEL, MLkNN και BPML [5, 6, 7, 10, 11]. Οι πρώτοι τρεις ανήκουν στην κατηγορία των *problem transformation* μεθόδων ενώ οι δύο εναπομείναντες στην κατηγορία των *algorithm adaptation* μεθόδων όπως φαίνεται και από τον πίνακα 1.2. Επίσης η βιβλιοθήκη Mulan προσφέρει ένα πλαίσιο αξιολόγησης αυτών των αλγορίθμων με πάρα πολλές μεθόδους αξιολόγησης οι οποίες θα χρησιμοποιηθούν ώστε να εξάγουμε ασφαλή αποτελέσματα για τις αποδόσεις των αλγορίθμων στο πρόβλημα της ταξινόμησης αρχείων λογισμικού.

2.4 Αναζήτηση στο Διαδίκτυο

Για τον εμπλουτισμό των αποτελεσμάτων της μηχανής αναζήτησης Minersoft, υπάρχει το Google Code[13] το οποίο είναι μηχανή αναζήτησης για κώδικα στο διαδίκτυο και μπορεί εύκολα να χρησιμοποιηθεί και να ενσωματωθεί σε κώδικα πολλών διαφορετικών γλωσσών προγραμματισμού μέσω του Google Search Api[14]. Παρόλο που αυτή η λύση φαίνεται ιδανική, έχει ένα σοβαρό μειονέκτημα, ο αριθμός των αναζητήσεων που μπορούν να εκτελεστούν από τη ίδια IP διεύθυνση είναι περιορισμένος.

Μια άλλη μηχανή αναζήτησης για λογισμικό είναι η Koders[15]. Μέσω της μηχανής αναζήτησης αυτής, ένας χρήστης μπορεί να εκτελέσει αναζήτηση πάνω από λογισμικό ανοιχτού κώδικα, το οποίο βρίσκεται στο διαδίκτυο. Ωστόσο δεν παρέχει κάποιες βιβλιοθήκες που θα μπορούσαν να χρησιμοποιηθούν για μια συνεχή, γρήγορη και αποτελεσματική χρήση του μέσα από ξένο κώδικα.

Τέλος, μία άλλη δημοφιλής μηχανή αναζήτησης είναι ή Yahoo και μέσω του Api της για αναζήτηση στο διαδίκτυο Yahoo Boss[16] προσφέρει ένα εύκολο και γρήγορο τρόπο ενσωμάτωσής του σε ξένο κώδικα. Επίσης δεν προβάλλει κανένα περιορισμό όσον αφορά τη συχνότητα των αναζητήσεων από την ίδια IP διεύθυνση.

Κεφάλαιο 3

Minersoft

Το θέμα της παρούσας διπλωματικής εργασίας είναι άρρηκτα συνδεδεμένο με τον τρόπο με τον οποίο εργάζεται το Minersoft, γι' αυτό είναι σημαντικό να γνωρίζουμε την αρχιτεκτονική του καλά ώστε να μπορούμε να εντάξουμε νέα στοιχεία σε αυτή. Σε αυτό το κεφάλαιο θα προβούμε σε μια ανασκόπηση των μοντέλων των υποδομών Grid, Cloud [27, 28]και του Minersoft καθώς και σε μια παρουσίαση της διεπαφής που υλοποιήθηκε για να εξυπηρετεί τη μηχανή αναζήτησης του Minersoft. Συγκεκριμένα, θα περιγράψουμε πώς κάποιος χρήστης μπορεί να κάνει αναζήτηση μέσω της διεπαφής, καθώς επίσης και ποιες επιλογές υπάρχουν για φιλτραρίσματα των αποτελεσμάτων. Η διεπαφή αυτή, όπως και οι πλείστες μηχανές αναζήτησης επιτρέπει στο χρήστη να εκτελέσει δυο διαφορετικά είδη αναζήτησης, την απλή αναζήτηση και την εξειδικευμένη αναζήτηση.

3.1 Ανασκόπηση πληροφοριακών δομών Πλέγματος (Grid) και Νεφέλης (Cloud)

3.1.1 Υποδομή Πλέγματος (Grid)

Το Grid είναι ένα υπολογιστικό δίκτυο μεγάλων διαστάσεων με πάρα πολλές χιλιάδες κατανεμημένες μηχανές σε πολλούς οργανισμούς. Οι μηχανές αυτές είναι ομαδοποιημένες σε αυτόνομους διαχειριστικούς τομείς οι οποίοι ονομάζονται *Grid sites* και επικοινωνούν μεταξύ τους σε γραμμές υψηλών ταχυτήτων. Οι οντότητες ενός Grid χωρίζονται σε τρεις κατηγορίες: επεξεργασίας, όπως ομάδες επεξεργαστών και πολυεπεξεργαστών, δικτύωσης όπως διακόπτες, δρομολογητές, πύλες κ.τ.λ. και αποθήκευσης, όπως βάσεις δεδομένων και συστοιχιών από δίσκους.

Η αρχιτεκτονική του Grid αποτελείται από τέσσερα επίπεδα:

- Fabric Layer: Επίπεδο στο οποίο παρέχονται οι πόροι που διαμοιράζονται στο Grid όπως χρόνο, αποθήκευση, δικτύωση κ.τ.λ.
- Core middleware: Επιτρέπει στο χρήστη να διαχειριστεί το προηγούμενο επίπεδο π.χ. επικοινωνία, απομακρυσμένη διαχείριση εργασιών, διαχείριση αποθηκευτικού χώρου.
- User-level middleware: Επιτρέπει μια ευκολότερη επικοινωνία με το προηγούμενο επίπεδο μέσω γραφικών διεπιφανειών.

- Grid application: Αυτό το επίπεδο επιτρέπει τη χρήση επιστημονικών εφαρμογών και εργαλείων στους χρήστες του Grid μέσα από το προηγούμενο επίπεδο.

EGEE Grid

Το EGEE (Enabling Grid for E-scienceE) Grid τέθηκε σε λειτουργία το Μάρτιο του 2004. Στόχος του είναι η παροχή σε ερευνητές τόσο του ακαδημαϊκού τομέα όσο και του βιομηχανικού τομέα, της δυνατότητας πρόσβασης σε υπολογιστικούς και αποθηκευτικούς χώρους μεγάλης κλίμακας ανεξαρτήτως τοποθεσίας. Τα κύρια χαρακτηριστικά του παρουσιάζονται στον παρακάτω πίνακα (3.1.1.1) .

ΥΠΟΔΟΜΗ EGEE			
140000 CPUS	20PB χωρητικότητα δίσκου	Υποστηρίζει 100000 Παράλληλες εργασίες	Είναι διαθέσιμος οποιαδήποτε στιγμή της μέρας

Πινακας 3.1.1.1: Χαρακτηριστικά EGEE.

Το EGEE χρησιμοποιεί το gLite middleware και περιλαμβάνει τις ακόλουθες κατηγορίες εργασιών και υπηρεσιών διαχείρισης δεδομένων:

- Computing Element (CE): Είναι ένα σύνολο από υπηρεσίες οι οποίες αντιπροσωπεύουν υπολογιστικούς πόρους που υπάρχουν σε κάποια τοποθεσία, όπως για παράδειγμα κάποιο cluster δηλαδή ενός συνόλου από μηχανές ή κάποιο computing farm. Το CE περιλαμβάνει το Grid Gate (GG), το οποίο λειτουργεί ως διεπαφή του cluster , το Local Resource Management System (LRMS) και το ίδιο το cluster στο οποίο εκτελούνται οι εργασίες.

- Workload Management System (WMS): Είναι υπεύθυνο για την αποδοχή των διάφορων διεργασιών που εκτελούνται από τους χρήστες, για τη διανομή των διεργασιών στους κατάλληλους πόρους (CE) καθώς επίσης και για την καταγραφή της κατάστασης εκτέλεσης (status) και εμφάνιση του αποτελέσματος. Οι υπηρεσίες του WMS τρέχουν στη μηχανή Resource Broker

(RB), ενώ για να μπορέσουν να εκτελεστούν οι διάφορες εργασίες χρησιμοποιείται η Job Description Language (JDL) η οποία είναι υπεύθυνη για την περιγραφή των διεργασιών προς εκτέλεση.

- Storage element (SE): Παρέχει πρόσβαση σε αποθηκευτικούς χώρους, από απλούς δίσκους που βρίσκονται σε κάποιους διακομιστές μέχρι μεγάλες συστοιχίες δίσκων μέσω διαφόρων πρωτοκόλλων επικοινωνίας όπως GSIFTP.

- EGEE site: Είναι μια ομάδα από τις προαναφερθείσες υπηρεσίες CE, WMS και SE. Παρέχεται πρόσβαση στους χρήστες της οντότητας αυτής, μέσω υπεύθυνων παροχής πρόσβασης όπως διάφορα ακαδημαϊκά ιδρύματα και οργανισμοί.

- Virtual organization (VO): Είναι μια δυναμική συλλογή ατόμων ή/και ιδρυμάτων που μοιράζονται προσυμφωνημένα κάποιους υπολογιστικούς πόρους, στους οποίους μπορούν να έχουν πρόσβαση εγγεγραμμένοι χρήστες του EGEE ώστε να χρησιμοποιήσουν εφαρμογές που χρειάζονται σε κάποιο VO όπου και να βρίσκεται.

3.1.2 Υποδομή Νεφέλης (Cloud)

Η έννοια του Cloud computing είναι ομώνυμη με τα μεγάλα κέντρα δεδομένων, τα οποία προσφέρουν μεγάλη υπολογιστική ισχύ καθώς και την ευελιξία του να πληρώνει κανείς μόνο για ότι χρησιμοποιεί. Οι υπολογιστικοί κόμβοι που ανήκουν στο Cloud καλούνται Cloud Virtual Servers. Μέσω αυτών οι χρήστες μπορούν να έχουν πρόσβαση σε αποθηκευτικούς χώρους, σε διάφορες πλατφόρμες λογισμικού ή να αναπτύξουν τις δικές τους εφαρμογές και μπορούν να επικοινωνήσουν με αυτούς μέσω πρωτοκόλλου SSH. Η αρχιτεκτονική του Cloud απαρτίζεται από τρία επίπεδα, το επίπεδο της υποδομής (infrastructure layer) στο οποίο υπάρχουν οι τεράστιοι αποθηκευτικοί χώροι και τα δίκτυα επικοινωνίας, το επίπεδο της πλατφόρμας (platform layer) το οποίο παρέχει ένα πιο ευέλικτο τρόπο διαχείρισης των πόρων που παρέχονται από το προηγούμενο επίπεδο και των εφαρμογών (application layer).

Amazon EC2 και S3

Το Amazon Elastic Computing Cloud αποτελεί μια συλλογή υπηρεσιών στο διαδίκτυο, οι οποίες παρέχουν υπολογιστική ισχύ και αποθηκευτικό χώρο τα οποία χαρακτηρίζονται με τον όρο ελαστικά και αυτό γιατί ο χρήστης ο οποίος ενοικιάζει αυτές τις υπηρεσίες μπορεί να τις φέρεις στις δικές του υπολογιστικές ανάγκες. Η Amazon παρέχει εικονικές μηχανές οι οποίες ονομάζονται *Amazon Machine Instances (AMIs)*. Κάθε τέτοια μηχανή αποτελείται από εικονικό δίσκο και περιλαμβάνει λειτουργικό σύστημα καθώς και κάποιο ήδη εγκατεστημένο λογισμικό. Ήδη αρχικοποιημένα AMIs ονομάζονται *server instances.* Το *Amazon S3 (Amazon Simple Storage Service)* χρησιμοποιείται για αποθήκευση δεδομένων από τους χρήστες διαμέσου μιας διεπαφής.

Rackspace cloud

Το Rackspace cloud είναι ένα σύστημα το οποίο ειδικεύεται σε υπηρεσίες cloud hosting. Παρέχει ακριβώς τις ίδιες υπηρεσίες με αυτές της Amazon με μόνη διαφορά ότι είναι πολύ πιο εύκολο στη χρήση. Οι τρεις υπηρεσίες που παρέχονται από το Rackspace cloud είναι το Cloud Sites, το Cloud Servers και το Cloud Files. Το Cloud Sites έχει ως κύρια λειτουργία αυτής της υπηρεσίας τη παροχή υπηρεσιών διαδικτύου. Η υπηρεσία αυτή έχει την ικανότητα κλιμάκωσης αυτόματα όποτε είναι απαραίτητο. Ο χρήστης αυτής της υπηρεσίας μπορεί να εγκαταστήσει οποιαδήποτε εφαρμογή. Το Cloud Servers δίνει στο χρήστη την ικανότητα να επιλέξει μέσα από ένα πλήθος server images ποιο από αυτά επιθυμεί να χρησιμοποιήσει. Στην υπηρεσία αυτή ο χρήστης πληρώνει βάσει των ωρών που κάνει χρήση των εικόνων που επέλεξε καθώς επίσης και με το bandwidth που χρησιμοποιήθηκε. Τέλος, το Cloud Files είναι η υπηρεσία που χρησιμοποιείται ως ένας διαδικτυακός αποθηκευτικός χώρος και όπως στο Amazon S3, τα volumes μπορούν να ενωθούν με τους cloud servers με αποτέλεσμα την αύξηση του μεγέθους του χώρου αποθήκευσης.

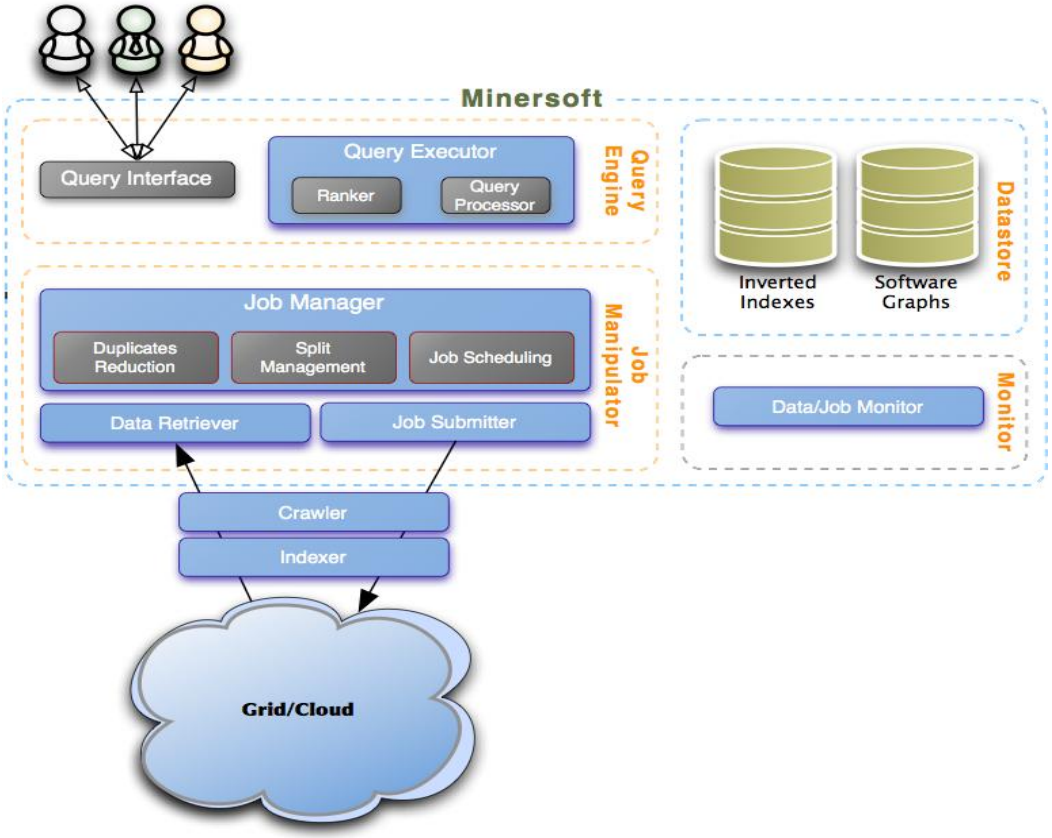
Στον πίνακα (3.1.2.1) που ακολουθεί γίνεται μια σύγκριση των δύο πληροφοριακών υποδομών Grid και Cloud.

	Παροχέας υπολογιστικών πόρων	Υπολογιστικοί πόροι	Αρχιτεκτονική υποδομής	Υποβολή εργασιών για εκτέλεση	Περιορισμοί σε χρόνο και υπολογιστική ισχύ
Grid	Παροχέας υπολογιστικών πόρων Grid	Grid site	Κατανεμημένη	Grid middleware	Ναι
Cloud	Παροχέας υπολογιστικών πόρων Cloud	Cloud Virtual Servers	Κεντρικοποιημένη	SSH, Web services, batch systems	Όχι

Πίνακας 3.1.2.1: Χαρακτηριστικά Grid και Cloud

3.2 Ανασκόπηση Minersoft

Ανασκόπηση αρχιτεκτονικής

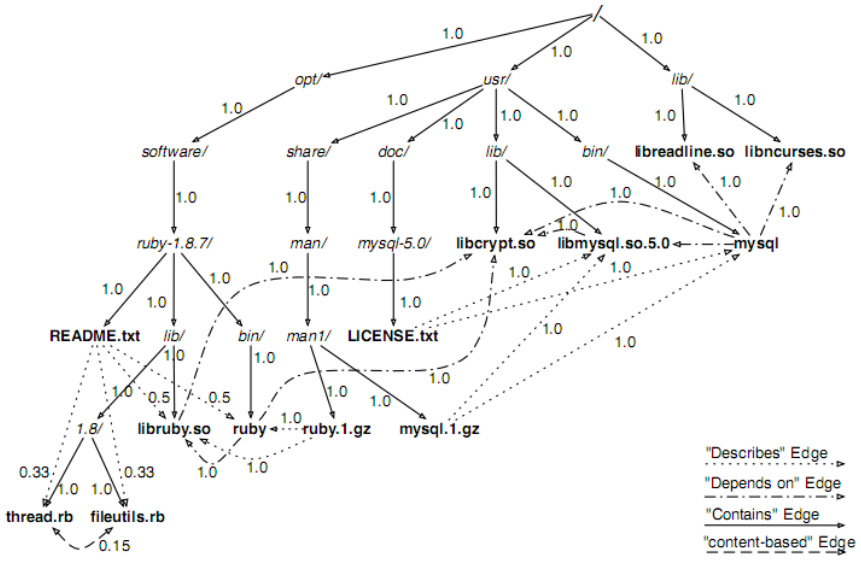


Σχήμα 3.2.1: Αρχιτεκτονική του MinerSoft

Το Minersoft έχει μια MAP-Reduce αρχιτεκτονική η οποία αναπαρίσταται στο σχήμα (3.2.1) ενώ η ανάκτηση των πληροφοριών (crawling) καθώς και η δημιουργία των ανεστραμμένων ευρετηρίων (indexing) εκτελείται παράλληλα και κατανεμημένα. Στο πιο πάνω σχήμα διακρίνουμε τις εξής οντότητες οι οποίες διαδραματίζουν σημαντικό ρόλο στη λειτουργία του Minersoft. Τον Job manipulator [1] ο οποίος αποτελείται από τρεις άλλες οντότητες τον Job manager, τον Job submitter και τον Job retriever και οι οποίοι έχουν ως έργο την ανάθεση του crawling, του indexing και της ανάκτησης των αποτελεσμάτων αντίστοιχα. Επίσης, υπεύθυνη για την επιτήρηση της ανάθεσης των εργασιών είναι η οντότητα του Monitor. Στο Datastore αποθηκεύονται όλα τα δημιουργημένα ανεστραμμένα ευρετήρια με σκοπό την τελική τους χρήση μέσω της μηχανής αναζήτησης από τους χρήστες.

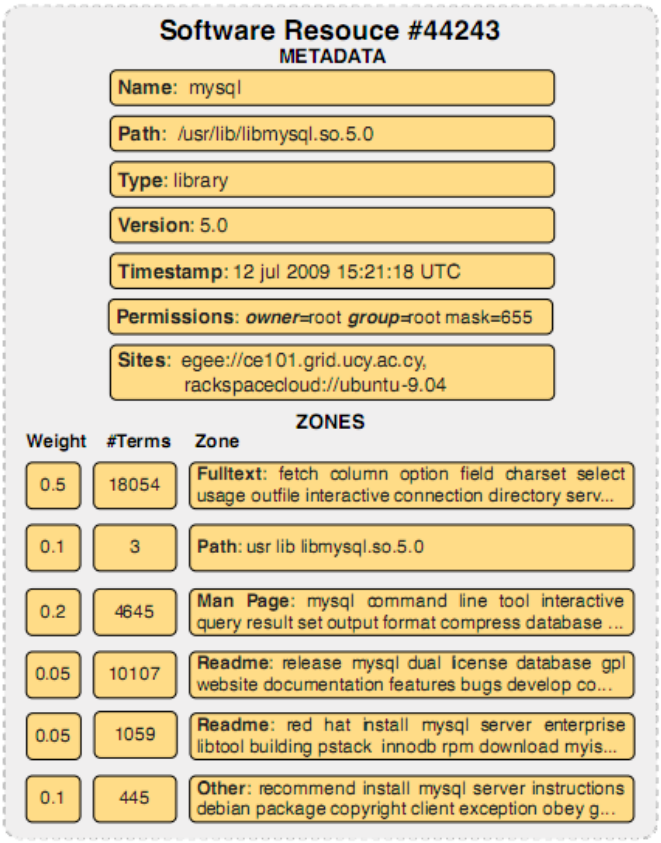
Ανασκόπηση κατασκευής του Software Graph του MinerSoft

Κατά τη διαδικασία της ανάκτησης πληροφοριών από ένα κόμβο στο Grid ή στο Cloud, ο crawler του Minersoft ανακτά τις πληροφορίες αυτές από το σύστημα αρχείων της μηχανής που βρίσκεται δημιουργώντας ένα **File System Tree** [1], αναγνωρίζοντας αρχεία που συσχετίζονται με λογισμικό και τα κατατάσσει σε διαφορετικές κατηγορίες (π.χ. binaries, libraries, documentation). Ακολούθως αποκόπτονται από το δέντρο αυτό αρχεία τα οποία δεν περιέχουν σημαντικές πληροφορίες (π.χ. log files) και ανιχνεύονται οποιεσδήποτε δομικές συσχετίσεις υπάρχουν μεταξύ των κόμβων του δέντρου. Το αποτέλεσμα είναι η δημιουργία της πρώτης μορφής του **Software Graph** [1]. Ο **Software Graph** – σχήμα (3.2.2) – είναι ένας γράφος G(V,E) με βάρη πλούσιος σε μεταδεδομένα, όπου κάθε κορυφή **V** του γράφου αυτού είναι ένα αρχείο λογισμικού ή ένας κατάλογος του δέντρου του συστήματος αρχείων του κόμβου. Οι ακμές **E** αντιπροσωπεύουν τις σχέσεις μεταξύ των κορυφών και μπορεί να είναι δομικές ή και περιεχομένου. Οι δομικές σχέσεις προκύπτουν μεταξύ κορυφών του γράφου όπου η μια κορυφή αναπαριστά ένα αρχείο λογισμικού και η άλλη ένα κατάλογο, ενώ οι σχέσεις περιεχομένου υπάρχουν μεταξύ αρχείων με ομοιότητα στο κείμενό τους. Ο γράφος αυτός είναι τυποποιημένος γιατί κάθε κορυφή του ανήκει σε μια κατηγορία τύπων αρχείου.



Σχήμα 3.2.2: Software Graph

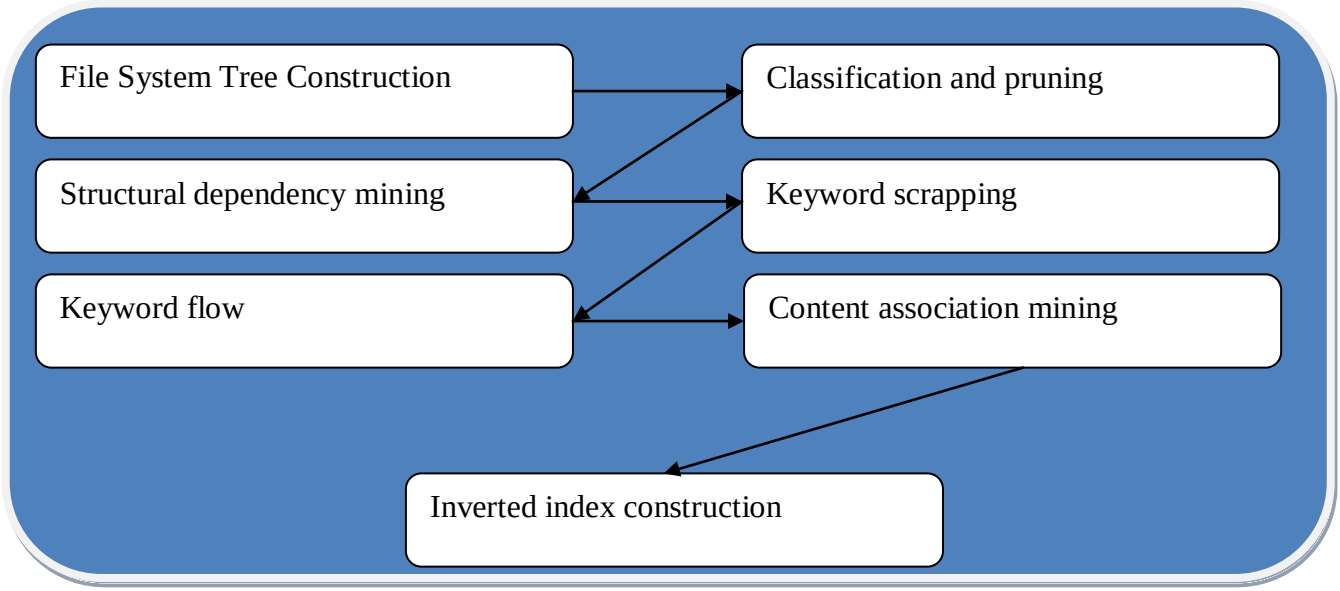
Επιπλέον, κάθε κορυφή περιέχει μεταδεδομένα τα οποία περιγράφουν το περιεχόμενο και το πλαίσιο στο οποίο ανήκουν, αλλά και ένα αριθμό από ζώνες σε αρχεία κειμένου τα οποία συσχετίζονται με τα αρχεία λογισμικού. Οι ζώνες αυτές *zones: zone$_i$(v), i = 1, ... , z_v* ανάλογα με τον βαθμό συσχέτισης του κάθε αρχείου παίρνουν ένα βάρος και έχουν συνολικό άθροισμα του βαρών 1. Οι ζώνες αυτές είναι σημαντικές γιατί εμπλουτίζουν τα αρχεία του γράφου με όρους οι οποίοι καθορίζουν την σημαντικότητα ενός αρχείου την ώρα της αναζήτησης από τους χρήστες. Αυτά τα δεδομένα αναπαριστώνται στο σχήμα (3.2.3).



Σχήμα 3.2.3: Μεταδεδομένα των κόμβων του γράφου

Ανασκόπηση ροών εργασιών (workflow) του MinerSoft

Το MinerSoft αποτελείται από συνολικά 7 ροές δεδομένων, μέρος των οποίων συμπεριλαμβάνεται στην ανασκόπηση αρχιτεκτονικής και στην ανασκόπηση κατασκευής του Software Graph του MinerSoft. Αυτές οι ροές δεδομένων αναπαριστώνται στο σχήμα (3.2.4).



Σχήμα 3.2.4: Minersoft workflow.

Στο ***Classification and pruning*** αποκόπτονται από τα αρχεία τα προθέματα και τα επιθέματα των ονομάτων των αρχείων και τα κανονικοποιημένα πλέον ονόματα αποθηκεύονται σαν μεταδεδομένα στο File System Tree. Ενώ χρησιμοποιούνται διάφορες ευριστικές μέθοδοι για την ταξινόμηση των κορυφών σε κατηγορίες όπως binaries, soutce κ.τ.λ.

Στο μέρος του ***Structural dependency mining*** το Minersoft ψάχνει για δομικές σχέσεις μεταξύ αρχείων και ενώνοντας τα αρχεία αυτά με ακμές δημιουργεί την πρώτη μορφή του γράφου όπως αναφέραμε προηγουμένως .

Το ***Keyword scraping*** είναι η διαδικασία όπου αναλύεται το περιεχόμενο αυτών των κορυφών. Απαλείφονται κοινές λέξεις, γίνεται στελέχωση και μένουν μόνο οι χρήσιμες πληροφορίες που θα αποτελέσουν τις λέξεις – κλειδιά και τις πληροφορίες των ανεστραμμένων ευρετηρίων.

Στη φάση του ***Keyword flow***, το Minersoft εμπλουτίζει τις πληροφορίες για αναζήτηση των αρχείων τύπου library, source executables με πληροφορίες που βρίσκονται μέσα σε manuals και readme files.

Στο μέρος του ***Content association mining*** επιπλέον συσχετίσεις με ακμές γίνονται μεταξύ αρχείων κώδικα με όμοια περιεχόμενα. Η σύγκριση γίνεται με cosine similarity.

Τέλος, δημιουργούνται τα ανεστραμμένα ευρετήρια για να υποστηρίξουν full-text αναζήτηση. Τα ανεστραμμένα ευρετήρια αποτελούνται από όρους, όπου κάθε όρος αντιστοιχεί σε ένα "posting list" με δείκτες στα αρχεία λογισμικού που περιέχουν τον όρο αυτό. Για την μελέτη της δημιουργίας των ανεστραμμένων ευρετηρίων, χρησιμοποιήσαμε το Minersoft και δημιουργήσαμε 30 ανεστραμμένα ευρετήρια – πίνακας (3.2.1). Τα 20 από αυτά είναι Amazon Machine Instances (AMIs) τα οποία προέρχονται από το EC2 της Amazon και τα υπόλοιπα 10 Cloud Virtual Servers τα οποία προέρχονται από το Rackspace cloud.

Cloud	Virtual Machine	Index Size (MB)
Amazon EC2	ami-00e41469	402
Amazon EC2	ami-0f42a966	954
Amazon EC2	ami-2a5fba43	553
Amazon EC2	ami-005aab69	1,333
Amazon EC2	ami-38c33651	446
Amazon EC2	ami-044cba6d	402
Amazon EC2	ami-74f0061d	454
Amazon EC2	ami-86db39ef	554
Amazon EC2	ami-0201f16b	319
Amazon EC2	ami-309c6d59	1,143
Amazon EC2	ami-0354b96a	214
Amazon EC2	ami-548c783d	311
Amazon EC2	ami-2547a34c	515
Amazon EC2	ami-8040aae9	968
Amazon EC2	ami-aa30c7c3	1521
Amazon EC2	ami-b209f8db	480
Amazon EC2	ami-d8f005b1	446
Amazon EC2	ami-f21aff9b	553
Amazon EC2	ami-f61dfd9f	534
Amazon EC2	ami-fefd0d97	481
Rackspace	server CentOS 5.4	243
Rackspace	server CentOS 5.5	243
Rackspace	server Debian 5.0 (Lenny)	177
Rackspace	server Fedora 13 (Goddard)	267

Rackspace	server Fedora 14 (Laughlin)	415
Rackspace	server Red Hat Enterprise Linux 5.4	375
Rackspace	server Red Hat Enterprise Linux 5.5	377
Rackspace	server Ubuntu 9.10 (Karmic Koala)	313
Rackspace	server Ubuntu 10.04 LTS (Lucid Lynx)	321
Rackspace	server Ubuntu 10.10 (Maverick Meerkat)	258

Πίνακας 3.2.1: Δημιουργημένα ευρετήρια

3.3 Γραφική διεπαφή χρήστη της μηχανής αναζήτησης Minersoft

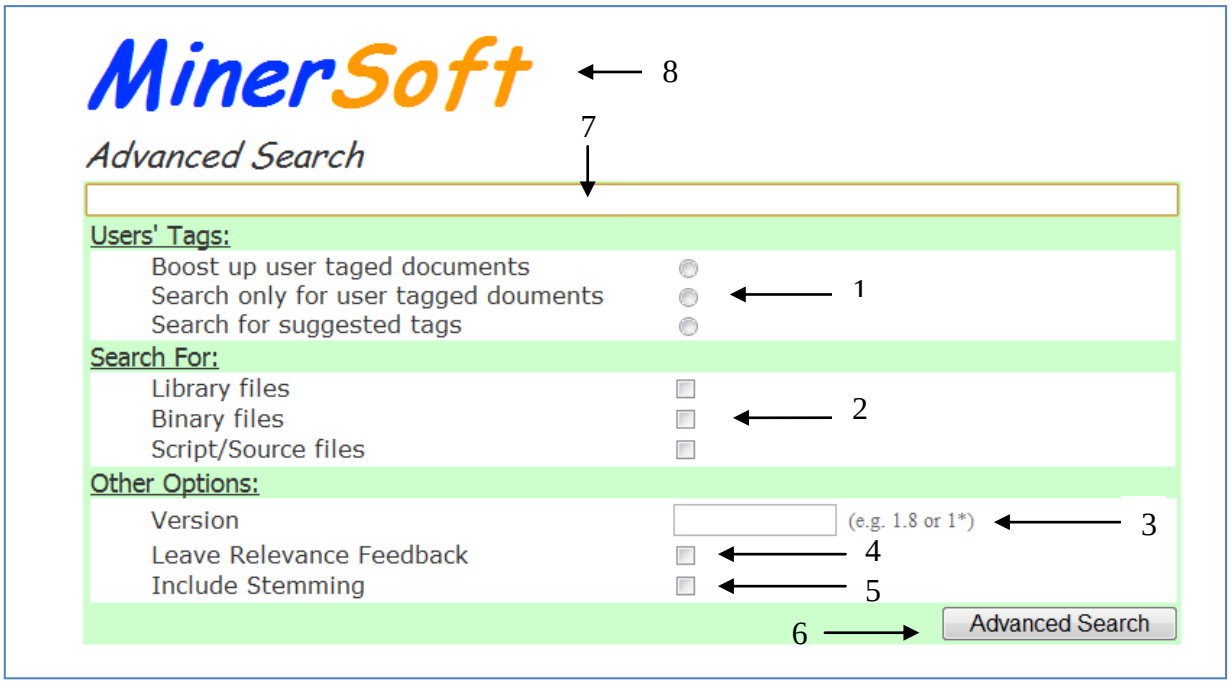
3.3.1 Απλή αναζήτηση

Η απλή αναζήτηση γίνεται μέσω της κεντρικής ιστοσελίδας της μηχανής αναζήτησης του Minersoft εικόνα (3.3.1.1) . Συγκεκριμένα, όπως και σε κάθε μηχανή αναζήτησης, ο χρήστης πρέπει να εισάγει τις λέξεις – κλειδιά για το λογισμικό που επιθυμεί να αναζητήσει, στο πεδίο που παρουσιάζεται με το βέλος (1) στην εικόνα (3.3.1.1) και ακολούθως η αναζήτηση και η εμφάνιση των αποτελεσμάτων εκτελείται με το πάτημα του πλήκτρου Search (βέλος 2). Με την επιλογή του συνδέσμου στο (βέλος 3) ο χρήστης μεταφέρεται στην ιστοσελίδα της εξειδικευμένης η οποία φαίνεται στην εικόνα (3.3.2.1) .

Εικόνα 3.3.1.1: Κεντρική οθόνη διεπαφής του MinerSoft

3.3.2 Εξειδικευμένη αναζήτηση

Η ενεργοποίηση της επιλογής αυτής γίνεται με το πάτημα του συνδέσμου Advanced Search που φαίνεται με το (βέλος 3) στην εικόνα (3.3.1.1). Αφού ο χρήστης ενεργοποιήσει την επιλογή αυτή παρουσιάζονται στο χρήστη οι επιλογές για την εξειδικευμένη αναζήτηση εικόνα (3.3.2.1).



Εικόνα 3.3.2.1: Οθόνη επιλογών εξειδικευμένης αναζήτησης

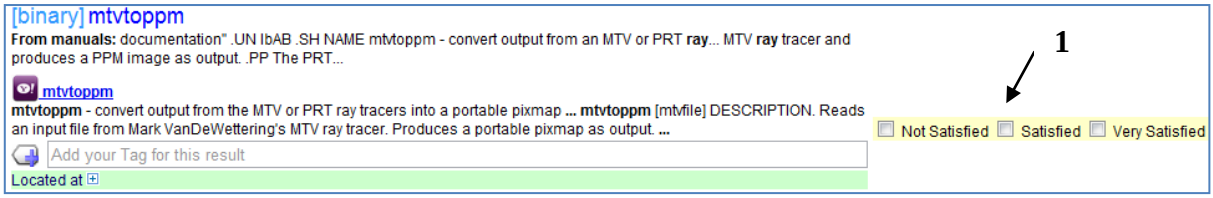
Στην εξειδικευμένη αναζήτηση υπάρχουν τρεις κατηγορίες επιλογών. Η πρώτη κατηγορία αφορά επιλογές αναζήτησης που σχετίζονται με τις ετικέτες που δίνουν οι χρήστες, η δεύτερη κατηγορία αφορά τις επιλογές των αρχείων λογισμικού που προσφέρονται στους χρήστες και η τρίτη κατηγορία αφορά άλλες επιλογές στην αναζήτηση.

Συγκεκριμένα, όπως φαίνεται στην εικόνα (3.3.2.1) στο (βέλος 1) για την πρώτη κατηγορία των επιλογών που εμφανίζονται αν ο χρήστης επιλέξει την πρώτη επιλογή στην εικόνα και αφού εκτελέσει μια αναζήτηση θα του εμφανιστούν πρώτα τα αποτελέσματα τα οποία τους έχουν προστεθεί ετικέτες από τους χρήστες. Επιλέγοντας την δεύτερη επιλογή στο (βέλος 1) και αφού εκτελέσει μια αναζήτηση, του εμφανίζονται μόνο τα αποτελέσματα τα οποία τους έχουν προστεθεί ετικέτες από τους χρήστες, στην τρίτη επιλογή θα εκτελείται αναζήτηση ακριβώς με τον ίδιο τρόπο με πριν με μόνη διαφορά ότι η αναζήτηση θα εκτελείται πάνω στις ετικέτες που προβλέφθηκαν από ο σύστημα με τον τρόπο που θα αναλύσουμε στον Κεφάλαιο 4.

Ο χρήστης έχει τη δυνατότητα να μην εκτελέσει αναζήτηση υποχρεωτικά και για τις τρείς κατηγορίες αρχείων (binary, library και Script/Source). Επομένως μέσω της εξειδικευμένης αναζήτησης πρέπει να επιλέξει για ποιες κατηγορίες αρχείων θα γίνει αναζήτηση ενεργοποιώντας τα αντίστοιχα check box που φαίνονται με το (βέλος 2).

Μέσω αυτής της οθόνης ο χρήστης μπορεί να καθορίσει την έκδοση του λογισμικού για το οποίο πρόκειται να εκτελέσει αναζήτηση πληκτρολογώντας τον αριθμό της έκδοσης στο πεδίο Version (βέλος 3). Επιπλέον, Το πεδίο αυτό υποστηρίζει αναζήτηση με είσοδο χαρακτήρα μπαλαντέρ. Ακόμη, ενεργοποιώντας την επιλογή Include Stemming (βέλος 5) ο όρος αναζήτησης ακολουθεί τους ίδιους κανόνες στελέχωσης κειμένου με αυτούς που ακολουθήθηκαν για την δημιουργία των ανεστραμμένων ευρετηρίων και η αναζήτηση διεξάγεται στους όρους που αποθηκεύτηκαν, αφότου στελέχωση κειμένου εφαρμόστηκε πάνω τους.

Με την επιλογή Leave Relevance Feedback (βέλος 4) δίπλα από κάθε αποτέλεσμα εμφανίζονται και τρείς επιλογές – εικόνα (3.3.2.2) (βέλος 1) – όπου ο χρήστης μπορεί να αναφέρει κατά πόσο το συγκεκριμένο αποτέλεσμα είναι ικανοποιητικό ή όχι.

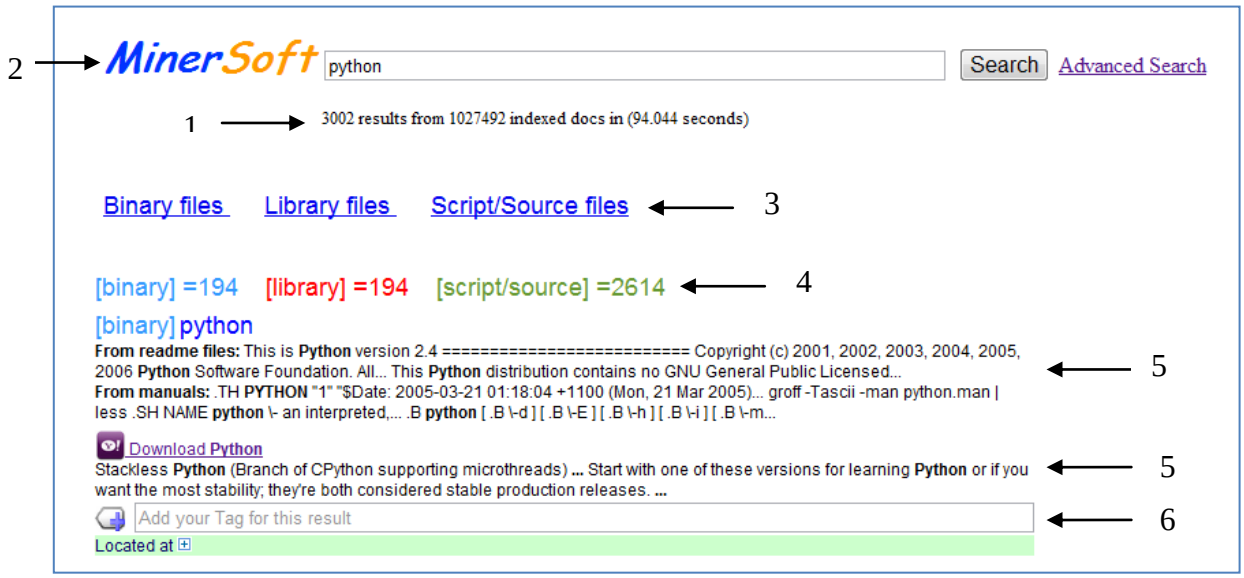


Εικόνα 3.3.2.2: Επιλογή Relevance feedback – Εξειδικευμένη αναζήτηση

Στη συνέχεια, αφού ο χρήστης καθορίσει τις επιλογές της εξειδικευμένης αναζήτησης πληκτρολογεί τον όρο αναζήτησης στο πεδίο με το (βέλος 7) της εικόνας (3.3.2.1) και με το πάτημα του πλήκτρου Advance Search (βέλος 4) εκτελείται εξειδικευμένη αναζήτηση με βάση τις επιλογές που έχουν καθορισθεί.

Ο χρήστης μπορεί να επιστρέψει στην κεντρική οθόνη της διεπαφής για απλή αναζήτηση με το πάτημα της εικόνας “MinerSoft” με το (βέλος 8) εικόνα (3.3.2.1).

3.3.3 Ιστοσελίδα αποτελεσμάτων



Εικόνα 3.3.3.1: Ιστοσελίδα αποτελεσμάτων αναζήτησης

Αφού ο χρήστης επιλέξει το είδος αναζήτησης που επιθυμεί (απλή ή εξειδικευμένη) και εκτελέσει την αναζήτηση θα εμφανιστεί η ιστοσελίδα αποτελεσμάτων αναζήτησης – εικόνα (3.3.3.1). Τα αποτελέσματα στην εικόνα αυτή προέκυψαν αφού ο χρήστης εκτέλεσε απλή αναζήτηση για την λέξη κλειδί ***python***.

Στο πάνω μέρος της οθόνης (βέλος 1) παρουσιάζονται οι πληροφορίες αναζήτησης και συγκεκριμένα το πλήθος των αποτελεσμάτων που επιστράφηκαν, το συνολικό μέγεθος των αρχείων που περιέχουν τα ανεστραμμένα ευρετήρια καθώς επίσης και ο χρόνος που χρειάστηκε για την συγκεκριμένη αναζήτηση και την επιστροφή των αποτελεσμάτων. Ακόμη, με την επιστροφή των αποτελεσμάτων εμφανίζονται στατιστικές πληροφορίες οι οποίες αφορούν των αριθμό των αρχείων λογισμικού που επεστράφησαν για τον κάθε ένα από τους τρείς τύπους αρχείων λογισμικού του MinerSoft δηλαδή Binary, Library και Script/Source. Όπως και στην ιστοσελίδα της εξειδικευμένης αναζήτησης, ο χρήστης με το πάτημα της εικόνας "MinerSoft" (βέλος 2) έχει την δυνατότητα να μεταβεί στην αρχική οθόνη της διεπαφής. Η εκτέλεση μιας νέας αναζήτησης μπορεί να διεξαχθεί και απευθείας από την υφιστάμενη ιστοσελίδα, καθώς και η επιλογή για εξειδικευμένη αναζήτηση.

Επιπρόσθετα, στην περίπτωση που ο χρήστης επιλέξει κάποιον από τους τρεις συνδέσμους που αντιστοιχούν στα Binary, Library και Script/Source files (βέλος 3), η μηχανή αναζήτησης του επιστρέφει μόνο τα αποτελέσματα του αντίστοιχου τύπου όπως φαίνεται και στην εικόνα (3.3.3.2).

Εικόνα 3.3.3.2: Ιστοσελίδα αποτελεσμάτων αναζήτησης – Επιστροφή μόνο Binary files

3.3.4 Περίληψη αποτελέσματος

Όλες οι καλές μηχανές αναζήτησης επιστρέφουν μαζί με κάθε αποτέλεσμα μια μικρή περίληψη η οποία περιγράφει συνοπτικά το συγκεκριμένο αποτέλεσμα. Στην αντίθετη περίπτωση ένας χρήστης θα δυσκολευόταν πολύ να διαλέξει ποιο από τα προβαλλόμενα αποτελέσματα τον ενδιαφέρει. Επομένως ένα τέτοιο χαρακτηριστικό δεν μπορεί να απουσιάζει από την μηχανή αναζήτησης του Minersoft. Υπάρχουν δύο γενικές κατηγορίες περιλήψεων οι στατικές περιλήψεις και οι δυναμικές. Οι στατικές περιλήψεις συνήθως υπακούουν σε απλούς ευριστικούς κανόνες όπως το να επιστρέφουν τους πρώτους 50 χαρακτήρες ενός αρχείου ή πιο περίπλοκους που αποσκοπούν στην εξαγωγή των σημαντικότερων στοιχείων ενός κειμένου. Οι δυναμικές περιλήψεις είναι οι περιλήψεις που δημιουργούνται δυναμικά και ταυτόχρονα με την επιστροφή των αποτελεσμάτων στο χρήστη, εφαρμόζοντας την περίληψη στις λέξεις κλειδιά που έδωσε ο χρήστης. Οι πιο συνήθεις τεχνικές για δημιουργία αυτού του είδους της περίληψης είναι η αναζήτησης της λέξης – κλειδί του

χρήστη σε ένα προκαθορισμένο αριθμό χαρακτήρων δεξιά και αριστερά από την λέξη – κλειδί (window).

Για να μπορεί όμως η μηχανή αναζήτησης να υλοποιεί οποιαδήποτε από τις δύο μεθόδους απαραίτητη προϋπόθεση είναι η ύπαρξη της αυθεντικής πληροφορίας όπως αυτή υπάρχει μέσα στα αρχεία κειμένου. Για το σκοπό αυτό τροποποιήσαμε μια από τις φάσεις δημιουργίας ανεστραμμένων ευρετηρίων του Minersoft προσθέτοντας της, επιπλέον εργασίες. Η φάση αυτή είναι η φάση του content association mining η οποία περιγράφηκε συνοπτικά στο Κεφάλαιο 3.2 και κατά τη διάρκεια της οποίας τα περιεχόμενα των αρχείων κειμένου τα οποία με κάποιους ευριστικούς αλγορίθμους βρέθηκαν να συσχετίζονται με τα αρχεία λογισμικού, αναλύονται με σκοπό να προσφέρουν περισσότερη πληροφορία σε μια αναζήτηση. Στη φάση αυτή αφού οι αλγόριθμοι βρήκαν τις συσχετίσεις για κάθε αρχείο λογισμικού αποθηκεύουν την πληροφορία όχι μόνο στην επεξεργασμένη αλλά και στην μη επεξεργασμένη της μορφή. Για τη δημιουργία λοιπόν των περιλήψεων που συνοδεύουν το κάθε αποτέλεσμα, χρησιμοποιήσαμε συνδυασμό της δυναμικής και της στατικής μεθόδου για δημιουργία περιλήψεων μέσω μιας απλής διαδικασίας. Αυτή η επιλογή προέκυψε από την ανάγκη για επιστροφή πληροφορίας όχι μόνο για αποτελέσματα των οποίων μέσα στο αυθεντικό τους κείμενο υπάρχει η λέξη – κλειδί που αναζητεί ο χρήστης αλλά και σε αποτελέσματα στα οποία παρόλο ότι υπάρχουν συσχετισμένα αρχεία κειμένου με αυτά η λέξη – κλειδί απουσιάζει από μέσα.

Για τη δημιουργία δυναμικής περίληψης βασισμένη στα ερωτήματα χρησιμοποιήθηκε η γνωστή κλάση Highlighter από την βιβλιοθήκη Lucene [17]. Ελέγχοντας πρώτα αν βρήκε επαρκείς πληροφορίες μέσα στα συσχετισμένα readme files, manuals ή και στο ίδιο το περιεχόμενο επιστρέφει από ποια από τις τρείς πιο πάνω κατηγορίες ανήκει και την περίληψη αυτή. Αν όμως οι λέξεις – κλειδιά δεν εντοπίζονται πουθενά τότε ο Highlighter δε θα επιστρέφει τίποτα.

Επομένως, στην περίπτωση που θα συμβεί αυτό, δημιουργούμε ένα άλλο είδος γενικής περίληψης εντοπίζοντας τις πιο σημαντικές προτάσεις από κάθε readme file ή manual το οποίο είναι συσχετισμένο με το αρχείο λογισμικού και μετά επιστρέφουμε μερικές από αυτές ώστε να δίνεται ακόμη και σε αυτή την περίπτωση μια περιγραφή. Η

σημαντικότητα των προτάσεων αυτών υπολογίζεται συγκρίνοντας τα βάρη TF-IDF καθώς και του μεγέθους τους, η οποία είναι μια συνήθης προσέγγιση όπως αναφέρεται και στο [34].

Συγκεκριμένα για κάθε πρόταση υπολογίστηκε το ακόλουθο score :

N: αριθμός όλων των προτάσεων, όλων των αρχείων που συσχετίζονται με ένα αρχείο λογισμικού.

df: αριθμός όλων των προτάσεων που περιέχουν τη λέξη αυτή.

tf: αριθμός εμφανίσεως της λέξης αυτής στην πρόταση.

ν: αριθμός λέξεων που έχει μια πρόταση.

$$\text{Score1} = \sum_{i=0}^{\nu} \text{tf} * \log10(\frac{\text{N}}{\text{df}}) \,/\max\{\sum_{i=0}^{\nu} \text{tf} * \log10(\frac{\text{N}}{\text{df}}))\}$$

Score2 = ν / max(ν)

Total score = Score1 + Score2

Αφού υπολογιστεί για όλες τις προτάσεις, αυτές ταξινομούνται σε φθίνουσα σειρά και επιλέγονται οι 5 οι οποίες συγκέντρωσαν πιο ψηλή βαθμολογία και οι οποίες αποτελούν την περίληψη που αφορά στο συγκεκριμένο αποτέλεσμα του Minersoft.

3.3.5 Συσχετισμός με πηγές από το διαδίκτυο

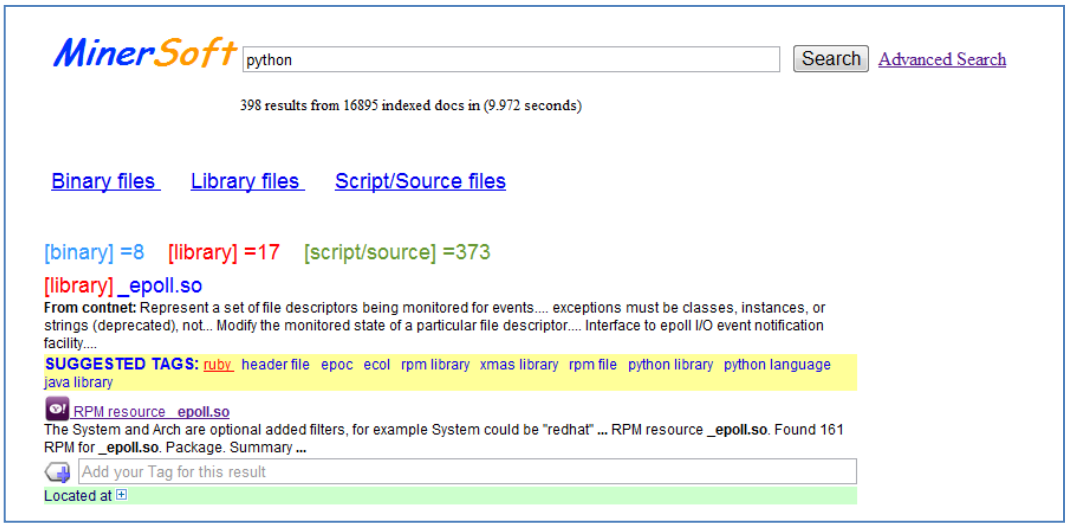
Εκτός από την περίληψη που βασίζεται σε πληροφορίες που ανακτήθηκαν από το σύστημα αρχείων των κόμβων του Cloud, αναπτύξαμε μια απλή διαδικασία για συνδυασμό του κάθε ενός αποτελέσματος της μηχανής αναζήτησης Minersoft με πληροφορίες από το διαδίκτυο.

Κάθε ένα αποτέλεσμα του Minersoft αποστέλλεται στην Yahoo και εκτελείται αναζήτηση βάση ονόματος. Ακολούθως στα 20 πρώτα αποτελέσματα της μηχανής αναζήτησης Yahoo εφαρμόζονται κάποιοι ευριστικοί κανόνες για την επιλογή του πιο σχετικού από αυτά τα 20 αποτελέσματα. Για την υλοποίηση αυτής της διαδικασίας και την επιστροφή των αποτελεσμάτων από την αναζήτηση στο διαδίκτυο χρησιμοποιήθηκε το **search API** της Yahoo, **BOSS** [2] το οποίο μας επιτρέπει την επιστροφή του συνδέσμου, του τίτλου και την περιγραφή ενός αποτελέσματος τα οποία και βλέπουμε στην εικόνα (3.3.3.1) στο (βέλος 5). Η περιγραφή της διαδικασίας αυτής,

καθώς και περισσότερες λεπτομέρειες για τους ευριστικούς κανόνες που χρησιμοποιήθηκαν σε αυτήν, περιλαμβάνονται στο Κεφάλαιο 5.

3.3.6 Πρόσθεση ετικέτας από χρήστη σε ένα αρχείο λογισμικού

Η πρόσθεση ετικέτας (tag), είναι μια διαδικασία κατά την οποία ο χρήστης εισάγει μικρή ποσότητα κειμένου που πάντα κατά τη δική του κρίση, πιστεύει ότι περιγράφει καλύτερα το αρχείο λογισμικού – εικόνα (3.3.3.1) (βέλος 6). Αυτή η μικρή ποσότητα κειμένου αποθηκεύεται στα ανεστραμμένα ευρετήρια και μέσω μιας διαδικασίας που θα αναλυθεί με λεπτομέρεια στο Κεφάλαιο 4, τα ανεστραμμένα ευρετήρια ανανεώνονται συνεχώς. Μέσα σε αυτή την αδιάκοπη διαδικασία ένας ταξινομητής ο οποίος εκπαιδεύεται βάση των ετικετών των χρηστών προτείνει κατηγορίες και για τα υπόλοιπα αρχεία του συστήματος. Οι κατηγορίες που προβλέφθηκαν για κάθε αρχείο λογισμικού αποθηκεύονται σε μια νέα ζώνη στο ανεστραμμένο ευρετήριο έτσι ώστε με το τέλος μιας αναζήτησης και την επιστροφή των αποτελεσμάτων, τα αρχεία λογισμικού που επιστρέφονται να περιέχουν και τις ετικέτες στις οποίες πιθανώς να ανήκουν. Επιπλέον, αν κάποιος χρήστης επιθυμεί να του επιστραφούν τα αρχεία για κάποια συγκεκριμένη ετικέτα από τα αποτελέσματα, τότε πατώντας απλά πάνω της γίνεται μια νέα αναζήτηση μόνο στη ζώνη των ετικετών ώστε να επιστραφούν μόνο αρχεία λογισμικού που περιγράφονται από αυτή την ετικέτα. Η διαδικασία αυτή απεικονίζεται στην εικόνα (3.3.6.1).

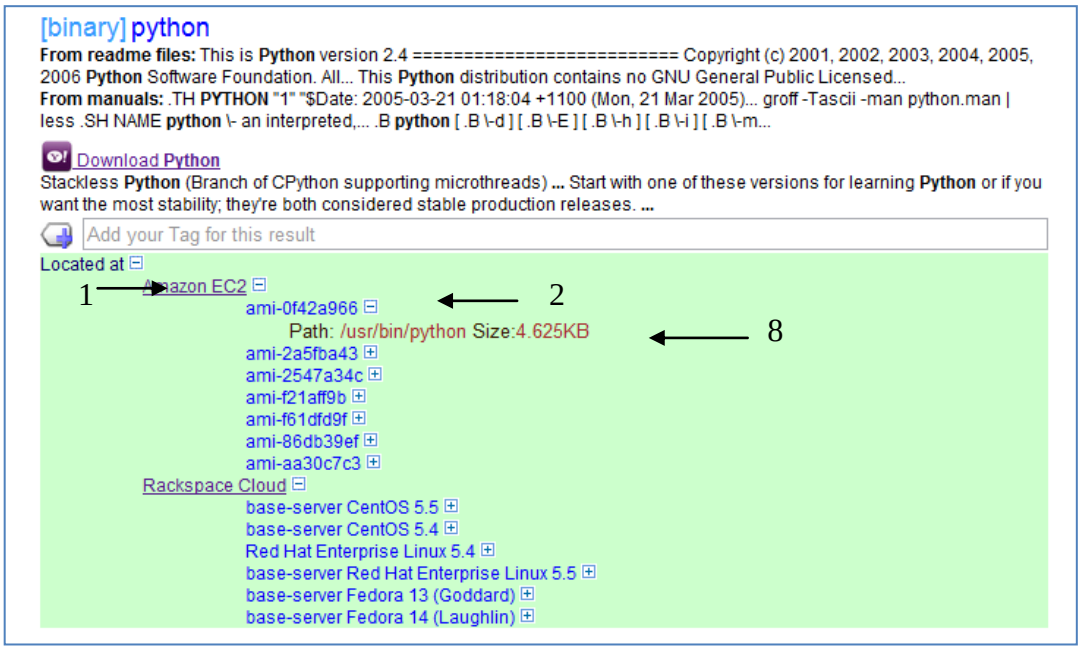


Εικόνα 3.3.6.1: Ιστοσελίδα αποτελεσμάτων αναζήτησης – Επιστροφή προτεινόμενων ετικετών και αναζήτηση βάση ετικέτας

3.3.7 Τοποθεσία αρχείων λογισμικού

Μια σημαντική πληροφορία που σίγουρα ενδιαφέρει τους χρήστες της μηχανής αναζήτησης αυτής, είναι το πού θα βρουν το λογισμικό το οποίο αναζητούν. Αυτή η πληροφορία επιστρέφεται στο χρήστη με τον τρόπο που αναπαριστάται στην εικόνα (3.3.7.1).

Συγκεκριμένα παρέχεται η πληροφορία σε ποιά Cloud υποδομή βρίσκεται (βέλος 1), σε ποια μηχανή (βέλος 2), καθώς και ποιο είναι το μονοπάτι στο οποίο είναι εγκατεστημένο το συγκεκριμένο λογισμικό αλλά και το μέγεθός του (βέλος 3).

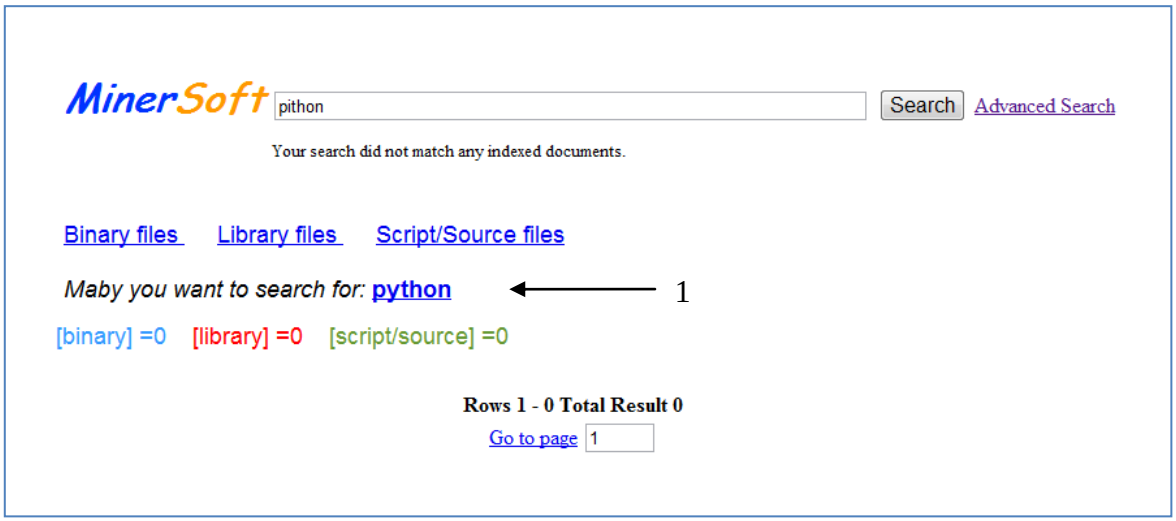


Εικόνα 3.3.7.1: Τοποθεσία αρχείου λογισμικού

3.3.8 Έλεγχος ορθογραφίας ερωτήματος

Ένα κοινό χαρακτηριστικό των μηχανών αναζήτησης στο διαδίκτυο είναι η προσφορά μιας εναλλακτικής αναζήτησης στην περίπτωση ύπαρξης ορθογραφικού λάθους στο ερώτημα του χρήστη (“*Did you mean?*”).

Πολλές φορές ένας χρήστης μιας μηχανής αναζήτησης, λόγω κεκτημένης ταχύτητας είναι φυσικό να κάνει κάποια ορθογραφικά λάθη που πιθανό να αλλοιώσουν το αποτέλεσμα που επιθυμεί να λάβει. Ένα τέτοιο παράδειγμα φαίνεται στην εικόνα (3.3.8.1). Ο χρήστης εισήγαγε λανθασμένα τη λέξη *python* με συνέπεια να μην επιστραφούν οποιαδήποτε αποτελέσματα. Ωστόσο η μηχανή αναζήτησης προσφέρει μια εναλλακτική αναζήτηση με τη λέξη αυτή ορθογραφικά σωστή (βέλος 1). Η δυνατότητα αυτή προσφέρεται χάρις στη κλάση SpellChecker της βιβλιοθήκης Lucene [17]. Τον έλεγχο αυτό τον πετυχαίνει με τη δημιουργία ενός ορθογραφικού λεξικού, το οποίο είναι εφαρμοσμένο στους όρους που περιέχονται στα ανεστραμμένα ευρετήρια του Minersoft ελέγχοντας κατά πόσο μια λέξη είναι όμοια με μια άλλη με την μέθοδο Edit Distance και καθιστά την μηχανή αναζήτησης πιο φιλική προς τους χρήστες αλλά και πιο αποτελεσματική.



Εικόνα 3.3.8: Ορθογραφικός έλεγχος ερωτήματος

Κεφάλαιο 4

Ταξινόμηση Λογισμικού

Εκτός από τις πληροφορίες που βρίσκονται ήδη μέσα στα ανεστραμμένα ευρετήρια, ένας άλλος τρόπος να παρέχεται επιπλέον πληροφόρηση για το λογισμικό που αναζητούν οι χρήστες είναι από τους ίδιους τους χρήστες. Η πείρα που έχουν κάποιοι χρήστες μπορούμε να την εκμεταλλευτούμε με τη χρήση των ετικετών, δηλαδή λέξεων οι οποίες περιέχουν σημαντική πληροφορία γιατί περιγράφουν κάποια δεδομένα και στην δική μας περίπτωση αρχεία λογισμικού από τα ανεστραμμένα ευρετήρια του Minersoft.

Για την εκμετάλλευση λοιπόν των πληροφοριών που θα προσφέρονται από τους χρήστες, είναι απαραίτητη η θεωρία της Εξόρυξης Δεδομένων[21, 22]. Η Εξόρυξη Δεδομένων (Data mining) ορίζεται ως η εξεύρεση (σημαντικών, άγνωστων και πιθανόν χρήσιμων) *πληροφοριών ή επαναλαμβανόμενων Προτύπων* (patterns). Η Εξόρυξη Δεδομένων αναγνωρίζεται ως ένα δυνατό εργαλείο και συνδυάζει στατιστική, αλγόριθμους ταξινόμησης, προέρχεται από το πεδίο των βάσεων δεδομένων, ενώ υπάρχει και αρκετή επικάλυψη με τη μηχανική μάθηση.

Η Μηχανική Μάθηση [21, 22] (Machine learning) θεωρείται ότι είναι μια ερευνητική περιοχή η οποία υπάγεται στην Τεχνητή Νοημοσύνη (Artificial Intelligence) και ερευνά το σχεδιασμό και την ανάπτυξη αλγορίθμων που έχουν ως σκοπό την υλοποίηση συστημάτων μάθησης.

4.1 Θεωρητικό Υπόβαθρο

Σε αυτή την υποενότητα θα γίνει μια συνοπτική περιγραφή γνωστών μεθόδων και εννοιών οι οποίες θα χρησιμοποιηθούν σε αυτή τη διπλωματική εργασία.

4.1.1 Μηχανική Μάθηση και ταξινόμηση

Όπως προαναφέραμε η μηχανική μάθηση ερευνά το σχεδιασμό και την ανάπτυξη αλγορίθμων που έχουν ως σκοπό την υλοποίηση συστημάτων μάθησης. Έτσι και στην περίπτωση της ταξινόμησης αρχείων γίνεται η αυτόματη ανάπτυξη ενός ταξινομητή ο οποίος χρησιμοποιεί ένα σύνολο από ήδη ταξινομημένα σε κατηγορίες αρχεία (σύνολο εκπαίδευσης) και αφού τα αναλύσει μαθαίνει ποια χαρακτηριστικά περιγράφουν κάθε

κατηγορία. Έτσι, είναι σε θέση να απαντήσει στο ερώτημα «σε ποιες κατηγορίες είναι πιθανό να ανήκουν κάποια αρχεία» δίνοντας κάποιες σχετικές προβλέψεις.

4.1.2 Ταξινόμηση αρχείων

Ένα αρχείο λογισμικού της μηχανής αναζήτησης Minersoft πιθανότατα να ανήκει σε περισσότερες από μια κατηγορίες και συνεπώς να αντιστοιχεί σε περισσότερες ετικέτες. Για παράδειγμα ένα αρχείο λογισμικού για ένα πρόγραμμα φυσικής, μπορεί να ανήκει στις κατηγορίες ***energy*** και ***physics,*** ετικέτες που ανέθεσαν οι χρήστες στο συγκεκριμένο αρχείο.

Το πρόβλημά μας, λοιπόν, είναι με ποιο τρόπο η γνώση, η οποία προσφέρεται από τους χρήστες στη μορφή ετικετών πάνω σε ορισμένα αρχεία λογισμικού, μπορεί να αξιοποιηθεί ώστε τα υπόλοιπα αρχεία λογισμικού να μπορούν να καταταγούν και αυτά σε σχετικές κατηγορίες. Αυτό γίνεται με τη διαδικασία της ταξινόμησης.

Η ταξινόμηση είναι η διαδικασία ανάθεσης μιας κατηγορίας c_j σε ένα αρχείο d_i όπου το c_j ανήκει σε ένα σύνολο κατηγοριών $C=\{c_1,c_2,..c_{|C|}\}$ και το d_i σε ένα σύνολο αρχείων $D=\{d_1,d_2,..d_{|D|}\}$. Επίσης, είναι απαραίτητη η χρήση ενός ταξινομητή (classifier), ο οποίος είναι υπεύθυνος να εκτελεί την ταξινόμηση και θεωρείται ως μια διαδικασία $h(d,c)\rightarrow\{True,False\}$ η οποία βάση κάποιων χαρακτηριστικών του d αποφασίζει αν αυτό ανήκει ή δεν ανήκει στην κατηγορία c. Η περίπτωση στην οποία απαιτείται η ανάθεση μόνο μιας κατηγορίας σε ένα αρχείο ονομάζεται *ταξινόμηση μονής ετικέτας* (single-label classification) και η οποία έχει μια ειδική περίπτωση τη *δυαδική ταξινόμηση* (binary classification) όπου ο συνολικός αριθμός των κατηγοριών είναι δύο. Το είδος όμως της ταξινόμησης που μας ενδιαφέρει εμάς είναι η *ταξινόμηση πολλαπλών τάξεων* (multiclass classification), καθώς ένα αρχείο λογισμικού από τα ανεστραμμένα ευρετήρια του Minersoft είναι πιθανό να ανήκει σε πολλές κατηγορίες.

4.1.3 Ταξινόμηση μονής ετικέτας

Όπως αναφέραμε και στην εισαγωγή η περίπτωση στην οποία απαιτείται η ανάθεση μόνο μιας κατηγορίας σε ένα αντικείμενο ονομάζεται *ταξινόμηση μονής ετικέτας*. Στην

ταξινόμηση μονής ετικέτας υπάρχουν πολλές κατηγορίες ταξινομητών όπως είναι οι πιθανοτικοί ταξινομητές, τα δέντρα αποφάσεων, οι ταξινομητές κανόνων, Νευρωνικά δίκτυα, ταξινομητές βάσει περιπτώσεων και οι μηχανές διανυσμάτων υποστήριξης.

4.1.3.1 Πιθανοτικοί ταξινομητές

Αντιπροσωπευτικός ταξινομητής αυτής της κατηγορίας είναι ο Naïve Bayes [22]. Οι ταξινομητές της κατηγορίας αυτής αποφασίζουν βάσει της πιθανότητας $P(A|\vec{B}) = \frac{P(A)\times P(\vec{B}|A)}{P(\vec{B})}$, δηλαδή της πιθανότητας το αρχείο που αναπαριστάται από το διάνυσμα $\vec{B}$ =(w1,w2,..w_N) να ανήκει στη κατηγορία Α. Ο ταξινομητή αυτός μπορεί να προβλέψει την τάξη με τη μεγαλύτερη πιθανότητα, όμως ο υπολογισμός της πιθανότητας $P(\vec{B}|A)$, δηλαδή της πιθανότητας ένα αντικείμενο της κατηγορίας Α να έχει την αναπαράσταση Β είναι πολύ δύσκολος αφού τα διανύσματα $\vec{B}$ είναι πάρα πολλά. Έτσι για να επιλυθεί αυτό το πρόβλημα γίνεται η υπόθεση ότι οι τιμές των χαρακτηριστικών μιας τάξης είναι ανεξάρτητες μεταξύ τους, άλλωστε ο αλγόριθμος Bayes για αυτό ονομάζεται αφελής (Naive). Έτσι πολλαπλασιάζονται οι πιθανότητας ύπαρξης όλων των χαρακτηριστικών μεταξύ τους, $P(\vec{B}|A) = \prod_{\kappa=1}^{|X|}(W(k)|A)$ και συνεπώς υπολογίζεται και αυτή με την πιο μεγάλη πιθανότητα.

4.1.3.2 Δέντρα αποφάσεων για ταξινόμηση

Τα δέντρα αποφάσεων είναι μια μέθοδος για ορθολογική λήψη αποφάσεων και έχουν ένα μεγάλο εύρος εφαρμογών καθώς χρησιμοποιούνται στη στατιστική, στην εξόρυξη δεδομένων και στη μηχανική μάθηση με σκοπό τη λήψη προγνωστικών αποφάσεων. Σε αυτή τη δεντρική δομή τα φύλλα αντιπροσωπεύουν κατηγοριοποιήσεις, ενώ οι διακλαδώσεις αντιπροσωπεύουν συνδέσμους χαρακτηριστικών που οδήγησαν σε αυτές τις κατηγοριοποιήσεις και τα χαρακτηριστικά αντιπροσωπεύονται από τους εσωτερικούς κόμβους του δέντρου. Οι ταξινομητές αυτοί πραγματοποιούν μία ταξινόμηση ξεκινώντας από τη ρίζα του δένδρου και εκτελώντας αναδρομικά τη διαδικασία αυτή:

- Για κάθε κόμβο που δεν είναι φύλλο του δέντρου εξετάζεται το βάρος του συγκεκριμένου χαρακτηριστικού και ακολουθεί την αντίστοιχη διακλάδωση. Αν

η διακλάδωση αυτή οδηγήσει σε φύλλο τότε ο ταξινομητής αποφάσισε την κατηγορία.

4.1.3.3 Μηχανές διανυσμάτων υποστήριξης

Οι ΜΔΥ είναι μια οικογένεια μεθόδων επιβλεπόμενης μάθησης, που χρησιμοποιούνται στην κατάταξη διαφόρων αντικειμένων σε κατηγορίες. Οι ΜΔΥ μπορούν γενικά να διαχωριστούν σε δυαδικές (binary) και πολλαπλών ετικετών (multi-label). Οι δυαδικές ΜΔΥ, προβάλλουν τα σημεία του συνόλου εκπαίδευσης σε έναν χώρο περισσοτέρων διαστάσεων και βρίσκουν το υπερεπίπεδο το οποίο διαχωρίζει βέλτιστα τα σημεία των δύο τάξεων. Τα άγνωστα σημεία ταξινομούνται σύμφωνα με την πλευρά του υπερεπίπεδου στην οποία βρίσκονται. Στην περίπτωση των multi-label ΜΔΥ, η απλούστερη περίπτωση είναι να υλοποιηθούν ως διαδοχική εφαρμογή δυαδικών ΜΔΥ. Ένα παράδειγμα ΜΔΥ είναι ο αλγόριθμος Σειριακής Ελάχιστης Βελτιστοποίησης (Sequential Minimal Optimization - SMO) [28], ο οποίος είναι υλοποιημένος στο Weka[12].

4.1.4 Ταξινόμηση πολλαπλών ετικετών

Η ταξινόμηση πολλαπλών ετικετών χωρίζεται σε δύο κατηγορίες μεθόδων: α) *μέθοδοι μετασχηματισμού προβλήματος* (problem transformation methods) και β) *μέθοδοι προσαρμογής αλγορίθμου* (algorithm adaptation methods). Στην πρώτη ομάδα το πρόβλημα της ταξινόμησης πολλαπλών ετικετών μετατρέπεται σε ένα ή και περισσότερα προβλήματα ταξινόμησης μονής ετικέτας για το οποίο υπάρχει πληθώρα αλγορίθμων που το επιλύει. Στη δεύτερη ομάδα γίνεται επέκταση κάποιων αλγορίθμων μάθησης από τις μεθόδους αυτές ώστε να μπορούν να χειρίζονται απευθείας δεδομένα πολλαπλών ετικετών.

4.1.4.1 Μέθοδοι Μετασχηματισμού Προβλήματος

Για να περιγράψουμε καλύτερα τις μεθόδους που ανήκουν σε αυτή την κατηγορία θα χρησιμοποιήσουμε το πεπερασμένο σύνολο ετικετών $E=\{\varepsilon_1, \varepsilon_2,\ldots\varepsilon_{|E|}\}$ σε ένα σύνολο από παραδείγματα $\Delta = \{(\overrightarrow{\chi 1}, Y_1), (\overrightarrow{\chi 2}, Y_2),\ldots(\overrightarrow{\chi|\Delta|}, Y|\Delta|)\}$, όπως και στο [21] όπου το

κάθε $\vec{\chi}$ είναι ένα διάνυσμα των χαρακτηριστικών του κάθε αρχείου ενώ το Y υποσύνολο του συνόλου των ετικετών E.

Παράδειγμα	Σύνολο ετικετών
1	$\{ \varepsilon_1, \varepsilon_4\}$
2	$\{ \varepsilon_3, \varepsilon_4\}$
3	$\{ \varepsilon_1\}$
4	$\{ \varepsilon_2, \varepsilon_3, \varepsilon_4\}$

Πίνακας 4.1.4.1.1: Παράδειγμα συνόλου πολλαπλών ετικετών

4.1.4.2 Μέθοδος δυναμοσυνόλου ετικετών (LabelPowerset)

Η μέθοδος *δυναμοσυνόλου ετικετών* Label Powerset (LP) είναι μία μέθοδος η οποία θεωρεί ότι κάθε σύνολο ετικετών που υπάρχει στα δεδομένα εκπαίδευσης είναι μια τάξη για ένα νέο πρόβλημα ταξινόμησης μονής ετικέτας. Με λίγα λόγια ο ταξινομητής, ο ταξινομητής απλής ετικέτας του LP έχει ως έξοδο την πιο πιθανή τάξη που όμως αντιστοιχεί σε ένα σύνολο από ετικέτες. Αυτό αναπαριστάται στο πίνακα (4.1.4.2.1) που ακολουθεί.

Παράδειγμα	Σύνολο ετικετών
1	$\varepsilon_1, \varepsilon_4$
2	$\varepsilon_3, \varepsilon_4$
3	ε_1
4	$\varepsilon_2, \varepsilon_3, \varepsilon_4$

Πίνακας 4.1.4.2.1: Παραδείγματα ετικετών που μετασχηματίστηκαν από την μέθοδο LP

4.1.4.3 Μέθοδος δυαδικής συνάφειας (Binary Relevance BR)

Η μέθοδος δυαδικής συνάφειας (Binary Relevance ή BR) [21] είναι μια μέθοδος μετασχηματισμού η οποία χρησιμοποιεί δυαδικούς ταξινομητές, δηλαδή ταξινομητές

που αποφασίζουν αν ένα παράδειγμα ανήκει η όχι σε μια κατηγορία. Η μέθοδος αυτή χρησιμοποιεί τόσους ταξινομητές όσες και οι ετικέτες που υπάρχουν στο σύνολο E, δηλαδή ένας για κάθε μια ετικέτα έτσι το αρχικό σύνολο των |Δ| δεδομένων μετασχηματίζεται σε |Δ| σύνολα δεδομένων που περιέχουν παραδείγματα του αρχικού συνόλου. Ένα παράδειγμα θεωρείται ότι ανήκει σε κάποια ετικέτα και συνεπώς είναι θετικό όταν το αρχικό σύνολο ετικετών E περιέχει την ετικέτα, αν όχι θεωρείται αρνητικό. Για να ταξινομηθεί ένα νέο αντικείμενο πρέπει πρώτα να συγχωνευτούν τα αποτελέσματα των προβλέψεων των |E| ταξινομητών. Στον πίνακα (4.1.4.3.1) παρουσιάζονται τα |Δ| σύνολα δεδομένων που προέκυψαν από τον μετασχηματισμό με την μέθοδο BR.

Παρ.	Ετικέτα
1	ε_1
2	$\neg\varepsilon_1$
3	ε_1
4	$\neg\varepsilon_1$

Παρ.	Ετικέτα
1	$\neg\varepsilon_2$
2	$\neg\varepsilon_2$
3	$\neg\varepsilon_2$
4	ε_2

Παρ.	Ετικέτα
1	ε_3
2	$\neg\varepsilon_3$
3	ε_3
4	$\neg\varepsilon_3$

Παρ.	Ετικέτα
1	ε_4
2	ε_4
3	$\neg\varepsilon_4$
4	ε_4

Πίνακας 4.1.4.3.1: Σύνολα δεδομένων μετά των μετασχηματισμό της μεθόδου BR

Η μέθοδος αυτή επομένως δεν λαμβάνει υπόψη τις συσχετίσεις μεταξύ των ετικετών, αντίθετα από την LP με αποτέλεσμα να έχει μικρότερη πολυπλοκότητα.

4.1.4.4 Μέθοδοι Προσαρμογής Αλγορίθμων

4.1.4.4.1 MultiLabelkNN

Ο MLkNN [10] ανήκει στην κατηγορία των αλγορίθμων οκνηρής μάθησης και είναι μια προσαρμογή του γνωστού αλγορίθμου ταξινόμησης μονής ετικέτας kNN (k Πλησιέστερων Γειτόνων) ο οποίος κάνει προβλέψεις βάση των *k* πλησιέστερων παραδειγμάτων όπου η παράμετρος *k* καθορίζεται από τους χρήστες. Ο MLkNN όπως και ο kNN εντοπίζει τους *k* πλησιέστερους γείτονες του συνόλου εκπαίδευσης, ακολούθως χρησιμοποιεί την στατιστική πληροφορία που υπάρχει στα γειτονικά

παραδείγματα και γίνεται χρήση της MAP (maximum a posteriori) αρχής για να ανατεθούν οι ετικέτες στη νέα περίπτωση. Ο τρόπος λειτουργίας του έχει ως εξής:

Έστω ότι λαμβάνονται υπόψη *k* κοντινότεροι γείτονες και ένα παράδειγμα $\pi = (\vec{\chi}, \vec{Y})$ όπου το $\vec{Y}$ υποσύνολο του συνόλου των ετικετών E, $\vec{Y} \subseteq \Upsilon$. Αν μια ετικέτα *l* που $l \in \Upsilon$ και υπάρχει στο $\vec{Y}$ τότε παίρνει τιμή 1 αλλιώς παίρνει τιμή 0. Επιπλέον, N(π) ορίζεται ως το σύνολο των *k* πλησιέστερων γειτόνων του παραδείγματος *π*, συνεπώς ένα διάνυσμα μέτρησης συμμετοχής (*membership counting vector*) ορίζεται ως εξής:

$$\overrightarrow{C_\pi}(l) = \sum_{a \in N(\pi)} \overrightarrow{Y_a}(l), l \in \Upsilon$$

Όπου το $\overrightarrow{C_\pi}(l)$ ορίζεται ως ο μετρητής των γειτόνων του *π* που ανήκουν στην κλάση *l*.

Για κάθε παράδειγμα ελέγχου t, ο MLkNN πρώτα εντοπίζει τους *k* γείτονές του, από το σύνολο εκπαίδευσης του. Με H_b^l , b={0,1} και όταν το b είναι 0 σημαίνει ότι η ετικέτα *l* δεν υπάρχει στο παράδειγμα t ενώ όταν είναι 1 υπάρχει. Επιπλέον, έστω ότι E_j^l (j={1,2,..*k*) συμβολίζει το γεγονός ότι μεταξύ των *k* πλησιέστερων γειτόνων του t, υπάρχουν ακριβώς j που έχουν την ετικέτα *l*. Συνεπώς, με βάση το διάνυσμα μέτρησης $\overrightarrow{C_\pi}(l)$, το διάνυσμα κλάσης $\overrightarrow{Y_t}$ καθορίζεται χρησιμοποιώντας την εκ' των υστέρων μέγιστη αρχή όπως φαίνεται πιο κάτω:

$$\overrightarrow{Y_t}(l) = \text{argmax}_{b=\{0,1\}} \text{P}(H_b^l | E^l_{\overrightarrow{C_t(l)}}),\ l \in E$$

Και σύμφωνα με τον κανόνα του Bayes η εξίσωση αυτή γράφεται ως:

$$\overrightarrow{Y_t}(l) = \text{argmax}_{b=\{0,1\}} \frac{\text{P}(H_b^l)\ \text{P}(E^l_{\overrightarrow{C_t(l)}} | H_b^l)}{\text{P}(E^l_{\overrightarrow{C_t(l)}})} = \text{argmax}_{b=\{0,1\}}\ \text{P}(H_b^l)\ \text{P}(E^l_{\overrightarrow{C_t(l)}} | H_b^l)$$

4.1.4.4.2 Backpropagation Multi-Label Learning

Ο αλγόριθμος BPMLL [11] είναι αλγόριθμος των νευρωνικών δικτύων και αποτελεί μία προσαρμογή του αλγορίθμου ανάστροφης μετάδοσης σφάλματος (error back propagation) για δεδομένα πολλαπλών ετικετών. Η κύρια μετατροπή στον αλγόριθμο είναι μία νέα συνάρτηση σφάλματος που λαμβάνει υπόψη τις πολλαπλές ετικέτες.

4.1.4.5 Random k-Labelsets (RAkEL)

Η μέθοδος RAkEL κατασκευάζει ένα σύνολο LP ταξινομητών, όπου ο κάθε ένας από αυτούς τους ταξινομητές εκπαιδεύεται πάνω σε ένα υποσύνολο του συνόλου E των ετικετών. Το υποσύνολο αυτό επιλέγεται τυχαία και ονομάζεται *labelset (ετικετοσύνολο)*. Η μέθοδος αυτή έχει σκοπό να λαμβάνει υπόψη τις συσχετίσεις μεταξύ των ετικετών όπως στον Label Powerset, αποφεύγοντας όμως τη μεγάλη πολυπλοκότητα που παρατηρείται σε αυτόν εξαιτίας της ποσότητας των ετικετών, καθώς οι LP ταξινομητές αντιμετωπίζουν απλούστερα προβλήματα μονής ετικέτας. Στη μέθοδο RAkEL συναντούμε δύο παραλλαγές την (disjoint – RAkELd) στην οποία τα *labelsets* είναι ξένα μεταξύ τους και την (overlapping – RAkELo) στην οποία η μέθοδος RAkEL επιστρέφει πολλαπλές προβλέψεις για την ίδια ετικέτα. Η μέθοδος αυτή αναλύεται με περισσότερη λεπτομέρεια στα[6,7,21].

4.2 Μεθοδολογία ταξινόμησης αρχείων λογισμικού

4.2.1 Ετικέτες (Tags) στο Διαδίκτυο

Οι ετικέτες (tags) στο Διαδίκτυο ονομάζονται οι λέξεις κλειδιά ή οι όροι που παρέχονται από τους χρήστες με σκοπό να περιγράψουν όσο το καλύτερο κάποια αντικείμενα (όπως εικόνες, άρθρα, αρχεία βίντεο κλπ). Δυο παραδείγματα συστημάτων που βασίζονται στην χρήση τέτοιων ετικετών είναι το del.icio.us™ [23] και το flickr™ [24]. Το del.icio.us™ είναι μια ιστοσελίδα όπου οι χρήστες μπορούν να αποθηκεύσουν τους σελιδοδείκτες τους για κάποια URI και να τους μοιραστούν με άλλους χρήστες της σελίδας αναθέτοντάς τους κάποιες ετικέτες που περιγράφουν τα URI όσο το δυνατό καλύτερα. Ακολούθως εκτελώντας αναζήτηση μπορούν να βρουν τα URIs αυτά αλλά και URIs άλλων χρηστών που εμπίπτουν στην ίδια κατηγορία. Το flickr είναι μια ιστοσελίδα, η οποία δημιουργήθηκε για να φιλοξενεί φωτογραφίες και βίντεο και που επιτρέπει στους χρήστες να τα αναγνωρίσουν με τη χρήση των ετικετών.

4.2.2 Ετικέτες(tags) στο Minersoft

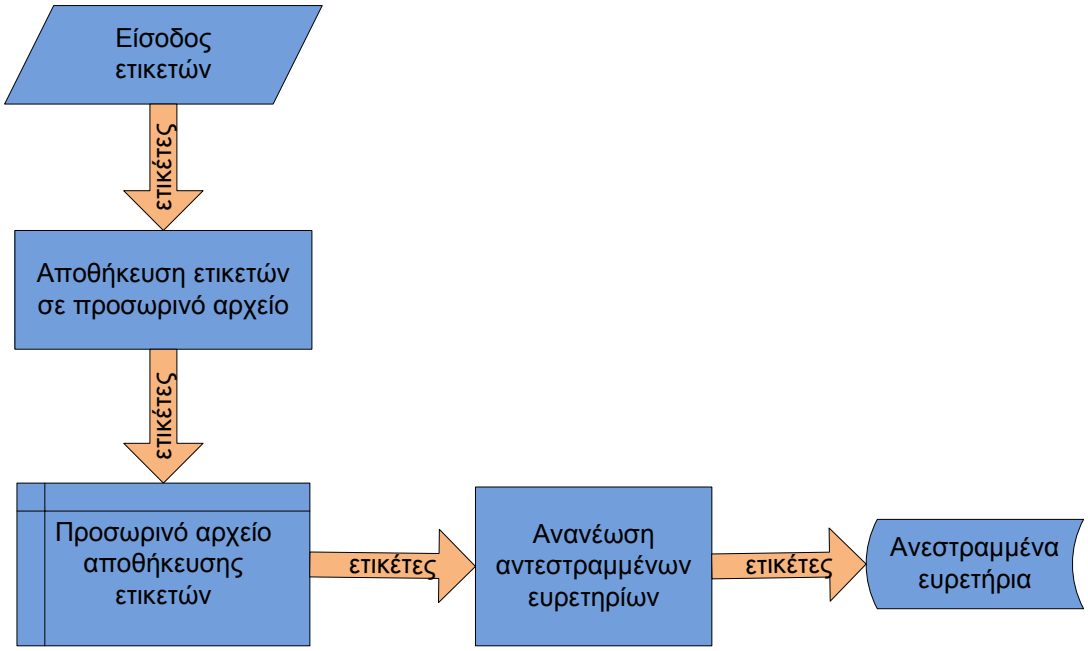
Όπως είπαμε στόχος της μηχανής αναζήτησης Minersoft είναι να προσφέρει αποτελέσματα με όσο το δυνατό περισσότερη πληροφορία, για λογισμικό το οποίο υπάρχει σε cloud υποδομές. Για να πετύχουμε αυτόν το στόχο, εισάγαμε τη χρήση των ετικετών στη μηχανή αναζήτησης Minersoft έτσι ώστε τα επιστρεφόμενα αποτελέσματα να είναι εμπλουτισμένα με πληροφορίες που του παρέχουν οι ίδιοι οι χρήστες.

Η διαδικασία της προσθήκης των ετικετών από τους χρήστες είναι απλή. Καταρχάς το αντικείμενο το οποίο θα επιδέχεται περιγραφή από ετικέτες είναι αρχεία λογισμικού (όπως binaries, libraries, source codes, scripts) τα οποία περιέχονται στα ανεστραμμένα ευρετήρια του Minersoft. Με το που ένας χρήστης εκτελεί μια αναζήτηση του επιστρέφονται κάποια σχετικά αποτελέσματα. Κάτω από αυτά τα αποτελέσματα εμφανίζεται ένα πεδίο στο οποίο οι χρήστες απλά εισάγουν τις ετικέτες που πιστεύουν ότι περιγράφουν καλύτερα το λογισμικό του αποτελέσματος. Ακολούθως οι ετικέτες αυτές αποθηκεύονται προσωρινά μέσα σε ένα αρχείο κειμένου και ανά κάποιο χρονικό διάστημα ανασύρονται για να αποθηκευτούν σε κάποιες νέες ζώνες στα ανεστραμμένα ευρετήρια. Η διαδικασία αυτή περιγράφεται λεπτομερώς στο σχήμα (4.2.2.1).

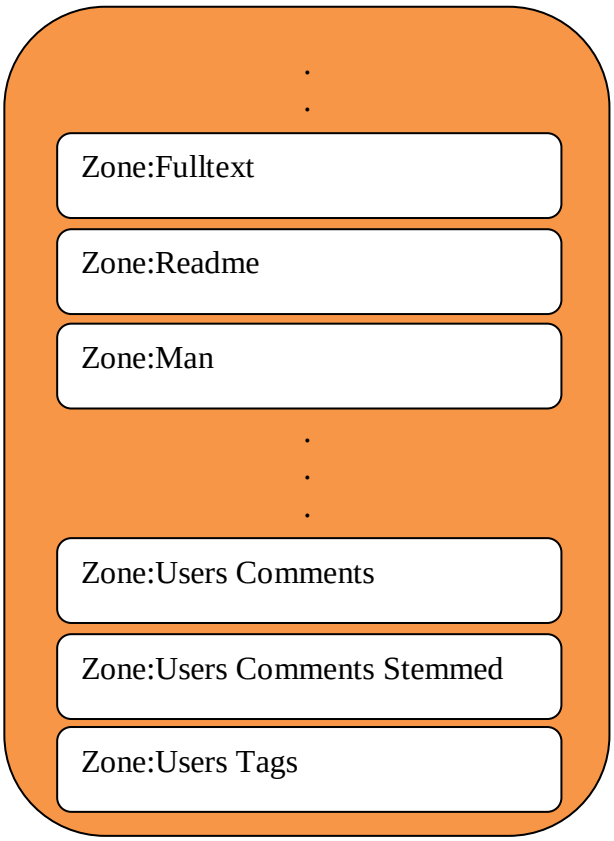
Όπως προαναφέραμε οι ετικέτες των χρηστών αποθηκεύονται σε νέες ζώνες και συγκεκριμένα σε τρεις, στη ζώνη *User Comments*, στη ζώνη *User Comments Stemmed* και στην *Users Tags* οι οποίες φαίνονται και στο σχήμα (4.2.2.2). Οι πρώτες δύο ζώνες δημιουργήθηκαν με σκοπό να γίνεται αναζήτηση πάνω τους, ενώ η τρίτη ζώνη αφορά την πληροφορία που θα παρέχεται ολόκληρη με την επιστροφή κάποιου αποτελέσματος καθώς και στη χρήση των πληροφοριών που έδωσαν οι χρήστες στην αρχική τους μορφή και που χρειάζονται σε μετέπιπτα στάδιο. Η πρώτη ζώνη αποθηκεύει τους όρους των χρηστών αφού τους εφαρμοστούν οι κανόνες επεξεργασίας φυσικής γλώσσας για να αποφευχθούν κοινές λέξεις και χαρακτήρες οι οποίοι δεν είναι χρήσιμοι στην αναζήτηση καθώς και μετατροπή όλων των χαρακτήρων από κεφαλαία σε μικρά. Η δεύτερη προβαίνει στην ίδια επεξεργασία με την πρώτη, αλλά σε αυτήν τη ζώνη επιβάλλεται και στελέχωση των λέξεων, ενώ στην τρίτη κατηγορία δεν επιβάλλεται

κάποιου είδους επεξεργασία αφού οι πληροφορίες πρέπει βρίσκονται στην αρχική τους μη επεξεργασμένη μορφή.

Οι δύο ζώνες οι οποίες συμπεριλαμβάνονται στην αναζήτηση αρχικά έχουν σταθμισμένο βάρος με τις υπάρχουσες ζώνες και χαμηλότερο από το βάρος των υπόλοιπων ζωνών. Με την πρόσθεση περισσότερων ετικετών σε μια ζώνη το βάρος της αυξάνεται, για παράδειγμα αν σε ένα αποτέλεσμα έχουν τοποθετηθεί 2 ετικέτες και σε ένα άλλο 4 τότε το δεύτερο έχει πιο ψηλό βάρος με αποτέλεσμα να πετυχαίνει υψηλότερη θέση στην κατάταξη των αποτελεσμάτων για το ίδιο ερώτημα.



Σχήμα 4.2.2.1: Διαδικασία προσθήκης ετικετών



Σχήμα 4.2.2.2: Ζώνες ετικετών

4.2.3 Προετοιμασία δεδομένων εκπαίδευσης ταξινομητή

Η ανάθεση ετικετών στα αποτελέσματα της μηχανής αναζήτησης του Minersoft δεν έχει μοναδικό σκοπό να εμπλουτίσει τις πληροφορίες για αναζήτηση. Ο κύριος σκοπός της ανάθεσης ετικετών είναι η περαιτέρω ταξινόμηση των αρχείων που αφορούν λογισμικό σε θεματικές κατηγορίες οι οποίες καθορίζονται από τους ίδιους τους χρήστες του Minersoft. Οι κατηγορίες αυτές είναι οι ετικέτες που αναθέτουν οι χρήστες, αφού είναι λέξεις κλειδιά που κατά την γνώμη των χρηστών περιγράφουν το αρχείο στο οποίο ανατέθηκαν.

Η ταξινόμηση των αρχείων λογισμικού εκτελείται ανά τακτά χρονικά διαστήματα έτσι ώστε νέες κατηγορίες που προκύπτουν από τις ετικέτες των χρηστών να λαμβάνονται υπόψη και να προσφέρουν επιπλέον γνώση για τα ήδη υπάρχοντα αρχεία λογισμικού. Αυτή η διαδικασία όπως αναφέραμε και πιο πάνω είναι επαναλαμβανόμενη ώστε ο ταξινομητής να εκπαιδεύεται συνεχώς από τις νέες ετικέτες.

Για την εκπαίδευση του ταξινομητή, είναι αναγκαία η δημιουργία δεδομένων παραδειγμάτων βάση των οποίων ο ταξινομητής θα μάθει πώς να τα διαχωρίζει σε κατηγορίες. Κάθε ένα παράδειγμα αποτελείται από τα *χαρακτηριστικά* μέσα από τα οποία αναζητά μοτίβα για τις *κατηγορίες* (ετικέτες) στις οποίες ανήκουν τα παραδείγματα αυτά. Τα χαρακτηριστικά στα οποία θα στηρίζεται η ταξινόμηση είναι οι πιο σημαντικοί όροι που υπάρχουν μέσα στα αρχεία λογισμικού καθώς και μέσα στα συσχετιζόμενα αρχεία κειμένου τους όπως αρχεία *readme* και *manuals*. Με τον όρο σημαντικοί εννοούμε, τους όρους που έχουν το πιο ψηλό df (Document frequency) σε όλα τα αρχεία του ανεστραμμένου ευρετηρίου των κατηγοριών *binary, library, script* και *source*.

Όπως απεικονίζεται και στο σχήμα (4.2.3.1) τα αρχεία λογισμικού που τους προστέθηκαν ετικέτες από τους χρήστες, φορτώνονται στη μνήμη για να επεξεργαστούν. Η επεξεργασία αυτή περιλαμβάνει τρεις διαδικασίες:

- *Ανεύρεση κατηγοριών*: Στη διαδικασία αυτή ανασύρονται όλες οι ετικέτες, από τα αρχεία των ανεστραμμένων ευρετηρίων. Οι ετικέτες αυτές είναι τοποθετημένες ώστε να περιγράφουν τα αρχεία λογισμικού που είχαν αναζητηθεί από τους χρήστες και οι ίδιοι οι χρήστες πρόσθεσαν αυτές τις περιγραφικές λέξεις. Αφού διασφαλιστεί ότι κάθε ετικέτα είναι μοναδική τότε έχουμε τις διαφορετικές κατηγορίες μας.

- *Επιλογή χαρακτηριστικών ταξινόμησης*: Η επιλογή χαρακτηριστικών τα οποία αποτελούν τις πληροφορίες που δίνονται στον ταξινομητή με σκοπό την ανεύρεση κάποιων μοτίβων για τη δημιουργία μοντέλου ταξινόμησης, επιλέγονται βάσει της συχνότητας εμφάνισης τους στο σύνολο των αρχείων του ευρετηρίου. Δηλαδή οι όροι με ψηλότερο DF (Document Frequency) είναι οι όροι που θα χρησιμοποιούνται ως χαρακτηριστικά ταξινόμησης.

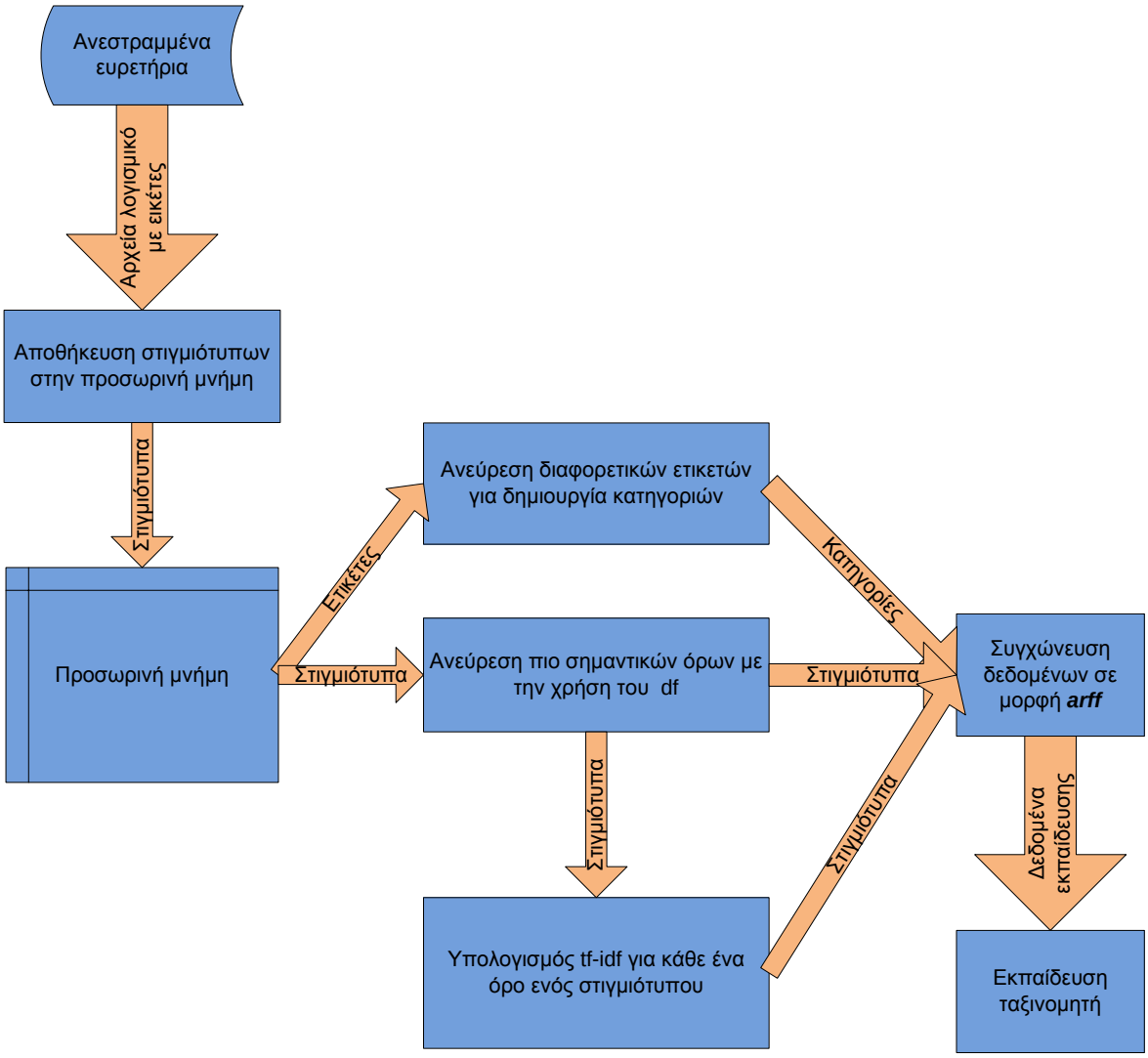
- *Ανάθεση βάρους*: Όπως έχουμε ήδη δείξει $\Delta = \{(\overrightarrow{\chi 1}, Y_1), (\overrightarrow{\chi 2}, Y_2), \ldots (\overrightarrow{\chi|\Delta|}, Y|\Delta|)\}$, όπου Δ το σύνολο των δεδομένων εκπαίδευσης και το διάνυσμα $\vec{\chi}$ αποτελεί το διάνυσμα των βαρών για το σύνολο των χαρακτηριστικών $\vec{\chi} = \{\beta_1, \beta_2, \beta_3 \ldots \beta_{|\chi|}\}$. Τα βάρη αυτά αφορούν το πόσο σημαντικός είναι ένας

όρος μέσα στο αρχείο και έτσι χαρακτηρίζουν και το ίδιο το αρχείο. Γι αυτό το λόγο θα χρησιμοποιηθούν ως βάρη τα TF-IDF των όρων -τύπος (4.2.3.1)- των αρχείων που ανήκουν στα δεδομένα εκπαίδευσης.

- *Δημιουργία* ***arff*** [20,22] *αρχείου εκπαίδευσης*: Αφού λοιπόν τελειώσουν οι τρεις προηγούμενες διαδικασίες και έχουν πλέον μετατραπεί τα αρχεία εκπαίδευσης σε μορφή διανύσματος τότε δημιουργείται ένα αρχείο της μορφής ***arff*** (Attribute-Relation File Format). Το αρχείο αυτό έχει τη μορφή που φαίνεται στο σχήμα (4.2.4.1) και αποτελεί την είσοδο του ταξινομητή, ο οποίος εκπαιδεύεται βάση αυτού του αρχείου.

$$\text{TF-IDF} = \text{Term Frequncy} \times \text{Inverse Document Frequency} = \text{TF} \times \log_{10}(\frac{N}{DF})$$

Τύπος 4.2.3.1: Βάρος των όρων στο Vector Space Model που θα χρησιμοποιηθεί στην ταξινόμηση των αρχείων λογισμικού



Σχήμα 4.2.3.1: Διαδικασία εκπαίδευσης ταξινομητή

4.2.4 Ταξινόμηση αρχείων

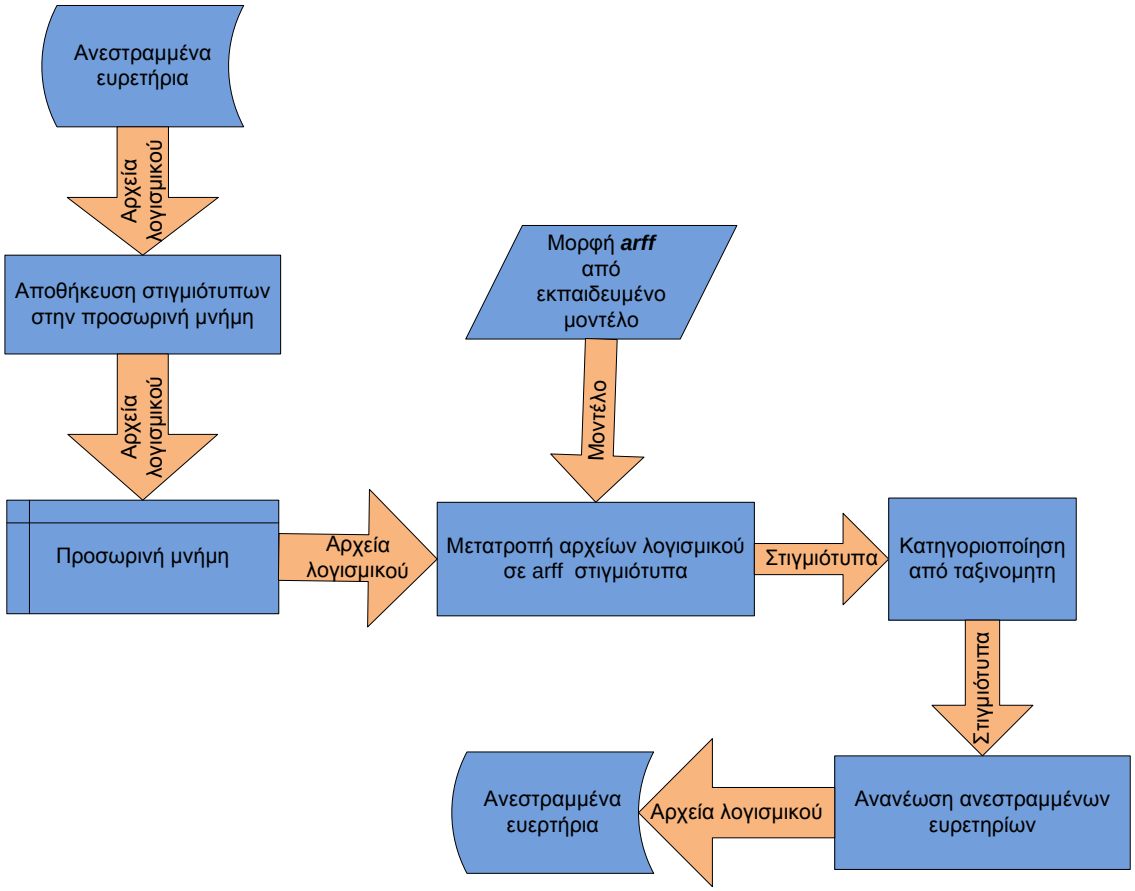
Αφού εκπαιδευτεί, λοιπόν, ο ταξινομητής, είναι σε θέση να δώσει πρόβλεψη για το ερώτημα «σε ποιες κατηγορίες πιθανόν να ανήκουν τα αρχεία λογισμικού που υπάρχουν στα ανεστραμμένα ευρετήρια του Minersoft». Η διαδικασία αυτή αναπαριστάται στο σχήμα 4.2.4.2.

```
% Title: Minersoft Data
%
@RELATION atlas

@ATTRIBUTE application  NUMERIC
@ATTRIBUTE software   NUMERIC
@ATTRIBUTE nucleon  NUMERIC
@ATTRIBUTE particle   NUMERIC
@ATTRIBUTE computing   {0,1}
@ATTRIBUTE physics   {0,1}
@ATTRIBUTE chemistry   {0,1}

@DATA
2.1,0.5,0,0,1,0,0
1.2,3.0,0.2,0,1,0,0
0,0.2,0,0.9,0,1,0
0,0.1,1.5,1.2,0,1,1
0,0,0,0.8,0,0,1
```

Σχήμα 4.2.4.1: Αρχείο μορφής *Attribute-Relation File Format*

Πρώτα πρέπει να φορτωθούν όλα τα αρχεία λογισμικού στη μνήμη, ακολούθως να φορτωθεί η μορφή του ***arff*** [20,22] – σχήμα (4.2.4.1) – η οποία έχει δημιουργηθεί κατά την διάρκεια της εκπαίδευσης του ταξινομητή και να μετατραπούν όλα τα αρχεία που αντιστοιχούν σε λογισμικό των ανεστραμμένων ευρετηρίων του Minersoft σε αυτή την μορφή, έτσι ώστε να είναι δυνατή η είσοδο τους στον ταξινομητή. Παίρνοντας λοιπόν ο ταξινομητής κάθε ένα από αυτά τα αρχεία (στιγμιότυπα) κάνει προβλέψεις κατατάσσοντας σε κατηγορίες τα αρχεία αυτά βάσει του εκπαιδευμένου μοντέλου που έχει δημιουργήσει.

Με το τέλος της ταξινόμησης όλων των αρχείων λογισμικού γίνεται ανανέωση των ανεστραμμένων ευρετηρίων. Κατά την ανανέωση των ευρετηρίων δημιουργούνται ή ανανεώνονται τρεις ζώνες με περιεχόμενο τις κατηγορίες που έχουν προβλεφθεί από τον ταξινομητή. Οι δύο ζώνες δημιουργούνται με σκοπό να προσφέρουν ένα επιπλέον είδος αναζήτησης, την αναζήτηση βάσει των κατηγοριών που προέρχονται από προβλέψεις του ταξινομητή με στελέχωση ή χωρίς, ενώ η τρίτη ζώνη θα περιέχει τις κατηγορίες αυτές χωρίς να αλλοιωθεί η αρχική τους μορφή με σκοπό να επιστραφούν ως έχουν στην γραφική διεπαφή.



Σχήμα 4.2.4.2: Διαδικασία ταξινόμησης

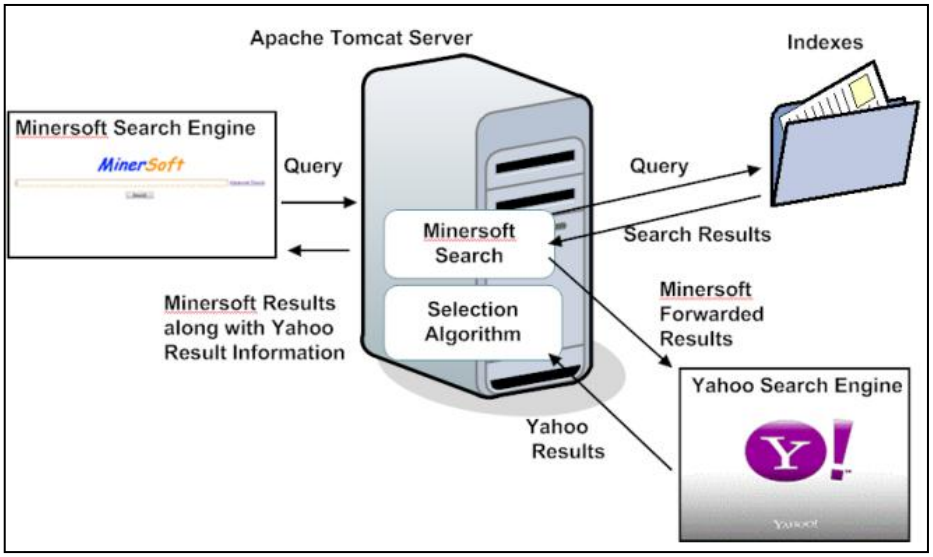
Κεφάλαιο 5

Εμπλουτισμός Αποτελεσμάτων από το Διαδίκτυο

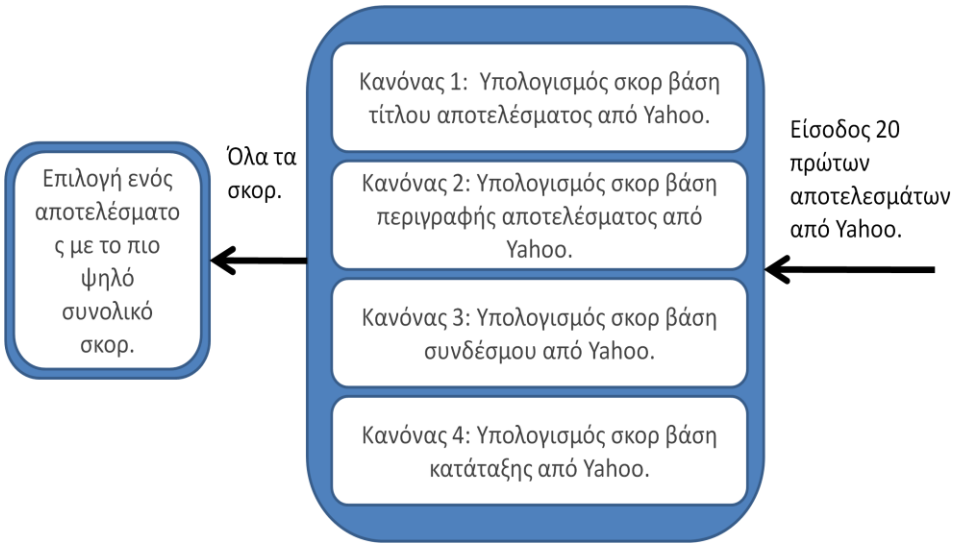
Η αρχιτεκτονική του αλγορίθμου είναι απλή καθώς χρησιμοποιεί την αρχιτεκτονική της μηχανής αναζήτησης για να συλλειτουργήσει και να συνεργαστεί με αυτή, ώστε να επιστρέψει περισσότερες πληροφορίες για κάθε αποτέλεσμα της μηχανής αναζήτησης Minersoft .

5.1 Οντότητες Αρχιτεκτονικής

Οι οντότητες που αποτελούν την αρχιτεκτονική του συστήματος είναι η ιστοσελίδα της μηχανής αναζήτησης από την οποία οι χρήστες εισάγουν τα ερωτήματά τους, τα ανεστραμμένα ευρετήρια τα οποία βρίσκονται εγκατεστημένα σε ένα εξυπηρετητή, η μηχανή αναζήτησης Yahoo και ο αλγόριθμος ο οποίος επεξεργάζεται τα αποτελέσματα που επιστρέφει η μηχανή αναζήτησης Yahoo.



Σχήμα 5.1: Αρχιτεκτονική αναζήτησης



Σχήμα 5.2: Αρχιτεκτονική αλγόριθμου επιλογής

5.2 Περιγραφή Αρχιτεκτονικής

Όπως αναπαριστάται στο σχήμα (5.1) μέσω της ιστοσελίδας ο χρήστης εισάγει το ερώτημα στη μηχανή αναζήτησης Minersoft και του επιστρέφονται τα αποτελέσματα. Πριν όμως επιστραφούν στον τελικό προορισμό δηλαδή το χρήστη οι τίτλοι των 10 πρώτων αποτελεσμάτων και κάθε 10 αποτελεσμάτων που επιλέγει να δει ο χρήστης, αποστέλλονται στη μηχανή αναζήτησης Yahoo. Η Yahoo εκτελεί αναζήτηση και επιστρέφει για κάθε ένα αποτέλεσμα τα 20 πρώτα αποτελέσματα σαν είσοδο στον αλγόριθμο επιλογής.

Ο αλγόριθμος επιλογής, που αναπαριστάται στο σχήμα (5.2), αφού λάβει τα 20 αυτά αποτελέσματα, με τη χρήση κάποιων ευριστικών κανόνων επιλέγει το πιο σχετικό αποτέλεσμα της Yahoo για κάθε ένα αποτέλεσμα του Minersoft. Τέλος τα 10 αποτελέσματα πλέον εμπλουτισμένα με πληροφορίες της Yahoo επιστρέφονται στο χρήστη.

5.3 Μεθοδολογία

Όπως αναφέραμε και πιο πάνω κάθε αποτέλεσμα της μηχανής αναζήτησης Minersoft, αποστέλλεται στην Yahoo και εκτελείται αναζήτηση βάσει ονόματος. Ακολούθως στα

20 πρώτα αποτελέσματα της μηχανής αναζήτησης Yahoo εφαρμόζονται κάποιοι ευριστικοί κανόνες για την επιλογή του πιο σχετικού από αυτά τα 20 αποτελέσματα.

Για την υλοποίηση αυτής της διαδικασίας και την επιστροφή των αποτελεσμάτων από την αναζήτηση στο διαδίκτυο χρησιμοποιήθηκε το **search API** της Yahoo **BOSS** [16] το οποίο μας επιτρέπει την επιστροφή του συνδέσμου, του τίτλου και την περιγραφή ενός αποτελέσματος .

5.3.1 Γενική Περιγραφή Ευριστικών Κανόνων

Οι 4 κανόνες που ορίσαμε χρησιμοποιούν πλήρως τα στοιχεία που επιστρέφει η αναζήτηση στην Yahoo .

Ο πρώτος κανόνας επεξεργάζεται τον τίτλο του κάθε αποτελέσματος που επιστρέφει η Yahoo σε σχέση με τον τίτλο του αποτελέσματος της αναζήτησης του Minersoft. Ο δεύτερος κανόνας επεξεργάζεται την περιγραφή του κάθε αποτελέσματος που επιστρέφει η Yahoo, ο τρίτος το URI και ο τέταρτος λαμβάνει υπόψη την κατάταξη των αποτελεσμάτων της Yahoo.

5.3.2 Ανάλυση Ευριστικών Κανόνων

Για την αξιολόγηση του κάθε αποτελέσματος εισαγάγαμε 4 απλούς κανόνες στους οποίους αναθέσαμε κάποια βάρη ώστε να δίνουμε περισσότερη βαρύτητα στα σκορ τα οποία καθορίζουν μια καλύτερη επιλογή για το χρήστη.
Οι κανόνες αυτοί λειτουργούν ως εξής:

- **Κανόνας 1:** Περνούμε το Levenshtein Distance για κάθε λέξη του τίτλου του αποτελέσματος της Yahoo με τη λιγότερη διαφορά και διαιρούμε με το μήκος της λέξης του ερωτήματος αφού πάντα μια λέξη θα έχουμε για ερώτημα στην Yahoo καθώς η λέξη αναζήτησης είναι ο τίτλος του αποτελέσματος του Minersoft. Το ποσοστό που προκύπτει το αφαιρούμε από την μονάδα για να βρούμε το μεγαλύτερο σκορ. Έτσι ώστε αν ένας τίτλος δεν περιέχει τη λέξη αλλά κάποια παράγωγό της και όχι ακριβώς αυτή, τότε πάλι θα πετύχει κάποιο

σκορ, ώστε όταν η λέξη που αναζητήθηκε δεν βρίσκεται ολόκληρη στον τίτλο να εξακολουθεί να μπορεί να εξάγει το καλύτερο δυνατό συμπέρασμα.

Score = (weight)* max(1-(Levenshtein's score(query , title word)/query length)).

- **Κανόνας 2:** Μετρούμε πόσες φορές προέκυψε η λέξη που αναζητούμε στην περιγραφή που επιστρέφει η Yahoo. Μετά απλά διαιρούμε αυτόν τον αριθμό με τον συνολικό αριθμό λέξεων που υπάρχουν στην περιγραφή. Ακολούθως πολλαπλασιάζουμε με το επιθυμητό βάρος.

  Score = (weight) *(occurrences/ words number in description).

- **Κανόνας 3:** Στον κανόνα αυτό γίνεται έλεγχος αν υπάρχει η λέξη του ερωτήματος κάπου μέσα στον URI που επιστρέφει για κάθε αποτέλεσμα η Yahoo και ανάλογα με το πού βρίσκεται παίρνει και διαφορετικό σκορ. Θεωρούμε ότι αν βρίσκεται στο authority μέρος του URI ότι είναι κάποιο σημαντικό λογισμικό με δική του ιστοσελίδα και γι' αυτό παίρνει όλο το βάρος που δόθηκε σε αυτό τον κανόνα ως σκορ. Αν όμως η λέξη του ερωτήματος δεν βρίσκεται μέσα στο authority μέρος του URI αλλά κάπου πιο μετά παίρνει μόνο το μισό βάρος ως σκορ. Τέλος αν δεν βρέθηκε η λέξη κάπου στο URI τότε το σκορ είναι 0.

- **Κανόνας 4:** Ο τελευταίος κανόνας λαμβάνει υπόψη τη σειρά με την οποία επιστρέφει η μηχανή αναζήτηση της Yahoo τα αποτελέσματα. Για παράδειγμα αν ένα αποτέλεσμα της Yahoo προηγείται κάποιου άλλου σημαίνει ότι έχει συγκεντρώσει πιο ψηλό σκορ. Έτσι αφού εμείς συγκρίνουμε τα 20 πρώτα αποτελέσματα το κάθε ένα από αυτά παίρνει διαφορετικό σκορ το οποίο μειώνεται αναλόγως της θέσης του αποτελέσματος.

  Score = (weight)* (1- ((1/20) * I-th place from top 20 results)).

Κεφάλαιο 6

Λεπτομέρειες Υλοποίησης

Στο κεφάλαιο αυτό θα αναφερθούμε στον τρόπο σχεδίασης και υλοποίησης των διαδικασιών που αναπτύξαμε με σκοπό να επεκτείνουν τη μηχανή αναζήτησης του Minersoft. Είναι σημαντικό να δείξουμε τον τρόπο με τον οποίο συσχετίζονται οι κλάσεις των διαδικασιών, καθώς και τα χαρακτηριστικά και τις μεθόδους που απαρτίζουν την κάθε κλάση. Αυτό γιατί θέλουμε να διατηρήσουμε την επεκτασιμότητα της δομής του Minersoft.

6.1 Σχεδιασμός διαδικασιών

6.1.1 Διαγράμματα UML (Unified Modeling Language)

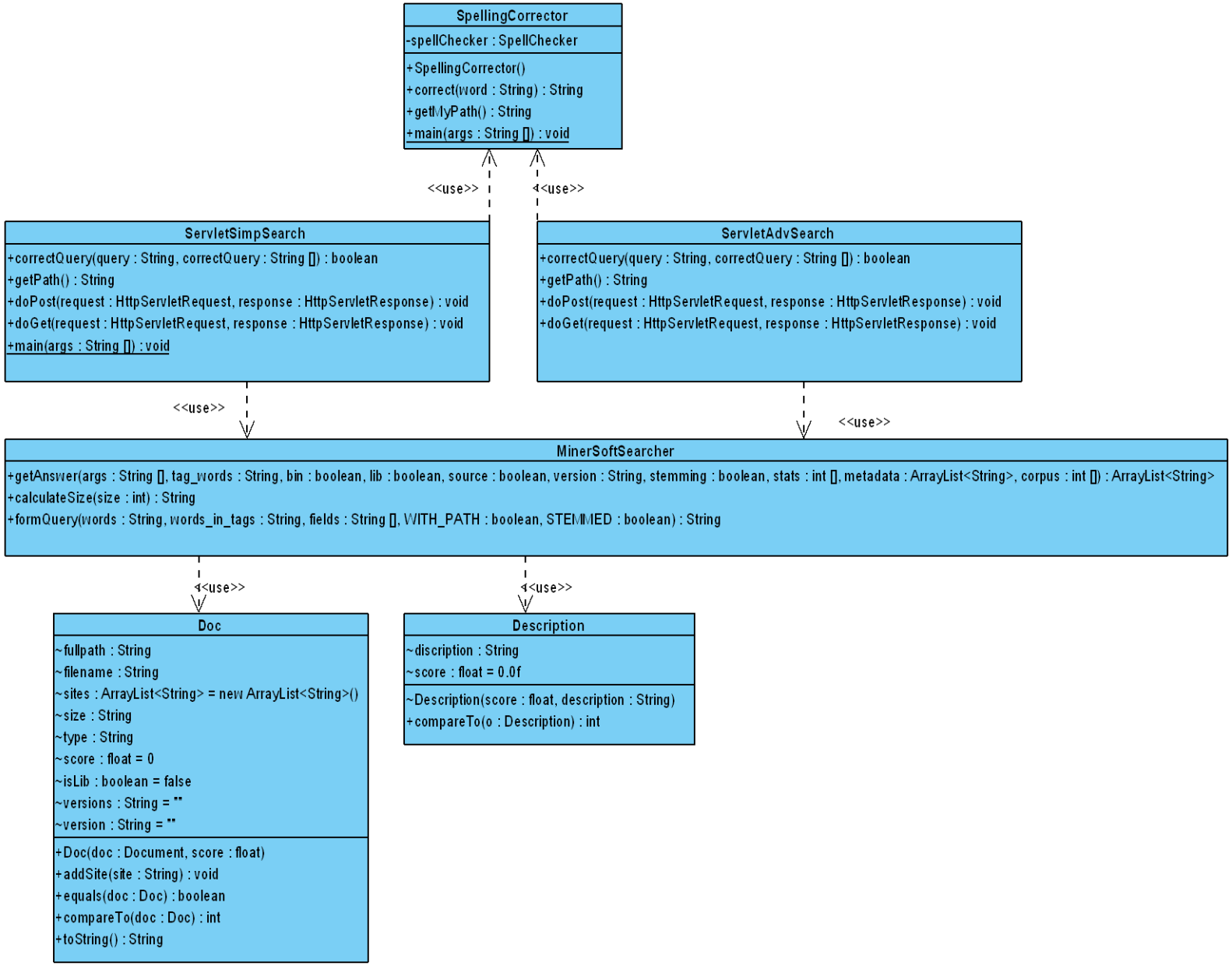
Η ενοποιημένη γλώσσα σχεδιασμού (*unified modeling language*) (UML) ορίζεται ως μια γραφική γλώσσα για την αναπαράσταση, τη διαμόρφωση προδιαγραφών και την τεκμηρίωση συστημάτων που βασίζονται σε λογισμικό. Η UML στοχεύει αποκλειστικά στο σχεδιασμό αντικειμενοστρεφών συστημάτων.

Εμείς θα κάνουμε χρήση της UML και συγκεκριμένα των διαγραμμάτων κλάσεων (class diagrams), όπως είπαμε και στην εισαγωγή, με σκοπό να δείξουμε τον τρόπο με τον οποίο συσχετίζονται οι κλάσεις των διαδικασιών, καθώς και τα χαρακτηριστικά και τις μεθόδους που απαρτίζουν την κάθε κλάση.

6.1.2 Διαδικασία αναζήτησης

Οι σημαντικότερες κλάσεις που απαρτίζουν τη διαδικασία της αναζήτησης όπως αναπαριστάται και στο σχήμα (6.1.2.1) είναι οι κλάσεις: ServletSimpSearch, ServletAdvSearch, SpellingCorector, MinersoftSearcher, Doc, Discription. Οι δύο πρώτες κλάσεις είναι υπεύθυνες για την επικοινωνία της γραφικής διεπαφής με τον υπόλοιπο java κώδικα. Συγκεκριμένα, η ServletSimpSearch αναλαμβάνει τη διαδικασία της απλής αναζήτησης, ενώ η ServletAdvSearch τη διαδικασία της εξειδικευμένης αναζήτησης. Αυτές οι κλάσεις χρησιμοποιούν τις κλάσεις SpellingCorector και MinersoftSearcher. Η κλάση SpellingCorector χρησιμοποιείται έχοντας ως σκοπό να διασφαλίζει την ορθογραφία των λέξεων – κλειδιών που εισάγουν οι χρήστες για εκτέλεση μιας απλής ή εξειδικευμένης αναζήτησης. Η κλάση MinersoftSearcher είναι

υπεύθυνη για την εκτέλεση της αναζήτησης και την επιστροφή των αποτελεσμάτων στους χρήστες της μηχανής αναζήτησης Minersoft, διαμέσου των κλάσεων ServletSimpSerch και ServletAdvSearch. Τέλος, οι κλάσεις Doc και Discription είναι συνυφασμένες με την προετοιμασία των αποτελεσμάτων μετά την εκτέλεση της αναζήτησης στην κλάση MinersoftSearcher. Αφού, δηλαδή, επιστραφούν τα αποτελέσματα από μια αναζήτηση τροποποιούνται ώστε να μπορούν να τύχουν διαχείρισης πιο εύκολα από τις μετέπειτα κλάσεις που θα χρησιμοποιήσουν τις πληροφορίες αυτές.

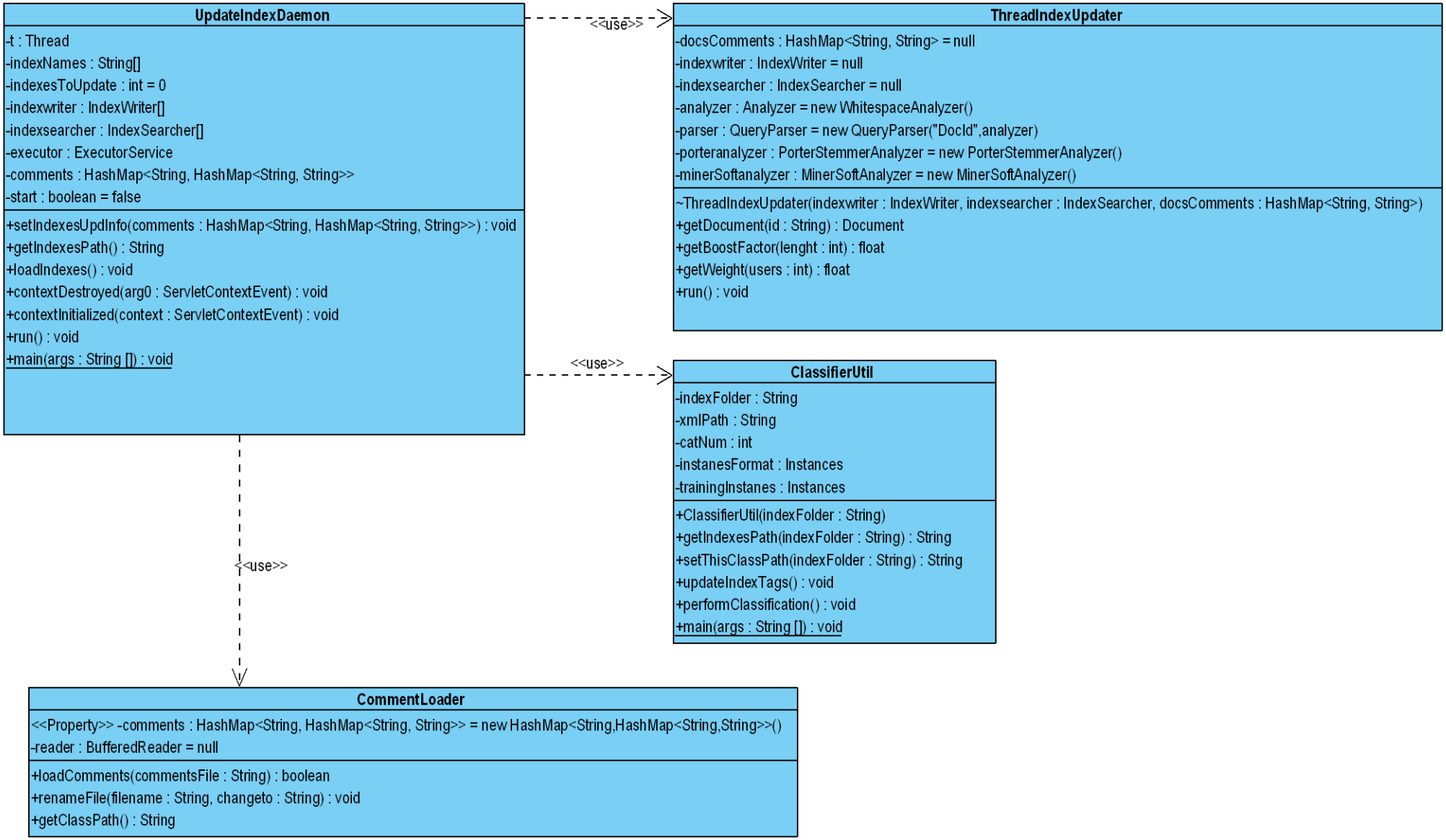


Σχήμα 6.1.2.1: Διάγραμμα κλάσεων για την διαδικασία της αναζήτησης

6.1.3 Διαδικασία ανανέωσης των ανεστραμμένων ευρετηρίων

Η διαδικασία της ανανέωσης των ανεστραμμένων ευρετηρίων, είναι μια συνεχής διαδικασία κατά την οποία τα ανεστραμμένα ευρετήρια μπορούν να ανανεωθούν με νέες πληροφορίες που προέρχονται απευθείας από τους χρήστες της μηχανής αναζήτησης ή από τον ταξινομητή ο οποίος – όπως περιγράψαμε στο Κεφάλαιο 4 – έχει ταξινομήσει σε διαφορές κατηγορίες τα αρχεία λογισμικού.

Όπως παρατηρούμε και από το σχήμα (6.1.3.1), στη διαδικασία αυτή περιλαμβάνονται 4 κλάσεις οι οποίες είναι: η UpdateIndexDeamon, η ThreadIndexUpdater, η ClassifierUtil και η CommentLoader. Η κλάση UpdateIndexDeamon είναι η υπεύθυνη κλάση για τη διασφάλιση της συνεχούς ανανέωσης των ευρετηρίων, η οποία χρησιμοποιεί τις υπόλοιπες τρεις για να το πετύχει αυτό. Συγκεκριμένα, χρησιμοποιεί την κλάση CommentLoader για να διαβάσει τις μέχρι στιγμής υπάρχουσες ετικέτες μέσα από το προσωρινό αρχείο κειμένου. Ακολούθως τις πληροφορίες αυτές τις αποστέλλει στην κλάση ThreadIndexUpdater με σκοπό τη μόνιμη αποθήκευση τους στα ανεστραμμένα ευρετήρια. Επίσης η κλάση UpdateIndexDeamon χρησιμοποιεί την κλάση ClassifierUtil για να εκπαιδεύσει σε πρώτο στάδιο τον ταξινομητή και μετά να ταξινομήσει όλα τα αρχεία λογισμικού των ευρετηρίων στις διάφορες κατηγορίες που προέκυψαν. Με το πέρας της ταξινόμησης, κάθε αρχείο λογισμικού ανανεώνει τις πληροφορίες του προσθέτοντας τις κατηγορίες που του έχουν ανατεθεί από τον ταξινομητή.



Σχήμα 6.1.3.1: Διάγραμμα κλάσεων για την διαδικασία της ανανέωσης των ευρετηρίων

6.1.4 Δημιουργία δεδομένων εκπαίδευσης ταξινομητή

Όπως έχουμε ήδη αναφέρει η κλάση ClassifierUtil είναι υπεύθυνη για τη δημιουργία των δεδομένων εκπαίδευσης του ταξινομητή, για την εκπαίδευση του ταξινομητή και για την ταξινόμηση των αρχείων λογισμικού. Γι' αυτές τις τρεις λειτουργίες χρησιμοποιεί τις κλάσεις InstanceIndexer, CategoryFinder, TrainInstanceCreator, ClassificationClass και XMLInstanceManipulator.

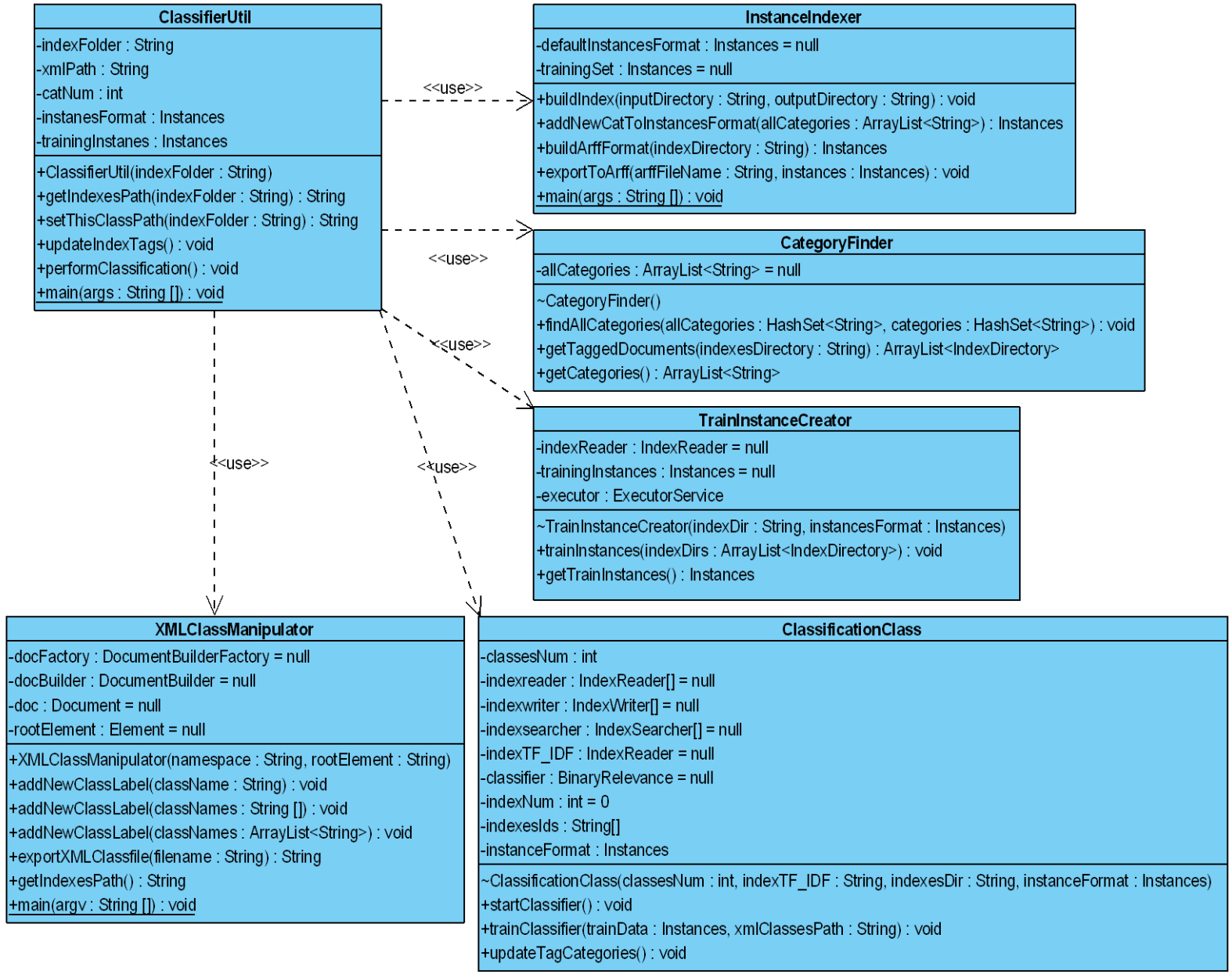
Η κλάση InstanceIndexer είναι υπεύθυνη να δημιουργήσει ένα ευρετήριο το οποίο απαρτίζεται μόνο από τα αρχεία τύπου *binary, library script* και *source.* Αυτό γιατί θέλουμε να γνωρίζουμε πληροφορίες σχετικά με το βάρος TF-IDF των όρων που περιέχουν αυτά τα αρχεία. Ακολούθως απομονώνει τα αρχεία τα οποία τους έχουν προστεθεί ετικέτες από τους χρήστες με σκοπό αυτά να μετατραπούν σε κατοπινό στάδιο σε στιγμιότυπα εκπαίδευσης.

Η κλάση CategoryFinder είναι υπεύθυνη να εντοπίσει τις διαφορετικές ετικέτες που υπάρχουν στα δεδομένα εκπαίδευσης και να τις μετατρέψει σε κατηγορίες βάσει των οποίων θα εκτελείται η εκπαίδευση του μοντέλου του ταξινομητή και η ταξινόμηση των δεδομένων. Στο σχήμα (6.1.4.2) φαίνεται η διαδικασία αυτή, καθώς η κλάση ThreadTagFinder χρησιμοποιεί την κλάση IndexDirectory σε συνδυασμό με την κλάση IndexDocument για να εντοπίσει τις διαφορετικές ετικέτες που υπάρχουν στα ανεστραμμένα ευρετήρια

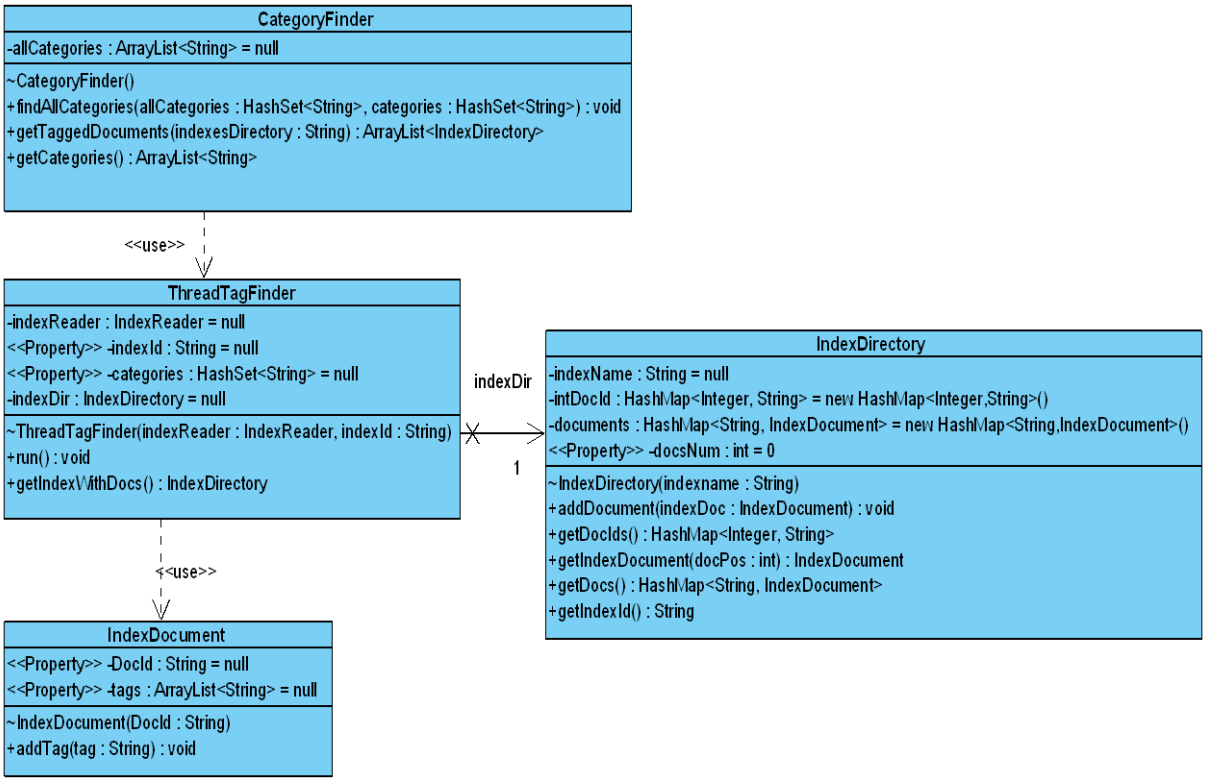
Αφού λοιπόν έχει δημιουργηθεί η μορφή των δεδομένων εκπαίδευσης, η κλάση TrainInstanceCreator θα καλεστεί να δημιουργήσει όλα τα στιγμιότυπα που αποτελούν τα δεδομένα εκπαίδευσης. Η κλάση TrainInstanceCreator, χρησιμοποιώντας την κλάση ThreadTrainInstanceCreator, δημιουργεί ξεχωριστά για κάθε ανεστραμμένο ευρετήριο τα δεδομένα εκπαίδευσης. Κάθε ένα στιγμιότυπο από αυτά τα δεδομένα αναπαριστάται με την κλάση TrainInstance.

Με το πέρας της δημιουργίας των δεδομένων εκπαίδευσης γίνεται η δημιουργία ενός XML αρχείου, με την χρήση της κλάσης XMLInstanceManipulator, με τις διαφορετικές κατηγορίες που βρέθηκαν.

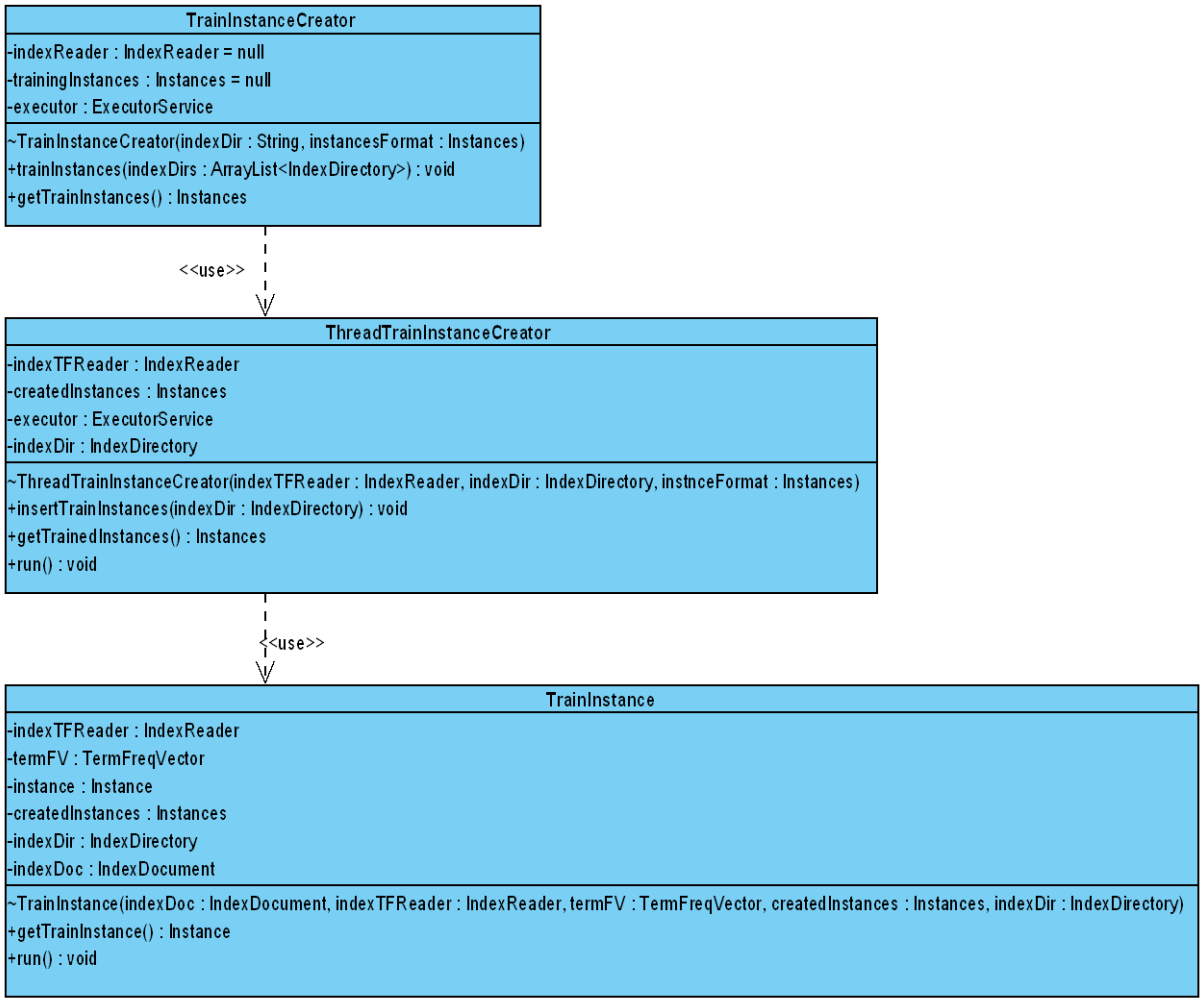
Τέλος, η κλάση ClassifierUtil θα χρησιμοποιήσει την κλάση ClassificationClass για να εκπαιδεύσει τον ταξινομητή και φυσικά να ταξινομήσει τα αρχεία λογισμικού των ανεστραμμένων ευρετηρίων. Το σχήμα (6.1.4.4) παρουσιάζει τη διαδικασία αυτή.



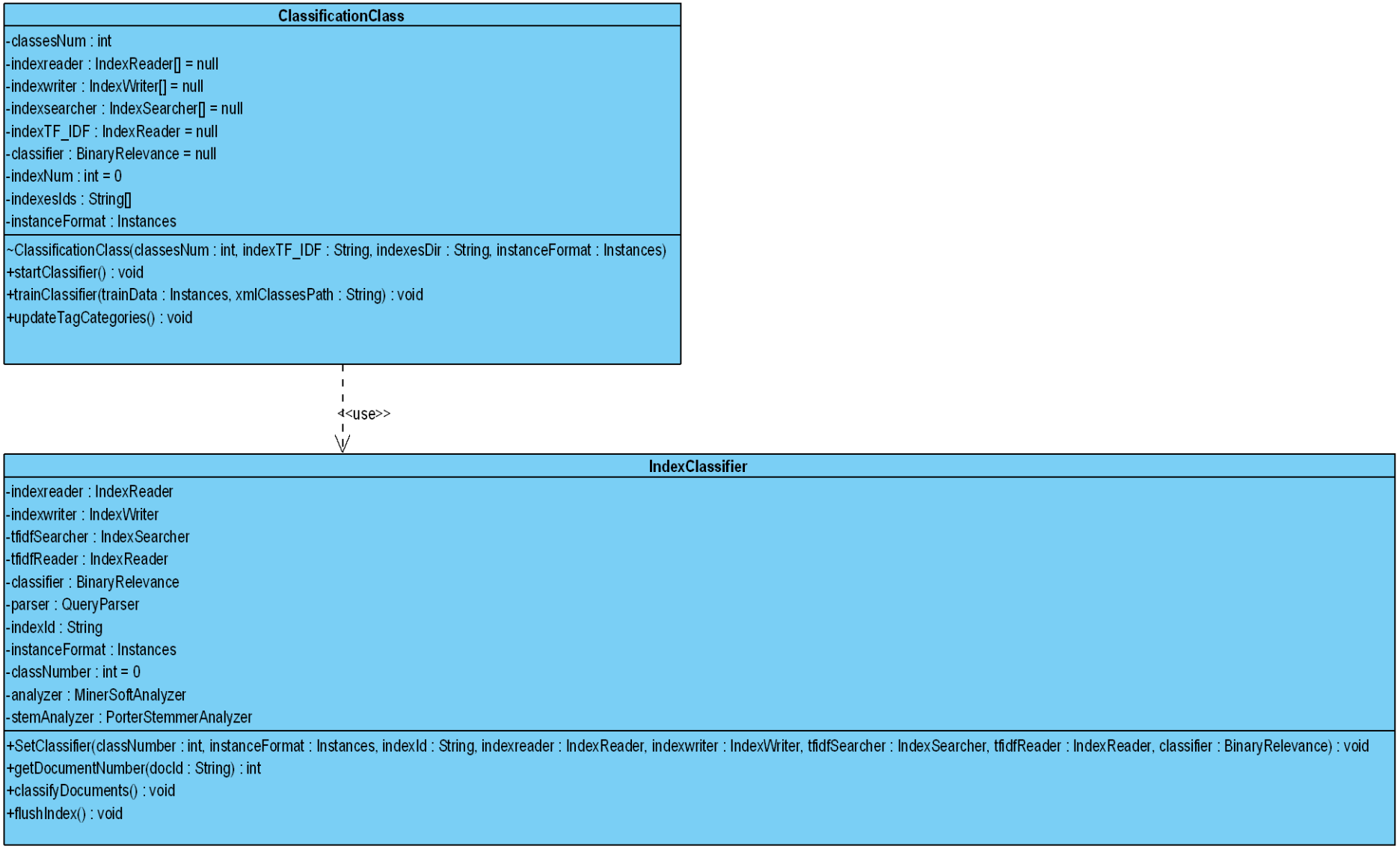
Σχήμα 6.1.4.1: Διάγραμμα κλάσεων για τη διαδικασία της δημιουργίας δεδομένων εκπαίδευσης ταξινομητή



Σχήμα 6.1.4.2: Διάγραμμα κλάσεων για τη διαδικασία της ανεύρεσης των κατηγοριών



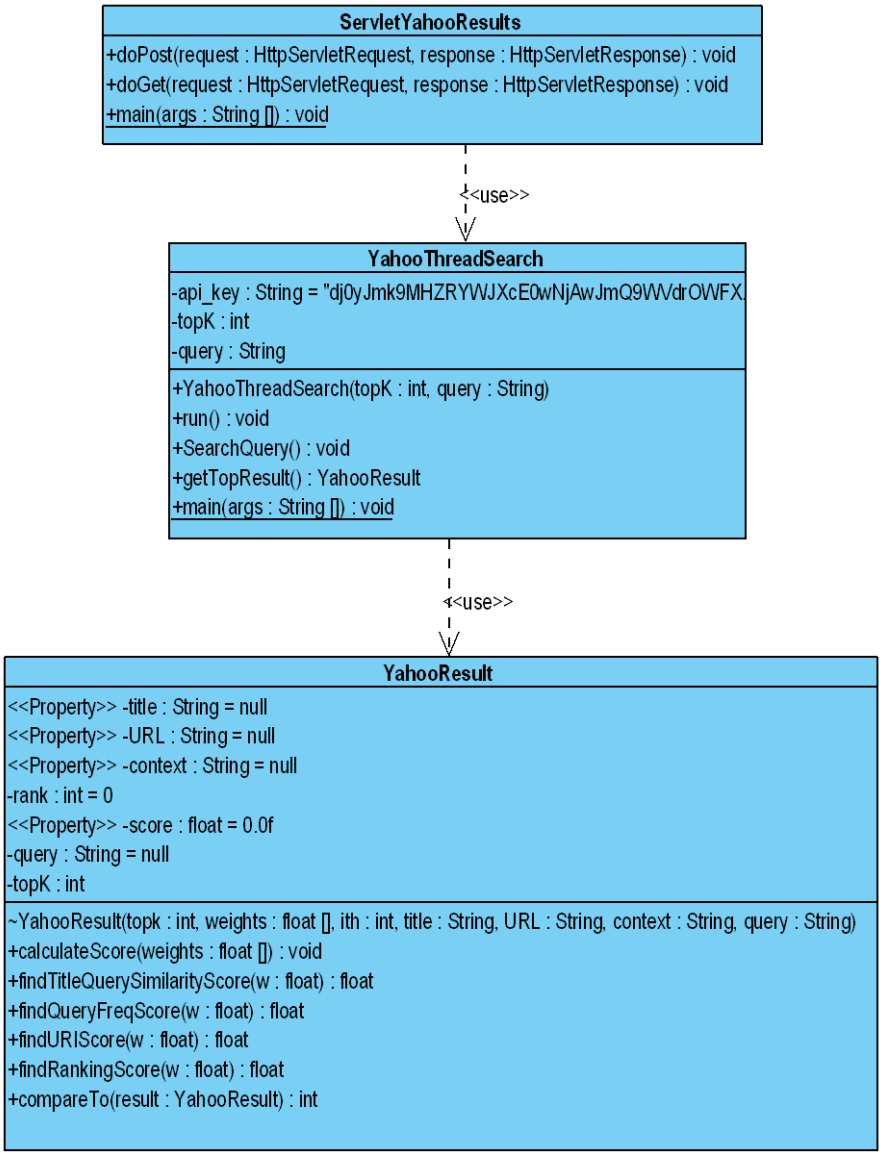
Σχήμα 6.1.4.3: Διάγραμμα κλάσεων για τη διαδικασία της δημιουργίας στιγμιότυπων



Σχήμα 6.1.4.4: Διάγραμμα κλάσεων για τη διαδικασία της ταξινόμησης

6.1.5 Διαδικασία εμπλουτισμού πληροφοριών από το διαδίκτυο

Οι τρεις κλάσεις που αντιπροσωπεύουν τη διαδικασία αυτή είναι η ServletYahooResults, η YahooThreadSearch και η YahooResult. Η πρώτη είναι υπεύθυνη να μεταφέρει τις πληροφορίες από τη γραφική διεπαφή στον κώδικα java και χρησιμοποιεί την κλάση YahooThreadSearch. Αυτή είναι υπεύθυνη να εκτελέσει τις αναζητήσεις για κάθε αποτέλεσμα της μηχανής αναζήτησης Minersoft και να επιστρέψει τα αποτελέσματα στην ServletYahooResults. Τέλος, η κλάση YahooResult χρησιμοποιείται από την YahooThreadSearch για να συγκρίνει κάθε αποτέλεσμα της μηχανής αναζήτησης Yahoo, αναδεικνύοντας το πιο σχετικό.

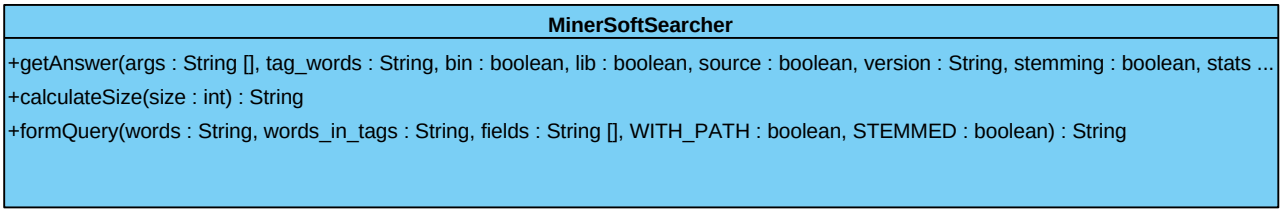


Σχήμα 6.1.5.1: Διάγραμμα κλάσεων για τη διαδικασία εμπλουτισμού πληροφοριών από το διαδίκτυο

6.2 Περιγραφή υλοποιημένων κλάσεων

Σε αυτήν την υποενότητα θα περιγραφούν συνοπτικά οι πιο κύριες κλάσεις από αυτές που αναφέρθηκαν στην προηγούμενη υποενότητα και συγκεκριμένα θα δοθεί περισσότερη έμφαση στις ιδιότητες και τις μεθόδους τους που τις αποτελούν. Οι ιδιότητες και οι μέθοδοι θα περιγραφούν με την σειρά με την οποία εμφανίζονται στα αντίστοιχα σχήματα.

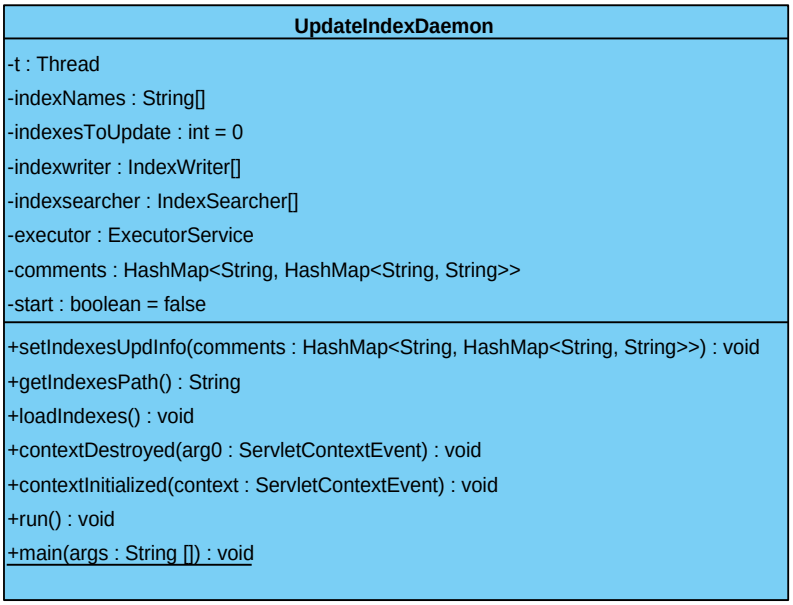
Κλάση MinersoftSearcher



Σχήμα 6.2.1: Η κλάση MinerSoftSearcher

Στο σχήμα (6.2.1) παρουσιάζεται η κλάση MinerSoftSearcher η οποία απαρτίζεται από τρεις μεθόδους την getAnswer η οποία είναι υπεύθυνη να εκτελεί την αναζήτηση στα ανεστραμμένα ευρετήρια του Minersoft επιστρέφοντας τα αποτελέσματα, την calculateSize η οποία μετασχηματίζει σε KB, MB, GB το μέγεθος του κάθε αρχείου λογισμικού που επιστρέφεται σε κάθε αποτέλεσμα της αναζήτησης και η formQuery η οποία μετατρέπει τα δεδομένα αναζήτησης σε μορφή η οποία είναι δεκτή από τις κλάσεις αναζήτησης της Lucene.

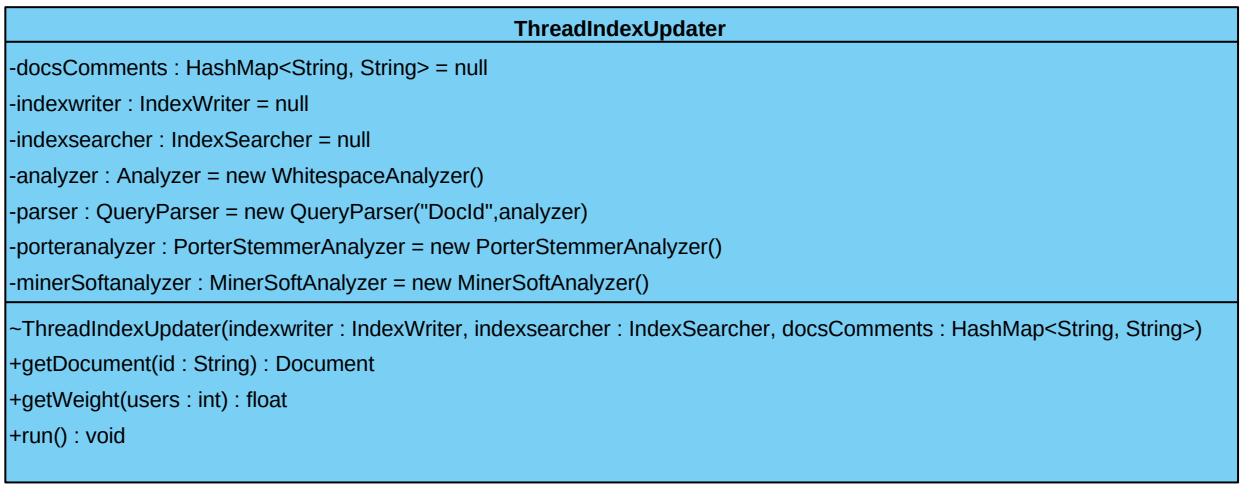
Κλάση UpdateIndexDeamon



Σχήμα 6.2.2: Η κλάση UpdateIndexDeamon

Στο σχήμα (6.2.2) παρουσιάζεται η κλάση UpdateIndexDeamon. Οι ιδιότητες της κλάσης αυτής είναι το νήμα συνεχούς εκτέλεσης της κλάσης, ο πίνακας με τα ονόματα των ευρετηρίων, ο αριθμός των ευρετηρίων που θα ανανεωθούν στην παρούσα φάση, ο πίνακας με τους Index Writer ένα για κάθε ευρετήριο, ο πίνακας με τους Index Searcher ένα για κάθε ευρετήριο, ένας Executor για να εκτελεί τα νήματα της κλάσης ανανέωσης των ευρετηρίων, η δομή με τις ετικέτες των χρηστών και η ένδειξη έναρξης νήματος. Οι μέθοδοι της κλάσης αυτής είναι οι εξής: η μέθοδος setIndexesUpdInfo η οποία αρχικοποιεί τις ιδιότητες της κλάσης UpdateIndexDeamon με τα νέα δεδομένα κάθε φορά, η getIndexPath η οποία εντοπίζει τον κατάλογο των ανεστραμμένων ευρετηρίων, η μέθοδος loadIndexes η οποία φορτώνει τα ανεστραμμένα ευρετήρια στον Index Writer και στον Index Searcher και οι μέθοδοι contextDestroyed, contectextInitialized και run που είναι οι κληρονομημένες μέθοδοι της κλάσης Thread.

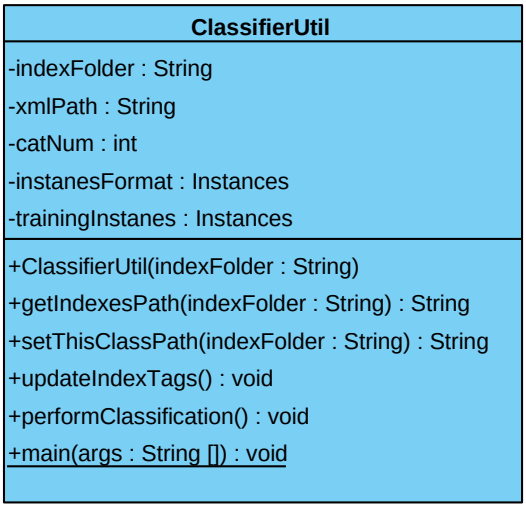
Κλάση ThreadIndexUpdater



Σχήμα 6.2.3: Η κλάση ThreadIndexUpdater

Στο σχήμα (6.2.3) παρουσιάζεται η κλάση ThreadIndexUpdater. Οι ιδιότητες αυτές είναι η δομή docsComments όπου ως κλειδί αποθηκεύεται ο αριθμός του αρχείου στο ανεστραμμένο ευρετήριο και ως πληροφορία οι ετικέτες των χρηστών, ο Index Writer και Index Searcher του συγκεκριμένου ανεστραμμένου ευρετηρίου που ανέλαβε να ανανεώσει η κλάση αυτή, ο analyzer ο οποίος είναι ο αναλυτής που θα χρησιμοποιηθεί από τον Index Writer, ο parser ο οποίος μετατρέπει τις αναζητήσεις των αρχείων βάσει το αριθμού τους στο ανεστραμμένο ευρετήριο στη μορφή που είναι αποδεκτή από τη Lucene, ο analyzer που εκτελεί στελέχωση κειμένου και θα χρησιμοποιηθεί στην ανάλυση των ετικετών των χρηστών και ο analyzer που δεν εκτελεί στελέχωση κειμένου και θα χρησιμοποιηθεί στην ανάλυση των ετικετών των χρηστών. Επιπλέον, η κλάση αυτή αποτελείται από τον κατασκευαστή της, τη μέθοδο getDocument η οποία εκτελεί αναζήτηση πάνω στο ανεστραμμένο ευρετήριο βάσει του αριθμού του αρχείου και επιστρέφει το αρχείο, την μέθοδο getWeight η οποία είναι υπεύθυνη να υπολογίζει το βάρος που έχει η ζώνη για τις ετικέτες στο κάθε αρχείο και που χρησιμοποιείται στην αναζήτηση στο ανεστραμμένο ευρετήριο και η run μέθοδος η οποία υπάρχει σε κάθε νήμα και εξυπηρετεί την ανανέωση ενός ανεστραμμένου ευρετηρίου παράλληλα με τα υπόλοιπα νήματα της ίδιας κλάσης.

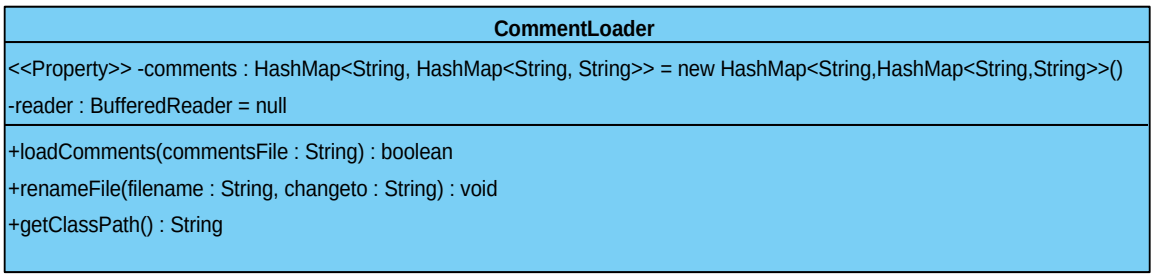
Κλάση ClassifierUtil



Σχήμα 6.2.4: Η κλάση ClassifierUtil

Στο σχήμα (6.2.4) παρουσιάζεται η κλάση ClassifierUtil. Οι ιδιότητες της κλάσης αυτής είναι το όνομα του καταλόγου στον οποίο υπάρχουν τα ανεστραμμένα ευρετήρια, το μονοπάτι όπου θα αποθηκευτεί το αρχείο XML με τις ετικέτες και συνεπώς τις ξεχωριστές κατηγορίες που εντοπίστηκαν, ο αριθμός των κατηγοριών που θα εντοπιστούν, η μορφή που θα έχει το αρχείο ***arff*** και τα δεδομένα εκπαίδευσης που θα εντοπιστούν. Επίσης η κλάση αυτή αποτελείται από τον κατασκευαστή της και τις μεθόδους getIndexesPath η οποία εντοπίζει το μονοπάτι που οδηγεί στον κατάλογο με τα ανεστραμμένα ευρετήρια, setThisClassPath η οποία εντοπίζει το μονοπάτι για την συγκεκριμένη κλάση μέσα στον εξυπηρετητή, updateIndexTags η οποία είναι υπεύθυνη να εντοπίσει τα αρχεία λογισμικού τα οποία περιέχουν ετικέτες από τους χρήστες και να τα μετατρέψει σε δεδομένα εκπαίδευσης, και η performClassiication η οποία είναι υπεύθυνη να εκτελέσει την ταξινόμηση.

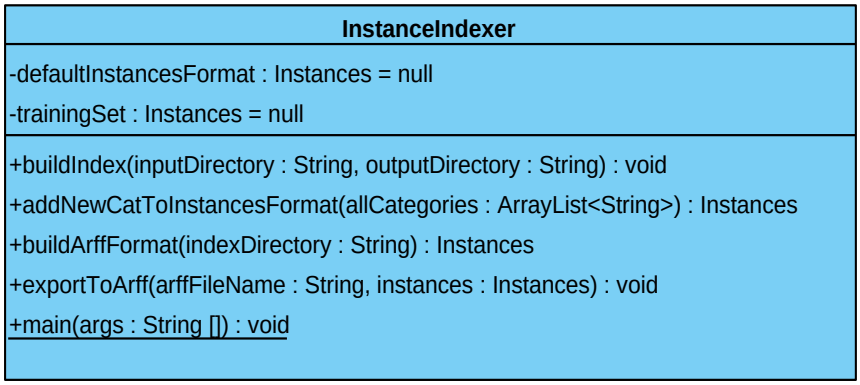
Κλάση CommentLoader



Σχήμα 6.2.5: Η κλάση CommentLoader

Στο σχήμα (6.2.5) παρουσιάζεται η κλάση CommentLoader. Οι ιδιότητες αυτής της κλάσης είναι η δομή comments στην οποία φορτώνονται οι πληροφορίες σχετικά με το ποια ευρετήρια πρέπει να ανανεωθούν, ποια αρχεία σε αυτά τα ευρετήρια και το ποιες ετικέτες αντιστοιχούν σε κάθε αρχείο, ο reader χρησιμοποιείται για να διαβάσει τις ετικέτες από το προσωρινό αρχείο και να τις περάσει στην δομή comments. Η κλάση αυτή αποτελείται από τη μέθοδο loadComments η οποία είναι υπεύθυνη να φορτώσει τις πληροφορίες αυτές στην δομή comments, τη μέθοδο renameFile η οποία διαχειρίζεται τα προσωρινά αρχεία αποθήκευσης των ετικετών και τέλος, τη μέθοδο getClassPath που είναι υπεύθυνη να επιστρέψει το μονοπάτι όπου βρίσκεται η κλάση αυτή μέσα στον εξυπηρετητή ώστε να εντοπιστούν τα προσωρινά αρχεία κειμένου που περιέχουν τις ετικέτες των χρηστών.

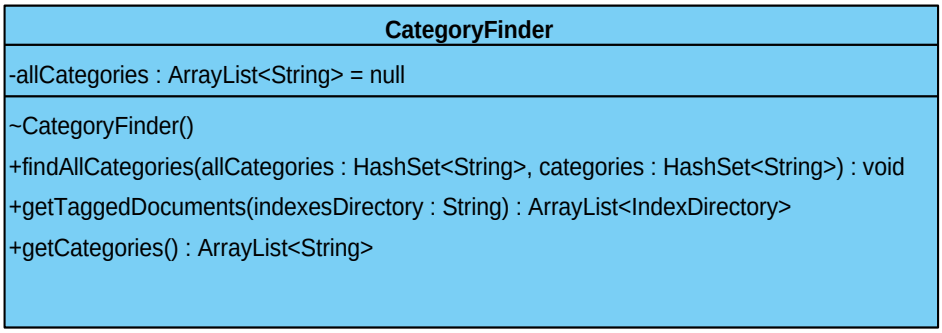
Κλάση InstanceIndexer



Σχήμα 6.2.6: Η κλάση InstanceIndexer

Στο σχήμα (6.2.6) παρουσιάζεται η κλάση InstanceIndexer. Η κλάση InstanceIndexer έχει ως ιδιότητες την default μορφή των στιγμιότυπων του ***arff*** αρχείου, δηλαδή ξέρει τα χαρακτηριστικά και τις κατηγορίες. Επιπλέον, στις ιδιότητες υπάρχουν τα δεδομένα εκπαίδευσης επίσης σε μορφή αρχείου ***arff.*** Στις μεθόδους της κλάσης αυτής βρίσκεται η μέθοδος builtIndex η οποία είναι υπεύθυνη να δημιουργεί ένα ανεστραμμένο ευρετήριο από όλα τα ανεστραμμένα ευρετήρια με μόνο τους όρους που συσχετίζονται με αρχεία λογισμικού, η μέθοδος addNewCatToInstaesFormat η οποία τοποθετεί τις κατηγορίες που υπολογίστηκαν μέσα στο αρχείο ***arff***. Επιπλέον, η μέθοδος builtArffFormat μετασχηματίζει όλα τα δεδομένα εκπαίδευσης σε αρχείο ***arff*** ενώ η μέθοδος exportToArff εξάγει τα δεδομένα αυτά στον κατάλογο της κλάσης αυτής.

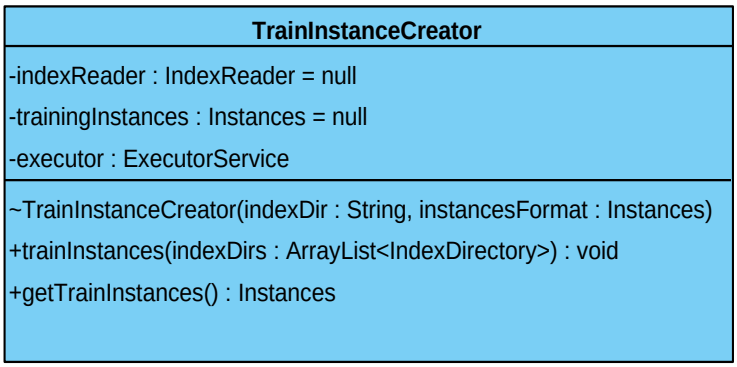
Κλάση CategoryFinder



Σχήμα 6.2.7: Η κλάση CategoryFinder

Στο σχήμα (6.2.7) παρουσιάζεται η κλάση CategoryFinder. Η κλάση CategoryFinder έχει στις ιδιότητες της μόνο τη δομή όπου θα αποθηκεύονται τα ονόματα των διαφορετικών κατηγοριών που θα εντοπιστούν. Η κλάση CategoryFinder απαρτίζεται από τον κατασκευαστή της, από τη μέθοδο findAllCategories η οποία εντοπίζει τις διαφορετικές κατηγορίες, από τη μέθοδο getTaggedDocuments η οποία εντοπίζει τα αρχεία από κάθε ανεστραμμένο ευρετήριο τα οποία τους έχουν προστεθεί ετικέτες και η μέθοδος getCategories επιστρέφει τις κατηγορίες.

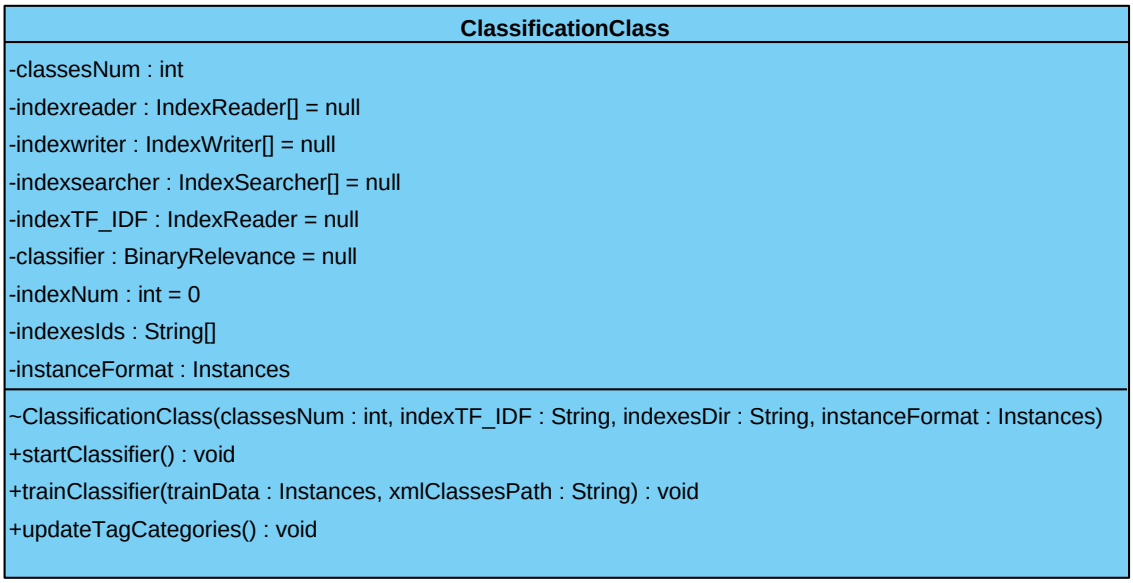
Κλάση TrainInstanceCreator



Σχήμα 6.2.8: Η κλάση TrainInstanceCreator

Στο σχήμα (6.2.8) παρουσιάζεται η κλάση TrainInstanceCreator. Η κλάση TrainInstanceCreator έχει ως ιδιότητες έναν Index Reader, τα δεδομένα εκπαίδευσης και έναν εκτελεστή νημάτων. Στις μεθόδους υπάρχει ο κατασκευαστής της κλάσης, trainInstances η οποία βάσει του ανεστραμμένου ευρετηρίου εντοπίζει τα αρχεία που θα χρησιμοποιηθούν στην εκπαίδευση του ταξινομητή και μέσω της μεθόδου getTrainInstnces τα επιστρέφει.

Κλάση ClassificationClass



Σχήμα 6.2.9: Η κλάση ClassificationClass

Στο σχήμα (6.2.8) παρουσιάζεται η κλάση ClassificationClass. Η κλάση αυτή έχει τις εξής ιδιότητες: αριθμός των κατηγοριών που εντοπίστηκαν σε προηγούμενο στάδιο, Index Reader, Writer και Searcher για τα ανεστραμμένα ευρετήρια, ένας Index Reader για το ανεστραμμένο ευρετήριο με τις πληροφορίες για το TF-IDF κάθε όρου, τον ταξινομητή που θα χρησιμοποιηθεί, ο αριθμός των ευρετηρίων, ο πίνακας με τις ταυτότητες των ευρετηρίων και η μορφή του αρχείου ***arff.*** Ακολουθούν ο κατασκευαστής της κλάσης ClassificationClass και οι μέθοδοι startClassifier η οποία αρχικοποιεί τον κατασκευαστή, trainClassifier η οποία είναι υπεύθυνη να εκπαιδεύσει τον ταξινομητή και updateTegCategories η οποία είναι υπεύθυνη να ταξινομήσει τα αρχεία λογισμικού.

Κεφάλαιο 7

Αξιολόγηση

Στο Κεφάλαιο 4 μιλήσαμε για τις προτεινόμενες μεθόδους ταξινόμησης των αρχείων λογισμικού σε κατηγορίες οι οποίες θα εισέρχονται από τους χρήστες διαμέσου της μηχανής αναζήτησης, ως ετικέτες (tags).

Σε αυτό το Κεφάλαιο θα συγκρίνουμε αυτές τις ήδη υλοποιημένες μεθόδους της βιβλιοθήκης Mulan[8,9], χρησιμοποιώντας δεδομένα που δημιουργήθηκαν από το Minersoft και να επιλέξουμε την πιο ιδανική μέθοδο για την διαδικασία της ταξινόμησης των αρχείων λογισμικού.

7.1 Δεδομένα αξιολόγησης μεθόδων

7.1.1 Δημιουργία δεδομένων

Τα δεδομένα τα οποία θα χρησιμοποιηθούν προέρχονται από τον Virtual Organization [1] atlas, πάνω στον οποίο βρίσκεται εγκατεστημένα μια πληθώρα λογισμικών από τους χρήστες του EGEE Grid[1]. Με την χρήση του Minersoft δημιουργήσαμε ανεστραμμένο ευρετήριο από το VO Atlas με τα χαρακτηριστικά που αναγράφονται στον πίνακα 7.1 και το οποίο χρησιμοποιήθηκε για την δημιουργία δεδομένων αξιολόγησης.

	Συνολικό μέγεθος	Συνολικός αριθμών αρχείων λογισμικού (bin, lib, script, source)
ATLAS (VO)	**22GB**	**124586**

Πίνακας 7.1: Στοιχειά VO Atlas

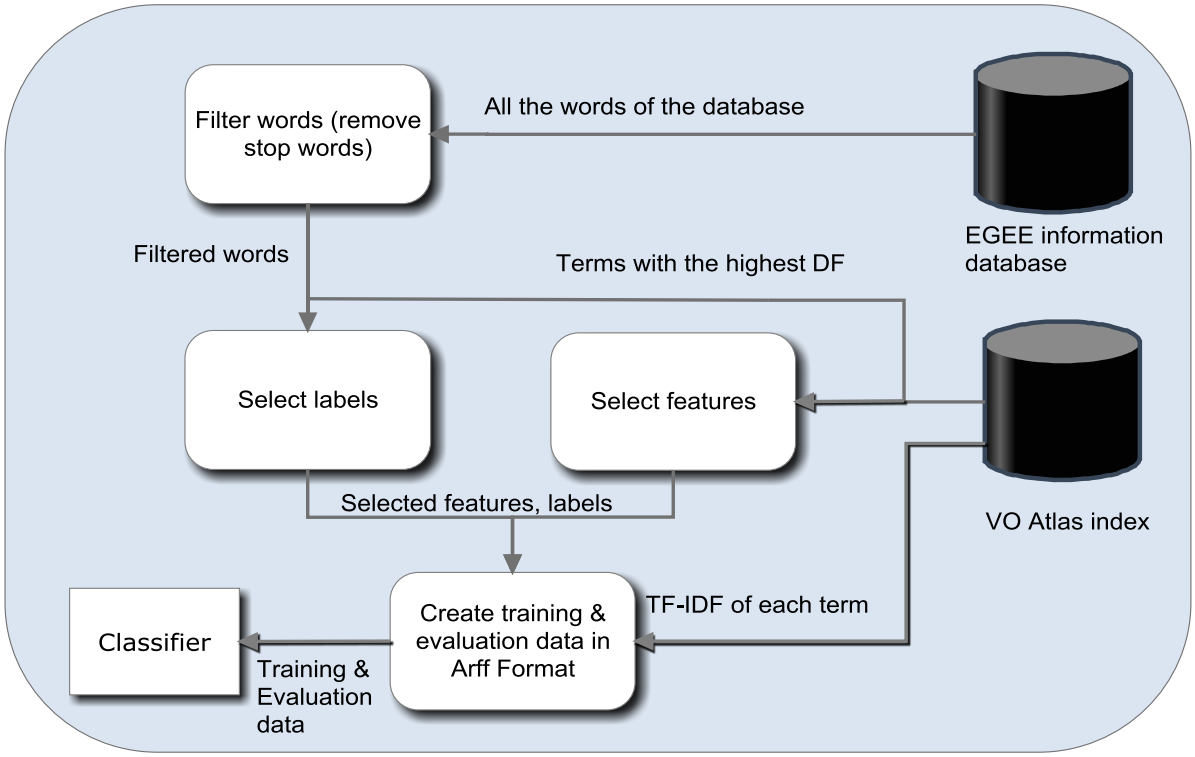
Στο σημείο αυτό θα οριστούν και θα επεξηγηθούν κάποιες βασικές έννοιες τις οποίες θα αναφέρουμε συχνά στο Κεφάλαιο αυτό.

- **Στιγμιότυπα:** Στιγμιότυπα ονομάζουμε τα αρχεία λογισμικού τα οποία θα χρησιμοποιηθούν στην εκπαίδευση των μοντέλων των μεθόδων ταξινόμησης. Κάθε ένα στιγμιότυπο περιέχει κάποιες πληροφορίες που θα χρησιμοποιηθούν από τις μεθόδους ταξινόμησης.
- **Χαρακτηριστικά:** Χαρακτηριστικά, είναι τα στοιχεία τα οποία επιλέγονται μέσα από τις πληροφορίες που περιέχει κάθε ένα στιγμιότυπο. Για την επιλογή των ιδανικότερων αυτών χαρακτηριστικών, επιλέξαμε τους όρους οι οποίοι

έχουν υψηλότερο Document Frequency μέσα σε ολόκληρο το σύνολο των αρχείων λογισμικού. Σε κάθε χαρακτηριστικό ανατίθεται το βάρος που έχει σε κάθε διαφορετικό στιγμιότυπο. Το βάρος αυτό είναι η τιμή TF-IDF του χαρακτηριστικού στο συγκεκριμένο στιγμιότυπο.

- **Ετικέτες**: Με τον όρο ετικέτες εννοούμε τις πιθανές κατηγορίες στις οποίες μπορεί να ανήκει ένα στιγμιότυπο και όπως επεξηγήθηκε στο Κεφάλαιο 4 οι ετικέτες αυτές είναι οι ετικέτες οι οποίες αναθέτονται από τους χρήστες της μηχανής αναζήτησης σε κάθε αρχείο λογισμικού.

Για την αξιολόγηση των αλγορίθμων χρησιμοποιήθηκε ένα μικρότερο υποσύνολο των αρχείων που ευρετηρίασε το Minersoft, της τάξης των 10000 αρχείων λογισμικού. Ωστόσο, σε αυτά τα αρχεία δεν έχουν ανατεθεί ετικέτες από χρήστες. Για να προσομοιώσουμε λοιπόν όσο το δυνατό καλύτερα την αξιολόγηση πάνω σε πραγματικά δεδομένα θα αναθέσουμε σε αυτά ετικέτες, λέξεις τις οποίες ανακτήσαμε από μια βάση δεδομένων η οποία περιέχει πληροφορίες για εφαρμογές στο EGEE Grid[1], όπως τον πλήρη τίτλο της εφαρμογής, τη θεματική κατηγορία στην οποία ανήκει και μια γενική περιγραφή της κάθε εφαρμογής. Ακολούθως, θα συσχετιστούν οι περιγραφικές λέξεις αυτές με αρχεία λογισμικού στα οποία έχουν εντοπιστεί ως όροι, για παράδειγμα, μια λέξη που υπάρχει στην βάση αυτή η οποία περιγράφει λογισμικό φυσικής είναι η λέξη ***ενέργεια*** αν λοιπόν αυτή η λέξη περιέχεται και σε ένα αρχείο λογισμικού τότε θεωρείται ετικέτα στην περίπτωση απουσίας τέτοιων λέξεων επιλέγονται λέξεις με υψηλό Document Frequency και οι οποίες επίσης υπάρχουν μέσα στο αρχείο λογισμικού. Έτσι θεωρούμε ότι το κάθε στιγμιότυπο θα περιγράφεται καλύτερα από τέτοιες λέξεις όπως ακριβώς πιστεύουν και οι χρήστες όταν εισάγουν τις δικές τους περιγραφικές λέξεις. Η διαδικασία της δημιουργίας δεδομένων αξιολόγησης αναπαριστάται στο σχήμα (7.1.1.1).



Σχήμα 7.1.1.1: Στο σχήμα αυτό αναπαριστάται η διαδικασία δημιουργίας δεδομένων αξιολόγησης.

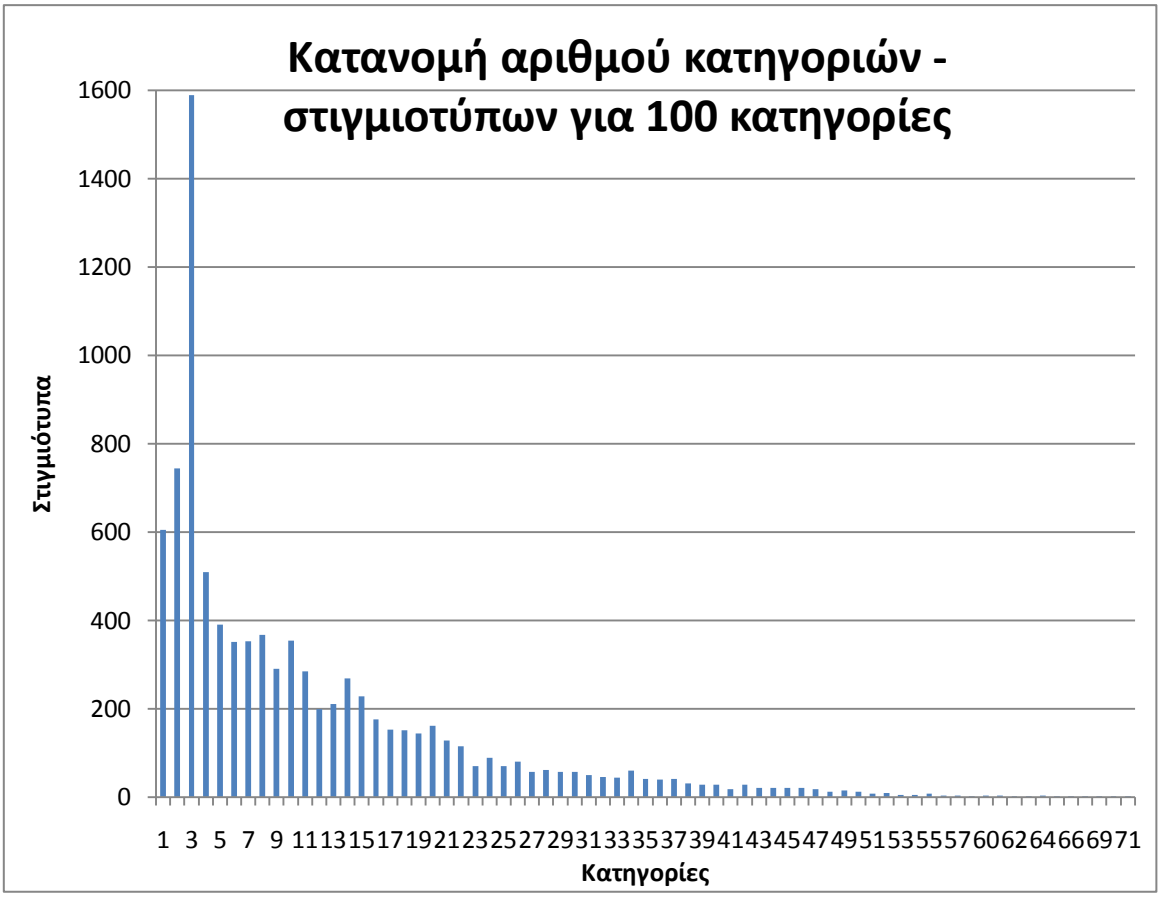
7.1.2 Παρουσίαση δεδομένων αξιολόγησης

Πιο κάτω στον πίνακα (7.1.2.1) βλέπουμε τα 5 διαφορετικά σύνολα στιγμιότυπων που συλλέξαμε από τον (VO) Atlas και διαφέρουν στον αριθμό των χαρακτηριστικών και τον αριθμό των κατηγοριών, αλλά είναι δημιουργημένα από το ίδιο υποσύνολο των 10000 στιγμιότυπων που περιγράφουν αρχεία λογισμικού και λήφθηκαν με τυχαίο τρόπο από το ανεστραμμένο ευρετήριο του (VO) Atlas. Είναι σημαντικό να αξιολογηθούν οι μέθοδοι ταξινόμησης στιγμιότυπων σε δεδομένα όσο το δυνατό πιο όμοια με τα πραγματικά, γι' αυτό πρέπει να υπάρχει ποικιλία όσον αφορά την ποσότητα των κατηγοριών που περιγράφουν κάθε ένα στιγμιότυπο. Για να επιβεβαιώσουμε αν αυτό ισχύει με τα δεδομένα μας, προχωρήσαμε στη δημιουργία τριών ιστογραμμάτων (7.1.2.1, 7.1.2.2, 7.1.2.3) τα οποία ακολουθούν παρακάτω και επιβεβαιώνουν την ανομοιόμορφη κατανομή των κατηγοριών στα στιγμιότυπα.

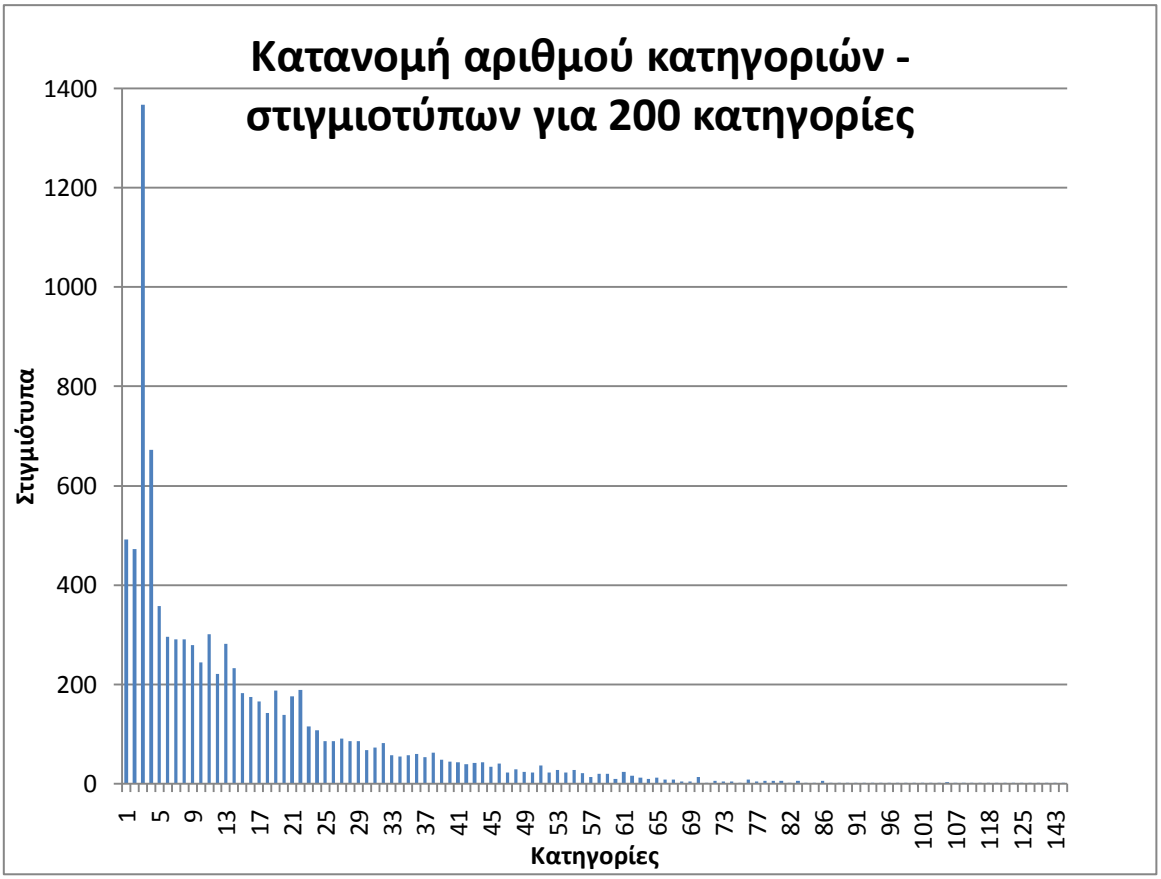
Σύνολο στιγμιότυπων από (VO) Atlas	D_1	D_2	D_3	D_4	D_5
Αριθμός στιγμιότυπων	10000	10000	10000	10000	10000
Αριθμός χαρακτηριστικών	1000	1000	1000	500	2000
Αριθμός κατηγοριών	100	200	400	200	200
Μέσος όρος κατηγοριών ανά στιγμιότυπο	11.06	15.43	19.84	15.43	15.43
Διάμεσος κατηγοριών	7	10	13	10	10
Μέγιστος αριθμός κατηγοριών	71	145	246	145	145
Ελάχιστος αριθμός κατηγοριών	1	1	1	1	1
Πιθανοί συνδυασμοί κατηγοριών	2^{100}	2^{200}	2^{400}	2^{200}	2^{200}
Συνδυασμοί κατηγοριών που παρατηρήθηκαν	5014	5660	6026	5660	5660

Πίνακας 7.1.2.1: Χαρακτηριστικά του συνόλου πειραμάτων των στιγμιότυπων

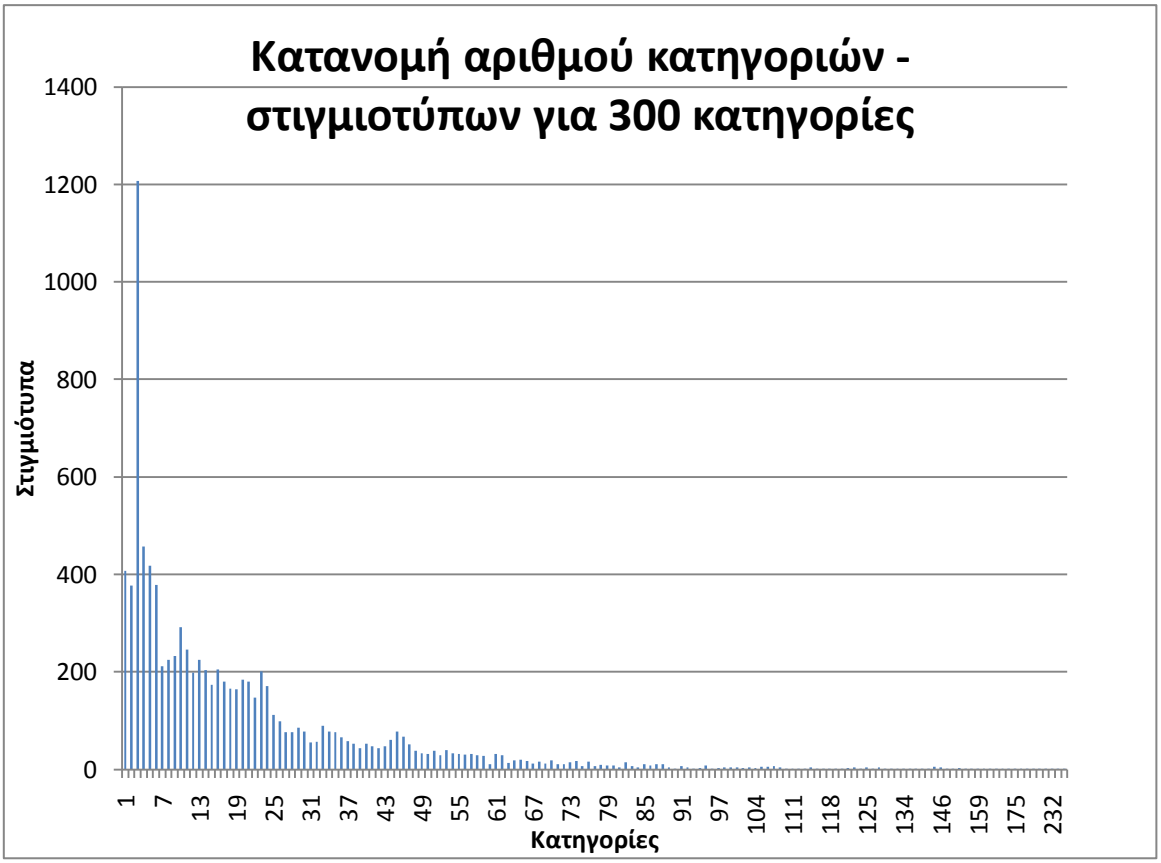
Μερικές παρατηρήσεις που προκύπτουν από την παρατήρηση του πίνακα (7.1.2.1) είναι ότι οι συνδυασμοί των κατηγοριών που παρατηρούνται σε όλα τα σύνολα στιγμιότυπων είναι πολύ λιγότεροι από τους πιθανούς συνδυασμούς. Παρ' όλα αυτά, για το μέγεθος του συνόλου των στιγμιότυπων το οποίο είναι της τάξης των 10000 στιγμιότυπων, οι συνδυασμοί σε όλα τα σύνολα στιγμιότυπων ξεπερνούν τις 5000 διαφορετικούς συνδυασμούς.



Γράφημα 7.1.2.1: Ιστόγραμμα κατανομής αριθμού κατηγοριών – στιγμιότυπων για το σύνολο D_1



Γράφημα 7.1.2.2: Ιστόγραμμα κατανομής αριθμού κατηγοριών – στιγμιότυπων για το σύνολο D_2



Γράφημα 7.1.2.3: Ιστόγραμμα κατανομής αριθμού κατηγοριών – στιγμιότυπων για το σύνολο D_3

Όπως παρατηρούμε από τα ιστογράμματα που παρουσιάζονται στα γραφήματα (7.1.2.1), (7.1.2.2) και (7.1.2.3) ο αριθμός των κατηγοριών ακολουθεί long-tail κατανομή. Αυτό σημαίνει ότι ο αριθμός των κατηγοριών δεν περιορίζεται σε κάποιο σταθερό πεδίο τιμών, αλλά είναι διαφορετικός για κάθε υποσύνολο στιγμιότυπων. Επίσης παρατηρούμε ότι τα περισσότερα στιγμιότυπα και στα τρία ιστογράμματα έχουν μικρότερους αριθμούς κατηγοριών.

7.2 Αξιολόγηση μεθόδων ταξινόμησης

7.2.1 Μετρικές αξιολόγησης

Για την αξιολόγηση των μεθόδων ταξινόμησης BR (Binary Relevance), LP (Label Powerset), RAKEL (RAndom k-labELsets), MLkNN (MultiLabel-kNN), BPMLL

(Backpropagation Multi-Label Learning) τις οποίες περιγράψαμε στο Κεφάλαιο 4, θα χρησιμοποιηθούν οι μετρικές Hamming Loss και Micro-averaged F-Measure οι οποίες προσφέρονται από την βιβλιοθήκη Mulan [8,9] όπως και οι πιο πάνω μέθοδοι. Επίσης η αξιολόγηση περιλαμβάνει συγκρίσεις σε απόδοση χρόνου αλλά και κατανάλωση σε μνήμη για την εκπαίδευση και ταξινόμηση του μοντέλου του κάθε ενός ταξινομητή.

Hamming Loss

Η μετρική αξιολόγησης Hamming Loss [10] είναι μια μετρική η οποία χρησιμοποιείται για να υπολογίσει πόσες φορές ένα ζεύγος στιγμιότυπου-κατηγορίας έχει κατηγοριοποιηθεί εσφαλμένα. Συγκεκριμένα, για κάποιο στιγμιότυπο, βλέπει αν έχει κατηγοριοποιηθεί σε μια κατηγορία στην οποία δεν ανήκει ή αν δεν έχει κατηγοριοποιηθεί σε κάποια κατηγορία στην οποία ανήκει. Όταν η μετρική Hamming Loss υπολογίζει μικρότερες τιμές, υποδηλώνει ότι έχει γίνει καλύτερη ταξινόμηση στιγμιότυπων ενώ όταν $\text{hloss}_S(h) = 0$ υποδηλώνει ότι έχει γίνει τέλεια ταξινόμηση. Στον τύπο (7.2.1.1) βλέπουμε τη μετρική Hamming Loss η οποία θα επεξηγηθεί περιληπτικά. Έστω ότι X είναι ένα σύνολο από στιγμιότυπα και το σύνολο $\mathcal{Y} = \{\lambda_1, \lambda_2, \lambda_3, \ldots \lambda_{|\mathcal{Y}|}\}$ είναι το σύνολο των κατηγοριών στις οποίες πιθανόν να ανήκουν τα στιγμιότυπα του X. Το σύνολο $T = \{(\chi_1, Y_1), (\chi_2, Y_2) \ldots (\chi_{|T|}, Y_{|T|})\}$ είναι το σύνολο των στιγμιότυπων για την αξιολόγηση του μοντέλου ενός ταξινομητή, όπου το στιγμιότυπο $\chi_i \in X$ και το υποσύνολο των κατηγοριών που περιγράφουν το στιγμιότυπο αυτό $Y_i \subseteq \mathcal{Y}$. Το Δ αντιπροσωπεύει την συμμετρική διαφορά μεταξύ δύο συνόλων ενώ το $h(x_i)$ είναι η μέθοδος ταξινόμησης που χρησιμοποιείται.

$$\text{hloss}_S(h) = \frac{1}{|T|} \sum_{i=1}^{|T|} \frac{1}{|\mathcal{Y}|} |h(x_i) \Delta Y_i|$$

Τύπος 7.2.1.1: Τύπος Hamming Loss

Micro-averaged F-Measure

Το Micro-averaged F-Measure [18,19] είναι μια ακόμα μετρική την οποία θα χρησιμοποιήσουμε για να αξιολογήσουμε τις 5 μεθόδους που προαναφέραμε βάσει των στιγμιότυπων που δημιουργήσαμε από το (VO) Atlas.

Οι αποφάσεις των μεθόδων ταξινόμησης πάνω στο σύνολο T που αναφέραμε πιο πάνω, χωρίζονται σε 3 κατηγορίες οι οποίες αναπαρίστανται στον πίνακα (7.2.2.1).

Αριθμός στιγμιότυπων που ανατέθηκαν σωστά στην κατηγορία i.	Αριθμός στιγμιότυπων που δεν ανήκουν στην κατηγορία i αλλά εσφαλμένα ταξινομήθηκαν σε αυτή.	Αριθμός στιγμιότυπων που ανήκουν στην κατηγορία i αλλά δεν ταξινομήθηκαν σε αυτή.
True Positives (TPi)	False Positives (FPi)	False Negatives (FNi)

Πίνακας 7.2.2.1: Περιγραφή True Positives, False Positives, False Negatives**.**

Από αυτές προκύπτουν δύο μετρικές απαραίτητες για τον υπολογισμό του Micro-averaged F-Measure, το Precision και το Recall. Το πρώτο εκφράζει το πόσες από τις κατηγορίες που ανατέθηκαν στο κάθε στιγμιότυπο ανατέθηκαν σωστά, ενώ το δεύτερο εκφράζει το πόσες από τις κατηγορίες που έπρεπε να ανατεθούν σε ένα στιγμιότυπο, έχουν ανατεθεί. Αυτό διαφαίνεται και από τους τύπους τους στο (7.2.1.2) όπου C είναι ο συνολικός αριθμός των κατηγοριών στις οποίες μπορεί να ανήκουν τα στιγμιότυπα.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{\sum_{i=1}^{C} TPi}{\sum_{i=1}^{C} (TPi + FPi)}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{\sum_{i=1}^{C} TPi}{\sum_{i=1}^{C} (TPi + FNi)}$$

Τύπος 7.2.1.2: Precision, Recall

Το Micro-averaged F-Measure λοιπόν υπολογίζεται καθολικά πάνω από το σύνολο των κατηγοριών, με σκοπό να δώσει εξισορροπημένη βαρύτητα στο Precision και το Recall υπολογίζοντας τον αρμονικό μέσο τους βάσει τον τύπο (7.2.1.3). Οι τιμές που προκύπτουν από την μετρική αυτή βρίσκονται στο διάστημα [0,1] ενώ ψηλότερες τιμές δηλώνουν καλύτερη ταξινόμηση.

$$\text{Micro-averaged F-Measure} = \frac{\mathbf{2 \times Precision \times Recall}}{\mathbf{Precision + Recall}}$$

Τύπος 7.2.1.3: Micro-averaged F-Measure

CPU time

Ο χρόνος πού χρειάζεται κάθε μέθοδος από τις 5 αυτές μεθόδους να εκπαιδεύσει το μοντέλο της, καθώς και να ταξινομήσει τα στιγμιότυπα που ανήκουν στο σύνολο δεδομένων αξιολόγησης.

Memory Consumption

Η μνήμη πού χρειάζεται κάθε μέθοδος για να εκπαιδεύσει το μοντέλο της, καθώς και να ταξινομήσει τα στιγμιότυπα που ανήκουν στο σύνολο δεδομένων αξιολόγησης.

7.2.2 Σχολιασμός αποτελεσμάτων

Στα πειράματα τα οποία ακολουθούν αυτό που θέλαμε να εξετάσουμε είναι ποια από τις μεθόδους ταξινόμησης πετυχαίνει καλύτερα αποτελέσματα σε Hamming Loss, Micro-averaged F-Measure, χρόνο εκπαίδευσης και ταξινόμησης του μοντέλου των 5 διαφορετικών μεθόδων και την απόδοση σε μνήμη, καθώς χρησιμοποιείται στη διαδικασία ταξινόμησης των αρχείων λογισμικού της μηχανής αναζήτησης Minersoft. Επιπρόσθετα, θέλαμε να εξετάσουμε ποία είναι η επίδραση που έχει η αύξηση του αριθμού των χαρακτηριστικών βάσει των οποίων γίνεται και η ταξινόμηση αλλά και του αριθμού των κατηγοριών, καθώς στη διαδικασία της ταξινόμησης βάσει των ετικετών των χρηστών ο αριθμός των κατηγοριών θα μεταβάλλεται. Επιπλέον για τις μεθόδους BR, LP και RAKEL οι οποίες χρησιμοποιούν ως base learners ταξινομητές μονής ετικέτας, πειραματιστήκαμε με 3 διαφορετικούς base learners, τους J48, Naïve Bayes και SMO.

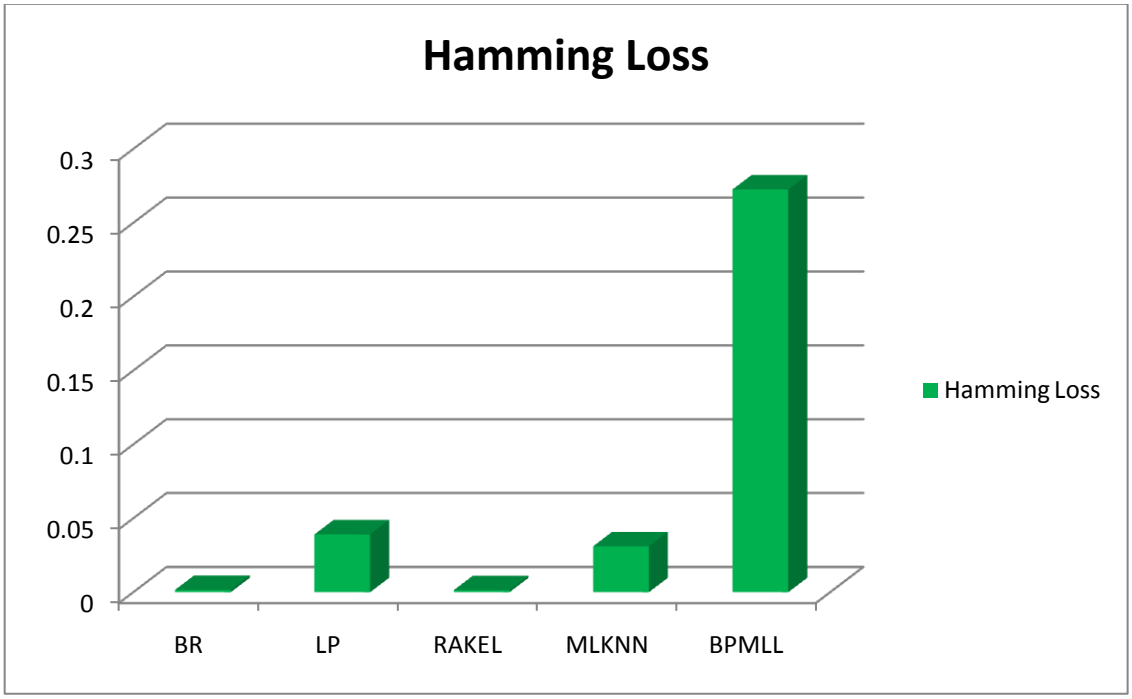
7.2.2.1 Πειράματα με μεταβολή στον αριθμό των χαρακτηριστικών

Πειράματα για 500 χαρακτηριστικά 200 κατηγορίες

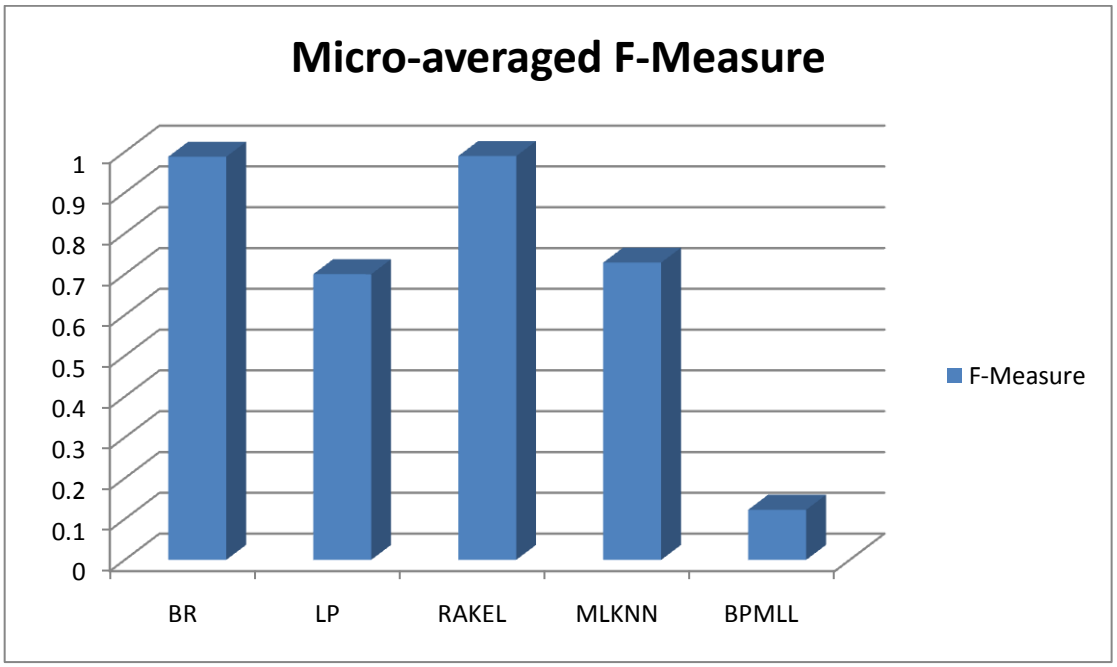
	BR	**LP**	**RAKEL**	**MLKNN**	**BPMLL**
Hamming Loss	0.0016	0.0391	**0.0014**	0.0311	0.2726
Micro-averaged F-Measure	0.9879	0.7001	**0.9896**	0.7285	0.1228
CPU time	845.78	**217.59**	5152.2	351.39	846.75

Memory consumption	**1108.69**	2311.977	1924.661	1902.244	1308.627

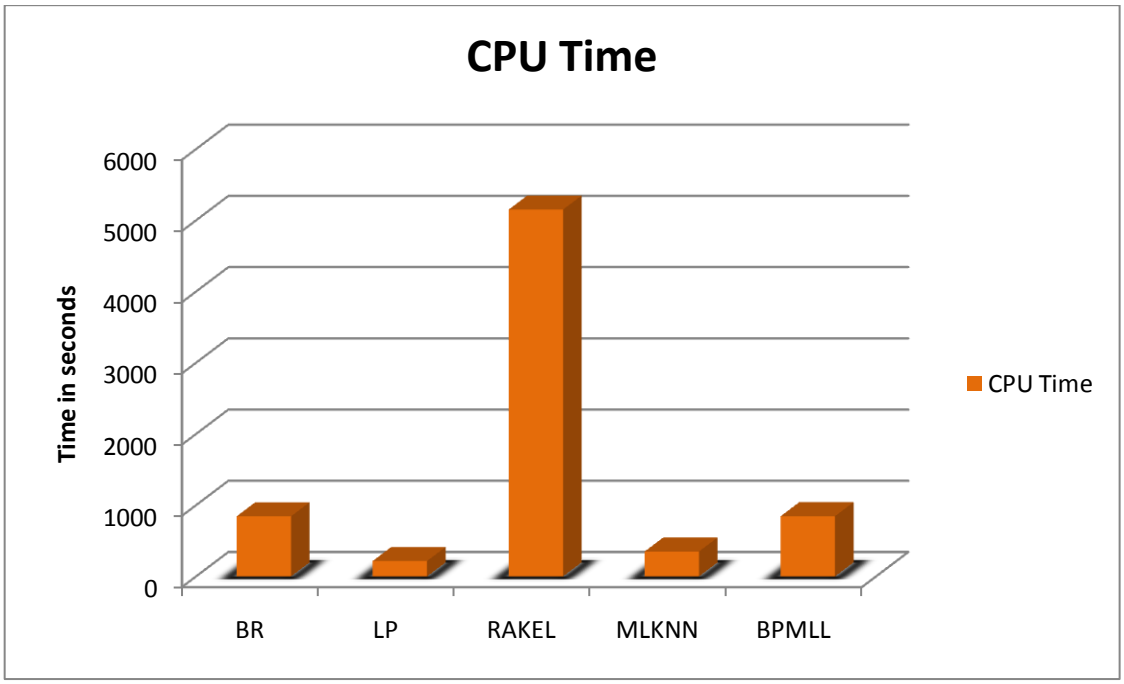
Πίνακας 7.2.2.1.1: Μετρήσεις που έχουν ληφθεί από πείραμα στο σύνολο στιγμιότυπων D_4



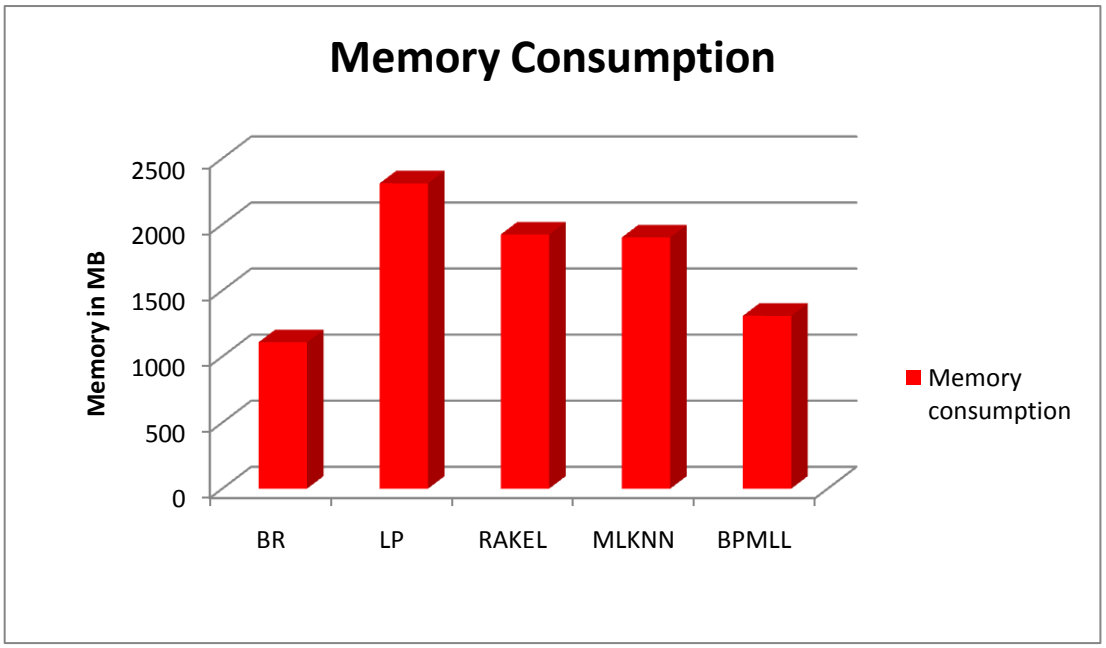
Γράφημα 7.2.2.1.1: Hamming Loss των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



Γράφημα 7.2.2.1.2: Micro-averaged F-Measure των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



Γράφημα 7.2.2.1.3: CPU time των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



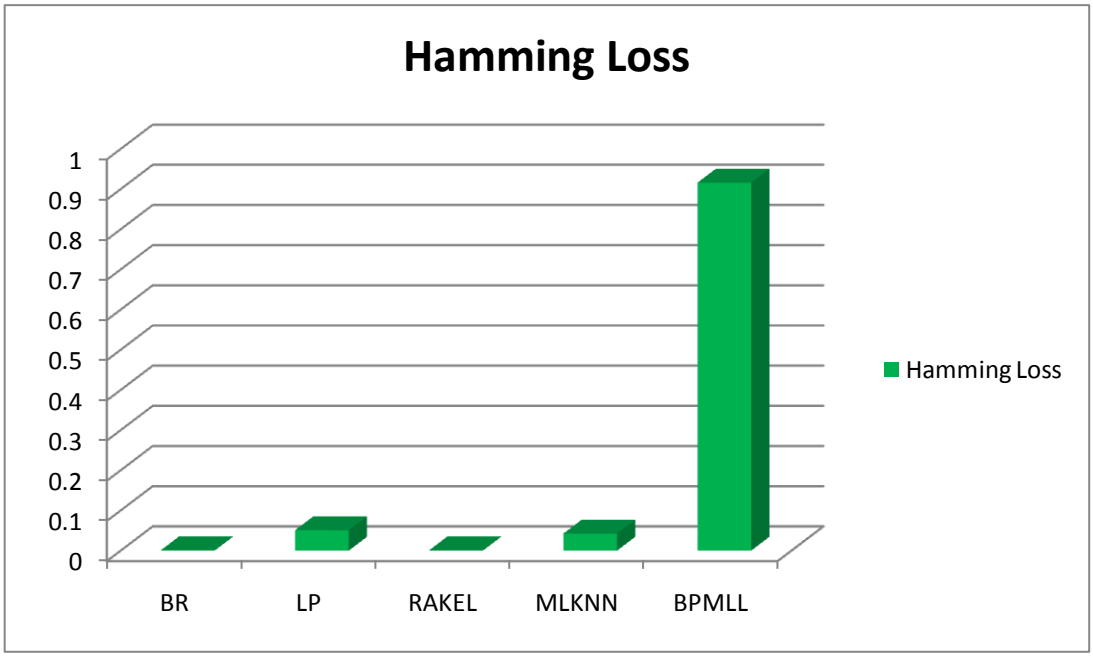
Γράφημα 7.2.2.1.4: Χρήση μνήμης των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4

Όπως παρατηρούμε από τις γραφικές παραστάσεις (7.2.2.1.1 - 7.2.2.1.4) οι μέθοδοι BR και RAKEL έχουν τα καλύτερα αποτελέσματα όσον αφορά τις μετρικές του Hamming Loss και Micro-averaged F-Measure, ενώ τα χείριστα αποτελέσματα τα έχει δημιουργήσει η μέθοδος BPMLL. Επίσης παρατηρούμε ότι οι γρηγορότερες μέθοδοι, όσον αφορά τον χρόνο CPU που σπατάλησαν για εκπαίδευση και αξιολόγηση του μοντέλου τους, είναι η LP και η MLKNN. Τέλος παρατηρούμε ότι η μέθοδος η οποία έχει δεσμεύσει τη λιγότερη μνήμη και με αρκετά μεγάλη διαφορά από τις υπόλοιπες 4 μεθόδους για τη διαδικασία της εκπαίδευσης του μοντέλου της είναι η BR.

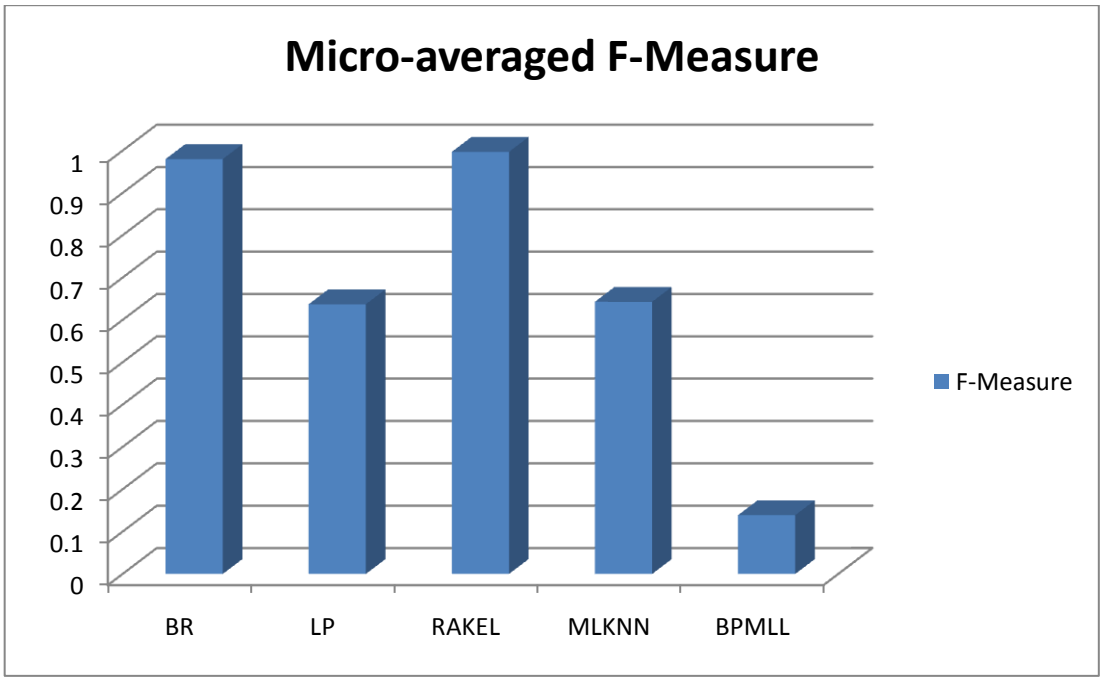
Πειράματα για 1000 χαρακτηριστικά 200 κατηγορίες

	BR	**LP**	**RAKEL**	**MLKNN**	**BPMLL**
Hamming Loss	0.0012	0.0509	**0.001**	0.0424	0.9161
Micro-averaged F-Measure	0.98	0.6371	**0.997**	0.6428	0.1388
CPU time	1072.24	**454.05**	5319.23	704.44	1803.03
Memory consumption	1755.88	2437.881	**1060.278**	1877.188	2009.794

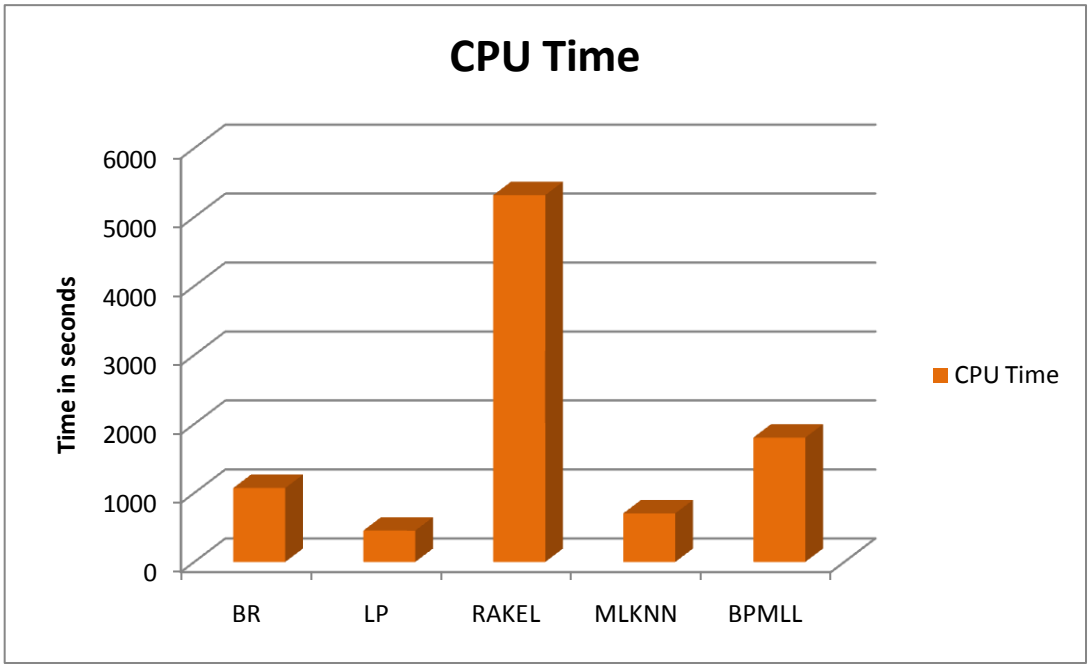
Πίνακας 7.2.2.1.2: Μετρήσεις που έχουν ληφθεί από πείραμα στο σύνολο στιγμιότυπων D_2



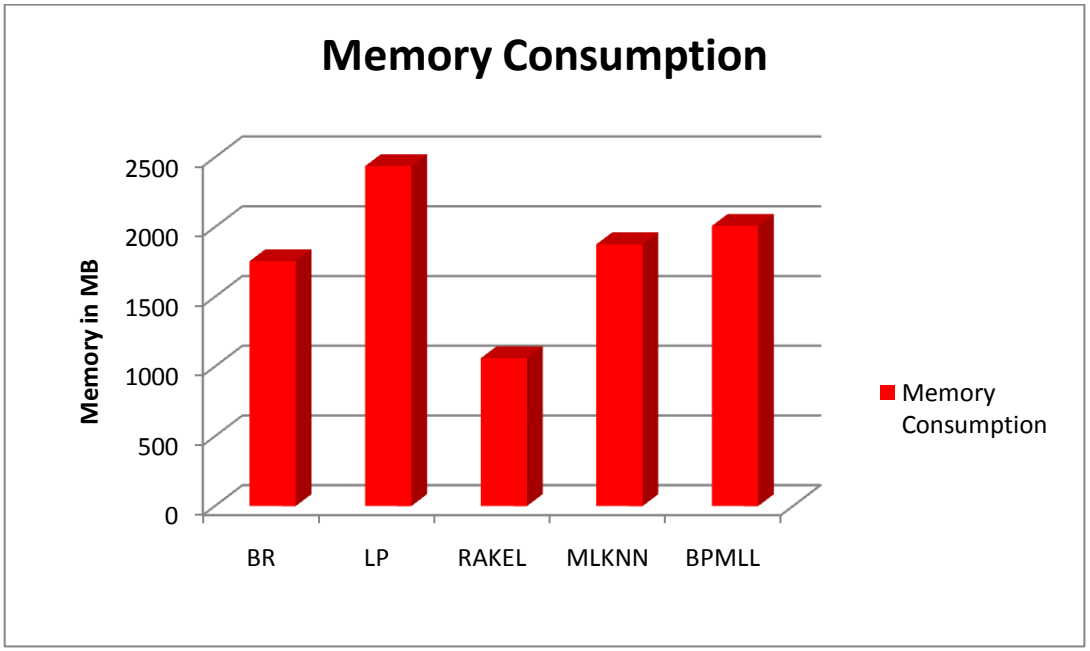
Γράφημα 7.2.2.1.5: Hamming Loss των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_2



Γράφημα 7.2.2.1.6: Micro-averaged F-Measure των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_2



Γράφημα 7.2.2.1.7: CPU time των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_2



Γράφημα 7.2.2.1.8: Χρήση μνήμης των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_2

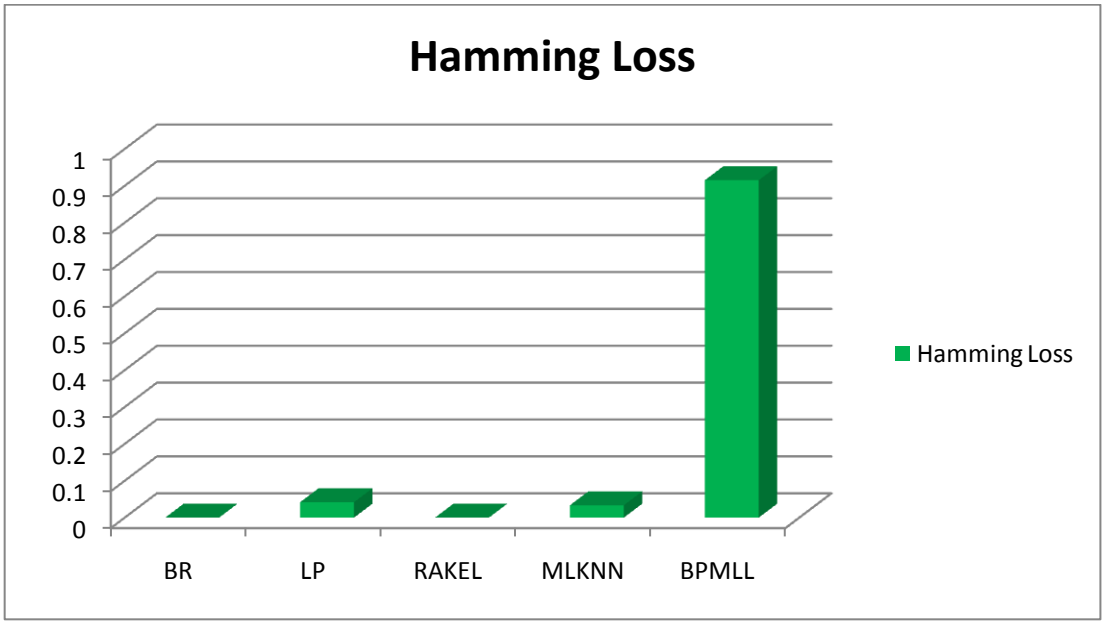
Όπως παρατηρούμε από τις γραφικές παραστάσεις (7.2.2.1.5 - 7.2.2.1.8) οι μέθοδοι BR και RAKEL έχουν τα καλύτερα αποτελέσματα όσον αφορά τις μετρικές του Hamming

Loss και Micro-averaged F-Measure, ενώ τα χείριστα αποτελέσματα τα έχει δημιουργήσει η μέθοδος BPMLL. Επίσης παρατηρούμε ότι οι γρηγορότερες μέθοδοι, όσον αφορά τον χρόνο CPU που σπατάλησαν για εκπαίδευση και αξιολόγηση του μοντέλου τους, είναι η LP και η MLKNN. Τέλος παρατηρούμε ότι η μέθοδος η οποία έχει δεσμεύσει τη λιγότερη μνήμη και με αρκετά μεγάλη διαφορά από τις υπόλοιπες 4 μεθόδους για την διαδικασία της εκπαίδευσης του μοντέλου της είναι η RAKEL.

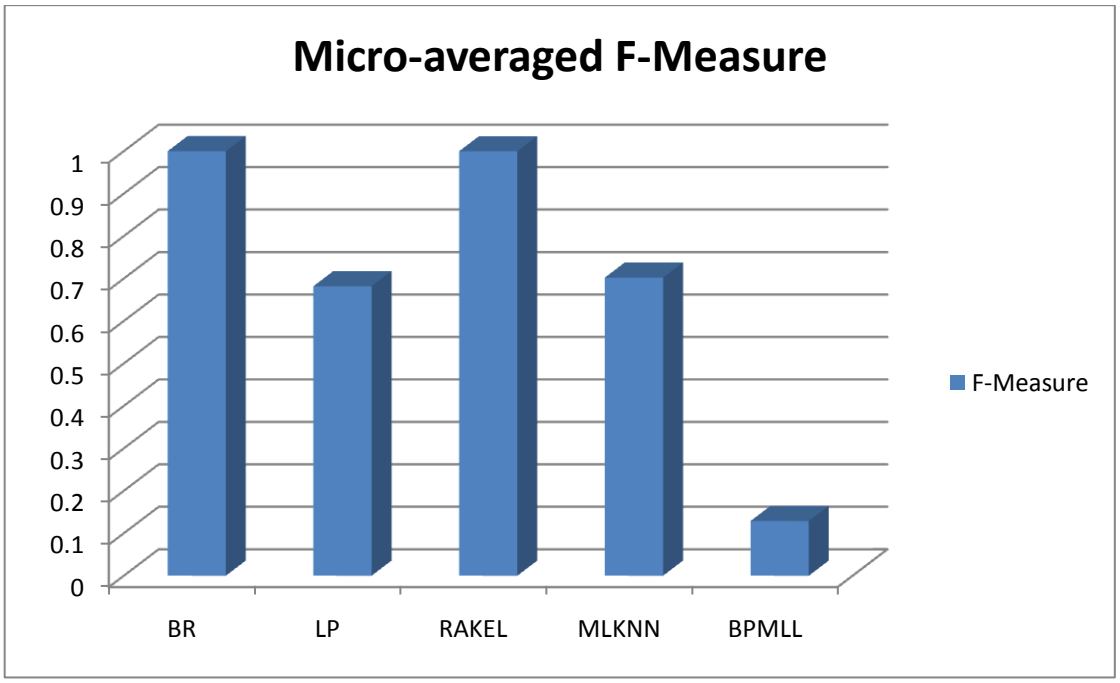
Πειράματα για 2000 χαρακτηριστικά 200 κατηγορίες

	BR	**LP**	**RAKEL**	**MLKNN**	**BPMLL**
Hamming Loss	**0.0**	0.0417	**0.0**	0.0335	0.9151
Micro-averaged F-Measure	**0.9998**	0.6817	0.9997	0.7021	0.1288
CPU time	1973.16	**682.86**	10779.04	1517.41	4394.72
Memory consumption	1149.673	**1104.567**	3756.309	2241.752	4442.644

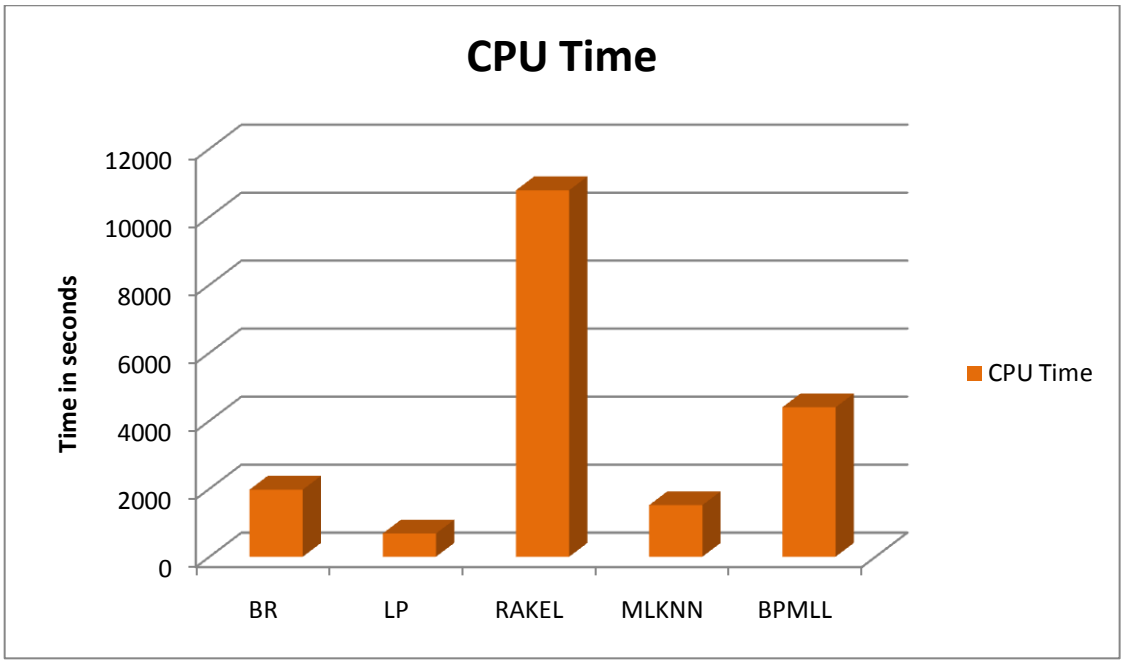
Πίνακας 7.2.2.1.3: Μετρήσεις που έχουν ληφθεί από πείραμα στο σύνολο στιγμιότυπων D_5



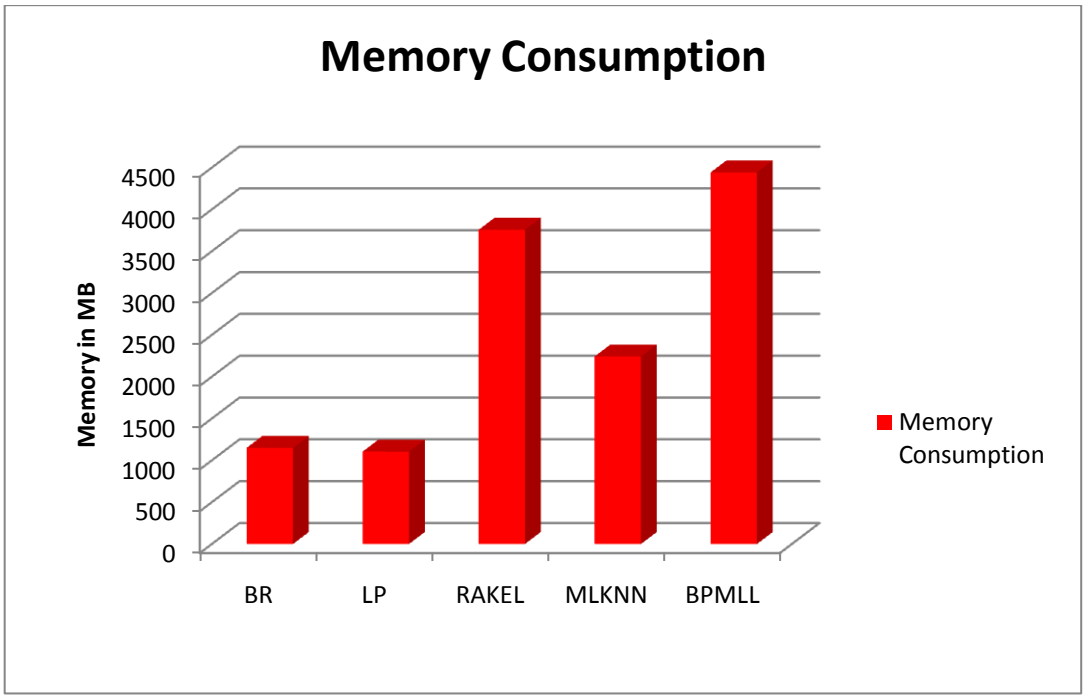
Γράφημα 7.2.2.1.9: Hamming Loss των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_5



Γράφημα 7.2.2.1.10: Micro-averaged F-Measure των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_5



Γράφημα 7.2.2.1.11: CPU time των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_5



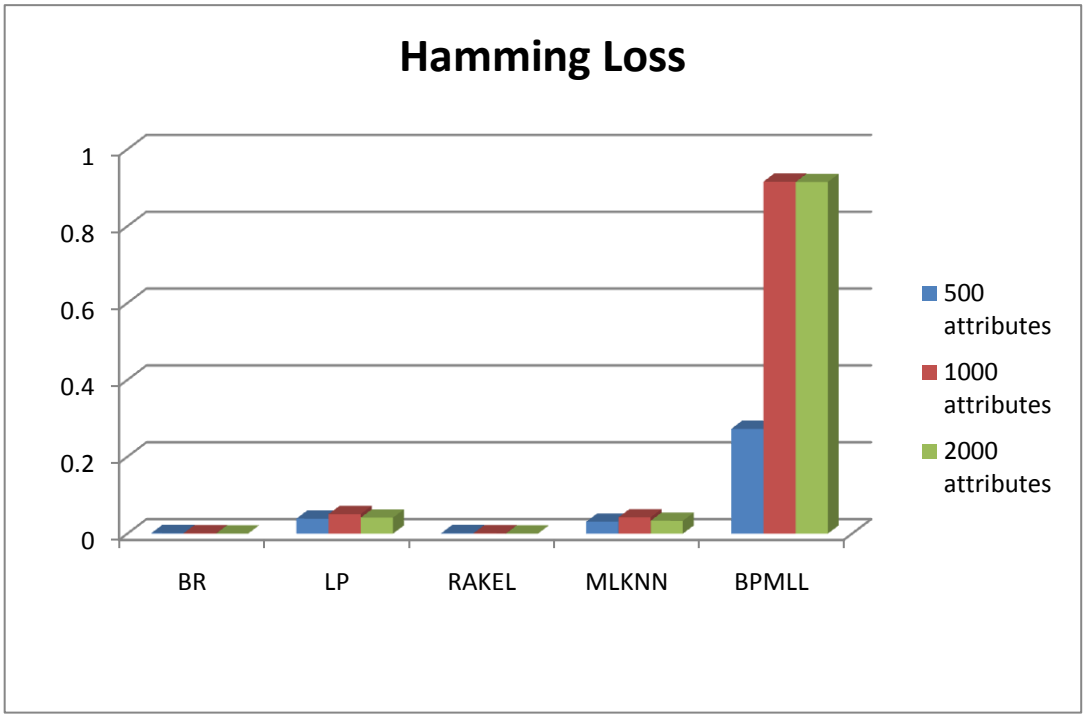
Γράφημα 7.2.2.1.12: Χρήση μνήμης των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_5

Όπως παρατηρούμε από τις γραφικές παραστάσεις (7.2.2.1.9 - 7.2.2.1.12), ομοίως με πριν οι μέθοδοι BR και RAKEL εξακολουθούν να έχουν τα καλύτερα αποτελέσματα όσον αφορά τις μετρικές του Hamming Loss και Micro-averaged F-Measure, ενώ τα χείριστα αποτελέσματα εξακολουθεί να τα έχει δημιουργήσει η μέθοδος BPMLL. Επίσης παρατηρούμε ότι οι γρηγορότερες μέθοδοι, όσον αφορά τον χρόνο CPU που σπατάλησαν για εκπαίδευση και ταξινόμηση του μοντέλου τους, είναι ξανά η LP και η MLKNN. Τέλος παρατηρούμε ότι οι μέθοδοι οι οποίες έχουν δεσμεύσει τις μικρότερες ποσότητες μνήμης είναι σε αυτό το πείραμα η BR και η LP.

Συγκρίσεις για όλες τις γραφικές παραστάσεις με μεταβαλλόμενο αριθμό χαρακτηριστικών και σταθερό αριθμό κλάσεων

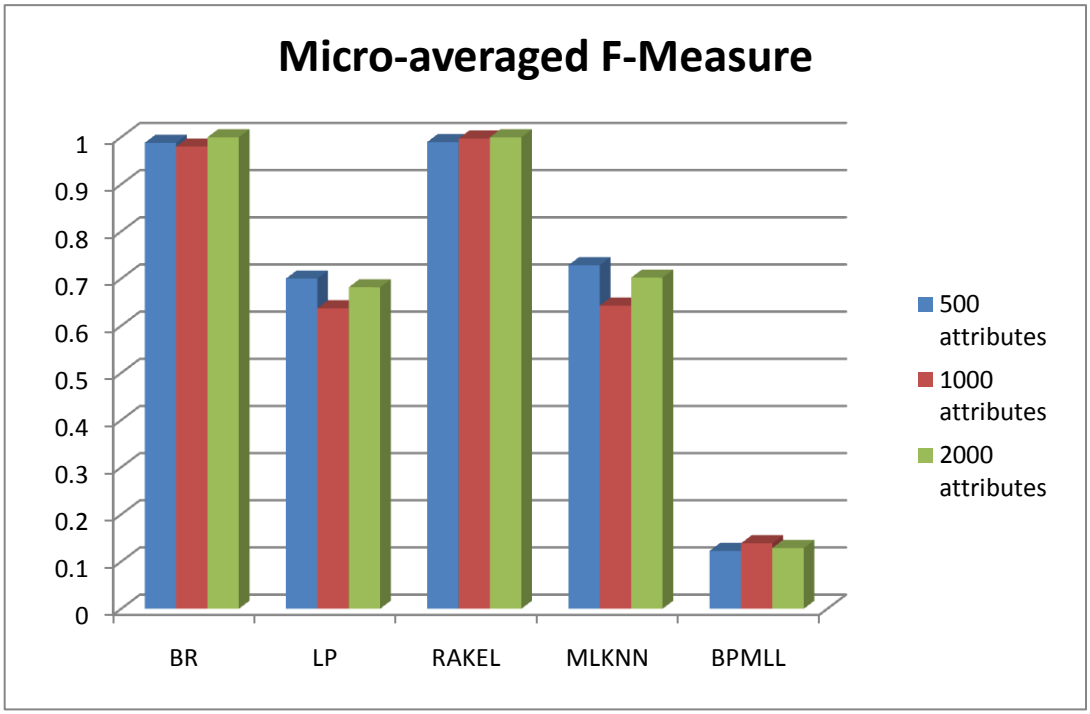
Multi-label classification methods	Data Sets	Hamming Loss	Micro-averaged F-Measure	CPU time	Memory consumption
BR	**D_2**	0.0012	0.98	1072.24	1755.88
	D_4	0.0016	0.9879	845.78	**1108.69**
	D_5	**0.0**	**0.9998**	1973.16	1149.673
LP	**D_2**	0.0509	0.6371	454.05	2437.881
	D_4	0.0391	0.7001	**217.59**	2311.977
	D_5	0.0417	0.6817	682.86	1104.567
RAKEL	**D_2**	0.001	0.997	5319.23	1060.278
	D_4	0.0014	0.9896	5152.2	1924.661
	D_5	**0.0**	0.9997	10779.04	3756.309
MLKNN	**D_2**	0.0424	0.6428	704.44	1877.188
	D_4	0.0311	0.7285	351.39	1902.244
	D_5	0.0335	0.7021	1517.41	2241.752
BPMLL	**D_2**	0.9161	0.1388	1803.03	2009.794
	D_4	0.2726	0.1228	846.75	1308.627
	D_5	0.9151	0.1288	4394.72	4442.644

Πίνακας 7.2.2.1.4: Συγκεντρωτικές μετρήσεις που έχουν ληφθεί από πείραμα στα σύνολο στιγμιότυπων D_2, D_4 και D_5



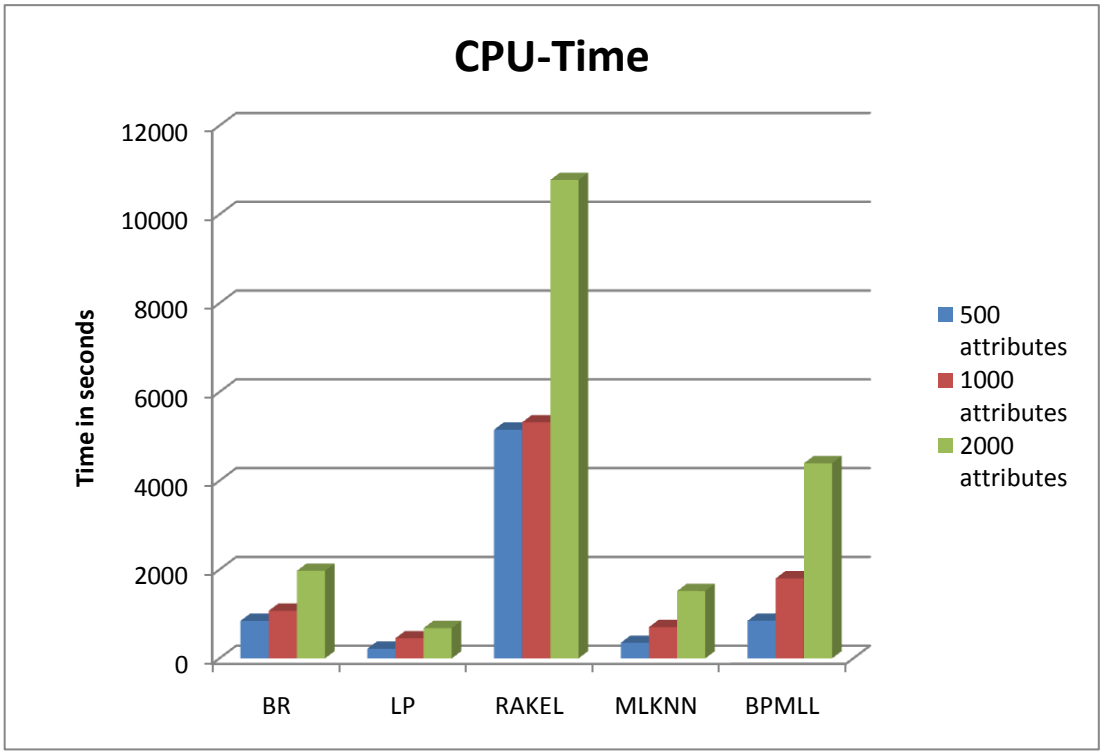
Γράφημα 7.2.2.1.13: Συγκριτικά αποτελέσματα για την μετρική Hamming Loss των 5 μεθόδων στα σύνολα στιγμιότυπων D_2, D_4 και D_5

Από τη γραφική παράσταση (7.2.2.1.13) παρατηρούμε ότι οι μέθοδοι BR, LP, RAKEL, MLKNN δεν επηρεάζονται από τη μεταβολή του αριθμού των χαρακτηριστικών και κρατούν σταθερά τα αποτελέσματα τους σε αντίθεση με την BPMLL. Επιπλέον, φαίνεται ότι τις καλύτερες μετρήσεις έχει συγκεντρώσει η μέθοδος RAKEL με ελάχιστη διαφορά από την BR.



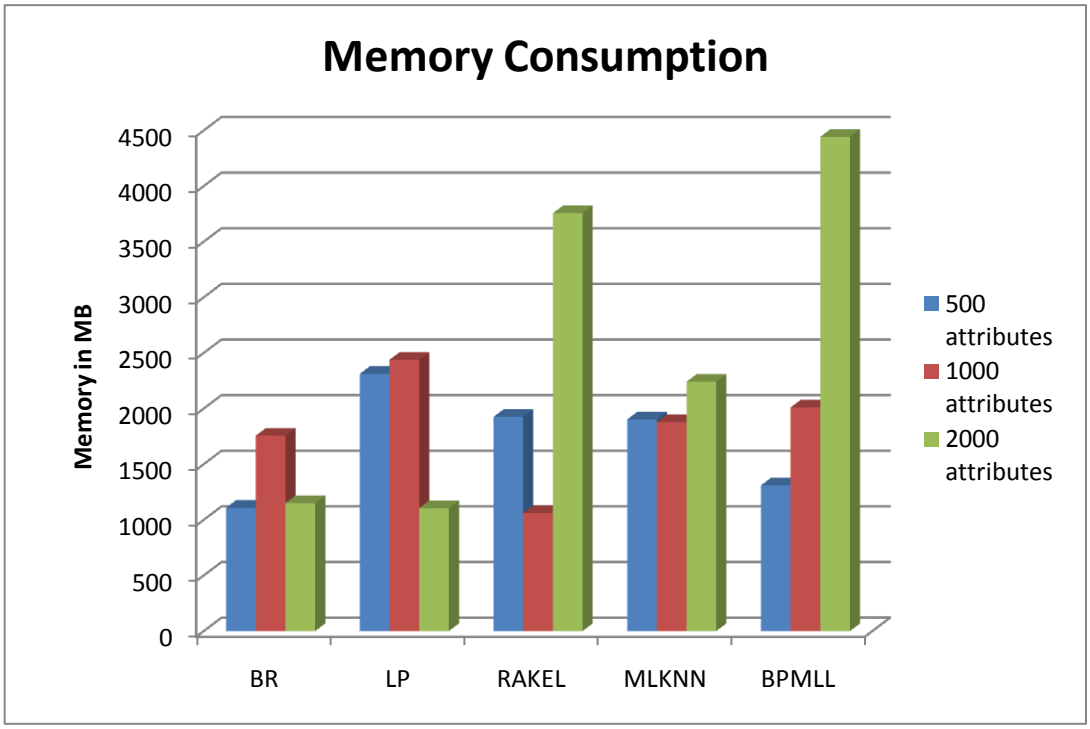
Γράφημα 7.2.2.1.14: Συγκριτικά αποτελέσματα την μετρική Micro-averaged F-Measure των 5 μεθόδων στα σύνολα στιγμιότυπων D_2, D_4 και D_5

Από την γραφική παράσταση (7.2.2.1.14) παρατηρούμε ότι οι μέθοδοι BR, LP, MLKNN, και BPMLL δεν επηρεάζονται από τη μεταβολή του αριθμού χαρακτηριστικών σε αντίθεση με την RAKEL της οποίας τα αποτελέσματα φαίνεται να αυξάνονται ανάλογα με τον αριθμό των χαρακτηριστικών. Οι μέθοδοι BR, LP και MLKNN έχουν αρκετά καλά αποτελέσματα όμως στην περίπτωση όπου δοκιμάστηκαν για 1000 χαρακτηριστικά η μετρική Micro-averaged F-Measure μειώθηκε και για τις δύο αυτές μεθόδους σε σημαντικό βαθμό. Η RAKEL όμως έχει σημειώσει τα καλύτερα αποτελέσματα και το ότι τα αποτελέσματα φαίνεται να αυξάνονται ανάλογα με τον αριθμό των χαρακτηριστικών ίσως προκύπτει από το γεγονός ότι η RAKEL είναι μια *ensemble* [7] μέθοδος, δηλαδή αποτελείται από πολλούς LP ταξινομητές, όπου ο κάθε ένας αναλαμβάνει να ταξινομήσει ένα τυχαίο σύνολο *k* κατηγορίες.



Γράφημα 7.2.2.1.15: Συγκριτικά αποτελέσματα για το CPU time των 5 μεθόδων στα σύνολα στιγμιότυπων D_2, D_4 και D_5

Όσον αφορά τη μετρική για το χρόνο CPU που χρειάστηκε για την εκπαίδευση και αξιολόγηση του μοντέλου της κάθε μεθόδου, είναι φανερό ότι η λιγότερη χρονοβόρα μέθοδος είναι η LP. Επίσης καλούς χρόνους για όλα τα μεγέθη των χαρακτηριστικών σημείωσαν οι μέθοδοι BR και MLKNN ενώ τους χειρότερους χρόνους η μέθοδος RAKEL. Ακόμη, παρατηρώντας τη γραφική παράσταση (7.2.2.1.15) φαίνεται ότι η αύξηση του χρόνου εκπαίδευσης και αξιολόγησης του μοντέλου κάθε ταξινομητή, οφείλεται στο γεγονός ότι με την αύξηση του αριθμού των χαρακτηριστικών το πρόβλημα γίνεται πιο περίπλοκο γι' αυτό απαιτείται περισσότερος χρόνος από όλες τις μεθόδους ταξινόμησης. Η μέθοδος RAKEL έχει σημειώσει τους χειρότερους χρόνους και αυτό γιατί όπως έχει εξηγηθεί και προηγουμένως πρόκειται για μια *ensemble* μέθοδο.



Γράφημα 7.2.2.1.16: Συγκριτικά αποτελέσματα για το Memory consumption των 5 μεθόδων στα σύνολα στιγμιότυπων D_2, D_4 και D_5

Τέλος, από τη γραφική παράσταση (7.2.2.1.16) προκύπτει ότι για όλα τα μεγέθη των χαρακτηριστικών οι μέθοδοι που σημείωσαν την λιγότερη χρήση μνήμης είναι οι BR, LP και MLKNN.

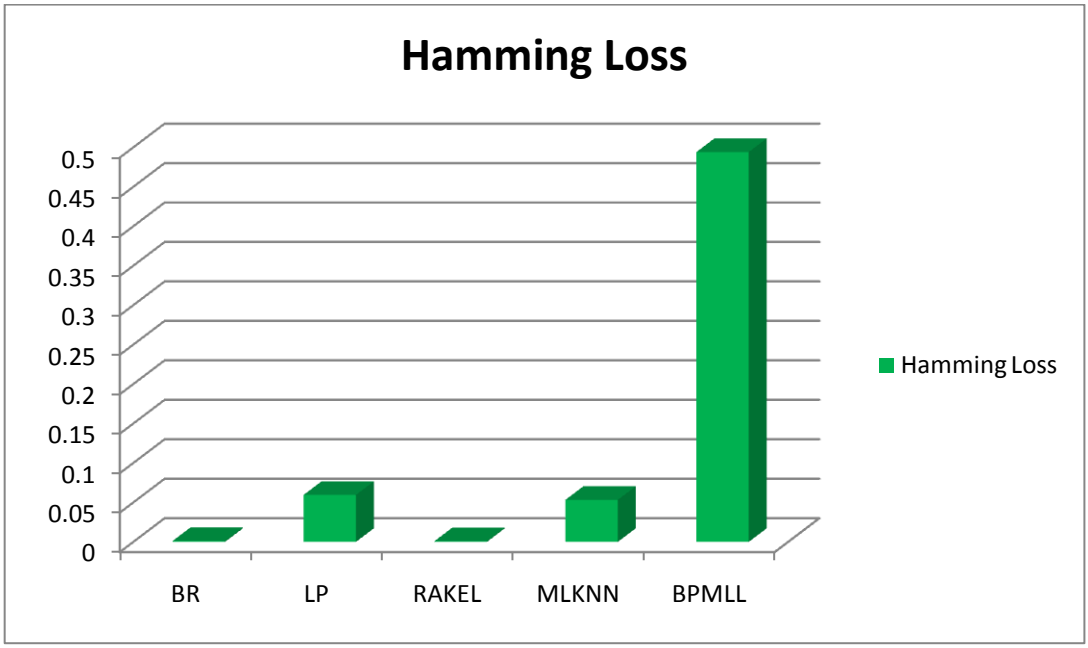
7.2.2.2 Πειράματα με μεταβολή στον αριθμό των κατηγοριών

Πειράματα για 1000 χαρακτηριστικά και 100 κατηγορίες.

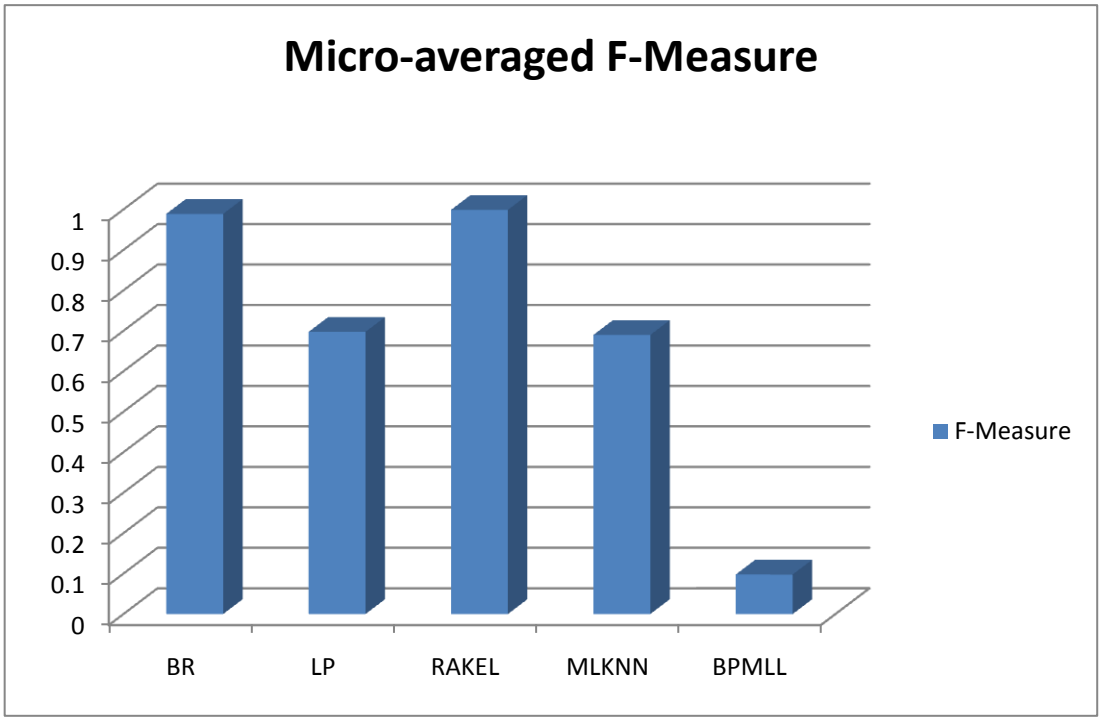
	BR	**LP**	**RAKEL**	**MLKNN**	**BPMLL**
Hamming Loss	0.0012	0.0595	**0.001**	0.0533	0.4941
Micro-averaged F-Measure	0.9889	0.6973	**0.9988**	0.6897	0.0976
CPU time	370.44	**262.57**	3851.03	581.51	670.19
Memory	1276.425	**606.2305**	1457.727	916.3946	1079.777

consumption					

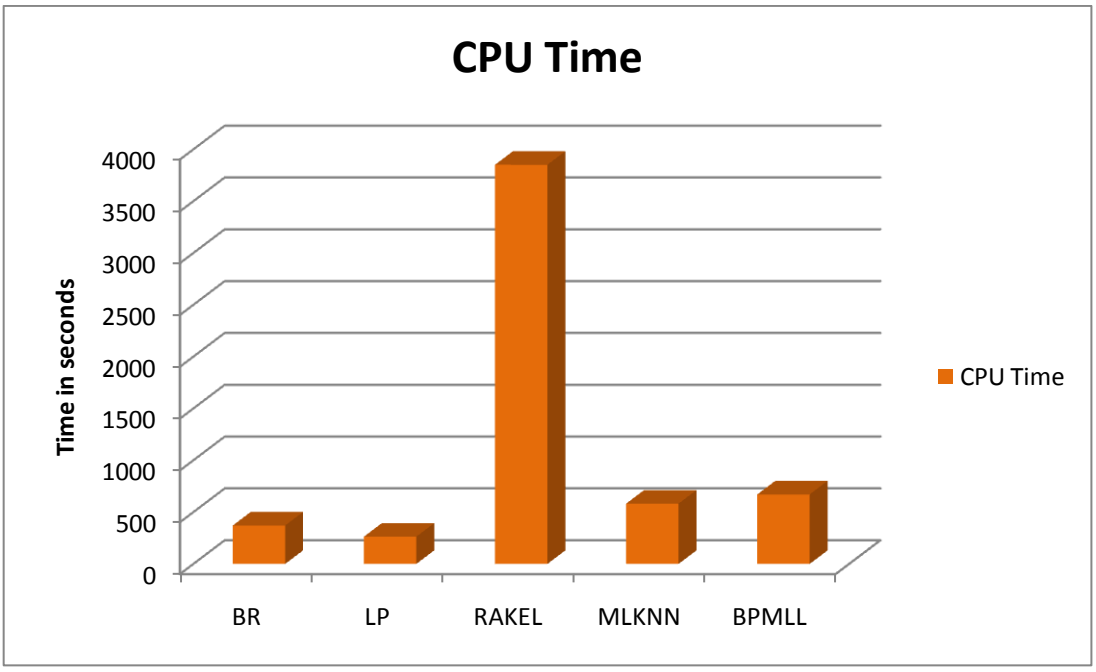
Πίνακας 7.2.2.2.1: Μετρήσεις που έχουν ληφθεί από πείραμα στο σύνολο στιγμιότυπων D1



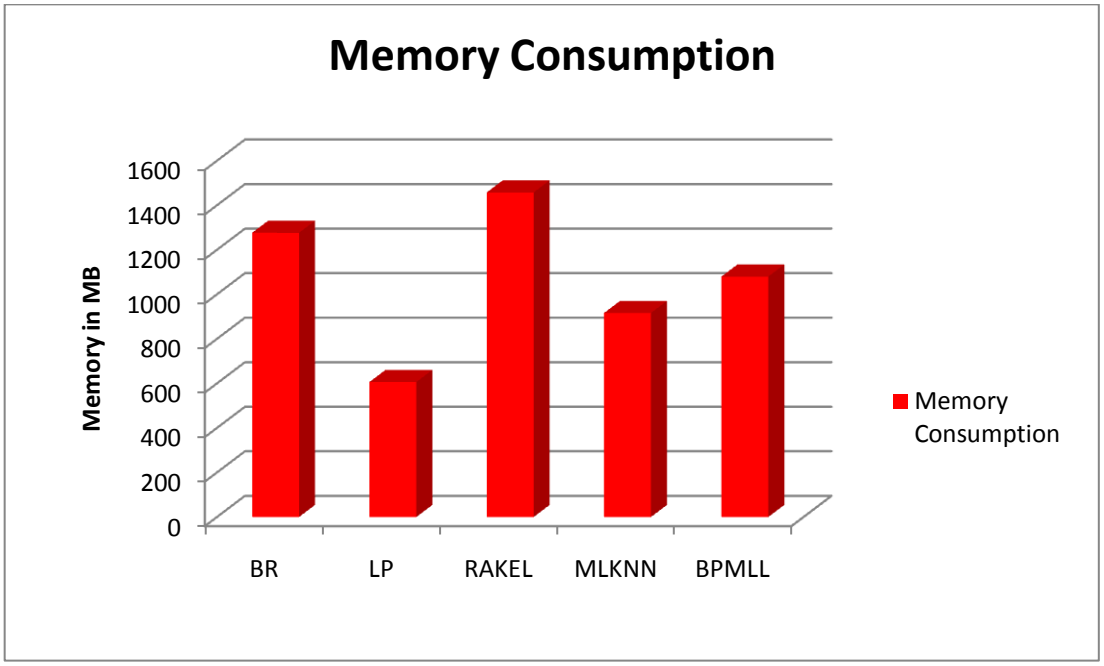
Γράφημα 7.2.2.2.1: Hamming Loss των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_1



Γράφημα 7.2.2.2.2: Micro-averaged F-Measure των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_1



Γράφημα 7.2.2.2.3: CPU time των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_1



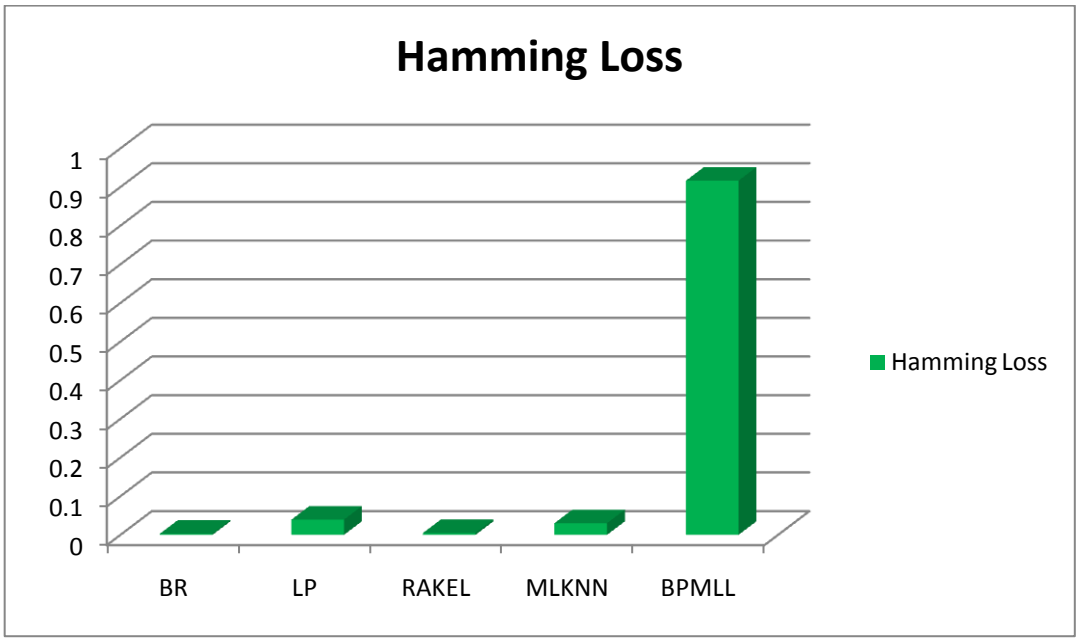
Γράφημα 7.2.2.2.4: Χρήση μνήμης των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_1

Από τις γραφικές παραστάσεις (7.2.2.2.1 - 7.2.2.2.4) προκύπτει ότι οι μέθοδοι BR και RAKEL εξακολουθούν να έχουν τα καλύτερα αποτελέσματα όσον αφορά τις μετρικές του Hamming Loss και Micro-averaged F-Measure, ενώ τα χείριστα αποτελέσματα εξακολουθεί να τα έχει δημιουργήσει η μέθοδος BPMLL. Επιπλέον, παρατηρούμε ότι οι πιο αποδοτικές μέθοδοι, από άποψης χρόνου CPU, που σπαταλήθηκε για εκπαίδευση του μοντέλου τους, είναι η BR και η LP. Τέλος παρατηρούμε ότι οι μέθοδοι οι οποίες έχουν δεσμεύσει τις μικρότερες ποσότητες μνήμης για αυτή την διαδικασία είναι η LP και η MLKNN.

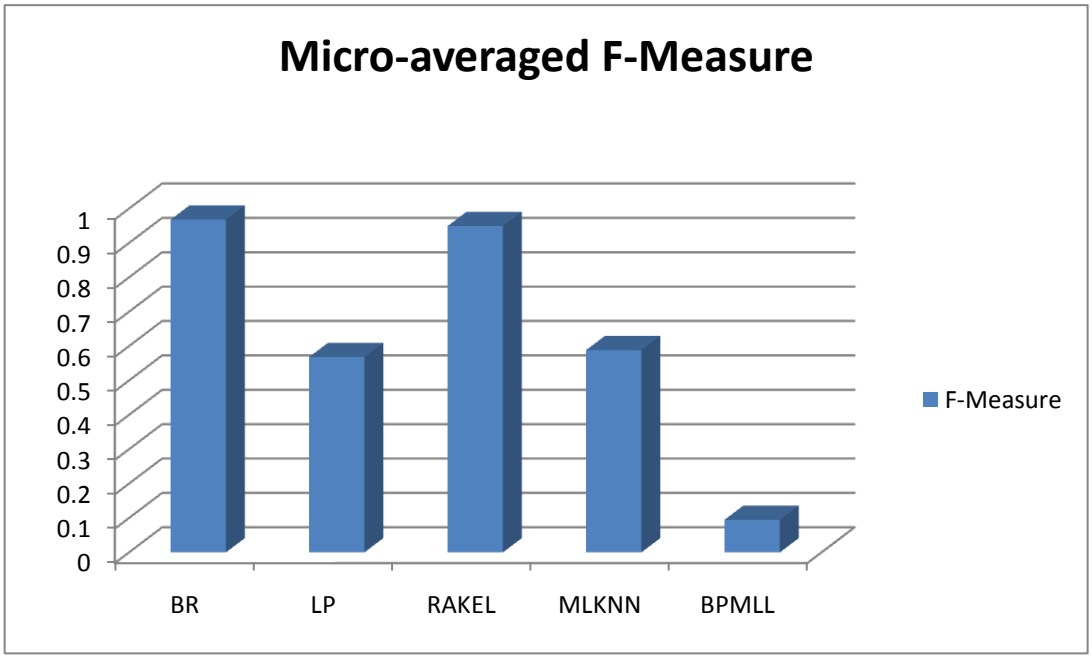
Πειράματα για 1000 χαρακτηριστικά και 400 κατηγορίες.

	BR	LP	RAKEL	MLKNN	BPMLL
Hamming Loss	**0.003**	0.0394	0.0047	0.0302	0.9165
Micro-averaged F-Measure	**0.9678**	0.568	0.949	0.5882	0.0947
CPU time	4287.02	**370.67**	10976.98	632.24	2980.96
Memory consumption	3361.699	3607.235	2135.372	3443.954	**1993.023**

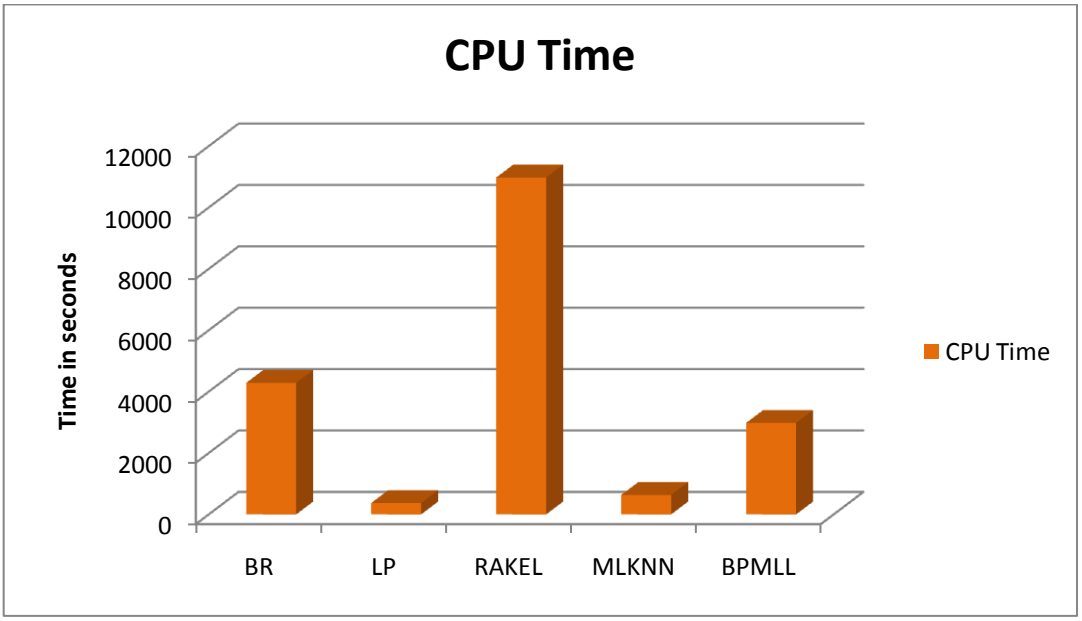
Πίνακας 7.2.2.2.2: Μετρήσεις που έχουν ληφθεί από πείραμα στο σύνολο στιγμιότυπων D3



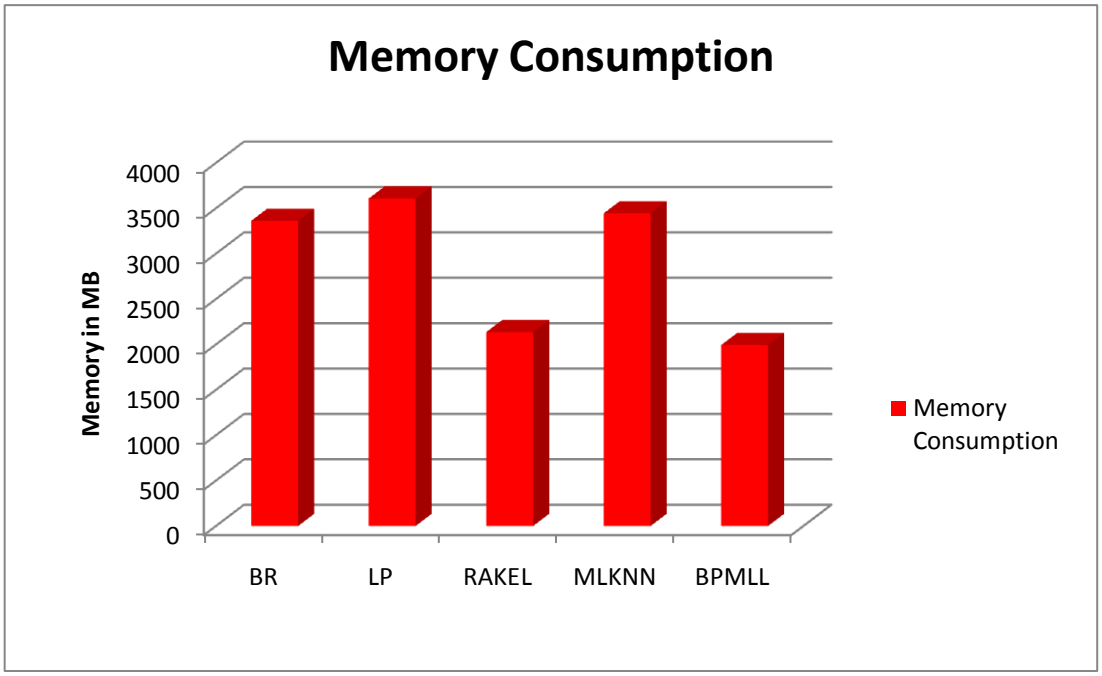
Γράφημα 7.2.2.2.5: Hamming Loss των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_3



Γράφημα 7.2.2.2.6: Micro-averaged F-Measure των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_3



Γράφημα 7.2.2.2.7: CPU time των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_3



Γράφημα 7.2.2.2.8: Χρήση μνήμης των 5 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_3

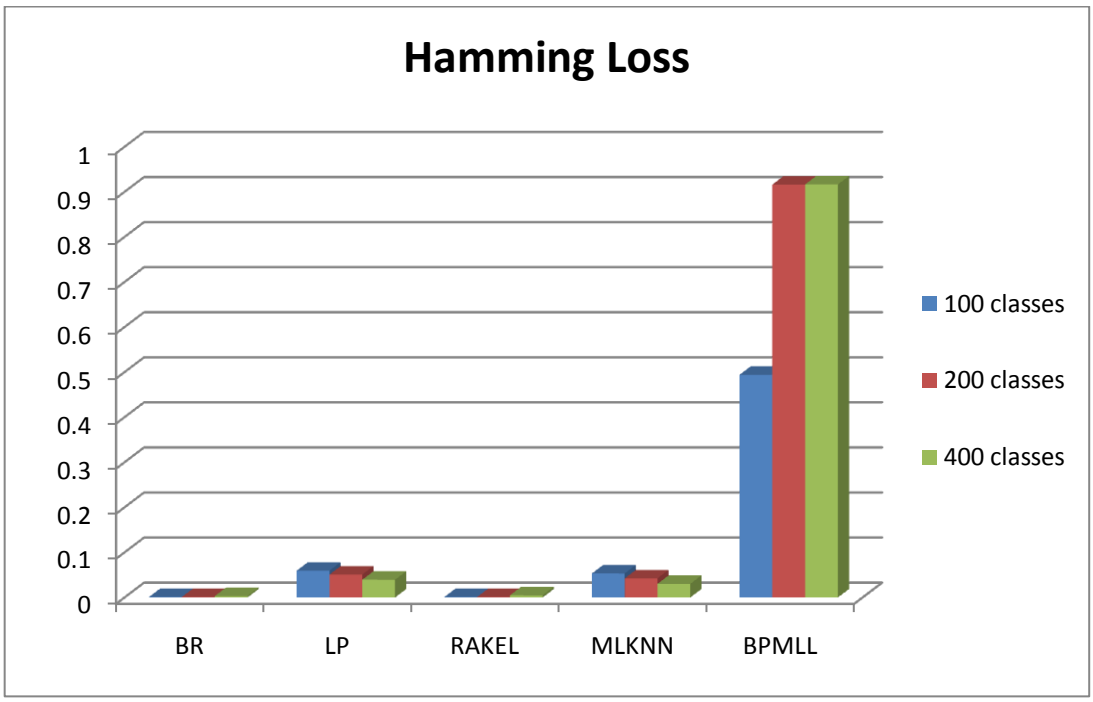
Από τις γραφικές παραστάσεις (7.2.2.2.4 - 7.2.2.2.8) προκύπτει ότι οι μέθοδοι BR και RAKEL εξακολουθούν να έχουν τα καλύτερα αποτελέσματα όσον αφορά τις μετρικές του Hamming Loss και Micro-averaged F-Measure, ενώ τα χείριστα αποτελέσματα εξακολουθεί να τα έχει δημιουργήσει η μέθοδος BPMLL. Επιπλέον, παρατηρούμε ότι οι πιο αποδοτικές μέθοδοι, από άποψης χρόνου CPU, που σπαταλήθηκε για εκπαίδευση του μοντέλου τους, είναι η LP και η MLKNN. Τέλος παρατηρούμε ότι οι μέθοδοι οι οποίες έχουν δεσμεύσει τις μικρότερες ποσότητες μνήμης για αυτή την διαδικασία είναι η RAKEL και η BPMLL.

Συγκρίσεις για όλες τις γραφικές παραστάσεις με μεταβαλλόμενο αριθμό χαρακτηριστικών και σταθερό αριθμό κλάσεων

Multi-label classification methods	Data Sets	Hamming Loss	Micro-averaged F-Measure	CPU time	Memory consumption
BR	D_1	0.0012	0.9889	370.44	1276.425
	D_2	0.0012	0.98	1072.24	1755.88

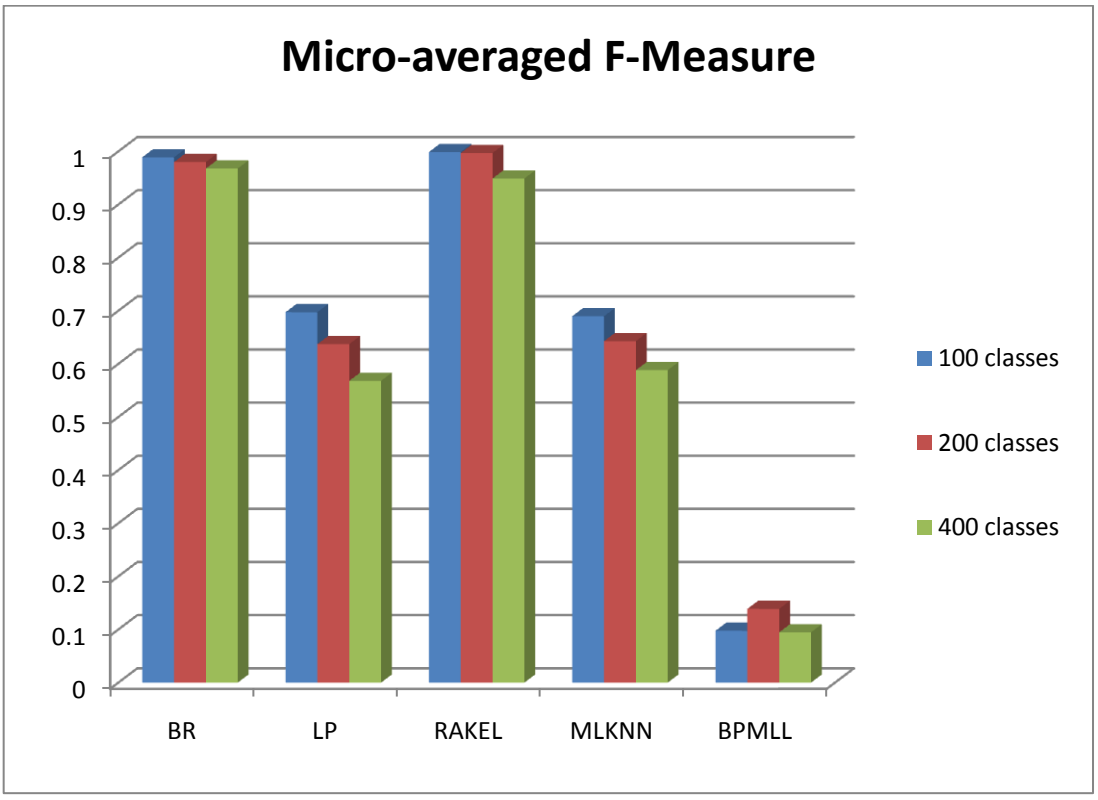
	D_3	0.003	0.9678	4287.02	3361.699
LP	**D_1**	0.0595	0.6973	**262.57**	**606.2305**
	D_2	0.0509	0.6371	454.05	2437.881
	D_3	0.0394	0.568	370.67	3607.235
RAKEL	**D_1**	**0.001**	**0.9988**	3851.03	1457.727
	D_2	**0.001**	0.997	5319.23	1060.278
	D_3	0.0047	0.949	10976.98	2135.372
MLKNN	**D_1**	0.0533	0.6897	581.51	916.3946
	D_2	0.0424	0.6428	704.44	1877.188
	D_3	0.0302	0.5882	632.24	3443.954
BPMLL	**D_1**	0.4941	0.0976	670.19	1079.777
	D_2	0.9161	0.1388	1803.03	2009.794
	D_3	0.9165	0.0947	2980.96	1993.023

Πίνακας 7.2.2.1.4: Συγκεντρωτικές μετρήσεις που έχουν ληφθεί από πείραμα στα σύνολο στιγμιότυπων D_1, D_2 και D_3



Γράφημα 7.2.2.2.9: Συγκριτικά αποτελέσματα για τη μετρική Hamming Loss των 5 μεθόδων στα σύνολα στιγμιότυπων D_1, D_2 και D_3

Παρατηρώντας τη γραφική παράσταση (7.2.2.2.9) και τις αντίστοιχες τιμές στον πίνακα (7.2.2.1.4) μπορούμε να διακρίνουμε ότι όταν ο αριθμός των κατηγοριών είναι μεγαλύτερος, οι μέθοδοι BR, RAKEL και BPMLL παρουσιάζουν αύξηση του Hamming Loss. Αυτό σημαίνει ότι κάνουν περισσότερες εσφαλμένες προβλέψεις. Αντιθέτως το Hamming Loss μειώνεται στην περίπτωση των μεθόδων LP και MLKNN. Οι πιο αποδοτικές μέθοδοι σε όλο το εύρος του αριθμού των κλάσεων παραμένουν οι BR και RAKEL.



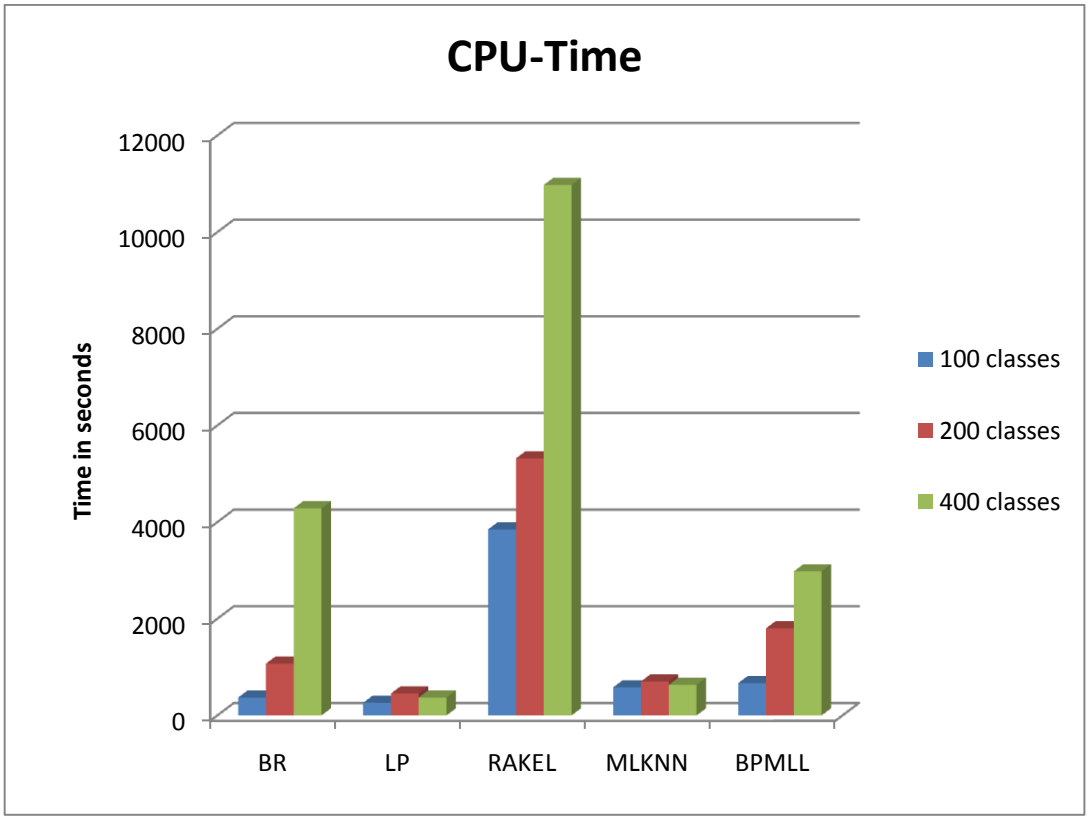
Γράφημα 7.2.2.2.10: Συγκριτικά αποτελέσματα την μετρική Micro-averaged F-Measure των 5 μεθόδων στα σύνολα στιγμιότυπων D_1, D_2 και D_3

Σύμφωνα με τη γραφική παράσταση (7.2.2.2.10) καθώς ο αριθμός των κατηγοριών αυξάνεται η μετρική Micro-averaged F-Measure μειώνεται για τις μεθόδους BR, LP, RAKEL και MLKNN ενώ η μέθοδος BPMLL εξακολουθεί να έχει τις χείριστες μετρήσεις όλων των μεθόδων.

Η γενική αυτή μείωση που παρατηρείται, οφείλεται στο γεγονός ότι με την αύξηση του αριθμού των κατηγοριών το πρόβλημα γίνεται πιο πολύπλοκο έτσι είναι φυσικό οι

μέθοδοι να οδηγούνται σε λανθασμένες προβλέψεις και κατά συνέπεια η μετρική αυτή να είναι αντιστρόφως ανάλογης της αύξησης του αριθμού των κατηγοριών.

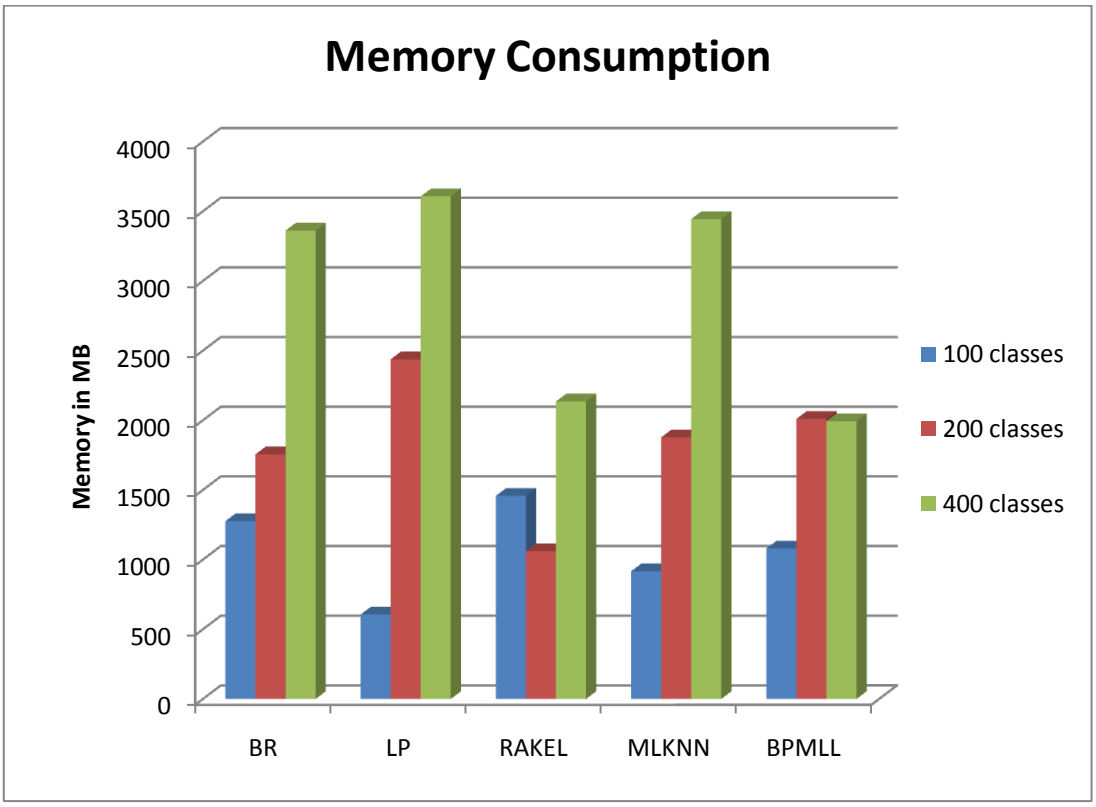
Παρόλο που η μετρική Micro-averaged F-Measure μειώνεται με την αύξηση του αριθμού των κατηγοριών, οι δύο μέθοδοι με τα ως τώρα καλυτέρα αποτελέσματα εξακολουθούν να είναι η BR και η RAKEL με τη δεύτερη να έχει ελαφρώς καλύτερες μετρήσεις.



Γράφημα 7.2.2.2.11: Συγκριτικά αποτελέσματα για το CPU time των 5 μεθόδων στα σύνολα στιγμιότυπων D_1, D_2 και D_3

Όσον αφορά το χρόνο CPU που χρειάστηκαν οι μέθοδοι για να εκπαιδεύσουν και να αξιολογήσουν τα μοντέλα τους, σύμφωνα με την γραφική παράσταση (7.2.2.2.11) οι μέθοδοι LP, MLKNN φαίνεται να μην επηρεάστηκαν από την αύξηση του αριθμού των κατηγοριών σε αντίθεση με τις υπόλοιπες μεθόδους. Οι μέθοδοι BR, RAKEL και BPMLL επηρεάζονται με την αύξηση του αριθμού των κατηγοριών όμως η μέθοδος RAKEL έχει σημειώσει τις χείριστες μετρήσεις αυτό προκύπτει από το γεγονός ότι η

RAKEL είναι μια *ensemble* [7] μέθοδος η οποία κάνει περισσότερους υπολογισμούς με αποτέλεσμα αυτό να επιβαρύνει τις μετρήσεις του CPU χρόνου.



Γράφημα 7.2.2.2.12: Συγκριτικά αποτελέσματα για το Memory consumption των 5 μεθόδων στα σύνολα στιγμιότυπων D_1, D_2 και D_3

Τέλος, παρατηρούμε από τη γραφική παράσταση (7.2.2.2.12), ότι όλες οι μέθοδοι έχουν επηρεαστεί όσον αφορά την δέσμευση μνήμης για την εκπαίδευση και αξιολόγηση του μοντέλου τους. Αυτό φαίνεται ξεκάθαρα από τις μεθόδους BR, LP και MLKNN όπου τα αποτελέσματα δείχνουν ότι η αύξηση της δέσμευσης μνήμης είναι ανάλογη με την αύξηση των κατηγοριών στα πειράματά μας. Επίσης αυτό που παρατηρούμε είναι ότι η RAKEL δεν συμφωνεί με τις μετρήσεις των υπόλοιπων μεθόδων που αυτό οφείλεται στο διαφορετικό τρόπο λειτουργίας της.

7.2.2.3 Χρήση Naïve Bayes ως αλγόριθμος εκπαίδευσης (base learner)

Στα προηγούμενα πειράματα χρησιμοποιήσαμε 5 διαφορετικές μεθόδους ταξινόμησης στιγμιότυπων εκ των οποίων η BR και η LP ανήκουν στην κατηγορία των Problem

Transformation μεθόδων, η RAKEL είναι μια *ensemble* μέθοδος και οι MLKNN, BPMLL ανήκουν στην κατηγορία των Algorithm Adaptation μεθόδων.

Από αυτές τις 3 μεθόδους ξεχώρισαν οι BR και RAKEL για τα ιδιαίτερα καλά αποτελέσματα τους όσον αφορά τις πιο σημαντικές μετρικές που είναι το Hamming Loss και το Micro-averaged F-measure. Αυτές οι δύο μέθοδοι multi-label ταξινόμησης, για να λειτουργήσουν αναγάγουν ένα πρόβλημα ταξινόμησης μονής ετικέτας (single-label) σε πρόβλημα ταξινόμησης πολλαπλών ετικετών χρησιμοποιώντας single-label αλγόριθμους ταξινόμησης του Weka [20].

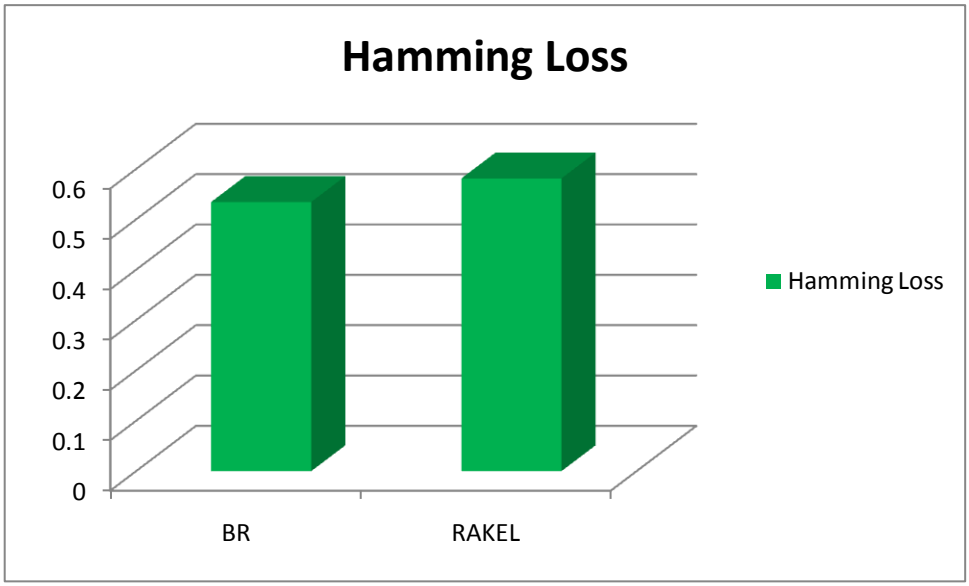
Μέχρι τώρα, τα πειράματά μας είχαν εφαρμοστεί ώστε να χρησιμοποιούν ως single-label ταξινομητές τα δέντρα αποφάσεως J48.

Ακολούθως, παρουσιάζουμε μερικά πειράματα στα οποία χρησιμοποιήσαμε ένα διαφορετικό είδος single-label ταξινομητή, τον Naive Bayes για να παρατηρήσουμε τι αλλαγές θα συμβούν στην απόδοση των 2 μεθόδων όσον αφορά τις μετρικές Hamming Loss, Micro Averaged F-measure, τον χρόνο CPU και τη δέσμευση μνήμης.

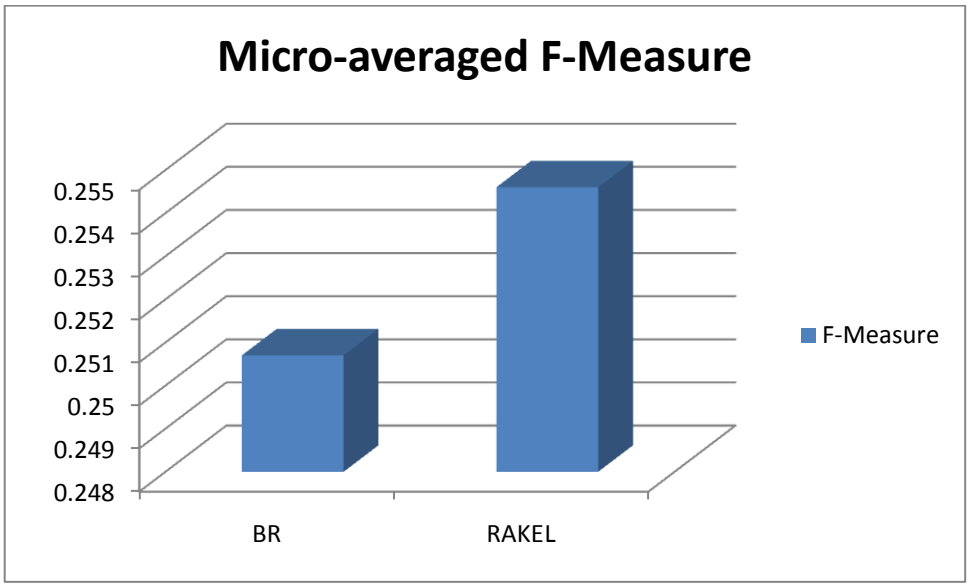
<u>500 χαρακτηριστικά και 200 κατηγορίες</u>

	BR	**RAKEL**
Hamming Loss	**0.2673**	0.2907
Micro-averaged F-Measure	0.2507	**0.2546**
CPU time	**794.83**	5632.04
Memory consumption	**967.5345**	1349.494

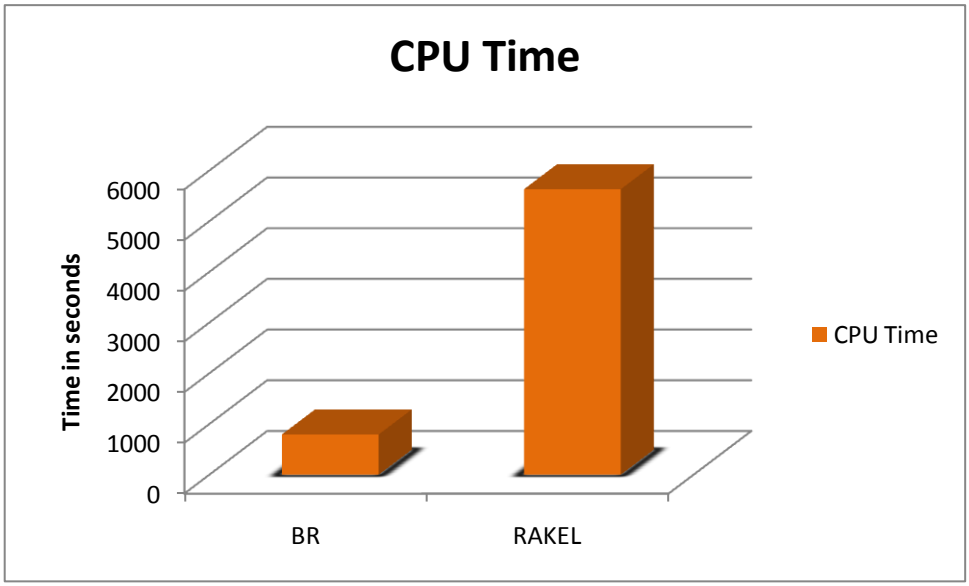
Πίνακας 7.2.2.3.1: Μετρήσεις που έχουν ληφθεί από πείραμα στο σύνολο στιγμιότυπων D4



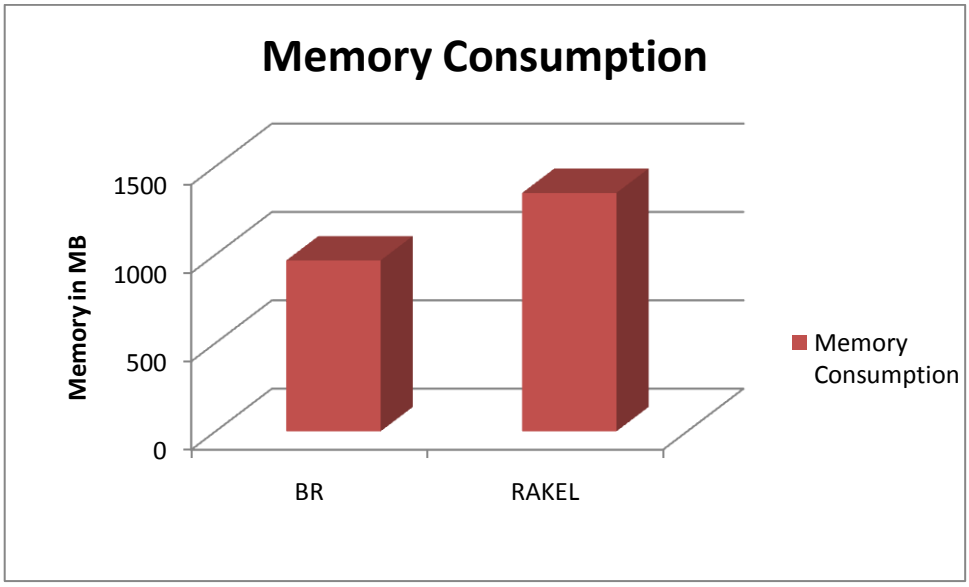
Γράφημα 7.2.2.3.1: Hamming Loss των 2 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



Γράφημα 7.2.2.3.2: Micro-averaged F-Measure των 2 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



Γράφημα 7.2.2.3.3: CPU time των 2 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



Γράφημα 7.2.2.3.4: Χρήση μνήμης των 2 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4

Σύμφωνα με τις πιο πάνω μετρήσεις, των γραφικών παραστάσεων (7.2.2.3.1 - 7.2.2.3.4), η καλύτερη μέθοδος προκύπτει να είναι η BR, όσον αφορά τις μετρήσεις

Hamming Loss, χρόνου CPU που χρειάστηκε για εκπαίδευση και αξιολόγηση του μοντέλου και δέσμευσης μνήμης, ωστόσο η RAKEL έχει ψηλότερο Micro Averaged F-Measure.

Από τις μετρήσεις αυτές μπορούμε να συμπεράνουμε ότι ο χρόνος CPU και η χρήση μνήμης επηρεάζονται από τον τρόπο με τον οποίο λειτουργούν οι μέθοδοι BR και RAKELL καθώς ο αλγόριθμος Naïve Bayes αποτελεί αλγόριθμο εκμάθησης και στις δύο αυτές μεθόδους.

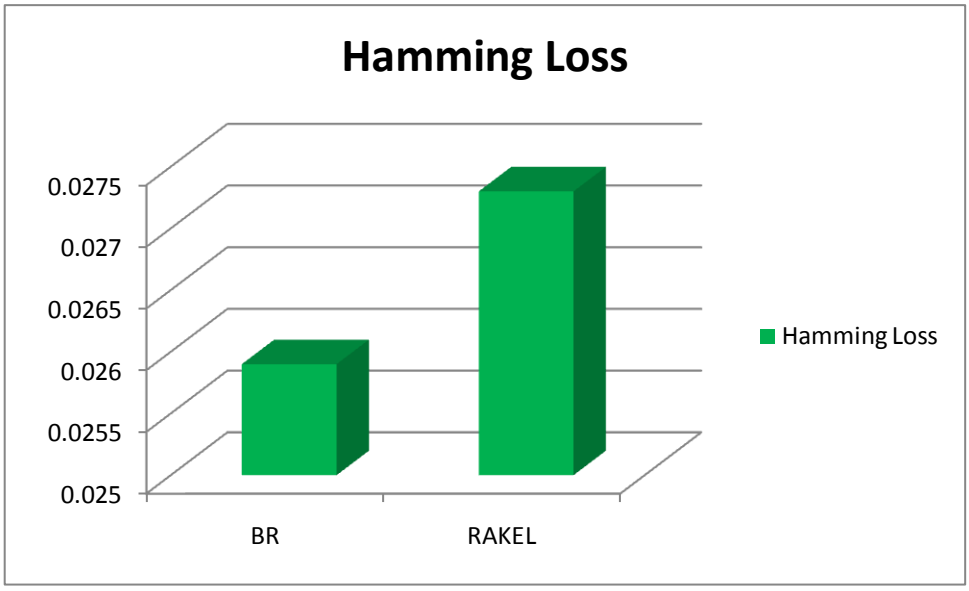
7.2.2.4 Χρήση SMO ως αλγόριθμος εκπαίδευσης (base learner)

Ακολούθως, παρουσιάζουμε μερικά πειράματα στα οποία χρησιμοποιήσαμε ένα διαφορετικό είδος single-label ταξινομητή ως αλγόριθμο βάσης, τον Sequential Minimal Optimization (SMO) [30] για να παρατηρήσουμε τι αλλαγές θα συμβούν στην απόδοση των 3 μεθόδων όσον αφορά τις μετρικές Hamming Loss, Micro Averaged F-measure, το χρόνο CPU και την δέσμευση μνήμης.

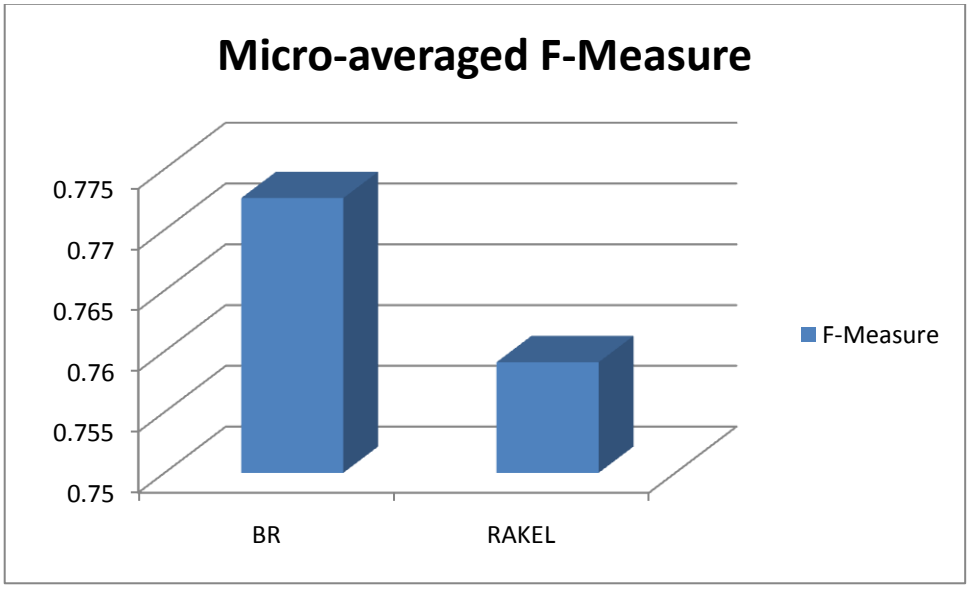
500 χαρακτηριστικά και 200 κατηγορίες

	BR	**RAKEL**
Hamming Loss	**0.0259**	0.0273
Micro-averaged F-Measure	**0.7726**	0.7591
CPU time	**10071.26**	27621.48
Memory consumption	**1370.1815**	2203.43685

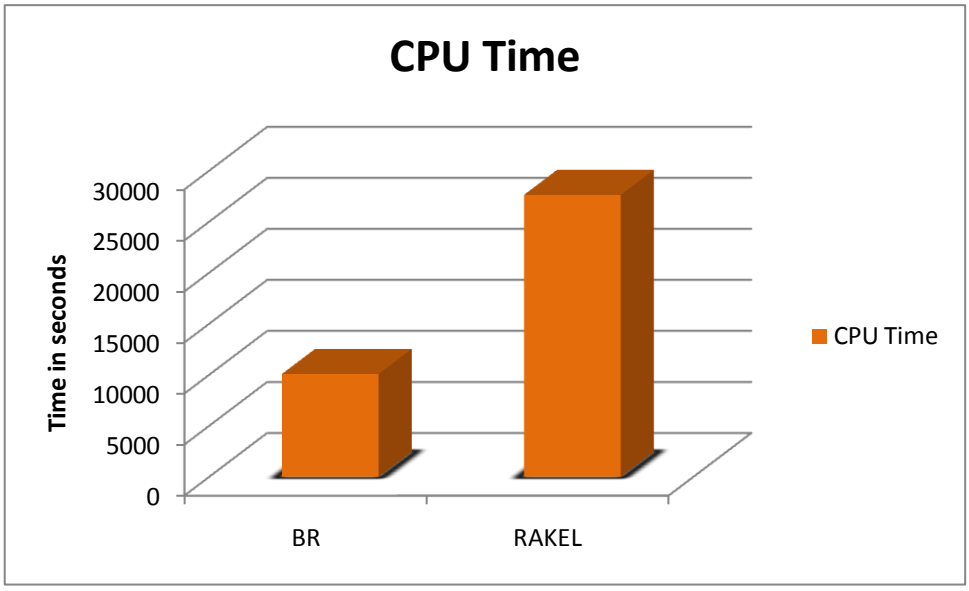
Πίνακας 7.2.2.4.1: Μετρήσεις που έχουν ληφθεί από πείραμα στο σύνολο στιγμιότυπων D3



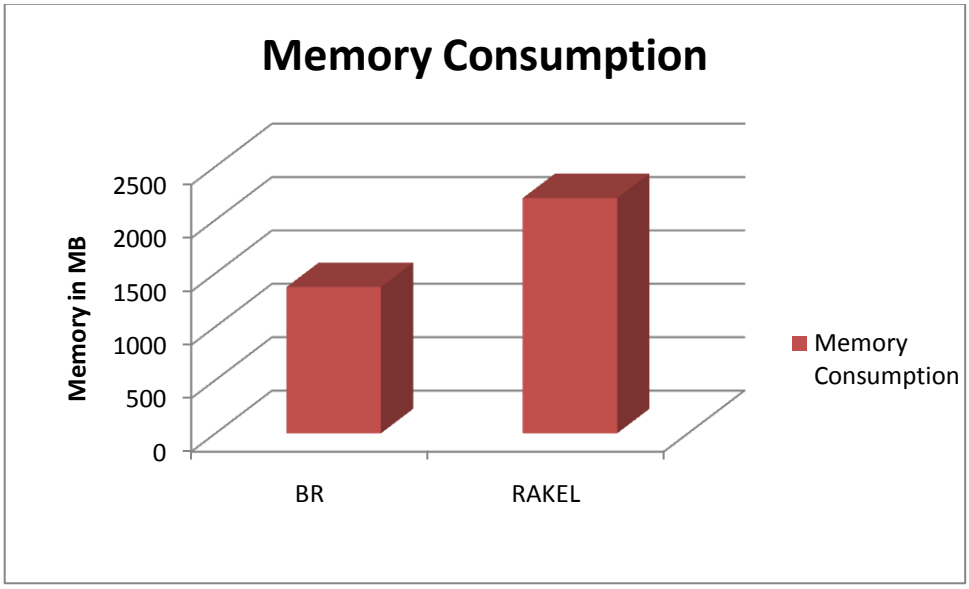
Γράφημα 7.2.2.4.1: Hamming Loss των 2 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



Γράφημα 7.2.2.4.2: Micro-averaged F-Measure των 2 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



Γράφημα 7.2.2.4.3: CPU time των 2 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4



Γράφημα 7.2.2.4.4: Χρήση μνήμης των 2 μεθόδων ταξινόμησης από πείραμα στο σύνολο στιγμιότυπων D_4

Σύμφωνα με τις πιο πάνω μετρήσεις, των γραφικών παραστάσεων (7.2.2.4.1 - 7.2.2.4.4), η καλύτερη μέθοδος προκύπτει να είναι η BR αφού παρουσιάζει καλύτερα

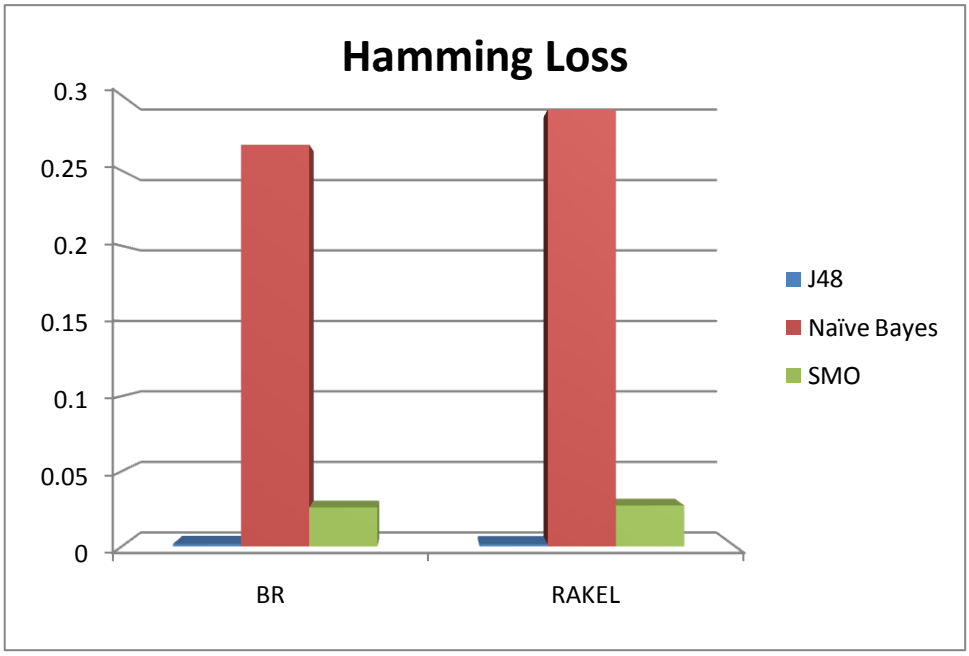
αποτελέσματα σε όλες τις μετρικές αξιολόγησης που χρησιμοποιήσαμε. Η διαφορά όμως όσον αφορά τις μετρικές Hamming Loss και Micro-averaged F-Measure των δύο αυτών μεθόδων είναι πάρα πολύ μικρή.

7.2.2.5 Σύγκριση BR και RAKEL με διαφορετικούς αλγόριθμους εκπαίδευσης

500 χαρακτηριστικά και 200 κατηγορίες

Multi-label Classification methods	Base Learners	Hamming Loss	Micro-averaged F-Measure	CPU time	Memory consumption
BR	J48	0.0016	0.9879	845.78	1108.69
	Naïve Bayes	0.2673	0.2507	**794.83**	**967.5345**
	SMO	0.0259	0.7726	10071.26	1370.1815
RAKEL	J48	**0.0014**	**0.9896**	5152.2	1924.661
	Naïve Bayes	0.2907	0.2546	5632.04	1349.494
	SMO	0.0273	0.7591	27621.48	2203.43685

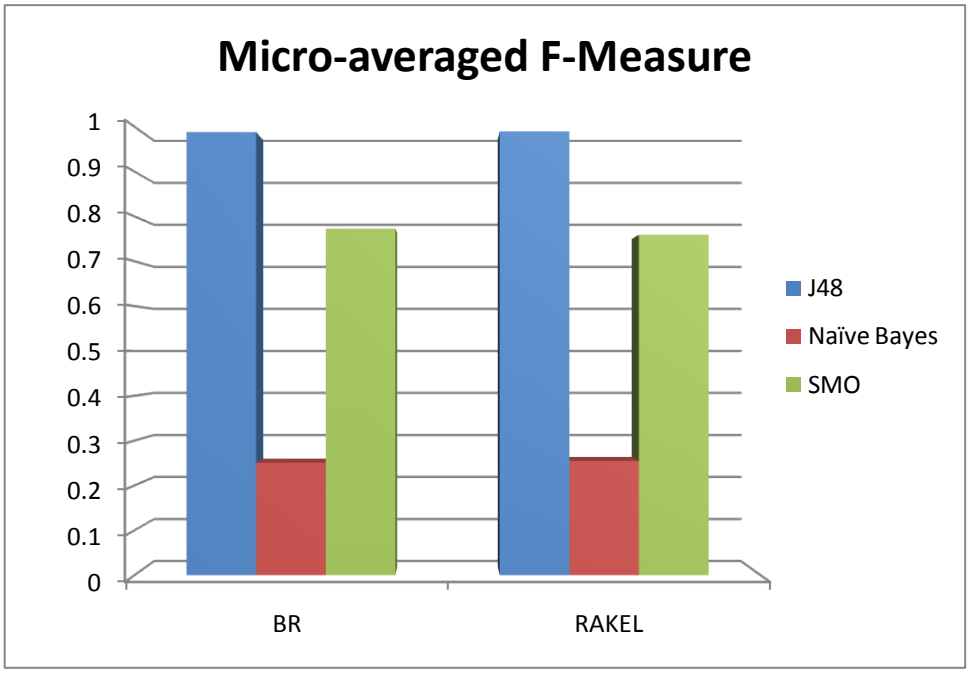
Πίνακας 7.2.2.5.1: Συγκεντρωτικές μετρήσεις για τις μεθόδους BR, LP, RAKEL χρησιμοποιώντας τους base learners J48, Naïve Bayes και SMO



Γράφημα 7.2.2.5.1: Συγκριτικά αποτελέσματα για τη μετρική Hamming Loss μεταξύ των ίδιων μεθόδων ταξινόμησης, αλλά με διαφορετικό αλγόριθμο εκπαίδευσης σε στιγμιότυπα των 500 χαρακτηριστικών και 200 κατηγοριών.

Παρατηρώντας την γραφική παράσταση (7.2.2.5.1) καταλαβαίνουμε ότι και οι δύο μέθοδοι παρουσιάζουν πιο καλά αποτελέσματα με την χρήση του ταξινομητή J48 παρά με τη χρήση του NB ή του SMO. Συγκεκριμένα η μέθοδος BR παρουσιάζει ελάχιστα χειρότερα αποτελέσματα από την RAKEL όταν χρησιμοποιεί τον J48, αλλά με τη χρήση του NB και SMO κάνει λιγότερα σφάλματα.

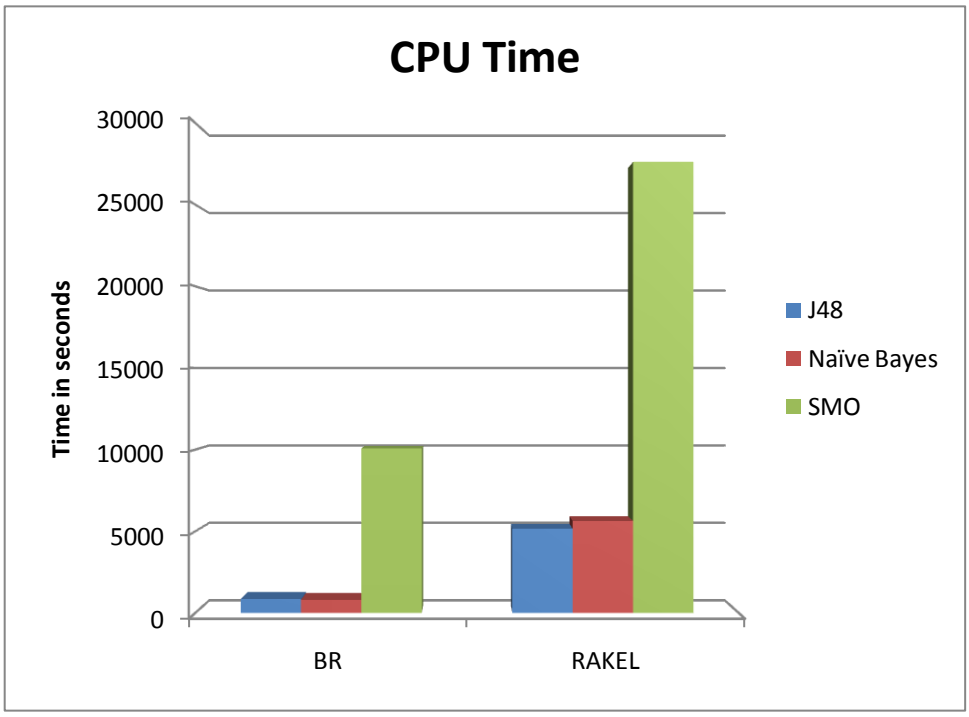
Αξιοσημείωτο είναι ότι και οι δύο μέθοδοι αντιδρούν με παρόμοιο τρόπο στη χρήση διαφορετικών αλγορίθμων εκπαίδευσης. Αυτό ίσως οφείλεται σε μεγάλο βαθμό στους αλγόριθμους εκπαίδευσης και όχι στις μεθόδους multi-label ταξινόμησης



Γράφημα 7.2.2.5.2: Συγκριτικά αποτελέσματα για τη μετρική Micro Average F-measure μεταξύ των ίδιων μεθόδων ταξινόμησης, αλλά με διαφορετικό αλγόριθμο εκπαίδευσης σε στιγμιότυπα των 500 χαρακτηριστικών και 200 κατηγοριών.

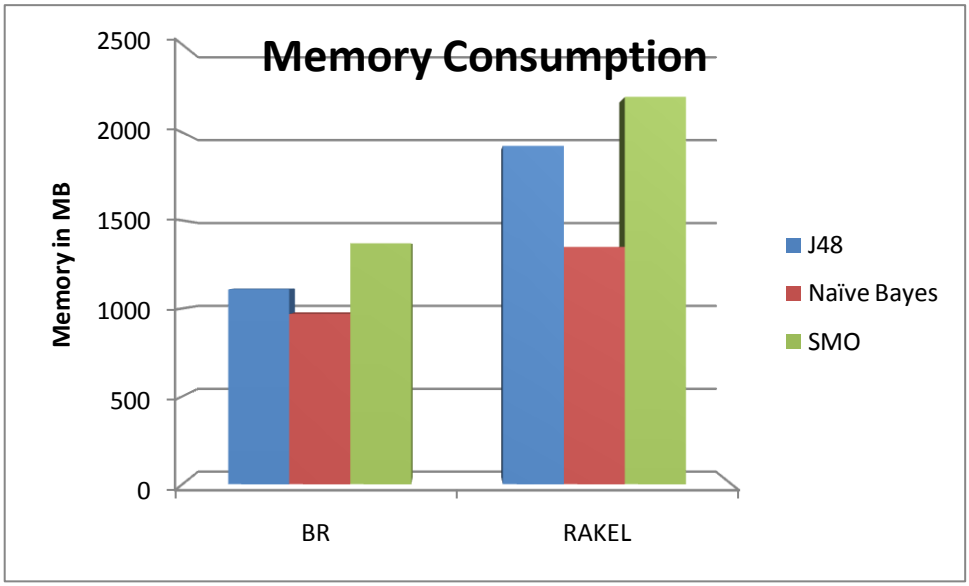
Μια γενική παρατήρηση που προκύπτει από την γραφική παράσταση (7.2.2.5.2) είναι ότι και οι δύο μέθοδοι παρουσιάζουν παρόμοια αποτελέσματα όπως και στη μετρική σφάλματος Hamming Loss (7.2.2.5.1). Στην γραφική παράσταση αυτή διαφαίνεται ότι η μέθοδος RAKEL έχει σημειώσει ελάχιστα καλύτερα αποτελέσματα από την BR για τους αλγόριθμους εκπαίδευσης J48 και Naïve Bayes, ενώ στην περίπτωση του SMO ισχύει το αντίθετο.

Ακόμα μια παρατήρηση που προκύπτει από την πιο πάνω γραφική παράσταση είναι ότι οι (single-label) αλγόριθμοι J48 και SMO παρουσιάζουν καλυτέρα αποτελέσματα από τον Naïve Bayes. Αυτό μπορεί να επεξηγηθεί εύκολα, αφού ο SMO και γενικά οι αλγόριθμοι της κατηγορίας Support Vector Machine (SVM) παρόλο ότι είναι αργοί ταξινομητές είναι πολύ ακριβείς, ενώ αλγόριθμοι που χρησιμοποιούν δέντρα απόφασης όπως ο J48 παρόλο ότι είναι πολύ γρήγοροι και δεν εκτελούν πολλούς υπολογισμούς, στην προκειμένη περίπτωση φαίνεται ότι τα δεδομένα ευνοούν τη χρήση τέτοιων αλγορίθμων ως αλγόριθμους εκπαίδευσης.



Γράφημα 7.2.2.5.3: Συγκριτικά αποτελέσματα για τον χρόνο CPU μεταξύ των ίδιων μεθόδων ταξινόμησης, αλλά με διαφορετικό αλγόριθμο εκπαίδευσης σε στιγμιότυπα των 500 χαρακτηριστικών και 200 κατηγοριών.

Όπως παρουσιάζεται και από τη γραφική παράσταση (7.2.2.5.3) η μέθοδος BR σημείωσε πολύ χαμηλότερους χρόνους από την RAKEL για όλους τους αλγόριθμους εκπαίδευσης. Αυτό επιβεβαιώνει το γεγονός ότι ο χρόνος CPU είναι ανάλογος της χρονικής πολυπλοκότητας των μεθόδων BR και RAKEL, όπως επίσης και το γεγονός ότι η χρονική πολυπλοκότητα των αλγόριθμων εκπαίδευσης J48, Naïve Bayes και SMO επηρεάζει το συνολικό αποτέλεσμα της μέτρησης.



Γράφημα 7.2.2.5.4: Συγκριτικά αποτελέσματα για τη δέσμευση μνήμης μεταξύ των δύο μεθόδων ταξινόμησης, αλλά με διαφορετικό αλγόριθμο εκπαίδευσης σε στιγμιότυπα των 500 χαρακτηριστικών και 200 κατηγοριών.

Τέλος, σχετικά με την μνήμη που έχουν δεσμεύσει οι δύο μέθοδοι για την εκπαίδευση και αξιολόγηση του μοντέλου, παρατηρούμε από τη γραφική παράσταση (7.2.2.5.4) ότι και για τις δύο μεθόδους BR και RAKEL δεσμεύτηκε περισσότερη μνήμη με τη χρήση δέντρου αποφάσεων J48 παρά με την χρήση NB, αλλά ακόμη περισσότερη μνήμη δεσμεύτηκε με τη χρήση του SMO. Όπως έχουμε παρατηρήσει και σε προηγούμενες γραφικές παραστάσεις η RAKEL χρειάζεται περισσότερη μνήμη για να εκτελεστεί αφού είναι μια πιο πολύπλοκη μέθοδος.

7.3 Αξιολόγησης διαδικασίας εμπλουτισμού αποτελεσμάτων από το Διαδίκτυο

Η αξιολόγηση θα γίνει με τη χρήση της μεθόδου precision@10 σε ένα σύνολο 10 ερωτημάτων τα οποία επιλέξαμε γνωρίζοντας ότι υπάρχουν στα ανεστραμμένα ευρετήρια του Minersoft και η μηχανή αναζήτησης θα επιστρέψει σχετικά αποτελέσματα.

Έτσι για κάθε ένα αποτέλεσμα του Minersoft επιστρέφεται ένα αποτέλεσμα από τη μηχανή αναζήτησης Yahoo το οποίο επιλέχτηκε από τα 20 πρώτα αποτελέσματα της

Yahoo. Αν το αποτέλεσμα αυτό σχετίζεται με στο Minersoft τότε το precision του αυξάνεται κατά ένα, με μέγιστο το 10.

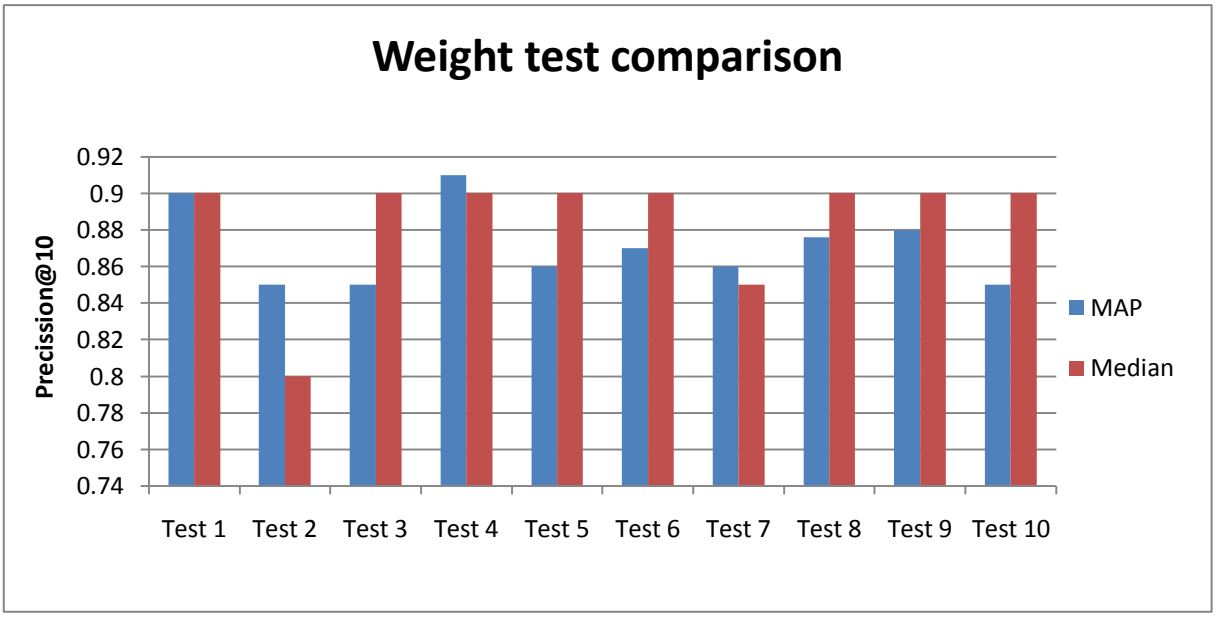
Επίσης έγιναν δοκιμές σε 10 διαφορετικά βάρη για κάθε κανόνα ώστε να επιλεχτεί αυτό που δίνει τα πιο ικανοποιητικά αποτελέσματα.

Tests	**Different weighting over 4 rules**			
	Rule 1	Rule 2	Rule 3	Rule 4
Weight test 1	0.5	0.2	0.2	0.1
Weight test 2	0.2	0.5	0.2	0.1
Weight test 3	0.2	0.2	0.5	0.1
Weight test 4	0.3	0.3	0.3	0.1
Weight test 5	0.2	0.3	0.4	0.1
Weight test 6	0.2	0.4	0.3	0.1
Weight test 7	0.3	0.2	0.4	0.1
Weight test 8	0.3	0.4	0.2	0.1
Weight test 9	0.4	0.2	0.3	0.1
Weight test 10	0.4	0.3	0.2	0.1

Πίνακας 7.3.1: Ανάθεση διαφορετικών βαρών για κάθε κανόνα

	Precision@10 for weight tests									
Query	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
ruby	0.9	0.8	0.9	1	0.9	0.8	0.9	0.9	0.9	0.9
python	0.9	0.8	0.9	0.9	0.9	0.9	0.8	0.9	0.9	0.9
gedit	1	0.8	0.9	0.9	0.8	0.8	0.8	0.7	0.8	0.7
php	0.9	0.9	0.9	0.9	0.8	0.8	0.8	0.8	0.9	0.8
perl	0.9	0.9	0.8	0.9	0.9	0.9	0.9	0.9	1	0.9
xml	1	1	0.9	0.9	0.7	1	0.9	0.9	0.9	0.9
mysql	0.8	0.8	0.7	0.8	0.9	0.8	0.8	0.9	0.8	0.8
C++	0.9	0.8	0.8	0.9	0.9	0.9	0.8	0.8	0.8	0.8
java	0.8	0.8	0.8	0.9	0.9	0.9	1	1	0.9	0.9
parser	0.9	0.9	0.9	1	0.9	0.9	0.9	0.9	0.9	0.9
MAP	0.9	0.85	0.85	0.91	0.86	0.87	0.86	0.876	0.88	0.85
Median	0.9	0.8	0.9	0.9	0.9	0.9	0.85	0.9	0.9	0.9

Πίνακας 7.3.2: Δοκιμές σε 10 ερωτήματα για κάθε μια από τις 10 διαφορετικές αναθέσεις βαρών στους 4 κανόνες



Γράφημα 7.3.1: Το σχήμα αναπαριστά τον Mean Average Precision και median για τις 10 δοκιμές βάρους σε σύνολο 10 ερωτημάτων.

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q}$$

Τύπος 7.3.1: Ο μέσος όρος όπου Q= 10, αριθμος των επερωτήσεων που δοκιμάστηκαν, ενώ AveP(q) είναι ο αριθμός των σχετικών αποτελεσμάτων από τα 10.

Όπως αναπαριστάται στον πίνακα (7.3.1) δοκιμάσαμε διαφορετικά βάρη κρατώντας μόνο το βάρος για τον κανόνα 4 σταθερό και με λιγότερο βάρος. Αυτό γιατί θέλουμε μεν να λαμβάνουμε υπόψη την κατανομή των αποτελεσμάτων της Yahoo αλλά όχι να εξάγουμε συμπεράσματα αποκλειστικά μόνο από αυτό. Οι τιμές του precision@10 που προκύπτουν για κάθε ένα από τα 10 ερωτήματα είναι στο εύρος 0.7 με 1 όπως αναπαρίστανται στον πίνακα (7.3.2). Όπως παρατηρούμε στη γραφική παράσταση (7.3.1) οι δύο ομάδες βαρών που έχουν τα ψηλότερα αποτελέσματα για τον Mean Average Precision -τύπος (7.3.1)- και median είναι η δοκιμή 1 και η 4, με ελάχιστη διαφορά η τελευταία να έχει πιο ψηλό Mean Average Precision. Επομένως το γενικό συμπέρασμα είναι ότι όταν ισορροπούν τα βάρη μεταξύ των πρώτων τριών κανόνων, δηλαδή όταν τους λαμβάνουμε και τους τρεις εξίσου υπόψη, παίρνουμε πιο ακριβή αποτελέσματα, ενώ σημαντικότεροι από τους τέσσερις αυτούς κανόνες κρίνονται ο πρώτος και ο δεύτερος. Τα δεύτερα καλύτερα αποτελέσματα, είναι οι δοκιμές 1, 2, 6, 9 και 10 όπου αντιστοιχούν στις ομάδες των βαρών με μεγαλύτερα τα βάρη του κανόνα 1 και 2. Αυτό συμβαίνει για είναι σημαντικό η λέξη που αναζητείται στη μηχανή

αναζήτησης Yahoo να υπάρχει στον τίτλο και στην περιγραφή του αποτελέσματος. Όσο για τον κανόνα 3 όταν είναι μέγιστος δεν επιστρέφονται τόσο καλά αποτελέσματα αφού δεν είναι όλα τα αποτελέσματα της αναζήτησης οικοσελίδες κάποιου λογισμικού.

Επίσης παρατηρώντας την τιμή του μέσου για όλες τις μετρήσεις είναι αρκετά ψηλή, της τάξης του 0.8.

Κεφάλαιο 8

Συμπεράσματα

8.1 Συμπεράσματα

Το Minersoft είναι μια μηχανή αναζήτησης η οποία εκτελεί αναζήτηση σε Cloud και Grid υπολογιστικές υποδομές, ώστε οι χρήστες του να μπορούν να πληροφορηθούν για την τοποθεσία λογισμικού, το οποίο ενδιαφέρονται να χρησιμοποιήσουν.

Για την εύκολη χρήση της μηχανής αναζήτησης του Minersoft, αναπτύξαμε μια γραφική διεπαφή μέσω της οποία οι χρήστες θα μπορούν να υποβάλλουν τις ερωτήσεις τους στη μηχανή αναζήτησης. Η γραφική διεπαφή προσφέρει δύο ειδών αναζητήσεις, την απλή αναζήτηση και την εξειδικευμένη. Στην απλή αναζήτηση ο χρήστης απλώς υποβάλει το ερώτημα του και παίρνει πίσω την απάντηση, ενώ στην εξειδικευμένη εκτός από το ερώτημα του μπορεί να επιλέξει αν θέλει να του επιστραφούν μόνο αποτελέσματα τα οποία θα περιγράφουν είτε *binary*, είτε *library*, είτε *script/source* αρχεία τα οποία είναι εγκατεστημένα στο Cloud.

Επιπλέον, στόχος της διπλωματικής αυτής εκτός από την δημιουργία της γραφικής διεπαφής ήταν να παρέχονται στους χρήστες περισσότερες πληροφορίες για το λογισμικό που επιστρέφεται με το τέλος της αναζήτησης.

Για την εκπλήρωση του στόχου αυτού, υλοποιήσαμε διαδικασίες οι οποίες επιτρέπουν στους χρήστες να εισάγουν σε κάθε αποτέλεσμα ετικέτες. Οι ετικέτες είναι ένας μικρός αριθμός από λέξεις οι οποίες περιγράφουν το αποτέλεσμα στο οποίο αναθέτονται. Αυτές οι επιπλέον πληροφορίες που ανατέθηκαν από τους χρήστες αποτελούν δεδομένα εκπαίδευσης ενός ταξινομητή. Ο ταξινομητής αυτός, αφού εκπαιδευτεί θα είναι σε θέση να ταξινομεί τα αρχεία λογισμικού σε διάφορες κατηγορίες οι οποίες δεν είναι άλλες από τις ετικέτες που έχουν δώσει οι χρήστες. Για την εκτέλεση της ταξινόμησης ο ταξινομητής στηρίζεται στις πληροφορίες πού περιέχει το κάθε αρχείο των ανεστραμμένων ευρετηρίων δηλαδή τους πιο σημαντικούς όρους σε όλα τα αρχεία και τη συχνότητα εμφάνισης τους σε κάθε ένα αρχείο, το γνωστό βάρος TF-IDF.

Για την επιλογή του πιο αποτελεσματικού και αποδοτικού ταξινομητή όπου θα χρησιμοποιηθεί για την ταξινόμηση των αρχείων των ανεστραμμένων ευρετηρίων που περιγράφουν λογισμικό, προχωρήσαμε σε μια σειρά πειραμάτων στην οποία

συγκρίναμε 5 γνωστές μεθόδους ταξινόμησης πολλαπλών ετικετών οι οποίες φημίζονται για τα καλά αποτελέσματα τους σε ταξινόμηση κειμένου. Οι μέθοδοι αυτές είναι οι BR (Binary Relevance),LP (Label Powerset), RAKEL (RAndom *k*-labELsets), ML*k*NN (Multi Label *k*-NN) και BPMLL (Backpropagation Multi-Label Learning).

Αυτό που θέλαμε να εξετάσουμε μέσω των πειραμάτων είναι ποια από τις μεθόδους ταξινόμησης πετυχαίνει καλύτερα αποτελέσματα στην ταξινόμηση των αρχείων και κάνει τα λιγότερα λάθη. Για να το μετρήσουμε αυτό χρησιμοποιήσαμε δύο διαφορετικές μετρικές, την μετρική Hamming Loss και την μετρική Micro-averaged F-Measure. Επίσης σημαντικό είναι πόσο μεγάλος είναι ο χρόνος εκπαίδευσης και ταξινόμησης του μοντέλου των 5 διαφορετικών μεθόδων ταξινόμησης καθώς και η κατανάλωση μνήμης για κάθε μια από τις μεθόδους.

Ακόμη, μας ενδιέφερε να εξετάσουμε την επίδραση που έχει στις 5 αυτές μεθόδους η αύξηση του αριθμού των χαρακτηριστικών βάσει των οποίων γίνεται η ταξινόμηση και η αύξηση του αριθμού των κατηγοριών.

Τα αποτελέσματα των πειραμάτων που αφορούν τη μεταβολή στον αριθμό των χαρακτηριστικών παρουσιάζονται στον πίνακα (8.1.1). Όπως έχουμε παρατηρήσει και από το Κεφάλαιο 7, η μέθοδος η οποία συγκέντρωσε τα καλύτερα αποτελέσματα όσον αφορά την ορθότητα της ταξινόμησης δηλαδή το μεγαλύτερο μέσο όρο στην μετρική Micro-averaged F-Measure καθώς και το μικρότερο σφάλμα -πίνακας (8.1.1)- πάνω σε 3 σύνολα δεδομένων των 500, 1000 και 2000 χαρακτηριστικών είναι η RAKEL. Παρ' όλα αυτά η μέθοδος Binary Relevance έχει ελάχιστα χειρότερες τιμές από την RAKEL στις δύο μετρικές που προαναφέρθηκαν. Επιπρόσθετα η μέθοδος BR έχει σημειώσει πολύ καλύτερο χρόνο από την RAKEL όσον αφορά την εκπαίδευση του ταξινομητή και την ταξινόμηση των δεδομένων, αν και όχι τον βέλτιστο. Επίσης στο πείραμα αυτό η μέθοδος BR έχει σημειώσει τη μικρότερη μέση κατανάλωση μνήμης για την εκπαίδευση του μοντέλου του ταξινομητή και την ταξινόμηση των δεδομένων.

Multi-label classification methods	Data Sets	Hamming Loss	Micro-averaged F-Measure	CPU time	Memory consumption
BR	**D_2**	0.0012	0.98	1072.24	1755.88
	D_4	0.0016	0.9879	845.78	1108.69
	D_5	0	0.9998	1973.16	1149.67
	Μέσος όρος	0.00093	0.98923	1297.06	**1338.08**
LP	**D_2**	0.0509	0.6371	454.05	2437.88
	D_4	0.0391	0.7001	217.59	2311.98
	D_5	0.0417	0.6817	682.86	1104.57
	Μέσος όρος	0.0439	0.67297	**451.5**	1951.48
RAKEL	**D_2**	0.001	0.997	5319.23	1060.28
	D_4	0.0014	0.9896	5152.2	1924.66
	D_5	0	0.9997	10779	3756.31
	Μέσος όρος	**0.0008**	**0.99543**	7083.49	2247.08
MLKNN	**D_2**	0.0424	0.6428	704.44	1877.19
	D_4	0.0311	0.7285	351.39	1902.24
	D_5	0.0335	0.7021	1517.41	2241.75
	Μέσος όρος	0.03567	0.69113	857.747	2007.06
BPMLL	**D_2**	0.9161	0.1388	1803.03	2009.79
	D_4	0.2726	0.1228	846.75	1308.63
	D_5	0.9151	0.1288	4394.72	4442.64
	Μέσος όρος	0.70127	0.13013	2348.17	2587.02

Πίνακας 8.1.1: Συγκεντρωτικά αποτελέσματα πειράματος αύξησης αριθμού χαρακτηριστικών

Τα αποτελέσματα των πειραμάτων που αφορούν τη μεταβολή στον αριθμό των κατηγοριών παρουσιάζονται στον πίνακα (8.1.2). Η μέθοδος η οποία συγκέντρωσε τα καλύτερα αποτελέσματα όσον τη μετρική Micro-averaged F-Measure πάνω σε 3 σύνολα δεδομένων των 100, 200 και 4000 κατηγοριών είναι και πάλι η RAKEL. Σε αντίθεση με πριν η μέθοδος Binary Relevance έχει τώρα σημειώσει καλύτερες τιμές όσον αφορά τις εσφαλμένες ταξινομήσεις από την RAKEL. Επίσης η μέθοδος BR έχει σημειώσει πολύ καλύτερο χρόνο από την RAKEL όσον αφορά την εκπαίδευση του ταξινομητή και την ταξινόμηση των δεδομένων όπως και στο προηγούμενο πείραμα. Ωστόσο στο πείραμα αυτό η μέθοδος RAKEL έχει σημειώσει τη μικρότερη μέση κατανάλωση μνήμης για την εκπαίδευση του μοντέλου του ταξινομητή και την ταξινόμηση των δεδομένων.

Οι δύο μέθοδοι με τις καλύτερες τιμές είναι η RAKEL και η BR, με την BR να είναι κατά πολύ πιο γρήγορη από την RAKEL αυτό οφείλεται στο διαφορετικό τρόπο που λειτουργούν οι δύο μέθοδοι. Η BR είναι μια απλή μέθοδος η οποία χρησιμοποιεί για κάθε ετικέτα ένα δυαδικό ταξινομητή ο οποίος εκτελεί μια απόφαση και στο τέλος όλο το σύνολο των δυαδικών αυτών ταξινομητών συγχωνεύει τα αποτελέσματά του. Σε αντίθεση η RAKEL είναι μια πιο πολύπλοκη μέθοδος η οποία χρησιμοποιεί LP ταξινομητές πάνω σε ένα τυχαίο υποσύνολο των κατηγοριών. Είναι αναμενόμενο λοιπόν ότι παρόλα τα καλά αποτελέσματα της η RAKEL θα είναι πολύ αργή μέθοδος.

Multi-label classification methods	**Data Sets**	**Hamming Loss**	**Micro-averaged F-Measure**	**CPU time**	**Memory consumption**
BR	**D_1**	0.0012	0.9889	370.44	1276.43
	D_2	0.0012	0.98	1072.24	1755.88
	D_3	0.003	0.9678	4287.02	3361.7
	Μέσος όρος	**0.0018**	0.9789	1909.9	2131.33
LP	**D_1**	0.0595	0.6973	262.57	606.231
	D_2	0.0509	0.6371	454.05	2437.88
	D_3	0.0394	0.568	370.67	3607.24

	Μέσος όρος	0.04993	0.63413	**362.43**	2217.12
RAKEL	**D_1**	0.001	0.9988	3851.03	1457.73
	D_2	0.001	0.997	5319.23	1060.28
	D_3	0.0047	0.949	10977	2135.37
	Μέσος όρος	0.00223	**0.9816**	6715.75	**1551.13**
MLKNN	**D_1**	0.0533	0.6897	581.51	916.395
	D_2	0.0424	0.6428	704.44	1877.19
	D_3	0.0302	0.5882	632.24	3443.95
	Μέσος όρος	0.04197	0.64023	639.397	2079.18
BPMLL	**D_1**	0.4941	0.0976	670.19	1079.78
	D_2	0.9161	0.1388	1803.03	2009.79
	D_3	0.9165	0.0947	2980.96	1993.02
	Μέσος όρος	0.77557	0.11037	1818.06	1694.2

Πίνακας 8.1.2: Συγκεντρωτικά αποτελέσματα πειράματος αύξησης αριθμού κατηγοριών

Τέλος για τις μεθόδους BR, LP και RAKEL οι οποίες χρησιμοποιούν ως base learners, ταξινομητές μονής ετικέτας, εξετάσαμε ποιος από τους base learners J48, Naïve Bayes και SMO παρουσιάζει καλύτερα αποτελέσματα χρησιμοποιώντας το σύνολο δεδομένων των 500 χαρακτηριστικών, 200 κατηγοριών και 10000 στιγμιότυπων. Όπως φαίνεται και στον πίνακα (8.1.3) τα καλυτέρα αποτελέσματα όσον αφορά την ορθότητα της ταξινόμησης τα συγκέντρωσε η μέθοδος RAKEL με base learner τον J48. Καλύτερα αποτελέσματα σε χρόνο τα σημείωσε η μέθοδος LP με base learner τον J48, ενώ τα καλύτερα αποτελέσματα από πλευράς μνήμης τα σημείωσε πάλι η μέθοδος LP όμως με base learner τον Naïve Bayes.

Εμπλουτισμός αποτελεσμάτων από το Διαδίκτυο

Εκτός από τις πληροφορίες που παρέχουν οι ίδιοι οι χρήστες μέσω των ετικετών και την αυτοματοποιημένη διαδικασία της ταξινόμησης, ένας άλλος τρόπος να εμπλουτιστούν τα αποτελέσματα της μηχανής αναζήτησης του Mineroft με περισσότερες πληροφορίες ήταν να συνδυαστούν με πληροφορίες από το Διαδίκτυο.

Η υλοποίηση της διαδικασίας η οποία είναι υπεύθυνη γι' αυτό, περιλαμβάνει τη χρήση του Yahoo BOSS ενός (API), για την εύκολη χρήση της αναζήτησης της Yahoo, καθώς και την εφαρμογή κάποιων ευριστικών κανόνων για την επιλογή του πιο σχετικού αποτελέσματος από την Yahoo.

Multi-label classification methods	**Base learners**	**Hamming Loss**	**Micro-averaged F-Measure**	**CPU time**	**Memory consumption**
BR	**J48**	0.0016	0.9879	845.78	1108.69
	Naïve Bayes	0.2673	0.2507	**794.83**	**967.5345**
	SMO	0.0259	0.7726	10071.26	1370.1815
RAKEL	**J48**	**0.0014**	**0.9896**	5152.2	1924.661
	Naïve Bayes	0.2907	0.2546	5632.04	1349.494
	SMO	0.0273	0.7591	27621.48	2203.43685

Πίνακας 8.1.3: Συγκεντρωτικές μετρήσεις για τις μεθόδους BR, LP, RAKEL χρησιμοποιώντας τους base learners J48, Naïve Bayes και SMO

8.2 Μελλοντική εργασία

Σχετικά με την μελλοντική εργασία υπάρχουν αρκετά πειράματα τα οποία θα μπορούσαν να ερευνηθούν ακόμα σχετικά με τους υφιστάμενους αλγορίθμους. Για παράδειγμα αν αλλάζει και πως αλλάζει η συμπεριφορά των αλγορίθμων για

διαφορετικού μεγέθους συνόλων δεδομένων. Πώς συμπεριφέρονται οι αλγόριθμοι όταν εκπαιδεύουν τα μοντέλα τους πάνω σε μεγάλα σύνολα δεδομένων.

Επίσης, θα μπορούσε να επεκταθεί ο πειραματισμός και σε άλλες μεθόδους ταξινόμησης πολλαπλών ετικετών που ανήκουν στην κατηγορία των Problem transformation μεθόδων όπως είναι η Calibrated Label Ranking (CLR) [30], ή της κατηγορίας των Algorithm adaptation μεθόδων που χρησιμοποιούν δέντρα αποφάσεων, πιθανοτικές μεθόδους, νευρωνικά δίκτυα, μηχανές υποστήριξης διανυσμάτων και αλγόριθμοι οκνηρής μάθησης.

Ακόμη, θα μπορούσαν να ερευνηθούν περισσότεροι αλγόριθμοι ταξινόμησης μονής ετικέτα ως αλγόριθμοι εκπαίδευσης ή να γίνουν περισσότερα πειράματα πάνω στους υφιστάμενους J48, Naïve Bayes και SMO όπου ο αριθμός των χαρακτηριστικών, ετικετών και στιγμιότυπων θα μεταβάλλεται.

Βιβλιογραφία

[1] A. Katsifodimos, “Minersoft: Searching software resources in large-scale grid and cloud infrastructures,” University of Cyprus, Nicosia, 2009.

[2] G. Pallis, A. Katsifodimos and M. D. Dikaiakos, “Effective Keyword Search for Software Resources installed in Large-scale Grid Infrastructures,” In WI-IAT '09 Proceedings of the 2009 IEEE/WIC/ACM International Joint. Conference on Web Intelligence and Intelligent Agent Technology - Volume 01, pages 482-489, Washington, DC, USA, 2009. IEEE Computer Society.

[3] C. D. Manning, P. Raghavan and H. Schutze, Introduction to Information Retrieval, Cambridge University Press, 2009.

[4] I. Katakis, G. Tsoumakas and I. Vlahavas, “Multilabel Text Classification for Automated Tag Suggestion,” In Proceedings of the ECML/PKDD 2008 Discovery Challenge, Antwerp, Belgium, 2008.

[5] K. Trohidis, G. Tsoumakas, G. Kalliris and I. Vlahavas, “Multilabel Classification of Music into Emotions,” Proc. 9th International Conference on Music Information Retrieval (ISMIR 2008), pp. 325-330, Philadelphia, PA, USA, 2008.

[6] G. Tsoumakas, I. Katakis and I. Vlahavas, “Random k-Labelsets for Multi-Label Classification,” IEEE Transactions on Knowledge and Data Engineering, (PrePrint).

[7] G. Tsoumakas and I. Vlahavas, “Random k-Labelsets: An Ensemble Method for Multilabel Classification,” Proc. 18th European Conference on Machine Learning (ECML 2007), pp. 406-417, Warsaw, Poland, 17-21 September 2007.

[8] G. Tsoumakas, J. Vilcek, E. Spyromitros and I. Vlahavas, “Mulan: A Java Library for Multi-Label Learning,” Journal of Machine Learning Research (accepted for publication conditioned on minor revisions), 2010.

[9] "Mulan: A Java Library for Multi-Label Learning." sourceforge.net, Web. 16 March 2011. <http://mulan.sourceforge.net/>.

[10] M.-L. Zhang and Z.-H. Zhou, “Ml-knn: A lazy learning approach to multi-label learning. Pattern Recognition,” vol. 40, no. 7, pp. 2038–2048, 2007.

[11] M.-L. Zhang and Z.-H. Zhou, “Multilabel neural networks with applications to functional genomics and text categorization,” IEEE Transactions on Knowledge and Data Engineering 18 (10) (2006) 1338-1351.

[12] "Weka 3 - Data Mining with Open Source Machine Learning Software in Java." University of Waikato, Web. 3 May 2011. <http://www.cs.waikato.ac.nz/ml/weka/>.

[13] “Google Code Search.” Google, Web. September 2010. <http://www.google.com/codesearch>.

[14] “Google Web Search API - Google Code.” Google, Web. September 2010. <http://code.google.com/apis/websearch/>.

[15] “Open Source Code Search Engine - Black Duck Koders.” Black Duck Software, Inc., Web. September 2010. <http://www.koders.com/>.

[16] “Yahoo! Search BOSS - YDN.” Yahoo! Inc., Web. March 2010. <http://developer.yahoo.com/search/boss/>.

[17] O. Gospodnetić and E. Hatcher, Lucene in action, 1st ed, Manning Publications, 2005

[18] G. Tsoumakas, I. Katakis, I. Vlahavas, "Mining Multi-label Data," Data Mining and Knowledge Discovery Handbook, O. Maimon, L. Rokach (Ed.), Springer, 2nd edition, 2010.

[19] S. Gopal and Y. Yang, "Multilabel classification with meta-level features," in Proc. SIGIR, 2010, pp.315-322.

[20] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009), "The WEKA Data Mining Software: An Update", SIGKDD Explorations, Volume 11, Issue 1.

[21] I. H. Witten and E. Frank, Data Mining Practical Machine Learning Tools and Techniques, 2nd ed, Morgan Kaufmann, San Francisco, 2005.

[22] Ι. Κατάκης, «Μέθοδοι Μηχανικής Μάθησης για Αυτόματη Ταξινόμηση Κειμένων,» Αριστοτελείο Πανεπιστημίο, Θεσσαλονίκη, 2009.

[23] "Delicious." Yahoo, Web. April 2011. <http://www.delicious.com/>.

[24] "Flickr." Yahoo, Web. April 2011. <http://www.flickr.com/>.

[25] "Attribute-Relation File Format (ARFF)." University of Waikato, 1 November 2008. Web. April 2011. <http://www.cs.waikato.ac.nz/~ml/weka/arff.html>.

[26] Likun Liu, Lianghong Xu, Yongwei Wu, Guangwen Yang and Gregory R. Ganger. "SmartScan: Efficient Metadata Crawl for Storage Management Metadata Querying in Large File Systems," Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-10-112, Oct. 2010.

[27] "Amazon Elastic Compute Cloud (Amazon EC2)." Amazon, Web. September 2010. <http://aws.amazon.com/ec2/>.

[28] “Cloud Computing, Cloud Hosting & Online Storage by Rackspace Hosting. Mosso is now the Rackspace Cloud..” Rackspace, US Inc., Web. September 2010. <http://www.rackspace.com/cloud/>.

[29] J. Platt, "Fast Training of Support Vector Machines using Sequential Minimal Optimization," Advances in Kernel Methods - Support Vector Learning, B. Sch-lkopf, C. Burges, and A. Smola, eds., MIT Press 1998.

[30] J. Furnkranz, E. Hullermeier, E.L. Mencia, and K. Brinker, “Multilabel Classification via Calibrated Label Ranking,” Machine Learning, vol. 73, no. 2, pp. 133-153, Nov. 2008.

[31] Z. Guan, C. Wang, J. Bu, C. Chen, K. Yang, D. Cai, and X. He, “Document Recommendation in Social Tagging Services,” Proc. 2010 ACM International Conference on World Wide Web, Raleigh, North Carolina, April, 2010.

[32] G.V. Menezes, J.M. Almeida, F. Belém, M.A. Gonçalves, A. Lacerda, E.S.D. Moura, G.L. Pappa, A. Veloso, and N. Ziviani, “Demand-Driven Tag Recommendation,” in Proc. ECML/PKDD (2), 2010, pp.402-417.

[33] Y.S. Maarek, D.M. Berry, and G.E. Kaiser, “An Information Retrievel Approach For Automatically Constructing Software Libraries,” IEEE Transactions On Software Engineering, Vol. 17, No. 8 Aug. 1991, pp 800-813.

[34] A. Turpin, Y. Tsegay, D. Hawking, and H. E. Williams, “Fast generation of result snippets in web search,” In SIGIR ’07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pages 127–134, New York, NY, USA, 2007. ACM17

Παράρτημα Α

Παράρτημα Α: Σημαντικότερα κομμάτια κώδικα από κλάσεις που εμπλέκονται στην διαδικασία ανανέωσης ευρετηρίων με τις ετικέτες που ανάθεσαν οι χρήστες

UpdateIndexDeamon.java

Μέρος της μεθόδου run()

```
/*
*Implementation of method 'run', checking consistently and updates indexes
*/
	public void run() {

		try {
			while (true) {
				//delay daemon 7 days
				t.sleep(7*24*3600*1000);
				days +=7;

				//add new user comments in the HashMap to be processed
				CommentLoader c = new CommentLoader();

	c.renameFile(c.getClassPath()+"comments.txt",c.getClassPath()+"user_comments.txt");

				if(c.loadComments("user_comments.txt")) {

					this.setIndexesUpdInfo(c.getComments());
					this.loadIndexes();
					for(int i=0; i<this.indexesToUpdate; i++){
						System.out.println("updating
index:"+this.indexNames[i]);
						ThreadIndexUpdater t = new
ThreadIndexUpdater(this.indexwriter[i],this.indexsearcher[i],this.comments.get(indexNames[i]));
						executor.execute(t);
					}
					executor.shutdown();
				}

				//don't stop until all the indexes have updated
				if(this.start)
					while(!executor.isTerminated())
						t.sleep(3000);

				//check if classification processes can be executed
				File indexes = new File(getIndexesPath());
				int taggedDocuments=0;
				for(int i=0; i<indexes.list().length;i++){
					try {
						taggedDocuments +=
getTotalDocsWithUserTags(getIndexesPath()+"/"+indexes.list()[i]);
					} catch (ParseException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}
				}

				if(days==14){
					days =0;
```

```
                                        if(taggedDocuments>5000)
                                                //classification process
                                                this.classificationProcess = new
ClassifierUtil("versioned");

                                                //find training data
                                                classificationProcess.generateTrainingData(softIndex);
                                                softIndex=true;
                                                //train classifier and perform classification
                                                classificationProcess.performClassification();
                                        }
                        }
```

ThreadIndexUpdater.java

Μέρος της μεθόδου run()

```
/*
*Update the fields 'users_tag_num', 'users_comments', *'users_comments_stemmed', 'user_tags' of the index
*/

        public void run() {

                //get all docids for all the documents of the index that is about to be updated
                Set<String> c = docsComments.keySet();

                Iterator<String> itr =c.iterator();
                while(itr.hasNext()){
                        String key = itr.next();

                        try {

                                Document doc = getDocument(key);
                                int users=0;
                                if(doc.getField("users_tag_num")!=null){
                                        users =
Integer.parseInt(doc.getField("users_tag_num").stringValue());
                                        if(users<=1000){
                                                doc.removeFields("users_tag_num");
                                                doc.add(new
Field("users_tag_num",String.valueOf(++users),Store.YES, Index.UN_TOKENIZED));
                                        }
                                }else
                                {
                                        doc.add(new
Field("users_tag_num",String.valueOf(++users),Store.YES, Index.UN_TOKENIZED));
                                }


                                //Update the field users_comments
                                Field tag_field = doc.getField("users_comments");
                                Field tag_field_to_stemmer = doc.getField("users_comments");
                                doc.removeFields("users_comments");
```

```
//System.out.println(this.docsComments.get(key));

Field addfield =null;
if(tag_field!=null){
	addfield = new Field("users_comments", minerSoftanalyzer
		.getTokenizedString("users_comments", new
StringReader(tag_field.stringValue()+this.docsComments.get(key))),
		Store.YES, Index.TOKENIZED);
}else{
	addfield = new Field("users_comments", minerSoftanalyzer
		.getTokenizedString("users_comments", new
StringReader(this.docsComments.get(key))),
		Store.YES, Index.TOKENIZED);
}

addfield.setBoost(getWeight(users));
doc.add(addfield);

doc.removeFields("users_comments_stemmed");
Field addfield_stemmed =null;
if(tag_field!=null){
	addfield_stemmed = new Field("users_comments_stemmed",
porteranalyzer

	.getTokenizedString("users_comments_stemmed", new
StringReader(tag_field_to_stemmer.stringValue()+this.docsComments.get(key))),
		Store.YES, Index.TOKENIZED);
}else{
	addfield_stemmed = new Field("users_comments_stemmed",
porteranalyzer

	.getTokenizedString("users_comments_stemmed", new StringReader(this.docsComments.get(key))),
		Store.YES, Index.TOKENIZED);
}

addfield_stemmed.setBoost(getWeight(users));
doc.add(addfield_stemmed);

String userTags[]= this.docsComments.get(key).split("<&&>");

//for each tag add a field in which the
for(String utag:userTags){
	doc.add(new Field("user_tags", utag, Store.YES,
Index.TOKENIZED, Field.TermVector.YES));

}
if(doc.getField("tagged")!=null)
	doc.add(new Field("tagged", "true", Store.YES,
Index.UN_TOKENIZED));


this.indexwriter.updateDocument(new
Term("DocId",doc.getField("DocId").stringValue()), doc);
```

Παράρτημα Β

Παράρτημα Β: Σημαντικότερα κομμάτια κώδικα από κλάσεις που εμπλέκονται στην διαδικασία δημιουργίας εκπαιδευτικών δεδομένων

ClassifierUtil.java

Μέρος της μεθόδου generateTrainingData ()

```
/*
*Based on the updated tags generate the training date
*/
        public void generateTrainingData(boolean createSoftwareIndex ) throws CorruptIndexException,
IOException, ParseException, InterruptedException{

                InstanceIndexer inin = new InstanceIndexer();

                String indexes = getIndexesPath(this.indexFolder);
                if(createSoftwareIndex==false)
                        inin.buildIndex(indexes,setThisClassPath("software_indexes_only"));

                //built arff format without categories
                inin.exportToArff(setThisClassPath("Only_Attributes"),
inin.buildArffFormat(setThisClassPath("software_indexes_only")));

                //find all categories from all tags from the indexes
                CategoryFinder cf = new CategoryFinder();
                this.catNum=cf.getCategories().size();
                ArrayList<IndexDirectory> indexDirs = cf.getTaggedDocuments(indexes);

                //add categories to instance format
                this.instanesFormat = inin.addNewCatToInstancesFormat(cf.getCategories());

                inin.exportToArff(setThisClassPath("Instances_Format"), this.instanesFormat);

                TrainInstanceCreator trainInstance = new
TrainInstanceCreator(setThisClassPath("software_indexes_only"),instanesFormat);

                trainInstance.trainInstances(indexDirs);
                this.trainingInstanes = trainInstance.getTrainInstances();


                inin.exportToArff(setThisClassPath("Training_Data"),trainInstance.getTrainInstances());


                XMLClassManipulator xml =new
XMLClassManipulator("http://mulan.sourceforge.net/labels",  "labels");
                xml.addNewClassLabel(cf.getCategories());
                this.xmlPath = xml.exportXMLClassfile("Labels_Classes");



        }
```

Μέρος της μεθόδου performClassification ()

```
public void performClassification(){

                try {
```

```
BufferedReader reader = new BufferedReader(
                        new FileReader(setThisClassPath("Training_Data.arff")));
this.trainingInstanes = new Instances(reader);
reader.close();
BufferedReader reader2 = new BufferedReader(
                        new FileReader(setThisClassPath("Instances_Format.arff")));
this.instanesFormat = new Instances(reader2);
reader.close();

int numClass=0;
for(int a=this.instanesFormat.numAttributes()-1; a>=0; a--){
        if(!this.instanesFormat.attribute(a).isNumeric())
                numClass++;
        else
                break;
}

this.xmlPath =  setThisClassPath("Labels_Classes.xml");

ClassificationClass n = new
ClassificationClass(numClass,setThisClassPath("software_indexes_only"),getIndexesPath(this.indexFolder),thi
s.instanesFormat);
n.startClassifier();
n.trainClassifier(this.trainingInstanes,  this.xmlPath);
n.updateTagCategories();
```

InstanceIndexer.java

Μέρος της μεθόδου buildIndex()

```
/*
* By getting the directory with all indexes generates a new index
* that contains the software documents of all indexes and their terms along
* with their tf-id which can be easily calculated.
*/
        public void buildIndex(String inputDirectory, String outputDirectory) throws CorruptIndexException,
IOException{
                File folder = new File(inputDirectory);
                IndexReader indexreader=null;
                IndexWriter indexWriter = new IndexWriter(outputDirectory, new StandardAnalyzer(),true);

                System.out.println("Start loading indexes ...");
                for(int j=0; j<folder.list().length;j++){
                        System.out.println("Loading index "+folder.list()[j]+" ...");
                        //for every index
                        indexreader= IndexReader.open(FSDirectory.getDirectory(inputDirectory +"/"+
folder.list()[j]));

                        for(int i=0; i<indexreader.maxDoc(); i++){
                                Document doc = indexreader.document(i);


                                //include only software documents in the new index
                                if(doc.getField("fulltext")!=null
&&(doc.getField("isBin").stringValue().equals("true") ||
doc.getField("isLib").stringValue().equals("true")||doc.getField("isSource").stringValue().equals("true")||doc.get
Field("isScript").stringValue().equals("true"))){

                                        Document newDoc = new Document();
```

```
                                        newDoc.add(new Field("IndexId",
folder.list()[j],Store.YES,Index.UN_TOKENIZED));
                                        newDoc.add(new Field("DocId",
doc.getField("DocId").stringValue(),Store.YES,Index.UN_TOKENIZED));


                                        String attr=doc.getField("fulltext").stringValue();

                                        if(doc.getFields("readme")!=null)
                                        for(Field field:doc.getFields("readme")){
                                                attr+=" "+field.stringValue();
                                        }

                                        if(doc.getFields("man")!=null)
                                        for(Field field:doc.getFields("man")){
                                                attr+=" "+field.stringValue();
                                        }

                                        newDoc.add(new Field("attribute",attr
,Store.YES,Index.TOKENIZED,Field.TermVector.YES));

                                        indexWriter.addDocument(newDoc);
                                }
                        }
                        indexreader.close();

                }


                System.out.println("OPTIMIZING NEW INDEX!");
                indexWriter.optimize();
                indexWriter.close();

                System.out.println("FINISH  NEW INDEX!");
                System.out.println("============================");

        }
```

Μέρος της μεθόδου buildArffFormat ()

```
/*
 * After generating the new index this function is responsible
 * to create the default format of 'instances' that the classifier
 * will be using.
 */
        public Instances buildArffFormat(String indexDirectory) throws CorruptIndexException,
IOException{
                IndexReader indexreader=null;
                indexreader= IndexReader.open(FSDirectory.getDirectory(indexDirectory));
                TermEnum termEnumeration = indexreader.terms();

                ArrayList<SortTerm> sortTerms = new ArrayList<SortTerm>();

                while(termEnumeration.next()){
                        if(termEnumeration.term().field() != "attribute" ){
                                continue;
                        }
                        sortTerms.add(new SortTerm(
termEnumeration.term().text(),termEnumeration.docFreq()));
```

```
		}

		Collections.sort(sortTerms);
		ArrayList <String> uniqueTerms = new ArrayList <String>();

		//add only the top df documents
		for(int i=0; i<sortTerms.size() && i<200; i++){
			uniqueTerms.add(sortTerms.get(i).text);
		}

		Collections.sort(uniqueTerms);

		FastVector allAttributes = new FastVector();

		for(String term:uniqueTerms) {
				allAttributes.addElement(new Attribute(term));
		}
		Instances isTrainingSet = new Instances("Training Data", allAttributes,0);

		this.defaultInstancesFormat = isTrainingSet;
		return isTrainingSet;
	}
```

IndexClassifier.java

Μέρος της μεθόδου classifyDocuments()

```
/*
 * Predicts document categories and ranks them in a way that the most relative appear first.
 * Then adds the top K predicted categories in the index and specifically in the document where
 * they belong.
 */
	public void classifyDocuments() throws CorruptIndexException, IOException, ParseException{

		for(int i=0; i<this.indexreader.numDocs(); i++){
			Document doc = this.indexreader.document(i);


				if(doc!=null && doc.getField("isLib")!=null && doc.getField("isBin")!=null &&
doc.getField("isSource")!=null && doc.getField("DocId")!=null)
					if(doc.getField("isLib").stringValue().equals("true") ||
doc.getField("isBin").stringValue().equals("true") || doc.getField("isSource").stringValue().equals("true")){

						Integer id
=getDocumentNumber(doc.getField("DocId").stringValue());

						if(id!=-1){
							TermFreqVector termFVattr
=this.tfidfReader.getTermFreqVector(id, "attribute");


							Instance instance;
							if(termFVattr!=null){
								//convert each document into Mulan instance in
order to be categorized

								instance = new
DenseInstance(instanceFormat.numAttributes());
								String[] fulltext_terms = termFVattr.getTerms()
```

```
                                                    int[] fulltext_freqs =
termFVattr.getTermFrequencies();
                                                    //set all tf-idf weights of all atributes 0

                                                    for(int a=0;
a<instanceFormat.numAttributes();a++)
                                                            instance.setValue(a,0);



                                                    //set all the tf-idf weights for each attribute
                                                    for(int a=0; a<fulltext_terms.length; a++){

        if(instanceFormat.attribute(fulltext_terms[a])!=null)

        instance.setValue(instanceFormat.attribute(fulltext_terms[a]).index(),
fulltext_freqs[a]*Math.log10(this.tfidfReader.numDocs()/this.tfidfReader.docFreq(new
Term("attribute",fulltext_terms[a]))));
                                                    }



                                                    String predictedCategories="";
                                                    String predictedCategoriesForSearch="";
                                                    try {

                                                            //predict the category
                                                            MultiLabelOutput output =
classifier.makePrediction(instance);

                                                            int ranking[] =output.getRanking();

                                                            int topOneThird = ranking.length/3;

                                                            for(int r=0; r<ranking.length &&
r<topOneThird; r++){


        predictedCategories+=instanceFormat.attribute((instanceFormat.numAttributes()-
(this.classNumber+1))+ranking[r]).name()+" & ";

        predictedCategoriesForSearch+=instanceFormat.attribute((instanceFormat.numAttributes()-
(this.classNumber+1))+ranking[r]).name();
                                                            }

                                                            //add the predicted categories to index

        if(doc.getField("predicted_classes")!=null)

        doc.removeFields("predicted_classes");

                                                            //add the predicted categories for search
to index

        if(doc.getField("predicted_search_classes")!=null)

        doc.removeFields("predicted_search_classes");


        if(doc.getField("predicted_search_classes_stemmed")!=null)

        doc.removeFields("predicted_search_classes_stemmed");
```

```
                                                                    doc.add(new
Field("predicted_classes",predictedCategories,Store.YES,Index.UN_TOKENIZED));

                                                                    doc.add(new
Field("predicted_search_classes",analyzer.getTokenizedString("null",new
StringReader(predictedCategories)),Store.YES,Index.TOKENIZED));

                                                                    doc.add(new
Field("predicted_search_classes_stemmed",stemAnalyzer.getTokenizedString("null",new
StringReader(predictedCategories)),Store.YES,Index.TOKENIZED));
```

Παράρτημα Γ

Παράρτημα Γ: Σημαντικότερα κομμάτια κώδικα από κλάσεις που εμπλέκονται στην διαδικασία της αναζήτησης αρχείων λογισμικού

MinerSoftSearcher.java

Μέρος της μεθόδου getAnswer()

```
//form query, parse query
                    if(constraints.equals(" +() ")){
                              queries   = parser.parse(formQuery(query,tag_words, new
String[]{"fulltext","readme", "man", "common_filename","users_comments" }, true, stemming ) +" +(isBin OR
isLib OR isSource) " +version_string);
                    }else{

                              queries   = parser.parse(formQuery(query,tag_words, new
String[]{"fulltext","readme", "man", "common_filename","users_comments"  }, true, stemming ) + constraints
+version_string);
                    }

                    System.out.println("Searching for: \"" + query + "\"");

                    Calendar cal = Calendar.getInstance();
                    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");


                    String timeKeyword= "time: "+ sdf.format(cal.getTime())+"  keyword(s): "+query+"\n";
                    logfile_out.append(timeKeyword);
                    logfile_out.close();
                    double time=0.0;
                    //search for the query
                    time = System.currentTimeMillis();
                    Hits hits= searcher.search(queries);
                    time = System.currentTimeMillis()-time;
                    time = time /(double)1000;

                    System.out.println("("+time +") secs");

                    time = System.currentTimeMillis();

                    ArrayList<Description> descriptions = new ArrayList<Description>();
                    HashMap<String,String> toUpdateIndexes = new HashMap<String,String>();
                    HashMap<String,String> predictions = new HashMap<String,String>();


                    HashMap<String, Doc> resultset = new HashMap<String,Doc>();

                    Iterator<Hit> it = (Iterator<Hit>)hits.iterator();


                    //to generate the description snippet under results
                    SnippetGenerator snippetGen = new SnippetGenerator();
                    snippetGen.highLightQuery(qHighlight.toString());


                    while(it.hasNext()){

                              Hit hit = it.next();
                              Doc doc = new Doc(hit.getDocument(),hit.getScore());
```

Παράρτημα Δ

Παράρτημα Δ: Σημαντικότερα κομμάτια κώδικα από κλάσεις που εμπλέκονται στην διαδικασία εμπλουτισμού αποτελεσμάτων από το διαδίκτυο

YahooThreadSearch.java

Μέθοδος SearchQuery()

```
/*
 * Used for each one of the 10 Minersoft results to search
 * in Yahoo using there title and retrieving the most relative,
 * chosen by our selection algorithm.
*/
	public void SearchQuery(){
		StringBuilder builder = new StringBuilder();

			try
			{
				// Convert spaces to +, etc. to make a valid URL
				String enquery = URLEncoder.encode(query, "UTF-8");

				// Give me back 10 results in JSON format
				URL url = new URL("http://boss.yahooapis.com/ysearch/web/v1/" +
enquery +
						"?appid=" + this.api_key +
"&count="+topK+"&format=json&abstract=long");
				URLConnection connection = url.openConnection();

				String line;

				BufferedReader reader = new BufferedReader(
						new InputStreamReader(connection.getInputStream()));
				while((line = reader.readLine()) != null) {

					builder.append(line);
				}

				String response = builder.toString();

				JSONObject json = new JSONObject(response);

				JSONArray ja = json.getJSONObject("ysearchresponse")
				.getJSONArray("resultset_web");

				for (int i = 0; i < ja.length(); i++) {

					JSONObject j = ja.getJSONObject(i);

					//choose weight for each of the four rules
					float weights[] = {0.3f,0.3f,0.3f,0.1f};

					this.TopKResults.add(new YahooResult(this.topK,weights,i,
j.getString("title"), j.getString("url"), j.getString("abstract"),this.query));

				}

			}
			catch (Exception e) {
				//in case of connection fails nothing can be done
```

```
                    System.err.println("Something went wrong...");

                }
            //sort results in descending order, in respect to their score
            if(TopKResults.size()!=0)
                Collections.sort(TopKResults);

    }
```

YahooResult.java

Μέθοδος findTitleQuerySimilarityScore()

```
/*
* Use Levenshtein's distance algorithm to find the most similar word
* in the title of a Yahoo result, then divide it by its length to
* make a similarity percentage, then subtract it from the unit to get
* each word's score. Finally return the highest score multiplied with the
 * weight given to the rule.
 */
    public float findTitleQuerySimilarityScore(float w){

        String title = this.title;
        title = title.replaceAll("<b>", "");
        title = title.replaceAll("</b>", "");
        title = title.toLowerCase();
        String words[] = title.split(" ");

        float highestScore=0;

        for(String word:words){
            int similarity = StringUtils.getLevenshteinDistance(query,word);
            float persentage = (float)similarity/query.length();
            if(persentage>1)persentage=1;

            if((1-persentage)>=highestScore)
                highestScore = 1-persentage;

        }
        return w*(highestScore);
    }
```

Μέθοδος findQueryFreqScore ()

```
/*
* Find the description's size in words, then find how many
* occurrences of the Minersoft title exist in Yahoo result's
* description. Finally make the division (occurrences/words length)
* and multiply with the weight given to this rule.
 */
    public float findQueryFreqScore(float w){

        String context = this.context;
        context = context.replaceAll("<b>", "");
        context = context.replaceAll("</b>", "");
        context = context.toLowerCase();

        int words= context.split(" ").length;
```

```
		Pattern p = Pattern.compile(query.toLowerCase());
		Matcher m = p.matcher(context);
		int occurrences= 0;

		while(m.find())
			occurrences++;

		return (w *((float)occurrences/words));
	}
```

Μέθοδος findURIScore ()

```
/*
* If the title of the Minersoft's result is not in the authoritative
* part of the URI but exist in the path returns half of the rule's given
* weight. If exist in the authoritative part of the URI then returns
* full score. If the URI doesn't contain Minersoft result's title then
* returns 0.
*/
	public float findURIScore(float w){

		float weight=0;

		if(this.URL.toLowerCase().contains(query))
			weight  = (float)w*0.5f;

		String firstHalhf  =this.URL.toLowerCase().replace("http://", "");
		firstHalhf =firstHalhf.substring(0, firstHalhf.indexOf("/"));

		if(firstHalhf.contains(query))
			weight =w;

		return weight;
	}
```

Μέθοδος findRankingScore ()

```
/*
* According to Yahoo ranking, multiply the rank of each Yahoo result
* with the weight for each result. Then subtract it from the unit to
* reverse it and find the weight that the result should get according
* its rank and finally multiply it to the weight given to rule.
*/
	public float findRankingScore(float w){

		return  (w*(1.0f-((1.0f/(float)this.topK)*(rank))));
	}
```