INTER-FACULTY MASTER PROGRAM on
COMPLEX SYSTEMS and NETWORKS
SCHOOL of MATHEMATICS
SCHOOL of BIOLOGY
SCHOOL of GEOLOGY
SCHOOL of ECONOMICS
ARISTOTLE UNIVERSITY of THESSALONIKI
http://cosynet.auth.gr/

MASTER THESIS

Analysis of recommender system algorithms in the Movielens dataset

Ανάλυση συστημάτων αυτόματων συστάσεων στο σύνολο δεδομένων Movielens

Anastasia Foudouli

Supervisor: Bratsas Charalampos, AUTH

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ στα
ΠΟΛΥΠΛΟΚΑ ΣΥΣΤΗΜΑΤΑ και ΔΙΚΤΥΑ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΤΜΗΜΑ ΒΙΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΓΕΩΛΟΓΙΑΣ
ΤΜΗΜΑ ΟΙΚΟΝΟΜΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
http://cosynet.auth.gr/

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάλυση συστημάτων αυτόματων συστάσεων στο σύνολο δεδομένων Movielens

Analysis of recommender system algorithms in the Movielens dataset

Αναστασία Φουδούλη

Επιβλέπων: Μπράτσας Χαράλαμπος, ΑΠΘ

Εγκρίθηκε από την Τριμελή Εξεταστική Επιτροπή την …. Δεκεμβρίου 2019.

………………………	………………………	………………………
Χ. Μπράτσας	Ν. Φαρμάκης	Ι. Αντωνίου
Μέλος ΕΔΙΠ, Α.Π.Θ.	Αν. Καθηγητής Α.Π.Θ.	Καθηγητής, Α.Π.Θ.

Θεσσαλονίκη, Δεκέμβριος 2019

…………………………………………..
Αναστασία Φουδούλη
Πτυχιούχος Μαθηματικός Α.Π.Θ.

Abstract

Recommender systems encompass a class of techniques and algorithms which are able to suggest "relevant" items to users. Ideally, the suggested items are as relevant to the user as possible, so that the user can engage with those items: YouTube videos, news articles, online products, and so on.Items are ranked according to their relevancy, and the most relevant ones are shown to the user. The relevancy is something that the recommender system must determine and is mainly based on historical data.

Recommender systems are generally divided into two main categories: collaborative filtering and content-based systems. In this thesis are focusing on Collaborative Filtering algorithms. In the first part, and introduction to recommendation engines, classification of recommender systems algorithms and evaluation metrics are defined. In chapter 2, the dataset is presented and alterations that were made to the original data due to limitation of the machine that run the algorithms. In chapter 3, a variety of algorithms that generate item to item recommendations is presented. These algorithms work solely on the interaction matrix and transformation that are based on natural language preprocessing techniques. In the 4th and 5th part, three algorithms that predict user – item rating are created. The first two of them, alternating least squares (ALS) and singular vale decomposition (SVD) are matrix factorization techniques and the last one is a recommender system based on a neural network of latent features. The evolution of errors as the training iterations pass is also presented.

Key Words

Recommender System, Collaborative Filtering, Alternating Least Squares, Singular Value Decomposition, Latent Features, Embeddings, Neural Networks, Item to item recommendations

Contents

Σύνοψη

Συστήματα συστάσεων

Τα συστήματα αυτόματων συστάσεων ή Recommender systems, είναι αλγόριθμοι που ο κύριος σκοπός τους είναι η δημιουργία μιας λίστας προς πρόταση, νέων αντικειμένων στους χρήστες του συστήματος. Τα αντικείμενα που προτείνονται θα πρέπει να συνάδουν με τις προτιμήσεις του κάθε χρήστη γι'αυτό και συχνά τέτοια συστήματα έχουν σας στόχο να προβλέψουν τον βαθμό που θα αρέσει το κάθε προϊόν ενός καταλόγου σε κάθε χρήστη του συστήματος ώστε να προτείνουν αυτά με τα υψηλότερα σκορ.

Η έρευνα στον τομέα των Recommender systems ξεκίνησε τα μέσα της δεκαετίας 1990 και έκτοτε τα συστήματα αυτά έχουν εξελιχθεί και έχουν διαχωριστεί μεταξύ τους ώστε να παρέχουν τις καλύτερες συστάσεις. Πρακτικά, πρόκειται για αλγορίθμους φιλτραρίσματος πληροφορίας οι οποίοι μπορούν να κατηγοριοποιηθούν σε τρείς κλάσεις, τεχνικές συνεργατικού φιλτραρίσματος (Collaborative filtering), τεχνικές φιλτραρίσματος με βάση το περιεχόμενο (content based filtering) καθώς και υβριδικές τεχνικές μεταξύ των δύο προαναφερθέντων τεχνικών. Οι τεχνικές αυτές συνεχίζουν να εξελίσσονται διαρκώς ώστε να παρέχουν καλύτερες συστάσεις με την τάση που επικρατεί να είναι η προσθήκη νέας πληροφορίας στο σύστημα, όπως πληροφορία από κοινωνικά δίκτυα, ροές δεδομένων του ίντερνετ των πραγμάτων (internet of things), γεωγραφικές πληροφορίες κ.α..

Ο τομέας των Recommender systems συνεχίζει να είναι υψίστου ενδιαφέροντος καθώς μπορεί να υπάρξει πληθώρα πρακτικών εφαρμογών. Οι αλγόριθμοι αυτοί μπορούν να εφαρμοστούν στην αναζήτηση στο διαδίκτυο, στις προτάσεις βιβλίων, ταινιών, μουσικής, paper, ειδήσεων κλπ.. Επακολούθως, αν και η ανάπτυξη των συστημάτων αυτών ξεκίνησε στον ακαδημαϊκό κλάδο, πλέον υπάρχει μία παράλληλη ροή βελτιστοποίησης των αλγορίθμων αυτών στον εταιρικό τομέα.

Είναι λοιπόν εμφανές ότι είναι δύο οι κύριοι στόχοι των συστημάτων αυτόματων συστάσεων. Ο πρώτος, ο οποίος και απευθύνεται στους χρήστες μια πλατφόρμας όπου γίνεται χρήση ενός τέτοιου συστήματος, είναι να αυξηθεί η ικανοποίηση του χρήστη μέσω της χρήσης της πλατφόρμας. Όσον αφορά τα πλεονεκτήματα που απορρέουν από τη ενσωμάτωση τέτοιων συστημάτων από εταιρίες, αυτές αυξάνουν την κερδοφορία τους μέσω αύξησης των πωλήσεων. Υπάρχει πολύ δυνατή συσχέτιση μεταξύ των δύο αυτών στόχων, άλλωστε δεδομένου ότι οι προτάσεις των νέων προϊόντων γίνονται στους καταναλωτές, βελτιστοποίηση της λίστας των προτεινόμενων προϊόντων, οδηγεί σε ικανοποίηση του πελάτη, περισσότερη κατανάλωση και αύξηση των κερδών της εταιρίας. Τα συστήματα αυτά έχουν ευρεία χρήση σε καταστήματα ηλεκτρονικού εμπορίου μετατρέποντας χρήστες της σελίδας σε αγοραστές, αυξάνοντας την

αφοσίωσή τους (loyalty) στο συγκεκριμένο site παρέχοντάς τους την δυνατότητα αγορών σε λιγότερα κλικ και παρέχοντας καλύτερες προσφορές σε συχνούς πελάτες.

Υπάρχει πληθώρα συστημάτων αυτοματοποιημένων συστάσεων. Κάθε ένα τέτοιο σύστημα αν και υποχρεούται να ακολουθεί βασικούς κανόνες, προσαρμόζεται στις ανάγκες των πελατών μιας εταιρίας καθώς διαφορετικά προβλήματα απαιτούν διαφορετικές λύσεις. Τα παραδείγματα που ακολουθούν δίνουν τη σφαιρική εικόνα τριών τέτοιων συστημάτων.

- Σύστημα προτάσεων βιβλίων
  Εδώ, ο στόχος του συστήματος είναι η προβολή νέων τίτλων στους χρήστες. Έτσι το ενδιαφέρων του χρήστη παραμένει αμείωτο και η κατανάλωσή του στη πλατφόρμα αυξάνεται.
- Σύστημα προτάσεων μετοχών
  Στο συγκεκριμένο παράδειγμα, μια λίστα των πιο κερδοφόρων μετοχών παρουσιάζεται στον χρήστη της πλατφόρμας. Εδώ, η προτάσεις νέων μετοχών δεν είναι απαιτούμενο του συστήματος. Μετοχές που έχουν ήδη αγοραστεί στο παρελθόν από τους χρήστες μπορούν να προταθούν ξανά, εφόσον αυτές παραμένουν κερδοφόρες
- Σύστημα προτάσεων προϊόντων
  Στο συγκεκριμένο σύστημα, πρέπει να υπάρχει ισορροπία στις συστάσεις μεταξύ νέων προϊόντων και προϊόντων που έχουν ήδη αγοραστεί παλαιότερα από τον πελάτη. Η συχνότητα με την οποία προϊόντα βρίσκονται σε παλαιότερα καλάθια ενός πελάτη, μπορεί να χρησιμοποιηθεί ώστε να γίνει επανασύστασή τους. Παρόλα αυτά, οι προτάσεις νέων προϊόντων είναι εξίσου σημαντικές, ειδάλλως το σύστημα κινδυνεύει να χάσει τον πελάτη, ο οποίος μπορεί να χάσει το ενδιαφέρον του.

Τα συστήματα αυτόματων συστάσεων μπορούν κατηγοριοποιηθούν σε τρεις μεγάλες κατηγορίες. Η κατηγοριοποίηση αυτή βασίζεται στους αλγορίθμους που χρησιμοποιούνται καθώς και την πληροφορία που χρειάζεται για την υλοποίησή τους. Τα βασικά είδη είναι τα παρακάτω:

- Collaborative filtering – «Συνεργατικό» φιλτράρισμα
- Content based filtering - φιλτραρίσματος με βάση το περιεχόμενο
- Hybrid systems – υβριδικά συστήματα

Collaborative Filtering

Οι μέθοδοι collaborative filtering για τα συστήματα αυτομάτων συστάσεων, είναι μέθοδοι που βασίζονται μόνο στις αλληλεπιδράσεις που είχε ένας χρήστης με τα προϊόντα στο παρελθόν για να παρέχουν νέες συστάσεις. Σε αυτές τις μεθόδους οι προτάσεις βασίζονται στην ανάλυση των παρελθοντικών δράσεων και τις προτιμήσεις των χρηστών καθώς και την ομοιότητα αυτών με άλλους χρήστες του συστήματος. Πρακτικά, ακολουθείται η παραδοχή ότι χρήστες που έχουν επιδείξει την ίδια συμπεριφορά στο παρελθόν, είναι πιθανό να συνεχίσουν να έχουν παρόμοια συμπεριφορά και στο μέλλον. Έτσι, αντικείμενα που βρίσκονται στη λίστα αντικειμένων με τα οποία έχει αλληλοεπιδράσει ένας χρήστης μπορούν να προταθούν σε έναν άλλο και αντίθετα.

Οι μέθοδοι collaborative filtering, μπορούν να διαχωριστούν επιπλέον σε άλλες υποκατηγορίες, με βάση τον αλγόριθμο που χρησιμοποιείται.

- Memory based τεχνικές
  Στην συγκεκριμένη υποκατηγορία βρίσκουμε τους lazy learners. Εδώ, δεν υπάρχει η παραδοχή για συγκεκριμένο μοντέλο και ο αλγόριθμος δουλεύει απευθείας με τις τιμές της αλληλεπίδρασης. Τέτοιες τεχνικές βασίζονται σε αναζητήσεις κοντινότερου γείτονα, βρίσκοντας τους πιο «συγγενικούς» χρήστες για κάθε χρήστη του συστήματος και προτείνοντας τα πιο δημοφιλή αντικείμενα της γειτονιάς που δημιουργείται.
- Model based τεχνικές
  Πρόκειται για την αντίθετη τεχνική, όπου πλέον υπάρχει η παραδοχή για ένα υποκείμενο γενετικό μοντέλο το οποίο έχει ως στόχο να «εξηγήσει» τον τρόπο με τον οποίο πραγματοποιούνται οι αλληλεπιδράσεις μεταξύ χρήστη – προϊόντος.

Οι memory based τεχνικές μπορούν να κατηγοριοποιηθούν επιπλέον σε user-user (χρήστης – προς – χρήστη) και item – item (προϊόν – προς – προϊόν) collaborative filtering.

- User – User collaborative filtering
  Σε αυτή τη μέθοδο γίνεται αναπαράσταση των χρηστών του συστήματος σε διανύσματα όπου καταγράφεται η αλληλεπίδραση τους με τα προϊόντα του συστήματος. Στη συνέχεια, η απόσταση μεταξύ των χρηστών υπολογίζεται. Σκοπός αυτών των αλγορίθμων είναι η εύρεση του πιο δυνατόν παρόμοιου χρήστη για κάθε έναν χρήστη του συστήματος και η προτάσεις νέων προϊόντων σε αυτούς, που βασίζονται στις προτιμήσεις του «παρόμοιου» χρήστη. Πρόκειται για μία τεχνική η οποία αν και έχει καλά αποτελέσματα, είναι συνήθως πολύ δαπανηρή υπολογιστικά, καθώς πρέπει να υπολογιστούν αποστάσεις μεταξύ κάθε δυνατού συνδυασμού χρηστών.
  Όπως ήδη αναφέρθηκε, αρχικά κάθε χρήστης μεταφράζεται στο διάνυσμα των αλληλεπιδράσεων του με τα διαφορετικά αντικείμενα του συστήματος. Στη συνέχεια μία μετρική ομοιότητας υπολογίζεται μεταξύ του χρήστη που μας ενδιαφέρει και όλους τους υπόλοιπους χρήστες του συνόλου δεδομένων. Η μετρική αυτή πρέπει να μπορεί να διακρίνει ως κοντινούς χρήστες, αυτούς με παρεμφερείς αλληλεπιδράσεις στο ίδιο σύνολο αντικειμένων. Στη συνέχεια, αφότου υπολογιστούν οι ομοιότητες, η πιο διαδεδομένη τεχνική είναι η διατήρηση των κ – κοντινότερων γειτόνων του χρήση και η πρόταση των πιο διαδεδομένων προϊόντων – συνήθως από τη λίστα των αντικειμένων με τα οποία δεν έχει αλληλοεπιδράσει ακόμη ο χρήστης – στη γειτονιά που δημιουργείται.
- Item – Item collaborative filtering

Σε αυτή την μέθοδο τα προϊόντα είναι αυτά που μεταφράζονται σε διανύσματα με τιμές που αφορούν την αλληλεπίδραση των χρηστών με αυτά και στη συνέχεια η απόστασή μεταξύ των αντικειμένων του συνόλου δεδομένων υπολογίζεται. Δύο προϊόντα θεωρούνται παρόμοια όταν οι περισσότεροι χρήστες που έχουν αλληλεπιδράσεις και με τα δύο, το έκαναν με παρόμοιο τρόπο. Για να παραχθούν προτάσεις προϊόντων σε έναν χρήστη, πρώτα κατασκευάζεται η λίστα με τα αντικείμενα με τον υψηλότερο βαθμό αλληλεπίδρασης και η ομοιότητα αυτών των αντικειμένων με τα υπόλοιπα προϊόντα του συνόλου δεδομένων υπολογίζεται. Στη συνέχεια δημιουργούνται γειτονιές κ – κοντινότερων γειτόνων για τα επιλεγμένα αντικείμενα και αυτά προτείνονται στον χρήστη.

Στις model – based τεχνικές βρίσκονται και οι μέθοδοι παραγοντοποίησης πινάκων (matrix factorization).

- Παραγοντοποίηση πινάκων
  Στην περίπτωση των συστημάτων συστάσεων, πρόκειται για τεχνικές παραγοντοποίησης πινάκων που προσπαθούν να παραγοντοποιήσουν έναν μεγάλο και πολύ αραιό πίνακα σε γινόμενο δύο μικρότερων και μη – αραιών πινάκων. Οι πίνακες αυτοί είναι ο πίνακας χαρακτηριστικών των χρηστών, ο οποίος περιέχει την αναπαράσταση των χρηστών του συστήματος σε πυκνά διανύσματα και ο πίνακας των χαρακτηριστικών των αντικειμένων που περιέχει την αναπαράσταση των αντικειμένων. Η παραδοχή που γίνεται σε αυτή την τεχνική, είναι ότι υπάρχει μία τέτοια αναπαράσταση σε χώρο μικρότερης διάστασης και ότι η αναπαράσταση αυτή μπορεί να εξηγήσεις τόσο τις προτιμήσεις των χρηστών αλλά και τα χαρακτηριστικά των προϊόντων με τέτοιο τρόπο, ώστε υπολογίζοντας το διανυσματικό γινόμενο μεταξύ των διανυσμάτων χαρακτηριστικών χρηστών και προϊόντων να προκύπτει η αριθμητικοποίηση της αλληπίδρασής τους.
  Πρέπει να επισημανθεί ότι τα χαρακτηριστικά αυτά δεν έχουν καμία συσχέτιση με χαρακτηριστικά που δίνονται στο μοντέλο σε τεχνικές φιλτραρίσματος περιεχομένου. Αντ’αυτού, πρόκειται για χαρακτηριστικά τα οποία «ανακαλύπτονται» από το σύστημα. Μάλιστα, καθώς πρόκειται για χαρακτηριστικά που μαθαίνονται και όχι για χαρακτηριστικά που ορίζουν παραμέτρους των χρηστών ή των προϊόντων, η σχέση που αναπαριστούν είναι πολλές φορές αδύνατο να κατανοηθεί από τον άνθρωπο. Παρόλα αυτά, με την χρήση αυτών των τεχνικών, η αναπαράσταση του χώρου που προκύπτει, είναι κοντά σε αυτή που ένας άνθρωπος θα σκεφτόταν ενστικτωδώς. Έτσι λοιπόν, σε αυτόν τον νέο χώρο που δημιουργείται τόσο προϊόντα με παρόμοια «χαρακτηριστικά», όπως και χρήστες έχουν κοντινές αναπαραστάσεις.

Content based filtering

Τα συστήματα που είναι κατασκευασμένα με αυτή την τεχνική, βασίζονται στο φιλτράρισμα χαρακτηριστικών των προϊόντων αλλά και των χρηστών προκειμένου να κατασκευαστεί ένα μοντέλο συστάσεων. Για να εφαρμοστεί ένας τέτοιος αλγόριθμος, προαπαιτείτε ένα βήμα αρχικοποίησης του συστήματος, όπου δημιουργούνται τα προφίλ των χρηστών καθώς και των προϊόντων. Παραδείγματος χάριν με την εγγραφή σε μία πλατφόρμα, οι χρήστες μπορούν να δημιουργήσουν μία λίστα πραγμάτων που τους ενδιαφέρουν καθώς και να παρέχουν δημογραφικά χαρακτηριστικά που τους αφορούν. Στη συνέχεια το σύστημα συστάσεων που κατασκευάζεται μπορεί να προτείνει αντικείμενα σε χρήστες της πλατφόρμας χρησιμοποιώντας την πληροφορία που αφορά τα προφίλ των προϊόντων και των χρηστών.

Η παραδοχή του συστήματος είναι ότι ο χρήστης ενδιαφέρεται για προϊόντα τα οποία έχουν παρόμοια χαρακτηριστικά με αυτά που έχει αλληλοεπιδράσει στο παρελθόν. Τα αντικείμενα του συνόλου δεδομένων αναπαριστώντα στον χώρο των χαρακτηριστικών που τα διέπουν και ομοιότητες μεταξύ των προϊόντων υπολογίζονται στον χώρο αυτό. Έτσι νέες συστάσεις γίνονται προτείνοντας όμοια αντικείμενα.

Σε αυτή την τεχνική δεν μας ενδιαφέρουν οι προτιμήσεις γειτονικών χρηστών, για αυτό και δεν είναι απαραίτητο ένα μεγάλο σύνολο δεδομένων με αλληλεπιδράσεις του χρήστη με διαφορετικά προϊόντα ώστε να αυξηθεί η ακρίβεια των προτάσεων του συστήματος. Η μόνη απαίτηση του συστήματος είναι να υπάρχει ένας αριθμός παρελθοντικών αλληλεπιδράσεων καθώς και να έχουν οριστεί χαρακτηριστικά των αντικειμένων στο προφίλ των χρηστών.

Υβριδικές τεχνικές

Τα υβριδικά συστήματα συστάσεων, συνδυάζουν τις δύο παραπάνω τεχνικές ώστε να ενισχύσουν τα προτερήματα και να μειώσουν τα μειονεκτήματα της κάθε τεχνικής. Η πιο διαδεδομένη υβριδική τεχνική συνδυάζει μεθόδους collaborative filtering μαζί με κάποια πληροφορία που αφορά το περιεχόμενο των αντικειμένων και των χρηστών.

Προτερήματα και μειονεκτήματα των μεθόδων

1. Collaborative Filtering

Τα συστήματα συστάσεων που βασίζονται σε τεχνικές Collaborative Filtering (τόσο memory όσο και model based) δεν απαιτούν περεταίρω πληροφορία, πέρα από τα αντικείμενα, τους χρήστες και την αλληλεπίδραση μεταξύ τους. Για ένα σύστημα που ο αριθμός αντικειμένων και χρηστών δεν αυξάνονται, νέες αλληλεπιδράσεις που καταγράφονται σε βάθος χρόνου, εισάγουν νέες πληροφορίες στο σύστημα. Καθώς δημιουργούνται νέα παραδείγματα, μπορεί να αυξηθεί η αποτελεσματικότητα του συστήματος. Ωστόσο, η απουσία επιπλέον χαρακτηριστικών για τα αντικείμενα και τους χρήστες αποτελεί το κύριο μειονέκτημα της χρήσης τέτοιων αλγορίθμων. Οι αλγόριθμοι Collaborative Filtering υποφέρουν από μια μεγάλη πρόκληση, το «cold start problem». Είναι αδύνατον να προτείνουν προϊόντα σε νέους χρήστες, ή να προτείνουν αντικείμενα με πολύ χαμηλό αριθμό καταγεγραμμένων αλληλεπιδράσεων. Αυτό μπορεί να καταπολεμηθεί με απλές προσεγγίσεις, όπως η πρόταση τυχαίων αντικειμένων σε νέους χρήστες ή νέων αντικειμένων σε τυχαίους χρήστες (random strategy), ή πρόταση δημοφιλών αντικειμένων σε νέους χρήστες, ή νέων αντικειμένων στους πιο ενεργούς χρήστες (maximum expectation strategy). Η βέλτιστη προσέγγιση ωστόσο είναι η χρήση μιας non-collaborative μεθόδου στην αρχή του κύκλου ζωής των νέων χρηστών και αντικειμένων. Η μέθοδος user-user βασίζεται στην αναζήτηση παρόμοιων χρηστών σε ό,τι αφορά την αλληλεπίδρασή τους με τα αντικείμενα. Γενικά, κάθε χρήστης έχει αλληλοεπιδράσει με λίγα μόνο αντικείμενα. Αυτό καθιστά τη μέθοδο σχετικά ευαίσθητη σε κάθε καταγεγραμμένη δράση. Από την άλλη, εφόσον η τελική σύσταση βασίζεται αποκλειστικά στις καταχωρημένες αλληλεπιδράσεις χρηστών που θεωρούνται παρόμοιοι με το χρήστη που μας ενδιαφέρει, τα αποτελέσματα είναι πιο εξατομικευμένα.

Αντίθετα, η μέθοδος item-item βασίζεται στην αναζήτηση παρόμοιων αντικειμένων, σε ό,τι αφορά την αλληλεπίδραση χρήστη-αντικειμένου. Καθώς, σε γενικές γραμμές, πολλοί χρήστες έχουν αλληλοεπιδράσει με ένα συγκεκριμένο αντικείμενο, το σύστημα δεν επηρεάζεται σε μεγάλο βαθμό από μεμονωμένες αλληλεπιδράσεις. Στον αντίποδα, αλληλεπιδράσεις που προέρχονται απ' όλους τους χρήστες (ακόμα και απ' αυτούς με πολύ διαφορετικό ιστορικό από του χρήστη που μας ενδιαφέρει) λαμβάνονται υπόψη για τη σύσταση. Συνεπώς, η προσέγγιση αυτή είναι λιγότερο εξατομικευμένη από τη μέθοδο user-user, αλλά πιο εύρωστη.

2. Content based filtering

Το κύριο πλεονέκτημα της content based filtering προσέγγισης σε σχέση με τη collaborative filtering προσέγγιση, συνίσταται στο γεγονός ότι το «cold start problem» μπορεί να αποφευχθεί όταν τα αντικείμενα διαθέτουν επαρκείς περιγραφές. Επιπλέον, συστήματα συστάσεων που έχουν κατασκευαστεί με την παραπάνω τεχνική μπορούν να προτείνουν προϊόντα σε χρήστες με ιδιαίτερες προτιμήσεις, βρίσκοντας αντικείμενα που αντιστοιχίζονται σ' αυτά τα κριτήρια. Μη δημοφιλή και καινούρια αντικείμενα μπορούν να συσταθούν, καθώς οι συστάσεις εξαρτώνται αποκλειστικά από τα χαρακτηριστικά των αντικειμένων και των χρηστών. Τέλος, η αναπαράσταση του περιεχομένου επιτρέπει τη χρήση μιας πληθώρας προσεγγίσεων ανάλυσης, όπως τεχνικές επεξεργασίας κειμένου, σημασιολογική πληροφορία κτλ. Ωστόσο, η εύρεση των κατάλληλων χαρακτηριστικών μπορεί να αποδειχθεί δύσκολη (λ.χ. εικόνες). Επίσης, οι χρήστες σπανίως συμπληρώνουν πληροφορίες σχετικά με τις προτιμήσεις τους στις πλατφόρμες, επομένως οι συστάσεις σε νέους χρήστες είναι δύσκολες. Καταληκτικά, αυτά τα συστήματα τείνουν να υπέρ-εξειδικεύονται. Δε δύνανται να προτείνουν αντικείμενα εκτός του προφίλ του χρήστη, ενώ τα άτομα διαθέτουν συνήθως πιο περίπλοκα ενδιαφέροντα.

Αξιολόγηση συστημάτων συστάσεων

Τα συστήματα αυτοματοποιημένων προτάσεων είναι πολύ δημοφιλή τόσο στον τομέα των επιχειρήσεων, όσο και στον ερευνητικό, όπου πολλοί αλγόριθμοι έχουν αναπτυχθεί και προταθεί για την βελτίωση των συστάσεων που παράγονται. Αυτοί οι αλγόριθμοι καλούνται να ανταπεξέλθουν σε ποικίλες διεργασίες σε διαφορετικά περιβάλλοντα. Γι' αυτό το λόγο είναι πολύ σημαντικό, τόσο για ερευνητικούς αλλά και για πρακτικού σκοπούς, να μπορεί να επιλεχθεί ο κατάλληλος αλγόριθμος που θα ταιριάζει καλύτερα στην εκάστοτε περίσταση. Ο τρόπος με τον οποίο γίνεται αυτή η διαλογή είναι συγκρίνοντας ένα πλήθος αλγορίθμων χρησιμοποιώντας μετρικές αξιολόγησης.

Υπάρχουν πολλές μετρικές στην βιβλιογραφία που έχουν προταθεί για την σύγκριση τέτοιων συστημάτων. Για συστήματα πρόβλεψης βαθμολογίας ενός προϊόντος από το χρήστη, οι πιο κοινώς χρησιμοποιούμενες στη βιβλιογραφία μετρικές είναι το μέσο απόλυτο σφάλμα, μέσο τετραγωνικό σφάλμα και ρίζα του μέσου τετραγωνικού σφάλματος.

Άλλες συνήθεις μετρικές, που όμως δεν χρησιμοποιούνται σε συστήματα πρόβλεψης βαθμολόγησης, είναι οι precision και recall. Το precision μετράει το πόσο ικανό είναι το σύστημα να παράξει σχετική πληροφορία με τον ελάχιστο αριθμό προτάσεων. Το recall μετράει την ικανότητα του συστήματος να βρει όλα τα σχετιζόμενα στοιχεία και να τα προτείνει στον χρήστη. Οι μετρικές αυτές χρησιμοποιούνται ως μετρήσεις σφάλματος σε προβλήματα δυαδικής κατηγοριοποίησης όπου είναι δυνατή η μέτρηση των true positive (TP), true negative (TN), false positive (FP) and false negative (FN) προβλέψεων.

Για να μπορέσουν να χρησιμοποιηθούν αυτές οι μετρικές στο σύστημα προτάσεων θα πρέπει να γίνουν οι παρακάτω παραδοχές. Έστω ότι οι προβλέψεις των αξιολογήσεων ανήκουν σε ένα διάστημα [0.5, 5]. Μπορούμε να θεωρήσουμε τότε πώς οι τιμές [0.5, 3] αντιπροσωπεύουν αρνητικές αξιολογήσεις, ενώ οι τιμές [3.5, 5] θετικές. Τότε οι TP, TN, FP και FN μπορούνε να μετρηθούνε. Επειδή όμως υπάρχει μόνο ένα πολύ μικρό κομμάτι του συνόλου αλληλεπιδράσεων χρήστη-προϊόντος, προκύπτει ένα ακόμη πρόβλημα.

Γενικότερα, θα πρέπει να αναγνωστείτε το βασικό μειονέκτημα του να μετριέται η ποιότητα ενός συστήματος προτάσεων μόνο με την χρήση μετρικών σφάλματος. Ο σκοπός ενός συστήματος προτάσεων είναι να προτείνει στους χρήστες προϊόντα τα οποία θα τους αρέσουν. Το πρόβλημα όμως εγγυάται στο ότι είναι αδύνατο να αξιολογηθεί αυτό καθώς τα δεδομένα μας για την εκπαίδευση του συστήματος δεν μπορούν ποτέ να περιέχουν τα επιθυμητά αποτελέσματα. Για παράδειγμα, σε ένα σύστημα προτάσεων για ταινίες, ξέρουμε ότι στον χρήστη Bob αρέσουν οι ταινίες A, B, C, D και E. Χωρίζοντας τα δεδομένα σε σύνολο εκπαίδευσης A, B, C , D και σύνολο ελέγχου E, αξιολογούμε αν ο αλγόριθμος πρότεινε την ταινία E. Όμως ένας αλγόριθμος ο οποίος σαν πρόβλεψη θα έβγαζε F, Z ή W θα μπορούσε να ήταν ακόμη καλύτερος από αυτόν που πρότεινε Ε. Η τεχνικής αξιολόγησης μέσω μετρικών μπορεί να ισχύει μόνο στην περίπτωση που γνωρίζουμε όλα τα προϊόντα που αρέσουν σε κάθε χρήστη του συστήματος.

Στην παρούσα εργασία, στα συστήματα πρόβλεψης αξιολόγησης, χρησιμοποιούνται μόνο οι κλασικές μετρικές (MAE, MAE και RMSE).Σε όλες τις περιπτώσεις, υπολογίζεται μόνο το σφάλμα ανάμεσα στις προβλέψεις και τις γνωστές αλληλεπιδράσεις του χρήστη με το προϊόν.

Περιγραφή ενοτήτων

Στην παρούσα εργασία χρησιμοποιείται το σύνολο δεδομένων Movielens. Αυτό συγκεντρώνεται κάθε χρόνο από το movielens.org όπου χρήστες βαθμολογούν ένα σύνολο ταινιών με σκοπό την βελτίωση συστημάτων συστάσεων. Το σύνολο αποτελείται από περίπου 29 εκατομμύρια βαθμολογίες ταινιών από 0.5 αστέρια έως 5 και περιέχει ταινίες μέχρι το 2018. Καθώς τα συστήματα και όλοι οι υπολογισμοί που πραγματοποιήθηκαν έτρεξαν σε ένα μηχάνημα περιορισμένης μνήμης, το σύνολο αυτό μειώθηκε σε 24 εκατομμύρια βαθμολογίες.

Οι αλγόριθμοι που παρουσιάζονται μπορούν να διαχωριστούν σε τρείς μεγάλες κατηγορίες, οι οποίες είναι κατανεμημένες στα κεφάλαια 3 έως 5 της διπλωματικής.

Στο κεφάλαιο 3, βρίσκεται η περιγραφή συστημάτων συστάσεων ταινιών, όπου ο χρήστης μπορεί να εισάγει τον τίτλο μιας ταινίας και στη συνέχεια να λάβει συστάσεις από το σύστημα. Αρχικά χρησιμοποιείται ένας αλγόριθμος που εφαρμόζει τεχνικές ομοιότητας κειμένων ώστε να βρει τον τίτλο που ταιριάζει στον εισαχθέν τίτλο στο σύστημα, σε σχέση με όλες τις ταινίες που υπάρχουν στο σύνολο δεδομένων. Η τεχνική βασίζεται στην απόσταση Levenstein, η οποία πρακτικά μας δίνει τον αριθμό τον αλλαγών (μετακίνηση χαρακτήρων, εισαγωγή νέων χαρακτήρων ή αφαίρεση χαρακτήρων) που πρέπει να γίνουν σε μία πρόταση ώστε να είναι όμοια με την άλλη, πχ οι λέξεις «πρύμνη» και «πρίμα» έχουν απόσταση 3 χαρακτήρων.

Στη συνέχεια ορίζονται αλγόριθμοι ο οποίοι είναι βασισμένοι σε

- Ιδιότητες συνόλων
- cosine similarity
- δημιουργία γειτονιών
- τεχνικές ομοιότητας κειμένων

Όλες οι παραπάνω τεχνικές εφαρμόζονται δημιουργώντας διανύσματα για κάθε εγγεγραμμένο χρήστη του συστήματος με τιμές είτε τις βαθμολογίες που έχουν δώσει στις διάφορες ταινίες είτε θεωρώντας διανύσματα με άσσους, που δηλώνουν μονάχα την παρακολούθησε κάθε ταινίας.

Συγκεκριμένα, τα συστήματα συνόλων, βασίζονται στην παραδοχή ότι παρόμοιες ταινίες έχουν και παρόμοιο κοινό. Έτσι δημιουργώντας ένα σύνολο όλων των ατόμων του συνόλου δεδομένων οι οποία παρακολούθησαν την συγκεκριμένη ταινία, μπορεί να οριστεί η επικάλυψη μεταξύ των συνόλων των θεατών των ταινιών. Ακολουθώντας την παραδοχή για τη χρήση αυτής της τεχνικής, δύο ταινίες ορίζονται ως πιο όμοιες όσο μεγαλύτερος είναι ο βαθμός της επικάλυψης. Παρόλα αυτά το σύστημα αυτό έχει την τάση να κάνει συστάσεις μονάχα δημοφιλών ταινιών καθώς είναι αυτές με το μεγαλύτερο σύνολο θεατών. Γι’ αυτό τον λόγο ένας ακόμη αλγόριθμος δημιουργήθηκε, όπου γίνεται χρήση μετρικών κατάλληλων για δυαδικά δεδομένα. Συγκεκριμένα χρησιμοποιούνται οι μετρικές ομοιότητας jaccard και dice o οποίες δίνουν παρόμοια αποτελέσματα. Το πλεονέκτημα χρήσης αυτών των μετρικών έναντι του σκορ επικάλυψης είναι ότι με αυτές το σκορ επικάλυψης κανονικοποιήται διαιρώντας με το σύνολο θεατών και «τιμωρούνται» οι πολύ δημοφιλείς ταινίες.

Στις τεχνικές που περιγράφονται παραπάνω δεν χρησιμοποιείται όλη η διαθέσιμη πληροφορία του συστήματος, καθώς οι βαθμολογίες των χρηστών δεν περιλαμβάνονται. Έτσι ταινίες που έχουν συγκεντρώσει χαμηλές βαθμολογίες μπορεί να μην διαφέρουν με αυτές με υψηλές βαθμολογίες αρκεί να έχουν παρόμοιους θεατές. Επιπλέον ταινίες με μικρότερο σύνολο θεατών είναι δύσκολο να εμφανιστούν στις συστάσεις του συστήματος. Έτσι ορίζεται αλγόριθμος που βασίζεται στην cosine similarity. Η cosine similarity, είναι μία μετρική ομοιότητας διανυσμάτων η οποία λαμβάνει υπόψιν της την κατεύθυνση δύο διανυσμάτων και όχι μέτρο αυτών. Παρόλα αυτά, σε ταινίες με μικρό αριθμό θεατών μόνο ταινίες όπου υπάρχι επικάλυψη θεατών μπορούν να έχουν υψηλή ομοιότητα, και έτσι λόγω αυτού του μειονεκτήματος το σύστημα έχει πολύ θόρυβο (noisy results).

Στη συνέχεια, γίνεται μία περιγραφή της μηχανικής μάθησης εν τάχει και παρουσιάζεται ο αλγόριθμος των Κ – κοντινότερων γειτόνων. Δημιουργήθηκε ένα σύστημα collaborative filtering με βάση αυτό τον αλγόριθμο. Αρχικά, για κάθε ταινία δημιουργείται το διάνυσμα των βαθμολογικώς της από τους χρήστες του συστήματος και στη συνέχεια για την ταινία που εισάγεται στο σύστημα υπολογίζεται η cosine similarity με τις υπόλοιπες ταινίες του συνόλου δεδομένων. Για κάθε ταινία σύνολο γειτόνων και αυτοί προτείνονται στον χρήστη.

Στο τελευταίο κομμάτι του κεφαλαίου παρουσιάζονται τεχνικές δανεισμένες από το τομέα του Natural language processing (NLP). Πρόκειται για τις τεχνικές LSA – Latent semantic analysis και TF-IDF – term frequency – inverse document frequency.

Η LSA είναι τεχνική κατά την οποία πραγματοποιείται παραγοντοποίηση του αρχικού πίνακα των αλληλεπιδράσεων χρήστη – προϊόντων (στην περίπτωσή μας βαθμολογιών ταινιών από τον χρήστη), σε γινόμενο δύο πινάκων με διαστάσεις μικρότερες του αρχικού. Στους δύο αυτούς πίνακες αποθηκεύεται μία αναπαράσταση των χαρακτηριστικών των ταινιών και των χρηστών του συστήματος. Τα χαρακτηριστικά που προκύπτουν, αν και αποτελούν μαθηματικές αναπαραστάσεις και είναι αδύνατο να εξηγηθούν σαν συνδυασμοί «κλασικών» χαρακτηριστικών, όπως παραδείγματος χάριν το είδος της ταινία ή η διάρκεια αυτής, βρίσκονται ωστόσο συχνά, κοντά σε μία αναπαράσταση που διαισθητικά θα σκεφτόταν κάποιος. Επακολούθως, η τεχνική ονομάζεται latent semantic analysis, καθώς φαίνεται να μπορεί να δημιουργεί σημασιολογικά χαρακτηριστικά που διέπουν τα αντικείμενα του συνόλου δεδομένων. Ο χρήστης του συστήματος καλείται κάθε φορά να επιλέγει το όνομα της ταινίας καθώς και τον αριθμό των «κρυφών χαρακτηριστικών» που υπολογίζονται.

Δεδομένων των ιδιαίτερων αποτελεσμάτων που προέκυψαν από την LSA, κατασκευάστηκε ένας ακόμη αλγόριθμος που χρησιμοποιείται στην επεξεργασία κείμενων, ο TF-IDF. Αυτή η τεχνική βρίσκει ευρεία εφαρμογή στις μηχανές αναζήτησης, καθώς δίνει μία βαρύτητα σε κάθε λέξη ενός κειμένου. Αυτό μπορεί να επιτευχθεί βρίσκοντας την συχνότητα της λέξης σε κάθε κείμενο, τον αριθμό των κειμένων σε μία συλλογή καθώς και τον συνολικό αριθμό κειμένων στα οποία εμφανίζεται η λέξη. Θεωρώντας ως κείμενο την κάθε ταινία και ως λέξη τον κάθε θεατή ήταν δυνατό να εφαρμοστεί η παραπάνω τεχνική για την δημιουργία μιας μηχανής συστάσεων ταινιών.

Πρέπει να παρατηρεί ότι οι δύο παραπάνω τεχνικές βρίσκουν ευρεία εφαρμογή σε content based filtering συστήματα συστάσεων, όμως στην παρούσα εργασία διατηρείται ο περιορισμός του συνόλου δεδομένων μόνο στις βαθμολογίες των ταινιών από τους χρήστες. Τέλος, σε αυτό το κεφάλαιο δεν γίνεται μέτρηση της ακρίβειας των συστημάτων με κάποια μετρική σφάλματος και η επιτυχία των συστάσεων είναι αντίστοιχη του γούστου του κάθε χρήστη.

Στο επόμενο κεφάλαιο παρουσιάζονται δύο αλγόριθμοι για παραγοντοποίησης πινάκων. Στόχος, είναι η χρήση της τεχνικής ώστε να δημιουργηθεί αυτή τη φορά ένα σύστημα προτάσεων ταινιών, οι οποίες όμως θα είναι εξατομικευμένες για κάθε χρήστη της πλατφόρμας όπου βρίσκεται το σύστημα. Συγκεκριμένα, και τα δύο συστήματα κάνουν πρόβλεψη βαθμολογίας της κάθε ταινίας από κάθε χρήστη και στη συνέχεια μπορούν να προτείνουν τις ταινίες με τις υψηλότερες βαθμολογίες. Οι αλγόριθμοι που παρουσιάζονται είναι ο ALS (Alternating Least Squares) και ο SVD (Singular value decomposition). Δοκιμάζονται διαφορετικοί συνδυασμοί παραμέτρων για κάθε έναν από αυτούς και ο βέλτιστος επιλέγεται με βάση το χαμηλότερο MSE.

Ο ALS, αποτελεί έναν από τους πιο διαδεδομένους αλγορίθμους που χρησιμοποιούνται για παραγοντοποίηση πινάκων στα πλαίσια δημιουργίας συστημάτων συστάσεων. Στον ALS, γίνεται η παραδοχή ότι υπάρχει μία αναπαράσταση των χαρακτηριστικών των χρηστών του συστήματος και μία των ταινιών του συστήματος, τέτοια ώστε το εσωτερικό γινόμενο του διανύσματος των χαρακτηριστικών του χρήστη επί το διάνυσμα των χαρακτηριστικών της ταινίας να δίνει την βαθμολογία της ταινίας αυτής από τον χρήστη. Για να μπορέσει να δημιουργηθεί ένα τέτοιο σύστημα, αρχικά ορίζεται η συνάρτηση σφάλματος που θα πρέπει να ελαχιστοποιηθεί από το μοντέλο. Στην περίπτωση αυτή είναι:

$$L = \|R - XxY^T\|_2 + \lambda(\|X\|_2 + \|Y\|_2)$$

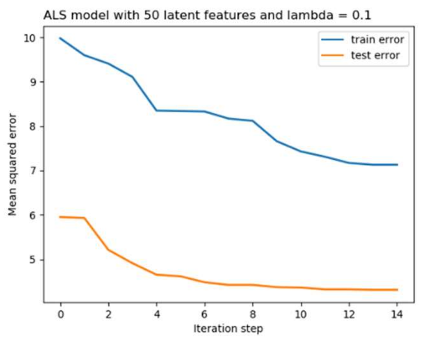
Ο πρώτος όρος είναι το μέσο τετραγωνικό σφάλμα μεταξύ της πραγματικής βαθμολογίας της ταινίας και της πρόληψης της από το σύστημα. Ο δεύτερος, αποτελεί έναν παράγοντα «σταθεροποίησης» της μηχανής συστάσεων, ώστε αυτή να μην μάθει το σύνολο δεδομένων σε τέτοιο βαθμό, ώστε να αδυνατεί να παράγει ακριβείς προβλέψεις σε νέα δεδομένα (overfitting).

Η λύση της συνάρτησης αυτής γίνεται μέσω gradient descent, η οποία είναι μία επαναληπτική διαδικασία βελτιστοποίησης κατά την οποία η παράγωγος της συνάρτησης προς βελτιστοποίηση υπολογίζεται ως προς τις μεταβλητές προς βελτιστοποίηση. Στην παρούσα περίπτωση όμως είναι δύο οι μεταβλητές, τα διανύσματα χαρακτηριστικών των χρηστών και των ταινιών του συστήματος. Έτσι η συνάρτηση προς βελτιστοποίηση είναι μη-κοίλη και επακολούθως αδύνατο να βελτιστοποιηθεί.

Παρόλα αυτά, μπορεί να ακολουθηθεί η παρακάτω τεχνική: Θεωρώντας τις τιμές ενός από τους δυο πίνακες σταθερή (έστω X), τότε μπορούμε να εφαρμόζουμε gradient descent ως προς Y. Τότε το πρόβλημα μετατρέπεται σε πρόβλημα επίλυσης ενός απλού γραμμικού συστήματος, η λύση του οποίου υπάρχει.

Έτσι χρησιμοποιώντας αυτή τη δυνατότητα, η ALS είναι μία τεχνική βελτιστοποίησης δύο βημάτων. Για την ολοκλήρωση μίας επανάληψης πρώτα κρατά σταθερές τις τιμές του πίνακα χαρακτηριστικών των χρηστών ή των ταινιών, κάνει τις απαραίτητες αλλαγές στα διανύσματα και στη συνέχεια επαναλαμβάνει την διαδικασία για τον άλλο πίνακα, κρατώντας τον πρώτο σταθερό. Ολόκληρος ο αλγόριθμος κατασκευάστηκε από την αρχή. Οι συναρτήσεις βασίζονται στο πακέτο *explicit* της Python.

Για τον έλεγχο της ποιότητας του αλγορίθμου από το σύνολο δεδομένων επιλέχθηκαν τυχαία 10 βαθμολογίες ταινιών από κάθε χρήστη και αφαιρέθηκαν ώστε να δημιουργηθούν δύο σετ δεδομένων, ένα για εκπαίδευση και ένα για αξιολόγηση του συστήματος. Στη συνέχεια οι τιμές των latent features (πίνακες χαρακτηριστικών) αρχικοποιήθηκαν σε τυχαίες τιμές και ο αλγόριθμος εκπαιδεύτηκε.



Τα βέλτιστα αποτελέσματα (με βάση την μέτρηση του MSE μόνο στις γνωστές τιμές των βαθμολογιών) επιτεύχθηκαν από αλγόριθμο με 50 latent features για τις ταινίες και τους χρήστες και παράμετρο σταθεροποίησης ίση με 0.01 τόσο για τον πίνακα των χρηστών όσο και για τον πίνακα χαρακτηριστικών των ταινιών.

Καθώς ο αλγόριθμος είναι κατασκευασμένος εξαρχής και δεν υπήρχε η δυνατότητα παραλληλοποίησης του τα αποτελέσματα δεν μπόρεσαν να βελτιωθούν παραπάνω καθώς δεν υπήρχε η δυνατότητα grid search, λόγω των μεγάλων χρόνων εκπαίδευσης (περίπου 6 ημέρες για κάθε συνδυασμό παραμέτρων).

Στη συνέχεια, στο ίδιο σύνολο δεδομένων εφαρμόστηκε η τεχνική SVD. Κατά την τεχνική αυτή ο αρχικός πίνακας A που περιέχει τις βαθμολογίες παραγοντοποιείται στο γινόμενο:

$$A = U \times \Sigma \times V^T$$

Όπου U και V είναι οι πίνακες χαρακτηριστικών των ταινιών και των χρηστών αντίστοιχα και είναι ορθογώνιοι, ενώ ο πίνακας Σ είναι ο διαγώνιος πίνακας των singular values τοποθετημένων σε φθίνουσα κατάταξη. Τότε, μία τιμή του πίνακα A, έστω A_{ui}, μπορεί να προσεγγιστεί από το γινόμενο $\sum_{k=1}^{K}(u_{uk}\Sigma_{kk}v_{ki}^{T})$ όπου K είναι ο αριθμός των latent factors του συστήματος.

Πρόκειται για μία τεχνική που εφαρμόζεται ευρύτατα σε πίνακες, όμως είναι αδύνατο να εφαρμοστεί στον πίνακα βαθμολογιών του παρόντος προβλήματος καθώς έχει εξαιρετικά αραιές τιμές (υπάρχει περίπου το 3% όλων των δυνατών βαθμολογιών). Ονομάζουμε singular values τις ιδιοτιμές του πίνακα $A^{T}A$, όμως στην περίπτωσή μας ο πίνακας A δεν είναι πλήρης και έτσι είναι αδύνατο να υπολογιστεί το γινόμενο και κατά συνέπεια να πραγματοποιηθεί ο αλγόριθμος SVD.

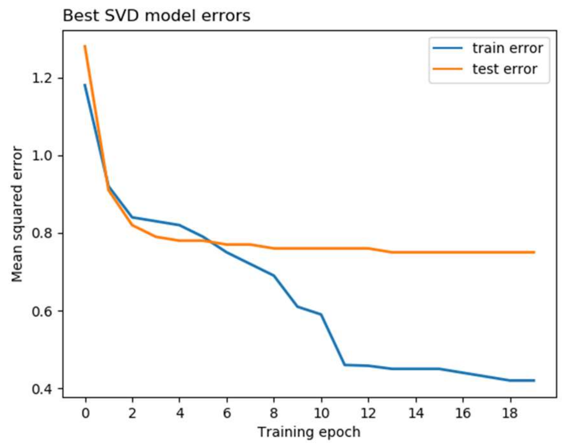
Μία προσεγγιστική λύση στο παραπάνω πρόβλημα δίνεται αν αναλογιστούμε το πρόβλημα από την αντίθετη οπτική. Θα πρέπει να βρεθούν οι πίνακες, οι οποίοι θα μπορούν να προσεγγίσουν τις τιμές του A. Για να οριστεί λύση στο παραπάνω πρόβλημα, πρέπει να οριστεί συνάρτηση βελτιστοποίησης. Για το SVD – Recommender system, χρησιμοποιήθηκε η συνάρτηση που όρισε ο Simon Funk κατά τη διάρκεια του Netflix prize και είναι η εξής:

$$L = \sum\nolimits_{u,i\in K}(a_{ui} - u_u^{T}v_i - \mu + b_i + b_u)^2 + \lambda(\|u_u\|^2 + \|v_i\|^2 + b_i^2 + b_u^2$$

Όπου μ είναι η μέση βαθμολογία όλων των ταινιών, b_i, η μέση βαθμολογία της ταινία i και b_u ο μέσος όρος των βαθμολογιών του χρήστη u.

Η συνάρτηση αυτή μπορεί να λυθεί με τη βοήθεια της gradient descent τεχνικής. Η Stochastic gradient descent (SGD) προτιμήθηκε σε αυτή την περίπτωση έναντι του κλασικού αλγορίθμου gradient descent, καθώς αν και ο αριθμός των εγγραφών (βαθμολογίες ταινιών) δεν αυξήθηκε, αυξήθηκε ο αριθμός των παραμέτρων της εξίσωσης και έτσι με την κλασική gradient descent ο αλγόριθμος θα καθυστερούσε.

Οι παράγωγοι υπολογίζονται σε κάθε βήμα για κάθε μία από τις παραμέτρους της παραπάνω συνάρτησης (b_u, b_i, u_u, v_i) και οι τιμές των διανυσμάτων ανανεώνονται. Το σφάλμα σε κάθε βήμα υπολογίζεται μόνο για τις γνωστές βαθμολογίες και έτσι μόνο μία προσέγγιση του αρχικού πίνακα A μπορεί να επιτευχθεί.



Για την δημιουργία του συστήματος συστάσεως ταινιών χρησιμοποιήθηκε ο αλγόριθμος του πακέτου *Surprise* της Python. Για την εύρεση του καλύτερου συνδυασμού των hyperparemeters του μοντέλου πραγματοποιήθηκε Grid search και το καλύτερο μοντέλο κατάφερε να έχει μέσο τετραγωνικό σφάλμα μόλις 0.75 στο σύνολο ελέγχου μετά από 20 εποχές εκπαίδευσης. Το μοντέλο έχει 100 latent features και 0.05 παράμετρο σταθεροποίησης.

Τέλος πρέπει να επισημανθεί ότι τα αποτελέσματα θεωρούνε αρκετά καλύτερα των αντίστοιχων αποτελεσμάτων του ALS, καθώς πλέον το σύστημα σταμάτησε να «υπερεκπαιδεύεται» στα δεδομένα.

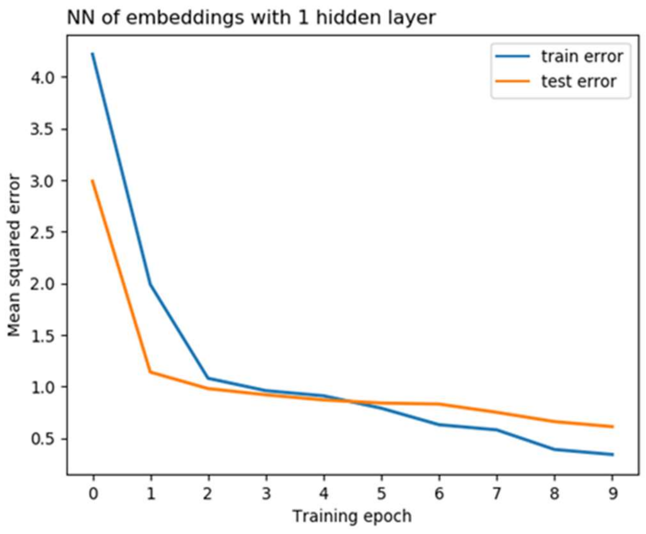
Στο τελευταίο μέρος της παρούσας διπλωματικής γίνεται χρήση τεχνικών deep learning για τη δημιουργία του συστήματος συστάσεων.

Αρχικά γίνεται μία σύντομη εισαγωγή στην αρχιτεκτονική του Multilayer perceptron νευρωνικού δικτύου. Ορίζονται οι τυπικές συναρτήσεις κόστους του δικτύου και οι συναρτήσεις ενεργοποίησης των νευρώνων. Επιπλέον γίνεται αναφορά και στον αλγόριθμο backpropagation, ο οποίος χρησιμοποιείται ευρέως στα νευρωνικά δίκτυα ώστε το δίκτυο να μπορέσει να μάθει τα βάρη των συνδέσεων των νευρώνων. Ο αλγόριθμος αυτός χρησιμοποιεί τον κανόνα της αλυσίδας ώστε να διανείμει το σφάλμα του αποτελέσματος «προς τα πίσω», δηλαδή στα βάρη των συνδέσεων και να μπορέσει να βελτιστοποιήσει τις τιμές αυτών ώστε να μειώσει το σφάλμα.

Ως embeddings ορίζονται οι αναπαραστάσεις ενός συνόλου κατηγορικών μεταβλητών σε έναν συνεχή χώρο αριθμών. Χαρακτηριστικό παράδειγμα αποτελεί η τεχνική one hot encoding. Όμως η τεχνική one hot encoding έχει δύο κύρια μειονεκτήματα. Δεν είναι εφικτό να εφαρμόζεται όταν η ηθικότητα των διαφορετικών τιμών μια μεταβλητής είναι μεγάλη καθώς αυξάνει πολύ την διάσταση του χώρου και κατά δεύτερο μέσω αυτής της τεχνικής, όμοια χαρακτηριστικά δεν τοποθετούνται «κοντά» στον χώρο των embeddings. Η βέλτιστη τεχνική είναι αυτή που μπορεί να ξεπεράσει και τα δύο αυτά προβλήματα και μπορεί να επιτευχθεί μέσω ορισμού ενός επιβλεπόμενου προβλήματος για λύση από το νευρωνικό δίκτυο. Τα embeddings λοιπόν προκύπτουν από τα βάρη του δικτύου και μαθαίνονται να είναι τέτοια ώστε το σφάλμα του δικτύου (η διαφορά μεταξύ προβλεπόμενης βαθμολογίας και πραγματικής βαθμολογίας) να είναι το ελάχιστο.

Για τη δημιουργία των συστημάτων συστάσεων, κατασκευάστηκαν δύο αρχιτεκτονικές δικτύου. Και οι δύο είναι κατασκευασμένες χρησιμοποιώντας την βιβλιοθήκη *Keras* της Python. Μετά το επίπεδο εισόδου, το πρώτο επίπεδο που ορίστηκε και στις δύο αρχιτεκτονικές είναι ένα layer embeddings. Στη συνέχεια τοποθετήθηκε ένα layer συνένωσης (merging layer) που υπολογίζει το διανυσματικό γινόμενο της αναπαράστασης των χρηστών επί την αναπαράσταση των ταινιών του συνόλου δεδομένων. Στην πρώτη αρχιτεκτονική, αυτό ήταν και το επίπεδο εξόδου. Στη δεύτερη περίπτωση ένα επιπλέον κρυφό επίπεδο τοποθετήθηκε μετά το merging layer.

Το τελευταίο μοντέλο εκπαιδεύτηκε για 10 εποχές, είχε ένα embeddings layer 50 latent features και ένα densely connected κρυφό επίπεδο 128 νευρώνων με σιγμοειδή συνάρτηση ενεργοποίησης, ενώ ο αλγόριθμος backpropagation πραγματοποιήθηκε με χρήση του Adam.



Το μοντέλο αυτό είχε τα καλύτερα αποτελέσματα και στα δύο σετ σε σχέση με όλους τους αλγόριθμους που δοκιμάστηκαν στην παρούσα διπλωματική και κατάφερε να έχει μέσο τετραγωνικό σφάλμα 0.61 στο σύνολο ελέγχου και 0.34 στο σύνολο εκπαίδευσης.

1. Introduction

1.1 Introduction to recommender systems

The explosive growth of internet has resulted in a phenomenon known as information abundance. In a way we are drowning in information but starving for knowledge, and it is mainly due to influx of data into the internet caused by people on one side and the scarcity of techniques to process the data to knowledge on the other side. So the current scenario demands new techniques that can assist us to discover resources of interest among the enormous options we are presented with. All of this paved way for the introduction of recommender systems which attempt to recommend items of interest to particular users by predicting a user's interest in an item based on related information about the items, the users and the interactions between items and users.

The first research paper in recommender systems came out in the mid 90s [1] and since then research in this area got diversified and various approaches were introduced to present better recommendations. Recommender System algorithms basically performs information filtering and can be classified into three types, namely collaborative filtering, content based filtering and hybrid filtering. With time newer strategies evolved from the basic categories with improved recommendations by including the social media information [2], information from internet of things [3], location information etc. A lot of work has happened in this area over the last decade on both industry and academia. Recommender systems still remains an area of high interest as it constitutes a problem-rich area and the possibilities it offer for practical applications. A wide range of applications including recommendations in web search, books, movies, music, restaurants, food, apparels, vehicles, targeted advertisements, medicines, news, potential customers for companies and many more.

There are two main purposes for a recommender system. The first one is user oriented, and is to increase a customer's satisfaction with the platform. The second one, business oriented [4], is to increase the company's revenue by increasing sales. It is obvious that these two are strongly correlated, since the better the recommendations are for a customer, the more they will consume and the better their overall experience will be. As recommendation engines are widely used by e-commerce sites, they are used to improves the user experience and at the same time benefiting the store. The system is able to convert browsers to buyers and cross-sell more items by means of suggestions while shopping. It increases the user loyalty by enabling them to purchase items in fewer clicks and also providing the frequent customers with good deals and offers. In short a recommender system is able to attract the interest of the customers by providing them fast and accurate recommendations .

There is a variety of recommender systems since each company's client's needs differ, and different business problems require different solutions. The following examples provide an overview of three such recommenders.

- Book recommendations.
  A recommender system's goal here is to expose new content to the user. This way the customer's interest can be sparked, and they are encouraged to consume more on the platform.
  The recommendation continues to have an impact even after the sale has been completed since if the customer is satisfied with it, it is more likely that they will choose one of the platform's recommendations next time that they are looking for a new book to read.
- Stock recommendations [5].
  In this example, a list of stock options that are most profitable, are recommended to the clients. Novelty is not a requirement in this case. Stocks that are in the list of the client's historical transactions can be recommended again if they remain a viable option.
- Product recommendations.
  Balance between recommending new products and products that have already been bought by the client is key in this case. Products found in the customer's previous transactions, serve as a reminder of their frequent purchases. However, recommending new products that the client may like or need is also very important, otherwise the client will lose interest on the platform and will stop using it.

Recommender systems can be divided into three main categories. The classification is based on the algorithms that are used and the kind of information that is needed to implement them. The main types of recommendation engines are [6] [7]:

- Collaborative filtering
- Content-based filtering
- Hybrid recommendation systems

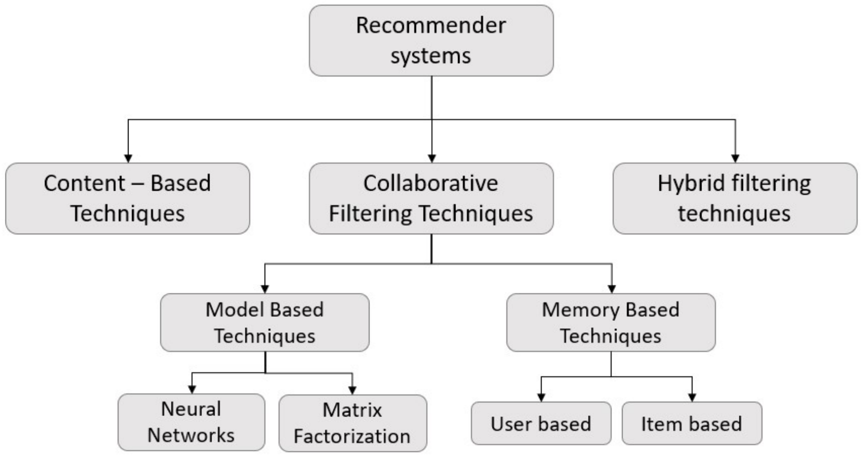
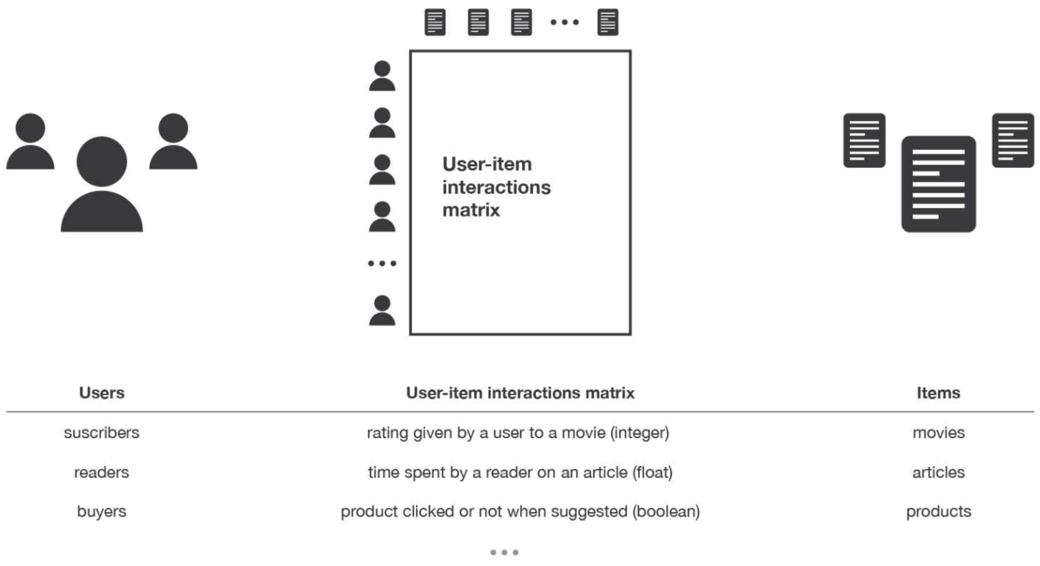


Figure 1 Recommender System Classification

Collaborative filtering

Collaborative filtering methods for recommender systems are methods that rely solely on past interactions recorded between users and items in order to produce new recommendations. These interactions are stored in the so-called “user-item interactions matrix”.



Users	User-item interactions matrix	Items
suscribers	rating given by a user to a movie (integer)	movies
readers	time spent by a reader on an article (float)	articles
buyers	product clicked or not when suggested (boolean)	products
	•••	

Figure 2 User - Item interaction matrix

In these methods predictions are based on an analysis of the past activities and preferences of the user, as well as similarity to other users. They are based on the assumption that people that have a similar behavior in the past, are likely to continue to have similar behavior in the future [8].

Thus items from the "interaction" history of one user may be of interest to the other and vice versa. To elaborate more, the underlying assumption is that if user A and user B have the same opinion on subjects X and Y, then user A is more likely to have a similar opinion with user B on subject Z, rather than a random user.

The class of collaborative filtering approach can be further split into subcategories, based on the collaborative filtering algorithm in use.

- Memory based
  Memory based (lazy learner) approach, assumes no model and directly works with the values of the recorded interactions. They are based on nearest neighbor search finding the closest users for a given user and suggesting the most popular items in this neighborhood.
- Model based
  This is the opposite of the previous approach. It assumes an underlying generative model [9] that try to explain the user-item interactions. In this approach, a scientist tries to discover the appropriate model that will make new predictions.

Memory based approached is further split into User-User and Item-Item collaborative filtering:

- User-User collaborative filtering
  This method represents users based on their interactions with items and evaluates the distance between them. The algorithm here tries to find the lookalike for a specific user (the user with the most similar "interactions profile") and present them with products based on the preferences of their lookalike. This approach, although it leads to very good results, is very computationally expensive as every customer pair must be analyzed and the number of users on an average e-commerce site is usually larger than the number of products.
  First every user is represented by their vector of interactions with different items. Then a similarity metric is computed between the user of interest and all other users in the dataset. This similarity measure should consider as close users those with similar interactions on the same set of items. After the similarities are computed, a common approach is to keep the k-nearest neighbor [10] of the user of interest and suggest to him the most popular items - from the list of items the user has not yet interacted with - amongst them.
- Item -Item collaborative filtering
  This method represents items based on interactions users had with them and evaluates distances between them. Two items are considered similar when most of the users that have interacted with both of them did so in a similar way. To make recommendations for a target user, first the list of the user's most liked items is constructed, and then those items are represented by their vector of interactions with all users. Similarities are computed between the most liked items of the target user, and all other items in the

dataset. Then the k-nearest neighbors of the selected item are recommended to the user. This approach can then be continued, for the second, third, .. nth most liked item by the user.

In the model based approaches lie the matrix-factorization methods.

- Matrix Factorization
  The goal of matrix factorization methods is to decompose the sparse user-item interaction matrix into a product of two smaller and denser matrices [11][12]: a user-factor matrix, that contains user representations, and a factor-item matrix, that contains item representations. The assumption here is that a low dimensional latent space of features (embeddings) exists and can represent both users and items in such a way that by computing the dot product between the dense user and item vectors, results in the interaction between them.
  These features cannot be given to the model directly, as they would be on content based approaches. Instead they are discovered by the system. Since they are learned features, and not features representing distinct attributes, the relationship between them is non-intuitive and can usually not be understood nor explained by humans. However, with the use of such algorithms, even without introducing distinct features to it, the representation that the model outputs is extremely close to an intuitive decomposition a human could think of. Users that are considered to be close with one-another in terms of preferences, and items with similar characteristics, end up having close representations in the latent space.

Content based filtering

As the name indicates, a content-based recommender system uses the content information of the items to build the recommendation model. A content recommender system typically contains a user-profile-generation step (user key words), item-profile-generation step (item key words) and a model-building step to generate recommendations for an active user. The content-based recommender system recommends items to users by taking the content or features of both items and user profiles. The common approach is to represent both the users and the items under the same feature space. Then similarity scores could be computed between users and items. The recommendation is made based on the similarity scores of a user towards all the items [13].

In simple terms, the system recommends items similar to those that the user has liked in the past. The similarity of the items is calculated based on the features associated with the other items they are being compared to, and is matched with the user's historical preferences.

This technique does not take into consideration the user's neighborhood preferences. Hence, it doesn't require a large user group's preference for items to increase its recommendation accuracy. It only considers the user's past preferences and the properties/features of the items.

Hybrid techniques

Hybrid recommendation engines combine the two previous recommendation techniques to reinforcine their advantages and reduce their disadvantages or limitations [14]. Most commonly, collaborative filtering is combined with some sort of content based technique in an attempt to avoid the problems, such as users with low number of interactions with catalogue items [15].

1.2 Recommended systems techniques comparison

Collaborative Filtering

Collaborative filtering approaches (both memory and model based) require no other information, but items, users and the interaction between them. For a fixed system of items and users, new interactions recorded over time bring new information to the system. As new examples are created, new recommendations can become more accurate.

However the luck of extra attributes for items and users is the main disadvantage of using such algorithms. Collaborative filtering algorithms do suffer from a major challenge, the 'cold start problem' [16,17]. It is impossible for them to recommend products to new users, or recommend items with a very low number of recorded interactions. This can be overcome with simple approaches such as recommending random items to new users or new items to random users (random strategy) [18], or recommending popular items to new users, or new items to the most active users (maximum expectation strategy). The most robust approach however is using a non-collaborative method for the beginning of the lifecycle of new users and items.

The user-user method is based on the search for similar users in terms of their interactions with items. In general, every user has interacted with a few items only. This makes the method pretty sensitive to any recorded interactions (high variance). On the other hand, as the final recommendation is only based on interactions recorded for users that are considered similar to the user of interest, the results obtained are more personalized (low bias).

Conversely, the item-item method is based on the search for similar items in terms of user-item interactions. As, in general, a lot of users have interacted with a specific item, the neighborhood search is far less sensitive to single interactions (lower variance). As a counterpart, interactions coming from all users (even those with very different history from the user of interest) are taken into consideration for the recommendation. This makes the method less personalized (more biased). Thus, this approach is less personalized than the user-user approach but more robust.

Content based filtering

The main advantage of content based filtering over collaborative filtering approaches, is that the cold start problem can be avoided when items have sufficient descriptions. Also recommender systems that are built with this technique are able to recommend to users with unique tastes finding items that much those criterions. Unpopular and new items can be recommended as recommendation solely rely on item and user characteristics. Last, the content representation allows for a variety of analysis approaches such as text preprocessing techniques, semantic information etc. [19].

However finding the appropriate features can be hard (e.g. images, news etc.). Also, users rarely fill information about their preferences on platforms, so recommendations for new users are hard. Last, these systems tend to over-specialize. They can not recommend items outside the user's profile, but people usually have more complex interest than that.

1.3 Evaluation of Recommender systems

Recommender systems are popular both commercially and in the research community, where many algorithms have been suggested for providing recommendations. These algorithms typically perform differently in various domains and tasks. Therefore, it is important from a research perspective, as well as from a practical view, to be able to choose an algorithm that is better suited for the domain and the task of interest. The standard way to make such decisions is by comparing a number of algorithms offline using some evaluation metric. Indeed, many evaluation metrics have been suggested for comparing recommendation algorithms [20].

Netflix, even started a competition (Netflix prize competition) to find an algorithm which could improve upon their accuracy by 10 percent for one million dollars. However, they later realized that accuracy of predictions had no relation with whether someone will be interested in watching the predicted movie as a person wants to see new movies that they would like to see, so a list of top-n movies should be put in front of users and we should measure how they react to these recommended movies.

Let us consider a matrix R of user – item interactions and the one predicted by the model, the matrix $\hat{R}$. Then, $\widehat{r_{ij}}$ is the predicted value of r_{ij}, that represents the quantitative interaction of user i with item j. Since the three main models that are built in this thesis are generating an approximation of these values, the metrics that are presented are for the evaluation of such models.

There is a list of traditional error metrics for recommender systems that are used in academia. We will be focusing solely on offline measures.

- Mean absolute error
  $MAE = \frac{\sum_{i\in U, j\in I} |r_{ij} - \widehat{r_{ij}}|}{N}$ (1.1)
- Mean squared errors
  $MSE = \frac{\sum_{i\in U, j\in I} (r_{ij} - \widehat{r_{ij}})^2}{N}$ (1.2)
- Root mean squared error
  $RMSE = \sqrt{\frac{\sum_{i\in U, j\in I} (r_{ij} - \widehat{r_{ij}})^2}{N}}$ (1.3)

Other commonly used metrics, that are however not used when predicting ratings, are precision and recall.

- Precision, measures how capable of delivering relevant information with the least amount of recommendations the system is.

$Precision = \frac{|Recommendet\ items\ \cap Relevant\ items|}{|Recommended\ items|}$ (1.4)

- Recall, measures how capable of finding all the relevant elements and recommend them to the user the model is.

$Recall = \frac{|Recommendet\ items\ \cap Relevant\ items|}{|Relevant\ items|}$ (1.5)

This measures are typical error metrics in binary classification problems where we can calculate the number of true positive (TP), true negative (TN), false positive (FP) and false negative (FN) predictions.

Then $precision = \frac{TP}{(TP+FP)}$, and $recall = \frac{TP}{(TP+F\ \ )}$.

In order for this metric to be used in recommendation systems the following assumptions must be made. Let us assume we are predicting movie ratings with values in [0.5, 5]. We can then assume that values in [0.5, 3] represent a negative rating to a movie and values in [3.5, 5] a positive rating to a movie. Then TP, TN etc. quantities can be measured.

However, since only a tiny amount with regards to the total number of possible user – item interactions exist, another problem arises. How can we evaluate the missing values, since the system is predicting a rating for them. One approach is to ignore them, like we do with the classic error metrics.

A second approach is to evaluate only the positive recommendations. Two new measures can then be defined. Precision at k (P@k) and Recall at k (R@k), where k is the number of evaluating recommendations sorted by value [21].

There are issues with the later approach as well. Let us imagine the original rating of three movies is 5,4,not rated and the predicted values were 4.5, 3.9 and 3.7. Then P@k = 2/3=0.67 and R@k = 2/2=1. Imagine now that the rating are 5,4,2 and another model predicted 4.5,3.9 and 2.1. Then

P@k = 2/2=1 and R@k = 1. Just looking at the metrics we would conclude that the second model is better than the first. But that is possibly not correct.

We must however acknowledge the main flaw of measuring the quality of a recommender system using error metrics. The purpose of a recommender system is to let the users discover items that they will like. The problem is that this is impossible to evaluate because our training data can never contain the results we are looking for. Let's think about a recommender for movies, we know user Bob likes movies A, B, C, D and E. We can split this into a train and test set with A, B, C and D in the train set. Then we evaluate if the algorithm recommends item "E" because we know the user likes E. The big problem is that an algorithm that recommends F, Z or W could be even better than the one that recommends E. This evaluation strategy is valid if and only if we know all the items the user likes, and we know for sure that the user does not like any item outside the ones in the list.

Refining our example let us say user Alice likes Pinocchio, Aladdin and Kill Bill. Based on this, our recommendation engine should pick things like Cinderella, Kill Bill II, etc. If we leave Pinocchio and Aladdin in the train set and put Kill Bill in the test set then our recommender will fail because there is no way to recommend Kill Bill based on Pinocchio and Aladdin. This means that splitting the training set can only lead to worst recommendations and that we can't use part of the training set as our test set.

So, to measure the accuracy of recommender system, we will continue the analysis in this thesis with the classic metrics (MAE, MAE and RMSE). In all cases, only the error between the prediction and the known interactions of the user with an item is calculated.

2. The dataset

There are two types of data used by recommendation systems. Explicit data are those that have some sort of rating. In this scenario the scientist is aware of how much a user liked or disliked an item by simply taking a glimpse at the provided rating. Data like that are usually really hard to find because users rarely spend the time to rate items.

The second type of datasets is implicit datasets. These are data gathered from user behavior. They could be items that a user purchased, how many times they repeated a song, how much time they spend on a page describing a specific product on a website etc. [22]. The upside is that there are lots of data like this, the downside is that they are more noisy and it is not always apparent what they mean.

In order to complete an analysis between different algorithms the famous movielens dataset was used. GroupLens is a research lab of the University of Minnesota that runs the website movielens.org. In this website, a large group of movie titles exist and users can sign up for free and provide their ratings for those movies. Each year they release two new datasets, the movielens 20M and the movielens 100M that contain 20 million and 100 million movie ratings respectively. The first dataset was chosen for this project. It contains interactions between viewers and movies about movies that were released up to 2018.

The dataset consists of anonymized user ids, movie ids (every movie is mapped on a specific numeric id) and the corresponding rating the user has given to that movie. The ratings range from 0.5 to 5.0.

There are 27.753.444 distinct interactions in the dataset. They are created by a set of 283.228 distinct users over a set of 53.889 movies. The distribution of movie ratings is severely skewed with most movies having a very low number of interactions with users. It is made very clear that only the very popular movies have a large number of interactions.

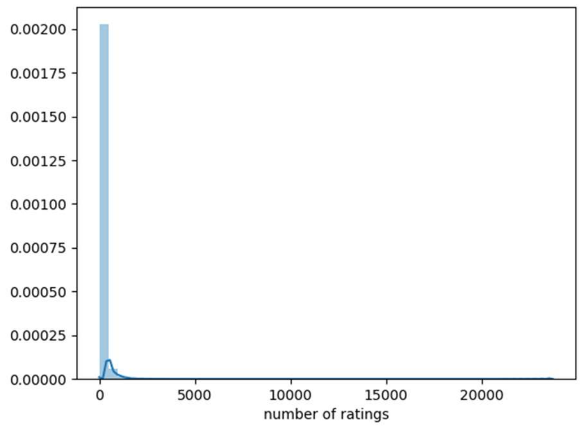


Figure 3 Distribution plot of movie ratings

statistic	Number of ratings
25% quartet	15
50% quartet	30
75% quartet	95
Min	1
Max	23.715
Mean	98
std	213

Similar behavior can be found regarding users. Users rarely do provide feedback in a dedicated manner and are also most likely to provide feedback only on items they liked.

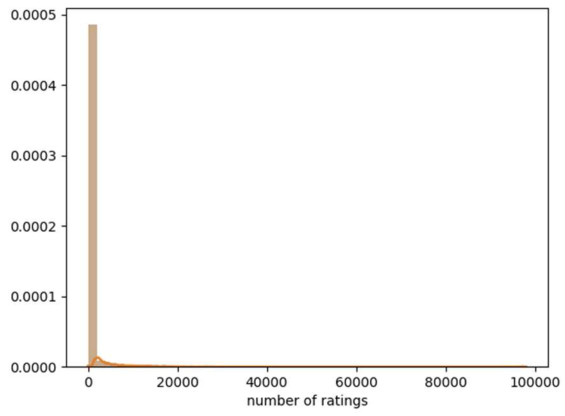


Figure 4 Distribution plot of rated movies

statistic	Number of ratings
25% quartet	2
50% quartet	7
75% quartet	48
Min	1
Max	97.999
Mean	515
std	2.934

Last, as it has already been mentioned, users usually will rate products that they like. This is also the case in this particular dataset, as 62% of the interactions have a rating of 3 stars or more.

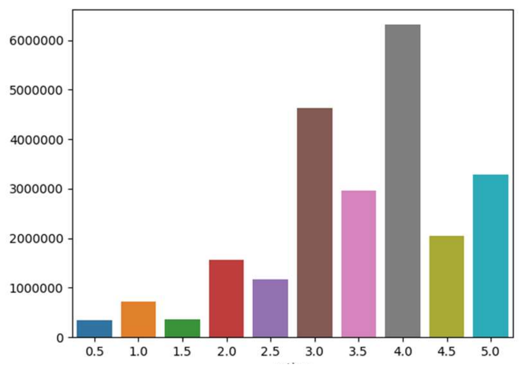


Figure 5 Bar chart of movie ratings

The dataset only contains viewer – movie pairs that have interacted in the past. In order to perform most of the algorithms that will be mentioned later, the data must be reshaped into a pivoted form. To be more precise, the data have to be represented into a matrix form with movies as the columns and the users as rows. The values of the cells are the rating this particular user has given to that movie. The size of this matrix will be the number of users times the number of movies and it will be very sparse as most viewers have only rated a small number of movies.

The aforementioned step demands memory resources. All steps run on a local machine with limited resources, so in order to improve the speed of computations, users and movies were removed from the dataset. By keeping movies that are rated by a minimum number of 300 users and users that have rated at least 50 movies, only 39% of different users is kept in the dataset (109.672) and 13% of different movies (6.964).

However, since the most 'dedicated' viewers and the more popular movies remain in the dataset, only 15% of the total number of interactions were removed.

Last, the data were turned into a pivoted form with sparsity 96.93%, meaning that only about 3% of the cells actually contain any values.

3. Item – to – item recommender systems

In the following section, a selections of algorithms that can generate movie recommendations are presented. First, algorithms that follow simple set based approaches are presented along with the recommended movie ids for a selected movie. Their disadvantages are discussed then a similarity based techniques and a KNN recommender are implemented. Last natural language preprocessing techniques are then introduced and engines are created to produce better recommendations, with LSA producing the most interesting results. In this section error metrics are not used.

3.0 Preliminaries

In the recommendation systems that are presented in this chapter a user needs to be able to provide human input, which in our case is the movie title, and the system must be able to find the id of that movie. Also, since a lot of movies have similar titles, and a user might not be completely sure about the actual title of a movie a string comparison method is needed. One way to implement such technique is by using the *fuzzywuzzy* python library. It performs a fuzzy matching of strings denoting that two strings are similar by giving a similarity index. The Levenshtein distance is then used to calculate the differences between the two sequences. The Levenshtein distance between two words is the minimum number of single character edits (insertions, deletions or substitutions) required, to change one word into another. The distance between two strings a and b is defined as follows:

$$Lev_{a,b}(i,j) = \begin{cases} \max(i,j) & if\ \min(i,j) = 0 \\ min\begin{cases} Lev_{a,b}(i-1,j)+1 \\ Lev_{a,b}(i,j-1)+1 \\ Lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & otherwise \end{cases} \quad (3.0.1)$$

Where $1_{(a_i \neq b_j)}$ is the indicator function, equal to 0 when $a_i = b_j$ and 1 elsewise. $Lev_{a,b}(i,j)$ is the distance of the first i character in a and the first j character in b.

Then the similarity can be calculated as follows:

$$fuzzy\_sim(a,b) = \frac{(|a|+|b|) - Lev_{a,b}(i,j)}{(|a|+|b|)} \quad (3.0.2)$$

The *fuzzywuzzy.fuzz.ratio* method uses this and generates values from 0 to 100, with 0 indicating a complete string mismatch and 100 a complete match. The steps in the algorithm are described bellow.

- Insert movie title
- Generate an empty dictionary of tuples
- Compute fuzzy ratio of all the titles in the dataset
- If the values is above 60, append a tuple of title and ratio value in the dictionary
- Sort the values in descending order of ratio value
- Return the movie title with the highest ratio match

3.1 Set based techniques

The easiest and fastest way one could create recommendations would be by ignoring the rating a user has provided for a movie and only keep this information in a binary form as movie watched/ not watched. Following this idea we can create recommendations by keeping a set of distinct users for every movie. The representation looks like the following:

110 : Braveheart (1995)	262144, 262146, 4, 131076, 131082, 15, 131087, 131095, 262169, …
1907: Mulan (1998)	166713, 232249, 166717, 35647, 101184, 166721, 265025, 68419,..
89745: The Avengers (2012)	198695, 67625, 100393, 231463, 133164, 133166, 34865, 2084, 197,..

The first column here shows the movie id. The second, contains the user ids that have rated that particular movie.

In an early naive approach, we can consider two movies to be similar when there is a large intersection of users that have rated both movies, considering them as users of similar taste. Therefore a first approach is to give recommendations based on the overlap score between two movies.

$$overlap\_score\ (A, B) = |(A \cap B)| \quad (3.1.1)$$

This method is very fast but the drawback is that popular movies are rated by the largest set of users. As a consequence, very popular movies will always have the largest overlap score.

Example:

Selected movie: Mulan (1998)

10 recommendations with the highest score:

- Recommended movie: Toy Story (1995), that shares 72.87% of its viewers
- Recommended movie: Matrix, The (1999), that shares 70.45% of its viewers
- Recommended movie: Lion King, The (1994), that shares 70.39% of its viewers
- Recommended movie: Forrest Gump (1994), that shares 69.58% of its viewers
- Recommended movie: Aladdin (1992), that shares 67.67% of its viewers
- Recommended movie: Star Wars: Episode IV - A New Hope (1977), that shares 66.1% of its viewers
- Recommended movie: Shrek (2001), that shares 64.84% of its viewers
- Recommended movie: Star Wars: Episode V - The Empire Strikes Back (1980), that shares 62.97% of its viewers
- Recommended movie: Jurassic Park (1993), that shares 62.24% of its viewers
- Recommended movie: Beauty and the Beast (1991), that shares 62.09% of its viewers

All of the above movies are extremely popular. We cannot consider this recommendation as being optimal. Matrix, Forrest Gump, Star wars are movies that one would probably not want to watch after watching Mulan, which is an animated film. These movies are in the list because of their popularity.

The most common way of dealing with this problem is by using metrics that are suited for binary data. For example Jaccard similarity index (or Jaccard similarity coefficient) compares members of two sets to see which members are shared and which are distinct. So, it normalizes the intersection size of two sets by dividing with the total number of users that have watched either movie [23]. This measure of similarity between sets of data ranges from 0% to 100% and higher percentage indicates more similar populations.

$$jaccard(a, b) = |a \cap b| / |a U b| \quad (3.1.2)$$

With changes in the algorithm to produce movie recommendations based on distanced data the top 10 recommendations are very different from the previous ones. Also notice that famous animated movies like Shrek and Lion King do not appear in the following list.

Top 10 recommendations of movie Mulan.

- Recommended movie: Tarzan (1999), with jaccard index of 0.2849.
- Recommended movie: Hercules (1997), with jaccard index of 0.2772.
- Recommended movie: Little Mermaid, The (1989), with jaccard index of 0.2591.
- Recommended movie: Emperor's New Groove, The (2000), with jaccard index of 0.2526.
- Recommended movie: Jungle Book, The (1967), with jaccard index of 0.2402.
- Recommended movie: Lilo & Stitch (2002), with jaccard index of 0.2308.

- Recommended movie: Sleeping Beauty (1959), with jaccard index of 0.2288.
- Recommended movie: Bambi (1942), with jaccard index of 0.2221.
- Recommended movie: 101 Dalmatians (One Hundred and One Dalmatians) (1961), with jaccard index of 0.2211.
- Recommended movie: Bug's Life, A (1998), with jaccard index of 0.2199.

There are other set distances that could be used here as well. For example the Dice-Sorensen coefficient could have been used instead. Again it measures the similarity between two sets.

$$dice(a, b) = 2*|a\cap b|/(|a|+|b|) \quad (3.1.3)$$

As both of these indexes are very similar ($D = 2J/(J+1)$ and $J = D/(2-D)$) the produced results end up being exactly the same in this case.

3.2 Similarity based techniques

While the results are starting to look respectable, metrics like the Jaccard distance bias the results, as they penalize movies that have dissimilarities in the number of users in their respective sets.

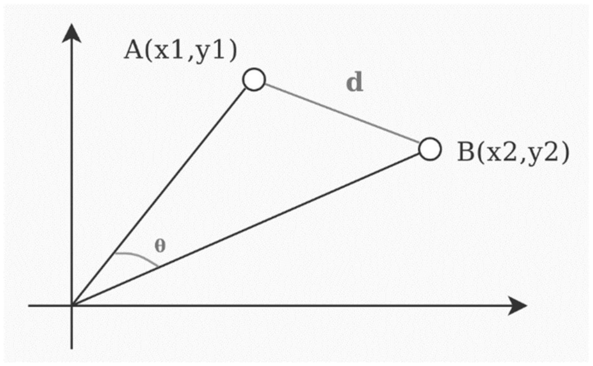
Although set based methods are fast and easy to interpret, they do not use all the available information. A viewer watches a movie and then rates it based on his preferences. However, in set – based methods, a user that dislikes a movie is treated the same as a user who likes the movie.

Cosine similarity is a measure of similarity between two non-zero vectors that is based on the calculation of the cosine of the angle between them. The measure takes into consideration the orientation of the vectors and not their magnitude [24]. In positive space, as the one where our vectors are represented, the values of the cosine are in the [0,1] interval. Vectors are then maximally similar when they are parallel and dissimilar if they are orthogonal. Lastly, it is a commonly used metric in high dimensional spaces such as the one we are representing our data.

Cosine similarity is generally used as a metric for measuring distance when the magnitude of the vectors does not matter. This happens for example when working with textual data represented by word counts. We could assume that when a word (e.g. science) occurs more frequent in document 1 than it does in document 2, that document 1 is more related to the topic of science. However, it could also be the case that we are working with documents of uneven lengths. Then, science probably occurred more in document 1 just because it was way longer than document 2. Cosine similarity corrects for this.

Although the term “distance metric” is commonly used to describe cosine similarity, it should not be confused with a distance metric. Since cosine similarity does not follow the constraints of the Schwarz inequality, it is not considered to be a distance in the formal sense.

Cosine similarity is advantageous to the Euclidean distance in our case, because two similar movies (in terms of viewers) would appear close, even if they differ in size, since in cosine similarity the closer the vectors are by angle, the higher the cosine similarity.



The mathematical formula for cosine similarity is:

$$cos\theta = \frac{\vec{a}\cdot\vec{b}}{(\|\vec{a}\|\cdot\|\vec{b}\|)} = \frac{\sum_{i=1}^{n} a_i b_i}{(\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2})} \quad (3.2.1)$$

Where = $\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$

To calculate the cosine similarity, all movies are mapped as sparse vectors. The values of these vectors are the ratings every user in the dataset has given to that particular movie. This leads to a set of 6853 movies in the 108K dimensional space.

The computations of the metric are not very intensive, because only the values that appear in the vectors are taken into consideration, and this amount is significantly lower than 108K. However since the metric must be computed pairwise for every movie pair, the number of combinations is very high and the algorithm needs a lot of time to complete. This problem can be tackled by performing the computations in parallel, as the value of every pair is only relevant to that pair and only needs to be stored after being calculated.

Again a set of 10 movies with the highest similarity score to the movie Mulan are produced and the recommendations are the following:

- Recommended movie: Tarzan (1999), with cosine similarity 0.5156.
- Recommended movie: Hercules (1997), with cosine similarity 0.4949.

- Recommended movie: Little Mermaid, The (1989), with cosine similarity 0.4677.
- Recommended movie: Emperor's New Groove, The (2000), with cosine similarity 0.4452.
- Recommended movie: Anastasia (1997), with cosine similarity 0.4296.
- Recommended movie: Bug's Life, A (1998), with cosine similarity 0.4182.
- Recommended movie: Lilo & Stitch (2002), with cosine similarity 0.4152.
- Recommended movie: Lion King, The (1994), with cosine similarity 0.415.
- Recommended movie: Beauty and the Beast (1991), with cosine similarity 0.4141.
- Recommended movie: Sleeping Beauty (1959), with cosine similarity 0.414.

The main advantage of using cosine similarity, is that it does not take into consideration the magnitude of the vector. This can lead to a lot of noise in the results, because movies with a very small audience can be generated in the results with high scores. This would happen when the audience of these movies has watched a small list of other movies as well. The movies with the small audience will get a high similarity score with the rest, as they share their audience.

3.3 Neighborhood based techniques

Machine learning (ML) is the scientific field of algorithms and statistical models that leverage computing power to perform specific tasks. These tasks usually rely on patterns that emerge from the data and no specific function that is used to generate explicit values is implied.

Machine learning algorithms can perform a plethora of different tasks. To perform a task, the user must define the learning style the algorithm will follow. The learning styles can be categorized as follows:

- Supervised learning
  In this case the task of learning a function, maps the input to an output, based on example input – output pairs.
  Those training examples form the training data which are labeled. They consist of the input (which is usually a vector of feature values) and the desired output – the supervised signal. Then the core objective of the algorithm is to generalize from the data and to infer the underlying process (function) that generated those data. Optimally, the algorithm will find the function and will correctly generate output values on new examples, on which the algorithm has not been trained. Tasks like Regression and Classification fall under this category.

- Unsupervised learning
  In this class, the system is presented with unlabeled, uncategorized data and the algorithms run on the data without prior training. Unsupervised learning algorithms can perform more complex processing tasks than supervised learning systems. However, unsupervised learning can be more unpredictable than the alternate model.
  Clustering and dimensionality reduction are the most famous classes of algorithms in unsupervised learning.
- Semi-supervised learning
  It is a class of machine learning tasks techniques that make use of unlabeled data and only a small amount of labeled data for training.
  This class falls between unsupervised learning (without any labeled data) and supervised learning (with completely labeled training data).

KNN or K nearest neighbors algorithm falls under the supervised learners hood. However it does not fully comply with the definition that is stated above. KNN is a lazy learner [25]. That translates into the fact that this algorithm does not need to learn a discriminative function from the training data but instead, it 'memorizes' the data themselves.

KNN is a classification algorithm that has a set of input features and a labeled target output. An example of a classification problem, is given the age, height and weight of a person to try to infer the sex of that person. Age, weight and height are called the set of predictors, while sex is the target label.

age	weight	height	Sex (target)
20	51	165	Female
25	77	178	Male
21	70	180	Male
29	54	160	Male
…	…	…	…
32	60	154	Female

The KNN algorithm, assumes that similarity exists in close proximity. In other words, similar items lie next to each other. For this algorithm to be useful, we must hinge on this assumption being true enough for the dataset on hand.

Steps of the KNN algorithms

- Load the data
- Initialize K, which is the number of the neighbors
- Calculate the distance between the current example and all the previous examples
- Add the distance and the index of the example in an ordered collection
- Sort the ordered collection of distances and indices in ascending order by the distances
- Pick the first K entries from the sorted collection along with their labels

- Classify new records with the mode of their K labels

The advantages of using KNN algorithm is that we do not need to build a model or tune several model parameters. However, while the number of examples and/or predictors increases, the algorithm gets significantly slower, which is KNN's main disadvantage. This makes it an impractical choice in environments where predictions need to be made rapidly and on big volume of data.

Provided we have sufficient computing resources to speedily handle the data we are using to make predictions, KNN can still be useful in solving problems that have solutions which depend on identifying similar objects. An example of this is using the KNN algorithm in recommender systems, an application of KNN-search [26].

Collaborative filtering recommender systems use the actions of users to recommend to others. In this section an item - based approach is employed with a KNN model.

The steps followed are described below:

- Transformation of the ratings data frame into an n by m matrix of n users and m movies. The values of the matrix are the actual rating of the movie of the particular user. If the value is missing, the cell is filled with 0s, so that linear operations can be performed later on.
- As not all users interact with all movies the data are very sparse. For this reason the matrix is transformed into a sparse representation with the use of *sparse* module of the *scipy* library in Python
- The training data are highly dimensional and the performance of the algorithm will suffer from the curse of dimensionality if we use the Euclidean distance as its objective function. Euclidean distance is unhelpful in high dimensions because the vectors are almost equidistant to the target movie's features. Instead the cosine similarity tackles that problem and is better suited for K nearest neighbor search [27].

Following the instructions of the first step, the data are pivoted with rows representing movies and columns representing users. Every cell with missing values is filled with a 0. In order to make the calculations faster and take advantage of sparsity in the data we turn the matrix into a sparse representation using the *sparse* module of the *scipy* python library. In a sparse representation only the non zero values are kept in memory and the calculations are performed on them.

In this thesis, the selected learner is implemented using the *NearestNeighbors* function of the neighbors module in the *sklearn* machine learning python library. The algorithm, the metric and the number of neighbors had to be defined.

The algorithm of choice is "brute force" because we want to calculate the pairwise distances of all points in the dataset. This is a very competitive choice over other approaches, as brute force algorithms do very well when the size of the data set is not very big. As for the metric, we are using the cosine similarity because of the dimensionality of the data is so big, that using the

Euclidean distance would not be a preferred option. Lastly we can set the number of neighbors to different integers, but 20 is the number of choice.

The user then inputs a title and the algorithm returns the top 10 movies with their respective distances.

Top 10 movies for Mulan using a KNN approach are the following:

- Sleeping Beauty (1959), with distance of 0.5867
- Beauty and the Beast (1991), with distance of 0.5864
- Lilo & Stitch (2002), with distance of 0.5856
- Lion King, The (1994), with distance of 0.5856
- Bug's Life, A (1998), with distance of 0.5825
- Anastasia (1997), with distance of 0.5709
- Emperor's New Groove, The (2000), with distance of 0.5557
- Little Mermaid, The (1989), with distance of 0.5328
- Hercules (1997), with distance of 0.5057
- Tarzan (1999), with distance of 0.4852

3.4 Text processing techniques

Singular value decomposition or SVD is a matrix decomposition technique. It a technique used to decompose the original matrix of ratings A into three matrices such that

$$A = U \times \Sigma \times V^T \quad (3.4.1)$$

The technique is described in more detail in chapter 4.2, but we have to be aligned with the general idea before moving further.

Matrix $\Sigma \in R^{kxk}$ is a diagonal matrix. In its diagonal are stored the singular values of the original matrix $A \in R^{mxn}$. The singular values of a non square matrix, are the eigenvalues of matrix $A^T A \in R^{nxn}$. A common convention in this technique is keeping the list of singular values in Σ in decreasing order. We can keep a limited number of singular values, by setting the value of k to a small number thus performing dimensionality reductions on the data.

Matrices U and $V \in R^{mxk}$ are both orthogonal. An orthogonal matrix, is a matrix with a norm that equals 1.

Since the matrices that are generated, have only limited amount of features, practically dimensionality reduction is performed on the item – user space. In other words, a low rank representation of the original input matrix is generated, that helps us gain insights of the data.

The use of SVD in this way, is called Latent Semantic Analysis (LSA) or Latent Semantic Indexing [28] [29].

LSA is a very famous technique used in the field of Natural Language Processing (NLP). It is regularly used to analyze the relationship between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. The main assumption is that words that are closely related in meaning will occur in similar pieces of text (similar documents).

The process is the following.

- Find the set of unique words that appear in the set of documents.
- Create a matrix, where each row represents a distinct word and each column a document.
- Fill the cells of the matrix with the word count of this word in the document.
- Decompose the matrix using SVD, reducing the number of lines while preserving the similarity structure among column.
- Compare two documents by taking the cosine of the angle between the two vectors formed by any two columns.
- Similar documents get values that are close to 1, while dissimilar documents get values close to 0.

Naturally, this technique has been used in content based recommender systems.

In our example of movie recommendation, a set of features can be created such as genre, runtime, release year, director, Oscar winning movie, starring actors, movie synopsis etc. All these values generate a text, and this text can be used as the input for the model.

However in the analysis that is presented in this thesis, we are not using any extra features apart from the rating information for a user – movie pair. This pair is represented in pivoted form. A different way of thinking, is to think of each user as a word, and the rating that this user has given to a specific movie, is then the “word count”. Following this idea, a recommender system can be built.

The steps in this item – to – item recommendation engine that was created are the following:

- The original ratings matrix along with the number of latent features is given as input by the user.
- SVD is performed (using *svds* function from *linalg* module of *scipy* package). It outputs the movie factors matrix, the viewers factor matrix and the diagonal matrix containing the singular values of the original rating matrix.
- The L2 norm of every vector of movie factors is computed and the values are kept into an nx1 vector, where n is the number of movies in the dataset.

- Each of the movie latent factors gets derived by its norm.
- The user inputs a movie name
- Fuzzy text matching is performed and the movie id is found in the dataset
- The vectors of the latent features of the selected movie is derived
- The dot product of all movie factors and the selected movie is calculated. Hence, the cosine similarity between all movie vectors and the selected movie vector are calculated, since all vectors are already divided by their corresponding L2 norm.
- The data regarding movie id and the cosine are stored into a data frame with descending order of cosine similarity
- Top N recommendations are output on the console.

Interesting recommendations are produced for the movie Her. It is a sci-fi, drama and romance movie, where a writer falls in love with an operating system. The system run with 40 latent factors

- Recommended movie: Whiplash (2014), with score: 0.9165.
- Recommended movie: Birdman: Or (The Unexpected Virtue of Ignorance) (2014), with score: 0.8998.
- Recommended movie: Grand Budapest Hotel, The (2014), with score: 0.8821.
- Recommended movie: Gone Girl (2014), with score: 0.8745.
- Recommended movie: Dallas Buyers Club (2013), with score: 0.8677.
- Recommended movie: Boyhood (2014), with score: 0.8395.
- Recommended movie: Ex Machina (2015), with score: 0.8322.
- Recommended movie: Nightcrawler (2014), with score: 0.8291.
- Recommended movie: Room (2015), with score: 0.7943.
- Recommended movie: Wolf of Wall Street, The (2013), with score: 0.7824.

Increasing the number of latent features to 60 included Moonrise Kingdom (2012) and Gravity (2013) to the resulting movies, while decreasing the number of latent features generated movies that included Wild Tales (2014), Spotlight (2015), The Lobster (2015) and Captain Fantastic (2016).

According to my personal preference, the recommendation engine with 20 latent features did a great job, recommending some out of the box movies that were never recommended by other systems.

Another way to create a recommender system, that has its roots in the field of text processing, is the TFIDF technique.

There are some amazing models in the field of information retrieval, that are used for calculating the similarity between query strings and text documents. Following the idea behind the LSA recommender system, we can easily adapt these models to our purpose, by treating each movie as a document and each user as a term in those documents.

Again, this is a technique that is preferred when working with actual text data, so it is usually found in content based filtering algorithms [30].

The concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) are used in information retrieval systemς. It is used to determine the relative importance of a document / article / news item / movie etc.

TF, or term frequency, is the frequency of a word in a document. While IDF, or inverse document frequency, is the inverse of the document frequency amongst the whole corpus of documents. The technique is mainly used for two reasons: Suppose we are searching using the following query on Google “Data Science Jobs in Thessaloniki”. It is certain that the word “in” will occur a lot more times than the word “Thessaloniki”, but the relative importance of the word “Thessaloniki” is higher in the search query point of view. TF-IDF practically penalized words with high frequency, determining the importance of an item.

A function that is usually used to penalize large values is the logarithmic function. Therefore the term frequency gets weighted by the logarithm. The formula of calculations is the following:

$$w_{t,d} = \begin{cases} 1 + log_{10} tf_{i,d} & if\ tf_{i,d} > 0 \\ 0 & elsewise \end{cases} \quad (3.4.2)$$

Where

- t is the term
- d is the document
- $tf_{i,d}$ is the term frequency of term t in document d
- $w_{t,d}$ he weighted term frequency of term t in document d

For example

TERM FREQUENCY	WEIGHTED TERM FREQUENCY
0	0
10	2
1000	4

It is clear that the effect of high frequency words is highly reduced and the values are more comparable as opposed to the original values of the term frequency.

IDF, or inverse document frequency is the second parameter of the TF-IDF algorithm. Having its basis on the principle that less frequent words are generally more informative, it helps us find the relevance of the words. The formula of the calculation is

$$IDF = log_{10}(N/DF) \quad (3.4.3)$$

Where

- N is the number of documents in the corpus
- DF is the number of documents in which we can observe at least one occurrence of the word

Then, by multiplying the relevance of the words (IDF) by the occurrence of the words in the document (TF) we get the TF-IDF score. The similarity of the query and the document is the TF-IDF distance which is just the cosine of the weighted vectors.

So, again we can consider the pivoted user - item interaction matrix as the matrix of item appearance in documents. Users are considered as the words and items (and in our case movies) as documents.

The problem that all the aforementioned algorithms are facing, is that they are ignoring the overall activity of each user. There are many users of the system, that only watch a handful of movies and probably only rate them highly. These users are treated the same as users that watch everything and their ratings range between all possible rating values. For purposes of calculating similarity, we want to weight people that are selective in what they are watching more than users that are not. IDF can achieve that by multiplying the logarithm of the inverse probability that a user will rate a movie.

To create a TF-IDF item – to – item recommendation engine, the following steps were executed:

- The original user – item interactions matrix is taken as input by the human
- A binary matrix based on the original matrix is created.
- Number of views per movies and total number of movies viewed per user is saved into data frame format
- IDF value for every user is calculated by dividing the total number of movies by the total number of movies that user has rated and taking the logarithm of that value
- TF is calculated on the original ratings matrix by taking the log of every observation and then adding the binary matrix values
- TF-IDF is computed by multiplying every row of TF matrix by the corresponding IDF
- The user then inputs a movie to calculate similarities
- The id of the movie is found using fuzzy text matching
- The cosine of the movie TFIDF vector and every other movie's TF-IDF vector is calculated
- Movie ids and cosine similarities are stored into a data frame with decreasing order in terms of similarity magnitude
- Top N recommendations are output on the console

The output of the model is very similar to LSA's output, there it is not printed here.

4. Matrix factorization techniques

4.1 Alternating least squares

Matrix factorization is a class of collaborative filtering algorithms that are used in the field of recommender systems. The idea is to decompose the user – item interaction matrix into two lower dimensionality matrices in a lower dimension latent space.

The original algorithm was proposed in 2006 during the Netflix Prize challenge by Simon Funk and became popular because of its effectiveness. Since then, many new and more sophisticated algorithms were constructed based upon his idea.

As it has been already stated, usually when creating a recommender system we have some sort of implicit or explicit data. These data are represented in a form of a matrix with rows representing users and columns representing items. Obviously these matrices are very sparse since not everyone can cover the whole list of items.

Here lies the strength of the matrix factorization algorithms. They can incorporate explicit feedback, that is information that is not directly given but can be derived from the data by analyzing user behavior [31]. Using this we can estimate if a user is going to like an item or not , even if they have not interacted with and provide a rating for this specific item. If the estimated rating is high, then this product will be recommended to the user.

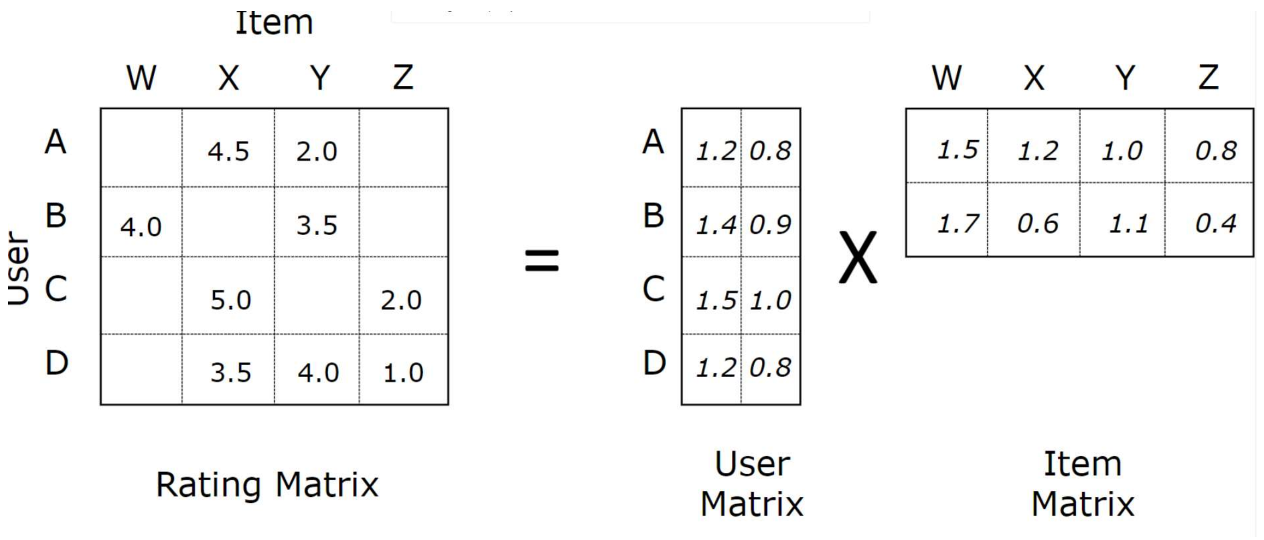


Figure 6 Visualization of matrix factorization technique

The image above is great on summarizing the core idea behind matrix factorization.

Practically, the idea is to take a high dimensional matrix and split it into two lower dimensionality matrices, with a dot product that equals the original matrix. It can be thought of as having a large integer number and factoring it into the product of smaller primes.

In recommender systems, the original matrix R has millions of different dimensions. However human taste is not nearly as “complex”. Even if a person would see hundreds of different items, he would express just a couple of different tastes. So in the context of collaborative filtering, we can explain the matrix factorization as reducing the dimension of all the items – all users matrix, into a smaller all items by some taste dimensions and all users by some taste dimensions. These dimensions are called latent hidden features and are learned from the data.

With this reduction, we are going to be working on fewer dimensions. Our computations will be more computationally efficient and also we are going to be getting better results because we can reason about items in this more compact “taste-space” [32].

If we can express each user as a vector of their taste values, and at the same time express each item as a vector of what tastes they represent, we can quite easily make a recommendation. This also gives us the ability to find connections between users who have no specific items in common but share common tastes.

It should also be noted that the latent features or tastes cannot be interpreted by a human as they can only be represented by a mathematical formula. We will not be able to label them “comedy” or “black and white” or “starring Jude Law”. These latent features don’t necessarily reflect any real metadata.

These features and their relevant values can be learned from the data. This is achieved by minimizing a loss function.

In the field of linear algebra, matrix factorization is a form of optimization process that aims to approximate the original matrix R, with two matrices X and Y, such as the following cost function is minimized.

$$L = \|R - XxY^T\|_2 + \lambda(\|X\|_2 + \|Y\|_2) \quad (4.1.1)$$

The first term in the formula of the loss function is the mean squared error (MSE) distance of the original matrix R and its approximation XxY^T. The second term is called a “regularization term” and is added to govern a generalized solution, to prevent overfitting to the local noise effects of the ratings.

The cost function introduces two new parameters: k and λ. While we are trying to minimize the loss function for given k and λ values, at the same time, it is essential to determine the optimal values for those parameters as well.

The loss function can be minimized with the help of gradient descent algorithm. Gradient descent is an optimization algorithm that is widely used in the field of machine learning. It is an iterative optimization process, that assumes the existence of a cost function and arbitrary initial values for the optimization variables. In every iteration, the gradient of the cost function is re-computed with respect to the optimization variables. The variables are then updated. The target of this process is to minimize the cost function, until the later converges to a minimum point. However, the gradient descent method can only guarantee convergence to a local minima [33].

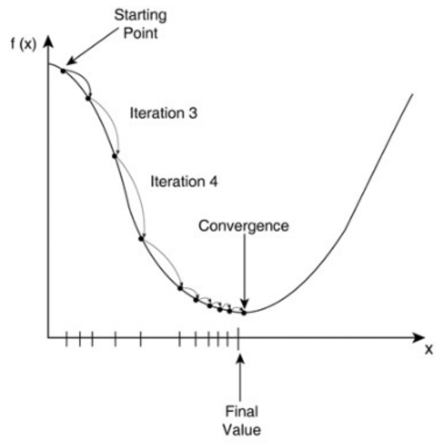


Figure 7 Visualization of gradient descent

Looking at the cost function of the matrix factorization algorithm, it appears that the aim is to learn two types of variables: those of X and those of Y. Since the cost function is the sum $\|R - XxY^T\|_2 = \sum_{u,i}(R_{ui} - x_u \cdot y_i)$ plus the regularization term, the fact that both X's and Y's values are unknown makes the cost function non convex.

We can, however, consider the following: If we fix the values of Y and optimize only for the values of X alone, the problem is reduced to a linear regression problem. In linear regression we solve the equation:

$$Y = \beta X + \varepsilon \text{ (4.1.2)}$$

Where

- Y is the vector of observed variables or the dependent variables
- X is a matrix of row vectors p_i, which are known as the independent variables
- β is the parameter vector, with values known as the regression coefficients and
- ε is the error value or noise

In linear regression we solve for β by minimizing the squared error

$$L = \|X\beta - Y\|^2 = \sum_{i=1}^{n}(\beta x_i - y_i)^2 \text{ (4.1.3)}$$

The loss functions is solved as:

$$L = \|X\beta - Y\|^2$$
$$= (X\beta - Y)^T(X\beta - Y)$$

$$= Y^TY - Y^TX\beta - \beta^TX^TY + \beta^TX^TX\beta \quad (4.1.4)$$

The loss is convex so the optimal solution lies when the gradient is at zero value. The solution is given by ordinary least squares (OLS).

$$\frac{\partial L}{\partial \beta} = \frac{\partial(Y^TY - Y^TX\beta - \beta^TX^TY + \beta^TX^TX\beta)}{\partial \beta}$$

$$= -2X^TY + 2\,X^TX\beta \quad (4.1.5)$$

Setting the gradient to 0 we get

$$-2X^TY + 2\,X^TX\beta = 0$$

$$X^TY = X^TX\beta$$

$$\beta = (X^TX)^{-1}X^TY \quad (4.1.6)$$

Alternating least squares is a two-step iterative optimization process that takes advantage of the convergence of OLS. In every iteration, first it keeps the values of X fixed and solves for Y and then does the same in reverse, keeping the values of Y fixed and solving for X. Since the OLS solution is unique and guarantees a minimal MSE, in either step the cost function can either decrease or stay unchanged but never increase. Alternating between the two steps guarantees a reduction of the cost function until convergence. Similarly to gradient descent optimization, it is guaranteed to converge to a local minima, and it ultimately depends on initial values for X and Y.

So, let R be the user – item interaction matrix, X the user matrix of k latent features and Y the item matrix of k latent features. Then, user u is represented by the vector x_u, while the item i is represented as y_i.

The actual rated score of user u on item i is r_{ui}. The prediction $\hat{r}_{ui}$ for the true rating can then be calculated as:

$$\hat{r}_{ui} = x_u^Ty_i = \sum_k x_{uk}y_{ki} \quad (4.1.7)$$

Where x_u^T is a row vector and y_i is a column vector. These are the latent vectors or low dimensional embeddings.

By minimizing the square of the difference between all ratings in the dataset (S) and their predictions, a loss function is produced. The formula of the function is:

$$L = \sum_{u,i\in S}(r_{ui} - x_u^T \cdot y_i) + \lambda_x \sum_u \|x_u\|^2 + \lambda_y \sum_i \|y_i\|^2 \quad (4.1.8)$$

In the end of the formula two L_2 regularization parameters are added to prevent overfitting on user or item data.

The goal is the minimization of the aforementioned function. This is achieved through gradient descent. At each step of ALS minimization, a set of latent vectors is kept constant. For simplicity let us assume that in the first step the item vectors are constant. Then the derivative of the loss function is calculated with respect to the other set of vectors (the user vectors). Because we are searching for a minimum, the derivative is set to equal zero and we solve for the non-constant vectors (the user vectors).

The alternating part, is the one that follows. After solving the derivative, the vectors for users are updated. We now hold them constant and take the derivative of the loss function with respect to the previously constant vectors (the item vectors). We alternate back and forth between these two steps until convergence.

To dive deeper into the math of the technique, let us hold the item vectors (y_i) constant and take the derivative of the loss function with respect to the user vectors (x_u).

$$\frac{\partial L}{\partial x_u} = -2\sum_i (r_{ui} - x_u^T y_u) y_i^T + 2\lambda_x x_u^T$$

$$0 = -(r_u - x_u^T Y^T)Y + \lambda_x x_u^T$$

$$x_u^T (Y^T Y + \lambda_x I) = r_u Y$$

$$x_u^T = r_u Y (Y^T Y + \lambda_x I)^{-1} \text{ (4.1.9)}$$

Let us assume that the original dimensionality of the interaction matrix is n by m, denoting that the dataset consists of n users and m items. Also let us assume, that the number of latent features is k.

Symbol Y, is used to refer to m by k representation of all item row vectors, that are vertically stacked on top of each other. The row vector r_u, is used to represent the row in the interaction matrix for user u. It contains all the ratings the user has given to all items. Its dimensions, therefore are a vector with length m. I is the identity matrix (unit diagonal matrix) with dimensions k by k.

So, for the kth dimension of the user's latent vector, the formula would be transformed into:

$$x_{uk} = r_{ui} Y_{ik} (Y_{ki} Y_{ik} + \lambda_\chi I_{\kappa\kappa})^{-1} \text{(4.1.10)}$$

Similarly, the derivation of the item vector is calculated.

$$\frac{\partial L}{\partial y_i} = -2\sum_i (r_{ui-} y_i^T x_u) x_u^T + 2\lambda_y y_i^T$$

$$0 = -(r_i - y_i^T X^T)X + \lambda_y y_i^T$$

$$y_i^T(X^TX + \lambda_y I) = r_i X$$

$$y_i^T = r_i X(X^TX + \lambda_y I)^{-1} \quad (4.1.11)$$

The pseudocode of the algorithm is provided below:

```
Algorithm 1: ALS for matrix factorization – ALS step
Initialize with random values X and Y
repeat:
    for u = 1, …, n do:
        $x_u = (\sum_{r_{ui}\in r_{u*}} y_i y_i^T + \lambda I_k)^{-1} \sum_{r_{ui}\in r_{u*}} r_{ui} y_i$
    end for
    for i = 1, …, m do:
        $y_i = (\sum_{r_{ui}\in r_{*i}} x_u x_u^T + \lambda I_k)^{-1} \sum_{r_{ui}\in r_{*i}} r_{ui} x_u$
    end for
until convergence
```

To train the algorithm and assert how the algorithm is doing we must first split the dataset into a training and a test set.

Since we have kept in the dataset only users with a minimum of 50 rated movies, 10 random movies that a user has rated are selected from every user, and are kept hidden. Then the data are reshaped into a pivoted form and two matrices are created, the training and the test set.

Notice, that this is not a classical split into an 80% - 20% training – test sets approach, but this is the approach that inspired the above defined split actions. In more detail, since 20% of 50 is 10, with 50 being the minimum number of distinct rated movies by a user, the number of 10 ratings for removal is selected. Ideally, to create an 80% - 20% split, the number of movies each user has rated should be multiplied by 0.2. However, we do not want to lose a large amount of data, since the data already consist of very sparse examples, and therefore the amount for movies to be removed was kept constant to 10. Also, for the test set, we wanted the number of examples for each user to be kept constant, so that we can assert the performance of the model on users with a limited amount of examples.

We also need an error metric to be optimized. This metric was chosen to be the mean squared error, since it is the most commonly used metric when calculating the performance of a recommender system. This is an outcome of this being the selected metric for optimization during the Netflix prize challenge, where participants were asked to reduce the metric by 10% over that of the then algorithm in use. Since then, MSE has remained the most liked metrics for the task.

The steps of the algorithm are defined in detail bellow.

First the system is initialized with the two matrices of latent vectors – the user and item latent features. These matrices are shaped n by k and k by m respectively, where n is the number of users, m is the number of items and k is the defined number of latent features. The matrices are initialized with random values generated by a normal distribution and they should lie in the same number interval of the actual ratings values.

After the system's initialization, the alternating squares step has to be defined. The algorithm goes into a loop, calculating the derivative for items and then users. So, in each step the latent and the fixed vectors are reversed and the corresponding lambda value is provided to the algorithm. The dot product of the fixed vectors is calculated and multiplied by the regularization parameter lambda. Then, for every point in the latent vectors, the loss function is minimized. This is achieved by computing the dot products of the ratings row or column that corresponds to the selected user or item respectively, by the fixed vector and solving against the dot product of the fixed vectors and the lambda of the eye matrix. To solve the loss function, the fast computation of the solve function from *numpy's linalg* module was selected. After solving, the values of the latent vectors are updated and returned for the next iteration step.

On the next step of the algorithm, the training begins. The number of iterations is defined by the user.

Next, the algorithm should be able to generate predictions. Two modules are defined: predict and predict all. Predict, works on a defined user and a defined item. Since the user and item vectors along with their trained values in their corresponding latent factors are defined, the prediction is the dot product of these two vectors. Predict all, loops over every user and every item and fills the values on a new matrix, with the same dimension as the original input matrix.

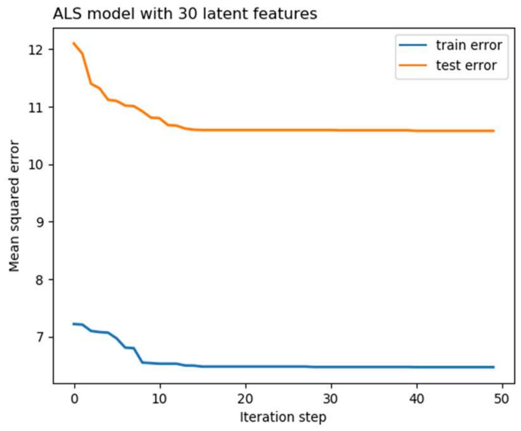
The predictions for every user – item combinations are calculated every time a full iteration of the algorithm is completed. Predictions are generated both for the test and training set. On this step, the error metric, mean squared error, is calculated separately over the actual data and the train set and over the actual data and the test set. The values are saved to be able to assert the model's performance over every iteration.

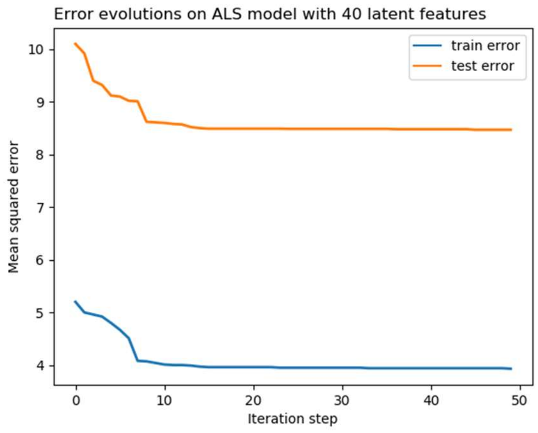
To keep track of the resulting error, a plotting function was also defined. It kept track of the error of the training and the test sets and how it evolved as the number of iterations increased.

All the computations were performed on local machine with 16GB RAM and on an intel i7-75000 processor. Since there are so many loops that are performed during every iteration the training time of the algorithm was very long. In more detail, training of the algorithm was performed with 30, 40 and 50 latent features and the training time was 6 and a half days, 8 and 10 days respectively.

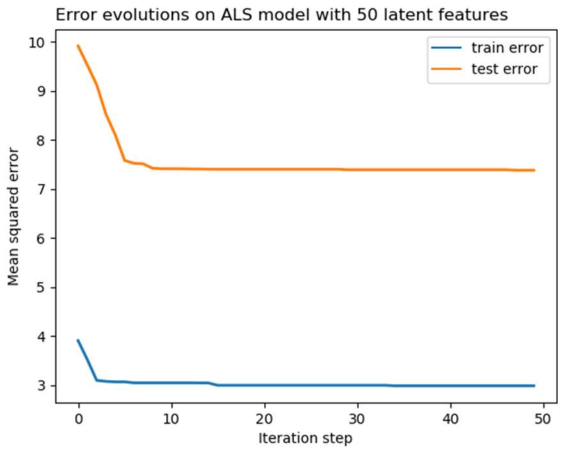
Because of the long training times, parameter tuning and optimization was very hard to perform.

On the following plots, we can observe the evolution of training and test MSE as the number of training iterations was increasing for models with different number of latent features. More specifically, tests were conducted for models with 30, 40 and 50 latent features. All three model run for a total of 50 iterations and had their regularization parameters set to zero for both item and user vectors.



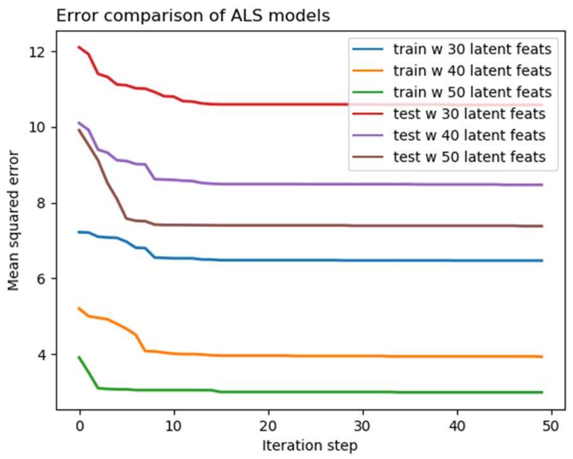


The model that was initialized with a total of 30 latent features started with a test MSE of 12.10 which started stabilizing around 10.6 after the 12th iteration of the algorithm. The training set had a much lower error on the first iteration (7.21), but it did not decrease by the same amount as the test error. The values of the training MSE were stable at a mean value of 6.5 after the 8th iteration step.



A similar behavior can be seen on the model that was initialized with 40 latent features. However the second plot differs significantly from the first, because there is a large drop on the 8th iteration on both train and test set, denoting that the algorithm's behaviors is becoming stable after that iteration.

At last, the alternating least squares recommender system was initialized with 50 latent features. At this test, the behavior of the system changed, as both train and test set errors decreased fast during the first 5 iterations and then they were stable for the rest of the iterations.



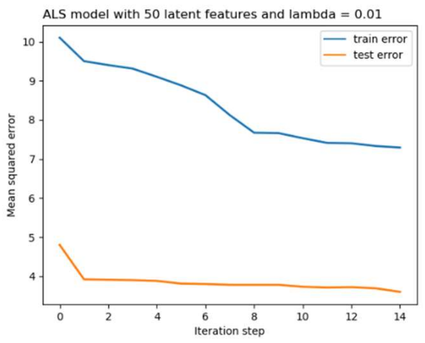
So, we can assume that increasing the number of latent features, the system is able to infer more accurate rules for both item and user behavior and generate more accurate results.

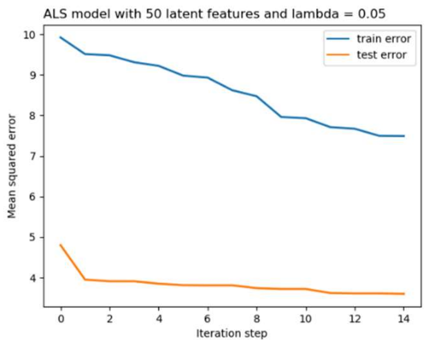
However, we do not want this number to increase dramatically, as it is also performed as a dimensionality reduction technique, so that the space of the features is a lot lower than the initial space.

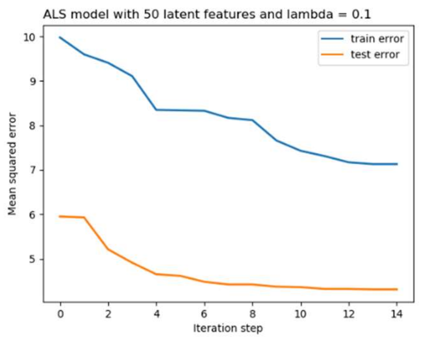
Testing on models with more than 50 latent features was not performed on that model because of the increase in training times.

All of the models were trained without regularization parameters. This lead to model overfitting which can be seen in all three plots. The test set MSE is always higher than the train set MSE, with values higher than those of the test set error more that 50%.

Since the model that worked best for this dataset is the model with 50 latent features, we further tested it by adding a regularization parameter, to alleviate the overfitting of the model. All of the models were tested up to 15 iterations to decrease the training time. Also the lambda values were set to the same number for both user and item matrices.



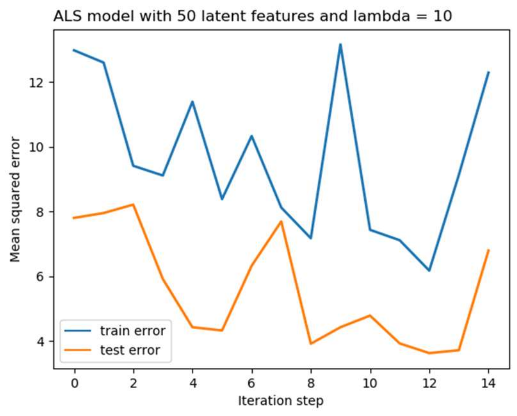




First, three models were generated with regularization parameters of 0.01, 0.05 and 0.1 values for both item and user matrices. The error values of the ALS model with lambda 0.05 seems to not having converged yet. They would probably get lower with the completion of more iterations.

In the model of 0.1 lambda, the interval between training and test set errors is the lowest of all three models. This is the model with the lower overfitting degree.

Of all three models, the model with the highest regularization parameter was fitting best the data.



Another model, with a lambda value of 10 was trained, with 50 latent features for 15 iterations. Here the resulting diagram is very different from the previous ones. The regularization parameter made the model unstable and did not let it reach convergence. The errors did not follow a decreasing pattern, but instead the MSE of both training and test set are alternating values in large values interval.

Ideally, to complete the analysis a grid search would be required.

Grid search is a technique, applied across machine learning algorithms, to calculate the best hyperparameters to use on a given model and a dataset. Via scanning of the data, the optimal values for the models hyperparameters can be defined. This technique is extremely computationally expensive, because the grid search is built on the model and calculates every possible combination of parameters. It stores every model with every possible parameter combinations per iteration.

To tune the hyperparameters of the ALS recommender system that was created, we should test over combinations of different latent features, distinct lambda values for both user and item matrices and different number of iteration steps. The model with the best overall score in terms of error would then be the best model and get selected. The above test was not performed, as the tests that have already been defined took very long to complete.

The model with 50 latent features on its 15th iteration with regularization of 0.1 for user and item matrices was considered to be the best.

4.2 Singular value decomposition

An alternative to the ALS method is Singular value decomposition or SVD. SVD is a matrix decomposition technique that has its roots in linear algebra. It decomposes the original matrix A into three matrices such that A = U x Σ x V^T where

- U∈ R^{mxk} and V∈ R^{mxk} are both orthogonal matrices (columns are orthogonal and their norm equals 1)
- Σ∈ R^{kxk} is a diagonal matrix with the singular values of A on its diagonal. The singular values of a matrix A∈ R^{mxn} are the square roots of the eigenvalues of matrix $A^TA \in R^{nxn}$. A common convention is keeping the list of singular values in Σ in descending order

Assuming that A is a matrix of user – item interactions, then each row in U would correspond to a user characteristic and each row in V to an item characteristic. For example $u_{i,k}$ is the number that quantifies the membership of user i to the characteristic k. Since U and V form orthogonal bases, the overall strength of characteristic K in the interaction matrix can be either deducted from U or V.

An interaction in matrix A, say A_{ui}, can be explained by a set of independent categories. For example for user u and item I, the interaction A_{ui} is decomposed to the following sum $\sum_{k=1}^{K}(u_{uk} * \Sigma_{kk} * v_{ki}^T)$ (4.2.1)

where

- U_{uk} is the kth latent factor value for user u
- v_{ki}^T is the kth latent factor value for item i
- Σ_{kk} is the overall weight of kth latent factor

It follows that any interaction between user u and item i is affected by K latent factors. These factors are each decomposed into the product of $u_{uk} * v_{ki}^T$ and the overall effect of this dimension Σ_{kk} on interactions across all users and items.

Lastly, since Σ is ordered by the size of the singular values of A in descending order, the cumulative sum $\sum_{k=1}^{K} \Sigma_{kk}$ accounts for the total variance (effect of the interactions), as is explained by the K strongest effects. Therefore, it is easy to make a rough assumption of what should the rank of A be (how many factors affect the interaction).

However, the above algorithm is practically impossible to implement on a recommender system as the original matrix A must have all of its values defined.

If A was dense, both U and V could be calculated easily. The columns of U are the eigenvectors of A^TA and the columns of V are the eigenvectors of AA^T. The associated eigenvalues make up the diagonal matrix Σ. While A is dense, there are many packages in various computing languages that can efficiently complete that task.

On the other hand, in the case of recommendation systems, the user – item interaction matrix is usually very sparse, therefore the matrices $A^T A$ and AA^T do not exist, and consequently, no complete SVD can be performed. Following the non existence of matrices $A^T A$ and AA^T the eigenvalues and eigenvectors of these matrices do not exist either and it is impossible to factorize A by the product UΣV^T. But, there is a way around. A first option that was used for some time is to fill the missing entries of A with some simple heuristic, e.g. the mean of the columns (or rows). Once the matrix is dense, we can compute its SVD using the traditional algorithm. This works OK, but the generated results are usually highly biased.

Another way of solving the SVD matrix factorization on a sparse matrix is by finding all the vectors u_u and v_i, such that u_u make the rows of U and v_i make the columns of V^T. The constraints that need to be followed are:

- $a_{ui} = u_u v_i$ for all u and i
- All the vectors u_u are mutually orthogonal as well as the vectors v_i.

Finding such vectors u_u and v_i can be achieved by solving the following optimization problem, while respecting the constraints over orthogonality.

$$\min_{\substack{u_u, v_i \\ \begin{cases} p_u \perp p_v \\ v_i \perp v_j \end{cases}}} \sum_{r_{ui} \in R} (a_{ui} - u_u v_i)^2 \quad (4.2.2)$$

In more detail, we want to find the values for vectors u_u and v_i that make the sum minimal. In other words, we are trying to match the optimal way of calculating the values of r_{ui} with their calculated value $u_u v_i$. Once these values are known, the construction of U and V is possible, and consequently we can get the SVD of the original interaction matrix A.

The aforementioned operation can be performed even on sparse matrices. This time the missing values are not treated as 0s, or some other arbitrary values, but they are simply ignored. However, we must also forget about the orthogonality constraints, because even if they are useful for interpretation purposes, constraining the vectors usually does not help us obtain more accurate predictions. So the optimization function changes to:

$$\min_{u_u, v_i} \sum_{r_{ui} \in R} (a_{ui} - u_u v_i)^2 \quad (4.2.3)$$

Again, the problem of a non convex function is found here, as in the alternating least squares loss function. This is because it is very difficult to find the values for the vectors u_u and v_i that make the sum minimal. Also, the optimal solution could be not unique. There are however techniques that can be used to approximate the solution.

The main idea, comes from the solution that Simon Funk proposed during the Netflix Prize competition. It is based on Stochastic Gradient Descent to solve the SVD system.

However Funk, proposed a slightly different loss function.

The loss function that was used along with the ALS recommender system tried to minimize the difference between the actual value in the interaction matrix and the calculated value from the dot product of the user and item matrices. Also, a regularization term was introduced to help the model regularize and not overfit the training data. This was a penalty term on the minimization equation that was introduced with a regularization factor lambda that was multiplied with the square sum of the magnitudes of the user and item vectors.

$$L = \sum_{u,i\in K}(a_{ui} - v_i^T u_u)^2 + \lambda(\|u_u\|^2 + \|v_i\|^2) (4.2.4)$$

To illustrate the usefulness of the regularization factor, imagine having an extreme case where a low rating is given by a user to a movie with no other rating from this user. The algorithm will minimize the error by giving v_i a large value. This will cause all rating from this user to other movies to be very low. This is intuitively wrong. By adding the magnitude of the vectors to the equation, giving vectors large value will minimize the equation and thus such situations will be avoided.

To reduce the error between the predicted and actual value, the algorithm makes use of some characteristics of the dataset. In particular for each user - item pair we can extract three parameters, called bias terms.

- μ, which is the average rating of all items
- b_i, which is the average rating of item i minus μ
- b_u, which is used to represent the average rating given by user u minus μ

Thus, the expected rating is:

$$\widehat{a_{ui}} = v_i^T u_u - \mu + b_i + b_u \ (4.2.5)$$

And the loss function is transformed into

$$L = \sum_{u,i\in K}(a_{ui} - u_u^T v_i - \mu + b_i + b_u)^2 + \lambda(\|u_u\|^2 + \|v_i\|^2 + b_i^2 + b_u^2) \ (4.2.6)$$

We have defined only one lambda value on the function above for simplicity. However, there can be multiple regularization parameters such as λ_u, λ_v and λ_{ub} etc.

The above equation can be minimized using the Stochastic Gradient Descent algorithm (SGD). With SGD, we take derivatives of the loss function, but we take the derivative with respect to each variable in the model. The "stochastic" aspect of the algorithm involves taking the derivative and updating feature weights one individual sample at a time. So, for each sample, we take the derivative of each variable, set them all equal to zero, solve for the feature weights, and update each feature.

We want to update each feature (user and item latent factors and bias terms) with each sample. The update for the user bias is given by:

$$b_u \leftarrow b_u - \eta \frac{\partial L}{\partial b_u} \ (4.2.7)$$

The derivative term is given by:

$$\frac{\partial L}{\partial b_u} = 2\big(a_{ui} - (\mu + b_u + b_i + u_u^T v_i)\big)(-1) + 2\lambda b_u$$

$$\frac{\partial L}{\partial b_u} = 2e_{ui}(-1) + 2\lambda b_u$$

$\frac{\partial L}{\partial b_u} = -e_{ui} + \lambda b_u$ (4.2.8)

Where η is the learning rate, that specifies how much the update modifies the feature weights. e_{ui} is a term to represent the error in our prediction, and 2 is dropped, as we can assume that it gets rolled up by the learning rate.

Thus, the rest of the features get updated as follows:

$$b_u \leftarrow b_u + \eta(e_{ui} - \lambda b_u)$$

$$b_i \leftarrow b_i + \eta(e_{ui} - \lambda b_i)$$

$$u_u \leftarrow u_u + \eta(e_{ui} v_i - \lambda u_u)$$

$v_i \leftarrow v_i + \eta(e_{ui} u_u - \lambda v_i)$ [34]

Apparently the number of parameters has greatly increased. The parameters that need tuning now are:

- Number of epochs, which is the number of full iterations over the algorithm
- Number of factors, which is the magnitude of the latent features
- The distribution, from which the random values for matrices U and V will be generated
- Lambda, which is the regularization parameter that will be used over all parameters
    - It can also be tuned per parameter
- Learning rate over all parameters

Surprise is a Python package, that has based its calculations on the very famous *scikit* package. It has a set of built in algorithms designed for recommender systems. Also, it contains a module that can help users perform fast cross validation to help with hyperparameter tuning.

Cross validation is a statistical method that is used to estimate the fit of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in estimates that generally have a lower bias than other methods. There are two main types of cross validation techniques: the exhaustive and non – exhaustive cross validation.

Exhaustive cross validation methods are cross-validation methods which learn and test on all possible ways to divide the original sample into a training and a validation set.

There are two main algorithms in use:

- LPOCV: Leave p out cross validation
  In this technique, p observations are used as the validation set and the remaining observations are the training set. This is repeated on all ways to cut the original sample on a validation set of *p* observations and a training set.
- LOOCV: Leave one out cross validation
  This method is a particular case of LPOCV, where p = 1

Non – exhaustive cross validation are methods that do not compute all possible ways of splitting the original sample dataset.

Methods included in this category are:

- K fold cross validation
  In this method, the sample is randomly partitioned into K equal sized subsamples. Of these, one subsample is retained to test the model, and the rest K-1 are used as the training set of the model. The validation process is repeated K times, so that each of the K subsamples are used one time as the validation set. The resulting scores of the K models, are then usually averaged in order to produce a single score.
  With this method all the observations are used for both training and validation set, and each observation is used in the validation data exactly one time. Most commonly used K fold cross validation method is 10-fold validation, although K remains an unfixed parameter.
- Holdout method
  In this cross validation method, points are randomly assigned to one of the two sets, training and validation set. The size of each of the sets is arbitrary, although typically the test set is smaller than the training set. The model is then trained on the training set and evaluated on the test set.

To create the SVD recommender system, the data were first split into training and test set. This was done holding the typical amount of 80% of the available data as training set and the remaining 20% as the test set.

To test the algorithm, and end up with an optimal solution regarding the dataset in hand, 5-fold cross validation was performed on the training set to test algorithms with different selection of parameters.

The original dataset contains 23.396.256 observations. 20% of those observations was randomly used as a test set (4.679.251 observations) and the rest of them as the training set (18.717.005 observation).

The training set is then randomly split into 5 folds of about 3.7 million observations each, and performs training five times, each time using a different fold as its test set.

The resulting scores are printed below.

The errors are calculated over the known rating. There are three error metrics printed in the matrices:

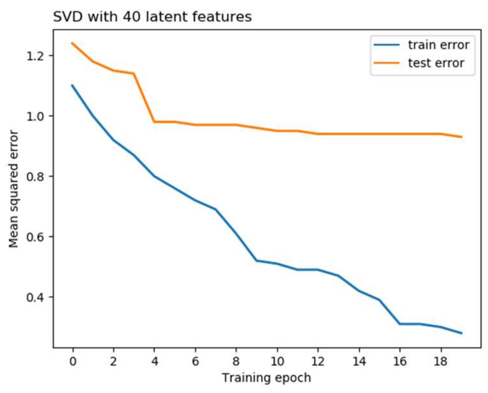
- MAE: mean absolute error $MAE = \frac{\sum(r_{ui} - \widehat{r_{ui}})}{N}$
- MSE: mean squared error $MSE = \frac{\sum(r_{ui} - \widehat{r_{ui}})^2}{N}$
- RMSE: root mean square error $RMSE = \sqrt{\frac{\sum(r_{ui} - \widehat{r_{ui}})^2}{N}}$

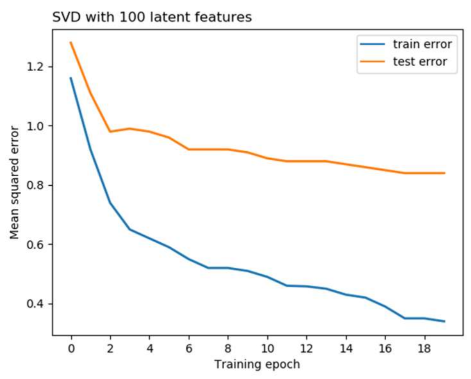
Cross validation run

- Number of training epochs: 20
- Number of latent factors: 40
- Regularization lambda 0.02
- Learning rate 0.005

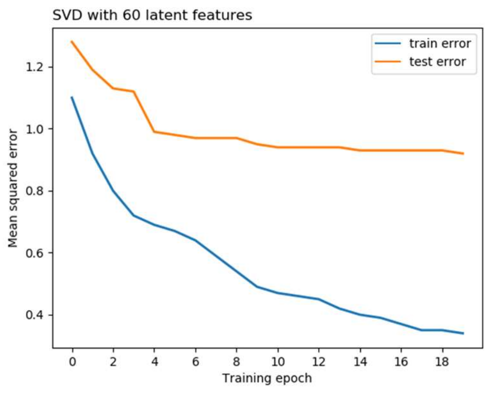
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (test set)	0.9720	0.9775	0.9774	0.9830	0.9717	0.9763	0.0039
MSE (test set)	0.9448	0.9556	0.9555	0.9663	0.9443	0.9532	0.0070
MAE (test set)	0.9718	0.9799	0.9796	0.9803	0.9792	0.9781	0.0004
Fit time	928.10	783.44	770.05	860.97	843.37	837.19	57.03
Test time	2537.47	3109.44	962.68	2007.17	1234.29	1970.21	797.21

Also, error diagrams were produced, as we run the algorithm on the train set and calculated the errors (MSE) on both training and test data with the same hyperparameters defined above, except the number of latent factors, that was set to 40, 60 and 100. The algorithm run for 1, 2, …, 20 epochs consequently.





It is apparent that test set error and training set error are not that far apart as they were on the ALS model. Also, it should be noted that the errors here are of a much lower magnitude. The test set error is about 0.95 after about 6 iterations regardless of the number of latent features selected. However it must be noted that, although the training error did not reduce any further, the test error has reduced with the increase of the number of latent features. We can thus assume that increasing further that amount might lead to a better model. That was not performed, as it is beyond the purpose of this thesis and the training times are getting longer the more we increase the number of features.



To get a better hint about the best model a grid search was performed.

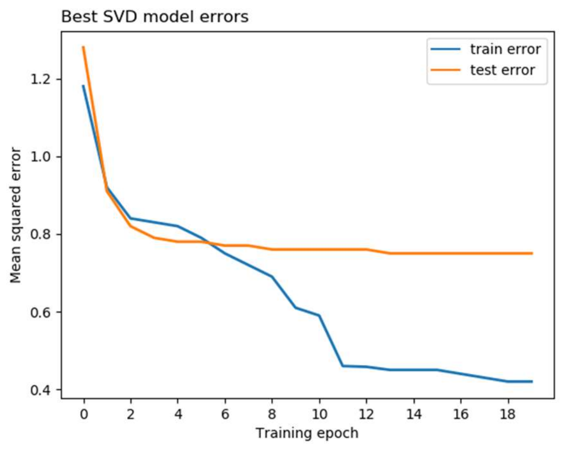
The parameters that were tested were:

- Training epochs: 6, 10, 15 and 20
- Number of latent factors: 60, 80, 100
- Regularization parameter: 0.01, 0.05, 0.1
- Learning rate: 0.001, 0.005, 0.01

All of the above combinations were tested against the whole dataset using a 3-fold cross validation set. The best model is selected based on the produced test set mean square error.

It turned out that the best model is trained for 20 epochs with 100 latent factors, regularization lambda 0.05 and learning rate 0.001 with MSE of 0.4218 on the training set and 0.7512 on the validation set.

The last iteration was performed over that model and the MSE is printed on the following error plot. This is the only model that was trained where the test error fell below the training set MSE for initial training epochs (up to 6). Hence we can be sure that the model is not overfitting the data.



Also since the selection of the best number of training epochs is 20 and the optimal number of latent features is 100, which both are the maximum values that where given for testing in the grid search, further exploration would be advised to be performed with higher values for those hyperparameters.

5. Deep learning techniques

5.0 Preliminaries

Moving further, this analysis would be incomplete if no deep learning algorithms were implemented.

Artificial neural networks of ANNs, are a set of graph based models inspired by the biological neural networks of human brains. The objective is to work like the human brain performing cognitive functions that the brain can perform like problem solving and being teachable. They interpret data through a series of actions, labeling or clustering raw input.

Neural nets only recognize numerical input contained in vectors, into which all real world data, let them be images, text or time series be labeled.

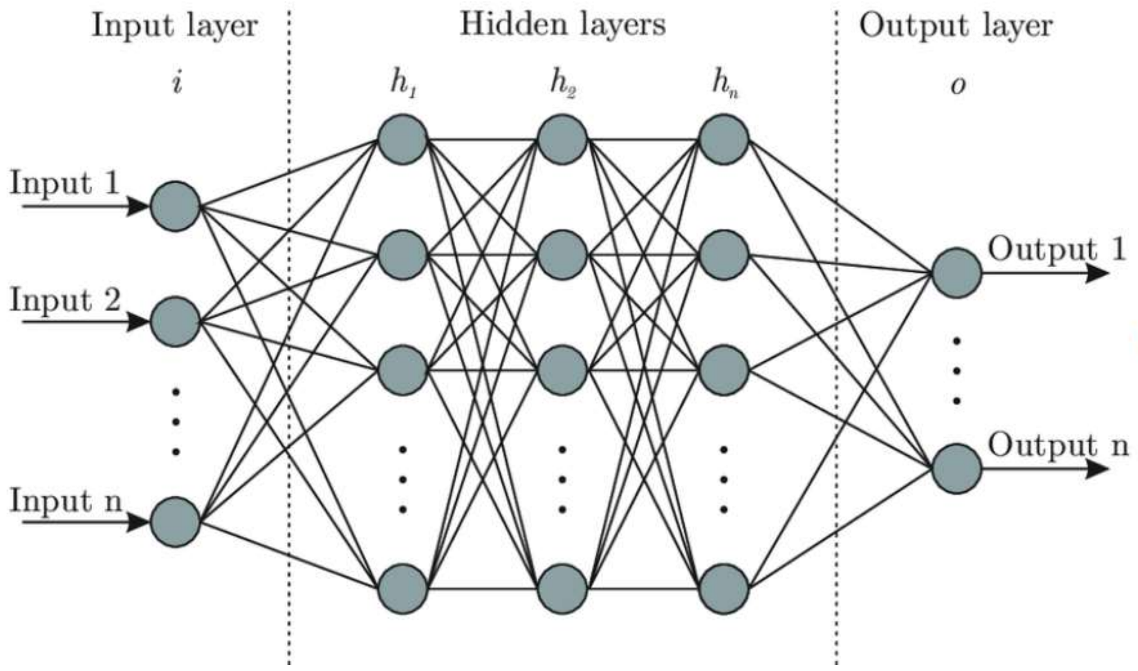


Figure 8 Fully connected ANN architecture

An artificial neural network has a large number of 'processors'. They operate in parallel but they are arranged in tiers. The first tier receives the raw input similar to how the optic nerve receives the raw information in human beings. Each successive tier then receives input from the tier before it and then passes on its output to the tier after it. The last tier processes the final output.

Each tier is made up of nodes. These nodes are also called neurons as they resemble the neurons of biological neural networks. These neurons are usually highly interconnected with the nodes in the previous layer as well as the next one. Every node in the network has its own sphere of knowledge, including rules that it was programmed with and rules that is has already learned by itself.

The learning process of each node is quite simple. The neurons are interconnected with weighted connections. Each of these weighted connections is attached with a real value. The value of a neuron that is placed in the previous layer is passed to a new neuron and it gets multiplied by the weight of the connection. Each node weights the importance of the input it received from the nodes in the previous layer. The input that contribute the most towards the right output are given the highest weight. The sum of all the connected neurons is the neuron's value. This value is then put through an activation function f(x) which mathematically transforms the value and assigns it to the connected neuron in the adjacent layer. This is propagated through the whole network.

So, in conclusion a neural network is a fully connected weighted graph. These weights must be learned in order for the network to make good inference from the data. "Looking" at two similar objects, the network could get confused and output the false answer. To prevent such an occurrence of event, we equip the network with a source of feedback mechanism that is known as back propagation. Using this algorithm, the network is able to go back and "double-check" the weights of its interconnections, to make sure that all the biases are correct and that all the connections are weighted properly.

There are various types of different architectures for neural network models. Each of these types is used to solve a different type of problem. In this project we are going to focus only on the multilayer perceptron.

The multilayer perceptron is a fully connected neural network that consists of a minimum of three layers: The input layer, at least one hidden layer and the output layer. Every single node in a layer is fully connected to the neurons of the previous and the following layers [35].

Each neuron in equipped with an activation function. The neurons of each layer first calculate the weighted sum of the output of all the neurons in the previous layer. This is the neuron's input. Then it adds a bias term. The bias term is defined on the layer, and is a value that is added to the input of all layers. Then the neuron decides if it should "fire" or not.

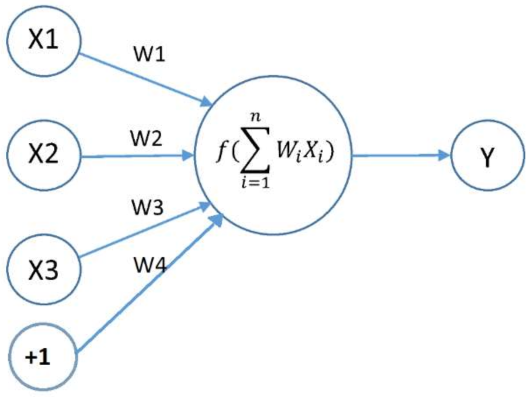


Figure 9 Neuron input and output in a NN

Let us consider a neuron. Then,

$$Y = \Sigma(weight * input) + bias$$

The value of Y is unconstrained. It can be anything from – infinity to + infinity. The neuron is agnostic of the bounds of the value. So, a process must be defined to allow the neuron to "fire" (pass its signal to the following layer) or not.

The purpose of the activation function, or transfer function, is exactly that: Check the produced Y value in the neuron and decide whether the outside connections should consider that this neuron is activated or not. Typically these functions are non linear, allowing the network to learn complex patterns in the data.

Various such functions exist. Some of the most widely used are [36]:

- Step function $f(x) = \begin{cases} 1, & if\ x > thresshold \\ 0, & otherwise \end{cases}$
- Sigmoid function $f(x) = \frac{1}{1+e^{-x}}$
- Tanh function $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
- ReLu $f(x) = \max(0, x)$

It must be noted that the input layer has no activation function and passes either raw of preprocessed information to the adjacent layers. The transfer function that is then found on each layer can be different per layer. Usually, the activation function that is used on all the layers of the "hidden part" of the network are the same, while the activation function of the output layer differs.

The learnable part of the neural network are the link weights. The weights need to get the optimal values, so that the network can generate and output the correct information. This is done via defining an error metric, which in this case is the loss function (or objective function) of the network. While propagating information back and forth the network need to make changes in the weights so that this function can converge to a minimum. In more detail, the loss function maps the values of variables into one number, intuitively representing some "cost" associated with these values.

The cost or loss function has an important job in that it must faithfully distill all aspects of the model down into a single number in such a way that improvements in that number are a sign of a better model. As the function must be used in backpropagation, it must satisfy two properties:

1. The cost function must be able to be written as an average $C = \frac{1}{N}\sum_x C_x$, over cost functions C_x of individual training examples x.
This is so it allows us to compute the gradient (with respect to weights and biases) for a single training example, and run Gradient Descent.
2. The cost function C must not be dependent on any activation values of a neural network besides the output values.
Technically a cost function can be dependent on any output values of the current layer. This restriction is crucial, so we can backpropagate, because the equation for

finding the gradient of the last layer is the only one that is dependent on the cost function (the rest are dependent on the next layer). If the cost function is dependent on other activation layers besides the output one, backpropagation will be invalid because the idea of "trickling backwards" no longer works.

Typically, the loss functions of choice for regression neural networks is the mean squared error, which satisfies both properties

- MAE $cost(s,y) = \frac{1}{N}\sum|s-y|$
- MSE $cost(s,y) = \frac{1}{N}\sum(s-y)^2$
- Huber $cost(s,y) = \begin{cases} \frac{1}{2}(y-s)^2, & if\ |y-s| < \delta \\ \delta\left((y-s) - \frac{1}{2}\delta\right), & otherwise \end{cases}$

Huber loss function is used in regression neural networks, however it is less sensitive to outliers than the MSE, as it treats errors as square only inside an interval [37]. Mean squared error has the tendency of getting dominated by outliers. However, in our case of building a recommender system, by creating a network that accurately predicts the values of the rating for a specific user – item pair, we need it to take care of low ratings. Therefore, the cost function that was chosen is MSE.

The values of the weights along with the bias terms are “learned” using the efficient algorithm of backpropagation (BP) [38]. Simply put, first a full forward pass is completed through the network. After that, backpropagation performs a backward pass while adjusting the model’s parameters. In more detail, via backpropagation, the neural network repeatedly adjusts the weights of the connections so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.

The derivative of a function C measures the sensitivity to change of the function value (output value) with respect to a change in its argument x (input value). In other words, the derivative tells us the direction C is going. The gradient shows how much the parameter x needs to change (in a positive or negative direction) to minimize C. So, the gradient helps us optimize the model parameters (weights and biases)

$$w \leftarrow w - \varepsilon \frac{\partial C}{\partial w} \quad (5.0.1)$$

$$b \leftarrow b - \varepsilon \frac{\partial C}{\partial b} \quad (5.0.2)$$

Where

- w and b are matrix representations of the weights and biases

- ε is the learning rate which determines the gradient's influence

We can compute the gradients using the chain rule technique. For a single weight (w_{jk}^{l}) and a single bias term (b_j^l) , where:

w_{jk}^{l} is the weight of the connection between neuron j the (l-1)-th layer and k in the l-th layer.

b_j^l is the bias term passed to neuron j in the l-th layer of the network.

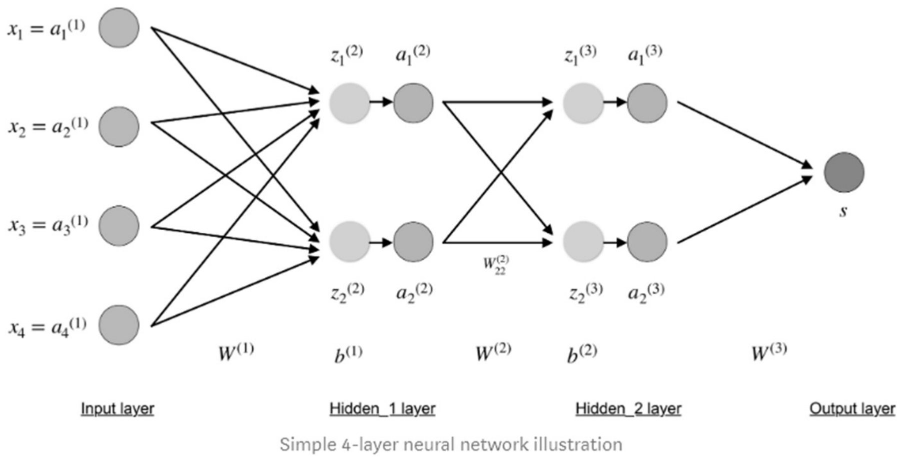


Figure 10 Neural Network architecture

The gradients can be calculated as follows, via the chain rule:

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^{l}} \quad (5.0.3)$$

Where

$z_j^l = \sum_{k=1}^{m} w_{jk}^{l} a_k^{l-1} + b_j^l$ (5.0.4)

a_k^{l-1} is the output of the previous layers (passes through nodes' activation function) that is input in k-th neuron of layer l

and m is the number of neurons in the l-th layer

$$\frac{\partial z_j^l}{\partial w_{jk}^{l}} = a_k^{l-1} \quad (5.0.5)$$

Then,

$$\frac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1} \cdot \frac{\partial C}{\partial z_j^l} \quad (5.0.6)$$

Similarly, the equations applied to b_j^l are calculated:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l}$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \quad (5.0.7)$$

And it is now obvious that the error gets propagated to the weight and bias values of the previous layers. Thus, optimization of those parameters can be achieved via backpropagation.

Consequently, the steps involved to complete an epoch of training are the following:

- Initiate the values of w and b with random values (typically chosen from a normal distribution)
- Forward pass the network and predict the output y for all input values. ($\hat{y} = s$)
- Evaluation of s and y through the calculation of the cost function C=cost(s,y). Based on the values of C, the model "knows" (via backpropagation of error) how much to adjust the parameters in order to get closer to the desired output y.
- Backpropagate the error and update the weight and bias matrices.

After each completion of training (:= training epoch), the value of the loss function is minimized and the weights and biases of the network are getting close to their optimal values.

5.1 Neural Network of embeddings recommender system

To create a recommendation engine that is using the functionality of neural network, we have constructed a neural network of embeddings.

An embedding is a mapping of a discrete — categorical — variable to a vector of continuous numbers. Practically, they "translate" high dimensional input into a lower dimension continuous vector representations of discrete variables. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space. Ideally, since the embeddings can be learned, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space.

The simplest form of an embedding technique is one-hot representation of the data.

One-hot encoding, maps categorical variables into a different vector. Every observation is mapped into a vector of 0s and a single 1, signaling the specific category. As a consequence, one-hot encoding has two main drawbacks.

1. Variables with high cardinality increase the dimensionality of the input space by a very large amount, usually making the dimensionality of the input space infeasible. With each additional category in a categorical variable, we have another number of one-

hot encoded features, so for example, a variable containing 5000 movie ids needs a 5000 – dimensional space to be represented with one-hot features.

2. The mapping is "uniform", meaning that similar categories are not placed close in the vector space.
   Measuring similarity between the vectors after one-hot encoding in the embedding space using cosine similarity, would end up 0 for every comparison between entities.

Considering these two problems, the ideal solution for representing categorical variables would require fewer numbers than the number of unique categories and would place similar categories closer to one another.

One-hot embeddings is an oversimplified transformation algorithm of representing categorical data into a numeric representation. The task is also performed without supervision, thus the resulting features are not able to capture similarities.

An improved embedding of the data can be achieved through a definition of a supervised task fed into a neural network. The embeddings are then learned: they are the weights of the network's connections between the layers. These weights are learned by the network - to their optimal values – minimizing the output error. The resulting vectors (the networks weights matrix) are a representation of categories, where similar categories – relative to the task – are close to one another.

So, to generate correct user to movie ratings, thus creating a list of high rated movies list for recommendations to a specific user, we can create a supervised task for a neural network of embeddings.

Since, the representation of data on hand, only contains minimal information of item id, movie id and the corresponding rating in [0,5] range, a neural network that embeds the movie id and the user id can be created.

For this project Python's deep learning framework *Keras* was used.

Keras is a high-level neural network API, written in Python and capable of running on top of TensorFlow, CNTK and Theano. It enables fast experimentation with a variety of different neural network architectures.

The following structure was modeled:

- Input layer: The input contains both user ids and movie ids, integer encoded.
- Hidden layer: One hidden layer of embeddings for user ids and movie ids.
- Output layer: Merge layer of dot product.

First, the neural network model is defined. The model is sequential, meaning that the output layer will consist of one neuron outputting a continuous value.

The Embeddings layer in a Keras neural network turns positive integers (indexes) into dense vectors of fixed size. This layer can only be used as the first layer of a model. The user must define the size of the vocabulary (i.e. the number of unique user and item indexes) and the dimension of the embedding. The embedding layer is then initialized with random weights and will learn an embedding for every word in the training set vocabulary (for every id). Also, in the implementation of Keras Embedding layer, embedding layers have no activation function.

The network is defined with two distinct input layers. One layers gets movie ids as input and the second gets the values of user ids. Two distinct embedding layers are trained for each of the movie and user ids alongside each other. These can be thought of as two distinct networks. It is possible to define the embedding layers to output movie embeddings on a different dimension space than user ids.

The two networks output the 2D matrix of embeddings which get flattened. Later, the networks are merged together using a merge layer that computes the dot product between the two input tensors. This is also the output layer of the network. The dot product is computed between all possible embeddings combinations, thus the training time of the neural network can also grow with the increase of the vocabulary.

The loss function of the network is the mean squared error, as it cares for outliers, which in our case are the low ratings of movies. The dot product of every possible combination of item and user embeddings is calculated, however the loss function is only computed on the training examples that have an accompanying rating.

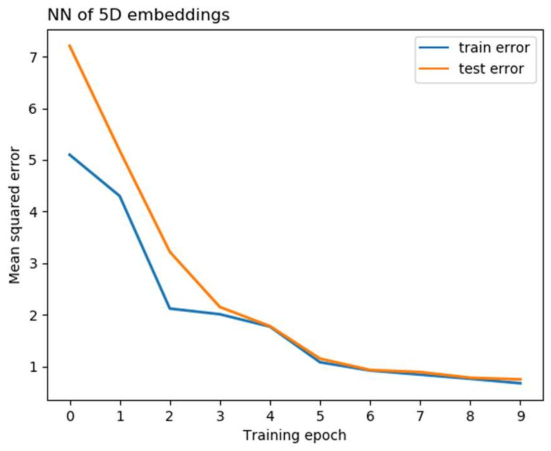
The errors are then back propagated through both embedding layers and the weights get adjusted with respect to one another. Instead of using classic gradient descent, Adam optimization algorithm was chosen, which is an extension of stochastic gradient descent [39].

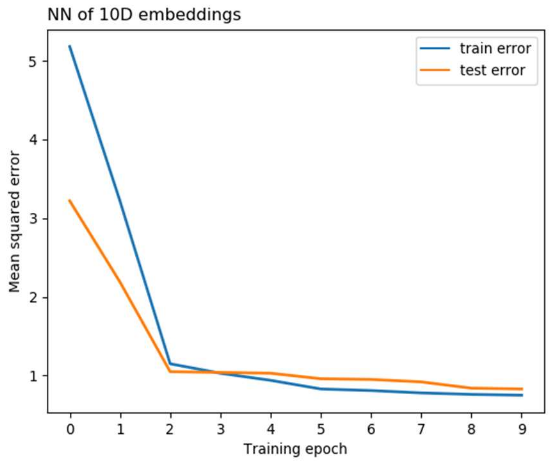
Distinct models were created that were trained from 1 up to 10 epochs. The learning rate of all tests was the default value of Adam optimizer, set to 0.001.

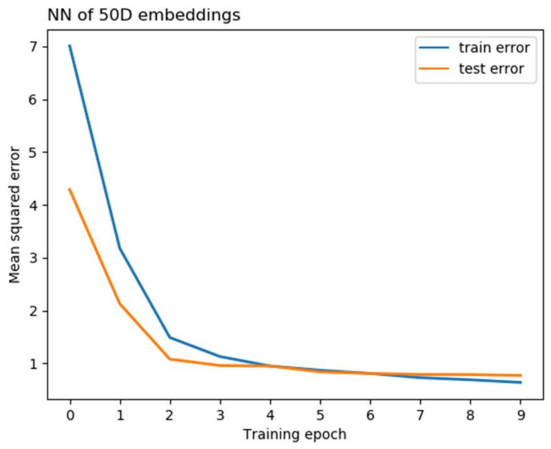
The training time of an algorithm trained for 10 epochs, takes approximately 4 days.

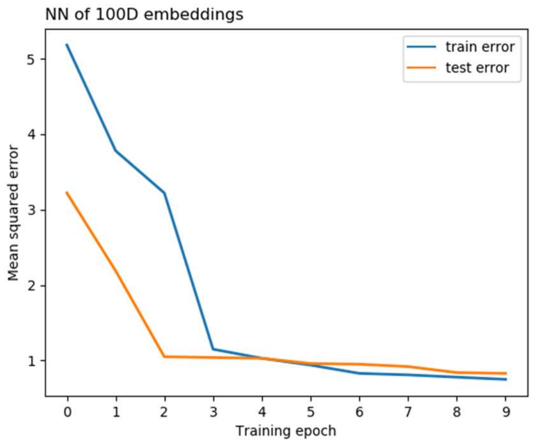
The following plots contain the evolution of MSE for both training and test set, as the number of training epochs increased.

No hyperparameter tuning was performed due to the long training times of the algorithm.







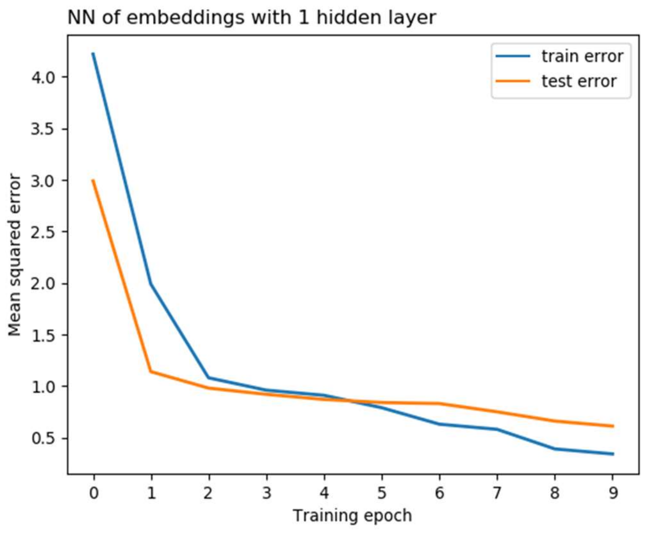


The resulting errors are very promising, as both train set and test set errors are almost the same and the model is generalizing well. Of the above models the best model is the one with 10D embeddings, as the resulting error after 10 epochs of training was 0.6178 for the training set and 0.7241 for the test set.

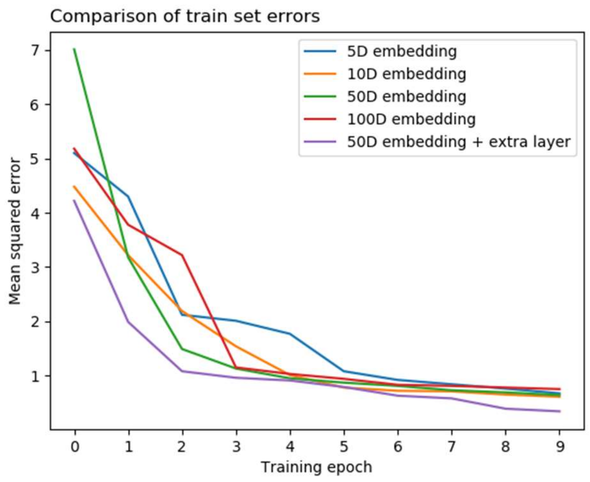
Since the resulting scores were very promising, another model was trained. That model had an extra hidden dense layer that was added after the merging layer of the embeddings layers. The embeddings layers produced embeddings of 50 – dimensional representations for both user and item ids. The dense layer consisted of 128 neurons with sigmoid activation function and it was placed after the merging layer of the embeddings. The output layer remained as a single output neuron. Backpropagation of errors was completed with Adam.

The training time of the model was the longest, in comparison to the rest of the NN model. It took a total of 9 days, so not much optimization was possible.

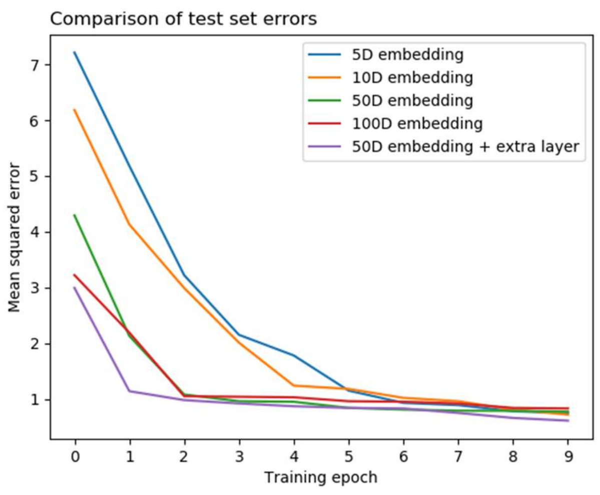
The model was able to achieve the lowest error for both training and test set after the completion of 10 training epochs. Train set mean squared error is 0.3413, while test set error 0.6112.



Error comparison diagrams were generated for all neural networks of embeddings models.



Model	**Train set error after 10 epochs**
5D embeddings	0.6725
10D embeddings	0.6178
50D embeddings	0.6415
100D embeddings	0.7508
50D embeddings + layer	0.3413



Model	**Test set error after 10 epochs**
5D embeddings	0.7474
10D embeddings	0.7241
50D embeddings	0.7723
100D embeddings	0.8319
50D embeddings + layer	0.6112

6. Conclusion

In this thesis an introduction to different methods for recommender systems was presented. The focus was to create good recommendations using only the data regarding the interaction between a user and an item. In the first part, a variety of methods was used to find similar item to recommend to users. On the second part algorithms were created to predict the rating a user would give an item.

The analysis that was presented, made clear that even though the dataset on hand had a very limited amount of information, consisting only with user and movie ids along with the given rating, a recommender system can be build achieving astonishing results in terms of error.

The strength of using latent features representations was underlined, as all algorithms that used them generated very promising results, being able to achieve very low errors in their test sets. Neural network architectures of embeddings showed the best results, having the lowest rating prediction error.

Bibliography

1. Sharma R, Singh R. Evolution of recommender systems from ancient times to modern era: A survey. *Indian J Sci Technol*. 2016;9(20). doi:10.17485/ijst/2016/v9i20/88005

2. Huang LW, Chen GS, Liu YC, Li DY. Enhancing recommender systems by incorporating social information. *J Zhejiang Univ Sci C*. 2013;14(9):711-721. doi:10.1631/jzus.CIIP1303

3. Felfernig A, Polat-Erdeniz S, Uran C, et al. An overview of recommender systems in the internet of things. *J Intell Inf Syst*. 2019;52(2):285-309. doi:10.1007/s10844-018-0530-7

4. Schröder G, Thiele M, Lehner W. Setting goals and choosing metrics for recommender system evaluations. *CEUR Workshop Proc*. 2011;811(January 2011):78-85.

5. Zibriczky D. Recommender systems meet finance: A literature review. *CEUR Workshop Proc*. 2016;1606(5):3-10.

6. P V A. a Survey of Recommender System Types and Its Classification. *Int J Adv Res Comput Sci*. 2017;8(9):486-491. doi:10.26483/ijarcs.v8i9.5017

7. Sielis GA, Tzanavari A, Papadopoulos GA. Recommender Systems Review of Types, Techniques, and Applications. *Encycl Inf Sci Technol Third Ed*. 2014:7260-7270. doi:10.4018/978-1-4666-5888-2.ch714

8. Bittner P. Modulempfehlungen über Vo rgänger- und Nachfolgemodule. *Lect Notes Informatics (LNI), Proc - Ser Gesellschaft fur Inform*. 2015;246(Section 3):725-733. doi:10.1155/2009/421425

9. Do M-PT, Nguyen D V., Nguyen L. Model-based Approach for Collaborative Filtering. *Proc 6th Int Conf Inf Technol Educ*. 2010;(January):217-225. https://goo.gl/BHu7ge.

10. Leben M. Applying Item-based and User-based collaborative filtering on the Netflix data. *Inf Retr Boston*. 2008:1-9.

11. Hayakawa M. MF Techniques. *Earthq Predict with Radio Tech*. 2015:199-207. doi:10.1002/9781118770368.ch6

12. Pu L, Faltings B. Understanding and improving relational matrix factorization in recommender systems. *RecSys 2013 - Proc 7th ACM Conf Recomm Syst*. 2013:41-48. doi:10.1145/2507157.2507178

13. Lu Z, Dou Z, Lian J, Xie X, Yang Q. Content-Based Collaborative Filtering for News Topic Recommendation. :217-223.

14. Analysis ID. Politecnico di Torino Porto Institutional Repository [Article] Hybrid Recommender Systems : A Systematic Literature Review. 2017;(November). doi:10.3233/IDA-163209

15. Modeling U, Burke R. Hybrid Recommender Systems : Survey and Experiments Hybrid Recommender Systems : 2014;(November 2002). doi:10.1023/A

16. Xu J, Yao Y, Tong H, Tao X, Lu J. Ice-Breaking: Mitigating cold-start recommendation problem by rating comparison. *IJCAI Int Jt Conf Artif Intell*. 2015;2015-January(Ijcai):3981-3987.

17. Elahi M. Chapter 1 Cold Start Solutions For Recommendation Systems. 2019;(May). doi:10.13140/RG.2.2.27407.02725

18. Nadimi-Shahraki MH, Bahadorpour M. Cold-start problem in collaborative recommender systems: Efficient methods based on ask-to-rate technique. *J Comput Inf Technol*. 2014;22(2):105-113. doi:10.2498/cit.1002223

19. Boratto L, Carta S, Fenu G, Saia R. Semantics-aware content-based recommender systems: Design and architecture guidelines. *Neurocomputing*. 2017;254(October 2017):79-85. doi:10.1016/j.neucom.2016.10.079

20. Gunawardana A, Shani G. A survey of accuracy evaluation metrics of recommendation tasks. *J Mach Learn Res*. 2009;10:2935-2962.

21. Beel J, Genzmehr M, Langer S, Nürnberger A, Gipp B. A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. *ACM Int Conf Proceeding Ser*. 2013:7-14. doi:10.1145/2532508.2532511

22. Hu Y, Volinsky C, Koren Y. Collaborative filtering for implicit feedback datasets. *Proc - IEEE Int Conf Data Mining, ICDM*. 2008:263-272. doi:10.1109/ICDM.2008.22

23. Ayub M, Ghazanfar MA, Maqsood M, Saleem A. A Jaccard base similarity measure to improve performance of CF based recommender systems. *Int Conf Inf Netw*. 2018;2018-January(January):1-6. doi:10.1109/ICOIN.2018.8343073

24. Gurjar K, Moon Y. A Comparative Analysis of Music Similarity Measures in. 2018;14(1):32-55.

25. Garcia EK, Feldman S, Gupta MR, Srivastava S. Completely lazy learning. *IEEE Trans Knowl Data Eng*. 2010;22(9):1274-1285. doi:10.1109/TKDE.2009.159

26. Wang H. Personalized recommendation system K- neighbor algorithm optimization. 2015;(Icitel):105-108.

27. Prasath VBS, Arafat H, Alfeilat A, Hassanat ABA, Lasassmeh O, Ahmad S. Effects of Distance Measure Choice on KNN Classifier Performance - A Review. :1-39.

28. Das U. Design of a Recommendation Model Considering Semantic Analysis. 2013;77(1):45-49.

29. Hofmann T. Latent Semantic Models for Collaborative Filtering. 2004;22(1):89-115.

30. Wang B, Ye F, Xu J. A Personalized Recommendation Algorithm Based on the User ' s Implicit Feedback in E-Commerce. 2018. doi:10.3390/fi10120117

31. Pilászy I, Zibriczky D. Fast ALS-based Matrix Factorization for Explicit and Implicit Feedback Datasets Categories and Subject Descriptors. 2010;(January). doi:10.1145/1864708.1864726

32. Konstan JA. MovieExplorer : Building an Interactive Exploration Tool from Ratings and Latent Taste Spaces. :1383-1392. doi:10.1145/3167132.3167281

33. Kao C. Introduction to gradient descent •. :1-12.

34. Katz G, Shani G. Using Wikipedia to Boost SVD Recommender Systems.

35. Dreiseitl S. Artificial Neural Networks textbook.

36. Nwankpa CE, Ijomah W, Gachagan A, Marshall S. Activation Functions : Comparison of Trends in Practice and Research for Deep Learning. :1-20.

37. Burges C, Drucker H, Golowich S, et al. Regression Estimation with Support Vector Learning Machines in collaboration with. 2003;(June 2013).

38. Li F, Johnson J, Yeung S. Lecture 4 : Backpropagation and Neural Networks. 2017.

39. Kingma DP, Ba JL. Adam: A method for stochastic optimization. 2015:1-15.