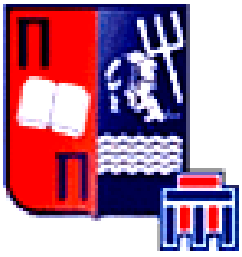


Πανεπιστήμιο Πειραιώς
Τμήμα Ψηφιακών Συστημάτων
Π.Μ.Σ. "Πληροφοριακά Συστήματα & Υπηρεσίες"
Κατεύθυνση : «Μεγάλα Δεδομένα και Αναλυτική»

Τίτλος Διατριβής	**ΤΕΧΝΙΚΕΣ ΑΝΑΠΤΥΞΗΣ ΣΥΣΤΗΜΑΤΩΝ ΣΥΣΤΑΣΕΩΝ ΜΕ ΒΑΣΗ ΠΛΗΡΟΦΟΡΙΑ ΑΠΟ ΚΟΙΝΩΝΙΚΑ ΔΙΚΤΥΑ**
Ονοματεπώνυμο Φοιτητή	**Παπάζογλου Πάνος**
Πατρώνυμο	**Ευάγγελος**
Αριθμός Μητρώου	**ΜΕ1612**
Υπεύθυνος Καθηγητής	**Μαρία Χαλκίδη**

Ημερομηνία Παράδοσης	**Μάρτιος 2019**

Διμελής Εξεταστική Επιτροπή

(υπογραφή)	**(υπογραφή)**
Μαρία Χαλκίδη	**Χρήστος Δουλκερίδης**
Επίκουρη Καθηγήτρια	**Επίκουρος Καθηγητής**

Ευχαριστίες

Η εργασία αυτή εκπονήθηκε στο πλαίσιο της μεταπτυχιακής μου διατριβής του τμήματος Ψηφιακών Συστημάτων με κατεύθυνση «Μεγάλα Δεδομένα και Αναλυτική». Σχετίζεται με το μάθημα "Συστήματα συστάσεων (recommendation systems) " της καθηγήτριας κα Μαρίας Χαλκίδη την οποία ευχαριστώ για την ανάθεση της, καθώς επίσης και για την βοήθεια σχετικά με τις τεχνικές κατευθύνσεις και γνώσεις που μου παρείχε.

Τέλος και περισσότερο απ' όλους θα ήθελα να ευχαριστήσω την οικογένεια μου και κυρίως την κόρη μου Νεφέλη-Μαρία που ήταν ήσυχη και υπομονετική, αν και μόλις τριών ετών, μέχρι να καταφέρω επιτέλους να την ολοκληρώσω επιτυχώς.

Περίληψη. Η παρούσα διπλωματική διατριβή παρουσιάζει, ένα σύστημα συστάσεων (recommendation system) το οποίο εκμεταλλεύεται την πληροφορία που είναι διαθέσιμη στα κοινωνικά δίκτυα. Πρόκειται για πληροφορία που είναι διαθέσιμη σε γνωστά τέτοια συστήματα και η οποία μπορεί να προσδώσει επιπλέον διαστάσεις στην ανάλυση συμπεριφοράς και επιλογών αλλά και αλληλεπίδρασης μεταξύ των χρηστών για την πιο ακριβή προσφορά προτάσεων.
Επιπλέον, γίνεται ανάλυση εκείνων των χαρακτηριστικών που είναι στην διάθεση αυτών των συστημάτων που θα μπορούσαν να έχουν επίδραση στα αποτελέσματα.
Γίνεται ενδελεχής περιγραφή του συστήματος, των πηγών της αρχικής πληροφορίας αλλά και των όποιων migration έγιναν καθώς επίσης και της αξιολόγησης του συστήματος.
Για το τελευταίο δε, εξετάζεται η απόδοση του συστήματος καθώς και γίνεται μια προσπάθεια δειγματοληπτικής βελτιστοποίησης των συντελεστών ενίσχυσης αυτών των χαρακτηριστικών στο μοντέλο, ενώ μέσω της χρήσης του API δίνεται η δυνατότητα δυναμικής αλλαγής των παραμέτρων από εξωτερικό σύστημα που θα μπορούσε βάση των αποτελεσμάτων να συγκλίνει σε βέλτιστες παραμετρικές τιμές. Τέλος παρατηρούνται τάσεις που καταλήγουν σε συμπεράσματα καθώς επίσης και προτάσεις για μελλοντική εργασία όπως η δυνατότητα κατανεμημένων συστημάτων για πιο γρήγορη επεξεργασία πιο αραιών dataset.

Abstract. This diploma thesis presents a recommendation system that exploits the information available on social networks. This kind of information is available in known such systems and which can add extra dimensions to behavioral analysis, proposals and interaction among users for the most accurate suggestions. In addition, an analysis is made of the features available in these systems and which could affect the results. A detailed architectural description of the system, the sources of the initial information as well as any migration and the evaluation of the system were presented.
For the latter, it examines the system performance as well as an attempt to sample optimization of the enhancement factors of these features in the model while the use of the API allows the dynamic change of the parameters from an external system that could be based on the results at optimal parametric values.
Finally, there are trends leading to conclusions and suggestions for future work such as use of distributed system for parallel and faster processing of more sparse data sets.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΚΟΝΕΣ

ΠΙΝΑΚΕΣ

1. ΕΙΣΑΓΩΓΗ

1.1 ΓΕΝΙΚΑ

Η δημιουργία ενός συστήματος συστάσεων (recommendation system) όπως αυτό που υλοποιήθηκε σε αυτή την εργασία, αφορά σε μια πλατφόρμα η οποία εκμεταλλεύεται την πληροφορία που είναι διαθέσιμη στα κοινωνικά δίκτυα. Πρόκειται για τον πυρήνα ενός συστήματος (framework) το οποίο μπορεί να χρησιμοποιηθεί είτε ως απλό σύστημα συστάσεων, είτε ως σύστημα εκπαίδευσης , επαλήθευσης αλλά και ανάλυσης αραιών συνόλων δεδομένων. Επιπλέον το σύστημα διαθέτει την υποδομή για την κατάλληλη επέκταση του (API / hooks), ώστε να είναι δυνατόν να χρησιμοποιηθεί είτε ως βιβλιοθήκη για άλλα συστήματα παρουσίασης (presentation layer), είτε και για συστήματα που θέλουν να κάνουν περαιτέρω ανάλυση χρησιμοποιώντας το ως βάση.

Στην συνέχεια θα περιγραφούν διεξοδικά οι λειτουργικές του δυνατότητες, ενώ θα γίνει αναφορά στις τεχνολογίες που χρησιμοποιήθηκαν. Ενδεικτικά θα αναφέρουμε ότι χρησιμοποιήθηκε για την υλοποίηση του συστήματος η γλώσσα προγραμματισμού Java καθώς επίσης και τεχνολογίες του ευρύτερου οικοσυστήματος της όπως maven, junit. Για λόγους τόσο εκπαιδευτικούς, απλότητας αλλά και για να υπάρξει η δυνατότητα προοδευτικής βελτίωσης, ώστε να είναι κατανοητή η κάθε νέα έκδοση για ποιο λόγο γίνεται, δεν έχουν χρησιμοποιηθεί έτοιμα frameworks όπως για παράδειγμα J2EE τεχνολογίες ή Spring κ.τ.λ. Εν' τούτης έχει στηθεί η υποδομή για διασύνδεση με την βάση τόσο σε απλό επίπεδο jdbc όσο και σε επίπεδο ORM(Object Relational Mapping) με JPA Annotations και ως provider (σύστημα υλοποίησης) το hibernate. Επιπλέον έχει γίνει χρήση τόσο της σχεσιακής βάσης Oracle όσο και της mongo. Η τελευταία χρησιμοποιήθηκε εκτεταμένα για την γρήγορα εισαγωγή των δεδομένων (dataset) αλλά και την περαιτέρω διαχείριση τους μικρότερα υποσύνολα με μεγαλύτερη πυκνότητα δεδομένων.

Τέλος, ιδιαίτερη σημασία έχει δοθεί στην ενσωμάτωση μέσα στο σύστημα διαχωριστή (splitter) και σύστημα επαλήθευσης (evaluator) ώστε να μην απαιτούνται εξωτερικά συστήματα. Αυτό δίνει την δυνατότητα μαζί με την κατάλληλη υποδομή που υλοποιήθηκε να είναι σε θέση το σύστημα να εκτελεί συνδυαστικά τρεξίματα με την ελάχιστη από τον χρήστη διάδραση και την μέγιστη δυνατή αποτύπωση των αποτελεσμάτων για την γρήγορη και εύκολη αποθήκευση τους και την περαιτέρω ανάλυση τους.

1.2 ΣΤΟΧΟΙ ΤΗΣ ΕΡΓΑΣΙΑΣ

Η θεματική ενότητα αφορούσε σε ένα Recomendation System σε συνδυασμό με social network. Ο βασικός στόχος της εργασίας ήταν καταρχάς η δημιουργία ενός απλού Recomendation συστήματος με βασική υλοποίηση Colaborative Filtering (User-User) και βασικό αλγόριθμο ομοιότητας cosine similarity και ελέγχου ποιότητας MSE, ενώ θα υπήρχαν extention points (hooks) και για άλλους αλγορίθμους αλλά και για εναλλαγή σε item-item με μικρό προγραμματιστικό efford (π.χ. αλλαγή των bindings των πινάκων user/item).

Σε ότι αφορά το κομμάτι του social network, η επιλογή του σωστού dataset για την όσο το δυνατόν περισσότερη πληροφορία συσχέτισης μεταξύ χρηστών (userUserRelationship) αλλά και ποιοτικών στοιχείων των ίδιων των items π.χ. κατηγορίες κτλ, προκειμένου να χρησιμοποιηθούν για τους πειραματικούς συνδυασμούς που ακολουθήθηκαν για να καταδείξουν πιθανά trends.

Η βασικότερη λειτουργία που υλοποιείται στα πλαίσια του social network είναι η ενδυνάμωση του αλγορίθμου ομοιότητας ώστε να κοιτά αν υπάρχει διασύνδεση μεταξύ των χρηστών (trust/absent) ώστε να δίνει ένα συντελεστή πάνω στον βασικό weighted average αλγόριθμο που υπολογίζει την μέση σταθμισμένη ομοιότητα για να υπολογιστεί αντίστοιχα και το rating για κάθε ένα item του συγκεκριμένου χρήστη στον Utility Matrix βάση των N χρηστών και T similarity Threshold (σε περίπτωση που τα δύο τελευταία είναι ενεργά).

Περεταίρω δοκιμές, λόγω της φύσης του συνόλου δεδομένων που δημιουργήθηκε, έγιναν με χρήση συγκεκριμένων χαρακτηριστικών των χρηστών και την μελέτη των αποτελεσμάτων σε σχέση με το κατά πόσο η χρήση τους είχε θετική ή αρνητική επίπτωση στην απόδοση του συστήματος.

2. ΕΙΣΑΓΩΓΗ ΣΤΑ ΣΥΣΤΗΜΑΤΑ ΣΥΣΤΑΣΕΩΝ

2.1 ΓΕΝΙΚΑ

Όπως ήδη αναφερθήκαμε και στο πρώτο κεφάλαιο της εργασία, ο σκοπός της εργασίας είναι η μελέτη και δημιουργία ενός συστήματος συστάσεων με χρήση δεδομένων από κοινωνικά δίκτυα. Πριν όμως από αυτό θα πρέπει πρώτα να αποσαφηνίσουμε ορισμένες έννοιες και προσεγγίσεις που σχετίζονται με το παρόν θέμα. Έτσι παρακάτω θα αναλυθούν συνοπτικά τα είδη των συστημάτων συστάσεων, έννοιες/μετρικές όπως αυτές της ομοιότητας και της απόδοσης ενός συστήματος, ενώ θα επεξηγήσουμε την ιδιαίτερη περίπτωση των συστημάτων συστάσεων με χρήση δεδομένων από κοινωνικά δίκτυα (Social Network-Based Recommendation).

Επιπλέον, θα αναφερθούμε σε διάφορες προσεγγίσεις και υλοποιήσεις που έχουν γίνει σε ότι αφορά το θέμα που μας ενδιαφέρει (Social Network-Based Recommendation) αλλά και γενικότερα για Recommendation Systems τόσο από την βιβλιογραφία όσο και από ευρύτερες πηγές.

2.1.1 Ανάγκη που εξυπηρετούν

Τα συστήματα συστάσεων, έχουν ως σκοπό τους να προσφέρουν συστάσεις σε χρήστες για διάφορα προϊόντα/υπηρεσίες(αντικείμενα) βάση των αξιολογήσεων τόσο του εκάστοτε χρήστη όσο και των άλλων χρηστών. Με αυτό τον τρόπο επιτυγχάνεται η καλύτερη δυνατή σύσταση που ταιριάζει με τον χρήστη προκειμένου αφενός να επιτευχθεί πιο εύκολα και σίγουρα μια αγορά, αφετέρου ο χρήστης να μείνει ευχαριστημένος από την αγορά του προϊόντος ή της υπηρεσίας που του προτάθηκε ώστε να επανέλθει και να είναι πιο καλή η εμπειρία αγοράς αλλά και η εμπιστοσύνη στο σύστημα συστάσεων.

Σε περίπτωση όμως που κάποια αγορά δεν είναι καλή και επικοινωνηθεί από τον χρήστη μέσω αρνητικής αξιολόγησης, είναι και αυτό σημαντικό καθώς το σύστημα θα προχωρήσει σε διορθωτικές κινήσεις μέσω αναβάθμισης του προφίλ του χρήστη αλλά και του συσχετισμού του με νέα άτομα ή γκρουπ ατόμων.

Τελικός στόχος είναι η αύξηση των πωλήσεων με την ταυτόχρονη καλυτέρευση της εμπειρίας και της πραγματικής ικανοποίηση του πελάτη.

2.1.2 Εφαρμογές

Τα συστήματα συστάσεων συναντώνται ως επί το πλείστον σε online καταστήματα προϊόντων αλλά αυτό δεν είναι η μόνη περίπτωση. Άλλη χρήση τους γίνεται από συστήματα που προσφέρουν υπηρεσίες όπως για παράδειγμα αυτό της Netflix, που είναι πλατφόρμα προτάσεων για παρακολούθηση ταινιών ή σειρών με μηνιαία χρέωση.

Σε επίπεδο φυσικών καταστημάτων, κυρίως μεγάλων πολυκαταστημάτων αλλά όχι μόνο, εμφανίζονται προτάσεις για χρήση έξυπνων συσκευών αγοράς ή ακόμη και υποδομής ελέγχου καλαθιού (sensitive basket) μέσω του οποίου ανάλογα με τις αγορές θα προτείνονται νέα προϊόντα ή ακόμη και συνταγές αν πρόκειται για τρόφιμα. Ένα παράδειγμα μπορεί να βρεθεί στο παρακάτω paper [5] Grocery shopping recommendations based on basket-sensitive random walk.

Σημαντική εφαρμογή φαίνεται να γίνεται και στο επίπεδο της ιατρικής και φαρμακευτικής. Στο παρακάτω paper [6] Health Recommender Systems: Concepts, Requirements, Technical Basics and Challenges

περιγράφεται το ότι λόγω των μεγάλων δεδομένων σε επίπεδο ιατρικού ιστορικού, συνταγογραφήσεων και αποτελεσμάτων ένα σύστημα συστάσεων θα μπορούσε να λειτουργεί υποστηρικτικά προς όφελος του ασθενή. εφαρμογές φαίνεται να έχει και σε επίπεδο.

2.2 ΓΕΝΙΚΕΣ ΕΝΝΟΙΕΣ ΚΑΙ ΜΕΤΡΙΚΕΣ

Κατά την ενασχόληση κάποιου με τα συστήματα συστάσεων, απαιτείται να έχει ξεκάθαρες τουλάχιστον δύο βασικές έννοιες. Οι έννοιες αυτές είναι η ομοιότητα και η απόδοση.

2.2.1 Ομοιότητα

Με τον όρο ομοιότητα θεωρείται το πόσο ίδιο με μαθηματικό τρόπο είναι μια οντότητα με μια άλλη. Έτσι όπως θα δούμε και παρακάτω μπορεί να ψάχνουμε να βρούμε την ομοιότητα ανάμεσα σε δύο χρήστες που έχουν προβεί σε αγορές ή αντίστοιχα στο πόσο ίδια είναι δύο αντικείμενα που έχουν αγοραστεί από κάποιους χρήστες. Ο τρόπος με τον οποίο καθορίζεται η ομοιότητα μπορεί να ποικίλει ανάλογα με την περίπτωση και την φύση των δεδομένων μας. Έτσι αν για παράδειγμα έχουμε χαρακτηριστικά που είναι δυαδικά μπορεί να συμφέρει η ομοιότητα Pearson.

$$similarity(A,B)=\frac{\sum_{i=1}^{N}(A_i-\mu o(A))*(B_i-\mu o(B))}{\sqrt{\sum_{i=1}^{N}(A_i-\mu o(A))^2}*\sqrt{\sum_{i=1}^{N}(B_i-\mu o(B))^2}}$$

Εικόνα 1 - Τύπος της μετρικής για «Ομοιότητα Pearson»

Από την άλλη αν τα χαρακτηριστικά μας έχουν τιμές τότε θα μπορούσαμε να χρησιμοποιήσουμε ομοιότητα συνημίτονου (cosine similarity)

$$similarity(A,B)=\cos(\theta)=\frac{\vec{A}*\vec{B}}{|\vec{A}||\vec{B}|}=\frac{\sum_{i=1}^{N}A_i*B_i}{\sqrt{\sum_{i=1}^{N}A_i^2}*\sqrt{\sum_{i=1}^{N}B_i^2}}$$

Εικόνα 2 - Τύπος της μετρικής για «Ομοιότητα Συνημίτονου»

Βεβαίως, εκτός από αυτούς τους δύο αλγόριθμους ομοιότητας υπάρχουν κι' άλλοι όπως Tanimoto, Dice αλλά και ο «μέσος όρος» που προκύπτει από την μέση τιμή της ομοιότητας των τεσσάρων προαναφερθέντων αλγορίθμων.

2.2.2 Απόδοση

Για την κατανόηση της έννοιας απόδοση ενός συστήματος συστάσεων θα πρέπει να αναφερθούμε στα σύνολα δεδομένων (dataset), το σύνολο εκπαίδευσης (training set) και τέλος στο σύνολο δοκιμής (test set). Έτσι ένα σύστημα συστάσεων κόβει ένα σύνολο δεδομένων σε κάποιο ποσοστό π.χ. 10% και δεν χρησιμοποιεί τα στοιχεία αυτά στην εκπαίδευση του. Στην συνέχεια το σύστημα εκπαιδεύεται με το υπόλοιπο ποσοστό (training set) και όταν ολοκληρωθεί η εκπαίδευση του θα πρέπει να ελεγχθεί η απόδοση του μέσω κατάλληλου μηχανισμού (evaluation). Αυτό γίνεται με τον υπολογισμό των τιμών που έχουμε ξεχωρίσει στο σύνολο δοκιμών (test set) και τον υπολογισμό για παράδειγμα των αξιολογήσεων σε κάποια προϊόντα από κάποιους χρήστες και στην συνέχεια στην αντιπαραβολή τους με τις πραγματικές αξιολογήσεις που διαθέτουμε σε αυτό το σύνολο δοκιμών (test set). Η απόδοση του συστήματος συνήθως αποδίδεται με το μέσο τετραγωνικό σφάλμα (MSE).

$$\text{ΜΤΣ}(T(\underset{\sim}{X}), \theta) = \mathbb{E}_\theta\Big(T(\underset{\sim}{X}) - g(\theta)\Big)^2$$

Εικόνα 3 - Τύπος της μετρικής «Μέσο Τετραγωνικό Σφάλμα (MSE)»

Είναι προφανές ότι όσο πιο χαμηλός είναι αυτός ο αριθμός (τείνει στο 0) τόσο πιο αξιόπιστο είναι το σύστημα μας.

2.3 ΕΙΔΗ ΣΥΣΤΗΜΑΤΩΝ ΣΥΣΤΑΣΕΩΝ

Υπάρχουν 3 βασικοί τύποι συστάσεων καθώς και διαφορετικές προσεγγίσεις :

- Βασισμένο στην μνήμη (Memory Based)
- Βασισμένα σε μοντέλο (Model-based)
- Υβριδικά συστήματα συστάσεων (Hybrid Recommendation Systems)

2.3.1 Βασισμένο στην μνήμη

Η προσέγγιση που βασίζεται στη μνήμη χρησιμοποιεί δεδομένα αξιολόγησης χρήστη για να υπολογίσει την ομοιότητα μεταξύ χρηστών ή αντικειμένων. Τυπικά παραδείγματα αυτής της προσέγγισης είναι οι βασισμένες στη γειτονιά CF και βασισμένες σε στοιχείο / βασισμένες σε χρήστες συστάσεις top-N. Για παράδειγμα, στις προσεγγίσεις που βασίζονται σε χρήστες, η τιμή του χρήστη που δίνει το στοιχείο u στο στοιχείο i υπολογίζεται ως συνάθροιση βαθμολογίας κάποιου παρόμοιου χρήστη του στοιχείου:

Στα “memory based” υπάρχουν δύο ειδών τα

- Επιλογή βάση περιεχομένου (Content-based Filtering)
- Φιλτράρισμα μέσω Συνεργασίας (Collaborative Filtering , User-User / Item-Item)

2.3.1.1 Επιλογή βάση περιεχομένου (Content-based Filtering)

Το πρώτο πρόβλημα που προκύπτει σε τέτοια συστήματα είναι αυτό που αποκαλείται “Cold Start Problem”. Για να μπορέσει να ξεκινήσει η διαδικασία παραγωγής συστάσεως σε ένα νέο σύστημα, και όχι σε μια εργασία, απαιτείται μια κρίσιμη μάζα πληροφορίας. Ενώ σε μια εργασία θα μπορούσε κάποιος απλά να βασιστεί σε ένα dataset, σε ένα πραγματικό νέο σύστημα με άγνωστα νέα προϊόντα / υπηρεσίες και νέους χρήστες θα απαιτείτο να περάσει κάποιος καιρός προκειμένου να γίνουν στα τυφλά κάποιες αξιολογήσεις μετά τις όποιες αγορές. Ο χρόνος αυτός θα ήταν ακόμη μεγαλύτερος από όσο αρχικά κάποιος θα υπολόγιζε αν σκεφτεί ότι όσοι κάνουν αγορές δεν συνεπάγεται ότι θα κάνουν και αξιολογήσεις.

Μια πρώτη προσέγγιση για αυτό το πρόβλημα είναι εκείνη των user/item profiles όπως περιγράφεται στο βιβλίο [2] Mining of Massive Datasets (κεφάλαιο 9). Βάση αυτής, τα αντικείμενα(προϊόντα / υπηρεσίες) αναλύονται στα ποιοτικά τους χαρακτηριστικά και

για κάθε αντικείμενο δημιουργείται το προφίλ του βάση των παραπάνω χαρακτηριστικών. Για παράδειγμα ένα φαγητό θα μπορούσε να αναλυθεί στις παραμέτρους γλυκό, ξινό, πικρό, αλμυρό , είδος (κρέας, ψάρι κτλ) και κάθε φαγητό να πάρει κάποιες τιμές βάση αυτών των χαρακτηριστικών. Οι τιμές αυτές θα μπορούσε να ήταν είναι δυαδικές (boolean) , είτε σταθμισμένες σε κάποια κλίμακα.

Μετά την δημιουργία των item profiles θα μπορούσε αντίστοιχα να δημιουργηθούν και τα user profiles. Έτσι θα μπορούσαμε για τα αντίστοιχα χαρακτηριστικά να ερωτηθούν οι χρήστες για το αν για παράδειγμα προτιμούν φαγητά αλμυρά , γλυκά , ξινά , καθώς επίσης και το είδος (κρέας, ψάρι κτλ) καθώς επίσης και το επίπεδο προτιμήσεις τους για τα παραπάνω.

Με αυτό τον τρόπο θα μπορούσε αρχικά το σύστημα ζητώντας από τους νέους χρήστες να επιλέξουν τα επίπεδα προτίμησης τους βάση των εκάστοτε ποιοτικών χαρακτηριστικών, να είναι σε θέση να τους προτείνει αντικείμενα(προϊόντα / υπηρεσίες) που λόγω των χαρακτηριστικών τους φαίνεται να ταιριάζουν στις προτιμήσεις τους.

Μια εναλλακτική προσέγγιση για το θέμα, και την οποία ακολουθεί η Netflix στην εφαρμογή της, είναι στην αρχή να ζητείται από τους χρήστες να επιλέξουν κάποιες ταινίες από διάφορα είδη ταινιών. Με αυτό τον τρόπο είναι σε θέση να βλέπει τις προτιμήσεις των χρηστών βάση των τιμών των χαρακτηριστικών που έχουν αυτές οι ταινίες χωρίς να τους εξαναγκάζει να δηλώσουν ότι προτιμούν α% δράση , β% αισθηματική , προτιμητέους ηθοποιούς ή σκηνοθέτες κτλ κάτι που θα ήταν αποτρεπτικό σε κάποιον που θέλει να μπει σε μια νέα πλατφόρμα/εφαρμογή και θέλει άμεσα να κάνει χρήση της.

2.3.1.2 Φιλτράρισμα μέσω Συνεργασίας (Collaborative Filtering)

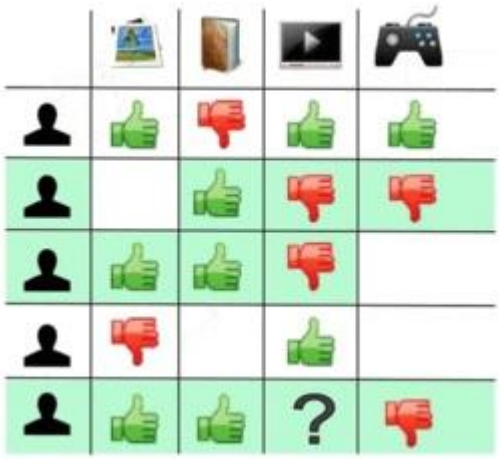
Αφού ξεπεραστεί το πρόβλημα της αρχικής έλλειψης δεδομένων (cold start problem), πλέον το σύστημα μπορεί να λειτουργήσει αλγοριθμικά και με διαφορετικές προσεγγίσεις. Δύο από τις πιο βασικές από αυτές είναι δημιουργώντας προφίλ αντικειμένων ή προφίλ χρηστών και να γίνεται ένα είδος ομαδοποίησης (clustering) ως ομαδοποίηση μεταξύ των όμοιων αντικειμένων ή χρηστών.

Έτσι έχουμε τα :

- Φιλτράρισμα μέσω Συνεργασίας μεταξύ χρηστών (User-User Collaborative Filtering)
- Φιλτράρισμα μέσω Συνεργασίας μεταξύ χρηστών (Item-Item Collaborative Filtering)

2.3.1.2.1 Μεταξύ Χρηστών (User-User Collaborative Filtering)

Στην περίπτωση του φιλτραρίσματος μέσω συνεργασίας χρήστη-χρήστη (user based collaborative filtering) το σύστημα βρίσκει για κάποιο χρήστη τα αντικείμενα που έχει αξιολογήσει. Στην συνέχεια για κάθε ένα από αυτά τα αντικείμενα βρίσκει ποιοι άλλοι χρήστες έχουν αξιολογήσει τουλάχιστον ένα από αυτά. Για κάθε έναν από αυτούς τους χρήστες το σύστημα υπολογίζει την ομοιότητα του αρχικού χρήστη με τον όμοιο χρήστη με κάποιο αλγόριθμό ομοιότητας π.χ. Pearson, Cosine Similarity κτλ. Στην συνέχεια και αφού έχει συγκρατήσει τις ομοιότητες τότε για κάθε αντικείμενο που δεν έχει αξιολογήσει ο χρήστης θα ελέγχει αν υπάρχει κάποιος από τους όμοιους του που το έχουν αξιολογήσει. Αν βρεθεί τότε θα υπολογίσει την αξιολόγηση του σταθμίζοντας το άθροισμα των αξιολογήσεων των όμοιων χρηστών που το έχουν αξιολογήσει.



Εικόνα 4 - User Based Collaborative Filtering

2.3.1.2.2 Μεταξύ Αντικειμένων (Item-Item Collaborative Filtering)

Στην περίπτωση του φιλτραρίσματος μέσω συνεργασίας αντικειμένου-αντικειμένου (item based collaborative filtering) το σύστημα όπως είναι προφανές βασίζεται στα αντικείμενα. Η λογική που το διέπει είναι «χρήστες που αγόρασαν το α προϊόν επίσης αγόρασαν το β. Ο αλγόριθμός στο πρώτο βήμα βρίσκει και διατηρεί σε κατάλληλη δομή δεδομένων την ομοιότητα μεταξύ όλων των ζευγών των αντικειμένων. Όπως και στην προηγούμενη περίπτωση η συνάρτηση της ομοιότητας μπορεί να έχει διαφορετικές μορφές. Στην συνέχεια το σύστημα προτείνει τα πιο όμοια αντικείμενα βασισμένο στις αξιολογήσεις των αντικείμενων που ένας χρήστης έχει ήδη βαθμολογήσει. Όπως προηγούμενος και σε αυτή την περίπτωση ο υπολογισμός αυτός είναι ένα σταθμισμένο άθροισμα ή μια γραμμική παλινδρόμηση.

2.3.2 Συστήματα Συστάσεων βασισμένα σε μοντέλα (Model Based RS)

Σε αυτήν την προσέγγιση, τα μοντέλα αναπτύσσονται χρησιμοποιώντας διαφορετικούς αλγόριθμους εξόρυξης δεδομένων, μηχανικούς μάθησης για την πρόβλεψη της βαθμολόγησης των χρηστών των μη διαβαθμισμένων αντικειμένων. Υπάρχουν πολλοί αλγόριθμοι CF που βασίζονται σε μοντέλα. Bayesian δίκτυα, μοντέλα συσσωμάτωσης, λανθάνοντα σημασιολογικά μοντέλα όπως αποσύνθεση μοναδικής τιμής, πιθανοτική λανθάνουσα σημασιολογική ανάλυση, πολλαπλό πολλαπλασιαστικό παράγοντα, λανθάνουσα κατανομή Dirichlet και μοντέλα που βασίζονται στη διαδικασία λήψης αποφάσεων Markov.

Μέσω αυτής της προσέγγισης, οι μέθοδοι μείωσης των διαστάσεων χρησιμοποιούνται κυρίως ως συμπληρωματική τεχνική για τη βελτίωση της ευρωστίας και της ακρίβειας της προσέγγισης που βασίζεται στη μνήμη. Με αυτή την έννοια, οι μέθοδοι όπως η αποσύνθεση μοναδιαίας τιμής, η ανάλυση βασικών συστατικών, γνωστών ως μοντέλων λανθάνων παραγόντων, συμπιέζουν τη μήτρα χρήστη-στοιχείου σε μια χαμηλής διάστασης αναπαράσταση με όρους λανθανόντων παραγόντων. Ένα πλεονέκτημα της χρήσης αυτής της προσέγγισης είναι ότι αντί να έχουμε μια μήτρα μεγάλης διαστάσεως που περιέχει άφθονο αριθμό ελλειπουσών τιμών θα έχουμε να κάνουμε με ένα πολύ μικρότερο πλέγμα στον κατώτερο διαστατό χώρο. Μια μειωμένη παρουσίαση θα μπορούσε να χρησιμοποιηθεί είτε για αλγορίθμους γειτονιάς που βασίζονται σε χρήστες

είτε για αντικείμενα που παρουσιάζονται στην προηγούμενη ενότητα. Υπάρχουν πολλά πλεονεκτήματα με αυτό το παράδειγμα. Αντιμετωπίζει την ακεραιότητα του αρχικού πίνακα καλύτερα από αυτά που βασίζονται στη μνήμη. Επίσης, η σύγκριση της ομοιότητας με τον προκύπτοντα πίνακα είναι πολύ πιο επεκτάσιμη, ιδίως όσον αφορά την αντιμετώπιση μεγάλων αραιών συνόλων δεδομένων.

2.3.3 Υβριδικά Συστήματα Συστάσεων (Hybrid Recommendation Systems)

Τέλος σε ορισμένες εφαρμογές συνδυάζουν τους αλγορίθμους CF που βασίζονται στη μνήμη και τον μοντέλο. Αυτά ξεπερνούν τους περιορισμούς των εγγενών προσεγγίσεων CF και βελτιώνουν την απόδοση πρόβλεψης. Είναι σημαντικό να ξεπεραστούν τα προβλήματα CF, όπως η έλλειψη πληροφοριών και η απώλεια πληροφοριών. Ωστόσο, έχουν αυξημένη πολυπλοκότητα και είναι δαπανηρές στην εφαρμογή τους. Για τους παραπάνω λόγους τα περισσότερα συστήματα πλέον είναι υβριδικά.

2.4 ΚΟΙΝΩΝΙΚΑ ΔΙΚΤΥΑ ΚΑΙ ΣΥΣΤΗΜΑΤΑ ΣΥΣΤΑΣΕΩΝ

Τα κοινωνικά δίκτυα έχουν γίνει πολύ σημαντικά για τη δικτύωση, την επικοινωνία και την κοινή χρήση περιεχομένου. Οι εφαρμογές κοινωνικής δικτύωσης παράγουν ένα τεράστιο όγκο δεδομένων σε καθημερινή βάση και τα κοινωνικά δίκτυα αποτελούν ένα αυξανόμενο πεδίο έρευνας, λόγω της ετερογένειας των δεδομένων και των δομών που διαμορφώνονται σε αυτά καθώς και του μεγέθους και της δυναμικής τους. Όταν αυτός ο πλούτος των δεδομένων αξιοποιείται από τέτοια συστήματά, η συνακόλουθη σύζευξη μπορεί να βοηθήσει περαιτέρω στην καλυτέρευση συστάσεων από τέτοια συστήματα. Έτσι οι πόροι των κοινωνικών μέσων χρησιμοποιούνται στα RS για τη σύσταση περιεχομένων, άρθρων, ειδήσεων, προϊόντων ηλεκτρονικού εμπορίου και χρηστών.

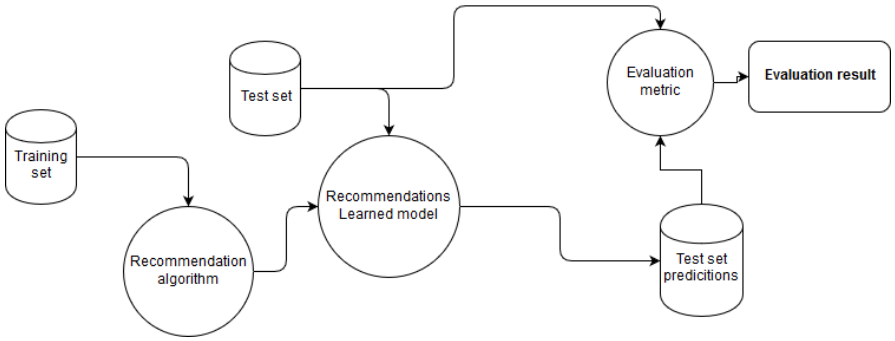
3. ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΟΥ & ΥΛΟΠΟΙΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

3.1 ΕΙΣΑΓΩΓΗ

Ως σημείο εκκίνησης για την ανάλυση του αλγορίθμου, προκειμένου να είναι εφικτή η υλοποίηση του συστήματος συστάσεων, ήταν η μελέτη του υλικού που δόθηκε τόσο από το βιβλίο [2] Mining of Massive Datasets (κεφάλαιο 9), καθώς επίσης και πληθώρα papers ή άλλου υλικού που βρέθηκε στο διαδίκτυο όπως για παράδειγμα του [3] Recommender Systems Based On Social Networks. Στην συνέχεια γίνεται μια συνοπτική περιγραφή του συστήματος καθώς επίσης και η ανάλυση του αλγορίθμου η οποία έχει τρία βασικά βήματα. Σε γενικές γραμμές βασιζόμαστε στην προσέγγιση CF user based κατά την οποία υπολογίζονται, με κατάλληλο αλγόριθμο ομοιότητας, οι βαρύτητες/ομοιότητες για όλους τους χρήστες που σχετίζονται με τον εκάστοτε ενεργό χρήστη. Στην συνέχεια, επιλέγονται οι N πιο ενεργοί, αν είναι ενεργοποιημένη αυτή η δυνατότητα και γίνεται η πρόβλεψη μιας αξιολόγησης βασισμένη στον σταθμισμένο μέσο όρων, με την όποια επαύξηση λόγω φιλίας, των όμοιων χρηστών.

3.2 ΣΥΝΟΠΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ

Στα πλαίσια της διπλωματικής και μετά από μελέτη της θεωρίας σχετικά με τα ήδη και τις προσεγγίσεις των recomendation systems, επιλέγχθηκε το Collaborative Filtering (CF) ως μια προσέγγιση που χρησιμοποιείται συχνά στα πλαίσια ερευνητικών μελετών για συστήματα συστάσεων. Το σύστημα και κυρίως το κομμάτι του αλγορίθμου που τροποποιεί την επιρροή ενός φίλου (πιο όμοιου σε σχέση με κάποιον άλλο), βάση της επαύξησης του συντελεστή ομοιότητας με κάποιο προδηλωμένο ποσοστό, θα μπορούσε κάλλιστα να εφαρμοστεί και στο CF item based απλά αυτή την φορά θα εφαρμοζόταν στο πιο όμοιο αντικείμενο. Αν θέλαμε να παρουσιάσουμε το σύστημα και τα υποσυστήματα του, σε όσο το δυνατόν πιο αφηρημένα αλλά και ολοκληρωμένα, τότε αυτό θα μπορούσε να απεικονιστεί με την παρακάτω εικόνα.



Εικόνα 5- Απεικόνιση Υποσυστημάτων του Συστήματος Συστάσεων

Ότι αφορά την ίδια την εφαρμογή, έχει υλοποιηθεί σε απλή java, ώστε να είναι κατανοητός ο κώδικας χωρίς να εμπλέκονται διάφορα frameworks. Ο κώδικας είναι έτσι γραμμένος με κατάλληλο τρόπο ώστε να βασίζεται σε γενικά archetypes όπως operations , commands, queries κτλ τα οποία κάνουν τον κώδικα να είναι ευκολότερα κατανοητός.

Ως τεχνολογία έχει επιπλέον γίνει χρήση του maven, log4j, ενώ υπάρχουν υποδομές για σύνδεση τόσο σε mongo άλλα και σε σχεσιακές βάσεις μέσω τόσο του hibernate όσο και του jdbc αν αυτό απαιτηθεί. Αν και έγιναν δοκιμές τόσο με επικοινωνία απευθείας με mongo αλλά και με oracle (μέσω hibernate) για το διάβασμα των δεδομένων, για την μεγαλύτερη ταχύτητα των αποτελεσμάτων δημιουργήθηκαν κατάλληλες δομές όπως, “Dataset”, “UtilityMatrix” k“EnchancedUsers” , “EnchancedItems” όπου για παράδειγμα κάθε χρήστης γνωρίζει ποια αντικείμενα έχει βαθμολογήσει και από αυτά ποιοι άλλοι τα έχουν βαθμολογήσει. Αυτό μας δίνει την δυνατότητα από ένα χρήστη με πολύ εύκολο προγραμματιστικό τρόπο να μπορούμε να πάρουμε τους σχετιζόμενους (όμοιους) χρήστες. Επιπλέον, δομές όπως το utilityMatrixOrig βοηθούν για το τελικό validation. Η εφαρμογή είναι σε θέση τόσο μέσω test να θέσει το dataset(hardcoded) όσο και από αρχεία csv (user.csv, item.csv, rating.csv).Επιπλέον ακόμη και στην ανυπαρξία του user.csv μπορούν να καταχωρηθούν οι users μέσω του αρχείου ratings.csv χωρίς να είναι απαραίτητη η ύπαρξη του users.csv, ενώ αν υπάρχουν και τα δύο τότε συμπεριλαμβάνονται σωρεύτηκα και χωρίς διπλότυπες εγγραφές.

Η εφαρμογή μπορεί να δουλέψει σε απλό mode διαβάζοντας από απλά datasets ή σε migration mode όπως αυτό που επεξηγήθηκε παραπάνω για να μπορέσει να υποστηρίξει την ειδική περίπτωση των δύο datasets των Yelp/YelpKC. Σε κάθε περίπτωση η

διαδικασία ξεκινά τοποθετώντας στην μνήμη όλες αυτές τις πληροφορίες (caching), προκειμένου όλοι οι υπολογισμοί να είναι ταχύτατοι πάνω στην μνήμη. Το σύστημα δύναται να τρέχει σε πολλαπλά threads ενώ διαθέτει την δυνατότητα εμφάνισης της ποσοστιαίας εκπαίδευσης του συστήματος με τον ελεγκτή του (controller) να είναι σε θέση να εκτελεί ταυτόχρονα πολλαπλά κομμάτια δειγμάτων, αλλά και ταυτόχρονα για κάθε ένα από αυτά διαφορετικά χαρακτηριστικά επαυξήσεις π.χ. "Friendship Boost Percentage". Αυτό προσδίδει στο σύστημα την ικανότητα να λειτουργεί αυτόνομα χωρίς την ανάγκη συνεχούς εποπτείας για την εναλλαγή των παραμέτρων, ενώ όλα τα στοιχεία για τις διάφορες εκτελέσεις, αποτυπώνονται ευκρινώς τόσο στην κονσόλα όσο και σε log αρχεία. Στην περίπτωση δε του random split υπάρχει η δυνατότητα του ορισμού του fold ώστε να μας τρέξε πολλές φορές με τον τυχαίο δειγματικό χώρο και να μας δώσει ένα συνολικό μέσο όρο. Έτσι στην περίπτωση των runs στο sparce Yelp, έγινε χρήση αυτής της δυνατότητας ώστε να σιγουρευτούμε ότι δεν πέφταμε σε ακραίες τιμές.

Μέσω του υποσυστήματος επαλήθευσης (validation), εμφανίζει την απόδοση του συστήματος. Η απόδοση μπορεί να τεθεί είτε επί του συνόλου των αποτελεσμάτων (mean square error) είτε επί των υπολογισμένων μόνο. Αυτό μας δίνει την δυνατότητα σε περίπτωση που λείπουν πολλά δεδομένα να μην χαλάει η απόδοση λόγω των μηδενικών τιμών που προέκυψαν από την μη ύπαρξη των δεδομένων. Το σύστημα διαθέτει και δυνατότητα αποτύπωσης του ποσοστού και του χρόνου ολοκλήρωσης το οποίο είναι χρήσιμο σε μεγαλύτερα dataset για να μπορούμε να αντιληφθούμε γρήγορα τις τάξεις μεγέθους εκτέλεσης. Σε περίπτωση που κάποιος θα ήθελε να δει πως δουλεύει το σύστημα εσωτερικά, μπορεί να ενεργοποιήσει σε μεγαλύτερο επίπεδο την καταγραφή (logging) και να δει μέσα στα log αρχεία ακριβώς πως υπολογίστηκαν οι εκάστοτε τιμές οι οποίες εν' τέλη αποθηκεύτηκαν στον τελικό Utility Matrix.

Τέλος υπάρχει η δυνατότητα επιλογής της καταγραφής σε κατάλληλο csv αρχείο του τελικού παραγόμενου "Utility Matrix" τόσο για λόγους manual μελέτης και ελέγχου (cross check) όσο και αν κάποιος επιθυμούσε να είχε την δυνατότητα να φορτωθεί στο σύστημα συστάσεως, το τελικά υπολογιζόμενο UtilityMatrix, χωρίς να απαιτηθεί ο εκ' νέου υπολογισμός του από τα αρχικά δεδομένα. Αυτό θα μπορούσε να έχει εφαρμογή και σε περιπτώσεις cluster, όπου θα μπορούσε να γίνει ο αρχικός υπολογισμός και μετά απλά να φορτωθούν τα αποτελέσματα σε διάφορα instances μέσα στο cluster που τρέχουν το

σύστημα συστάσεων για λόγους balancing, χωρίς να πρέπει κάθε ένα από αυτά τα συστήματα να προβεί στην χρονοβόρα και κοστοβόρα διαδικασία της εκ' νέου εκπαίδευσης του UtilityMatrix προκειμένου να είναι σε θέση να λειτουργήσει και να παράγει συστάσεις παραγωγικά. Το σύστημα είναι σε θέση να προτείνει συστάσεις για έναν συγκεκριμένο χρήστη που ορίζεται στο αρχείο παραμέτρων για το PoC αλλά μπορεί να οριστεί και μέσω του API.

3.3 ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΟΥ

Κατά την ανάλυση του αλγορίθμου θα αναφερθούμε τόσο στον συνολικό αλγόριθμό όσο και το πώς αλλάζει η συνάρτηση πρόβλεψης rating με βάση την πληροφορία από social networks. Σε γενικές γραμμές ο αλγόριθμός ακολουθεί τα εξής βασικά βήματα :

- Εισαγωγή δεδομένων και τοποθέτηση τους στον Utility Matrix και UtilityMatrixOrig μέσω του υποσυστήματος «Data Importer»
- Διαχωρισμός των δεδομένων (Rand/Seq) για δημιουργία train set και test set μέσω του υποσυστήματος «Spliter».
- Εύρεση της ομοιότητας των χρηστών που έχουν βαθμολογήσει τουλάχιστον ένα από τα αντικείμενα που έχει βαθμολογήσει ο εκάστοτε χρήστης μέσω του υποσυστήματος «UserEqualityCalculator».
- Ενεργοποίηση των N πιο όμοιων χρηστών και του K threashold ομοιότητας για την επιλογή των όμοιων χρηστών μέσω του υποσυστήματος «UserCounterUserEqualityThreshold»
- Εύρεση ratings για τα αντικείμενα των χρηστών που δεν έχουν αξιολογηθεί από τους ίδιους βάση των όμοιων χρηστών τους, με ταυτόχρονη επαύξηση των εκάστοτε συντελεστών σε περίπτωση φιλίας και άλλων κοινωνικών χαρακτηριστικών μέσω του κύριου υποσυστήματος «RatingCalculator».
- Τελικός υπολογισμός απόδοσης MSE με χρήση του επικαιροποιημένου UtilityMatrix και του UtilityMatrixOrig στο υποσύστημα »Evaluator»

Πιο συγκεκριμένα, μετά την είσοδο των δεδομένων από κατάλληλη δομή και αφότου τα στοιχεία αυτά βρεθούν στην μνήμη, ξεκινά η διαδικασία του collaborative filtering (user/user) , μέσω της οποίας το σύστημα για κάθε ένα χρήστη (σε κάθε thread) εντοπίζει ποια αντικείμενα από τον UtilityMatrix έχει βαθμολογήσει και στην συνέχεια για αυτά τα αντικείμενα βρίσκει ποιο άλλοι χρήστες στον utility matrix έχουν επίσης βαθμολογήσει τουλάχιστον έναν από αυτά. Το σύστημα καταχωρεί σε κατάλληλη δομή κάθε ένα από αυτούς τους χρήστες και τις ομοιότητες τους ελέγχοντας αν έχουν ξανά εισαχθεί και επιστρέφει αυτή την λίστα (ώστε να μην την ξανά υπολογίζει). Για κάθε έναν από αυτούς τους χρήστες το σύστημα θα υπολογίσει τον βαθμό ομοιότητας τους με τους

σχετιζόμενους άλλους χρήστες. Η υλοποίηση του αλγορίθμου ομοιότητας είναι παραμετρική. Έτσι μπορεί κάποιος για παράδειγμα να επιλέξει αν επιθυμεί να χρησιμοποιήσει την συγκεκριμένη υλοποίηση της ομοιότητας που αφορά στον "Cosine Similarity" implementer, που είναι και ο default, είτε τον " Jaccard" implementer, ή για παράδειγμα κάποιον άλλον που μπορεί να φτιαχτεί και να τεθεί παραμετρικά στο σύστημα σαν κλάση τύπου similarity implementeter. Το τελευταίο είναι έτσι σχεδιασμένο για επεκτασιμότητα.

Στην συνέχεια το σύστημα για κάθε ένα από τα αντικείμενα που υπάρχουν στον utlityMatrix και για κάθε έναν από τους όμοιους χρήστες του αρχικού χρήστη, θα υπολογίσει στα αντικείμενα που ο αρχικός χρήστης δεν έχει βαθμολογήσει, την τιμή που προκύπτει από τους όμοιους του χρήστες. Η διαδικασία αυτού του υπολογισμού παίρνει ως είσοδο δύο πίνακες (arrays). Ο μεν πρώτος έχει τις τιμές που έχουν δώσει οι χρήστες, ο δε δεύτερος έχει τον βαθμό ομοιότητας τους και υπολογίζει την τελική τιμή όχι ισόποσα σε σχέση με το πλήθος τους αλλά θέτοντας υπόψη και το βάρος τους (weighted average).

Στην περίπτωση δε που είναι ενεργοποιημένη η συσχέτιση τους ως φίλοι στα social media το σύστημα επαυξάνει την τιμή του "φίλου" με ένα ποσοστό το οποίο καθορίζει παραμετρικά ο χρήστης πριν ξεκινήσει η εκτέλεση του συστήματος.

Πέραν αυτών υπάρχει η δυνατότητα το σύστημα να τρέξει συντηρητικά λαμβάνοντας υπόψη για κάθε χρήστη μόνο τις βαθμολογίες που έχει δώσει διαβάζοντας από τον "UtlityMatrixOrig". Εναλλακτικά, θα μπορούσε να εκτελεστεί και πιο δυναμικά, διαβάζοντας από τον "UtilityMatrix" ο οποίος ανανεώνεται συνεχώς. Αυτό σημαίνει πρακτικά ότι αν ο χρήστης Α είχε αρχικά βαθμολογήσει 3 από τα 7 αντικείμενα, και κατά την εκπαίδευση του συστήματος υπολογιστούν και τα υπόλοιπα 4 αντικείμενα, τότε όταν έρθει στην συνέχεια ένας νέος χρήστης που σχετίζεται με αυτόν (τον Α), τότε θα μπορούσαν να συνυπολογιστούν στις νέες υπολογιζόμενες βαθμολογίες του Β και οι νέο-υπολογιζόμενες του Α και όχι μόνο οι αρχικές του. Αυτό θα μπορούσε στην αρχή πιθανόν να δημιουργήσει κάποια προβλήματα στην απόδοση του συστήματος αφού το σύστημα δεν θα εκπαιδευόταν αμιγώς με τα πραγματικά δεδομένα αλλά θα βασιζόταν και στα υπολογιζόμενα με το όποιο ρίσκο αρχικού σφάλματος αλλά και μετέπειτα σωρευτικά σε προγενέστερες λάθος υπολογιζόμενες εκτιμήσεις. Εντούτοις, η συγκεκριμένη δυνατότητα θα μπορούσε να ενεργοποιηθεί σε δεύτερο χρόνο, μετά την αρχική εκπαίδευση του

συστήματος και κατά την διαδικασία που το σύστημα θα γίνονταν δυναμικό σε πραγματικό χρόνο και θα επικαιροποιούσαι το μοντέλο του με κάθε νέα εισαγωγή.

Για την διαδικασία επαλήθευσης (validation) του συστήματος, έχει δημιουργηθεί η δυνατότητα του διαχωρισμού (spliter) και της εκτίμησης (evaluation). Έτσι το σύστημα στο τέλος της αρχικοποίησης (Initialization / pre-operartion) και πριν εκκινήσει την διαδικασία των υπολογισμών, διαβάζοντας επίσης παραμετρικά από το αρχεία ρυθμίσεων, είναι σε θέση να κόψει το δείγμα στο ποσοστό που θα του έχει καθοριστεί. Επιπλέον είναι δυνατόν να δηλωθεί αν επιθυμούμε το δείγμα να κοπεί τυχαία ή σειριακά, προκειμένου να μπορέσουν τα αποτελέσματα (ranks) να είναι επαληθεύσιμα. Στην περίπτωση του σειριακού μπορούμε ακόμη και να καθορίσουμε το offset πάνω στο αρχείο από το οποίο θέλουμε να το χωρίσουμε ώστε να πάρουμε για παράδειγμα πολλά κομμάτια του δειγματικού χώρου και όχι μόνο την αρχή.

Αν δεν κάποιος θεωρούσε ότι επιλέγοντας συγκεκριμένα σειριακά κομμάτια, για παράδειγμα από την αρχή, την μέση και το τέλος, όπως έτρεξαν οι μετρήσεις μας, θα κατέληγε σε λανθασμένα συμπεράσματα λόγω του Bias της επιλογής, θα υπέπιπτε σε λανθασμένα συμπεράσματα δεδομένου ότι ο αρχικός δειγματικός χώρος έχει προκύψει με την εντολή random της mongo και export σε csv διασφαλίζοντας την τυχαιότητα των δεδομένων που επιλέγονται σύμφωνα με το documentation.

Θα πρέπει τέλος να τονίσουμε άλλες δύο παραμετρικές δυνατότητες που δίνονται στο σύστημα και αυτές είναι του “SIMILARITY_THRESHOLD_VALUE” και του “NUMBER_OF_SIMILAR_USERS”. Τα δύο αυτά χαρακτηριστικά μπορούν να ενεργοποιηθούν από το χρήστη μέσω του αρχείου ρυθμίσεων καθώς επίσης και τον καθορισμό των τιμών τους.

Για το μεν πρώτο "SIMILARITY_THRESHOLD_VALUE", αν ενεργοποιηθεί θα θεωρηθούν από το σύστημα ως όμοιοι χρήστες (similarUsers) σε σχέση με τον αρχικό /τρέχον χρήστη, μόνο εκείνοι που έχουν βαθμό ομοιότητας πάνω από το κατώτατο όριο (threshold). Είναι προφανές ότι τα παραπάνω μπορούν να δουλέψουν ταυτόχρονα για περισσότερες δοκιμές.

Επιπλέον, δημιουργήθηκε ο λεγόμενος “MultiFieldsRateProposer” ο οποίος λαμβάνει υπόψη του και άλλα πεδία που γίνονται register σε αυτόν και τα οποία θέτουν το

όριο (threshold) αλλά και το πόσο επαύξησης που θα έχει η επιρροή ενός χρήστη στον αρχικό χρήστη σε σχέση με την τιμή που έδωσε για κάποιο αντικείμενο (βαρύτητα). Έτσι έχουν δημιουργηθεί και γίνει register τα ακόλουθα :

- Το "AverageStarsSimilarityBoost" που θέτει το πόσο επαύξησης βάση του μέσου όρου αστεριών που έχει από τους άλλους χρήστες. Σε αυτό υπολογίζεται η απόλυτη διαφορά των μέσων αστεριών η οποία διαιρείτε με το μέγιστο ποσό αστερίων (5) και το ποσό αυτό αφαιρείτε από την μονάδα. Το αποτέλεσμα που προκύπτει πολλαπλασιάζεται με τον συντελεστή επαύξησης και επιστρέφεται προκειμένου να προστεθεί με τα υπόλοιπες επαυξήσεις πάνω στην αρχική βαρύτητα/ομοιότητα
- Το "ReviewCountSimilarityBoost" που θέτει το όριο αλλά και το πόσο επαύξησης βάση του πλήθους των κριτικών (reviews) που έχει κάνει ο χρήστης αυτός
- Το "FansSimilarityBoost" που θέτει το όριο αλλά και το πόσο επαύξησης βάση του πόσους ακολούθους (fans) έχει ο συγκεκριμένος χρήστης
- Το "UsefullSimilarityBoost" που θέτει το πόσο επαύξησης βάση του πόσο χρήσιμα θεωρούν οι άλλοι χρήστες τις κριτικές του συγκεκριμένου χρήστη

Κάθε ένα από τα παραπάνω, διαθέτει έναν αντίστοιχο διακόπτη υπό μορφή δυαδικής μεταβλητής (boolean) στο αρχείο συστήματος, μέσω της οποίας καθορίζεται αν θα εφαρμοστεί ο εκάστοτε υπολογισμός στον συνολικό αλγόριθμο ή όχι.

3.3.1 Συνάρτηση Πρόβλεψης Rating με Βάση Πληροφορία από Social Networks

Παρακάτω θα παρουσιαστεί ο τρόπος που γίνεται ο υπολογισμός του σταθμισμένου μέσου όρου των αξιολογήσεων των όμοιων χρηστών με βάση την ομοιότητα που έχει υπολογισθεί για τον καθέναν, καθώς επίσης και πως επηρεάζεται η ομοιότητα ως μέγεθος με βάση την πληροφορία από social networks.

Σε πρώτη φάση θα πρέπει να γίνει κατανοητό πως υπολογίζεται ο σταθμισμένος μέσος όρος μέσω του παρακάτω μαθηματικού τύπου αλλά και μιας μεθόδου του κώδικα που εκτελεί το σύστημα συστάσεων που κατασκευάστηκε.

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}, \quad (1)$$

όπου :

$\bar{x}$: ο σταθμισμένος μέσος όρος των ratings

w : το εκάστοτε βάρος του εκάστοτε χρήστη όπως αυτό θα έχει προκύψει (βλέπε παρακάτω)

x : το εκάστοτε rate του εκάστοτε χρήστη

n : το πλήθος των χρηστών που λαμβάνουν χώρα στον υπολογισμό ενός rating για τον χρήστη του οποίου τα rating που λείπουν προσπαθούν να υπολογιστούν

```java
/**
 * Calc weighted average
 *
 * @param numbers
 * @param weights
 *
 * @return calcWeightedAverage
 *          The calculated weighted average
 */
public static Double calcWeightedAverage(List<Double> numbers, List<Double> weights) {
    Double result=null;
    Double numerator  = 0.0;
    Double denominator= sumNumberedList(weights);
    int counter=0;
    for (Double number : numbers) {
        numerator += number*weights.get(counter);
        counter++;
    }
    result=numerator/denominator;
    result = roundNum(result, Constants.DECIMAL_NUMBERS);
    return result;
}
```

Εικόνα 6 - Κώδικας υπολογισμού Σταθμισμένου Μέσου Όρου

Μέσω του configuration δίνεται η δυνατότητα να επιλεγεί ο κατάλληλος "RatingProposer". Όπως φαίνεται και στο παρακάτω παράδειγμα υπάρχουν 3 :

- Για λειτουργία ως απλού RS χρησιμοποιείται ο βασικός Recommender μέσω της αντίστοιχης κλάσης «WeightedAverageRateProposerOperationImpl».
- Για λειτουργία ως απλού RS με Social Friendship association Data, χρησιμοποιείται ο «SocialFriendshipRateProposerOperationImpl»
- Τέλος για την λειτουργία του πιο σύνθετου RS με χρήση δεδομένων από κοινωνικά δίκτυα (Friendship Association + social characteristics), γίνεται χρήση του «MultiFieldsRateProposerOperationImpl».

```
    /**
     * The class of the rating proposer to run against
     */
    //public static final String RATE_PROPOSED_CLASS="gr.papazogloup.rs.impl.engine.collaborative.core.rating.proposer.bl
.op.WeightedAverageRateProposerOperationImpl";
    //public static final String RATE_PROPOSED_CLASS="gr.papazogloup.rs.impl.engine.collaborative.core.rating.proposer.bl
.op.SocialFriendshipRateProposerOperationImpl"; //$NON-NLS-1$
    public static final String RATE_PROPOSED_CLASS="gr.papazogloup.rs.impl.engine.collaborative.core.rating.proposer.bl.op
.MultiFieldsRateProposerOperationImpl"; //$NON-NLS-1$
```

Εικόνα 7 - Επιλογή Rating Proposer

Στην απλή αυτή περίπτωση απλά γίνεται η κλήση του παραπάνω κώδικα (εικόνα 6) με BoostedWeight όπως αυτό προέκυψε από τον επιλεγμένο αλγόριθμο ομοιότητας του τρέχοντος χρήστη με τον εκάστοτε όμοιο χρήστη του.

$$\boldsymbol{BW = SV} \quad (2)$$

όπου :

BW : BoostedWeight ,

SV : SimilarityValue

Στην τελευταία περίπτωση που είναι και η ενεργή στο σύστημα μας, περιλαμβάνει ταυτόχρονα και την επαύξηση λόγω φιλίας μεταξύ των χρηστών μέσω συνυπολογισμού των "SIMILARITY_THRESHOLD_VALUE" και "NUMBER_OF_SIMILAR_USERS", αλλά και τις επαυξήσεις από το λοιπά χαρακτηριστικά των δεδομένων των κοινωνικών δικτύων έχουμε τα ακόλουθα.

Πιο συγκεκριμένα όπως φαίνεται στον παρακάτω κώδικα οι βαθμολογίες και οι αντίστοιχες βαρύτητες/ομοιότητες των χρηστών (για ένα οσυγκεκριμένο αντικείμενο του εκάστοτε χρήστη) δεν υπολογίζεται απευθείας μέσω του κώδικα που αναφέρθηκε στην εικόνα 6, αλλά πρώτα γίνεται ο υπολογισμός των επαυξημένων βαρυτήτων/ομοιοτήτων από τα social data.

Πιο συγκεκριμένα όπως φαίνεται στην παρακάτω εικόνα, ο υπολογισμός είναι το άθροισμα των επιμέρους υπολογισμού που αφορούν στην συσχέτιση των χρηστών λόγω φιλίας αλλά και στα άλλα (4) χαρακτηριστικά.

```java
    /**
     * Calculates the weight of the similar user rate based
     * on dynamic fields boosted between current and similar user.
     * At first the weight of the rate for a similar user is the same
     * as his similarity value among him (the similar user) and the current user.
     * This type of calc add boosted weight for a series of dynamic fields.
     * For example if there is a friendship between the current user and the similar then
     * the weight is the base weight (similarityValue) plus the friendship percentage boost for friendship multiplied with the base weight (similarity value)
     * The same addition happens with all the other fields.
     * That means that the final boosted weight depends in all those factors if they apply and their percentage boost.
     * That makes the final boosted weight between a similar user and a current user a simple addition of percentage multiplies
     * but on the other hand a dynamic percentage that takes in consideration a number of factors in order to come up with the
     * final boosted weight.
     * That final boosted weights are those that will be multiplied with each similar user's rate and all together added to come up with the final proposed by the system rate.
     * There is however the possibility of negative percentage in the logic of punishment that makes the hall calculation more business dynamic.
     * In a nutshell this type of calculation of boosted weights works in the logic of addition between all the dynamic fields boosted weights
     * That doesn't mean that this should be the only algotithmic logic.
     * Perhapse another algorithm could add more business logic before  add in advanced all those boosts as the final calculated boosted weight.
     * That's only one algorithmic logic that could be extract to class so we could have more different implementations.
     *
     * @return boostedWeight
     *          calculated boosted weight
     *
     * @throws LogicException
     * @throws DataException
     */
    private Double calcBoostedWeight() throws DataException, LogicException {
        Double boostedWeight = similarityValue;
        boostedWeight+= friendShipContribution();
        boostedWeight+= averageStarsSimilarityContribution();
        boostedWeight+= usefulContribution();
        boostedWeight+= fansContribution();
        boostedWeight+= reviewCountContribution();
        return boostedWeight;
    }
```

Εικόνα 8 - Συνολικός υπολογισμός επαυξημένων βαρυτήτων λόγω Social Data

$$BW = FBW + ASSBW + UBW + FABW + RCBW + .. \quad (3)$$

όπου :

BW : Boosted weight ,

FBW: Friendship boosted weight

ASSBW : Average stars similarity boosted weight

UBW : Usefull boosted weight

FABW : Fans boosted weight

RCBW : Review count boosted weight

Για την βασική περίπτωση που αφορά στην φιλία όπως φαίνεται και στον παρακάτω κώδικα, βλέπουμε ότι επιστρέφεται για επαύξηση στην αρχική βαρύτητα το γινόμενο της αρχικής βαρύτητας με το ποσοστό επαύξησης.

```
1   /**
2    * Calculates the friendShip boosted weight
3    *
4    * @return friendShipBoosteddWeight
5    *          The friendShip boosted weight
6    */
7   private Double friendShipContribution() {
8       Double friendShipBoosteddWeight = 0.0;
9       if(friendSimilarityBoostPercentage!=0 && isFriend()) {
10          friendShipBoosteddWeight = similarityValue * friendSimilarityBoostPercentage/100;
11      }
12      return friendShipBoosteddWeight;
13  }
```

Εικόνα 9 - Υπολογισμός επαύξησης λόγω φιλίας

Συνεπώς σε αυτή την περίπτωση έχουμε

$$BW = SV + (SV*FSBP/100) \quad (4)$$

όπου :

BW : BoostedWeight ,

SV : SimilarityValue ,

FSBP : FriendSimilarityBoostPercentage

Θα πρέπει να σημειώσουμε εδώ ότι αν είναι ενεργοποιημένο το “SIMILARITY_THRESHOLD_VALUE” θα περνάνε στους όμοιους χρήστες που θα δίνονται

για υπολογισμό στον κώδικα της εικόνας 6, μόνοι οι χρήστες που έχουν ομοιότητα πάνω από το threshold.

Αντίστοιχα, σε περίπτωση που είναι ενεργοποιημένο το «NUMBER_OF_SIMILAR_USERS» θα περάσουν ως όμοιοι χρήστες μόνο οι N πρώτοι πιο ίδιοι χρήστες.

Στην περίπτωση των χαρακτηριστικών η λογική τους χωρίζεται σε δύο κατηγορίες. Στην πρώτη κατηγορία έχουμε το χαρακτηριστικό “average stars”. Σε αυτή την περίπτωση η λογική όπως βλέπουμε στον παρακάτω κώδικα είναι η εξής :

- Υπολογίζεται η απόλυτη τιμή “averageStarsAbsDiff” ανάμεσα στα αστέρια που έχουν οι δύο χρήστες (ο τρέχον και ο όμοιος του)
- Υπολογίζεται ως averageStarsSimilarNormilized ο λόγος της προηγούμενης υπολογισθείσας τιμής “averageStarsAbsDiff” ως προς την μέγιστη τιμή αστεριών (5) και το αποτέλεσμα αυτό αφαιρείτε από το 1 προκειμένου να έχουμε μια κοινωνικοποιημένη τιμή
- Τέλος το αποτέλεσμα επαύξησης θα είναι το γινόμενο της αρχικής ομοιότητας ”similarityValue” επί του ποσού που μόλις υπολογίσθηκε «averageStarsSimilarNormalized” επί του ποσού του ποσοστού επαύξησης για τους μέσους όρους των αστεριών δια 100. Το ποσό αυτό θα επιστραφεί, για να προστεθεί με τα υπόλοιπα όπως έχουμε εξηγήσει προηγουμένως.

```
1   /**
2    * Calculates the normalized average star similarity between user and similar user
3    * and multiplies it with the Configuration#AVERAGE_STARS_SIMILARITY_BOOST_VALUE
4    * in order to find the final average stars similarity boosted weight.
5    *
6    * @return averageStarsBoostedWeight
7    *         The average star boosted weight
8    */
9   private Double averageStarsSimilarityContribution() {
10      Double averageStarsBoostedWeight = 0.0;
11      Float userAverageStars = user.getAverage_stars();
12      Float similarUserAverageStars = similarUser.getAverage_stars();
13      if(userAverageStars!=null && similarUserAverageStars!=null) {
14          float averageStarsAbsDiff = Math.abs(userAverageStars-similarUserAverageStars);
15          Float averageStarsAbsDiffNormalized = averageStarsAbsDiff/Constants.MAX_RATE_VALUE;
16          Float averageStarsSimilarNormalized = 1-averageStarsAbsDiffNormalized;
17          averageStarsBoostedWeight = similarityValue * averageStarsSimilarNormalized * Configuration.AVERAGE_STARS_SIM
ILARITY_BOOST_VALUE/100;
18      }
19      return averageStarsBoostedWeight;
20  }
```

Εικόνα 10 - Υπολογισμός Επαύξησης λόγω Μέσων Αστεριών

Επομένως, ο υπολογισμός είναι :

$$ASBW = SV * [1 – (|UAS - SUAS |/MRV)] * ASSBV /100 \quad (5)$$

όπου :

ASBW : Average stars boosted weight,

SV : Similarity value ,

UAS : User average stars,

SUAS : Similar user average stars,

MRV : Max rate value,

ASSBV : Average stars similarity boost value

Τέλος για τα υπόλοιπα τρία χαρακτηριστικά ο κώδικας που υπολογίζει της επαυξήσεις της βαρύτητας παρουσιάζεται στην παρακάτω εικόνα.

```
/**
 * Calculates the boost weight contribution for a social field that
 * is beeing calculated using a threashold and a boost for that field.
 *
 * @return socialThresholdFieldBoostWeight
 *          Social thredshold field boosted weight
 */
private Double calcSocialThresholdFieldBoost() {
    Double boostedWeight = 0.0;
    User similarUser = socialThresholdFieldBoostBean.getSimilarUser();
    Double similarityValue = socialThresholdFieldBoostBean.getSimilarityValue();
    Integer threshold = socialThresholdFieldBoostBean.getThreshold();
    Integer boost = socialThresholdFieldBoostBean.getBoost();
    Integer socialFieldValue = socialThresholdFieldBoostBean.getSocialFieldValue();
    Boolean friendsOnly = socialThresholdFieldBoostBean.getFriendsOnly();

    if(threshold<0 || !isFriendshipContintionsValid(similarUser, friendsOnly) ){
        return boostedWeight;
    }

    if(socialFieldValue!=null && socialFieldValue>=threshold) {
            boostedWeight = similarityValue * boost/100;
    }

    return boostedWeight;
}
```

Εικόνα 11 - Υπολογισμός επαυξήσεων για άλλα χαρακτηριστικά Social

Όπως και στην περίπτωση που αφορούσε απλά τον συνυπολογισμό της επαύξησης λόγω φιλίας, έτσι και για τα 3 πεδία (Fans , Usefull & Review Count) το “BoostedWeight“ είναι το

άθροισμα του SimilarityValue με το γινόμενο του SimilarityValue και του BoostPercentage για το εκάστοτε χαρακτηριστικό.

$$BW = SV + (SV*BP/100) \quad (6)$$

όπου :

BW : Boosted weight,

SV : Similarity value ,

BP : Boost percentage

Θα πρέπει να σημειώσουμε τέλος, πως για να ενεργοποιηθεί η επίδραση αυτών των χαρακτηριστικών θα πρέπει οι τιμές αυτές να ξεπερνούν το αριθμητικό όριο το οποίο ορίζεται ξεχωριστά για το κάθε ένα από αυτά στο αρχείο ρυθμίσεων.

4. ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

4.1 ΓΕΝΙΚΑ

Στο παρόν κεφάλαιο, θα παρουσιαστούν τα σύνολα δεδομένων και οι προεργασία που απαιτήθηκε, καθώς επίσης και τα ευρήματα μέσω των πολλαπλών εκτελέσεων του συστήματος συστάσεων με δεδομένα από κοινωνικά δίκτυα. Εκ' πρώτης έγινε διερεύνηση πιθανών dataset, προκειμένου να υπάρχει η δυνατότητα για ανάλυση "Colaborative Filtering User-User" με ένα dataset που να περιέχει ταυτόχρονα "user-item-rating" αλλά και "userUserRelationship" με όσο το δυνατόν περισσότερα στοιχεία/χαρακτηριστικά, τόσο για τους users όσο και για τα items (π.χ. movie Genres).

4.2 ΔΕΔΟΜΕΝΑ

Όπως έχουμε ήδη αναφέρει και στην εισαγωγή, στην αρχή έγινε η προσπάθεια δημιουργίας ενός απλού συστήματος συστάσεων (recommendation system). Πρόκειται για ένα σύστημα "Collaborative Filtering (user/user)". Για την δημιουργία λοιπόν ενός τέτοιου συστήματος, θα ήταν αρκετή εκ' πρώτης η χρήση ενός απλού dataset όπως του "Movielens (100k)". Το σύστημα έτρεξε με το συγκεκριμένο dataset προκειμένου να εξεταστεί η απόδοση του ως απλού non-social recommender. Ο χρόνος εκτέλεσης του ήταν 5 λεπτά και η απόδοση του άγγιζε την τιμή των 1,3 MSE κάτι που θεωρήθηκε επαρκές ως πρώτο milestone προκειμένου να συνεχίσουμε με δεδομένα που διαθέτουν και συσχετίσεις που σχετίζονται με κοινωνικά δίκτυα.

Στην δεύτερη φάση που θα έπρεπε να συνδεθεί και το social κομμάτι εξετάστηκα μεταξύ άλλων τα : douban-dataset, epinions, filmTrust, Flixster-dataset, hetrec2011-movielens, imdb, lastFm. Η έρευνα των διάφορων dataset κατέληξε στο «ciao» το οποίο είχε τα βασικά χαρακτηριστικά που θα θέλαμε για ένα social recommender. Θα πρέπει να σημειώσουμε ότι το «Chiao» dataset, το οποίο αναφέρεται και στο librec library, είχε απόδοση MSE περίπου 2. Τα χαρακτηριστικά αυτά ήταν τουλάχιστον δύο αλλά όχι μόνο.

Το πρώτο χαρακτηριστικό που θα έπρεπε να διαθέτει το dataset θα ήταν να απορρέει (είτε με 3 αρχεία είτε με 2 που έχει όμως και τις 3 πληροφορίες κ.τ.λ.) η πληροφορία της τριπλέτας user/item/rating.

Το δεύτερο χαρακτηριστικό που θα έπρεπε να διαθέτει θα ήταν η συσχέτιση φιλίας μεταξύ των φίλων (friendship association).

Πέραν των δύο παραπάνω χαρακτηριστικών και προκειμένου να μπορεί να μελετηθεί και σε μεγαλύτερο βάθος η όποια αλληλοσύνδεση ποιοτικών κοινωνικών χαρακτηριστικών, και με την καθοδήγηση της καθηγήτριας έγινε η επιλογή του yelp το οποίο είχε περισσότερες δυνατότητες στα πεδία που προσέφερε και άρα θα είχε μεγαλύτερες δυνατότητες συνδυασμών.

Έτσι εκτός από τα δύο βασικά χαρακτηριστικά που αναφέραμε παραπάνω, για κάθε χρήστη διέθετε και πληθώρα χαρακτηριστικών όπως:

- "Μέσος Όρος Αστεριών" (average stars) που έχει να κάνει με τις ψηφοφορίες σε προϊόντα
- "Πλήθος κριτικών" (review count) που έχει να κάνει με το πόσες κριτικές έχει κάνει ο συγκεκριμένος χρήστης
- "Πόσο χρήσιμες είναι οι κριτικές (usefull similarity) που έχει να κάνει με το πόσες κριτικές έχει κάνει ο συγκεκριμένος χρήστης
- "Πόσους ακολούθους" έχει ο εκάστοτε χρήστης (UsefullSimilarityBoost)

Για το λόγο αυτό έγινε η προσπάθεια ενσωμάτωσης του στο σύστημα. Αυτό οδήγησε στις εξής παρατηρήσεις σε σχέση με το dataset τις οποίες και θα έπρεπε να θέσουμε ενώπιον μας. Το "Yelp" dataset έχε ναι μεν την συσχέτιση μεταξύ φίλων/χρηστών (user association) αλλά τα δεδομένα είναι τρομερά αραιά. Από την άλλη το YelpKC (Kaggle Challedge 2013) είναι λιγότερο αραιό αλλά δεν έχει user associations.

Επιπλέον, θέλαμε την δυνατότητα της τριπλέτας "user,item,rating" για να υλοποιήσουμε το "Collaborative Filtering (user/user)" αλλά και το "Friendship Association" το οποίο θα μας έδινε την δυνατότητα του social networking χωρίς όμως μεγάλο sparce αλλά και με τα επιπλέον κοινωνικά χαρακτηριστικά. Το πρώτο πρόβλημα, ήταν ότι χάθηκε πάρα πολύς χρόνος για να γίνει κατανοητό ότι το dataset είναι πολύ αραιό sparce. Στην αρχή, στην προσπάθεια να τρέξουν σε όλο το dataset τα runs, οι χρόνοι ήταν απαγορευτικοί. Αυτό είχε σαν αποτέλεσμα όπως φαίνεται και στα γραφήματα που θα ακολουθήσουν στο κεφάλαιο 4 να προσπαθήσουμε, στην αρχή με τυχαίο τρόπο, να πάρουμε sample (mongo-sample no bias) για να μικρύνουμε το δείγμα. Στην συνέχεια ακολουθήθηκε η προσέγγιση της λήψης δειγματικού χώρου με συνθήκες π.χ. άτομα που έχουν ψηφίσει πάνω από 5000 επιχειρήσεις ώστε να σιγουρευτούμε ότι οι χρήστες θα είναι

λίγοι και άρα το πλήθος των αξιολογήσεων θα ήταν πιο περιορισμένο και συνεπώς και το dataset, αφού θα είχαμε λίγους χρήστες που θα είχαν ψηφίσει πάνω από 5000 αντικείμενα καθώς επίσης και τα ratings τους για αυτά.

Σε συνέχεια των προπαρασκευαστικών ενεργειών μας για την εξεύρεση του βέλτιστου dataset και αφού παρατηρήθηκε ότι ούτε αυτές οι δυο προσεγγίσεις δεν ήταν σε θέση να καλυτερεύσουν την κατάσταση, έγινε προσπάθεια εξεύρεσης καλύτερου dataset. Στην έρευνα αυτή μέσα από την μελέτη κάποιων papers βρέθηκε αυτό που είχαμε εντοπίσει και πειραματικά, το γεγονός δηλαδή ότι το Yelp dataset ήταν τρομερά sparce καθώς αναγραφόταν καθαρά. Με ακόμη περαιτέρω μελέτη βρεθήκαμε στο Kaggle dataset. Το πρόβλημα με αυτό ήταν ότι λόγω του ότι αφορούσε διαγωνισμό που είχε λήξει, δεν ήταν διαθέσιμο και ήταν αρκετά δύσκολο ο εντοπισμός του.

Αφού έγινε δυνατός ο εντοπισμός και η απόκτηση του εν' λόγω dataset, και ενώ έδειχνε στις εκτελέσεις εκπαίδευσης του συστήματος (runs) ότι ήταν αρκετά λιγότερο αραιό (sparce) και μάλιστα ολοκληρώνονταν σε πολύ λιγότερο χρόνο, εν' τούτης στην προσπάθεια ενεργοποίησης της φιλικής συσχέτισης διαπιστώθηκε ότι δεν υπήρχε φιλική συσχέτιση σε αυτό το dataset. Προφανώς, επειδή ο διαγωνισμός ήταν παλιός και δεν υπήρχε η ανάγκη για "recommendation με social network" αλλά απλά "most efficient recomentation algorithm" δεν υπήρχε στο dataset αυτό η φιλική συσχέτιση.

4.3 MIGRATION

Η λύση που προτάθηκε ήταν ο συγκερασμός μεταξύ των δύο dataset δεδομένου ότι πρόκειται για το ίδιο dataset Yelp απλά το ένα σύγχρονο και sparce ενώ το άλλο συμπυκνωμένο για τον διαγωνισμό Kaggel Challedge 2013 (YelpKC). Κατά την διαδικασία ελέγχου για τον συσχετισμό των userIdies των δύο αυτών dataset ώστε να βρεθούν τα friendshop userIdies στο dataset του Kaggle που είναι λιγότερο sparce έγινε εμφανές ένα ακόμη πρόβλημα. Τα idies από το 2013 και το dataset που προσφέρεται σήμερα ήταν διαφορετικά. Για να λυθεί αυτό το πρόβλημα έγινε προσπάθεια να συνδεθούν μέσω συντεταγμένων που ήταν κοινά. Αυτή η στρατηγική έδειξε πολύ σύντομα την αδυναμία της λόγω του ότι είχαν αλλάξει στο νέο dataset κατά 3 ψηφία λιγότερα τις συντεταγμένες , ενώ επιπλέον στις ίδιες συντεταγμένες υπήρχαν νέα αντικείμενα (καταστήματα) που είχαν και ως business διαφορετικά categories/tags.

Ο μόνος κοινός τόπος μεταξύ των 2 datasets ήταν δυστυχώς τα σχόλια του εκάστοτε χρήστη τα οποία ως πεδίο ήταν text και δεν μπορούσε να γίνει κάποιας μορφής indexing. Για την διευκόλυνση της διαδικασίας, έγινε ξανά χρήση της mongo ώστε με aggregation να γίνει εφικτή η διασύνδεση σε ένα ενδιάμεσο πίνακα και τέλος να προκύψουν με export τα κατάλληλα user.csv που περιείχαν τα idies τόσο του παλιού όσο και του νέου σε νέα στήλη. Με κατάλληλο migration κώδικα στην pre-process operation του recomender, κατέστει εν' τέλη δυνατή η διαδικασία να γίνονται cached όλα τα στοιχεία user.csv, business,csv, ratijngs.csv . ενώ τα idies των φίλων γύρισαν όλα στα idies του Yelp Kaggle ώστε να μπορεί το σύστημα να εντοπίζει το friendship association. Η όλη παραπάνω διαδικασία ήταν αρκετά επίπονη αφού επί της ουσίας για ένα απλό dataset το οποίο θα είχε την τριπλέτα, user.item,rating και το friendship association καταναλώθηκε αρκετός χρόνος δυσανάλογος μόνο και μόνο για να υπάρχει ένα καλό dataset. Επειδή όμως, και πάλι το migration αυτό ήταν πολύ χρονοβόρο, έγινε και σε αυτό mongo-random sampling και το μέγιστο στο οποίο έγιναν εκτελέσεις ήταν αυτό των 5k επιχειρήσεων αντί για 11k, ενώ ο χρόνος για κάθε run ήταν 7 ώρες με single thread.

Στην συνέχεια η εφαρμογή έγινε και multithreaded με αποτέλεσμα να είναι μελλοντικά δυνατή και η εκτέλεση του συνόλου των 11k επιχειρήσεων σε εύλογο χρονικό διάστημα αφού μέσω multihtrading (10 threads) η εφαρμογή είναι περίπου 400% πιο efficient. (μετρήσεις στα 1000 αντικείμενα από 11 λεπτά σε 3). Επομένως είχαμε για migration δύο datasets το "**Yelp**" και το "**Yelp Kaggle** (ή **YelpKC**)":

Το **"Yelp"** το οποίο είναι τρομερά sparce και το οποίο περιέχει..

> 1,5M χρήστες

> 200K Businesses

> 6M reviews

και το οποίο εισήχθη στην mongo με τις παρακάτω εντολές :

```
> use Yelp;
> mongoimport --db Yelp --collection user --file yelp_academic_dataset_user.json          [1.518.169]   > db.user.find().count();
> mongoimport --db Yelp --collection business --file yelp_academic_dataset_business.json  [  188.593]   > db.business.find().count();
> mongoimport --db Yelp --collection review --file yelp_academic_dataset_review.json      [5.996.996]   > db.review.find().count();
```

Παρακάτω βρίσκονται τα urls από όπου έγινε δυνατή η απόκτηση τους :

https://www.yelp.com/dataset (somepath/.../yelp.tar.gz)

https://www.yelp.com/dataset/documentation/main

https://github.com/Yelp/dataset-examples

Το **"Yelp Kagle"** το οποίο ήταν για ένα διαγωνισμό το οποίο περιέχει :

> 44k χρήστες

> 12K Businesses

> 6M reviews

και το οποίο εισήχθη στην mongo με τις παρακάτω εντολές :

```
> use YelpKC;
> mongoimport --db YelpKC --collection user --file yelp_academic_dataset_user.json [ 43.873] > db.user.find().count();
> mongoimport --db YelpKC --collection business --file yelp_academic_dataset_business.json [ 11.537] > db.business.find().count();
> mongoimport --db YelpKC --collection review --file yelp_academic_dataset_review.json [ 229.907] > db.review.find().count();
```

4.4 ΕΚΤΕΛΕΣΕΙΣ

Το σύνολο των εκτελέσεων και κατ' επέκταση δεδομένων που θα παρουσιαστούν φτάνουν στα 1000. Τα ευρήματα αυτά υποστηρίζονται από τα ατόφια δεδομένα (raw data) που μπορούν να βρεθούν στο Παράρτημα Κεφ 7.2. "Τιμές για τα διάφορα τρεξίματα", ενώ παρουσιάζονται μέσω γραφημάτων για καλύτερη κατανόηση και παρουσίαση. Λόγω των πολλαπλών εκτελέσεων και μεγάλου πλήθους πληροφορίας, προκειμένου να είναι το δυνατόν ευκολότερη η εξαγωγή ποιοτικής γνώσης από αυτά, υπήρξε η συμπυκνωμένη αποτύπωση τους μέσω γραφημάτων Sparkline, Τα γραφήματα χωρίζονται στο αρχικό dataset Yelp το οποίο ήταν πολύ αραιό, και στο migrated Yelp-YelpKC. Επιπλέον, γίνεται διαχωρισμός σε random split (πέραν του random dataset split από την mongo) για τα slices μεταξύ εκπαίδευσης και test (για το evaluation του συστήματος), καθώς επίσης και σε sequencial split (στο random dataset split από την mongo για περιορισμό του dataset / υποσύνολο) προκειμένου να είναι επαληθεύσιμα τα αποτελέσματα σε περίπτωση επανεκτέλεσης της εκπαίδευσης του συστήματος.

Σε γενικές γραμμές, έγιναν εκτελέσεις σε διαφορετικά κομμάτια του υποσυνόλου του δείγματος προκειμένου να έχουμε όσο το δυνατόν πιο αντικειμενική εικόνα για την συμπεριφορά του. Έτσι για τα κομμάτια που αφορούν το τεστ της απόδοσης του συστήματος είτε στο 10% είτε στο 25% πάρθηκαν αυτά τα κομμάτια από την αρχή την μέση και το τέλος. Επιπλέον θα πρέπει να αναφερθεί ότι παρουσιάζεται η απόδοση του συστήματος MSE, αλλά και η MSE* (ratings>0) δηλαδή πάνω στις τιμές που είναι μεγαλύτερες από το 0, αφού το σύστημα ότι δεν γνωρίζει στον Utility Matrix το κάνει 0, και άρα θέλουμε να είμαστε σίγουροι ότι ελέγχουμε την απόδοση για πραγματικές τιμές.

Τα διάφορα τρεξίματα (runs) χωρίζονται στο σύστημα ως απλού recommender, ως recommender με σταθμισμένη επαύξηση της ψήφου των φίλων χρηστών με κάποιο συντελεστή (friendship Boost %) όταν αυτοί αναγνωρίζονται στο migrated dataset, ενώ επιπλέον εφαρμόζονται πλήθος άλλων χαρακτηριστικών όπως το SimilarityThreshold (πόσο ίδιοι είναι και αν είναι από ένα σημείο και πάνω να εφαρμόζεται η επαύξηση), similarUsers (τους N πρώτους πιο ίδιους χρήστες να χρησιμοποιηθούν για την επαύξηση) αλλά και τον συνδυασμό τους.

Τέλος γίνεται αναφορά και στα χαρακτηριστικά που δίνει το συγκεκριμένο dataset KagleKC όπως reviewCount, fans κτλ ,τα οποία χρησιμοποιήθηκαν από το σύστημα με κατάλληλη επέκταση του συστήματος προγραμματιστικά.

4.5 YELP

Όπως αναφερθήκαμε ήδη προηγουμένως, το dataset Yelp είναι ένα σύνολο δεδομένων το οποίο περιέχει την τριπλέτα χρήστες, αντικείμενα (επιχειρήσεις) και αξιολογήσεις, καθώς επίσης και την συσχέτιση (φιλία) μεταξύ των χρηστών. Το πρόβλημα με το συγκεκριμένο dataset είναι ότι είναι πάρα πολύ αραιό με αποτέλεσμα να είναι δύσκολος ο χειρισμός για απλά συστήματα συστάσεων αφού η εκπαίδευση στο σύνολο ενός τέτοιου αραιού dataset θα απαιτούσε πολύ μνήμη, επεξεργαστική ισχύ και χρόνο εκτός και αν το σύστημα φτιαχνόταν ειδικά για τέτοιου είδους αραιά σύνολα δεδομένων.

Παρακάτω θα αναφέρουμε, το πως βρεθήκαμε αντιμέτωποι με αυτό το πρόβλημα και πως έγινε η προσπάθεια για την αντιμετώπιση του, μέχρι που τελικά αποφασίστηκε να συνδυάσουμε άλλο ένα dataset και να κάνουμε migration όπως έχει ήδη αναφερθεί προκειμένου να γίνει λιγότερο αραιό και να έχουμε κάποια ευρήματα.

4.5.1 Yelp - Τυχαίο Τεστ Δείγμα

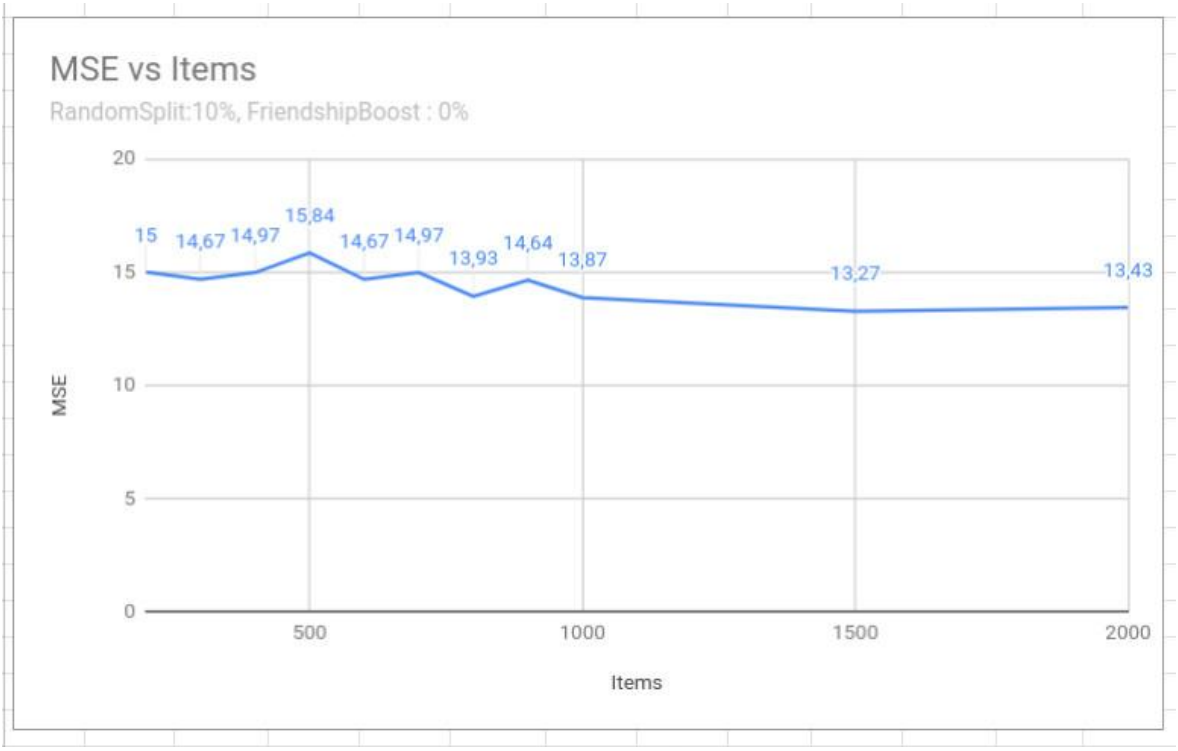
Σε πρώτη φάση, πήραμε το αρχικό dataset και το περάσαμε μέσα σε μια mongoDB. Αυτό μας έδωσε την δυνατότητα να μπορούμε να πάρουμε υποσύνολα του dataset αφού κατέστη άμεσα προφανές ότι ο χρόνος εκτέλεσης και η μνήμη που θα απαιτείτο ήταν εκτός σχεδιαστικών απαιτήσεων για την παρούσα διπλωματική.

Στον παρακάτω πίνακα βλέπουμε τα στοιχεία που αφορούν στο Yelp Dataset. Πιο συγκεκριμένα έχουμε τα items, τους χρήστες, τα ratings, την πυκνότητα ανά subset (το subset έχει γίνει με mongo sampling), τον χρόνο εκπαίδευσης και την απόδοση (MSE).

Να σημειωθεί ότι με πορτοκαλί βλέπουμε την συμπεριφορά του συστήματος για το dataset “MovieLens100k”. Έτσι θα δούμε ότι για 1682 αντικείμενα, 943 χρήστες και 100.000 αξιολογήσεις, με πυκνότητα 6,305% (αρκετά μεγάλη), το σύστημα μας έχει απόδοση 1,3 MSE σχετικά ικανοποιητικό.

Users	**943**	2521	5502	9451	11643	13499	18362	19550	26,440	28,053	28,846	39,760	57,668
Items	**1682**	100	200	300	400	500	600	700	800	900	1000	1500	2000
Ratings	**100,000**	2540	5630	9847	12240	14207	19601	20696	28,702	30,449	31,983	45,264	67,290
Density%	**6.305**	1.008	0.512	0.347	0.263	0.21	0.178	0.151	0.136	0.121	0.111	0.076	0.058
Time	**5.00m**	4s	13s	1.20m	6.36m	6.60m	8.30m	10.80m	34.58m	37.10m	18.97m	43.00m	3.2h
MSE	**1.3**	17	15	14.67	14.97	15.84	14.67	14.97	13.93	14.64	13.87	13.27	13.43

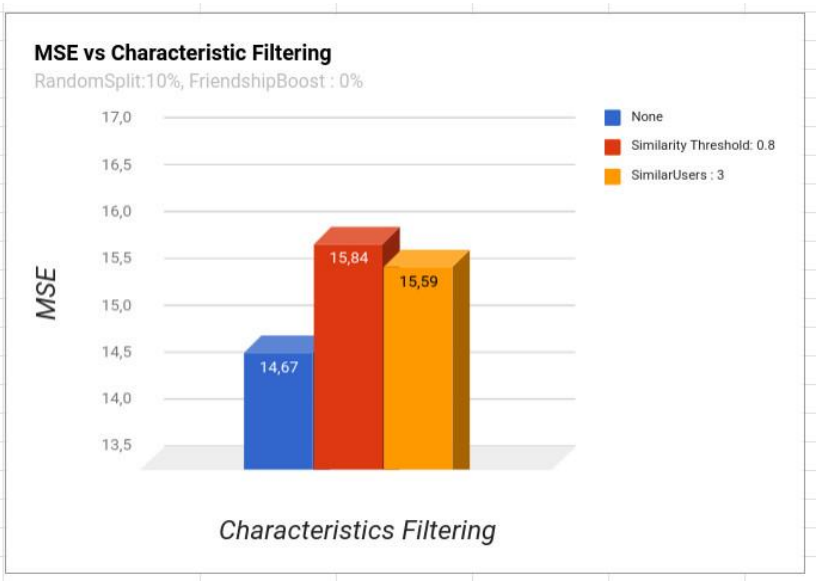
Πίνακας 1 - Αποδόσεις Recommender για τυχαίο δείγμα Yelp



Εικόνα 12 - Απόδοση Recommender για τυχαίο δείγμα Yelp

Αυτό που παρατηρούμε παραπάνω, είναι μια τάση καλυτέρευσης του recommender, αλλά λόγω του ότι είναι εξαιρετικά αραιό το dataset το σύστημα δεν μπορεί να λειτουργήσει τουλάχιστον μέχρι το σημείο που κατάφερε να τρέξει. Το συγκεκριμένο dataset (Yelp) έχει 1.518.169 χρήστες, 188.593 αντικείμενα και 5.996.996 κρητικές με πυκνότητα 0,002%. Στο τρέξιμο των 2000 αντικειμένων απαιτήθηκαν πάνω από 3 ώρες. Αυτό κατέδειξε από νωρίς ότι με το σύστημα που έχει υλοποιηθεί και το συγκεκριμένο dataset όπου όλα θα αυξάνονταν εκθετικά και σε μνήμη και σε υπολογιστική ισχύ, δεν υπήρχε περιθώριο να συνεχίσουμε στο ίδιο μοτίβο, αυξάνοντας δηλαδή το sampling σε items μέσω mongo sampling από το αρχικό αραιό Yelp dataset.

Στην προσπάθεια, μας να δούμε έστω και σε αυτή την χαμηλή απόδοση πως συμπεριφέρεται το σύστημα με την ενεργοποίηση των δύο βασικών χαρακτηριστικών SimilarityThreshold και SimilarUsers, παρατηρήθηκε χειροτέρευση της απόδοσης. Τα αποτελέσματα αυτά προέκυψαν ως μέσος όρος 10 εκτελέσεων (fold : 10) μιας και επρόκειτο για τυχαίο κομμάτι για το test set (split 10%) για το evaluation.

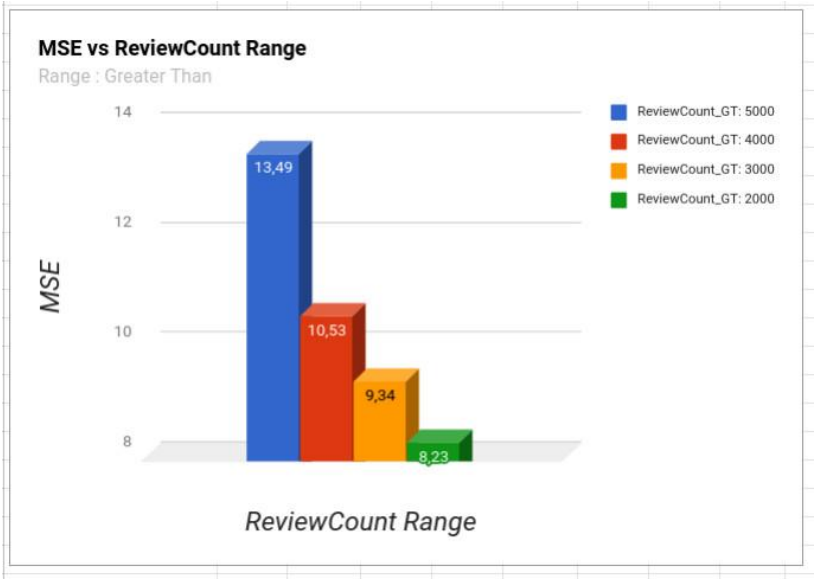


Εικόνα 13 - MSE vs Characteristic Filtering

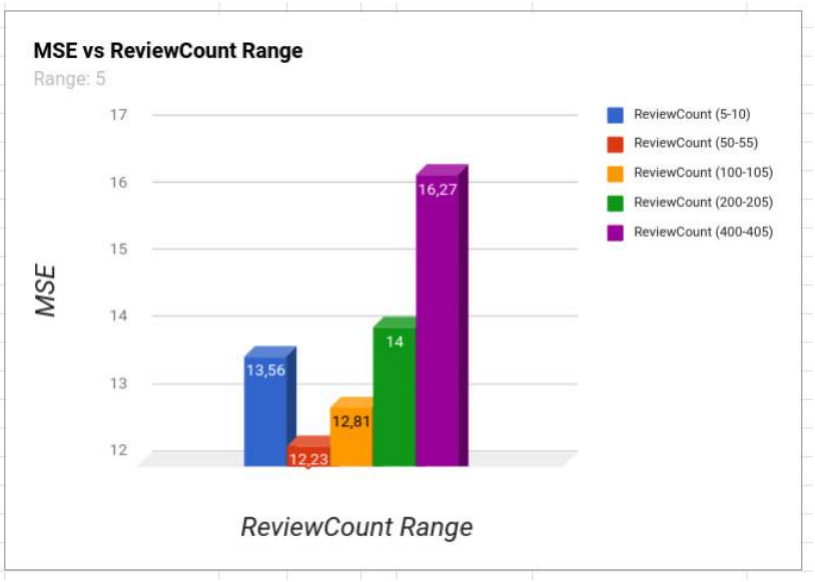
4.5.2 Yelp - Τυχαίο Δείγμα τεστ με Bias

Δεδομένου, ότι η προηγούμενη προσέγγιση λόγω του αραιού του συνόλου δεδομένων Yelp, έφτανε στα όρια της σε επίπεδο επεξεργασίας (χρόνος τρεξίματος, μνήμη κτλ) με αποδόσεις πολύ μικρές, η επόμενη αντιμετωπίσει ήταν ο διαχωρισμός του sparce dataset βάση του πλήθους των αξιολογήσεων. Έτσι, εξήχθησαν υποσύνολα του αρχικού dataset μέσω επερωτήσεων της mongo με κλειστά όρια στο πλήθος των αξιολογήσεων. Αυτό μας έδωσε την δυνατότητα να παίρνουμε ένα υποσύνολο το οποίο έχει μεγαλύτερη πυκνότητα λόγω του ότι οι χρήστες αυτοί είχαν κάνει αρκετές αξιολογήσεις και συνεπώς θα είχαμε δεδομένα για ανάλυση από το να συμπεριλαμβάνουμε χρήστες αδρανείς.

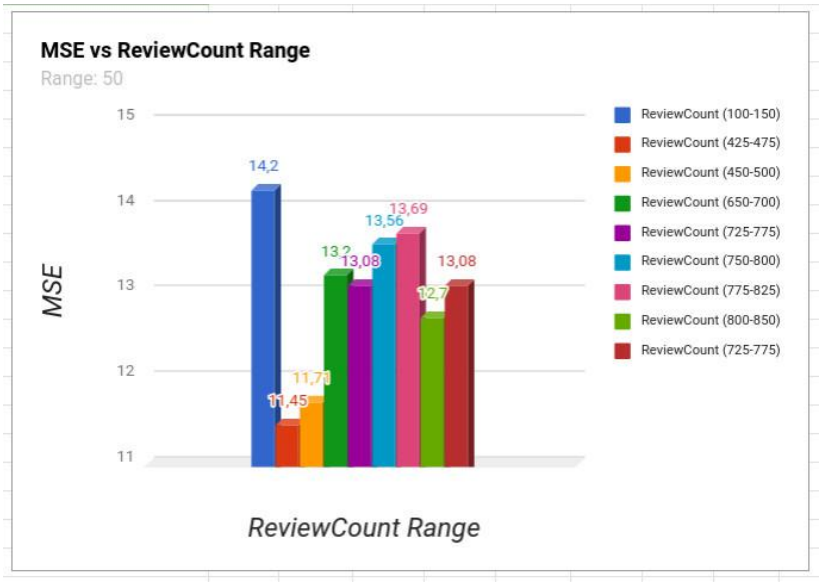
Τα δεδομένα στο νέο υποσύνολο είχαν μόνο τους χρήστες αυτούς που αναφέραμε παραπάνω με μεγαλύτερο/μικρότερο από το καθορισμένο όριο πλήθους αξιολογήσεων, και αντίστοιχα μόνο αντικείμενα που σχετίζονται με αυτούς τους χρήστες και τις αξιολογήσεις τους. Το παραπάνω εμπλουτίστηκε με την διαφορετικότητα του εύρους αλλά και του ελάχιστου ποσού. Έτσι όπως θα δούμε παρακάτω δημιουργήθηκαν υποσύνολα για εύρη 5,50,100,200 και αντίστοιχα ελάχιστο πλήθος αξιολογήσεων από 100 έως 5000, καθώς επίσης και εύρη με ανοικτό το άνω όριο από 100.



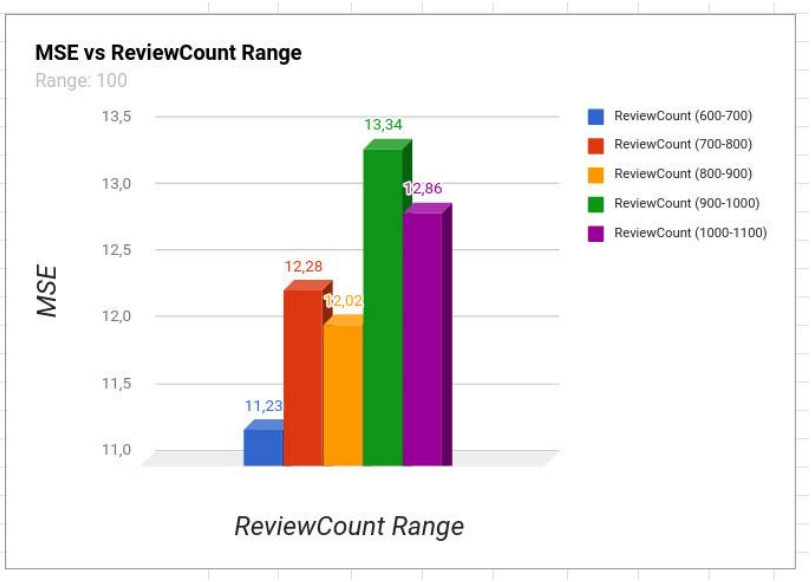
Εικόνα 14 - MSE vs ReviewCount Range (Greater Than)



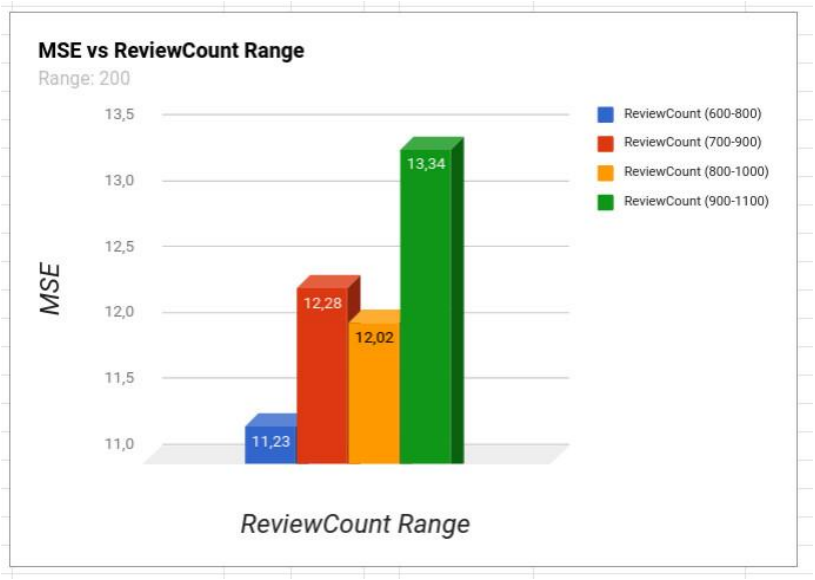
Εικόνα 15 - MSE vs ReviewCount Range (Range : 5)



Εικόνα 16 - MSE vs ReviewCount Range (Range : 50)



Εικόνα 17 - MSE vs ReviewCount Range (Range : 100)



Εικόνα 18 - MSE vs ReviewCount Range (Range : 200)

Απ' όλα τα παραπάνω διαφαίνεται ότι το σύστημα είχε καλύτερη απόδοση με αυτή την στρατηγική επιλογής υποσυνόλου. Είναι σαφές ότι το υποσύνολο με πλήθος κριτικών ανά χρήστη άνω των 2000 (ανοικτό εύρος στο πρώτο γράφημα με βάση τα 2000), είχε πιο

ικανοποιητική απόδοση, αλλά αυτό έχει να κάνει με το γεγονός ότι είχε περισσότερα αντικείμενα, σε ένα βέβαια πιο πυκνό δείγμα λόγω της στρατηγικής της φύσης της επιλογής που επιλέχθηκε. Τέλος, δεν είναι σαφές κάποιο συμπέρασμα για το πιο εύρος έχει καλύτερα αποτελέσματα και ακόμη περισσότερο σε πιο σημείο (λίγο πλήθος κριτικών, μεσαίο πλήθος κριτικών, μεγάλο πλήθος κριτικών).

4.6 YELPKC

Όπως έχουμε ήδη αναφερθεί, δημιουργήθηκαν κατάλληλα υποσύνολα μέσω migration, από τα το αρχικό YelpKC dataset και του Yelp dataset που περιείχε τις συσχετίσεις των χρηστών, ανάλογα με το πλήθος των αντικείμενων. Πιο συγκεκριμένα, το δείγμα κόπηκε σε 100 αντικείμενα, 1000, και 5000 σε σύνολο 11.537. Ο λόγος που έγινε κάτι τέτοιο είχε να κάνει με την ταχύτητα εξαγωγής αποτελεσμάτων από το σύστημα συστάσεων δεδομένου ότι τόσο από μεριάς mongo (migration) το migration του συνόλου θα απαιτούσε 1 εβδομάδα πλήρους υπολογιστικής ισχύς ενώ και η εκτέλεση του ίδιου του συστήματος συστάσεων (ιδίως πριν γίνει multithreating) είχε μεγάλους χρόνους εκτέλεσης αλλά και μνήμης.

Παρακάτω παρουσιάζονται οι εκτελέσεις που έγιναν σε δείγματα των 100, 1000 και 5000 αντικειμένων, σε τυχαία κομμένο δείγμα (mongo sampling), με διαφορετικά ποσοστά split για το evaluation 10% , 25% σειριακής επιλογής ανά slice (αρχή, μέση, τέλος) με διαφορετικά boost ανά χαρακτηριστικά π.χ. FriendshipBoost, SimilarityThreshold, SimilarUsers κ.α.

4.6.1 Yelpkc – Τυχαίο τέστ δείγμα 10%

Παρακάτω θα δούμε για πρώτη φορά την συμπεριφορά της απόδοσης του συστήματος ως απλού συστήματος συστάσεως και ως συστήματος που λαμβάνει υπόψη δεδομένα από κοινωνικά δίκτυα και πιο συγκεκριμένα την αλληλοσυσχέτιση μεταξύ των χρηστών σε επίπεδο γνωριμίας (φιλίας/αλληλοσύνδεσης).

4.6.1.1 Yelpkc – Τυχαίο τέστ δείγμα 10% - Απλός Recommender

Στα δύο παρακάτω γραφήματα , κάνουμε τυχαίο διαχωρισμό εκπαίδευσης/ελέγχου από το σύνολο των δεδομένων σε ποσοστό 10%. Διατηρούμε σταθερό το FriendshipBoost σε 0% και αποτυπώνουμε την συμπεριφορά του συστήματος σε σχέση με το πλήθος των Items. Αυτό που παρατηρείται, είναι η αύξηση της απόδοσης όσο αυξάνεται το dataset (περισσότερα items). Είναι προφανές εδώ ότι πρόκειται για τον έλεγχο συμπεριφοράς ενός απλού recommender και όχι ενός social recommender αφού η ποσοστιαία επαύξηση λόγω φιλίας είναι μηδενική. Σκοπός ήταν απλά να επιβεβαιωθεί η ανοδική συμπεριφορά της απόδοσης, όταν το σύνολο των δεδομένων αυξάνεται, όπως και επαληθεύθηκε, καθώς και να δούμε σε πραγματικές τιμές την απόδοση του συστήματος.

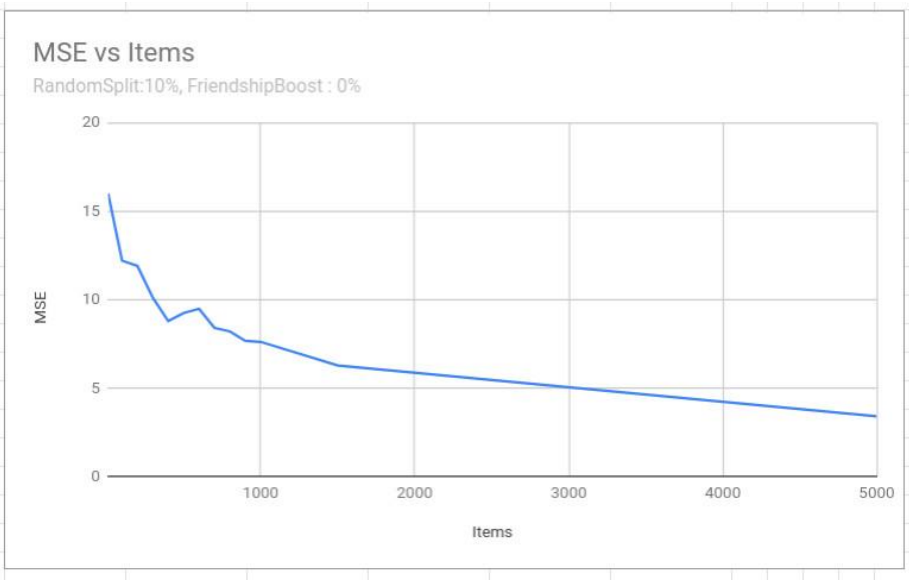
Στον παρακάτω πίνακα αποτυπώνονται οι τιμές που υποστηρίζουν τα επόμενα γραφήματα καθώς επίσης και περαιτέρω ποιοτικές πληροφορίες ανά πλήθος αντικειμένων, όπως το πλήθος των χρηστών, τα ratings, την πυκνότητα του δείγματος αλλά και τον χρόνο εκτελέσεις από τον recommender.

Users	241	2.073	2.973	4.435	5.696	6.083	7.706	7.366	7.869	9.489	9.827	14.090	29.301
Items	10	100	200	300	400	500	600	700	800	900	1000	1500	5000
Ratings	250	2.524	3.753	6.249	8.860	9.481	12.762	12.070	13.278	16.895	18.175	30.168	97.596
Density%	10,373	1,218	0,631	0,470	0,389	0,312	0,276	0,234	0,211	0,198	0,185	0,143	0,067
Time	0h:0m:1s	0h:0m:4s	0h.0m:14s	0h.0m:28s	0h.1m:1s	0h.1m:8s	0h.1m:10s	0h.1m:23s	0h.2m:15s	0h.4m:55s	0h.3m:33s	0h.19m:44s	2h.33m:17
MSE	16	12.2	11.9	10.1	8.78	9.24	9.48	8.4	8.2	7.67	7.61	6.28	3.4

Split : 10%						
Slice 1						
1-[10% of dataset]						
GraphId	**Dataset**	**Kind of Sampling**	**Type of Sampling**	**Items**	**Boost Characteristic**	**Boost Values (%)**
1	YelpKC	Item	Random (of Mongo Random)	10-5000	None	None

Πίνακας 2 - YelpKC (Random Split 10% , Friendship Boost : 0%)

Εικόνα 19 - MSE vs Items (Random Split 10% , Friendship Boost : 0%)



Εικόνα 20 - MSE vs Items (RandomSplit 10%, FriendshipBoost : 0%)

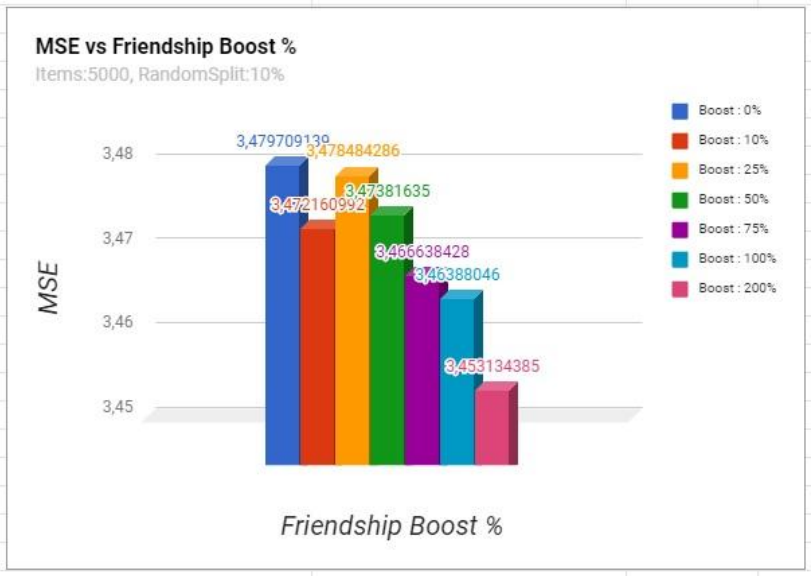
4.6.1.2 Yelpkc – Τυχαίο τέστ δείγμα 10% & Κοιν. Δεδ.

Στο παρακάτω γράφημα βλέπουμε πλέον, πάλι για τυχαίο διαχωρισμό 10% αλλά και για 5000 αντικείμενα (το μισό dataset) πως δείχνει να συμπεριφέρεται το FriendshipBoost ανάμεσα στις τιμές 0,10,25,50,75,100,200%.

Users	Items	Ratings	[	Ratings	/	Max Cells	Density%
29,301	5000	97,596	[	97,596	/	146,505,000	0.067

Split : 10%						
Random						
-						
GraphiId	**Dataset**	**Kind of Sampling**	**Type of Sampling**	**Items**	**Boost Characteristic**	**Boost Values (%)**
2	YelpKC	Item	Random (of Mongo Random)	5000	FriendShip	0,10,25,50,75,100,200

Πίνακας 3 - YelpKC (Random Split 10% , Friendship Boost : 0,10,25,50,75,100,200%)



Εικόνα 21 - MSE vs Friendship Boost % (Items : 5000, Boosts 0,10,25,50,75,100,200%)

Το αποτέλεσμα δείχνει μια ξεκάθαρη τάση καλυτέρευσης της απόδοσης του recommender όσο αυξάνεται η επαύξηση της φιλίας (friendshipboost %). Ο χρόνος εκτέλεσης ήταν 2 ώρες με multithreading (7,5 με ένα thread) σε σύνολο 5000 αντικείμενα (μισό dataset), για κάθε ένα από τους διαφορετικούς συντελεστές επαύξησης. Συνεπώς, σε σύνολο επτά εκτελέσεων (runs) σε 15 περίπου ώρες σε Random διαχωρισμό test set 10% η συμπεριφορά του Friend Boost,χαρακτηριστικού βασισμένο σε δεδομένα από κοινωνικά δίκτυα, δείχνει ευκρινώς ότι τέτοιου είδους πληροφορία μπορεί να βοηθήσει σε ένα σύστημα συστάσεων.

4.6.2 Yelpkc - Σειριακό τεστ δείγμα

Προκειμένου, να είναι δυνατή η επαναλληψιμότητα των αποτελεσμάτων και να μην βασιζόμαστε στην τυχαιότητα, ακολουθήθηκε η στρατηγική της ποσοστιαίας σταθερής επιλογής του τεστ δείγματος από την αρχή την μέση και το τέλος του. Έτσι τα αποτελέσματα μπορούν να επαληθευτούν και να μην χάνονται μέσα στην τυχαιότητα του δείγματος. Με αυτό τον τρόπο δίνεται η δυνατότητα μελέτης και συγκερασμού των αποτελεσμάτων από τα 3 κομμάτια (slices) απ' όπου λαμβάνεται το σύνολο για την δοκιμή της απόδοσης του συστήματος (test set). Να εξηγήσουμε βέβαια ότι αυτό δεν μπορεί να θεωρηθεί ως BIAS αφού ο δειγματικός χώρος αποτελεί προϊόν τυχαίου υποσυνόλου του αρχικού dataset. Έτσι τα πρώτα για παράδειγμα δεδομένα που θα χρησιμοποιούνταν ως test set δεν είναι πράγματι τα πρώτα σειριακά δεδομένα του αρχικού dataset αλλά τυχαία λόγω του mongo sampling.

4.6.2.1 Yelpkc–Σειρ. Τεστ 10%,25% & Κοιν. Δεδ.

Παρακάτω θα δούμε τις αποδόσεις του συστήματος με test set 10% και 25% , με ενεργοποιημένα τα δεδομένα από κοινωνικά δίκτυα που μας παρέχει το migrated Yelp/YelpKC dataset. Όπως ήδη είδαμε κάθε γράφημα, από πάνω του θα συνοδεύεται με την ταυτότητα του, που αφορά στους 3 πίνακες που το συνοδεύουν. Στον πρώτο πίνακα εμφανίζονται χαρακτηριστικά όπως το πλήθος των χρηστών, των αντικείμενων και το αξιολογήσεων καθώς επίσης και την πυκνότητα. Στο δεύτερο πίνακα θα γίνεται αναφορά στα κομμάτια που επιλέχθηκαν για test set (slices) με επακριβή παρουσίαση των εγγραφών (από – έως). Ενώ στο τρίτο πίνακα γίνεται αναφορά στον κωδικό γραφήματος, το dataset, στο είδος δειγματοληψίας (π.χ. αντικείμενα), τον τύπο δειγματοληψίας για το test set (τυχαίο, σειριακό), το πλήθος των αντικειμένων (ανάλογα με το είδος της δειγματοληψίας). Επιπλέον αναφέρεται πιο χαρακτηριστικό λαμβάνει χώρα ως επαύξηση και με τι συντελεστή επαύξησης.

4.6.2.1.1 Αντικείμενα (100)

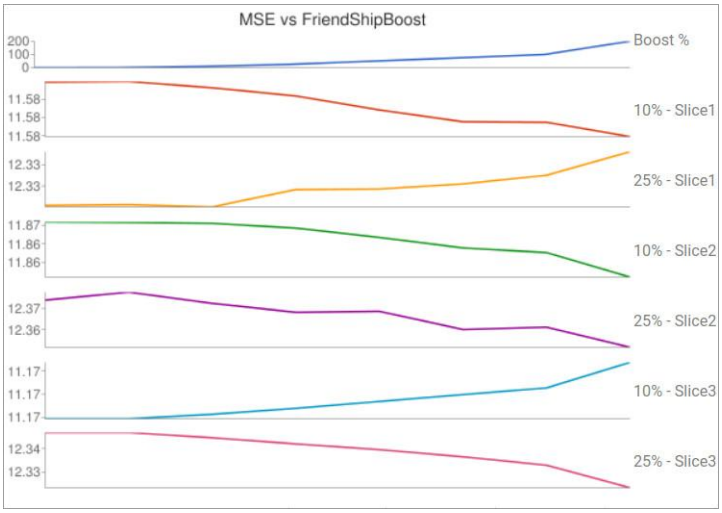
Όπως είναι εμφανές από την ταυτότητα του γραφήματος όπως περιγράφεται παρακάτω, έχουμε να κάνουμε με 2 ποσοστά διαχωρισμού του dataset (10%,25%) και τρία σειριακά κομμάτια, ενώ τα αντικείμενα που διαθέτει το υποσύνολο μας είναι 100. Να σημειώσουμε ότι γίνεται χρήση του χαρακτηριστικού φιλίας το οποίο προμοτάρεται με σταθμισμένο μέσο όρο στους συντελεστές 0,1,10,25,50,75,100,200. Στην ίδια λογική και διαβάζοντας την ταυτότητα (3 πίνακες) των γραφημάτων θα είναι κάποιος σε θέση να αναγνωρίζει τα χαρακτηριστικά του γραφήματος που αφορούν στις αποδόσεις ανάλογα με τις εκτελέσεις (runs) που έγιναν.

Users	Items	Ratings	[	Ratings	/	Max Cells	Density %
2,073	100	2,524	[	2,524	/	207,300	1.218

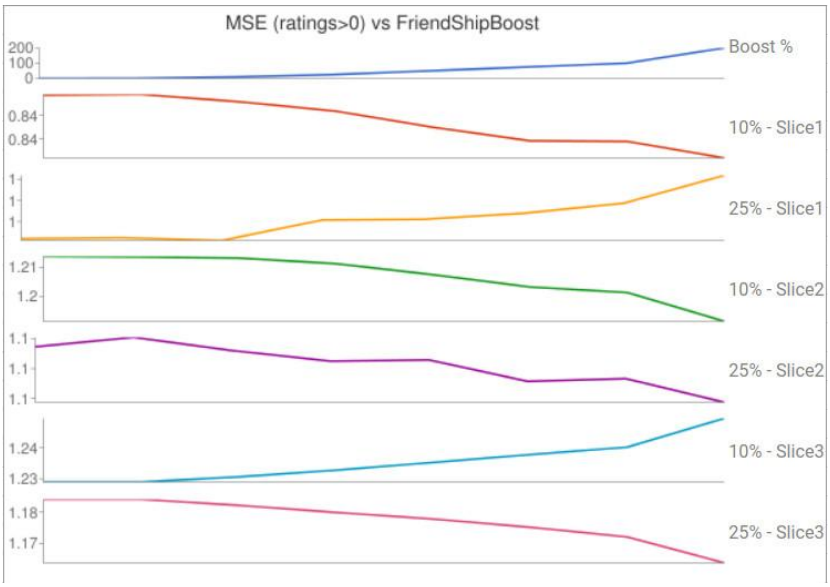
Split : 10%				**Split : 25%**		
Slice 1	**Slice 2**	**Slice 3**		**Slice 1**	**Slice 2**	**Slice 3**
1-253	*1262-1514*	*2270-2522*		*1-632*	*1262-1894*	*1885-2517*

GraphId	Dataset	Kind of Sampling	Type of Sampling	Items	Boost Characteristic	Boost Values (%)
3	YelpKC	Item	Seq (of Mongo Random)	100	FriendShip	0, 10,25,50,75,100,200

Πίνακας 4 - YelpKC (Seq Split 10/25% , Items: 100 Friendship Boost : 0,10,25,50,75,100,200%)



Εικόνα 22 – MSE vs FriendshipBoost (Items 100) Seq-MultiSlice



Εικόνα 23 – MSE (Ratings > 0) vs FriendshipBoost (Items 100) Seq-MultiSlice

Είναι εμφανές, ότι σε ένα υποσύνολο με τόσο λίγα αντικείμενα, η απόδοση του συστήματος συστάσεως είναι μη αποδοτική. Πρακτικά το σύστημα είναι άχρηστο να κάνει προβλέψεις αφού με τέτοιο MSE (12), το λάθος αγγίζει τις 4 βαθμίδες. Έτσι αν κάποιος είχε δώσει 1 αστέρι τότε το σύστημα θα μπορούσε να θεωρήσει ότι η αξιολόγηση θα ήταν 5 αστέρια. Εν' τούτης διαφαίνεται μια μικρή τάση καλυτέρευσης όσο αυξάνεται η επαύξηση του χαρακτηριστικού φιλίας στα δύο από τα τρία (1ο και 2ο) test sets για 10%, και αντίστοιχα για δύο από τα τρία (2ο και 3ο) test sets για 25%. Αυτό πέρα από μια

πρώτη αμυδρά γενική εκτίμηση και τάση που επιδέχεται στη συνέχεια επαλήθευση σε καλύτερες αποδόσεις δεν μπορεί να λέει πολλά για την βοήθεια που προσφέρουν τα δεδομένα κοινωνικών δικτύων σε ένα σύστημα συστάσεων.

4.6.2.1.2 Αντικείμενα (1000)

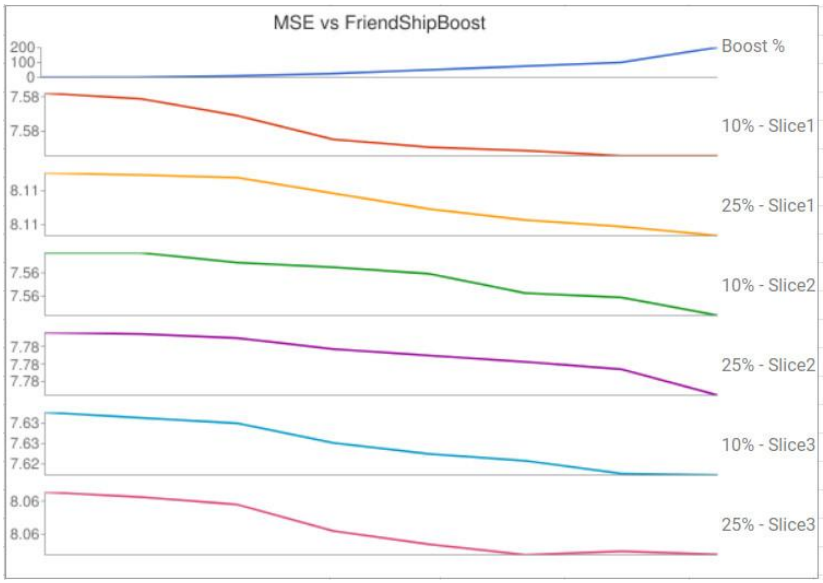
Στην ίδια λογική με το προηγούμενο με μόνη διαφορά ότι έχουμε 1000 αντικείμενα.

Users	Items	Ratings	[	Ratings	/	Max Cells	Density%
9,827	1000	18,175	[	18,175	/	9,827,000	0.185

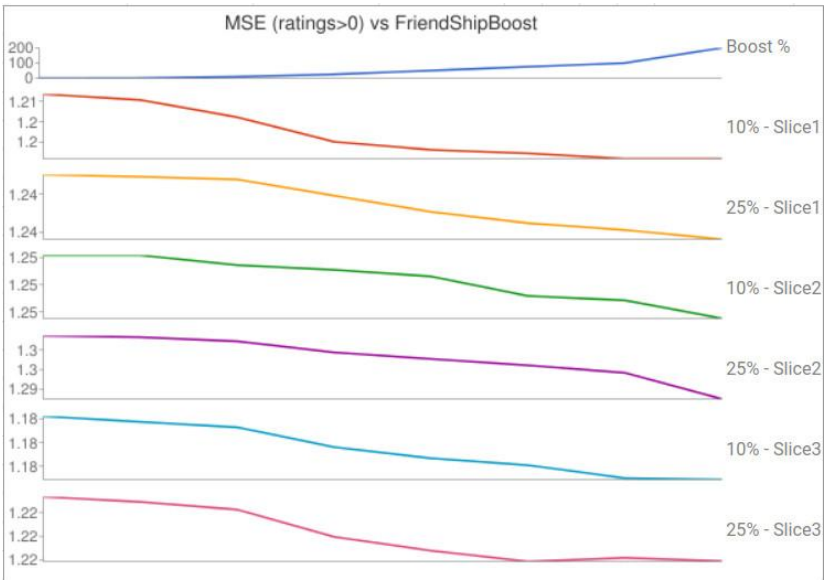
Split : 10%				**Split : 25%**		
Slice 1	**Slice 2**	**Slice 3**		**Slice 1**	**Slice 2**	**Slice 3**
1-1819	*9087-10905*	*16357-18173*		*1-4543*	*6815-11358*	*13630-18173*

GraphiId	Dataset	Kind of Sampling	Type of Sampling	Items	Boost Characteristic	Boost Values (%)
4	YelpKC	Item	Seq (of Mongo Random)	1000	FriendShip	0,1,10,25,50,75,100,200

Πίνακας 5 - YelpKC(Seq Split 10/25% , Items: 1000 Friendship Boost : 0,10,25,50,75,100,200%)



Εικόνα 24 – MSE vs FriendshipBoost (Items 1000) Seq-MultiSlice



Εικόνα 25 – MSE (Ratings > 0) vs FriendshipBoost (Items 1000) Seq-MultiSlice

Παρατηρείται και πάλι μια καλυτέρευση αυτή την φορά σε όλα τα test set slices και των δύο ποσοστών όσο μεγαλώνει η επαύξηση της φιλίας με ανώτερο το 200%. Η ερμηνεία του 200% λόγω του ότι πρόκειται για σταθμισμένο μέσο όρο είναι σαν ο λόγος του συγκεκριμένου χρήστη να λειτουργεί διπλά, δηλαδή σαν να διπλοψηφίζει σε ένα σύνολο προφανώς που αυξάνει κατά ένα τους ψηφισάντες. Με αυτό τον τρόπο όμως η άποψη του μετρά διπλά.

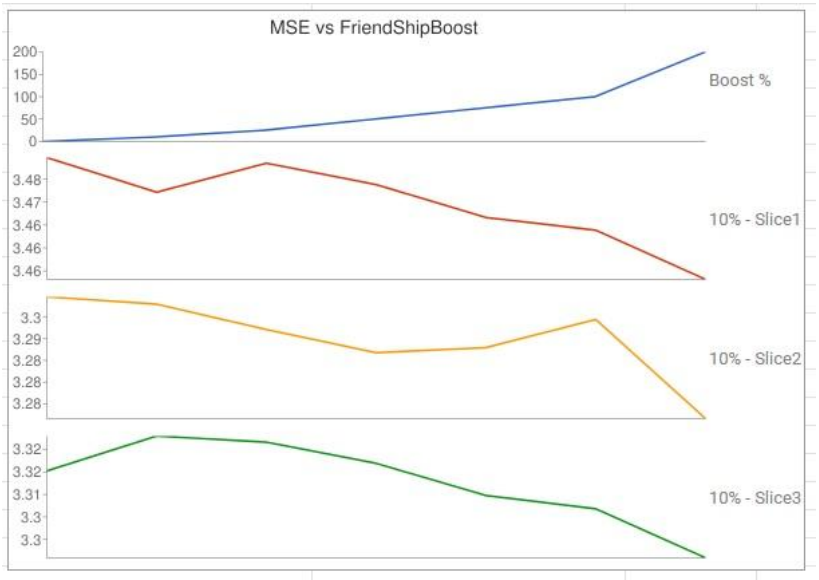
4.6.2.1.3 Αντικείμενα (5000)

Παρακάτω παρατηρούμε την συμπεριφορά στα 5000 αντικείμενα σε 3 slices σειριακά την απόδοση του συστήματος MSE με διάφορα social friendship boost.

Users	Items	Ratings	[	Ratings	/	Max Cells	Density%
29,301	5000	97,596	[	97,596	/	146.505.000	0.067

Split : 10%						
Slice 1			**Slice 2**		**Slice 3**	
1-9760			*48798-58558*		*87834-97594*	
GraphiId	**Dataset**	**Kind of Sampling**	**Type of Sampling**	**Items**	**Boost Characteristic**	**Boost Values (%)**
5	YelpKC	Item	Random (of Mongo Random)	5000	FriendShip	0, 10,25,50,75,100,200

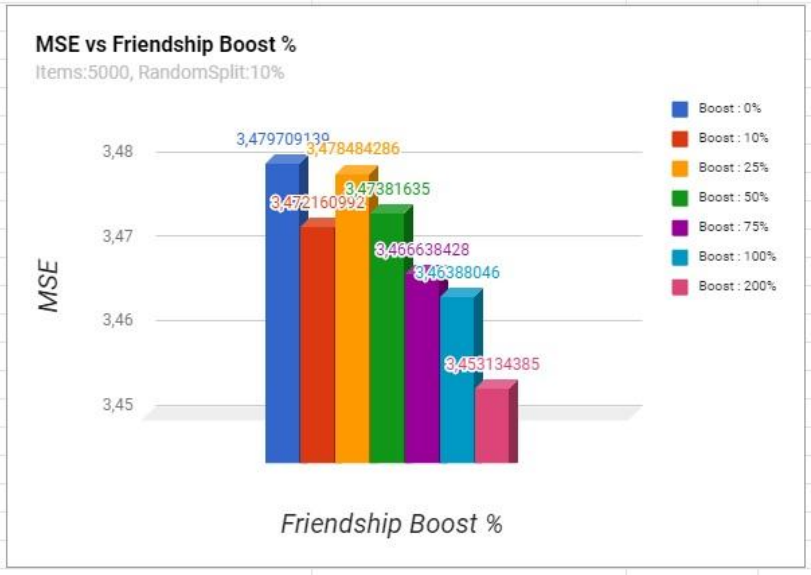
Πίνακας 6 - YelpKC(Seq Split 10% , Items: 5000 Friendship Boost : 0,10,25,50,75,100,200%)



Εικόνα 26 - MSE vs FriendshipBoost (Items 5000) Seq-MultiSlice

Παρακάτω μπορούμε να δούμε πιο καθαρά την απόδοση στα 5000 αντικείμενα (που αποτελούν και το μισό dataset), στο πρώτο σειριακό slice για τις διαφορετικές επαυξήσεις

της φιλικής απόδοσης (friendship boost), προκειμένου να είναι πιο κατανοητές οι διακυμάνσεις.



Εικόνα 27 - MSE vs FriendshipBoost (Items 5000) Seq – Slice 1

Το σύστημα φέρεται να αποδίδει θετικά μέχρι και το 200% με κάποιες μικρές αναταράξεις στην απόδοση στην αρχή (10% με 50%). Οι συγκεκριμένες αποκλείσεις είναι δεν είναι εντυπωσιακές, εν' τούτης το πλήθος είναι το μισό του δειγματικού χώρου και σειριακό (όχι random) άρα επαληθεύσιμο. Μπορούμε λοιπόν από τις συγκεκριμένες μετρήσεις να θεωρήσουμε σε ένα τόσο μεγάλο σύνολο, ότι είναι υπαρκτή μια τάση καλυτέρευσης της απόδοσης ενός συστήματος συστάσεων με κοινωνικά δεδομένα όταν οι φιλία μεταξύ των χρηστών προμοτάρεται και μάλιστα μέχρι το 200%.

Να σημειωθεί ότι σε προπαρασκευαστικές εκτελέσεις φάνηκε ότι πάνω από το 200% τα αποτελέσματα ήταν άκρως αρνητικά και αυτός είναι ο λόγος που οι επαυξήσεις φτάνουν μέχρι αυτό το ποσοστό.

4.6.2.2 Yelpkc–Σειρ. Τεστ 10%,25% Όριο Ομοιότητας

Στο σημείο αυτό παρουσιάζονται τα αποτελέσματα από εκτελέσεις πάλι με σειριακό test set σε ποσοστά 10 και 25% αντίστοιχα, με 100 και 1000 αντικείμενα (10% του αρχικού δειγματικού χώρου του Yelp/YelpKC migrated) και με ενεργοποιημένο αυτή την φορά το χαρακτηριστικό που λειτουργεί ως κόφτης κάτω από το οποίο δεν συνυπολογίζει την φιλία μεταξύ των χρηστών. Πιο συγκεκριμένα, αν κάποιος φίλος δεν είναι όμοιος αρκετά (κάτω από το similarity threshold) τότε δεν λογίζεται ως φίλος και δεν συνυπολογίζονται σε αυτόν οι συντελεστές επαύξησης που γίνονται στα διάφορα runs.

4.6.2.2.1 Όριο Ομοιότητας - Αντικείμενα : 100

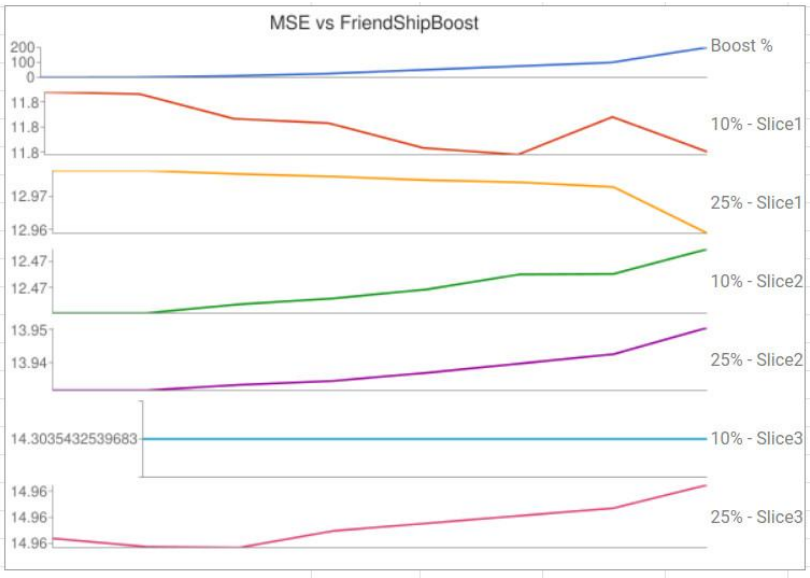
Ξεκινάμε και πάλι με 100 αντικείμενα σε 2 test sets των 10 και 25%, και αντίστοιχα 3 slices, με όλα τις γνωστές επαυξήσεις φιλίας αλλά για διαφορετικά similarity thresholds (0.2 – πέρνα όσους είναι λίγο όμοιοι και επέτρεψε τους να είναι φίλοι αφού βρέθηκαν ως τέτοιοι, 0,5 – πέρνα όσοι είναι κατά το ήμισυ όμοιοι, 0,8 – πέρνα όσοι είναι μεταξύ τους πολύ όμοιοι).

Users	Items	Ratings	[	Ratings	/	Max Cells	Density%
2,073	100	2,524	[	2,524	/	207.300	1.218

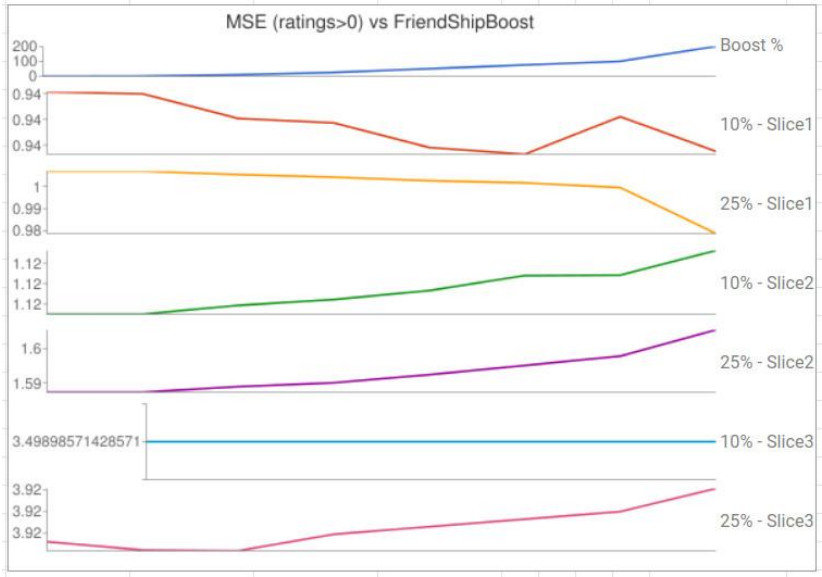
Split : 10% (Slice 1 , 1-253)				**Split : 25% (Slice1 , 1-632)**		
Simil. Threshold	**Simil. Threshold**	**Simil. Threshold**		**Simil. Threshold**	**Simil. Threshold**	**Simil. Threshold**
0.2	*0.5*	*0.8*		*0.2*	*0.5*	*0.8*

GraphId	**Dataset**	**Kind of Sampling**	**Type of Sampling**	**Items**	**Boost Characteristic**	**Boost Values (%)**
6	YelpKC	Item	Seq (of Mongo Random)	100	FriendShip	0,1,10,25,50,75,100,200

Πίνακας 7 - YelpKC(Seq Split 10% , Items: 100 Friendship Boost : 0,10,25,50,75,100,200%)



Εικόνα 28 - MSE vs FriendshipBoost (Items 100) Seq-MultiSlice – Όριο Ομοιότητας



Εικόνα 29 - MSE* vs FriendshipBoost (Items 100) Seq-MultiSlice – Όριο Ομοιότητας

Στα συγκεκριμένα αποτελέσματα παρατηρείται ότι μάλλον όταν επιτρέπουμε να λογίζονται ως πιο σημαντικοί χρήστες "φίλοι" που δεν έχουν και πολλά κοινά τότε αυτό είναι καλό για την απόδοση του συστήματος, ενώ όσο πιο ψηλά κόβουμε τους φίλους αν δεν είναι όμοιοι και δεν τους συμπεριφερόμαστε ως τέτοιους τότε χάνουμε σε απόδοση. Να σημειώσουμε και εδώ ότι το σύνολο δεδομένων είναι μικρό με τεράστια λάθη στην απόδοση (MSE>11) και άρα πιθανόν αυτές οι τάσεις να είναι και τυχαίες.

4.6.2.2.2 Όριο Ομοιότητας - Αντικείμενα : 1000

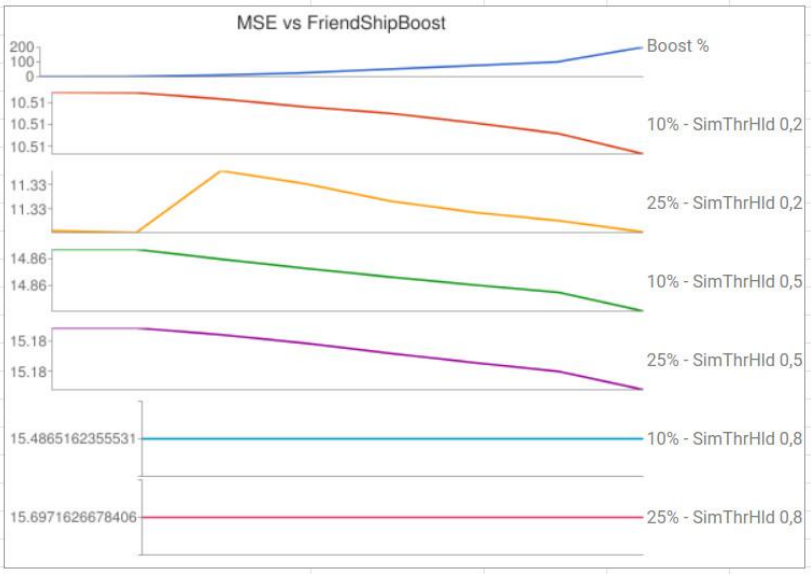
Στα συγκεκριμένα γραφήματα, σε συνέχεια του προηγούμενου, έχουμε δειγματικό χώρο με 1000 αντικείμενα.

Users	Items	Ratings	[	Ratings	/	Max Cells	Density %
9,827	1000	18,175	[	18,175	/	9,827,000	0.185

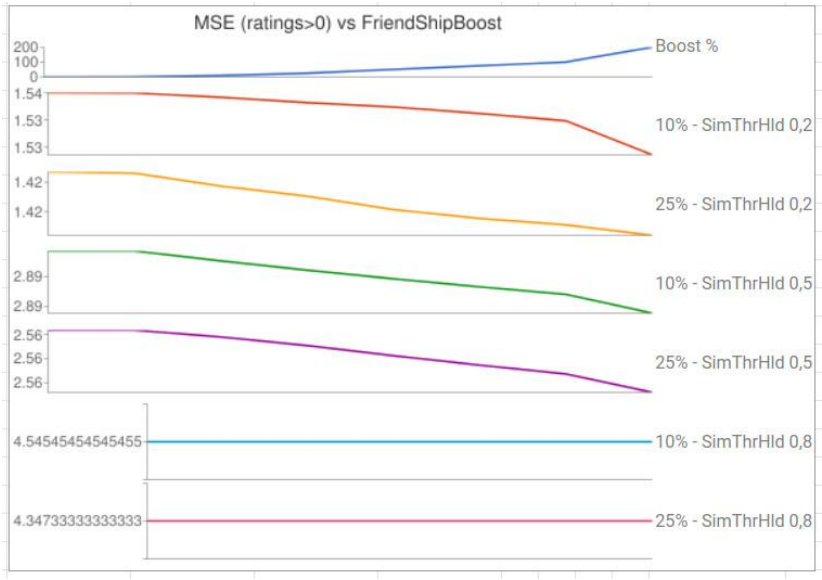
Split : 10% (Slice 1 , 1-253)				**Split : 25% (Slice1 , 1-632)**		
Simil. Threshold	**Simil. Threshold**	**Simil. Threshold**		**Simil. Threshold**	**Simil. Threshold**	**Simil. Threshold**
0.2	*0.5*	*0.8*		*0.2*	*0.5*	*0.8*

GraphId	**Dataset**	**Kind of Sampling**	**Type of Sampling**	**Items**	**Boost Characteristic**	**Boost Values (%)**
7	YelpKC	Item	Seq (of Mongo Random)	1000	FriendShip	0,1,10,25,50,75,100,200

Πίνακας 8 - YelpKC(Seq Split 10% , Items: 1000 Friendship Boost : 0,1,10,25,50,75,100,200%)



Εικόνα 30 - MSE vs FriendshipBoost (Items 1000) Seq-MultiSlice – Όριο Ομοιότητας



Εικόνα 31 – MSE* vs FriendshipBoost (Items 1000) Seq-MultiSlice – Όριο Ομοιότητας

Τα ευρήματα με τα 100 αντικείμενα, ως τάση, δείχνουν να επιβεβαιώνονται και σε αυτή την περίπτωση, με τον συντελεστή να θέλει να μείνει χαμηλά καθιστώντας αυτή την μεταβλητή ομοιότητας από κοινωνικά δεδομένα μάλλον όχι και τόσο χρήσιμη ή τουλάχιστον η χρήση της πρέπει να γίνεται σε χαμηλά όρια. Υπάρχει βέβαια η πιθανότητα μιας και παίζουμε με το 10% του δειγματικού χώρου (1000 αντικείμενα από τα 11.000) να μην έχει καταφέρει να απόδοση σωστά. Αυτό θα ήταν ένα πεδίο προς διερεύνηση στην περίπτωση που θα τρέχαμε το συγκεκριμένο χαρακτηριστικό σε μεγαλύτερο dataset.

4.6.2.3 Yelpkc–Σειρ. Τεστ 10%,25% Όμοιοι Χρήστες

Στο παρόν γίνεται διερεύνηση κατά πόσο το κατώφλι που αφορά στο πλήθος των όμοιων χρηστών έχει θετική ή όχι απόδοση στο σύστημα. Στα παρακάτω αποτελέσματα θα ελεγχθεί η απόδοση όταν επιτρέπονται μόνο οι 1, 3 ή 10 πιο ίδιο χρήστες να μπουν σε συνθήκες επαυξημένης ψήφου.

4.6.2.3.1 Κατώφλι όμοιων χρηστών - Αντικείμενα : 100

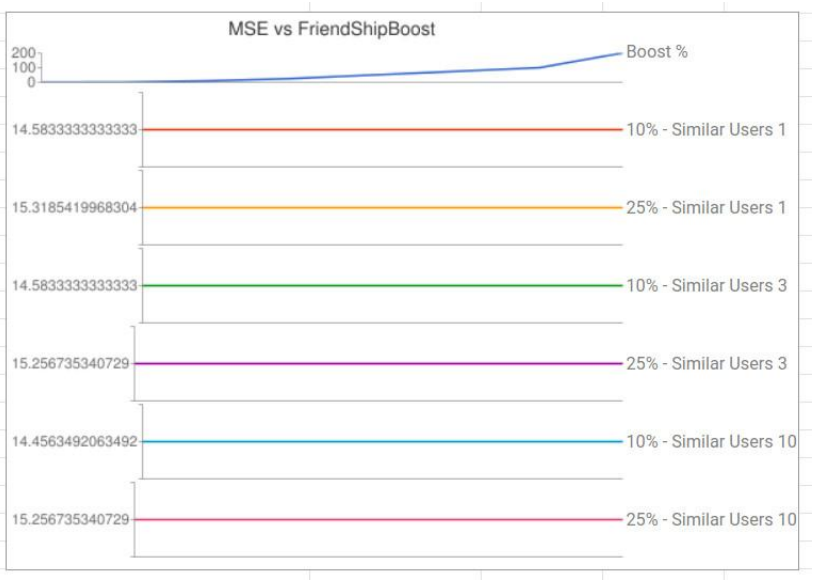
Ξεκινάμε όπως πάντα με 100 αντικείμενα σε 3 slices των 10 και 25% test set με σειριακή επιλογή.

Users	Items	Ratings	[	Ratings	/	Max Cells	Density%
2,073	100	2,524	[	2,524	/	207.300	1.218

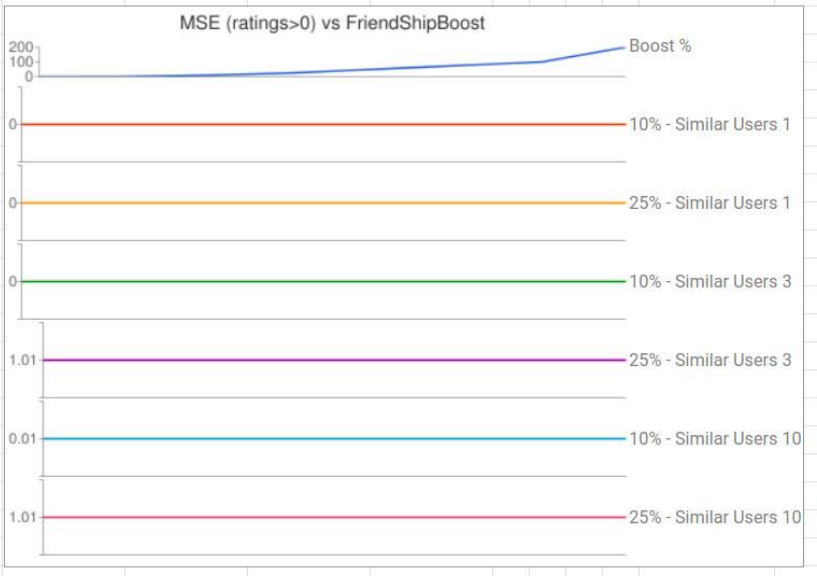
Split : 10% (Slice 1 , 1-253)				**Split : 25% (Slice1 , 1-632)**		
Similar Users	**Similar Users**	**Similar Users**		**Similar Users**	**Similar Users**	**Similar Users**
1	*3*	*10*		*1*	*3*	*10*

GraphId	Dataset	Kind of Sampling	Type of Sampling	Items	Boost Characteristic	Boost Values (%)
8	YelpKC	Item	Seq (of Mongo Random)	100	FriendShip	0,1,10,25,50,75,100,200

Πίνακας 9 - YelpKC(Seq Split 10% , Items: 100 ,Boost : 0-200%) – Όριο Όμοιων Χρήστων



Εικόνα 32 - MSE vs FriendshipBoost (Items 100) Seq-MultiSlice – Όριο Όμοιων Χρήστων



Εικόνα 33 - MSE* vs FriendshipBoost (Items 100) Seq-MultiSlice – Όριο Όμοιων Χρήστων

Στις παρούσες συνθήκες, με το πολύ μικρό δείγμα των 100 αντικείμενων, η συγκεκριμένη μεταβλητή φαίνεται να μην κάνει απολύτως τίποτα στην απόδοση του συστήματος, ενώ είναι εμφανές ότι η αστοχία του συστήματος είναι τρομερά μεγάλη για οποιαδήποτε εξαγωγή ασφαλούς συμπεράσματος (MSE > 14,5).

4.6.2.3.2 Κατώφλι όμοιων χρηστών - Αντικείμενα : 1000

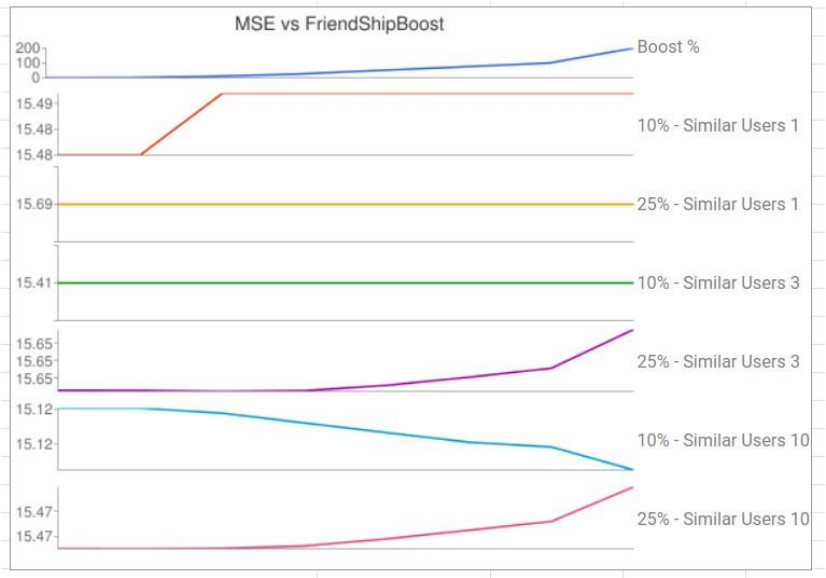
Συνεχίζουμε με 1000 αντικείμενα σε 3 slices των 10 και 25% test set με σειριακή επιλογή.

Users	Items	Ratings	[	Ratings	/	Max Cells	Density%
9,827	1000	18,175	[	18,175	/	9,827,000	0.185

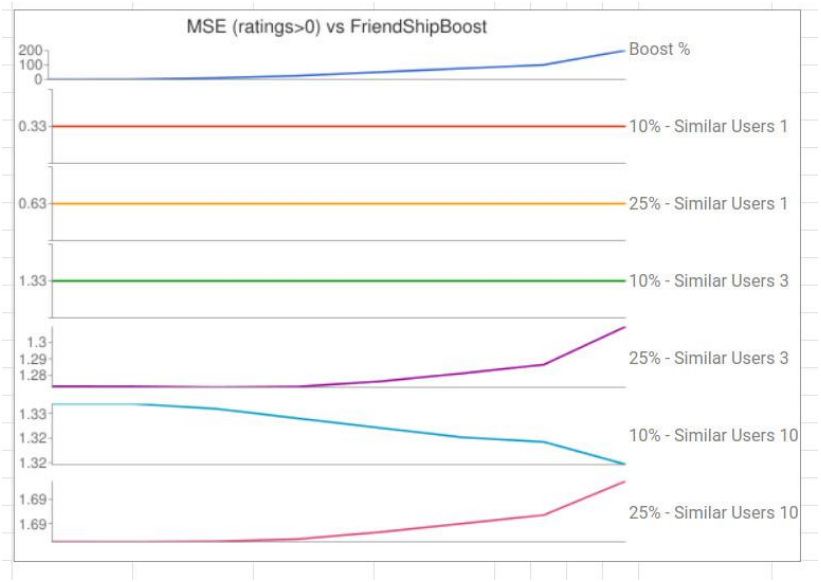
Split : 10% (Slice 1 , 1-253)				**Split : 25% (Slice1 , 1-632)**		
Similar Users	**Similar Users**	**Similar Users**		**Similar Users**	**Similar Users**	**Similar Users**
1	*3*	*10*		*1*	*3*	*10*

GraphId	**Dataset**	**Kind of Sampling**	**Type of Sampling**	**Items**	**Boost Characteristic**	**Boost Values (%)**
9	YelpKC	Item	Seq (of Mongo Random)	1000	FriendShip	0,1,10,25,50,75,100,200

Πίνακας 10 - YelpKC(Seq Split 10% , Items: 1000 ,Boost : 0-200%) – Όριο Όμοιων Χρήστων



Εικόνα 34 - MSE vs FriendshipBoost (Items 1000) Seq-MultiSlice – Όριο Όμοιων Χρήστων



Εικόνα 35 - MSE* vs FriendshipBoost (Items 1000) Seq-MultiSlice – Όριο Όμοιων Χρήστων

Είναι εμφανές και στα 1000 αντικείμενα ότι η συγκεκριμένη μεταβλητή δεν φαίνεται να κάνει την διαφορά στο κατώφλι λίγο χρηστών, ενώ φαίνεται να συνυφαίνει θετικά όταν κόβει μετά τους 10 χρήστες και με ποσοστό 200% και αυτό στο 10% test set. Αυτό πρακτικά σημαίνει ότι γενικότερα είναι προτιμότερο το 10% test για να μην χάνονται δεδομένα αφού παίζουμε με μόλις το 10% του δειγματικού χώρου. Επίσης σημαίνει ότι το 200% πάντα κάνει εμφανή μια τάση από τα μικρότερα ποσοστά (το έχουμε δει και σε προηγούμενες περιπτώσεις). Ενώ τέλος όταν το κατώφλι φτάνει τους 10 πιο όμοιους χρήστες, πρακτικά τους περνά όλους. Επομένως, είναι σαν να είναι άλλη μια μεταβλητή που συνεισφέρει από λίγο έως ελάχιστα στην απόδοση του συστήματος, τουλάχιστον μέχρι το dataset του 10% του αρχικού δηλαδή με τα 1000 αντικείμενα.

4.6.2.4 Yelpkc–Σειρ. Τεστ 10%,25% Όριο Ομοιότητας & πλήθους Όμοιων Χρηστών

Τα παρακάτω αποτελέσματα δείχνουν την αποτελεσματικότητα ή μη στην απόδοση του συστήματος σε περίπτωση συνδυαστικής χρήσης των παραπάνω χαρακτηριστικών “Κατώφλι ομοιότητας χρηστών” (Similarity Threshold) και “Κατώφλι πλήθους όμοιων χρηστών” (SimilarityThreshold). Τα παρακάτω αποτελέσματα θα γίνουν πάλι σε test set των 10% και 25% και για αντικείμενα 100 και 1000 και για το σύνολο των επαυξήσεων σε επίπεδο φιλίας.

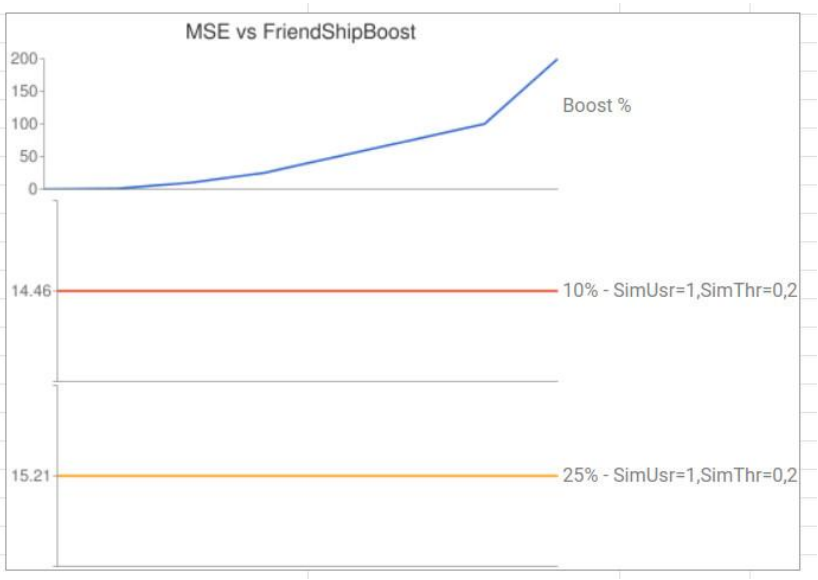
4.6.2.4.1 Όριο Ομοιότητας & Όριο πλήθους Όμοιων Χρηστών - Αντικείμενα : 100

Ξεκινάμε με 100 αντικείμενα σε test set των 10% και 25% και για το σύνολο των επαυξήσεων σε επίπεδο φιλίας.

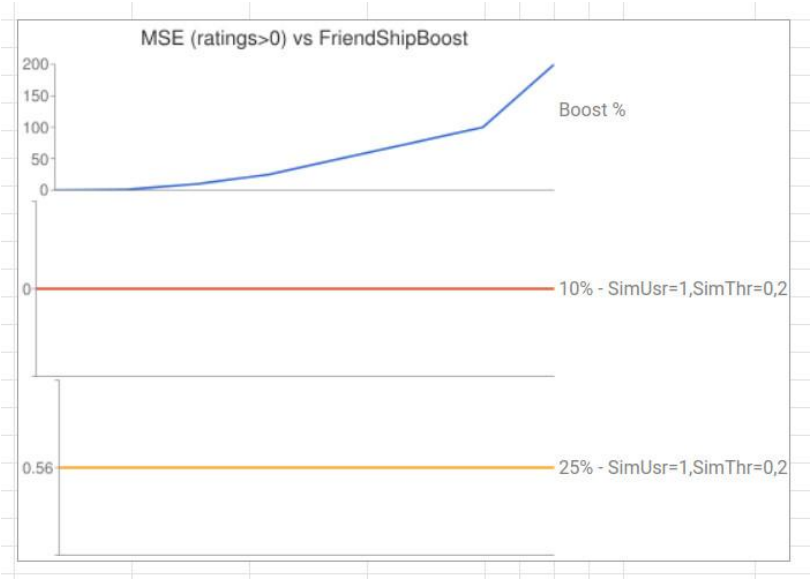
Users	Items	Ratings	[	Ratings	/	Max Cells	Density%
2,073	100	2,524	[	2,524	/	207.300	1.218

Split : 10% (Slice 1 , 1-253)				**Split : 25% (Slice1 , 1-632)**		
Similar Users =1 , Similarity Threshold = 0.2						
GraphiId	**Dataset**	**Kind of Sampling**	**Type of Sampling**	**Items**	**Boost Characteristic**	**Boost Values (%)**
10	YelpKC	Item	Seq (of Mongo Random)	100	FriendShip	0,1,10,25,50,75,100,200

Πίνακας 11 - YelpKC(Items: 100 ,Boost : 0-200%) - Όριο Ομοιότητας & Όμοιων Χρήστων



Εικόνα 36 - MSE vs Boost (Items 100) Seq-MultiSlice – Όριο Ομοιότητας&Όμοιων Χρήστων



Εικόνα 37 - MSE* vs Boost (Items 100) Seq-MultiSlice–Όριο Ομοιότητας&Όμοιων Χρήστων

Και στα δύο test set (10% και 25%) οι σύμπραξη των συγκεκριμένων μεταβλητών με όριο πλήθους χρηστών : 1 και ομοιότητα 0,2 δεν επειρεάζει καθόλου την απόδοση του συστήματος.

4.6.2.4.2 Όριο Ομοιότητας & Όριο πλήθους Όμοιων Χρηστών - Αντικείμενα : 1000

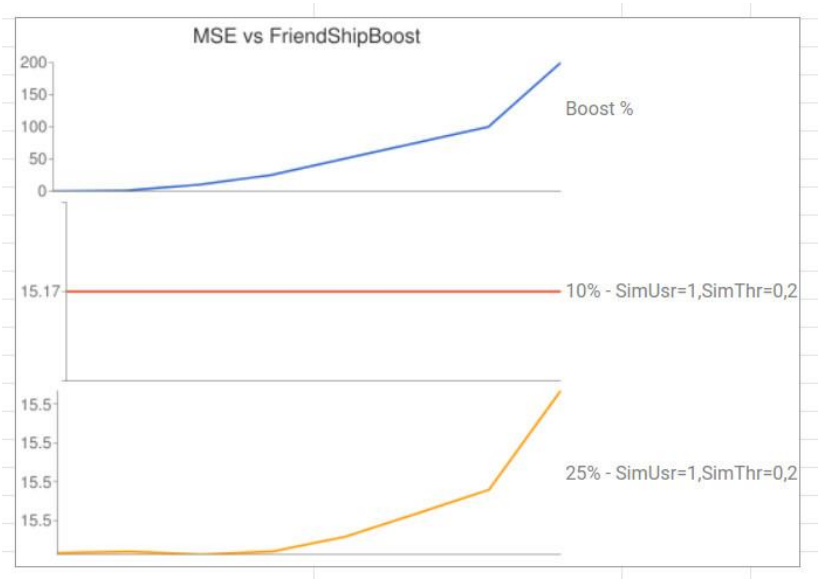
Στην συνέχεια ακολουθεί η περίπτωση των 1000 αντικείμενα σε test set των 10% και 25% για το σύνολο των επαυξήσεων σε επίπεδο φιλίας.

Users	Items	Ratings	[	Ratings	/	Max Cells	Density%
9,827	1000	18,175	[	18,175	/	9,827,000	0.185

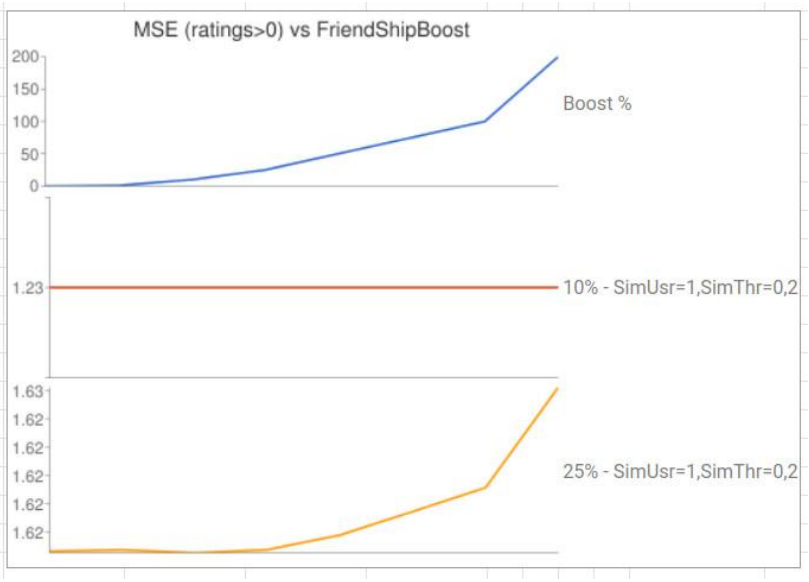
Split : 10% (Slice 1 , 1-253)				**Split : 25% (Slice1 , 1-632)**		
Similar Users	**Similar Users**	**Similar Users**		**Similar Users**	**Similar Users**	**Similar Users**
1	*3*	*10*		*1*	*3*	*10*

GraphiId	Dataset	Kind of Sampling	Type of Sampling	Items	Boost Characteristic	Boost Values (%)
11	YelpKC	Item	Seq (of Mongo Random)	1000	FriendShip	0,1,10,25,50,75,100,200

Πίνακας 12 - YelpKC(Items: 1000 ,Boost : 0-200%) - Όριο Ομοιότητας & Όμοιων Χρηστών



Εικόνα 38 - MSE vs Boost (Items 1000) Seq-MultiSlice – Όριο Ομοιότητας&Όμοιων Χρηστών

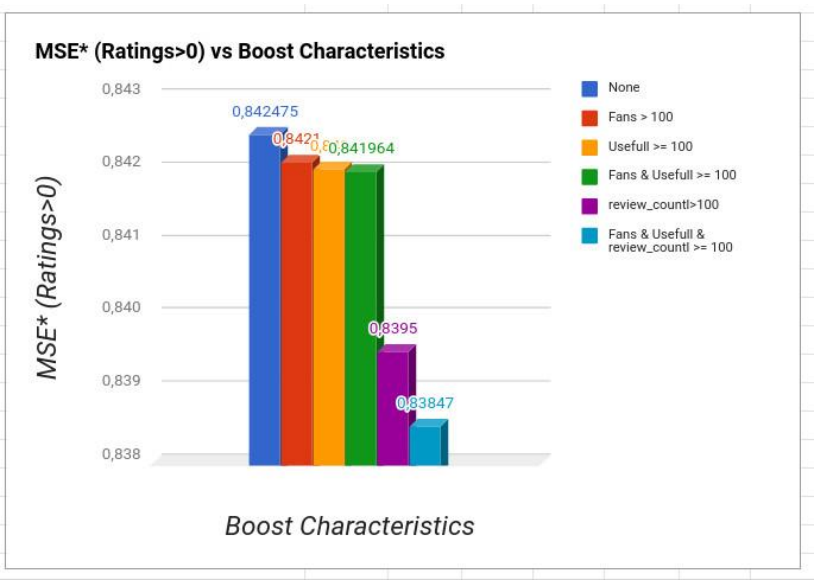


Εικόνα 39 - MSE* vs Boost (Items 1000) Seq-MultiSlice–Όριο Ομοιότητας&Όμοιων Χρηστών

Βλέπουμε και σε αυτή την περίπτωση στο δε 10% test set να μην επηρεάζει καθόλου την συμπεριφορά του συστήματος, ενώ στο 25% να την χειροτερεύει κιόλας. Να ξαναθυμίσουμε ότι και εδώ βρισκόμαστε σε πάρα πολύ κακή απόδοση του συστήματος συστάσεων λόγω του πολύ χαμηλού συνόλου δεδομένων (10% σε σχέση με το αρχικό – 1000 αντικείμενα).

4.6.2.5 Yelpkc–Σειρ. Τεστ - Κοινωνικά Χαρακ/κά (Fans , Usefull, Review Count)

Το YelpKC ως dataset, διαθέτει και κάποια επιπλέον χαρακτηριστικά τα οποία θα μπορούσαν να μελετηθούν ως προς την απόδοση που προσδίδουν σε ένα σύστημα συστάσεων με δεδομένα από κοινωνικά δίκτυα. Μερικά από αυτά, τα οποία και χρησιμοποιήθηκαν για τους σκοπούς της εργασίας, ήταν το πλήθος των ακολούθων (Fans), το πλήθος των ατόμων που θεωρούν χρήσιμες τις αξιολογήσεις ενός ατόμου (Usefull) καθώς επίσης και το πλήθος των αξιολογήσεων (ReviewCount). Παρακάτω παρουσιάζεται μια μεσοσταθμική απεικόνιση της συμπεριφοράς των παραπάνω κοινωνικών χαρακτηριστικών σε σχέση με την απόδοση (MSE*) που αυτά προσδίδουν στο σύστημα.



Εικόνα 40- MSE* vs Boost (Items 5000) Seq-MultiSlice – (Fans , Usefull, Review Count)

Όπως γίνεται καταφανές και από το παραπάνω γράφημα, το πλήθος των αξιολογήσεων ενός χρήστη φαίνεται να προσδίδει μεγαλύτερη απόδοση στο σύστημα, με την χρησιμότητα αυτών των αξιολογήσεων να ακολουθεί και τέλος το πλήθος των ακολούθων να έπεται μεν αλλά να συνεισφέρει και αυτό θετικά. Επιπλέον μπορούμε να δούμε ότι τόσο ο συνδυασμός των ακολούθων και των χρήσιμων συμβουλών τείνει να είναι ακόμη πιο θετικός από τον συνδυασμό τους αλλά λιγότερο σημαντικός από το πλήθος των αξιολογήσεων. Ενώ, ο συνδυασμός και των τριών αποδίδει ακόμη καλύτερα από όλα τα παραπάνω.

4.7 ΣΥΜΠΕΡΑΣΜΑΤΑ

Η χρήση δεδομένων από κοινωνικά δίκτυα και πιο συγκεκριμένα του συσχετισμού φιλίας μεταξύ χρηστών και ιδίως στο 200% επαύξησης κατέδειξε πιο καθαρά στο μισό δειγματικό χώρο των 5000 αντικειμένων, ότι το σύστημα είχε καλυτέρευση στην απόδοση του. Για το μεν πρώτο "Κατώφλι ομοιότητας χρηστών" (Similarity Threshold) αν ενεργοποιηθεί θα θεωρηθούν "φίλοι" μόνο εκείνοι που έχουν βαθμό ομοιότητας πάνω από το κατώτατο όριο (threshold) και το οποίο φαίνεται ότι θα πρέπει να το έχουμε σε χαμηλό επίπεδο. Μάλιστα, μέσα από τις διάφορες εκτελέσεις παρατηρήθηκε ότι αν περάσει πάνω από το 0.8 (80% similarity) έχει μάλλον κακές συνέπειες αφού χαμηλώνει την ποιότητα του συστήματος αφού μάλλον κόβει σημαντική πληροφορία έστω και από χρήστες που δεν είναι τόσο όμοιοι.

Από την άλλη το "Κατώφλι πλήθους όμοιων χρηστών" (SimilarityThreshold) επίσης μπορεί να μειώσει το πλήθος των όμοιων χρηστών επιστρέφοντας μόνο τους "NUMBER_OF_SIMILAR_USERS" πιο παρόμοιους χρήστες. Και σε αυτό παρατηρήθηκε ότι τιμές όπως 1 και 2 δεν έχουν καλή συμπεριφορά γιατί επίσης μειώνουν την πληροφορία από χρήστες που επίσης θα είχαν να προσφέρουν έστω και με χαμηλότερη ομοιότητα. Ο δε συνδυασμός τους μάλλον πρέπει να θεωρηθεί απαγορευτικός.

Τα χαρακτηριστικά που διαθέτει το ίδιο το Yelp Kaggle (fan, reviewCount, Usefull) φαίνονται ότι επίσης βοηθούν και μάλιστα αυξάνουν την απόδοση του συστήματος και συνδυαστικά.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

5.1 ΓΕΝΙΚΑ

Στα πλαίσια αυτής της διπλωματικής εργασίας τα συμπεράσματα είναι ότι τελικά τόσο το social friendship αλλά και τα άλλα τρία (fan,usefull,reviewCount) δείχνουν να βοηθάνε στην απόδοση του συστήματος. Αυτό σημαίνει ότι υπάρχει μια τάση ότι όσα περισσότερα ποιοτικά χαρακτηριστικά έχουμε για το σύστημα τόσο καλύτερα για την απόδοση του αφού δημιουργεί τους κατάλληλους συσχετισμούς πιο δυναμικά.

Σε ότι αφορά τα άλλα χαρακτηριστικά που ενεργοποιήσαμε δεν φαίνεται να βελτιστοποιούν στις περισσότερες των περιπτώσεων και ίσα ίσα μερικά χειροτερεύουνε την απόδοση του συστήματος, όπως για παράδειγμα τα "Κατώφλι ομοιότητας χρηστών" (Similarity Threshold) και του "Κατώφλι πλήθους όμοιων χρηστών" (SimilarityThreshold).

5.2 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Η μελλοντική εργασία που θα μπορούσε να γίνει θα αφορούσε καταρχάς περισσότερες εκτελέσεις, κυρίως πιο χρονοβόρες στον δειγματικό χώρο των 5000 αντικειμένων ή ακόμη και του συνόλου των 11.000, προκειμένου να καταδειχτούν με μεγαλύτερη ασφάλεια τα συμπεράσματα που σχετίζονται με τα χαρακτηριστικά και την απόδοση τους στο σύστημα.

Επιπλέον αναγκαία θα ήταν η γενικότερα την κλιμάκωση του συστήματος. Έτσι είναι προφανές ότι σε μεγαλύτερο scale ίσως απαιτηθεί μεγαλύτερη διαχείριση της μνήμης και τότε, είτε μια in memory database (π.χ. redis), είτε και σε συνδυασμό με μια κανονική μνήμη είτε με την δυνατότητα που δίνουν τα first και second level caching του hibernate να ήταν δυνατό το scale up.

Ακόμη μεγαλύτερη κλιμάκωση θα μπορούσε να γίνει αν η εφαρμογή έσπαγε σε πολλαπλούς κόμβους (clusters) και είχε ένα συντονιστή (orchestrator) ο οποίος θα αποθήκευε σε μια βάση όλα τα δεδομένα και θα έκοβε ισόποσα κομμάτια για τον κάθε κόμβο ώστε να εκτελεί, μέσω microservices για παράδειγμα, τις εργασίες και να επιστρέφει τα αποτελέσματα. Αυτό θα μπορούσε πραγματικά να κάνει το λεγόμενο scale out σε επίπεδο κόμβων που γεννιούνται κατ' επιλογή και όχι με περισσότερη

επεξεργαστική ισχύ για το εκάστοτε σύστημα (scale up) χωρίς να έχουμε να αντιμετωπίσουμε το limitation σε resources ενός συστήματος.

Επιπλέον, θα μπορούσε να είναι η περαιτέρω ανάπτυξη του API του με κατάλληλο Orchestrator module και υποδομή για διασύνδεση με γνωστά frameworks που σχετίζονται με κατανεμημένα συστήματα ώστε να είναι δυνατή η ταυτόχρονη εκτέλεση από πολλούς κόμβους (nodes) ακόμη πιο γρήγορα και χωρίς περιορισμούς σε μνήμη, επεξεργαστική ισχύ και σύνολο δεδομένων.

Τέλος, οι πολλαπλές εκτελέσεις και εισαγωγή των δεδομένων μαζί με τις διάφορες παραμέτρους σε ένα νευρωνικό δίκτυο, προκειμένου να εντοπιστεί το βέλτιστες ρυθμίσεις για τον σύστημα συστάσεων, θα ήταν επίσης άλλος ένας ενδιαφέρον τομέας στον οποίο πιθανόν να άξιζε κάποιος να ασχοληθεί και να επεκτείνει.

6. ΑΝΑΦΟΡΕΣ / ΒΙΒΛΙΟΓΡΑΦΙΑ

Οι παρακάτω αναφορές είναι τύπου "Plain Text Citation" όπως παρέχονται από το https://www.researchgate.net

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages.

[2] Leskovec, J & Rajaraman, A & Ullman, J.D.. (2014). Mining of massive datasets: Second edition. 10.1017/CBO9781139924801.

[3] Sun, Zhoubao & Han, Lixin & Huang, Wenliang & Wang, Xueting & Zeng, Xiaoqin & Wang, Min & Chong, Yan-Wen. (2014). Recommender Systems Based On Social Networks. Journal of Systems and Software. 99. 10.1016/j.jss.2014.09.019.

[4] Li, Ming & Dias, Malcolm & Jarman, Ian & El-Deredy, Wael & Lisboa, Paulo. (2009). Grocery shopping recommendations based on basket-sensitive random walk. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 1215-1224. 10.1145/1557019.1557150.

[5] Wiesner, Martin & Pfeifer, Daniel. (2014). Health Recommender Systems: Concepts, Requirements, Technical Basics and Challenges. International journal of environmental research and public health. 11. 2580-607. 10.3390/ijerph110302580.

7. ΠΑΡΑΡΤΗΜΑ

7.1 ΑΡΧΕΙΑ ΡΥΘΜΙΣΕΩΝ

Παρακάτω βρίσκονται μερικά από τα αρχεία ρυθμίσεων του συστήματος μέσω των οποίων μπορεί να αλλάξουν οι παράμετροι εκτέλεσης του.

7.1.1 Αρχείο Σταθερών Constants

Στο αρχείο αυτό τίθενται οι διάφορες σταθερές του συστήματος όπως για παράδειγμα τα csv suffix, τα Date_Seperators, τα Resource Base Path , DECIMAL_NUMBERS κ.α.

```
package gr.papazogloup.rs.impl.common;
/**
 * System's Constants
 */
@SuppressWarnings("javadoc")
public class Constants {
	/**
	 * Private constructor as it's not meant to be instantiated
	 */
	private Constants() {
		// EMPTY
	}
	public static final String  MONGODB_DATABASE_NAME="Recommendation";  // Recommendation , RecommendationFull //$NON-NLS-1$
	public static final boolean MONGODB_RESET_DB=false;
	public static final String CSV_SUFFIX = ".csv"; //$NON-NLS-1$
	public static final String NEW_LINE = "\n"; //$NON-NLS-1$
	public static final String DATE_SEPERATOR = "/"; //$NON-NLS-1$
	public static final String CSV_USERS_FRIENDS_SEPERATOR = ","; //$NON-NLS-1$
	public static final String EMPTY = ""; //$NON-NLS-1$
	//public static final String RESOURCE_BASE_PATH_SMALL = "src/main/resources/dataset/small/";

	/**
	 *
	 */
	public static final String RESOURCE_BASE_PATH_CUSTOM = "/mnt/development/0.panos/vcs/svn/msc2/recommendation/trunk/recommendation/rs-batch-rsrc/src/main/resources/dataset/movielens/custom/"; //$NON-NLS-1$
```

```
	public static final String RESOURCE_BASE_PATH_CUSTOM2 =
"/mnt/development/0.panos/vcs/svn/msc2/recommendation/trunk/recommendation/rs-batch-
rsrc/src/main/resources/dataset/movielens/custom2/"; //$NON-NLS-1$


	public static final String RESOURCE_BASE_PATH_SMALL =
"/mnt/development/0.panos/vcs/svn/msc2/recommendation/trunk/recommendation/rs-batch-
rsrc/src/main/resources/dataset/movielens/standard/"; //$NON-NLS-1$


	public static final String RESOURCE_BASE_PATH_CIAO =
"/mnt/development/0.panos/vcs/svn/msc2/recommendation/trunk/recommendation/rs-batch-
rsrc/src/main/resources/dataset/ciao/standard/";	//$NON-NLS-1$


	public static final String RESOURCE_BASE_PATH_YELP =
"/mnt/development/0.panos/vcs/svn/msc2/recommendation/trunk/recommendation/rs-batch-
rsrc/src/main/resources/dataset/yelp/standard/"; //$NON-NLS-1$


	public static final String RESOURCE_BASE_PATH_YELPKC =
"/mnt/development/0.panos/vcs/svn/msc2/recommendation/trunk/recommendation/rs-batch-
rsrc/src/main/resources/dataset/yelp/challenge/";	//$NON-NLS-1$


	public static final String RESOURCE_BASE_PATH = RESOURCE_BASE_PATH_YELPKC;


	/**
	 * Default recommendation batch user
	 */
	public static final String RSBAT = "RSBAT";  //$NON-NLS-1$


	/**
	 * How many decimal numbers we want the calculating to offer
	 */
	public static final int DECIMAL_NUMBERS = 2;


	/**
	 * Progress indicator master counter of all the ratings that will be calculated
	 */
	public static long masterCounter = 0;


	/**
	 * Progress indicator temp counter limit of ratings before check of average and ratingsPerSec indigator
	 * Doesn't seem to impact much between 100.000 and 1.000.000
	 * so its more prefered the 100.000 so we can have more frequent monitoring status.
	 */
	public static long tempCounterLimit = 1000000;


	/**
```

```
	 * Progress indicator starting time
	 */
	public static Long startTime = null;

	/**
	 * Progress indicator  temp counter for the ratings
	 */
	public static long tempCounter = 0;

	/**
	 * Calculated ratings to be filled
	 */

	public static Long ratingsToBeCalculated = null;

	/**
	 * Minimum rate value
	 */
	public static Float MIN_RATE_VALUE = 0f;

	/**
	 * Maximum rate value
	 */
	public static Float MAX_RATE_VALUE = 5f;
}
```

7.1.2 Αρχείο Ρυθμίσεων Configuration

Στο αρχείο αυτό τίθενται οι διάφορες ρυθμίσεις του συστήματος όπως για παράδειγμα τα "NUMBER_OF_THREADS" , "NUMBER_OF_SIMILAR_USERS" , "USE_ONLY_ORIG_RATINGS" , "SIMILARITY_THRESHOLD_VALUE" , "RANDOM_SPLIT","PERCENTAGE_SPLIT","FRIENDSHIP_BOOST_STEPS_VALUES" , "AVERAGE_STARS_SIMILARITY_BOOST_VALUE" κ.α.

```
package gr.papazogloup.rs.impl.engine;
import gr.papazogloup.rs.def.engine.collaborative.core.bl.op.FillUserItemMissingRatingOperation;
import gr.papazogloup.rs.def.engine.utilitymatrix.beans.UtilityMatrix;
/**
```

```
 * System's configurations
 */
public interface Configuration {
		/**
		 * Number of threads to run the recommender computations {@link FillUserItemMissingRatingOperation}
		 */
		public static final Integer NUMBER_OF_THREADS=1;


		/**
		 * Indicator if the algorithm should set under consideration the number of similar user
		 */
		public static final Boolean SHOULD_SET_NUMBER_OF_SIMILAR_USERS = Boolean.FALSE;


		/**
		 * Number of similar users  x 10
		 *
		 * 1 ,  3  ,  10
		 */
		public static final Integer NUMBER_OF_SIMILAR_USERS = 10;

		/**
		 * Indicator if the algorithm should set under consideration the similarity threshold value
		 */
		public static final Boolean SHOULD_SET_SIMILARITY_THRESHOLD_VALUE = Boolean.FALSE;


		/**
		 * Similarity threshold value x 10b 0.9 overffiting
		 * 0.2   0.5   0.8
		 */
		public static final Double SIMILARITY_THRESHOLD_VALUE = 0.2;


		/**
		 * This indigator stands for the explicity usage of the given users' ratings.
		 * TRUE: The system considers only the original ratings of the users
		 * FALSE: The system considers both the original and the calculated ratings of the user.
		 */
		public static final Boolean USE_ONLY_ORIG_RATINGS = Boolean.FALSE;


		/**
		 * Miration Mode Switch
```

```
 *
 * * KCUid : Kaggle Challedge uid , less sparce but no friendship association
 * * Yuid : Yale uid , sparce but with friendship association
 *
 * TRUE : Read user_mig.csv
 *                          Create KCuid , Yui mapping
 *                          Update ratings.uid to Yuid (using above) map , during ratings dataset process.
 */
public static final Boolean MIGRATION = Boolean.TRUE;


/**
 * Indicator if we need a random generated list of {@link Integer} numbers indicators
 * or a sequential one for the training of {@link UtilityMatrix}.
 *
 * <ul>
 *   <li>TRUE : Random generated List of {@link Integer} numbers</li>
 *   <li>FALSE : Sequential generated List of {@link Integer} numbers</li>
 * </ul>
 */
public static final Boolean RANDOM_SPLIT = Boolean.FALSE;


/**
 * Split percentage indicates the percentage of the dataset that will cut down and it's rating become null
 * in order to evaluate systems accuracy
 */
public static final Integer PERCENTAGE_SPLIT = 10;


/**
 * The base offset for the creation of the sequential split {@link Integer} numbered indexes
 * for the training of {@link UtilityMatrix}.
 * <p>
 * SPLIT : 10%<br>
 * <ul>
 * <li> 100: [1- 253] , [ 1262- 1514] , [ 2270- 2522]
 * <li>1000: [1-1819] , [ 9087-10905] , [16357-18173]
 * <li>5000: [1-9760] , [48798-58558] , [87834-97594]
 * </p>
 *
 * <p>
 * SPLIT : 25%<br>
 * <ul>
```

```
 * <li> 100: [1-  632] , [ 1262- 1894] , [ 1885- 2517]
 * <li>1000: [1- 4543] , [ 6815-11358] , [13630-18173] ***
 * <li>5000: [1-24400] , [36600-61000] , [73190-97590]
 * </p>       *
 */
//public static final Integer[] SEQUENTIAL_SPLIT_OFFSETS = new Integer[] {0,6815,13630};
public static final Integer[] SEQUENTIAL_SPLIT_OFFSETS = new Integer[] {0};


/**
 * Friendship similarity boost percentage steps values.
 * <p>
 * Similarity Boost percentage into a friends weight/
 * <br><br>
 * Example :
 * <ul>
 * <li>Friend's 'A' Initial Similarity Weight : 0.7</li>
 * <li>friendSimilarityBoost : 30   (30%)</li>
 * </ul>
 * Final friend's 'A' similarity weight : 0.91 [0.7 * (30/100)]
 * </p>
 *
 */
//public static final Integer[] FRIENDSHIP_BOOST_STEPS_VALUES = new Integer[] {0,1,10,25,50,75,100,200};
public static final Integer[] FRIENDSHIP_BOOST_STEPS_VALUES = new Integer[] {50};

/**
 * Average stars friendship boost
 */
public static final Integer AVERAGE_STARS_SIMILARITY_BOOST_VALUE = 0;

/**
 * Average stars friendship boost friends only flag.
 * TRUE: Boost only for friends
 * FALSE : Boost for everyone
 */
public static final Boolean AVERAGE_STARS_SIMILARITY_BOOST_FRIENDS_ONLY = Boolean.TRUE;


/**
 * The Useful threshold defines the value equal or above of which the
 * system will use the below Configuration#USEFULL_FRIENDSHIP_BOOST_VALUE
 * to boost the weight of the similar user's similarity for his contribution
```

```
 * in the rate calculation.
 * If we don't want this calculation to happen then simply set this
 * constants to -1
 */
//public static final Integer USEFUL_SOCIAL_THRESHOLD = 100;   // 0.8420 vs 0.8424 better!!
public static final Integer USEFUL_SOCIAL_THRESHOLD = -1;


/**
 * Useful social boost
 */
public static final Integer USEFUL_SOCIAL_BOOST_VALUE = 50;


/**
 * Useful social boost friends only flag.
 * TRUE: Boost only for friends that are beyond threshold
 * FALSE : Boost for everyone that are beyond threshold
 */
public static final Boolean USEFUL_SOCIAL_BOOST_FRIENDS_ONLY = Boolean.TRUE;


/**
 * The Fans social threshold defines the value equal or above of which the
 * system will use the below Configuration#FANS_FRIENDSHIP_BOOST_VALUE
 * to boost the weight of the similar user's similarity for his contribution
 * in the rate calculation.
 * If we don't want this calculation to happen then simply set this
 * constants to -1
 */                                     //0.841964 vs 0.8424 with above open Even better!!
//public static final Integer FANS_SOCIAL_THRESHOLD = 100;  //0.8421 vs 0.8424 better!!
public static final Integer FANS_SOCIAL_THRESHOLD = -1;    //0.8421 vs 0.8424 better!!


/**
 * Fans social boost value.
 */
public static final Integer FANS_SOCIAL_BOOST_VALUE = 50;


/**
 * Fans social boost friends only flag.
 * TRUE: Boost only for friends that are beyond threshold
 * FALSE : Boost for everyone that are beyond threshold
 */
public static final Boolean FANS_SOCIAL_BOOST_FRIENDS_ONLY = Boolean.TRUE;
```

```
/**
 * The Review count social threshold defines the value equal or above of which the
 * system will use the below Configuration#REVIEW_COUNT_FRIENDSHIP_BOOST_VALUE
 * to boost the weight of the similar user's similarity for his contribution
 * in the rate calculation.
 * If we don't want this calculation to happen then simply set this
 * constants to -1
 */                                          // 0.83847 vs 0.842475 all two above open even even better!!
//public static final Integer REVIEW_COUNT_SOCIAL_THRESHOLD = 100; // 0.8395 vs 0.842475
public static final Integer REVIEW_COUNT_SOCIAL_THRESHOLD = -1;   // 0.8395 vs 0.842475


/**
 * Review count social boos value.
 */
public static final Integer REVIEW_COUNT_SOCIAL_BOOST_VALUE = 50;


/**
 * Review count social boost friends only flag.
 * TRUE: Boost only for friends that are beyond threshold
 * FALSE : Boost for everyone that are beyond threshold
 */
public static final Boolean REVIEW_COUNT_SOCIAL_BOOST_FRIENDS_ONLY = Boolean.TRUE;


/**
 * Indicator if should display the splited (test set) cells of the {@link UtilityMatrix}
 * at the evaluation process
 */
public static final Boolean SHOW_UTILITY_MATRIX_EVALUATION_CELLS = Boolean.FALSE;



/**
 *  How many times it has to cut the dataset in the {@link Configuration#PERCENTAGE_SPLIT}
 *  equals parts to run the process to see the accuracy deviation.
 */
public static final Integer CROSS_VALIDATION_FOLD = 1;


/**
 * The class of the rating proposer to run against
 */
```

```
		//public static final String RATE_PROPOSED_CLASS="gr.papazogloup.rs.impl.engine.collaborative.core.rating.proposer.bl.op.WeightedAverageRateProposerOperationImpl";

		//public static final String RATE_PROPOSED_CLASS="gr.papazogloup.rs.impl.engine.collaborative.core.rating.proposer.bl.op.SocialFriendshipRateProposerOperationImpl"; //$NON-NLS-1$

		public static final String RATE_PROPOSED_CLASS="gr.papazogloup.rs.impl.engine.collaborative.core.rating.proposer.bl.op.MultiFieldsRateProposerOperationImpl"; //$NON-NLS-1$


		/**
		 * If yes then the utility matrix csv will be created as a csv log file
		 */
		public static final Boolean CREATE_UTILITY_MATRIX_CSV = Boolean.TRUE;


		/**
		 * If yes then the original utility matrix will be exported in the same csv
		 * as the calculated as well in order to be able to check the diferrences
		 */
		public static final Boolean EXPORT_UTILITY_MATRIX_ORIG_ON_TOP_OF_CSV = Boolean.TRUE;


		/**
		 * Number of maximum returned recomendations
		 */
		public static final Integer NUMBER_OF_MAXIMUM_RETURNED_RECOMMENDATIONS = 4;


		/**
		 * UserId of the user that we want to see system's recommendation
		 */
		public static final String CHECK_RECOMMENDATION_USER_ID = "US72XuETHCnMS0lItZGzMA"; //$NON-NLS-1$
}
```

7.1.3 Αρχείο Ρυθμίσεων Καταγραφής (log4j.properties)

```
# Set everything to be logged to the console
rs.log.dir=../logs
rs.log.file=recommender.log
rs.log.UtilityMatrix.file=utilityMatrix.log.csv
log4j.rootLogger=INFO,console,FILE
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n
```

```
log4j.appender.console.threshold=INFO
log4j.appender.FILE=org.apache.log4j.RollingFileAppender
log4j.appender.FILE.Append=true
log4j.appender.FILE.MaxFileSize=10MB
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n
log4j.appender.FILE.File=${rs.log.dir}/${rs.log.file}
log4j.logger.gr.papazogloup.rs.impl.engine.collaborative.export.bl.op.ExportUtilityMatrixCsvLogFileOperationImpl=INFO,
UTILITYMATRIX
log4j.appender.UTILITYMATRIX=org.apache.log4j.RollingFileAppender
log4j.appender.UTILITYMATRIX.Append=false
log4j.appender.UTILITYMATRIX.MaxFileSize=100MB
log4j.appender.UTILITYMATRIX.layout=org.apache.log4j.PatternLayout
log4j.appender.UTILITYMATRIX.File=${rs.log.dir}/${rs.log.UtilityMatrix.file}
log4j.additivity.gr.papazogloup.rs.impl.engine.collaborative.export.bl.op.ExportUtilityMatrixCsvLogFileOperationImpl=false
```

7.2 ΕΝΤΟΛΕΣ MONGO ΓΙΑ SAMPLING SUBSET

Στο παρακάτω αρχείο αποτυπώνονται οι εντολές μέσω των οποίον ήταν δυνατό να δημιουργηθεί ένα τυχαίο υποσύνολο μέσω των δύο merged βάσεων και βάση του πλήθους των items που επιθυμούσαμε ώστε να είναι δυνατή η εκτέλεση πιο γρήγορα σε μικρότερα subset σε σχέση με το σύνολο του.

```
[0]
> mongo
> show dbs;
> use Yelp;
> show collections;
> DBQuery.shellBatchSize = 10
[1]
Import Dataset to MongoDB
-----------------------------------------
mongoimport --db Yelp --collection user --file yelp_academic_dataset_user.json              [1.518.169]
mongoimport --db Yelp --collection business --file yelp_academic_dataset_business.json    [   188.593]
mongoimport --db Yelp --collection review --file yelp_academic_dataset_review.json         [5.996.996]
mongoimport --db Yelp --collection tip --file yelp_academic_dataset_tip.json               [1.185.348]
mongoimport --db Yelp --collection photo --file yelp_academic_dataset_photo.json            [   280.992]
mongoimport --db Yelp --collection checkin --file yelp_academic_dataset_checkin.json       [   157.075]
 [1,5]
Create Indexes
--------------
```

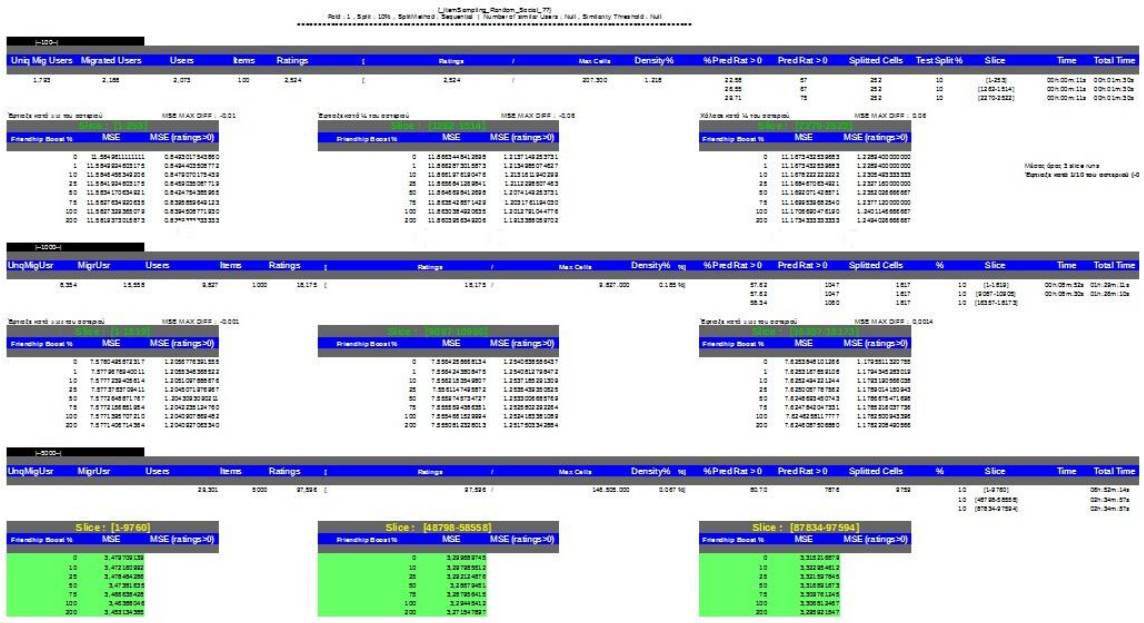
```
db.businessStandard.ensureIndex({ "business_id":1 });
db.review.ensureIndex({ "review_id":1,"user_id": 1 , "business_id": 1  });
db.user.ensureIndex({ "user_id":1 });
 [2]
Business Subset (Random sample of 1500 businesses
------------------------------------------------------------------------
db.business.aggregate([
{$sample: { size: 100 } },
{$out: "businessStandard100"}
]);
db.businessStandard100.ensureIndex({ "business_id":1 });
db.businessStandard100.find().length();
 [3]
Review Subset
----------------------
db.review.aggregate([
{ "$lookup": {
  "localField": "business_id",
  "from": "businessStandard100",
  "foreignField": "business_id",
  "as": "businessinfo"
} },
  { "$unwind": "$businessinfo" },
  { "$project": {
  "review_id":1,
  "user_id":1,
  "business_id":1,
  "stars":1,
  "date":1,
  "text":1,
  "useful":1,
  "funny":1,
  "cool":1
  } }, {$out: "reviewStandard100"}

]);
> db.reviewStandard100.distinct('user_id').length
 [4]
User Subset .... manual...distict user_id.. ??  (den einai sosto ...)
------------------------------------------------------------------------
db.reviewStandard.aggregate([
```

```
{ "$lookup": {
  "localField": "user_id",
   "from": "user",
   "foreignField": "user_id",
   "as": "userinfo"
} },
  { "$unwind": "$userinfo" },
{'$group': {
   '_id': '$from._id'
}},
{$out: "userStandard"}
]);
[*]
Counts..
----------
db.reviewStandard100.distinct('user_id').length;
db.businessStandard100.find().length();
db.reviewStandard100.find().length();
[5]
Export csv - specific fields
---------------------------------------
mongoexport --host localhost --db Yelp --collection businessStandard100 --csv --out items.csv --fields business_id,name
mongoexport --host localhost --db Yelp --collection reviewStandard100 --csv --out ratings.csv --fields user_id,business_id,stars
mongoexport --host localhost --db Yelp --collection businessStandard100 --csv --out items.csv --fields
_id,business_id,name,stars,review_count
mongoexport --host localhost --db Yelp --collection reviewStandard100 --csv --out ratings.csv --fields
_id,review_id,user_id,business_id,stars,date,useful
[A]
Find matching userId-ies
-------------------------------------
db.user.aggregate([
{$match: {  'user_id': {  $in: ["s4FoIXE_LSGviTHBe8dmcg","SgYDjNCecPidsRB_su5-tw","lzlZwIpuSWXEnNS91wxjHw"]  } }},
{$out: "userTiny"}
]);
```

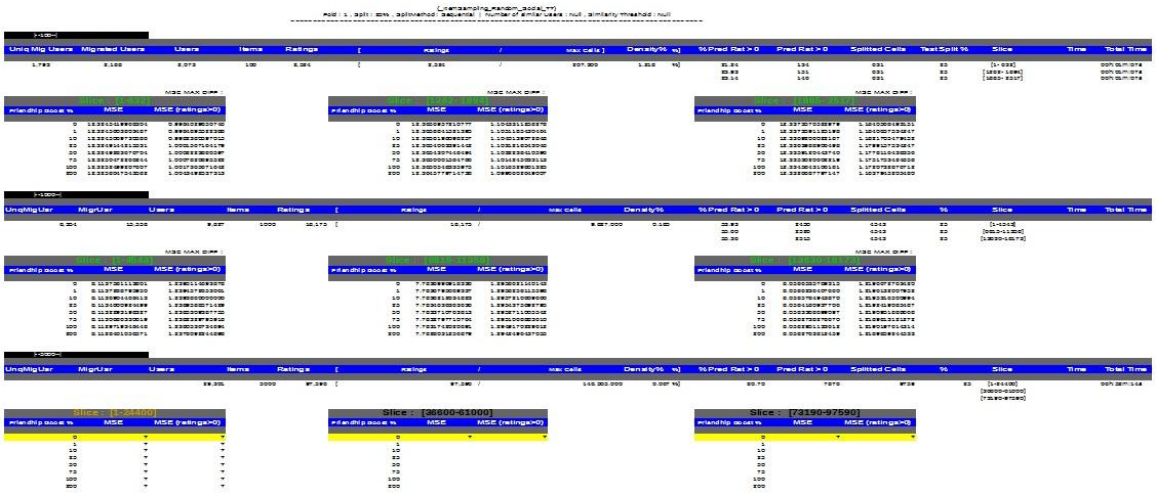
7.3 ΤΙΜΕΣ ΑΠΟ ΔΙΑΦΟΡΑ ΤΡΕΞΙΜΑΤΑ

Στα παρακάτω αρχεία βρίσκονται οι τιμές (raw data) που έχουν παραχθεί από το σύστημα για κάποια από τα τρεξίματα που έχουν παρουσιαστεί στο 4ο κεφάλαιο μέσω γραφημάτων. Πρόκειται δειγματοληπτικά για κάποια από τα MSE και MSE (ratings>0) έχουν αναφερθεί.

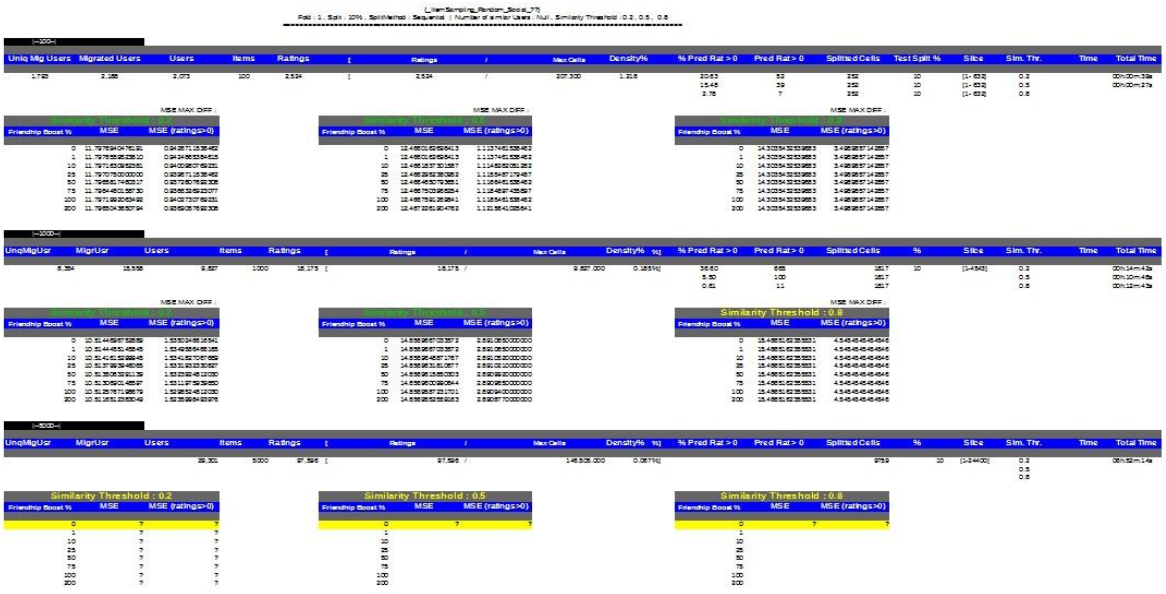
7.3.1 KC_ItemSampling_Random_Social_FriendShip(Seq-Spli_10%)



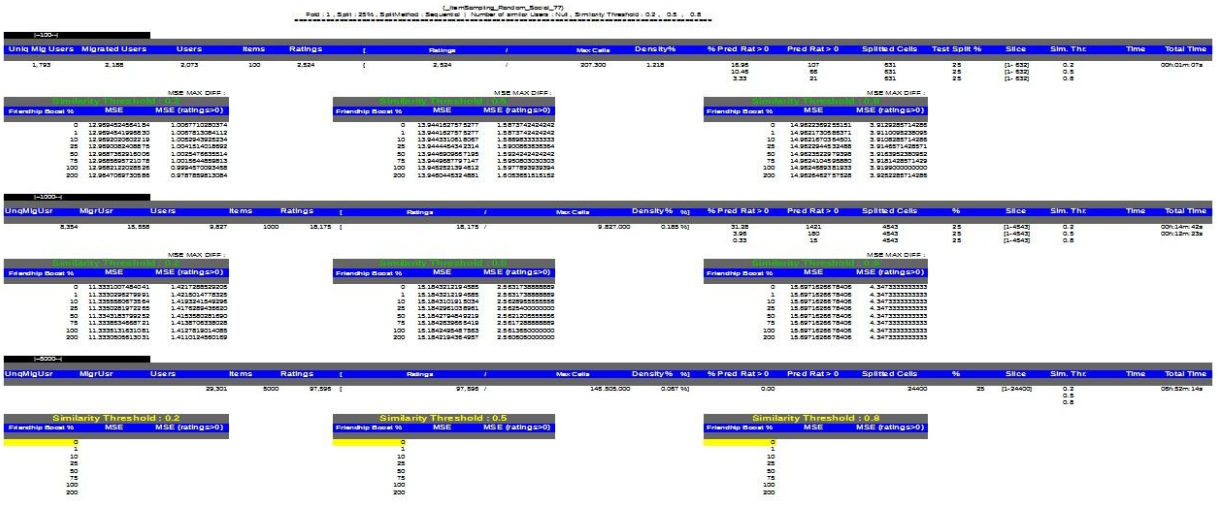
7.3.2 KC_ItemSampling_Random_Social_FriendShip(Seq-Spli_25%)



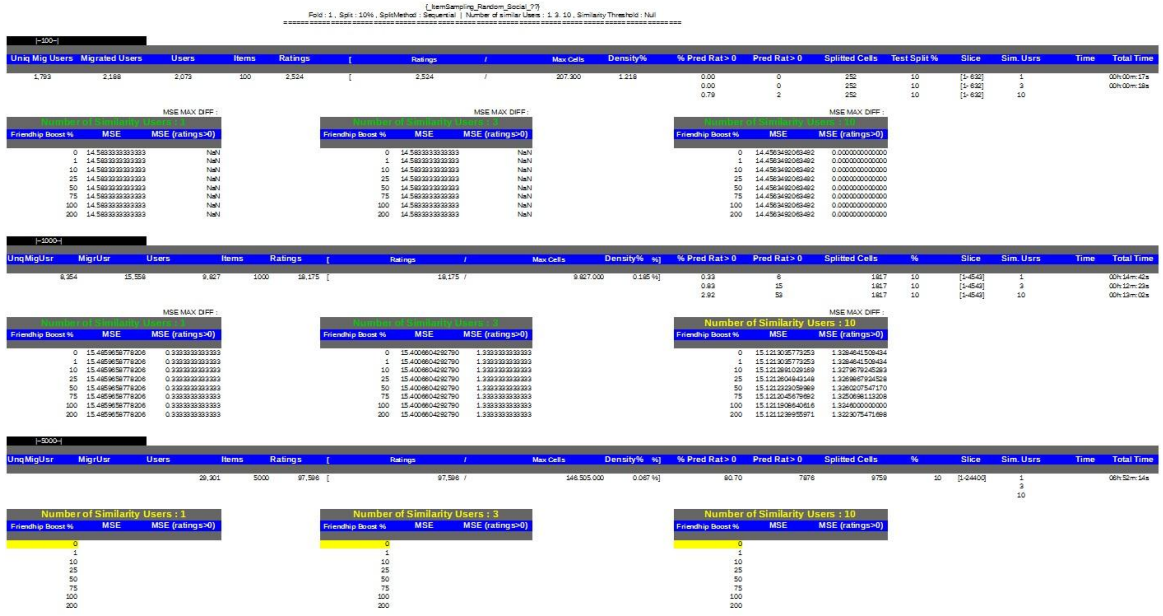
7.3.3 KC_ItemSampling_Random_Social_FriendShip(Seq-Spli_10%_SimThrHld_0.N)



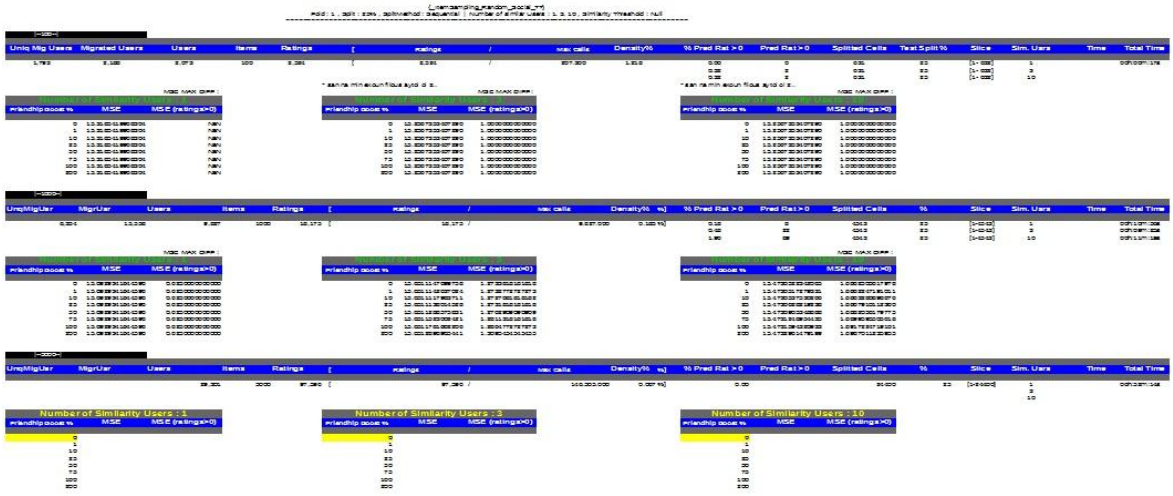
7.3.4 KC_ItemSampling_Random_Social_FriendShip(Seq-Spli_25%_SimThrHld_0.N)



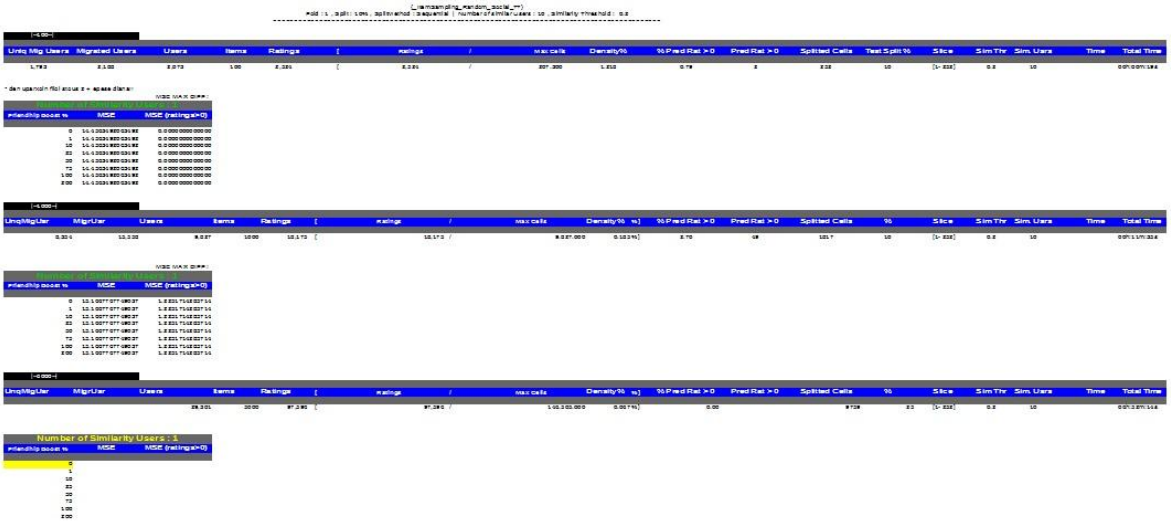
7.3.5 KC_ItemSampling_Random_Social_FriendShip(Seq-Spli_10%_SimUsrs_N)



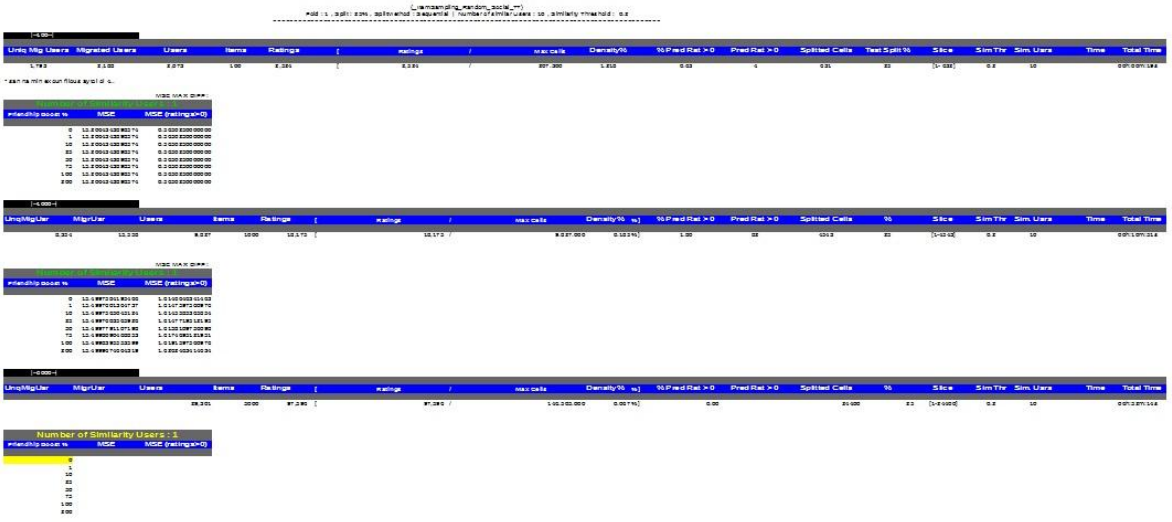
7.3.6 KC_ItemSampling_Random_Social_FriendShip(Seq-Spli_25%_SimUsrs_N)



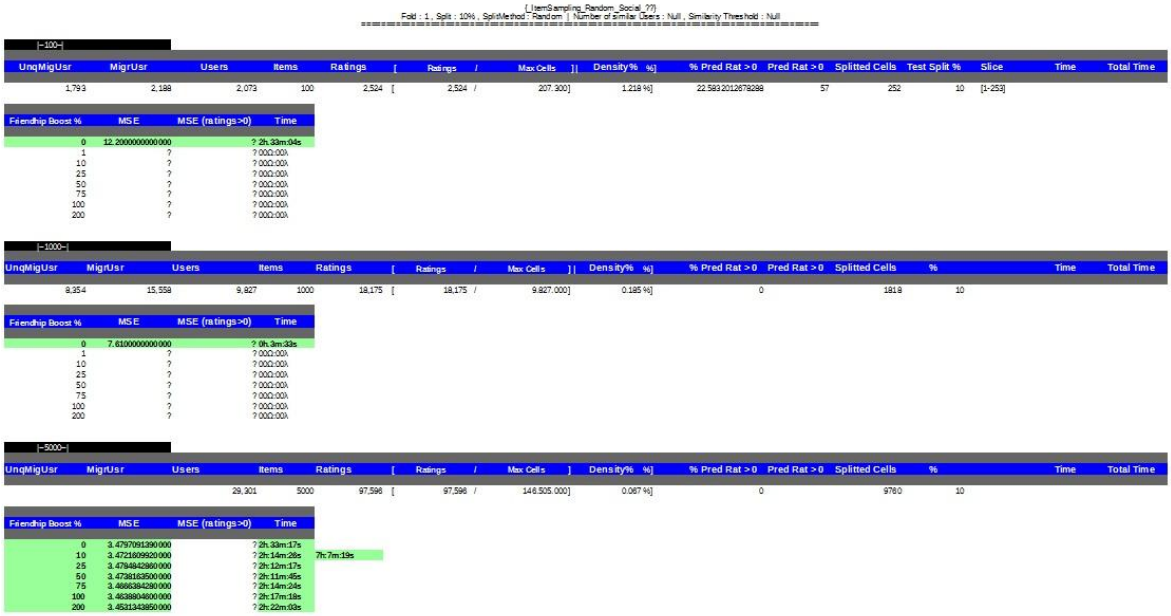
7.3.7 KC_ItemSampling_Random_Social_FriendShip(Seq-Spli_10%_SimThrHld_0.2_SimUsrs_10)



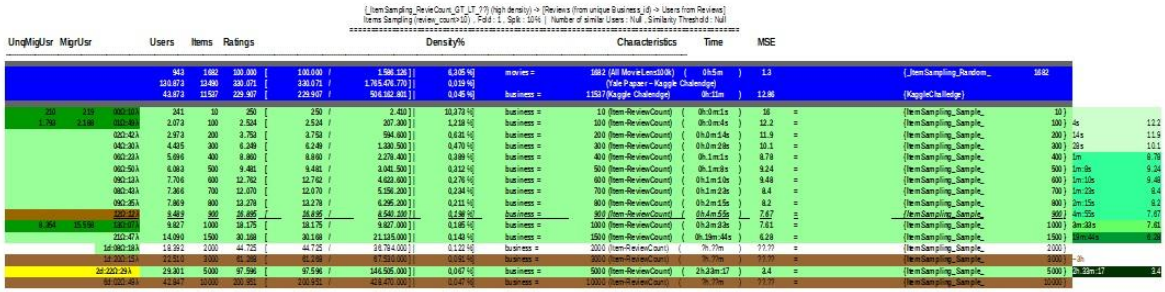
7.3.8 KC_ItemSampling_Random_Social_FriendShip(Seq-Spli_25%_SimThrHld_0.2_SimUsrs_10)



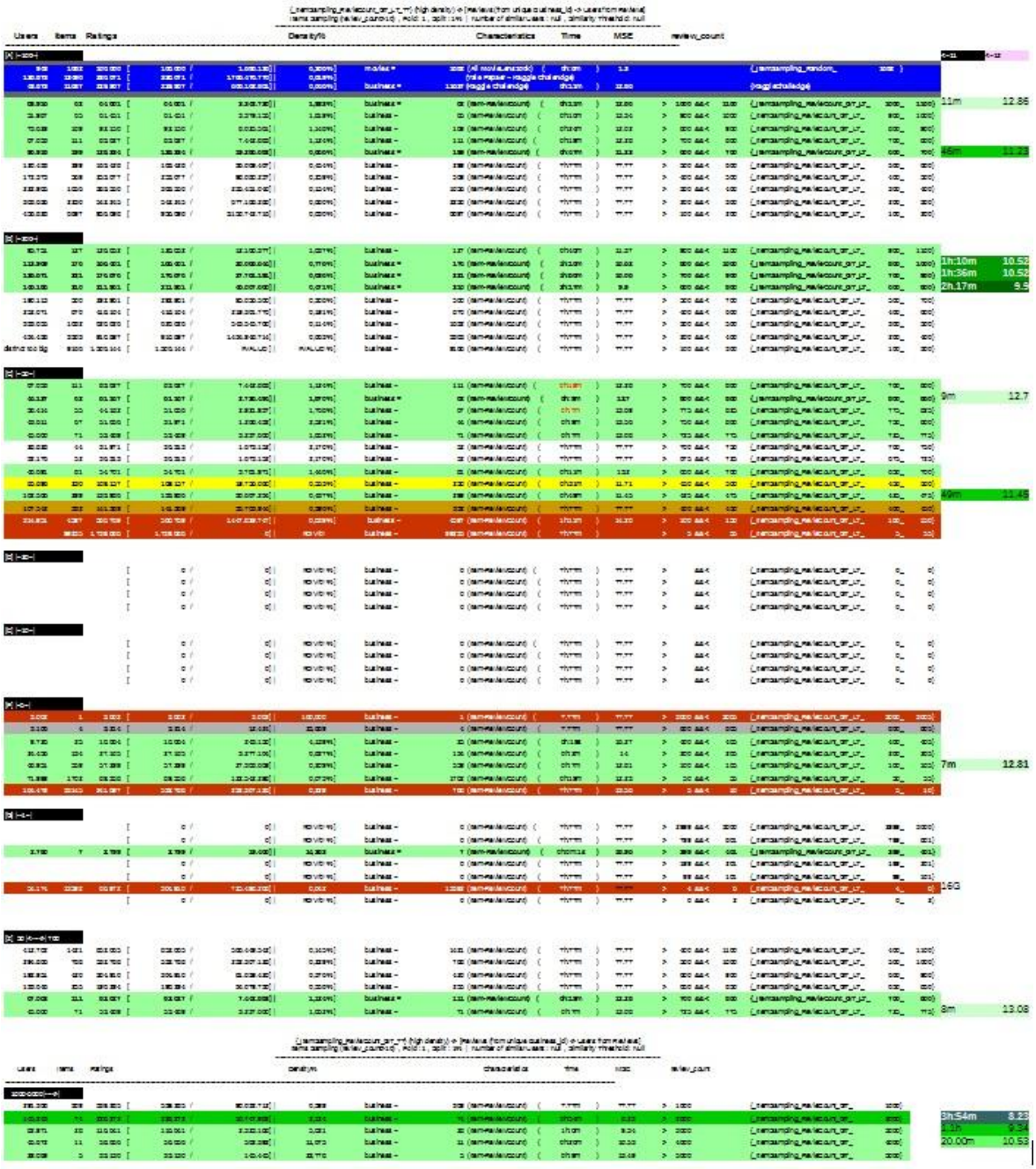
7.3.9 KC_ItemSampling_Random_Social_FriendShip (Rand-Spli _10%)



7.3.10 KC_ItemSampling_Random_Split_10%



7.3.11 ItemSampling_RevieCount_GT_LT



7.3.12 ItemSampling_Random