ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. «Πληροφοριακά Συστήματα & Υπηρεσίες»

ΚΑΤΕΥΘΥΝΣΗ:

«ΜΕΓΑΛΑ ΔΕΔΟΜΕΝΑ ΚΑΙ ΑΝΑΛΥΤΙΚΗ»

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ :

ΜΕΛΕΤΗ ΣΥΣΤΗΜΑΤΟΣ ΣΥΣΤΑΣΕΩΝ ΑΠΟ ΟΜΑΔΕΣ ΧΡΗΣΤΩΝ ΜΕ ΒΑΣΗ ΤΗΝ ΔΙΚΑΙΟΣΥΝΗ

ΓΕΩΡΓΟΠΟΥΛΟΣ ΔΗΜΟΣΘΕΝΗΣ Α.Μ: ΜΕ 1602

ΕΠΙΒΛΕΠΟΥΣΑ ΚΑΘΗΓΗΤΡΙΑ: ΜΑΡΙΑ ΧΑΛΚΙΔΗ

Διμελής Εξεταστική Επιτροπή

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

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτρια κα. Μαρία Χαλκίδη για την εξαιρετική συνεργασία που είχαμε και την πολύτιμη καθοδήγηση της κατά την διάρκεια εκπόνησης της διπλωματικής μου εργασίας.

ΠΕΡΙΛΗΨΗ

Στα πλαίσια εκπόνησης της παρούσας διπλωματικής εργασίας, γίνεται μελέτη πάνω στην υπολογιστική βελτιστοποίηση μιας ειδικής κατηγορίας συστημάτων συστάσεων. Συγκεκριμένα, στόχος είναι η δημιουργία ενός συστήματος, βασισμένο σε έναν ευρεστικό αλγόριθμο, το οποίο αντιμετωπίζει το πολύπλοκο πρόβλημα πρότασης πακέτων αντικειμένων σε ομάδες χρηστών λαμβάνοντας σαν περιορισμό την δίκαιη ευχαρίστηση όλων των μελών της ομάδας με ένα αποδοτικό υπολογιστικά τρόπο. Ο όρος «δίκαιη» προκύπτει από το ότι τα προτεινόμενα αντικείμενα πρέπει να πληρούν ένα συγκεκριμένο κριτήριο για κάθε μέλος της ομάδας και ο αλγόριθμος αποσκοπεί στην «έξυπνη» επιλογή τους. Περεταίρω, γίνεται μια γενικότερη εισαγωγή στα συστήματα συστάσεων, τις διαφορετικές προσεγγίσεις που υπάρχουν, ποια προβλήματα προσπαθούν να καλύψουν, την χρήση περιοριστικών παραμέτρων καθώς και την επίδραση επιφέρουν οι περιορισμοί αυτοί στην απόδοση τους. Στη συνέχεια, γίνεται περιγραφή του προσεγγιστικού ευρεστικού αλγορίθμου που αναπτύχθηκε για την δημιουργία προτάσεων που καλύπτουν περιορισμούς «δικαιοσύνης» μεταξύ μελών μιας ομάδας και αναλύονται τα αποτελέσματα συγκριτικά με τις πλήρως αναλυτικές λύσεις. Τέλος, γίνεται ανάλυση όλων των παραμέτρων που επηρεάζουν την απόδοση του αλγορίθμου και εξάγονται συμπεράσματα που μπορούν αποτελέσουν εφαλτήριο για μελλοντικές έρευνες.

ABSTRACT

The present master thesis focus on the computational optimization of a special category of recommendations systems. In particular, the objective is to create a system, based on a greedy algorithm, that addresses efficiently the complex problem of recommending packages of objects to user groups which are constrained by the fairness between all the members of the group. The "fairness" term derives from the fact that the proposed objects must meet a specific criterion among the members of the group and the algorithm aims on the "smart" selection of those. Moreover, an introduction is made to the theory of recommendations systems, the different implementation approaches that exist, what problems they are trying to solve, the use of constrains on such systems and the effect these limitations have on their performance. Furthermore, a detailed description is presented of the approximation algorithm that was developed for generating fairness constrained based recommendations where the generated results are evaluated with the equivalent complete brute force solutions. Finally, all the parameters that affect the performance of the algorithm are analyzed and conclusions can be drawn that can be used for future research.

ΠΕΡΙΕΧΟΜΕΝΑ

1. Εισαγωγή

Η λήψη μιας απόφασης, ακόμα και απλής όπως η αγορά κάποιου αντικειμένου ή η επιλογή μιας ταινίας, βασίζεται στην συλλογή και ανάλυση πληροφοριών. Συνεπώς, το «επιτυχημένο» αποτέλεσμα της απόφασης συνδέεται άμεσα με το πλήθος των πληροφοριών που έχουμε πρόσβαση. Ωστόσο, ο μεγάλος όγκος πληροφοριών έχει σαν παρενέργεια ότι αυξάνεται ο απαιτούμενος χρόνος για την ανάλυση τους και κατά συνέπεια επιβραδύνεται η διαδικασία της επιλογής. Στη εποχή μας, ο τεράστιος όγκος δεδομένων που είναι διαθέσιμος ανέδειξε την ανάγκη να δημιουργηθεί ένα σύστημα/διαδικασία διαχείρισής τους που θα εξάγει προτάσεις με χρονικά αποδοτικό τρόπο. Τη λύση στο πρόβλημα αυτό ήρθαν να δώσουν τα συστήματα συστάσεων (recommendation systems) τα οποία σήμερα έχουν εδραιωθεί στην καθημερινότητάς μας σαν σημείο αναφοράς για την λήψη αποφάσεων.

Ο κλάδος των ηλεκτρονικών αγορών αποτελεί μια περίπτωση όπου η χρήση συστημάτων συστάσεων πραγματικά αποτελεί αναπόσπαστο κομμάτι όχι μόνο για τους καταναλωτές αλλά και εμπορική επιτυχία των επιχειρήσεων. Από την πλευρά των καταναλωτών όσο μεγαλύτερος είναι ο διαθέσιμος αριθμός προϊόντων , τόσο πιο πολύπλοκη γίνεται η διαδικασία λήψης απόφασης για το ποιο ικανοποιεί τις ανάγκες του, το οποίο όμως αντιμετωπίζεται με τα συστήματα συστάσεων αφού αναλαμβάνουν να φιλτράρουν τον τεράστιο όγκο πληροφορίας και αναδείξουν τις διαθέσιμες επιλογές που καλύπτουν καλύτερα τις ανάγκες του κάθε καταναλωτή. Παραδείγματα επιτυχημένων επιχειρήσεων με μεγάλη επισκεψιμότητα που βασίζονται σε συστήματα συστάσεων αποτελούν η Amazon, το YouTube και το Netflix.

Η λειτουργία των συστημάτων αυτών βασίζεται σε αλγορίθμους οι οποίοι αποσκοπούν στην βέλτιστη δυνατή πρόβλεψη προϊόντων ή υπηρεσιών που μπορεί να καλύπτουν τις ανάγκες/ενδιαφέροντα κάποιου χρήστη. Τα κριτήρια δημιουργίας συστάσεων που βασίζονται οι αλγόριθμοι μπορεί να είναι χαρακτηριστικά των χρηστών (πχ ηλικία, φύλο), η συμπεριφορά τους σαν χρήστες (πχ τι έχουν αξιολογήσει θετικά ή τι τύπου προϊόντα αναζητούν συνήθως), οι σχέσεις μεταξύ χρηστών σε πλατφόρμες κοινωνικής δικτύωσης (πχ φίλοι ή επίπεδο εμπιστοσύνης) ή ακόμα και χαρακτηριστικά γνωρίσματα για τα ως προς σύσταση αντικείμενα (πχ ταινίες τύπου sci-fi). Συνοπτικά, η αρχή λειτουργίας των συστημάτων αυτών αποτελείται από δύο στάδια, όπου αρχικά δημιουργείται ένα προφίλ χαρακτηριστικών για το άτομο στο οποίο γίνεται η πρόταση, με ή χωρίς κάποια δεδομένα εισόδου, και στην συνέχεια φιλτράρουν τα διαθέσιμα αντικείμενα και προτείνουν ένα υποσύνολο με βάση τη συσχέτιση που έχουν με το παραπάνω προφίλ. Συνεπώς, δεδομένου ότι υπάρχουν διαφορετικές προσεγγίσεις για τα κριτήρια με τα οποία πρέπει να συσχετιστεί ένα άτομο με ένα αντικείμενο, τα συστήματα συστάσεων μπορούν να διαχωριστούν σε πολλές κατηγορίες όπως Content Based που είναι βασισμένα στο περιεχόμενο/χαρακτηριστικά των αντικειμένων, Collaborative Systems που βασίζονται στην συσχέτιση μεταξύ χρηστών ή αντικειμένων, Knowledge- Based Systems όπου βασίζονται στη γνώση του τρόπου κάλυψης των αναγκών των χρηστών, Hybrid Systems χρησιμοποιούν ένα συνδυασμό συστημάτων συστάσεων κ.ο.κ.

Περεταίρω, η μελέτη και ανάπτυξη των συστημάτων συστάσεων έχει επεκταθεί για την αντιμετώπιση πιο σύνθετων προβλημάτων, όπου δεν επαρκεί μόνο μια βέλτιστη πρόταση επιλογών από το σύστημα αλλά να μπορεί παράλληλα να εξασφαλίζει την ικανοποίηση περιορισμών. Για παράδειγμα, έστω ένα σύστημα που δημιουργεί προτάσεις για πακέτα διακοπών σε ομάδες χρηστών. Το σύστημα περιορίζεται αρχικά από το γεγονός ότι τα προτεινόμενα μέρη περιορίζονται χωρικά αλλά παράλληλα πρέπει και όλα τα μέλη της γκρουπ

να ευχαριστηθούν εξίσου (πχ αν είναι πακέτο οικογενειακών διακοπών να περιλαμβάνονται επιλογές που θα ευχαριστήσουν τόσο τα παιδιά όσο και τους ενήλικες). Συνεπώς, κατά την δημιουργία της σύστασης το σύστημα πρέπει να ελέγχει ότι δεν παραβιάζει κάποιο περιορισμό το οποίο αυξάνει δραματικά την πολυπλοκότητα του και να μην το καθιστά αποδοτικό σε εφαρμογές όπου απαιτούνται αποτελέσματα σε πραγματικό χρόνο.

Στα πλαίσια της παρούσας εργασίας στόχος είναι η ανάπτυξη ενός υπολογιστικά αποδοτικού συστήματος συστάσεων που θα βασίζεται σε έναν παραμετροποιήσιμο αλγόριθμο για την δημιουργία προτάσεων πακέτων από αντικείμενα σε ομάδες χρηστών με περιορισμό την δίκαιη ευχαρίστηση όλων των μελών της ομάδας. Απώτερος σκοπός είναι το σύστημα να μπορεί παράλληλα να μεγιστοποιήσει την ικανοποίηση όλων των χρηστών μιας ομάδας αλλά και να εξασφαλίσει ότι και όλοι οι χρήστες θα έχουν το «ίδιο βαθμό» ευχαρίστησης. Επιπλέον, θα μελετηθεί η απόδοση του αλγορίθμου τόσο σε υπολογιστικό κόστος όσο και βέλτιστων προτάσεων σε σχέση με τους τύπους των ομάδων στις οποίες γίνονται οι προτάσεις αλλά και με το πλήθος των διαθέσιμων αντικειμένων σε σχέση με το πλήθος των περιορισμών. Η υλοποίηση του συστήματος συστάσεων θα γίνει σε Python.

2. Εισαγωγή στα Συστήματα Συστάσεων

Στο κεφάλαιο αυτό περιγράφονται βασικά στοιχεία των συστημάτων συστάσεων, τρόποι υλοποίησης τους αλλά περιπτώσεις προβλημάτων που καλούνται να επιλύσουν.

2.1. Βασικοί Όροι

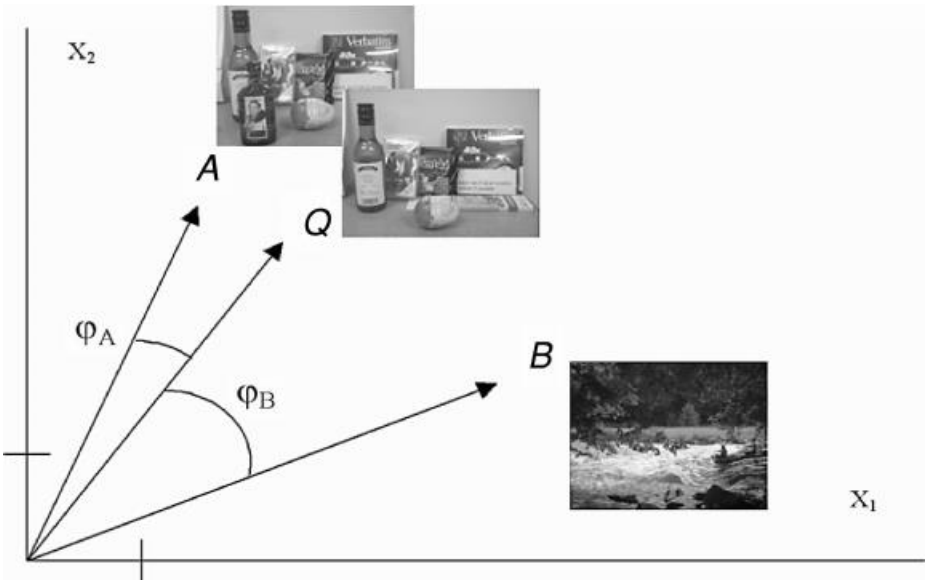
2.1.1. Ομοιότητα

Η αλγοριθμική «ομοιότητα» μεταξύ αντικειμένων γίνεται με ποικίλους τρόπους και συνδέεται άμεσα την απόδοση του συστήματος. Μερικοί από τους τρόπους που υπολογίζεται η ομοιότητα είναι η μέση τετραγωνική απόσταση, η ομοιότητα κατά Pearson και η ομοιότητα Jaccard. Ο A. Agarwal [1] καταγράφει αναλυτικά τους τρόπους αξιολόγησης του μέτρου ομοιότητας που χρησιμοποιούνται γενικότερα στα συστήματα συστάσεων και η επιλογή της μεθόδου εξαρτάται κυρίως από την φύση των χαρακτηριστικών.

Στα πλαίσια της παρούσας εργασίας η μέθοδος πάνω στην οποία θα βασιστεί το σύστημα συστάσεων είναι η ομοιότητα συνημιτόνου (cosine similarity). Για τον υπολογισμό του cosine similarity, μετασχηματίζονται τα χαρακτηριστικά μιας οντότητας (χρήστης ή αντικείμενο) σε ένα κανονικοποιημένο διάνυσμα ***a***. Στη συνέχεια ο υπολογισμός της ομοιότητας της με κάποια άλλη οντότητα ***b*** μπορεί να υπολογιστεί αλγεβρικά με τον τύπο:

$$\cos(a, b) = \frac{\vec{a} \cdot \vec{b}}{\|a\| \, \|b\|} \quad (1)$$

Ένα μειονέκτημα της τεχνικής είναι ότι κατά την διαδικασία των υπολογισμών οι κενές (*Null)* τιμές που μπορεί να εμφανίζονται πρέπει να αντικατασταθούν με κάποια αριθμητική τιμή (0 συνήθως) το οποίο όμως ενισχύει το σφάλμα του αποτελέσματος. Στην εικόνα 1 παρουσιάζεται ένα παράδειγμα για το πώς λειτουργεί η ομοιότητα συνημιτόνων μεταξύ αντικειμένων, όπου όσο μικρότερη είναι η γωνία μεταξύ δυο αντικειμένων τόσο πιο «όμοια» είναι μεταξύ τους.



Εικόνα 1: Παράδειγμα υπολογισμού της ομοιότητας συνημιτόνου(https://www.researchgate.net)

2.1.2. Απόδοση Συστημάτων Συστάσεων

Περεταίρω, πρέπει σε ένα σύστημα συστάσεων να γίνει ένας έλεγχος της ποιότητας των παραγόμενων αποτελεσμάτων. Η μεθοδολογία για την αξιολόγηση της απόδοσης των συστημάτων συστάσεων, βάσει του J. L. Herlocker [2] μπορεί να διαχωριστεί σε τρεις κατηγορίες:

- Predictive Accuracy Metrics
- Classification Accuracy Metrics
- Precision and Recall and Related Measures

Στην περίπτωση των Predictive Accuracy Metrics αξιολογείται πόσο κοντά είναι η εκτιμώμενη τιμή αξιολόγησης που υπολογίζει το σύστημα συστάσεων για ένα χρήστη σε σχέση με τη πραγματική βαθμολογία του. Η αξιολόγηση γίνεται με το μέσο απόλυτο σφάλμα (Mean Absolute Error ή MAE) , το οποίο υπολογίζει για κάθε αντικείμενο την απόλυτη διαφορά της εκτιμώμενης βαθμολογίας από την πραγματική βαθμολογία που έχει εισάγει ο κάθε χρήστης με τον παρακάτω τύπο:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{N} |r_{ui} - r'_{ui}|}{N} \tag{2}$$

Όπου N είναι το πλήθος των συστάσεων του συστήματος που θα αξιολογηθούν, r_{ui} είναι η πραγματική αξιολόγηση ενός χρήστη u για ένα αντικείμενο I και r'_{ui} είναι η αντίστοιχη υπολογισμένη τιμή από το σύστημα. Επίσης, για αυτή την κατηγορία αξιολόγησης χρησιμοποιείται και η ρίζα του μέσου τετραγωνικού σφάλματος (Root Mean Squared Error ή RMSE) με την σχέση:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (r_{ui} - r'_{ui})^2}{N}} \tag{3}$$

Η διαφορά του RMSE και MAE είναι στο πως επιδρούν οι επιμέρους διαφορές στο τελικό αποτέλεσμα. Στην περίπτωση του MAE όλες οι αποκλίσεις πρόβλεψης – πραγματικής τιμής είναι εξίσου σταθμισμένες σε αντίθεση με το RMSE όπου οι διαφορές υψώνονται στο τετράγωνο με αποτέλεσμα να δίδεται σχετικά μεγαλύτερο βάρος σε μεγάλες διαφορές.

Τα Classification accuracy metrics χρησιμοποιούνται για να αξιολογήσουν την επιτυχία ενός αλγορίθμου στο να καθορίσει σωστά τη κατηγορία που ανήκει το κάθε αντικείμενο.

Η τρίτη κατηγορία που περιλαμβάνει τις μετρικές Precision και Recall εφαρμόζονται στη λίστα των αντικειμένων που προτείνει το σύστημα για κάθε χρήστη και η συνολική ικανότητα του συστήματος να κατηγοριοποιεί σωστά τα αντικείμενα.

2.2. Βασικές Προσεγγίσεις Συστημάτων Συστάσεων

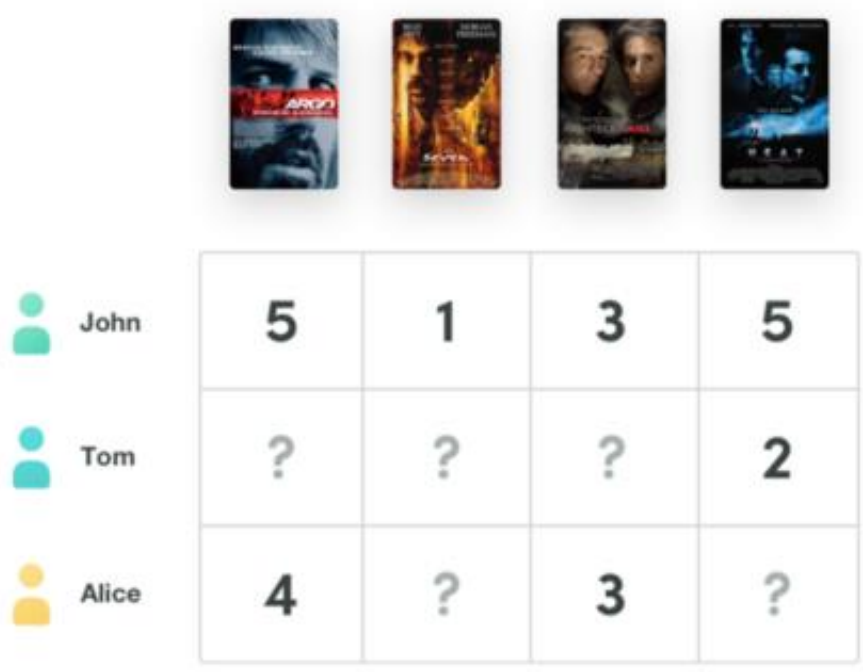
Αρχικά, η υλοποίηση ενός συστήματος συστάσεων δεν αποτελεί απλή υπόθεση δεδομένου ότι η βελτιστοποίηση των παραγόμενων αποτελεσμάτων δεν αποτελεί την μόνη παράμετρο που καθορίζει τον τρόπο σχεδιασμού του. Ανάλογα με την προσέγγιση που θα επιλεγεί,

διαφοροποιούνται οι περιορισμοί και τα προβλήματα εκάστοτε συστήματος, συνεπώς η αξιολόγηση παραμέτρων όπως το υπολογιστικό κόστος ή οι μεταβολές των δεδομένων είναι εξίσου σημαντικές. Παρακάτω αναλύεται η αρχή λειτουργίας για ορισμένους τύπους συστημάτων συστάσεων αλλά και οι περιορισμοί που προκύπτουν ανά περίπτωση.

2.2.1. Συνεργατικό φιλτράρισμα (Collaborative Filtering)

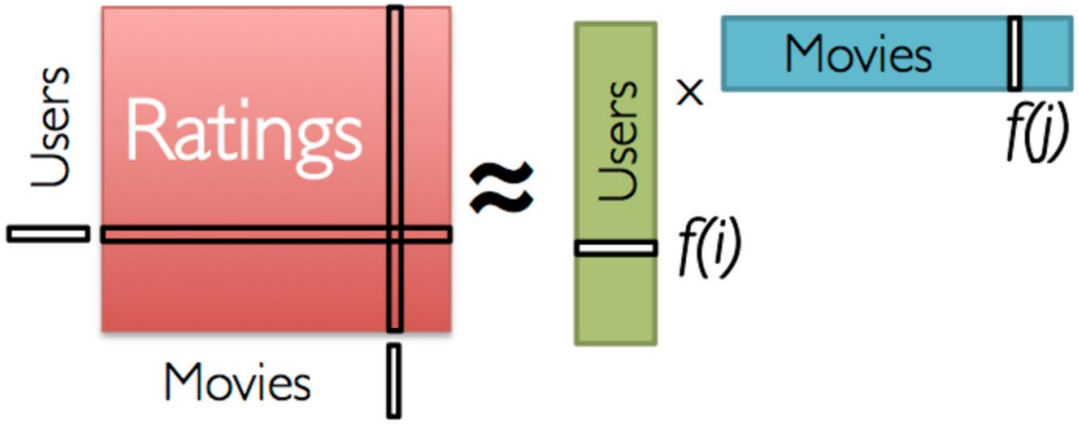
Το συνεργατικό φιλτράρισμα αποτελεί μια από τις πιο δημοφιλείς τεχνικές υλοποίησης συστημάτων συστάσεων, όπου για την δημιουργία συστάσεων οι αλγόριθμοι βασίζονται στις βαθμολογήσεις των χρηστών για να καθορίσουν κάποια σχέση ομοιότητας μεταξύ είτε των χρηστών είτε των αντικειμένων και να εξάγουν προβλέψεις σε βαθμολογήσεις που θα έδινε κάθε χρήστης σε κάποιο νέο αντικείμενο. Ανάλογα με τον τρόπο που υπολογίζεται η ομοιότητα υπάρχουν παραλλαγές των συστημάτων αυτών, οι οποίες θα περιγραφούν παρακάτω.

Αρχικά, σε κάθε σύστημα συστάσεων που βασίζεται στο συνεργατικό φιλτράρισμα όλα τα δεδομένα που σχετίζονται με τις προτιμήσεις (αξιολογήσεις) των χρηστών καταγράφονται σε ένα πίνακα (Utility Matrix) του οποίου οι γραμμές αντιστοιχούν σε διαφορετικούς χρήστες, οι κολώνες στα διαθέσιμα αντικείμενα και κάθε κελί υποδεικνύει τη προτίμηση του χρήστη U για το αντικείμενο I. Στην εικόνα 2 δίνεται ένα παράδειγμα, όπου οι προτιμήσεις των χρηστών ως προς κάποιες ταινίες έχουν απεικονιστεί σε ένα Utility Matrix.



Εικόνα 2:Παράδειγμα Utility Matrix (https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed)

Τα συστήματα που χρησιμοποιούν το συνεργατικό φιλτράρισμα μπορούν να διαχωριστούν βάση τον τρόπο που γίνονται οι υπολογισμοί σε αυτά που είναι βασιζόμενα στη μνήμη (memory-based) και αυτά που βασιζόμενα σε κάποιο υπολογισμένο μοντέλο (model-based). Στα memory-based συστήματα οι αλγόριθμοι διατρέχουν την τελευταία εικόνα του Utility Matrix και χρησιμοποιούν όλο το σύνολο των αντικειμένων που έχει βαθμολογήσει κάθε χρήστης στο παρελθόν για να εξάγουν για αυτόν συστάσεις. Αντίθετα, στην περίπτωση των model-based συστημάτων οι αλγόριθμοι που χρησιμοποιούνται λαμβάνουν ένα στιγμιότυπο του Utility Matrix και το μετασχηματίζουν πχ σε γινόμενο πινάκων βάση των θεμελιωδών διανυσμάτων (SVD method) και συνεπώς εκμεταλλεύονται μόνο το μέρος των αντικειμένων που έχει βαθμολογήσει ο κάθε χρήστης και έχει συμπεριληφθεί στον μετασχηματισμό για να γίνουν προβλέψεις βαθμολογιών του χρήστη σε νέα αντικείμενα. Παράδειγμα του μετασχηματισμού του Utility Matrix στην περίπτωση ενός model-based συστήματος παρουσιάζεται στην εικόνα 3.



Εικόνα 3: Μετασχηματισμός του Utility Matrix για model-based σύστημα συστάσεων (https://nycdatascience.com/blog/student-works/ninkasi-beer-recommender-system)

Περεταίρω, οι τεχνικές συνεργατικού φιλτραρίσματος μπορεί να κατηγοριοποιηθούν επιπλέον σε user-based και item-based. Στην περίπτωση της user-based (βασιζόμενη στο χρήστη) τεχνικής για τον υπολογισμό του rating ενός νέου αντικειμένου για ένα χρήστη αρχικά υπολογίζεται η ομοιότητα του με τους υπόλοιπους χρήστες και στην συνέχεια καθορίζεται το rating βάση του πως έχουν αξιολογήσει το αντικείμενο οι πιο «όμοιοι» χρήστες με αυτόν. Αντίστοιχα, στα item-based συστήματα συστάσεων υπολογίζεται αρχικά η ομοιότητα μεταξύ αντικειμένων, χρησιμοποιώντας τις βαθμολογήσεις των χρηστών του Utility Matrix, και στη συνέχεια υπολογίζεται το rating κάποιου χρήστη βάση το της αξιολόγησης που έχουν λάβει «όμοια» αντικείμενα.

Τα κύρια πλεονεκτήματα στα συστήματα συνεργατικού φιλτραρίσματος είναι [3] :

- Δεν απαιτείται γνώση του περιεχομένου των αντικειμένων αλλά μόνο η αξιολόγηση τους από τους χρήστες του συστήματος. Συνεπώς το συνεργατικό φιλτράρισμα δεν περιορίζεται ως προς το είδος των αντικειμένων για το οποία δημιουργούνται συστάσεις, δηλαδή η υλοποίηση θα είναι η ίδια ανεξάρτητα από το αν πρόκειται για ταινίες ή βιβλία.

- Με την χρήση και ενημέρωση του συστήματος βελτιώνεται η ακρίβεια των προβλέψεων του.

Αντίστοιχα τα συστήματα συστάσεων που βασίζονται στο συνεργατικό φιλτράρισμα αντιμετωπίζουν και κάποιους περιορισμούς/προβλήματα. Ορισμένα από αυτά αναφέρονται παρακάτω.

- Το cold-start πρόβλημα παρουσιάζεται όταν δεν υπάρχουν αρκετές αξιολογήσεις για ένα χρήστη ή αντικείμενο γιατί τα συστήματα συνεργατικού φιλτραρίσματος στηρίζονται μόνο στις αξιολογήσεις των χρηστών στο Utility Matrix για να παράγουν συστάσεις. Επομένως, όταν ένας νέος χρήστης εισάγεται στη βάση δεδομένων, το σύστημα δεν έχει τιμές για να υπολογίσει μέτρα ομοιότητας και για αυτό δεν μπορεί να του προτείνει αντικείμενα. Κατά αντιστοιχία, ένα νέο αντικείμενο που δεν έχει αξιολογηθεί θα μπορεί να προταθεί μόνο όταν το αξιολογήσει ένας ικανοποιητικός αριθμός από χρήστες ώστε να υπολογιστεί η «σχέση ομοιότητας» με τα υπόλοιπα.

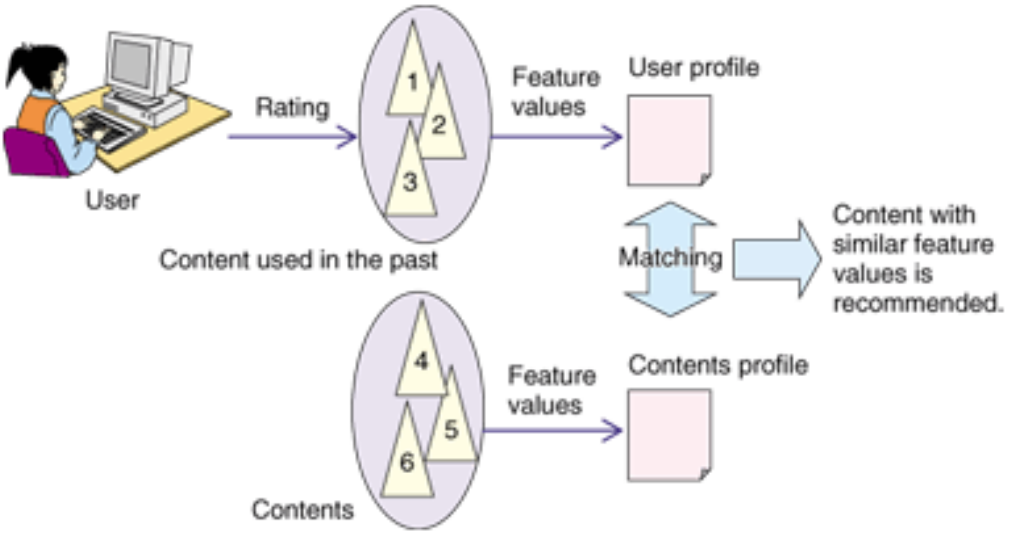
- Το grey sheep πρόβλημα, το οποίο σχετίζεται με την δυσκολία αυτού του τύπου συστημάτων συστάσεων να προτείνουν επιτυχώς αντικείμενα σε χρήστες που δεν είναι εύκολο να ταξινομηθούν σε μια ομάδα χρηστών με παραπλήσια ενδιαφέροντα. Για παράδειγμα ένας χρήστης που αξιολογεί όλες τις ταινίες που βλέπει με το ίδιο ακριβώς rating δεν παρουσιάζει κάποια συμπεριφορά που να επιτρέπει στο σύστημα να αναγνωρίζει πιθανές προτάσεις.

- Η επεκτασιμότητα αυτού του τύπου συστημάτων αποτελεί επίσης θέμα. Δεδομένου ότι η αρχιτεκτονική τους βασίζεται σε έναν πίνακα με γραμμές όσοι οι χρήστες και κολώνες ισάριθμες με το πλήθος των αντικειμένων, ο οποίος παρουσιάζει και πολύ μικρή πυκνότητα δεδομένων το κάνει μη αποδοτικό κυρίως για memory-based προσεγγίσεις σε εφαρμογές πραγματικού χρόνου.

2.2.2. Φιλτράρισμα με βάση το περιεχόμενο (Content-based Filtering)

Τα συστήματα που χρησιμοποιούν φιλτράρισμα με βάση το περιεχόμενο βασίζονται στην μελέτη των χαρακτηριστικών των αντικειμένων για την δημιουργία συστάσεων. Ως χαρακτηριστικά ορίζονται είτε τα μεταδεδομένα που περιγράφουν τα αντικείμενα όπως για παράδειγμα σε ένα σύστημα που προτείνει ταινίες αυτά μπορεί να είναι ο σκηνοθέτης ή οι πρωταγωνιστές είτε το «περιεχόμενο» τους όπως πχ ποια βιβλία που περιέχουν την λέξη-χαρακτηριστικό «ανάλυση».

Συνεπώς, για να παραχθεί μια πρόταση, το σύστημα συλλέγει το σύνολο των αντικειμένων που έχει αξιολογήσει ο χρήστης, γίνεται εξαγωγή των χαρακτηριστικών τους και δημιουργείται ένα προφίλ προτιμήσεων. Στη συνέχεια, γίνεται αντιστοίχιση με τα χαρακτηριστικά των αντικειμένων που υπάρχουν και προτείνονται εκείνα που καλύπτουν καλύτερα το προφίλ του χρήστη. Στην εικόνα 4 απεικονίζεται η αρχή λειτουργίας ενός τέτοιου συστήματος.



Εικόνα 4: Αναπαράσταση λειτουργίας ενός συστήματος συστάσεων με βάση το περιεχόμενο(http://findoutyourfavorite.blogspot.com/2012/04/content-based-filtering.html)

Σημαντικά πλεονεκτήματα των συστημάτων που βασίζονται στο φιλτράρισμα με βάση το περιεχόμενο είναι τα ακόλουθα[4]:

- Έχουν μεγαλύτερη επιτυχία στο να προτείνει αντικείμενα σε χρήστες που έχουν γούστα πολύ διαφορετικά σε σχέση με τους υπόλοιπους χρήστες.

- Για να προταθεί ένα αντικείμενο σε ένα χρήστη δεν είναι απαραίτητο να έχει αξιολογηθεί από άλλους χρήστες, όπως στα συστήματα που βασίζονται στο συνεργατικό φιλτράρισμα, αλλά αρκούν τα χαρακτηριστικά του, το οποίο σημαίνει ότι η πιθανότητα πρότασης ενός αντικειμένου δεν εξαρτάται από το πόσο δημοφιλές είναι.

Ωστόσο, και σε αυτή την περίπτωση συστημάτων υπάρχουν περιορισμοί, ορισμένοι από τους οποίους αναφέρονται παρακάτω:

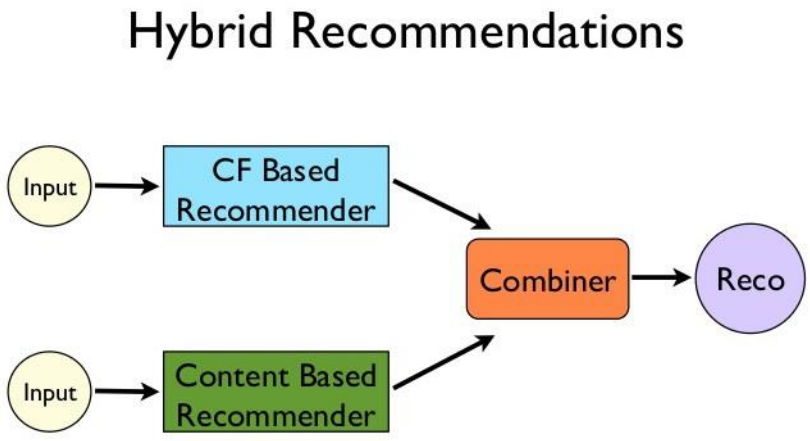
- Δεδομένου ότι η απόδοση του συστήματος επηρεάζεται σε σημαντικό βαθμό από την ορθή επιλογή χαρακτηριστικών που θα αποτελέσουν την βάση περιγραφή τους δεν είναι εύκολη η υλοποίησή τους για όλους τους τύπου αντικειμένων. Πιο συγκεκριμένα στην περίπτωση όπου ένα τέτοιο σύστημα χρησιμοποιηθεί για συστάσεις βιβλίων, υπάρχει η δυνατότητα ώστε η εξαγωγή των χαρακτηριστικών να γίνεται με ένα αυτοματοποιημένο και συγκεκριμένο τρόπο για κάθε αντικείμενο με τεχνικές ανάλυσης κειμένου. Αντίθετα, στη περίπτωση όπου τα προτεινόμενα αντικείμενα είναι ταινίες ή τραγούδια είναι δυσκολότερο ή αδύνατο να εφαρμοσθούν αυτόματες τεχνικές εξαγωγής των χαρακτηριστικών. Σε αυτή την περίπτωση η εξαγωγή τους εξαρτάται από τον ανθρώπινο παράγοντα, το οποίο εκτός του ότι μπορεί να είναι αδύνατο σε ορισμένες περιπτώσεις, μπορεί να δημιουργεί ασυνέπεια ως προς τα χαρακτηριστικά των αντικειμένων (πχ 2 διαφορετική χρήστες να κατατάσσουν την ίδια ταινία σε δύο διαφορετικές κατηγορίες).

- Και σε αυτό τον τύπο συστημάτων , υπάρχει το πρόβλημα του νέου χρήστη. Εάν ο χρήστης δεν έχει αξιολογήσει κάποια αντικείμενα , το σύστημα δεν μπορεί να εξάγει το περιεχόμενο τους και να προτείνει παρόμοια.

- Τα συστήματα φιλτραρίσματος με βάση το περιεχόμενο μπορεί να «εγκλωβιστούν» και να παράγουν προτάσεις από ένα πολύ συγκεκριμένο σύνολο αντικειμένων. Το πρόβλημα προκύπτει από το γεγονός ότι τα συστήματα αυτά προτείνουν μόνο αντικείμενα που καλύπτουν το προφίλ του χρήστη, το οποίο έχει σχηματιστεί με βάση αυτά που έχει αξιολογήσει ότι του άρεσαν. Κατά συνέπεια, αν ένας χρήστης έχει βαθμολογήσει μια συγκεκριμένη κατηγορία αντικειμένων το σύστημα δε μπορεί να προτείνει αντικείμενα που δεν υπάγονται σε αυτήν.

2.2.3. Υβριδικά συστήματα συστάσεων (Hybrid Recommendation Systems)

Ως υβριδικά ορίζονται τα συστήματα αυτά που συνδυάζουν δύο ή περισσότερες προσεγγίσεις συστημάτων συστάσεων[5]. Πιο συγκεκριμένα, σαν πρώτο βήμα συλλέγονται τα αποτελέσματα από διαφορετικούς αλγορίθμους επί των οποίων τρέχει κάποια δεύτερη διαδικασία αξιολόγησης τους (πχ υπολογίζεται κάποιος σταθμικός μέσος όρος) για την εξαγωγή της τελικής σύστασης. Με τον τρόπο αυτό μπορούν να χρησιμοποιήσουν τα θετικά χαρακτηριστικά των μεθόδων που συνενώνονται αλλά και να ξεπεράσουν παράλληλα τους περιορισμούς που παρουσιάζει η κάθε μια προσέγγιση μεμονωμένα. Αυτό βελτιώνει την ποιότητα των συστάσεων και καθιστά τα υβριδικά συστήματα μια ιδιαίτερα ελκυστική προσέγγιση.



Εικόνα 5:Παράδειγμα υλοποίησης υβριδικού συστήματος συστάσεων, όπου συνυπολογίζονται τα αποτελέσματα από δυο διαφορετικές προσεγγίσεις για την δημιουργία μιας πρότασης αντικειμένου σε ένα χρήστη(https://dataconomy.com/2015/03/an-introduction-to-recommendation-engines/).

2.3. Multi Objective Συστήματα Συστάσεων

Οι βασικές προσεγγίσεις συστημάτων συστάσεων που αναφέρθηκαν στην παράγραφο 2.2 έχουν σαν αποκλειστικό στόχο την κάλυψη ενός μεμονωμένου κριτηρίου όπως για παράδειγμα ποιο αντικείμενο ευχαριστεί ένα χρήστη. Ωστόσο, τα συστήματα συστάσεων μπορούν να σχεδιαστούν έτσι ώστε να καλύπτουν περισσότερους από ένα στόχους, όπως για παράδειγμα να προτείνουν πακέτα αντικειμένων σε ομάδες χρηστών [7] όπου πρέπει να συνυπολογιστούν τα χαρακτηριστικά όλων των χρηστών ή να παράγουν συστάσεις που υπόκεινται περιορισμούς (πχ εύρεση του καλύτερου εστιατορίου σε μια πόλη για συγκεκριμένο τύπο φαγητού)[6]. Όμως, η ύπαρξη πολλαπλών στόχων ή περιορισμών κατατάσσει συνήθως τις προσεγγίσεις αυτές σε επίπεδο πολυπλοκότητας στην κατηγορία των NP-Hard προβλημάτων δεδομένου ότι δεν είναι εύκολη η εύρεση αποδοτικών υπολογιστικά αλγορίθμων που να το αντιμετωπίζουν ντετερμινιστικά.

2.3.1. Δημιουργία Συστάσεων Σε Ομάδες Χρηστών (Group Recommendation Systems)

Αυτός ο τύπος συστήματος, για την δημιουργία συστάσεων δεν περιορίζεται στα ατομικά χαρακτηριστικά ενός χρήστη αλλά εστιάζει περισσότερο στις γενικότερες προτιμήσεις της ομάδας που αυτός ανήκει. Για παράδειγμα ένας τέτοιος τύπος συστήματος θα μπορούσε να χρησιμοποιηθεί στην περίπτωση της δημιουργίας συστάσεων πακέτων διακοπών σε γκρουπ τουριστών, ωστόσο, η μελέτη των μοντέλων αυτόν δεν περιορίζεται στο χώρο των τουριστικών επιχειρήσεων αλλά επεκτείνεται και σε κλάδους όπως αυτός της υγείας [7]. Η δυσκολία στην ανάπτυξη τέτοιων συστημάτων βρίσκεται στην εύρεση ενός μοντέλου υπολογιστικά αποδοτικού που μετρά τις προτιμήσεις μιας ομάδας και συστήνει αντικείμενα τα οποία θα είναι χρήσιμα για τη πλειοψηφία των ατόμων της ομάδας.

Περεταίρω, οι προσεγγίσεις που χρησιμοποιούνται στα group recommendation systems εξαρτώνται και από τον τύπο των ομάδων στις οποίες πρέπει να προτείνουν αντικείμενα. Οι ομάδες χρηστών χωρίζονται σε 2 κατηγορίες [8] τις σταθερές (persistent groups) και τι εφήμερες (ephemeral groups). Στα persistent groups η δομή είναι σταθερή και υπάρχει το ιστορικό της αλληλεπίδρασης μεταξύ των μελών τους, πράγμα που σημαίνει ότι μπορεί μια ομάδα χρηστών να αντιμετωπιστεί σαν μια ενιαία οντότητα. Αντίθετα, στη περίπτωση των ephemeral groups δεν υπάρχει σχέση αλληλεπίδρασης μεταξύ των χρηστών και αυτά δημιουργούνται βάση ομαδοποίησης των ατομικών προτιμήσεων τους. Ωστόσο, όμως υπάρχουν και περιπτώσεις όπου λαμβάνεται σαν παράμετρο στο σύστημα η σχέση αλληλεπίδρασης μεταξύ των χρηστών [11] για την δημιουργία συστάσεων.

2.3.2. Fairness σε Συστήματα Συστάσεων

Ο όρος Fairness στα συστήματα συστάσεων δεν είναι μονοσήμαντα ορισμένος και η μορφή του εξαρτάται από τις ανάγκες που προσπαθεί να καλύψει η αντίστοιχη υλοποίηση. Αρχικά,

στην γενικότερη περιγραφή, τα συστήματα συστάσεων αποτελούν πλατφόρμες [9] όπου ο ιδιοκτήτης (provider) παρέχει σε κάποιον καταναλωτή (consumer) μια υπηρεσία. Με βάση αυτή την περιγραφή τα συστήματα που βασίζονται στο Fairness μπορούν να χωριστούν σε 3 κατηγορίες:

- Σε αυτά που εστιάζουν στο Fairness των consumers (C-fairness). Παράδειγμα τέτοιου συστήματος θα μπορούσε να είναι ένα σύστημα που κάνει συστάσεις για εύρεση εργασίας. Στην περίπτωση αυτή, οι συστάσεις που γίνονται θα πρέπει να μη επηρεάζονται π.χ. από το φύλο του υποψήφιου γιατί μπορεί να υπάρχει μεγαλύτερη πιθανότητα για τους άντρες ή τις γυναίκες να επιλεγούν για κάποια συγκεκριμένη δουλειά. Συνεπώς, θα πρέπει να προτείνει θέσης εργασίας με δίκαιο τρόπο.

- Σε αυτά που εστιάζουν στο Fairness των providers (P-fairness). Η Airbnb αποτελεί παράδειγμα υλοποίησης ενός τέτοιου συστήματος. Όταν κάποιος χρήστης αναζητά διαθέσιμα καταλύματα σε ένα προορισμό δεν πρέπει στα παραγόμενα αποτελέσματα να εμφανίζονται μόνο οι καλά βαθμολογημένοι ιδιοκτήτες αλλά όλοι όσοι καλύπτουν τα κριτήρια της αναζήτησής του.

- Σε αυτά που εστιάζουν στην διατήρηση του Fairness και για τις δύο πλευρές (CP-fairness)

Περεταίρω, ο μαθηματικός φορμαλισμός του Fairness στην περίπτωση Group Recommendation συστημάτων παρουσιάζεται παρακάτω. Έστω ότι γίνεται σύσταση σε μια ομάδα G που αποτελείται από χρήστες U μια λίστα από αντικείμενα I και S(U,i) είναι μια συνάρτηση που περιγράφει την συσχέτιση του αντικειμένου με τον χρήστη τότε μπορούν να χρησιμοποιηθούν οι παρακάτω σχέσεις σαν μέτρα του Fairness:

- Least Misery : $F(g,I) = min\{\mathrm{S(U,I)}, \forall\ \mathrm{U} \in \mathrm{G}\}$ (4)

- Variance : $F(g,I) = 1\ /\ Var(\{\mathrm{S(U,I)}, \forall\ \mathrm{U} \in \mathrm{G}\})$ (5)

- Min-Max Ratio : $F(g,I) = \frac{min\{\mathrm{S(U,I)}, \forall\ \mathrm{U} \in \mathrm{G}\}}{max\{\mathrm{S(U,I)}, \forall\ \mathrm{U} \in \mathrm{G}\}}$ (6)

2.4. Fairness Aware Package To Group Recommendation Systems

Αυτός ο τύπος συστήματος προσπαθεί να καλύψει πολλαπλούς στόχους και προκύπτει από τον συνδυασμό των βασικών τεχνικών που αναφέρονται στην παράγραφο 2 με τις προσεγγίσεις που παρουσιάζονται στην παράγραφο 3. Παράδειγμα Συνεπώς, αποτελεί μια περίπτωση συστήματος αρκετά περίπλοκη και δεν υπάρχει κάποιος ιδανικός τρόπος υλοποίησης του.

Μια εκτενής μελέτη για την υλοποίηση αυτού του είδους συστήματος συστάσεων έγινε από τον D. Serbos et al [10], όπου η πολυπλοκότητα του προβλήματος της «δικαιοσύνης» με μετρικές fairness (proportionality, envy-freeness) μεταξύ χρηστών με ευρεστικούς αλγορίθμους οι οποίοι βασίζονται σε «greedy» επιλογές αντικειμένων οι οποίες ικανοποιούν την πλειοψηφία των μη εκπληρωμένων περιορισμών. Δηλαδή, κύριος στόχος του συστήματος είναι να μπορέσει να δημιουργήσει ένα πακέτο από αντικείμενα όπου η διαδοχική διαλογή τους να στοχεύει πρωτίστως στην κάλυψη των περιορισμών. Περεταίρω εξετάζει την λειτουργία των αλγορίθμων με επιπλέον περιορισμούς όπως το πλήθος των διαθέσιμων αντικειμένων ανά κατηγορία που μπορούν να επιλεχθούν ανά σύσταση και τη «χωρική» απόσταση μεταξύ αντικειμένων. Αντίστοιχη προσέγγιση ευρεστικού αλγορίθμου παρουσιάζεται από τον L. Xiao et al [8]. Το NP-Hard πρόβλημα βελτιστοποίησης, προσπαθεί να το αντιμετωπίσει με μια Pareto Optimal γραμμική λύση που περιλαμβάνει τα βάρη του Social Welfare (ευχαρίστηση) και του Fairness.

3. Σύστημα συστάσεων σε ομάδες χρηστών με βάση την δικαιοσύνη

3.1. Δημιουργία Συστάσεων με Collaborative Filtering

Σαν βασικό σύστημα συστάσεων στην εργασία χρησιμοποιήθηκε η προσέγγιση του user-user collaborative filtering για την δημιουργία ενός utility matrix όπου για κάθε χρήστη έχει υπολογιστεί το rating για κάθε αντικείμενο. Αρχικά, το dataset χωρίζεται σε test/train και υπολογίζεται ο πίνακας ομοιότητας μεταξύ των χρηστών. Στην συνέχεια υπολογίζονται τα ratings στα αντικείμενα του κάθε χρήστη βάσει της σχέσης:

$$P_{ui} = \overline{r_u} + \frac{\sum_{j \in U} sim(u,j)\ (r_{ji} - \bar{r}_j)}{\sum_{j \in U} |sim(u,j)|} \qquad (12)$$

όπου P_{ui} είναι η προβλεπόμενη αξιολόγηση του χρήστη u για ένα αντικείμενο I, $\overline{r_u}$ είναι η μέση τιμή αξιολόγησης των αντικειμένων του χρήστη u και $sim(u,j)$ είναι η ομοιότητα συνημιτόνου του χρήστη u με κάποιο άλλο χρήστη j.

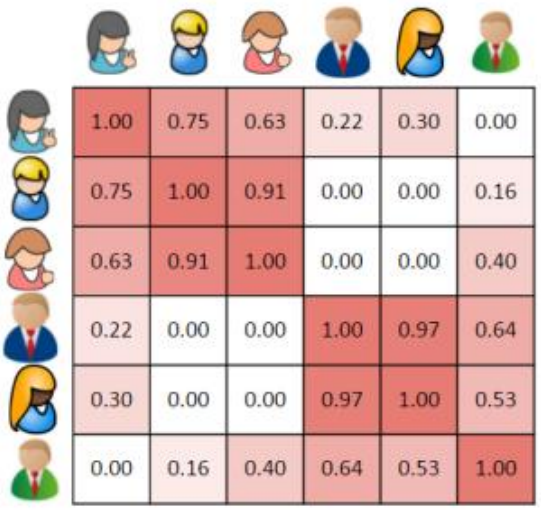
3.2. Δημιουργία Ομάδων

Δεδομένου ότι το dataset δεν έχει ομάδες χρηστών ορισμένες, για την εκτέλεση των πειραμάτων αυτές δημιουργούνται δυναμικά τριών ειδών ομάδες:

- Ομάδες ομοιότητας, όπου τα μέλη τους «μοίαζουν».
- Ομάδες διαφορετικότητας, που υπάρχει διαφοροποίηση μεταξύ των μελών.
- Ομάδες τυχαιότητας, που δημιουργούνται με τυχαία επιλογή χρηστών.

Στις δύο πρώτες περιπτώσεις, αρχικά υπολογίζεται ο πίνακας ομοιότητας (εικόνα 6) μεταξύ των χρηστών.

Στην συνέχεια επιλέγεται ένας χρήστης τυχαία από τους χρήστες του dataset που αποτελεί τον πρώτο χρήστη της ομάδας. Στη συνέχεια επιλέγονται και προστίθενται διαδοχικά χρήστες που παρουσιάζουν την μέγιστη ομοιότητα, στην περίπτωση των ομάδων ομοιότητας, ή ελάχιστη ομοιότητα, στην περίπτωση των ομάδων διαφορετικότητας σε σχέση με τα υπάρχοντα μέλη της ομάδας.



Εικόνα 6: Παράδειγμα πίνακα ομοιότητας (https://buildingrecommenders.wordpress.com/2015/11/18/overview-of-recommender-algorithms-part-2/)

Αναλυτικότερα, η αλγοριθμική διαδικασία που ακολουθείται για την παραγωγή των ομάδων για ομάδες «ομοιότητας» και «διαφορετικότητας» παρουσιάζεται παρακάτω:

Input: Available_Users, Sim_Matrix, Group_Size, Type{max,min}
Result: List of Groups

While Available_Users > 0 do:

Create empty Group_G_j
Add random $user_i$ from Available_Users to Group_G_j
Remove $user_i$ from Available_Users

While Group_G_j < Group_Size do:
Add $user_k$ from Available_Users to Group_G_j where similarity of $user_k$ is max/min
with vs user ∈ Group_G_j
end
end

3.3. Περιγραφή Αλγορίθμου

Η παρούσα εργασία πραγματεύεται το πολύπλοκο πρόβλημα του fairness aware package to group recommendation. Για αυτή την κατηγορία συστημάτων συστάσεων οι πλήρως αναλυτικές λύσεις, όπου υπολογίζονται όλοι οι δυνατοί και επιτρεπτοί συνδυασμοί αντικειμένων και επιλέγεται ο βέλτιστος, δεν είναι υπολογιστικά αποδοτικές και κατά συνέπεια δεν εύκολο να υλοποιηθούν σε πραγματικές εφαρμογές. Συνεπώς, η παρούσα μελέτη στοχεύει στην εύρεση ενός αλγορίθμου που θα μπορεί να ικανοποιεί τους περιορισμούς της «δικαιοσύνης» μεταξύ των μελών μίας ομάδας ενώ παράλληλα θα επιτυγχάνει την βέλτιστη ικανοποίηση σε χρόνους που θα επέτρεπαν υλοποίηση ιδανικά και σε μια εμπορική real time εφαρμογή.

Η αρχή πάνω στην οποία έχει σχεδιαστεί ο αλγόριθμος είναι η παρακάτω:

- Αρχικά για μία ομάδα χρηστών αξιολογούνται και επιλέγονται τα top αντικείμενα για ένα πακέτο από ένα σύνολο επιτρεπτών επιλογών βάσει κάποιων συντελεστών/παραμέτρων.
- Στην συνέχεια υπολογίζεται το μέτρο της δικαιοσύνης μεταξύ των χρηστών και εντοπίζεται ο πιο «αδικημένος».
- Στο σύνολο των επιτρεπτών αντικειμένων ενισχύονται τα αντικείμενα που ικανοποιούν τον πιο αδικημένο χρήστη και ξαναγίνεται επιλογή των αντικειμένων του πακέτου.
- Η διαδικασία επαναλαμβάνεται ώσπου το σύστημα να επιτύχει τον περιορισμό της δικαιοσύνης και να βελτιστοποιήσει την ευχαρίστηση των χρηστών.

Αναλυτικότερα, για την ανάπτυξη του συστήματος συστάσεων που παρουσιάζεται στην παρούσα εργασία θεωρούμε ότι έστω έχουμε ένα γκρουπ U από χρήστες u_i και ένα σύνολο S αντικειμένων s_j που μπορούν να προταθούν στους χρήστες του U. Αρχικά, παράγονται οι εκτιμώμενες αξιολογήσεις των χρηστών για κάθε αντικείμενο με την χρήση ενός user-user collaborative filtering σύστημα συστάσεων. Στην συνέχεια για κάθε χρήστη της ομάδας επιλέγονται K_i πιο επιθυμητά αντικείμενα και από αυτά δημιουργείται η συνολική λίστα L με διαθέσιμα για το γκρουπ.

Στη συνέχεια το σύστημα στοχεύει να δημιουργήσει ένα πακέτο P με n αντικείμενα όπου:

- Έχοντας την ευχαρίστηση του χρήστη σαν μετρική που ορίζεται από την σχέση:

$$Satisfaction(u_i) = \frac{\sum_{s_j \in L} r_{s_j} \cdot x_{s_j}}{\sum_{s_j \in K_i} r_{s_j}} \quad (7)$$

με r_{s_j} την αξιολόγηση του χρήστη για το αντικείμενο s_j και $x_{s_j} = \begin{cases} 1, & \alpha\nu\ s_j \in K_i \\ 0, & \alpha\nu\ s_j \notin K_i \end{cases}$, αποσκοπεί στην μεγιστοποίηση των σχέσεων 4 – 6 συναρτήσει της σχέσης 7.

- Εξασφαλίζει ότι στα αντικείμενα του προτεινόμενου πακέτου υπάρχουν τουλάχιστον m από τα n που ανήκουν στις κορυφαίες προτιμήσεις K_i για κάθε χρήστη της ομάδας.

Για την δημιουργία σύστασης ο αλγόριθμος αξιολογεί αρχικά τα διαθέσιμα αντικείμενα από την λίστα L βάσει του πλήθους των χρηστών που καλύπτει το καθένα αλλά και το μέσο rating που έχουν λάβει από τα μέλη της ομάδας παραμετρικά. Η συνάρτηση αξιολόγησης του κάθε αντικειμένου από τον αλγόριθμο είναι:

- $Score\ of\ item_\iota = rating\ factor * rating\ of\ item_{i\ avg} + coverage\ factor * user\ coverage\ of\ item_\iota$ (8)

όπου το $user\ coverage\ of\ item_\iota$ περιγράφει το ποσοστό των χρηστών για τους οποίους ένα αντικείμενο i ανήκει στις K κορυφαίες προτιμήσεις τους και περιγράφεται από την σχέση:

- $user\ coverage\ of\ item_\iota = \frac{number\ of\ users\ in\ group\ for\ which\ item_i \text{ is in } top\ items}{number\ of\ users\ in\ group}$ (9)

το $rating\ of\ item_{i\ avg}$ περιγράφει την μέση αξιολόγηση των χρηστών της ομάδας για αυτό το αντικείμενο και υπολογίζεται από την σχέση:

- $rating\ of\ item_{i\ avg} = \frac{\sum ratings\ of\ user's\ of\ item_i}{number\ of\ users\ in\ group}$ (10)

το $rating\ factor$ και το $coverage\ factor$ είναι μια παράμετροι (βάρη) που καθορίζουν την στον υπολογισμό της σχέσης (9) τη σημαντικότητα του $user\ coverage\ of\ item_\iota$ και του $user\ coverage\ of\ item_\iota$.

Στην συνέχεια, με βάση το «σκορ» που έχει προκύψει επιλέγει τα πρώτα n για την δημιουργία πακέτου και αξιολογεί κατά πόσο καλύπτεται ο περιορισμός του ελάχιστου αριθμού αντικειμένων ανά χρήστη. Ακολούθως, αν δεν καλύπτεται για μέλος της ομάδας ο περιορισμός, ο αλγόριθμος αναβαθμολογεί (ενισχύει) τα αντικείμενα της λίστας L που ανήκουν στα top αντικείμενα του χρήστη για τον οποίο δεν ισχύει ο περιορισμός και επιλέγει ξανά αντικείμενα για την δημιουργία πακέτου. Η διαδικασία επαναλαμβάνεται μέχρι να καλύπτεται ο περιορισμός για όλους τους χρήστες. Αντίστοιχα, για την βελτιστοποίηση της ευχαρίστησης της ομάδας, εφόσον έχουν καλυφθεί οι περιορισμοί, ο αλγόριθμος βρίσκει τον χρήστη για τον οποίο η συνάρτηση (7) παρουσιάζει την ελάχιστη τιμή και αναβαθμολογεί τα αντικείμενα της λίστας L που ανήκουν στα πιο επιθυμητά του και επιλέγονται ξανά αντικείμενα για την δημιουργία πακέτου.

Input: List_of_Groups, Utility_Matrix, Score_Params, Constrain_of_top_items,Package_size, Boost_param
Result: List of Group_Packages

Create a list of all the items that can be recommended per group
For $Group_k$ in List_of_Groups do:
 For $User_i$ in $Group_k$ do:
 Add $User_i$ N top items to $Group_k$_list_of_items
 end
end

For $Group_k$ in List_of_Groups do:
 For $Item_j$ in $Group_k$_list_of_items do:
  Score $Item_j$ based on the Score_Params of eq (8)
 end

 While (Constrain_of_top_items <> True and Delta<0) or (Iterations <max iterations) do:
  Temp_package = Select top items based on Package_size
  Calculate $Group_k$_$Satisfaction_s$
  Delta = $Group_k$_$Satisfaction_s$ – max_$Group_k$_$Satisfaction_s$
   If Delta>0 do:
    If Constrain_of_top_items is False do:
     Find least satisfied $user_m$
     For each $Item_j$ of $user_m$ in $Group_k$_list_of_items do:
      Score($Item_j$) = Score($Item_j$) * Boost_param
     end
    else do:
     max_$Group_k$_$Satisfaction_s$ = $Group_k$_$Satisfaction_s$
     $Group_k$_Package = Temp_package
  end
end

Αφού ολοκληρωθεί η διαδικασία, τα αποτελέσματα του αλγορίθμου ελέγχονται σε σχέση με την βέλτιστη λύση. Η βέλτιστη λύση υπολογίζεται με έλεγχο όλων των δυνατών συνδυασμών αντικειμένων της λίστας L (n) που δημιουργούν ένα πακέτο με m αντικείμενα. Το πλήθος των συνδυασμών υπολογίζεται από την σχέση:

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \qquad (11)$$

Στην συνέχεια, εξετάζεται η μελέτη των παραμέτρων (βάρη) που λαμβάνουν μέρος στον υπολογισμό του «σκόρ» που χρησιμοποιείται για την δημιουργία συστάσεων πακέτων από τον αλγόριθμο και πως επηρεάζουν την απόκλιση του αποτελέσματος από τον «ιδανικό» συνδυασμό. Επιπλέον, γίνεται μελέτη της συσχέτισης του συντελεστή αναβαθμολόγησης με το χρόνος εκτέλεσης, το ελάχιστο πλήθος επαναλήψεων για την εύρεση της βέλτιστης λύση δυνατής λύσης από τον αλγόριθμο αλλά και την σύγκλιση του αλγορίθμου.

4. ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

4.1. Collaborative Filtering

Το dataset που χρησιμοποιήθηκε για την παρούσα πειραματική μελέτη ήταν το movielens με 100k αξιολογήσεις από 1000 χρήστες σε 1700 ταινίες (http://files.grouplens.org/datasets/movielens/ml-100k.zip). Συγκεκριμένα, τα δεδομένα είναι της μορφής:

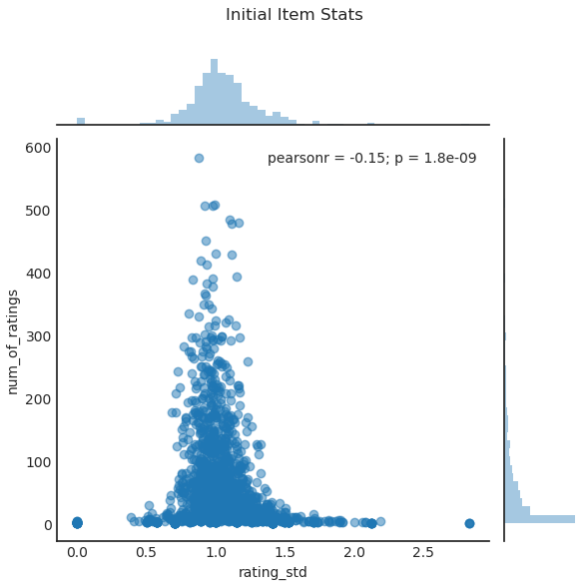
userId : Το id του χρήστη που έχει αξιολογήσει μια ταινία

movieId: Τι Id της ταινίας που αξιολογήθηκε

rating: Η αξιολόγηση που έδωσε ο χρήστης σε μια κλίμακα {0-5}

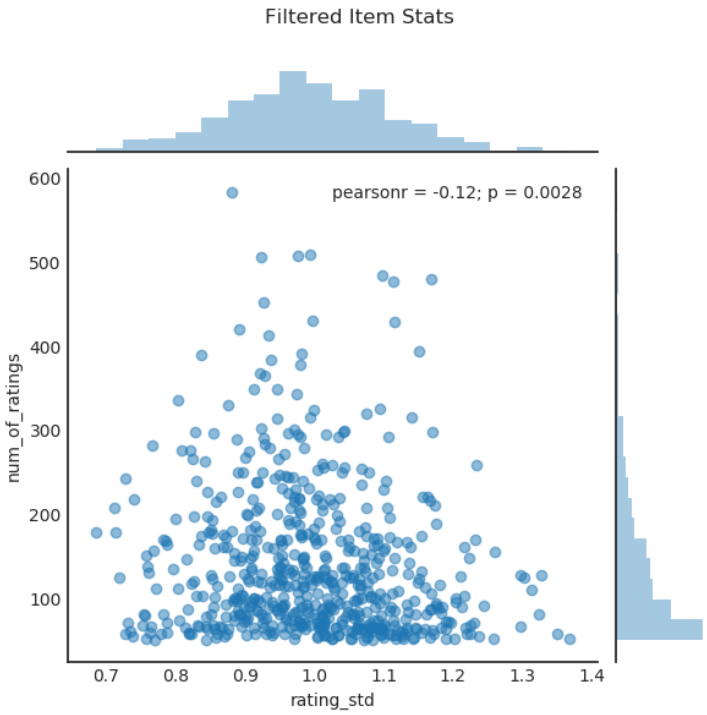
Δεδομένου ότι κύριος στόχος είναι η μελέτη του της επίδρασης του Fairness από το dataset, με την βιβλιοθήκη pandas της Python,αφαιρέθηκαν χρήστες και αντικείμενα που δημιουργούσαν θόρυβο (πχ. δεν υπήρχε διακύμανση στις επιλογές τους, βάζοντας παντού τον ίδιο βαθμό) ή ακόμα και δυσκολίες στην εφαρμογή των constrains (πχ να πρέπει να υπάρχουν 3 αντικείμενα για κάθε χρήστη από τα top items του στην τελική λίστα αλλά μόνο 2 να μπορούν να γίνουν recommended για κάποιον επειδή είχε λίγα ratings στο αρχικό dataset).

Αρχικά, φιλτραρίστηκαν αντικείμενα που είχαν αξιολογηθεί λιγότερες από 50 φορές ώστε να αυξηθεί η πιθανότητα να είναι διαθέσιμο για πρότασης σε πολλούς χρήστες ταυτόχρονα και να είναι εφικτή η δημιουργία συστάσεων που καλύπτουν τους περιορισμούς. Αντίστοιχα, αντικείμενα όπου παρουσίαζαν τυπική απόκλιση μικρότερη από 0.7 αφαιρέθηκαν ώστε να περιοριστούν τα αντικείμενα που αρέσουν σε πολλούς χρήστες εξίσου. Στην γράφημα 1, παρουσιάζεται για τα αντικείμενα του αρχικού dataset η κατανομή του πλήθους αξιολογήσεων συναρτήσει της τυπικής απόκλισης των αξιολογήσεων που έλαβαν.



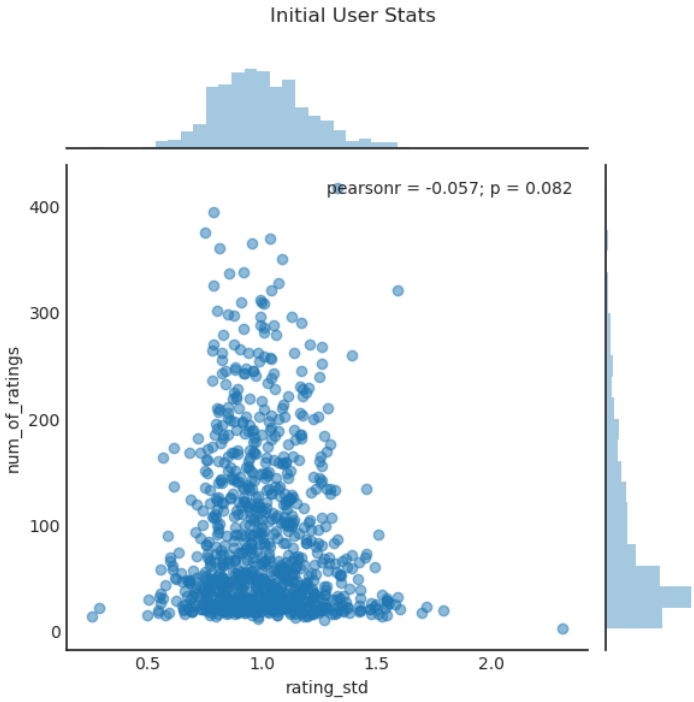
Γράφημα 1: Το πλήθος αξιολογήσεων των αντικειμένων συναρτήσει της τυπική απόκλισης των αξιολογήσεων τους στο αρχικό δείγμα.

Έχοντας εφαρμόσει τα φίλτρα που αναφέρθηκαν παραπάνω, η κατανομή του πλήθους αξιολογήσεων συναρτήσει της τυπικής απόκλισης των αξιολογήσεων λαμβάνει την μορφή που παρουσιάζεται στο γράφημα 2.



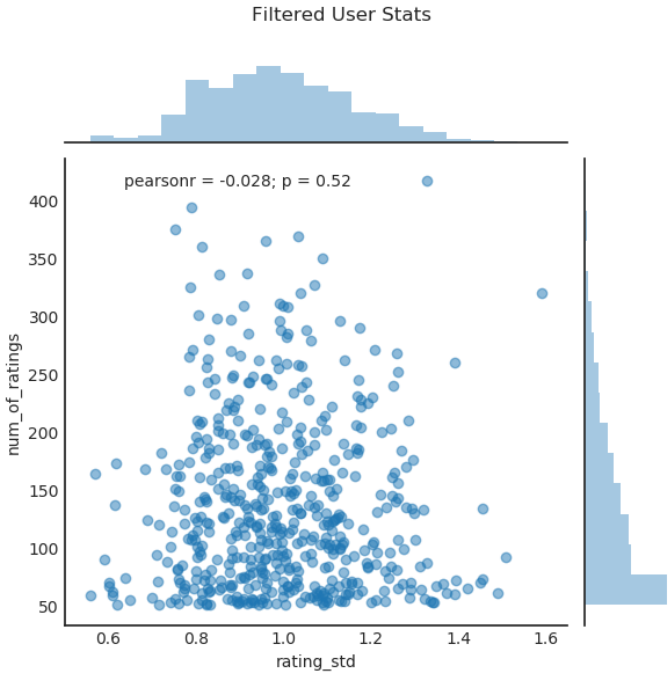
Γράφημα 2: Το πλήθος αξιολογήσεων των αντικειμένων συναρτήσει της τυπική απόκλισης των αξιολογήσεων τους στο φιλτραρισμένο δείγμα.

Στην συνέχεια, έχοντας εφαρμόσει τα παραπάνω φίλτρα στο dataset, επαναλαμβάνεται η αντίστοιχη διαδικασία για τους χρήστες. Αρχικά φιλτράρουμε τους χρήστες που έχουν κάνει αξιολόγηση σε λιγότερα από 50 αντικείμενα, ώστε κατά την δημιουργία συστάσεων από το σύστημα να υπάρχει επαρκής αριθμός διαθέσιμών αντικειμένων. Επίσης, αφαιρούνται χρήστες οι οποίοι παρουσιάζουν μικρή τυπική απόκλιση ως προς τις αξιολογήσεις τους, δίνουν δηλαδή το ίδιο περίπου rating σε όλα τα αντικείμενα. Στο γράφημα 3 παρουσιάζεται η αρχική κατανομή του πλήθους αξιολογήσεων που έχουν κάνει οι χρήστες συναρτήσει της τυπικής απόκλισης των αξιολογήσεων που έχουν δώσει στο φιλτραρισμένο dataset.



Γράφημα 3: Το πλήθος αξιολογήσεων των χρηστών συναρτήσει της τυπική απόκλισης των αξιολογήσεων τους στο φιλτραρισμένο δείγμα.

Αντίστοιχα, από την εφαρμογή των φίλτρων πάνω στους χρήστες του dataset παράγεται το τελικό δείγμα (γράφημα 4) που περιλαμβάνει 518 χρήστες και 593 αντικείμενα για την δημιουργία του Utility Matrix που θα αποτελέσει βάση για Collaborative filtering και την δημιουργία ομάδων χρηστών.



Γράφημα 4:Το πλήθος αξιολογήσεων των χρηστών συναρτήσει της τυπική απόκλισης των αξιολογήσεων τους στο τελικό dataset.

Η παραπάνω διαδικασία γίνεται μέσω παραμετρικού αρχείο και στο επεξεργασμένο dataset με τους 518 χρήστες και 593 αντικείμενα εκτελείται ο αλγόριθμος του Collaborative Filtering. Το dataset χωρίζεται σε 65% trainset και 35% testset και από την εκτέλεση του αλγορίθμου γεμίζει με τιμές ο Utility Matrix. Σε επίπεδο ακρίβειας του υπολογισμού αξιολογήσεων των χρηστών, το MSE που προκύπτει είναι περίπου 2.9 το οποίο θα ήταν μικρότερο αν γινόταν χρήση κάποιας πιο αναλυτικής προσέγγισης αλλά είναι εκτός των σκοπών της παρούσας εργασίας.

4.2. Ικανότητα Παραγωγής Συστάσεων (Success Rate)

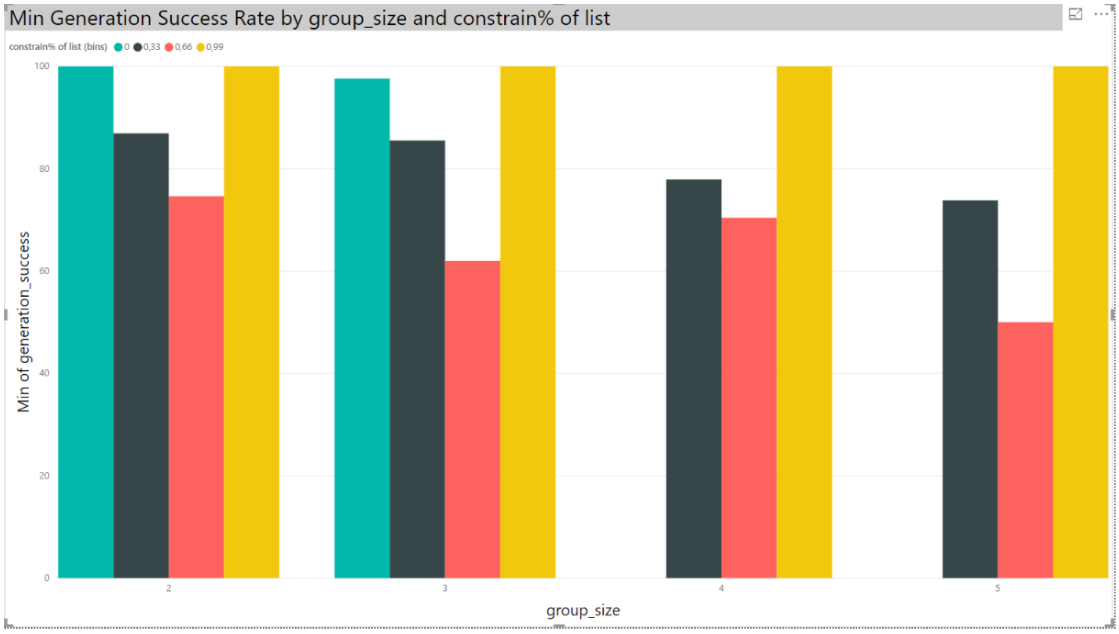
Δεδομένου ότι η ικανοποίηση του constrain του ελάχιστου αριθμού επιθυμητών αντικειμένων ανά χρήστη δεν αποτελεί εύκολο πρόβλημα, εξετάζουμε αρχικά το κατά πόσο μπορεί ο αλγόριθμος να παράγει συστάσεις που ικανοποιούν τους περιορισμούς. Για τον έλεγχο αυτό, δημιουργούνται πρώτα γκρουπ χρηστών, διαφορετικού μεγέθους και τύπου, και υπολογίζεται ανά γκρουπ ο ιδανικός συνδυασμός αντικειμένου που ικανοποιούν τον περιορισμό. Στη συνέχεια, με διαφορετικές τιμές στις παραμέτρους του αλγορίθμου coverage factor,rating factor και boost factor (βάρη), γίνονται πολλαπλές εκτελέσεις για να μπορέσουμε να υπολογίσουμε την ικανότητα του να παράγει συστάσεις σε σχέση με την brute force λύση. Άρα το Success rate υπολογίζεται ως:

$$\text{Success rate} = \frac{Succesfully\ Generated\ Combinations}{Possible\ Combinations\ with\ Brute\ Force} \qquad (13)$$

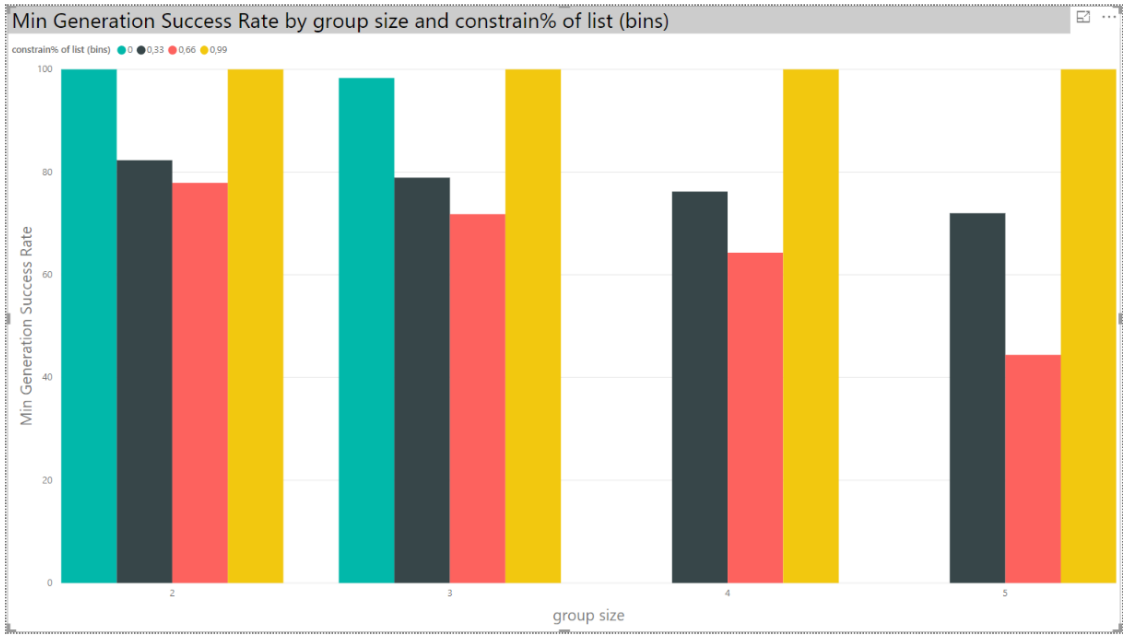
Επιπλέον, λαμβάνουμε υπόψιν στην διαδικασία αυτή και το πόσο περιοριστικό είναι το constrain των αντικειμένων. Πιο συγκεκριμένα, πως ο αριθμός των constrained αντικειμένων ως προς των αριθμό των αντικειμένων που πρέπει να περιλαμβάνονται πως επηρεάζουν το Success Rate. Ο λόγος που το κάνουμε αυτό είναι για να μπορούμε να συγκρίνουμε τα αποτελέσματα ανεξάρτητα από το μέγεθος του γκρουπ και συνεπώς ορίζουμε το παρακάτω μέτρο:

$$Constrain\ \%\ of\ Package = \frac{number\ of\ constrained\ items}{number\ of\ package\ items} \qquad (14)$$

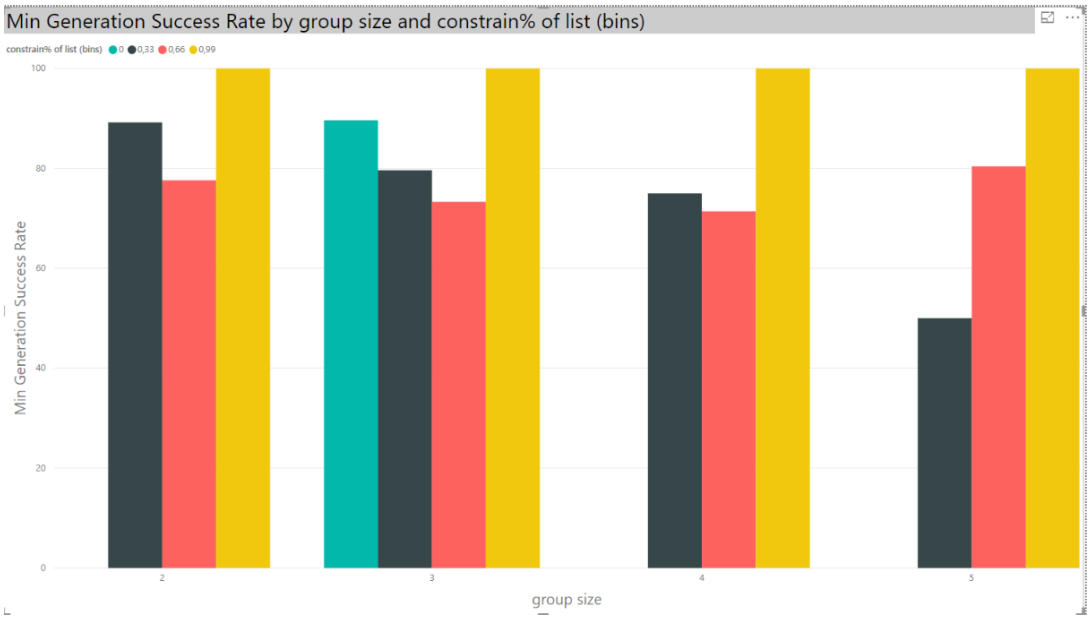
Στα γραφήματα 5 - 7 παρουσιάζεται το ελάχιστο της απόδοσης του αλγορίθμου από το σύνολο των εκτελέσεων για τα τρία διαφορετικά είδη γκρουπ. Για καλύτερη οπτική αναπαράσταση τα αποτελέσματα της σχέσης 14 έχουν ομαδοποιηθεί σε bins με αρχή την τιμή που απεικονίζεται και εύρος 0.33. Από τα παρακάτω γραφήματα παρατηρούμε ότι καθώς αυξάνεται το group size και το constrain % (για constrain % bin <0.99) μειώνεται το success rate του αλγορίθμου, πράγμα το οποίο είναι λογικό δεδομένου ότι αυξάνεται η πολυπλοκότητα της λύσης. Ωστόσο, για constrain % bin = 0.99 έχουμε 100% success rate ανεξαρτήτως group size. Αυτό, προκύπτει από το γεγονός ότι στην περίπτωση όπου έχουμε τον ίδιο αριθμό αντικειμένων constrain και μέγεθος λίστας, αν αυτός ο συνδυασμός αντικειμένων μπορεί να υπάρξει, τότε τα αντικείμενα αυτά έχουν επιλεγεί από όλους χρήστες και είναι «εύκολο» για τον αλγόριθμο να τα επιλέξει.



Γράφημα 5: Αναπαράσταση του Success Rate του αλγορίθμου ανά group size και constrain % bin για γκρουπ ομοιότητας



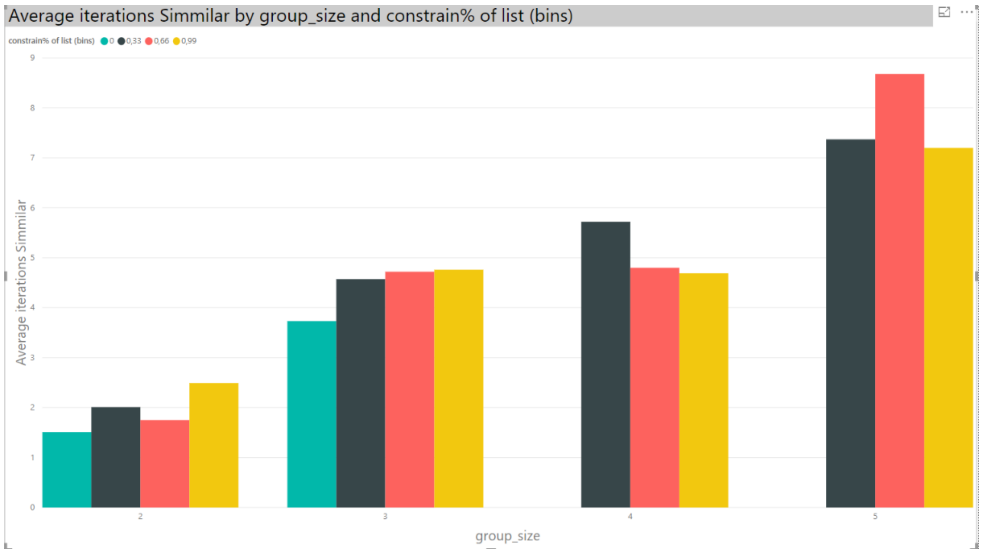
Γράφημα 6: Αναπαράσταση του Success Rate του αλγορίθμου ανά group size και constrain % bin για random γκρουπ



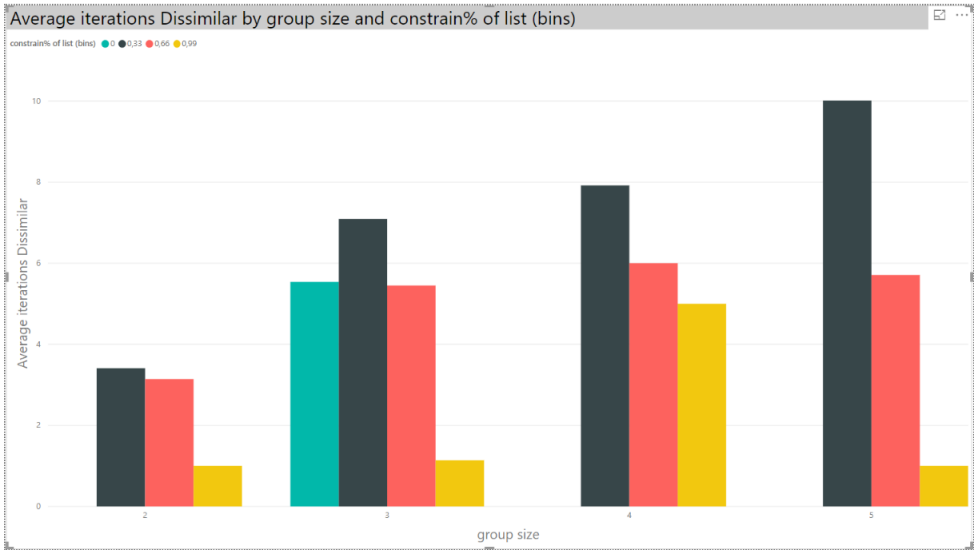
Γράφημα 7 : Αναπαράσταση του Success Rate του αλγορίθμου ανά group size και constrain % bin για γκρουπ διαφορετικότητας

4.3. Ταχύτητα Σύγκλισης Σε Λύση

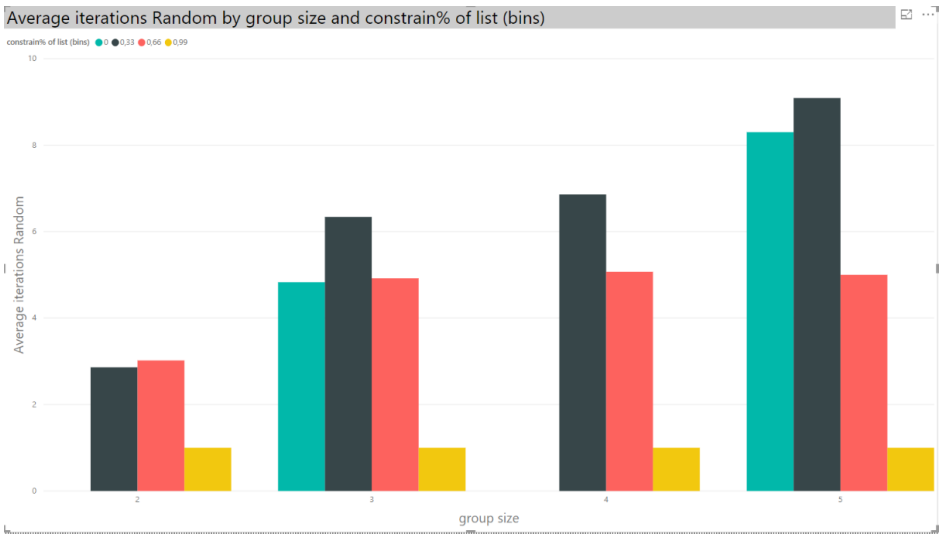
Στα παρακάτω γραφήματα 8 έως 10 παρουσιάζεται ο μέσος αριθμός επαναλήψεων που πρέπει να εκτελέσει ο αλγόριθμος μέχρι τη εύρεση κάποιας βέλτιστης λύσης που ικανοποιεί τους περιορισμούς. Από τα γραφήματα αυτά παρατηρούμε ότι για constrain % bin = 0.33 αυξάνεται το πλήθος των επαναλήψεων με την αύξηση του group size. Επίσης, ενδιαφέρον παρουσιάζει η περίπτωση για constrain % bin = 0.99, το οποίο στην περίπτωση των dissimilar group και των random παραμένει σταθερά μικρό ενώ στην περίπτωση των similar groups αυξάνεται με την αύξηση του μεγέθους του group.



Γράφημα 8: Αναπαράσταση του μέσου αριθμού επαναλήψεων του αλγορίθμου ανά group size και constrain % bin για γκρουπ ομοιότητας

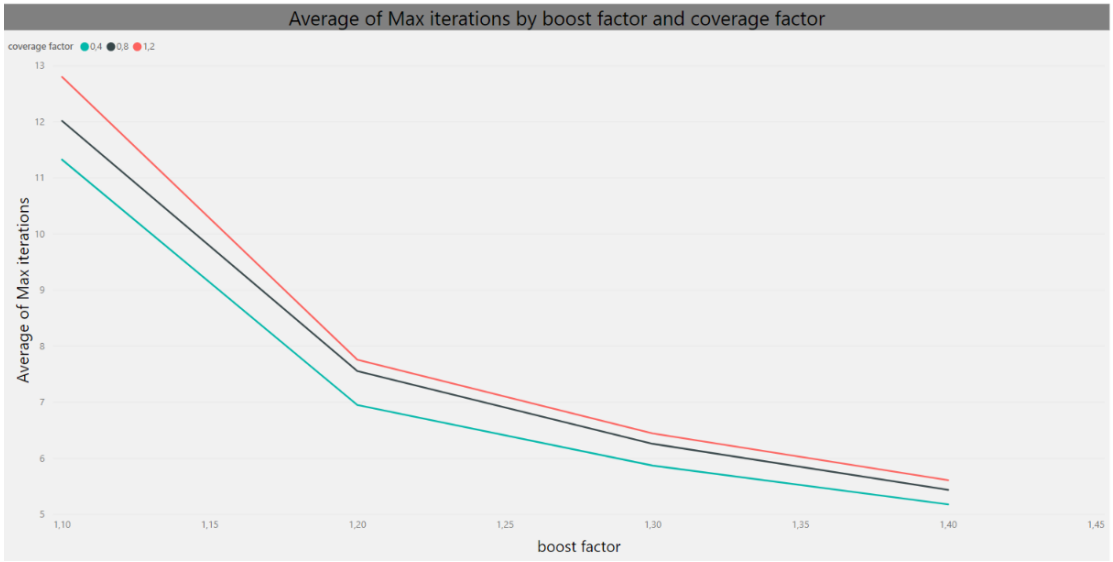


Γράφημα 9:Αναπαράσταση του μέσου αριθμού επαναλήψεων του αλγορίθμου ανά group size και constrain % bin για γκρουπ διαφορετικότητας



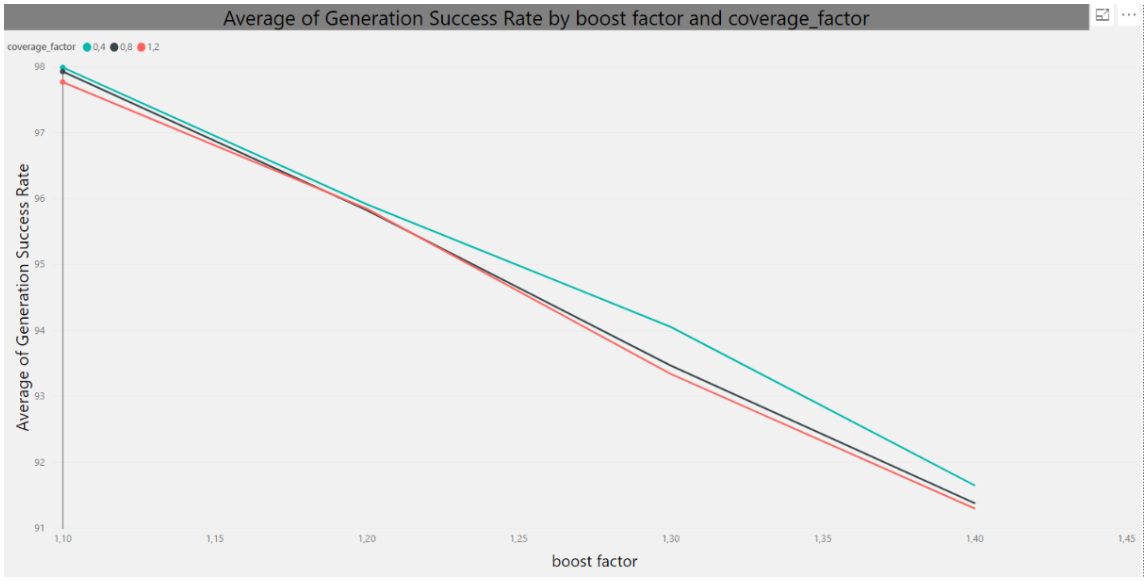
Γράφημα 10:Αναπαράσταση του μέσου αριθμού επαναλήψεων του αλγορίθμου ανά group size και constrain % bin για random γκρουπ

Περεταίρω, μπορούμε να δούμε πως σχετίζεται η παράμετρος αναβαθμολόγησης (boost factor) για την σύγκλιση του αλγορίθμου σε δημιουργία σύστασης. Στο γράφημα 11 το μέσο πλήθος επαναλήψεων που απαιτούνται συναρτήσει του boost factor για τρεις διαφορετικές τιμές coverage factor ανεξάρτητα από τον τύπο ή μέγεθος ομάδας. Από το γράφημα φαίνεται ότι καθώς αυξάνεται η τιμή του boost factor τόσο πιο γρήγορα συγκλίνει ο αλγόριθμός το οποίο όμως έχει αρνητική επίδραση στην ικανότητα σύγκλισης του συστήματος όπως θα δούμε παρακάτω. Επιπλέον, από το γράφημα φαίνεται ότι η υψηλή τιμή του coverage factor αυξάνει το πλήθος επαναλήψεων για την δημιουργία σύστασης από τον αλγόριθμο.



Γράφημα 11: Το μέσο πλήθος επαναλήψεων συναρτήσει του boost factor για διαφορετικές τιμές coverage factor.

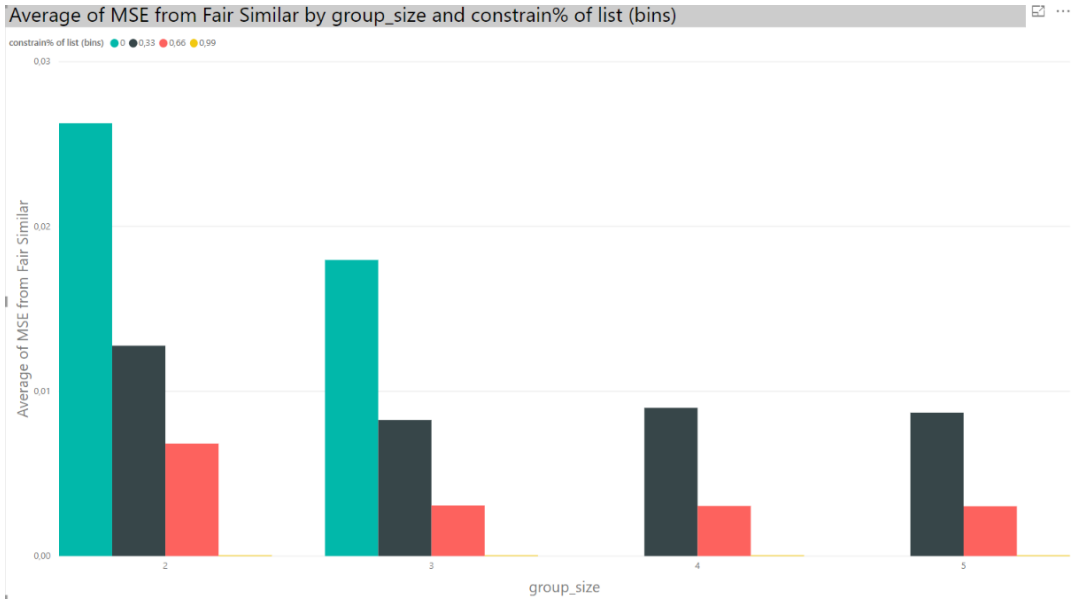
Στο γράφημα 12 απεικονίζεται η μέση τιμή του ποσοστού επιτυχούς παραγωγής αποτελέσματος από τον αλγόριθμο συναρτήσει του boost factor. Από το γράφημα, παρατηρούμε την αρνητική επίδραση του boost factor στην ικανότητα παραγωγής αποτελεσμάτων από τον αλγόριθμο ανεξάρτητα από την τιμή του coverage factor. Αυτό μπορεί να οφείλεται στο γεγονός ότι η υψηλή τιμή της παραμέτρου κατά την διαδικασία αναβαθμολόγησης αντικειμένων για την κάλυψη των περιορισμών του «αδικημένου» χρήστη μπορεί να δημιουργεί πρόβλημα στην κάλυψη των περιορισμών για τους υπολοίπους. Δηλαδή, του σύστημα στην προσπάθεια του να «δικαιώσει» ένα χρήστη να αδικεί τα υπόλοιπα μέλη της ομάδας.



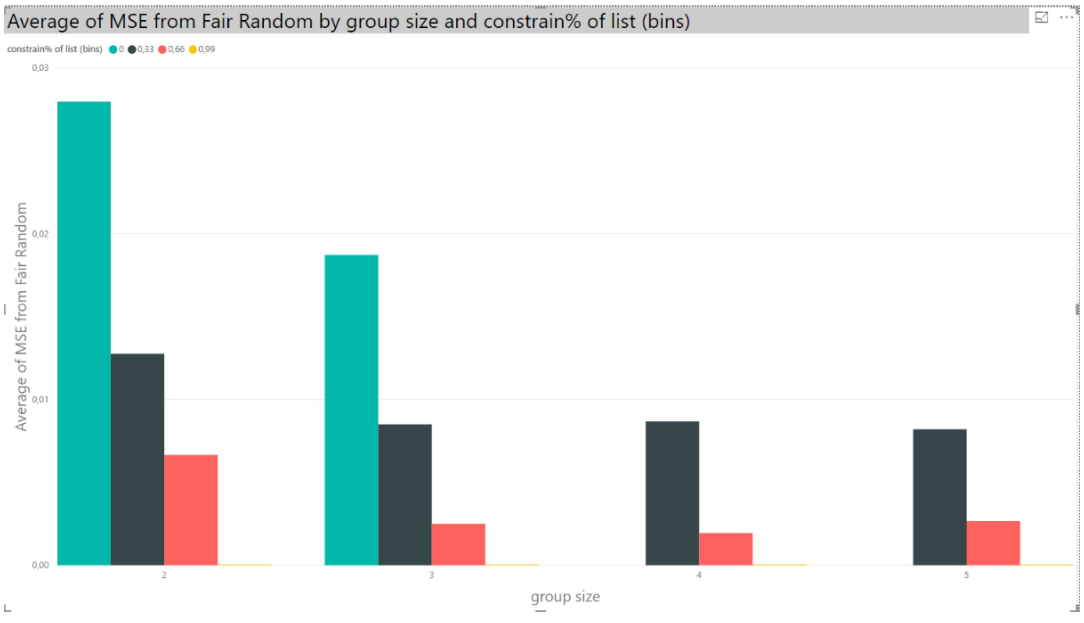
Γράφημα 12: Η μέση τιμή του ποσοστού επιτυχούς παραγωγής αποτελεσμάτων από τον αλγόριθμο συναρτήσει του boost factor για διαφορετικές τιμές coverage factor.

4.4. Απόδοση αλγορίθμου

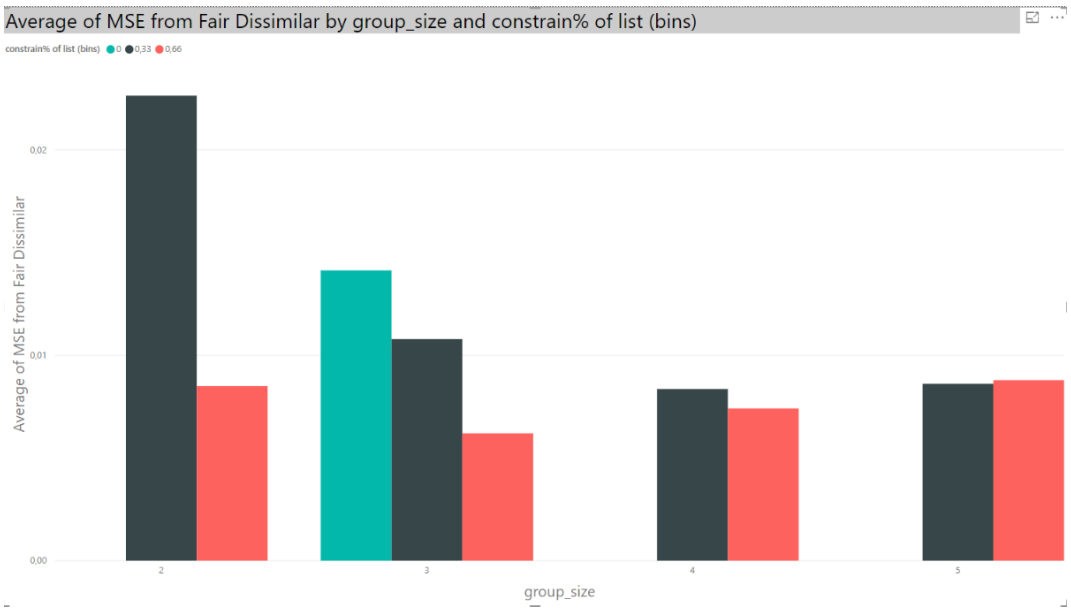
Έχοντας πλέον μελετήσει την δυνατότητα του αλγορίθμου να παράγει συστάσεις που καλύπτουν του περιορισμούς του Fairness, θα εστιάσουμε στο κατά πόσο τα παραγόμενα αποτελέσματα μεγιστοποιούν την ευχαρίστηση των χρηστών κάθε γκρουπ. Σε αντιστοιχία με το βήμα 2 των πειραματικών αποτελεσμάτων, τα αποτελέσματα του αλγορίθμου συγκρίνονται με την ιδανική λύση που έχει προκύψει με brute force υπολογισμούς και καταγράφεται το MSE του user Satisfaction από αυτήν. Στα γραφήματα 13-15 παρατηρούμε ότι για group size μεγαλύτερο του 2 η μέση απόκλιση του αλγορίθμου από την βέλτιστη λύση διατηρείται σχεδόν σταθερή ανεξάρτητα από τον τύπο της ομάδας.



Γράφημα 13:Αναπαράσταση του μέσου MSE δοκιμών του αλγορίθμου ανά group size και constrain % bin για γκρουπ ομοιότητας

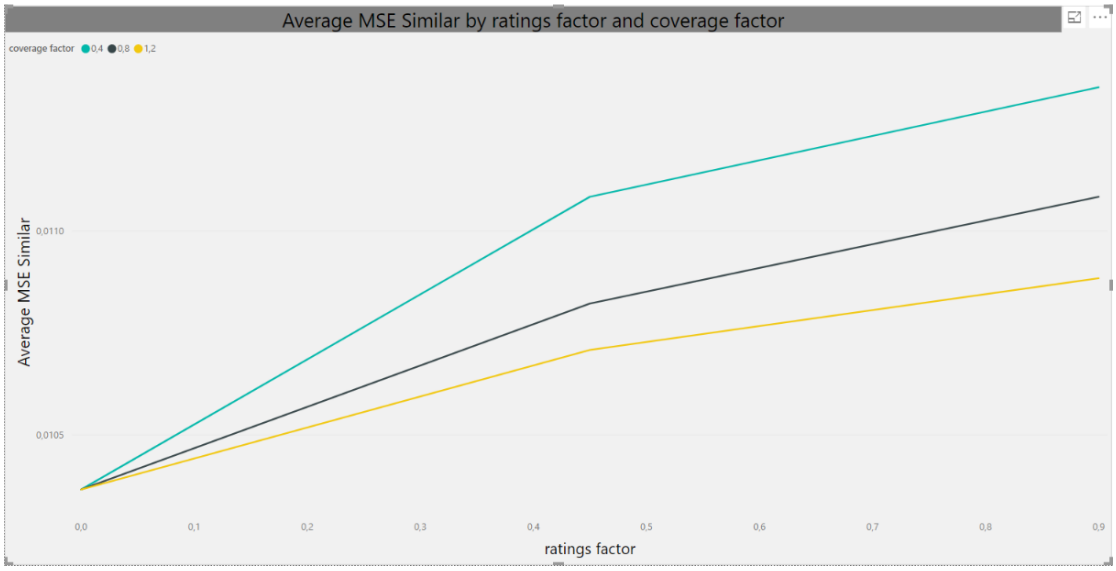


Γράφημα 14: Αναπαράσταση του μέσου MSE δοκιμών του αλγορίθμου ανά group size και constrain % bin για random γκρουπ



Γράφημα 15:Αναπαράσταση του μέσου MSE δοκιμών του αλγορίθμου ανά group size και constrain % bin για γκρουπ διαφορετικότητας

Περαιτέρω, εστιάζουμε στη επίδραση των παραμέτρων του coverage factor και του boost factor στην επίδραση του αλγορίθμου. Στο γράφημα 16 παρατηρούμε ότι καθώς αυξάνεται η τιμή του ratings factor, δηλαδή να ενισχύονται βαθμολογικά αντικείμενα με υψηλό μέσο rating, αποκλίνει περισσότερο κατά μέσο όρο η σύσταση που θα παράγει ο αλγόριθμος από την ιδανική. Αντιθέτως, η αύξηση της τιμής του coverage factor φαίνεται να βελτιώνει την απόδοση του αλγορίθμου.



Γράφημα 16: Η μέση τιμή του MSE συναρτήσει του ratings factor για διαφορετικές τιμές coverage factor.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Η δημιουργία συστάσεων που βασίζονται στο fairness αποτελεί ένα από τα πλέον ενδιαφέροντα ερευνητικά θέματα δεδομένου ότι επεκτείνουν σημαντικά το εύρος των εφαρμογών που μπορούν να χρησιμοποιηθούν τα recommendation systems. Επιπλέον, η μελέτη τέτοιων συστημάτων συνεισφέρει και στην εύρεση νεών προσεγγίσεων για την επίλυση υπολογιστικά σύνθετων προβλημάτων.

Στην παρούσα εργασία έγινε μελέτη πάνω στην ανάπτυξη μιας ευρεστικής προσέγγισης για την αντιμετώπιση των δυσκολιών που εισάγει ο περιορισμός του Fairness σε ένα package to group recommendation system. Η προσέγγιση αυτή βασίστηκε στην χρήση του μέσου rating των αντικειμένων σε επίπεδο ομάδας και του ποσοστού των χρηστών για τους οποίους αποτελούσε βέλτιστη επιλογή το εκάστοτε αντικείμενο σαν χαρακτηριστικά με βάση τα οποία γίνεται η δημιουργία της σύστασης. Επιπλέον, χρησιμοποιήθηκε ένας διορθωτικός παράγοντας, όπου μέσω μιας επαναληπτικής διαδικασίας ήταν εφικτό να επιτευχθεί η εύρεση συνδυασμού αντικειμένων που καλύπτουν τους περιορισμού του fairness. Από τα αποτελέσματα των πειραμάτων, φάνηκε να υπάρχει συσχέτιση μεταξύ των coverage factor,rating factor και boost factor ως προς την ικανότητα παραγωγής αποτελεσμάτων από τον αλγόριθμο, την βελτιστοποίηση των αποτελεσμάτων αλλά και την ταχύτητα σύγκλισης του αλγορίθμου. Ωστόσο, όμως υπάρχουν και άλλες παράμετροι που θα μπορούσαν να χρησιμοποιηθούν αλλά και να μεταβληθεί η μορφή της συνάρτησης αντίστοιχα στα πλαίσια κάποιας μελλοντικής έρευνας ή ακόμα και να γίνει χρήση κάποιου αλγορίθμου μηχανικής μάθησης που να «ελέγχει» τις παραμέτρους δυναμικά για την αξιολόγηση των αντικειμένων.

Επιπλέον, εκτός από την παραμετρική συμπεριφορά του αλγορίθμου, εξετάστηκε και η απόδοση του σε διαφορετικούς τύπους ομάδων. Οι τρείς τύποι ομάδων καλύπτουν σε μεγάλο βαθμό πιθανά σενάρια πραγματικών εφαρμογών όπου οι χρήστες έχουν παρόμοιες προτιμήσεις έως εντελώς διαφορετικά ενδιαφέροντα και ανέδειξαν περιπτώσεις όπου ο τύπος της ομάδας επηρέαζε την απόδοση του αλγορίθμου. Ενδιαφέρον για μελλοντική έρευνα η μελέτη της επίδρασης σχέσεων χρηστών από κοινωνικά (πχ φίλοι, βαθμός εμπιστοσύνης), οι οποίες θα λαμβάνονταν σαν είσοδο από τον αλγόριθμο και θα όριζαν κάποια προτεραιότητα ως προς την ικανοποίηση των μελών της ομάδας.

Περαιτέρω, θα μπορούσε να μελετηθεί η απόδοση του συστήματος ως προς την ικανότητα δημιουργίας συστάσεων αλλά και την μεγιστοποίηση της ευχαρίστησης των χρηστών αν ο περιορισμός του Fairness αποτελούσε ένα soft constrain το οποίο θα μπορούσε να παραβιασθεί μερικώς κατά την δημιουργία συστάσεων.

Τέλος, ενδιαφέρον θα ήταν να προστεθούν επιπλέον περιορισμοί στο σύστημα, όχι μόνο από την πλευρά των χρηστών(C-fairness) αλλά και από την πλευρά του συστήματος που κάνει τις συστασεις (P-fairness) όπως η περίπτωση του item coverage και να διερευνηθεί η απόδοση στα πλαίσια μιας εφαρμογής.

6. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Agarwal, A., Chauhan, M., & Ghaziabad (2017). Similarity Measures used in Recommender Systems:A Study.

[2] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* 22, 1 (January 2004), 5-53.

[3] Matev Kunaver and Toma Porl. 2017. Diversity in recommender systems A survey. *Know.-Based Syst.* 123, C (May 2017), 154-162.

[4] Prasad, RVVSV. (2012). A Categorical Review of Recommender Systems. International Journal of Distributed and Parallel systems. 3. 73-83. 10.5121/ijdps.2012.3507.

[5] Mohan, Geetha & Iqubal, Safa & Fancy, C & Saranya, D. (2018). A Hybrid Approach using Collaborative filtering and Content based Filtering for Recommender System. Journal of Physics: Conference Series. 1000. 012101. 10.1088/1742-6596/1000/1/012101.

[6] Felfernig, Alexander & Friedrich, Gerhard & Jannach, Dietmar & Zanker, Markus. (2015). Constraint-Based Recommender Systems. 10.1007/978-1-4899-7637-6_5.

[7] Dimitris Serbos, Shuyao Qi, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2017. Fairness in Package-to-Group Recommendations. In *Proceedings of the 26th International Conference on World Wide Web* (WWW '17). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 371-379.

[8] Lin Xiao, Zhang Min, Zhang Yongfeng, Gu Zhaoquan, Liu Yiqun, and Ma Shaoping. 2017. Fairness-Aware Group Recommendation with Pareto-Efficiency. In *Proceedings of the Eleventh ACM Conference on Recommender Systems* (RecSys '17). ACM, New York, NY, USA, 107-115.

[9] Burke, Robin D. “Multisided Fairness for Recommendation.” *CoRR* abs/1707.00093 (2017)

[10] Serbos, Dimitris, Shuyao Qi, Nikos Mamoulis, Evaggelia Pitoura and Panayiotis Tsaparas. “Fairness in Package-to-Group Recommendations.” *WWW* (2017).

[11] Amirali Salehi-Abari and Craig Boutilier. 2015. Preference-oriented Social Networks: Group Recommendation and Inference. In *Proceedings of the 9th ACM Conference on Recommender Systems* (RecSys '15). ACM, New York, NY, USA, 35-42.

[12] 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.

7. ΠΑΡΑΡΤΗΜΑ

Παρακάτω παρατίθενται ορισμένα κομμάτια από τον κώδικα python που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας.

7.1. mult_exp_controller.py

Το συγκεκριμένο script της python διαχειριζόταν την διαδοχική εκτέλεση πειραμάτων για διαφορετικές παραμέτρους του ευρεστικού αλγορίθμου. Ορίζοντας τον τύπο του γκρουπ των χρηστών, το εύρος τιμών των παραμέτρων boost_factor, coverage_factor, ratings_factor καθώς και το βήμα μεταβολής τους αναλάμβανε την διαδοχική εκτέλεση πειραμάτων.

```
from Upini_thesis_project.Config import Config
from Upini_thesis_project.Utilities.Datasource import Datasource
from Upini_thesis_project.entities.GroupGenerator import GroupGenerator
from Upini_thesis_project.Utilities import Calculations, Utils
from Upini_thesis_project.Utilities.GreedyTrain import optimize_greedy
from time import sleep
from Upini_thesis_project.Utilities.Logger import Logger
import numpy as np
import time

Export_data_for_optimization_analysis = False
Multirun = True

boost_factor_min = 1.1
boost_factor_max = 1.5
boost_factor_stp = 0.1
coverage_factor_min = 0.4
coverage_factor_max = 1.3
coverage_factor_stp = 0.4
ratings_factor_min = 0.0
ratings_factor_max = 1.0
ratings_factor_stp = 0.45

# Load Settings and Start Logging process
cfg = Config()

# group_type = ['similar','dissimilar','random']
group_type = ['random']
fairn = ['least_misery', 'variance', 'min_max_ratio']
top_it = range(1, 8, 2)
rec_list = range(2, 8, 2)
const = range(2, 6)
total_test = len(group_type) * len(top_it) * len(rec_list) * len(const) * len(fairn)
test_num = 0
starttime = time.clock()
for gt in group_type:
```

```
for fr in fairn:
    for ti in top_it:
        for rl in rec_list:
            for ct in const:

                if ti >= ct and rl >= ct:
                    test_num += 1
                    time_est = round(((time.clock() - starttime) / test_num) * (total_test - test_num + 1), 2)
                    print('\n\n=================================================================')
                    print('type:', gt, 'fair:', fr, 'top_items:', ti, 'rec_list_size:', rl, 'constrained_items:',
                          ct)
                    print('test_run:', test_num, '/', total_test)
                    print('eta:', time_est)
                    print('=================================================================\n')
                    cfg.group_type = gt
                    cfg.fairness_measure = fr

                    cfg.number_of_top_items = ti
                    cfg.number_of_rec_items = rl
                    cfg.number_of_min_covered_items = ct

                    cfg.greedy_coverage_factor = 0.82
                    cfg.greedy_ratings_factor = 0.09
                    cfg.boost_factor = 1.3

                    cfg.log_file_name()
                    log = Logger(cfg)

                    log.log_task('Loading data from recommender system')
                    dataset = Datasource(cfg.dataframe_dir, cfg.constrain_dir)
                    dataset.get_users()
                    dataset.get_items()
                    item_stats = dataset.items_stats_map
                    all_users_map = dataset.index_to_user_obj_map

                    log.log_task('Load for each user the available')
                    Utils.load_possible_items(all_users_map, dataset.constrains, dataset.dataframe)

                    log.log_task('Generate groups of users')
                    group = GroupGenerator(dataset, cfg.group_size)

                    if cfg.group_type == 'similar':
                        group.generate_similar_group()
                    elif cfg.group_type == 'dissimilar':
                        group.generate_dissimilar_group()
                    elif cfg.group_type == 'random':
                        group.generate_random_group()

                    log.log_task('Load for each user the top items')
                    Utils.select_top_items(cfg.number_of_top_items, all_users_map, group.group_map)
```

```
            log.log_task('Generate for its group the list of recommendable items')
            Utils.create_group_recommendation_list_of_available_items(group.group_map, all_users_map)
            sleep(0.3)

            print('==================Calculations=====================================')

            log.log_task('combination_test_brute')
            Calculations.combination_test_brute(group.group_map, all_users_map, log, cfg)
            sleep(0.1)

            if Multirun:
              log.log_task('Multirun_test_greedy')
              for boost_factor in np.arange(boost_factor_min, boost_factor_max, boost_factor_stp):
                for coverage_factor in np.arange(coverage_factor_min, coverage_factor_max,
                                                 coverage_factor_stp):
                  for ratings_factor in np.arange(ratings_factor_min, ratings_factor_max,
                                                  ratings_factor_stp):
                    #
print('boost_factor:',boost_factor,'coverage_factor:',coverage_factor,'ratings_factor:',ratings_factor)
                    cfg.boost_factor = boost_factor
                    cfg.greedy_coverage_factor = coverage_factor
                    cfg.greedy_ratings_factor = ratings_factor

                    Calculations.combination_test_greedy(group.group_map, all_users_map, log, cfg)
                    Calculations.greedy_algorith_deviation(group.group_map, log, cfg)


            else:
              log.log_task('combination_test_greedy')
              Calculations.combination_test_greedy(group.group_map, all_users_map, log,
                                                   cfg)
              Calculations.greedy_algorith_deviation(group.group_map, log)

            if Export_data_for_optimization_analysis:
              log.log_task('optimize_greedy')
              optimize_greedy(group.group_map, all_users_map, log, cfg)

            log.log_task('top_combination_analysis')
            Calculations.top_combination_analysis(group.group_map, item_stats, log,
                                                  all_users_map)

            log.end()
          else:
            test_num += 1
            print('==NOT POSSIBLE CONSTRAIN COVERAGE==')
```

7.2. Config.py

Στο Config.py υλοποιήθηκε η κλάση Config όπου σε κάθε χαρακτηριστικό ήταν αποθηκευμένες απαραίτητες πληροφορίες για την εκτέλεση των πειραμάτων. Μερικά από αυτά είναι:

- Ορισμός θέσης του source αρχείου και οι κολώνες που πρέπει να γίνουν import.
- Φίλτρα στο αρχικό dataset για το import συγκεκριμένων περιπτώσεων χρηστών και αντικειμένων.
- Ορισμός των παραμέτρων των επιλεγμένων αντικειμένων για την δημιουργία των συστάσεων.
- Ορισμός του ορίου του περιορισμού για την δημιουργία συστάσεων

```
class Config:

    def __init__(self):

        '''
        Import dataset settings:
            1.dataset dir
            2.Imported csv files have the format user_id,item_id,rating.The rec_csv_read_indexes idfentifies the
position of each entity

        '''
        self.dataset_dir =
'/home/dimos/PycharmProjects/Py_projects/Upini_thesis_project/files/1_raw_data/movielens/u.data'
        self.csv_r_ind = {'user_id': 1, 'item_id': 2, 'rating': 3}



        '''
        Filtering data settings, Recommender split sets and show visualizations setting
        '''

        self.min_std_of_item_ratings = 0.7
        self.min_num_of_item_ratings = 50
        self.min_std_of_user_ratings = 0.5
        self.min_num_of_user_ratings = 50
        self.test_size = 0.35
        self.show_charts = True



self.base_dataframe_dir='/home/dimos/PycharmProjects/Py_projects/Upini_thesis_project/files/2_processed/da
taframes/'

        '''
        group_type Settings
                1.similar
                2.dissimilar
```

```
            3.random
    '''
    self.group_type = 'dissimilar'


    '''
    Combination Test Settings:
        1.group_size : set how many users are contained in a group
        2.number_of_top_items: the number of top K items that would be recommendable to a user
                        from all the available items
        3.rec_repeatability_of_item : is the max number of users that can get the same item
        4.the percentage of users who have seen the item
    '''

    self.group_size = 5
    self.number_of_top_items = 3
    self.number_of_rec_items = 4
    self.threshold_cov=0.99
    self.number_of_min_covered_items = 1

    self.greedy_coverage_factor = 0.82
    self.greedy_ratings_factor = 0.09
    self.boost_factor = 1.3
    self.max_iterations = 40

    '''
    Fairness_measure Settings
        1.least_misery
        2.variance
        3.min_max_ratio
    '''
    self.fairness_measure = 'min_max_ratio'


    '''
    Dynamically generated file names
    '''
    self.log_dir = ''
    self.dataframe_fname = ''
    self.dfrm_file_name()
    self.dataframe_dir = self.dataframe_fname
    self.constrain_dir = self.dataframe_fname+'_constrains'
    # self.dataframe_dir = 'files/1_raw_data/dt'
    # self.constrain_dir = 'files/1_raw_data/constrains' # test dirs
    self.log_file_name()

def __str__(self):

    string = '=======================TEST SETTINGS======================='
    for prop in self.__dict__:
        if prop != 'csv_r_ind':
```

```
            string += '\n' + prop + ' : ' + str(self.__dict__[prop])

        string += '\n' + '============================================================' + '\n'
        return string


    def dfrm_file_name(self):

        string = 'df_'
        string += 'itms_' + str(self.min_std_of_item_ratings)
        string += '_itmn_' + str(self.min_num_of_item_ratings)
        string += '_urms_' + str(self.min_std_of_user_ratings)
        string += '_urmn_' + str(self.min_num_of_user_ratings)
        string += '_split_' + str(self.test_size)

        self.dataframe_fname = self.base_dataframe_dir + string

    def log_file_name(self):

        string = '/home/dimos/PycharmProjects/Py_projects/Upini_thesis_project/files/2_processed/'
        string += self.fairness_measure
        string += '_grp_tp_' + self.group_type
        string += '_grp_sz_' + str(self.group_size)
        string += '_top_it_' + str(self.number_of_top_items)
        string += '_item_rec_' + str(self.number_of_rec_items)
        string += '_item_con_' + str(self.number_of_min_covered_items)
        string += '_userfact_' + str(self.greedy_coverage_factor)
        string += '_ratefact_' + str(self.greedy_ratings_factor)
        string += '_boostfact_' + str(self.boost_factor)

        string += '.txt'
        self.log_dir = string
```

7.3. Uesr.py

Κλάση που κρατάει πληροφορίες σε επίπεδο χρήστη.

class User(object):

```
    def __init__(self, user_id):
        self.id = user_id
        #self.constrain_flag = 1
        '''
        for the possible items get:
        1 the map item-rating
        2 the map of top_item-rating
        2 the satisfaction_factor that s = SUM(Ratings of top Items)
        '''
        self.map_possible_items_ratings = {}
        self.map_top_items_ratings = {}
```

```
        self.satisfaction_factor = -1
```

7.4. GroupOfUsers.py

Κλάση που κρατάει πληροφορίες σε επίπεδο ομάδας χρηστών.

```
'''
The User_Group class the holds all the properties per Group
and stat results
'''
class GroupOfUsers:

    def __init__(self, group_id, user_list):
        self.id = group_id
        self.users = user_list
        self.user_map = {}

        self.rlist_of_items = []
        '''
        self.result_obj contains:
        :keys the item combination
        :params
        '''
        self.result_obj = {}
        self.best_combination = {}
```

7.5. movielens_data_prep2.py

Script για την επεξεργασία των αρχικών δεδομένων, το οποίο φίλτραρε περιπτώσεις χρηστών και αντικειμένων που δεν πληρούσαν τα απαιτούμε κριτήρια. Επίσης, δημιουργεί γραφήματα με την στατιστική συμπεριφορά των δεδομένων.

```
from Upini_thesis_project.Config import Config
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from Upini_thesis_project.Utilities.Recommender import collaborative_filtering

from matplotlib.pyplot import axis
cf = Config()
plt.interactive(False)
sns.set_style('white')

#Variables
data_dir = os.path.relpath(cf.dataset_dir)

min_std_of_item_ratings = cf.min_std_of_item_ratings
min_num_of_item_ratings = cf.min_num_of_item_ratings
```

```
min_num_of_user_ratings = cf.min_num_of_user_ratings
min_std_of_user_ratings = cf.min_std_of_user_ratings
print('==========Start Loading Raw data==========')


'''
Load movielens dataset into a dataframe and merge with movie
'''
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv(data_dir , sep='\t', names=column_names)

print('Filter users with less than', min_num_of_user_ratings, 'ratings')


n_users = df.user_id.nunique()
n_items = df.item_id.nunique()
print('init_n_users',n_users)
print('init_n_items',n_items)

'''
Filter Items
   1.count ratings grouped by Item and create new dataframe
   2. filter out Items with less ratings than threshold (min_std_of_item_ratings)
   3. filter out Items with less ratings_std than threshold (min_std_of_item_ratings)
   4. Join with initial dataset (inner join) to keep only items with the above constrains
   5. Drop aggregated columns
'''
grouped_by_item_stats = pd.DataFrame(df.groupby('item_id')['rating'].std())
grouped_by_item_stats.rename(columns={'rating': 'rating_std'}, inplace=True)
grouped_by_item_stats['num_of_ratings'] = pd.DataFrame(df.groupby('item_id')['rating'].count())
p1 = sns.jointplot(x='rating_std', y='num_of_ratings', data=grouped_by_item_stats, alpha=0.5)
p1.fig.subplots_adjust(top=0.9)
p1.fig.suptitle('Initial Item Stats', fontsize=12)

grouped_by_item_stats = grouped_by_item_stats[grouped_by_item_stats['rating_std'] >
min_std_of_item_ratings]
grouped_by_item_stats = grouped_by_item_stats[grouped_by_item_stats['num_of_ratings'] >
min_num_of_item_ratings]
p2 = sns.jointplot(x='rating_std', y='num_of_ratings', data=grouped_by_item_stats, alpha=0.5)
p2.fig.subplots_adjust(top=0.9)
p2.fig.suptitle('Filtered Item Stats', fontsize=12)

df = pd.merge(df, grouped_by_item_stats, left_on='item_id', right_index=True)
df = df.drop(['rating_std','num_of_ratings'], axis=1)

print('df',df)

n_users = df.user_id.nunique()
n_items = df.item_id.nunique()

print('n_users',n_users)
```

```
print('n_items',n_items)


'''
Filter Users
   1.count ratings grouped by Users and create new dataframe
   2. filter out Users with less ratings than threshold (min_std_of_user_ratings)
   3. filter out Users with less ratings_std than threshold (min_std_of_user_ratings)
   4. Join with initial dataset (inner join) to keep only user with the above constrains
   5. Drop aggregated columns
'''
grouped_by_user_stats = pd.DataFrame(df.groupby('user_id')['rating'].std())
grouped_by_user_stats.rename(columns={'rating': 'rating_std'}, inplace=True)
grouped_by_user_stats['num_of_ratings'] = pd.DataFrame(df.groupby('user_id')['rating'].count())
p3 = sns.jointplot(x='rating_std', y='num_of_ratings', data=grouped_by_user_stats, alpha=0.5)
p3.fig.subplots_adjust(top=0.9)
p3.fig.suptitle('Initial User Stats', fontsize=12)

grouped_by_user_stats = grouped_by_user_stats[grouped_by_user_stats['rating_std'] >
min_std_of_user_ratings]
grouped_by_user_stats = grouped_by_user_stats[grouped_by_user_stats['num_of_ratings'] >
min_num_of_user_ratings]
p4 = sns.jointplot(x='rating_std', y='num_of_ratings', data=grouped_by_user_stats, alpha=0.5)
p4.fig.subplots_adjust(top=0.9)
p4.fig.suptitle('Filtered User Stats', fontsize=12)

df = pd.merge(df, grouped_by_user_stats, left_on='user_id', right_index=True)
df = df.drop(['rating_std','num_of_ratings'], axis=1)

print('df',df)

n_users = df.user_id.nunique()
n_items = df.item_id.nunique()

print('n_users',n_users)
print('n_items',n_items)

if cf.show_charts:
   plt.show()
'''
   Initialize Recommender
'''
collaborative_filtering(df,cf )
```

7.6. Recommender.py

```
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import pairwise_distances
```

```
from Upini_thesis_project.Utilities.Utils import
entity_to_index_map,utility_matrix_populate,index_to_entity_map,export_array_to_csv_dataframe
from Upini_thesis_project.Utilities.RecomCalculations import generate_prediction_matrix,calculate_RMSE
import numpy as np


def collaborative_filtering(raw_data_dataframe,cf):

    df = raw_data_dataframe


    print('=============Initialize Recommender=============')

    '''
    1.Get number of unique users and items
    2.Get list of unique users and items
    3.Generate dictionary with indexes
    '''

    n_users = df.user_id.nunique()
    n_items = df.item_id.nunique()
    print('n_users', n_users)
    print('n_items', n_items,'\n')

    user_list = df.user_id.unique()
    item_list = df.item_id.unique()

    user_index_map = entity_to_index_map(user_list)
    item_index_map = entity_to_index_map(item_list)

    index_user_map = index_to_entity_map(user_list)
    index_item_map = index_to_entity_map(item_list)

    '''
    1. Split data to test and trainset
    2. Populate 2 utility matrixes (train/test)
    3. Calculate user similarity matrix
    '''



    print('Split dataset to train/test:', 1-cf.test_size,'/',cf.test_size)
    train_data, test_data = train_test_split(df, test_size=cf.test_size)

    train_data_matrix = np.zeros((n_users, n_items))
    test_data_matrix = np.zeros((n_users, n_items))

    train_data_matrix = utility_matrix_populate(train_data_matrix, user_index_map, item_index_map, train_data,
cf)
    test_data_matrix = utility_matrix_populate(train_data_matrix, user_index_map, item_index_map, train_data,
cf)
```

```
print('Calculate User-similarity Matrix')
user_similarity = pairwise_distances(train_data_matrix, metric='cosine')

'''
1. Generate prediction matrix for users
2. Test accuracy
'''
print('Calculate Predicted Values Matrix')
user_prediction = generate_prediction_matrix(train_data_matrix, user_similarity)

print('Calculate RMSE','\n')
rmse_error = calculate_RMSE(user_prediction, test_data_matrix)

print('User-based CF RMSE: ' + str(rmse_error))

export_array_to_csv_dataframe(cf.dataframe_fname, user_prediction, test_data_matrix, index_item_map, index_user_map)
```

7.7. RecomCalculations.py

Περιλαμβάνει όλους τους υπολογισμούς που είναι απαραίτητοι για την λειτουργία του recommender.

```
from sklearn.metrics import mean_squared_error

from math import sqrt

import numpy as np

'''

Recommender: Generate the predictions from the initial utility matrix and the similarity matrix

'''



def generate_prediction_matrix(initial_utility_matrix, entity_similarity_matrix, recommendation_type='user'):

    '''

    :param initial_utility_matrix: the utility matrix as it was loaded from the raw data

    :param entity_similarity_matrix: the table that contains the similarity between users

    :param recommendation_type: select user for user-user or item for item collaborative filtering default = 'user'

    :return: a fully completed utility matrix with calculated values
```

```
    '''

    if recommendation_type == 'user':
        '''
        1. For each user calculate average rating behaviour..
        2. Convert average rating shape by adding a dimension m-> m x 1
        3. Calculate rating difference rating-mean_rating
        4. Calculate the product of the entity_similarity_matrix matrix of users with normalized
           initial_utility_matrix(ratings_diff) row-> column and sum to calculate each rating
        5. Calculate normalization factor of each predicted rating for each movie
        6.Calculate matrix with predictions
        '''
        mean_user_rating = initial_utility_matrix.mean(axis=1)
        mean_user_rating_dim = mean_user_rating[:, np.newaxis]
        ratings_diff = (initial_utility_matrix - mean_user_rating_dim)
        table_product = np.dot(entity_similarity_matrix, ratings_diff)
        normalizing_factor = np.array([np.abs(entity_similarity_matrix).sum(axis=1)]).T
        matrix_of_predicted_ratings = mean_user_rating_dim + (table_product / normalizing_factor)

    elif recommendation_type == 'item':
        table_product = np.dot(initial_utility_matrix, entity_similarity_matrix)
        normalizing_factor = np.array([np.abs(entity_similarity_matrix).sum(axis=1)])
        matrix_of_predicted_ratings = table_product / normalizing_factor

    else:
        return []
    return matrix_of_predicted_ratings

'''
Recommender: Metric to calculate Recommender accuracy
```

```
    '''
    def calculate_RMSE(matrix_with_predictions, matrix_with_test_data):
        '''
        :param matrix_with_predictions: np_array
        :param matrix_with_test_data: np_array
        :return: RMSE

        1. Get the indexes of test values (rows-columns) from the test data matrix of the non-zero elements
        2. Get the values of the calculated ratings based on the indexes_of_test_values
        3. Get the values of the test ratings based on the indexes_of_test_values
        4. Calculate RMSE
        '''
        indexes_of_test_values = matrix_with_test_data.nonzero()
        predicted_values = matrix_with_predictions[indexes_of_test_values]
        test_values = matrix_with_test_data[indexes_of_test_values]

        return sqrt(mean_squared_error(predicted_values, test_values))
```

7.8. Calculations.py

Περιλαμβάνει συναρτήσεις που υπολογίζουν:

- Τον βέλτιστο συνδυασμού αντικειμένων με brute force.
- Τον βέλτιστο συνδυασμού αντικειμένων με τον ευρεστικό αλγόριθμο.
- Μετρικά όπως η κάλυψη των χρηστών από ένα αντικείμενο ή το score ενός συνδιασμού

```
from Upini_thesis_project.Utilities import Utils
from Upini_thesis_project.Utilities.ProgressBar import ProgressBar
from sklearn.metrics import mean_squared_error
from math import sqrt
import numpy as np
import time

'''
1.Iterate over all groups
2.Generate combinations based on the number of users
3.Iterate over each combination and check if the item should be given to each user based on the list of available objects
4.Keep valid combinations and their reting in an dictionary for further processing
'''
```

```
def combination_test_brute(groups_map, users_map, log, conf_object):
    size_of_recommendation_list = conf_object.number_of_rec_items
    threshold_cov = conf_object.threshold_cov
    fairness_mes = conf_object.fairness_measure
    min_covered_items = conf_object.number_of_min_covered_items
    number_of_all_groups = len(list(groups_map.keys()))
    log.log_static_metric('number_of_all_groups', number_of_all_groups)
    log.log_static_metric('fairness_mes', fairness_mes)

    progress = ProgressBar(number_of_all_groups, fmt=ProgressBar.FULL)

    print('number_of_all_groups:', number_of_all_groups)  #:todo ADD LOGGING OF GENERATED/valid
combinations

    for group_id in groups_map:
        users_ids = groups_map[group_id].users
        item_combination_list = Utils.combinations_generator(groups_map[group_id].rlist_of_items,
                                            size_of_recommendation_list)

        progress.current += 1
        progress()

        total_combinations = 0
        valid_combinations = 0
        best_comb = None
        best_score = -1

        fair_comb = None
        fair_score = -1

        for comb in item_combination_list:
            metric = {}
            total_combinations += 1
            tmp_satisfaction_map = {}
            tmp_usr_coverage_map = {}
            for user_id in users_ids:
                tmp_satisfaction_map[user_id] = 0
                tmp_usr_coverage_map[user_id] = 0

            for item in comb:
                item_avail = 0

                for user_id in users_ids:
                    user_obj = users_map[user_id]

                    if item not in user_obj.map_possible_items_ratings:
                        item_avail += 1

                    elif item in user_obj.map_top_items_ratings:
```

```
                rating = user_obj.map_top_items_ratings[item]
                tmp_satisfaction_map[user_id] += rating / user_obj.satisfaction_factor
                tmp_usr_coverage_map[user_id] += 1

            if item_avail / len(users_ids) > threshold_cov:
                break

        result_list = list(tmp_satisfaction_map.values())
        metric[fairness_mes] = metric_calculation(result_list, fairness_mes)

        if user_coverage_check(tmp_usr_coverage_map, min_covered_items):
            valid_combinations += 1
            if fair_score < metric[fairness_mes]:
                fair_score = metric[fairness_mes]
                fair_comb = comb

        if best_score < metric[fairness_mes]:
            best_score = metric[fairness_mes]
            best_comb = comb

    groups_map[group_id].result_obj['best_comb'] = [best_comb, best_score]
    groups_map[group_id].result_obj['fair_comb'] = [fair_comb, fair_score]
    # LOG result
    header = 'group_id,total_combinations,valid_combinations'
    key = str(group_id) + ',' + str(total_combinations)
    # log.log_dynamic_metric(header, key, valid_combinations)

  progress.done()


def metric_calculation(satisfaction_list, fairness_mes):
  if fairness_mes == 'least_misery':
    return min(satisfaction_list)
  elif fairness_mes == 'variance':
    return 1 / (np.var(satisfaction_list) + 1)
  elif fairness_mes == 'min_max_ratio':
    return min(satisfaction_list) / max(satisfaction_list)


def user_coverage_check(coverage_map, min_covered_items):
  for user in coverage_map:
    covered_items = coverage_map[user]
    if covered_items < min_covered_items:
      return False
  return True


def combination_test_greedy(groups_map, users_map, log, conf_object):
  n = conf_object.number_of_rec_items
  ratings_factor = conf_object.greedy_ratings_factor
  coverage_factor = conf_object.greedy_coverage_factor
```

```
fairness_mes = conf_object.fairness_measure
min_covered_items = conf_object.number_of_min_covered_items
boost_factor = conf_object.boost_factor
max_iterations = conf_object.max_iterations

# log.log_static_metric('min_covered_items', min_covered_items)
# log.log_static_metric('ratings_factor', ratings_factor)
# log.log_static_metric('coverage_factor', coverage_factor)

start = time.clock()
for group_id in groups_map:
    users_ids = groups_map[group_id].users
    tmp_item_stats = {}
    tmp_item_score = {}
    iterations_list = []

    # Get Score features for each item
    for user_id in users_ids:
        user_obj = users_map[user_id]
        item_ratings = user_obj.map_top_items_ratings

        for item in item_ratings:
            if item not in tmp_item_stats:
                tmp_item_stats[item] = {'users': 1 / len(users_ids), 'ratings': [item_ratings[item]]}

            else:
                tmp_item_stats[item]['users'] += 1 / len(users_ids)
                tmp_item_stats[item]['ratings'].append(item_ratings[item])

    for item in tmp_item_stats:
        tmp_item_score[item] = greedy_algorith_score_function(ratings_factor, coverage_factor,
                                              tmp_item_stats[item]['users'],
                                              tmp_item_stats[item]['ratings'])

    # ITERATE until constrain reached
    score = -1
    iterations = 0
    sol_iter = max_iterations
    delta_threshold = 0.5
    delta = -1
    best_score = -2
    best_comb = []

    while (score < 0 or delta > delta_threshold) and max_iterations > iterations:
        iterations += 1

        top = {k: tmp_item_score[k] for k in sorted(tmp_item_score, key=lambda k: -tmp_item_score[k])[:n]}
        comb = list(top.keys())
        score, user_id = calculate_combination_score(comb, users_ids, users_map, min_covered_items,
                                         fairness_mes)
```

```
            if best_score < score:
                delta = score - best_score
                best_score = score
                best_comb = comb
                sol_iter = iterations

            for item in users_map[user_id].map_top_items_ratings:
                tmp_item_score[item] = tmp_item_score[item] * boost_factor

        iterations_list.append(iterations)
        duration = time.clock() - start
        groups_map[group_id].result_obj['greedy_comb'] = [best_comb, best_score, sol_iter, duration]


def greedy_algorith_score_function(ratings_factor, coverage_factor, users, ratings_list):
    score = ratings_factor * users
    score += coverage_factor * np.mean(ratings_list)
    return score

'''
CALCULATE THE SCORE OF EACH COMBINATION
'''
def calculate_combination_score(comb, users_ids, users_map, min_covered_items, fairness_mes):
    tmp_satisfaction_map = {}
    tmp_usr_coverage_map = {}
    for user_id in users_ids:
        tmp_satisfaction_map[user_id] = 0
        tmp_usr_coverage_map[user_id] = 0

    for item in comb:
        for user_id in users_ids:
            user_obj = users_map[user_id]

            if item in user_obj.map_top_items_ratings:
                rating = user_obj.map_top_items_ratings[item]
                tmp_satisfaction_map[user_id] += rating / user_obj.satisfaction_factor
                tmp_usr_coverage_map[user_id] += 1

    result_list = list(tmp_satisfaction_map.values())
    # score = 1 / np.var(result_list)
    if user_coverage_check(tmp_usr_coverage_map, min_covered_items):
        score = metric_calculation(result_list, fairness_mes)
        return score, min(tmp_satisfaction_map, key=tmp_usr_coverage_map.get)
    else:
        return -1, min(tmp_usr_coverage_map, key=tmp_usr_coverage_map.get)


'''
EXPORT THE RESULT SCORE OF EACH COMBINATION
'''
```

```
def greedy_algorith_deviation(groups_map, log, conf_object):
    ratings_factor = conf_object.greedy_ratings_factor
    coverage_factor = conf_object.greedy_coverage_factor
    fairness_mes = conf_object.fairness_measure
    boost_factor = conf_object.boost_factor

    header = 'Combination Result\n'
    header += 'coverage_factor,ratings_factor,boost_factor,'
    header += 'mse_from_best,mse_from_fair,average_iterations,max_iterations,'
    header += 'generation_success,duration_avg(s),'
    header += 'total_best_score,avg_best_score,total_fair_score,avg_fair_score,total_grd_score,avg_grd_score'
    list_best = []
    list_fair = []
    list_grdy = []
    list_iter = []

    list_time = []
    possible_recom = 0
    generated_recom = 0

    for group_id in groups_map:
        group_combinations = groups_map[group_id].result_obj
        if group_combinations['greedy_comb'][1] > 0 and group_combinations['fair_comb'][1] > 0:
            generated_recom += 1
            possible_recom += 1

            list_best.append(group_combinations['best_comb'][1])
            list_fair.append(group_combinations['fair_comb'][1])
            list_grdy.append(group_combinations['greedy_comb'][1])
            list_iter.append(group_combinations['greedy_comb'][2])
            list_time.append(group_combinations['greedy_comb'][3])

        elif group_combinations['greedy_comb'][1] < 0 and group_combinations['fair_comb'][1] > 0:
            possible_recom += 1

    if len(list_iter)>0:
        mse_from_best = sqrt(mean_squared_error(list_grdy, list_best))
        mse_from_fair = sqrt(mean_squared_error(list_grdy, list_fair))
        average_iterations = round(np.mean(list_iter), 2)
        max_iterations = max(list_iter)
        generation_success = round((generated_recom / possible_recom) * 100, 1)
        average_duration = np.mean(list_time)
        total_best_score = sum(list_best)
        avg_best_score = round(np.mean(list_best), 5)

        total_fair_score = sum(list_fair)
        avg_fair_score = round(np.mean(list_fair), 5)

        total_grd_score = sum(list_grdy)
        avg_grd_score = round(np.mean(list_grdy), 5)
```

```
    line = str(coverage_factor) + ',' + str(ratings_factor) + ',' + str(boost_factor)
    line += ',' + str(mse_from_best) + ',' + str(mse_from_fair) + ',' + str(average_iterations)
    line += ',' + str(max_iterations) + ',' + str(generation_success) + ',' + str(average_duration)

    line += ',' + str(total_best_score) + ',' + str(avg_best_score) + ',' + str(total_fair_score)
    line += ',' + str(avg_fair_score) + ',' + str(total_grd_score) + ',' + str(avg_grd_score)

    log.log_static_metric(header, line)


'''
FOR EACH COMBINATION TYPE COLLECT ITEM/USER COVERAGE STATS
(HOW MANY SATISFIED USERS PER PACKAGE AND THEIR RATING)
'''


def top_combination_analysis(groups_map, item_stats, log, users_map):
  for comb_type in groups_map[0].result_obj:

    for group_id in groups_map:
      users_ids = groups_map[group_id].users
      group_combinations = groups_map[group_id].result_obj
      comb = group_combinations[comb_type][0]

      if comb == None:
        break

      for item in comb:
        item_stats[item]['number_of_times_given'] += 1
        satisfied_users = 0
        for user_id in users_ids:
          user_obj = users_map[user_id]

          '''
          if top items +1 satisfied user else just rating score
          '''
          if item in user_obj.map_top_items_ratings:
            item_stats[item]['rating_list'].append(user_obj.map_top_items_ratings[item])
            satisfied_users += 1

        item_stats[item]['satisfied_users'].append(satisfied_users)

    item_stats_analysis(comb_type, item_stats, log)
    item_stats_reset(item_stats)


'''
ITEM STATS ANALYSIS
'''
def item_stats_analysis(calaulation_type, item_stats, log):
```

```
    # LOG result
    header = calaulation_type + '\n'
    header +=
'item_id,number_of_times_given,number_of_groups_offered,average_rating,rating_variation,average_covered_
users,variation_covered_users'
    for item in item_stats:
        num_of_times = item_stats[item]['number_of_times_given']
        if num_of_times > 0:
            avg_rating = np.mean(item_stats[item]['rating_list'])
            var_rating = np.var(item_stats[item]['rating_list'])
            avg_users_covered = np.mean(item_stats[item]['satisfied_users'])
            var_users_covered = np.var(item_stats[item]['satisfied_users'])
        else:
            avg_rating = 0
            var_rating = 0
            avg_users_covered = 0
            var_users_covered = 0

        key = str(item) + ',' + str(num_of_times)
        key += ',' + str(avg_rating)
        key += ',' + str(var_rating)
        key += ',' + str(avg_users_covered)

def item_stats_reset(item_stats):
    for item in item_stats:
        item_stats[item]['number_of_times_given'] = 0
        item_stats[item]['rating_list'] = []
        item_stats[item]['satisfied_users'] = []
```

7.9. Utils.py

Περιλαμβάνει βοηθητικές συναρτήσεις.

```
import itertools

import numpy as np

import os


import Upini_thesis_project.entities.GroupOfUsers

from Upini_thesis_project.entities import GroupOfUsers

'''

It takes the index_to_user_obj_map and for each user based on the constrain matrix returns
  for each user the

available objects

'''
```

```
def load_possible_items(users, constrain_matrix, dataframe):
    constrain_flag = 0

    for i in users:
        temp = constrain_matrix.loc[users[i].id, :]

        for item in temp[(temp != constrain_flag)].keys():
            rating = dataframe.loc[users[i].id, item]
            users[i].map_possible_items_ratings[item] = rating
            #users[i].possible_items_list.append(item) :todo remove list



'''
1.Iterate over all groups
2.Get the available items of each user of each group
3.Get a shorted dict for each user with the top items (to user object)
4.Calculate the factor for the satisfaction of each user (to user object)

'''



def select_top_items(number_of_top_items, all_users_map, groups_map): #:Done

    n = number_of_top_items
    for group_id in groups_map:

        for user_id in groups_map[group_id].users:
            user_obj = all_users_map[user_id]
            tmp_dict = user_obj.map_possible_items_ratings
            user_satisfaction_factor=0
```

```
            top = {k: tmp_dict[k] for k in sorted(tmp_dict, key=lambda k: -tmp_dict[k])[:n]}

            for item in top:
                user_satisfaction_factor += top[item]

            user_obj.map_top_items_ratings = dict(top)
            user_obj.satisfaction_factor = user_satisfaction_factor




    '''
    1.Iterate over all groups
    2.Iterate over all users of each group
    3.Get top K items of each user and create a list with the unique recommendation items for the group

    '''



    def  create_group_recommendation_list_of_available_items(groups_map,  all_users_map): #:Done
        for group_id in groups_map:

            tmp_pool_of_recommendable_items = []
            for user_id in groups_map[group_id].users:
                user_top_items_list = all_users_map[user_id].map_top_items_ratings.keys()

                for item in user_top_items_list:
                    tmp_pool_of_recommendable_items.append(item)

            groups_map[group_id].rlist_of_items = list(set(tmp_pool_of_recommendable_items))
```

```
    '''
    1.Generate all the permutations of items
    2.Add combination only if it does not exist (filter symmetric combinations)
    '''


    def combinations_generator(list, repeat):
        #           https://www.mathplanet.com/education/algebra-2/discrete-mathematics-and-probability/permutations-and-combinations
        item_combination_map = {}

        for item_combination in itertools.combinations(list, repeat): #:todo changed permutation to combination
            item_combination_map[item_combination] = ''

        return item_combination_map


    '''
    1.Generate all the permutations of items
    '''


    def combinations_generator_raw(list2, repeat):
        return itertools.permutations(list2, repeat)



    def entity_to_index_map(list):
```

```
    map_object = {}
    index = 0
    for i in list:
        map_object[i] = index
        index += 1
    return map_object


'''
gets a list of unique items and returns  a hashmap of index -> item
'''


def index_to_entity_map(list):
    map_object = {}
    index = 0
    for i in list:
        map_object[index] = i
        index += 1
    return map_object


'''
Recommender: Populate utility matrix initially from raw data using the config file
'''


def utility_matrix_populate(matrix, user_index, item_index, data, config_file):
    '''

    :param matrix: an existing matrix with zero values
```

```
        :param user_index: dictionary that maps each user_id to an array index
        :param item_index: dictionary that maps each item_id to an array index
        :param data: the dataframe from the raw file(filtered) with user,item,rating format
        :param config_file: contains the info of the entity position in the dataframe (data)
        :return: populated utility matrix
        '''
        for line in data.itertuples():
            user_id = line[config_file.csv_r_ind['user_id']]
            item_id = line[config_file.csv_r_ind['item_id']]
            rating = line[config_file.csv_r_ind['rating']]
            matrix[user_index[user_id], item_index[item_id]] = rating

        return matrix



    '''
    Recommender: For the export of the of the 2 files required for the constrain process
    '''



    def export_array_to_csv_dataframe(dir, pred_array, test_array, index_to_item_dict,
idex_to_user_dic):
        '''
        :param dir: directory of expoirted files
        :param pred_array: np array that contains the predicted values
        :param test_array: np array that contains the test values
        :param index_to_item_dict: dictionary that gives item_id from the array column index
        :param idex_to_user_dic: dictionary that gives user_id from the array row index
        :return: nothing
        '''
        if os.path.exists(dir):
```

```
    os.remove(dir)
    os.remove(dir + "_constrains")

fp = open(dir, "a")
fc = open(dir + "_constrains", "a")

line = 'user'
for item_index, value in np.ndenumerate(pred_array[0, :]):
    line += ',' + str(index_to_item_dict[item_index[0]])

fp.write(line)
fc.write(line)

for user_index, items_list in enumerate(pred_array):
    line = "\n" + str(idex_to_user_dic[user_index])

    for item_index, value in enumerate(items_list):
        line += ',' + str(value)

    fp.write(line)

for user_index, items_list in enumerate(test_array):
    line = "\n" + str(idex_to_user_dic[user_index])

    for item_index, value in enumerate(items_list):
        line += ',' + str(value)
    fc.write(line)
return
```

7.10. GreedyTrain.py

```
from Upini_thesis_project.Utilities.Calculations import  greedy_algorith_score_function,metric_calculation
import os


def optimize_greedy(groups_map, users_map, log, conf_object):
    ratings_factor = conf_object.greedy_ratings_factor
    coverage_factor = conf_object.greedy_coverage_factor
    fairness_mes = conf_object.fairness_measure
    boost_factor = conf_object.boost_factor

    dir = '/home/dimos/PycharmProjects/Py_projects/Upini_thesis_project/files/2_processed/opt_'
    dir += fairness_mes + '_cf_'+str(coverage_factor)+'_rf_'+str(ratings_factor)+'_bf_'+str(boost_factor)
    dir += '.csv'
    if os.path.exists(dir):
        os.remove(dir)

    fp = open(dir, "a")

    line = 'item,users,ratimgs,flag'
    fp.write(line)
    for group_id in groups_map:
        users_ids = groups_map[group_id].users

        comb = groups_map[group_id].result_obj['fair_comb'][0]


        if comb != None:
            tmp_item_stats, tmp_item_score = greedy_item_scoring(users_ids, users_map, ratings_factor,
coverage_factor)
            evl_item_stats, evl_item_score = reference_item_scoring(users_ids, comb, users_map, ratings_factor,
coverage_factor)

            for item in tmp_item_stats:
                top_choices_flag = 0
                if item in evl_item_stats:
                    top_choices_flag = 1

                line = '\n' + str(item) + ',' + str(tmp_item_stats[item]['users'])
                line += ',' + str(metric_calculation(tmp_item_stats[item]['ratings'], fairness_mes))
                line += ',' + str(top_choices_flag)

                fp.write(line)



def greedy_item_scoring(users_ids, users_map, ratings_factor, coverage_factor):
    tmp_grd_item_stats = {}
    tmp_item_score = {}

    for user_id in users_ids:
```

```
        user_obj = users_map[user_id]
        item_ratings = user_obj.map_top_items_ratings
        item_iteration(users_ids, item_ratings, tmp_grd_item_stats)

    for item in tmp_grd_item_stats:
        tmp_item_score[item] = greedy_algorith_score_function(ratings_factor, coverage_factor,
                                                              tmp_grd_item_stats[item]['users'],
                                                              tmp_grd_item_stats[item]['ratings'])

    return tmp_grd_item_stats, tmp_item_score


def item_iteration(users_ids, item_ratings, tmp_item_stats):  # done
    for item in item_ratings:
        if item not in tmp_item_stats:
            tmp_item_stats[item] = {'users': 1 / len(users_ids), 'ratings': [item_ratings[item]]}
        else:
            tmp_item_stats[item]['users'] += 1 / len(users_ids)
            tmp_item_stats[item]['ratings'].append(item_ratings[item])


def reference_item_scoring(users_ids, comb, users_map, ratings_factor, coverage_factor):
    tmp_item_score = {}
    tmp_item_stats = {}
    for item in comb:
        for user_id in users_ids:
            user_obj = users_map[user_id]
            item_ratings = user_obj.map_top_items_ratings

            if item in item_ratings:
                if item not in tmp_item_stats:
                    tmp_item_stats[item] = {'users': 1 / len(users_ids), 'ratings': [item_ratings[item]]}
                else:
                    tmp_item_stats[item]['users'] += 1 / len(users_ids)
                    tmp_item_stats[item]['ratings'].append(item_ratings[item])

    for item in tmp_item_stats:
        tmp_item_score[item] = greedy_algorith_score_function(ratings_factor, coverage_factor,
                                                              tmp_item_stats[item]['users'],
                                                              tmp_item_stats[item]['ratings'])

    return tmp_item_stats, tmp_item_score
```

7.11. Logger.py

Η κλάση που χρησιμοποιήθηκε για την καταγραφή των ενδιάμεσων αποτελεσμάτων που παράγονταν από τα πειράματα

```
import time
import os
```

```
class Logger:

    def __init__(self,config_obj):
        print('=====LOG START=====')

        self.task = ''
        self.str_time = 0
        self.end_time = 0
        self.task_map = {}
        self.static_metric_map = {}
        self.dynamic_metric_map = {}
        self.log_dir=config_obj.log_dir
        self.config_obj=config_obj


    def log_task(self,task_name):

        if self.task == '':
            self.task = task_name
            self.str_time = time.clock()

        else:
            self.end_time = time.clock()
            res = ' Duration: ' + str(self.end_time-self.str_time)
            self.task_map [self.task] = res
            print(self.task+res)
            self.task = task_name
            self.str_time = time.clock()


    def log_static_metric(self, metric, value):
        if metric not in self.static_metric_map:
            self.static_metric_map[metric] = [value]
        else:
            self.static_metric_map[metric].append(value)


    def log_dynamic_metric(self, metric, metric_key,value):
        if metric not in self.dynamic_metric_map:
            self.dynamic_metric_map[metric] = {}
            self.dynamic_metric_map[metric][metric_key] = value
        else:
            if metric_key not in self.dynamic_metric_map[metric]:
                self.dynamic_metric_map[metric][metric_key] = value
            else:
                self.dynamic_metric_map[metric][metric_key] = value


    def end(self):
        self.end_time = time.clock()
```

```
        self.task_map[self.task] = ' Duration: ' + str(self.end_time - self.str_time)

        res_separator = '\n\n==================================================\n'
        if os.path.exists(self.log_dir):
            os.remove(self.log_dir)
        f = open(self.log_dir, "a")

        for task in self.task_map:
            f.write(task + self.task_map[task]+'\n')


        for metric in self.static_metric_map:
            f.write(res_separator+metric+'\n')

            for value in self.static_metric_map[metric]:
                f.write(str(value)+'\n')


        for metric in self.dynamic_metric_map:
            f.write(res_separator+metric+'\n')

            for metrickey in self.dynamic_metric_map[metric]:
                value = self.dynamic_metric_map[metric][metrickey]
                f.write(str(metrickey)+','+str(value)+'\n')

        f.close()
```

7.12. exported_data_analysis_gran_split.py

Κώδικας που χρησιμοποιήθηκε για την συλλογή και συσταδοποίηση των αποτελεσμάτων που προέκυψαν από τα πειράματα με τις διαφορετικές παραμέτρους.

```
import os

dir = '/home/dimos/PycharmProjects/Py_projects/Upini_thesis_project/files/2_processed'
data_header =
'coverage_factor,ratings_factor,boost_factor,mse_from_best,mse_from_fair,average_iterations,max_iterations,g
eneration_success,duration_avg(s),total_best_score,avg_best_score,total_fair_score,avg_fair_score,total_grd_sc
ore,avg_grd_score'
files = os.listdir(dir)
sep = ','

exp_head =
'metric,group_type,group_size,top_items,package_size,constrain,rel_constrain_to_top_items,rel_constrain_to_r
ec_list,' + data_header


group_type = ['_similar','_dissimilar','_random']
fairn = ['least_misery', 'variance', 'min_max_ratio']

for gr in group_type:
    for fr in fairn:
```

```
    exp_dir = "aa_"+ gr + "_" + fr + ".csv"
    if os.path.exists(exp_dir):
       os.remove(exp_dir)
    file = open(exp_dir, "a")
    file.write(exp_head + '\n')

    for f in files:
       if os.path.isfile(f) and fr in f and gr in f:

       # if f
==='variance_grp_tp_random_grp_sz_5_top_it_3_item_rec_6_item_con_3_userfact_0.82_ratefact_0.09_boostfac
t_1.3.txt':
          ws = f.split("_")

          if 'least' in f:
             metric = ws[0] + "_" + ws[1]
             group_type = ws[4]
             group_size = (ws[7])
             top_items = (ws[10])
             package_size = (ws[13])
             constrain = (ws[16])

          elif 'min_max' in f:
             metric = ws[0] + "_" + ws[1]
             group_type = ws[5]
             group_size = (ws[8])
             top_items = (ws[11])
             package_size = (ws[14])
             constrain = (ws[17])


          elif 'variance' in f:
             metric = ws[0]
             group_type = ws[3]
             group_size = (ws[6])
             top_items = (ws[9])
             package_size = (ws[12])
             constrain = (ws[15])

          print(f)
          # print(ws)
          # print('metric',metric)
          # print('group_type', group_type)
          # print('group_size', group_size)
          # print('top_items', top_items)
          # print('package_size', package_size)
          # print('constrain', constrain)

          data = open(f)

          data_start_flag = False
```

```
        for row, line in enumerate(data):

            #
'metric,group_type,group_size,top_items,package_size,constrain,rel_constrain_to_top_items,rel_constrain_to_r
ec_list'

            if data_start_flag and len(line) > 3:
                newline = str(metric) + sep + str(group_type) + sep + str(group_size) + sep + str(top_items)
                newline += sep + str(package_size) + sep + str(constrain)

                rel_constrain_to_top_items = round(int(constrain) / int(top_items), 2)
                rel_constrain_to_rec_list = round(int(constrain) / int(package_size), 2)

                newline += sep + str(rel_constrain_to_top_items) + sep + str(rel_constrain_to_rec_list)
                newline += sep + line

                # print('row',row,'line',line)
                file.write(newline)

            elif data_start_flag:
                break

            if data_header in line:
                data_start_flag = True

    file.close()
```