ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Βελτιστοποίηση Αλγορίθμων Πρόβλεψης Προτιμήσεων Χρηστών σε Περιβάλλον Έξυπνης Τηλεόρασης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΥΓΕΝΙΑ ΔΙΚΑΙΟΥ

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου,
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Βελτιστοποίηση Αλγορίθμων Πρόβλεψης Προτιμήσεων Χρηστών σε Περιβάλλον Έξυπνης Τηλεόρασης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΥΓΕΝΙΑ ΔΙΚΑΙΟΥ

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου,
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή τη 17η Φεβρουαρίου 2017.

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Εμμανουήλ Βαρβαρίγος	Βασίλειος Λούμος
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2017

.....
Ευγενία Δικαίου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στη σύγχρονη κοινωνία, ο όγκος των διαθέσιμων πληροφοριών, προϊόντων και υπηρεσιών, οι γρήγοροι ρυθμοί ζωής, αλλά και οι διαρκώς αυξανόμενες απαιτήσεις των ανθρώπων καθιστούν απαραίτητη τη χρήση αλγορίθμων πρότασης περιεχομένου ανάλογου με τις προτιμήσεις του εκάστοτε χρήστη.

Σκοπός της παρούσας διπλωματικής εργασίας είναι η βελτιστοποίηση ενός αλγορίθμου πρόβλεψης προτιμήσεων χρηστών, έτσι ώστε να μειωθεί η απόκλιση των προβλεπόμενων από τις πραγματικές, αλλά και ο απαιτούμενος χρόνος για την ολοκλήρωση της διαδικασίας. Βάση για τη διαμόρφωση της εργασίας αυτής αποτελεί το σύστημα που αναπτύχθηκε στα πλαίσια του SAM project, το οποίο με τη χρήση second screen προτείνει περιεχόμενο σχετικό με ταινίες. Από αυτό πηγάζουν η βάση δεδομένων και ο αρχικός αλγόριθμος που χρησιμοποιούνται. Επομένως, στόχος της εργασίας είναι η ακριβέστερη και ταχύτερη πρόβλεψη σε πραγματικό χρόνο της βαθμολογίας ενός χρήστη για μία ταινία, προκειμένου να καθοριστεί αν θα αποτελέσει προτεινόμενο για αυτόν περιεχόμενο. Η πρόβλεψη βασίζεται στη βαθμολογία που προκύπτει μέσω sentiment analysis από τα σχόλια για την ταινία άλλων χρηστών με παρόμοιες προτιμήσεις.

Προκειμένου να επιτευχθεί ο παραπάνω στόχος, η αρχική βάση δεδομένων υπό μορφή γράφου (neo4j) εμπλουτίζεται με επιπλέον σχέσεις και στοιχεία, τα οποία ανανεώνονται και υπολογίζονται εκ νέου όταν συγκεντρωθεί όγκος δεδομένων ικανός να δημιουργήσει αποκλίσεις στα αποτελέσματα. Ακόμη, αναπτύσσεται σε java αλγόριθμος που αποτυπώνει πιο αντιπροσωπευτικά την ομοιότητα των προτιμήσεων δυο οποιωνδήποτε χρηστών, τόσο λόγω διαφορετικού τρόπου υπολογισμού αυτής όσο και λόγω εξειδικευμένης αντιμετώπισης ιδιαίτερων περιπτώσεων και βαθμολογικών μοτίβων. Επιπρόσθετα, βελτιστοποιείται ο εξαιρετικά διαδεδομένος για recommendation systems, k-NN (k-nearest neighbour) αλγόριθμος και περιορίζεται η επίδραση ή απομονώνονται από τη διαδικασία χρήστες των οποίων η συμπεριφορά επηρεάζει αρνητικά την ορθότητα των προβλέψεων. Επιπλέον, αντιμετωπίζονται διαφορετικά χρήστες που αναμένεται ότι θα παρουσιάσουν ιδιάζουσα συμπεριφορά όταν πραγματοποιούνται προβλέψεις για αυτούς (gray και black sheep). Τέλος, περιορίζονται οι on demand υπολογισμοί που απαιτούνται κάθε φορά, το πλήθος των προσβάσεων στη βάση δεδομένων και ο συνολικός καταναλισκόμενος σε αυτές χρόνος, αυξάνοντας σημαντικά την ταχύτητα του αλγορίθμου.

Λέξεις Κλειδιά

συστήματα πρότασης περιεχομένου, αλγόριθμοι πρόβλεψης προτιμήσεων, βελτιστοποίηση, k-NN αλγόριθμος, k-nearest neighbour αλγόριθμος, ομοιότητα, βάση δεδομένων neo4j, gray sheep, black sheep

Abstract

In modern society, the immense amount of the available data, products and services, the hectic pace of life and the continuously growing people's demands necessitate the usage of algorithms for recommending content according to each user's preferences.

The present diploma thesis' goal is the optimisation of an algorithm used in recommendation systems so that the deviation of predicted and real preference values as well as the amount of time needed to complete the process are diminished. The system developed for the SAM project which uses a second screen environment to recommend content relevant to movies is considered to be the basis for this thesis. The database and the initial algorithm which are used derive from this project. Thus, this thesis' goal is a more accurate and fast real-time prediction of a user's rating for a movie in order to determine if it will be recommended to him. Each prediction is calculated by taking into account ratings that stem from the sentiment analysis of a movie's comments written by similar, to the first one, users.

In order for this goal to be achieved, the initial graph database (neo4j) is enriched with new relationships and properties which are refreshed and recalculated whenever a large amount of data, capable of creating a noticeable divergence in results, is gathered. Furthermore, an algorithm implemented in java represents, more accurately, the similarity between users due to a different way of calculating it, as well as handling special cases and rating patterns. In addition, the k-NN (k-nearest neighbour) algorithm which is extremely popular among the recommendation systems is optimised and, users whose behaviour has a negative impact on the predictions' accuracy are completely or partially secluded. Users expected to have an irregular behaviour when predicting their preferences are handled differently (gray and black sheep). Finally, the on demand calculations, the number of queries executed to retrieve data from the database and the total amount of time needed for the execution of those queries is downsized, improving the performance of the algorithm.

Keywords

recommendation systems, recommender systems, prediction algorithms, optimisation, k-NN algorithm, k-nearest neighbour algorithm, similarity, neo4j database, gray sheep, black sheep

Ευχαριστίες

Η παρούσα διπλωματική εκπονήθηκε στο Εργαστήριο Distributed Knowledge and Media Systems Group του τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη της Καθηγήτριας Θεοδώρας Βαρβαρίγου.

Αρχικά, θα ήθελα να ευχαριστήσω θερμά την επιβλέπουσα καθηγήτρια μου, κ. Θεοδώρα Βαρβαρίγου, για την εμπιστοσύνη που μου έδειξε, καθώς και για τη δυνατότητα που μου έδωσε να διεκπεραιώσω την παρούσα διπλωματική εργασία.

Επιπρόσθετα, ιδιαίτερες ευχαριστίες οφείλω στον υποψήφιο Διδάκτορα του Ε.Μ.Π., Αλέξανδρο Ψύχα, για την άριστη καθοδήγηση και τη βοήθειά του καθ' όλη τη διάρκεια εκπόνησης της εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω πολύ την οικογένεια και τους φίλους μου για την αμέριστη υποστήριξή τους όλα αυτά τα χρόνια σε κάθε μου βήμα.

Ευγενία Δικαίου,
Αθήνα, Φεβρουάριος 2017

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κατάλογος Listings

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή του χώρου εφαρμογής της εργασίας

Η σύγχρονη κοινωνία χαρακτηρίζεται από την ταχεία εξέλιξη των υπολογιστών και του ψηφιακού κόσμου και την όλο και εντονότερη παρουσία τους σε όλες τις εκφάνσεις της καθημερινότητας. Το διαδίκτυο αποτελεί, πλέον, αναπόσπαστο τμήμα της ζωής κάθε ανθρώπου, καθώς μεγάλο μέρος των δραστηριοτήτων του σχετίζεται με αυτό. Ο όγκος των δεδομένων του διαδικτύου είναι τεράστιος και εξακολουθεί να αυξάνεται, εφόσον διαρκώς εμπλουτίζεται με νέες πληροφορίες, προϊόντα και υπηρεσίες.

Κάθε χρήστης για τον εντοπισμό της επιθυμητής πληροφορίας καλείται να φιλτράρει τις διαθέσιμες, αποκλείοντας όσες θεωρεί ότι δεν του χρειάζονται. Η διαδικασία αυτή δύναται να γίνει ιδιαίτερα χρονοβόρα, αν λάβουμε υπόψη τον όγκο των δεδομένων από τα οποία πρέπει να επιλέξει κάθε φορά, καθώς και τον συνολικό αριθμό αναζητήσεων που θα χρειαστεί να πραγματοποιήσει ο εκάστοτε χρήστης προκειμένου να καλύψει όλες τις ανάγκες του. Επιπρόσθετα, θεωρείται σημαντικό να αναφερθεί ότι οι πληροφορίες στις οποίες καταλήγει κάθε φορά δεν είναι κατ' ανάγκη οι βέλτιστες για αυτόν, αφού είναι δύσκολη η εποπτεία και η σύγκριση του συνόλου των διαθέσιμων για ένα θέμα πληροφοριών. Ακόμη, υπάρχει περίπτωση να επιθυμεί την ενημέρωσή του όταν προκύπτουν νέα δεδομένα σχετικά με αυτά για τα οποία έχει ήδη εκδηλώσει ενδιαφέρον. Τέλος, μπορεί να αποβεί πολύ χρήσιμη η γνωστοποίηση και σύσταση στον χρήστη πληροφοριών που παρουσιάζουν κάποια συσχέτιση, άμεση ή έμμεση, με τον ίδιο ή τα θέματα που τον ενδιαφέρουν. Προλαμβάνονται, έτσι, μελλοντικές αναζητήσεις του για αυτά, αλλά και του κοινοποιούνται πληροφορίες που ενδεχομένως δε θα αναζητούσε και ενδέχεται να του καλύπτουν καλύτερα κάποια υπάρχουσα ή καινούρια ανάγκη.

Όλα τα προαναφερθέντα, σε συνδυασμό με τις διαρκώς αυξανόμενες απαιτήσεις των χρηστών, καταδεικνύουν την αναγκαιότητα ύπαρξης συστημάτων που προτείνουν σε καθένα από αυτούς περιεχόμενο βάσει των εξατομικευμένων αναγκών του. Τέτοια συστήματα χρησιμοποιούνται ήδη για να προβάλλουν στο χρήστη περιεχόμενο, προϊόντα και υπηρεσίες που, ενδεχομένως, τον ενδιαφέρουν, βασιζόμενα σε προσωπικά του δεδομένα προσβάσιμα από το σύστημα, στη δραστηριότητά και τις προτιμήσεις του, αλλά και στα αντίστοιχα άλλων χρηστών που συνδέονται ή παρουσιάζουν μεγάλη ομοιότητα με αυτόν. Τα συστήματα αυτά, χρησιμοποιώντας τα παραπάνω στοιχεία, προβλέπουν, με κάποια απόκλιση, τη στάση του χρήστη απέναντι σε κάποιο περιεχόμενο και του προτείνουν αυτό για το οποίο προβλέπεται να είναι πιο θετικά διακείμενος. Ο σχεδιασμός και η ανάπτυξη τέτοιων συστημάτων είναι μια ιδιαίτερα απαιτητική εργασία, καθότι συνδυάζει γνώσεις και δεξιότητες πολλών διαφορετικών πεδίων της επιστήμης υπολογιστών.

Τα τελευταία χρόνια έχουν αναπτυχθεί αρκετές μέθοδοι-αλγόριθμοι προς επίτευξη αυτού του σκοπού, οι οποίες συχνά αναπροσαρμόζονται και συνδυάζονται προκειμένου να επιτευχθούν κάθε φορά όσο το δυνατόν καλύτερα - από πλευράς ακρίβειας πρόβλεψης, υπολογιστικού κόστους και ταχύτητας - αποτελέσματα.

1.2 Σκοπός και Δομή της εργασίας

Η παρούσα διπλωματική εργασία έχει ως σκοπό τη βελτιστοποίηση αλγορίθμου πρόβλεψης της συμπεριφοράς χρηστών, προκειμένου να μειωθεί η απόκλιση της προβλεπόμενης από την πραγματική, καθώς και ο χρόνος υπολογισμού αυτής. Ως εφαλτήριο για το σκοπό αυτό χρησιμοποιείται το σύστημα που αναπτύχθηκε στα πλαίσια του SAM (Socialising Around Media) [1], το οποίο, στη συνέχεια, εμπλουτίζεται και εξελίσσεται περαιτέρω για τις ανάγκες της εργασίας

Το προαναφερθέν project επιτρέπει στο χρήστη να καταναλώσει περιεχόμενο πολυμέσων από διάφορες πηγές, χρησιμοποιώντας διαφορετικές συγχρονισμένες συσκευές. Μέσω των συνδεδεμένων συσκευών επιτυγχάνεται η διαδραστικότητα, καθώς ο χρήστης δύναται να σχολιάσει, να προτείνει και να αναζητήσει περιεχόμενο σχετικό με ταινίες, τηλεοπτικά προγράμματα και on demand βίντεο, αναπτύσσοντας έτσι το φαινόμενο του Second Screen. Χάρη στο SAM, ο κάθε χρήστης, μέσω του Second Screen, μαθαίνει περισσότερα για αυτά που τον αφορούν, εφόσον παρουσιάζονται διάφορες πηγές σχετικών πληροφοριών· εντοπίζεται δυναμικά περιεχόμενο που συνδέεται με όσα προβάλλονται στην τηλεόραση, προτείνονται συσχετιζόμενα αντικείμενα και παρόμοια προγράμματα που υπάρχει περίπτωση να τον ενδιαφέρουν. Ακόμη, ο χρήστης ειδοποιείται για σχετικές με το πρόγραμμα συζητήσεις σε μέσα κοινωνικής δικτύωσης. Τέλος, το SAM παρέχει στατιστικά σχετικά με τη χρήση του και ελέγχει την καταλληλότητα του περιεχομένου ανάλογα με το άτομο στο οποίο απευθύνεται.

Σκοπός, λοιπόν, της εργασίας είναι η κατά το δυνατόν ακριβέστερη και ταχύτερη πρόβλεψη σε πραγματικό χρόνο της συμπεριφοράς του χρήστη απέναντι σε κάποιο περιεχόμενο, προκειμένου να διαμορφωθεί ένα σύνολο προτεινόμενων προς αυτόν στοιχείων. Έτσι, αναβαθμίζεται η εμπειρία του χρήστη, καθώς του προσφέρεται - χωρίς να χρειαστεί ο ίδιος να το αναζητήσει - περιεχόμενο που τον ενδιαφέρει.

Η εργασία δομείται ως εξής:

- Κεφάλαιο 2: Στο κεφάλαιο αυτό γίνεται αναφορά στις βασικές έννοιες που διέπουν τα συστήματα που προτείνουν περιεχόμενο στο χρήστη και στους αλγορίθμους που προβλέπουν τη συμπεριφορά του. Επιπλέον, προσδιορίζεται η προσέγγιση που θα ακολουθήσουμε σχετικά με τους αλγορίθμους αυτούς και η μορφή της βάσης δεδομένων που θα χρησιμοποιήσουμε.

- Κεφάλαιο 3: Το παρόν κεφάλαιο αφορά στις τεχνικές προδιαγραφές του συστήματος. Τελείται περιγραφή του προϋπάρχοντος συστήματος, των εργαλείων και λογισμικών που χρησιμοποιήθηκαν.

- Κεφάλαιο 4: Σε αυτό το κεφάλαιο αναλύονται οι τροποποιήσεις που πραγματοποιήθηκαν στη βάση δεδομένων, καθώς και ο αλγόριθμος που αναπτύχθηκε συμπεριλαμβανομένων όλων των αλλαγών και προσθηκών που έγιναν στο δοθέν σύστημα.

- Κεφάλαιο 5: Στο παρόν κεφάλαιο παρουσιάζονται τα αποτελέσματα της διαδικασίας και οι βελτιώσεις σε επίπεδο ακρίβειας-ορθότητας και ταχύτητας πρόβλεψης.

- Κεφάλαιο 6: Στο κεφάλαιο αυτό παρατίθενται τα συμπεράσματα στα οποία καταλήξαμε και προτείνονται κατευθύνσεις για πιθανή μελλοντική επέκτασή της.

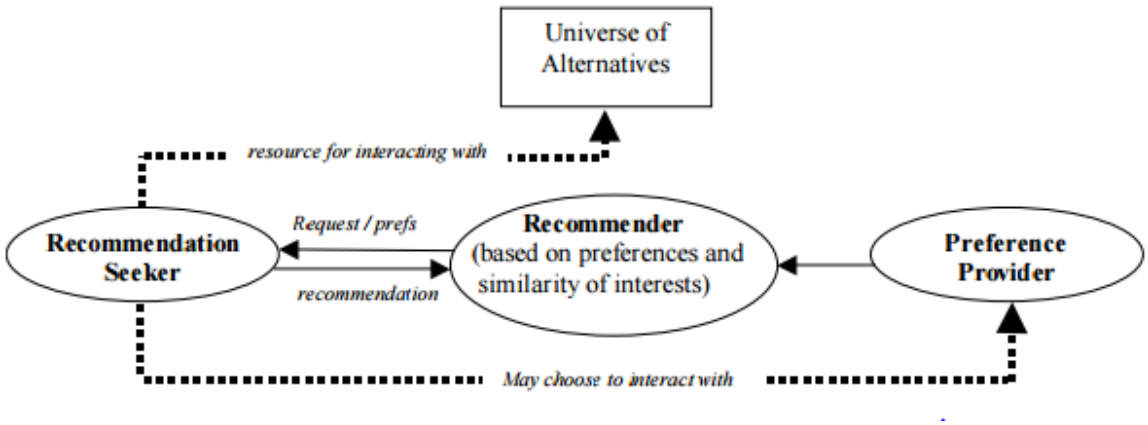
Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Recommendation Systems

Ως recommendation systems ή recommender systems [2][3] χαρακτηρίζουμε τα συστήματα φιλτραρίσματος πληροφοριών που προτείνουν στο χρήστη περιεχόμενο και, πιο συγκεκριμένα, προσπαθούν να προβλέψουν τις προτιμήσεις του ή τη βαθμολογία που θα έβαζε σε κάποιο αντικείμενο[1]. Τα συστήματα αυτά αντιμετωπίζουν το πρόβλημα του υπερβολικού όγκου δεδομένων απομονώνοντας χρήσιμες για το χρήστη πληροφορίες ανάλογα με τις προτιμήσεις, τα ενδιαφέροντα και την παρατηρούμενη συμπεριφορά όχι μόνο του ίδιου, αλλά και άλλων, παρόμοιων ή συσχετιζόμενων με αυτόν, χρηστών. Χάρη στα συστήματα αυτά, η εμπειρία του χρήστη αναβαθμίζεται, καθώς δε χρειάζεται, πλέον, να αναζητήσει μόνος του τις πληροφορίες ή τα αντικείμενα που τον ενδιαφέρουν. Τα recommendation systems έχουν αποδειχθεί ιδιαίτερα χρήσιμα, εφόσον μειώνουν τα κόστη εντοπισμού και επιλογής των κατάλληλων για το χρήστη αντικειμένων, βελτιώνουν τη διαδικασία λήψης απόφασης και την ποιότητα αυτής [4].

Στο ακόλουθο σχήμα αποτυπώνεται ένα γενικευμένο μοντέλο της διαδικασίας πρότασης περιεχομένου στο χρήστη.



Σχήμα 2.1: Μοντέλο της διαδικασίας πρότασης περιεχομένου [5]

Όπως προκύπτει από το άνωθεν σχήμα, μπορεί είτε ο Recommendation Seeker να κάνει αίτημα για να του προταθεί περιεχόμενο είτε ο Recommender να παράγει προτεινόμενο περιεχόμενο χωρίς κάποια προτροπή. Οι Seeker έχουν τη δυνατότητα είτε να

[1] Με τον όρο αντικείμενο, όταν αναφέρεται σε αυτό το πλαίσιο, εννοούμε οτιδήποτε προτείνεται στον χρήστη από το σύστημα.

γνωστοποιήσουν από μόνοι τους τις προτιμήσεις τους είτε αυτές να τους ζητηθούν από τον Recommender. Βασιζόμενος σε ένα σύνολο γνωστών προτιμήσεων - των δικών του, του Seeker και άλλων ανθρώπων - ο Recommender προτείνει στον Seeker περιεχόμενο που είναι πιθανό να τον ενδιαφέρει. Επιπρόσθετα, ο Recommender δύναται να αναγνωρίσει άτομα με παρόμοια ενδιαφέροντα. Ο Seeker μπορεί να αξιοποιήσει τις προτάσεις για να επιλέξει αντικείμενα μεταξύ των διαθέσιμων ή να επικοινωνήσει με παρόμοιους με αυτόν χρήστες.

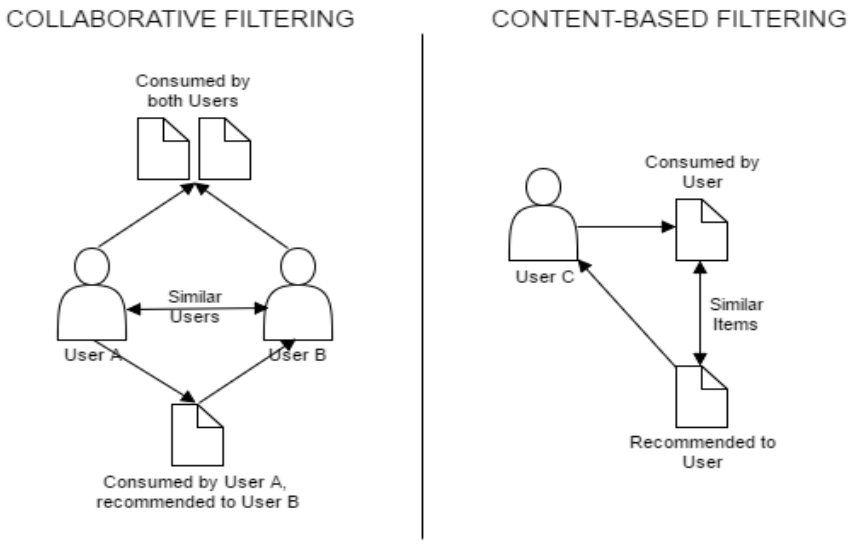
Τα recommendation systems χρησιμοποιούν αρκετές διαφορετικές τεχνολογίες. Μπορούμε να κατηγοριοποιήσουμε αυτά τα συστήματα βάσει δύο βασικών διαφορετικών προσεγγίσεων [3][6]:

- Τα συστήματα που βασίζονται σε collaborative filtering δημιουργούν ένα μοντέλο στηριζόμενα στην μέχρι τώρα συμπεριφορά του χρήστη, αλλά και στη συμπεριφορά άλλων χρηστών με παρόμοια γνωρίσματα και προτιμήσεις. Το μοντέλο αυτό χρησιμοποιείται, στη συνέχεια, για να προβλέψει αντικείμενα (ή βαθμολογία αντικειμένων) που μπορεί να ενδιαφέρουν το χρήστη.

- Τα συστήματα που βασίζονται σε content-based filtering χρησιμοποιούν μια σειρά διακριτών χαρακτηριστικών ενός αντικειμένου, προκειμένου να προτείνουν επιπλέον αντικείμενα με παρόμοια χαρακτηριστικά και ιδιότητες. Πιο συγκεκριμένα, εμφανίζουν στο χρήστη αντικείμενα παρόμοια με αυτά για τα οποία έχει επιδείξει κάποιο ενδιαφέρον στο παρελθόν [4].

Τέλος, αξίζει να αναφερθεί ότι υπάρχουν και υβριδικές τεχνικές που συνδυάζουν τις δύο προαναφερθείσες προσεγγίσεις.

Παρακάτω απεικονίζονται σχηματικά οι δύο αυτές προσεγγίσεις:



Σχήμα 2.2: Collaborative filtering versus Content-based filtering

2.1.1 Collaborative filtering

Τα συστήματα που χρησιμοποιούν collaborative filtering [4][7] βασίζονται στη συγκέντρωση και ανάλυση μεγάλης ποσότητας πληροφοριών σχετικών με τις συμπεριφορές, τις δραστηριότητες ή τις προτιμήσεις των χρηστών και στην πρόβλεψη των μελλοντικών τους προτιμήσεων βάσει της ομοιότητάς τους με άλλους χρήστες (ή της ομοιότητας σε επίπεδο βαθμολογίας μεταξύ αντικειμένων). Μια βασική αρχή στην οποία στηρίζονται αυτά τα συστήματα είναι ότι αν δυο χρήστες συμφωνούν για την ποιότητα ή τη συσχέτιση κάποιων αντικειμένων, τότε θεωρείται πιθανό να συμφωνούν και για άλλα αντικείμενα. Με αυτήν τη λογική, τα αντικείμενα που αρέσουν στον έναν και δεν τα γνωρίζει ο άλλος, αποτελούν περιεχόμενο το οποίο αξίζει να προταθεί. Αντίστοιχα, αν δυο αντικείμενα έχουν παρόμοιες βαθμολογίες ή τρόπο συσχέτισης από μέρος των χρηστών, τότε θα έχουν την ίδια βαθμολογία ή τρόπο συσχέτισης και από τους υπόλοιπους χρήστες.

Βασικό πλεονέκτημα της collaborative filtering προσέγγισης [2][8] είναι ότι δεν απαιτείται βαθιά κατανόηση και ανάλυση του ίδιου του, ενδεχομένως, πολύπλοκου αντικειμένου από το σύστημα, προκειμένου να γίνουν ορθές προτάσεις· και αυτό, γιατί στηρίζεται κυρίως σε βαθμολογίες και προτιμήσεις. Είναι αποτελεσματική και εύκολη στην υλοποίηση. Κύριο μειονέκτημα της προσέγγισης αυτής είναι το πρόβλημα του cold start, σύμφωνα με το οποίο εάν δεν υπάρχουν διαθέσιμες βαθμολογίες των αντικειμένων ή υπάρχουν ελάχιστες, τότε δεν μπορούν να παραχθούν προτάσεις περιεχομένου ή έχουν μικρή ακρίβεια αντίστοιχα. Άλλο μειονέκτημα των collaborative filtering συστημάτων είναι η επεκτασιμότητα (scalability). Το πρόβλημα αυτό παρουσιάζεται όταν αυξάνεται πολύ το πλήθος των χρηστών και των αντικειμένων, καθώς απαιτείται μεγαλύτερη υπολογιστική ισχύς για τους υπολογισμούς. Τέλος, εξίσου σημαντικό θεωρείται και το πρόβλημα της ισχνότητας (sparsity), που παρατηρείται, κυρίως, όταν τα αντικείμενα είναι πάρα πολλά και οι χρήστες έχουν βαθμολογήσει ένα μικρό υποσύνολο της βάσης δεδομένων, με αποτέλεσμα ακόμα και δημοφιλή αντικείμενα να έχουν ελάχιστες βαθμολογίες.

Τα collaborative filtering συστήματα διακρίνονται σε memory-based, model-based και υβρίδια αυτών.

- Οι memory-based αλγόριθμοι πραγματοποιούν προβλέψεις δρώντας πάνω σε δεδομένα - χρήστες, αντικείμενα και βαθμολογίες - αποθηκευμένα στη μνήμη. Βασικά πλεονεκτήματα των αλγορίθμων αυτών είναι η ευκολία κατανόησης και επεξήγησης των αποτελεσμάτων, αλλά και η ευκολία υλοποίησης τους και ένταξης νέων δεδομένων στη διαδικασία. Επίσης, κρίνεται σημαντική η ανεξαρτησία περιεχομένου των προτεινόμενων αντικειμένων. Κύριο μειονέκτημα των memory-based αλγορίθμων θεωρείται η μείωση της επίδοσής τους όταν τα δεδομένα είναι αραιά, γεγονός που επιβαρύνει την επεκτασιμότητα (scalability) των αλγορίθμων σε μεγάλα σύνολα δεδομένων. Οι memory-based αλγόριθμοι διακρίνονται σε user-based και item-based αλγόριθμους [9].

- Οι user-based αλγόριθμοι για να προτείνουν σε κάποιον χρήστη περιεχόμενο, αναζητούν χρήστες με παρόμοια μοτίβα (pattern) βαθμολόγησης των αντικειμένων με τον πρώτο και, βάσει των δικών τους βαθμολογιών, προβλέπουν τις αντίστοιχες του χρήστη που μας ενδιαφέρει σε αντικείμενα που δεν έχει βαθμολογήσει ακόμη.

- Οι item-based αλγόριθμοι για να προτείνουν σε κάποιον χρήστη περιεχόμενο, αναζητούν αντικείμενα με παρόμοια μοτίβα (pattern) στη βαθμολογία τους. Αν δυο αντικείμενα τείνουν να έχουν τους ίδιους χρήστες να τα βαθμολογούν παραπλήσια, τότε θεωρούνται όμοια και αναμένεται να βαθμολογούνται το ίδιο από ένα χρήστη.

- Στους model-based αλγόριθμους [9][10] εξάγεται ένα μοντέλο και χρησιμοποιείται για να προταθεί περιεχόμενο. Πιο συγκεκριμένα, απομονώνονται κάποιες πληροφορίες από το σύνολο των δεδομένων και χρησιμοποιούνται ως ένα μοντέλο πρότασης περιεχομένου, δίχως να απαιτείται η χρήση ολόκληρου του συνόλου των δεδομένων κάθε φορά. Τα προαναφερθέντα μοντέλα αναπτύσσονται μέσω data mining και machine learning αλγορίθμων, ώστε να εντοπιστούν κάποια μοτίβα (pattern) στηριζόμενα στα χρησιμοποιούμενα δεδομένα (training data). Από αυτά πραγματοποιούνται προβλέψεις για πραγματικά δεδομένα. Βασικές περιπτώσεις εφαρμογής αυτής της προσέγγισης είναι οι τεχνικές clustering, οι τεχνικές association, τα Bayesian networks κ.ά. Οι αλγόριθμοι αυτοί αντιμετωπίζουν καλύτερα το πρόβλημα της ισχνότητας (sparsity) από τους memory-based αλγορίθμους. Ακόμη, βοηθούν στο πρόβλημα της επεκτασιμότητας (scalability) σε μεγάλα σύνολα δεδομένων και γενικά είναι γρηγορότεροι. Ωστόσο, ιδιαίτερα ακριβή μπορεί να αποβεί η κατασκευή ενός τέτοιου μοντέλου. Τέλος, το μοντέλο που δημιουργείται είναι δυνατόν να έχει χαμηλότερη ακρίβεια πρόβλεψης, εφόσον για τη δημιουργία του χρησιμοποιείται μέρος των δεδομένων.

- Τα υβριδικά συστήματα συνδυάζουν αλγορίθμους memory-based και model-based και επιτυγχάνουν έτσι καλύτερα αποτελέσματα, καθώς υπερβαίνουν τους περιορισμούς που θέτουν οι κλασικές προσεγγίσεις, βελτιώνουν τα αποτελέσματα και αποτρέπουν σε κάποιο βαθμό προβλήματα, όπως η ισχνότητα (sparsity) και η απώλεια πληροφοριών. Αυξάνουν, όμως, την πολυπλοκότητα και είναι ακριβά στην υλοποίηση.

2.1.2 Content-based filtering

Το content-based filtering [2][4] αποτελεί μια άλλη προσέγγιση που υιοθετείται κατά το σχεδιασμό συστημάτων πρόβλεψης περιεχομένου και βασίζεται σε μια περιγραφή των αντικειμένων και σε ένα προφίλ με τις προτιμήσεις των χρηστών. Σε ένα τέτοιο σύστημα χρησιμοποιούνται λέξεις-κλειδιά (keywords) για να περιγραφούν τα αντικείμενα και το προφίλ χρήστη δημιουργείται για να υποδείξει τον τύπο αυτών που αρέσουν στον

συγκεκριμένο χρήστη. Συνεπώς, αυτοί οι αλγόριθμοι προτείνουν αντικείμενα που είναι όμοια με εκείνα που ενδιέφεραν το χρήστη στο παρελθόν (ή εξετάζονται στο παρόν). Πιο συγκεκριμένα, διάφορα υποψήφια προς πρόταση αντικείμενα συγκρίνονται με όσα ο εκάστοτε χρήστης έχει βαθμολογήσει ή έχει επιδείξει κάποιο ενδιαφέρον και εν τέλει προτείνονται εκείνα που κρίνεται ότι ταιριάζουν καλύτερα με τα υψηλότερα βαθμολογημένα ή με αυτά που τον ενδιέφεραν αντίστοιχα. Επομένως, η content-based filtering προσέγγιση επικεντρώνεται περισσότερο στην ανάλυση των χαρακτηριστικών των αντικειμένων για την παραγωγή προβλέψεων και πηγάζει από τη σχετική με ανάκτηση δεδομένων (information retrieval) και φιλτράρισμα πληροφοριών (information filtering) έρευνα.

Για να συνοψιστούν τα χαρακτηριστικά των αντικειμένων εφαρμόζονται κατάλληλοι αλγόριθμοι, όπως ο tf-idf. Για να δημιουργηθεί το προφίλ χρήστη, το σύστημα εστιάζει σε δύο τύπους πληροφοριών: σε ένα μοντέλο των προτιμήσεων του χρήστη και στο ιστορικό της αλληλεπίδρασης του με το σύστημα. Δημιουργείται ένα content-based προφίλ χρήστη βασισμένο σε ένα διάνυσμα με βάρη για τα χαρακτηριστικά των αντικείμενων. Τα βάρη δηλώνουν τη σημασία κάθε χαρακτηριστικού για το χρήστη, ο οποίος δύναται μέσω του feedback να τα τροποποιήσει.

Βασικό πλεονέκτημα του content-based filtering είναι η δυνατότητα πρότασης νέων αντικειμένων, ακόμα κι αν δεν διατίθενται βαθμολογίες από χρήστες, καθώς γίνεται ανάλυση μόνο του αντικειμένου και του προφίλ του χρήστη. Επίσης, αν οι προτιμήσεις του αλλάξουν, ο αλγόριθμος προσαρμόζει τις προτάσεις του στα νέα δεδομένα σε σύντομο χρονικό διάστημα. Ακόμη, προστατεύει τα προσωπικά δεδομένα, καθώς ο χρήστης δε χρειάζεται να μοιραστεί τα στοιχεία του. Σημαντικό μειονέκτημα του content-based filtering θεωρείται η εξάρτησή του από τα μεταδεδομένα των αντικειμένων· και αυτό, γιατί απαιτείται πλούσια περιγραφή τους και πλήττεται η ποιότητα πρόβλεψης αν δεν περιλαμβάνονται αρκετές ή ακριβείς πληροφορίες για τη σαφή διάκριση τους. Επιπρόσθετα, υπάρχει το πρόβλημα της υπερειδίκευσης (over-specialisation), αφού - λόγω της διαδικασίας που ακολουθείται για τις προβλέψεις - παρουσιάζεται κάποιο όριο στο επίπεδο πρότασης καινοτόμου περιεχομένου. Κατ' επέκταση, ο χρήστης θα λαμβάνει, ως επί το πλείστον, προτάσεις πανομοιότητες με τη δραστηριότητά του. Τέλος, για να καταστεί δυνατή η πρόταση περιεχομένου, θα πρέπει να υπάρχει ένα άρτια οργανωμένο προφίλ χρήστη.

2.1.3 Hybrid filtering

Πρόσφατες έρευνες έχουν δείξει ότι ο συνδυασμός collaborative filtering και content-based filtering τεχνικών ενδέχεται να έχει καλύτερα αποτελέσματα σε κάποιες περιπτώσεις [2][3][4]. Η κεντρική ιδέα πάνω στην οποία στηρίζονται οι υβριδικές αυτές τεχνικές είναι ότι με τη χρήση συνδυασμού αλγορίθμων μπορεί να γίνει υπέρβαση των μειονεκτημάτων και υπερκάλυψη των αδυναμιών που παρουσιάζει ο καθένας ξεχωριστά και τελικά να προκύψουν ορθότερες προβλέψεις. Υβριδικές τεχνικές δύναται να υλοποιηθούν με διάφορους τρόπους: πραγματοποιώντας προβλέψεις βάσει collaborative

filtering και content-based τεχνικών και συνδυάζοντας τα αποτελέσματά τους, προσθέτοντας content-based δυνατότητες σε collaborative filtering αλγορίθμους και το αντίστροφο ή ενοποιώντας τις δυο προσεγγίσεις σε ένα μοντέλο.

Υπάρχουν εφτά βασικές υβριδικές τεχνικές:

- Weighted: Συνδυάζει αριθμητικά, με τη χρήση βαρών, τα αποτελέσματα διαφορετικών τεχνικών πρόβλεψης. Τα βάρη κάθε τεχνικής προσαρμόζονται ανάλογα με την ποιότητα των αποτελεσμάτων που έχει δώσει.

- Switching: Μέσω μιας ευριστικής τεχνικής το σύστημα επιλέγει κάθε φορά έναν από τους διαθέσιμους αλγορίθμους πρόβλεψης ανάλογα με την ικανότητά τους για ακριβή αποτελέσματα.

- Mixed: Χρησιμοποιεί και παρουσιάζει ταυτόχρονα τα αποτελέσματα από διαφορετικές τεχνικές πρόβλεψης.

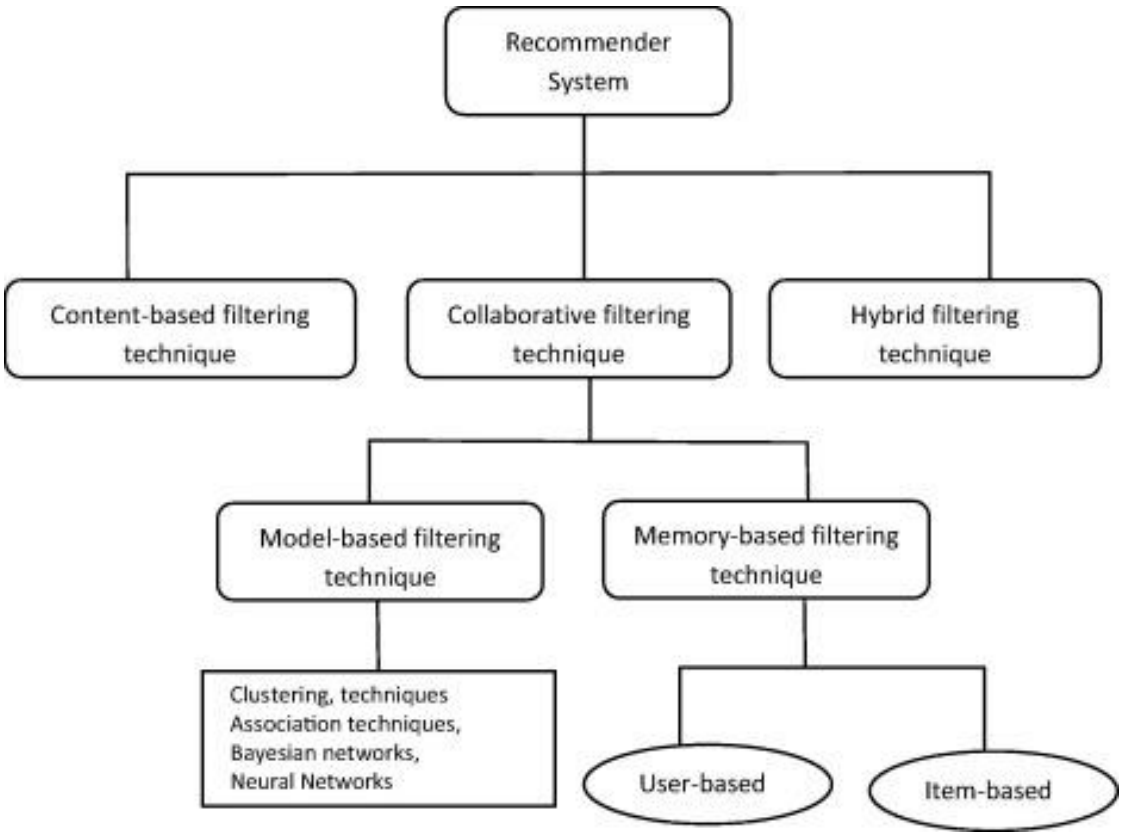
- Feature Combination: Χαρακτηριστικά που πηγάζουν από διαφορετικές πηγές συνδυάζονται για τη δημιουργία ενός αλγορίθμου πρότασης περιεχομένου.

- Feature Augmentation: Ένα χαρακτηριστικό ή σύνολο χαρακτηριστικών υπολογίζονται μέσω μιας τεχνικής πρότασης περιεχομένου και στη συνέχεια χρησιμεύουν ως μέρος της εισόδου της επόμενης τεχνικής.

- Cascade: Εφαρμόζεται μια επαναληπτική διαδικασία βελτίωσης της κατάταξης των προτιμήσεων για διαφορετικά αντικείμενα. Το προτεινόμενο περιεχόμενο που εξάγεται από μια τεχνική βελτιώνεται από την επόμενη και η βασική κατάταξη καθορίζεται από την πρώτη τεχνική.

- Meta-level: Το εσωτερικό μοντέλο που παράγεται από μια τεχνική πρότασης περιεχομένου χρησιμοποιείται ως είσοδος σε μια άλλη. Το μοντέλο αυτό είναι πάντα πλουσιότερο σε πληροφορία από μια απλή βαθμολογία.

Στο ακόλουθο σχήμα παρουσιάζονται οι βασικές κατηγορίες στις οποίες διακρίνονται τα συστήματα πρότασης περιεχομένου, όπως αναλύθηκαν παραπάνω.



Σχήμα 2.3: Κατηγορίες διάκρισης συστημάτων παραγωγής προτεινόμενου περιεχομένου [4]

2.2 Βασικά παραδείγματα Αλγορίθμων για Recommendation Systems

Όπως αναπτύχθηκε παραπάνω, τα recommendation systems διακρίνονται σε διάφορες κατηγορίες ανάλογα με την προσέγγιση που ακολουθούν. Υπάρχουν πολυάριθμοι αλγόριθμοι που εντάσσονται στις κατηγορίες αυτές και προβλέπουν τις προτιμήσεις των χρηστών. Ακολούθως, θα αναφερθούν ορισμένοι καίριοι τέτοιοι αλγόριθμοι.

2.2.1 Αλγόριθμος k-Nearest Neighbours

Στην αναγνώριση μοτίβων (pattern recognition), ο αλγόριθμος k-Nearest Neighbours (ή k-NN) [11] είναι μια μη-παραμετρική μέθοδος που χρησιμοποιείται για classification και regression. Και στις δυο περιπτώσεις η είσοδος αποτελείται από τα k κοντινότερα

δείγματα στον χώρο χαρακτηριστικών (feature space) [12][2]. Η έξοδος εξαρτάται από το αν ο k-NN χρησιμοποιείται για classification ή regression:

- Στην περίπτωση του k-NN classification, η έξοδος είναι η κλάση στην οποία αποφασίζεται ότι το αντικείμενο ανήκει. Η κατηγοριοποίηση ενός αντικειμένου πραγματοποιείται βάσει της πλειοψηφίας των γειτόνων του. Ανατίθεται στην κλάση που ανήκουν οι περισσότεροι από τους k κοντινότερους γείτονές του (όπου το k είναι θετικός, σχετικά μικρός ακέραιος).

- Στην περίπτωση του k-NN regression, η έξοδος είναι η τιμή ιδιότητας για το αντικείμενο. Αυτή η τιμή είναι ο μέσος όρος των αντίστοιχων τιμών των k κοντινότερων γειτόνων.

Ο k-NN είναι από τους απλούστερους machine learning αλγορίθμους. Μπορεί να θεωρηθεί χρήσιμη η ανάθεση κάποιου βάρους στη συμβολή των γειτόνων, έτσι ώστε οι κοντινότεροι να συμβάλλουν περισσότερο στο μέσο όρο από τους πιο μακρινούς. Θα μπορούσε, για παράδειγμα, να ανατεθεί σε κάθε γείτονα βάρος 1/d, όπου d η απόσταση του γείτονα από το αντικείμενο. Οι γείτονες λαμβάνονται από ένα σύνολο αντικειμένων για τα οποία η κλάση (στον k-NN classification) ή η τιμή ιδιότητας (στον k-NN regression) είναι γνωστή.

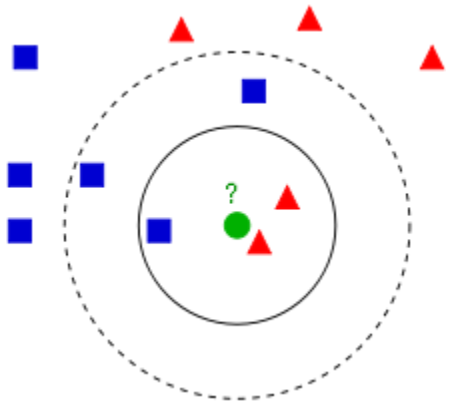
Τα χρησιμοποιούμενα δείγματα είναι διανύσματα ενός πολυδιάστατου χώρου χαρακτηριστικών (feature space), καθένα με μια ετικέτα κλάσης (class label). Σε πρώτο επίπεδο, ο αλγόριθμος αποθηκεύει τα διανύσματα χαρακτηριστικών (feature vectors) και τις ετικέτες των κλάσεων των δειγμάτων αυτών. Στη φάση της κατηγοριοποίησης (classification), με το k να είναι μια καθοριζόμενη από το χρήστη σταθερά, αποφασίζεται η ετικέτα ενός - μη-κατηγοριοποιημένου ως τότε - διανύσματος, βάσει της συχνότερα εμφανιζόμενης, μεταξύ των k πλησιέστερων γειτόνων, ετικέτας.

Μια συχνά χρησιμοποιούμενη, από τον αλγόριθμο, μετρική απόστασης για συνεχείς μεταβλητές είναι η Ευκλείδεια απόσταση. Για διακριτές μεταβλητές είναι δυνατή η εφαρμογή μιας άλλης μετρικής, της overlap metric (ή Hamming distance). Ακόμα, ο k-NN αξιοποιεί και συντελεστές συσχέτισης (correlation coefficients), όπως ο Pearson και ο Spearman.

Η "πλειοψηφική" κατηγοριοποίηση ("majority voting" classification) παρουσιάζει προβλήματα όταν η κατανομή των κλάσεων είναι στρεβλωμένη. Για παράδειγμα, τα στοιχεία της πιο συχνά εμφανιζόμενης κλάσης τείνουν να κυριαρχούν στην πρόβλεψη ενός προς κατηγοριοποίηση στοιχείου, γιατί εμφανίζονται συχνότερα μεταξύ των k πλησιέστερων γειτόνων λόγω του μεγάλου αριθμού τους. Ένα τρόπο αντιμετώπισης αυτού του προβλήματος αποτελεί η χρήση βαρών ίσων με το αντίστροφο της απόστασης

[2] Ως χώρο χαρακτηριστικών (feature space) [12] ορίζουμε το διανυσματικό χώρο που περιλαμβάνει τα διανύσματα χαρακτηριστικών (feature vectors). Διάνυσμα χαρακτηριστικών (feature vector) ονομάζουμε το διάνυσμα ν-διαστάσεων το οποίο αποτελείται από αριθμητικά χαρακτηριστικά που αντιπροσωπεύουν κάποιο αντικείμενο.

του προς κατηγοριοποίηση στοιχείου από τον εκάστοτε γείτονα, τα οποία πολλαπλασιάζονται με την κλάση (ή την τιμή σε περίπτωση προβλήματος regression).



Σχήμα 2.4: Παράδειγμα για k-NN classification [11]

Στο άνωθεν σχήμα παρουσιάζεται ένα παράδειγμα για την περίπτωση του k-NN classification. Οι υπάρχουσες κλάσεις είναι δυο, παριστάνονται με τα τετράγωνα και τα τρίγωνα και κάθε νέο προς κατηγοριοποίηση στοιχείο θα πρέπει να εντάσσεται σε μια από τις δυο αυτές κλάσεις. Στο παράδειγμα αυτό, το προς κατηγοριοποίηση στοιχείο παριστάνεται με τον κύκλο. Αν k=3 (συμπαγής-solid κύκλος), τότε το στοιχείο ανατίθεται στην κλάση με τα τρίγωνα, εφόσον στον κύκλο περιέχονται δυο τρίγωνα και ένα τετράγωνο. Αν k=5 (διακεκομμένος-dashed κύκλος), τότε το στοιχείο ανατίθεται στην κλάση με τα τετράγωνα, αφού στον κύκλο περιέχονται τρία τετράγωνα και δυο τρίγωνα.

Η βέλτιστη επιλογή ενός k εξαρτάται από τα εκάστοτε δεδομένα. Γενικά, μεγαλύτερες τιμές του k μειώνουν την επίδραση του θορύβου στην κατηγοριοποίηση (classification), αλλά υπολογιστικά είναι πιο ακριβές, καθιστούν τα όρια μεταξύ των κλάσεων λιγότερο διακριτά και μας απομακρύνουν από τη λογική του k-NN που επιθυμεί να λάβει υπόψη μόνο τους πλησιέστερους γείτονες. Μέσω ευριστικών αλγορίθμων μπορεί να εντοπιστεί ένα καλό k. Η ακρίβεια του k-NN ελαττώνεται δραματικά λόγω της παρουσίας θορύβου, μη-σχετικών χαρακτηριστικών ή αν η κλιμάκωση των χαρακτηριστικών δεν αντιστοιχεί στη σπουδαιότητά τους. Για την αντιμετώπιση αυτών έχουν αναπτυχθεί δυο βασικές προσεγγίσεις: η χρήση εξελικτικών αλγορίθμων (evolutionary algorithms) και η κλιμάκωση των χαρακτηριστικών των κοινών πληροφοριών μεταξύ των διαθέσιμων δεδομένων (training data) και κλάσεων (training classes).

Στην περίπτωση του k-NN regression, ο αλγόριθμος k-NN χρησιμοποιείται για να υπολογίσει συνεχείς μεταβλητές. Ένας τέτοιος αλγόριθμος χρησιμοποιεί το σταθμικό μέσο των k πλησιέστερων γειτόνων, με βάρος το αντίστροφο της απόστασης τους από το στοιχείο. Η τυπική διαδικασία που ακολουθείται είναι η εξής:

1. Υπολογισμός της απόστασης (π.χ. Ευκλείδεια απόσταση) μεταξύ του στοιχείου που μας ενδιαφέρει και των υπόλοιπων.

2. Διάταξη των στοιχείων κατά αύξουσα απόσταση.

3. Εύρεση του βέλτιστου k βάσει του RMSE.

4. Υπολογισμός σταθμικού μέσου αντίστροφης απόστασης με τη συμβολή των k πλησιέστερων γειτόνων.

2.2.2 Αλγόριθμος Singular Value Decomposition

Ο Singular Value Decomposition (SVD) [13] [14] είναι μια μέθοδος για τη μετατροπή ενός πραγματικού ή μιγαδικού πίνακα σε γινόμενο πινάκων ή καλύτερα για την ανάλυση του σε ιδιάζουσες τιμές. Ένας *mxn* $\boldsymbol{R}$ παραγοντοποιείται στη μορφή:

$$\boldsymbol{R}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \text{ , με } \boldsymbol{U}^T\boldsymbol{U}=\boldsymbol{U}\boldsymbol{U}^T=\boldsymbol{I} \text{ και } \boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{V}\boldsymbol{V}^T=\boldsymbol{I}$$

όπου ο $\boldsymbol{U}$ και $\boldsymbol{V}$ είναι ορθοκανονικοί πραγματικοί ή μιγαδικοί πίνακες *mxm* και *nxn* αντίστοιχα και ο $\boldsymbol{\Sigma}$ είναι "ορθογώνια διαγώνιος" *mxn* πίνακας με μη-αρνητικά πραγματικά στοιχεία στη διαγώνιο. Ο $\boldsymbol{\Sigma}$ περιλαμβάνει στη διαγώνιό του τις ιδιάζουσες τιμές του $\boldsymbol{R}$: $\{\sigma_1,\sigma_2,...,\sigma_r\}$ των οποίων το πλήθος είναι ίσο με την τάξη (rank) του πίνακα $\boldsymbol{R}$. Κοινή σύμβαση αποτελεί οι ιδιάζουσες αυτές τιμές να τοποθετούνται σε φθίνουσα σειρά.

Σε εφαρμογές πολύ σπάνια χρησιμοποιείται η πλήρης αυτή μορφή του SVD, καθώς επιθυμούμε οικονομία χρόνου και χώρου. Μια από τις βασικές απλοποιημένες μορφές του SVD είναι ο compact SVD, στον οποίο υπολογίζονται οι *r* στήλες του $\boldsymbol{U}$ και οι *r* γραμμές του $\boldsymbol{V}^T$ που αντιστοιχούν στις μη-μηδενικές ιδιάζουσες τιμές του $\boldsymbol{\Sigma_r}$. Τα υπόλοιπα στοιχεία των πινάκων δεν υπολογίζονται. Ο $\boldsymbol{U_r}$ γίνεται *mxr*, ο $\boldsymbol{\Sigma_r}$ *rxr* διαγώνιος και ο $\boldsymbol{V_r}^T$ *rxn*.

Οι πίνακες που λαμβάνονται με αυτήν τη μέθοδο είναι ιδιαίτερα χρήσιμοι, εφόσον ο SVD παρέχει τις καλύτερες χαμηλότερης τάξης (rank) προσεγγίσεις του αρχικού πίνακα $\boldsymbol{R}$, βάσει του Frobenius norm. Ο *rxr* $\boldsymbol{\Sigma_r}$ είναι δυνατόν να τροποποιηθεί διατηρώντας μόνο τις k μεγαλύτερες τιμές της διαγωνίου, ώστε να παραχθεί ο πίνακας $\boldsymbol{\Sigma_k}$, με k<r. Αν και οι υπόλοιποι πίνακες τροποποιηθούν αντιστοίχως, τότε θα έχουμε τον $\boldsymbol{R_k}=\boldsymbol{U_k}\boldsymbol{\Sigma_k}\boldsymbol{V_k}^T$, που είναι ο πλησιέστερος στον $\boldsymbol{R}$ k τάξης πίνακας. Ισοδύναμα, ο $\boldsymbol{R_k}$ ελαχιστοποιεί το Frobenius norm $||\boldsymbol{R}-\boldsymbol{R_k}||$ σε σύγκριση με οποιονδήποτε άλλο πίνακα k τάξης.

Στα recommendation systems η μέθοδος SVD μπορεί να χρησιμοποιηθεί για την εκτέλεση δυο διαφορετικών εργασιών. Αρχικά, χρησιμοποιείται για τη διάκριση λανθανουσών σχέσεων μεταξύ χρηστών και αντικειμένων που επιτρέπουν τον υπολογισμό του προβλεπόμενου ενδιαφέροντος του χρήστη για ένα συγκεκριμένο αντικείμενο. Επίσης, ο SVD χρησιμοποιείται για την παραγωγή μιας λίγων διαστάσεων αναπαράστασης του αρχικού χώρου χρήστη-αντικειμένου και για τον υπολογισμό των γειτόνων στον ελαττωμένο αυτό χώρο. Κατόπιν, γίνεται χρήση των παραπάνω για τη δημιουργία μιας λίστας των top-N προτεινόμενων αντικειμένων για χρήστες.

Αρχικά, υπάρχει ένας $\boldsymbol{R}$ χρηστών-αντικειμένων με στοιχεία βαθμολογίες που είναι πολύ αραιός. Για να εντοπιστούν χρήσιμες λανθάνουσες σχέσεις αντιμετωπίζεται η ισχνότητα (sparsity) με το γέμισμα του πίνακα χρηστών-αντικειμένων. Για το σκοπό αυτό, μπορεί να γίνει χρήση της μέσης βαθμολογίας των προϊόντων. Κατόπιν, δύναται να χρησιμοποιηθεί μια τεχνική ομαλοποίησης (normalisation technique), όπως η αφαίρεση της μέσης βαθμολογίας του χρήστη από κάθε βαθμολογία. Τελικά, προκύπτει ο $\boldsymbol{R}_{norm}=\boldsymbol{R}+\boldsymbol{NPR}$, όπου ο $\boldsymbol{NPR}$ παρέχει "αφελείς" μη-εξατομικευμένες προτάσεις περιεχομένου. Πραγματοποιείται παραγοντοποίηση του $\boldsymbol{R}_{norm}$ και προκύπτει μια χαμηλής τάξης (rank) προσέγγιση, εφόσον εφαρμοστούν τα ακόλουθα βήματα:

1. Τελείται παραγοντοποίηση του $\boldsymbol{R}_{norm}$ με χρήση της μεθόδου SVD για την παραγωγή των $\boldsymbol{U},\boldsymbol{\Sigma},\boldsymbol{V}$.

2. Ο $\boldsymbol{\Sigma}$ ελαττώνεται σε k διαστάσεις.

3. Υπολογίζεται η τετραγωνική ρίζα του ελαττωμένου $\boldsymbol{\Sigma}_k$, ώστε να ληφθεί ο $\boldsymbol{\Sigma}_k^{1/2}$.

4. Υπολογίζονται οι πίνακες: $\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{1/2}$ και $\boldsymbol{\Sigma}_k^{1/2}\boldsymbol{U}_k^T$.

Οι πίνακες αυτοί μπορούν να χρησιμοποιηθούν για τον υπολογισμό της πρόβλεψης της βαθμολογίας κάθε αντικειμένου (p) από κάθε χρήστη (c). Οι διαστάσεις του $\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{1/2}$ είναι *mxk* και του $\boldsymbol{\Sigma}_k^{1/2}\boldsymbol{U}_k^T$ *kxn*. Η πρόβλεψη προκύπτει από το γινόμενο της $c^{ης}$ γραμμής του $\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{1/2}$ με την $p^{η}$ στήλη του $\boldsymbol{\Sigma}_k^{1/2}\boldsymbol{U}_k^T$ επαυξημένο κατά το μέσο όρο των βαθμολογιών του χρήστη, όπως φαίνεται ακολούθως:

$$C_{p^{pred}} = \bar{C} + \boldsymbol{U}_k \sqrt{\boldsymbol{\Sigma}_k}^{T} (c)\ \sqrt{\boldsymbol{\Sigma}_k}\ \boldsymbol{U}_k^{T} (p)$$

Στο σημείο αυτό κρίνεται σκόπιμο να αναφερθεί ότι παρά την πυκνότητα του $\boldsymbol{R}_{norm}$, η δομή του πίνακα $\boldsymbol{NPR}$ επιτρέπει τη χρήση αραιών (sparse) SVD αλγορίθμων, των οποίων η πολυπλοκότητα είναι σχεδόν γραμμική σε σχέση με τον αριθμό των μη-μηδενικών στοιχείων στον αρχικό πίνακα $\boldsymbol{R}$.

Πέραν της προαναφερθείσας διαδικασίας, ιδιαίτερα χρήσιμος μπορεί να καταδειχθεί ένας χώρος λιγότερων διαστάσεων ως βάση για τον εντοπισμό των γειτόνων (neighbours), των οποίων οι προτιμήσεις βοηθούν να προταθεί στο χρήστη μια λίστα N αντικειμένων. Για το σκοπό αυτό, τα δεδομένα των χρηστών θεωρούνται δυαδικά και, πιο συγκεκριμένα, θεωρείται ως "1" οποιοδήποτε μη μηδενικό στοιχείο. Δηλαδή, είναι σημαντικό το αν καταναλώθηκε ένα αντικείμενο από το χρήστη και όχι το πόσο του άρεσε. Ακολουθούνται τα βήματα της προηγούμενης διαδικασίας μέχρι και τον υπολογισμό του $\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{1/2}$. Αυτός ο *mxk* πίνακας είναι η k-διαστάσεων αναπαράσταση των *m* χρηστών. Ακολούθως, εφαρμόζεται κάποια μέθοδος υπολογισμού της ομοιότητας (similarity), ώστε να δημιουργηθεί το σύνολο των γειτόνων στο χώρο αυτό. Στη συνέχεια, εξετάζονται οι γείτονες ενός συγκεκριμένου χρήστη και αναλύονται τα αντικείμενα που καταναλώθηκαν, έτσι ώστε να του προταθούν τα N καταλληλότερα. Αυτό επιτυγχάνεται αν εξεταστεί το σύνολο όλων των αντικειμένων που καταναλώθηκαν από τους k γείτονες και πραγματοποιηθεί μέτρηση των συχνοτήτων τους. Τέλος, η προκύπτουσα λίστα των προϊόντων ταξινομείται και τα N συχνότερα καταναλισκόμενα

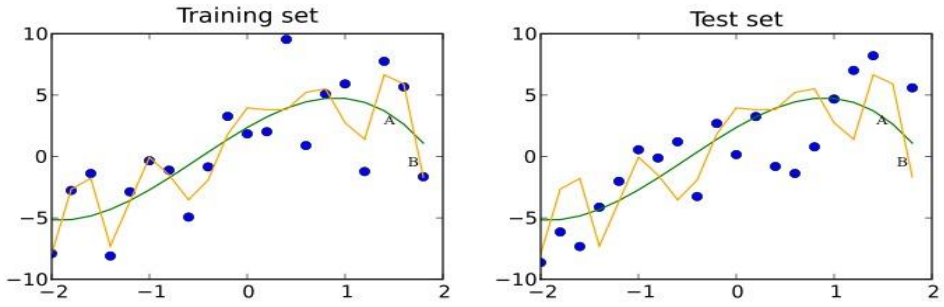
αντικείμενα επιστρέφονται ως προτεινόμενα στον χρήστη που έχει επιλεγεί. Η προαναφερθείσα διαδικασία ονομάζεται πρόταση συχνότερου αντικειμένου (most frequent item recommendation).

2.3 Μέθοδοι Ελέγχου Εγκυρότητας Αλγορίθμων

Η εύρεση σχέσεων από δεδομένα, οι οποίες μπορούν να χρησιμεύσουν στην παραγωγή προβλέψεων, αποτελεί μια άκρως σημαντική εργασία. Η ανακάλυψη των σχέσεων αυτών πραγματοποιείται, σε πρώτη φάση, με τη βοήθεια του training set, ενώ ένα test set χρησιμοποιείται για την αξιολόγηση τους [15]. Δηλαδή, το training set είναι ένα σύνολο δεδομένων που χρησιμοποιείται για την εύρεση σχέσεων που πιθανόν μπορούν να συντελέσουν στην παραγωγή προβλέψεων. Το test set είναι ένα σύνολο δεδομένων που χρησιμοποιείται για την εκτίμηση και την επαλήθευση της σωστής λειτουργίας των σχέσεων που προέκυψαν από το training set. Το test set είναι ανεξάρτητο του training set, αλλά ακολουθεί την ίδια κατανομή πιθανότητας με αυτό.

Πολλές φορές, πέραν των δυο αυτών set διαμορφώνεται και ένα ακόμη· το λεγόμενο validation set, που χρησιμοποιείται προκειμένου να αποφευχθεί η υπερβολική προσαρμογή των προκυπτουσών σχέσεων στις ιδιαιτερότητες των παρεχόμενων δεδομένων (overfitting), όταν κάποια παράμετρος κατηγοριοποίησης (classification) χρειάζεται προσαρμογή.

Το σύνολο των δεδομένων μπορεί να κατανεμηθεί στα προαναφερθέντα set με διάφορους τρόπους. Συνήθως, στο training set αποδίδεται το 60% με 80% των δεδομένων και στο test set το υπόλοιπο 40% με 20%. Ιδιαίτερα δημοφιλείς είναι οι αναλογίες 70%/30% ή 75%/25% ή 80%/20% για training και test set αντίστοιχα. Στην περίπτωση ύπαρξης validation set χρησιμοποιούνται συχνά οι αναλογίες 60%/20%/20% ή 70%/15%/15% για training, validation και test set αντιστοίχως. Αν ένα μοντέλο έχει εξίσου καλά αποτελέσματα τόσο για το training όσο και για το test set, τότε υπάρχει ελάχιστο overfitting. Αντίθετα, αν ένα μοντέλο παρουσιάζει πολύ καλύτερα αποτελέσματα στο training set σε σχέση με το test set, τότε είναι πιθανή η ύπαρξη overfitting.



Σχήμα 2.5: Training και Test Set [15]

Στο προηγούμενο σχήμα (Σχήμα 2.5) παρουσιάζονται με κουκίδες τα δεδομένα του training set στην αριστερή γραφική παράσταση και του test set στη δεξιά. Δυο μοντέλα προβλέψεων έχουν προσαρμοστεί στα δεδομένα του training set και απεικονίζονται τόσο με τα δεδομένα αυτού (αριστερή γραφική παράσταση) όσο και με τα δεδομένα του test set (δεξιά γραφική παράσταση). Στο training set το μέσο τετραγωνικό σφάλμα (Mean Squared Error ή MSE) του μοντέλου που απεικονίζεται με τη γραμμή Α είναι 9, ενώ εκείνου που απεικονίζεται με τη γραμμή Β είναι 4. Στο test set οι αντίστοιχες τιμές είναι 13 και 15. Επομένως, στο μοντέλο που παριστάνεται με τη γραμμή Β παρατηρείται έντονα το φαινόμενο του overfitting, καθώς παρουσιάζει σχεδόν 4 φορές μεγαλύτερο MSE στο test set. Στο μοντέλο που παριστάνεται με τη γραμμή Α το overfitting είναι συγκριτικά λιγότερο έντονο, εφόσον το test set παρουσιάζει MSE λιγότερο από 2 φορές μεγαλύτερο από αυτό του training set.

2.4 Βασικές Προκλήσεις, Περιορισμοί και Προβλήματα των Recommendation Systems

Στις παραπάνω ενότητες αναφέρθηκαν ορισμένα από τα προβλήματα των recommendation systems ή συγκεκριμένων κατηγοριών αυτών. Ακολούθως, θα παρουσιαστούν εκτενέστερα τα σημαντικότερα προβλήματα, περιορισμοί και προκλήσεις [9][16] των συστημάτων αυτών.

- Data Sparsity: Συχνά, τα σύνολα δεδομένων πάνω στα οποία εργάζονται οι αλγόριθμοι πρότασης περιεχομένου είναι αρκετά μεγάλα. Σε αυτήν την περίπτωση ο πίνακας χρηστών-αντικειμένων μπορεί να είναι πολύ μεγάλος και αραιός, με αποτέλεσμα να μη διατίθενται όσα δεδομένα απαιτούνται κάθε φορά για ακριβείς υπολογισμούς και προτάσεις περιεχομένου.

- Cold-start [17]: Στα content-based συστήματα, το πρόβλημα αυτό παρατηρείται όταν το σύστημα δεν έχει διαμορφώσει ακόμη ένα ολοκληρωμένο προφίλ χρήστη. Μέχρι τότε, το σύστημα απλά δέχεται δεδομένα από το χρήστη, δίχως να μπορεί να του προτείνει περιεχόμενο. Στα collaborative filtering συστήματα, το πρόβλημα αυτό παρατηρείται τόσο σε νέους χρήστες, εφόσον δεν υπάρχει δείγμα των προτιμήσεών τους όσο και σε αντικείμενα για τα οποία δεν υπάρχει κάποια βαθμολογία.

- Scalability: Το πρόβλημα αυτό εντοπίζεται όταν το πλήθος των χρηστών και των αντικειμένων αυξάνεται σε μεγάλο βαθμό, με αποτέλεσμα το σύστημα να δυσκολεύεται στο χειρισμό τους και να απαιτείται μεγαλύτερη υπολογιστική ισχύς.

- Synonyms: Η συνωνυμία αναφέρεται στην τάση ενός αριθμού ίδιων ή εξαιρετικά παρόμοιων αντικειμένων να έχουν διαφορετικά ονόματα ή καταχωρήσεις. Πολλά συστήματα πρότασης περιεχομένου δεν είναι ικανά να αναγνωρίσουν τις

υποβόσκουσες αυτές συσχετίσεις και κατ' επέκταση τα αντιμετωπίζουν ως διαφορετικά.

- Gray Sheep: Ο όρος "γκρίζο πρόβατο" (gray sheep) αποδίδεται σε χρήστες των οποίων οι προτιμήσεις παρουσιάζουν ασυνέπεια και, επομένως, δεν μπορούν να ενταχθούν σε κάποια υπάρχουσα κατηγορία χρηστών. Οι χρήστες που ανήκουν σε αυτήν την κατηγορία εν μέρει μόνο συμφωνούν ή διαφωνούν με τις προτιμήσεις των υπόλοιπων χρηστών και, συνήθως, είναι δύσκολη η ακριβής πρόταση περιεχομένου σε αυτούς. Ακόμη, είναι δυνατό να επηρεάσουν αρνητικά την ορθότητα των προτάσεων περιεχομένου και για την υπόλοιπη κοινότητα [18].

- Black Sheep: Ο όρος "μαύρο πρόβατο" (black sheep) αποδίδεται στους χρήστες των οποίων οι ιδιαίτερες προτιμήσεις μπορούν να συσχετιστούν με ελάχιστους ή κανέναν χρήστες. Επομένως, η πρόταση περιεχομένου σε αυτούς είναι ιδιαίτερα δύσκολη ή και αδύνατη [19].

- Fraud / Shilling Attacks: Είναι πιθανό οι εταιρείες παραγωγής ενός προϊόντος να χρησιμοποιήσουν αθέμιτα μέσα προκειμένου να αυξήσουν τη βαθμολογία των προϊόντων τους εν συγκρίσει με αυτήν ανταγωνιστικών προϊόντων· γνωστό ως push attacks. Αντίστοιχα, είναι δυνατό να προσπαθήσουν να ελαττώσουν τη βαθμολογία των ανταγωνιστών· γνωστό ως nuke attacks [19][20].

- Diversity και Long Tail: Τα recommendation systems αναμένεται να αυξήσουν την ποικιλία, εφόσον συμβάλλουν στον εντοπισμό νέων αντικειμένων. Ωστόσο, κάποιοι αλγόριθμοι μπορεί να έχουν το αντίθετο αποτέλεσμα. Όταν προτείνονται τα πιο υψηλά ή περισσότερο βαθμολογημένα αντικείμενα, μένουν στην αφάνεια άλλα με περιορισμένα δεδομένα ιστορικού. Η κατάσταση αυτή ευνοεί τα δημοφιλή αντικείμενα και αυτό μπορεί να αποτρέψει την πρόταση αντικειμένων που ίσως ταίριαζαν καλύτερα στο χρήστη. Έτσι, δεν ευνοείται η ποικιλία και το long tail.

- Overfitting [21]: Προκειμένου να γίνουν αξιόπιστες προβλέψεις χρειάζεται η ανάπτυξη ενός μοντέλου που να ταιριάζει στο training set, αλλά να μπορεί να παράγει ακριβείς προβλέψεις και για γενικά δεδομένα που δεν ανήκουν σε αυτό. Στο overfitting ένα μοντέλο περιγράφει τυχαίο σφάλμα (random error) ή θόρυβο (noise) αντί για την πραγματική σχέση. Το overfitting παρουσιάζεται όταν το μοντέλο που χρησιμοποιείται είναι εξαιρετικά πολύπλοκο και έχει πολλές παραμέτρους σε σχέση με τον αριθμό των δεδομένων. Οι προβλέψεις του αλγορίθμου για δεδομένα εκτός του training set είναι λιγότερο ακριβείς, καθώς ο αλγόριθμος αντιδρά υπερβολικά σε μικρές διακυμάνσεις. Το overfitting μπορεί να συμβεί επειδή το κριτήριο που χρησιμοποιείται για την ανάπτυξη του μοντέλου δεν είναι το ίδιο με αυτό που χρησιμοποιείται για να κριθεί η αποτελεσματικότητα ενός μοντέλου. Πιο συγκεκριμένα, ένα μοντέλο δύναται να αναπτυχθεί κατά τέτοιο τρόπο, ώστε να μεγιστοποιεί την επίδοσή του στα δεδομένα του training set. Ωστόσο, η αποτελεσματικότητά του κρίνεται από την ικανότητά του να παράγει ακριβή αποτελέσματα σε νέα δεδομένα. Δηλαδή, αντί

να "μαθαίνει" να γενικεύει μέσω των δεδομένων του training set, αυτό "απομνημονεύει" τα δεδομένα αυτά και προσαρμόζεται για να λειτουργεί άρτια μόνο πάνω σε αυτά.

2.5 Υπολογισμός Ομοιότητας

Ως μέτρο ομοιότητας (similarity measure) ή συνάρτηση ομοιότητας (similarity function) [22] ορίζεται μια πραγματική συνάρτηση που προσδιορίζει ποσοτικά την ομοιότητα μεταξύ δυο αντικειμένων. Κατά μια έννοια, τα μέτρα αυτά είναι αντίστροφα των μετρικών απόστασης, εφόσον λαμβάνουν μεγάλες τιμές αν τα αντικείμενα είναι κοντά μεταξύ τους και μικρές τιμές αν απέχουν πολύ. Ακολούθως, θα αναφερθούν μερικά αξιοσημείωτα μέτρα υπολογισμού similarity.

2.5.1 Euclidean Distance

Ως βάση πολλών μέτρων ομοιότητας (similarity) ή καλύτερα διαφορετικότητας (dissimilarity) θεωρείται η Ευκλείδεια απόσταση (Euclidean distance). Ως Ευκλείδεια απόσταση [23] μεταξύ δυο σημείων ορίζεται το μήκος της ευθείας γραμμής που τα συνδέει. Έστω διανύσματα $\boldsymbol{p}=(p_1,p_2,..,p_n)$ και $\boldsymbol{q}=(q_1,q_2,...,q_n)$. Η Ευκλείδεια απόστασή τους είναι:

$$d(\boldsymbol{p},\boldsymbol{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Τα q_i και p_i μπορούν να αντικατασταθούν από τις βαθμολογίες δυο χρηστών για αντίστοιχα αντικείμενα. Με άλλα λόγια, η Ευκλείδεια απόσταση είναι η τετραγωνική ρίζα του αθροίσματος των τετραγώνων των διαφορών μεταξύ αντίστοιχων στοιχείων των δυο διανυσμάτων [24]. Μεγάλη τιμή της απόστασης αυτής σημαίνει μικρό similarity και το αντίστροφο.

2.5.2 Cosine Similarity

Η ομοιότητα συνημίτονου (cosine similarity) [25] είναι ένα μέτρο ομοιότητας μεταξύ δυο μη-μηδενικών διανυσμάτων που υπολογίζει το συνημίτονο της μεταξύ τους γωνίας. Οι τιμές που λαμβάνει περιορίζονται, προφανώς, στο διάστημα [-1,1]. Το συνημίτονο των 0° είναι 1 και για κάθε άλλη γωνία μικρότερο του 1. Το cosine similarity εκφράζει προσανατολισμό. Δυο διανύσματα με ίδιο προσανατολισμό έχουν similarity 1 που υποδεικνύει ότι είναι ίδια, με γωνία 90° έχουν similarity 0 που δείχνει ότι είναι

ασυσχέτιστα μεταξύ τους, ενώ αν είναι διαμετρικά αντίθετα έχουν similarity -1 που σημαίνει ότι είναι ακριβώς αντίθετα, ανεξαρτήτως του μέτρου τους. Έστω διανύσματα ***A*** και ***B*** με similarity:

$$similarity = cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

όπου A_i, B_i συνιστώσες των ***A*** και ***B*** διανυσμάτων.

2.5.3 Pearson Correlation Coefficient

Ο συντελεστής συσχέτισης του Pearson (Pearson correlation coefficient/ PCC ή Pearson's r ή Pearson product-moment correlation coefficient/ PPMCC) [26] αποτελεί ένα ακόμη πολύ διαδεδομένο μέτρο ομοιότητας. Υποδεικνύει τη γραμμική εξάρτηση ή συσχέτιση μεταξύ δυο μεταβλητών και λαμβάνει τιμές στο διάστημα [-1,1], όπου η τιμή 1 υποδηλώνει μια πλήρως θετική γραμμική συσχέτιση, η τιμή 0 την απουσία γραμμικής συσχέτισης και η τιμή -1 μια πλήρως αρνητική γραμμική συσχέτιση. Όταν επιθυμούμε τον υπολογισμό του συντελεστή συσχέτισης του Pearson για δυο σύνολα δεδομένων (datasets) $\{x_1,x_2,...,x_n\}$ και $\{y_1,y_2,...,y_n\}$ που ανήκουν σε ένα δείγμα του πληθυσμού, χρησιμοποιούμε τον ακόλουθο τύπο:

$$r = \frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\overline{y})^2}}$$

όπου $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ είναι ο μέσος και αντίστοιχα για το $\overline{y}$.

Στην περίπτωση των recommendation systems οι τιμές x_i, y_i αποτελούν, συνήθως, τις βαθμολογίες δυο χρηστών σε κοινά βαθμολογημένα αντικείμενα και από τους δυο. Δηλαδή, η τιμή x_i είναι η βαθμολογία του χρήστη *x* για το αντικείμενο *i*. Η τιμή $\overline{x}$ παριστάνει το μέσο που προκύπτει από τις βαθμολογίες του *x* στο σύνολο των κοινών αντικειμένων που έχει με τον *y*. Αντίστοιχα προκύπτουν τα y_i και $\overline{y}$ για το χρήστη *y*.

2.6 Άμεσες και Έμμεσες Βαθμολογίες

Γενικά, υπάρχουν δυο τρόποι εξαγωγής πληροφοριών σχετικών με τις προτιμήσεις του χρήστη: άμεσα (explicitly) και έμμεσα (implicitly). Η άμεση συγκέντρωση πληροφοριών από τον χρήστη περιλαμβάνει τη συνειδητή παροχή βαθμολογιών για αντικείμενα συνήθως σε διακριτή αριθμητική κλίμακα. Από την άλλη πλευρά, η έμμεση συγκέντρωση πληροφοριών από το χρήστη προκύπτει όταν από τη συμπεριφορά του

εξάγονται συμπεράσματα τα οποία μεταφράζονται σε like ή dislike σχετικά με διάφορα αντικείμενα. Η μέθοδος αυτή, σε αντίθεση με την πρώτη, δεν απαιτεί από τον χρήστη να απαντήσει σε ερωτήσεις, προκειμένου το σύστημα να συλλέξει πληροφορίες για αυτόν [27].

Εφόσον οι explicit βαθμολογίες εκφράζουν ξεκάθαρα τη γνώμη του χρήστη για ένα αντικείμενο, έχουν ύψιστη σημασία. Ωστόσο, δεν είναι πάντα συνεπείς. Για παράδειγμα, ένας χρήστης μπορεί να βαθμολογήσει ένα αντικείμενο επηρεαζόμενος από τον κοινωνικό του περίγυρο και όχι καθαρά από την προσωπική του άποψη. Επίσης, είναι πιθανό δυο χρήστες στους οποίους άρεσε εξίσου ένα αντικείμενο να το βαθμολογήσουν διαφορετικά, καθώς αντιλαμβάνονται διαφορετικά την κλίμακα των βαθμολογιών και ο ένας τείνει να είναι περισσότερο ή λιγότερο συγκρατημένος σε βαθμολογικό επίπεδο. Τέλος, υπάρχει γενικά η τάση ο χρήστης να βαθμολογεί υψηλότερα ένα αντικείμενο όταν η διαδικασία βαθμολόγησης δεν απέχει πολύ χρονικά από την ημερομηνία κατανάλωσης [28].

Μέσω των implicit βαθμολογιών αντιμετωπίζονται ως ένα βαθμό τα προαναφερθέντα προβλήματα· παρουσιάζονται, ωστόσο, νέα. Αν χρησιμοποιηθεί ως πρακτική η μελέτη του χρόνου που ο χρήστης καταναλώνει σε μια ιστοσελίδα, είναι δύσκολο να διακριθεί αν στον χρήστη αρέσει ή όχι το περιεχόμενο της σελίδας ή αν πραγματικά κοιτάζει τη σελίδα ή ασχολείται με κάτι άλλο. Αν χρησιμοποιηθούν σελίδες που προβλήθηκαν ή αντικείμενα που καταναλώθηκαν, πάλι δεν είναι σίγουρο αν ο χρήστης είδε τη σελίδα τυχαία ή αν το αντικείμενο εν τέλει του άρεσε [28].

Παραδείγματα explicit τρόπων συγκέντρωσης βαθμολογιών [2] είναι τα εξής:

- Αίτημα προς το χρήστη να βαθμολογήσει ένα αντικείμενο σε μια αναλογική κλίμακα.
- Αίτημα προς το χρήστη να πραγματοποιήσει αναζητήσεις.
- Αίτημα προς το χρήστη να κατατάξει μια λίστα αντικειμένων από το αγαπημένο προς το λιγότερο αγαπημένο για αυτόν.
- Παρουσίαση δυο αντικειμένων και επιλογή ενός από το χρήστη.
- Αίτημα προς το χρήστη για δημιουργία μιας λίστας αντικειμένων που τον ενδιαφέρουν.

Παραδείγματα implicit τρόπων συγκέντρωσης βαθμολογιών [2] είναι τα εξής:

- Παρατήρηση των αντικειμένων που ο χρήστης βλέπει σε ένα διαδικτυακό κατάστημα.
- Ανάλυση αριθμού προβολών αντικειμένων / χρηστών.
- Διατήρηση αρχείου των αντικειμένων που ένας χρήστης αγοράζει μέσω διαδικτύου.

- Λήψη λίστας των προϊόντων που ένας χρήστης βλέπει ή ακούει στον υπολογιστή του.

- Ανάλυση του κοινωνικού δικτύου του χρήστη και ανακάλυψη κοινών like και dislike.

Και οι δυο στρατηγικές μπορούν εύκολα να συνδυαστούν για καλύτερα αποτελέσματα.

2.7 Βελτίωση Προβλέψεων με χρήση Contextual Πληροφοριών και Σχολίων Χρήστη

Τα περισσότερα recommendation systems λειτουργούν αξιοποιώντας πληροφορίες που σχετίζονται μόνο με το χρήστη και το αντικείμενο (δισδιάστατος χώρος), δίχως να λαμβάνονται υπόψη επιπρόσθετες συναφείς (contextual) πληροφορίες (πολυδιάστατος χώρος), που ενδεχομένως είναι καίριες για κάποιες εφαρμογές [29]. Σε πολλές περιπτώσεις, η χρησιμότητα ενός αντικειμένου εξαρτάται από το χρόνο (εποχή, μήνα, μέρα, ώρα), τους ανθρώπους με τους οποίους αυτό καταναλώνεται ή μοιράζεται και τις εκάστοτε συνθήκες. Για παράδειγμα, όταν προτείνεται ένα πακέτο διακοπών θα πρέπει να ληφθεί υπόψη η εποχή του χρόνου που ο χρήστης θα ταξιδέψει, τα άτομα με τα οποία θα πάει διακοπές, οι συνθήκες του ταξιδιού και οι ενδεχόμενοι περιορισμοί. Ακόμη, ένας χρήστης είναι πιθανό να έχει σημαντικά διαφορετικές προτιμήσεις σε ταινίες ανάλογα με το πότε και με ποιους τις βλέπει. Δηλαδή, άλλη ταινία θα επιλέξει ο χρήστης να δει Σάββατο βράδυ με το σύντροφό του και άλλη Τετάρτη απόγευμα με την οικογένειά του.

Πολλοί δισδιάστατοι αλγόριθμοι για recommendation δεν είναι εφικτό να επεκταθούν άμεσα σε πολυδιάστατους. Για αυτές τις περιπτώσεις, προτείνεται μια προσέγγιση που χρησιμοποιεί για την πρόβλεψη των προτιμήσεων μόνο τις βαθμολογίες που αναφέρονται στο πλαίσιο των κριτηρίων που έχουν προσδιοριστεί από το χρήστη (reduction-based recommendation approach). Επί παραδείγματι, στη reduction-based recommendation προσέγγιση για να προταθεί μια ταινία σε ένα άτομο που επιθυμεί να τη δει στο σινεμά Σάββατο βράδυ, χρησιμοποιούνται μόνο οι διαθέσιμες βαθμολογίες των ταινιών που προβάλλονται στις αίθουσες τα Σάββατα, εφόσον αυτές οι παράμετροι επηρεάζουν τη συμπεριφορά του χρήστη. Η προσέγγιση αυτή, με την επιλογή μόνο των βαθμολογιών που σχετίζονται με το πλαίσιο της πρότασης περιεχομένου, προβάλλει τον πολυδιάστατο "κύβο" προβλέψεων στις δυο βασικές διαστάσεις (χρήστης και αντικείμενο).

Οι κριτικές - σχόλια περιέχουν τη γνώμη των χρηστών για ένα αντικείμενο ή την άποψή τους για χαρακτηριστικά του. Τεχνικές "εξόρυξης" γνώμης (opinion mining techniques) εφαρμόζονται σε κριτικές χρηστών για να καθορίσουν το συναισθηματικό προσανατολισμό καθενός σχετικά με κάθε χαρακτηριστικό του αντικειμένου, υποδεικνύοντας έτσι αν στο χρήστη άρεσε ή όχι το αντικείμενο υπό το πρίσμα αυτού του χαρακτηριστικού. Επίσης, καθορίζεται ο ολικός προσανατολισμός κάθε κριτικής, ώστε

να κατανοήσουμε αν ο χρήστης έχει θετική, αρνητική ή ουδέτερη εικόνα για το αντικείμενο συνολικά. Δηλαδή, οι απόψεις του χρήστη, όπως προκύπτουν από τις κριτικές του, αντικατοπτρίζουν την εικόνα του για την ποιότητα του αντικειμένου. Μια θετικά διακείμενη κριτική υποδηλώνει ικανοποίηση με ορισμένα τουλάχιστον χαρακτηριστικά του. Αναγνωρίζοντας και απομονώνοντας αυτά τα χαρακτηριστικά, είναι δυνατός ο καθορισμός των αντικειμένων που μπορεί να ενδιαφέρουν το χρήστη. Συνεπώς, παράγοντας μοτίβα (patterns) και εφαρμόζοντας κανόνες συσχέτισης γίνεται εφικτό να προταθούν στο χρήστη αντικείμενα που συμπίπτουν με τις προτιμήσεις του [30].

Με τον τρόπο αυτό, προτείνεται στο χρήστη πρωτότυπο και διαφορετικό, συνήθως, περιεχόμενο βάσει των προτιμήσεων του, όπως εκφράζονται από τις κριτικές του. Ακόμη, οι προτάσεις αυτές αντικατοπτρίζουν κάποιες φορές καλύτερα τις προτιμήσεις και απαιτήσεις του, εφόσον προκύπτουν από τα μεμονωμένα χαρακτηριστικά που τον ενδιαφέρουν σε κάθε αντικείμενο. Επιπλέον, η εξαγωγή βαθμολογιών από σχόλια επιτρέπει τη συλλογή περισσότερων δεδομένων για το χρήστη από το σύστημα εν συγκρίσει μια απλή βαθμολόγηση του αντικειμένου. Τέλος, στη σύγχρονη κοινωνία είναι πολύ συχνό και σχεδόν δεδομένο ότι ο χρήστης θα κάνει σχόλια στα κοινωνικά δίκτυα για τα αντικείμενα που καταναλώνει. Με τη χρήση, λοιπόν, αυτών των σχολίων δεν τον υποβάλλουμε στη διαδικασία βαθμολόγησης των αντικειμένων· δεν απαιτούμε, δηλαδή, από αυτόν επιπλέον κόπο και χρόνο.

2.8 Neo4j Βάση Δεδομένων

Η Neo4j [31] είναι μια βάση δεδομένων υπό μορφή γράφου που αναπτύχθηκε από τη Neo Technology, Inc. Η Neo4j συμμορφώνεται στο σύνολο ιδιοτήτων ACID: Ατομικότητα (Atomicity), Συνέπεια (Consistency), Απομόνωση (Isolation), Διάρκεια/ Ανθεκτικότητα (Durability). Είναι υλοποιημένη σε Java και προσβάσιμη από άλλες γλώσσες με τη χρήση της γλώσσας ερωταποκρίσεων Cypher μέσω ενός HTTP σημείου συναλλαγών (transactional HTTP endpoint).

Η αρχιτεκτονική έχει σχεδιαστεί για τη βελτίωση της διαχείρισης, του αποθηκευτικού χώρου και της διάσχισης κόμβων και ακμών. Με αυτήν την προσέγγιση, η διάσχιση πραγματοποιείται πολύ γρήγορα, εφόσον η αναζήτηση γίνεται τοπικά, στην ευρύτερη γειτονιά του κόμβου. Επομένως, η ποσότητα των αποθηκευμένων δεδομένων στη βάση δεν επηρεάζει το χρόνο εκτέλεσης των λειτουργιών. Η υψηλή επεκτασιμότητα και η αποδοτικότητα λειτουργιών μνήμης αποτελούν βασικά πλεονεκτήματα της Neo4j. Η ύπαρξη γράφου επιτρέπει τη συνεπή χρήση του ίδιου μοντέλου σε όλη τη διάρκεια της επινόησης, σχεδίασης, υλοποίησης, αποθήκευσης και οπτικοποίηση οποιουδήποτε τμήματος ή σεναρίου χρήσης.

Η Neo4j υποστηρίζει τη γρήγορη ανάπτυξη συστημάτων υπό μορφή γράφου. Η ανάπτυξή της προέρχεται από την ανάγκη να τρέξουμε σε πραγματικό χρόνο (real-time) ερωτήματα (queries) που αφορούν σε συναφείς πληροφορίες· κάτι που καμία άλλη βάση

δεδομένων δεν μπορεί να προσφέρει. Μπορεί να αποθηκεύσει εκατοντάδες τρισεκατομμύρια οντότητες, να μειώσει το κόστος και την πολυπλοκότητα λειτουργιών [32].

Στη Neo4j όλα είναι αποθηκευμένα σε μορφή κόμβου, ακμής ή ιδιότητας. Κάθε κόμβος ή ακμή μπορούν να έχουν οποιονδήποτε αριθμό ιδιοτήτων. Τόσο οι κόμβοι όσο και οι ακμές είναι δυνατόν να έχουν ετικέτες. Οι ετικέτες μπορούν να χρησιμοποιηθούν για τον περιορισμό των αναζητήσεων και την ανάθεση ρόλων ή τύπων. Κατόπιν, προστέθηκε και η λειτουργία ευρετηρίου (indexing). Γενικά, οι κόμβοι παριστάνουν οντότητες του συστήματος (χρήστες, αντικείμενα), ενώ οι ακμές σχέσεις που συνδέουν τις οντότητες αυτές μεταξύ τους. Οι ακμές αυτές έχουν κατεύθυνση, αλλά κατά την προσπέλασή τους μπορεί να αγνοηθεί, εφόσον αυτό είναι επιθυμητό. Παρακάτω πραγματοποιείται μια εισαγωγή στη γλώσσα που επιτρέπει την προσπέλαση των βάσεων Neo4j.

2.8.1 Γλώσσα ερωταποκρίσεων Cypher

Η Cypher [33] είναι μια δηλωτική γλώσσα ερωταποκρίσεων για γράφους που επιτρέπει την πραγματοποίηση εκφραστικών και αποδοτικών queries και την ενημέρωση του αποθηκευμένου γράφου. Είναι σχετικά απλή γλώσσα, αλλά με πληθώρα δυνατοτήτων. Πολύπλοκα queries μπορούν εύκολα να εκφραστούν μέσω της Cypher, γεγονός που επιτρέπει να εστιάσουμε στην ουσία και όχι στον τρόπο πρόσβασης στη βάση δεδομένων.

Η δομή και πολλές από τις λέξεις-κλειδιά (keywords) που χρησιμοποιούνται είναι εμπνευσμένα από την SQL. Το ταίριασμα προτύπων (pattern matching) δανείζεται προσεγγίσεις έκφρασης από τη SPARQL. Τέλος, δανείζεται στοιχεία σε σημασιολογικό επίπεδο από τις γλώσσες Haskell και Python.

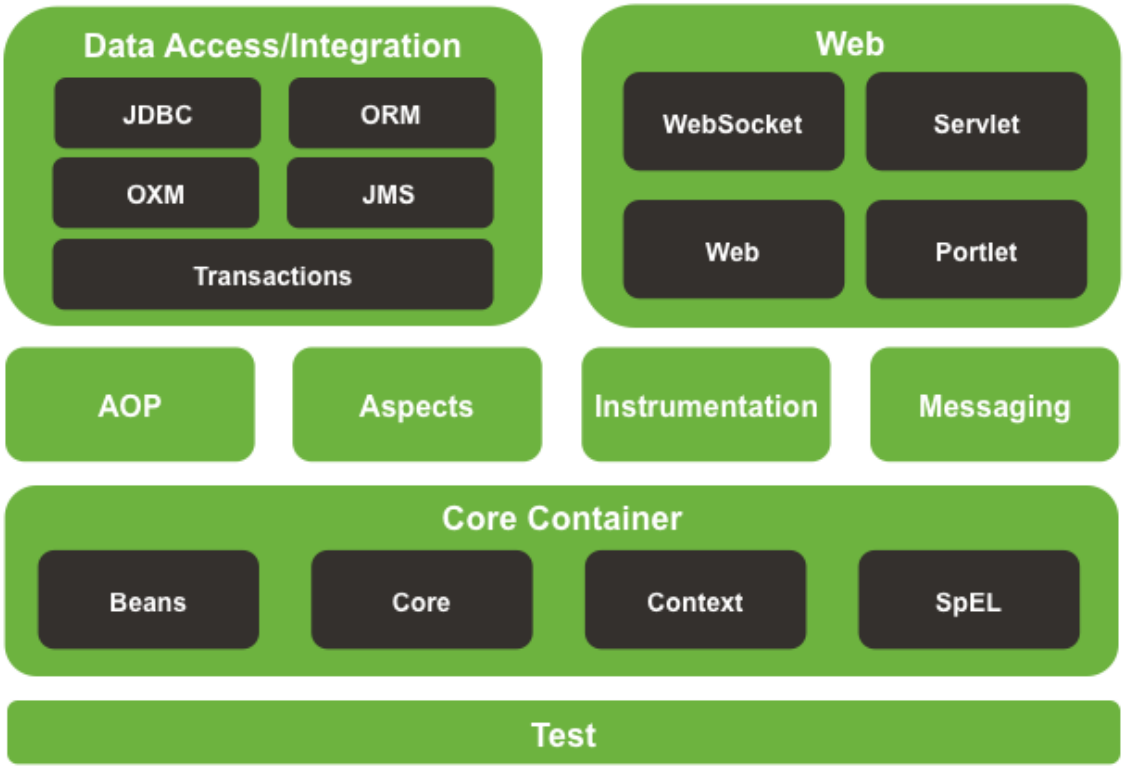
Κεφάλαιο 3

Τεχνικές Προδιαγραφές και Παρεχόμενο Σύστημα

3.1 Spring Framework

Η διαδικασία ανάπτυξης μεγάλων Java εφαρμογών έχει αλλάξει, εφόσον η χρήση "βαρέων" αρχιτεκτονικών (Heavyweight Architecture), όπως τα Enterprise Java Beans (EJB), μειώνεται και αντικαθίσταται από πιο "ελαφριά" περιβάλλοντα εργασίας (frameworks). Ένα τέτοιο ιδιαίτερα διαδεδομένο, εύχρηστο και ποιοτικό περιβάλλον εργασίας είναι το Spring Framework [34]. Το project SAM που αποτέλεσε βάση για την πραγματοποίηση της παρούσας διπλωματικής υλοποιήθηκε με τη βοήθεια του ελεύθερου αυτού λογισμικού (open source).

Τα χαρακτηριστικά του πυρήνα (core) του framework μπορούν να χρησιμοποιηθούν σε οποιαδήποτε εφαρμογή Java και υπάρχουν επεκτάσεις για τη δημιουργία διαδικτυακών εφαρμογών πάνω σε πλατφόρμα Java EE. Το Spring Framework περιλαμβάνει διάφορα τμήματα (modules) και όσα από αυτά κρίνονται χρήσιμα για την εκάστοτε εφαρμογή, μπορούν να χρησιμοποιηθούν από το χρήστη. Στο ακόλουθο σχήμα παρουσιάζονται τα modules του Spring Framework:



Σχήμα 3.1: Spring Framework modules [35]

Τα modules αυτά παρέχουν ένα μεγάλο εύρος υπηρεσιών:

- Spring Core Container: Είναι το βασικό module του Spring Framework και είναι υπεύθυνο για τη βασική του λειτουργικότητα. Το module αυτό παρέχει τους Spring containers (BeanFactory, ApplicationContext).

- Aspect-oriented programming (AOP): Επιτρέπει τη σωστή διαχείριση των cross-cutting concerns [36], δηλαδή τμημάτων ενός προγράμματος που βασίζονται σε ή επηρεάζουν άλλα τμήματα του συστήματος και είναι δύσκολο να γίνει η απομόνωσή τους από το υπόλοιπο σύστημα τόσο στο σχεδιασμό όσο και στην υλοποίηση. Αυτό μπορεί να οδηγήσει στην ύπαρξη αντιγράφων του κώδικα (scattering), σε σημαντικές εξαρτήσεις μεταξύ συστημάτων (tangling) ή και στα δυο.

- Authentication και authorisation: Διαδικασίες ασφαλείας που υποστηρίζουν ένα εύρος προτύπων, πρωτοκόλλων, εργαλείων και πρακτικών μέσω του Spring Security.

- Convention over configuration: Μια γρήγορη λύση ανάπτυξης εφαρμογής για βασισμένες σε Spring εφαρμογές παρέχεται στο Spring Roo module.

- Data access: Η εργασία με συστήματα διαχείρισης σχεσιακών βάσεων σε Java πλατφόρμα με χρήση JDBC, object-relational mapping εργαλείων και NoSQL βάσεων δεδομένων.

- Inversion of control container: Ρύθμιση των δομικών στοιχείων της εφαρμογής και διαχείριση του κύκλου ζωής των αντικειμένων Java, κυρίως μέσω του dependency injection.

- Messaging: Προσφέρεται υποστήριξη για ενσωμάτωση συστημάτων ανταλλαγής μηνυμάτων [37].

- Model-view-controller: Ένα πλαίσιο εργασίας βασισμένο σε HTTP και servlet που παρέχει δυνατότητα για επεκτάσεις και προσαρμογές σε διαδικτυακές εφαρμογές και διαδικτυακές υπηρεσίες RESTful.

- Remote access framework: Επιτυγχάνει την εργασία με διάφορες τεχνολογίες βασισμένες σε RPC, διαθέσιμες στην Java πλατφόρμα τόσο για τη συνδεσιμότητα του πελάτη όσο και για τη διαλογή αντικειμένων στους servers. Το πιο σημαντικό προσφερόμενο χαρακτηριστικό είναι η διευκόλυνση της διαμόρφωσης και χρήσης αυτών των τεχνολογιών με το συνδυασμό του AOP και της αντιστροφής ελέγχου (inversion of control).

- Transaction management: Ενοποιεί διάφορα APIs διαχείρισης συναλλαγών και συντονίζει συναλλαγές αντικειμένων Java.

- Remote management: Προβολή και διαχείριση αντικειμένων Java για τοπική ή απομακρυσμένη ρύθμιση μέσω JMX.

- Testing: Υποστήριξη κλάσεων για έλεγχο μονάδων και ενσωμάτωσης.

3.2 Gradle

Το Gradle [38][39] είναι ένα σύστημα για αυτοματοποίηση του build και υποστήριξη της ανάπτυξης κώδικα σε διάφορες γλώσσες προγραμματισμού. Προσφέρει ένα ευέλικτο μοντέλο που δύναται να υποστηρίξει ολόκληρο τον κύκλο ζωής των προγραμμάτων. Έχει ενσωματωμένα εργαλεία ανάπτυξης και servers που επιτρέπουν τη διαρκή ενσωμάτωση (continuous integration). Έτσι διατηρείται ενημερωμένος ο κώδικας, οι βιβλιοθήκες και άλλες πηγές που δημιουργούν εξαρτήσεις (dependencies) ή πιθανές συγκρούσεις (conflicts). Χρησιμοποιεί έναν κατευθυνόμενο ακυκλικό γράφο για να αποφασίσει τη σειρά με την οποία μπορούν να εκτελεστούν οι εργασίες. Το Gradle σχεδιάστηκε για να γίνεται build πολλαπλών project, ενδεχομένως πολύ μεγάλων, και υποστηρίζει σταδιακά αυξανόμενα builds, αποφασίζοντας ποια τμήματα είναι ενημερωμένα, ώστε οποιαδήποτε εργασία εξαρτώμενη από αυτά να μην ξαναεκτελεστεί.

3.3 Βάση Δεδομένων και Σύνδεσή της με το Σύστημα

Το σύστημα συνδέεται με neo4j βάση δεδομένων, στην οποία πραγματοποιείται η πρόσβαση από το URL: http://localhost:7474. Η διαθέσιμη βάση αφορά σε ταινίες και χρήστες που αλληλεπιδρούν με αυτές. Πιο συγκεκριμένα, η βάση δεδομένων αποτελείται από τους ακόλουθους κόμβους (nodes) που κατηγοριοποιούνται βάσει label:

- Person: Οι κόμβοι αυτοί είναι, ουσιαστικά, οι αποθηκευμένοι στη βάση χρήστες. Το πλήθος τους είναι 619, όπως προκύπτει από το Cypher query:

```
1. MATCH (n:Person)
2. RETURN COUNT(n)
```

Listing 3.1: Cypher query για πλήθος χρηστών

- Asset: Οι κόμβοι αυτοί είναι οι αποθηκευμένες στη βάση ταινίες. Είναι 30 στο πλήθος, όπως προκύπτει από το Cypher query:

```
MATCH (n:Asset)
RETURN COUNT(n)
```

Listing 3.2: Cypher query για πλήθος asset

- Keyword: Οι κόμβοι αυτοί αποτελούν τις αποθηκευμένες στη βάση δεδομένων λέξεις-κλειδιά. Είναι 98 στο σύνολο, όπως φαίνεται από την εκτέλεση του Cypher query:

```
MATCH (n:Keyword)
RETURN COUNT(n)
```

Listing 3.3: Cypher query για πλήθος keywords

Οι σχέσεις (relationships) που υπάρχουν στη βάση δεδομένων είναι οι εξής:

- CONSUMES: Οι σχέσεις αυτές συνδέουν έναν κόμβο Person με έναν κόμβο Asset, με προσανατολισμό από τον πρώτο στο δεύτερο και δείχνουν ότι ο χρήστης Person κατανάλωσε την ταινία Asset. Το πλήθος των σχέσεων αυτών είναι 7038, όπως φαίνεται από την εκτέλεση του Cypher query:

```
MATCH p=()-[r:CONSUMES]->()
RETURN COUNT(p)
```

Listing 3.4: Cypher query για πλήθος σχέσεων CONSUMES

- COMMENTS: Οι σχέσεις αυτές συνδέουν έναν κόμβο Person με έναν κόμβο Asset, με προσανατολισμό από τον πρώτο στο δεύτερο και δείχνουν ότι ο χρήστης Person σχολίασε την ταινία Asset. Είναι 7038 σε πλήθος, όπως προκύπτει από το Cypher query:

```
MATCH p=()-[r:COMMENTS]->()
RETURN COUNT(p)
```

Listing 3.5: Cypher query για πλήθος σχέσεων COMMENTS

- HAS_KEYWORD: Οι σχέσεις αυτές συνδέουν έναν κόμβο Asset με έναν κόμβο Keyword, με προσανατολισμό από τον πρώτο στο δεύτερο και δείχνουν με ποιες λέξεις-κλειδιά σχετίζεται κάθε Asset. Το πλήθος των σχέσεων αυτών είναι 147, όπως φαίνεται από την εκτέλεση του Cypher query:

```
MATCH p=()-[r:HAS_KEYWORD]->()
RETURN COUNT(p)
```

Listing 3.6: Cypher query για πλήθος σχέσεων HAS_KEYWORD

Πέραν των προαναφερθέντων κόμβων και σχέσεων υπάρχουν στη βάση και αρκετά Property keys που χρησιμεύουν ως ιδιότητες τους. Ακολούθως, θα αναφερθούν τα σημαντικότερα εξ αυτών:

- personId: Είναι ιδιότητα κάθε κόμβου Person και μάλιστα αποτελεί το διακριτικό μεταξύ τους χαρακτηριστικό, δηλαδή κάθε χρήστης έχει διαφορετική τιμή σε αυτό το πεδίο. Έχει τη μορφή "user:αριθμός/id".

- assetId: Είναι ιδιότητα κάθε κόμβου Asset και αποτελεί το διακριτικό μεταξύ τους χαρακτηριστικό, δηλαδή κάθε ταινία έχει διαφορετική τιμή σε αυτό το πεδίο. Έχει τη μορφή "sam:αριθμός/id".

- title: Εντάσσεται στις ιδιότητες κάθε κόμβου Asset και περιέχει το όνομα κάθε ταινίας και σε παρένθεση τη χρονολογία που προβλήθηκε για πρώτη φορά.

- keywordId: Αποτελεί ιδιότητα κάθε κόμβου Keyword και περιέχει τη λέξη ή φράση-κλειδί του.

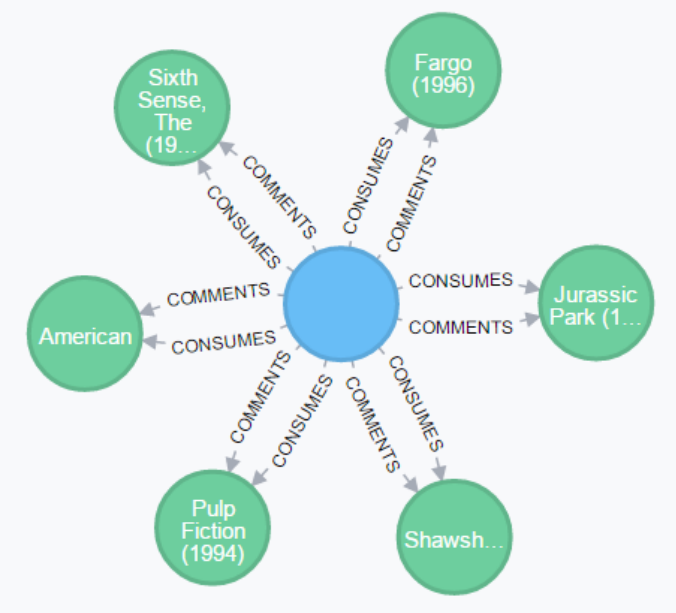
- isTrain: Αποτελεί ιδιότητα των σχέσεων COMMENTS και δείχνει ποιες ανήκουν και ποιες όχι στο training set. Όσες ανήκουν έχουν τιμή 1 σε αυτήν την ιδιότητα, ενώ όσες δεν ανήκουν έχουν τιμή 0. Η κατανομή των σχέσεων σε training και test set έγινε με ποσοστά 70%/30% αντίστοιχα.

- sentimentCategory: Εντάσσεται στις ιδιότητες των σχέσεων COMMENTS και δείχνει αν η ολική άποψη του χρήστη για το Asset, όπως αυτή προκύπτει από το sentiment analysis του comment που έγραψε, είναι θετική ή αρνητική. Λαμβάνει τιμές "positive" ή "negative" αντίστοιχα.

- intensity: Εντάσσεται στις ιδιότητες των σχέσεων COMMENTS και δείχνει, κατά απόλυτη τιμή, την ένταση της άποψης του χρήστη για το Asset. Ποσοτικοποιεί, δηλαδή, την άποψη που έχει ο χρήστης για ένα Asset, όπως αυτή προκύπτει από το sentiment analysis του comment. Λαμβάνει τιμές από 0 ως 1. Σε συνδυασμό με το sentimentCategory προσφέρει ολοκληρωμένη εικόνα για τη γνώμη του χρήστη σε σχέση με την εκάστοτε ταινία.

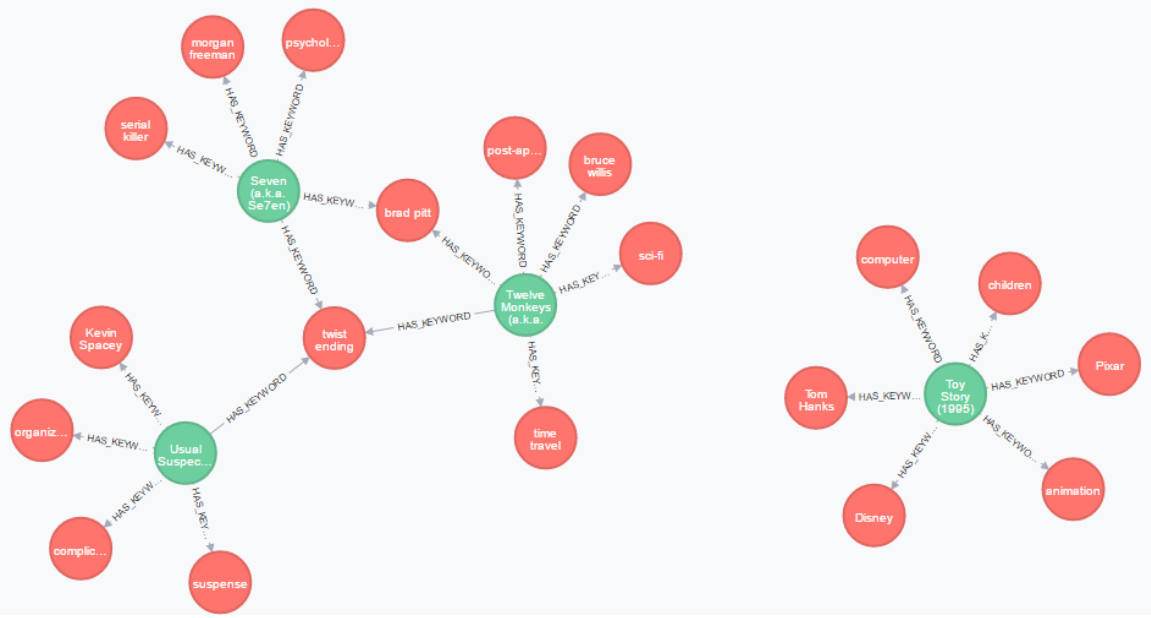
- timeOfAction: Αποτελεί ιδιότητα των σχέσεων COMMENTS και CONSUMES και εκφράζει τον ακριβή χρόνο στον οποίο έγινε το σχόλιο ή καταναλώθηκε η ταινία αντίστοιχα.

Στη συνέχεια, για μεγαλύτερη εξοικείωση, παρατίθενται σχηματικά κάποιες οντότητες και σχέσεις από τη βάση δεδομένων:



Σχήμα 3.2: Απεικόνιση σε γράφο αλληλεπίδρασης Person με διάφορα Asset

Στο παραπάνω σχήμα (Σχήμα 3.2) ο κεντρικός κόμβος παριστάνει ένα χρήστη και οι κόμβοι με τους οποίους συνδέεται τις ταινίες που έχει καταναλώσει (σχέση CONSUMES) ή σχολιάσει (σχέση COMMENTS).

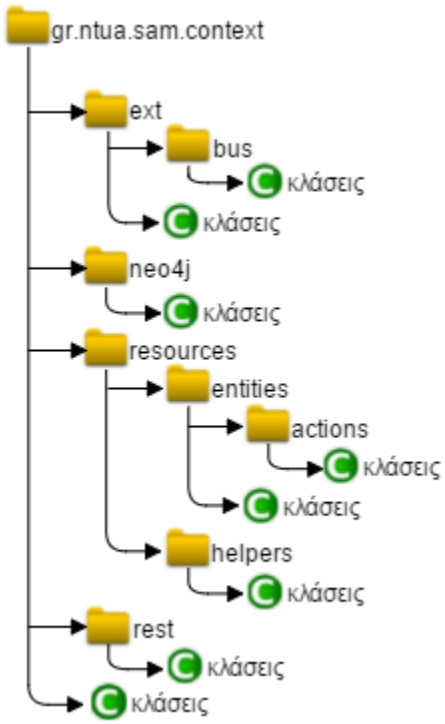


Σχήμα 3.3: Απεικόνιση σε γράφο σύνδεσης Assets με Keywords

Στο Σχήμα 3.3 παρουσιάζονται 4 κόμβοι Asset καθένας από τους οποίους συνδέεται (σχέση HAS_KEYWORD) με κόμβους Keyword και μάλιστα κάποιοι από τους δεύτερους είναι κοινοί σε δυο ή και παραπάνω Asset. Δηλαδή, κάποιες λέξεις-κλειδιά σχετίζονται με πάνω από μία ταινίες.

3.4 Δομή και Κλάσεις Συστήματος

Το σύστημα υλοποιείται με τη χρήση Java και αποτελείται από πληθώρα κλάσεων, οργανωμένων σε διαφορετικούς φακέλους ανάλογα με το σκοπό και τη λειτουργικότητά τους. Ακολούθως, θα αναπτυχθούν όσες από αυτές απαιτούνται για να τεθούν οι βάσεις για την κατανόηση και ανάπτυξη της παρούσας διπλωματικής εργασίας. Στο ακόλουθο σχήμα παρουσιάζεται με κάποια απλοποίηση η δομή του συστήματος σε επίπεδο διάρθρωσης φακέλων και κλάσεων.



Σχήμα 3.4: Διάρθρωση φακέλων και κλάσεων συστήματος

Στο άνωθεν σχήμα (Σχήμα 3.4) παρουσιάζεται απλοποιημένα η διάρθρωση των βασικών φακέλων και κλάσεων του συστήματος. Ο φάκελος ext δε μας ενδιαφέρει άμεσα, επομένως δε θα προχωρήσουμε σε ανάλυσή του. Ο φάκελος neo4j του project περιέχει τα repositories με τη συνδρομή των οποίων γίνεται η ανάκτηση δεδομένων από τη βάση neo4j. Ο φάκελος resources περιέχει τους φακέλους entities και helpers. Ο φάκελος entities περιλαμβάνει τις οντότητες του συστήματος και το φάκελο actions που περιέχει

κλάσεις σχετικές με τις ενέργειες που δύναται να πραγματοποιηθούν σε αυτές τις οντότητες. Ο φάκελος helpers περιέχει κλάσεις που περιλαμβάνουν τα στοιχεία του συστήματος που θεωρούνται χρήσιμα κάθε φορά ως σύνολο και συμβάλλουν στην επεξεργασία των δεδομένων. Περιέχει, δηλαδή, κλάσεις που σε καθεμία από αυτές υπάρχουν ομαδοποιημένα τα στοιχεία εκείνα του συστήματος που θα φανούν χρήσιμα αποτελώντας μια ενιαία μονάδα. Ο φάκελος rest περιλαμβάνει κλάσεις μέσω των οποίων τελείται η επιθυμητή επεξεργασία των δεδομένων. Τέλος, οι κλάσεις που βρίσκονται στο ίδιο επίπεδο με τους φακέλους ext, neo4j, resources και rest αφορούν στη λειτουργία του συστήματος και στη σύνδεσή του με τη βάση δεδομένων και δε θα αναπτυχθούν περαιτέρω. Στη συνέχεια, αναφέρονται οι σημαντικότερες κλάσεις, πεδία και συναρτήσεις που αφορούν στην παρούσα εργασία, με σειρά που εξυπηρετεί την καλύτερη κατανόησή τους.

Οι βασικότερες κλάσεις του φακέλου entities (εκτός των κλάσεων που περιέχονται στον φάκελο actions) είναι οι εξής:

- Asset
  Η κλάση αυτή αντιπροσωπεύει κάθε ταινία της βάσης δεδομένων. Τα σημαντικότερα πεδία που περιέχει είναι τα ακόλουθα:
  - id (τύπου Long): Αποτελεί το id του στοιχείου στο γράφο της βάσης δεδομένων.
  - assetId (τύπου String): Αποτελεί το id της ταινίας και είναι της μορφής "sam:αριθμός/id". Η λήψη δεδομένων από τη βάση για κάποια ταινία πραγματοποιείται βάσει αυτού του πεδίου.
  - title (τύπου String): Στο πεδίο αυτό αναγράφεται ο τίτλος της ταινίας και σε παρένθεση η χρονολογία που αυτή προβλήθηκε πρώτη φορά.

  Ακόμη, υπάρχουν κατασκευαστές (constructors) για αντικείμενα της κλάσης· ένας constructor δίχως και ένας με παραμέτρους που αρχικοποιεί βάσει αυτών τα πεδία του αντικειμένου. Τέλος, υπάρχουν και οι λεγόμενοι "getters" και "setters", δηλαδή μέθοδοι που επιτρέπουν στο χρήστη να λάβει ή να ορίσει μεμονωμένα τα οριζόμενα πεδία του κάθε αντικειμένου: getId(), setId(Long id), getAssetId(), setAssetId(String assetId), getTitle(), setTitle(String title).

- Person
  Η κλάση αυτή αντιπροσωπεύει κάθε χρήστη της βάσης δεδομένων. Τα βασικότερα πεδία που περιέχει είναι τα ακόλουθα:
  - id (τύπου Long): Αποτελεί το id του στοιχείου στο γράφο της βάσης δεδομένων.
  - personId (τύπου String): Αποτελεί το id του χρήστη και είναι της μορφής "user:αριθμός/id". Η λήψη δεδομένων από τη βάση για κάποιο χρήστη, πραγματοποιείται βάσει αυτού του πεδίου.

  Επιπλέον, υπάρχουν τρεις κατασκευαστές (constructors) για αντικείμενα της κλάσης· ένας χωρίς παραμέτρους, ένας με ορισμένες παραμέτρους, ώστε να δοθεί τιμή σε κάποια βασικά πεδία, και ένας πλήρης constructor που έχει μια

παράμετρο για κάθε πεδίο του αντικειμένου. Τέλος, υπάρχουν και οι μέθοδοι: getId(), setId(Long id), getPersonId(), setPersonId(String personId).

Οι βασικότερες κλάσεις του φακέλου actions είναι οι ακόλουθες:

- Comment
  Η κλάση αυτή αναφέρεται στη σχέση "COMMENTS" της βάσης δεδομένων. Τα κυριότερα πεδία που περιέχει είναι τα εξής:
  - id (τύπου Long): Αποτελεί το id της σχέσης στο γράφο της βάσης δεδομένων.
  - person (τύπου Person): Private πεδίο που χρησιμοποιείται για τον καθορισμό του εναρκτήριου κόμβου της σχέσης, δηλαδή του ατόμου που σχολίασε.
  - asset (τύπου Asset): Private πεδίο που χρησιμοποιείται για τον καθορισμό του καταληκτικού κόμβου της σχέσης, δηλαδή της ταινίας που σχολιάστηκε.
  - intensity (τύπου Double): Είναι η τιμή που αποδίδει κατά απόλυτη τιμή την ένταση της άποψης του χρήστη για την ταινία και λαμβάνει τιμές από 0 ως 1.
  - sentimentCategory (τύπου String): Εκφράζει το συναισθηματικό προσανατολισμό της γνώμης του χρήστη και λαμβάνει τιμές "positive" και "negative".
  - isTrain (τύπου Integer): Δηλώνει αν το συγκεκριμένο σχόλιο εντάσσεται στο training set. Σε περίπτωση ένταξης έχει τιμή 1, διαφορετικά 0.
  - timeOfAction (τύπου Date): Δηλώνει το χρόνο δημιουργίας του σχολίου.

  Επιπρόσθετα, υπάρχουν δυο κατασκευαστές (constructors) για αντικείμενα της κλάσης· ένας χωρίς και ένας με παραμέτρους. Τέλος, διατίθενται και οι μέθοδοι: getPerson(), setPerson(Person person), getAsset(), setAsset(Asset asset), getIntensity(), setIntensity(Double intensity), getSentimentCategory(), setSentimentCategory(String sentimentCategory), getIsTrain(), setIsTrain(Integer isTrain), getTimeOfAction(), setTimeOfAction(Date timeOfAction).

- Consume
  Η κλάση αυτή αναφέρεται στη σχέση "CONSUMES" της βάσης δεδομένων. Τα κυριότερα πεδία που περιέχει είναι τα εξής:
  - id (τύπου Long): Αποτελεί το id της σχέσης στο γράφο της βάσης δεδομένων.
  - person (τύπου Person): Private πεδίο που χρησιμοποιείται για τον καθορισμό του εναρκτήριου κόμβου της σχέσης, δηλαδή του ατόμου που είδε την ταινία.
  - asset (τύπου Asset): Private πεδίο που χρησιμοποιείται για τον καθορισμό του καταληκτικού κόμβου της σχέσης, δηλαδή της ταινίας που προβλήθηκε.
  - timeOfAction (τύπου Date): Δηλώνει το χρόνο προβολής της ταινίας.

Ακόμη, υπάρχουν τρεις κατασκευαστές (constructors) για αντικείμενα της κλάσης· ένας χωρίς παραμέτρους, ένας με κάποιες παραμέτρους, ώστε να δοθεί τιμή σε κάποια βασικά πεδία, και ένας πλήρης constructor που έχει μια παράμετρο για κάθε πεδίο του αντικειμένου. Τέλος, υπάρχουν και οι μέθοδοι: getPerson(), setPerson(Person person), getAsset(), setAsset(Asset asset), getTimeOfAction(), setTimeOfAction(Date timeOfAction).

Οι σημαντικότερες κλάσεις του φακέλου helpers είναι οι ακόλουθες:

- UserCorrelation
  Η κλάση αυτή περιλαμβάνει ένα πεδίο assetId, δυο πεδία intensity (intensity, intensity2) και δυο πεδία sentimentCategory (sentimentCategory, sentimentCategory2). Επιπροσθέτως, περιέχει δυο constructors (έναν χωρίς και έναν με παραμέτρους) και μεθόδους "getters" και "setters" για όλα τα παραπάνω πεδία.

- UserList
  Η κλάση UserList περιλαμβάνει ένα πεδίο personId, ένα πεδίο intensity και ένα sentimentCategory. Ακόμα, περιέχει δυο constructors (έναν χωρίς και έναν με παραμέτρους) και μεθόδους "getters" και "setters" για όλα τα παραπάνω πεδία.

- UserRatings
  Η κλάση αυτή περιέχει ένα πεδίο intensity και ένα sentimentCategory. Ακόμα, περιλαμβάνει δυο constructors (έναν χωρίς και έναν με παραμέτρους) και μεθόδους "getters" και "setters" για τα παραπάνω πεδία.

- UserSimilarity
  Η κλάση αυτή περιέχει ένα πεδίο personId και ένα similarity (τύπου Double). Ακόμα, περιλαμβάνει δυο constructors (έναν χωρίς και έναν με παραμέτρους) και μεθόδους "getters" και "setters" για τα παραπάνω πεδία.

Ο φάκελος neo4j περιέχει κλάσεις repositories με βασικότερες τις εξής:

- AssetRepository
  Η κλάση αυτή περιέχει διάφορες μεθόδους που επιστρέφουν τα ζητούμενα - σχετικά με ταινίες - δεδομένα από τη βάση βάσει ενός query. Οι βασικότερες εξ αυτών είναι οι ακόλουθες:
  - findByAssetId(String assetId): Επιστρέφει αντικείμενο Asset. Η μέθοδος αυτή επιστρέφει ολόκληρο το αντικείμενο Asset στο οποίο αντιστοιχεί το assetId που της δίνεται.
  - all(): Επιστρέφει List<Asset>, δηλαδή μια λίστα αντικειμένων Asset. Η μέθοδος αυτή μας δίνει μια λίστα με όλα τα αντικείμενα Asset του συστήματος.

- PersonRepository
  Η κλάση αυτή περιέχει διάφορες μεθόδους που επιστρέφουν τα ζητούμενα - σχετικά με χρήστες - δεδομένα από τη βάση βάσει ενός query. Οι σημαντικότερες εξ αυτών είναι οι ακόλουθες:
  - findByPersonId(String personId): Επιστρέφει αντικείμενο Person. Η μέθοδος αυτή επιστρέφει ολόκληρο το αντικείμενο Person στο οποίο αντιστοιχεί το personId που της δίνεται.
  - all(): Επιστρέφει List<Person>, δηλαδή μια λίστα αντικειμένων Person. Η μέθοδος αυτή μας δίνει μια λίστα με όλα τα αντικείμενα Person του συστήματος.
  - getMeanRatingByPerson(String personId): Επιστρέφει μια λίστα αντικειμένων UserRatings (List<UserRatings>). Η μέθοδος αυτή δέχεται ένα personId και μας δίνει μια λίστα με όσα από τα rating (intensity και sentimentCategory) του χρήστη ανήκουν στο training set.
  - getUserList(String assetId): Επιστρέφει List<UserList>, δηλαδή λίστα αντικειμένων UserList. Η μέθοδος αυτή δέχεται ένα assetId και επιστρέφει τα personId όσων ατόμων το έχουν κάνει rate μαζί με το rating αυτό (intensity και sentimentCategory), με την προϋπόθεση ότι οι σχέσεις ανήκουν στο training set.
  - getUserCorrelation(String personId, String person2Id): Επιστρέφει List<UserCorrelation>, δηλαδή λίστα αντικειμένων UserCorrelation. Η μέθοδος αυτή δέχεται δυο personId (personId, person2Id) και επιστρέφει τα assetId των ταινιών που έχουν σχολιάσει και οι δυο χρήστες, καθώς και το rating (intensity και sentimentCategory) που έχει βάλει σε καθεμία από αυτές ο κάθε χρήστης, με την προϋπόθεση ότι και οι δυο σχέσεις COMMENTS ανήκουν στο training set.
  - getUserRating(String assetId, String personId): Επιστρέφει ένα αντικείμενο UserRatings. Η μέθοδος αυτή δέχεται ένα assetId και ένα personId και επιστρέφει το rating (intensity και sentimentCategory) του χρήστη αυτού για την επιλεγμένη ταινία, υπό την προϋπόθεση ότι η σχέση αυτή ανήκει στο training set.

Οι κυριότερες κλάσεις του φακέλου rest είναι οι εξής:

- CFController
  Η κλάση αυτή υλοποιεί έναν user-based collaborative filtering αλγόριθμο. Δέχεται ένα personId και ένα assetId και προβλέπει τη βαθμολογία που θα βάλει ο χρήστης που αντιστοιχεί στο personId στην ταινία που αντιστοιχεί στο assetId. Αρχικά, εντοπίζεται η λίστα με όλες τις βαθμολογίες του χρήστη που μας ενδιαφέρει. Κατόπιν, βάσει αυτής της λίστας υπολογίζεται ο μέσος όρος των βαθμολογιών του. Στη συνέχεια, εντοπίζεται η λίστα των ατόμων και των rating που έχουν βάλει αυτά στο asset που μας ενδιαφέρει. Ακολούθως, υπολογίζεται το similarity του χρήστη που μας ενδιαφέρει με καθέναν από τους χρήστες που βρίσκονται στην προαναφερθείσα λίστα, μέσω του Pearson correlation

coefficient. Τέλος, η πρόβλεψη του rating του χρήστη για την ταινία που μας ενδιαφέρει προκύπτει από τον ακόλουθο τύπο [6]:

$$p_{a,i} = \overline{r_a} + \frac{\sum_{u \in K}(r_{u,i} - \overline{r_u})\, w_{a,u}}{\sum_{u \in K} w_{a,u}}$$

όπου $p_{a,i}$ είναι η πρόβλεψη για τη βαθμολογία που θα βάλει ο χρήστης *α* στην ταινία *i*, $\overline{r_a}$ είναι ο μέσος όρος των βαθμολογιών του χρήστη *α* που μας ενδιαφέρει, $w_{a,u}$ το similarity μεταξύ των χρηστών *α* και *u*, όπως προέκυψε από τον συντελεστή συσχέτισης του Pearson , $r_{u,i}$ η βαθμολογία του χρήστη *u* για την ταινία *i* και $\overline{r_u}$ ο μέσος όρος του χρήστη *u* για τις κοινές με τον χρήστη *α* ταινίες και *K* είναι το σύνολο των κοινών μεταξύ των χρηστών *α* και *u* ταινιών.

- <u>KNNController</u>

  Η κλάση αυτή υλοποιεί έναν knn αλγόριθμο. Δέχεται ένα personId και ένα assetId και προβλέπει τη βαθμολογία που θα βάλει ο χρήστης που αντιστοιχεί στο personId στην ταινία που αντιστοιχεί στο assetId. Αρχικά, δημιουργείται μια λίστα όλων των χρηστών του συστήματος και για καθέναν από αυτούς σχηματίζεται μια λίστα με τα κοινά assetId που έχουν με το χρήστη που μας ενδιαφέρει και τις αντίστοιχες βαθμολογίες του χρήστη αυτού και τις δικές τους. Κατόπιν, υπολογίζεται το similarity του χρήστη που μας ενδιαφέρει με καθέναν από τους υπόλοιπους χρήστες βάσει του cosine similarity. Τα similarities αυτά τοποθετούνται μαζί με το personId κάθε χρήστη σε μια λίστα, η οποία ταξινομείται βάσει του similarity. Ακολούθως, από τη λίστα αυτή λαμβάνονται υπόψη για τον υπολογισμό της πρόβλεψης όσοι από τους 10 πιο similar χρήστες με αυτόν που μας ενδιαφέρει έχουν κάνει rate (μέσω σχολίων) την ταινία με το ζητούμενο assetId. Ο τύπος που χρησιμοποιείται για τον υπολογισμό αυτό είναι ο εξής:

$$p_{a,i} = \frac{\sum_{u \in M} r_{u,i}\, w_{a,u}}{j}$$

όπου $p_{a,i}$ είναι η πρόβλεψη για τη βαθμολογία που θα βάλει ο χρήστης *α* στην ταινία *i*, $w_{a,u}$ το similarity μεταξύ των χρηστών *α* και *u*, όπως προέκυψε από το cosine similarity, $r_{u,i}$ η βαθμολογία του χρήστη *u* για την ταινία *i*, *M* το σύνολο των χρηστών που ανήκουν στους 10 πιο similar και ταυτόχρονα έχουν κάνει rate (μέσω σχολίου) την ταινία και *j* το πλήθος των χρηστών που ανήκουν στο σύνολο *L*.

Στο σημείο αυτό, χρειάζεται να διευκρινιστεί ότι όποτε γίνεται λόγος για rating αναφερόμαστε στις βαθμολογίες χρηστών για ταινίες, όπως αυτές έχουν προκύψει από τα σχετικά σχόλιά τους. Ακόμα, το rating προκύπτει από το συνδυασμό των πεδίων intensity και sentimentCategory ως εξής:

– Αν το sentimentCategory έχει τιμή "positive", τότε *rating= (+1) intensity*.

– Αν το sentimentCategory έχει τιμή "negative", τότε *rating= (–1) intensity*.

Κεφάλαιο 4

Τροποποιήσεις Συστήματος και Βελτιστοποίηση Αλγορίθμου Πρόβλεψης Προτιμήσεων Χρήστη

Στο κεφάλαιο αυτό παρουσιάζονται οι αλλαγές και οι βελτιστοποιήσεις στη βάση δεδομένων, στον κώδικα, στον τρόπο και τη συχνότητα συλλογής δεδομένων από τη βάση. Τα αναλυτικά αριθμητικά αποτελέσματα των προαναφερθέντων παρουσιάζονται εκτενώς στο επόμενο κεφάλαιο. Στο σημείο αυτό, κρίνεται σκόπιμο να αναφερθεί ότι η ανάπτυξη του κώδικα πραγματοποιείται με τη βοήθεια του IntelliJ IDEA, ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης Java (Java integrated development environment, IDE).

4.1 Αλλαγές στη Neo4j Βάση Δεδομένων

Η αρχική βάση δεδομένων εμπλουτίστηκε με επιπλέον σχέσεις και ιδιότητες για τη βελτίωση των αποτελεσμάτων και την εξυπηρέτηση των σκοπών της παρούσας εργασίας. Πιο συγκεκριμένα, προστέθηκαν:

- Η σχέση SIMILARITY, η οποία έχει ως εναρκτήριο κόμβο ένα Person και ως καταληκτικό ένα άλλο Person και καταδεικνύει την ομοιότητα μεταξύ τους, όπως αυτή προκύπτει από τις βαθμολογίες που διαμορφώνονται μέσω των σχολίων τους στις κοινές τους ταινίες.

- Η ιδιότητα similarity της σχέσης SIMILARITY, η οποία λαμβάνει ως τιμή (τύπου double) την ομοιότητα μεταξύ δυο χρηστών.

- Η ιδιότητα meanAsstRating κάθε κόμβου Asset, η οποία λαμβάνει ως τιμή (τύπου double) τη μέση βαθμολογία (mean) του εκάστοτε Asset, όπως προκύπτει από τους χρήστες που το σχολίασαν.

- Η ιδιότητα stdevAsstRating κάθε κόμβου Asset, η οποία λαμβάνει ως τιμή (τύπου double) την τυπική απόκλιση (standard deviation) των βαθμολογιών του εκάστοτε Asset. Μικρή τυπική απόκλιση καταδεικνύει ότι τα δεδομένα δεν απέχουν πολύ από τη μέση τιμή, ενώ αντίθετα μεγάλη τυπική απόκλιση καταδεικνύει ότι τα δεδομένα βρίσκονται διάσπαρτα σε μεγαλύτερο εύρος τιμών.

- Η ιδιότητα numOfRatings κάθε κόμβου Asset, η οποία λαμβάνει ως τιμή (τύπου int) το πλήθος των φορών που έχει βαθμολογηθεί (μέσω σχολίων) η εκάστοτε ταινία.

- Η ιδιότητα meanUsrRating κάθε κόμβου Person, η οποία λαμβάνει ως τιμή (τύπου double) το μέσο όρο (mean) των βαθμολογιών του εκάστοτε Person, όπως προκύπτει από τις βαθμολογίες του μέσω σχολίων σε ταινίες.

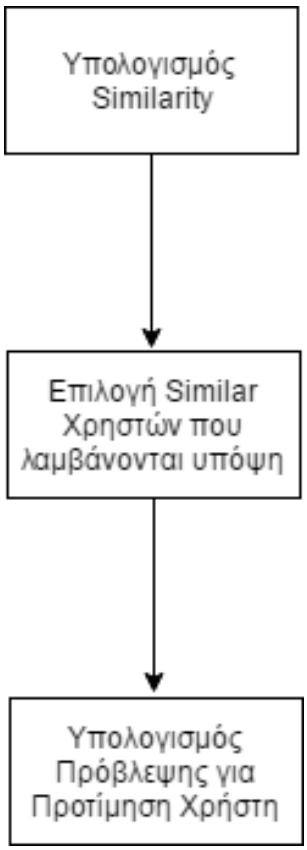
- Η ιδιότητα stdevUsrRating κάθε κόμβου Person, η οποία λαμβάνει ως τιμή (τύπου double) την τυπική απόκλιση (standard deviation) των βαθμολογιών του εκάστοτε Person. Μικρή τυπική απόκλιση καταδεικνύει ότι τα δεδομένα δεν απέχουν πολύ από τη μέση τιμή, δηλαδή ο χρήστης βαθμολογεί με παρόμοιες τιμές όλες τις ταινίες μέχρι στιγμής. Αντίθετα, μεγάλη τυπική απόκλιση καταδεικνύει ότι τα δεδομένα βρίσκονται διάσπαρτα σε μεγαλύτερο εύρος τιμών, συνεπώς ο χρήστης χρησιμοποιεί περισσότερες τιμές της κλίμακας βαθμολογιών.

- Η ιδιότητα numOfRated κάθε κόμβου Person, η οποία λαμβάνει ως τιμή (τύπου int) το πλήθος των βαθμολογημένων (μέσω σχολίων) ταινιών από κάθε χρήστη.

4.2 Βελτιστοποιήσεις και Προσθήκες στο Σύστημα

Σκοπός της εργασίας είναι η ακριβέστερη και ταχύτερη πρόβλεψη των προτιμήσεων ενός χρήστη. Για να επιτευχθεί, λοιπόν, ο προαναφερθείς σκοπός θα πρέπει να πραγματοποιηθούν κάποιες αλλαγές στο σύστημα. Οι αλλαγές αυτές αφορούν στον τρόπο υπολογισμού κάποιων στοιχείων, στις πληροφορίες που αποθηκεύονται στη βάση δεδομένων, στην επιλογή, τον τρόπο ανάκτησης και επεξεργασίας τους από το σύστημα.

Στο πλαίσιο αυτό, πραγματοποιείται τροποποίηση και βελτιστοποίηση του παρεχόμενου k-NN αλγορίθμου. Για καλύτερη εποπτεία παρουσιάζονται στη συνέχεια σχηματικά και κατηγοριοποιημένα τα τρία βασικά τμήματα-λειτουργίες του k-NN αλγορίθμου που διατίθεται. Τα τμήματα αυτά είναι ο υπολογισμός του similarity του χρήστη που μας ενδιαφέρει με κάθε άλλο χρήστη της βάσης δεδομένων, η διαδικασία επιλογής των similar χρηστών που λαμβάνονται κάθε φορά υπόψη από τον αλγόριθμο και ο υπολογισμός της πρόβλεψης για την προτίμηση του αρχικού χρήστη σχετικά με μια συγκεκριμένη ταινία. Σε όλα αυτά τα τμήματα πραγματοποιούνται, λοιπόν, τροποποιήσεις με σκοπό τη βελτίωση καθενός ξεχωριστά και κατ' επέκταση του συνολικού αλγορίθμου.



Σχήμα 4.1: Βασικές λειτουργίες του αρχικού k-NN

Στις ακόλουθες υποενότητες διατυπώνονται και αναλύονται όλες οι τροποποιήσεις που πραγματοποιούνται στα τμήματα αυτά και στο σύνολο του συστήματος.

4.2.1 Βελτιστοποίηση και Αποθήκευση Ομοιότητας στη Βάση Δεδομένων

Μια αλλαγή που στοχεύει στη βελτίωση της ταχύτητας πρόβλεψης των προτιμήσεων του χρήστη είναι ο περιορισμός των υπολογισμών που χρειάζεται να γίνουν on demand κάθε φορά και η πραγματοποίηση πολλών εξ αυτών offline από το σύστημα ανά κάποια διαστήματα, όποτε κρίνεται ότι έχει συγκεντρωθεί όγκος νέων δεδομένων ικανός να επηρεάσει την ποιότητα των αποτελεσμάτων.

Χαρακτηριστικό τέτοιο παράδειγμα αποτελεί ο εκ νέου υπολογισμός των similarities όλων των χρηστών του συστήματος με τον χρήστη Α για τον οποίο είναι επιθυμητή η

πραγματοποίηση πρόβλεψης προτίμησης για κάποια ταινία. Σε περίπτωση, λοιπόν, που πραγματοποιηθεί ξανά πρόβλεψη για την προτίμηση αυτού του χρήστη Α σε σχέση με κάποια άλλη ταινία, απαιτείται και πάλι ο υπολογισμός όλων των similarities. Ακόμα, όμως, και για την πρόβλεψη της προτίμησης κάποιου άλλου χρήστη Β θα υπολογιστεί και πάλι το similarity αυτού με το χρήστη Α, ενώ είχε ήδη υπολογιστεί προηγουμένως.

Θα ήταν ωφέλιμο, επομένως, για το σύστημα να αποθηκεύεται στη βάση δεδομένων το similarity των χρηστών, αντί να υπολογίζεται εκ νέου κάθε φορά. Έτσι, περιορίζεται όχι μόνο το υπολογιστικό κόστος, αλλά και το πλήθος των προσβάσεων στη βάση δεδομένων, καθώς οι πολυάριθμες προσβάσεις για την ανάκτηση από αυτήν όλων των απαραίτητων, για τον υπολογισμό των similarities, δεδομένων αντικαθίστανται από μία και μοναδική πρόσβαση για την συγκέντρωση των επιθυμητών similarities. Δημιουργείται, λοιπόν, μια κλάση SimilarityController για τον υπολογισμό και την αποθήκευση του similarity στη βάση δεδομένων υπό μορφή σχέσης μεταξύ δυο κόμβων Person.

Όπως προαναφέρθηκε, ο υλοποιημένος k-NN χρησιμοποιεί το cosine similarity για τον υπολογισμό της ομοιότητας δυο χρηστών. Ωστόσο, το cosine similarity δε δίνει σε όλες τις περιπτώσεις τιμή που είναι αντιπροσωπευτική της πραγματικής ομοιότητας των δυο χρηστών. Και αυτό, γιατί το cosine similarity εκφράζει την ομοιότητα δυο διανυσμάτων (καθένα αντιπροσωπεύει τις βαθμολογίες των χρηστών στις κοινές τους ταινίες) βάσει της μεταξύ τους γωνίας. Συνεπώς, σε περιπτώσεις που οι χρήστες έχουν μία μόνο κοινή ταινία, τα διανύσματα αυτά εκφυλίζονται σε σημεία, με αποτέλεσμα το similarity να προκύπτει 1 ή -1 ακόμα και στην περίπτωση που έχουν βαθμολογήσει διαφορετικά την ταινία.

Έστω $X \neq 0$ ένα rating του πρώτου χρήστη, $Y \neq 0$ ένα rating του δεύτερου και k το πλήθος των κοινών τους ταινιών. Αντίστοιχο πρόβλημα δημιουργείται όταν ισχύουν ταυτόχρονα οι τρεις ακόλουθες προτάσεις:

1. Ο πρώτος χρήστης έχει βαθμολογήσει με το ίδιο intensity $|X|$ όλες τις κοινές με το δεύτερο χρήστη ταινίες.
2. Αντίστοιχα, ο δεύτερος χρήστης έχει βαθμολογήσει με intensity $|Y|$ όλες τις κοινές με τον πρώτο χρήστη ταινίες.
3. Το γινόμενο XY έχει το ίδιο πρόσημο για όλες τις κοινές ταινίες.

Όταν ισχύουν οι τρεις παραπάνω προτάσεις ταυτόχρονα, τότε το similarity προκύπτει και πάλι 1 ή -1 ακόμα και αν $|X| \neq |Y|$, εφόσον:

$$similarity = cos(\theta) = \frac{\sum_{i=1}^{k} A_i B_i}{\sqrt{\sum_{i=1}^{k} A_i^2}\sqrt{\sum_{i=1}^{k} B_i^2}} = \frac{XY+XY+\cdots+XY}{\sqrt{X^2+X^2+\cdots+X^2}\sqrt{Y^2+Y^2+\cdots+Y^2}} = \frac{k\,(XY)}{\sqrt{kX^2}\sqrt{kY^2}}$$

$$= \frac{k\,(XY)}{\sqrt{k^2X^2Y^2}} = \frac{XY}{|XY|} = \begin{cases} 1, \alpha\nu\ XY > 0 \\ -1, \alpha\nu\ XY < 0 \end{cases}$$

Στις παραπάνω περιπτώσεις, αντί για τη χρήση του cosine similarity χρησιμοποιούνται οι εξής τύποι:

– Αν οι δυο χρήστες έχουν για κάθε ταινία το ίδιο sentimentCategory, τότε γίνεται χρήση του τύπου:

$$similarity = 1 - \frac{\big||X| - |Y|\big|}{2}$$

– Αν οι δυο χρήστες έχουν για κάθε ταινία αντίθετο sentimentCategory, τότε γίνεται χρήση του τύπου:

$$similarity = -1 + \frac{\big||X| - |Y|\big|}{2}$$

Τέλος, στην περίπτωση που οι χρήστες δεν έχουν κοινές μεταξύ τους ταινίες ή ο παρονομαστής του τύπου του cosine similarity προκύψει 0, τότε το similarity θεωρείται 0. Ακολούθως, παρουσιάζεται η κλάση SimilarityController, όπως διαμορφώθηκε μετά την προσθήκη των προαναφερθέντων:

```
package gr.ntua.sam.context.rest;

import gr.ntua.sam.context.neo4j.PersonRepository;
import gr.ntua.sam.context.resources.entities.Person;
import gr.ntua.sam.context.resources.helpers.UserCorrelation;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;
import org.springframework.data.neo4j.template.Neo4jOperations;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.*;
```

```
@RestController
@Configuration
@EnableNeo4jRepositories
public class SimilarityController {

    @Autowired
    private PersonRepository personRepository;

    @Autowired
    Neo4jOperations neo4jTemplate;

    @ApiOperation(value = "calculate and save similarity", tags =
"saveSimilarity")
    @RequestMapping(value = "/saveSimilarity", method = RequestMethod.GET)
    public ResponseEntity calculateSimilarity() {

        List<Person> AllPersonsA = personRepository.all();//list of all users
        double sumOfMultiA = 0.0; //for cosine's similarity numerator
        double sumOfSqrUser1A = 0.0; //for cosine's similarity denominator
        double sumOfSqrUser2A = 0.0; //for cosine's similarity denominator
        double similarityA = 0.0;
        int elmnt = 0;
        int i = 0;
        boolean sameIntensity=true; //the 4 following variables are used in
case the 2 vectors
        int allSameSign=0; //have 0° or 180° angle but different amplitudes
        double intensity1; //(since cosine similarity has a value of 1 even
when the users have rated the assets differently)
        double intensity2;
        Person next = AllPersonsA.get(0);
        UserCorrelation UVector1;
        for (Person APersonA : AllPersonsA) {
            elmnt++;
            for (i = elmnt; i < AllPersonsA.size(); i++) {
                next = AllPersonsA.get(i);
                similarityA = 0.0;
                sumOfMultiA = 0.0;
                sumOfSqrUser1A = 0.0;
                sumOfSqrUser2A = 0.0;
                sameIntensity=true;
                allSameSign=0;
                List<UserCorrelation> UserVectorA =
personRepository.getUserCorrelation(APersonA.getPersonId(),
next.getPersonId());

                if (UserVectorA.size() > 0) {
                    UVector1 = UserVectorA.get(0);
                    intensity1 = UVector1.getIntensity();
                    intensity2 = UVector1.getIntensity2();
                    for (UserCorrelation UVectorA : UserVectorA) {
                        if
(UVectorA.getSentimentCategory().equals(UVectorA.getSentimentCategory2())) {
                            sumOfMultiA += UVectorA.getIntensity() *
UVectorA.getIntensity2();
                            allSameSign++;
                        }
                        else {
                            sumOfMultiA -= UVectorA.getIntensity() *
UVectorA.getIntensity2();
                            allSameSign--;
                        }
                        sumOfSqrUser1A += Math.pow(UVectorA.getIntensity(), 2);
```

```
                        sumOfSqrUser2A += Math.pow(UVectorA.getIntensity2(), 2);
                        if (sameIntensity) {
                            if ((intensity1 == UVectorA.getIntensity()) && (in-
tensity2 == UVectorA.getIntensity2())) {
                                intensity1 = UVectorA.getIntensity();
                                intensity2 = UVectorA.getIntensity2();
                            }
                            else {
                                sameIntensity = false;
                            }
                        }
                    }
                }

                //save similarity!
                //If (Math.sqrt(sumOfSqrUser1A) * Math.sqrt(sumOfSqrUser2A)) =
0 then similarity = 0
                if ((Math.sqrt(sumOfSqrUser1A * sumOfSqrUser2A)) == 0) {
                    similarityA = 0.0; // it also includes the case :
UserVectorA.size()=0
                }
                else if ( (Math.abs(allSameSign) == UserVectorA.size()) &&
(sameIntensity) ){ //if the vectors have zero angle
                    UVector1 = UserVectorA.get(0); // if we reach this point,
the UVector1 already exists, but we have to assign it again because
the 1st assignment was inside an if-statement
                    // intensity and intensity2 of UVector1 are sufficient for
similarity calculation, since user1 rated all assets using the same intensity;
so did user2
                    if (allSameSign>0) { //if users have the same
sentimentCategory
                        similarityA = 1 - (Math.abs(UVector1.getIntensity() -
UVector1.getIntensity2()) / 2);
                    }
                    else { // if users have different sentimentCategory
                        similarityA = -1 + (Math.abs(UVector1.getIntensity() -
UVector1.getIntensity2()) / 2);
                    }
                }
                else { // cosine similarity
                    similarityA = sumOfMultiA / (Math.sqrt(sumOfSqrUser1A *
sumOfSqrUser2A));
                    // to solve a java's precision issue (double values that
are not power of 2 or sums of powers of 2 cannot be represented precisely if
they have many digits)
                    if (similarityA > 1) {
                        similarityA = 1.0;
                    }
                    else if (similarityA < -1){
                        similarityA = -1.0;
                    }
                }
                personRepository.saveSimilarity(APersonA.getPersonId(),
next.getPersonId(), similarityA); //save similarity to database
            }
        }
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Listing 4.1: Κλάση SimilarityController.java

Στην παραπάνω κλάση γίνεται υπολογισμός του similarity μεταξύ όλων των χρηστών με την εφαρμογή όσων προαναφέρθηκαν. Εφόσον υπάρχουν 619 χρήστες, οι σχέσεις που δημιουργούνται στη βάση ανέρχονται σε $\Sigma_{618} = 1 + 2 + \cdots + 618 = \frac{618\,(1+618)}{2} =$ 191.271.

Η μέθοδος getUserCorrelation(String personId, String person2Id) του PersonRepository, η οποία χρησιμοποιείται, έχει τροποποιηθεί, ώστε να επιστρέφει για τους χρήστες των οποίων τα personId δέχεται ως παραμέτρους όλες τις κοινές μεταξύ τους ταινίες και τις βαθμολογίες τους για αυτές ανεξαρτήτως του αν ανήκουν ή όχι στο training set. Η τροποποιημένη μέθοδος getUserCorrelation(String personId, String person2Id) παρουσιάζεται ακολούθως:

```
@Query("MATCH (a:Person)-[r:COMMENTS]->(b:Asset) , (a2:Person)-[r2:COMMENTS]-
>(b:Asset) " +
        "WHERE a.personId={0} AND a2.personId={1}  " +
        "RETURN  b.assetId as assetId, r.intensity as intensity,
r.sentimentCategory as sentimentCategory , r2.intensity as intensity2,
r2.sentimentCategory as sentimentCategory2 ")
List<UserCorrelation> getUserCorrelation (String personId, String person2Id);
```

Listing 4.2: Μέθοδος getUserCorrelation(String personId, String person2Id)

Η μέθοδος saveSimilarity(String personId, String person2Id, Double similarity) του PersonRepository δέχεται δυο personId και το similarity μεταξύ των δυο ατόμων που αντιπροσωπεύουν αυτά και δημιουργεί στη βάση μοναδική σχέση SIMILARITY (μεταξύ αυτών των δυο) έχοντας ως τιμή ιδιότητας το similarity που λαμβάνει ως τρίτη παράμετρο. Σε περίπτωση που η σχέση αυτή έχει δημιουργηθεί ήδη (σε περιπτώσεις που εκτελείται ξανά ο SimilarityController για εκ νέου υπολογισμό των similarities όταν έχουν συγκεντρωθεί αρκετά νέα δεδομένα), η τιμή του similarity ενημερώνεται. Η μέθοδος αυτή παρουσιάζεται παρακάτω:

```
//save(create) or update similarity
@Query("MATCH (a:Person),(b:Person) "+
        "WHERE a.personId={0} AND b.personId={1} "+
        "CREATE UNIQUE (a)-[r:SIMILARITY]->(b) "+
        "SET r.similarity ={2} ")
Void saveSimilarity(String personId, String person2Id, Double similarity);
```

Listing 4.3: Μέθοδος saveSimilarity(String personId, String person2Id, Double similarity)

4.2.2 Αλλαγή τρόπου Υπολογισμού Πρόβλεψης στον k-NN

Πλέον, δεν απαιτείται από την κλάση KNNController ο υπολογισμός του similarity και η κλάση αυτή αντικαθίσταται από την KNNSimilarController που εφαρμόζει τον k-NN αλγόριθμο με τη χρήση των τιμών similarities που λαμβάνει από τη βάση δεδομένων. Η κλάση αυτή εμφανίζεται ακολούθως:

```
package gr.ntua.sam.context.rest;

import gr.ntua.sam.context.neo4j.AssetRepository;
import gr.ntua.sam.context.neo4j.PersonRepository;
import gr.ntua.sam.context.resources.entities.Asset;
import gr.ntua.sam.context.resources.entities.Person;
import gr.ntua.sam.context.resources.helpers.UserRatings;
import gr.ntua.sam.context.resources.helpers.UserSimilarity;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;
import org.springframework.data.neo4j.template.Neo4jOperations;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import java.util.*;

@RestController
@Configuration
@EnableNeo4jRepositories
public class KNNSimilarController {

    @Autowired
    private PersonRepository personRepository;

    @Autowired
    private AssetRepository assetRepository;

    @Autowired
    Neo4jOperations neo4jTemplate;

    @ApiOperation(value = "calculate prediction For KNN2", tags = "KNN")
    @RequestMapping(value = "/knn2/{personId}/{assetId}", method =
RequestMethod.GET)
    public ResponseEntity calculatePredictionKNNa(@PathVariable("personId")
String personId, @PathVariable("assetId")String assetId) {
        //get Person with personId or return error if not valid
        Person person = personRepository.findByPersonId(personId);
        if (person == null) {
            return new ResponseEntity<>("Person id not valid.",
HttpStatus.BAD_REQUEST);
        }
        //get Asset with assetId or return error if not valid
```

```
        Asset currentAsset = assetRepository.findByAssetId(assetId);
        if (currentAsset == null) {
            return new ResponseEntity<>("Asset does not exist",
HttpStatus.BAD_REQUEST);
        }

        Integer k=10; //value of k used
        int j=0;
        double sumOfSimilarities=0.0; //for prediction's denominator
        List<UserSimilarity> USLa =
personRepository.getMostSimilar(personId,k); //get k most similar with personId
users
        System.out.println("Size:....."+USLa.size());
        k=0;
        double meanKNNR=0.0; //to store the prediction
        for(UserSimilarity ULa: USLa){
                k++;
                UserRatings UR = personRepository.getUserRating(assetId,
ULa.getPersonId());
                if(UR!=null){
                    sumOfSimilarities+=Math.abs(ULa.getSimilarity());
                    j++;
                    if("positive".equals(UR.getSentimentCategory())){
                        meanKNNR+= UR.getIntensity()*ULa.getSimilarity();
                     }
                    else if("negative".equals(UR.getSentimentCategory())){
                        meanKNNR-= UR.getIntensity()*ULa.getSimilarity();}
                }
                if(k==10){break;}
        }
        if ((j==0)|| (sumOfSimilarities==0.0)){
            meanKNNR=0.0; //if no one of the most similar users have rated the
asset or the denominator is 0
        }
        else {
            meanKNNR = meanKNNR / sumOfSimilarities; //weighted mean
        }
        System.out.println(k+"...meanKNNR:"+meanKNNR+"--j:"+j);
        return new ResponseEntity<>(meanKNNR, HttpStatus.OK);
    }
}
```

Listing 4.4: Κλάση KNNSimilarController.java

Στην παραπάνω κλάση υπολογίζεται μια πρόβλεψη για τη βαθμολογία που πρόκειται να βάλει ένας χρήστης με personId σε μια ταινία με assetId. Για το χρήστη αυτό επιστρέφεται η λίστα με τους 10 πιο similar με αυτόν χρήστες και το similarity τους. Κατόπιν, για καθέναν από αυτούς επιστρέφεται η βαθμολογία που έχει βάλει στην επιλεγμένη ταινία σε περίπτωση που αυτή υπάρχει. Στη συνέχεια, χρησιμοποιείται ο τύπος του σταθμικού μέσου (weighted mean) για τον υπολογισμό της προβλεπόμενης βαθμολογίας του χρήστη για την ταινία αυτή:

$$p_{a,i} = \frac{\sum_{u \in M} r_{u,i}\, w_{a,u}}{\sum_{u \in M} |w_{a,u}|}$$

όπου $p_{a,i}$ είναι η πρόβλεψη για τη βαθμολογία που θα βάλει ο χρήστης a στην ταινία i, $w_{a,u}$ το similarity μεταξύ των χρηστών a και u, όπως προέκυψε από το SimilarityController.java, $r_{u,i}$ η βαθμολογία του χρήστη u για την ταινία i και M το σύνολο των χρηστών που ανήκουν στους 10 πιο similar με το χρήστη που μας ενδιαφέρει και ταυτόχρονα έχουν κάνει rate (μέσω σχολίου) την ταινία. Στον παρονομαστή χρησιμοποιείται το άθροισμα των απόλυτων τιμών των similarities, καθώς ο σταθμικός μέσος ορίζεται για θετικά βάρη. Στον αριθμητή τα γινόμενα υπολογίζονται κανονικά, θεωρώντας ότι τυχόν αρνητικό πρόσημο στο εκάστοτε γινόμενο οφείλεται στη βαθμολογία και όχι στο similarity.

Σε περίπτωση που ο παρονομαστής στον παραπάνω τύπο προκύψει μηδέν ή κανένας από τους 10 πιο similar χρήστες δεν έχουν βαθμολογήσει την ταινία, τότε η τιμή της πρόβλεψης προκύπτει μηδέν.

Για τη χρήση της κλάσης UserSimilarity από την KNNSimilarController απαιτείται η προσθήκη της ακόλουθης γραμμής και του παρακάτω annotation στον κώδικα της πρώτης:

```
import org.springframework.data.neo4j.annotation.QueryResult;

@QueryResult
```

Listing 4.5: Προσθήκες σε κλάση UserSimilarity.java

Η μέθοδος getMostSimilar(String personId, Integer k) του PersonRepository, η οποία παρουσιάζεται παρακάτω, δέχεται το personId του χρήστη για τον οποίο επιθυμούμε να βρούμε τους πιο similar χρήστες και έναν ακέραιο k που καθορίζει πόσοι από τους πιο similar αυτούς χρήστες θα επιστραφούν. Η μέθοδος αυτή, επομένως, επιστρέφει μια λίστα μήκους k με τα personId και τα similarities των πιο όμοιων χρηστών με τον χρήστη που μας ενδιαφέρει με φθίνουσα σειρά των similarities.

```
//returns k most similar users and their similarity (descending order)
@Query("MATCH (a:Person)-[r:SIMILARITY]-(b:Person) "+
        "WHERE a.personId={0} "+
        "RETURN b.personId AS personId, r.similarity AS similarity  "+
        "ORDER BY similarity DESC LIMIT {1}  ")
List<UserSimilarity> getMostSimilar(String personId, Integer k);
```

Listing 4.6: Μέθοδος getMostSimilar(String personId, Integer k)

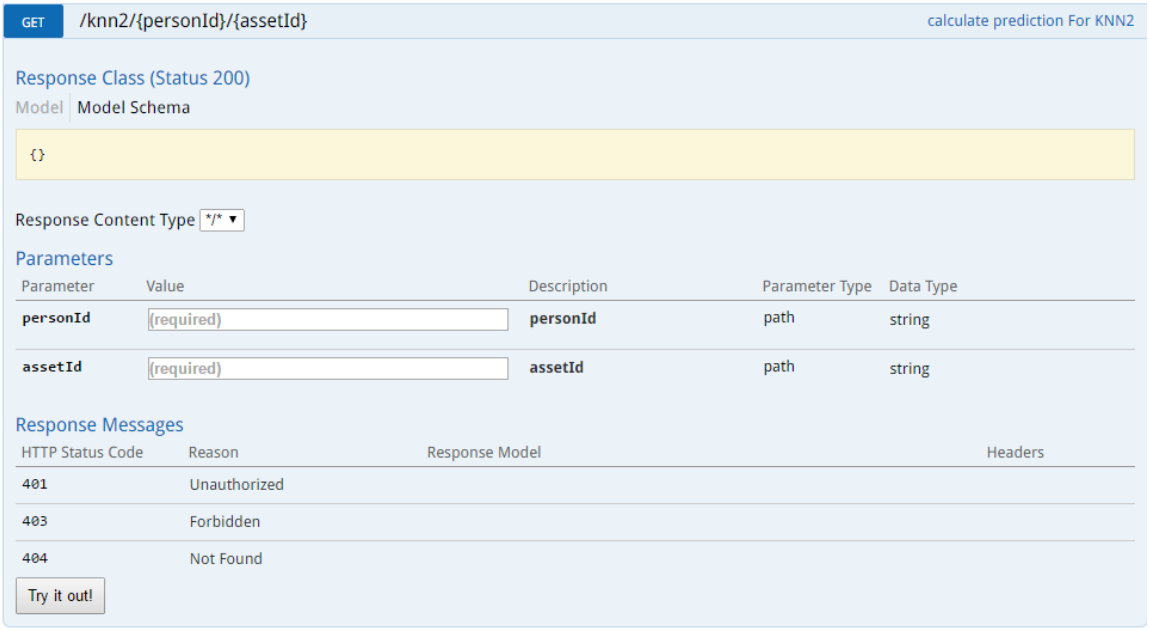
Η μέθοδος getUserRating(String assetId, String personId) του PersonRepository, η οποία χρησιμοποιείται, έχει τροποποιηθεί, ώστε να επιστρέφει τη βαθμολογία του χρήστη που δέχεται ως παράμετρο για την ταινία που, επίσης, δέχεται ως παράμετρο ανεξαρτήτως

του αν η συγκεκριμένη σχέση COMMENTS ανήκει ή όχι στο training set. Η τροποποιημένη μέθοδος getUserRating(String assetId, String personId) παρουσιάζεται ακολούθως:

```
@Query("MATCH (a:Person)-[r:COMMENTS]->(b:Asset) " +
        "WHERE b.assetId={0} AND a.personId={1} " +
        "RETURN  r.intensity as intensity, r.sentimentCategory as
sentimentCategory ")
UserRatings getUserRating (String assetId,String personId);
```

Listing 4.7: Μέθοδος getUserRating(String assetId, String personId)

Στο σημείο αυτό, κρίνεται σκόπιμο να αναφερθεί ο τρόπος που παρέχονται στις περισσότερες κλάσεις Controller τα δεδομένα για τα οποία εξάγονται αποτελέσματα. Μέσω του http://localhost:8080/ πραγματοποιείται η μεταβίβαση των επιθυμητών αυτών δεδομένων, όπως φαίνεται στην ακόλουθη εικόνα:



Σχήμα 4.2: Παροχή δεδομένων στις κλάσεις

Η παραπάνω εικόνα αφορά στον τρόπο εισαγωγής δεδομένων στην κλάση KNNSimilarController, αλλά αντίστοιχα εισάγονται τα δεδομένα που απαιτούν και οι υπόλοιπες κλάσεις Controller. Στο πρώτο κενό πεδίο εισάγεται το personId του χρήστη που μας ενδιαφέρει και στο δεύτερο το assetId της επιθυμητής ταινίας. Τέλος, επιλέγεται το "Try it out!" στο κάτω μέρος της εικόνας και αποστέλλονται τα δεδομένα στον

Controller. Από τον κώδικα της εκάστοτε κλάσης καθορίζονται τα στοιχεία που εμφανίζονται κάθε φορά στην παραπάνω εικόνα.

4.2.3 Αποδοτικότερη Αξιοποίηση της Βάσης Δεδομένων

Μετά τις αλλαγές στον τρόπο υπολογισμού του similarity και της πρόβλεψης του k-NN, αλλά και την αποθήκευση του πρώτου στη βάση δεδομένων, στόχος είναι η περαιτέρω μείωση του χρόνου και του αριθμού των προσβάσεων που απαιτούνται στη βάση δεδομένων και ταυτόχρονα η βελτίωση της ποιότητας των προβλέψεων. Όπως φάνηκε στην κλάση KNNSimilarController γίνονται δυο προσβάσεις στη βάση δεδομένων μέσω των μεθόδων findByPersonId(String personId) και findByAssetId(String assetId) για την εύρεση των αντικειμένων Person και Asset αντίστοιχα, μια πρόσβαση μέσω της μεθόδου getMostSimilar(String personId, Integer k) για τη συγκρότηση μιας λίστας με τους k πιο similar χρήστες με τον personId και για καθέναν από αυτούς πραγματοποιείται άλλη μια πρόσβαση στη βάση δεδομένων για την εύρεση της βαθμολογίας τους για την ταινία που μας ενδιαφέρει. Δηλαδή, πραγματοποιούνται συνολικά 2+1+k προσβάσεις και πιο συγκεκριμένα, για k=10 πραγματοποιούνται 13 προσβάσεις στη βάση δεδομένων κάθε φορά που είναι επιθυμητή η πραγματοποίηση μια πρόβλεψης. Για τις δυο πρώτες προσβάσεις δεν πραγματοποιείται κάποια αλλαγή. Σχετικά με τις υπόλοιπες, εξετάζεται αν θα ήταν αποδοτικότερο χρονικά και καλύτερο ποιοτικά να επιστρέφεται κατευθείαν από τη βάση δεδομένων μια λίστα με τους 10 πιο similar χρήστες με το χρήστη με personId, οι οποίοι, όμως, ταυτόχρονα να έχουν βαθμολογήσει την ταινία. Με τον τρόπο αυτό, οι 1+k προσβάσεις στη βάση θα αντικατασταθούν από μια, πιο πολύπλοκου περιεχομένου, αλλά μοναδική πρόσβαση. Ο αλγόριθμος k-NN, όπως διαμορφώνεται μετά τις προαναφερθείσες τροποποιήσεις, παρουσιάζεται ακολούθως:

```
package gr.ntua.sam.context.rest;

import gr.ntua.sam.context.neo4j.AssetRepository;
import gr.ntua.sam.context.neo4j.PersonRepository;
import gr.ntua.sam.context.resources.entities.Asset;
import gr.ntua.sam.context.resources.entities.Person;
import gr.ntua.sam.context.resources.helpers.UserSimilarityRating;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;
import org.springframework.data.neo4j.template.Neo4jOperations;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import java.util.*;

@RestController
@Configuration
@EnableNeo4jRepositories
public class KNNSimilarRatedController {

    @Autowired
    private PersonRepository personRepository;

    @Autowired
    private AssetRepository assetRepository;

    @Autowired
    Neo4jOperations neo4jTemplate;

    @ApiOperation(value = "calculate prediction For KNN3 ", tags = "KNN")
    @RequestMapping(value = "/knn3/{personId}/{assetId}", method =
RequestMethod.GET)
    public ResponseEntity calculatePredictionKNNb(@PathVariable("personId")
String personId, @PathVariable("assetId")String assetId) {
        //get Person with personId or return error if not valid
        Person person = personRepository.findByPersonId(personId);
        if (person == null) {
            return new ResponseEntity<>("Person id not valid.",
HttpStatus.BAD_REQUEST);
        }
        //get Asset with assetId or return error if not valid
        Asset currentAsset = assetRepository.findByAssetId(assetId);
        if (currentAsset == null) {
            return new ResponseEntity<>("Asset does not exist",
HttpStatus.BAD_REQUEST);
        }

        Integer k=10; //value of k used
        int j=0;
        double sumOfSimilarities=0.0; //for prediction's denominator
        //returns the k most similar users with personId that have rated
assetId
        List<UserSimilarityRating> USLa =
personRepository.getMostSimilarAndRated(personId,assetId,k);
        System.out.println("Size:....."+USLa.size());
        double meanKNNR=0.0; //to store the prediction
        for(UserSimilarityRating ULa: USLa){
            System.out.println(k+"..."+ULa.getPersonId()+"--
getSimilarity"+ULa.getSimilarity());
            sumOfSimilarities+=Math.abs(ULa.getSimilarity());
            j++;

System.out.println(k+"....."+ULa.getSentimentCategory()+"...."+ULa.getIntensity
());
            if("positive".equals(ULa.getSentimentCategory())){
                meanKNNR+= ULa.getIntensity()*ULa.getSimilarity();
            }
            else if("negative".equals(ULa.getSentimentCategory())){
                meanKNNR-= ULa.getIntensity()*ULa.getSimilarity();
            }
        }
        if ((j==0)|| (sumOfSimilarities==0.0)){
            meanKNNR=0.0; //if no one has rated the asset or the denominator is
```

```
0
        }
        else {
            meanKNNR = meanKNNR / sumOfSimilarities; //weighted mean
        }
        System.out.println(k+"...meanKNNR:"+meanKNNR+"--j:"+j);
        return new ResponseEntity<>(meanKNNR, HttpStatus.OK);

    }
}
```

Listing 4.8: Κλάση KNNSimilarRatedController.java

Όπως φαίνεται παραπάνω η κλάση αυτή μοιάζει αρκετά με την KNNSimilarContoller, με τη διαφορά ότι επιστρέφεται λίστα με τους 10 πιο similar χρήστες με αυτόν που μας ενδιαφέρει, οι οποίοι όμως έχουν κάνει ταυτόχρονα και rate την επιθυμητή ταινία. Η λίστα αυτή περιέχει τα personId των χρηστών αυτών, το similarity τους με τον αρχικό χρήστη, και τη βαθμολογία τους για την ταινία στην οποία αναφερόμαστε. Επομένως, στη συνέχεια δεν χρειάζεται να γίνουν επιπλέον queries στη βάση για τη συγκέντρωση των βαθμολογιών τους. Η προσέγγιση αυτή δημιουργεί μια διαφοροποίηση στον αριθμό των similar χρηστών που χρησιμοποιούνται κάθε φορά στον υπολογισμό της πρόβλεψης σε σχέση με προηγουμένως. Δηλαδή, στην περίπτωση αυτή λαμβάνονται υπόψη κάθε φορά ακριβώς 10 similar χρήστες που έχουν βαθμολογήσει την ταινία (λιγότεροι θα ληφθούν υπόψη μόνο σε περιπτώσεις που δεν έχουν βαθμολογήσει αρκετοί χρήστες της βάσης την ταινία). Στην προηγούμενη περίπτωση, χρησιμοποιούνται το πολύ 10 χρήστες για τον υπολογισμό της πρόβλεψης, εφόσον είναι πιθανό να μην έχουν βαθμολογήσει όλοι οι similar χρήστες την ταινία. Το μεγάλο πλεονέκτημα της νέας προσέγγισης που χρησιμοποιείται είναι ότι αποφεύγονται περιπτώσεις στις οποίες ο υπολογισμός της πρόβλεψης αποτυγχάνει γιατί κανένας από τους 10 πιο similar χρήστες που επιστρέφονται δεν έχει κάνει rate την ταινία. Ακόμα, οι προβλέψεις που προκύπτουν είναι πιο ισορροπημένες, καθώς πηγάζουν από βαθμολογίες 10 χρηστών, ενώ στην προηγούμενη περίπτωση υπάρχει η πιθανότητα να χρησιμοποιηθούν εν τέλει ένας ή δυο χρήστες για την πρόβλεψη, γεγονός που σημαίνει ότι το αποτέλεσμα διαμορφώνεται μόνο από αυτούς (biased).

Για να υλοποιηθούν τα παραπάνω ο φάκελος helpers εμπλουτίζεται με την κλάση UserSimilarityRating, η οποία περιλαμβάνει ένα πεδίο personId, ένα πεδίο similarity, ένα intensity και ένα sentimentCategory. Επιπροσθέτως, περιέχει δυο constructors (έναν χωρίς και έναν με παραμέτρους) και μεθόδους "getters" και "setters" για όλα τα παραπάνω πεδία. Η κλάση αυτή παρουσιάζεται ακολούθως:

```java
package gr.ntua.sam.context.resources.helpers;
import org.springframework.data.neo4j.annotation.QueryResult;

@QueryResult
public class UserSimilarityRating {

    String personId;
    Double similarity;
    Double intensity;
    String sentimentCategory;

    public UserSimilarityRating()  {
    }

    public UserSimilarityRating(String personId, Double similarity,Double
intensity, String sentimentCategory) {
        this.personId = personId;
        this.similarity = similarity;
        this.intensity = intensity;
        this.sentimentCategory = sentimentCategory;
    }

    public String getPersonId() {
        return personId;
    }

    public void setPersonId(String personId) {
        this.personId = personId;
    }

    public Double getSimilarity() {
        return similarity;
    }

    public void setSimilarity(Double similarity) {
        this.similarity = similarity;
    }

    public Double getIntensity() {
        return intensity;
    }

    public void setIntensity(Double intensity) {
        this.intensity = intensity;
    }

    public String getSentimentCategory() {
        return sentimentCategory;
    }

    public void setSentimentCategory(String sentimentCategory) {
        this.sentimentCategory = sentimentCategory;
    }

}
```

Listing 4.9: Κλάση UserSimilarityRating.java

Η μέθοδος getMostSimilarAndRated(String personId, String assetId, Integer k) του PersonRepository, η οποία παρουσιάζεται παρακάτω, δέχεται το personId του χρήστη για τον οποίο αναζητούνται οι πιο similar χρήστες, το assetId της ταινίας που επιθυμούμε να έχουν κάνει rate οι similar χρήστες που θα επιστραφούν και έναν ακέραιο k που καθορίζει πόσοι από αυτούς θα επιστραφούν. Η μέθοδος αυτή, επομένως, επιστρέφει μια λίστα μήκους k με τα personId, τα similarities και τις βαθμολογίες για την ταινία με assetId των πιο όμοιων χρηστών με τον χρήστη που μας ενδιαφέρει με φθίνουσα σειρά των similarities.

```
@Query("MATCH (a:Person)-[r:SIMILARITY]-(b:Person)-[r2:COMMENTS]->(c:Asset) "+
        "WHERE a.personId={0} AND c.assetId={1} "+
        "RETURN b.personId AS personId, r.similarity AS similarity,
r2.intensity AS intensity, r2.sentimentCategory AS sentimentCategory  "+
        "ORDER BY similarity DESC LIMIT {2}  ")
List<UserSimilarityRating> getMostSimilarAndRated(String personId, String
assetId, Integer k);
```

Listing 4.10: Μέθοδος getMostSimilarAndRated(String personId, String assetId, Integer k)

4.2.4 Αναζήτηση βέλτιστου k για τον k-NN

Μετά την εφαρμογή των παραπάνω αλλαγών στο σύστημα, τίθεται το ζήτημα της επιλογής του βέλτιστου για τη βάση θετικού ακέραιου k του k-NN αλγορίθμου. Έτσι δημιουργούνται νέες κλάσεις και προστίθενται νέες μέθοδοι για τον υπολογισμό του k (εντός ενός λογικού για τη βάση εύρους τιμών) που οδηγεί σε ελάχιστο σφάλμα στην πρόβλεψη της βαθμολογίας των χρηστών. Αρχικά, εντοπίζεται το k για το οποίο προκύπτει ελάχιστο σφάλμα στα δεδομένα του training set και ελέγχεται αν με τη χρήση αυτού στα δεδομένα του test set προκύπτει αντίστοιχα μικρό σφάλμα, ώστε να έχουν μειωθεί οι πιθανότητες ύπαρξης overfitting. Πιο εκτενής περιγραφή της διαδικασίας που ακολουθείται και των αποτελεσμάτων που προκύπτουν πραγματοποιείται στο επόμενο κεφάλαιο.

Η βέλτιστη τιμή του k που προκύπτει από τις διαδικασίες που ακολουθούνται - καθώς και οι αμέσως επόμενες τιμές του - είναι εξαιρετικά μικρή (k=2) και δεν μπορεί να χρησιμοποιηθεί ως ιδανική. Ένα τόσο μικρό k δεν είναι δυνατό να το γίνει δεκτό ως βέλτιστο, εφόσον λαμβάνει υπόψη ελάχιστους χρήστες και δημιουργεί μια εξαιρετικά biased πρόβλεψη. Το αποτέλεσμα αυτό θεωρείται πιθανό να οφείλεται στο γεγονός ότι η βάση τη δεδομένη χρονική στιγμή δε διαθέτει μεγάλο όγκο δεδομένων και, συνεπώς, μακροπρόθεσμα δε θα είναι αποδοτικό. Επίσης, στο αποτέλεσμα που προέκυψε ενδέχεται να συμβάλλει η ύπαρξη gray και black sheep στο σύστημα. Επομένως, ως βέλτιστη υιοθετείται η αρχική τιμή του k (k=10), δεδομένου ότι και για αυτήν προκύπτει μικρό σφάλμα πρόβλεψης, και, στη συνέχεια, γίνεται προσπάθεια απομόνωσης ή ελάττωσης

της επίδρασης των παραπάνω χρηστών, ώστε να παράγονται όσο το δυνατόν ακριβέστερες προβλέψεις, ακόμα και μακροπρόθεσμα μετά από εμπλουτισμό της βάσης.

4.2.5 Εμπλουτισμός Βάσης Δεδομένων με επιπλέον στοιχεία

Προκειμένου να επιτευχθεί ο παραπάνω στόχος, δηλαδή η μείωση της επίδρασης των gray και black sheep στις προβλέψεις του συστήματος, είναι απαραίτητος ο υπολογισμός και η αποθήκευση στη βάση δεδομένων επιπρόσθετων πληροφοριών σχετικών με τους χρήστες και τις ταινίες. Με τον τρόπο αυτό, είναι εφικτή η ευκολότερη αναγνώριση ιδιαίτερων συμπεριφορών και προτιμήσεων χρηστών. Πιο συγκεκριμένα, υπολογίζεται η μέση (mean) βαθμολογία και η τυπική απόκλιση (standard deviation) των βαθμολογιών κάθε χρήστη και οι αντίστοιχες τιμές για κάθε ταινία. Συνδυάζοντας και συγκρίνοντας κατάλληλα αυτές τις τιμές μπορούν να εντοπιστούν ιδιάζουσες συμπεριφορές χρηστών. Ακόμα, υπολογίζεται ο αριθμός των ταινιών που έχει βαθμολογήσει κάθε χρήστης, αλλά και ο αριθμός των χρηστών που έχουν βαθμολογήσει κάθε ταινία. Και αυτό, για να αποφανθεί αν το πλήθος των βαθμολογιών είναι αρκετό για την εξαγωγή σχετικά ασφαλών συμπερασμάτων. Όλες οι προαναφερθείσες τιμές υπολογίζονται για το σύνολο των δεδομένων της βάσης και αποθηκεύονται σε αυτήν προκειμένου να μην απαιτείται κάθε φορά να υπολογιστούν ξανά. Οι τιμές ανανεώνονται όταν συγκεντρωθεί αρκετός όγκος νέων δεδομένων.

Αρχικά, προστίθενται στην κλάση Person τα πεδία:

```
double meanUsrRating;
double stdevUsrRating;
int numOfRated;
```

Listing 4.11: Προσθήκη πεδίων σε Person.java

Βάσει των πεδίων αυτών ενημερώνονται οι κατασκευαστές που δέχονται παραμέτρους και δημιουργούνται οι αντίστοιχες "getters" και "setters" μέθοδοι, οι οποίες παρουσιάζονται ακολούθως:

```
public double getMeanUsrRating() {return this.meanUsrRating; }

public void setMeanUsrRating(double meanUsrRating) {
this.meanUsrRating=meanUsrRating; }

public double getStdevUsrRating() {return this.stdevUsrRating; }

public void setStdevUsrRating(double stdevUsrRating) {
this.stdevUsrRating=stdevUsrRating; }
```

```
public int getNumOfRated() {return this.numOfRated; }

public void setNumOfRated(int numOfRated) { this.numOfRated=numOfRated; }
```

Listing 4.12: Προσθήκη get και set μεθόδων σε Person.java

Στη συνέχεια, δημιουργείται η κλάση MeanSTDevUsersController προκειμένου να υπολογιστεί η μέση βαθμολογία, η τυπική απόκλιση και ο αριθμός των ταινιών που βαθμολόγησε κάθε χρήστης. Η κλάση MeanSTDevUsersController παρουσιάζεται παρακάτω:

```
package gr.ntua.sam.context.rest;

import gr.ntua.sam.context.neo4j.PersonRepository;
import gr.ntua.sam.context.resources.entities.Person;
import gr.ntua.sam.context.resources.helpers.UserRatings;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;
import org.springframework.data.neo4j.template.Neo4jOperations;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.*;

@RestController
@Configuration
@EnableNeo4jRepositories
public class MeanSTDevUsersController {

    @Autowired
    private PersonRepository personRepository;

    @Autowired
    Neo4jOperations neo4jTemplate;


    @ApiOperation(value = "calculate and save mean and stdev for all users",
tags = "meanStDevUsers")
    @RequestMapping(value = "/meanStDevUsers", method = RequestMethod.GET)
    public ResponseEntity calculateMeanSTDevUsers() {
        List<Person> All_Persons = personRepository.all(); //list with all us-
ers
        double mean=0.0;
        double stdev=0.0;
        int listLength=0;
        double sum=0.0;
        double sumofsquare=0.0;

        for (Person APerson : All_Persons) {
            System.out.println("personId:"+APerson.getPersonId());
            mean=0.0;
```

```
            stdev=0.0;
            sum=0.0;
            listLength=0;
            sumofsquare=0.0;
            List<UserRatings> User_Vector =
personRepository.getUserComments(APerson.getPersonId()); //all intensity kai
sentimentCategory of APerson

            //calculate mean
            listLength=User_Vector.size(); // number of list elements (list
size)
            for (UserRatings UV : User_Vector) {
                if("positive".equals(UV.getSentimentCategory())){
                    sum+= UV.getIntensity();
                }
                else if("negative".equals(UV.getSentimentCategory())){
                    sum-= UV.getIntensity();
                }
            }
            if (listLength==0){
                mean=0.0;
            }
            else{
                mean=sum/listLength;
            }
            APerson.setMeanUsrRating(mean); //save mean to APerson

            //calculate stdev
            sum=0.0;
            for (UserRatings UV : User_Vector) {
                if("positive".equals(UV.getSentimentCategory())){
                    sum=UV.getIntensity()-mean;
                }
                else if("negative".equals(UV.getSentimentCategory())){
                    sum= ((-1)*UV.getIntensity())-mean; //or
sum=UV.getIntensity())+mean; it's the same since it will be raised to 2nd power
                }
                sumofsquare    +=Math.pow(sum, 2);
            }
            if (listLength==0 || listLength==1){
                stdev=0.0;
            }
            else {
                stdev=Math.sqrt((sumofsquare/(listLength-1)));
            }
            APerson.setStdevUsrRating(stdev); //save stdev to APerson
            APerson.setNumOfRated(listLength); //save numOfRated assets to
APerson
           //save all values to database
         personRepository.saveValuesForUser(APerson.getPersonId(),mean,stdev,lis
tLength);
            //print values
            System.out.println("mean:"+mean);
            System.out.println("mean:"+APerson.getMeanUsrRating());
            System.out.println("stdev:"+stdev);
            System.out.println("stdev:"+APerson.getStdevUsrRating());
            System.out.println("numOfRated:"+listLength);
            System.out.println("numOfRated:"+APerson.getNumOfRated());
        }
```

```
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Listing 4.13: Κλάση MeanSTDevUsersController.java

Όπως φαίνεται στον παραπάνω κώδικα, επιστρέφεται μια λίστα με όλους τους χρήστες και κατόπιν για καθέναν από αυτούς συγκροτείται λίστα με όλες του τις βαθμολογίες. Από τα intensity και τα sentimentCategory υπολογίζεται η μέση βαθμολογία τους και η τυπική απόκλιση που προκύπτει. Από το μήκος της λίστας των βαθμολογιών καθορίζεται το πλήθος των ταινιών που βαθμολόγησε ο χρήστης.

Η μέθοδος getUserComments(String personId) του PersonRepository που χρησιμοποιείται δέχεται το personId του χρήστη του οποίου τις βαθμολογίες επιθυμούμε να συγκεντρώσουμε και επιστρέφει μια λίστα με τα intensity και sentimentCategory που υπάρχουν καταχωρημένα για αυτόν στο σύστημα. Η μέθοδος αυτή παρατίθεται ακολούθως:

```
@Query("MATCH (a:Person)-[r:COMMENTS]->(b:Asset) "+
        "WHERE a.personId={0}  "+
        "RETURN r.intensity as intensity, r.sentimentCategory as
sentimentCategory  ")
List<UserRatings> getUserComments(String personId);
```

Listing 4.14: Μέθοδος getUserComments(String personId)

Η μέθοδος saveValuesForUser(String personId, double meanUsrRating, double stdevUsrRating, int numOfRated) του PersonRepository που χρησιμοποιείται δέχεται το personId ενός χρήστη, τη μέση τιμή, την τυπική απόκλιση και το πλήθος των ταινιών που βαθμολόγησε και αποθηκεύει τα τρία τελευταία ως πεδία του Person με personId. Η μέθοδος αυτή χρησιμοποιείται για κάθε χρήστη και παρατίθεται ακολούθως:

```
@Query("MATCH (a:Person) "+
        "WHERE a.personId={0}  "+
        "SET a.meanUsrRating={1},a.stdevUsrRating={2},a.numOfRated={3}  ")
Void saveValuesForUser(String personId,double meanUsrRating, double
stdevUsrRating,int numOfRated);
```

Listing 4.15: Μέθοδος saveValuesForUser(String personId, double meanUsrRating, double stdevUsrRating, int numOfRated)

Αντίστοιχα, προστίθενται στην κλάση Asset τα πεδία:

```
double meanAsstRating;
double stdevAsstRating;
int numOfRatings;
```

Listing 4.16: Προσθήκη πεδίων σε Asset.java

Μετά την προσθήκη των πεδίων αυτών ενημερώνεται ο κατασκευαστής που δέχεται παραμέτρους και δημιουργούνται οι αντίστοιχες "getters" και "setters" μέθοδοι, οι οποίες φαίνονται ακολούθως:

```
public double getMeanAsstRating() {return this.meanAsstRating; }

public void setMeanAsstRating(double meanAsstRating) {
this.meanAsstRating=meanAsstRating; }

public double getStdevAsstRating() {return this.stdevAsstRating; }

public void setStdevAsstRating(double stdevAsstRating) {
this.stdevAsstRating=stdevAsstRating; }

public int getNumOfRatings() {return this.numOfRatings; }

public void setNumOfRatings(int numOfRatings) { this.numOfRatings=numOfRatings;
}
```

Listing 4.17: Προσθήκη get και set μεθόδων σε Asset.java

Εν συνεχεία, συντάσσεται η κλάση MeanSTDevAssetsController προκειμένου να υπολογιστεί η μέση βαθμολογία, η τυπική απόκλιση και το σύνολο των φορών που βαθμολογήθηκε κάθε ταινία. Η κλάση MeanSTDevAssetsController παρατίθεται παρακάτω:

```
package gr.ntua.sam.context.rest;

import gr.ntua.sam.context.neo4j.AssetRepository;
import gr.ntua.sam.context.resources.entities.Asset;
import gr.ntua.sam.context.resources.helpers.UserRatings;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;
import org.springframework.data.neo4j.template.Neo4jOperations;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.*;


@RestController
```

```
@Configuration
@EnableNeo4jRepositories
public class MeanSTDevAssetsController {


    @Autowired
    private AssetRepository assetRepository;

    @Autowired
    Neo4jOperations neo4jTemplate;


    @ApiOperation(value = "calculate and save mean and stdev for all assets",
tags = "meanStDevAssets")
    @RequestMapping(value = "/meanStDevAssets", method = RequestMethod.GET)
    public ResponseEntity calculateMeanSTDevAssets() {
        List<Asset> All_Assets = assetRepository.all(); //list with all assets
        double mean=0.0;
        double stdev=0.0;
        int listLength=0;
        double sum=0.0;
        double sumofsquare=0.0;

        for (Asset AnAsset : All_Assets) {
            System.out.println("assetId:"+AnAsset.getAssetId());
            mean=0.0;
            stdev=0.0;
            sum=0.0;
            listLength=0;
            sumofsquare=0.0;
            List<UserRatings> Asset_Vector =
assetRepository.getAssetComments(AnAsset.getAssetId()); //all intensity kai
sentimentCategory of AnAsset
            //calculate mean
            listLength=Asset_Vector.size(); // number of list elements (list
size)
            for (UserRatings AV : Asset_Vector) {
                if("positive".equals(AV.getSentimentCategory())){
                    sum+= AV.getIntensity();
                }
                else if("negative".equals(AV.getSentimentCategory())){
                    sum-= AV.getIntensity();
                }
            }
            if (listLength==0){
                mean=0.0;
            }
            else{
                mean=sum/listLength;
            }
            AnAsset.setMeanAsstRating(mean); // save mean to AnAsset

            //calculate stdev
            sum=0.0;
            for (UserRatings AV : Asset_Vector) {
                if("positive".equals(AV.getSentimentCategory())){
                    sum=AV.getIntensity()-mean;
                }
                else if("negative".equals(AV.getSentimentCategory())){
                    sum= ((-1)*AV.getIntensity())-mean; //or
sum=UV.getIntensity())+mean; it's the same since it will be raised to 2nd power
```

```
                }
                sumofsquare    +=Math.pow(sum, 2);
            }
            if (listLength==0 || listLength==1){
                stdev=0.0;

            }
            else{
                stdev=Math.sqrt((sumofsquare/(listLength-1)));
            }
            AnAsset.setStdevAsstRating(stdev); //save standard deviation to
AnAsset
            AnAsset.setNumOfRatings(listLength); //save numOfRatings (how many
times the asset has been rated) to AnAsset

assetRepository.saveValuesForAsset(AnAsset.getAssetId(),mean,stdev,listLength);
            System.out.println("mean:"+mean);
            System.out.println("mean:"+AnAsset.getMeanAsstRating());
            System.out.println("stdev:"+stdev);
            System.out.println("stdev:"+AnAsset.getStdevAsstRating());
            System.out.println("numOfRatings:"+listLength);
            System.out.println("numOfRatings:"+AnAsset.getNumOfRatings());
        }
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Listing 4.18: Κλάση MeanSTDevAssetsController.java

Όπως φαίνεται παραπάνω, επιστρέφεται μια λίστα με όλες τις ταινίες και κατόπιν για καθεμία από αυτές συγκροτείται λίστα με όλες της τις βαθμολογίες. Από τα intensity και τα sentimentCategory υπολογίζεται η μέση βαθμολογία και η τυπική απόκλιση που προκύπτει για κάθε ταινία. Το μήκος της λίστας των βαθμολογιών καθορίζει το πλήθος των χρηστών που βαθμολόγησαν καθεμία.

Η μέθοδος getAssetComments(String assetId) του AssetRepository που χρησιμοποιείται δέχεται το assetId της ταινίας για την οποία είναι επιθυμητή η συγκέντρωση των βαθμολογιών της και επιστρέφει μια λίστα με τα intensity και sentimentCategory που υπάρχουν καταχωρημένα για αυτή στο σύστημα. Η μέθοδος getAssetComments(String assetId) παρατίθεται ακολούθως:

```
@Query("MATCH (a:Person)-[r:COMMENTS]->(b:Asset) "+
        "WHERE b.assetId={0}  "+
        "RETURN r.intensity as intensity, r.sentimentCategory as
sentimentCategory  ")
List<UserRatings> getAssetComments(String assetId);
```

Listing 4.19: Μέθοδος getAssetComments(String assetId)

Η μέθοδος saveValuesForAsset(String assetId, double meanAsstRating, double stdevAsstRating, int numOfRatings) του AssetRepository που χρησιμοποιείται δέχεται το assetId μιας ταινίας, τη μέση τιμή, την τυπική απόκλιση και το πλήθος των βαθμολογήσεων που υπάρχουν για αυτή και αποθηκεύει τα τρία τελευταία ως πεδία του Asset με assetId. Η μέθοδος αυτή χρησιμοποιείται για κάθε ταινία και παρατίθεται ακολούθως:

```
@Query("MATCH (a:Asset) "+
        "WHERE a.assetId={0}  "+
        "SET a.meanAsstRating={1},a.stdevAsstRating={2},a.numOfRatings={3}  ")
Void saveValuesForAsset(String assetId,double meanAsstRating, double
stdevAsstRating,int numOfRatings);
```

Listing 4.20: Μέθοδος saveValuesForAsset(String assetId, double meanAsstRating, double stdevAsstRating, int numOfRatings)

4.2.6 Χειρισμός Χρηστών Ιδιάζουσας Συμπεριφοράς

Μετά την προσθήκη των παραπάνω πληροφοριών στο σύστημα, υπάρχει πλέον η δυνατότητα χειρισμού των χρηστών που παρουσιάζουν ιδιάζουσα συμπεριφορά σε βαθμολογικό επίπεδο. Επί παραδείγματι, αν ένας χρήστης βαθμολογεί ακριβώς το ίδιο όλες τις ταινίες, τότε θεωρείται ότι παρουσιάζει ιδιάζουσα συμπεριφορά. Το ίδιο ισχύει για χρήστες που βαθμολογούν πολύ διαφορετικά από τους υπόλοιπους μια ταινία και ταυτόχρονα παρουσιάζουν μεγάλη τυπική απόκλιση. Γενικά, δίνεται ιδιαίτερη έμφαση στην ύπαρξη χρηστών με similarity = 1 με το χρήστη που μας ενδιαφέρει, καθώς η γνώμη τους είναι πανομοιότυπη με εκείνου μέχρι τη δεδομένη στιγμή και το πιο πιθανό είναι να συμφωνήσουν βαθμολογικά για την εξεταζόμενη ταινία.

Αρχικά, διαμορφώνονται δυο διαφορετικές προσεγγίσεις για τη βελτίωση των προβλέψεων του KNNSimilarRatedController.java και, κατόπιν, τροποποιείται περαιτέρω καθεμία από αυτές για καλύτερα αποτελέσματα. Για καλύτερη εποπτεία των προσεγγίσεων αυτών πραγματοποιείται μια περιγραφή των βημάτων που ακολουθούνται για καθεμία. Παρατίθεται, στη συνέχεια, ένα activity diagram και, τέλος, τα σχετικά με την προσέγγιση τμήματα κώδικα.

Για τη διευκόλυνση της περιγραφής, θα χρησιμοποιηθούν τα εξής συμβολικά ονόματα:

– mean_Asset, όταν γίνεται αναφορά στη μέση βαθμολογία μιας ταινίας

– stdev_Asset, όταν γίνεται αναφορά στην τυπική απόκλιση μιας ταινίας

– mean_User, όταν γίνεται αναφορά στη μέση βαθμολογία ενός χρήστη

– stdev_User, όταν γίνεται αναφορά στην τυπική απόκλιση ενός χρήστη

– numOfRatings, όταν γίνεται αναφορά στο πλήθος των βαθμολογιών μιας ταινίας

– numOfRated, όταν γίνεται αναφορά στο πλήθος των βαθμολογημένων από ένα χρήστη ταινιών

– rating, όταν γίνεται αναφορά στη συγκεκριμένη βαθμολογία ενός χρήστη για μια ταινία

KNNSimRatedSheepController4a

Στην πρώτη προσέγγιση που ακολουθείται τα βήματα είναι τα εξής:

– Ως πρώτο βήμα ελέγχεται αν είναι πιθανό να έχει ιδιάζουσα συμπεριφορά ο χρήστης για τον οποίο πραγματοποιείται πρόβλεψη της προτίμησης του. Ένας χρήστης θεωρείται ότι έχει ιδιάζουσα συμπεριφορά αν

$$(mean\_Asset) \notin [(mean\_User) - (stdev\_User), (mean\_User) + (stdev\_User)]$$

$$KAI$$

$$numOfRated \geq 4$$

(≡ Condition_1)

Ο χρήστης είναι δύσκολο να βαθμολογήσει εκτός του εύρους βαθμολογιών που χρησιμοποιεί, εφόσον έχει βαθμολογήσει ένα ικανό πλήθος ταινιών, ώστε να θεωρηθεί ότι έχει διαμορφωθεί ένα προφίλ συμπεριφοράς για αυτόν. Αν, λοιπόν, η μέση βαθμολογία της ταινίας δεν περιέχεται εντός αυτού του εύρους, τότε ο χρήστης προβλέπεται να τη βαθμολογήσει διαφορετικά, ώστε να συμπίπτει με τη διαμορφωμένη συμπεριφορά του.

Στη συνέχεια, εξετάζονται τα ακόλουθα για κάθε χρήστη της λίστας των similar χρηστών που έχουν βαθμολογήσει την ταινία:

– Ελέγχεται η ύπαρξη χρήστη με similarity=1 με τον αρχικό χρήστη που εξετάζεται. Προφανώς, αν υπάρχει τέτοιος χρήστης θα είναι πρώτος στη λίστα. Άρα ελέγχεται αν *για τον πρώτο χρήστη της λίστας ισχύει similarity = 1 (≡ Condition_2).*

– Αν *ο χρήστης που μας ενδιαφέρει δεν έχει ιδιάζουσα συμπεριφορά KAI ο χρήστης της λίστας ο οποίος ελέγχεται έχει similarity < 1 με τον αρχικό KAI βρίσκεται από την 5η και κάτω θέση στη λίστα KAI* $|(mean\_Asset) - (rating)| \geq 0.3$ *KAI numOfRatings > 5 (≡ Condition_3a),* τότε ο χρήστης της λίστας που ελέγχεται θεωρείται ως "ύποπτος" και εξετάζεται περαιτέρω. Δηλαδή, σε περίπτωση που δεν ισχύει το *Condition_1*, ως

"ύποπτος" θεωρείται ένας χρήστης που δεν ταυτίζονται απόλυτα οι προτιμήσεις του με τον αρχικό και δε βρίσκεται στις πρώτες θέσεις στη λίστα των πιο similar με αυτόν χρηστών και, προφανώς, η βαθμολογία του παρουσιάζει αρκετή απόκλιση από τη μέση βαθμολογία της ταινίας, με την προϋπόθεση ότι η ταινία αυτή έχει βαθμολογηθεί κάποιες φορές. Στην περίπτωση, λοιπόν, που ισχύει το *Condition_3a*:

– Υπολογίζεται ο *μέσος όρος των βαθμολογιών* (αρχικών ή αναπροσαρμοσμένων) όσων similar χρηστών λαμβάνονται υπόψη *μέχρι εκείνο το σημείο (≡current_mean).*
– Αν *stdev_User ≥ 0.5 ΚΑΙ* $|(current\_mean) - (rating)| > 0.2$ *(≡ Condition_4a)*, τότε δεν λαμβάνεται υπόψη.
– Αλλιώς αν *stdev_User = 0 ΚΑΙ* $|(current\_mean) - (rating)| > 0.2$ *(≡ Condition_4b)*, τότε δεν λαμβάνεται υπόψη.
– Αλλιώς αν $|(current\_mean) - (rating)| > 0.2$ *ΚΑΙ similarity < 0.99 (≡ Condition_4c)*, τότε το rating αναπροσαρμόζεται ως εξής:

$$rating = \frac{[\,(rating) + (mean\_Asset) + (current\_mean)]}{3}$$

– Σε κάθε άλλη περίπτωση λαμβάνεται κανονικά υπόψη η βαθμολογία του.

– Αλλιώς αν *ο χρήστης που μας ενδιαφέρει έχει ιδιάζουσα συμπεριφορά ΚΑΙ ο χρήστης της λίστας ο οποίος ελέγχεται έχει similarity < 0.99 με τον αρχικό ΚΑΙ βρίσκεται από την 5η και κάτω θέση στη λίστα (≡ Condition_3b)*, τότε εξετάζεται περισσότερο ο χρήστης της λίστας:

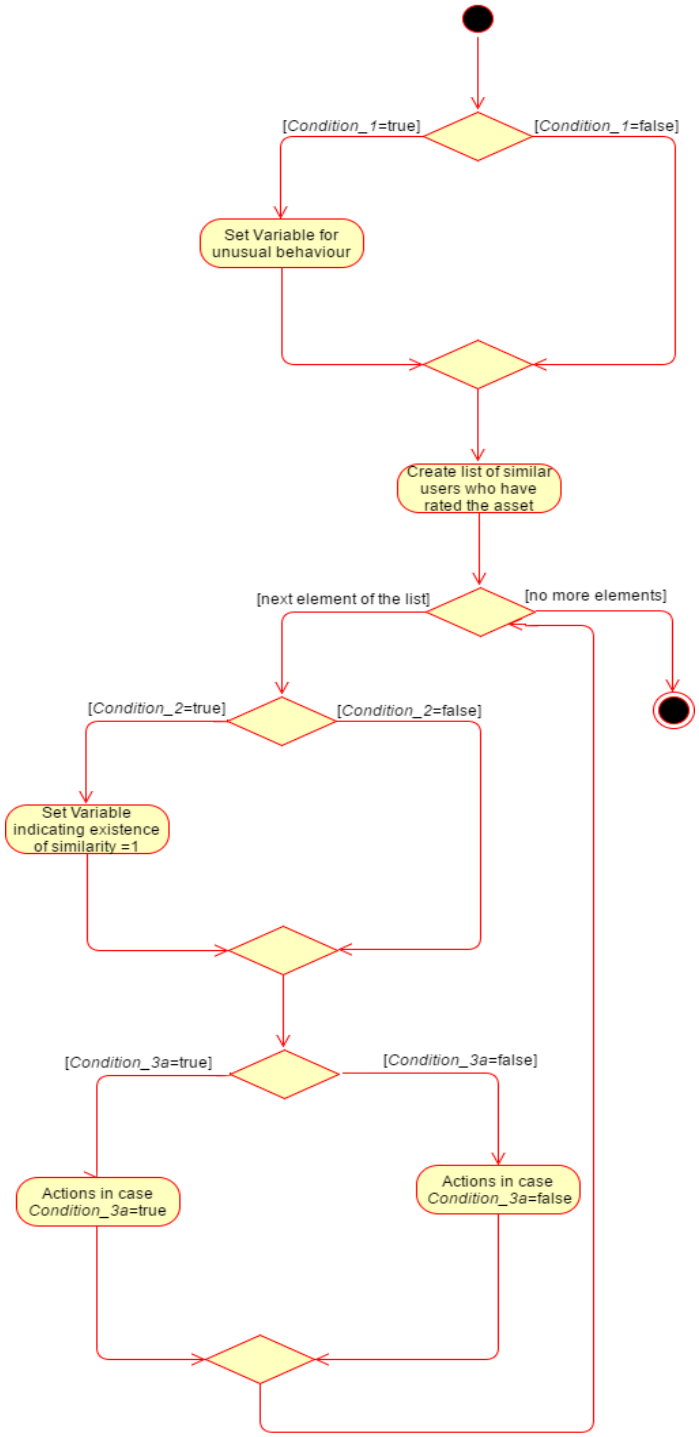
– Υπολογίζεται ο *μέσος όρος των βαθμολογιών* (αρχικών ή αναπροσαρμοσμένων) όσων similar χρηστών λαμβάνονται υπόψη *μέχρι εκείνο το σημείο (≡current_mean).*
– Αν $|(current\_mean) - (rating)| > 0.2$ *ΚΑΙ δεν υπάρχει χρήστης με similarity = 1 στη λίστα (≡ Condition_5a)*, τότε το rating αναπροσαρμόζεται ως εξής:

$$\begin{cases} rating = (current\_mean) - 0.2, αν\ rating < (current\_mean) - 0.2 \\ (\equiv\ Condition\_6) \\ rating = (current\_mean) + 0.2, αν\ rating > (current\_mean)\ + 0.2 \end{cases}$$

– Αλλιώς αν $|(current\_mean) - (rating)| > 0.2$ *ΚΑΙ υπάρχει χρήστης με similarity = 1 στη λίστα (≡ Condition_5b)*, τότε δεν λαμβάνεται υπόψη, αφού διαφοροποιείται αρκετά από την προτίμηση των χρηστών με similarity = 1.
– Σε κάθε άλλη περίπτωση λαμβάνεται κανονικά υπόψη η βαθμολογία του.

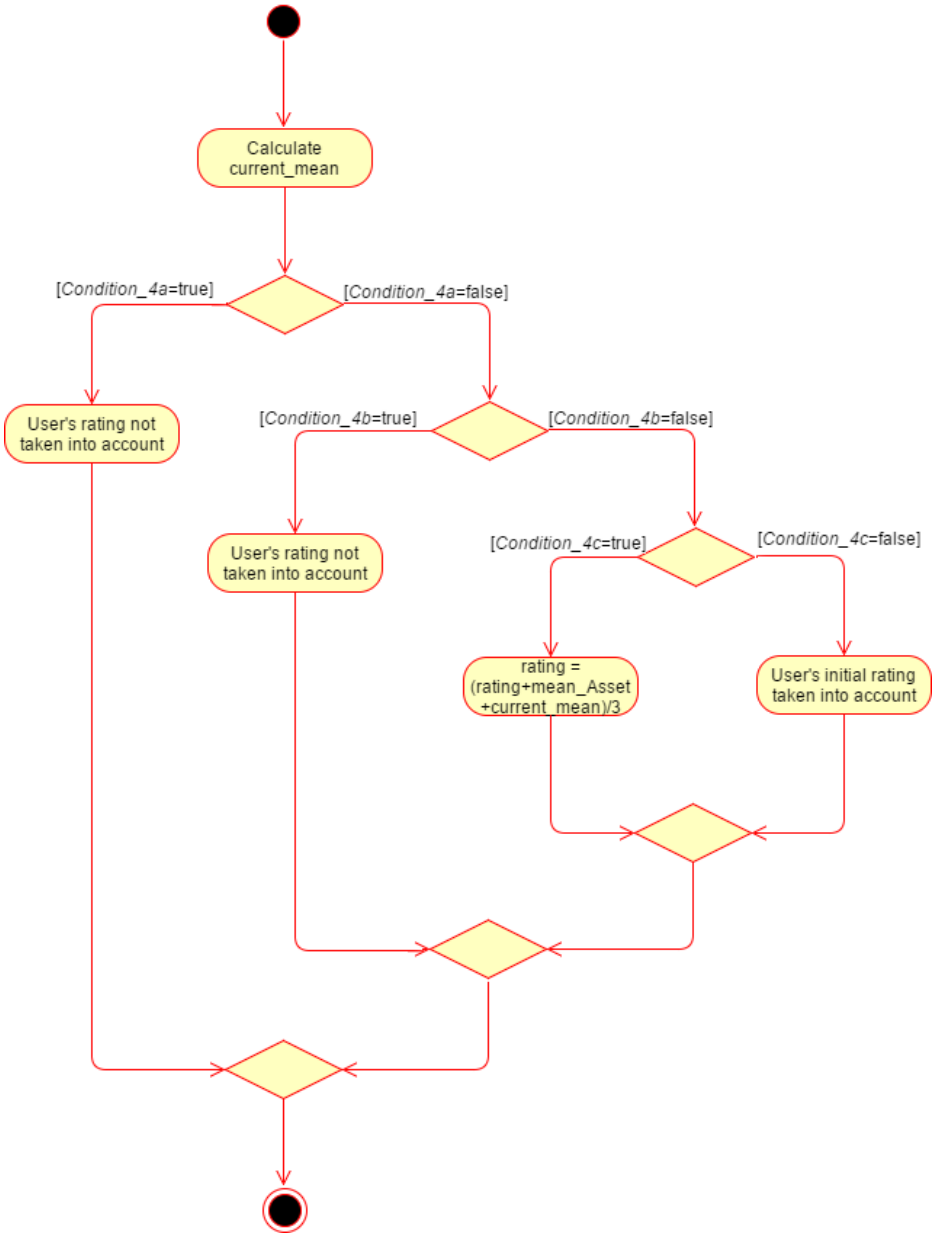
– Σε κάθε άλλη περίπτωση λαμβάνεται κανονικά υπόψη η βαθμολογία του.

Όλα τα παραπάνω συνοψίζονται στο ακόλουθο activity diagram:



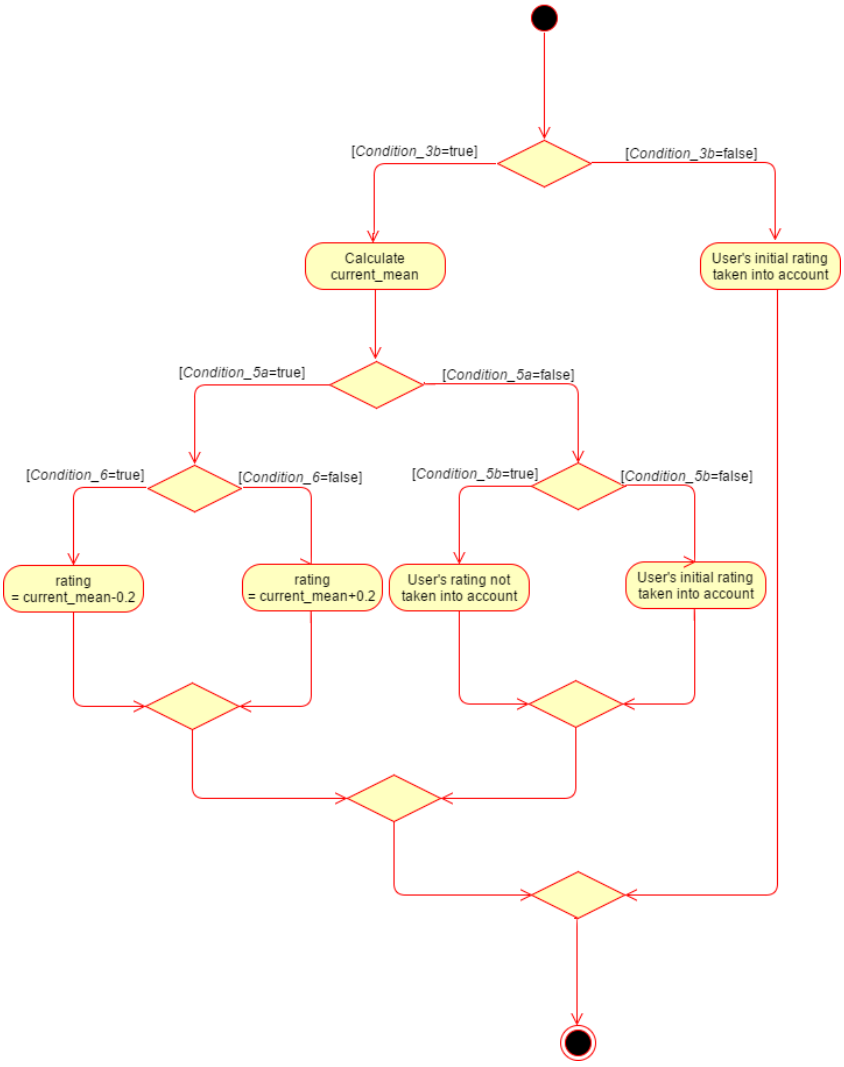
Σχήμα 4.3: Activity Diagram για KNNSimRatedSheepController4a.java

Για λόγους καλύτερης εποπτείας τα "Actions in case Condition_3a = true" και "Actions in case Condition_3a = false" δεν αναλύονται στο παραπάνω σχήμα, αλλά στη συνέχεια:



Σχήμα 4.4: Activity Diagram για "Actions in case Condition_3a = true" της KNNSimRatedSheepController4a.java

Το παραπάνω διάγραμμα παρουσιάζει αναλυτικά τις ενέργειες που περιλαμβάνονται στο "Actions in case Condition_3a = true" Activity Node του Σχήματος 4.2. Αντίστοιχα, για το "Actions in case Condition_3a = false":



Σχήμα 4.5: Activity Diagram για "Actions in case Condition_3a = false" της KNNSimRatedSheepController4a.java

Η προσέγγιση αυτή υλοποιείται στην κλάση KNNSimRatedSheepController4a μέρος της οποίας παρατίθεται ακολούθως:

```
person = personRepository.findByPersonId(personId);
currentAsset = assetRepository.findByAssetId(assetId);
//check for uncommon behaviour
uncommonBehaviour=0;
if ((person.getNumOfRated()>=4) && (
((person.getMeanUsrRating()+person.getStdevUsrRating())<currentAsset.getMeanAss
tRating()) ||  ((person.getMeanUsrRating()-
person.getStdevUsrRating())>currentAsset.getMeanAsstRating()) ) ){
    uncommonBehaviour=1;
}
k=10;
```

```
j=0;
sumOfSimilarities=0.0;
List<PersonSimilarityRating> USLa =
personRepository.getInfoMostSimilarAndRated(personId,assetId,k);
System.out.println("Size:..."+USLa.size());
System.out.println("MeanAssetRating:..."+currentAsset.getMeanAsstRating()+"--
numOfRatings:"+currentAsset.getNumOfRatings());
k=0;
meanKNNR=0.0;
meanList=0.0;
sumList=0.0;
count=0;
similarity1exists=0;
for(PersonSimilarityRating ULa: USLa){
    j++;
    //check for similarity=1
    if ((j == 1) && (ULa.getSimilarity() == 1.0)) {
        similarity1exists = 1;
        System.out.println("THERE IS A SIMILARITY=1");
    }
    //get rating
    if("positive".equals(ULa.getSentimentCategory())){
        rating= ULa.getIntensity();
    }
    else if("negative".equals(ULa.getSentimentCategory())){
        rating= (-1)*ULa.getIntensity();
    }
    System.out.println("user:"+ULa.getPersonId()+"--asset:"+assetId+"--
rating:"+rating+"--STDEV:"+ULa.getStdevUsrRating());
    whiteUser=1;
    notAveraged=1;
    //check if the rating is significantly different from asset's mean rating
provided that the asset has already been rated
    //and the similarity is lower than 1 and the user isn't very high on the
similarity list
    if ((uncommonBehaviour==0) && (ULa.getSimilarity()<1) && (j>=5) &&
(Math.abs(currentAsset.getMeanAsstRating()-rating) >= 0.3) &&
currentAsset.getNumOfRatings() > 5){
        meanList=sumList/count; //calculate mean of ratings up until now
        if ((ULa.getStdevUsrRating() >= 0.5) && (Math.abs(meanList-rating)>0.2)
){
            // if the user has a great standard deviation and the rating is
significantly different from other more similar users' ratings =>
            //we don't take it into account
                whiteUser=0;
        } // if stdev=0 and his rating significantly differs from other similar
users' ratings we don't take him into account
        else if ((ULa.getStdevUsrRating() == 0.0) && (Math.abs(meanList-
rating)>0.2)){
            whiteUser=0;
        } // if his rating significantly differs from other similar users'
ratings and his similarity is lower than 0.99 his rating is averaged
        else if ((Math.abs(meanList-rating)>0.2) &&
(ULa.getSimilarity()<0.99)){
            notAveraged=0;
            rating=(rating+currentAsset.getMeanAsstRating()+meanList)/3;
        }
    }
    else if ((uncommonBehaviour==1) && (ULa.getSimilarity()<0.99) && (j>=5)){
        meanList=sumList/count; //calculate mean of ratings up until now
        if ((Math.abs(meanList-rating)>0.2)  && (similarity1exists==0)){
```

```
                notAveraged=0;
                if (rating<meanList-0.2){
                    rating=meanList-0.2;
                }
                else if (rating>meanList+0.2){
                    rating=meanList+0.2;
                }
            } else if ((Math.abs(meanList-rating)>0.2) && (similarity1exists==1)){
                whiteUser=0; // we don't take him into account
            }
        }

        if  (notAveraged==0) { // if the user's rating is averaged
            k--;
        }
        if (whiteUser==0) { //if the user shouldn't be taken into account
            k--;
            count--;
            sumList -= rating;
        }

        k++;
        count++;
        sumList+=rating;
        System.out.println("White User:..."+whiteUser);
        System.out.println("Not Averaged:..."+notAveraged);
        sumOfSimilarities+=whiteUser*Math.abs(ULa.getSimilarity());
        meanKNNR+=whiteUser*rating*ULa.getSimilarity();
    }
    if ((k==0)|| (sumOfSimilarities==0.0)){
        meanKNNR=0.0;
    }
    else {
        meanKNNR = meanKNNR / sumOfSimilarities;
    }
```

Listing 4.21: Κλάση KNNSimRatedSheepController4a.java

Για να υλοποιηθούν τα παραπάνω, απαιτείται η προσθήκη στο φάκελο helpers της κλάσης PersonSimilarityRating, η οποία περιλαμβάνει ένα πεδίο personId, ένα meanUsrRating, ένα stdevUsrRating, ένα numOfRated, ένα similarity, ένα intensity και ένα sentimentCategory. Επιπροσθέτως, περιέχει δυο constructors (έναν χωρίς και έναν με παραμέτρους) και μεθόδους "getters" και "setters" για όλα τα παραπάνω πεδία. Η κλάση αυτή παρουσιάζεται ακολούθως:

```
package gr.ntua.sam.context.resources.helpers;
import org.springframework.data.neo4j.annotation.QueryResult;

@QueryResult
public class PersonSimilarityRating {
    String personId;
    Double meanUsrRating;
    Double stdevUsrRating;
    Integer numOfRated;
    Double similarity;
```

```
    Double intensity;
    String sentimentCategory;
    public PersonSimilarityRating()  {
    }

    public PersonSimilarityRating(String personId, Double similarity,Double
intensity, String sentimentCategory, Double meanUsrRating,Double
stdevUsrRating,Integer numOfRated) {
        this.personId = personId;
        this.meanUsrRating=meanUsrRating;
        this.stdevUsrRating=stdevUsrRating;
        this.numOfRated=numOfRated;
        this.similarity = similarity;
        this.intensity = intensity;
        this.sentimentCategory = sentimentCategory;
    }

    public String getPersonId() {
        return personId;
    }

    public void setPersonId(String personId) {
        this.personId = personId;
    }

    public Double getMeanUsrRating() {
        return meanUsrRating;
    }

    public void setMeanUsrRating(Double meanUsrRating) {
        this.meanUsrRating = meanUsrRating;
    }

    public Double getStdevUsrRating() {
        return stdevUsrRating;
    }

    public void setStdevUsrRating(Double stdevUsrRating) {
        this.stdevUsrRating = stdevUsrRating;
    }

    public Integer getNumOfRated() {
        return numOfRated;
    }

    public void setNumOfRated(Integer numOfRated) {
        this.numOfRated = numOfRated;
    }

    public Double getSimilarity() {
        return similarity;
    }

    public void setSimilarity(Double similarity) {
```

```
        this.similarity = similarity;
    }

    public Double getIntensity() {
        return intensity;
    }

    public void setIntensity(Double intensity) {
        this.intensity = intensity;
    }

    public String getSentimentCategory() {
        return sentimentCategory;
    }

    public void setSentimentCategory(String sentimentCategory) {
        this.sentimentCategory = sentimentCategory;
    }
}
```

Listing 4.22: Κλάση PersonSimilarityRating.java

Η μέθοδος getInfoMostSimilarAndRated(String personId, String assetId, Integer k) του PersonRepository που χρησιμοποιείται δέχεται το personId του χρήστη για τον οποίο πραγματοποιείται πρόβλεψη συμπεριφοράς, το assetId της ταινίας που μας ενδιαφέρει και έναν ακέραιο k, ο οποίος συμβολίζει το μήκος της λίστας που επιστρέφεται. Η λίστα αυτή περιέχει τα personId, meanUsrRating, stdevUsrRating, numOfRated των k πιο similar χρηστών με το χρήστη που μας ενδιαφέρει, με την προϋπόθεση οι χρήστες αυτοί να έχουν βαθμολογήσει την ταινία. Περιέχει, ακόμα, το similarity τους με τον αρχικό χρήστη, καθώς και το intensity και sentimentCategory τους για την ταινία. Η μέθοδος getInfoMostSimilarAndRated(String personId, String assetId, Integer k) παρατίθεται ακολούθως:

```
@Query("MATCH (a:Person)-[r:SIMILARITY]-(b:Person)-[r2:COMMENTS]->(c:Asset) "+
        "WHERE a.personId={0} AND c.assetId={1} "+
        "RETURN b.personId AS personId, b.meanUsrRating AS meanUsrRating,
b.stdevUsrRating AS stdevUsrRating, b.numOfRated AS numOfRated, r.similarity AS
similarity, r2.intensity AS intensity, r2.sentimentCategory AS
sentimentCategory  "+
        "ORDER BY similarity DESC LIMIT {2}  ")
List<PersonSimilarityRating> getInfoMostSimilarAndRated(String personId, String
assetId, Integer k);
```

Listing 4.23: Μέθοδος getInfoMostSimilarAndRated(String personId, String assetId, Integer k)

KNNSimRatedSheepController4b

Στη δεύτερη προσέγγιση που ακολουθείται τα βήματα είναι τα εξής:

– Το πρώτο βήμα είναι ίδιο με το αντίστοιχο της προηγούμενης προσέγγισης. Ελέγχεται, δηλαδή, αν είναι πιθανό να έχει ιδιάζουσα συμπεριφορά ο χρήστης για τον οποίο πραγματοποιείται πρόβλεψη της προτίμησης του. Ένας χρήστης θεωρείται ότι έχει ιδιάζουσα συμπεριφορά αν

$$(mean\_Asset) \notin [(mean\_User) - (stdev\_User), (mean\_User) + (stdev\_User)]$$

$$KAI$$

$$numOfRated \geq 4$$

(≡ Condition_1)

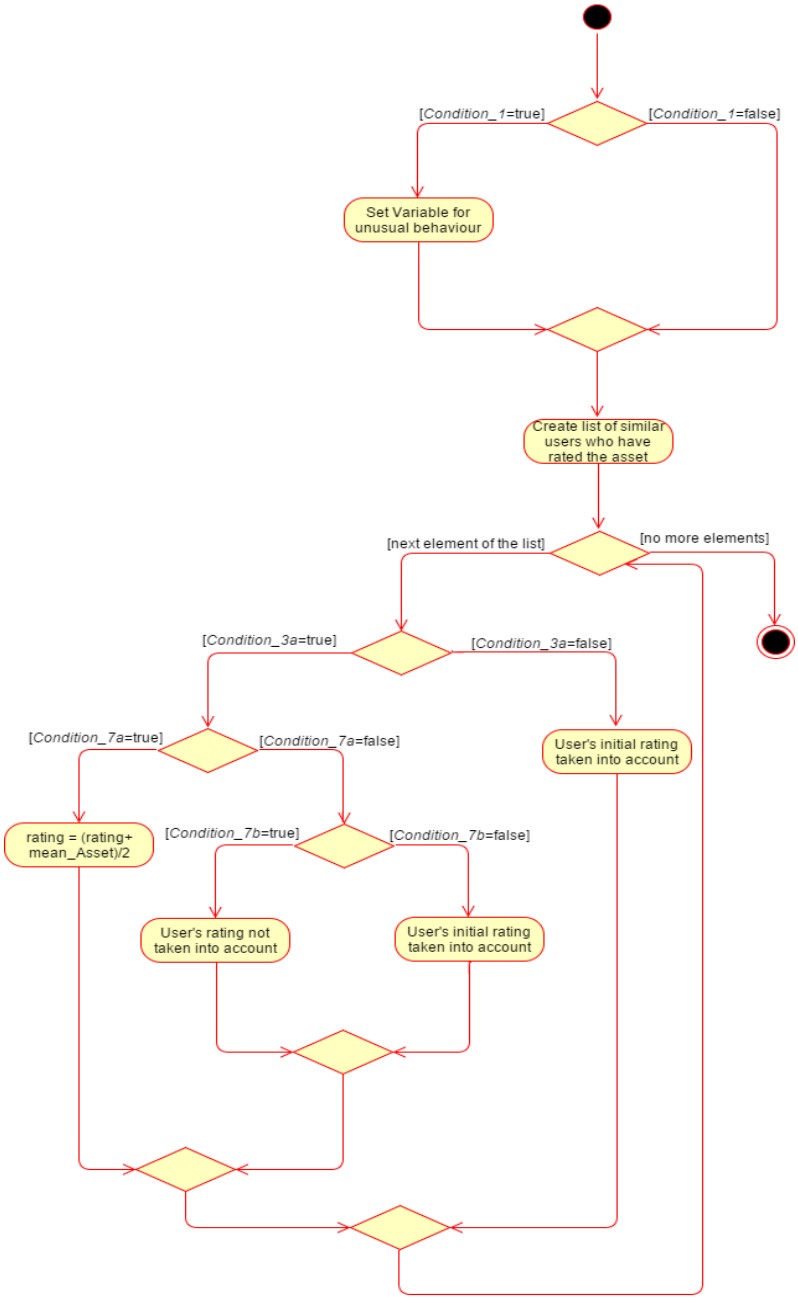
Στη συνέχεια, εξετάζονται τα ακόλουθα για κάθε χρήστη της λίστας των similar χρηστών που έχουν βαθμολογήσει την ταινία:

– Αν *ο χρήστης που μας ενδιαφέρει δεν έχει ιδιάζουσα συμπεριφορά KAI ο χρήστης της λίστας ο οποίος ελέγχεται έχει similarity < 1 με τον αρχικό KAI βρίσκεται από την 5η και κάτω θέση στη λίστα KAI* $|(mean\_Asset) - (rating)| \geq 0.3$ *KAI numOfRatings > 5 (≡ Condition_3a)*, τότε ο χρήστης της λίστας που ελέγχεται θεωρείται ως "ύποπτος" και εξετάζεται περαιτέρω. Δηλαδή, και εδώ εφαρμόζεται συνθήκη της προηγούμενης προσέγγισης. Στην περίπτωση, λοιπόν, που ισχύει το *Condition_3a*:

> – Αν *stdev_User ≥ 0.5 (≡ Condition_7a)*, τότε το rating αναπροσαρμόζεται ως εξής:
>
> $$rating = \frac{[(rating)+(mean\_Asset)]}{2}$$
>
> Δηλαδή, αν παρατηρείται μεγάλη τυπική απόκλιση είναι προτιμότερο να αντικατασταθεί το rating με το μέσο όρο αυτού και της μέσης βαθμολογίας της ταινίας.
> – Αλλιώς αν *stdev_User = 0 KAI numOfRated ≥ 4 (≡ Condition_7b)*, τότε δεν λαμβάνεται υπόψη, εφόσον βαθμολογεί πάντα ίδια όλες τις ταινίες και έχει βαθμολογήσει έναν ικανό αριθμό ταινιών, ώστε να μπορούν να εξαχθούν συμπεράσματα για τη συμπεριφορά του.
> – Σε κάθε άλλη περίπτωση λαμβάνεται κανονικά υπόψη η βαθμολογία του.

– Σε κάθε άλλη περίπτωση λαμβάνεται κανονικά υπόψη η βαθμολογία του.

Όλα τα παραπάνω συνοψίζονται στο ακόλουθο activity diagram:



Σχήμα 4.6: Activity Diagram για KNNSimRatedSheepController4b.java

Η προσέγγιση αυτή υλοποιείται στην κλάση KNNSimRatedSheepController4b μέρος της οποίας παρατίθεται ακολούθως:

```
person = personRepository.findByPersonId(personId);
currentAsset = assetRepository.findByAssetId(assetId);
uncommonBehaviour=0;
if ((person.getNumOfRated()>=4) && (
((person.getMeanUsrRating()+person.getStdevUsrRating())<currentAsset.getMeanAss
tRating()) ||  ((person.getMeanUsrRating()-
person.getStdevUsrRating())>currentAsset.getMeanAsstRating()) ) ){
    uncommonBehaviour=1;
}
k=10;
j=0;
sumOfSimilarities=0.0;
List<PersonSimilarityRating> USLa =
personRepository.getInfoMostSimilarAndRated(personId,assetId,k);
System.out.println("Size:"+USLa.size());
System.out.println("MeanAssetRating:"+currentAsset.getMeanAsstRating()+"--
numOfRatings:"+currentAsset.getNumOfRatings());
k=0;
meanKNNR=0.0;
for(PersonSimilarityRating ULa: USLa){
    j++;
    //get rating
    if("positive".equals(ULa.getSentimentCategory())){
        rating= ULa.getIntensity();
    }
    else if("negative".equals(ULa.getSentimentCategory())){
        rating= (-1)*ULa.getIntensity();
    }
    System.out.println("user:"+ULa.getPersonId()+"--asset:"+assetId+"--
rating:"+rating+"--STDEV:"+ULa.getStdevUsrRating());
    whiteUser=1;
    notAveraged=1;
    if ((uncommonBehaviour==0) && (ULa.getSimilarity() < 1) && (j>=5) &&
(Math.abs(currentAsset.getMeanAsstRating()-rating) >= 0.3) &&
currentAsset.getNumOfRatings() > 5){
        //if the user has a great standard deviation his rating is averaged
        if ((ULa.getStdevUsrRating() >= 0.5) ){
            notAveraged=0;
            rating=(rating+currentAsset.getMeanAsstRating())/2;
        }//if the user always votes the same we don't take him into account
        else if ((ULa.getStdevUsrRating() == 0.0) && (ULa.getNumOfRated()>=4)){
            whiteUser=0;
        }
    }
    if ((whiteUser==0)|| (notAveraged==0)){ // if the user shouldn't be taken
into account
        k--;
    }
    k++;
    System.out.println("White User:..."+whiteUser);
    System.out.println("Not Averaged:..."+notAveraged);
    sumOfSimilarities+=whiteUser*Math.abs(ULa.getSimilarity());
    meanKNNR+=whiteUser*rating*ULa.getSimilarity();
}
if ((k==0)|| (sumOfSimilarities==0.0)){
    meanKNNR=0.0;
}
else {
    meanKNNR = meanKNNR / sumOfSimilarities;
}
```

Listing 4.24: Κλάση KNNSimRatedSheepController4b.java

KNNSimRatedSheepController5a & KNNSimRatedSheepController5b

Ακολούθως, παρουσιάζονται οι τροποποιήσεις που πραγματοποιήθηκαν στις προαναφερθείσες προσεγγίσεις. Συγκεκριμένα, η βασική προσθήκη και στις δυο αυτές προσεγγίσεις είναι η ειδική αντιμετώπιση των περιπτώσεων στις οποίες υπάρχουν χρήστες με similarity = 1 στη λίστα που συγκροτείται. Επειδή θεωρείται πως η ύπαρξη τέτοιων χρηστών είναι ιδιαίτερα σημαντική για την παραγωγή μιας αξιόπιστης πρόβλεψης, αναπτύσσεται διαφορετικός τρόπος χειρισμού των περιπτώσεων αυτών.

Στην περίπτωση, λοιπόν, που στη λίστα των πιο similar χρηστών που έχουν βαθμολογήσει την ταινία *παρατηρηθεί έστω κι ένα similarity = 1 - προφανώς αν υπάρχει θα βρίσκεται στην πρώτη θέση της λίστας - (≡ Condition_2)*, ακολουθείται η εξής διαδικασία:

– Υπολογίζεται *ο μέσος όρος (≡ mean_similar) και η τυπική απόκλιση (≡ stdev_similar) των βαθμολογιών των χρηστών που έχουν similarity ≥ 0.99* (για να υπάρχει η δυνατότητα κάποιας διαφοροποίησης) και δημιουργείται το διάστημα

$$[(mean\_similar) - (stdev\_similar), (mean\_similar) + (stdev\_similar)]$$

– Οι βαθμολογίες των χρηστών με similarity ≥ 0.99 λαμβάνονται υπόψη στον υπολογισμό της πρόβλεψης χωρίς κάποια τροποποίηση.

– Για κάθε άλλη βαθμολογία της λίστας αν $rating < (mean\_similar) - (stdev\_similar)$ *(≡ Condition_8α)* τότε το rating αναπροσαρμόζεται ως εξής:

$$rating = (mean\_similar) - (stdev\_similar)$$

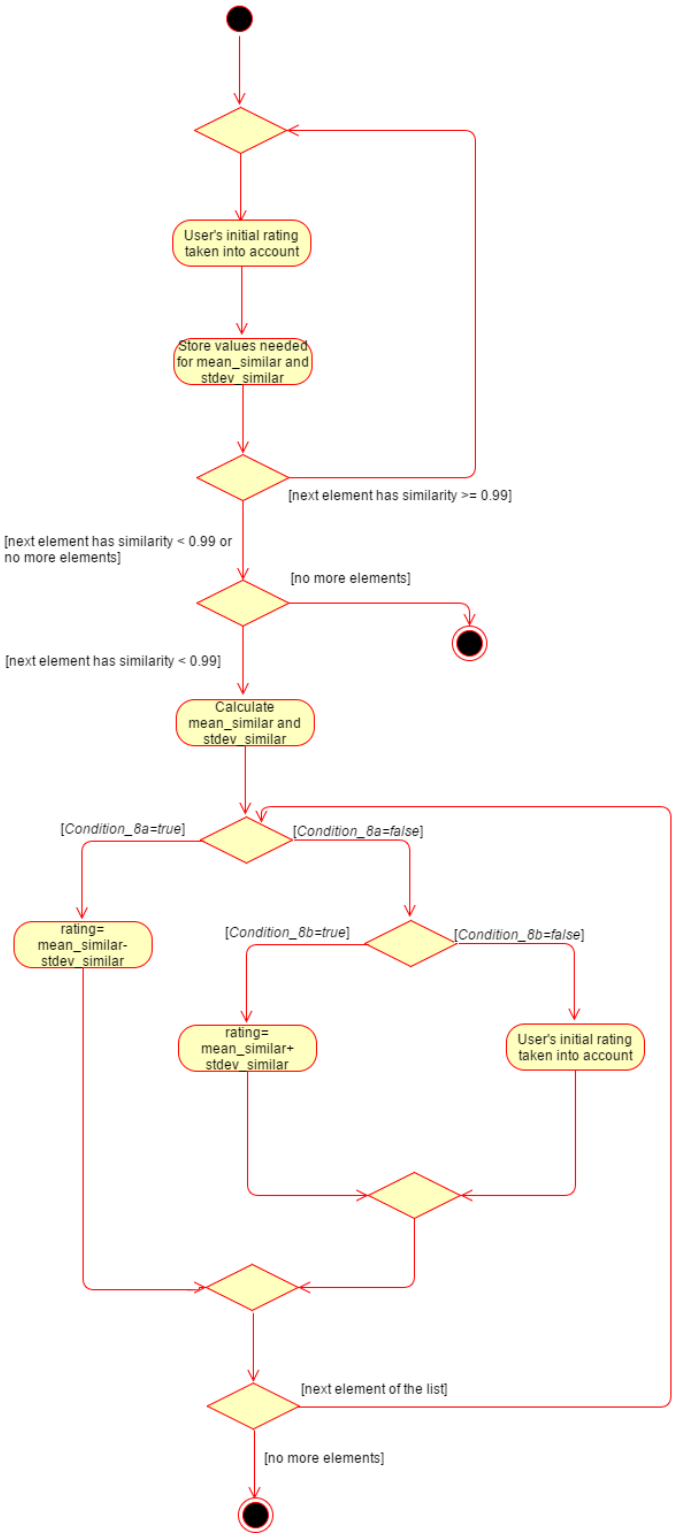
– Αλλιώς αν $rating > (mean\_similar) + (stdev\_similar)$ *(≡ Condition_8b)* τότε το rating αναπροσαρμόζεται ως εξής:

$$rating = (mean\_similar) + (stdev\_similar)$$

– Σε κάθε άλλη περίπτωση ($rating \in [(mean\_similar) - (stdev\_similar), (mean\_similar) + (stdev\_similar)]$) λαμβάνεται κανονικά υπόψη η βαθμολογία του.

Αν δεν υπάρχουν χρήστες με similarity = 1 στη λίστα, τότε η KNNSimRatedSheepController4a.java που υλοποιεί την πρώτη προσέγγιση ακολουθεί την αρχική διαδικασία που ορίστηκε, εκτός του *Condition_5b* που δεν έχει πλέον νόημα, εφόσον οι περιπτώσεις με similarity = 1 αντιμετωπίζονται διαφορετικά. Έτσι δημιουργείται η κλάση KNNSimRatedSheepController5a. Αντίστοιχα, αν δεν υφίστανται χρήστες με similarity = 1 στη λίστα, η KNNSimRatedSheepController4b.java που υλοποιεί τη δεύτερη προσέγγιση ακολουθεί τη δική της αρχική διαδικασία. Παράγεται, λοιπόν, η κλάση KNNSimRatedSheepController5b.

Οι προαναφερθείσες αλλαγές για την περίπτωση που παρατηρείται χρήστης με similarity = 1 στη λίστα απεικονίζονται στο ακόλουθο activity diagram:



Σχήμα 4.7: Activity Diagram για χειρισμό περιπτώσεων με similarity = 1

Σε μορφή κώδικα οι παραπάνω τροποποιήσεις υλοποιούνται ως εξής:

```
person = personRepository.findByPersonId(personId);
currentAsset = assetRepository.findByAssetId(assetId);
uncommonBehaviour=0;
if ((person.getNumOfRated()>=4) && (
((person.getMeanUsrRating()+person.getStdevUsrRating())<currentAsset.getMeanAss
tRating()) ||  ((person.getMeanUsrRating()-
person.getStdevUsrRating())>currentAsset.getMeanAsstRating()) ) ){
    uncommonBehaviour=1;
}
k=10;
j=0;
sumOfSimilarities=0.0;
List<PersonSimilarityRating> USLa =
personRepository.getInfoMostSimilarAndRated(personId,assetId,k);
System.out.println("Size:..."+USLa.size());
System.out.println("MeanAssetRating:..."+currentAsset.getMeanAsstRating()+"--
numOfRatings:"+currentAsset.getNumOfRatings());
k=0;
meanKNNR=0.0;
similarity1exists=0;
meanSim=0.0;
stdevSim=0.0;
diff=0.0;
count=0;
flagContinue=1;
for(PersonSimilarityRating ULa: USLa) {
    j++;
    if ((j == 1) && (ULa.getSimilarity() == 1.0)) {
        similarity1exists = 1;
        System.out.println("THERE IS A SIMILARITY=1");
    }
    if ((ULa.getSimilarity()<0.99) && (flagContinue==1)){
        flagContinue=0;
        System.out.println("flagContinue became 0");
    }
    if ((similarity1exists==1) && (flagContinue==1) ){
        System.out.println("similarity1exists is 1.. flagContinue is 1");
        k++;
        count++;
        sumOfSimilarities+=Math.abs(ULa.getSimilarity());
        if("positive".equals(ULa.getSentimentCategory())){
            meanKNNR+= ULa.getIntensity()*ULa.getSimilarity();
            meanSim+= ULa.getIntensity();
        }
        else if("negative".equals(ULa.getSentimentCategory())){
            meanKNNR-= ULa.getIntensity()*ULa.getSimilarity();
            meanSim-= ULa.getIntensity();
        }
        System.out.println("user:" + ULa.getPersonId() + "--asset:" + assetId +
"--Absolute Rating:"+ULa.getIntensity()+"k:..."+k);
    } else if ((similarity1exists==1) && (flagContinue==0) ){
        System.out.println("similarity1exists is 1 .. flagContinue is 0");
        flagContinue=2;
        //calculate mean
        if (count==0){
            meanSim=0.0;
        }
```

```
        else{
            meanSim=meanSim/count;
        }
        System.out.println("Mean of most Similar Users from List:.."+meanSim);
        //calculate stdev
        for (i=0;i<count;i++){
            ListEl=USLa.get(i);
            System.out.println("user in for i<count:" + ListEl.getPersonId() +
"--i: "+i);
            if("positive".equals(ListEl.getSentimentCategory())){
                diff=ListEl.getIntensity()-meanSim;
            }
            else if("negative".equals(ListEl.getSentimentCategory())){
                diff= ((-1)*ListEl.getIntensity())-meanSim; //or
sum=UV.getIntensity())+mean; it's the same since it will be raised to 2nd power
            }
            stdevSim +=Math.pow(diff, 2);
        }
        if (count==0 || count==1){
            stdevSim=0.0;
        }
        else {
            stdevSim=Math.sqrt((stdevSim/(count-1)));
        }
        System.out.println("Stdev of most Similar Users from List:...
"+stdevSim);
        //check this element
        //get rating
        if ("positive".equals(ULa.getSentimentCategory())) {
            rating = ULa.getIntensity();
        } else if ("negative".equals(ULa.getSentimentCategory())) {
            rating = (-1) * ULa.getIntensity();
        }
        sumOfSimilarities+=Math.abs(ULa.getSimilarity());
        // change rating if it isn't included in [mean-stdev,mean+stdev]
        // in this case we don't increment k
        if (rating<meanSim-stdevSim) {
            rating=meanSim-stdevSim;
            k--;
            System.out.println("Rating changed");
        } else if (rating>meanSim+stdevSim) {
            rating=meanSim+stdevSim;
            k--;
            System.out.println("Rating changed");
        }
        k++;
        System.out.println("user:" + ULa.getPersonId() + "--asset:" + assetId +
"--Rating: "+rating+"k:..."+k);
        meanKNNR+=rating*ULa.getSimilarity();
    } else if ((similarity1exists==1) && (flagContinue==2) ){
        System.out.println("similarity1exists is 1.. flagContinue is 2");
        //get rating
        if ("positive".equals(ULa.getSentimentCategory())) {
            rating = ULa.getIntensity();
        } else if ("negative".equals(ULa.getSentimentCategory())) {
            rating = (-1) * ULa.getIntensity();
        }
        sumOfSimilarities+=Math.abs(ULa.getSimilarity());
        // change rating if it isn't included in [mean-stdev,mean+stdev]
        // in this case we don't increment k
```

```
        if (rating<meanSim-stdevSim) {
            rating=meanSim-stdevSim;
            k--;
            System.out.println("Rating changed");
        } else if (rating>meanSim+stdevSim) {
            rating=meanSim+stdevSim;
            k--;
            System.out.println("Rating changed");
        }
        k++;
        System.out.println("user:" + ULa.getPersonId() + "--asset:" + assetId +
"--Rating: "+rating+"k:.. "+k);
        meanKNNR+=rating*ULa.getSimilarity();
    } else if (similarity1exists == 0) {
```

Listing 4.25: Κλάση KNNSimRatedSheepController5a.java και KNNSimRatedSheepController5b.java

Από όλες τις παραπάνω κλάσεις το μικρότερο σφάλμα προέκυψε για την KNNSimRatedSheepController5b. Η προσέγγιση που αντικατοπτρίζεται από την προαναφερθείσα κλάση βελτιώνεται περαιτέρω.

KNNSimRatedSheepController6

Επειδή δεν προβλέπεται για την προσέγγιση αυτή χειρισμός των περιπτώσεων στις οποίες *ο χρήστης είναι πιθανό να παρουσιάσει ιδιάζουσα συμπεριφορά (ισχύει Condition_1)* ΚΑΙ *δεν υπάρχει similarity = 1 στη λίστα των 10 πιο similar χρηστών που έχουν βαθμολογήσει την ταινία (δεν ισχύει Condition_2)*, εντάσσεται ο τρόπος που εφαρμόζεται στην πρώτη προσέγγιση για την αντιμετώπιση τους. Πιο συγκεκριμένα, αν *ο χρήστης που μας ενδιαφέρει έχει ιδιάζουσα συμπεριφορά ΚΑΙ ο χρήστης της λίστας ο οποίος ελέγχεται έχει similarity < 0.99 με τον αρχικό ΚΑΙ βρίσκεται από την 5η και κάτω θέση στη λίστα (≡ Condition_3b)*, τότε εξετάζεται περαιτέρω ο χρήστης της λίστας:

– Υπολογίζεται ο *μέσος όρος των βαθμολογιών* (αρχικών ή αναπροσαρμοσμένων) όσων similar χρηστών λαμβάνονται υπόψη *μέχρι εκείνο το σημείο (≡current_mean).*

– Αν $|(current\_mean) - (rating)| > 0.2$ *ΚΑΙ δεν υπάρχει χρήστης με similarity = 1 στη λίστα (≡ Condition_5a)*, τότε το rating αναπροσαρμόζεται ως εξής:

$$\begin{cases} rating = (current\_mean) - 0.2, αν\ rating < (current\_mean) - 0.2 \\ (\equiv\ Condition\_6) \\ rating = (current\_mean) + 0.2, αν\ rating > (current\_mean)\ + 0.2 \end{cases}$$

Έτσι, διαμορφώνεται η KNNSimRatedSheepController6.java.

KNNSimRatedSheepController7

Η επόμενη τροποποίηση που εφαρμόζεται είναι ο υπολογισμός του *μέσου όρου των βαθμολογιών* (αρχικών ή αναπροσαρμοσμένων) όσων similar χρηστών λαμβάνονται υπόψη *μέχρι εκείνο το σημείο (≡current_mean)* και όταν ισχύει το *Condition_3a*. Όταν, λοιπόν, ισχύει το *Condition_3a*, αλλά δεν ισχύουν τα *Condition_7a* και *Condition_7b*, τότε ελέγχεται αν ισχύει το *Condition_4c* και σε περίπτωση που αυτό ισχύει:

$$\begin{cases} rating = (current\_mean) - 0.2, αν\ rating < (current\_mean) - 0.2 \\ (\equiv\ Condition\_6) \\ rating = (current\_mean) + 0.2, αν\ rating > (current\_mean)\ + 0.2 \end{cases}$$

Από την τροποποίηση αυτή προκύπτει η κλάση KNNSimRatedSheepController7. Σκοπός της είναι ο χειρισμός περιπτώσεων που ο χρήστης θεωρείται "ύποπτος", αλλά δεν έχει μεγάλη ή μηδενική τυπική απόκλιση ώστε να περιοριστεί η επίδρασή του ή να απομονωθεί αντίστοιχα· ωστόσο απέχει από τη μέση βαθμολογία των πιο similar χρηστών για την ταινία και δεν έχει υπερβολικά μεγάλο similarity με τον αρχικό χρήστη. Με τον τρόπο αυτό, ομαλοποιείται η επίδραση τέτοιων χρηστών στις προβλέψεις του συστήματος.

KNNSimRatedSheepController8

Στη συνέχεια, το *Condition_7b* της KNNSimRatedSheepController4b.java αντικαθίσταται με συνδυασμό αυτού και του *Condition_4b* της KNNSimRatedSheepController4a.java. Πιο συγκεκριμένα, αν ισχύει το *Condition_3a*, τότε:

– Αν *stdev_User = 0 ΚΑΙ (* $|(current\_mean) - (rating)| > 0.2$ *'Η numOfRated ≥ 4) (≡ Condition_9),* τότε δεν λαμβάνεται υπόψη ο χρήστης.

Αυτό σημαίνει ότι σε περίπτωση που ένας χρήστης είναι "ύποπτος" και μέχρι τώρα έχει βαθμολογήσει ίδια όλες τις ταινίες θα απομονωθεί αν η συμπεριφορά αυτή έχει παρατηρηθεί αρκετές φορές ή αν απέχει πολύ βαθμολογικά από τη μέση βαθμολογία των πιο similar από αυτόν χρηστών. Έτσι δημιουργείται η κλάση KNNSimRatedSheepController8.

KNNSimRatedSheepController9

Η επόμενη αλλαγή αφορά στο *Condition_7a*. Σε περίπτωση που η συνθήκη αυτή ισχύει, δηλαδή ο χρήστης παρουσιάζει μεγάλη τυπική απόκλιση, η αναπροσαρμογή του rating τελείται, πλέον, ως εξής:

$$rating = \frac{[\,(rating)+(mean\_Asset)\,+(current\_mean)]}{3}$$

Με αυτή τη μέθοδο, το αναπροσαρμοσμένο rating επηρεάζεται όχι μόνο από το αρχικό του χρήστη και τη μέση βαθμολογία της ταινίας, αλλά και από την άποψη των πιο similar χρηστών. Η παραπάνω αλλαγή υλοποιείται στην κλάση KNNSimRatedSheepController9.

KNNSimRatedSheepController10

Η τελευταία τροποποίηση που πραγματοποιήθηκε σχετίζεται με το χειρισμό των περιπτώσεων στις οποίες παρατηρείται similarity = 1 στη λίστα των πιο similar χρηστών (ισχύει, δηλαδή, το *Condition_2*). Στις περιπτώσεις αυτές, λοιπόν, αφού υπολογιστούν τα mean_similar και stdev_similar, ελέγχονται:

– Αν $(mean\_similar) + (stdev\_similar) > 1$ *(≡ Condition_10a)*, τότε:

– Αν $rating < (mean\_similar) - \frac{(stdev\_similar)}{2}$ *(≡ Condition_11)*, τότε το rating αναπροσαρμόζεται ως εξής:

$$rating = (mean\_similar) - \frac{(stdev\_similar)}{2}$$

– Σε κάθε άλλη περίπτωση λαμβάνεται κανονικά υπόψη η βαθμολογία του.

Δηλαδή, αν ισχύει το *Condition_10a*, τότε όλα τα rating βρίσκονται (είτε από πριν είτε αναπροσαρμόστηκαν) εντός του διαστήματος:

$$\left[(mean\_similar) - \frac{(stdev\_similar)}{2}, 1\right]$$

Περιορίζεται, λοιπόν, το εύρος των τιμών που επιτρέπονται ως rating. Αυτό συμβαίνει, γιατί όταν το άθροισμα μέσης τιμής και τυπικής απόκλισης υπερβαίνει τη μέγιστη τιμή που μπορεί να παρατηρηθεί (στην περίπτωσή μας τη μονάδα), υπάρχουν πολλές υψηλές τιμές στα δεδομένα μας, δηλαδή πολλά υψηλά rating.

Επομένως, αυξάνεται το κάτω άκρο του διαστήματος, ώστε να αντικατοπτρίζει το γεγονός αυτό.

– Αλλιώς αν $(mean\_similar) - (stdev\_similar) < -1$ *(≡ Condition_10b)*, τότε:

– Αν $rating > (mean\_similar) + \frac{(stdev\_similar)}{2}$ *(≡ Condition_12)* τότε το rating αναπροσαρμόζεται ως εξής:

$$rating = (mean\_similar) + \frac{(stdev\_similar)}{2}$$

– Σε κάθε άλλη περίπτωση λαμβάνεται κανονικά υπόψη η βαθμολογία του.

Δηλαδή, αν ισχύει το *Condition_10b*, τότε όλα τα rating βρίσκονται (είτε από πριν είτε αναπροσαρμόστηκαν) εντός του διαστήματος:

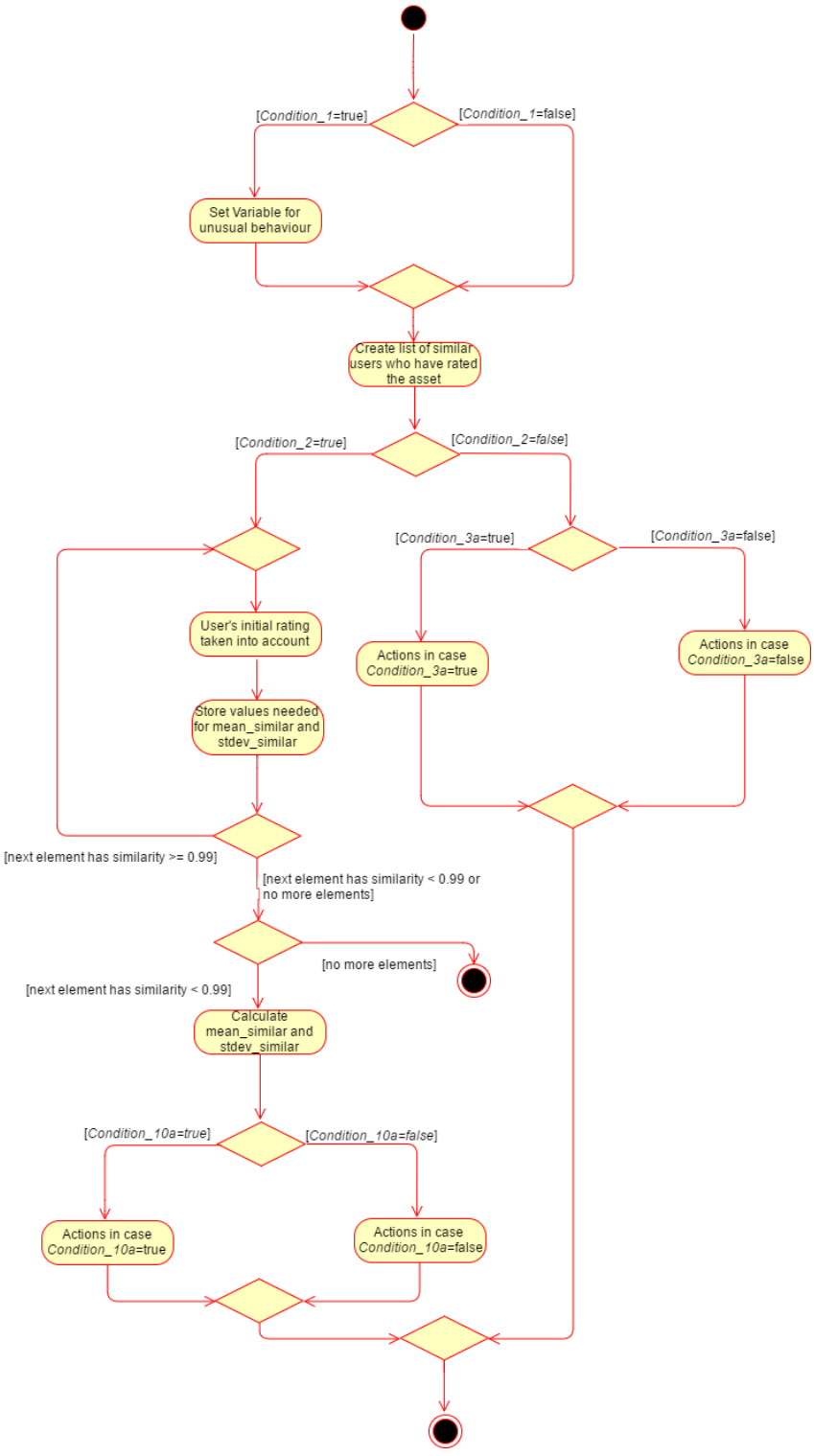
$$\left[-1\,, (mean\_similar) + \frac{(stdev\_similar)}{2}\right]$$

Περιορίζεται, λοιπόν, και πάλι το εύρος των τιμών που επιτρέπονται ως rating. Η αιτιολόγηση είναι αντίστοιχη με προηγουμένως, με τη διαφορά ότι στην περίπτωση αυτή υπάρχουν πολλές χαμηλές βαθμολογίες.

– Σε κάθε άλλη περίπτωση ακολουθούνται οι διαδικασίες που ορίστηκαν στις προηγούμενες κλάσεις.

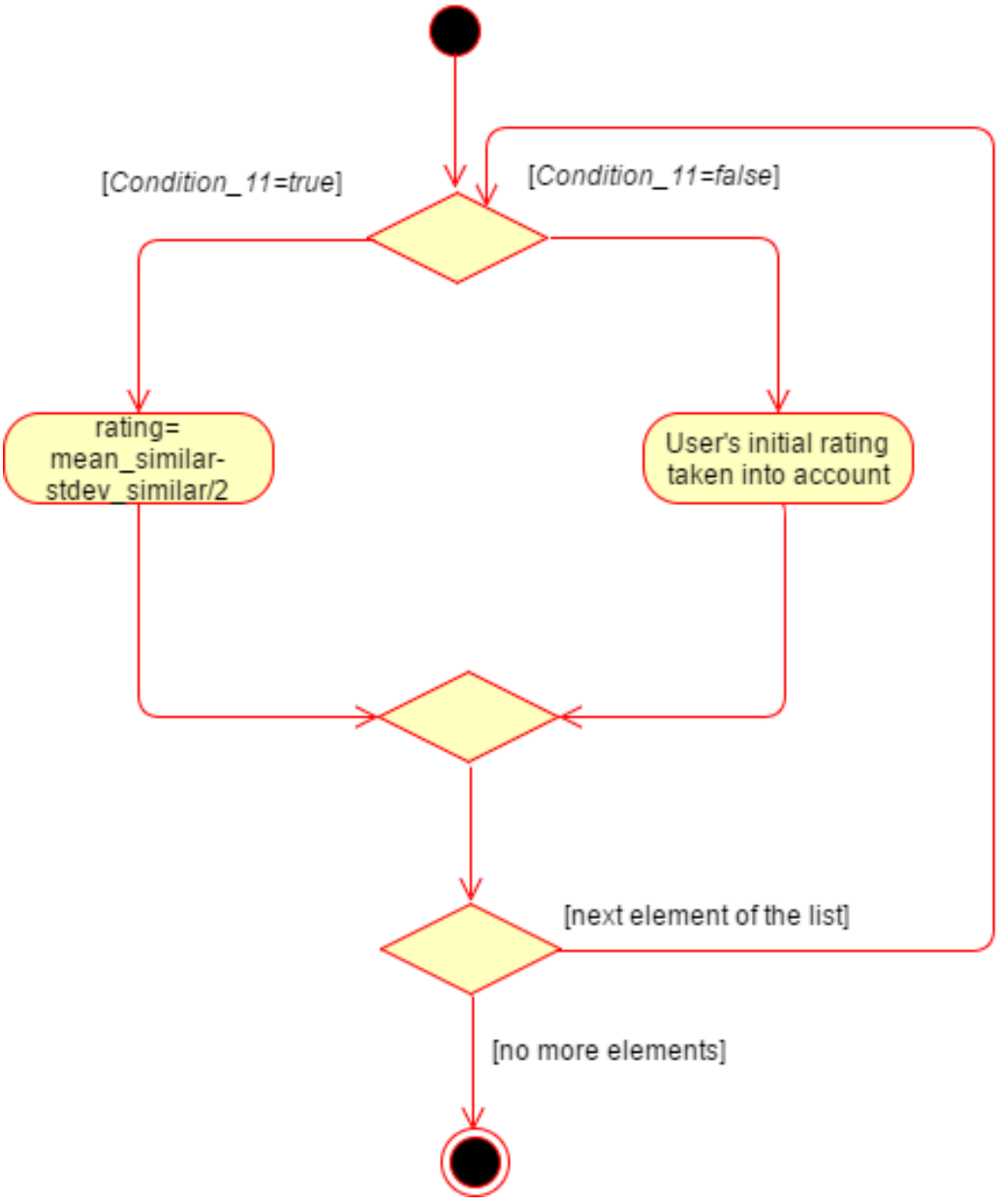
Οι τροποποιήσεις αυτές εντάσσονται στην κλάση KNNSimRatedSheepController10.

Ακολούθως, παρατίθεται το activity diagram της κλάσης αυτής:



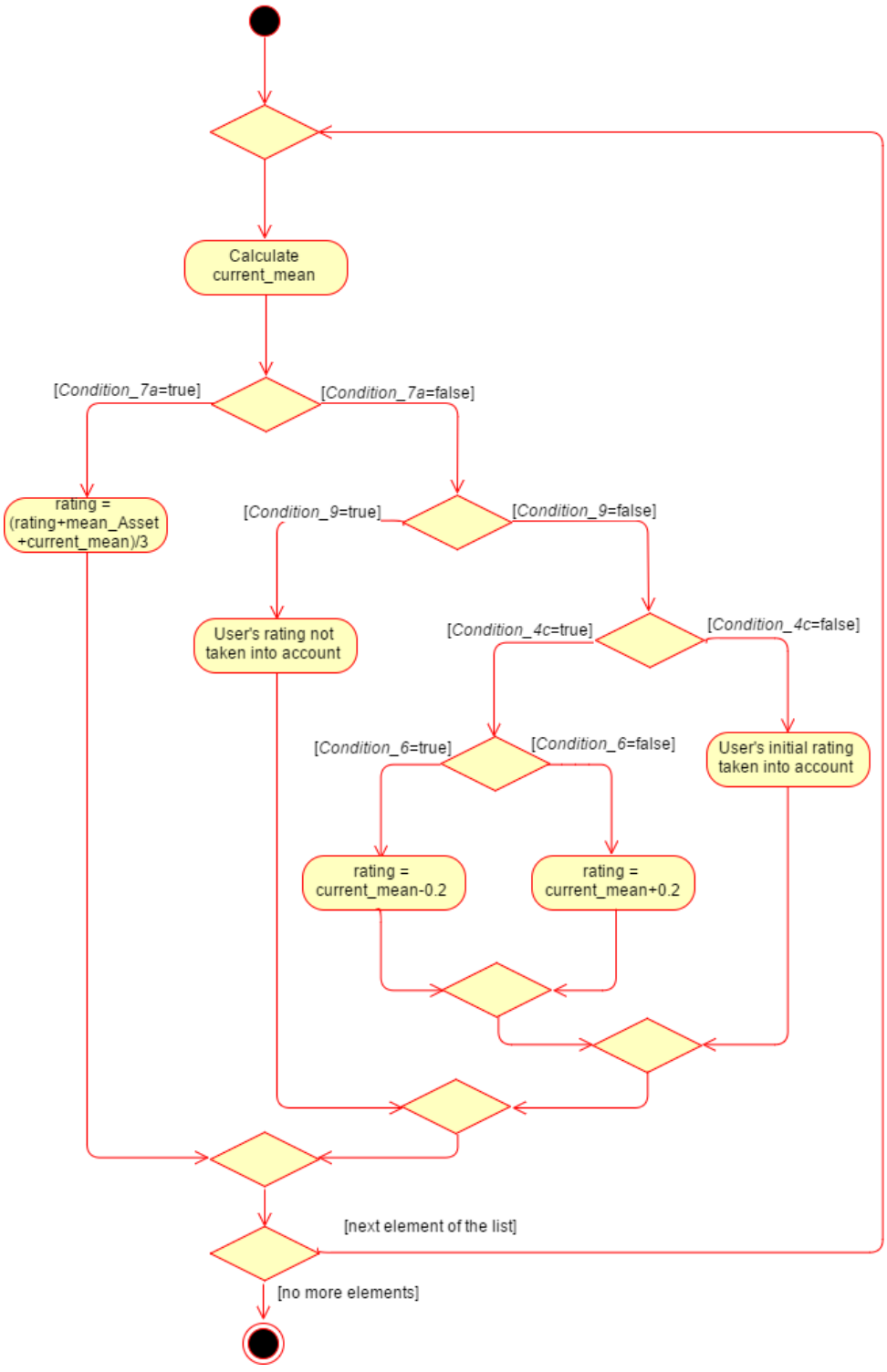
Σχήμα 4.8: Activity Diagram για KNNSimRatedSheepController10.java

Για καλύτερη εποπτεία, οι ενέργειες που περιλαμβάνονται στα Activity Nodes "Actions in case Condition_10a=true", "Actions in case Condition_10a=false", "Actions in case Condition_3a = true" και "Actions in case Condition_3a = false" παρουσιάζονται στα ακόλουθα activity diagrams:

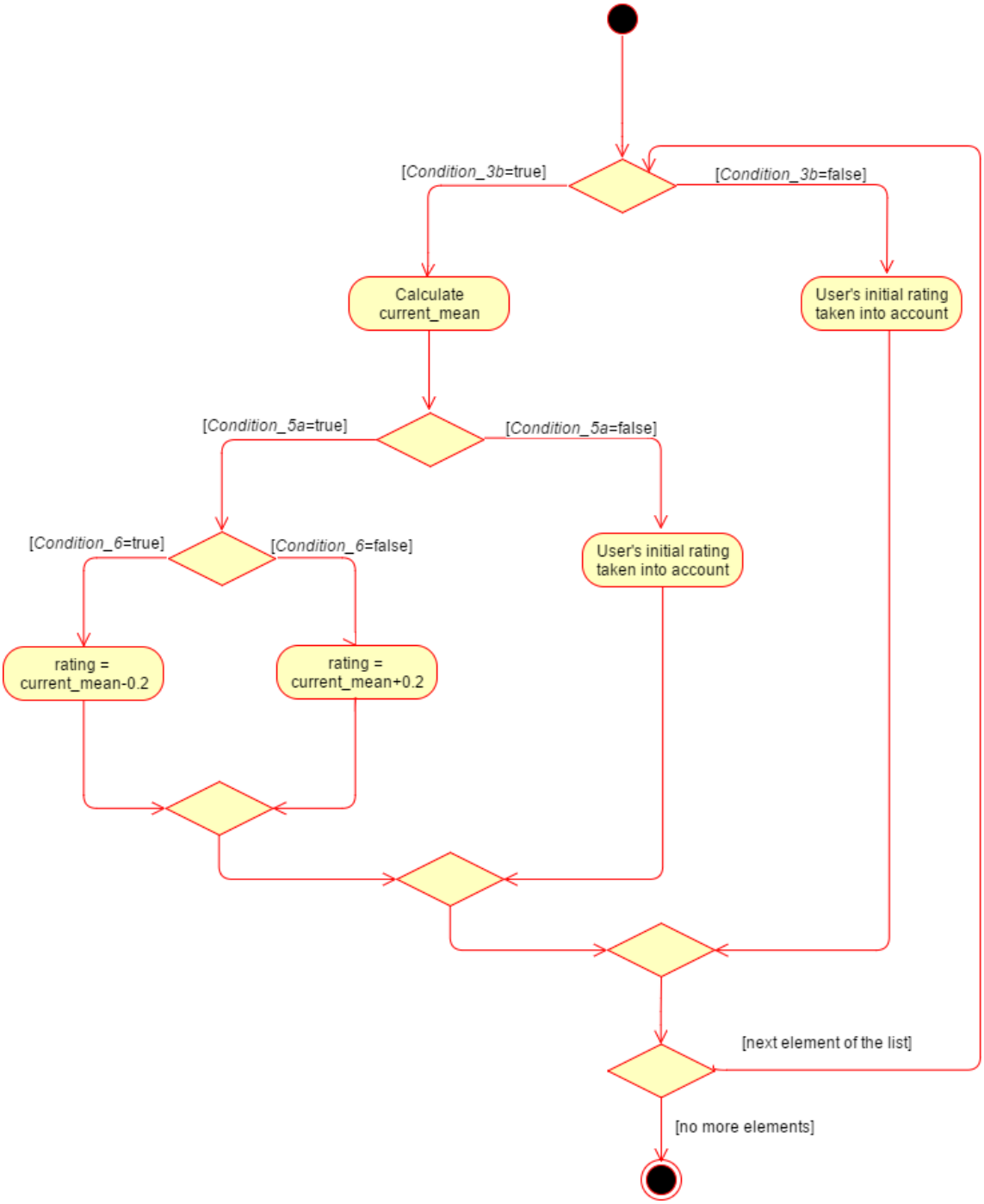


Σχήμα 4.9: Activity Diagram για "Actions in case Condition_10a = true" της KNNSimRatedSheepController10.java

Σχήμα 4.10: Activity Diagram για "Actions in case Condition_10a = false" της KNNSimRatedSheepController10.java



Σχήμα 4.11: Activity Diagram για "Actions in case Condition_3a = true" της KNNSimRatedSheepController10.java



Σχήμα 4.12: Activity Diagram για "Actions in case Condition_3a = false" της KNNSimRatedSheepController10.java

Η τελική κλάση KNNSimRatedSheepController10 παρουσιάζεται παρακάτω:

```
package gr.ntua.sam.context.rest;

import gr.ntua.sam.context.neo4j.AssetRepository;
import gr.ntua.sam.context.neo4j.PersonRepository;
import gr.ntua.sam.context.resources.entities.Asset;
import gr.ntua.sam.context.resources.entities.Person;
import gr.ntua.sam.context.resources.helpers.PersonSimilarityRating;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;
import org.springframework.data.neo4j.template.Neo4jOperations;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.*;

@RestController
@Configuration
@EnableNeo4jRepositories
public class KNNSimRatedSheepController10 {
    @Autowired
    private PersonRepository personRepository;

    @Autowired
    private AssetRepository assetRepository;

    @Autowired
    Neo4jOperations neo4jTemplate;

    @ApiOperation(value = "calculate prediction For 10th (my) KNN", tags =
"KNN")
    @RequestMapping(value = "/knn10/{personId}/{assetId}", method =
RequestMethod.GET)
    public ResponseEntity calculatePrediction10thKNNb(@PathVariable("personId")
String personId, @PathVariable("assetId") String assetId) {

        Person person = personRepository.findByPersonId(personId);
        if (person == null) {
            return new ResponseEntity<>("Person id not valid.",
HttpStatus.BAD_REQUEST);
        }
        Asset currentAsset = assetRepository.findByAssetId(assetId);
        if (currentAsset == null) {
            return new ResponseEntity<>("Asset does not exist",
HttpStatus.BAD_REQUEST);
        }

        double sumOfSimilarities=0.0;
        double meanKNNR=0.0;
        Integer k=10;
        int j=0; // to count which list element we're processing
```

```
        double rating=0;
        int whiteUser=1; // the weight of the user's rating considering his
overall behaviour
        int notAveraged=1;
        int uncommonBehaviour=0; // to check if asset's mean is included in
[meanUsr-stdevUsr,meanUsr+stdevUsr]
        int similarity1exists=0; //to check if there are similarities=1
        double meanSim=0.0; // to calculate mean if there is similarity=1
        double stdevSim=0.0; // to calculate stdev if there is similarity=1
        double diff=0.0; // for stdev calculation
        int count=0; // for mean and stdev calculation
        int flagContinue=1;
        int i=0;
        double meanList=0.0; // to calculate mean rating up until the current
user
        double sumList=0.0; // to sum ratings up until the current user
        PersonSimilarityRating ListEl;
        int changedRatings=0; // number of changed ratings
        int unusedRatings=0; // number of ratings not taken into account
        int high=2; //to check if mean+stdev>1 or mean-stdev<-1
        uncommonBehaviour=0;
        // to check if the person HAS uncommon Behaviour
        if ((person.getNumOfRated()>=4) && (
((person.getMeanUsrRating()+person.getStdevUsrRating())<currentAsset.getMeanAss
tRating()) ||  ((person.getMeanUsrRating()-
person.getStdevUsrRating())>currentAsset.getMeanAsstRating()) ) ){
            uncommonBehaviour=1;
        }
        List<PersonSimilarityRating> USLa =
personRepository.getInfoMostSimilarAndRated(personId,assetId,k);
        k=0;
        for(PersonSimilarityRating ULa: USLa) {
            j++;
            // to check if there is a user with similarity = 1
            if ((j == 1) && (ULa.getSimilarity() == 1.0)) {
                similarity1exists = 1;
                System.out.println("THERE IS A SIMILARITY=1");
            }
            // to check when the similarity becomes lower than 0.99 for the
first time
            if ((ULa.getSimilarity()<0.99) && (flagContinue==1)){
                flagContinue=0;
                System.out.println("flagContinue became 0");
            }
            if ((similarity1exists==1) && (flagContinue==1) ){
                System.out.println("similarity1exists is 1... flagContinue is
1");
                k++;
                count++;
                sumOfSimilarities+=Math.abs(ULa.getSimilarity());
                if("positive".equals(ULa.getSentimentCategory())){
                    meanKNNR+= ULa.getIntensity()*ULa.getSimilarity();
                    meanSim+= ULa.getIntensity();
                }
                else if("negative".equals(ULa.getSentimentCategory())){
                    meanKNNR-= ULa.getIntensity()*ULa.getSimilarity();
                    meanSim-= ULa.getIntensity();
                }
                System.out.println("user:" + ULa.getPersonId() + "--asset:" +
assetId + "--Absolute Rating: "+ULa.getIntensity()+"--k: "+k);
```

```
            }
             else if ((similarity1exists==1) && (flagContinue==0) ){
                System.out.println("similarity1exists is 1... flagContinue is
0");
                flagContinue=2; // to prevent from entering a 2nd time
                //calculate mean of ratings for users with similarity of 0.99
or higher
                if (count==0){
                    meanSim=0.0;
                }
                else{
                    meanSim=meanSim/count;
                }
                System.out.println("Mean of most Similar Users from List:...
"+meanSim);
                //calculate stdev of ratings for users with similarity of 0.99
or higher
                for (i=0;i<count;i++){
                    ListEl=USLa.get(i);
                    System.out.println("user in for i<count:" +
ListEl.getPersonId() + "--i: "+i);
                    if("positive".equals(ListEl.getSentimentCategory())){
                        diff=ListEl.getIntensity()-meanSim;
                    }
                    else if("negative".equals(ListEl.getSentimentCategory())){
                        diff= ((-1)*ListEl.getIntensity())-meanSim; //or
sum=UV.getIntensity())+mean; it's the same since it's gonna be raised to 2nd
power
                    }
                    stdevSim +=Math.pow(diff, 2);
                }
                if (count==0 || count==1){
                    stdevSim=0.0;
                }
                else {
                    stdevSim=Math.sqrt((stdevSim/(count-1)));
                }
                System.out.println("Stdev of most Similar Users from
List:..."+stdevSim);
                if (meanSim+stdevSim>1){
                    high=1;
                }
                else if (meanSim-stdevSim<-1){
                    high=0;
                }
                //check this element
                //get rating
                if ("positive".equals(ULa.getSentimentCategory())) {
                    rating = ULa.getIntensity();
                } else if ("negative".equals(ULa.getSentimentCategory())) {
                    rating = (-1) * ULa.getIntensity();
                }
                sumOfSimilarities+=Math.abs(ULa.getSimilarity());
                // change rating if needed
```

```
                    // in this case we don't increment k
                    if ((high==1) && (rating<meanSim-(stdevSim/2))){
                        rating=meanSim-(stdevSim/2);
                        k--;
                        changedRatings++;
                        System.out.println("Rating changed");
                    }
                    else if ((high==0) && (rating>meanSim+(stdevSim/2))){
                        rating=meanSim+(stdevSim/2);
                        k--;
                        changedRatings++;
                        System.out.println("Rating changed");
                    }
                    else if ((high==2) && (rating<meanSim-stdevSim)) {
                        rating=meanSim-stdevSim;
                        k--;
                        changedRatings++;
                        System.out.println("Rating changed");
                    }
                    else if ((high==2) && (rating>meanSim+stdevSim)) {
                        rating=meanSim+stdevSim;
                        k--;
                        changedRatings++;
                        System.out.println("Rating changed");
                    }
                    k++;
                    System.out.println("user:" + ULa.getPersonId() + "--asset:" +
assetId + "--Rating: "+rating+"--k: "+k);
                    meanKNNR+=rating*ULa.getSimilarity();
                } else if ((similarity1exists==1) && (flagContinue==2) ){
                    System.out.println("similarity1exists is 1.. flagContinue is
2");
                    //get rating
                    if ("positive".equals(ULa.getSentimentCategory())) {
                        rating = ULa.getIntensity();
                    } else if ("negative".equals(ULa.getSentimentCategory())) {
                        rating = (-1) * ULa.getIntensity();
                    }
                    sumOfSimilarities+=Math.abs(ULa.getSimilarity());
                    // change rating if needed
                    // in this case we don't increment k
                    if ((high==1) && (rating<meanSim-(stdevSim/2))){
                        rating=meanSim-(stdevSim/2);
                        k--;
                        changedRatings++;
                        System.out.println("Rating changed");
                    }
                    else if ((high==0) && (rating>meanSim+(stdevSim/2))){
                        rating=meanSim+(stdevSim/2);
                        k--;
                        changedRatings++;
                        System.out.println("Rating changed");
                    }
                    else if ((high==2) && (rating<meanSim-stdevSim)) {
                        rating=meanSim-stdevSim;
                        k--;
                        changedRatings++;
                        System.out.println("Rating changed");
                    }
                    else if ((high==2) && (rating>meanSim+stdevSim)) {
                        rating=meanSim+stdevSim;
                        k--;
```

```
                    changedRatings++;
                    System.out.println("Rating changed");
                }
                k++;
                System.out.println("user:" + ULa.getPersonId() + "--asset:" +
assetId + "--Rating: "+rating+"--k: "+k);
                meanKNNR+=rating*ULa.getSimilarity();
            } else if (similarity1exists == 0) {
                System.out.println("similarity1exists is 0");
                // if there is no user with similarity=1 on the list
                //get rating
                if ("positive".equals(ULa.getSentimentCategory())) {
                    rating = ULa.getIntensity();
                } else if ("negative".equals(ULa.getSentimentCategory())) {
                    rating = (-1) * ULa.getIntensity();
                }
                System.out.println("user:" + ULa.getPersonId() + "--asset:" +
assetId + "--rating:" + rating + "--STDEV:" + ULa.getStdevUsrRating());
                whiteUser = 1;
                notAveraged = 1;
                //check if the rating is significantly different from asset's
mean rating provided that the asset has already been rated
                //only if the user isn't one of the 4 most similar users on the
list
                //note: assets have initial mean value of 0 in case they
haven't been rated
                if ((uncommonBehaviour == 0) && (j >= 5) &&
(Math.abs(currentAsset.getMeanAsstRating() - rating) >= 0.3) &&
currentAsset.getNumOfRatings() > 5) {
                    meanList = sumList / count;
                    //if the user has a great standard deviation we average his
rating
                    if ((ULa.getStdevUsrRating() >= 0.5)) {
                        notAveraged = 0;
                        changedRatings++;
                        rating = (rating +
currentAsset.getMeanAsstRating()+meanList) / 3;
                    } //if the user has 0 standard deviation we don't take into
account his rating
                    else if ((ULa.getStdevUsrRating() == 0.0) &&
((ULa.getNumOfRated() >= 4) || (Math.abs(meanList - rating) > 0.2))) {
                        whiteUser = 0;
                        unusedRatings++;
                    } //if the user has rated the asset differently from
previous users of the list and his similarity is lower than 0.99, his rating is
changed
                    else if ((Math.abs(meanList - rating) > 0.2) &&
(ULa.getSimilarity() < 0.99)) {
                        if (rating < meanList - 0.2) {
                            rating = meanList - 0.2;
                            notAveraged = 0;
                            changedRatings++;
                        } else if (rating > meanList + 0.2) {
                            rating = meanList + 0.2;
                            notAveraged = 0;
                            changedRatings++;
                        }
                    }
                } // if the user HAS uncommon Behaviour AND his similarity is
lower than 0.99 AND he's not one of the 4 most similar users, we change his
rating
```

```
                else if ((uncommonBehaviour == 1) && (ULa.getSimilarity() <
0.99) && (j >= 5)) {
                    meanList = sumList / count;
                    if ((Math.abs(meanList - rating) > 0.2) &&
(similarity1exists == 0)) {
                        if (rating < meanList - 0.2) {
                            rating = meanList - 0.2;
                            notAveraged = 0;
                            changedRatings++;
                        } else if (rating > meanList + 0.2) {
                            rating = meanList + 0.2;
                            notAveraged = 0;
                            changedRatings++;
                        }
                    }
                }
                if ((whiteUser == 0) || (notAveraged == 0)) { // if the user
shouldn't be taken into account
                    k--;
                }
                k++;
                count++;
                sumList += rating;
                System.out.println("White User:... " + whiteUser);
                System.out.println("Not Averaged:..." + notAveraged);
                sumOfSimilarities += whiteUser * Math.abs(ULa.getSimilarity());
                meanKNNR += whiteUser * rating * ULa.getSimilarity();
            } // if similarity1exists==0
        }
        if ((k == 0) || (sumOfSimilarities == 0.0)) {
            meanKNNR = 0.0;
        } else {
            meanKNNR = meanKNNR / sumOfSimilarities;
        }


System.out.println("PersonId:..."+personId+"..assetId:..."+assetId+"..meanKNNR:
... "+meanKNNR+"--k:"+k+"--changedRatings:"+changedRatings+"--
unusedRatings:"+unusedRatings+"--uncommonBehaviour:"+uncommonBehaviour+"--
similarity1exists:"+similarity1exists);
        return new ResponseEntity<>(meanKNNR,HttpStatus.OK);

    }
}
```

Listing 4.26: Κλάση KNNSimRatedSheepController10.java

Συνεπώς, ο τελικός αλγόριθμος ελέγχει, αρχικά, αν υπάρχουν χρήστες με similarity = 1 στη λίστα των πιο similar χρηστών που έχουν βαθμολογήσει την ταινία. Σε περίπτωση που υπάρχει έστω και ένας τέτοιος χρήστης, διαδραματίζει ρόλο ύψιστης σημασίας στον υπολογισμό της πρόβλεψης, αφού μέχρι τώρα η γνώμη του ταυτίζεται με αυτή του χρήστη που μας ενδιαφέρει. Συνεπώς, η βαθμολογία του, όπως και αυτή όσων χρηστών έχουν πολύ υψηλό similarity, λαμβάνεται υπόψη ως έχει για τον υπολογισμό της πρόβλεψης. Παράλληλα, υπολογίζεται η μέση τιμή (≡ mean_similar) και η τυπική απόκλιση (≡ stdev_similar) των βαθμολογιών αυτών, ώστε να δημιουργηθεί ένα

διάστημα επιτρεπόμενων τιμών rating για τους υπόλοιπους χρήστες. Κατόπιν, ελέγχεται αν το άθροισμα της μέσης τιμής και της τυπικής απόκλισης είναι μεγαλύτερο από την υψηλότερη δυνατή τιμή rating ή αν η διαφορά τους είναι μικρότερη από τη χαμηλότερη δυνατή τιμή rating. Αν ισχύει το πρώτο, σημαίνει ότι πολλές βαθμολογίες είναι αρκετά υψηλές, οπότε οι επιτρεπόμενες τιμές rating περιέχονται στο διάστημα [(mean_similar)-(stdev_similar)/2, 1]. Αν ισχύει το δεύτερο, υπάρχουν πολλές βαθμολογίες αρκετά χαμηλές, οπότε οι επιτρεπόμενες τιμές rating περιέχονται στο διάστημα [-1, (mean_similar)+(stdev_similar)/2]. Αν δεν ισχύει τίποτα από τα δυο, οι επιτρεπόμενες τιμές rating ανήκουν στο διάστημα [(mean_similar) - (stdev_similar), (mean_similar) + (stdev_similar)]. Η βαθμολογία κάθε χρήστη της λίστας με μικρότερο similarity λαμβάνεται υπόψη ως έχει αν ανήκει στο κατάλληλο διάστημα, διαφορετικά αναπροσαρμόζεται στο πλησιέστερο άκρο του.

Αν δεν υπάρχουν χρήστες με similarity = 1 και ο χρήστης για τον οποίο πραγματοποιείται πρόβλεψη δεν είναι πιθανό να παρουσιάσει ιδιάζουσα συμπεριφορά, ακολουθείται διαφορετική προσέγγιση. Δηλαδή, λαμβάνονται υπόψη ως έχουν οι βαθμολογίες των πρώτων τεσσάρων χρηστών της λίστας, εφόσον δε θεωρείται λογικό να αγνοηθούν οι πιο similar χρήστες. Μετά από αυτούς, όμως, αν παρουσιαστεί μεγάλη απόκλιση μεταξύ της μέσης βαθμολογίας της ταινίας και του rating του εκάστοτε χρήστη, τότε ο χρήστης αυτός θεωρείται ως "ύποπτος" και πραγματοποιούνται περαιτέρω έλεγχοι. Όπως παρατηρείται, αυτή τη φορά τον "ύποπτο" χρήστη καθορίζει όχι η απόκλιση της βαθμολογίας του από αυτή των πιο similar χρηστών, αλλά η απόκλιση από τη μέση βαθμολογία της ταινίας. Και αυτό, γιατί οι πιο similar χρήστες σε αυτήν την περίπτωση δεν έχουν τόσο υψηλό similarity και συνεπώς δεν μπορούμε να στηριχτούμε εξολοκλήρου σε αυτούς. Στην περίπτωση, λοιπόν, που η απόκλιση που παρατηρείται είναι μεγάλη (με την προϋπόθεση ότι υπάρχει ήδη ικανός αριθμός rating για την ταινία), ελέγχεται αν η τυπική απόκλιση του χρήστη είναι υψηλή και αν αυτό ισχύει, τότε η βαθμολογία αναπροσαρμόζεται ως ο μέσος όρος αυτής, της μέσης βαθμολογίας της ταινίας και του μέσου όρου των βαθμολογιών των πιο similar χρηστών. Αν η τυπική απόκλιση του χρήστη είναι μηδενική, δηλαδή ο χρήστης έχει ίδιο rating σε όσες ταινίες βαθμολόγησε, τότε θεωρείται ως "ένοχος" και δε λαμβάνεται υπόψη. Στην περίπτωση που η τυπική απόκλιση είναι μηδενική, αλλά ο χρήστης έχει βαθμολογήσει ελάχιστες ταινίες, οπότε δεν μπορούν να εξαχθούν ασφαλή συμπεράσματα για τη συμπεριφορά του, ελέγχεται αν η βαθμολογία του απέχει αρκετά από το μέσο όρο των βαθμολογιών των πιο similar χρηστών και αν αυτό ισχύει, δε λαμβάνεται υπόψη, καθώς θεωρείται πιθανό στο μέλλον να εμπέσει στην προηγούμενη κατηγορία. Αν ο χρήστης είναι "ύποπτος", αλλά δεν έχει ούτε υψηλή ούτε μηδενική τυπική απόκλιση, τότε ελέγχεται αν το similarity του δεν είναι υπερβολικά υψηλό και η βαθμολογία του απέχει αρκετά από το μέσο όρο των βαθμολογιών των πιο similar χρηστών. Σε περίπτωση που ισχύουν αυτά, η βαθμολογία του αναπροσαρμόζεται σε τιμή μεγαλύτερη ή μικρότερη κατά 0.2 από το μέσο όρο ανάλογα με το αν είναι αρκετά υψηλότερη ή χαμηλότερη από αυτόν. Σε

όλες τις άλλες περιπτώσεις που ο χρήστης θεωρείται "ύποπτος" βάσει της παραπάνω συνθήκης λαμβάνεται κανονικά υπόψη.

Αν δεν υπάρχουν χρήστες με similarity = 1 και ο χρήστης για τον οποίο πραγματοποιείται πρόβλεψη είναι πιθανό να παρουσιάσει ιδιάζουσα συμπεριφορά, ελέγχονται τα ακόλουθα. Εφόσον η συμπεριφορά του προβλέπεται να είναι ιδιάζουσα, δεν έχει νόημα να πραγματοποιηθούν συγκρίσεις με τη μέση βαθμολογία της ταινίας, γιατί το πιο πιθανό είναι να μη συμφωνεί με αυτή. Επομένως, βασικός γνώμονας για την πρόβλεψη της προτίμησης τέτοιων χρηστών είναι οι προτιμήσεις των χρηστών που έχουν παρόμοια συμπεριφορά με αυτούς. Λαμβάνονται, λοιπόν, υπόψη ως έχουν οι βαθμολογίες των τεσσάρων πιο similar χρηστών της λίστας και για κάθε άλλο χρήστη αυτής ελέγχεται αν ο μέσος όρος των βαθμολογιών των πιο similar από αυτόν χρηστών απέχει αρκετά από τη βαθμολογία του. Αν κάτι τέτοιο ισχύει, τότε η βαθμολογία του αναπροσαρμόζεται σε τιμή μεγαλύτερη ή μικρότερη κατά 0.2 από το μέσο όρο ανάλογα με το αν είναι αρκετά υψηλότερη ή χαμηλότερη από αυτόν. Σε όλες τις υπόλοιπες περιπτώσεις, οι βαθμολογίες των similar χρηστών λαμβάνονται υπόψη ως έχουν.

Κεφάλαιο 5

Αξιολόγηση Τροποποιημένου Συστήματος

Στο παρόν κεφάλαιο παρουσιάζονται τα αποτελέσματα των βελτιστοποιήσεων που πραγματοποιήθηκαν και περιγράφηκαν προηγουμένως. Οι αλγόριθμοι που προέκυψαν συγκρίνονται τόσο σε επίπεδο σφαλμάτων πρόβλεψης της αναμενόμενης συμπεριφοράς των χρηστών όσο και σε επίπεδο ταχύτητας.

5.1 Μέτρα Υπολογισμού Σφαλμάτων

Για την εκτίμηση των σφαλμάτων που προκύπτουν μετά από κάθε τροποποίηση και κατ' επέκταση της ακρίβειας χρησιμοποιούνται τρία διαφορετικά μέτρα:

- Root-mean-square error (RMSE)
  Χρησιμοποιείται για τον υπολογισμό των διαφορών μεταξύ των τιμών που προβλέπονται από ένα μοντέλο ή εκτιμητή και των πραγματικών. Αποτελεί, γενικά, αξιόπιστο μέτρο ακρίβειας και προκύπτει από τον ακόλουθο τύπο [40]:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - r_i)^2}{n}}$$

  όπου p_i είναι η τιμή που προβλέπεται, r_i είναι η πραγματική τιμή που παρατηρείται και n είναι το σύνολο των προβλέψεων που πραγματοποιήθηκαν.

- Mean Absolute Error (MAE)
  Χρησιμοποιείται και αυτό για τον υπολογισμό της εγγύτητας των προβλεπόμενων τιμών με τις πραγματικές. Υπολογίζεται με χρήση του εξής τύπου [41]:

$$MAE = \frac{\sum_{i=1}^{n}|p_i - r_i|}{n}$$

  όπου p_i είναι η τιμή που προβλέπεται, r_i είναι η πραγματική τιμή που παρατηρείται και n είναι το σύνολο των προβλέψεων που πραγματοποιήθηκαν.

- Mean Percentage Error (MPE)
  Χρησιμοποιείται για να εκφράσει το ποσοστιαίο σφάλμα που προκύπτει μεταξύ των προβλεπόμενων και των πραγματικών τιμών. Υπολογίζεται από τον ακόλουθο τύπο [42]:

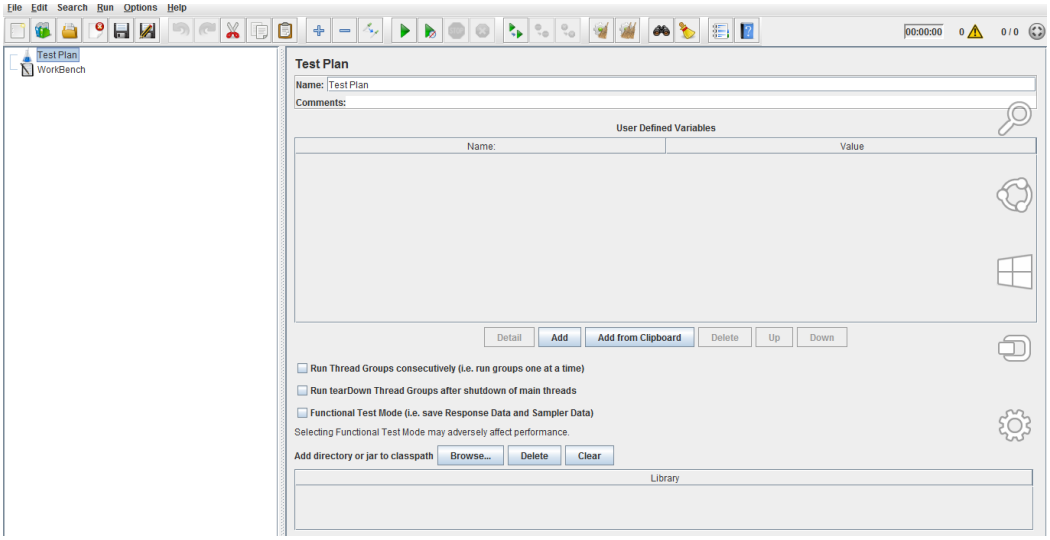
$$MPE = \frac{100\%}{n} \sum_{i=1}^{n} \frac{r_i - p_i}{r_i}$$

  όπου p_i είναι η τιμή που προβλέπεται, r_i είναι η πραγματική τιμή που παρατηρείται και n είναι το σύνολο των προβλέψεων που πραγματοποιήθηκαν.

5.2 Υπολογισμός Χρονικής Απόκρισης Συστήματος

Για τη σύγκριση του χρόνου που καταναλώνεται σε συναλλαγές με τη βάση δεδομένων από κάθε κλάση Controller, αλλά και το συνολικό χρόνο που απαιτείται, χρησιμοποιείται ένα ελεύθερο λογισμικό, το Apache JMeter [43][44]. Το Apache JMeter είναι ένα εργαλείο για ανάλυση και μέτρηση της επίδοσης διάφορων υπηρεσιών, στατικών και δυναμικών πηγών. Χρησιμοποιείται, δηλαδή, για τη μέτρηση της απόκρισης ενός συστήματος ή συσκευής υπό συνθήκες δημιουργίας ζήτησης. Μπορεί να αξιοποιηθεί για την προσομοίωση μεγάλου φορτίου σε ένα server, σύνολο από servers, δίκτυο ή αντικείμενο για τον έλεγχο της συμπεριφοράς του ή την ανάλυση της ολικής επίδοσης του υπό διαφορετικού τύπου φορτία.

Ανοίγοντας την εφαρμογή η αρχική οθόνη που εμφανίζεται είναι η ακόλουθη:



Σχήμα 5.1: Αρχική οθόνη Apache JMeter

Κάθε φορά, ανάλογα με τους επιθυμητούς ελέγχους και μετρήσεις, προστίθενται επιπλέον στοιχεία στο Test Plan, μεταβάλλονται παράμετροι και πραγματοποιούνται αντίστοιχες ρυθμίσεις.

5.3 Σύγκριση αρχικών βελτιστοποιημένων Αλγορίθμων

Οι αλλαγές που πραγματοποιήθηκαν στις ενότητες 4.2.1, 4.2.2, 4.2.3 επηρεάζουν τόσο την ακρίβεια των προβλέψεων όσο και την ταχύτητα υπολογισμού τους. Η επίδραση των αλλαγών αυτών διαφαίνεται στις ακόλουθες υποενότητες

5.3.1 Σύγκριση Ακρίβειας Πρόβλεψης

Οι τροποποιήσεις που πραγματοποιήθηκαν στον αρχικό k-NN στις ενότητες 4.2.1, 4.2.2, 4.2.3 επηρεάζουν, όπως αναφέρθηκε, την ακρίβεια των προβλέψεων. Για να διαπιστωθούν ποσοτικά τα αποτελέσματα που προκύπτουν ακολουθείται η διαδικασία που περιγράφεται ακολούθως. Πιο συγκεκριμένα, εφόσον είναι επιθυμητή η χρήση των τύπων υπολογισμού σφαλμάτων της ενότητας 5.1, είναι απαραίτητο για κάθε πρόβλεψη που πραγματοποιεί ο εκάστοτε k-NN αλγόριθμος να υπάρχει η αντίστοιχη πραγματική τιμή. Δηλαδή, αν πραγματοποιηθεί πρόβλεψη για το rating ενός χρήστη σε μια ταινία, θα πρέπει να διατίθεται στη βάση δεδομένων το πραγματικό rating του χρήστη αυτού για τη συγκεκριμένη ταινία. Θα υπολογιστεί, λοιπόν, η ακρίβεια των αλγορίθμων βάσει των σφαλμάτων που προκύπτουν για όλες τις καταχωρημένες στο σύστημα βαθμολογίες. Για να πραγματοποιηθεί αυτό, απαιτείται ο εκάστοτε k-NN αλγόριθμος να δέχεται κάθε φορά ως είσοδο ένα ζευγάρι (χρήστης, ταινία) για το οποίο υπάρχει καταχωρημένη βαθμολογία στη βάση. Προκειμένου να επιτευχθεί αυτό, εκτελείται το ακόλουθο query:

```
1. MATCH (a:Person)-[r:COMMENTS]->(b:Asset)
2. RETURN a.personId, b.assetId, CASE r.sentimentCategory
3. WHEN 'positive' THEN r.intensity
4. ELSE (-1)*r.intensity
5. END
```

Listing 5.1: Cypher query για ανάκτηση χρηστών, ταινιών και αντίστοιχων βαθμολογιών

Τα αποτελέσματα του query αυτού εξάγονται στο user_asset_rating.csv με τη σειρά που καταδεικνύεται στο όνομα του αρχείου και χωρισμένα μεταξύ τους με κόμματα. Από το παραπάνω αρχείο λαμβάνει ο εκάστοτε k-NN γραμμή προς γραμμή το χρήστη και την ταινία (χωρίς όμως το αντίστοιχο rating) και προχωρά σε υπολογισμό πρόβλεψης της βαθμολογίας που θα βάλει ο χρήστης αυτός στη συγκεκριμένη ταινία. Η διαδικασία αυτή

συνεχίζεται μέχρι να διαβαστούν όλα τα ζευγάρια (χρήστης, ταινία) του αρχείου και να αποθηκευτούν οι προβλέψεις του εκάστοτε k-NN σε txt. Κατόπιν, με χρήση του Excel υπολογίζεται η ακρίβεια κάθε αλγορίθμου. Τα αποτελέσματα παρατίθενται στον παρακάτω πίνακα όπου κάθε αλγόριθμος εκπροσωπείται από το όνομα της κλάσης που τον υλοποιεί:

	RMSE	**MAE**	**MPE**
KNNController	0.5110	0.4025	26.9%
KNNSimilarController	0.4171	0.2473	39.2%
KNNSimilarRatedController	0.2015	0.1407	-15.6%

Πίνακας 5.1: Ακρίβεια πρόβλεψης αρχικών βελτιστοποιημένων αλγορίθμων

Συνεπώς, βέλτιστος αλγόριθμος σε επίπεδο ακρίβειας είναι αυτός που υλοποιείται από την κλάση KNNSimilarRatedController.

5.3.2 Σύγκριση Καταναλισκόμενου Χρόνου σε Συναλλαγές με τη Βάση Δεδομένων

Οι παραπάνω κλάσεις διαφέρουν μεταξύ τους όχι μόνο σε επίπεδο ακρίβειας, αλλά και σε επίπεδο χρόνου. Καθεμία από αυτές, όπως αναφέρθηκε στις ενότητες 4.2.1, 4.2.2, 4.2.3, διαφέρει στον αριθμό των προσβάσεων στη βάση δεδομένων και στην πολυπλοκότητα των queries προς αυτή. Προφανώς, οι συγκρίσεις αναφέρονται στα διαφορετικά queries των κλάσεων και όχι σε όσα έχουν κοινά. Πιο συγκεκριμένα, η κλάση KNNController χρησιμοποιεί ένα query για να επιστρέψει μια λίστα με όλους τους χρήστες της βάσης (μέθοδος all() του PersonRepository), 618 queries για να επιστραφούν οι λίστες με τα κοινά asset μεταξύ του χρήστη που μας ενδιαφέρει και καθενός από τους υπόλοιπους 618 χρήστες (μέθοδος getUserCorrelation (String personId, String person2Id) του PersonRepository) και 10 queries για να επιστραφεί η βαθμολογία ενός χρήστη για μια ταινία (μέθοδος getUserRating (String assetId,String personId) του PersonRepository). Συνεπώς, χρησιμοποιεί 1+618+10 = 629 σχετικά απλά queries.

Η κλάση KNNSimilarController χρησιμοποιεί 1 σύνθετο query για να επιστραφεί μια λίστα με τους k πιο similar χρήστες με το χρήστη που μας ενδιαφέρει (μέθοδος getMostSimilar(String personId, Integer k) του PersonRepository) και 10 queries για να επιστραφεί η βαθμολογία ενός χρήστη για μια ταινία (μέθοδος getUserRating (String assetId,String personId) του PersonRepository). Επομένως, εκτελούνται 1+10 = 11 queries.

Τέλος, η κλάση KNNSimilarRatedController χρησιμοποιεί 1 ακόμα πιο σύνθετο query για να επιστραφεί μια λίστα με τους k πιο similar χρήστες με το χρήστη που μας ενδιαφέρει που έχουν βαθμολογήσει την επιθυμητή ταινία (μέθοδος

getMostSimilarAndRated(String personId, String assetId, Integer k) του PersonRepository). Ωστόσο, δε χρειάζεται κάποιο επιπλέον query για τον υπολογισμό της πρόβλεψης της προτίμησης του χρήστη. Επομένως, διαμορφώνεται ο ακόλουθος πίνακας:

	Συνολικός Αριθμός Queries
KNNController	629
KNNSimilarController	11
KNNSimilarRatedController	1

Πίνακας 5.2: Πλήθος queries αλγορίθμων

Τονίζεται ότι στον παραπάνω πίνακα δεν λαμβάνονται υπόψη τα queries που υπάρχουν κοινά και στις τρεις κλάσεις. Με τη σύγκριση του μέσου χρόνου απόκρισης (Average Response Time) των παραπάνω κλάσεων διαπιστώνεται αν είναι προτιμότερο να πραγματοποιούνται πολλές και απλές προσβάσεις στη βάση ή λίγες αλλά σύνθετες. Χάριν ευκολίας η αναφορά στο εκάστοτε query γίνεται με το όνομα της μεθόδου του PersonRepository που το αντιπροσωπεύει.

Για τον υπολογισμό του Average Response Time απαιτείται η χρήση του Apache JMeter για κάθε διαφορετικό query. Για κάθε 1 φορά που εκτελείται κάποιο query στον k-NN, θα εκτελείται 1,000 φορές στο JMeter, ώστε να προκύψει ένας αξιόπιστος μέσος όρος για το Average Response Time.

Για το query που αντιστοιχεί στη μέθοδο all() πραγματοποιούνται οι εξής ρυθμίσεις στο JMeter:

Στο Thread Group:

– Loop Count: 1000

- HTTP Request:
  – Path: /db/data/transaction/
  – Body Data:

```
{
  "statements": [
    {
      "statement": "MATCH (n:Person) RETURN n",
      "parameters": null,
      "resultDataContents": [
        "row",
        "graph"
      ],
      "includeStats": true
    }
  ]
}
```

Listing 5.2: Body Data στο JMeter για το query που αντιστοιχεί στη μέθοδο all()

- HTTP Request Defaults:
  – Server Name or IP: localhost
  – Port Number: 7474
- HTTP Header Manager:
  – Headers Stored in the Header Manager:

Name:	Value
Content-Type	application/json;charset=utf-8
Accept	application/json, text/plain, */*

Πίνακας 5.3: Headers Stored in the Header Manager στο JMeter για τη μέθοδο all()

- View Results Tree
- Response Time Graph
- Graph Results
- Aggregate Graph:
  – Filename: C:\Users\Venia\Desktop\all.csv

Για τον υπολογισμό του Average Response Time του query που αντιστοιχεί στη μέθοδο getUserCorrelation(String personId, String person2Id) οι ρυθμίσεις του JMeter είναι αντίστοιχες με προηγουμένως, με τη διαφορά ότι ο αριθμός επαναλήψεων (Loop Count) είναι 618,000, εφόσον το query αυτό εκτελείται 618 φορές στον k-NN. Ακόμη, το Body Data και το Filename προσαρμόζονται κατάλληλα, ώστε να αναφέρονται στο query αυτό. Τέλος, προστίθενται τα εξής στο Thread Group για την είσοδο των δεδομένων που απαιτούνται για την εκτέλεση του query:

- CSV Data Set Config:
  – Filename: C:\Users\Venia\Desktop\personIds.csv
  – Variable Names (comma-delimited): personId
  – Allow quoted data?: False
  – Recycle on EOF?: True
  – Stop thread on EOF?: False

Ακόμη, για να υπολογιστεί το Average Response Time του query που αντιστοιχεί στη μέθοδο getUserRating(String assetId,String personId) οι ρυθμίσεις του JMeter είναι αντίστοιχες με αυτές για το query που αντιστοιχεί στη μέθοδο all(), με τη διαφορά ότι ο αριθμός επαναλήψεων (Loop Count) είναι 10,000, αφού το query αυτό εκτελείται 10 φορές στον k-NN. Ακόμη, το Body Data και το Filename προσαρμόζονται κατάλληλα, ώστε να αναφέρονται στο query αυτό.

Για τον υπολογισμό του Average Response Time του query που αντιστοιχεί στη μέθοδο getMostSimilar(String personId, Integer k), αλλά και αυτού που αντιστοιχεί στη μέθοδο

getMostSimilarAndRated(String personId, String assetId, Integer k) οι ρυθμίσεις του JMeter είναι αντίστοιχες με αυτές για το query που αντιστοιχεί στη μέθοδο all(), με τη διαφορά ότι το Body Data και το Filename προσαρμόζονται κατάλληλα, ώστε να αναφέρονται στο καθένα από τα queries αυτά.

Από όλα τα παραπάνω, προκύπτουν τα εξής Average Response Time για το συνολικό αριθμό εκτελέσεων κάθε query στον k-NN:

	Αριθμός Εκτελέσεων Query	**Average Response Time (ms)**
all()	1	189
getUserCorrelation(String personId, String person2Id)	618	4528
getUserRating(String assetId,String personId)	10	46
getMostSimilar(String personId, Integer k)	1	7
getMostSimilarAndRated(String personId, String assetId, Integer k)	1	10

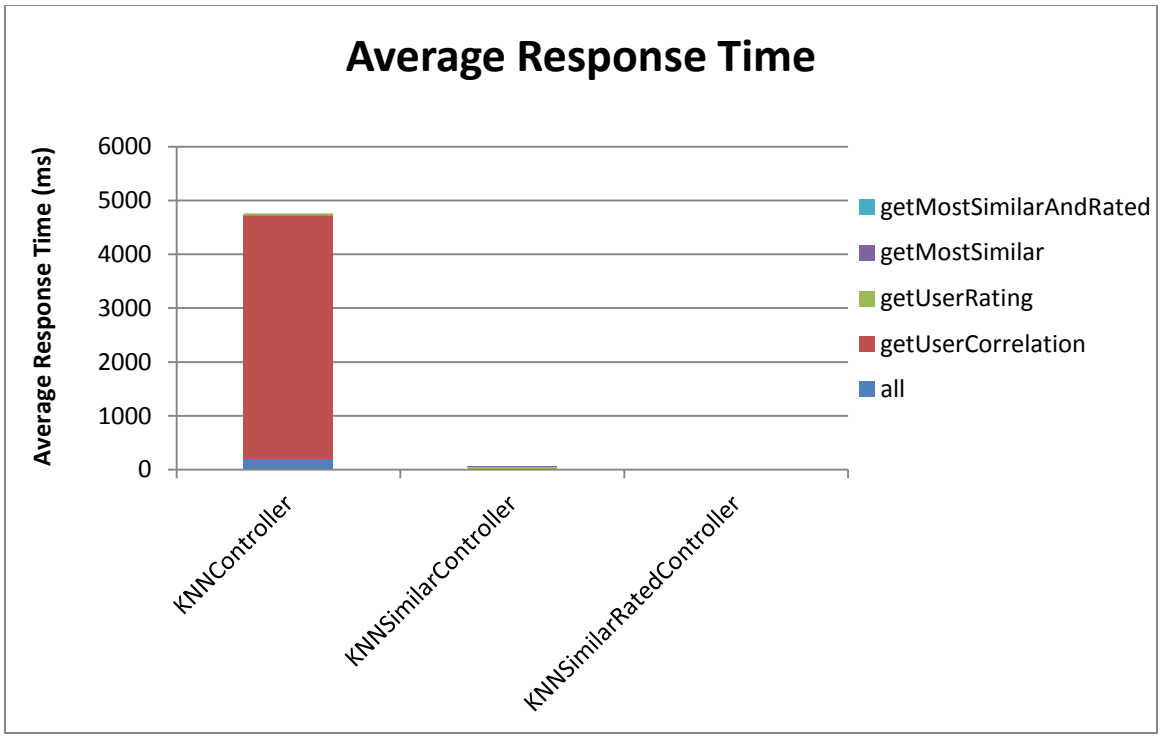
Πίνακας 5.4: Average Response Time και αριθμός εκτελέσεων για κάθε query

Βάσει του Πίνακα 5.4 προκύπτει το Average Response Time που απαιτείται για το σύνολο των queries των αλγορίθμων που υλοποιούνται στις κλάσεις KNNController, KNNSimilarController, KNNSimilarRatedController. Έτσι διαμορφώνεται ο παρακάτω πίνακας:

	Average Response Time (ms)
KNNController	4763
KNNSimilarController	53
KNNSimilarRatedController	10

Πίνακας 5.5: Συνολικό Average Response Time των queries κάθε αλγορίθμου

Από τα παραπάνω προκύπτει ότι παρόλο που το query του KNNSimilarRatedController είναι πολύ σύνθετο, εκμηδενίζει το overhead των πολλών συναλλαγών (transactions) με τη βάση δεδομένων. Τα προαναφερθέντα παρουσιάζονται και γραφικά:



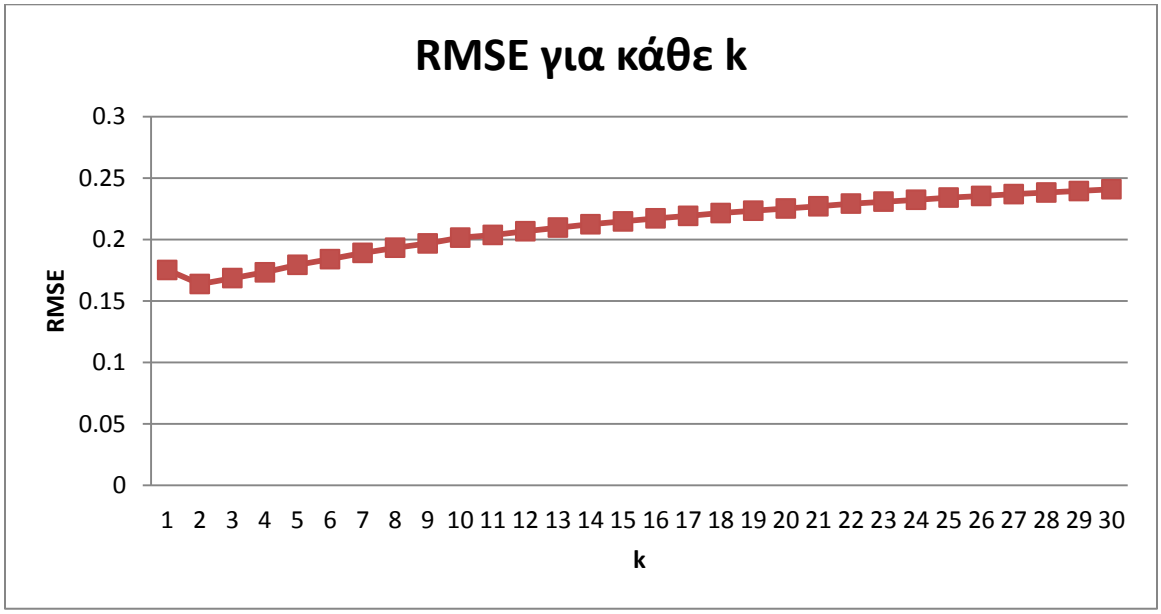
Σχήμα 5.2: Συνολικό Average Response Time των queries κάθε αλγόριθμου

Συνεπώς, και από πλευράς χρόνου (Average Response Time) καλύτερα αποτελέσματα προκύπτουν για τον αλγόριθμο που υλοποιείται από την κλάση KNNSimilarRatedController.

5.4 Σύγκριση Αποτελεσμάτων για διάφορες τιμές του k

Όπως αναφέρθηκε στην ενότητα 4.2.4, αναζητούμε τη βέλτιστη από πλευράς ακρίβειας τιμή για την παράμετρο k του k-NN αλγορίθμου. Ένα λογικό - για τη βάση δεδομένων που διατίθεται - εύρος τιμών εντός του οποίου αναζητείται το βέλτιστο k είναι οι ακέραιες τιμές του διαστήματος [1,30]. Για να εντοπιστεί το βέλτιστο αυτό k υπολογίζεται το RMSE που προκύπτει κάθε φορά για τα δεδομένα του training set. Κατόπιν, για το k που δίνει το μικρότερο RMSE, ελέγχονται τα αποτελέσματα που προκύπτουν για τα δεδομένα του test set. Αν, λοιπόν, το RMSE που υπολογίζεται για τα δεδομένα του test set δε διαφέρει πολύ από εκείνο που προκύπτει για τα δεδομένα του training set, τότε θεωρείται ότι έχει περιοριστεί ο κίνδυνος του overfitting και γίνεται αποδεκτό ως βέλτιστο το k αυτό.

Στην ακόλουθη γραφική παράσταση παρουσιάζονται για τα διάφορα k τα αντίστοιχα RMSE που προκύπτουν για τα δεδομένα του training set:



Σχήμα 5.3: RMSE του training set για κάθε k

Όπως φαίνεται στο Σχήμα 5.3 η τιμή του k για την οποία προκύπτει ελάχιστο RMSE για τα δεδομένα του training set είναι το 2 και ισχύει $RMSE_{tr(k=2)} = 0.1638$. Για την τιμή αυτή το RMSE που προκύπτει για τα δεδομένα του test set είναι $RMSE_{test(k=2)} = 0.1675$. Ωστόσο, δε χρησιμοποιείται ως βέλτιστη η τιμή k=2, εφόσον είναι πολύ μικρή και δεν μπορεί να θεωρηθεί αντιπροσωπευτική και αξιόπιστη για την πραγματοποίηση μελλοντικών προβλέψεων. Στο αποτέλεσμα αυτό, όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, συντελεί το γεγονός ότι η βάση δε διαθέτει μεγάλο όγκο δεδομένων τη συγκεκριμένη χρονική στιγμή, αλλά και η ενδεχόμενη ύπαρξη gray και black sheep στο σύστημα. Συνεπώς, υιοθετείται η αρχική τιμή k=10, εφόσον και για αυτήν παρουσιάζεται μικρό RMSE ($RMSE_{(k=10)} = 0.2015$) και, κατόπιν, απομονώνονται ή ελαττώνεται η επίδραση των χρηστών αυτών, ώστε να εξάγονται όσο το δυνατόν ακριβέστερες προβλέψεις ακόμα και μετά από εμπλουτισμό της βάσης μακροπρόθεσμα.

5.5 Σύγκριση Αλγορίθμων Χειρισμού Χρηστών με Ιδιάζουσα Συμπεριφορά

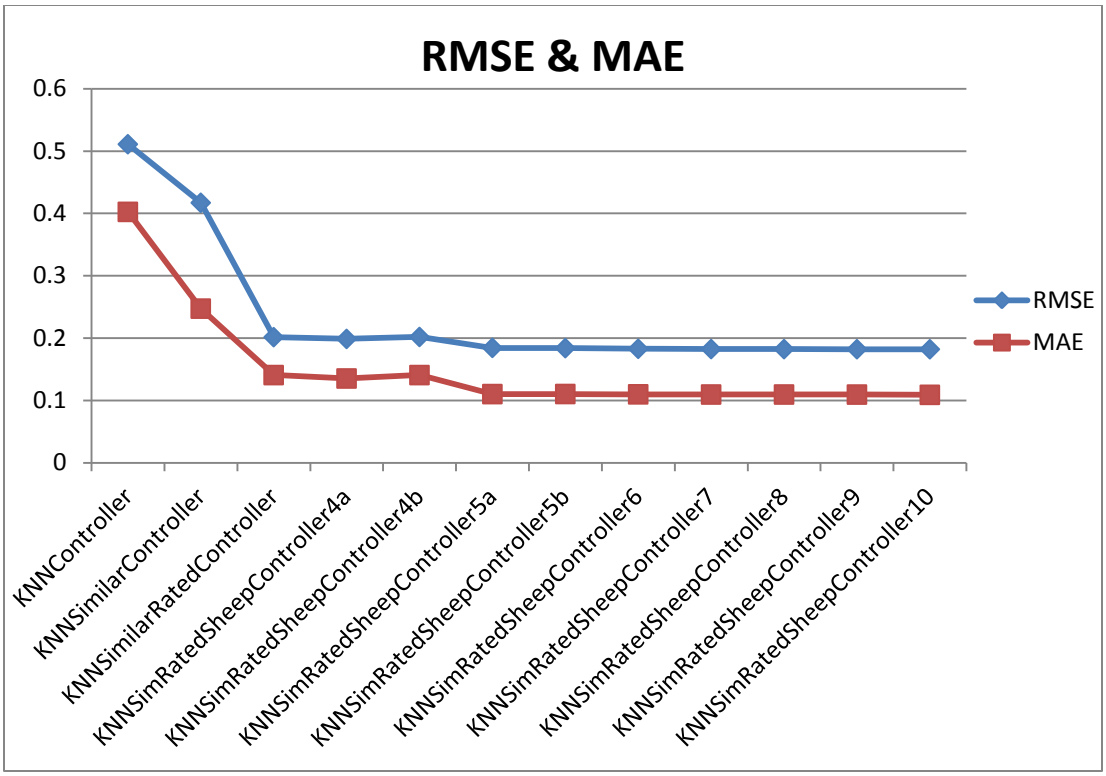
Ο Πίνακας 5.1 εμπλουτίζεται με τα RMSE, MAE, MPE των αλγορίθμων που χρησιμοποιήθηκαν για την αντιμετώπιση χρηστών ιδιάζουσας συμπεριφοράς:

	RMSE	**MAE**	**MPE**
KNNController	0.5110	0.4025	26.9%
KNNSimilarController	0.4171	0.2473	39.2%
KNNSimilarRatedController	0.2015	0.1407	-15.6%
KNNSimRatedSheepController4a	0.1987	0.1352	-13.8%
KNNSimRatedSheepController4b	0.2018	0.1407	-15.3%
KNNSimRatedSheepController5a	0.1843	0.1102	-10.9%
KNNSimRatedSheepController5b	0.1839	0.1103	-11.3%
KNNSimRatedSheepController6	0.1828	0.1098	-11.5%
KNNSimRatedSheepController7	0.1824	0.1095	-11.3%
KNNSimRatedSheepController8	0.1823	0.1095	-11.3%
KNNSimRatedSheepController9	0.1822	0.1094	-11.3%
KNNSimRatedSheepController10	0.1821	0.1090	-11.4%

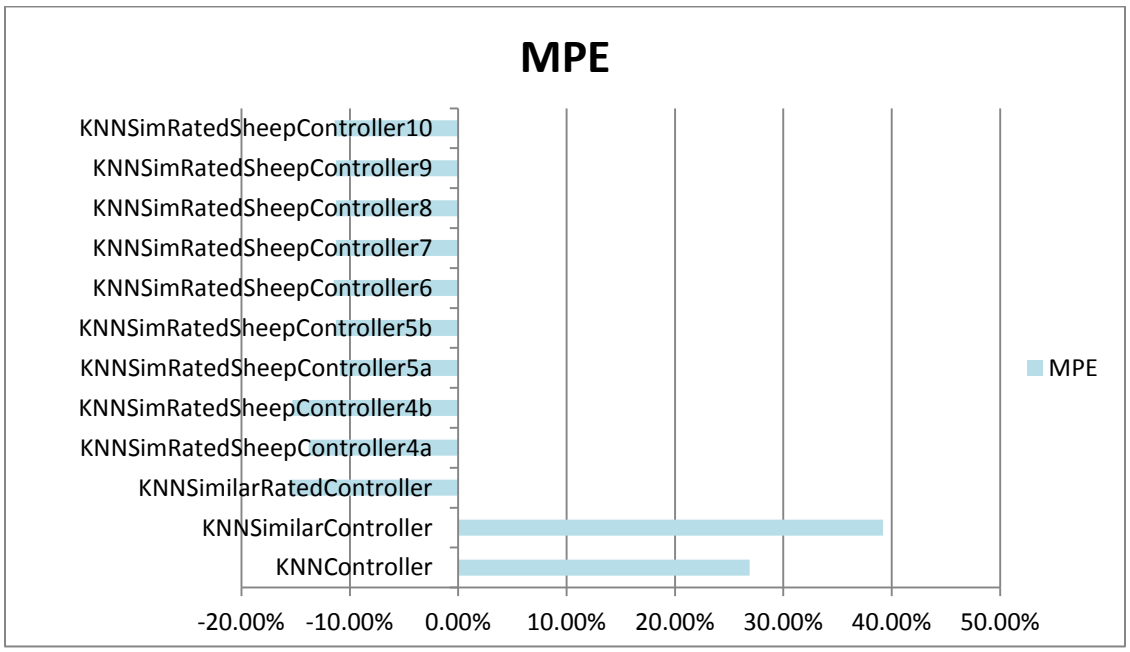
Πίνακας 5.6: Ακρίβεια πρόβλεψης αρχικού και βελτιστοποιημένων αλγορίθμων

Οι διαφορές που παρουσιάζονται μεταξύ των RMSE (καθώς και MAE, MPE) από ένα σημείο και μετά είναι μικρές, γεγονός που οφείλεται στο ότι η βάση δεν περιέχει μεγάλο όγκο δεδομένων, ώστε να φανεί σε μεγάλη έκταση η βελτίωση λόγω του διαφορετικού τρόπου αντιμετώπισης χρηστών με ιδιάζουσα συμπεριφορά. Αξίζει να αναφερθεί ότι το βασικό μέτρο ακρίβειας για τον εντοπισμό του βέλτιστου αλγορίθμου είναι το RMSE, καθώς δίνει ιδιαίτερη βαρύτητα στην ύπαρξη μεγάλων αποκλίσεων. Εφόσον είναι επιθυμητή η αποφυγή αυτών, επιλέγεται να ελαχιστοποιηθεί όσο γίνεται το RMSE, ακόμα και αν κάποιο εκ των δυο άλλων μέτρων υπολογισμού ακρίβειας παρουσιάζεται λίγο αυξημένο.

Παρακάτω παρουσιάζεται γραφικά το περιεχόμενο του Πίνακα 5.6:



Σχήμα 5.4: RMSE και MAE αρχικού και βελτιστοποιημένων αλγορίθμων



Σχήμα 5.5: MPE αρχικού και βελτιστοποιημένων αλγορίθμων

Στο Σχήμα 5.4 παρουσιάζεται η πορεία των RMSE και MAE, καθώς εξελίσσεται και τροποποιείται ο αρχικός αλγόριθμος. Στο Σχήμα 5.5 παριστάνεται το MPE όλων των σταδίων εξέλιξης του αρχικού αλγόριθμου στον τελικό. Συνεπώς, βέλτιστος αλγόριθμος είναι αυτός που υλοποιείται στην κλάση KNNSimRatedSheepController10.

Ο καταναλισκόμενος σε συναλλαγές με τη βάση δεδομένων χρόνος παρουσιάζει αμελητέα διαφορά, εφόσον η μόνη αλλαγή είναι ότι το query που αντιστοιχεί στη μέθοδο getMostSimilarAndRated(String personId, String assetId, Integer k) της κλάσης KNNSimilarRatedController αντικαθίσταται από το query που αντιστοιχεί στη μέθοδο getInfoMostSimilarAndRated(String personId, String assetId, Integer k) που χρησιμοποιούν όλες οι επόμενες κλάσεις. Η μόνη διαφορά των queries αυτών είναι ότι το δεύτερο επιστρέφει κάποια επιπλέον πεδία του χρήστη.

5.6 Αξιολόγηση τελικού βελτιστοποιημένου Αλγορίθμου

Ο τελικός βελτιστοποιημένος αλγόριθμος υλοποιείται στην κλάση KNNSimRatedSheepController10 και στις υποενότητες που ακολουθούν συγκρίνεται σε επίπεδο ακρίβειας με τους αρχικούς που διατίθενται, αλλά και με τα αποτελέσματα δημοφιλών machine learning αλγορίθμων που παρατίθενται στο [45]. Επιπροσθέτως, συγκρίνεται η μέση χρονική απόκριση του τελικού αλγορίθμου με την αντίστοιχη των αρχικών.

5.6.1 Σύγκριση Ακρίβειας τελικού βελτιστοποιημένου Αλγορίθμου με αρχικούς και με δημοφιλείς machine learning Αλγορίθμους

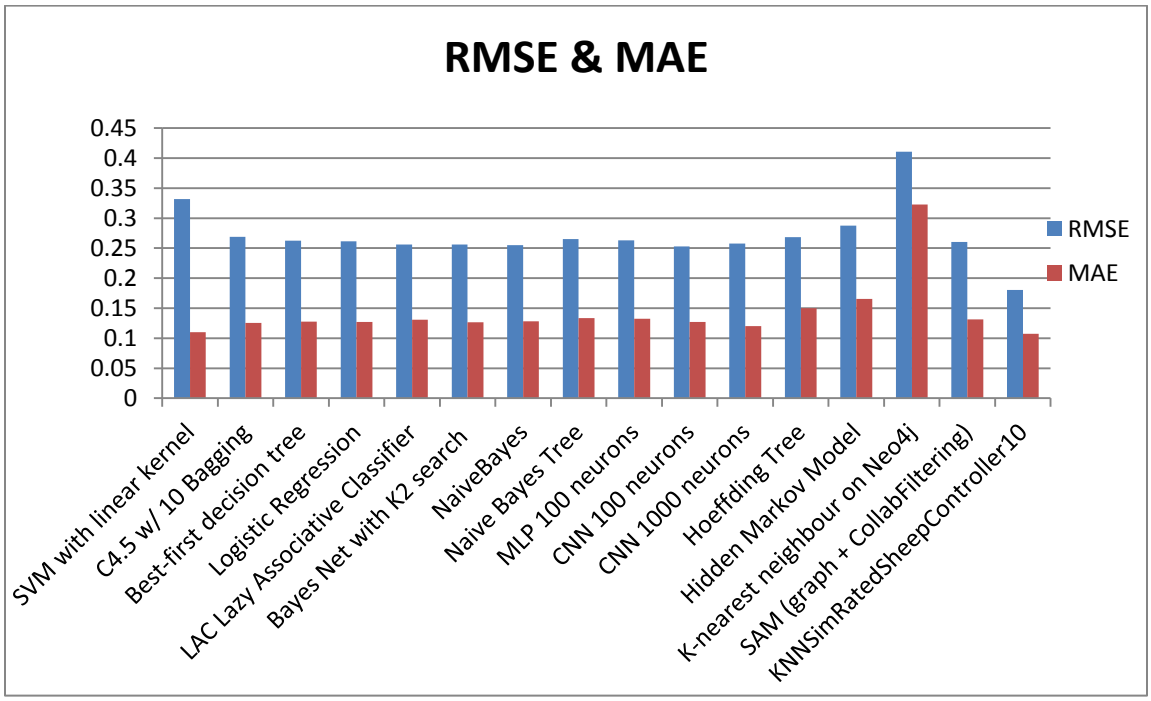
Ο πίνακας που παρατίθεται στο [45] παρουσιάζει την ακρίβεια και τα σφάλματα state-of-the-art machine learning αλγορίθμων, αλλά και τα αντίστοιχα των αλγορίθμων που χρησιμοποιούνται στο SAM project. Τα αποτελέσματα αυτά υπολογίστηκαν μόνο με τη χρήση του training set και εμπλουτίζονται με τα αντίστοιχα του τελικού αλγορίθμου.

Έτσι προκύπτει ο ακόλουθος πίνακας:

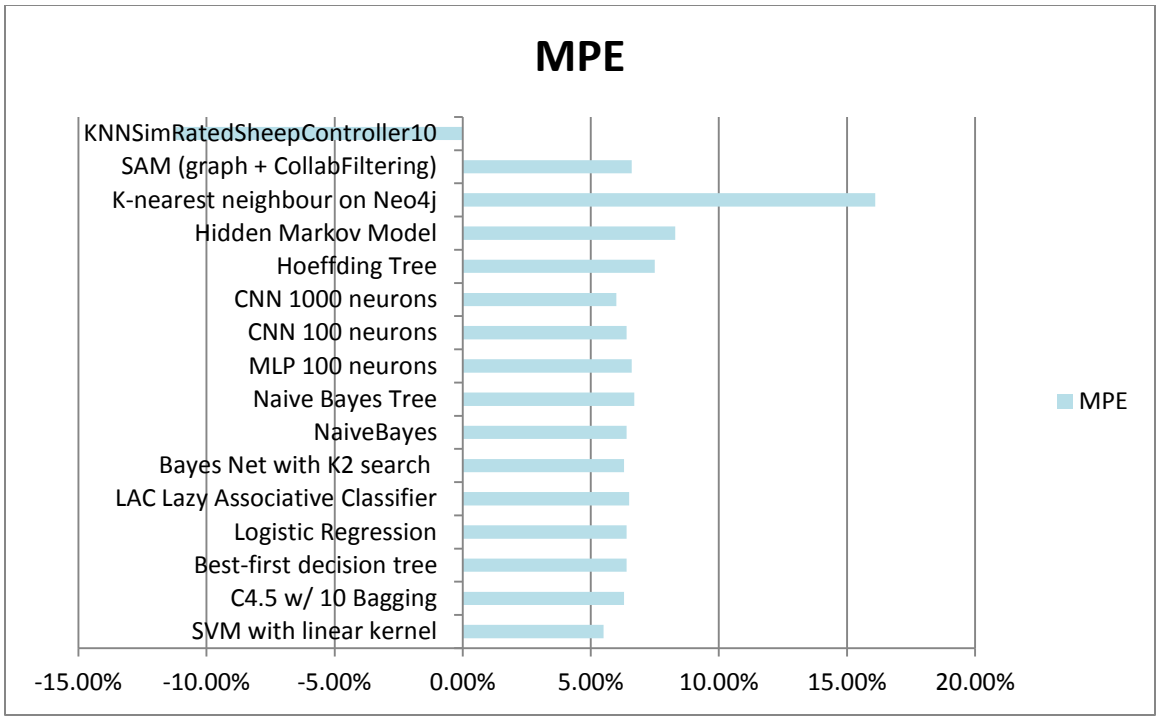
	RMSE	MAE	MPE
SVM with linear kernel	0.3315	0.1099	5.5%
C4.5 w/ 10 Bagging	0.2688	0.1253	6.3%
Best-first decision tree	0.2626	0.1274	6.4%
Logistic Regression	0.2612	0.1269	6.4%
LAC Lazy Associative Classifier	0.2563	0.1306	6.5%
Bayes Net with K2 search	0.2559	0.1263	6.3%
NaiveBayes	0.2551	0.1283	6.4%
Naive Bayes Tree	0.2652	0.1334	6.7%
MLP 100 neurons	0.2632	0.1324	6.6%
CNN 100 neurons	0.2530	0.1269	6.4%
CNN 1000 neurons	0.2578	0.1202	6.0%
Hoeffding Tree	0.2684	0.1499	7.5%
Hidden Markov Model	0.2875	0.1653	8.3%
K-nearest neighbour on Neo4j	0.4108	0.3226	16.1%
SAM (graph + CollabFiltering)	0.2604	0.1312	6.6%
KNNSimRatedSheepController10	0.1802	0.1075	-11.2%

Πίνακας 5.7: Ακρίβεια πρόβλεψης τελικού βελτιστοποιημένου αλγορίθμου, αρχικών και δημοφιλών machine learning αλγορίθμων

Ακολούθως, παρουσιάζονται γραφικά όσα αναφέρονται στον Πίνακα 5.7:



Σχήμα 5.6: RMSE και MAE τελικού βελτιστοποιημένου, αρχικών και δημοφιλών machine learning αλγορίθμων



Σχήμα 5.7: MPE τελικού βελτιστοποιημένου, αρχικών και δημοφιλών machine learning αλγορίθμων

Όπως προκύπτει από τον παραπάνω πίνακα ο τελικός βελτιστοποιημένος αλγόριθμος παρουσιάζει μικρότερο RMSE και MAE σε σύγκριση με τους αρχικούς και με δημοφιλείς machine learning αλγορίθμους. Το MPE είναι λίγο μεγαλύτερο (το πρόσημο σχετίζεται με την ερμηνεία του MPE, όχι με το μέγεθός του) σε σχέση με αυτό των περισσότερων άλλων αλγορίθμων, αλλά, όπως αναφέρθηκε, οι συγκρίσεις πραγματοποιούνται προσδίδοντας μεγαλύτερη βαρύτητα στο RMSE, καθώς είναι επιθυμητός ο περιορισμός των μεγάλων σφαλμάτων στις προβλέψεις των προτιμήσεων των χρηστών.

Επιπροσθέτως, προκύπτει $RMSE_{test} = 0.1864$, δηλαδή το RMSE του test set είναι εξίσου μικρό με αυτό του training set. Το RMSE που προκύπτει τελικά για όλη τη βάση είναι RMSE = 0.1821, όπως αναφέρθηκε και σε προηγούμενη ενότητα. Συνεπώς, η ακρίβεια του k-NN αλγορίθμου έχει σημειώσει σημαντική βελτίωση με την κλάση KNNSimRatedSheepController10.

5.6.2 Σύγκριση Μέσου Χρόνου Απόκρισης τελικού βελτιστοποιημένου Αλγορίθμου με αρχικούς

Για την πλήρη αξιολόγηση του τελικού αλγορίθμου, απαιτείται η σύγκρισή του με τους αρχικούς και από πλευράς χρόνου. Πιο συγκεκριμένα, με τη χρήση του Apache JMeter υπολογίζεται το Average Response Time του αρχικού k-NN και Collaborative Filtering, αλλά και του τελικού βελτιστοποιημένου k-NN αλγορίθμου. Οι πραγματοποιούμενες στο Apache JMeter ρυθμίσεις για τον υπολογισμό της μέσης χρονικής απόκρισης του αρχικού k-NN είναι οι εξής:

Στο Thread Group:

– Loop Count: 1000

- HTTP Request:
  – Path: /knn/${personId}/${assetId}
- HTTP Request Defaults:
  – Server Name or IP: localhost
  – Port Number: 8080
- HTTP Header Manager:
  – Headers Stored in the Header Manager:

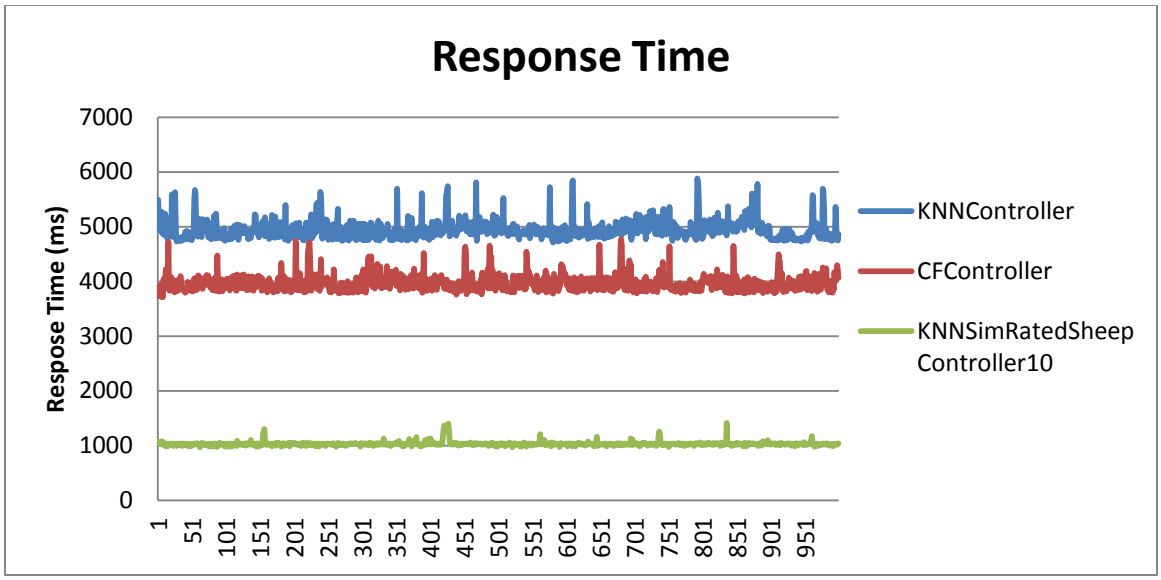
Name:	Value
Content-Type	application/json
Accept	application/json;charset=UTF-8

Πίνακας 5.8: Headers Stored in the Header Manager στο JMeter για αρχικό k-NN

- CSV Data Set Config:
  – Filename: C:\Users\Venia\Desktop\personId_assetId.csv
  – Variable Names (comma-delimited): personId,assetId
  – Allow quoted data?: True
  – Recycle on EOF?: True
  – Stop thread on EOF?: False
- View Results Tree
- Response Time Graph
- Graph Results
- Aggregate Graph:
  – Filename: C:\Users\Venia\Desktop\knn1_results.csv

Για τον υπολογισμό του Average Response Time του Collaborative Filtering αλγορίθμου που δίνεται στην κλάση CFController, αλλά και του τελικού βελτιστοποιημένου k-NN αλγορίθμου της κλάσης KNNSimRatedSheepController10 οι ρυθμίσεις του JMeter είναι αντίστοιχες με τις προαναφερθείσες, με τη διαφορά ότι το Path στο HTTP Request και το Filename στο Aggregate Graph προσαρμόζονται κατάλληλα.

Ακολούθως αποτυπώνεται γραφικά το Response Time για 1000 εκτελέσεις των αλγορίθμων από το Apache JMeter:



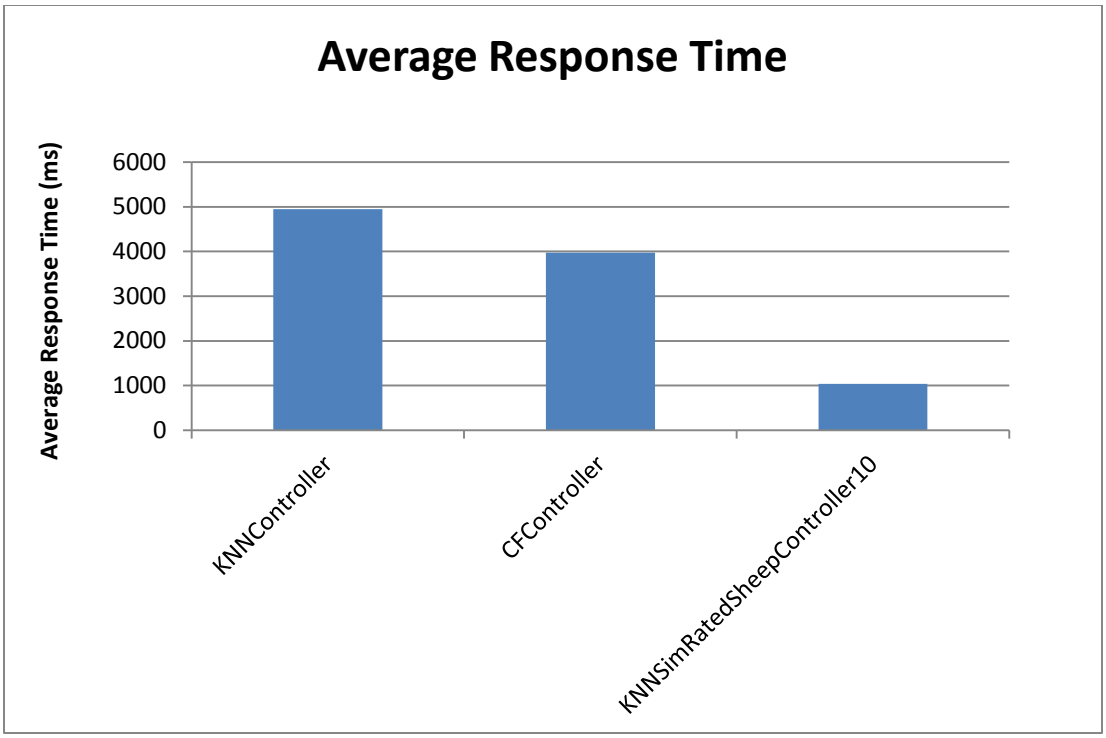
Σχήμα 5.8: Response Time αρχικών και τελικού βελτιστοποιημένου αλγορίθμου σε 1000 εκτελέσεις

Τα αποτελέσματα που προκύπτουν για το Average Response Time παρατίθενται ακολούθως:

	Average Response Time (ms)
KNNController	4949
CFController	3971
KNNSimRatedSheepController10	1036

Πίνακας 5.9: Average Response Time αρχικών και τελικού βελτιστοποιημένου αλγορίθμου

Ο Πίνακας 5.9 παρουσιάζεται κα γραφικά:



Σχήμα 5.9: Average Response Time αρχικών και τελικού βελτιστοποιημένου αλγορίθμου

Συνεπώς, και από πλευράς χρόνου συμφέρει η χρήση του αλγορίθμου που αναπτύσσεται στην κλάση KNNSimRatedSheepController10. Μάλιστα, ο αλγόριθμος αυτός είναι περίπου 4 φορές γρηγορότερος από τους αρχικούς.

Κεφάλαιο 6

Συμπεράσματα και Προοπτικές για Μελλοντική Έρευνα

6.1 Σύνοψη - Συμπεράσματα

Με τον αλγόριθμο που αναπτύσσεται και τον εμπλουτισμό της βάσης δεδομένων με επιπλέον χρήσιμες πληροφορίες επιτυγχάνεται η πρόβλεψη των προτιμήσεων των χρηστών με μεγαλύτερη ακρίβεια και ορθότητα και ο περιορισμός των σφαλμάτων. Ακόμη, ο καταναλισκόμενος σε συναλλαγές με τη βάση δεδομένων χρόνος, καθώς και ο αριθμός των queries που εκτελούνται περιορίζονται σημαντικά. Κατ' επέκταση, ελαττώνεται σε μεγάλο βαθμό η μέση χρονική απόκριση του αλγορίθμου και, πιο συγκεκριμένα, ο τελικός k-NN αλγόριθμος είναι περίπου 4 φορές ταχύτερος από τους αρχικούς k-NN και Collaborative Filtering αλγορίθμους (Σχήμα 5.9).

Ακρογωνιαίο λίθο στην επίτευξη των προαναφερθέντων αποτελούν οι αλλαγές στον τρόπο υπολογισμού της ομοιότητας μεταξύ δυο χρηστών, αλλά και της πρόβλεψης της προτίμησης ενός χρήστη. Με τον εντοπισμό και ειδικό χειρισμό των περιπτώσεων και βαθμολογικών μοτίβων για τα οποία δεν υπήρχε, αρχικά, η δυνατότητα αντιπροσωπευτικής αποτύπωσης της ομοιότητας δημιουργούνται οι βάσεις για την πραγματοποίηση ακριβέστερων προβλέψεων. Οι αλλαγές στον τύπο υπολογισμού της παραγόμενης από τον αλγόριθμο πρόβλεψης έχει αντίστοιχα αποτελέσματα. Η ακρίβεια των προβλέψεων βελτιώνεται περαιτέρω με τον ξεχωριστό χειρισμό ιδιαίτερων συμπεριφορών χρηστών. Η αναγνώριση τέτοιων συμπεριφορών επιτυγχάνεται με τη συμβολή κάποιων επιπλέον πληροφοριών σχετικών με τους χρήστες και τις ταινίες, οι οποίες επαναϋπολογίζονται και αποθηκεύονται εκ νέου στη βάση, όποτε κρίνεται απαραίτητο. Η σπουδαιότητα του ειδικού χειρισμού τέτοιων στοιχείων καταδεικνύεται ακόμα περισσότερο με τον περαιτέρω εμπλουτισμό της βάσης δεδομένων μακροπρόθεσμα. Εξάγεται, λοιπόν, το συμπέρασμα ότι σε επίπεδο ακρίβειας καθοριστική είναι η όσο το δυνατόν καλύτερη αποτύπωση των συσχετίσεων μεταξύ των δεδομένων, αλλά και η χρήση κατάλληλων τύπων για τον υπολογισμό των προβλεπόμενων τιμών. Ακόμα, στα περισσότερα πραγματικά συστήματα παρατηρούνται στοιχεία που λειτουργούν αποπροσανατολιστικά, σαν "θόρυβος", και επιδρούν αρνητικά στην εξαγωγή ορθών αποτελεσμάτων. Ο περιορισμός ή - σε κάποιες περιπτώσεις - η εξάλειψη της επίδρασης τέτοιων στοιχείων, μέσω του ιδιαίτερου χειρισμού τους, αποτελεί καθοριστικό παράγοντα για τη βελτίωση της ποιότητας των προβλέψεων.

Σπουδαίο ρόλο στη μείωση της μέσης χρονικής απόκρισης του αλγορίθμου και στη βελτίωση της ταχύτητας του διαδραματίζει η αποφυγή περιττών υπολογισμών, επανάληψης υπολογισμών που έχουν ήδη πραγματοποιηθεί παλαιότερα, η δρομολόγηση πολλών εξ αυτών offline από το σύστημα, όποτε κρίνεται σκόπιμο, και η αποθήκευσή τους στη βάση δεδομένων για μελλοντική χρήση. Γενικά, λοιπόν, βασικός κρίνεται ο περιορισμός των υπολογισμών που απαιτούνται κάθε φορά για την παραγωγή on demand προβλέψεων, όπου αυτό είναι δυνατό. Ιδιαίτερα σημαντική προς αυτήν την κατεύθυνση

θεωρείται, επίσης, η ελάττωση του αριθμού των προσβάσεων στη βάση δεδομένων, εφόσον διαπιστώνεται ότι μεγάλο μέρος του χρόνου καταναλώνεται σε queries επικοινωνίας με τη βάση.

Με την εφαρμογή όλων των προαναφερθέντων επιτυγχάνεται η βελτιστοποίηση - τόσο σε επίπεδο ακρίβειας όσο και σε επίπεδο χρόνου - αλγορίθμων πρόβλεψης των προτιμήσεων των χρηστών.

6.2 Προοπτικές για Μελλοντική Έρευνα

Στο πλαίσιο της εξέλιξης και περαιτέρω ανάπτυξης της εργασίας προτείνονται ορισμένες προοπτικές προς μελλοντική εξέταση:

- Για τον έλεγχο των τροποποιήσεων του συστήματος θα μπορούσε να χρησιμοποιηθεί κάποια cross-validation μέθοδος (όπως η 10-fold cross-validation) αντί των κλασικών training (70% του συνόλου των δεδομένων) και test set (30% του συνόλου των δεδομένων), ειδικά σε περίπτωση εμπλουτισμού της βάσης δεδομένων.

- Για την παραγωγή προβλέψεων θα μπορούσαν να λαμβάνονται υπόψη και διάφορα user interactions. Όπως αναφέρθηκε, για τη δημιουργία προβλέψεων χρησιμοποιήθηκαν οι βαθμολογίες που προέκυψαν μέσω sentiment analysis από σχόλια χρηστών σχετικά με τις ταινίες. Θα ήταν δυνατό να χρησιμοποιηθούν, επίσης, τα like/dislike σε ταινία ή σε σχετικό widget (εφόσον, όπως αναφέρθηκε, το SAM project αφορά σε περιβάλλον second screen), καθώς και το dismiss ή show more για σχετικό widget. Προφανώς, όλα αυτά θα αντιστοιχίζονταν στη χρησιμοποιούμενη βαθμολογική κλίμακα και θα τους προσδιδόταν κάποιο βάρος ανάλογα με τη σημασία τους, ώστε μετά από συμψηφισμό τους να αντικατοπτρίζουν πιο ολοκληρωμένα την προτίμηση του εκάστοτε χρήστη και να συντελούν στην παραγωγή ακριβέστερων και πιο αξιόπιστων προβλέψεων.

- Σε περαιτέρω βελτίωση των αποτελεσμάτων ενδέχεται να συμβάλλει ο συνδυασμός του βελτιστοποιημένου k-NN με άλλους αλγορίθμους πρόβλεψης προτιμήσεων.

Βιβλιογραφία

[1] "SAM – Socialising Around Media." SAM Socialising Around Media. N.p., n.d. Web. <http://samproject.net/>.

[2] "Recommender system." *Wikipedia*. Wikimedia Foundation, 05 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Recommender_system>.

[3] "Introduction to approaches and algorithms." *Recommender systems, Part 1*. N.p., 12 Dec. 2013. Web. <https://www.ibm.com/developerworks/library/os-recommender1/>.

[4] "Recommendation systems: Principles, methods and evaluation." *Recommendation systems: Principles, methods and evaluation*. N.p., n.d. Web. <http://www.sciencedirect.com/science/article/pii/S1110866515000341?np=y>.

[5] Terveen, Loren, and Will Hill. "Beyond recommender systems: Helping people help each other." *HCI in the New Millennium* 1.2001 (2001): 487-509.

[6] Melville, Prem, and Vikas Sindhwani. "Recommender systems." *Encyclopedia of machine learning*. Springer US, 2011. 829-838.

[7] Lee, Joonseok, Mingxuan Sun, and Guy Lebanon. "A comparative study of collaborative filtering algorithms." *arXiv preprint arXiv:1205.3193* (2012).

[8] "Collaborative Filtering Recommender Systems – Benefits and Disadvantages." *Recommender Systems / Recommendation engines explained*. N.p., n.d. Web. <http://recommender.no/info/collaborative-filtering-approach/>.

[9] "Collaborative filtering." *Wikipedia*. Wikimedia Foundation, 17 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Collaborative_filtering>.

[10] "Algorithms." *Algorithms :: Model-based Algorithms*. N.p., n.d. Web. <http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/modelbased.html>.

[11] "K-nearest neighbors algorithm." *Wikipedia*. Wikimedia Foundation, 10 Feb. 2017. Web. <https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm>.

[12] "Feature vector." *Wikipedia*. Wikimedia Foundation, 20 Jan. 2017. Web. <https://en.wikipedia.org/wiki/Feature_vector>.

[13] "Singular value decomposition." *Wikipedia*. Wikimedia Foundation, 13 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Singular_value_decomposition>.

[14] Sarwar, Badrul, et al. "Application of Dimensionality Reduction in Recommender System-A Case Study." (2000).

[15] "Test set." *Wikipedia*. Wikimedia Foundation, 06 Jan. 2017. Web. <https://en.wikipedia.org/wiki/Test_set>.

[16] Lakshmi, Soanpet Sree, and T. Adi Lakshmi. "Recommendation Systems: Issues and challenges." *IJCSIT) International Journal of Computer Science and Information Technologies* 5.4 (2014): 5771-5772.

[17] "Cold start." *Wikipedia*. Wikimedia Foundation, 01 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Cold_start>.

[18] Ghazanfar, Mustansar Ali, and Adam Prugel-Bennett. "Fulfilling the Needs of Gray-Sheep Users in Recommender Systems, A Clustering Solution."

[19] Solanki, Shano. *Recommender System using Collaborative Filtering and Demographic*. Diss. THAPAR UNIVERSITY PATIALA, 2015.

[20] Zuva, Tranos, et al. "A survey of recommender systems techniques challenges and evaluation metrics." *International Journal of Emerging Technology and Advanced Engineering* 2.11 (2012): 382-386.

[21] "Overfitting." *Wikipedia*. Wikimedia Foundation, 31 Jan. 2017. Web. <https://en.wikipedia.org/wiki/Overfitting>.

[22] "Similarity measure." *Wikipedia*. Wikimedia Foundation, 17 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Similarity_measure>.

[23] "Euclidean distance." *Wikipedia*. Wikimedia Foundation, 16 Jan. 2017. Web. <https://en.wikipedia.org/wiki/Euclidean_distance>.

[24] Sondur, Mr Sridhar Dilip, Mr Amit P. Chigadani, and Shantharam Nayak. "Similarity Measures for Recommender Systems: A Comparative Study." *Journal for Research| Volume* 2.03 (2016).

[25] "Cosine similarity." *Wikipedia*. Wikimedia Foundation, 12 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Cosine_similarity>.

[26] "Pearson correlation coefficient." *Wikipedia*. Wikimedia Foundation, 17 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Pearson_correlation_coefficient>.

[27] Rafter, Rachael. *Evaluation and conversation in collaborative filtering*. University College Dublin, 2010.

[28] Levinas, Claudio Adrian. "An Analysis of Memory Based Collaborative Filtering Recommender Systems with Improvement Proposals." (2014).

[29] Adomavicius, Gediminas, and Alexander Tuzhilin. "Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions."

[30] "Artificial Intelligence in Theory and Practice III." *Google Books*. N.p., n.d. Web. <https://books.google.gr/books?id=2l6rCAAAQBAJ>.

[31] "Neo4j." *Wikipedia*. Wikimedia Foundation, 19 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Neo4j>.

[32] "Chapter 1. Introduction." *Chapter 1. Introduction - The Neo4j Developer Manual v3.1*. N.p., n.d. Web. <https://neo4j.com/docs/developer-manual/current/introduction/>.

[33] "Chapter 3. Cypher." *Chapter 3. Cypher - The Neo4j Developer Manual v3.1*. N.p., n.d. Web. <https://neo4j.com/docs/developer-manual/current/cypher/>.

[34] "Spring Framework." *Wikipedia*. Wikimedia Foundation, 18 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Spring_Framework>.

[35] "2. Introduction to the Spring Framework." *2. Introduction to the Spring Framework*. N.p., n.d. Web. <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html>.

[36] "Cross-cutting concern." *Wikipedia*. Wikimedia Foundation, 31 Jan. 2017. Web. <https://en.wikipedia.org/wiki/Cross-cutting_concern>.

[37] "32. Messaging." *32. Messaging*. N.p., n.d. Web. <http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-messaging.html>.

[38] "Gradle." *Wikipedia*. Wikimedia Foundation, 21 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Gradle>.

[39] "Accelerate developer productivity." *Gradle Build Tool*. N.p., n.d. Web. <https://gradle.org/>.

[40] "Root-mean-square deviation." *Wikipedia*. Wikimedia Foundation, 16 Nov. 2016. Web. <https://en.wikipedia.org/wiki/Root-mean-square_deviation>.

[41] "Mean absolute error." *Wikipedia*. Wikimedia Foundation, 30 Dec. 2016. Web. <https://en.wikipedia.org/wiki/Mean_absolute_error>.

[42] "Mean percentage error." *Wikipedia*. Wikimedia Foundation, 27 Nov. 2016. Web. <https://en.wikipedia.org/wiki/Mean_percentage_error>.

[43] "Apache JMeter." *Wikipedia*. Wikimedia Foundation, 18 Feb. 2017. Web. <https://en.wikipedia.org/wiki/Apache_JMeter>.

[44] "Apache JMeter™." *Apache JMeter - Apache JMeter™*. N.p., n.d. Web. <http://jmeter.apache.org/index.html>.

[45] Fotis Aisopos, Angelos Valsamis, Alexandros Psychas, Andreas Menychtas and Theodora Varvarigou, "Efficient Context Management and Personalized User Recommendations in a Smart Social TV environment", GECON2016 Conference, Athens, 2016