Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας, Πληροφορικής & Υπολογιστών

Δημιουργία Συστήματος Συστάσεων με Νευρωνικά Δίκτυα Γράφων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΩΑΝΝΗΣ ΑΘΑΝΑΣΙΟΥ

Επιβλέπων : Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας, Πληροφορικής & Υπολογιστών

Δημιουργία Συστήματος Συστάσεων με Νευρωνικά Δίκτυα Γράφων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΩΑΝΝΗΣ ΑΘΑΝΑΣΙΟΥ

Επιβλέπων : Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 13η Μαρτίου 2023.

Γεώργιος Στάμου Καθηγητής Ε.Μ.Π.	Αθανάσιος Βουλόδημος Επίκουρος Καθηγητής Ε.Μ.Π.	Στέφανος Κόλλιας Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2023

Ιωάννης Αθανασίου

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

Περίληψη

Στο σύγχρονο διαδίκτυο, η αφθονία δεδομένων μπορεί να αποβεί αφόρητη για τους χρήστες. Για την βελτίωση της εμπειρίας χρήστη, οι εξατομικευμένες προτάσεις γίνονται όλο και πιο σημαντικές. Εμπνευσμένη από την αυξανόμενη τάση των προσεγγίσεων που βασίζονται σε γράφους στα συστήματα συστάσεων, αυτή η διπλωματική εργασία διερευνά τις δυνατότητες των νευρωνικών δικτύων γράφων (GNN) για την ανάπτυξη ενος συστήματος προτάσεων ταινιών. Η προσέγγισή μας περιλαμβάνει την αντιμετώπιση του προβλήματος ως την διαδικασία πρόβλεψης βάρους των ακμών σε ένα διμερές γράφημα, αποτελούμενο από χρήστες και ταινίες ως κόμβους, και τις αντίστοιχες βαθμολογήσεις ως ακμές. Πραγματοποιούμε την έρευνά μας σε ένα σχετικά νέο σύνολο δεδομένων, ονομαζόμενο The Movies Dataset, το οποίο δεν έχει χρησιμοποιηθεί ευρέως σε δημοσιεύσεις για συστήματα συστάσεων. Το σύνολο δεδομένων αυτό περιέχει βαθμολογήσεις από πραγματικούς χρήστες σε ταινίες, καθώς και πολυάριθμα μεταδεδομένα σχετικά με το περιεχόμενο κάθε ταινίας, τα οποία έχουν εξαχθεί από ποικίλες διαδιτκυακές πηγές. Μοντελοποιούμε το σύνολο δεδομένων ως γράφο και το αποθηκεύουμε σε ένα σύστημα διαχείρισης βάσεων δεδομένων γράφων. Για την αξιοποίηση των πολυάριθμων μεταδεδομένων που σχετίζονται με το περιεχόμενο της κάθε ταινίας, τα κωδικοποιούμε ως εμφυτεύματα κόμβων χρησιμοποιώντας ποικίλες τεχνικές. Το υλοποιημένο μοντέλο συστάσεων λαμβάνει ως είσοδο τον διμερή γράφο και προβλέπει τις ακριβείς τιμές των νέων βαθμολογήσεων. Μετά από την διεξαγωγή πολλαπλών πειραμάτων και τον συντονισμό των πολυάριθμων υπερπαραμέτρων, η λύση μας πετυχαίνει επιδόσεις συγκρινόμενες με λύσεις τελευταίας τεχνολογίας, που βασίζονται σε γράφους, στο αντίστοιχο σύνολο δεδομένων MovieLens. Δείχνουμε ότι χρησιμοποιώντας κατάλληλους αλγορίθμους για την δημιουργία των εμφυτευμάτων κόμβων που αντιπροσωπεύουν το περιεχόμενο της κάθε ταινίας, μπορούμε να βελτιώσουμε την ακρίβεια των προβλέψεων σε όλες τις αρχιτεκτονικές νευρωνικών δικτύων γράφων που διερευνήθηκαν σε αυτήν την διατριβή. Επιπλέον της ανωτέρω έρευνας, αναπτύσσουμε και μία πλήρη διαδικτυακή εφαρμογή, που αποτελείται από πολυάριθμες υπηρεσίες στην πλευρά του διακοσμητή, και από μία διεπαφή χρήστη. Η πλατφόρμα δίνει στους χρήστες την δυνατότητα να εξερευνούν και να βαθμολογούν ταινίες, συμβάλλοντας στην επέκταση του συνόλου δεδομένων με νέους χρήστες και αξιολογήσεις. Επιπλέον, η ενοποίηση του μοντέλου συστάσεων με τα υπόλοιπα συστατικά μέρη της πλατφόρμας, προσφέρει μία εισαγωγή στο πεδίο των MLOps, ενώ ταυτόχρονα επιτρέπει την δοκιμή του συστήματος συστάσεων σε πραγματικές συνθήκες.

Λέξεις κλειδιά

Γράφοι, Νευρωνικά Δίκτυα Γράφων, Βάσεις Δεδομένων Γράφων, Συστήματα Συστάσεων, Ταινίες, Πρόβλεψη βάρους ακμής, Εμφυτεύματα κόμβων

Abstract

In today's web, the abundance of available data can be overwhelming for users. To enhance the user experience, personalized recommendations are becoming increasingly important. Inspired by the growing trend of graph-based approaches in recommendation systems, this thesis investigates the potential of Graph Neural Networks (GNNs) for developing a movie recommendation system. Our approach involves treating the problem as the link weight prediction task on a bipartite graph, consisting of users and movies as nodes, and the corresponding ratings as edges. We experimented with a relatively new dataset, named The Movies Dataset, that contains real-world ratings from users to movies as well as numerous metadata regarding each movie's content, which were scrapped from multiple web resources. We model the dataset as a graph and store it in a graph database management system. To utilize the numerous metadata related to each movie's content, we encode them as node embeddings using various techniques. The implemented recommendation model receives as input the bipartite graph and predicts the exact ratings values. After conducting multiple experiments and tuning the numerous hyperparameters, our solution achieved an RMSE value compared to state-of-the-art graph-based solutions on the corresponding MovieLens dataset. We demonstrate that by utilizing appropriate algorithms to generate node embeddings that represent the content of each movie, we can improve the accuracy of the predictions in all of the GNN architectures explored in this thesis. In addition to the above research, we develop a complete web application, consisting of multiple REST APIs and a user interface. The platform enables users to explore and rate movies, contributing to the expansion of the dataset with new users and ratings. Additionally, the integration of the recommendation model with the other components of the platform, offers a glimpse into the field of MLOps, while concurrently enabling the testing of the recommender system in real-world conditions.

Key words

Graphs, Graph Neural Networks, Graph Databases, Recommender Systems, Movies, Link Weight Prediction, Node Embeddings

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Στάμου για την πολύτιμη καθοδήγησή του, και την ευκαιρία να ερευνήσω ένα τόσο ενδιαφέρον θέμα κατά την εκπόνηση της διπλωματικής μου εργασίας. Θα ήθελα να ευχαριστήσω τον διδακτορικό φοιτητή Ορφέα Μενή Μαστρομιχαλάκη, για την πολύτιμη και συνεχή καθοδήγηση και υποστήριξή του, καθώς και για το πλήθος των δημιουργικών και στοχευμένων ιδεών και σκέψεων που μοιράστηκε μαζί μου στα πλαίσια εκπόνησης αυτής της εργασίας.

Θα ήθελα επίσης να ευχαριστήσω τους γονείς μου, Δώρα και Γιώργο, και την αδερφή μου Θάλεια, για την απεριόριστη στήριξή τους κατά την διάρκεια των σπουδών μου. Τέλος, δεν θα μπορούσα να μην ευχαριστήσω τους συμφοιτητές μου, Αντρέα και τον Θάνο, για το πλήθος των ιδεών που μοιράστηκαν μαζί μου, και τις στιγμές που μοιραστήκαμε κατά την διάρκεια των σπουδών μας.

Ιωάννης Αθανασίου,

Αθήνα, 13η Μαρτίου 2023

Contents

List of Tables

List of Figures

Chapter 1

Εκτεταμένη περίληψη στα Ελληνικά

1.1 Θεωρητικό υπόβαθρο

Στην παρακάτω ενότητα θα αναφερθούν περιληπτικά οι βασικές έννοιες γύρω από τους γράφους που είναι αναγκαίες για την ενασχόληση με Βάσεις Δεδομένων Γράφων, όπως η Neo4j (2012), και για την εμβάθυνση στην Μηχανική Μάθηση πάνω σε γράφους, καθώς και οι αρχές που σχετίζονται με τα νευρωνικά δίκτυα γράφων, τα συστήματα προτάσεων, και τον συνδυασμό των ανωτέρω τεχνολογιών.

1.1.1 Βασικές έννοιες Γράφων

Definition 1.1.1 (Γράφος): *Γράφος είναι ένα ζεύγος $G = (V, E)$ από σύνολα, τέτοια ώστε $E \subseteq |V|^2$. Για να αποφύγουμε σημασιολογικές ασάφειες, θεωρούμε ότι $V \cap E = \emptyset$. Τα στοιχεία του συνόλου V ονομάζονται κορυφές ή κόμβοι του γράφου G, ενώ τα στοιχεία του συνόλου E ονομάζονται ακμές του γράφου. Diestel (2010)*

Οι αναλυτικοί ορισμοί των βασικών εννοιών γύρω από τους γράφους παρατίθενται στο κεφάλαιο 3.1.1. Μία από τις πιο βασικές έννοιες είναι ο χαρακτηρισμός των γράφων ως κατευθυνόμενων (ή μη). Σε έναν **κατευθυνόμενο γράφο**, οι δύο κόμβοι κάθε ακμής διαχωρίζονται στον αρχικό και τελικό κόμβο, και λέμε ότι η ακμή έχει κατεύθυνση από τον αρχικό στον τελικό. Μία οπτικοποίηση της διαφοροποίησης αυτής των γράφων φαίνεται διαισθητικά στην εικόνα 3.2. Δύο κορυφές ονομάζονται **γειτονικές κορυφές** όταν υπάρχει μία ακμή στον γράφο που να προσπίτει σε αυτές τις κορυφές. Ο **βαθμός κόμβου** είναι ο αριθμός των ακμών που προσπίπτουν σε αυτόν. Ως **απόσταση κόμβων** ορίζεται το μήκος του συντομότερου μονοπατιού με αφετηρία τον πρώτο κόμβο και τερματισμό τον δεύτερο, όπου **μονοπάτι** ονομάζουμε μία ακολουθιά από διαδοχικές ακμές και κόμβους στον γράφο, όπου όλες οι κορυφές εμφανίζονται το πολύ μία φορά. Δύο γράφοι ονομάζονται **ισομορφικοί**, εάν, διαισθητικά, υπάρχει μία αντιστοιχία μεταξύ των κορυφών τους, που να διατηρεί την συνδεσιμότητα του γράφου. Μία συνήθης μαθηματική αναπαράσταση ενός γράφου είναι ο πίνακας γειτνίασής του. Ο τρόπος υπολογισμού του πίνακα γειτνίασης ενός γράφου φαίνεται στην εικόνα 3.4.

Συμπληρωματικά με τις προαναφερθείσες βασικές έννοιες γύρω από τους γράφους, στην παρούσε μελέτη αξιοποιήθηκαν εκτενώς και πιο σύνθετοι τύποι γραφημάτων. Η πρώτη έννοια από αυτές είναι η έννοια των **σταθμισμένων γραφημάτων**, στα οποία ένας αριθμός (βάρος) έχει αντιστοιχηθεί στην κάθε ακμή. Τα σταθμισμένα γραφήματα έχουν πληθώρα εφαρμογών σε συστήματα που προσομοιώνουν ρεαλιστικά δίκτυα, με τα βάρη των ακμών να αντιπροσωπέυουν την ένταση ή το κόστος της αλληλεπίδρασης μεταξύ των κόμβων. Μία οπτικοποίηση ενός σταθμισμένου γραφήματος

βρίσκεται στην εικόνα 3.5. Συνεχίζοντας, μία περαιτέρω διάκριση των γράφων γίνεται με βάση την κατηγοριοποίηση ή μη των κόμβων τους. Υπό αυτό το πρίσμα, κάποιοι γράφοι χαρακτηρίζονται από ένα σύνολο **ετικετών** που ανατίθενται στους κόμβους τους. Οι γράφοι, με βάση τις ετικέτες αυτές διαχωρίζονται σε **ομογενείς** και **ετερογενείς** (3.6) γράφους. Όπως υποδηλώνει και το όνομα της κάθε κατηγορίας, οι κόμβοι ενός ομογενούς γράφου χαρακτηρίζονται από έναν τύπο ετικέτας, ενώ οι στους ετερογενείς γράφους, οι κόμβοι χαρακτηρίζονται από περισσότερα από ένα είδη ετικετών (3.6). Μία ειδική περίπτωση ετερογενή γράφου, όπου όλες οι ακμές συνδέοουν κόμβους διαφορετικής ετικέτας, είναι οι **διμερείς γράφοι**. Οι διμερείς γράφοι, χρησιμοποιούνται συχνά για την μοντλεποίηση της πληροφορίας που σχετίζεται με ένα σύστημα προτάσεων, όπου η μία κατηγορία κόμβων είναι οι χρήστες του συστήματος, η δεύτερη κατηγορία είναι τα αντικείμενα που αξιολογούνται, και οι ακμές εκφράζουν την αλληλεπίδραση μεταξύ χρηστών και αντικειμένων.

1.1.2 Ο αλγόριθμος Weisfeiler-Lehman

Η έννοια του ισομορφισμού των γράφων, που παρατέθηκε προηγουμένως, αποτελεί μία έννοια στην οποία στηρίζεται σε σημαντικό βαθμό η εφαρμογή μεθόδων μηχανικής μάθησης σε γράφους. Ο **αλγόριθμος Weisfeiler-Lehman** (Leman (2018)) είναι ένας αποδοτικός αλγόριθμος που αποσκοπεί στο να λύσει το πρόβλημα της απόφασης σχετικά με το αν δύο γράφοι είναι ισομορφικοί ή όχι. Πιο συγκεκριμένα, αξίζει να καλύφθει η διαίσθηση πίσω από μία συγκεκριμένη εκδοχή του αλγορίθμου, τον **1-WL αλγόριθμο**. Ο αλγόριθμος 1-WL, αποτελεί έναν επαναληπτικό αλγόριθμο, που στηρίζεται στην τεχνική color refinement, ώστε να αναθέσει ένα χρώμα (ετικέτα) σε κάθε κόμβο του γράφου, και να αναπαραστήσει τον γράφο ως ένα υπερσύνολο από αυτές τις ετικέτες. Αν τα δύο υπερσύνολα που αντιπροσωπεύουν δύο γράφους καταλήξουν να είναι διαφορετικά, τότε οι δύο γράφοι σίγουρα δεν είναι ισομορφικοί. Υπάρχουν, ωστόσο, κάποιοι μη ισομορφικοί γράφοι, τους οποίους ο αλγόριθμος 1-WL δεν καταφέρνει να διαχωρίσει. Ένα χαρακρητιστικό παράδειγμα ενός τέτοιου ζεύγους γράφων, φαίνεται στην εικόνα 3.7. Όσον αφορά την διαδικασία χρωματισμού του κάθε κόμβου, η λογική είναι ότι σε κάθε βήμα του αλγορίθμου, το νέο χρώμα του κάθε κόμβου προκύπτει από τον κατακερματισμό του τρέχοντος χρώματος του κόμβου, και των τρεχώντων χρωμάτων των γειτονικών του κόμβων. Αυτή η διαδικασία της άσκησης επιρροής στον κάθε κόμβο από την γειτονιά του, αξιοποιείται σε σημαντικό βαθμό από τα Νευρωνικά Δίκτυα Γράφων, τα οποία θα αναλυθούν στην συνέχεια της διατριβής.

1.1.3 Μηχανική Μάθηση σε Γράφους

Στην ενότητα αυτή θα εισαγάγουμε το βασικό υπόβαθρο γύρω από την εφαρμογή μεθόδων μηχανικής μάθησης σε γράφους.

1.1.3.1 Κίνητρο

Όσον αφορά το κίνητρο γύρω από την εμβάθυνση σε αυτόν τον τομέα, ο κύριος λόγος είναι ότι μία **πληθώρα πραγματικών συστημάτων** και συνόλων δεδομένων μπορούν να μοντελοποιηθούν φυσικά ως γράφοι. Με χαρακτηριστικό παράδειγμα τον γράφο τον οποίο σχηματίζουν οι χρήστες μίας πλατφόρμας κοινωνικής δικτύωσης, η εφαρμογή μεθόδων μηχανικής μάθησης πάνω στους

γράφους μπορεί να επιφέρει πληθώρα χρήσιμων προβλέψεων, όπως την πρόβλεψη νέων ακμών, που θα αντιπροσωπεύουν νέες φιλίες στο δίκτυο.

1.1.3.2 Προκλήσεις

Όπως είναι αναμενόμενο, η μηχανική μάθηση στους γράφους εμφανίζει και ένα σοβαρό πλήθος προκλήσεων.

Η πιο σημαντική πρόκληση είναι η αδυναμία των παραδοσιακών μεθόδων μηχανικής μάθησης να εφαρμοστούν αποτελεσματικά αυτούσιες σε γράφους. Οι παραδοσιακές μέθοδοι βαθιάς μηχανικής μάθησης, όπως έχουν εφαρμοσθεί σε τομείς όπως η επεξεργασία φυσικής γλώσσας και η όραση υπολογιστών, απαιτούν ως είσοδο στο σύστημα μία μοντελοποίηση της εισόδου με την μορφή διανυσμάτων. Η αναπαράσταση ενός γράφου με αυτόν τον τρόπο δεν είναι καθόλου προφανής, λόγω της έλλειψης κανονικότητας στην μορφή του. Η άμεση αναπαράσταση ενός γράφου με τον πίνακα γειτνίασής του εμφανίζει αδυναμίες. Στην περίπτωση αραιών γράφων, ο πίνακας γειτνίασης καταλαμβάνει σημαντικά μεγαλύτερο χώρο από τον απαιτούμενο, αυξάνοντας την πολυπλοκότητα του συστήματος. Ταυτόχρονα, ο ίδιος γράφος μπορεί να εκφραστεί με πολλούς διαφορετικούς πίνακες γειτνίασης, αν προηγηθεί μία αναδιάταξη των κόμβων του. Στην περίπτωση, επομένως, που χρησιμοποιηθεί ο πίνακας γειτνίασης ενός γράφου ως είσοδος σε ένα μοντέλο μηχανικής μάθησης, θα πρέπει να εξασφαλιστεί ότι αυτό το μοντέλο θα παράγει την ίδια έξοδο (πρόβλεψη) ανεξαρτήτως αναδιατάξεων.

Μία εξίσου σημαντική διαφοροποίηση της μηχανικής μάθησης στους γράφους, σε σχέση με την παραδοσιακή μηχανική μάθηση σε άλλους τομείς είναι οι αλληλεξαρτήσεις μεταξύ των δειγμάτων της εκπαίδευσης και αξιολόγησης ενός μοντέλου. Για παράδειγμα, όταν ένα μοντέλο εκπαιδεύεται στον τομέα της όρασης υπολογιστών, κάθε εικόνα, δηλαδή κάθε οντότητα, του συνόλου δεδομένων, είναι ανεξάρτητη από την επόμενη. Στην περίπτωση ωστόσο που πραγματοποιούμε μία πρόβλεψη που αφορά στους κόμβους ενός γράφου, οι οντότητες του συνόλου δεδομένων, εμφανίζουν αλληλεξαρτήσεις, αφού μπορεί να συνδέονται με ακμές. Αυτές οι αλληλεξαρτήσεις χρήζουν ειδικής αντιμετώπισης κατά την εκτέλεση διάφορων εργασιών στην εκπαίδευση και στην αξιολόγηση ενός μοντέλου, όπως κατά τον διαχωρισμό του συνόλου δεδομένων, και κατά την εκπαίδευση με χρήση της τεχνικής mini-batching.

1.1.3.3 Ταξινόμηση

Οι εφαρμογές μηχανικής μάθησης στους γράφους μπορούν να κατηγοριοποιηθούν μία πληθώρα κριτηρίων. Η πρωτεύουσα ταξινόμησή τους μπορεί να γίνει με βάση την οντότητα με την οποία σχετίζονται οι προβλέψεις τους. Όπως φαίνεται στην εικόνα 3.11, με βάση αυτό το κριτήριο μπορούν να χωριστούν σε **εφαρμογές επιπέδου γράφου**, **υπογράφου**, **κόμβου**, **και ακμής**. Μερικές χαρακτηριστικές υποεφαρμογές κάθε κατηγορίας είναι η κατηγοριοποίηση γράφων, η πρόβλεψη ετικετών για τους κόμβους, και η πρόβλεψη νέων ακμών στον γράφο αντίστοιχα. Ταυτόχρονα, ομοίως με την παραδοσιακή μηχανική μάθηση, οι εφαρμογές μπορούν να διαχωριστούν, με βάση το σύνολο δεδομένων που χρησιμοποιούν και τον τύπο της πρόβλεψης, σε **επιβλεπόμενες**, **ημί-επιβλεπόμενες**, **και μη-επιβλεπόμενες εφαρμογές**.

1.1.3.4 Παραδοσιακές Τεχνικές

Οι παραδοσιακές τεχνικές γύρω από την μηχανική μάθηση σε γράφους στηρίζονται στην **εξαγωγή "χειροποίητης" πληροφορίας** για κάθε κόμβο, και στην αξιοποίησή της μέσω κάποιου παραδοσιακού μοντέλου μηχανικής μάθησης. Η πληροφορία που θα εξαχθεί για κάθε κόμβο, είναι συνήθως κάποια κλασική μετρική που συνδέεται με την θεωρία γραφημάτων, όπως ο βαθμός ή η κεντρικότητα ενός κόμβου. Στην συνέχεια χρησιμοποιείται για την σύγκριση των οντοτήτων και την εξαγωγή συμπερασμάτων. Το κύριο μειονέκτημα των παραδοσιακών τεχνικών είναι ότι η επιλογή της πληροφορίας που θα εξαχθεί ποικίλει στις διάφορες εφαρμογές, με αποτέλεσμα οι τεχνικές **να μην μπορούν να γενικεύουν** σε διαφορετικές εφαρμογές.

Στις **εφαρμογές επιπέδου κόμβου**, χρησιμοποιούνται μετρικές όπως ο βαθμός και τα διάφορα είδη κεντρικότητας κόμβων. Οι αναλυτικές εξισώσεις υπολογισμού αυτών των μετρικών δίνονται στο κεφάλαιο 3.2.4.1. Στις **εφαρμογές επιπέδου ακμής**, συνηθίζεται η χρήση μετρικών που εκφράζουν την συσχέτιση μεταξύ δύο κόμβων, ώστε να εξαχθούν συμπεράσματα γύρω από την ύπαρξη ακμής μεταξύ των κόμβων αυτών. Μια κατηγοριοποίηση των μετρικών αυτών είναι σε εκείνες που εστιάζουν στην απόσταση των κόμβων, στην επικάλυψη της τοπικής γειτονιάς των κόμβων, και στην επικάλυψη της ολικής γειτονιάς των κόμβων. Μία πιο εκτενής έκθεση των πιο γνωστών μετρικών που χρησιμοποιούνται γι αυτόν τον σκοπό παρατίθεται στο κεφάλαιο 3.2.4.2. Όσον αφορά τις **εφαρμογές επιπέδου γράφων και υπογράφων**, χρησιμοποιείται έντονα η τεχνική των συναρτήσεων πυρήνα (kernel functions), για την αποτελεσματική σύγκριση μεταξύ δύο γράφων. Ταυτόχρονα, επικρατεί μία ακόμα τεχνική, γνωστή στην διεθνή βιβλιογραφία ως **bag-of-nodes**, η οποία προάγει την εξαγωγή ενός συνόλου χαρακτηριστικών για κάθε κόμβο του γράφου, και την αντιμετώπιση του γράφου ως το σύνολο των χαρακτηριστικών αυτών. Το κεφάλαιο 3.2.4.2 εμβαθύνει περαιτέρω σε αυτές τις μεθόδους.

Μία ευρέως διαδεδομένη έννοια γύρω από την μηχανική μάθηση στους γράφους είναι η έννοια των **εμφυτευμάτων κόμβων** (node embeddings). Η διαδικασία παραγωγής εμφυτευμάτων κόμβων μπορεί να ιδωθεί ως η διαδικασία αντιστοίχισης των κόμβων ενός γράφου σε έναν **χώρο εμφυτευμάτων**, όπου η γεωμετρική συσχέτιση μεταξύ δύο εμφυτευμάτων αντιπροσωπεύει την συσχέτιση των κόμβων στον αρχικό γράφο. Τα εμφυτεύματα κόμβων είναι εμπνευσμένα από τον τομέα της επεξεργασίας φυσικής γλώσσας (Natural Language Processing).

Μία γενική προσέγγιση για την παραγωγή εμφυτευμάτων κόμβων είναι το σύστημα κωδικοποιητή-αποκωδικοποιητή, όπου τα εμφυτεύματα όλων των κόμβων του γράφου παράγονται σε έναν πίνακα εμφυτευμάτων, και στην συνέχεια η ανάκτησή τους ολοκληρώνεται μέσω της επιλογής της αντίστοιχης στήλης του πίνακα. Οι μέθοδοι παραγωγής των εμφυτευμάτων βασίζονται σε μία ποικιλία τεχνικών, με πιο γνωστές τις τεχνικές **παραγοντοποίησης πινάκων**, **τυχαίων περιπάτων** και **τυχαίων προβολών**. Στις τεχνικές τυχαίων περιπάτων, η διαίσθηση είναι ότι πραγματοποιείται ένα σύνολο από τυχαίες διασχίσεις του γράφου, που έχουν ωστόσο παραμετροποιήσιμη συμπεριφορά (βλ. 3.2.5.5), και η απόσταση μεταξύ των εμφυτευμάτων δύο κόμβων εξαρτάται από το πλήθος των περιπάτων στους οποίους εμφανίζονται και οι δύο κόμβοι. Όσον αφορά την μέθοδο των τυχαίων προβολών, η διαίσθηση είναι ότι παράγεται μία χαμηλότερης διάστασης αναπαράσταση του γράφου, μέσω της κατασκευής τυχαίων προβολών που αναπαριστούν τις ιδιότητες των κόμβων. Μία πιο εκτενής ανάλυση των πιο διαδεδομένων μεθόδων για την παραγωγή εμφυτευμάτων κόμβων, καθώς και το αλγοριθμικό/μαθηματικό υπόβαθρό τους, δίνεται στο κεφάλαιο 3.2.5.

1.1.4 Πρόβλεψη βάρους ακμής

Το πρόβλημα της **πρόβλεψης βάρους ακμής** σε έναν γράφο, ανήκει προφανώς στην κατηγορία των προβλημάτων **επιπέδου ακμής**. Ο σκοπός του είναι, δεδομένου ενός γράφου G και ενός υποσυνόλου E των ακμών του, να αναπτυχθεί ένα μοντέλο που να μπορεί να προβλέψει το βάρος κάθε ακμής $e \epsilon E$. Το **κίνητρο** γύρω από αυτό το πρόβλημα, είναι ότι σε πραγματικά συστήματα, η επίγνωση της έντασης της αλληλεπίδρασης μεταξύ δύο κόμβων δίνει **περισσότερη πληροφορία** από την επίγνωση της ύπαρξης ή μη της ακμής αυτής. Χαρακτηριστικά, σε μία εφαρμογή κοινωνικής δικτύωσης, η επίγνωση ότι δύο χρήστες έχουν ανταλλάξει 500 μηνύματα, μπορεί να φανεί πολύ πιο χρήσιμη από την πληροφορία ότι οι χρήστες αυτοί έχουν απλώς ανταλλάξει κάποια μηνύματα.

Μία πληθώρα τεχνικών αντιμετώπισης, που βασίζονται τόσο σε παραδοσιακές, όσο και σε πιο σύγχρονες τεχνικές, όπως η βαθιά μηχανική μάθηση και τα νευρωνικά δίκτυα γράφων, έχουν αναπτυχθεί γύρω από την συγκεκριμένη εφαρμογή. Όσον αφορά τις τεχνικές βαθιάς μάθησης, ξεχωρίζει το Model R (Hou and Holder (2017)) και η μετεξέλιξή του, Model S (Hou and Holder (2018)), που διαχωρίζει το στάδιο της παραγωγής των εμφυτευμάτων κόμβων από την τροφοδότησή τους στα κατάλληλα στρώματα νευρωνικών δικτύων, με σκοπό την πρόβλεψη του βάρους της εκάστοτε ακμής.

1.1.5 Νευρωνικά Δίκτυα Γράφων

1.1.5.1 Κίνητρο και Προκλήσεις

Όπως προαναφέρθηκε στα προηγούμενα κεφάλαια, οι παραδοσιακές τεχνικές μηχανικής μάθησης για ευκλείδια δεδομένα δεν μπορούν να εφαρμοστούν άμεσα στους γράφους. Την ίδια στιγμή, οι παραδοσιακές τεχνικές μηχανικής μάθησης πάνω σε γράφους, εμφανίζουν σημαντικούς περιορισμούς, όπως η αδυναμία γενίκευσης σε νέα δεδομένα (κόμβους ή γράφους), και σε διαφορετικού τύπου προβλήματα. Τα Νευρωνικά Δίκτυα Γράφων έχουν ως **κίνητρο** να αντιμετωπίσουν αυτές τις αδυναμίες, και να βελτιώσουν τα αποτελέσματα των μεθόδων μηχανικής μάθησης πάνω σε γράφους. Όσον αφορά στις κύριες **προκλήσεις** που έχουν να αντιμετωπίσουν, αυτές έχουν αναφερθεί στο κεφάλαιο 1.1.3.2.

1.1.5.2 Βασικές έννοιες

Δύο από τις πιο βασικές έννοιες που σχετίζονται με την υλοποίηση και την συμπεριφορά των νευρωνικών δικτύων γράφων, είναι η αμετάβλητη συμπεριφορά τους στις **μεταθέσεις της εισόδου (permutation invariance)**, και η λογική του **περάσματος μηνυμάτων (message passing)** μεταξύ των κόμβων του γράφου.

Η διαίσθηση πίσω από την ανάγκη για αμετάβλητη συμπεριφορά των νευρωνικών δικτύων γράφων στις μεταθέσεις της εισόδου, είναι ότι με είσοδο τον ίδιο γράφο, θα πρέπει να παράγεται η ίδια έξοδος, ανεξάρτητα από την σειρά με την οποία δίνονται ως είσοδος οι κόμβοι του γράφου.

Η τεχνική του περάσματος μηνυμάτων μεταξύ των κόμβων του γράφου είναι, σύμφωνα με τους Gilmer et al. (2017), μία γενίκευση της λειτουργικότητας πολλών νευρωνικών δικτύων γράφων, και στηρίζεται στην ιδέα ότι το εμφύτευμα κάθε κόμβου πρέπει να παράγεται με βάση τα εμφυτεύματα των κόμβων και την δομή της τοπικής του γειτονιάς, όπως φαίνεται στην εικόνα 3.26. Μία εκτενέστερη ανάλυση της τεχνικής αυτής βρίσκεται στο κεφάλαιο 3.4.3.1. Συνοπτικά, σε κάθε πέρασμα k προς

τα εμπρός (forward pass) του αλγορίθμου, κάθε κόμβος υπολογίζει το νέο του εμφύτευμα h_v^{k+1} σύμφωνα με την σχέση Hamilton (2020):

$$m_{N(v)}^{(k)} = AGGREGATE^{(k)}(\{h_u^{(k)}, \forall u \epsilon N(v)\}) \tag{1.1}$$

$$h_v^{k+1} = UPDATE^{(k)}(h_v^{(k)}, m_{N(v)}^{(k)}) \tag{1.2}$$

Όπως φαίνεται στις εξισώσεις 1.1 η συνάρτηση **AGGREGATION**, η οποία αναλαμβάνει την συσσωμάτωση των εμφυτευμάτων των κόμβων που ανήκουν στην γειτονιά του τρέχοντα κόμβου, λαμβάνει το μη διατεταγμένο σύνολο που τα περιέχει. Έτσι, η τεχνική του περάσματος μηνυμάτων, τηρεί την μη μεταβλητότητα της εξόδου στις μεταθέσεις της εισόδου.

Μετά από K βήματα του αλγορίθμου, έχουν παραχθεί τα εμφυτεύματα $h_v^{(K)}$ για κάθε κόμβο v. Στο σημείο αυτό, μπορεί να εφαρμοσθεί το βήμα της ανάγνωσης (readout) του γράφου, ώστε να παραχθεί ένα ολικό εμφύτευμα για ολόκληρο τον γράφο. Η διαδικασία αυτή μοντελοποιείται μαθηματικά ως εξής:

$$\hat{y} = R(\{h_v^{(k)} | v \epsilon G\}) \tag{1.3}$$

Αξίζει να σημειωθεί ότι οι συναρτήσεις **AGGREGATE**, **UPDATE**, και **R** είναι διαφοροποιήσιμες συναρτήσεις που μαθαίνονται από το μοντέλο, κι επομένως στην πράξη μπορούν να αντικατασταθούν από ολόκληρα νευρωνικά δίκτυα.

1.1.5.3 Κατηγοριοποίηση με βάση την Αρχιτεκτονική

Με βάση την αρχιτεκτονική τους, σύμφωνα με τους Wu et al. (2021) και Zhou et al. (2022), τα νευρωνικά δίκτυα γράφων μπορούν να διαχωριστούν στις εξής κατηγορίες:

- **Ανατροφοδοτούμενα (Recurrent) Νευρωνικά Δίκτυα Γράφων (RecGNNs)**
- **Συνελικτικά (Convolutional) Νευρωνικά Δίκτυα Γράφων (ConvGNNs)**
- **Αυτόματοι Κωδικοποιητές Γράφων (GAEs)**
- **Χωροχρονικά (Spatial-Temporal) Νευρωνικά Δίκτυα Γράφων (STGNNs)**

1.1.5.4 Συνελικτικά νευρωνικά δίκτυα γράφων

Η παρούσα διατριβή εστιάζει στην θεωρητική μελέτη και πειραματική αξιολόγηση τεσσάρων συγκεκριμένων τύπων νευρωνικών δικτύων γράφων, που υπάγονται στην κατηγοριά των **συνελικτικών αρχιτεκτονικών**. Όλα τα νευρωνικά δίκτυα που χρησιμοποιήθηκαν, ακολουθούν την διαδικασία του **περάσματος μηνυμάτων**. Οι διαφορές τους εντοπίζονται κυρίως στην υλοποίηση των συναρτήσεων **AGGREGATE** και **UPDATE** της περιγραφείσας διαδικασίας.

Το πρώτο είδος συνελικτικού νευρωνικού δικτύου γράφου που ερευνήθηκε είναι το **GraphSAGE** (Hamilton et al. (2017a)). Η ιδιαιτερότητα του μοντέλου αυτού σε σχέση με την βασική τεχνική του περάσματος μηνυμάτων είναι ότι στο βήμα aggregate δεν λαμβάνονται υπόψην όλοι οι γείτονες του κόμβου, αλλά ένα σταθερό σε μέγεθος δείγμα τους. Η τεχνική αυτή εξομαλύνει τον χρόνο που απαιτείται για την εκτέλεση του βήματος σε κάθε batch, στην περίπτωση χρήσης της τεχνικής mini-batching. Η διαίσθηση πίσω από την διαδικασία οπτικοποιείται στην εικόνα 3.29.

Στην συνέχεια, μελετήθηκε το είδος **k-GNN** (Morris et al. (2018)). Η ιδιαιτερότητα αυτού του τύπου GNN, είναι ότι χρησιμοποιεί μία ακολουθία από πολλά νευρωνικά δίκτυα γράφων k-GNNs, το καθένα από τα οποία εστιάζει στον γράφο σε έναν διαφορετικό βαθμό λεπτομέρειας k. Διαισθητικά, τα k-GNN πραγματοποιούν την διαδικασία του περάσματος μηνυμάτων μεταξύ υπογράφων μεγέθους k, και όχι μεταξύ μεμονωμένων κόμβων (εκτός της προφανούς περίπτωσης $k = 1$). Ως απόρροια, συνδυάζουν αναπαραστάσεις διαφορετικού επιπέδου του γράφου, και καταφέρνουν να εξαγάγουν σύνθετες πληροφίες για την δομή του. Μία οπτικοποίηση της διαδικασίας αυτής φαίνεται στην εικόνα 3.30.

Ένα επιπρόσθετο είδος νευρωνικών δικτύων γράφων που μελετήθηκε είναι τα **Graph Attention Networks (GAT)** (Veličković et al. (2017)). Διαισθητικά, η πρωτοπορία αυτών των δικτύων αυτών είναι ότι εφαρμόζουν τον μηχανισμό της **προσοχής (attention)**, ώστε να λάβουν υπόψην το εμφύτευμα κάθε κόμβου-γείτονα με διαφορετική βαρύτητα κατά την εκτέλεση της συνάρτησης **AGGREGATE**. Για τον σκοπό αυτό, κατά την εκπαίδευση του δικτύου, εκπαιδεύονται και οι αντίστοιχοι **συντελεστές προσοχής**, οι οποίοι συντονίζουν τον μηχανισμό.

Το τελευταίο είδος νευρωνικών δικτύων γράφων που αξιοποιήθηκε ήταν το **Graph Isomorphism Network (GIN)** (Xu et al. (2018)). Το θεωρητικό υπόβαθρο του νευρωνικού αυτού δικτύου βασίζεται στον αλγόριθμο 1-WL, και επιχειρεί, με την χρήση συναρτήσεων ένα-προς-ένα στο βήμα **AGGREGATE**, να αντιστοιχήσει σύνολα εμφυτευμάτων που προέρχονται από διαφορετικές γειτονιές, σε διαφορετικά παραγόμενα εμφυτεύματα. Βασίζεται στο θεώρημα καθολικής προσέγγισης (Universal Approximation Theorem, Hornik (1991)), και χρησιμοποιεί ολόκληρα πολυστρωματικά perceptrons στο στάδιο **AGGREGATE**, ώστε να μιμηθεί ένα-προς-ένα συναρτήσεις. Μία ελαφρώς εκτενέστερη ανάλυση για την λειτουργία του μπορεί να βρεθεί στο κεφάλαιο 3.5.4.

1.1.6 Συστήματα προτάσεων

Ένα σύστημα προτάσεων έχει ως στόχο να προβλέψει την αξιολόγηση ή την προτίμηση ενός χρήστη προς ένα αντικείμενο. Τα συστήματα προτάσεων χρησιμοποιούνται ευρέως στο διαδίκτυο, με χαρακτηριστικό παράδειγμα ένα σύστημα προτάσεων ταινιών, όπως το Netflix. Η διαθέσιμη πληροφορία για ένα σύστημα προτάσεων, μπορεί να αφορά αλληλεπιδράσεις μεταξύ χρηστών και αντικειμένων, δημογραφικά στοιχεία για τον χρήστη, αλλά και πληροφορίες σχετικές με τα αντικείμενα. Μερικές από τις **προκλήσεις** που καλούνται να αντιμετωπίσουν τα συστήματα προτάσεων είναι η **αραιή φύση των αλληλεπιδράσεων** μεταξύ χρηστών και αντικειμένων, η παραγωγή προτάσεων για **νέους χρήστες**, ή η ένταξη **νέων αντικειμένων** στα προτεινόμενα αντικείμενα, καθώς και η ανταπόκριση στα **ιδιόμορφα ενδιαφέροντα** ορισμένων χρηστών του συστήματος.

Με βάση τον τύπο της πληροφορίας που αξιοποιεί ένα σύστημα προτάσεων, η διαδικασία μπορεί να διακριθεί σε **Συνεργατικό Φιλτράρισμα (Collaborative Filtering)**, όπου αξιοποιείται το ιστορικό των αλληλεπιδράσεων μεταξύ των χρηστών και των αντικειμένων για την παραγωγή νέων προτάσεων, **Φιλτράρισμα με βάση το περιεχόμενο (Content-based Filtering)** όπου αξιοποιείται η πληροφορία που σχετίζεται με το περιεχόμενο των αντικειμένων για την παραγωγή νέων προτάσεων, και σε **Υβριδικές Μεθόδους**, όπου συνδυάζονται τεχνικές των δύο προηγούμενων κατηγοριών. Η κατηγοριοποίηση αυτή φαίνεται και στο διάγραμμα της εικόνας 3.33. Κάθε μία μέθοδος παρουσιάζει πλεονεκτήματα και μειονεκτήματα σε σχέση με τις υπόλοιπες. Για παράδειγμα, το φιλτράρισμα με βάση το περιεχόμενο συχνά παρουσιάζει προβλήματα κλιμάκωσης (scalability), αλλά λειτουργεί

καλύτερα σε περιπτώσεις νέων χρηστών. Αντίστοιχα, το συνεργατικό φιλτράρισμα μπορεί να λειτουργήσει χωρίς την απαίτηση ύπαρξης πλούσιας πληροφορίας για τα αντικείμενα, και να παρέχει ποικιλία στις προβλέψεις για κάθε χρήστη, αλλά συχνά υστερεί εξαιτίας των αραιών αλληλεπιδράσεων μεταξύ των χρηστών και των αντικειμένων.

Όσον αφορά την αξιολόγηση ενός συστήματος προτάσεων, εμφανίζεται μία πληθώρα κλασικών, αλλά και πιο ιδιαίτερων μετρικών. Η πρόβλεψη ενός συστήματος προτάσεων, μπορεί να παράγει είτε μία ακριβή βαθμολογία που θα αναθέσει ο χρήστης στο αντικείμενο (regression), είτε μία τιμή αληθείας που να εκφράζει την αρέσκει ή μη του αντικειμένου από τον χρήστη (classification). Ανάλογα με το είδος των προβλέψεων, το σύστημα μπορεί να αξιολογηθεί με τις παραδοσιακές αντίστοιχες μετρικές της μηχανικής μάθησης, είτε με μετρικές προσανατολισμένες στα συστήματα προτάσεων, όπως το Κανονικοποιημένο Προεξοφλημένο Αθροιστικό Κέρδος (Normalized Discounted Cumulative Gain), η ποικιλία (diversity) και η καινοτομία (novelty) των προβλέψεων.

1.1.7 Νευρωνικά Δίκτυα Γράφων και Συστήματα προτάσεων

Η υλοποίηση ενός **συστήματος προτάσεων**, μπορεί να ιδωθεί, μετά από συγκεκριμένη μοντελοποίηση του προβήματος, ως το πρόβλημα της **πρόβλεψης ακμών** (link prediction) σε έναν γράφο.

Τα δεδομένα μπορούν να μοτελοποιηθούν ως ένας **διμερής γράφος**, όπου οι χρήστες και τα αντικείμενα του συστήματος είναι οι δύο τύποι κόμβων. Οι αλληλεπιδράσεις γενικότερα μεταξύ χρηστών και αντικειμένων μπορούν να αποτελέσουν τις ακμές του γράφου. Στην περίπτωση που χτίζουμε ένα σύστημα που να προβλέπει την βαθμολογία που θα υποβάλει ένας χρήστης σε ένα αντικείμενο, η ακμή θα μπορεί να είναι **σταθμισμένη** (weighted), και η πρόβλεψη που καλείται να επιτελέσει το σύστημα να αφορά το βάρος της ακμής.

Επομένως, η μέθοδος των νευρωνικών δικτύων γράφων μπορεί να εφαρμοστεί για την υλοποίηση ενός συστήματος προτάσεων μέσω της επίλυσης του προβλήματος πρόβλεψης ακμής. Σύμφωνα με τους Zhang (2022), οι μέθοδοι γύρω από την πρόβλεψη του βάρους ακμών με χρήση νευρωνικών δικτύων γράφων, κατηγοριοποιούνται σε τεχνικές βασιζόμενες στους κόμβους (node-based), και σε υπογράφους (subgraph-based) του γράφου. Στο πλαίσιο της παρούσας διπλωματικής, χρησιμοποιούμε μία γενικευμένη αρχιτεκτονική, η οποία βασίζεται στα GAEs (Ng et al. (2019)), και διαχωρίζει πλήρως την διαδικασία της παραγωγής εμφυτευμάτων κόμβων από την διαδικασία της αποκωδικοποίησής τους στο βάρος (weight) της αντίστοιχης ακμής. Υπό αυτό το πρίσμα, χρησιμοποιούμε την αρχιτεκτονική του σχήματος 3.35, όπου το σύστημα "prediction head" λαμβάνει ως είσοδο εμφυτεύματα κόμβων, και εξάγει το βάρος της ακμής που αντιστοιχεί σε κάθε ζεύγος χρήστη-αντικειμένου.

1.1.8 Υπάρχουσες μελέτες βασιζόμενες στους γράφους

Επιπροσθέτως της μελέτης που πραγματοποιήσαμε στις αρχιτεκτονικές νευρωνικών δικτύων γράφων της ενότητας 1.1.5.4, μελετήσαμε και ένα πλήθος τεχνικών οι οποίες βασίζονται σε γράφους και πετυχαίνουν state-of-the-art επιδόσεις στο σύνολο δεδομένων MovieLens (Grouplens).

Αξίζει να σημειώσουμε ότι στην υπάρχουσα βιβλιογραφία, η ανάπτυξη ενός συστήματος συστάσεων, η οποία στην παρούσα διπλωματική αντιμετωπίζεται ως το πρόβλημα της πρόβλεψης βάρους ακμής, αντιμετωπίζεται συχνά και ως το πρόβλημα της συμπλήρωσης του πίνακα (matrix completion) που αποτελείται από τις αλληλεπιδράσεις των χρηστών με τα αντικείμενα.

Υπό αυτό το πρίσμα, οι Berg et al. (2017), χρησιμοποιούν τα συνελικτικά νευρωνικά δίκτυα γράφων και τεχνικές συμπλήρωσης πινάκων, ώστε να παράξουν δύο πίνακες που περιέχουν αναπαραστάσεις για τους χρήστες και τις ταινίες. Στην συνέχεια χρησιμοποιούν την πράξη του εσωτερικού γινομένου μεταξύ των αναπαραστάσεων, ώστε να παράξουν τις αντίστοιχες προβλεπόμενες βαθμολογήσεις από κάθε χρήστη για κάθε ταινία. Οι Zhang and Chen (2019) επεκτείνουν την τεχνική αυτή, χρησιμοποιώντας τεχνικές παραγοντοποίησης πινάκων που λειτουργούν επαγωγικά, δηλαδή μπορούν να παράξουν αναπαραστάσεις για νέους χρήστες και νέα αντικείμενα, πάνω στα οποία δεν έχει εκπαιδευτεί το δίκτυο. Μία περαιτέρω επέκταση των δημοσιεύσεων αυτών, αποτελεί το έργο των Hekmatfar et al. (2022), το οποίο αξιοποιεί τον μηχανισμό της προσοχής (attention) κατά την συλλογή των εμφυτευμάτων των κόμβων γειτόνων.

1.2 Υλοποίηση του συστήματος προτάσεων

Σκοπός της διατριβής, ήταν η υλοποίηση ενός συστήματος προτάσεων ταινιών. Η υλοποίηση αυτή πραγματοποιήθηκε μοντελοποιώντας το σύνολο δεδομένων **The Movies Dataset** (Banik (2017)) ως γράφο στην βάση δεδομένων γράφων Neo4j (2012), και αξιοποιώντας τα συνελικτικά νευρωνικά δίκτυα γράφων για την πρόβλεψη των μελλοντικών αξιολογήσεων των ταινιών.

1.2.1 Σύνολο Δεδομένων

Το σύνολο δεδομένων που χρησιμοποιήθηκε για την εκπαίδευση του μοντέλου, αλλά και για την συμπλήρωση της βάσης δεδομένων ήταν το The Movies Dataset (Banik (2017)). Το σύνολο δεδομένων αυτό, αποτελεί μία μετεξέλιξη των ευρέως γνωστών συνόλων δεδομένων MovieLens (Grouplens), και περιλαμβάνει ένα σύνολο από ταινίες, αξιολογήσεις των ταινιών από χρήστες, καθώς και ένα πλούσια μετα-πληροφορία σχετικά με το περιεχόμενο της κάθε ταινίας. Αξίζει να σημειωθεί ότι το The Movies Dataset παρέχει δύο εκδόσεις, που χαρακτηρίζονται από διαφορετικό πλήθος καταγεγραμμένων αξιολογήσεων (100 χιλιάδες αξιολογήσεις στην σύντομη έκδοση του συνόλου δεδομένων, και 25 εκατομμύρια αξιολογήσεις στην πλήρη έκδοσή του). Στα πειράματα της διπλωματικής αυτής, αξιοποιήθηκε η σύντομη έκδοση του συνόλου δεδομένων.

Περνώντας σε πιο τεχνικές λεπτομέρειες, το σύνολο δεδομένων που χρησιμοποιήθηκε αποτελείται από ένα σύνολο αρχείων, όπου το καθένα παρέχει διαφορετικού είδους πληροφορία για το τελικό σύστημα. Για παράδειγμα, υπάρχουν αρχεία αφιερωμένα στην μετα-πληροφορία των ταινιών, αρχεία που αφορούν τις αξιολογήσεις των ταινιών, καθώς και βοηθητικά για την συνένωση των ανωτέρω. Περισσότερες τεχνικές λεπτομέρειες για το περιεχόμενο του συνόλου δεδομένων, θα αναφερθούν στο κεφάλαιο 1.2.2.

1.2.2 Βάση Δεδομένων Γράφων

1.2.2.1 Κίνητρο

Για την αποθήκευση του συνόλου δεδομένων επιλέχθηκε η τεχνολογία των **βάσεων δεδομένων γράφων**. Οι βάσεις δεδομένων γράφων διαφέρουν από τις παραδοσιακές σχεσιακές βάσεις δεδομένων, ως προς το ότι αναπαριστούν την πληροφορία στο εσωτερικό τους υπό την μορφή **κόμβων και ακμών**. Ορίζουν όλες τις πρωταρχικές λειτουργίες τους και τους κανόνες ακεραιότητάς τους υπό

το πρίσμα των γράφων. Έτσι, υπερτερούν έναντι των παραδοσιακών βάσεων δεδομένων στην ταχύτητα διεκπεραίωσης των λειτουργιών που επικεντρώνονται στις **συσχετίσεις μεταξύ των οντοτήτων**, οι οποίες να παρομοιαστούν με περιπάτους σε έναν γράφο.

Στην περίπτωση μίας πλατφόρμας εξερεύνησης και σύστασης ταινιών, είναι σύνηθες να ζητείται πληροφορία που επικεντρώνεται στις αλληλεξαρτήσεις μεταξύ των οντοτήτων (πχ ταινιών και ηθοποιών), οι οποίες ανακτώνται πιο αποδοτικά μέσω μίας βάσης δεδομένων γράφων. Ταυτόχρονα, η μοντελοποίηση του συνόλου δεδομένων σε μία βάση δεδομένων γράφων όπως είναι η Neo4j (2012), μας επέτρεψε να αξιοποίησουμε μία ποικιλία αλγορίθμων που προσφέρει, ώστε να εξαγάγουμε χρήσιμη πληροφορία από τον γράφο, και να την αξιοποιήσουμε ως επιπρόσθετη είσοδο του συστήματος προτάσεων.

1.2.2.2 Μοντελοποίηση του συνόλου δεδομένων ως γράφο - Προεπεξεργασία

Όσον αφορά την **διαδικασία μοντελοποίησης** του **The Movies Dataset** ως γράφο, επιλέξαμε να αναπαραστήσουμε με ξεχωριστό **τύπο κόμβων** τις ταινίες, τα είδη των ταινιών, τις λέξεις-κλειδιά τους, τα μέλη-συνεργεία τους, τις χώρες, εταιρείες και γλώσσες παραγωγής τους, καθώς και τους χρήστες του συστήματος. Προφανώς, στην βάση αποθηκεύτηκαν οι αντίστοιχοι **τύποι ακμών** για να δηλώσουν τις συσχετίσεις μεταξύ των ανωτέρω οντοτήτων, όπως για παράδειγμα το γεγονός ότι μία ταινία υπάγεται σε ένα συγκεκριμένο είδος. Η λίστα των τύπων κόμβων και συσχετίσεων που χρησιμοποιήθηκαν αναλύεται στο κεφάλαιο 4.2.3. Αναφορικά με την τεχνική υλοποίηση της διαδικασίας αυτής, χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python, και η βιβλιοθήκη Py2neo (2020) για το διάβασμα και την επεξεργασία των αρχείων του συνόλου δεδομένων, και αποθήκευσή τους στην βάση δεδομένων υπό την κατάλληλη μορφή.

Στην συνέχεια, αξιοποιήθηκε η βιβλιοθήκη GDS (2022), η οποία παρέχει ένα σύνολο αποδοτικών υλοποιήσεων ευρέως διαδεμένων αλγορίθμων μηχανικής μάθησης σε γράφους. Υπό την κατάλληλη επεξεργασία του πηγαίου κώδικα, συντονίστηκε μία διαδικασία παραγωγής εμφυτευμάτων κόμβων για τις ταινίες του γράφου, ώστε να κωδικοποιείται κατάλληλα η μετα-πληροφορία που σχετίζεται με το περιεχόμενό τους στον γράφο (είδη, λέξεις-κλειδιά, κλπ). Τα εμφυτεύματα αυτά παράχθηκαν με τρεις διαφορετικούς αλγορίθμους, που ακολουθούν διαφορετικές τεχνικές, και στην συνέχεια αποθηκεύτηκαν ως πληροφορίες για την κάθε ταινία, ώστε να αξιοποιηθούν ως είσοδος στο νευρωνικό δίκτυο γράφων που θα πραγματοποιεί τις προβλέψεις. Αξιοποιήθηκαν οι αλγόριθμοι Node2vec (Grover and Leskovec (2016)), FastRP (Chen et al. (2019)), και το νευρωνικό δίκτυο GraphSAGE (Hamilton et al. (2017a)). Στα διαγράμματα 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19 έχουν οπτικοποιηθεί με την βιβλιοθήκη scikit-learn (Pedregosa et al. (2011)), μετά από μείωση διαστατικότητας„ χαρακτηριστικές περιπτώσεις των εμφυτευμάτων που παρήχθησαν.

1.2.3 Το μοντέλο

Το κύριο μέρος του συστήματος προτάσεων, δηλαδή το νευρωνικό δίκτυο γράφων, υλοποιήθηκε στην γλώσσα προγραμματισμού Python, με την χρήση της βιβλιοθήκης Pytroch Geometric (Fey and Lenssen (2019)). Η βιβλιοθήκη αυτή, χρησιμοποείται για την ανάπτυξη μοντέλων σε μη ευκλείδια δεδομένα, όπως οι γράφοι. Προσφέρει αποδοτικές υλοποιήσεις από ένα σημαντικό αριθμό νευρωνικών δικτύων γράφων, αλλά και παραμετροποιήσιμες υλοποιήσεις μεμονωμένων στρωμάτων τους.

Όσον αφορά την αρχιτεκτονική που ακολουθήθηκε για την υλοποίηση των μοντέλων στην παρούσα διατριβή, χρησιμοποιήθηκαν δύο υπό-μοντέλα, που ανέλαβαν αντίστοιχα την κωδικοποίηση της πληροφορίας του γράφου σε εμφυτεύματα κόμβων, και την αποκωδικοποίηση αυτών των εμφυτευμάτων σε προβλέψεις των βαρών των αντίστοιχων ακμών. Η αρχιτεκτονική αυτή οπτικοποιείται στο διάγραμμα 4.20, όπου διακρίνεται εποπτικά ο ρόλος του **κωδικοποιητή (GNNEncoder)** και του **αποκωδικοποιητή (EdgeDecoder)**.

1.2.3.1 Ο κωδικοποιητής

Ο κωδικοποιητής (GNNEncoder) είναι ένα **βαθύ νευρωνικό δίκτυο γράφων**, που δέχεται ως είσοδο τον **διμερή γράφο** αποτελούμενο από ταινίες, χρήστες, και αξιολογήσεις, και παράγει ως έξοδο τα **εμφυτεύματα κόμβων** που αντιστοιχούν σε κάθε ταινία και χρήστη. Για την υλοποίηση του κωδικοποιητή μελετήθηκαν τα **συνελικτικά νευρωνικά δίκτυα γράφων** GraphSAGE (3.5.1), k-GNN (3.11), GAT (3.5.3), και GIN (3.5.4). Όπως θα αναλυθεί στην συνέχεια, μία πληθώρα **υπερπαραμέτρων** συντονίστηκε κατά την διεξαγωγή των πειραμάτων. Μία από αυτές τις υπερπαραμέτρους, η οποία σχετίζεται με την δομή του κωδικοποιητή, είναι το **βάθος** του δικτύου. Διαισθητικά, με την χρήση ενός στρώματος, ο κάθε κόμβος υπολογίζει το νέο του εμφύτευμα λαμβάνοντας υπόψην του μόνο τα εμφυτεύματα των άμεσων γειτόνων του. Ομοίως, με χρήση δύο στρωμάτων, λαμβάνει υπόψην του και τα εμφυτεύματα των γειτόνων των άμεσων γεινότων του, κοκ. Το σύνολο των κόμβων αυτών, που ρυθμίζεται από το βάθος του νευρωνικού δικτύου, ονομάζεται **δεκτικό πεδίο** κάθε κόμβου. Επομένως, όπως οπτικοποιείται στην εικόνα 4.23, η αλόγιστη αύξηση του βάθους, μπορεί να οδηγήσει σε σημαντικές επικαλύψεις των δεκτικών πεδίων των κόμβων του γράφου, με αποτέλεσμα την σύγκλιση των εμφυτευμάτων που παράγονται. Αυτή η σύγκλιση είναι αρνητική για την ποιότητα του μοντέλου, και αντιμετωπίζεται είτε με περιορισμό του βάθους του δικτύου, είτε με χρήση ειδικών τεχνικών που παρουσιάζονται αναλυτικά στο κεφάλαιο 4.3.3.1. Η επίσημη αγγλική ορολογία του φαινομένου δίνεται από τον όρο **oversmoothing**.

1.2.3.2 Ο αποκωδικοποιητής

Ο αποκωδικοποιητής είναι ένα απλό πολυστρωματικό feed-forward νευρωνικό δίκτυο, που δέχεται ως είσοδο την έξοδο του κωδικοποιητή, δηλαδή τα παραγόμενα εμφυτεύματα για τις ταινίες και τους χρήστες, και παράγει ως έξοδο τις προβλέψεις για τα βάρη όλων των ακμών, δηλαδή τις αξιολογήσεις μεταξύ χρηστών και ταινιών.

1.2.4 Αξιολόγηση του συστήματος προτάσεων

Για να αποφασιστούν οι κατάλληλες μετρικές για την αξιολόγηση του συστήματος, μελετήθηκε η υπάρχουσα έρευνα γύρω από το σύνολο δεδομένων MovieLens (Grouplens), λόγω της ομοιότητάς του με το σύνολο δεδομένων που χρησιμοποιήθηκε στα πειράματα, και της πανομοιότυπης φύσης των προβλέψεων που συνηθίζονται στα δύο σύνολα δεδομένων. Όπως παρατηρήθηκε από μία πληθώρα δημοσιεύσεων (Han et al. (2021), Rashed et al. (2019), Wu et al. (2021), Darban and Valipour (2022)), η πρόβλεψη βάρους ακμής στο MovieLens αντιμετωπίζεται ως ένα πρόβλημα παλινδρόμησης, κι επομένως οι πιο ευρέως διαδεδομένες μετρικές για την αξιολόγηση των λύσεών του είναι οι μετρικές που χρησιμοποιούνται παραδοσιακά στην μηχανική μάθηση στα προβλήματα αυτής της

κατηγορίας. Οι μετρικές που επιτεύχθηκαν στις ανωτέρω δημοσιεύσεις θα χρησιμοποιηθούν ως ένα αρχικό μέτρο αναφοράς για τα πειράματά μας. Ταυτόχρονα, όπως θα αναλυθεί και στην ενότητα που θα αφιερωθεί στα πειράματα που διεξήχθησαν στην παρούσα διατριβή, ως μέτρο αναφοράς για την αξιολόγηση των μοντέλων που αναπτύσσονται, θα χρησιμοποιηθεί ένα αρχικό απλοϊκό μοντέλο GraphSAGE, το οποίο δεν θα χρησιμοποιεί την μετα-πληροφορία που παρέχεται για τις ταινίες, ώστε να διαπιστώσουμε πιο συστηματικά την επίδραση της μετα-πληροφορίας συναρτήσει της αρχιτεκτονικής και των υπερπαραμέτρων του συστήματος στην ποιότητα των προβλέψεων, υπό τις ίδιες συνθήκες εκτέλεσης των πειραμάτων.

1.3 Υλοποίηση της πλατφόρμας

Ένα σημαντικό μέρος της προσπάθειας που καταβλήθηκε στην παρούσα διατριβή, αφιερώθηκε στην υλοποίηση μίας ολοκληρωμένης πλατφόρμας εξερεύνησης και σύστασης ταινιών.

1.3.1 Γενικά στοιχεία και κίνητρο

Αναπτύξαμε έξι κύρια ανεξάρτητα συστατικά στοιχεία, τα οποία συγχρονίστηκαν ώστε να λειτουργούν συνδυαστικά και παρέχουν μία ολοκληρωμένη και απρόσκοπτη εμπειρία χρήστη. Δύο βασικά συστατικά στοιχεία της πλατφόρμας είναι το **νευρωνικό δίκτυο γράφων** που πραγματοποιεί τις προβλέψεις, και η **βάση δεδομένων γράφων** η οποία είναι υπεύθυνση για την αποθήκευση των δεδομένων. Αναπτύχθηκαν τρεις ακόμα ανεξάρτητες **υπηρεσίες στην πλευρά του server** (back-end), και μία **υπηρεσία στην πλευρά του client**, για την πρόσβαση στις λειτουργικότητες των ανωτέρω υπηρεσιών μέσω ενός περιηγητή ιστού. Η αρχιτεκτονική του συστήματος οπτικοποιείται στο διάγραμμα 5.1. Η υλοποίηση της πλατφόρμας εμπνεύστηκε από την ιστοσελίδα MovieLens, η οποία αποτελεί μία ιστοσελίδα σύστασης ταινιών, που έχει αναπτυχθεί από ερευνητές στο πανεπιστήμιο της Minnesota, και βρίσκεται σε λειτουργία από το 1997. Σκοπός της πλατφόρμας είναι να διευκολύνει τον τελικό χρήστη να εξερευνήσει απρόσκοπτα την υποκείμενη δομή του γράφου, ανακαλύπτοντας νέες ταινίες και είδη ταινιών, αλλά και να του παρέχει ποιοτικές προβλέψεις οι οποίες είναι σύμφωνες με τις προτιμήσεις του.

1.3.2 Συστατικά Στοιχεία

Η γενικότερη ιδέα στην οποία βασίστηκε η αρχιτεκτονική του συστήματος, είναι η **αποσύζευξη ανεξάρτητων λειτουργιών** σε διαφορετικές υπηρεσίες (services) που αναπτύσσονται ακολουθώντας τις αρχές των REST προγραμματιστικών διεπαφών, με σκοπό την κάλυψη ζητημάτων **επίδοσης**, αναγκών **κλιμάκωσης**, και ζητημάτων **ασφάλειας** στην αναπτυσσόμενη πλατφόρμα. Πιο αναλυτικά, οι τρεις υπηρεσίες που υλοποιήθηκαν στην **πλευρά του server** είναι οι εξής:

- **Movies API**: Αναλαμβάνει τον χειρισμό των μετα-δεδομένων των ταινιών. Υλοποιήθηκε με την βιλβιοθήκη NodeJS (Foundation (2009)), χρησιμοποιώντας ως γλώσσα προγραμματισμού την Typescript (Corporation (2012)).
- **Users API**: Αναλαμβάνει τον χειρισμό των χρηστών, των αξιολογήσεων, και την αυθεντικοποίηση των χρηστών. Υλοποιήθηκε με την βιλβιοθήκη NodeJS (Foundation (2009)), χρησιμοποιώντας ως γλώσσα προγραμματισμού την Typescript (Corporation (2012)).

- **Model API**: Αναλαμβάνει την ενθυλάκωση του μοντέλου που πραγματοποιεί τις προβλέψεις σε μία υπηρεσία που εξωτερικεύει τις απαιτούμενες λειτουργικότητές του. Υλοποιήθηκε με την βιλβιοθήκη Flask (Team (2010)), χρησιμοποιώντας ως γλώσσα προγραμματισμού την Python.

Η διεπαφή χρήστη αναπτύχθηκε ως ανεξάρτητη υπηρεσία λαμβάνει την απαιτούμενη πληροφορία από την πλευρά του server, και παρέχει στον χρήστη μία φιλική διεπαφή για την αξιοποίησή της. Υλοποιήθηκε με την βιβλιοθήκη ReactJS (Facebook (2013)), χρησιμοποιώντας ως γλώσσες προγραμματισμού την Javascript (Eich (1995)) και Typescript (Corporation (2012)).

1.3.3 Ενσωμάτωση του μοντέλου στην πλατφόρμα

Η ενσωμάτωση των λειτουργιών του μοντέλου στην συνολική πλατφόρμα, και η εναρμόνιση της λειτουργίας του συστήματος είναι ένα σύνθετο πεδίο, το οποίο μπορεί να αποτελέσει ανεξάρτητο πεδίο έρευνας. Στην παρούσα διπλωματική, οι σχετικές σχεδιαστικές αποφάσεις λήφθηκαν με γνώμονα την ασφάλεια και την δυνατότητα κλιμάκωσης των λειτουργιών του μοντέλου. Το μοντέλο τοποθετήθηκε σε **χωριστή υπηρεσία**, ώστε να μπορούν να του ανατεθούν περισσότεροι **υπολογιστικοί πόροι** από τις υπόλοιπες υπηρεσίες για την ταχεία διεκπεραίωση των σύνθετων λειτουργιών του. Την ίδια στιγμή επιλέχθηκε για **λόγους ασφάλειας** να απαγορευτεί η άμεση επικοινωνία μεταξύ του μοντέλου και του client μέρους της πλατφόρμας, ώστε ο κάθε χρήστης να μην έχει πρόσβαση μόνο στα δεδομένα που αφορούν τους υπόλοιπους χρήστες. Η πρόσβαση στις ακμές που προβλέπει το μοντέλο για τον κάθε χρήστη, γίνονται **μέσω του Users API**, το οποίο διαθέτει **μηχανισμό αυθεντικοποίησης** των χρηστών. Παράλληλα αφιερώθηκε σημαντικό μέρος της διαδικασίας ανάπτυξης του λογισμικού, ώστε να διασφαλιστεί η αποδοτική λειτουργία του αντίστοιχου API. Πιο συγκεκριμένα, διασφαλίσαμε την **διαθεσιμότητα** του Model API, κατά την διάρκεια διεκπεραίωσης απαιτητικών λειτουργιών, όπως η επανεκπαίδευση του μοντέλου, μέσω της χρήσης **πολλαπλών χωριστών νημάτων** εκτέλεσης των διεργασιών. Ταυτόχρονα, αναπτύξαμε μία μέθοδο, ώστε να προσπεράσουμε την αδυναμία του μοντέλου μας να χειριστεί γράφους με περισσότερους χρήστες από τον αρχικό γράφο στον οποίο έχει εκπαιδευτεί. Υλοποιήσαμε την **δυναμική** προσθήκη στηλών στον **πίνακα βαρών** του πρώτου στρώματος του νευρωνικού δικτύου γράφων του μοντέλου μας, κατά την προσθήκη ενός **νέου χρήστη** στο σύστημα, ώστε να μπορεί να πραγματοποιείται ο αντίστοιχος πολλαπλασιασμός πινάκων στο εσωτερικό του μοντέλου. Όσον αφορά την **αρχικοποίηση** των νέων στηλών, αποφασίσαμε, με γνώνομα την δυνατότητα λήψης ουσιαστικών προβλέψεων για τους νέους χρήστες χωρίς την ανάγκη επανεκπαίδευσης του μοντέλου, να χρησιμοποιήσουμε τις μέσες τιμές των αντίστοιχων βαρών των υπάρχοντων χρηστών. Κατά αυτόν τον τρόπο, οι νέοι χρήστες, κληρονομούν τα βάρη του **μέσου χρήστη** του συστήματός μας, και λαμβάνουν τις αντίστοιχες προβλέψεις, με βάση τις βαθμολογήσεις που έχουν υποβάλει, χωρίς την ανάγκη επανεκπαίδευσης του μοντέλου. Καταφέραμε, κατ'αυτόν τον τρόπο να προσεγγίσουμε την **επαγωγική** (inductive) λειτουργία των νευρωνικών δικτύων γράφων.

1.3.3.1 Διεπαφή Χρήστη

Η διεπαφή χρήστη σχεδιάστηκε με γνώμονα την απρόσκοπτη εξερεύνηση της πληροφορίας και την γρήγορη πρόσβαση στις προβλέψεις του μοντέλου. Για τον σκοπό αυτό, υλοποιήθηκαν συνοπτικές

όψεις για κάθε οντότητα, που επιτρέπουν την άμεση πρόσβαση στις συνδεόμενες εγγραφές της βάσης δεδομένων. Υλοποιήθηκαν οπτικοποιήσεις του γράφου με την ακριβή μορφή που έχει αποθηκευτεί στην βάση δεδομένων, και λειτουργίες αναζήτησης ταινιών με βάση τον τίτλο τους ή τις συνδεόμενες οντότητες, όπως τα είδη και οι λέξεις-κλειδιά τους. Στις εικόνες 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, και 5.8 φαίνεται ένα δείγμα της πληθώρας όψεων που παρέχει η πλατφόρμα υπό αυτό το πρίσμα.

1.4 Πειράματα

1.4.1 Αξιολόγηση

Η κύρια μετρική που χρησιμοποιήθηκε για την αξιολόγηση των προβλέψεων που πραγματοποιούν τα μοντέλα που αναπτύξαμε, ήταν η μετρική Root Mean Square Error, που αποτελεί κλασικό μέτρο αξιολόγησης για προβλήματα γραμμικής παλινδρόμησης. Η μετρική αυτή χρησιμοποιείται σε ένα εύρος προϋπάρχουσων δημοσιεύσεων που σχετίζονται με το σύνολο δεδομένων MovieLens Grouplens.

1.4.2 Υπερπαράμετροι

Το μεγαλύτερο μέρος της πειραματικής διαδικασίας που ακολουθήθηκε, αφιερώθηκε στην εύρεση κατάλληλων τιμών για την πληθώρα των υπερπαραμέτρων που διέπουν το σύστημά μας, ώστε να πετύχουμε όσο το δυνατόν ποιοτικότερες προβλέψεις για τους χρήστες της πλατφόρμας. Εποπτικά, οι υπερπαράμετροι του συστήματος μπορούν να ομαδοποιηθούν στις εξής κατηγορίες:

1. **Αρχιτεκτονική του Νευρωνικού Δικτύου Γράφων** Η **αρχιτεκτονική** του Νευρωνικού Δικτύου Γράφων που χρησιμοποιείται στον κωδικοποιητή του μοντέλου, αποτελεί έναν καθοριστικό παράγοντα για τις παραγόμενες προβλέψεις. Η αρχιτεκτονική αυτή καθορίζει τις συναρτήσεις που πραγματοποιούν την άθροιση των εμφυτευμάτων κόμβων των γειτόνων κάθε κόμβου, κατά την διαδικασία του **περάσματος μηνυμάτων** (**Message Passing Framework**). Κατά αυτόν τον τρόπο, μπορεί να καθορίσει σε σημαντικό βαθμό, όπως θα φανεί στην συνέχεια στα πειράματα που εκτελέσαμε, τις επιδόσεις του εκάστοτε μοντέλου. Οι αρχιτεκτονικές που χρησιμοποιήσαμε εμείς είναι οι τέσσερις αρχιτεκτονικές που έχουν αναλυθεί στο κεφάλαιο 1.1.5.4, δηλαδή οι αρχιτεκτονικές **GraphSAGE**, **GAT**, **k-GNN**, και **GIN**.

2. **Αλγόριθμος για την παραγωγή εμφυτευμάτων κόμβων** Σε ένα πλήθος πειραμάτων της παρούσας μελέτης, χρησιμοποιούμε ένα σύνολο **εμφυτευμάτων κόμβων** το οποίο κωδικοποιεί την μεταπληροφορία των ταινιών, και δίδεται ως είσοδος στο Νευρωνικό Δίκτυο Γράφων υπό την μορφή χαρακτηριστικών των ταινιών-κόμβων. Είναι αναμενόμενο ο αλγόριθμος που έχει χρησιμοποιηθεί για την παραγωγή των εμφυτευμάτων αυτών, να επηρεάζει σε σημαντικό βαθμό την ποιότητά τους, και τον βαθμό στον οποίο καταφέρνουν να κωδικοποιήσουν την δομή που υποβόσκει στον αρχικό γράφο. Υπό αυτό το πρίσμα πειραματιζόμαστε με τρεις διαφορετικούς αλγορίθμους, οι οποίοι διαφέρουν στην τεχνική με την οποία εξάγουν τα εμφυτεύματα, ώστε να διαπιστώσουμε ποιος αλγόριθμος οδηγεί σε καλύτερες επιδόσεις στο παρόν σύστημα. Οι αλγόριθμοι που χρησιμοποιούμε έχουν καταγραφεί στο κεφάλαιο 1.2.2.2, και είναι οι Node2Vec (Grover and Leskovec (2016)), FastRP (Chen et al. (2019)), καθώς και το νευρωνικό δίκτυο γράφων GraphSAGE (Hamilton et al. (2017a)).

3. **Υπογράφος για την παραγωγή εμφυτευμάτων κόμβων** Μία επιπρόσθετη συνιστώσα που καθορίζει την ποιότητα των παραγόμενων εμφυτευμάτων κόμβων για την μεταπληροφορία των ταινιών είναι ο αντίστοιχος υπογράφος που χρησιμοποιείται. Για παράδειγμα, εφαρμόζοντας κάποιον από τους προαναφερθέντες αλγορίθμους στον διμερή υπογράφο που αποτελείται μόνο από ταινίες και λέξεις-κλειδιά, μπορούμε να παράξουμε για κάθε ταινία ένα εμφύτευμα το οποίο κωδικοποιεί τις λέξεις-κλειδιά της. Υπό αυτό το πρίσμα, κατηγοριοποιούμε τα πειράματά μας ως προς την επιλογή του υπογράφου παραγωγής των εμφυτευμάτων κόμβων σε τρεις κύριες περιπτώσεις:

    - Δεν χρησιμοποιούμε εμφυτεύματα κόμβων που να σχετίζονται με το περιεχόμενο των ταινιών για καμία ταινία
    - Για κάθε ταινία, παράγουμε ένα σύνολο εμφυτευμάτων κόμβων, κάθε ένα από τα οποία κωδικοποιεί τις συσχετίσεις της με μία από τις υπόλοιπες οντότητες, έχοντας παραχθεί στον αντίστοιχο διμερή υπογράφο. Για παράδειγμα, για κάθε ταινία παράγουμε ένα εμφύτευμα που να κωδικοποιεί τα είδη της, ένα εμφύτευμα που να κωδικοποεί τις λέξεις κλειδία της, κλπ.
    - Για κάθε ταινία, παράγουμε ένα εμφύτευμα κόμβων, που κωδικοποιεί ολόκληρο το περιεχόμενό της έχοντας παραχθεί σε ολόκληρο τον υπογράφο των ταινιών και των συσχετιζόμενων οντοτήτων τους (εκτός των χρηστών)

4. **Δομή του μοντέλου** Ένα σημαντικό σύνολο υπερπαραμέτρων του μοντέλου, αποτελεί τις κλασικές υπερπαραμέτρους που δοκιμάζονται παραδοσιακά στην βαθιά μηχανική μάθηση. Χαρακτηριστικά παραδείγματα υπερπαραμέτρων που υπάγονται σε αυτήν την κατηγορία είναι το πλήθος των στρωμάτων του κωδικοποιητή, το πλήθος των στρωμάτων του αποκωδικοποιητή, και ο αριθμός των κρυφών κόμβων σε κάθε στρώμα.

1.4.3 Μέτρο αναφοράς

Για να σχηματίσουμε ένα μέτρο αναφοράς για τις επιδόσεις των πειραμάτων που πραγματοποιήσαμε, κινηθήκαμε σε δύο άξονες. Αφενός μελετήσαμε τις επιδόσεις **υπάρχουσων μελετών** που έχουν γίνει σε αντίστοιχα σύνολα δεδομένων, και αφετέρου καταγράψαμε την επίδοση μίας **αρχικής απλοϊκής έκδοσης της αρχιτεκτονικής** που χρησιμοποιούμε. Όσον αφορά τις υπάρχουσες μελέτες, δεδομένου ότι το The Movies Dataset (Banik (2017)) είναι ένα σχετικά καινούργιο σύνολο δεδομένων που απαιτεί ευρεία προεπεξεργασία για να αξιοποιηθεί στην ανάπτυξη συστημάτων συστάσεων, δεν βρήκαμε κάποια επίσημη μελέτη που να το αφορά, στο πρόβλημα της πρόβλεψης βάρους ακμής. Γι αυτόν τον λόγο, εστιάσαμε σε μελέτες που έχουν πραγματοποιηθεί στο ίδιο πρόβλημα στο σύνολο δεδομένων **MovieLens** (Grouplens), το οποίο άλλωστε αποτελεί τον κορμό του συνόλου δεδομένων μας. Ταυτόχρονα, εκτελέσαμε κάποια αρχικά πειράματα σε μία απλοϊκή έκδοση της αρχιτεκτονικής μας, χρησιμοποιώντας το μοντέλο GraphSAGE ως το νευρωνικό δικτύο γράφων μας, χωρίς να αξιοποιούμε εμφυτεύματα κόμβων που να σχετίζονται με μεταδεδομένα των ταινιών, και με κάποιες αρχικές αυθαίρετες τιμές για τις παραδοσιακές υπερπαραμέτρους της αρχιτεκτονικής. Η πορεία του αντίστοιχου πειράματος οπτικοποιείται στα διαγράμματα 6.1, και 6.2.

Μία συνολική εικόνα των επιδόσεων που παρατηρήθηκαν σε υπάρχουσες μελέτες, και της επίδοσης του αρχικού μας πειράματος, βρίσκεται στον πίνακα 6.1.

1.4.4 Περαιτέρω Πειράματα στην Σύγκριση Μοντέλων

Ως επόμενο βήμα, μετά από τον καθορισμό των μέτρων αναφοράς για τις επιδόσεις των μοντέλων μας, προχωρήσαμε σε διαδοχικά πειράματα με σκοπό την εύρεση κατάλληλων τιμών για τις υπερπαραμέτρους του μοντέλου.

1.4.4.1 Αρχική Σύγκριση Αρχιτεκτονικών GNN

Το πρώτο πείραμα που πραγματοποιήσαμε αποσκοπεί σε μία αρχική σύγκριση των επιδόσεων που πετυχαίνει κάθε μία από τις τέσσερις αρχιτεκτονικές νευρωνικών δικτύων γράφων με τις οποίες ασχοληθήκαμε στην παρούσα διπλωματική.

Επιλέγοντας κάποιες αρχικές τιμές για τις παραδοσιακές υπερπαραμέτρους του προβλήματος, και χωρίς να αξιοποιούμε εμφυτεύματα κόμβων που να σχετίζονται με την μεταπληροφορία των ταινιών, εκπαιδεύσαμε από ένα μοντέλο για κάθε μία βασική αρχιτεκτονική. Τα αποτελέσματα έχουν οπτικοποιηθεί στα διαγράμαατα 6.3, και 6.4. Εκ πρώτης όψεως, φαίνεται ότι οι αρχιτεκτονικές GraphSAGE και GAT αποδίδουν καλύτερα, ενώ μας προβληματίζει η ιδιαίτερα ασταθής εκπαίδευση της αρχιτεκτονικής GIN.

1.4.4.2 Αριθμός Στρωμάτων στον Κωδικοποιητή

Στην συνέχεια, πειραματιστήκαμε με τον αριθμό των στρωμάτων στο νευρωνικό δίκτυο γράφων του κωδικοποιητή, έχοντας υπόψην το πρόβλημα του **oversmoothing** που περιγράφεται στην ενότητα 1.2.3.1.

Πραγματοποιήσαμε χωριστά πειράματα σε κάθε αρχιτεκτονική νευρωνικών δικτύων γράφων, διαχωρίζοντας ταυτόχρονα τις περιπτώσεις αξιοποίησης των εμφυτευματών κόμβων που σχετίζονται με την μεταπληροφορία των ταινιών, ή μη αξιοποίησής τους. Πράγματι, όπως είναι αναμενόμενο, παρατηρούμε ότι είτε αξιοποιώντας τα εμφυτεύματα κόμβων που σχετίζονται με την διαθέσιμη μεταπληροφορία για τις ταινίες, είτε όχι, ο βέλιστος αριθμός στρωμάτων για το νευρωνικό δίκτυο γράφων είναι μικρός. Τα αποτελέσματα φαίνονται αναλυτικά στα διαγράμματα 6.5, 6.6, 6.7, και 6.8, καθώς και στους πίνακες 6.2, και 6.3. Όσον αφορά την παράλληλη σύγκριση μεταξύ των αρχιτεκτονικών του νευρωνικόυ δικτύου γράφων, παρατηρούμε ότι οι αρχιτεκτονικές **GraphSAGE**, **GAT**, και **GIN** παρουσιάζουν αρκετά παρόμοιες επιδόσεις, με την αρχιτεκτονική **k-GNN (GraphConv)** να υστερεί σημαντικά.

1.4.4.3 Αριθμός Νευρώνων στα Στρώματα των Νευρωνικών Δικτύων

Ως επόμενο βήμα, πειραματιζόμαστε με τον αριθμό των νευρώνων στα στρώματα των νευρωνικών δικτύων, σε κάθε ένα από τα μοντέλα που ξεχωρίσαμε στην προηγούμενη ενότητα.

Για κάθε αρχιτεκτονική νευρωνικού δικτύου, και τον βέλτιστο έως τώρα συνδυασμό της με τον αριθμό των στρωμάτων και την αξιοποίηση ή μη των εμφυτευμάτων κόμβων, πειραματιστήκαμε με το πλήθος των νευρώνων στο κάθε στρώμα εξερευνώντας τιμές στο εύρος [16, 128]. Για κάθε περίπτωση καταλήγουμε σε έναν διαφορετικό αριθμό νευρώνων ως βέλτιση επιλογή, με τα αποτελέσματα να συνοψίζονται στους πίνακες 6.4, 6.5 και στα διαγράμματα 6.9, 6.10, 6.11, και 6.12.

1.4.4.4 Αξιοποίηση Εμφυτευμάτων Κόμβων για την Μεταπληροφορία των Ταινιών

Ακολούθως, προχωρήσαμε σε πληθώρα πειραμάτων για να καταλήξουμε σε ένα συμπέρασμα σχετικά με την επίδραση της χρήσης των εμφυτευμάτων κόμβων που κωδικοποιούν την μεταπληροφορία των ταινιών, στην τελική επίδοση των μοντέλων μας.

Για τον σκοπό αυτό, ξεκινήσαμε από την βέλτιστη έως τώρα έκδοση κάθε μοντέλου, υπό την προϋπόθεση αξιοποίησης των εμφυτευμάτων, και πειραματιστήκαμε στην διαδικασία παραγωγής των εμφυτευμάτων. Εξερευνήσαμε τους τρεις διαφορετικούς αλγορίθμους παραγωγής Node2Vec, FastRP, και GraphSAGE, ενώ παράλληλα δοκιμάσαμε πληθώρα συνδυασμών των υπογράφων που χρησιμοποιούνται στην διαδικασία. Η πορεία των πειραμάτων για τις αρχιτεκτονικές GraphSAGE και GAT οπτικοποιείται στα διαγράμματα 6.13, και 6.14. Οι επιδόσεις των βέλτιστων συνδυασμών για κάθε μοντέλο συνοψίζονται στον πίνακα 6.6.

Αξίζει να τονίσουμε ότι το μοντέλο GraphSAGE φαίνεται να πετυχαίνει την χαμηλότερη απώλεια από τις υπόλοιπες αρχιτεκτονικές. Ένα σημαντικό συμπέρασμα εξάγεται, επίσης, από την γραφική παράσταση 6.15, όπου συγκρίνεται για κάθε αρχιτεκτονική η απώλεια στην περίπτωση μη αξιοποίησης των εμφυτευμάτων κόμβων, και η απώλεια στην περίπτωση αξιοποίησής της. Παρατηρούμε ότι σε όλες τις αρχιτεκτονικές, καταφέραμε να βελτιώσουμε τις επιδόσεις των μοντέλων.

1.5 Προβλέποντας τις κριτικές ενός συγκεκριμένου χρήστη

Στα πειράματα της προηγούμενης ενότητας, ρυθμίσαμε τις τιμές των ποικίλων υπερπαραμέτρων των πειραμάτων και αξιολογήσαμε τα παραγόμενα μοντέλα με βάση την μετρική Root Mean Square Error στο σύνολο δεδομένων.

Στην τρέχουσα ενότητα, υπό την σκέψη της συμμετοχής των μοντέλων που χτίστηκαν σε μία πλατφόρμα συστάσεων ταινιών, επιλέγουμε το καλυτερο μοντέλο GraphSAGE της προηγούμενης ενότητας, και προβαίνουμε σε κάποια πιο πρακτικά πειράματα. Πιο συγκεκριμένα, παρατηρούμε τις προβλέψεις του μοντέλου για έναν συγκεκριμένο χρήστη, συγκρίνοντάς τες με την έως τώρα συμπεριφορά του χρήστη στην πλατφόρμα.

1.5.1 Προσωποποιημένες προβλέψεις

Στο πλαίσιο της οικογένειας των πιο πρακτικών πειραμάτων, ξεκινάμε ελέγχοντας κατά πόσο οι προβλέψεις που πραγματοποιεί το μοντέλο είναι προσωποποιημένες στην συμπεριφορά του κάθε χρήστη.

Επιλέγουμε δύο χρήστες με διαφορετική συμπεριφορά, η οποία εκφράζεται ως προς την κατανομή των τιμών των βαθμολογιών που έχουν δημοσιεύσει έως τώρα. Στα διαγράμματα 6.16 και 6.17 παρατηρούμε ότι πράγματι, οι κατανομές των προβλεπόμενων βαθμολογιών για κάθε χρήστη είναι σε αντιστοιχία με την έως τώρα συμπεριφορά του.

1.5.2 Προσαρμογή σε αλλαγές συμπεριφοράς ενός χρήστη

Ένα βασικό στοιχείο το οποίο καλούμαστε να εισάγουμε στο σύστημα συστάσεων που υλοποιούμε, είναι η παροχή προβλέψεων που συνάδουν με την τρέχουσα βαθμολογική συμπεριφορά ενός χρήστη.

Πιο συγκεκριμένα, στο συγκεκριμένο πείραμα, εντοπίζουμε έναν χρήστη της πλατφόρμας ο οποίος υποβάλλει συνεχώς υψηλές βαθμολογίες, όπως φαίνεται στην κατανομή του διαγράμματος 6.18. Όπως είναι αναμενόμενο, η κατανομή των προβλεπόμενων βαθμολογιών γι αυτόν τον χρήστη κινείται σε υψηλές τιμές. Μοντελοποιούμε μία αλλαγή συμπεριφοράς του χρήστη, μέσω της υποβολής χαμηλών βαμολογιών από τον συγκεκριμένο χρήστη στην πλατφόρμα, έτσι ώστε η κατανομή των βαθμολογιών του να αλλάξει όπως φαίνεται στο διάγραμμα 6.19, μετατοπιζόμενη σημαντικά προς χαμηλότερες τιμές. Επανεκπαιδεύουμε το μοντέλο για ένα αριθμό εποχών μικρότερο της αρχικής εκπαίδευσης που χρειάστηκε, και παρατηρούμε στο διάγραμμα 6.19 ότι οι προβλεπόμενες βαθμολογίες για τον συγκεκριμένο χρήστη έχουν μετακινηθεί προς χαμηλότερες τιμές, ώστε να βρίσκονται σε αντιστοιχία με την συνολική συμπεριφορά του χρήστη.

Η δυνατότητα αυτή του μοντέλου μπορεί να αποτελέσει καθοριστικό παράγοντα για την ευχαρίστηση των μακροχρόνιων χρηστών της πλατφόρμας, αφού θα λαμβάνουν συνεχώς προτάσεις συντονισμένες με την συμπεριφορά τους.

1.5.3 Κατανόηση πιο σύνθετων συμπεριφορών

Στα έως τώρα πειράματα, η κατανομή των βαθμολογιών ενός χρήστη βρισκόταν επί το πλείστον συγκεντρωμένη σε ένα συγκεκριμένο και σχετικά περιορισμένο εύρος τιμών. Ένα σημαντικό πείραμα για την αξιολόγηση του μοντέλου μας, είναι πώς αυτό ανταποκρίνεται σε πιο σύνθετες συμπεριφορές χρηστών, όπου οι βαθμολογίες τους εκτείνονται σε μεγαλύτερο εύρος τιμών.

Στο συγκεκριμένο πείραμα, προσομοιώνουμε την συμπεριφορά ενός χρήστη, ο οποίος παραθέτει βαθμολογίες σε ένα μεγάλο εύρος τιμών, βαθμολογώντας και τις ταινίες που του αρέσουν και εκείνες που δεν του αρέσουν. Επιλέγουμε τυχαία έναν χρήστη του συστήματος και μιμούμαστε την αρέσκειά του σε ταινίες του είδους *Έγκλημα* και την δυσαρέσκειά του απέναντι σε ταινίες του είδους *Περιπέτεια*. Για να το πετύχουμε αυτό, βαθμολογούμε με σχετικά χαμηλή τιμή (στο εύρος [0.5, 2]) ταινίες του είδους *Περιπέτεια*, και με υψηλή τιμή (στο εύρος [3, 4.5]) ταινίες του είδους *Έγκλημα*. Κατ' αυτόν τον τρόπο, σχηματίζεται η κατανομή του διαγράμματος 6.20. Παρατηρούμε ότι το μοντέλο προσεγγίζει την πιο σύνθετη κατανομή αυτή με μικρότερη ακρίβεια από ότι προσέγγιζε τις πιο απλοϊκές κατανομές των προηγούμενων πειραμάτων. Ωστόσο, παρατηρούμε μία τάση να σχηματιστούν δύο συστάδες βαθμολογιών. Η πιο σημαντική παρατήρηση στο συγκεκριμένο πείραμα γίνεται οπτικοποιώντας χωριστά τις προβλεπόμενες βαθμολογίες που αφορούν τον χρήστη για κάθε ένα από τα δύο είδη ταινιών. Όπως φαίνεται σε αυτήν την οπτικοποίηση στο διάγραμμα 6.21, το σύστημα τείνει να προβλέψει ότι οι περισσότερες ταινίες του είδους *Έγκλημα* πρέπει να βαθμολογηθούν με μεγαλύτερη τιμή από ότι οι ταινίες του είδους *Περιπέτεια*, δείχνοντας να προσεγγίζει αυτήν την πιο σύνθετη κατανομή βαθμολογιών για τον συγκεκριμένο χρήστη.

Ωστόσο, μπορούμε να παρατηρήσουμε την μικρότερη ακρίβεια που παρουσιάζουν οι προβλέψεις του μοντέλου, σε σχέση με τα προηγούμενα πειράματα. Προβληματιζόμενοι από το γεγονός αυτό, πραγματοποιούμε έναν δειγματοληπτικό έλεγχο στην κατανομή των βαθμολογιών που εμφανίζουν οι χρήστες του συνόλου δεδομένων, και παρατηρούμε ότι η πλειοψηφία των χρηστών εμφανίζει πιο απλοϊκή συμπεριφορά. Επομένως, λαμβάνοντας υπόψην και το σχετικά περιορισμένο μέγεθος του συνόλου δεδομένων που χρησιμοποιήσαμε, συμπεραίνουμε ότι το σύστημα προτάσεων δεν ανταποκρίνεται εξίσου καλά σε πιο σύνθετες συμπεριφορές χρηστών, λόγω της περιορισμένης εκπαίδευσής του σε τέτοια παραδείγματα.

Chapter 2

Introduction

2.1 Recommender Systems

The field of recommendation systems has gained significant attention in recent years. The increasing amount of available content on the web has increased the need for personalized recommendations. Due to the increasing availability of large-scale rating data, there is a growing interest in developing graph-based recommendation systems. By modeling the relationships between items and users as a graph, we can utilize graph-based methods to leverage the graph's rich structural information.

2.2 Existing Approaches

The development of recommender systems that perform personalized and useful recommendations for each user has been an area of research for multiple years. The existing approaches are in general divided into collaborative filtering, content-based filtering, and hybrid approaches. In collaborative filtering approaches, the recommendations are based on past interactions between users and items, without leveraging metadata about the items. In contrast, content-based approaches rely on utilizing the metadata on each item's content. Finally, hybrid approaches combine both previous methods. Our contribution, which will be reported in the following section, can be categorized as a hybrid approach. In the following chapters, and more specifically in chapter 3, we will report the most important aspects of existing graph-based approaches for implementing recommender systems based on the MovieLens 100K dataset (Grouplens).

2.3 Our Contribution

In this thesis, we focus on the problem of movie recommendation. We develop an end-to-end movie recommendations platform, leveraging the data structure of graphs in every part of the process. In order to mimic a real-world application, we choose a relatively new dataset, The Movies Dataset (Banik (2017)) which combines a real-world historical record of users and movies interaction, and is enriched with a variety of movie metadata. We model the certain dataset as a graph, in the graph database system Neo4j (Neo4j (2012)), and execute multiple graph algorithms to generate node embeddings and gain insights into the complex relationships between the movies, based on their metadata graph structure. Subsequently, we leverage these embeddings, enriching the bipartite graph of users and movies as node features, and feed the new graph into graph neural networks. Utilizing an architecture with two main modules, a GNN Encoder, and an Edge Decoder, we predict the weights

of the edges between users and movies on the bipartite graph. Equivalently, we predict the rating that a user would submit for a certain movie. Leveraging these predicted ratings, we can recommend new movies to a user, based on his/her past interactions and on the movies' metadata. We perform multiple experiments, developing and exploring a variety of GNN architectures, using the Pytorch Geometric framework (Fey and Lenssen (2019)) and tuning the system's hyperparameters. We show that after tuning the hyperparameters related to the node embeddings generation, we can achieve predictions with lower losses in all the model architectures, compared to not using these embeddings at all. The source code that was developed for the purpose of this thesis can be found in the corresponding GitHub repository, at https://github.com/John-Atha/diploma.

2.4 Thesis outline

The outline of this thesis is the following:

- To provide a comprehensive understanding of our study, we begin with a thorough introduction to the fundamental concepts of graph theory, and machine learning on graphs. Subsequently, we explore the link weight prediction task and its relationship with recommender systems. At the same time, we delve deeper into the topic, analyzing the **theoretical background** of graph neural networks as a promising solution for these problems.
- As a next step, we cover the process of **implementing the recommendation** process that powers our platform, beginning from modeling the dataset as a graph in our graph database and finishing with the architecture and the technical details of our machine learning model.
- Followingly, we dive into some technical aspects of the **main components of the platform**, exploring their respective roles, as long as the process of integrating the recommendations model into the system.
- As a natural next step, we present the most important parts of our **experimental process**. We showcase the experiments performed to tune the model's hyperparameters, and some practical experiments on the model's predictions, trying to identify the strengths and the weaknesses a potential user would experience with our platform.

Chapter 3

Theoretical Background

3.1 Graphs

This section will cover the basic concepts around Graphs. These concepts are utilized to represent a real-world dataset as a Graph, and store it in a Graph Database Management System, such as Neo4j (2012). Moreover, the majority of these concepts are necessary to understand and extend the traditional techniques around Machine Learning on Graphs (3.2.3.2), and the implementation of Graph Neural Networks(3.4).

3.1.1 Main definitions

Definition 3.1.1 (Graph): *A **graph** is a pair $G = (V, E)$ of sets such that $E \subseteq |V|^2$; thus, the elements of E are 2-element subsets of V . To avoid notational ambiguities, we shall always assume tacitly that $V \cap E = \emptyset$. The elements of V are the vertices (or nodes, or points) of the graph G, the elements of E are its edges (or lines). Diestel (2010)*

The usual way to visualize a Graph, is by drawing a dot for its vertices, and a line for its edges, with each line connecting the corresponding dots. A visual representation of a Graph can be seen in figure 3.1.

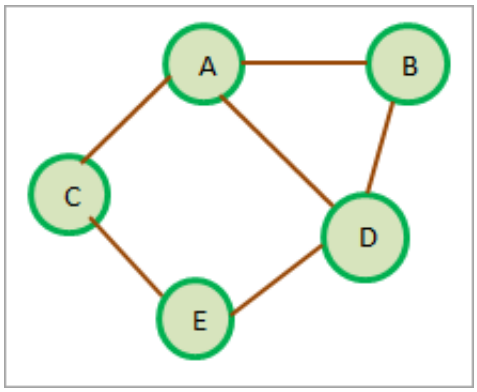

Figure 3.1: Visual Representation of a Graph
SoftwareHelpingTest

Definition 3.1.2 (Directed Graph): *A **directed graph** (or digraph) is a pair (V,E) of disjoint sets (of vertices and edges) together with two maps $init : E \rightarrow V$ and $ter : E \rightarrow V$ assigning to every edge e an initial vertex $init(e)$ and a terminal vertex $ter(e)$. The edge e is said to be directed from $init(e)$ to*

$ter(e)$. *Note that a directed graph may have several edges between the same two vertices* x, y. *Such edges are called multiple edges; if they have the same direction (say from x to y), they are parallel. If* $init(e) = ter(e)$, *the edge e is called a loop.* Diestel (2010)

The usual way to visualize a Directed Graph, is by drawing an arrow (instead of a simple line) for the edges, indicating the direction. The different visualizations of a Directed and an Undirected Graph can be seen in figure 3.2.

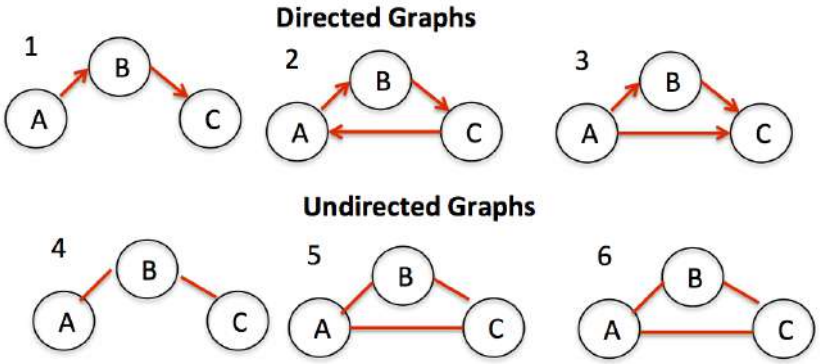

Figure 3.2: Visual Comparison of a Directed and an Undirected Graph
Christuniversity

Definition 3.1.3 (Subgraph and Supergraph): *Let* $G = (V, E)$ *and* $G' = (V', E')$ *be two graphs. If* $V \subseteq V'$ *and* $E \subseteq E'$, *then G' is a* ***subgraph*** *of G (and G a* ***supergraph*** *of G'), written as* $G' \subseteq G$. *Less formally, we say that G contains G'.* Diestel (2010)

Definition 3.1.4 (Graph Isomorphism): *Let* $G = (V, E)$ *and* $G' = (V', E')$ *be two graphs. A map* $\phi : V \longrightarrow V$ *is a* ***homomorphism*** *from* G *to* G' *if it preserves the adjacency of vertices, that is, if* $\{\phi(x), \phi(y)\} \epsilon E'$ *whenever* $\{x, y\} \epsilon E$. *Then, in particular, for every vertex* x' *in the image of* ϕ *its inverse image* $\phi^{-1}(x')$ *is an independent set of vertices in G. If* ϕ *is bijective and its inverse* ϕ^{-1} *is also a homomorphism (so that* $xy \epsilon E \Longleftrightarrow \phi(x)\phi(y) \epsilon E' \forall x, y \epsilon V$ *), we call* ϕ *an* ***isomorphism****, say that* G *and* G' *are isomorphic, and write* $G \simeq$*G'.*Diestel (2010)

An example of two isomorphic graphs can be seen in figure 3.3.

Definition 3.1.5 (Incident Vertices): *A vertex v is* ***incident*** *with an edge e if* $v \in e$; *then e is an edge at v. The two vertices incident with an edge are its endvertices or ends, and an edge joins its ends. An edge x, y is usually written as xy (or yx).* Diestel (2010)

Definition 3.1.6 (Adjacent or Neighbour Vertices and Edges): *Two vertices x, y of G are* ***adjacent****, or* ***neighbours****, if x, y is an edge of G. Two edges* $e \neq f$ *are adjacent if they have an end in common.* Diestel (2010)

Definition 3.1.7 (Vertices distance): *The* ***distance*** $d_G(x, y)$ *in G of two vertices x, y is the length of a shortest x–y path in G; if no such path exists, we set* $d(x, y) := \infty$. Diestel (2010)

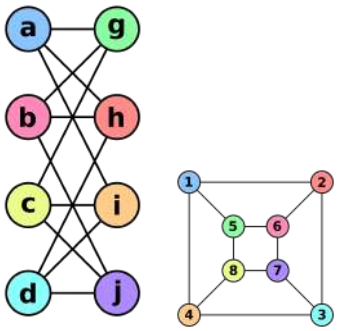

Figure 3.3: Two Isomorphic Graphs
Wikipedia (a) Wikipedia (b)

Definition 3.1.8 (Node degree): *The **degree** (or valency) $d_G(v) = d(v)$ of a vertex v is the number $|E(v)|$ of edges at v. A vertex of degree 0 is **isolated**. Diestel (2010)*

Definition 3.1.9 (Vertex i[th] Neighbourhood): *Let $G = (V, E)$ be a (non-empty) graph. The set of neighbours of a vertex v in G is denoted by $N_G(v)$, or briefly by $N^1_{(v)}$, and is called the **neighbourhood** of the vertex. The set of the vertices with distance at most i from the vertex v, is called the **ith neighourhood** of the vertex and is denoted by $N^i_{(v)}$.*

Definition 3.1.10 (Walk, Path): *A **walk** (of length k) in a graph G is a non-empty alternating sequence $v_0e_0v_1e_1...e_{k-1}v_k$ of vertices and edges in G such that $e_i = v_i, v_{i+1}$ for all $i < k$. If $v_0 = v_k$, the walk is closed. If the vertices in a walk are all distinct, it defines an obvious **path** in G. Diestel (2010)*

3.1.2 Mathematical Representation of Graphs

A well-known mathematical representation of a finite Graph G is its **Adjacency Matrix** $\boldsymbol{A}$. For a finite Graph with N nodes, the Adjacency matrix is a square $N \times N$ matrix. An example of the Adjacency Matrix of a Graph is seen in figure 3.4. The value of the element A_{ij} is calculated based on the formula:

$$Aij = \begin{cases} 1 & (i,j) \in E \\ 0 & otherwise \end{cases}$$

In the case of an undirected graph, the Adjacency Matrix is symmetric.

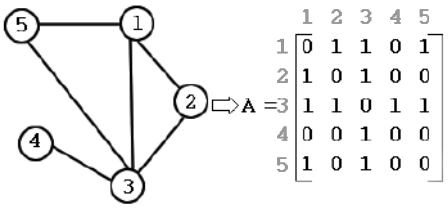

Figure 3.4: Adjacency Matrix of a directed Graph
Yepke

3.1.3 Complex Graphs

3.1.3.1 Weighted Graphs

Definition 3.1.11 (Weighted Graph): *Let $G = (V, E)$ be a graph. With each edge e of G let there be associated a real number w(e), called its weight. Then G, together with these weights on its edges, is called a* ***weighted*** *graph. Bondy and Murty (1976)*

In recent decades, there is increasingly challenging care to study weighted networks. As stated by Boccaletti et al. (2006), this is motivated by the fact that in most real cases, a complex topology is often associated with a large heterogeneity in the capacity and intensity of the connections.

In weighted Graphs, the Adjacency Matrix is slightly modified, and computed by the following formula:

$$Aij = \begin{cases} w(i,j) & (i,j) \in E \\ 0 & otherwise \end{cases}$$

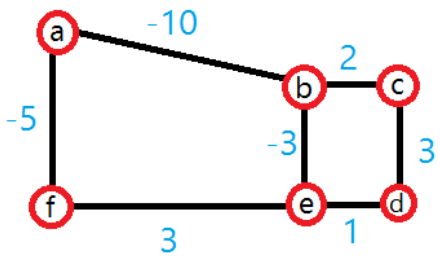

Figure 3.5: Visual Representation of a Weighted Graph
Milad

3.1.3.2 Labeled Graphs

A **Labeled** Graph is a type of Graph, whose vertices are assigned to ”labels”, i.e. an element from a set of symbols. Taking into account these labels, the nodes of the Graph can be grouped into categories.

3.1.3.3 Homogeneous and Heterogeneous Graphs

When modeling real-world data in a Graph, it is common case that we want to distinguish the nodes and the edges, by grouping them into categories.

Definition 3.1.12 (Heterogeneous Graphs): *In heterogeneous graphs, nodes are imbued with types, meaning that we can partition the set of nodes into disjoint sets $V = V_1 \cup V_2 \cup ... \cup V_k$ where $V_i \cap V_j = \varnothing, \forall i \neq j$. Edges in heterogeneous graphs generally satisfy constraints according to the node types, most commonly the constraint that certain edges only connect nodes of certain types. Hamilton (2020)*

Graphs that are not Heterogeneous, are called Homogeneous. A common example of a Homogeneous real-world Graph would be a social media Graph, consisting of nodes that represent users, and edges that represent friendships between the users.

A common example of a Heterogeneous real-world Graph is a Graph with users and movies as nodes, and ratings as edges between a user and a movie. An example Graph of this type can be seen in figure 3.6.

3.1.3.4 Bipartite Graphs

Definition 3.1.13 (Bipartite Graph): *Let $r \leq 2$ be an integer. A graph $G = (V, E)$ is called **r-partite** if V admits a partition into r classes such that every edge has its ends in different classes: vertices in the same partition class must not be adjacent. Instead of '2-partite' one usually says **bipartite**. Diestel (2010)*

Bipartite Graphs are commonly used to represent the data for a recommender system, as shown in figure 3.6, with users belonging to the one set, and the content to be recommended belonging to the other set.

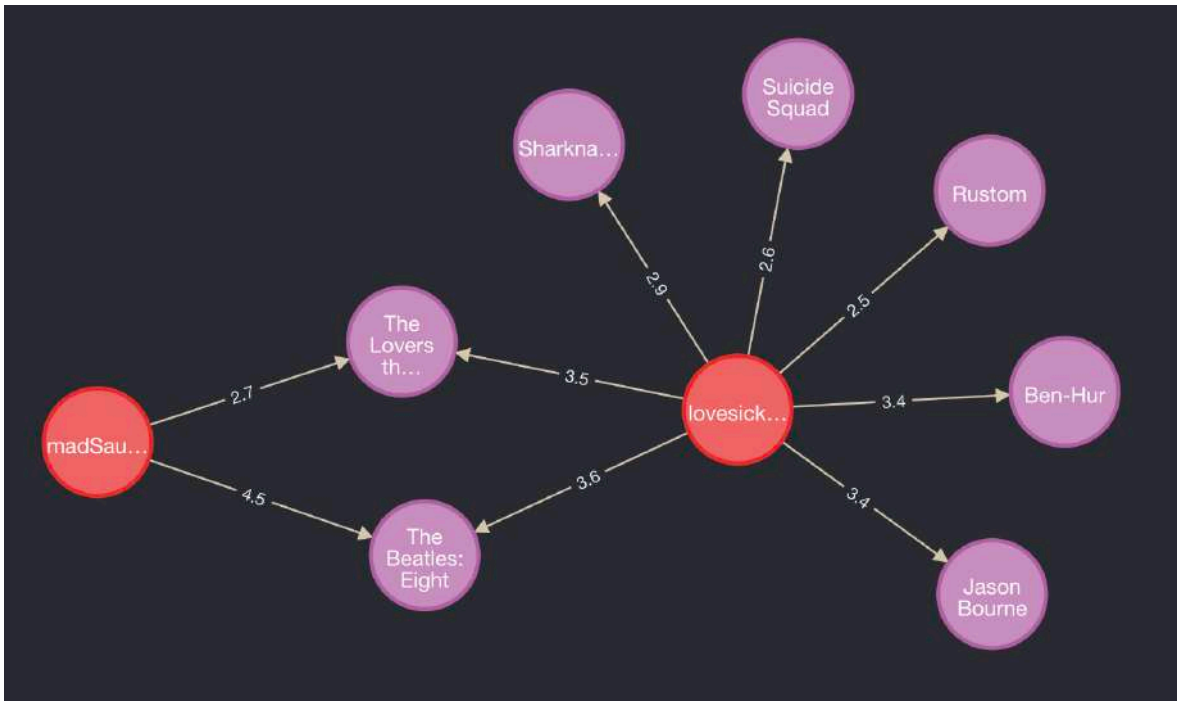

Figure 3.6: Visual Representation of a Heterogeneous, Directed and Weighted Graph. Users are nodes denoted with red color, movies are nodes denoted with purple color, and ratings are modeled as directed and weighted edges, with the weight corresponding to the exact value of the rating. This image corresponds to a subgraph of the graph database instance that was utilized for our research, visualized with Neo4j (2021).

3.1.4 The Weisfeiler-Lehman algorithm on Isomorphism

As stated in the definition 3.1.1, two graphs are considered isomorphic if there is a mapping between the graph's nodes, such that the adjacencies are the same. The graph isomorphism task aims to decide whether two graphs are isomorphic, and is a computationally expensive task. The **Weisfeiler-Lehman algorithm** (Leman (2018)) is a graph-isomorphism test. To test the isomorphism of two graphs, the algorithm produces a canonical form for each one of them. If the canonical forms are

not equivalent, then the graphs are certainly non-isomorphic. However, it is important to note that two non-isomorphic graphs can have two equivalent canonical forms, and therefore the algorithm can fail two distinguish these graphs. An example of such an edge case can be seen in figure 3.7. The algorithm has multiple variants, such as the 1-WL, the k-WL, and the k-FWL. The **1-WL** variant is the most classic variant of the algorithm and is based on the **color refinement** method. As per Sato (2020), for a more strict mathematical representation of the algorithm, the following elements are defined:

The **input** of the algorithm is a pair of graphs $G = (V, E, X)$ and $H = (U, F, Y)$.

The **output** of the algorithm is to decide the existence of a bijection $f : V- > U$ such that $X_v = Y_{f(v)} \forall v \epsilon V$ and $\{u, v\} \epsilon E$ iff $\{f(u), f(v)\} \epsilon F$.

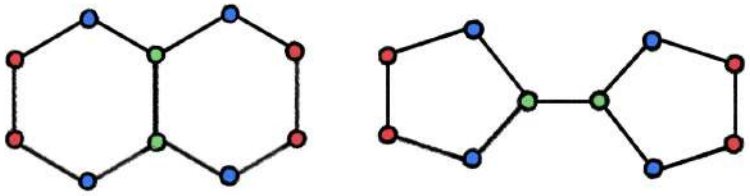

Figure 3.7: Visual Representation of two non-isomorphic graphs that cannot be distinguished by the WL test

Bronstein

3.1.4.1 The 1-WL algorithm

The most standard WL test variant is based on the color refinement method. That is, it is based on the iterative recoloring of the Graph's nodes, starting with identical colors on all nodes. An overview of the algorithm's steps is described in Hamilton (2020):

1. Given two graphs $G1$ and $G2$, we assign an initial label $l^{(0)}_{G_i}(v)$ to each node in each graph. In most graphs, this label is simply the node degree, i.e., $l^{(0)}(v) = d_v \forall v \epsilon V$, but if we have discrete features (i.e., one-hot-features x_v) associated with the nodes, then we can use these features to define the initial labels.

2. Next, we iteratively assign a new label to each node in each graph by hashing the multi-set of the current labels within the node's neighborhood, as well as the node's current label:

$$l^{(i)}_{G_i}(v) = HASH(l^{(i-1)}_{G_i}(v), \{\{l^{(i-1)}_{G_i}(u) \forall u \epsilon N(v)\}\})$$

   where the double braces are used to denote a multi-set and the HASH function maps each unique multi-set to a unique new label.

3. We repeat Step 2 until the labels for all nodes in both graphs converge, i.e., until we reach an iteration K where $l^{(K)}_{G_j}(v) = l^{(K-1)}_{G_i}(v), \forall v \epsilon V_j, j = 1, 2$.

4. Finally, we construct multi-sets

$$L_{G_j} = \{\{L^{(i)}_{G_j}(v), \forall v \epsilon V_j, i = 0, ..., K-1\}\}$$

summarizing all the node labels in each graph, and we declare G1 and G2 to be isomorphic if and only if the multi-sets for both graphs are identical, i.e., if and only if $L_{G_1} = L_{G_2}$.

3.2 Machine Learning on graphs

This section covers the basic concepts around Machine Learning on Graphs. At first, the motivation behind performing Machine Learning Tasks on Graphs is analyzed. Next, the main challenges of these types of Tasks are covered. Lastly, we dive into more technical ideas, such as the Taxonomy of these Tasks, and the concept of node embeddings.

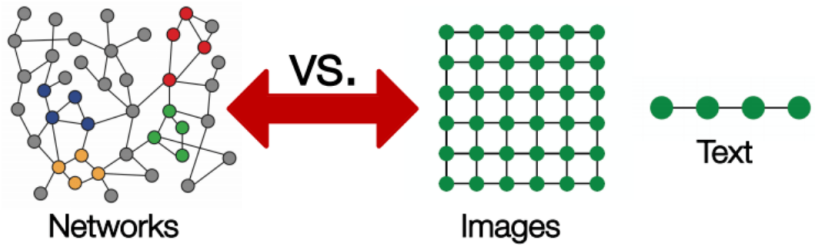

Figure 3.8: Visual Comparison of Graph (ir)regularity in graphs, images, and text Stanford (2021)

3.2.1 Motivation

Many real-world systems and interactions can be naturally represented as Graphs. Graphs exist in multiple scenarios around us, and therefore the ability to extract meaningful information from them can seem really useful. The scale and the complexity of graphs and other non-Euclidean, geometric data handled by software systems make their management a natural target for machine learning techniques. As stated by Bronstein et al. (2017), **Geometric Deep Learning** is an umbrella term for emerging techniques attempting to generalize (structured) deep neural models to non-Euclidean domains such as graphs and manifolds. Some examples of large-scale real-world applications of Geometric Deep Learning are the following:

- By analyzing the structure of a social network Graph, the engineers of a **Social Media Platform** can produce interesting insights, and apply Machine Learning Techniques to identify communities or recommend new friends to users based on their Graph neighborhood.
- The engineers of a **Movies Platform** can utilize the Graph structure of the available data (such as users, movies, genres, ratings, actors, etc.) and make useful recommendations to users about new movies.
- Graphs directly work for the representation of **molecules**, having a variety of applications in the fields of chemistry and materials science.

Apart from these obvious Graph structures around us, the concepts of Graph Theory can be also applied to other types of data, such as text and images, making Graphs relative to Machine Learning Domains such as **Natural Language Processing** and **Computer Vision**. A visualization relative to representing Natural Language sentences as a Graph can be seen in figure 3.9.

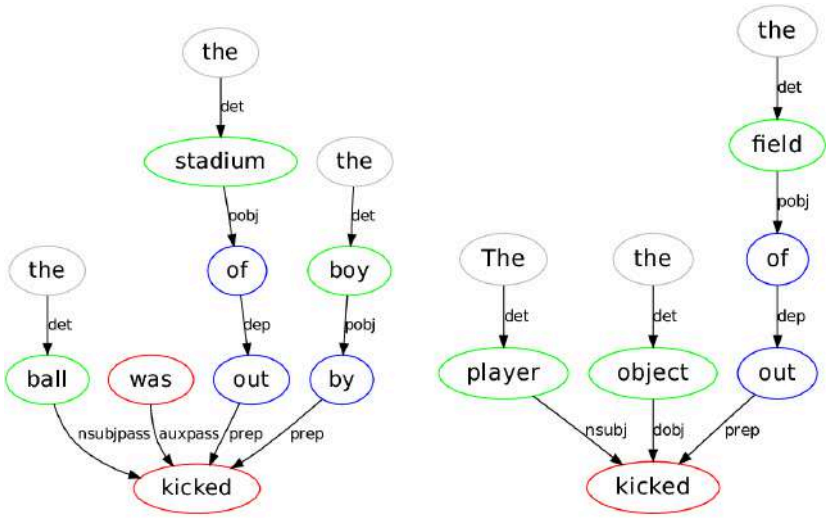

Figure 3.9: Visual Representation of Natural Language Sentences as a Graph Røkenes (2012)

3.2.2 Challenges

Deep Neural Networks have been extremely successful on data with an underlying Euclidean or grid-like structure, such as images or text. However, these Euclidean data have significant differences from the general-case natural Graph.

First, the main difference between the two types of data is the **regularity** of their structure, as can be seen in figure 3.8. Euclidean data, such as images, have a certain structure and can be modeled as regular Graphs, meaning that all nodes have an equal degree. Images can be visualized as fixed-size grid structures, as can be seen in figure 3.10. Whereas, in general, a Graph does not necessarily have a fixed number of nodes, and is not necessarily regular, meaning that each node can have a different-sized neighborhood.

As stated by Sanchez-Lengeling et al. (2021), another decision that needs to be made for performing Machine Learning Tasks on Graphs, is how Graphs will be represented in order to be compatible with neural networks. These complex data structures have up to four types of information that a Machine Learning model could utilize to make predictions; nodes, edges, global context, and connectivity. Nodes, edges and global context can be stored in matrices, that can be processed by the Deep Neural Networks that specialize in Euclidean data. In contrast, representing the connectivity of a Graph in an efficient way is much more complicated. A straightforward way to represent a Graph's connectivity would be to use the **Adjacency Matrix** of the Graph. But, this technique involves the risk of using a **sparse** Adjacency Matrix and therefore is **not optimal in terms of memory usage**. An additional danger of using Adjacency Matrices to encode the Graph connectivity as input to a Neural Network is that they are not **permutation invariant**. In other words, there may be multiple Adjacency Matrices that encode the same Graph connectivity, but do not produce the same output as input to the same Neural Network.

Secondly, traditional deep learning has a core assumption that the **instances within a dataset are independent** of each other. But, this is not the case in Graphs. When performing Machine Learning Tasks on the nodes of a Graph, each node depends on its neighborhood. Therefore, assuming this

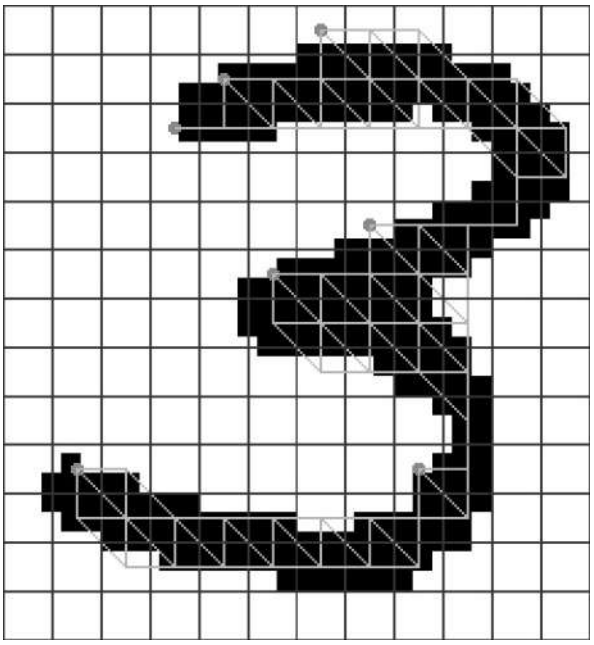

Figure 3.10: Visual Representation of an Image as a Regular Graph

independence can lead to multiple problems, such as **information leakage** between the train and the test set during the training of a Deep Learning Model on a Graph. It also complicates well-known techniques during the training of a model, such as **mini-batch training**, where the split of the mini-batches must be done based on the neighborhood structure of each node.

3.2.3 Tasks Taxonomy

Considering the complexity of the information that a Graph encodes, it is natural that more than one general type of prediction task on Graphs exists. We can differentiate these tasks by taking into account the entity that they aim to predict, and by considering the usage of labeled/unlabeled data.

3.2.3.1 Graph, Node, and Edge Level Tasks

As seen in figure 3.11, a way to differentiate these tasks is based on the entity that they aim to predict. In this section, the basic principles of these types of tasks will be covered, and some popular examples will be provided for each category.

The first type of Machine Learning tasks on Graphs are the **graph-level** tasks. As stated by Sanchez-Lengeling et al. (2021), in these tasks the goal is to predict a property for the entire graph. **Graph classification**, **regression**, and **matching** tasks, which require graph-level representation to be modeled, belong to the category of graph-level tasks. For example, the aim of Graph classification is to predict a value representing the whole Graph and use this information to assign the Graph to a certain class. Graph-level tasks can be seen as analogous to image classification in **Computer Vision**, or sentiment analysis in **Natural Language Processing**, where the goal is to extract information for the entire entity (image/sentence) provided. A popular example of a graph-level task is **Drug Discovery**, where atoms are represented as nodes and chemical bonds between them as edges. The

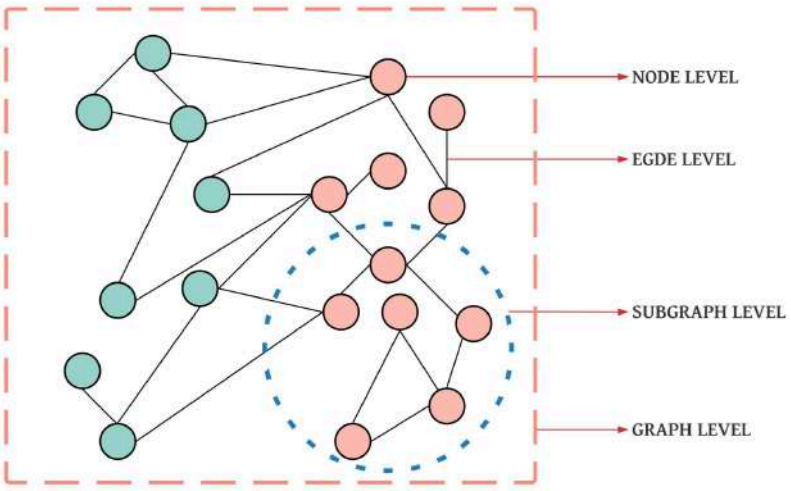

Figure 3.11: Visual Differentiation of Graph Tasks
Waikhom and Patgiri (2021)

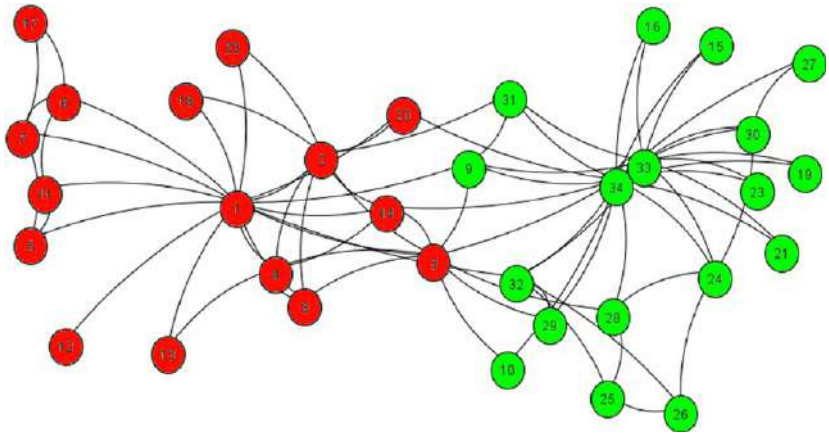

Figure 3.12: Communities in Zachary's Karate Club
Silva and Zhao (2012)

graph-level tasks can also be of subgraph level. An example of a subgraph-level task, is traffic prediction, by considering road network as a graph.

Node-level tasks are mainly concerned with predicting the identity or role of each node within the graph. As per Zhou et al. (2018), **node-level** tasks include node classification, node regression, and node clustering. Node classification tries to categorize nodes into several classes, and node regression predicts a continuous value for each node. Node clustering aims to partition the nodes into several disjoint groups, where similar nodes should be in the same group. A classic example of a node-level task is Zachary's karate club Zachary (1977). Zachary's karate club is a social network of a US university karate club in the 1970s. The network captures 34 members of the karate club, representing the interactions of the members outside of the club as links. The network is used for the **community detection** task, where the goal is to assign a label to each node, indicating the community to which the node belongs.

The goal of **edge-level** tasks is to predict information about the edges of a Graph. Such tasks include **link prediction**, **edge classification**. The aim of the link prediction task is to predict, given a pair of nodes of the Graph, whether an edge between these two nodes exists. A more advanced version of link prediction is the **link weight prediction** task, which is applied to weighted graphs, and aims in predicting not just the existence of a link between a pair of nodes, but also in predicting the weight of that link. In many cases, this prediction is far more informative than the simple link prediction. For example, as stated by Hou and Holder (2017), when describing the connection of two users in a social network, the description "Alice texts Bob 128 times per day" is more informative than "Alice likes Bob". Link weight prediction can also seem useful in recommender systems in general, as for example in the Movie recommender system of figure 3.6.

3.2.3.2 Supervised, Semi-supervised, and Unsupervised Tasks

In traditional Machine Learning Applications, learning can be categorized into **supervised**, **semi-supervised**, and **unsupervised**. In supervised learning, labeled datasets are used, meaning that there is prior knowledge of what the model predictions should be. In contrast, in unsupervised learning, datasets are unlabeled, and the task is to understand the structure of the provided data points. Semi-supervised learning is another categorization of learning, that combines the previous categories. In semi-supervised learning, there exists a small amount of labeled data and a large amount of unlabeled data. Some common subcategories of supervised learning are **classification** and **regression** tasks, whereas a common subcategory of unsupervised task is **clustering**.

However, this categorization is neither so informative nor so straightforward in Machine Learning Tasks on Graphs. An example of supervised learning on Graphs is a node-classification task, where all nodes are already assigned to labels. An example of unsupervised learning on Graphs is the community detection problem.

3.2.4 Traditional Approaches

Before the analysis of Modern Deep Learning on Graphs, it is necessary to introduce the traditional approaches that were used to implement Machine Learning Tasks on Graphs. The techniques that will be covered in this section, will introduce some key concepts for understanding modern approaches.

The traditional pipeline used for Machine Learning on Graphs was to extract some **hand-crafted features** from the Graphs and use these features as input to a traditional machine learning classifier. These features could focus either on the whole Graph or on certain parts, such as nodes or edges. A disadvantage of this approach is that hand-crafted features are usually **task-specific**, and are therefore not able to generalize across different types of prediction tasks.

3.2.4.1 Node-Level Features

This section will cover the most common node-level features that were used in traditional Machine Learning on Graphs. The purpose of their usage is to capture the structure and the position of the nodes in the network.

Node degree is the most straightforward node feature to examine. Counting the number of incident edges to the node is one of the most informative node features when performing Machine

Learning Tasks on a Graph. In cases of directed graphs, there is a differentiation between the **in-degree** and the **out-degree** of a node. The in-degree of a node is equal to the number of the incident edges, with direction towards the nodes, whereas the out-degree of a node is the number of the incident to the node edges, with direction away from the node.

Node centrality is an additional group of node features that have been widely used in traditional node-level tasks. These features aim to quantify the **importance** of a particular node within a network. Node degree is not a sufficient metric for capturing this importance, as can be seen in figure 3.13. In this example Graph, node A has a small degree but has a crucial role in the connectivity of the network. There is a variety of centrality metrics, that aim to capture this kind of importance.

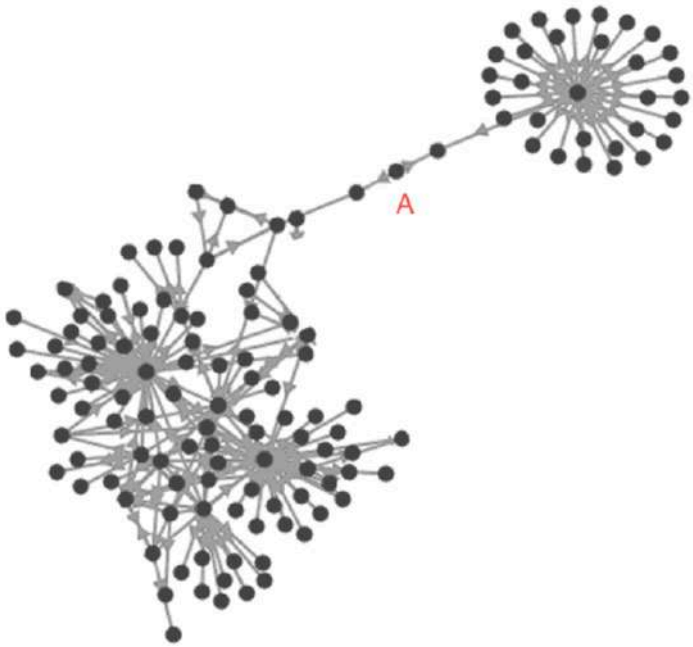

Figure 3.13: Inability of node degree to fully capture the importance of node (A) on the graph Golbeck (2013)

Eigenvector centrality is a node centrality measure, that calculates a node's importance for the network, considering the importance of its neighbors. For example, as stated by Golbeck (2013), a node with 300 relatively unpopular friends on Facebook would have lower eigenvector centrality than some with 300 very popular friends. The eigenvector centrality of node v, denoted by e_v, is computed by the recursive formula:

$$e_v = \frac{1}{\lambda} \sum_{u \epsilon N(v)} e_u$$

Closeness centrality indicates how close a node is to all the other nodes in the network (Golbeck (2013)). Lower values of closeness centrality, express that the node's average distance to all the other nodes is lower, and therefore the node is more influential to the network. This calculation can be represented as:

$$c_v = \frac{1}{\sum_{u \neq v} \text{shortest path length between u and v}}$$

Betweenness centrality measures how important a node is to the shortest paths through the network (Golbeck (2013)). As it can be seen in figure 3.16, the more the shortest paths between all pairs of nodes that contain the node, the greater the betweenness centrality of this node. The betweenness centrality of node v is denoted by c_v, and is computed by the formula:

$$c_v = \sum_{s \neq v \neq t} \frac{\text{\#(shortest paths between s and t that contain v)}}{\text{\#(shortest paths between v and t)}}$$

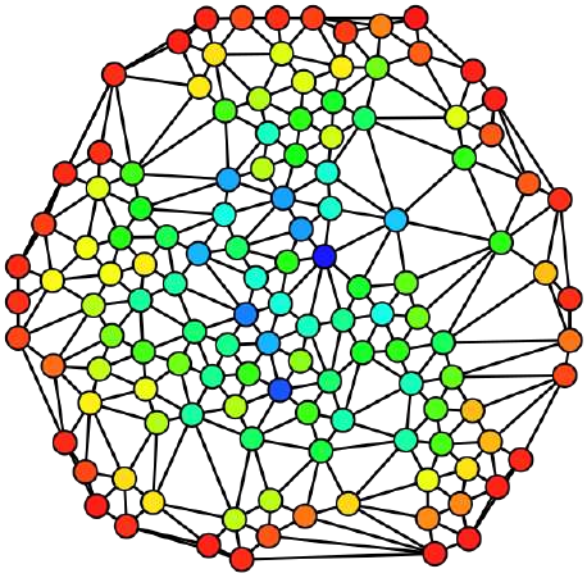

Figure 3.14: An undirected graph colored based on the betweenness centrality of each vertex from least (red) to greatest (blue).
Rocchini

Clustering coefficient measures the proportion of closed triangles in a node's local neighborhood (Hamilton (2020)). It measures how tightly clustered a node's neighborhood is. It is computed by the formula:

$$e_v = \frac{\text{\#(edges among neighboring nodes)}}{\text{\#(node pairs among neighboring nodes)}}$$

The general concept behind **clustering coefficient** can be generalized to counting other types of pre-specified subgraphs (called **graphlets**), instead of closed triangles, in the local neighborhood of the node.

As far as embedding nodes is concerned, **graphlets** are small, induced, rooted, connected, and not isomorphic to each other subgraphs. Some graphlets with 3-5 nodes are visualized in figure 3.15. By counting the occurrences of each graphlet in the local neighborhood of a node, a new node embedding, called **Graphlet Degree Vector**, can be generated. An analogy between Node Degree, Clustering Coefficient, and Graphlet Degree Vector can be found in table 3.1.

The concept of representing whole graphs, based on the graphlets that they contain, will be covered in subsection 3.2.4.2.

Table 3.1: Degree, Clustering Coefficient, Graphlet Degree Vector Analogy

Metric of node u	Target counted in neighborhood of node u
Degree	nodes
Clustering Coefficient	closed triangles
Graphlet Degree Vector (GDV)	graphlets

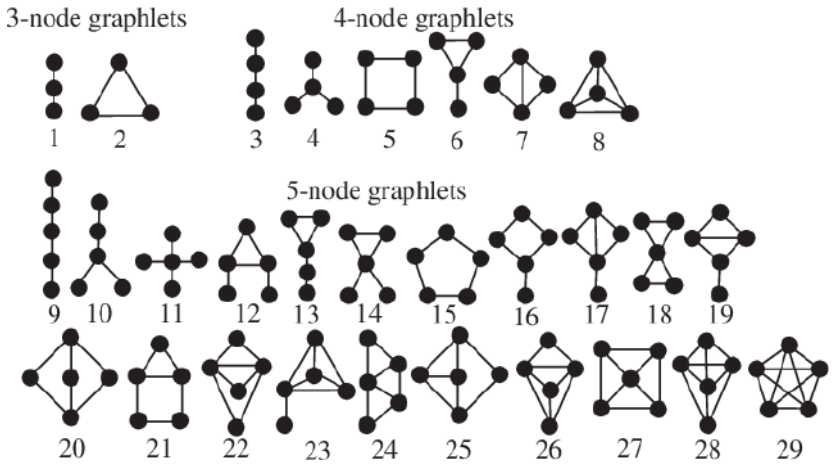

Figure 3.15: Some graphlets with 3-5 nodes
Przulj et al. (2004)

3.2.4.2 Link-Level Features

The previously defined features are useful for multiple classification tasks, but they do not quantify the relationships between nodes. Link-level features aim to achieve this quantification and use this information to predict the existence or features of edges. They can be categorized into the **distance-based** features, those that aim to capture the **Local Neighborhood Overlap**, and those that aim to capture the **Global Neighborhood Overlap**.

Shortest-path length between two nodes is the most straightforward **distance-based** feature, that can be used to predict the link-existence between a pair of nodes. In other words, the closer two nodes are placed in the graph, the more possible that a link between these two nodes exists. However, this feature has the disadvantage that it does not capture the overlap of the node's neighborhoods.

Counting the **Common Neighbors** is the simplest way to measure the **Local Neighborhood Overlap** between a pair of nodes. This metric can be formalized by the formula (Hamilton (2020)):

$$S[u, v] = |N(u) \cap N(v)|$$

Jaccard's coefficient is an extension to the technique of just counting the common neighbors of two nodes, that aims to quantify the **Local Neighborhood Overlap** between two nodes. Its aim is to normalize any biases in the result because of the node's degrees, by dividing the number of common neighbors by the size of the union of the node's neighborhoods. It is computed by the

formula (Hamilton (2020)):

$$S_{Jaccard}[u, v] = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

In addition to these types of measures, that just quantify the **Local Neighborhood Overlap**, there are measures that aim to take into account the importance of the common neighbors. For example, the **Adamic-Adar index** takes into account the degree of each one of the common neighbors. It considers more important the common neighbors with low degrees. The intuition behind this is that the low-degree common neighbors are more informative about the relationship between the target nodes than the high-degree common neighbors. To achieve that, it uses the inverse logarithm of the neighbor's degrees, and is computed based on the formula (Hamilton (2020)):

$$S_{AA}[u1, u2] = \sum_{v \epsilon N(u1) \cap N(u2)} \frac{1}{\log d_u}$$

All these measures of **Local Neighborhood Overlap** have a common disadvantage. They only consider the local neighborhood of the graph, and as a result, they do not correlate a pair of nodes without common neighbors. In a real-world network, it is possible that two nodes, without any common neighbors, belong to the same community, and therefore a link can be formed between them.

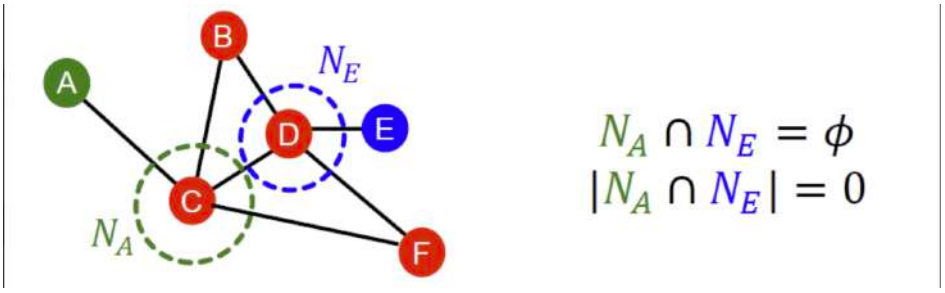

Figure 3.16: Nodes A and E neighborhoods are not overlapping, but a link could be formed between these nodes in the future.
Stanford (2021)

This limitation can be resolved by **Global Neighborhood Overlap** metrics, that take into account the entire graph. **Katz index** belongs to the category of **Global Neighborhood Overlap** metrics. The Katz index between a pair of nodes is computed by counting the walks of all lengths between these nodes. It is computed by the formula (Hamilton (2020)):

$$S_{Katz}[u, v] = \sum_{i=1}^{\infty} \beta^i A^i[u, v]$$

where the index i denotes the length of the path between nodes u and v, and b is a discount factor, that regulates the influence of larger paths on the final result.

3.2.4.3 Graph-Level Features

Graph-Level features are essential in graph-level tasks, such as Graph Classification. These features aim to extract global information and characterize the structure of the whole graph.

Kernel functions are generally used in various Machine Learning Tasks. A brief introduction to kernel methods can be found at Nikolentzos et al. (2021).

Definition 3.2.1 (Kernel function): *Nikolentzos et al. (2021) Given a non-empty set X, we say that a function $k : X \times X \to R$ is a kernel if there exists a Hilbert space H and some map $\phi : X \to H$ that satisfies:*

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_H \forall x, x' \epsilon X$$

Kernel functions can be also used for graphs. In this case, they are called graph kernels. A **graph kernel** is a kernel function that computes the similarity of two graphs via their inner product.

Multiple graph kernels are based on counting the occurrences of defined substructures in the graph. A widely used technique to generate graph kernel functions is by aggregating node-level features. The most straightforward concept that falls under this category is known as **Bag of nodes**. For example, a graph kernel can be generated if the graph is seen as a **bag of node degrees**, or as **bag of node centralities**. A visualization of this approach can be seen in figure 3.17. An important weakness of this approach is that the result is produced entirely by node-level features, and therefore some information that can be extracted only by the global structure of the graph will be ignored.

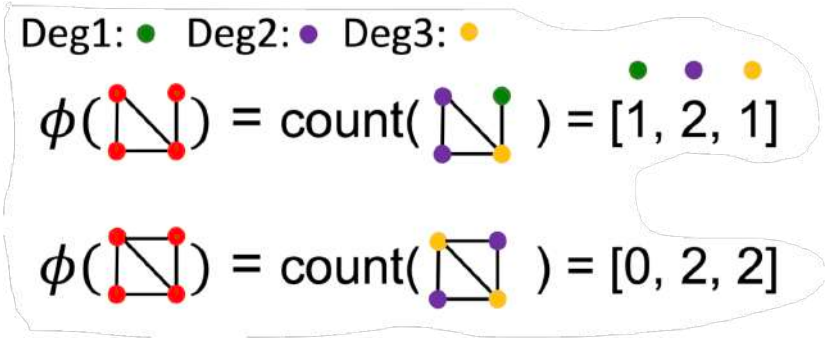

Figure 3.17: Graphs as Bag of Node Degrees
Stanford (2021)

Another concept that has been widely used in traditional machine learning approaches on graphs, and has already been reported in section 3.2.4.1, is the concept of **Graphlets**. An important difference between using graphlets for node-level and for graph-level tasks is that, for graph-level tasks, we consider that graphlets are not rooted, and are not necessarily connected. Following the bag of nodes approach, we can define another graph kernel, known as **graphlet kernel**, by counting how many times each graphlet is contained in a graph. That way, the graph can be represented as **bag of graphlets**. However, this approach has severe scalability limitations, as counting graphlets of size k for a graph of size n by enumeration takes $O(n^k)$.

To define a more efficient graph kernel, the color refinement variant of the Weisfeiler-Lehman isomorphism test (3.1.3.4) has been used. This kernel, known as **Weisfeiler-Lehman Kernel**, is based

on the concept that after K steps of the algorithm, the label of node v, denoted by $l_{G^{(K)}}(v)$, summarizes the structure of its K[th] neighborhood. Therefore, after K steps of the color refinement method, each node is assigned to a label describing its neighborhood, and as a result, each graph can be represented as a **bag of labels**. Then, the WL kernel is computed by the inner product of the label count vectors.

3.2.5 Node Embeddings

This subsection is an overview of the basic concepts around node embeddings, and a high-level report of the main approaches and techniques that have been developed for their calculation.

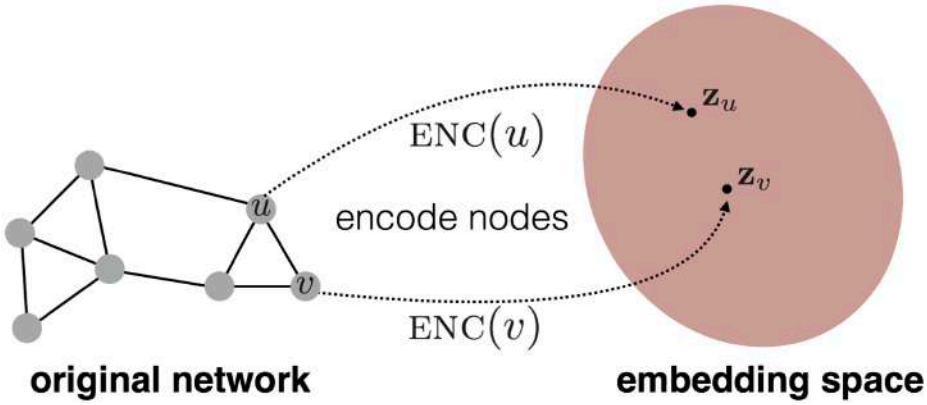

Figure 3.18: Projection of nodes into the embedding space
Stanford (2021)

3.2.5.1 Main concepts

Node embeddings are low-dimensional, real-valued vectors, that summarize the position of the node in the Graph and capture the structure of the node's local neighborhood. The aim of the projection of figure 3.20 is to find an embedding space, where the geometric relations of the embeddings z_u and z_v, correspond to the relations of their nodes u and v in the original graph.

Embeddings have been used in various domains of machine learning in general, such as in **Natural Language Processing**. In this domain, a **word embedding** is a representation of a word as a low-dimensional vector, that captures the meaning of a word, in such a way that words with similar meanings are expected to have embeddings with a "small" distance in the embeddings space. As a result, the mapping of the words to an **embedding space** can be used as an intermediate step in multiple NLP tasks.

In the same way, nodes of a Graph that are in a similar position, or belong to the same neighborhood, are expected to have similar node embeddings. This type of information can be utilized in various tasks, such as in **Link Prediction**, or **Community Detection**. A visualization of the relationship between detected communities and computed node embeddings in Zachary's Karate Club dataset can be seen in figure 3.19.

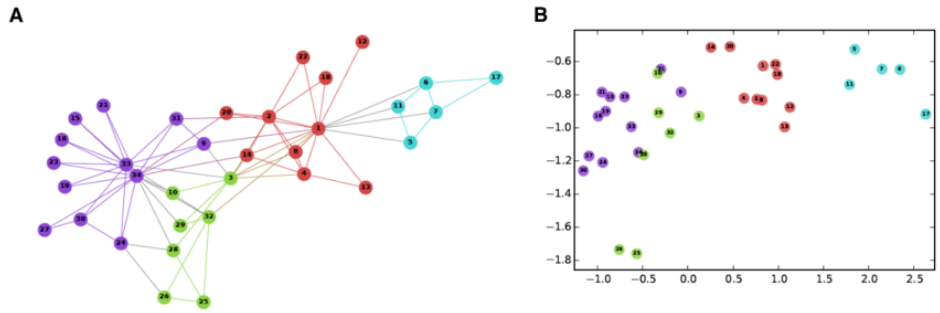

Figure 3.19: Communities and node embeddings in Zachary's Karate Club Graph
Perozzi et al. (2014)

3.2.5.2 The Encoder-Decoder Framework

A well-known approach used for node embeddings is the **Encoder-Decoder** framework. As per Hamilton et al. (2017b), the framework consists of two basic mapping functions, an **encoder** and a **decoder**.

The encoder is responsible for the mapping of the graph nodes to low-dimensional embeddings. Formally, the encoder has the signature (Hamilton (2020)):

$$ENC : V \rightarrow R^d$$

Encoders are commonly based on the **shallow embedding** approach. In this approach, the encoder is just an embedding lookup in the embedding matrix, that contains a column for the embedding vector of each node of the graph. In this case, we can define the decoder mapping as:

$$ENC(u) = Z[u], \text{ where Z is the embedding matrix}$$

There have been developed multiple methods for generating the embedding matrix. Some of them, such as the ones used in Node2Vec (3.2.5.5) and Fast Random Projection (3.2.5.6), will be seen in the following chapters.

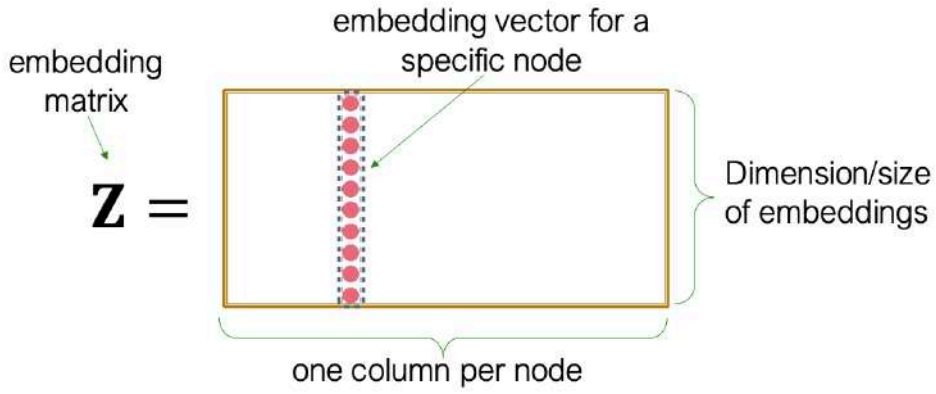

Figure 3.20: Shallow encoding for the generation of node embeddings
Stanford (2021)

The decoder is responsible for the reverse mapping, i.e. obtaining structural information about the graph by the embeddings. In general, multiple decoders are possible, but the usual practice is a decoder that maps a pair of node embeddings (**pairwise decoder**) to a real value that represents the similarity of these nodes in the original graph. The signature of the decoder is the following (Hamilton (2020)):

$$DEC : R^d \times R^d \rightarrow R^+$$

The output of a pairwise decoder is a *reconstruction* of the similarity between a pair of nodes. In order to evaluate the quality of the encoder and the decoder mappings, there must be a function that maps a pair of nodes to a real-valued similarity measure. This **pairwise similarity function** is denoted by $s_G(v_i, v_j)$, and is a user-defined, graph-based similarity measure between nodes, defined over the graph G Hamilton et al. (2017b). It has the following signature:

$$s_G : V \times V \rightarrow R^+$$

The evaluation of the decoder and encoder mappings is based on a loss function that defines the comparison between the **pairwise decoder** output and the **pairwise similarity function** output for the same pair of nodes and their embeddings. The goal is to build the encoder and decoder functions in a way that this loss is minimized, and therefore the following relationship stands Hamilton et al. (2017b):

$$DEC(ENC(v_i, v_j)) = DEC(z_i, z_j) \approx s_G(v_i, v_j)$$

The **shallow embedding** approaches are mainly based on matrix factorization techniques, and on random walks. The following subsections present an overview of these two methods.

3.2.5.3 Factorization-Based Approaches

The methods in this section are referred to as matrix-factorization techniques because they define the loss function based on the form:

$$L \approx ||Z^T Z - S||_2^2$$

where S is the pairwise similarities matrix, and Z is the node embeddings matrix.

They can be divided into two sub-categories, the **Laplacian eigenmaps** techniques, and the **Inner product** methods. Table 3.2 shows the general formulas for the decoder mapping and the loss function in these techniques.

Table 3.2: Decoder mapping and loss function in Laplacian-Eigenmaps and Inner-product approaches of the encoder-decoder framework

$DEC(z_i, z_j)$	$Loss function$
$\|\|z_i - z_j\|\|_2^2$	$\sum_{(v_i,v_j)\epsilon D} DEC(z_i, z_j) \cdot s_G(v_i, v_j)$
$z_i^T z_j$	$\sum_{(u_i,u_j)\epsilon D} \|\|DEC(z_i, z_j) - s_G(u_i, u_j)\|\|_2^2$

3.2.5.4 Random Walks

In recent years, there has been a shift towards using **stochastic methods** to capture a node's local neighborhood. Following this direction, a method that has been widely used to map nodes to an embedding space is the method of random walks.

Given a graph and a starting point, we select a neighbor of it at random and move to this neighbor; then we select a neighbor of this point at random and move to it, etc. This (random) sequence of points selected this way is a **random walk** on the graph. Lovász (1996)

More formally, as per Grover and Leskovec (2016), the i_{th} node of a random walk of fixed length l , given a source node $c_0 = u$ is generated by the distribution:

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & (v, x) \in E \\ 0 & otherwise \end{cases}$$

where π_{vx} is the unnormalized transition probability between nodes v and x, and Z is the normalizing constant.

The main concept of encoding nodes using random walks is that a pair of nodes should have similar embeddings if these nodes tend to co-occur on short-length random walks over the graph. The concept of using random walks to project nodes to the embedding space is beneficial in terms of scalability. Methods that utilize random walks are especially useful when one can either only partially observe the graph or the graph is too large to measure in its entirety Goyal and Ferrara (2018).

A widely-used example of a real-world algorithm that utilizes the concept of random walks, is the **node2vec** algorithm.

3.2.5.5 Node2vec

Node2vec Grover and Leskovec (2016) is a semi-supervised algorithm for scalable feature learning in networks. Intuitively, node2vec performs some biased random walks and returns feature representations that maximize the likelihood of preserving network neighborhoods in a d-dimensional feature space. For the purpose of this thesis, this subsection will focus on the **search strategy** of node2vec. A detailed description of the algorithm and information about its time and memory complexity can be found in the original paper (Grover and Leskovec (2016)).

The goal of a search strategy is, given a source node u, to sample its neighborhood $N_S(u)$. Two classic, but extreme in terms of search behavior, strategies are the **Breadth First Search** and the **Depth First Search** algorithms. Intuitively, the **Breadth First Search** algorithm focuses on the local neighborhood of the source node, exploring at first the immediate neighbors of each node. In contrast, the **Depth First Search algorithm** focuses on exploring nodes as far as possible from the root node along each branch, before backtracking. Figure 3.21 displays a visualization of the nodes that each algorithm has visited, starting from node u, and after making three steps.

In general, prediction tasks on nodes usually capture two types of similarities: **homophily** and **structural equivalence**. According to the homophily hypothesis, nodes that are highly connected, are similar to each other, and should therefore have close embeddings. According to the structural equivalence hypothesis, nodes that have a similar role in the structure of the network, are alike

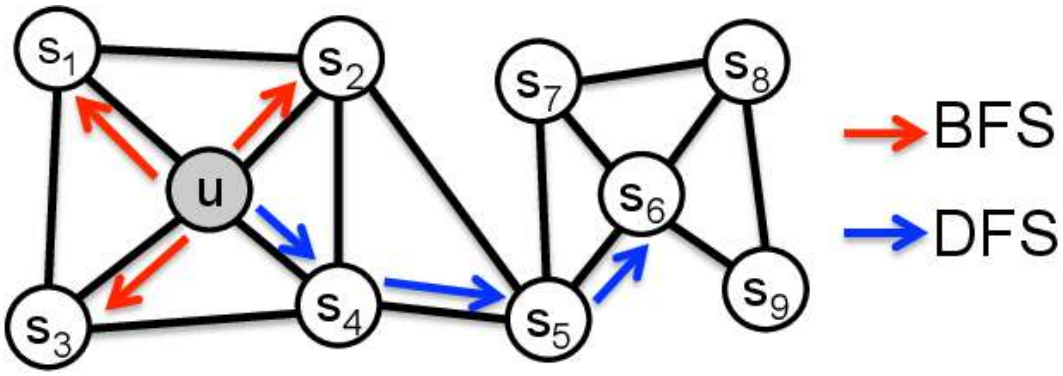

Figure 3.21: BFS and DFS strategies
Grover and Leskovec (2016)

to each other and, regardless of their connectivity, should have close embeddings. In real-world networks, node similarity is a mixture of these two criteria.

One could say that the **BFS** algorithm focuses on identifying the **structural equivalence** of two nodes, after focusing on the structure of their local neighborhood. In contrast, the **DFS** algorithm focuses on identifying the **homophily** of two nodes, after exploring the node's connections.

Node2vec uses two tunable parameters, the **return parameter** p and the **in-out parameter** q, that give control over the behavior of the biased walks, and therefore over the search space. By tuning these parameters, the algorithm can achieve a mixture of homophily and structural similarity to generate expressive node embeddings.

In short, node2vec defines a 2^{nd} order random walk, where the transition probability π_{vx} is computed as follows Grover and Leskovec (2016): Considering the latest traverse to be from node t to node v, the transition probabilities π_{vx} for each neighbor x of node v are: $\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx}$, where:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & d_{tx} = 0 \\ 1 & d_{tx} = 1 \\ \frac{1}{q} & d_{tx} = 2 \end{cases}$$

and d_{tx} denotes the shortest path between nodes t and x.

As can be inferred by this formula:

- The **return parameter p** controls the probability of revisiting the previous node in the random walk. When p has a low value, this probability is large, and it is therefore expected that the walk stays in the local neighborhood of node v.

- The **In-out parameter q** controls the *inward* or *outward* direction of the walks. If q<1, the probability of visiting nodes with $d_{tx} = 2$ is large, and therefore the walk tends to visit nodes that are further from node t, reminding of the DFS algorithm.

3.2.5.6 Fast Random Projection (FastRP)

FastRP (Chen et al. (2019)) is a scalable and performant algorithm for learning distributed node representations in a graph. According to the original paper, FastRP is over 4000 times faster than

state-of-the-art methods such as node2vec (Grover and Leskovec (2016)) and DeepWalk (Perozzi et al. (2014)), and it also achieves comparable performance on various tasks.

The FastRP algorithm perceives the generation of network embedding as a process of two main components. The first component is the construction of a **node similarity matrix**, and the second one is the **dimensionionality reduction** on this matrix to produce the node embeddings. The outline of the algorithm, as it can be found in the original paper, is presented in figure 3.22. The following sections contain the basic mathematical background and the intuition behind these processes, as they are presented in the original paper.

Algorithm 1 FastRP(A)

Input:
graph transition matrix $\mathbf{A}$, embedding dimensionality d, maximum power k, normalization strength β, weights $\alpha_1, \alpha_2, \ldots, \alpha_k$

Output: matrix of node representations $\mathbf{N} \in \mathbb{R}^{n \times d}$

1: Produce $\mathbf{R} \in \mathbb{R}^{n \times d}$ according to Eq. 6
2: $\mathbf{N}_1 \leftarrow \mathbf{A} \cdot \mathbf{L} \cdot \mathbf{R}$ where $\mathrm{L}_{ij} = \left(\frac{d_j}{2m}\right)^{\beta}$
3: **for** $i = 2$ to n **do**
4: $\quad \mathbf{N}_i \leftarrow \mathbf{A} \cdot \mathbf{N}_{i-1}$
5: **end for**
6: $\mathbf{N} = \alpha_1 \mathbf{N}_1 + \ldots + \alpha_k \mathbf{N}_k$
7: **return** N

Figure 3.22: Outline of the FastR algorithm
Chen et al. (2019)

Dimension Reduction in FastRP

As far as the **dimension reduction** is concerned, the FastRP algorithm utilizes the **very sparse random projection** algorithm (Li et al. (2006)), which is an extension of the **random projection** (Vempala) and **sparse random projection** (Achlioptas (2003)) algorithms.

In short, **Random projection** is a method to reduce the dimensionality of a set of points, while preserving the pairwise distances between them. The general concept is that, in order to reduce a $n \times m$ feature matrix M to a $n \times d$ matrix N, where $d << m$, we need the matrix multiplication:

$$N = M \cdot R$$

where R is random projection matrix, with all of its entries to be independent and identically distributed with zero mean value. In general, the random projection methods mainly differ in the way that they compute the R matrix. These differences are summarized in the following paragraphs.

In **Gaussian Random Projection**, the entries of R are sampled i.i.d from a Gaussian distribution: $R_{ij} \sim N(0, \frac{1}{d})$. R is a dense $m \times d$ matrix, and therefore the time complexity of Gaussian random projection is $O(n \cdot m \cdot d)$.

An improvement to the Gaussian random projection is **sparse random projection**. In this method, entries of R are sampled i.i.d from:

$$R_{ij} = \begin{cases} \sqrt{s} & \text{with probability} \frac{1}{2s} \\ 0 & \text{with probability} 1 - \frac{1}{s} \\ -\sqrt{s} & \text{with probability} \frac{1}{2s} \end{cases} \tag{3.1}$$

with $s = 3$. As a result, $1 - \frac{1}{s} = \frac{2}{3}$ of the R entries are zero, and this triples up the speed of the process.

Very sparse random projection is an extension to the **sparse random projection** method, that proposes the usage of $s = \sqrt{D}$ for the computation of the matrix R in the equation 3.1, in order to generate node embeddings of dimension D. This achieves $\sqrt{D}$ times speedup over the Gaussian random projection.

In the FastRP algorithm, **very sparse random projection** with $D = m$ is used. As the only computation needed for dimensionality reduction is matrix multiplication, accelerators such as GPUs are utilized better, and parallelization is achieved. In addition, because of the associative property of matrix multiplication, the computation of the random projection $N = A^k \cdot R$ of the node similarity matrix A^k, becomes even more efficient. As reported in the original paper, the time complexity for the random projection of each power of the matrix A is $O(m \cdot k \cdot d)$.

Node similarity matrix construction in FastRP

There are two main points about the existing methods, that are worth mentioning. Firstly, it is important to keep the **high-order proximity** in the input graph, which is usually achieved by the powers of the transition matrix (A^k). Secondly, existing methods usually skip the **element-wise normalization** on the similarity matrix, before the dimension reduction, for scalability reasons.

But, in the original FastRP publication, it is shown that for a particular entry A^k_{ij} of the node similarity matrix, we have that

$$A^k_{ij} \to \frac{d_j}{2m} \text{ when } k \to \infty \tag{3.2}$$

In addition, the majority of the real-world graphs are scale-free, and the degree of the scale-free networks follows a heavy-tailed power-law distribution. So, it follows that the entries in A^k have a **heavy-tailed** distribution, which causes problems with dimensionality reduction methods. For this purpose, the FastRP algorithm uses a scaled version of the Tukey transformation (Tukey (1957)) to normalize the similarity matrix in an effective way. Each feature y is transformed into y^λ where λ controls the normalization strength. In the original publication, it is proved that, because of the equation 3.2, the normalized version of the node similarity matrix is computed as $(A^k_{ij})^\lambda$. Therefore, the time complexity for the computation of the similarity matrix is $O((n \cdot d)/s) = O(n \cdot \sqrt{d})$.

Iteration weights

In the algorithm presented in figure 3.22, one can notice the weights $\alpha_1, \alpha_2, ..., \alpha_k$, also referred to as iteration weights in bibliography. Intuitively, they are used to tune the impact of each node on the embeddings of the other nodes, based on their distance on the graph.

3.3 The Link Weight Prediction Task

The link weight prediction task is classified as an edge-level task of machine learning on graphs, as it has already been reported in 3.2.3. A visualization of the link prediction task can be seen in figure 3.23. Multiple approaches have been developed for this task, starting from some traditional machine learning techniques, later moving onto deep learning approaches, and finally utilizing Graph Neural Networks. In this chapter, the first two categories of approaches will be covered. The utilization of Graph Neural Networks for link weight prediction will be covered in section 3.7.

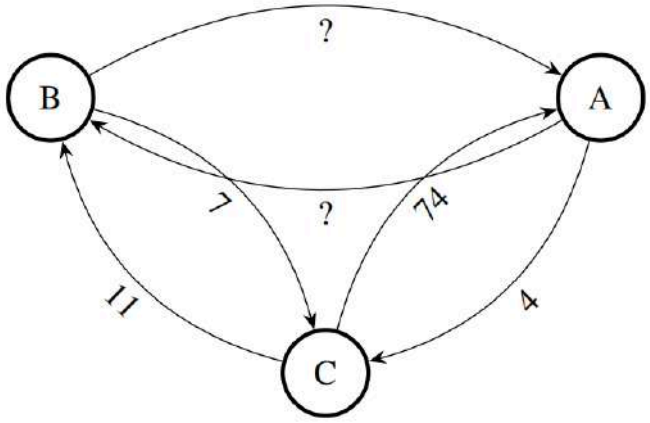

Figure 3.23: Visual representation of a graph with unknown weights on some edges. The aim of the link prediction task is to predict the weight on these edges.
Hou and Holder (2017)

3.3.1 Problem Definition

In Hou and Holder (2017), the link weight prediction problem is defined as follows, for a weighted directed graph:

Definition 3.3.1 (Link Weight Prediction): *Given a weighted directed graph with the node set V and link subset E, the aim of the link prediction task is to build a model that can predict the weight $w = f(x, y)$, for any link $(x, y) \epsilon E$.*

3.3.2 Motivation

The main motivation behind the link weight prediction problem is that in a real-world system, having some kind of knowledge on the interaction between two nodes, can be far more informative than just knowing that the interaction exists. For example, in a movie recommender system, where the ratings of users on movies are modeled as links of a graph, knowing the exact rating is more useful than just knowing that a user has rated, liked, or seen a movie.

3.3.3 Traditional Approaches

Multiple approaches to the link weight prediction task were developed before the usage of deep learning. An overview of these methods is reported in Hou and Holder (2017). Some of them, such as the **node similarity model**, depend on the link-level features of a graph (3.2.4.2) and other more complex methods, such as **Stochastic Block Model** (Holland et al. (1983)), only use link existence information.

In the **node similarity model**, the weight w_{xy} of a link (x, y) between nodes x and y is considered to be proportional to the similarity s_{xy} of these nodes and is computed by the linear regression model:

$$w_{xy} = k \cdot s_{xy} \tag{3.3}$$

where k is the regression coefficient. The similarity s_{xy} is based on the common neighbors of these nodes, and is calculated by the formula:

$$s_{xy} = \sum_{z \epsilon N(x) \cap N(y)} F \tag{3.4}$$

where F is an index factor that can have multiple forms and is calculated with respect to the common neighbors of the nodes or other metrics such as Adamic-Adar (3.2.4.2) and Resource Allocation.

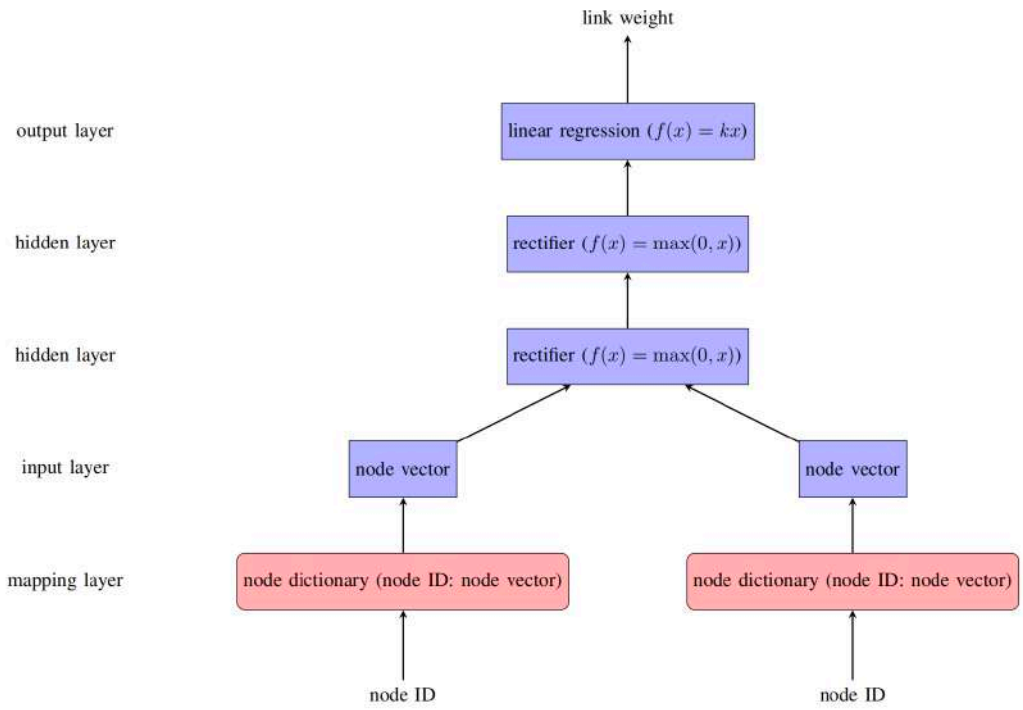

Figure 3.24: Visual representation of the architecture of Model R
Hou and Holder (2017)

3.3.4 Deep Learning Approaches

Deep learning has outperformed traditional machine learning approaches in multiple domains, such as Computer Vision and Natural Language Processing. As a result, in addition to the traditional approaches which were reported in the previous section, multiple deep-learning approaches were developed for the link weight prediction task.

One of the first deep learning approaches for link weight prediction was **Model R** (Hou and Holder (2017)). This model is a fully connected deep neural network and uses a supervised approach to predict link weights. As it can be seen in figure 3.24, the model maps each node to a node vector, and then applies multiple hidden layers of rectified linear units, and a linear regression on its output layer to generate the predicted link weight.

A following approach to the same task is **Model S**, proposed in Hou and Holder (2018). The key concept for the development of Model S was to decouple the process of Model R into two sub-processes, the generation of the node embeddings, and the following link weight prediction. The main benefit of this decoupling is that any node embedding technique can be used as an input to the model (such as node2vec 3.2.5.5), making it also compatible with any techniques that might be developed in the future. As stated in the original paper, Model S is a generalization of Model R, or vice-versa, Model R is a special case of Model S when the pair of node embeddings is produced by itself. A visualization of Model S can be seen in figure 3.25.

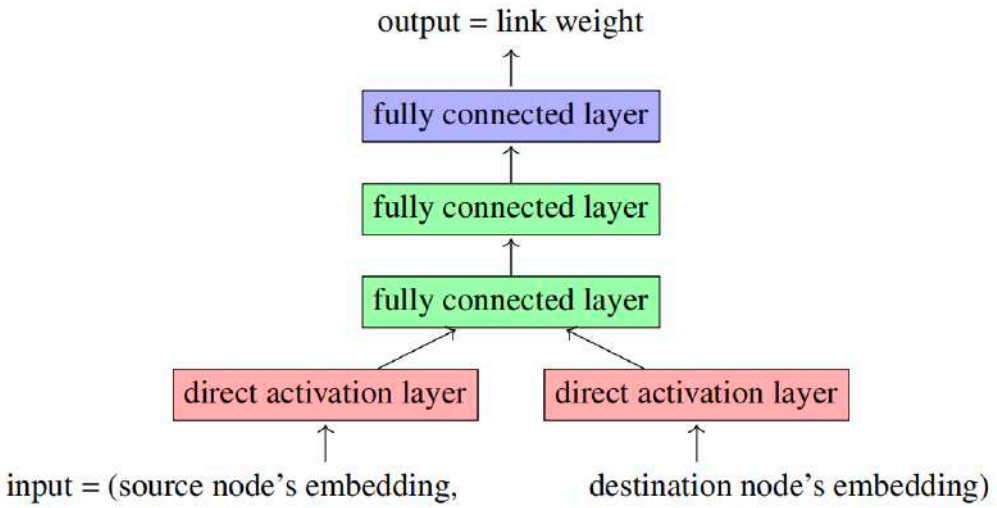

Figure 3.25: Visual representation of the architecture of Model S, with one input layer (red), two hidden layers (green), and one output layer (blue)
Hou and Holder (2018)

3.4 Graph Neural Networks

This section will present the basic concepts around Graph Neural Networks, also referred to as **GNNs**, starting with the motivation behind their recent growth. Subsequently, some more technical details about their functionality will be covered, as long as the most common architectural patterns

for developing a GNN. Closing this section, we will present some interesting insights and challenges that occur when training and testing a Graph Neural Network in a real-world task.

3.4.1 Motivation

In the previous sections, there was made an extensive coverage of the need to perform Machine Learning Tasks on Graphs, and multiple traditional approaches in this direction were reported. The main concepts that have been traditionally used for Graph Learning Representation are **node embeddings**, which used a shallow embedding approach to produce node representations. But, most of these embedding techniques, introduced some serious limitations. For example, they do not incorporate node features and capture only the structural information of the network. In addition, they are **transductive**, meaning that they cannot generate embeddings for nodes that have not been seen during the training phase.

For these reasons, **Graph Neural Networks** were developed. As per Hamilton (2020), the **GNN** formalism is a general framework for defining deep neural networks on graph data. GNNs are able to encode structural information of the node inside the network, and simultaneously take into account any existing **node features**.

3.4.2 Challenges

In general, the challenges around deep learning on graphs have already been covered in chapter 3.2.1. In summary, the main challenge is to define a new deep learning architecture, that extends existing widely-spread architectures of Deep Learning, such as **Convolutional Neural Networks** for Computer Vision and **Recurrent Neural Networks** for Natural Language Processing, and that can perform machine learning tasks on **non-euclidean data**.

3.4.3 Key concepts

This subsection will cover in a high-level the key concepts around the functionality of GNNs.

3.4.3.1 Permutation Invariance and Equivariance

The most straightforward way to handle a graph with a deep learning model would be to use as input to the model its adjacency matrix. The issue with this approach is that changing the order of the nodes in the adjacency matrix obviously produces a different adjacency matrix for the same graph. As a result, the output of the model for a graph would be dependent on the order of the nodes in the adjacency matrix, and the transformation would not be permutation invariant. Ideally, the model should learn a representation of the node features, that is **permutation invariant**, meaning that it should be independent of the order of the nodes.

A formal definition of the permutation invariance and permutation equivariance is provided in Meltzer et al. (2019).

Definition 3.4.1 (Permutation Invariance): *Let P_n be the set of all valid permutation matrices of order n, then a function f is invariant to:*

- *row permutation iff* $f(X) = f(P_\pi X), \forall X \epsilon R^{n \times m}, P_\pi \epsilon P_n$

- *column permutation iff* $f(X) = f(XP_\pi^T), \forall X \epsilon R^{m \times n}, P_\pi \epsilon P_n$

Definition 3.4.2 (Permutation Equivariance): *Let* P_n *be the set of all valid permutation matrices of order* n*, then a function* f *is equivariant to:*

- *row permutation iff* $P_\pi f(X) = f(P_\pi X), \forall X \epsilon R^{n \times m}, P_\pi \epsilon P_n$
- *column permutation iff* $f(X)P_\pi^T = f(XP_\pi^T), \forall X \epsilon R^{m \times n}, P_\pi \epsilon P_n$

Intuitively, as far as graphs are concerned, a permutation invariant function is a function that is independent of the order of the rows/columns in the adjacency matrix, and e permutation equivariant function is a function that its output is permuted in a consistent way when the adjacency matrix is permuted.

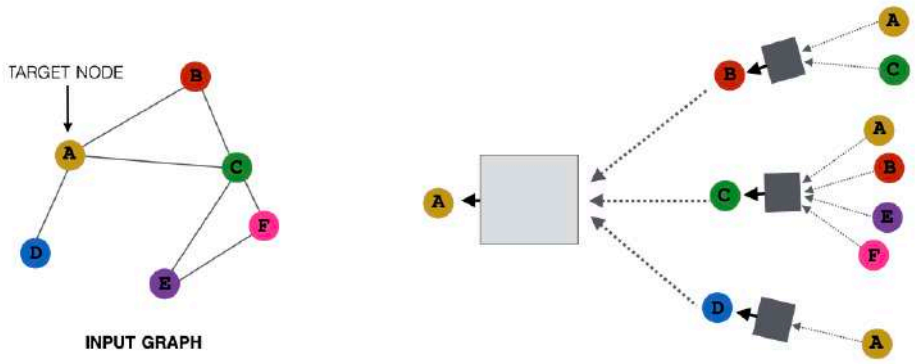

Figure 3.26: The Message Passing Framework, used for neighbors' aggregation. The embedding of node A is generally generated by aggregating the embeddings of its neighbor nodes. The message-passing process that is visualized here is a two-layer version, extending this aggregation to also reach the nodes with a distance of 2 from node A.
Stanford (2021)

3.4.3.2 The Message Passing Framework

As stated by Gilmer et al. (2017), this framework is an abstraction of some common points in the most promising existing models for graph-structured data. The key idea behind this framework is that the embedding of each node should be generated based on the embeddings and the structure of its local neighborhood. In this section, we present a high-level approach to the framework's functionality. In Gilmer et al. (2017), a more formal mathematical formulation can be found.

For simplicity, the basic concepts of the framework will be presented on undirected graphs, denoted by G, with node features denoted by x_v and edge features denoted by e_{vw}.

The forward pass of the training has two phases, the **message-passing** phase, and the **readout** phase. As it can be seen in figure 3.26, each node's A embedding is affected by the embeddings of its neighborhood (denoted by N_A). The message-passing phase runs for T time steps. At each time step, each node v has a hidden state vector h_v. During the time step t, the hidden state h_v^t of each node v is updated based on the messages m_v^{t+1} that the node receives from its neighborhood.

The messages are aggregated by an **AGGREGATE** function, and then an **UPDATE** function is applied to this aggregated vector, producing the hidden state h_v^{t+1}. The message-passing phase can be mathematically formulated as follows, as stated by Hamilton (2020):

$$m_{N(v)}^{(k)} = AGGREGATE^{(k)}(\{h_u^{(k)}, \forall u \epsilon N(v)\}) \tag{3.5}$$

$$h_v^{k+1} = UPDATE^{(k)}(h_v^{(k)}, m_{N(v)}^{(k)}) \tag{3.6}$$

Note that in equation 3.5 the aggregation function takes as input a set of the neighbors' hidden states, and therefore GNNs following the message-passing framework are permutation equivariant by design. After running K steps of the message-passing phase, we can use the hidden state $h_v^{(K)}$ of each node v as the node's embedding.

In the readout phase, there is computed a feature $\hat{y}$ for the whole graph. This feature vector is extracted via a readout function R, according to the formulation:

$$\hat{y} = R(\{h_v^{(k)} | v \epsilon G\}) \tag{3.7}$$

The functions **AGGREGATE**, **UPDATE**, and **R** are learned differentiable functions. That means that, for example, a whole neural network can be used to perform the aggregation. Various GNN types, that utilize these functions in different ways, will be covered in the following sections.

An important property of the message-passing framework is that it requires that all the nodes v have some initial hidden state $h_v^{(0)}$. The hidden state of the nodes can be initialized either with structural properties, such as the degree or the centrality of each node, or, in real-world networks, with information related to the entities that the nodes represent, about the nodes, as for example demographic information about a user in a social media network.

In the classic approach, the embedding aggregation takes place over the immediate neighbors of each node. An interesting note is that then, the node's v hidden state $h_v^{(k)}$, at each step k, is generated based on the embeddings of the k^{th} neighborhood of the node. This behavior of aggregating the local neighborhood features is similar to the convolutional kernels used in **Convolutional Neural Networks** for Computer Vision.

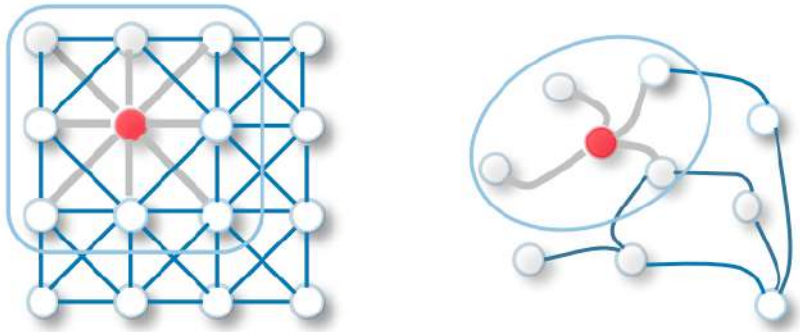

Figure 3.27: Visual Comparison between 2D convolution on an image and Graph Convolution Wu et al. (2021)

3.4.4 Taxononomy

This section will focus on the Taxonomy of the GNNs, as it presented by Wu et al. (2021) and Zhou et al. (2022). This taxonomy is based on multiple criteria, such as architecture, training type, and task type.

3.4.4.1 Architecture

- **Recurrent Graph Neural Networks (RecGNNs)**

  RecGNNs (Huang and Carley (2019)) generalize classic RNNs in order to process graph-structured data and learn node representations. They introduced the idea of message-passing, which was later on inherited by Graph Convolutional Neural Networks. They assume a constant message-passing process between a node and its neighbors until a stable equilibrium is reached. A commonly used variant of Recurrent Graph Neural Networks is GraphLSTM (Liang et al. (2016)), which generalizes the vanilla LSTM that is being used for sequential data, to perform on graph data. A high-level visualization of the RecGNNs functionality can be seen in figure 3.28.

- **Convolutional graph neural networks (ConvGNNs)**

  ConvGNNs (Kipf and Welling (2016a)) generalize classic CNNs (O'Shea and Nash (2015)) to process graph data, instead of images. Following the concept of convolution kernels in CNNs, ConvGNNs generate each node's embedding by aggregating the embeddings of this node's neighbors. This process can be seen in figure 3.27. As will be seen in the following chapters, ConvGNNs can be further distinguished into spatial and spectral.

- **Graph autoencoders (GAEs)**

  GAEs (Ng et al. (2019)) are unsupervised learning frameworks that encode graphs or nodes into a latent vector space and then reconstruct them from the encoded data.

- **Spatial-temporal graph neural networks (STGNNs)**

  are GNN architectures that operate on graphs that evolve over time, also known as spatial-temporal graphs. These types of GNNS consider spatial and temporal dependency at the same time, exploring tasks that have been increasingly popular in a variety of applications.

3.4.4.2 Task

Another way to categorize Graph Neural Networks is by the task type that they aim to solve. This taxonomy of machine learning tasks on graphs has already been analyzed in section 3.2.3.

3.4.4.3 Training

Graph Neural Networks can also be categorized based on the training frameworks used. That is, GNNs can be categorized into those that perform supervised, unsupervised and semi-supervised tasks. The differentiation between these types of tasks, as long as some examples, have already been reported in section 3.2.3.1.

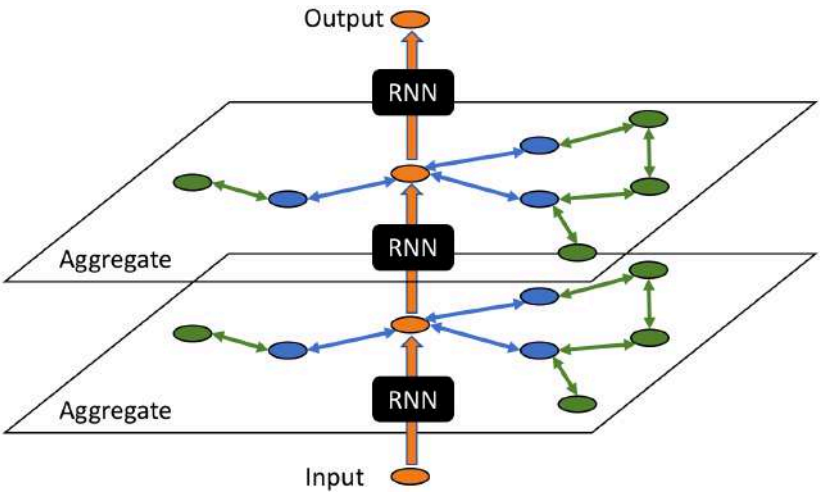

Figure 3.28: Visual Representation of a Recurrent GNN
Huang and Carley (2019)

3.5 Some Graph Convolutional Neural Networks

In this section, we will cover the theoretical background behind the Graph Convolutional Neural Networks that were used in our experiments.

3.5.1 GraphSAGE

GraphSAGE (Hamilton et al. (2017a)) stands for SAmple and aggreGatE and is a general framework for inductive node embedding. Traditional methods for generating node embeddings, usually depend on matrix factorization, and therefore cannot generalize to unseen nodes. GraphSAGE is inductive, meaning that it can generalize to unseen nodes, and even to unseen graphs. As it is reported in the original paper, *the key concept behind GraphSAGE is that instead of training a separate embedding vector for each node, it trains a set of aggregator functions that learn to aggregate information about a node's local neighborhood.* A visualization of the forward pass of GraphSAGE can be seen in figure 3.29.

The detailed algorithm of GraphSAGE can be found in the original paper. In this section, a high-level overview of the algorithm will be provided. The forward pass of the algorithm follows the message-passing concept of the general Message Passing Framework (3.4.3.1). GraphSAGE performs K iterations, and in each iteration, it executes the equations 3.5 and 3.6 as follows:

$$m_{N(v)}^{(k)} = AGGREGATE^{(k)}(\{h_u^{(k-1)}, \forall u \epsilon N(v)\}) \tag{3.8}$$

$$h_v^{k+1} = \sigma(W^k \cdot CONCAT(h_v^{(k-1)}, m_{N(v)}^{(k)})) \tag{3.9}$$

The **AGGREGATE** function can be any function, as long as it operates over an unordered set of vectors. It can be any function that ideally is symmetric and trainable. Some candidate functions presented in the original paper are a mean operator, a max pooling operator, and even a more complex aggregator based on the LSTM architecture. The weight matrices $W^k \ \forall k \epsilon 1, ..., K$ of the

UPDATE step are trainable parameters. Thus, this step is feeding the concatenated vectors through a fully connected layer with a non-linear activation function σ.

An interesting concept around the GraphSAGE framework is its relation to the Weisfeiler-Lehman Isomorphism test (3.1.3.4). As it is shown in the original paper, a variant of GraphSAGE is an instance of the WL test, known as "naive vertex refinement". This similarity between GraphSAGE and the WL test, gives the theoretical background for the algorithm, to represent effectively the structure of each node's local neighborhood.

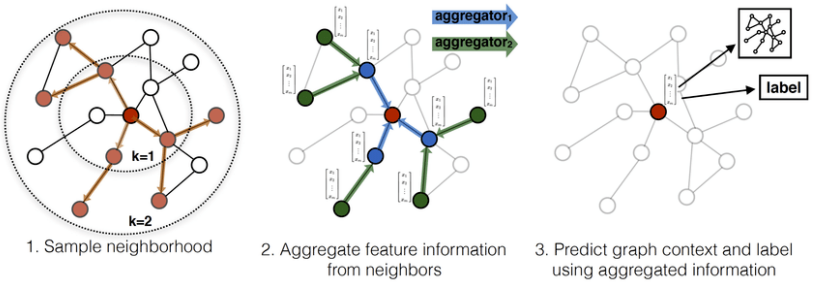

Figure 3.29: Visual Representation of the sample and aggregate technique of SAGE Hamilton et al. (2017a)

3.5.2 k-GNNs

k-GNNs (Morris et al. (2018)) are neural network architectures based on the k-dimensional WL algorithm. The intuition on this category of GNNs is that they perform the message-passing process between subgraph structures, instead of nodes.

As stated in the original paper, k-GNNs are generalizations of **1-GNNs**, which is the most basic GNN model. The general concept of 1-GNNs, has already been covered in the Message Passing Framework section 3.4.3.1 of this thesis. In short, their functionality is formulated by the equations:

$$f^{(t)}(u) = \sigma(f^{(t-1)}(u) \cdot W_1^{(t)} + \sum_{w \epsilon N(v)} f^{(t-1)}(w) \cdot W_2^{(t)}) \tag{3.10}$$

where $W_1^{(t)}$ and $W_2^{(t)}$ are parametrized matrices, and σ denotes a component-wise non-linear function. The generalization of the equation 3.10, where the aggregation of the neighborhood features and their merge with the current hidden state of the node can be performed by any differentiable, permutation invariant function, is equivalent to the process formulated by the equations 3.5 and 3.6 of the Message Passing Framework (3.4.3.1).

The mathematical formulation of k-GNNs is based on computing feature vectors for k-element subsets $[V(G)]^k$ over the set $V(G)$ of the graph's vertices. In each layer of a k-GNN, the equation 3.10, that computes the new feature vector of the subset s is modified as follows:

$$f_k^{(t)}(s) = \sigma(f_k^{(t-1)}(s) \cdot W_1^{(t)} + \sum_{u \epsilon N_L(s) \cap N_G(s)} f_k^{(t-1)}(u) \cdot W_2^{(t)}) \tag{3.11}$$

where $N_L(s)$ denotes the local neighborhood of s, and $N_G(s)$ denotes the global neighborhood of s.

A key feature of **k-GNNs** is the existence of an **Hierarchical Variant**, that combines layers with different values of k, and therefore combines different representations of different granularities. This concept is visualized in figure 3.30. A detailed formulation of this variant can be found in the original paper.

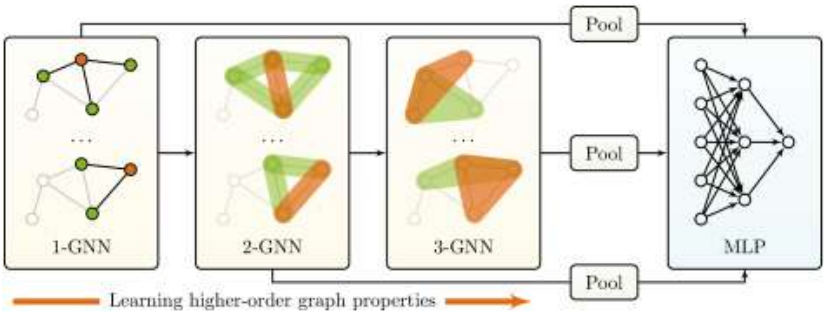

Figure 3.30: Visual Representation of the Hierarchical Variant of k-GNNs. Multiple k-GNNs are combined, with the extracted representations coming from different granularities. Hamilton et al. (2017a)

3.5.3 GAT

GAT (Veličković et al. (2017)) stands for **Graph Attention Network** and is an extension to the graph convolutional neural networks that have been reported in the previous sections. Graph Attention Networks utilize the concept of attention mechanisms, which have been widely used in many sequence-based tasks of machine learning. The general intuition is that they extend the **AGGREGATE** method of the message-passing framework, combining it with attention mechanisms so that the messages coming from each neighbor have different importance on the newly produced embedding. This is opposed to other GNNs, such as GraphSAGE, that assign the same importance to all the nodes of the same neighborhood.

More formally, a single layer of a graph attention network takes as input a set of node features $h = \{\vec{h_1}, \vec{h_2}, ..., \vec{h_N}\}, \vec{h_i} \epsilon R_F$ and transforms them to a set of output node features $h' = \{\vec{h'_1}, \vec{h'_2}, ..., \vec{h'_N}\}, \vec{h'_i} \epsilon R_{F'}$. At first, a linear transformation, which is parametrized by a weight matrix W, is applied to the input features. Then, the attention takes place on each node, via a shared attention mechanism α, that maps features pairs into **attention coefficients** e_{ij}. The attention coefficients are defined for pairs of nodes, and intuitively compute the importance of node's j features, for the computation of the features of node i. An illustration of this process can be seen in figure 3.31. Mathematically, attention coefficients are defined as

$$e_{ij} = \alpha(W\vec{h_i}, W\vec{h_j}) \tag{3.12}$$

In practice, as the original paper states, these attention coefficients are computed only for the pairs of nodes that are neighbors in the graph. An interesting extension, which is covered in the original paper, is the usage of multi-headed attention. The intuition is that multiple independent attention transformations a_k are trained, and the features produced by each transformation are aggregated, with concatenation in hidden layers and mean pooling in the final layer, to produce

expressive feature vectors. An analytical mathematical formulation of this multi-headed attention is out of the scope of this section and can be found in the original paper.

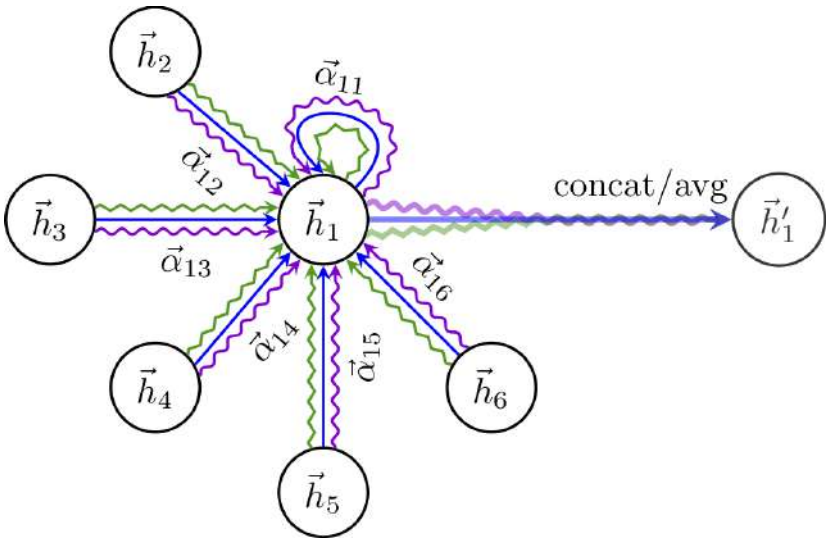

Figure 3.31: Visual Representation of the message-passing step on a Graph Attention Network. The trainable attention coefficients $a_{1v}^{\vec{}}$ lead to the embedding of each node v being aggregated with a different level of importance for the computation of the embedding $\vec{h'_1}$. The three colors illustrate that the attention is multi-headed (with three attention heads). Veličković et al. (2017)

3.5.4 GIN

GIN (Xu et al. (2018)) stands for **Graph Isomorphism Network** and is a GNN variant with a deep theoretical background. The general concept of the original paper is that the **expressivity** of graph neural networks is highly dependent on the **aggregation** function that is used in the message-passing step. As one can understand by the term **Graph Isomorphism Network**, GINs are significantly affected by the concept of graph isomorphism. More specifically, the studies around GINs have focused on the Weisfeiler-Lehman test on graph isomorphism.

The power of the WL test in distinguishing non-isomorphic graphs is due to the **injective** nature of the neighborhood's embeddings aggregation at each iteration step. Injective functions are known for mapping distinct elements to distinct values. In the context of GNNs, the embeddings on a node's neighborhood can be seen as a multiset and the aggregation scheme as a function that takes as input that multiset. Therefore, **an injective aggregation function should be suitable for mapping different node neighborhoods to different node embeddings**.

The key concept of the original paper is that GNNs are at most as powerful as the WL test in distinguishing different graph neighbors. It is proved that GNNs with injective aggregation and readout functions are **as expressive as the WL test**.

Next, this concept is combined with the Universal Approximation Theorem (Hornik (1991), which intuitively states that *standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function* can approximate any continuous function within a specific range. This theorem leads to the idea that the aggregation and the read-

out functions of a GNN can be multi-layer perceptrons, that can learn and approximate injective functions.

In the original paper, is it proven that these ideas can be formalized, to generate the node's v next hidden state at step k with the equation:

$$h_v^{(k)} = MLP^{(k)}((1 + \epsilon^{(k)}) \cdot h_v^{(k-1)} + \sum_{u \epsilon N(v)} h_u^{(k-1)}) \tag{3.13}$$

where ϵ is a trainable parameter or a fixed scalar.

In summary, as stated in the original paper, these concepts lead to the conclusion that GIN is a provably maximally powerful GNN under the neighborhood aggregation framework.

3.6 Recommender Systems

This section will cover the basic theoretical background of recommender systems, such as their **taxonomy** and the most common ideas of the **traditional approaches** around them. Subsequently, the **evaluation metrics** of recommender systems, as long as the most significant **challenges** that they are facing till now will be reported. Last but not least, this section will be a transitional stage before the experimental study of this thesis, as it will connect the task of content recommendation to **Graph Neural Networks**.

3.6.1 Basic concepts

Definition 3.6.1 (Recommender System): *A recommender system or a recommendation system is a subclass of* ***information filtering system*** *that seeks to predict the rating or preference that a user would give to an item. Raghuwanshi and Pateriya (2019)*

The studies around Recommender Systems have increased significantly over the last decades. In the real world, applications of recommender systems are all around us, from product recommendations at Amazon.com to movie recommendations on Netflix.com.

Recommender systems emerged as an independent area of research during the mid-1990s. The typical description of the problem is to predict ratings that users would give to unseen items. This recommendation process can be formalized as follows (Adomavicius and Tuzhilin (2005)):

Let

- C be the set of all users
- S be the set of all items that can be recommended
- $u : C \times S \rightarrow R$ a utility function that measures the usefulness of each item for each user, where R is a totally ordered set

Then, the goal of a recommendation system is to find, for each user u', the items $s' \epsilon S$ that maximize the value of the utility function $u(c, s)$. This goal can be formulated as:

$$\forall c \epsilon C, s'_c = argmax_{s \epsilon S} u(c, s) \tag{3.14}$$

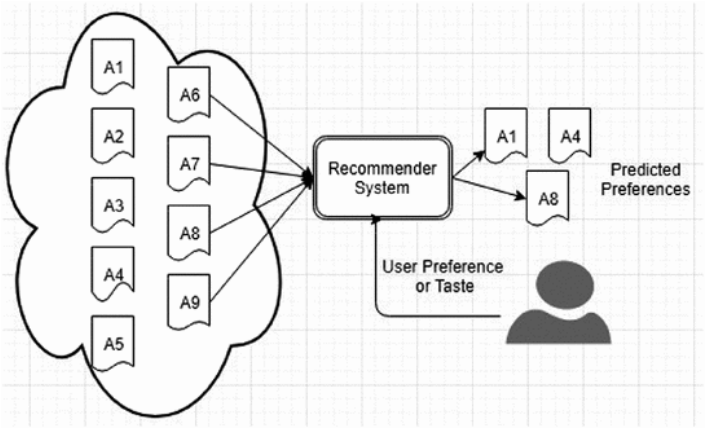

Figure 3.32: Visualization of a Recommender System
Raghuwanshi and Pateriya (2019)

In real-world scenarios, this utility function usually exists as a **rating** submitted by the user to a certain item. There are real-world cases when these kinds of ratings are not available, and the utility function is defined based on other metrics, such as the type of interaction that took place between the user and the item.

Except for the **user-item interactions**, there is also additional information that a real-world recommender system can utilize. Usually, each user has a set of features, called the user's **profile**. At the same time, the items being recommended, are also characterized by a set of **item features**. For example, in a real-world movie recommender, users' profiles can include data such as the users' age or gender, and each movie has features such as the title, the release year, its genres, etc. All these types of information, along with the history of user-items interactions can be utilized to achieve successful recommendations for each user. Several approaches exist for deciding between using the items' features or the user-item interactions. The approach to this dilemma, along with other design choices, lead to multiple types of recommender systems.

3.6.2 Main Challenges

A multitude of challenges exists when developing a recommender system. In this section, we will refer to some of the most important ones.

Multiple real-world recommender systems suffer from the challenge of **data sparsity**. This challenge occurs when there are large item sets, and therefore there are few rating values for each item. This sparsity can lead to poor predictions and is relevant to the **cold-start** problem. This problem occurs both for new users and new items in a system. Intuitively, when a new user is registered into the system, there is no information about the user's tastes, and therefore recommending content to this user is more complex. In the same way, when a recommender system follows a collaborative-filtering approach, new items cannot be suggested to users, until at least one user has submitted a rating for them. A usual approach to solving the cold-start problem is making new users submit a specific number of ratings so that the system can gain some knowledge of their preferences.

Another challenge that can occur in real-world recommender systems, is recommending content to users with special tastes. These users are referred to as **gray/black sheep**. In other words,

the existence of users whose taste is not consistent with any other user in the system, makes the recommendation process extremely complex. A more detailed analysis of this problem, as long as some proposed solutions can be found in Alabdulrahman and Viktor (2021).

3.6.3 Taxonomy

As per Raghuwanshi and Pateriya (2019), the multitude of recommender systems can be categorized, according to the information that they utilize to generate their predictions, in the following groups, that are also visualized in figure 3.33:

- **Collaborative Filtering (CF)**

  In these types of recommenders, the history of user-item interactions is being used.

- **Content-based recommending**

  In contrast, content-based recommenders, focus on user profiles and items features, ignoring any history of user-items interactions

- **Hybrid approaches**

  These approaches combine collaborative and content-based methods, utilizing both items' features and interactions history.

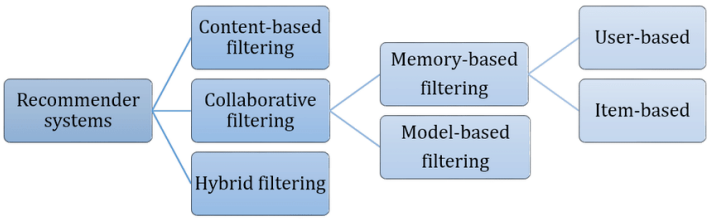

Figure 3.33: Visualization of a traditional Recommender System approaches Belkacem (2021)

3.6.4 Traditional Approaches for Collaborative Filtering

In collaborative filtering, the recommender system aims to find patterns in the rating behavior of different users, and utilize them to associate users with each other. As can be seen in figure 3.33, collaborative filtering approaches can be usually divided into **memory-based** and **model-based** approaches. The input to a collaborative-filtering algorithm is usually a user-rating matrix, that stores all the existing ratings. A visualization of the user-rating matrix can be seen in figure 3.34.

3.6.4.1 Memory-based Collaborative Filtering

Memory-based approaches for collaborative filtering are also referred to as **neighborhood-based** approaches. As per Melville and Sindhwani (2017), the main concept behind these algorithms is

that the rating of each user u for each product p is a weighted combination of the ratings that the other users v have submitted for the same product. The steps that these algorithms follow to predict the rating of user u for a product p can be formalized as follows:

1. A weight $w_{u,v}$, expressing the similarity between the past behavior of users u and v is generated for all users v.

2. The k users with the highest values of $w_{u,v}$ are selected. The set of these users is usually called the **neighborhood** of user u.

3. A weighted combination of the ratings that the neighborhood has submitted for product p is computed. This combination is the predicted rating of user u on product p.

The weights of step (1) can be computed based on numerous similarity metrics. A mathematical analysis of the most common metrics such as Pearson correlation, and cosine similarity, and a formulation of the computation of the weighted combination in step (3), can be found in Melville and Sindhwani (2017).

A disadvantage of using this approach is that the computation of the weights $w_{u,v}$ that correlate all the users pairs on the system is not scalable. As it can be easily understood, in large-scale real-world recommender systems, millions of items and users exist, and computing the similarity for all users is a computationally expensive process. For this reason, extensions, such as **Item-based collaborative filtering**, were invented. In these approaches, similarities are computed between rated items, and the predicted rating by a user u for a product p is computed as a weighted combination of the ratings on similar items.

3.6.4.2 Model-based Collaborative Filtering

Conversely, the key concept of Model-based techniques is to use the existing ratings to learn a model, that can predict future ratings. In contrast to neighborhood-based approaches, they rely on the fact that the similarity between users and items is explained by some hidden lower-dimensional representation in the data. Therefore, they do not need to load an entire dataset into the memory and perform the computations of memory-based techniques. An analysis of the most common techniques used in model-based collaborative filtering can be found in Do et al. (2010).

3.6.4.3 Advantages and Disadvantages

An important asset of collaborative filtering techniques is that they do not require **rich domain knowledge** about the items in order to perform recommendations. In addition, filtering based on user similarities provides **novelty** in the new recommendations, as the user can be related to products that differ from his/her past interactions. A usual difficulty in developing real-world recommender systems that are based on the collaborative-filtering technique is the **sparsity** of the provided data, meaning that the existing ratings are little compared to the ratings that the recommender has to predict.

Figure 3.34: Visualization of the user-item matrix used in a collaborative-filtering movie recommender system

Di Noia and Ostuni (2015)

3.6.5 Traditional Approaches for content-based recommendations

Content-based recommenders predict the new ratings of a user by comparing the user's features and his/her past interactions with items, to the items' features. The main difference with the previous technique is that content-based recommenders do not consider the interactions of other users.

A traditional mathematical formulation of this approach is to represent both each user as a vector u, and each item as a vector i, and recommend items to users based on the cosine similarity of their vectors.

3.6.5.1 Advantages and Disadvantages

A common advantage of content-based recommendations is that they offer **personalization**, as they can capture the unique interests of each user, as long as other users are not considered during the rating prediction. However, they cannot provide **novelty** in their recommendations, as they always recommend items that are similar to already high-rated items. An additional disadvantage of theirs is that they need the existence of **rich item features**, which are not always present in real-world systems, in order to perform useful recommendations.

3.6.6 Hybrid Traditional Approaches

As reported in the previous sections, both collaborative filtering and content-based recommendations have their advantages and disadvantages. In order to combine the advantages of each approach, there have been developed multiple **hybrid approaches**. As per Adomavicius and Tuzhilin (2005), the hybrid approaches can be divided into the following categories:

- Implement collaborative and content-based recommender systems separately and then combine their predictions.
- Introduce content-based concepts into collaborative filtering methods.
- Introduce collaborative-filtering concepts into content-based methods.
- Develop a general model that utilizes both types of characteristics.

3.6.7 Evaluation Metrics

There are multiple metrics to evaluate a recommender system, based on the recommendation task type, and other factors such as the provided dataset. In general, as per Raghuwanshi and Pateriya (2019) evaluation metrics can be classified into **prediction accuracy metrics** and **classification accuracy metrics**.

3.6.7.1 Traditional Prediction Accuracy metrics

Prediction accuracy metrics are used to evaluate the recommender on regression tasks, where the goal is to predict the exact rating value. In this case, the most common metrics are **Mean Absolute Error** and **Root Mean Square Error**.

Definition 3.6.2 (Mean Absolute Error): *Mean absolute error is the average of the absolute difference between the predictions and the actual values. Raghuwanshi and Pateriya (2019)*

$$MAE = \frac{1}{N}\sum_{i=1}^{m}\sum_{j=1}^{n}|r_{i,j} - \hat{r_{i,j}}| \tag{3.15}$$

Definition 3.6.3 (Root Mean Square Error): *Root Mean Square Error is computed by the square root of the average of the difference between predictions and actual values. Raghuwanshi and Pateriya (2019)*

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{m}\sum_{j=1}^{n}|r_{i,j} - \hat{r_{i,j}}|^2} \tag{3.16}$$

3.6.7.2 Traditional Classification Accuracy metrics

Classification accuracy metrics are used to evaluate recommenders that perform some kind of classification, such as a binary classification of the items as recommended or not recommended. The most common metrics of this category are **Precision**, **Recall**, and **F-measure**. These metrics are calculated based on the True-Positives, True-Negatives, False-Positives, and False-Negatives of the predictions, which are defined in table 3.3.

Table 3.3: Classification of a recommendation result for a specific item and a specific user

	Recommended	Not recommended
Used	True-Positive (TP)	False-Negative (FN)
Not used	False-Positive (FP)	True-Negative (TN)

Definition 3.6.4 (Precision): *Precision is a measure of exactness calculated by the fraction of relevant items retrieved out of all items retrieved.*

$$Precision = \frac{TP}{TP + FP} \tag{3.17}$$

Definition 3.6.5 (Recall): *Recall is a measure of completeness calculated by the fraction of relevant items retrieved out of all relevant retrieved.*

$$Recall = \frac{TP}{TP + FN} \tag{3.18}$$

Definition 3.6.6 (F-measure): *F-measure is the harmonic mean of precision and recall:*

$$F\_measure = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall} \tag{3.19}$$

3.6.7.3 More sophisticated evaluation metrics

Some additional, more sophisticated metrics, that capture the effectiveness of recommender systems are the following:

- Normalized Discounted Cumulative Gain (NDCG)
- Receiver Operating Characteristic (ROC) and Area Under the Curve, which are used for binary classification tasks
- Diversity and novelty, which are additional important aspects for the evaluation of a recommender system

These metrics were not utilized in our experiments, and therefore their detailed definitions are out of scope for the current analysis.

3.7 Graph Neural Networks and Recommender Systems

This section will connect the concepts of recommender systems and link prediction with graph neural networks. The main ideas presented in this section will be used in the implementation of our recommender, which will be described in the next sections.

3.7.1 Recommendations as Link Prediction

It has already been reported that the implementation of a recommender system can be seen, under a certain formalization of the problem, as a link (weight) prediction task. For example, in a movie recommender system, the users and the movies can be represented as the nodes of a graph, where an edge between a user and a movie represents a submitted rating. As a result, by performing link weight prediction on such a graph, one can predict the weight of the graph's edges, and therefore the rating that a user will submit for a movie. This way, by sorting the predicted ratings in descending order, the system can recommend new movies to users.

3.7.2 Graph Neural Networks for Link Prediction

There are multiple approaches for utilizing graph neural networks in the link prediction task. In general, as per Zhang (2022), these approaches can be grouped into two categories, the **node-based** methods, and the **subgraph-based** methods, depending on how the representation of the links takes place. Node-based methods represent the edges as aggregations of the node embeddings, whereas

subgraph-based methods extract a local subgraph around each edge. This section focuses on the basic concepts of node-based methods and introduces the **GNN pipeline** that will be used in the experimental part of the thesis.

3.7.2.1 Node-based methods

Node-based methods represent edges by combining the embeddings of the corresponding nodes. A well-known family of Graph Neural Networks, that belong to this category, are **Graph AutoEncoders (GAEs)**, which have been reported in the chapter 3.4.4 of GNNs taxonomy. As an interesting extension of GAEs, Kipf and Welling (2016b) introduce the **variational version of Graph Autoencoders** (VGAE). In the original paper, they demonstrate the model by using a Graph Convolutional Network **encoder**, responsible for generating expressive node embeddings, and an inner product **decoder**, that is used to represent the links of a graph, based on the corresponding node embeddings. The main idea of GAEs and VGAEs can be generalized into a **General GNN pipeline**, which can be seen in figure 3.35.

3.7.2.2 A general GNN pipeline

The graph convolutional neural networks analyzed in section 3.5 utilize the **Message Passing framework** to produce expressive node embeddings. In other words, the output of a GNN of this type is a set of node embeddings. But, to perform various machine learning tasks on graphs, these embeddings require further processing to be transformed to the expected output. For example, in graph-level tasks, these embeddings must be transformed to an output that corresponds to the entire graph, and in edge-level tasks, they must be translated to a prediction about graph edges. A usual GNN pipeline that implements this concept, begins with a GNN and concatenates a **prediction head** system to it. The role of the prediction head is to transform the node embeddings produced by the GNN, into task-specific predictions. The pipeline is visualized in figure 3.35. The pipeline is utilized in multiple researches and publications, such as Berg et al. (2017), and Dziugaite and Roy (2015).

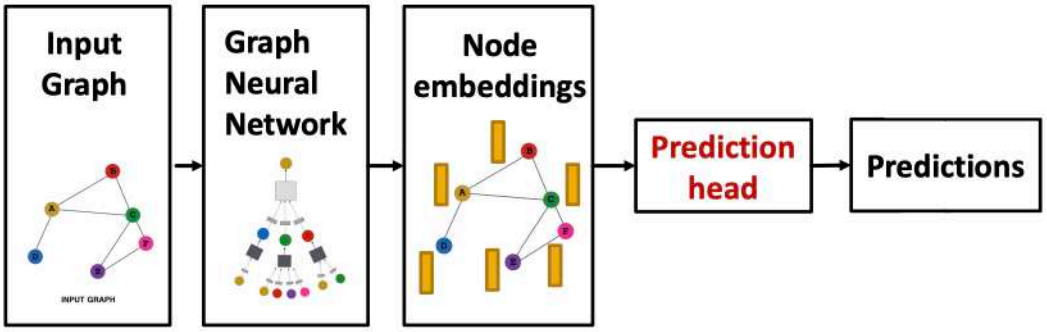

Figure 3.35: Visual Representation of the GNN training pipeline. The node embeddings produced by the GNN, are used as input to the prediction head, which transforms them into the predicted entity.

Stanford (2021)

3.7.2.3 Prediction heads for link weight prediction

The previous pipeline can be utilized to perform the **link weight prediction** task with graph neural networks. The concept of decoupling the link prediction process into two main subprocesses, the generation of node embeddings, and the prediction of the link corresponding to these nodes, is similar to the main idea of **Model S** of section 3.3.4.

The prediction heads used in the link weight prediction task, operate on pairs of node embeddings and can perform various types of transformations to predict the weight of the incident edges. Simpler methods, such as a dot product of the two node embeddings, or more complex ones, such as concatenating the embeddings and feeding them to whole multi-layer perceptrons, can be applied to generate the desired output.

3.7.3 Advantages

The main advantage of using Graph Neural Networks for the link weight prediction task is that the system can utilize both the graph structure and the node features.

In a real-world graph, there may exist multiple node features, that can seem useful in capturing the relationship between nodes. The message-passing process, which takes place inside graph neural networks, expresses the connectivity of the graph, and captures effectively the underlying structural role of the graph's nodes. In addition, it utilizes the existing node features, passing them effectively to each node's local neighborhood. Therefore, recommender systems based on predicting the edges of a graph with Graph Neural Networks, can be categorized as hybrid recommender systems, combining the benefits of content-based and collaborative-filtering approaches.

At the same time, in contrast with various traditional techniques for link prediction, graph neural networks can perform inductive learning, and under certain conditions, generalize to unseen nodes or whole new graphs. This makes graph neural networks a promising approach on the area of recommender systems.

3.8 Graph-based Related Work

In addition to the GNN architectures that are reported in section 3.5, this section reports some publications that achieve state-of-the-art performance on the **MovieLens** Grouplens dataset, in the context of recommender systems. The ideas introduced in these publications inspired the design space in our experiments, and their achieved results were used as baselines, helping us gain some intuition on the metrics our models should achieve.

It is important to note that in general, the link weight prediction task can be seen as the matrix completion problem, under the concept of recommender systems. In matrix completion, the goal is to predict missing values in a matrix by utilizing the observed values. In the context of recommender systems, this matrix represents the user-item interactions, where each row corresponds to a user and each column corresponds to an item. The goal is to predict the missing entries in the matrix to recommend items to users, likewise predicting the weight of the user-item edges in the link weight prediction task.

The publication **"Graph Convolutional Matrix Completion"** (GC-MC) by Berg et al. (2017), incorporates graph convolutional neural networks into the recommendation task, to learn expressive

node embeddings, that capture the underlying graph structure. Intuitively, a matrix completion technique is utilized to generate a user matrix and an item matrix, that are initialized with the GCN node embeddings. After these matrices are learnt, the predicted ratings for each user and movie pair can be computed as the inner product of the corresponding rows of the matrices.

However, GC-MC is a transductive learning method, meaning that it learns and makes predictions on the same set of nodes, not being able to generalize to unseen nodes without retraining. For this purpose, publications such as "**Inductive Matrix Completion Based on Graph Neural Networks**" by Zhang and Chen (2019) exist, that build on the idea of GC-MC by incorporating graph neural networks to learn node embeddings, but this time utilizing more advanced matrix factorization techniques that allow inductive learning.

A widely used extension to these works, is the "**Attetion-Based Recommendation on Graphs**" paper by Hekmatfar et al. (2022), that introduces an attention-based model that leverages the power of the attention mechanism to aggregate information from neighboring nodes in the graph. This allows the model to capture more complex relationships between items and users, which can be critical for making accurate recommendations.

Chapter 4

Implementation of the Recommender

The main part of the project was the development of a Recommender System on **The Movies Dataset** (Banik (2017)) with the usage of Graph Neural Networks. The recommendation task is modelized as a **link weight prediction** problem on the bipartite graph consisting of users and movies as nodes, and ratings as weighted edges.

4.1 The Dataset

This section will be an introduction to **The Movies Dataset** (Banik (2017)), which is the main dataset of the task, and **MovieLens** (Grouplens), on which the main dataset was based.

4.1.1 Movielens

The MovieLens dataset is a collection of movie ratings data made available by the GroupLens Research lab at the University of Minnesota. The dataset includes ratings, movie information, and demographic data on users, gathered from the MovieLens website. It is commonly used in research on recommendation systems and has been used in a number of published papers. Over time, the dataset has been expanded to include more and different types of data, such as movie genre information.

The MovieLens dataset has been released in different versions with varying sizes over the years. A detailed analysis of the history around these datasets can be found in Harper and Konstan (2015). Some of the most commonly used versions are:

- MovieLens 100K: This is the original version of the dataset and includes 100,000 ratings given by 943 users to 1,682 movies.
- MovieLens 1M: This version of the dataset includes 1 million ratings given by 6,040 users to 3,706 movies.
- MovieLens 10M: This version of the dataset includes 10 million ratings given by 69,878 users to 10,677 movies.
- MovieLens 20M: This version of the dataset includes 20 million ratings given by 138,493 users to 27,278 movies.
- MovieLens 25M: This is the most recent version of the dataset which includes 25 million ratings, and 3 million tag applications, and applies to 58,000 movies, and 280,000 users.

4.1.2 The Movies Dataset

The dataset that we used for our experimental research on the recommender system was The Movies Dataset (Banik (2017)). This dataset can be seen as an extension of the Full MovieLens dataset, as it contains additional metadata for its movies. The data was scraped from various sources such as IMDb and TMDB. It is mainly used for Recommender Systems and Natural Language Processing (NLP) tasks. Both datasets can be used to develop a recommender system, but they have some important differences.

Of course, the most important difference between the two datasets is the metadata that is contained in The Movies Dataset. The MovieLens datasets are more ready-to-use for developing a recommender system, as they mainly focus on the submitted ratings, and do not require much preprocessing. At the same time, The Movies Dataset provides an interesting case study, as the utilization of the additional metadata on the recommending process requires more effort in the data preprocessing part, and seems promising to help the recommender system learn more accurate representations of the movies, and by providing additional context for the user-item interaction.

The metadata that was mainly utilized in the development of the recommender systems includes the title, crew, cast, genres, spoken languages, production countries, and production companies of each movie.

In the Kaggle page of The Movies Dataset, there are provided two versions of the dataset. The full version provides 25M ratings between users and movies, and the small version provides 100K ratings. The 100K version of the dataset was used in our experiments.

Figure 4.1: Example of a keywords JSON array

Figure 4.2: Example of a genres JSON array

More specifically, the dataset consists of the following csv files:

- **credits.csv** This file contains the three columns **id**, **cast**, and **crew**. Obviously, the first column corresponds to the id of the movie, and the other two columns contain **JSON arrays**, where each object is a cast member or a crew member of the movie. The typical form of an object representing a cast and a crew member can be seen in figures 4.3 and 4.4.

- **keywords.csv** This file contains the two columns **id** and **keywords**. The keywords column contains a list of JSON objects, where each object corresponds to a genre. The typical form of a genres list can be seen in figure 4.2.

- **movies_metadata.csv** This CSV file contains multiple columns about the movies' metadata. Each row contains multiple pieces of information, such as the movie's title, tagline, overview, genres, production companies, production countries, and spoken languages. An interesting property of this file is that the columns that correspond to the genres, production companies, production countries, and spoken languages of each movie, are formatted as **JSON arrays**. The format of a movie's genres array can be seen in figure 4.2.

- **ratings.csv** and **ratings_small.csv** These files contain the rating records of the dataset. The full version contains 25M ratings, and the small one contains 100K ratings. The CSV files consist of the four columns **userId**, **movieId**, **rating**, and **timestamp**.

- **links.csv** and **links_small.csv** These files have an auxiliary role and contain some mappings with the keys of the movies, that are used to "join" the previously reported CSV files. They map each **movieId** that is used on the rating files, to their ids in the **movies_metadata.csv** and the other files.

Figure 4.3: Example of a cast JSON object

Figure 4.4: Example of a crew JSON object

4.2 Graph database

This chapter will be the first chapter of this thesis that deals with more practical issues and technologies around the development of the recommender system. It begins with a short introduction to graph databases in general, and to Neo4j, which is the graph database we used in our recommender system. Then, the motivation behind modeling The Movies Dataset as a graph, and the process of storing it in Neo4j (Neo4j (2012)) will be covered.

4.2.1 Graph Databases

As per Angles (2018), a **graph database system** is a system designed for handling graph-like data following the basic principles of database systems. Graph databases are gaining interest in applications where the data are highly connected to each other in a complex way, such as social networks.

The fundamental abstraction behind all database systems is the **database model**. This model defines three main components:

- A set of data structure types
- A set of query operators on the data
- A set of integrity rules

As far as the graph databases are concerned, the database model is called **Graph Database Model**. All three components of a database model are defined in a graph-wised manner. Data structures are modeled as graphs, query operators are defined as operations on graphs, and integrity rules are defined over the graph structure. A formal definition of the **property graph database model**, which is utilized by multiple graph database systems, such as Neo4j, is presented by Angles (2018).

4.2.2 Neo4j

Neo4j (Neo4j (2012)) is a graph database management system that allows managing a large amount of data, powered by a native graph database. It is based on the **property graph model**, meaning that data is represented as nodes and edges, rather than tables and rows.

Neo4j provides a powerful query language, called **Cypher**, that takes advantage of data connections and allows traversing graph paths avoiding the complex joins of traditional relational databases. In addition, there exists a multitude of **built-in algorithms** for graph analytics, that allow extracting additional features from the graph. At the same time, there have been developed multiple libraries, that enhance the utilization of Neo4j on machine learning and data science tasks, such as the **graph data science library** (GDS (2022)). An extensive analysis of these algorithms and tools will be made in the following chapters, during the description of the process of modeling our dataset as a graph in Neo4j. Last but not least, Neo4j offers seamless integration with other libraries and frameworks, such as **Py2neo**(Py2neo (2020), that allows the development of external components and services, such as REST APIs, over the Neo4j graph database.

4.2.3 The dataset modeled as a graph

The first design choice that had to be taken during the implementation of the recommender system, was the way of modeling the dataset as a graph. **The target was to design the database in such a generic way, that its rich context can be utilized in various other tasks**, except for just the link weight prediction task of our recommender system.

The resulting graph should be optimized in the context of the **space complexity**, meaning that repeated entities, such as genres and keywords, that can be related to multiple movies, should be modeled as nodes. This formulation, is also **time efficient**, as the usual queries will be about finding all the related entities to a specific movie, or finding all the movies related to a specific entity, such as the genre *Animation*. Therefore, modeling these entities as separate nodes, instead of keeping them as node attributes of each movie, will utilize the nature of graph databases to efficiently solve these types of queries. The graph is heterogeneous, as there should be more than one type of nodes and edges.

The following nodes types were created:

- **Movie**: Each node corresponds to a movie, with rich **metadata** features, such as the title, tagline, description, and release date.
- **Genre**, **Keyword**, **Production Company**, **Production Country**, **Language**: The role of each node can be easily inferred by the name of its label. All these nodes have only two node attributes, that correspond to a **primary key** (such as an internal id for the keywords nodes, or an iso_3166_1 for the production countries nodes) and a **name**.

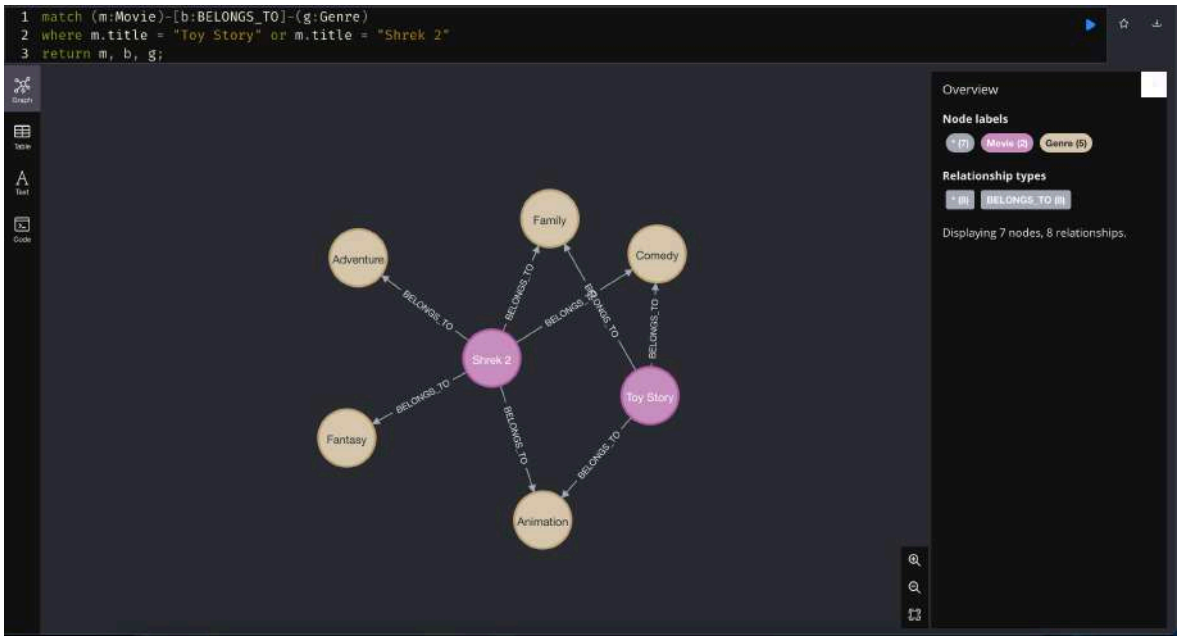

Figure 4.5: A subgraph with two specific movies (pink-colored nodes) and their genres (yellow-colored nodes), visualized by Neo4j Desktop (Neo4j (2021))

- **Person**: Nodes of this label are used to represent the **cast and crew members** of each movie. We avoid representing the same person with a separate node every time he/she participates in a separate movie to gain some space and time efficiency. The information about the relationship between a movie and a user will be saved onto the corresponding edge, as we will analyze in the proceeding of this section.

- **User**: Each node bearing this label corresponds to a user of the recommender system. In The Movies Dataset, there are not provided any demographic or other user-related information. In order to allow the database to be used by external APIs and services, we generated a random and unique **username** and a simple **password** for each user, to make the user data more realistic.

The design choices around the edges are equally important for the quality of the resulting graph. The various nodes that were reported, are connected to each other via the following edges types:

- **BELONGS_TO**: This type of edge connects a movie to a genre. Obviously, a movie can belong to multiple genres, and a genre can be related to multiple movies. A subgraph containing two random movies and their genres is visualized in figure 4.5.

- **HAS_KEYWORD**: This type of edge is similar to the **BELONGS_TO** edges, but is used to connect a **Movie** node to the corresponding **Keyword** nodes.

- **PRODUCED_BY**, **PRODUCED_IN**, **SPEAKING**: In the same context, these types of edges are used to relate **Movie** nodes to their **Production Company**, **Production Country**, and **Language** nodes.

- **HAS_CAST**: These edges are used to relate **Movie** nodes to the corresponding **Person** nodes, that have participated as cast in the movie. An interesting part about these edges, is that they

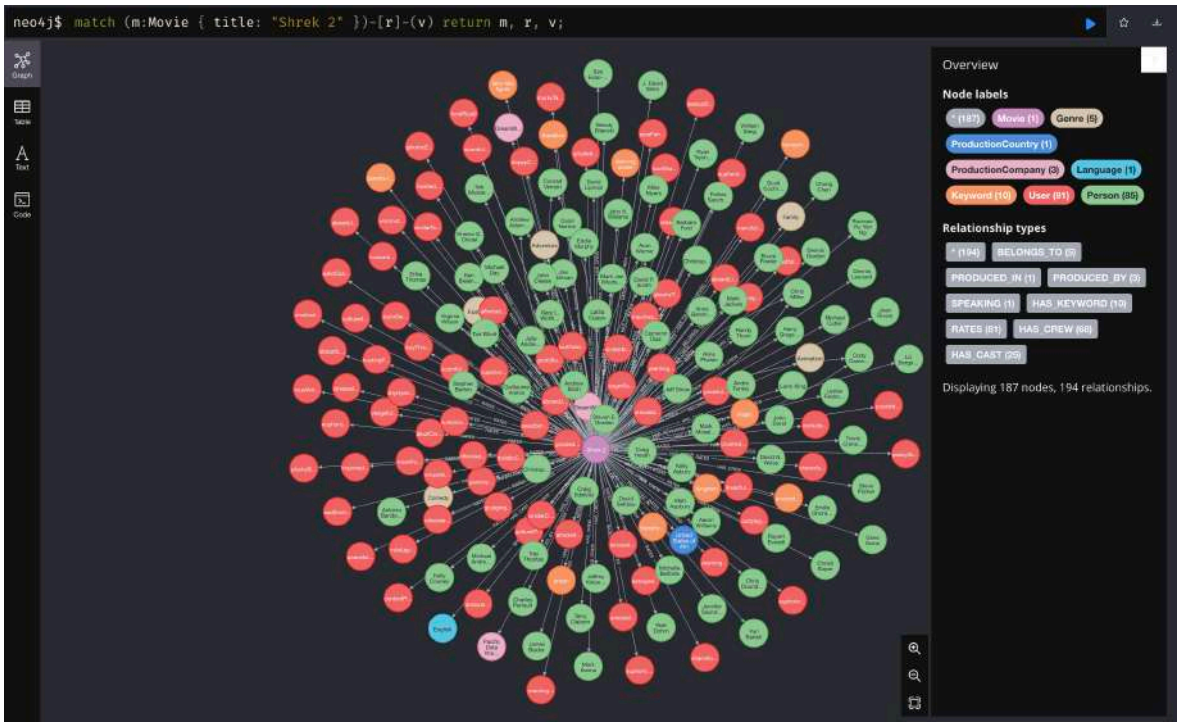

Figure 4.6: A subgraph with the 1-hop neighborhood of a specific movie. Multiple types of nodes (denoted by different colors) are easily fetched with a simple Cypher query. Visualized by Neo4j Desktop (Neo4j (2021))

are the first type of edges used till now in our graph, which bears some information on them. More specifically, the **HAS_CAST** edge between a **Movie** and a **Person**, has the attributes **character** and **order**. This formulation frees us from creating a new node for the same person, each time that he/she participates in a different movie as a different character.

- **HAS_CREW**: In a similar way to the **HAS_CAST** edges, these edges relate **Movie** nodes to the **Person** nodes, where the person has participated as a member of the crew in the movie. These types of edges, bear information that defines the role of the person in the specific movie, with the attributes **department** and **job**.

- **RATES**: Each edge of this type represents a rating submitted by a **User** node to a **Movie** node. The **RATES** edges have the attributes **datetime** and **rating**. Our recommender system will aim to perform link weight prediction, on the **rating** attribute of the **RATES** edges.

This design of the graph database allows performing simpler or more complex queries to retrieve a wide range of insights about the graph structure and the connectivity of the entities. A simple, but yet useful, query that indicates the effectiveness of the design is finding all the related entities of a specific movie. The Cypher code, as long as the visualization of the resulting subgraph, can be seen in figure 5.8.

4.2.4 Technical background of the graph database initialization

An important and challenging process for the development of the recommender system, was the process of storing the original CSV files of the dataset in a Neo4j instance, following the principles

that were analyzed in the previous section. For this purpose, reusable Python scripts were developed, that allowed building multiple Neo4j instances for the different-sized versions of the dataset, with different graph properties.

The development **Neo4j** instances were created and managed with the **Neo4j Desktop application** (Neo4j (2021)). As stated in the original website, **Neo4j Desktop** is a local development environment for working with Neo4j and comes out-of-the-box with multiple extensions, such as the **Neo4j Browser**, that enable executing and visualizing queries in the connected Neo4j instances.

The process of parsing the dataset files, and storing them in the database, after the corresponding preprocessing, was made with code in Python. More specifically, the python library Pandas (McKinney et al. (2010)) was used to parse the CSV files into dataframes. Afterward, the proper processing was performed on them with Python scripts, and the data about nodes and edges were sent to the database via the Python library Py2neo (2020).

Py2neo is a client library for working with Neo4j from within Python applications. The main library features that were used for the development of our recommender system, are the variety of ways to execute queries towards the database, as long as features specifically designed for operating with a large amount of data.

Py2neo provides multiple ways to connect and execute code in a Neo4j instance, from within a Python application. For our development process, we executed both raw Cypher queries, sent from Py2neo to Neo4j as raw python strings and queries utilizing the Py2neo matching module. Using the matching module, the developer can execute multiple types of queries, without having to write complex and error-prone Cypher syntax. At the same time, Py2neo offers high-level methods, to store a large amount of data in a Neo4j instance in a time-efficient way. More specifically, the Py2neo bulk operations API allows executing operations to the database in a bulk way, making them run more effectively. This utility, along with the built-in functionality of Neo4j to load **large CSV files** by applying **periodic commits**, were heavily exploited to efficiently construct graphs with hundreds of thousands of nodes.

4.2.5 Small version of the dataset (100K ratings)

Following the initialization of the graph, multiple queries were run to visualize some aggregated metrics, and estimate the need for preprocessing the data before using them as input to the recommender system. In this context, multiple visualizations regarding the density and the connectivity of the graph were generated. The queries were executed from python notebooks, using Py2neo to communicate with the Neo4j instance, and the results were visualized using the matplotlib Python library.

Initially, these scripts were run to examine the graph built from the small version of the dataset, where 100K ratings exist. We try to capture the density of the ratings in the graph, visualizing the distributions of the ratings per user and per movie in figures 4.8 and 4.7.

At the same time, some aggregating queries are being run, to compute the average values of these metrics. We count that there exist 9067 movie nodes, 672 user nodes, and 99802 rating edges between them. The average number of ratings per movie is 11, and the average number of ratings per user is 148.5. Considering the distribution of the ratings per movie, we observe that most of the movies are related to a small number of ratings. We expect that this **sparsity** of the graph might

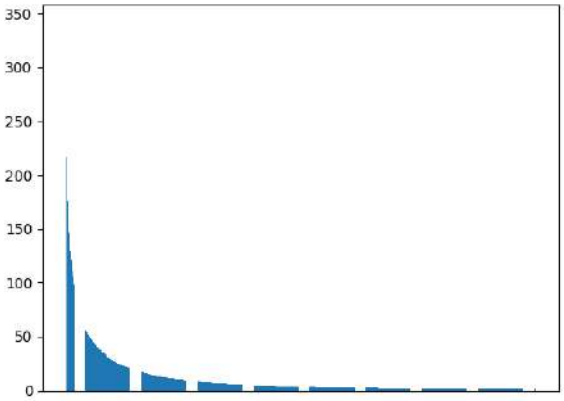

Figure 4.7: Number of ratings per movie distribution in the original version of the small 100K dataset. Visualized with matplotlib.

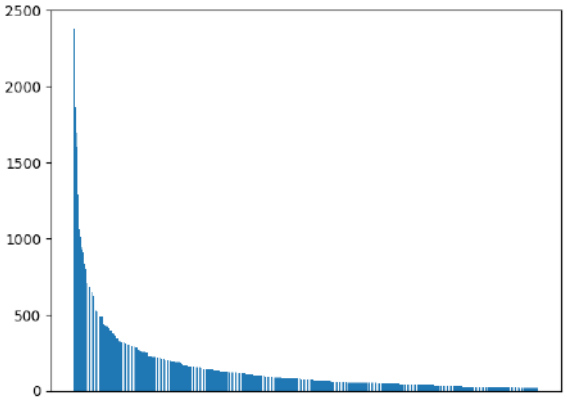

Figure 4.8: Number of ratings per user distribution in the original version of the small 100K dataset. Visualized with matplotlib

cause limitations to the learning ability of the recommender system. For this reason, as it will be reported subsequently in this section, we consider that the large version of the dataset with 26M ratings could be used to build a more dense version of the graph.

Another important factor that needs to be considered about the quality of the small version of the produced graph, is the distribution of the rating values. This distribution is visualized in 4.9. As can be seen, the ratings with values around 4 are exceeding the ratings with values less than 3, and the dataset can be considered slightly imbalanced.

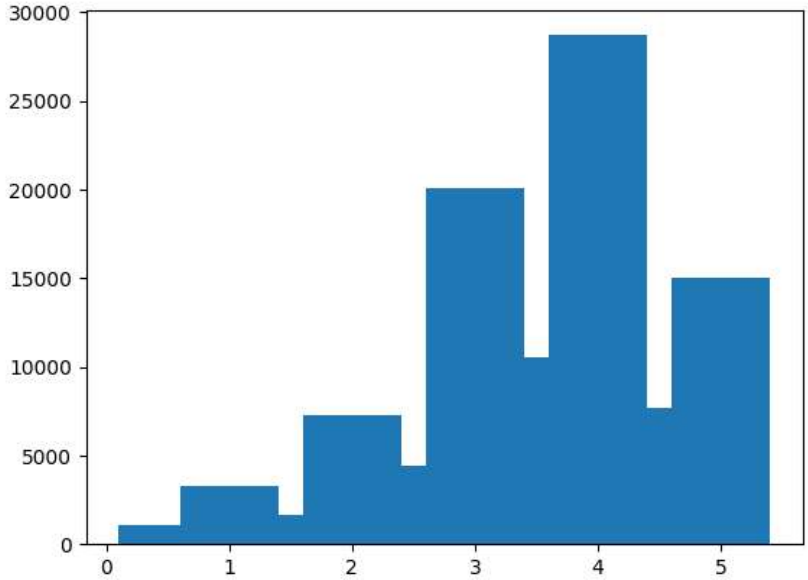

Figure 4.9: The rating values distribution in the original version of the small 100K dataset. Visualized with matplotlib.

To address this issue, as long as the sparsity issue, we utilize the reusable Python code that was developed initially and build some additional database instances, that aim to resolve these restric-

tions of the original dataset. We left the exploration of the metrics our model achieves on these custom datasets as future steps.

4.2.6 Encoding the movie content

An important design decision for the development of the recommender system was the way to utilize the **rich movie metadata** of The Movies Dataset.

In contrast to the MovieLens dataset, which contains few metadata available for each movie (such as the title, and its genres), The Movies Dataset contains rich movie features. When performing machine learning tasks on MovieLens, one can easily encode the title and genres as vectors, by using various NLP techniques for the first, and techniques such as one-hot encoding for the latter.

This approach can be partly applied to The Movies Dataset, too. As far as the title of each movie is concerned, we used Sentence Transformers to encode it into a sentence embedding. The technical details behind the functionality of Sentence Transformers are out of the scope of this thesis. An extensive analysis of them can be found in Reimers and Gurevych (2019). The main issue with The Movies Dataset, is that the one-hot vectors technique appears quite restrictive for encoding information about related entities that appear in large numbers. As an example, both in MovieLens and in The Movies Dataset, there exist only 20 genres. Therefore, encoding the genres of each movie as a one-hot vector of dimension 20 is a straightforward solution. In contrast, when it comes to encoding the cast or crew members of a movie, there exist around 150K Person nodes in the small version of The Movies Dataset. Encoding the cast members of each movie as a one-hot vector would lead to a large dimension in the movie's features, and would not capture the underlying graph structure of the dataset.

Bearing in mind these limitations, in the performed experiments we utilize well-known node embedding algorithms, to express the correlation between the contents of the movies in an effective way. These algorithms generate node embeddings of controlled dimensions and can capture the underlying graph structure using techniques such as random walks and matrix factorization.

4.2.7 Node embeddings with Neo4j Graph Data Science Library

In order to encode the movies' content with node embeddings, the Neo4j Graph Data Science Library (GDS (2022)) was used, to generate node embeddings based on various techniques. In this section, some technical details about this process will be covered, and a variety of the generated embeddings will be visualized.

4.2.7.1 Neo4j Graph Data Science Library

Neo4j GDS (2022) is a library that extends the functionalities of a Neo4j instance, providing optimized implementations of common graph algorithms. These algorithms are exposed as simple Cypher procedures and were used in the development of our recommender system to generate node embeddings that encode information about the content of the movies. The integration of the Neo4j Graph Data Science with Neo4j is straightforward and made it possible to run a variety of algorithms with ease and then store the produced embeddings as node attributes back to the database.

In advance of analyzing the specific algorithms that were used, it is important to report the outline of the general process, independently of the algorithm choice. The generation of the embeddings takes place in a graph data model which is a projection of the Neo4j property graph data model. The developer can specify the nodes, edges and attributes that the projected graph will contain, and store the corresponding graph projection in an in-memory container of projected graphs, called the **Graph Catalog**.

For the production of the movie content embeddings, it is important to ignore the **User** nodes and the **rating** edges. Taking these entities into account can lead to **information leakage** during the training of the model. For this reason, we restrict the node embeddings to be generated considering only the nodes and edges that are related to the content of each movie. The main approaches to the process can be categorized based on how the induced graph is constructed, and are the following:

- Node embeddings on **"node pairs"**: In this approach, we generate **multiple separate** node embeddings for each movie, each time based on a different related entity. For example, we can generate an embedding only about the genres of the movies, based on the induced graph that contains only the **Movie** and **Genre** nodes. We repeat the same process, of generating induced graphs with only two types of nodes at each time for the **Keyword**, **Production Company**, **Production Country**, **Country**, and **Person** nodes. After generating the node embeddings and writing them back to the graph database for each movie node, each movie is characterized by a set of independent node embeddings, that are used as input features to the recommender system. An example induced graph, containing only **Movie** and **Keyword** nodes can be seen in figure 4.10.

- **Node embeddings on the "whole content"**: In this approach, we consider the induced graph that contains information about the whole content of each movie. More specifically, the induced graph contains all the **Movie**, **Genre**, **Keyword**, **Production Company**, **Production Country**, **Country**, and **Person** nodes. After running the chosen algorithm, each movie node bears one embedding that encodes the whole content of the movie.

4.2.7.2 Generated Node Embeddings

Neo4j Graph Data Science Library offers a variety of algorithms for the generation of node embeddings on projected graphs. We chose the algorithms in such a way as to cover the three main implemented categories, which are random walks, random projections, and graph neural networks. The Python scripts that implement the integration to the Neo4j GDS were written in a generic way so that multiple types of embeddings can be generated with almost no modification of the original code. The target of this process was to compare the expressiveness between the multiple types of algorithms, tuning only the most common hyperparameters of theirs, without excessive focus on the specialized hyperparameters of each algorithm.

The first chosen algorithm is **node2vec**. The main concepts of the theoretical background of this algorithm are provided in section 3.2.5.5. In the experimental process of utilizing node2vec for node embeddings, mainly the default hyperparameters provided by the corresponding GDS procedure were used. The hyperparameter of the algorithm that was tuned to produce more expressive node

Figure 4.10: The induced graph containing only movies and keywords. For simplicity purposes, a limit on the number of visualized nodes and edges was applied. Visualized with Neo4j (2021).

embeddings is the **embeddings dimension**, which was set to 256 for the node pairs embeddings, and 512 for the node embeddings encoding the whole content for each movie. The visualizations of node2vec embeddings that correspond to the Movie-Genres, Movie-Production Companies induced graphs, and to the whole content graph, are visualized in figures 4.11, 4.12, and 4.13.

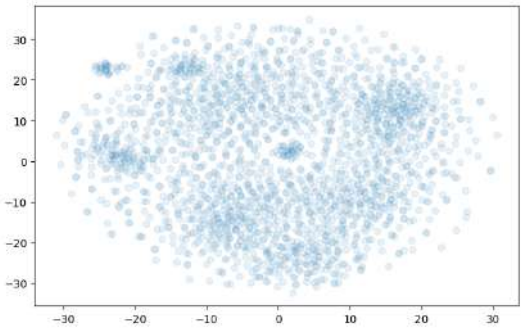

Figure 4.11: Movies embeddings on the **Movie-Genre** induced graph, generated by **Node2Vec** algorithm. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

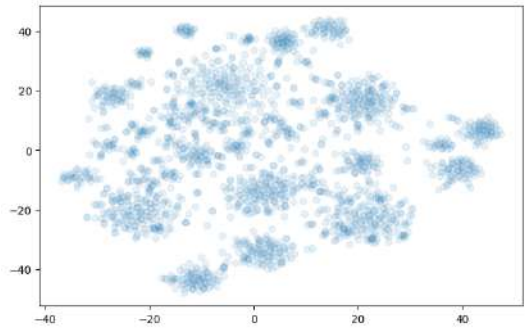

Figure 4.12: Movies embeddings on the **Movie-Production Company** induced graph, generated by **Node2Vec**. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

An interesting outcome of these visualizations is that the Node2vec algorithm seems to be able to detect communities in the case of the Movie-Production Company nodes graph, and in the case of the graph containing the whole movie content, but seems to not be able to differentiate communities in the case of the Movie-Genre induced graph, which in theory has a simpler structure. It is impor-

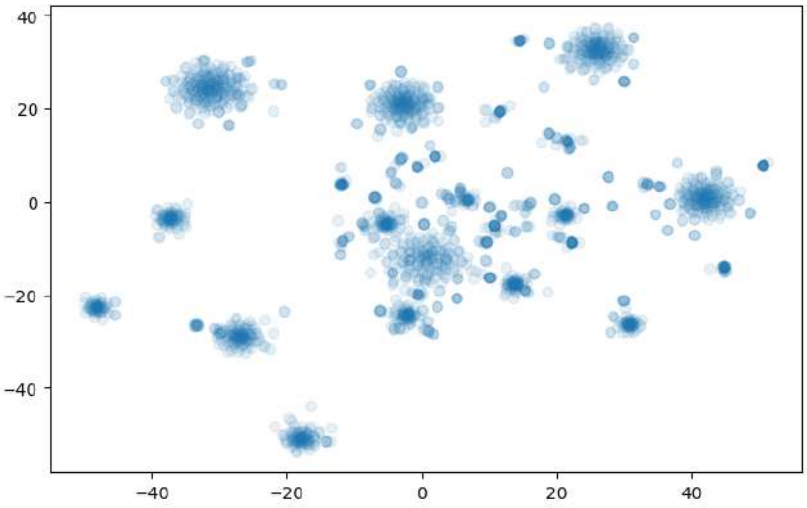

Figure 4.13: Movies embeddings on the **whole movies-content** induced graph, generated by **Node2Vec** algorithm. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

tant to note that node embeddings produced by the Node2vec algorithm in multiple other induced subgraphs, such as on the subgraph with movies and cast members, seem to fail in distinguishing the movies.

Subsequently, the **Fast Random Projection** algorithm was utilized to generate the node embeddings. The basic theoretical background of this algorithm is covered in section 3.2.5.6. The hyperparameters of the algorithm that were tuned in our experiments are the embedding dimension and the iteration weights. As stated in the official documentation of the algorithm, the iteration weights tune the importance of nodes of each distance in the generation of the current node's embeddings. Some visualizations of the FastRP embeddings for the 165K version of the dataset, can be seen in figures 4.14, 4.15, and 4.16.

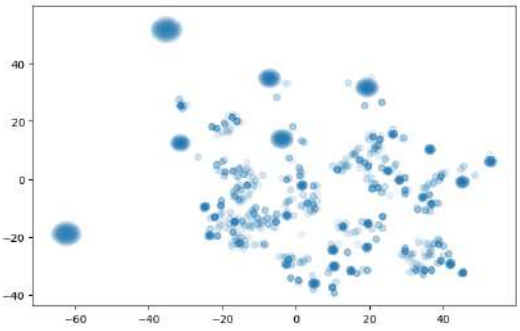

Figure 4.14: Movies embeddings on the **Movie-Genre** induced graph, generated by **FastRP** algorithm. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

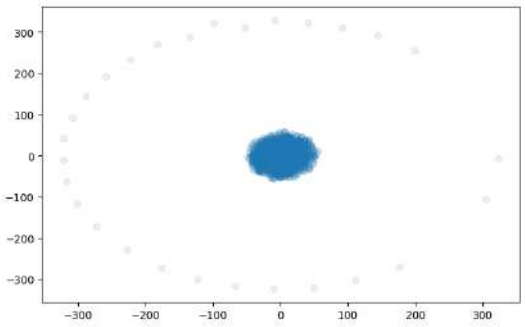

Figure 4.15: Movies embeddings on the **Movie-Keyword** induced graph, generated by **FastRP**. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

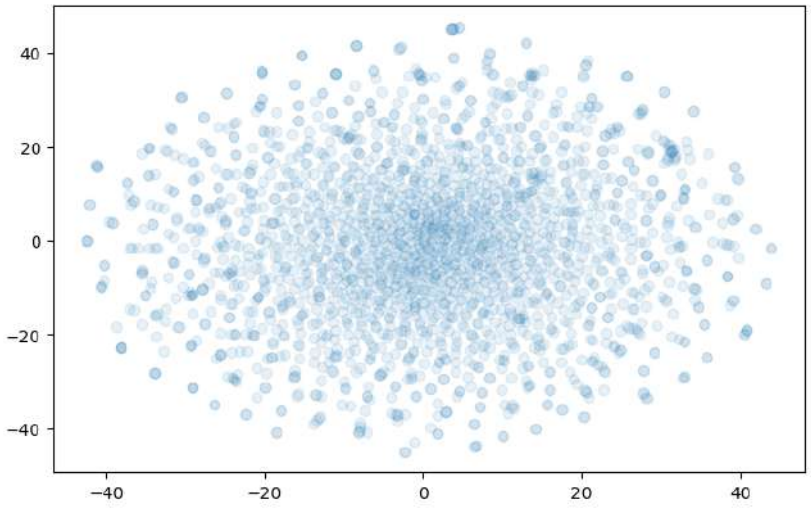

Figure 4.16: Movies embeddings on the **whole movies-content** induced graph, generated by **FastRP** algorithm. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

These visualizations indicate that the FastRP algorithm detects movie communities according to their genres in an effective way, but does not operate at the same level on the Movie-Keyword and on the whole movie content graph.

As a final approach to the challenge of encoding the content of each movie with node embeddings, we can leverage the Graph Neural Network **GraphSAGE**. Once again we focused on tuning the most common hyperparameters, such as the embedding dimension. An important note is that the **GraphSAGE** algorithm requires some initial node features for each node in the projected graph. In our case, the target was to capture the structural relationship between nodes. As a result, providing input to the GraphSAGE model features such as the title of a movie or the name of a genre, would not help. As reported in the official documentation of the module, in such cases, some topological features for each node, such as the node degree, can be provided as initial node features for the generation of the node embeddings. After generating and writing back to the database the embeddings for each movie, we visualized them, to gain some intuition on their expressiveness. Some examples of these visualizations can be seen in figures 4.17, 4.18, and 4.19.

It is important to note that the movie embeddings generated by the GraphSAGE algorithm, seem to form clusters, that are separated in a clearer way than in the previous methods. We expect these conclusions to be verified by the experiments that will be reported in the following parts of this thesis.

4.3 The model

This section will cover the technical background behind the development of the model that performs the link weight prediction task for our recommender system. At first, a short reference to the Pytorch Geometric library (Fey and Lenssen (2019)) is made, which is the Python library that was used to implement, train and test the Graph Neural Networks of our experiments. Subsequently, we proceed with the architecture of our model, the main types of Graph Neural Networks that were utilized,

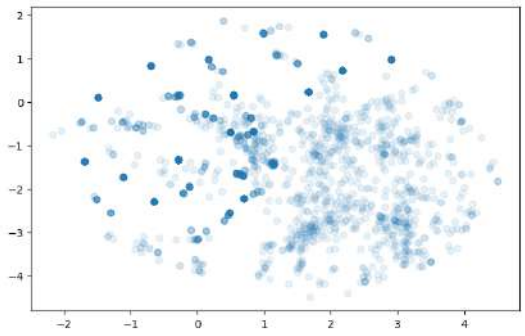

Figure 4.17: Movies embeddings on the **Movie-Genre** induced graph, generated by **GraphSAGE**. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

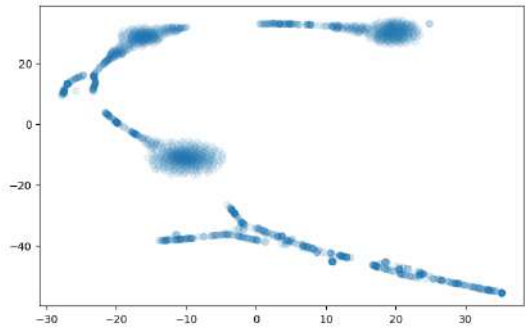

Figure 4.18: Movies embeddings on the **movies-cast members** graph, generated by **GraphSAGE**. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

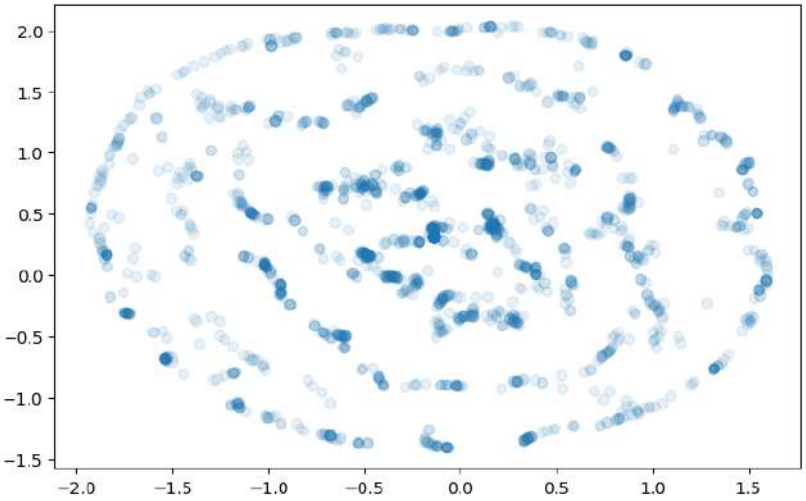

Figure 4.19: Movies embeddings on the **whole movies-content** induced graph, generated by **GraphSAGE**. Visualized with matplotlib, after dimensionality reduction with the scikit-learn library (Pedregosa et al. (2011)).

and finally, we present an overview of the hyperparameters that were related to the model structure and that were tuned during the experiments.

4.3.1 Pytorch Geometric

For the development of our recommender system, Pytorch Geometric (Fey and Lenssen (2019)) was used to build the whole machine learning model. Pytorch Geometric is a Python library for deep learning on irregularly structured data, such as graphs. It integrates seamlessly with Pytorch, and therefore allows developers to easily utilize traditional tensor computation when performing machine learning tasks on graphs.

The library offers support for a variety of convolutional graph neural networks, such as GraphSAGE (3.5.1), GAT (3.5.3), and GIN (3.5.4). It provides fully-customizable implementations of the

message-passing layers used in these architectures, allowing developers to easily combine them and perform multiple experiments. In addition, Pytorch Geometric provides seamless integration with a variety of commonly-used datasets, that are used as benchmarks to evaluate machine learning models on multiple types of tasks. It helps developers download and use these datasets, by specifying only the dataset's name. One of these datasets, that are provided out-of-the-box with Pytroch Geometric, is, for example, the small version of the MovieLens dataset.

4.3.2 Architecture

As far as the architecture of our model is concerned, the pipeline of section 3.7.2.2 was followed. Our model consists of two main submodels: a Graph Neural Network (GNN) encoder, and an Edge Decoder. The general pipeline is visualized in figure 3.35, but with the Edge Decoder being in the position of the Prediction Head in our case.

Intuitively, the GNN Encoder receives as input the bipartite graph consisting only of **Movie** and **User** nodes, and the **rating** edges between them. The graph containing information about the movies' content, such as their genres and keywords, is encoded as a feature of each movie, via the node embeddings techniques that were described in sections 4.2.6 and 4.2.7. The GNN Encoder learns representations for the movies and users. These node embeddings are then passed as input to the Edge Decoder module. The Edge Decoder is a simple Multi-Layer Perceptron, that concatenates the user and movie embeddings and predicts the weight of the corresponding edge, which is the rating that we aim to predict. This pipeline is visualized in figure 4.20.

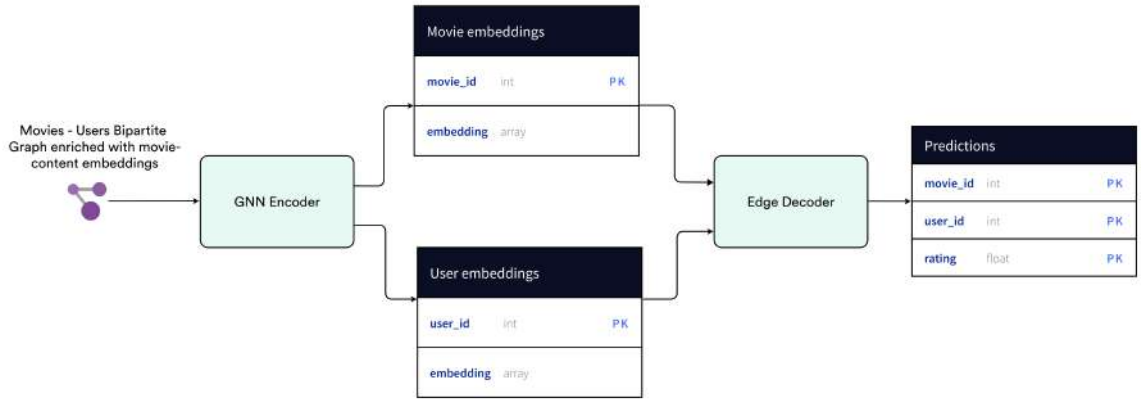

Figure 4.20: The architecture of the model. GNN Encoder receives the bipartite graph of users and movies as input and produces the embedding of each node. Edge Decoder receives these embeddings, and generates the predicted ratings for each pair. Visualized with terrastruct.com.

4.3.3 GNN Encoder

The GNN Encoder component is a simple Graph Neural Network, consisting of multiple layers. For the purposes of this project, as will be reported in the experiments section, four types of Graph Convolutional Neural Networks were used. These variants are GraphSAGE 3.5.1, GraphConv 3.11, GAT 3.5.3, and GIN 3.5.4. The main hyperparameters that were tuned, and that are related to the

GNN structure, are the number of the GNN layers, the number of hidden channels per layer, and the existence of skip connections. The concept of skip connections will be explained in the following section.

4.3.3.1 Stacking Multiple GNN Layers

The theoretical background of operating with a single GNN layer differs slightly for each GNN variant and is described in section 3.5. When developing a Graph Neural Network, an important design choice is the number of its layers. In general, when stacking K GNN layers, the features of each node are computed based on the K-hop neighborhood of the node. For example, with a single GNN layer, the embedding of each node is computed after aggregating the embeddings of the node's immediate neighbors. The set of neighbors that participate in this aggregation, form the **receptive field** of the node.

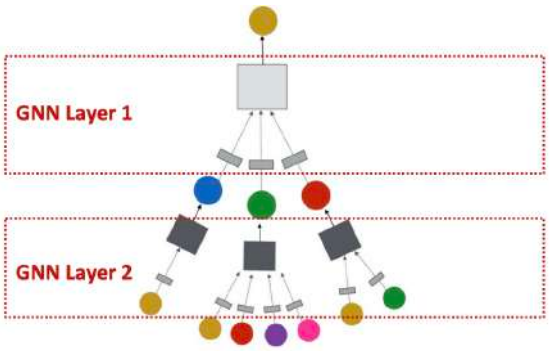

Figure 4.21: The message-passing process for a single node, in the case of a GNN with two layers. With two GNN layers, node features with a maximum distance of two are aggregated. Stanford (2021)

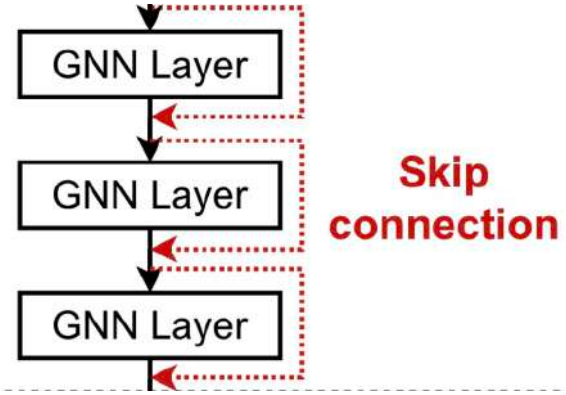

Figure 4.22: Visualization of a GNN with 3 layers, where skip connections are used as shortcuts between the layers, to increase the impact of earlier layers on the final node embeddings. Stanford (2021)

Stacking multiple GNN layers can lead to the phenomenon of **oversmoothing**. Oversmoothing refers to the situation where the representations of the graph nodes become indistinguishable, even though the nodes might actually have different roles in the network. An extensive analysis of this phenomenon can be found in Chen et al. (2020). The connection between oversmoothing and stacking multiple layers of GNNs, is that intuitively, in deep GNNs, the receptive fields of multiple nodes tend to have high overlaps. Therefore, as the features of multiple nodes are computed based on the same neighbors, these features tend to have the same values. As a result, the embeddings produced for these nodes do not express their differences in the real network. A high-level intuition of this phenomenon is visualized in figure 4.23.

A general approach to empowering deep graph convolutional neural networks can be found in Li et al. (2019). A widely-used solution for tackling the oversmoothing problem is the technique of **skip connections**. The usage of skip-connections, was one of the hyperparameters that were tuned in our experiments, as will be seen in the experimental part of this report. Intuitively, skip connections, increase the impact of the earlier layers on the final embeddings, by adding some shortcuts, as can be seen in figure 4.22.

4.3.4 Edge Decoder

Edge Decoder is a simple multi-layer perceptron that receives the user and movie embeddings generated by the GNN Encoder. The hyperparameters that are related to the model structure are the number of then hidden layers, as long as the number of hidden channels within each layer. The output of the Edge Decoder is a vector with the predicted rating values for each one of the provided pairs of node embeddings.

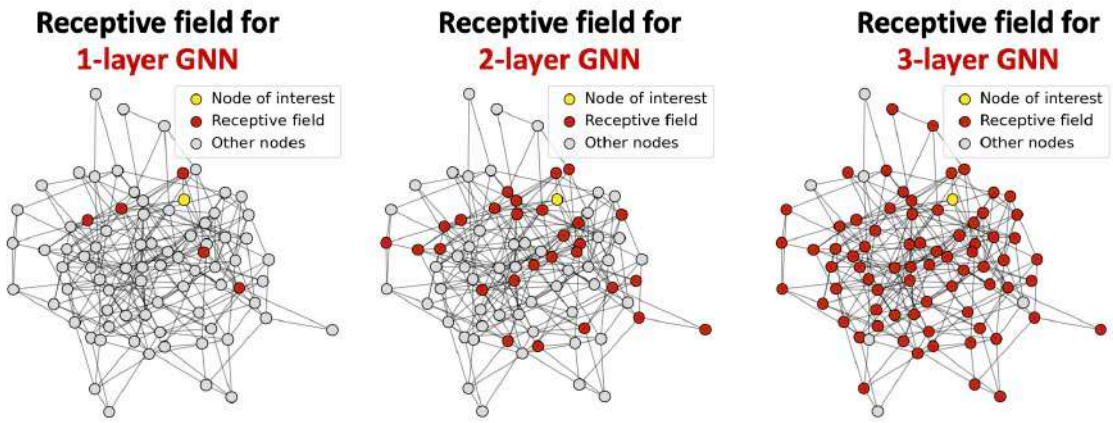

Figure 4.23: As the layers of the GNN increase, the receptive field of the yellow node tends to capture the whole graph. We expect the embeddings of multiple nodes to converge to the same values.

Stanford (2021)

4.4 Evaluation of the Recommender System

Evaluating a recommender system can be based on a variety of metrics. Bearing in mind that the link weight prediction task can be seen as a regression task, in our experiments, we focus on metrics that are generally used for evaluation on regressions tasks.

Therefore, the metric that was used for the evaluation of the recommender systems that we built was the Root Mean Square Error (RMSE). The definition of this metric is reported in section 3.6.7.1. RMSE is the metric that is commonly used to evaluate predictions in the MovieLens dataset, as can be read in numerous related papers (Han et al. (2021), Rashed et al. (2019), Zhang and Chen (2019), Darban and Valipour (2022)) on the corresponding PapersWithCode leaderboard.

It is important to note that The Movies Dataset is a relatively new dataset, and there could not be found many benchmarks available for its evaluation. For that reason, the most common benchmarks on the MovieLens dataset are reported to gain intuition on the expected performance of our models. The baseline for our experiments will be the performance of a simple Graph Neural Network, that is chosen arbitrarily to be GraphSAGE, with a relatively small number of hidden layers, to avoid the oversmoothing phenomenon, that is described in section 4.3.3.1. In the baseline experiment, we will not utilize the various movie metadata that exists in the dataset. We will perform the predictions utilizing only the user-movie interactions, and the title of each movie.

Chapter 5

Implementation of the Platform

5.1 Motivation

A significant proportion of the development effort was dedicated to constructing a fully operational platform. The main motivation behind this decision was to experiment with the **integration** of a machine-learning model that performs movie recommendations, in a real-world scenario. At the same time, the usage of the platform can be seen as particularly favorable for expanding the database with additional data. Deploying the whole web application, and making it available to multiple users, can increase the available ratings, and even add demographic information about the users, leading eventually to the formation of a **new version of The Movies Dataset**. This dataset, can afterward become available to the developing community, and lead to a series of new experiments, assisting the academic community in advancing the research in the field of recommender systems. Ultimately, the research around implementing a set of **scalable** services, that handle the decades of thousands of the available data in our system, collaborating with a **graph database** and an **ML model**, and exposing a multitude of functionalities through a friendly **user interface**, can be considered as a separate area of study.

5.2 Related work

The development of the platform was inspired by multiple platforms that exist on the web, where users can create an account and receive personalized movie recommendations. The platform that seems to be the most similar one to our implementation, is the MovieLens website. MovieLens is a movie recommendation website that was developed by researchers at the University of Minnesota. It has been operating since 1997, and after receiving multiple updates, it currently offers movie and TV shows recommendations. The main feature of the MovieLens website, which inspired our implementation, is that users get predictions on the exact rating they could submit for each movie on the website.

5.3 Architecture - Components

The platform consists of **multiple independent components**, that are designed for serving distinct purposes, in favor of scalability. Figure 5.1 shows a high-level visualization of the system architecture. More specifically, except for the system's **database**, which is a Neo4j (2012) instance, there exist **three APIs**, following the RESTFul principles, and a **front-end web application**. The interac-

tions between the components are performed as **asynchronous** calls, and the exchanged data follow the **JSON** format. As one can observe in the corresponding diagram, not all the components interact with each other. These design choices were made for **security** and **scalability** reasons. More details will be reported in the section that corresponds to each component.

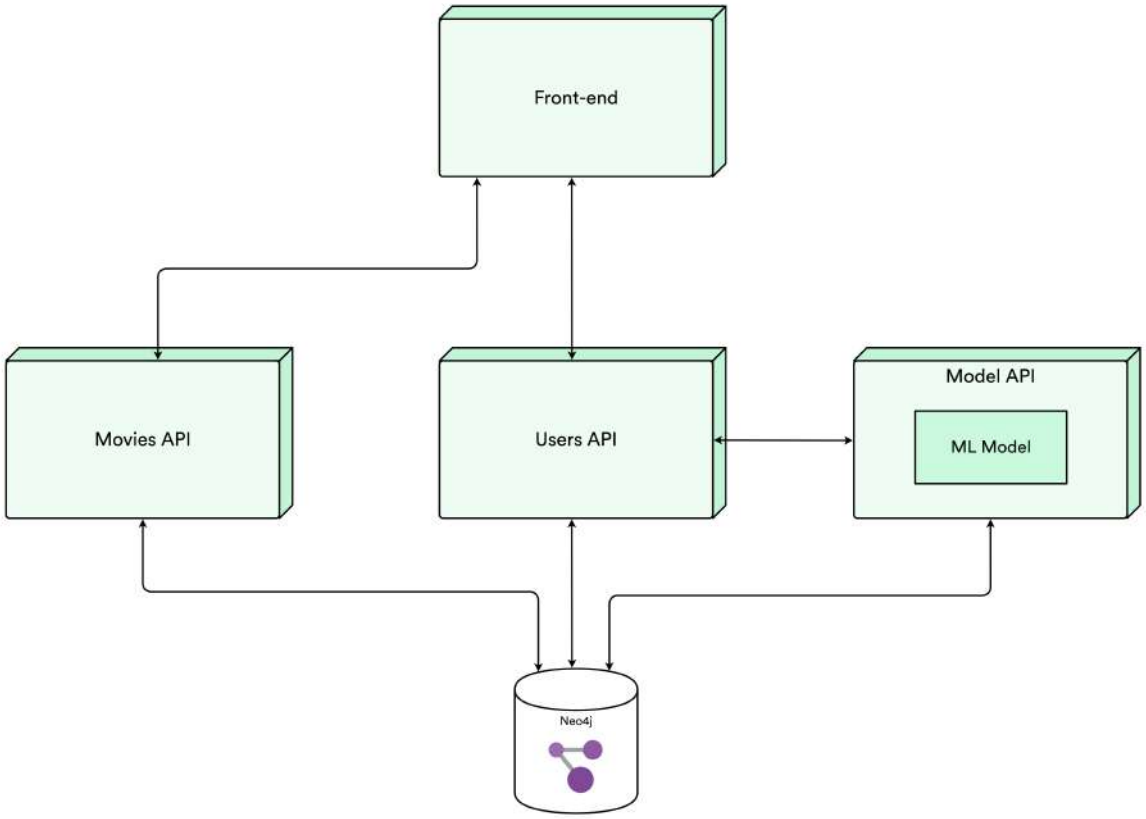

Figure 5.1: The architecture of the platform, as a high-level component diagram. The platform consists of the graph database, three REST APIs, and a separate front-end web application. Visualized with app.terrastruct.com.

5.3.1 Graph Database

An extensive approach to the technologies used for the development of the system's database has already been made in numerous sections of this thesis. In short, the database is a Neo4j (2012) instance, meaning that it belongs to the category of **graph databases**. It stores the data natively as nodes and edges, being able to perform effectively complex queries, utilizing the underlying graph structure of the data. It is initialized with a preprocessed version of The Movies Dataset (Banik (2017)), containing **node embeddings** for each node, generated by numerous graph algorithms, that are provided out-of-the-box by GDS (2022). The graph database **interacts only with the three APIs**. We forbid direct communication between the graph database and our front-end, to avoid storing the database connection credentials in the front-end, as that would introduce a **security vulnerability** to our system.

5.3.2 REST APIs

The architecture of the REST APIs was the most important design choice as far as the structure of the system's components is concerned. The decision to partition the functionalities that the system's **back-end** will perform, into multiple APIs, is mainly based on scalability, performance, and security concerns. For the development of the APIs, more than one programming languages and frameworks were used. Each one will be reported in the corresponding section.

5.3.2.1 The Movies API

The purpose of the **movies API** is to serve general information about the movies and their content. This service performs only **read operations** on the database. The main reason that it was developed as a separate service is the lack of need for user **authentication**, as it does not expose any personal information related to a specific user. It interacts only with the database, and the front-end.

As far as the tech stack of this service is concerned, it was developed in NodeJS (Foundation (2009)), with the usage of Typescript (Corporation (2012)) as the main programming language. Numerous third-party libraries were utilized for a rapid and safe development process, but an extensive listing of them is out of scope for the current report and can be found in the GitHub repository of the project.

5.3.2.2 The Users API

The purpose of the **users API** is to handle the requests that require user authentication, such as the manipulation of user accounts, and the ratings that are submitted by each user. In addition, **users API** is responsible for the interaction with the **model API**, to get the predicted ratings for each user. The reasons that led to this decision will be covered in section 5.4. This service was developed in NodeJS (Foundation (2009)), with the usage of Typescript (Corporation (2012)) as the main programming language, likewise the **movies API**. In addition to the variety of third-party libraries that were used there, this service utilizes libraries that specialize in handling user **authentication and authorization** in a secure and developer-friendly way.

5.3.2.3 The model API

This service is responsible for encapsulating the machine learning model that performs the recommendations. The general concept is that an initial trained version of the model is saved in this service. This saving process requires serialization and deserialization of the model in a file that can be saved on the disk of the service's machine. These utilities were provided by the Pytorch library (Paszke et al. (2019)). The REST API was developed with Python, using the Flask web framework (Team (2010)). The service exposes, through four simple endpoints, the basic functionalities of the model. These functionalities will be analyzed in the section 5.4. It is important to note, that the **model API** does not interact directly with the front-end. The reason that forced us to this design choice will be covered in the section 5.4.

5.3.3 Front end

The purpose of developing a front-end module for the recommender system was to expose the multitude of implemented functionalities in a user-friendly way, making them accessible from the web in an easy-to-understand way. In addition, by making an interactive and easy-to-use web application, we aim to attract multiple users, and gather enough ratings to be able to experiment with the process of re-training the recommender system as the available data increase. For the development of the front-end part of the platform, the Javascript library ReactJS (Facebook (2013)) was utilized. The main programming languages used for the development of this service were Javascript (Eich (1995)) and Typescript (Corporation (2012)). A variety of third-party libraries were utilized, such as Material-UI (Team (2020)), and React-Query (tannerlinsley (2020)).

5.4 Integrating the model into the platform

The integration of the machine learning model into the platform was one of the most important aspects of the development process. An important design choice for our system was to deploy the model in a dedicated REST API, named *Model API*. In this section, we will cover the main reasons that led us to this decision, the functionalities that the Model API exposes to the rest of the components, as long as the main challenges we met in the development process.

5.4.1 Motivation behind deploying the model in a dedicated API

5.4.1.1 Computational Resources

The main reason for deploying the model as a separate service, is the additional needs of computing power that it might need. Re-training the model in a production environment, could be performed with extended computing resources, such as GPU, that are not necessary for the operation of the rest back-end services. Therefore, deploying the model in a separate machine, allows the developers to scale the resources allocated to it independently of the rest back-end. This flexibility helps the system in terms of scalability and performance.

5.4.1.2 Technology Stack

In addition, it is important to note that the programming language for the development of the model was Python. Decoupling the model functionalities in a separate REST API, allows us to use a variety of programming languages and frameworks to develop the rest of the back-end services. This freedom enables the developer to choose any other technology stack he/she prefers, such as NodeJS (Foundation (2009)) with Typescript (Corporation (2012)). The resulting system can be of higher quality, as the technologies that fit better to each service can be utilized. At the same time, this design choice strengthens the **maintainability** of the system, as any specialized libraries and modules related to machine learning, are totally separated from the rest of the back-end services.

5.4.1.3 User Authentication

An additional area of significance in the procedure, was the handling of **user authentication and authorization** across the multiple services. The target was to keep all the code related to user au-

thentication in a single service for **maintainability** reasons, and at the same time decouple other unrelated functionalities into separate services. In our system, the need for authentication is related to two categories of functionalities. The first category is the **CRUD** (create-read-update-delete) operations on the **ratings** and the **user accounts**, and the second category is the retrieval of the **model predictions** for a single user. The first category of these functionalities is implemented in the **users API**, as it is not related to the ML model. The retrieval of the model predictions of a specific user is obviously implemented in the **model API**. But, the noteworthy part is that the predicted ratings for a specific user should be accessed only by the same user. For this purpose, requests that originate from the front-end and are targeted to the **model API**, at first pass through the **users API**, which authenticates the user that performed the request, and forwards the request to the **model API**, acting as an **authentication middleware**.

5.4.2 Functionalities

The functionalities that are exposed through the **model API**, they are the following:

- Re-train the model for N epochs
- Refresh the in-memory dataset of the model, by re-fetching it from the graph database
- Get the rating predictions of a specific user
- Recommend top N movies to a specific user

In a production environment, the first two operations will not be accessed by the users of the platform. These will have only access to retrieving their predicted ratings and getting their personalized recommendations.

5.4.3 Challenges

5.4.3.1 Retraining Frequency

The coordination of the operations regarding the re-training of the model, and the re-fetching of the dataset, are a whole separate area of study. The automation of their execution requires extensive experiments. Multiple approaches can be found to re-training the model in the bibliography, such as an automated re-train based on time intervals, a performance-based trigger, a trigger based on data changes, and re-training on demand. The extensive experimentation on their effects on our recommender system could be considered a potential avenue for further research.

5.4.3.2 Model API Availability

Another issue was that the functionalities of **re-training the model** and **refreshing the in-memory dataset**, require an importantly larger amount of time, compared to other requests, such as predicting ratings for a specific user. As a result, whenever such a request was made, the model API was **blocked** and was unable to respond to other requests, as it was occupied with the execution of the heavy task. To address this issue, and improve the **responsiveness** of our API, we developed a mechanism to execute these functionalities in separate **threads**, allowing the API to continue handling other requests while the task is being executed in the **background**.

5.4.3.3 Handling new users

One additional challenge our system had to overcome, was how to handle new users, without having to retrain the model.

In general, graph neural networks can be trained in an **inductive** way, meaning that they can generalize to **unseen** nodes or even to whole unseen graphs. However, in our experimental setup, the graph neural networks were trained in a **transductive** way. Transductive train-test split is a popular technique in graph-based machine learning that involves using the entire graph for both training and testing a model. It is mainly used on relatively small datasets, because of the limitation that it requires the entire graph to be loaded into memory, and because inductive train-test split requires excluding multiple edges from our dataset.

In our experiments, training and testing the model with the transductive technique made it **difficult to add new nodes** to the graph. Intuitively, this arose due to the **weight matrices** of the first GNN layer being **dimensioned** based on the size of the **initial graph** it was trained on. As a result, when passing as input to the model a graph with additional nodes, the corresponding matrix multiplication could not take place. For that reason, we implemented a custom solution, that we could not find after a short research in the existing bibliography. More specifically, we **dynamically** updated the weight matrix of the first GNN layer of our model, by **adding a new column**, each time a new user was added to our system. That way, the matrix multiplication could be completed successfully.

At first, a design decision had to be made, regarding the values that would fill the corresponding vector in the weight matrix of the first GNN layer. Initially, it was considered to fill the vector with **zeros**. However, that approach would not take into account any ratings the new user would submit until the model was re-trained and the corresponding weights were updated. As a result, it was decided to fill the new column that corresponds to the user, with the average value of the corresponding weights for the existing users of the platform. That way, intuitively, the new user would receive the predictions based on the **average user** of our platform. The key feature of our approach is that the predictions for the new users can be generated without re-training the model. This represents a step towards an inductive approach in our recommender system.

5.5 Platform usage

This section reports a short description of the user interaction with our platform. As stated before, the purpose of the platform's design is to allow users to navigate quickly and easily through numerous movies, exploring the connections between the entities in the underlying graph structure. At the same time, users can get a prediction for the rating that they will submit to each movie, as it is generated by our ML model.

A user can explore the movies on the platform without creating an account. The home page of the platform is captured in figure 5.2. Users can see a list of the latest movies, the top movies and cast members, and the top genres and keywords.

The target of the system is to allow users to easily explore movies and their related entities and at the same time show users the predicted ratings. By clicking on a specific genre, the user can see the latest and the top related movies, as seen in figure 5.3.

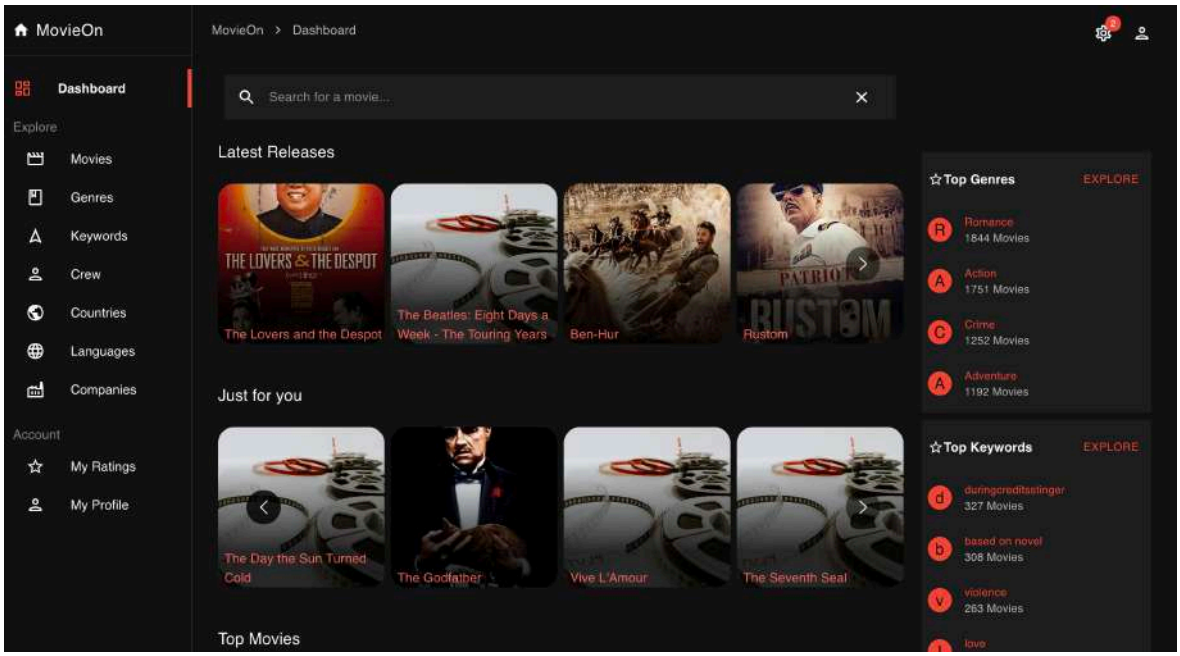

Figure 5.2: The home screen of the web application

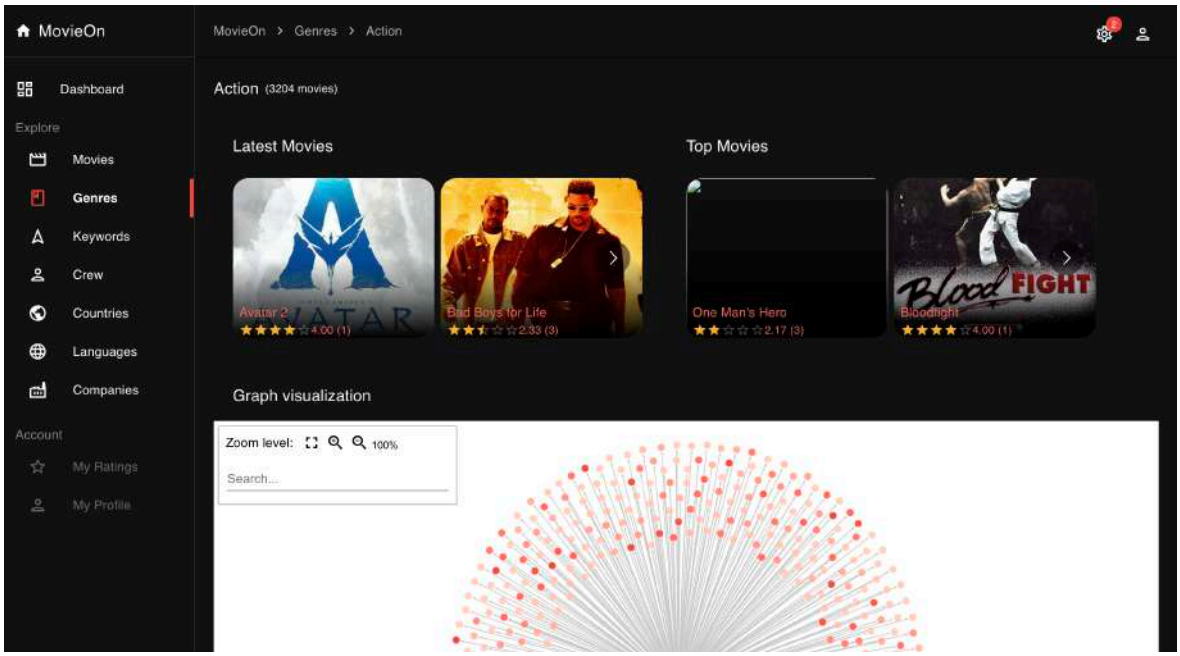

Figure 5.3: The detailed view of a specific genre. Here, users can see the latest and the top movies related to the **Action** genre.

Except for showing the top and latest movies related to an entity, such as the genre *Action*, users can also observe a sample of the neighborhood of the corresponding node in the graph database. This feature is visualized in figure 5.4, and aims to show the connectivity of the graph database to more advanced users.

In cases where the entities' relationships can be visualized in more advanced ways, the platform utilizes more advanced diagrams. For example, as it can be seen in figure 5.5, the production coun-

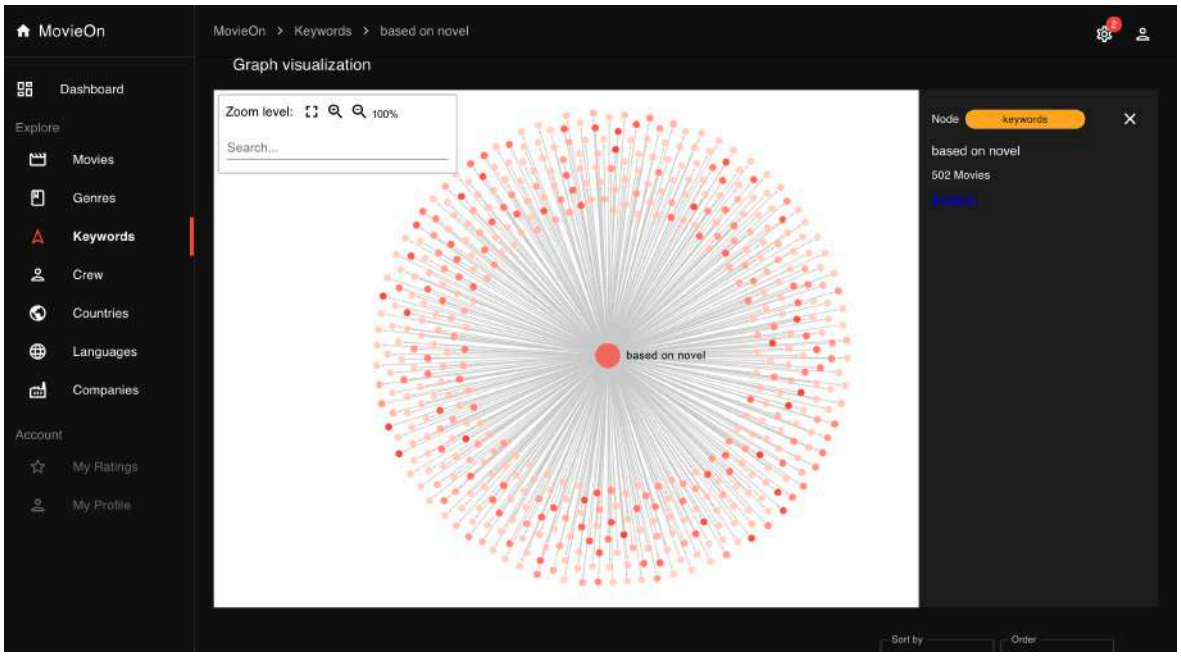

Figure 5.4: The visualization of the *"based on a novel"* keyword's neighborhood in the original graph.

tries are visualized using a choropleth map, allowing the users to explore movies easily based on this criterion.

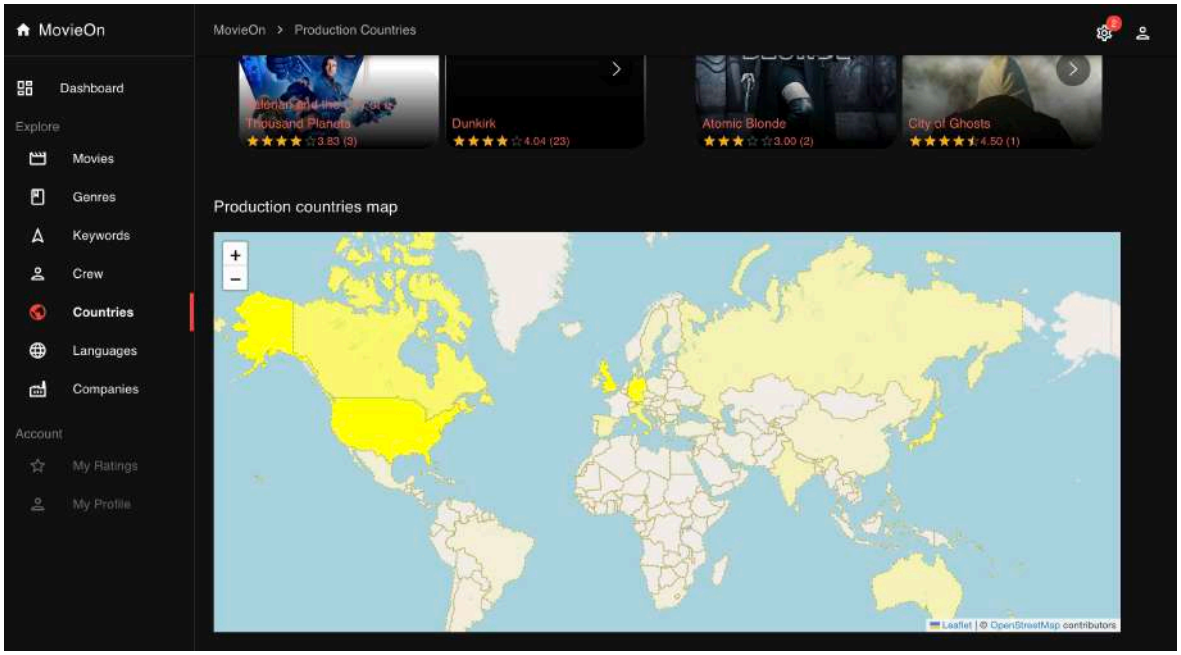

Figure 5.5: A choropleth map, colored by the number of movies produced in each country.

The exploration of new movies, can be based on the participating cast. A separate profile page is assigned to each cast/crew member, which shows the available metadata, and the related movies. This profile page is captured in figure 5.6.

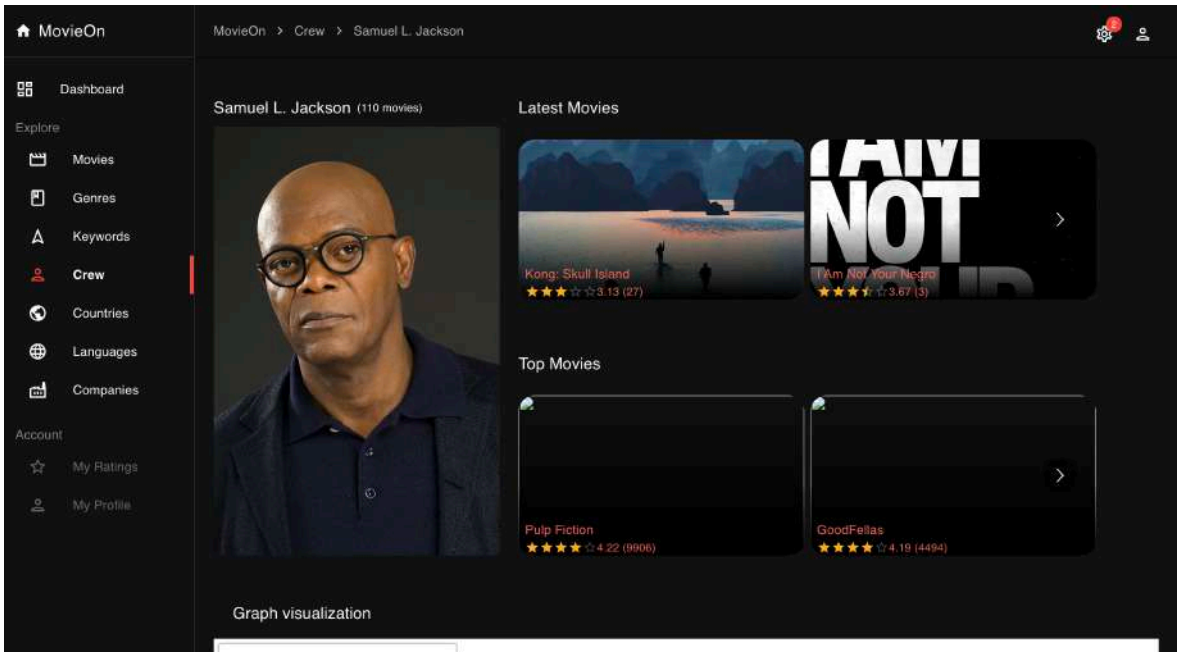

Figure 5.6: The profile page of a cast member. Here, users can see the latest and the top movies related to the person, as long as the visualization of the person's neighbor.

While navigating over the variety of the aforementioned entities, users can be led easily to a specific movie's page, with an example visualized in figure ??. There, a detailed view of the movie's metadata is gathered. In addition to this information, authenticated users have access to the predicted rating for this movie by the ML model, as long as the option to submit their rating.

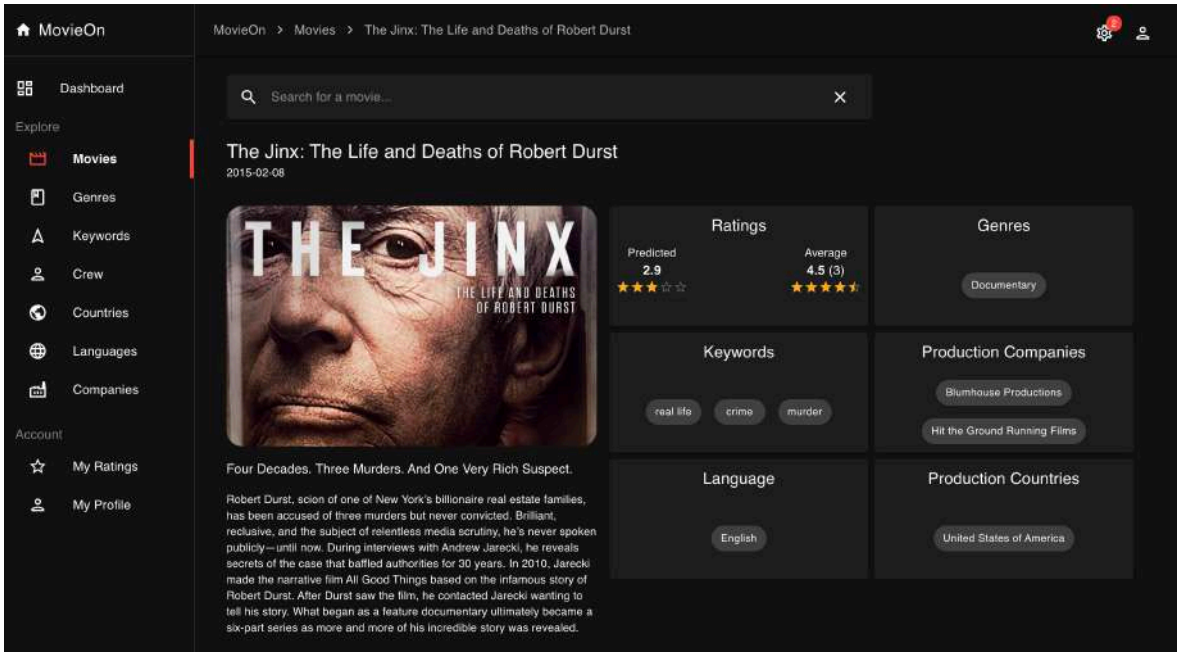

Figure 5.7: The page of a specific movie. The predicted rating of the authenticated user, as long as the movie's metadata are gathered here.

Last but not least, on the same page that refers to a specific movie, a visualization of the movie's neighborhood in the original graph takes place, as it can be seen in figure 5.8.

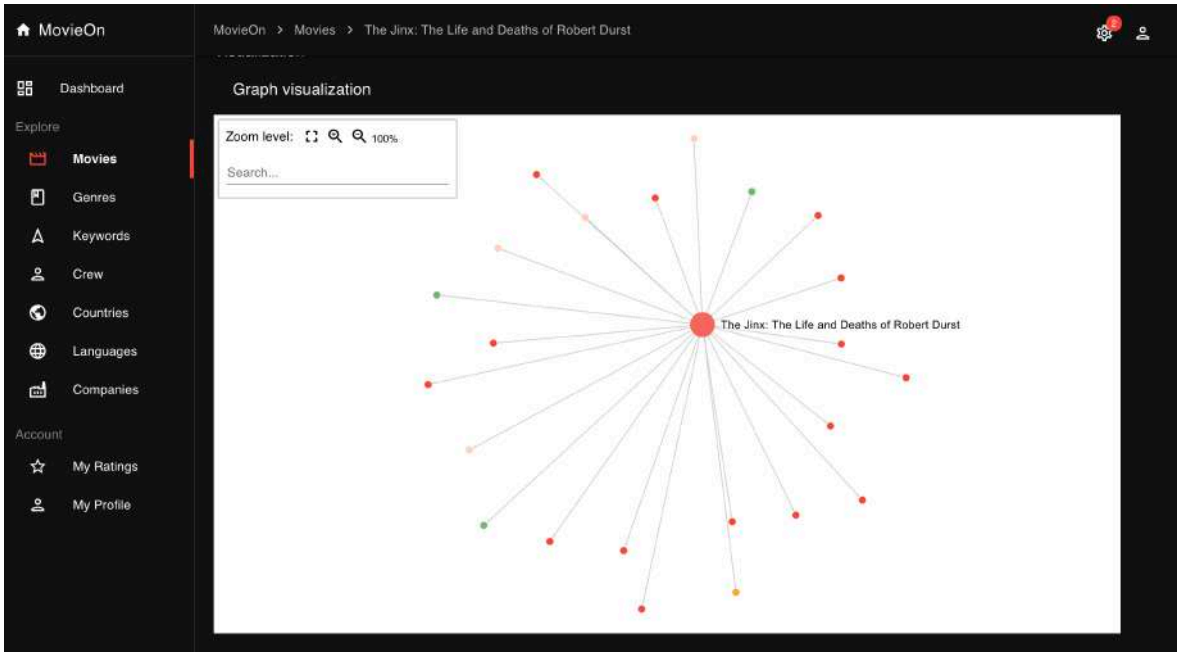

Figure 5.8: The neighborhood of the node that corresponds to a specific movie in the original graph.

It is noteworthy, that the platform has been designed in a way to simplify the access of each user to the predicted ratings of his. Therefore, users can easily see these ratings, by ”hovering” over the corresponding card of each movie. An example is visualized in figure 5.9, where the user is currently ”hovering” over the ”Ben-Hur” movie card.

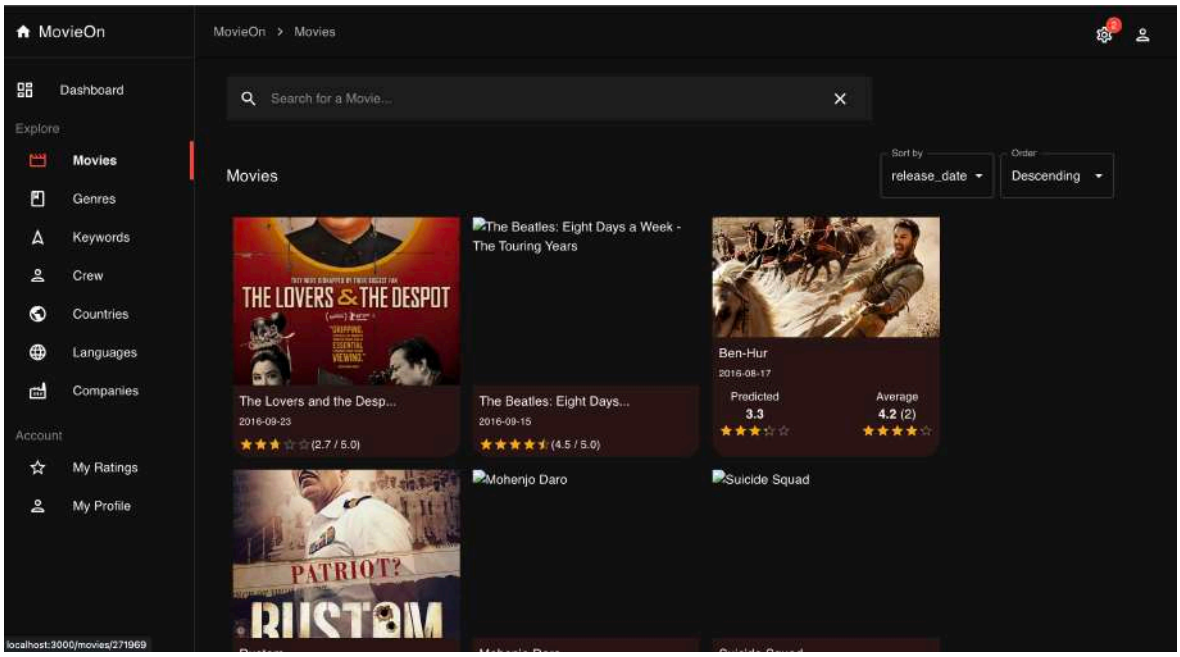

Figure 5.9: User is hovering over the ”Ben-Hur” card. As a result, he/she gets immediate access to the predicted rating for him on this movie, and the average rating of the movie.

Chapter 6

Experiments

6.1 Evaluation

The aim of this thesis was to build a movie recommender system that can accurately predict the ratings that users would give to movies. As reported in section 4.4, the metric used for the training and the evaluation of the recommender is the **Root Mean Squared Error (RMSE)**. This decision was motivated by the nature of the problem, which is a **regression task**, and by the examination of relevant **prior research** on the MovieLens (Grouplens) dataset.

6.2 Hyperparameters

One of the critical aspects of the experimental part of this thesis was the exploration of multiple hyperparameters, related to the architecture of the model, the generation of the node embeddings that encode the movies' content, and the version of the dataset. More specifically, the hyperparameters of the experimental part were divided into the following four main categories:

1. **Convolutional Graph Neural Network (GNN) architecture**: The architecture of the Graph Neural Network is a critical component of the system. It determines the exact computations followed in the message-passing process, specifying the functions that perform the aggregation of the neighbors' embeddings, and updating the current node's embeddings at each step. In our experiments, we explored four types of Convolutional Graph Neural Networks, including the **GraphSAGE** network (3.5.1), **K-GNNs** (3.11), **Graph Attention networks** (GAT 3.5.3), and **Graph Isomorphism Networks** (GIN 3.5.4).

2. **Node Embedding Algorithm**: Simultaneously, the predicted ratings can be highly affected by the quality of the node embeddings that encode the movies' content, prior to the usage of the GNN. For this reason, we experimented with the algorithms that can be utilized for embedding the movies. Three algorithms were used for generating node embeddings, including **FastRP** (3.2.5.6), **Node2Vec** (3.2.5.5), and **GraphSAGE** (3.5.1). Some sample visualizations of the generated node embeddings by these algorithms can be seen in the figures of section 4.2.7.

3. **Movies Content Subgraph Selection**: An additional important factor that affects the quality of the node embeddings about the movies' content, is the subgraph on which they were generated. The subgraph containing all the metadata on the content of the movies, consists of multiple nodes; Genres, Keywords, Production Companies, Production Countries, Languages, and People (Cast and Crew). To encode the content of a certain movie, we can utilize **multiple**

different subgraphs of this graph. For example, we can generate embeddings to encode only the genres, relying on the bipartite graph of movies and genres, or embeddings that encode the whole content of each movie, using the whole content subgraph. Taking this into consideration, we generated and utilized node embeddings encoding the content of the movies in the three following ways:

- Do not produce **any node embeddings** on the content of the movies. Utilize only the movies' titles and the ratings between users and movies. This methodology approaches the nature of **collaborative filtering** techniques.
- Generate multiple node embeddings encoding the content of each movie, with each embedding being generated on a different bipartite subgraph, that includes the different relationships of the movie. In other words, for each movie, generate a genres-embedding based on the movies-genres subgraph, a keywords-embedding based on the movies-keywords subgraph, etc.
- Generate one node embedding for each movie, that encodes its whole content on the graph. In this approach, we apply the node embedding algorithms on the whole subgraph.

4. **Model Structure**: The structure of the model relies on numerous hyperparameters. Many of them, including the number of **layers** and the **hidden channels** of the GNN Encoder and the Edge Decoder were explored.

6.3 Baseline

The baselines of the recommender system built in this thesis can be divided into two main categories. The first category involves **prior experiments** conducted on the MovieLens 100K dataset, that have already been reported in section 3.8. The second one involves a relatively simple version of the proposed architecture. This simple version uses **GraphSAGE** (3.5.1) as its GNN, and will be referred to as the **"SAGE baseline"**. By comparing the performance of our experiments to prior studies, we can evaluate the effectiveness of the proposed models and estimate the quality of our implementation. The SAGE Baseline serves as a useful reference for the more complex and optimized versions of the recommender system that are developed later in this thesis.

6.3.1 A simple GraphSAGE baseline

6.3.1.1 The model

The SAGE baseline is a relatively straightforward implementation of the proposed architecture, aiming to provide information on the basic functioning of the system, and to provide a starting point for further optimization of the numerous hyperparameters we introduced in the previous section.

More specifically, the **GNN Encoder** of the **SAGE baseline** consists of 4 GraphSAGE layers and 16 hidden channels. The **Edge Decoder** is an MLP (Multi-Layer Perceptron) of 5 layers and 16 hidden channels. As far as the selection of the **Movies Content Subgraph** is concerned, the baseline experiment uses only the movie titles as metadata input, without incorporating any node embeddings that encode the movie's metadata subgraph.

6.3.1.2 Baseline Experiment

For the baseline experiment, we trained the described GraphSAGE model on an 80%-10%-10% split of the dataset (train-validation-test sets) for 200 epochs. The RMSE values on the train and the validation sets for each epoch of the training are visualized in figures 6.1 and 6.2 respectively.

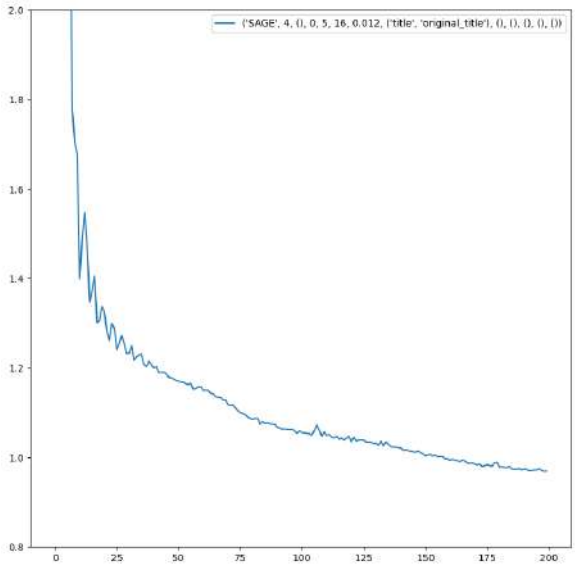

Figure 6.1: The training RMSE loss of the GraphSAGE baseline for each epoch. Visualized with matplotlib.

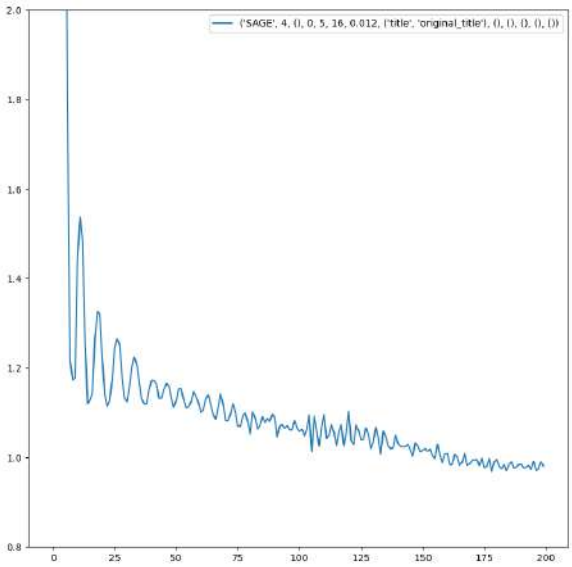

Figure 6.2: The validation RMSE loss of the GraphSAGE baseline for each epoch. Visualized with matplotlib.

The RMSE loss on the validation set reached a value of **0.97**. This value, along with the RMSE losses of the reported prior work, can be summarized in the table 6.1.

Table 6.1: RMSE baselines on the 100K datasets. The RMSE values on the MovieLens dataset are reported in the publication of Hekmatfar et al. (2022) and are run under the same experimental setup. It is important to note that our SAGE baseline cannot be directly compared to the other methods, due to the different experimental setup.

Model	Dataset (100K)	RMSE
GARec (Hekmatfar et al. (2022))	MovieLens	0.880
GC-MC (Berg et al. (2017))	MovieLens	0.905
IGMC (Zhang and Chen (2019))	MovieLens	0.927
PinSAGE (Ying et al. (2018))	MovieLens	0.962
SAGE baseline	The Movies Dataset	0.968

6.4 Further Experiments on Tuning the Hyperparameters

Subsequently to defining the baselines, we begin the process of conducting multiple experiments, in order to optimize the numerous hyperparameters of the task.

6.4.1 Initial GNN architectures comparison

The first experiment we performed, bearing mainly a diagnostic role, compares the predictions of the model for the different **Convolutional Graph Neural Network (GNN) architectures**. Keeping the same parameters as the baseline approach, we compare the four architectures, and the RMSE values of figures 6.3 and 6.4 are produced for the training and the validation set respectively. We observe that GraphSAGE and GIN achieve the lowest training losses, and that GraphSAGE achieves the lowest RMSE values on the validation set for the most epochs. The validation RMSE of the Graph Isomorphism Network is unstable during the epochs.

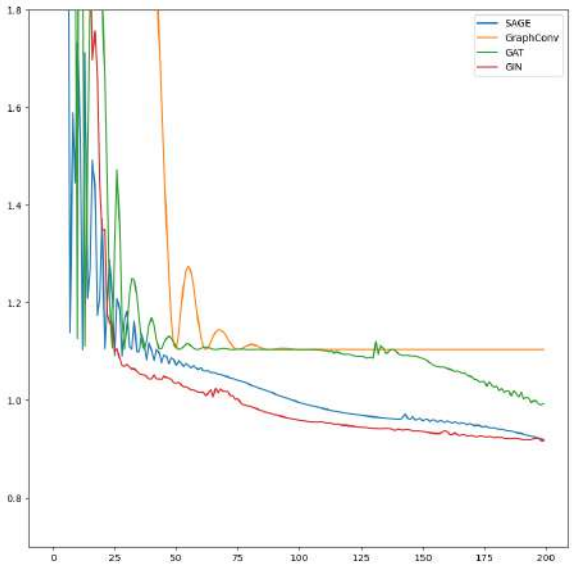

Figure 6.3: The training RMSE loss of the GraphSAGE, GraphConv, GAT, and GIN for each epoch. Visualized with matplotlib.

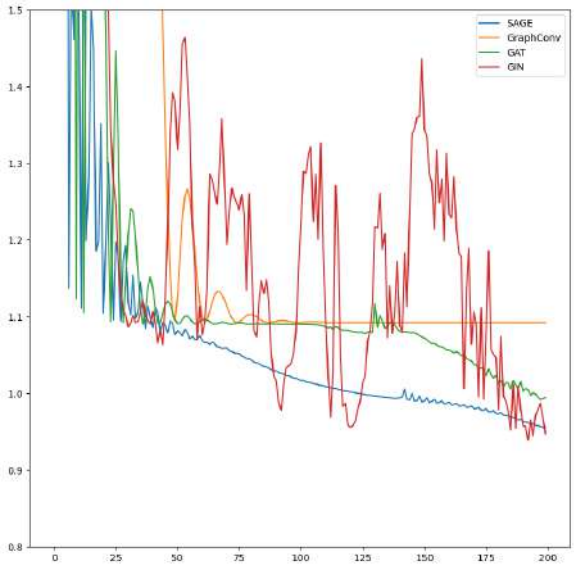

Figure 6.4: The validation RMSE loss of the GraphSAGE, GraphConv, GAT, and GIN for each training epoch. Visualized with matplotlib.

6.4.2 Number of layers

As a subsequent step in the experimental process, we conducted an exploration of the number of layers for the GNN Encoder and for the Edge Decoder. As far as the layers of the GNN Encoder are concerned, the existence of the oversmoothing problem that was analyzed in section 4.3.3.1, makes us expect that the optimal number of layers for the GNN Encoder will be relatively small. We perform multiple experiments for each GNN architecture separately, tuning the layers of the GNN Encoder and the Edge Decoder.

In general, to perform these experiments, a grid-search approach was followed. The number of layers for the GNN Encoder was varied between 2 and 6 in increment steps of 2, whereas the number of layers for the Edge Decoder was varied between 2 and 10 with steps of 4. For each combination, a separate instance was trained and evaluated by the RMSE loss on the same validation set.

6.4.2.1 Experiments without utilizing movie-content embeddings

In order to gain an intuition on the effection that the number of GNN layers has on the quality of the predictions, we conduct the experiments of figures 6.5 and 6.6 on a GraphSAGE architecture. We observe the existence of the oversmoothing phenomenon, as architectures with less hidden layers on the GNNEncoder achieve lower RMSE values on the validation set.

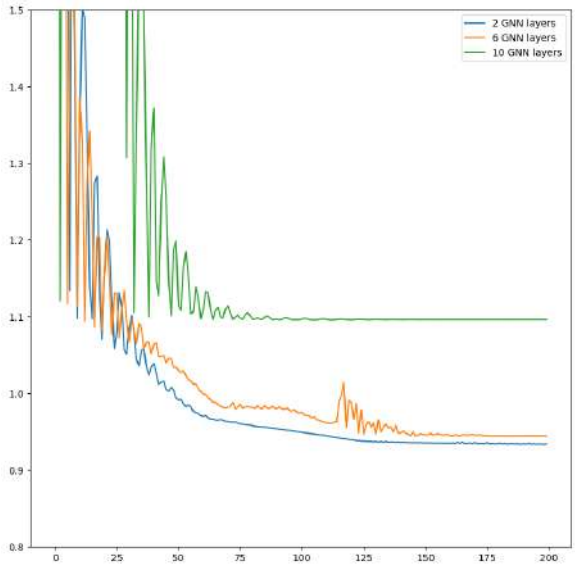

Figure 6.5: The validation RMSE loss of the GraphSAGE, with 2, 6, and 10 GNN layers, for each epoch. Visualized with matplotlib.

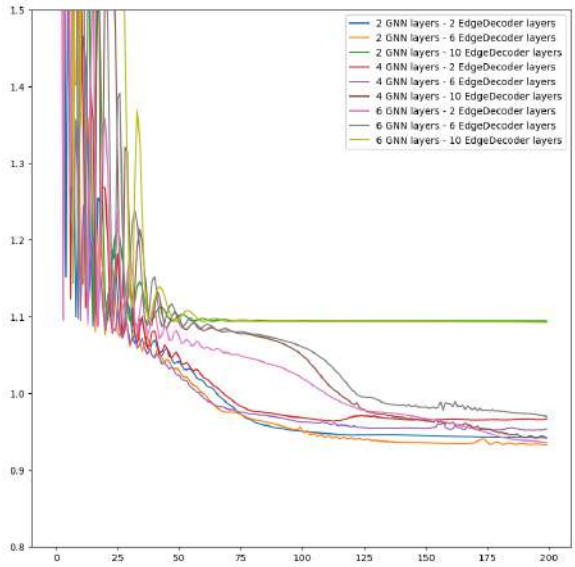

Figure 6.6: The validation RMSE loss of the GraphSAGE, with multiple combinations of layers, for each epoch. Visualized with matplotlib.

We conduct similar experiments for all the architectures of our experiments, with the produced RMSE values summarized in table 6.2. It is important to note, that the RMSE values achieved on the validation set after tuning the number of layers have already improved significantly, in comparison to the GraphSAGE baseline model.

Table 6.2: The layers combination with the lowest validation RMSE for each GNN architecture on the 100K dataset

Model	GNN Encoder layers	Edge Decoder layers	Validation RMSE
SAGE	2	6	0.933
GAT	2	10	0.929
GraphConv	2	10	1.050
GIN	2	6	0.937

6.4.2.2 Experiments utilizing movie-content embeddings

Next, we ensure that the layers number affects the predictions in the same way, even when utilizing the movie-content embeddings. For this purpose, we generate node embeddings for the movies, using the FastRP algorithm 3.2.5.6 provided by GDS (2022), based on the whole movie-content subgraph.

Following the same configuration of the experiments, at first we conduct experiments on the number of layers for the GraphSAGE network. The RMSE losses on the validation set are visualized in figures 6.7 and 6.8. Based on the visualized experiments, and on the same experiments on the other three GNN architectures, we observe that the same behaviour with the previous experiments group is followed. The RMSE losses achieved on the validation set are summarized in table 6.3.

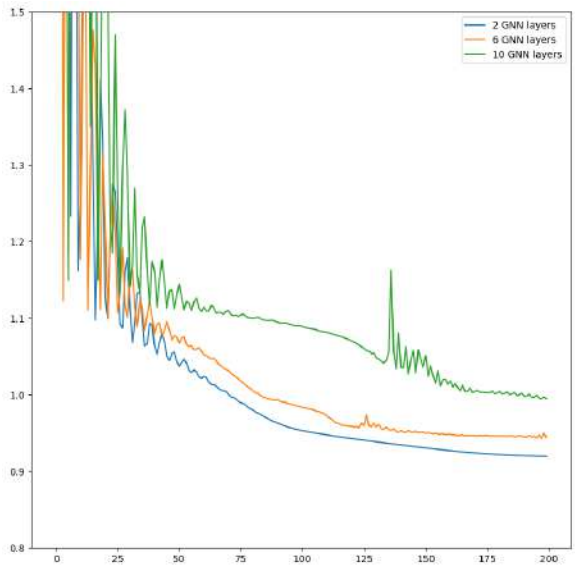

Figure 6.7: The validation RMSE loss of the GraphSAGE, with 2, 6, and 10 GNN layers, for each epoch, using FastRP node embeddings. Visualized with matplotlib.

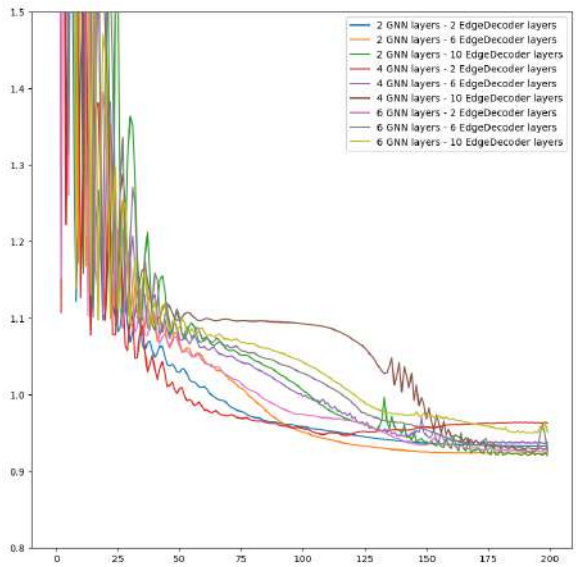

Figure 6.8: The validation RMSE loss of the GraphSAGE, with multiple combinations of layers, for each epoch, using FastRP node embeddings. Visualized with matplotlib.

We conduct similar experiments for all the architectures of our experiments, with the produced RMSE values summarized in table 6.3.

Table 6.3: The layers combination with the lowest validation RMSE for each GNN architecture on the 100K dataset, utilizing the FastRP node embeddings for the movies-content

Model	GNN Encoder layers	Edge Decoder layers	Validation RMSE
SAGE	2	10	0.920
GAT	2	10	0.941
GraphConv	2	10	0.972
GIN	2	6	0.933

6.4.3 Number of hidden channels

The next step of the experimental process was to try different numbers of hidden channels for the GNN Encoder and the Edge Decoder, bearing in mind the ways that the movie-content embeddings can be utilized. For the two edge cases of the embeddings utilization, we perform a search over the possible hidden channels number, in the range of [16, 128]. For each GNN architecture, we use the layers number that achieved the minimum RMSE loss on the validation set, during the previous group of experiments.

6.4.3.1 Experiments without utilizing movie-content embeddings

We create a different instance for each one of the models in table 6.2, with a variable number of hidden channels. The conducted experiments for GraphSAGE and GAT are visualized in figures 6.9 and 6.10 respectively.

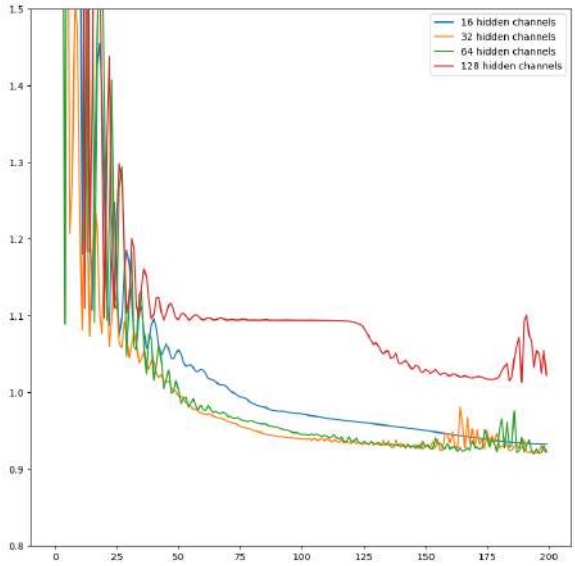

Figure 6.9: The validation RMSE loss of the GraphSAGE, with 2 GNN layers, 6 layers in the Edge Decoder, and variable number of hidden channels, without using movie-content embeddings. Visualized with matplotlib.

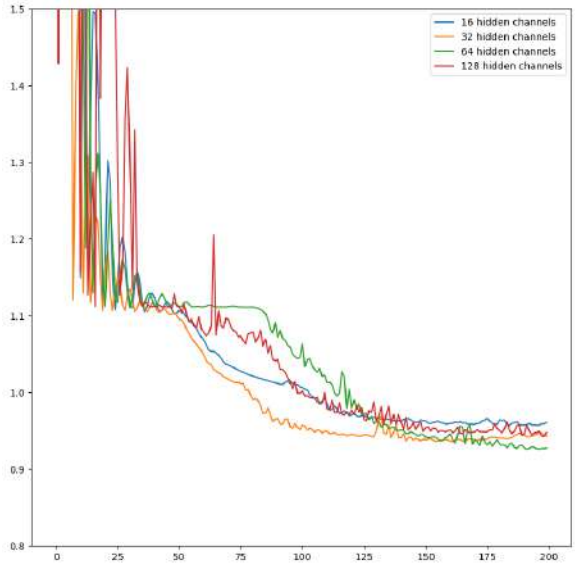

Figure 6.10: The validation RMSE loss of GAT, with 2 GNN layers, 10 layers in the Edge Decoder, and variable number of hidden channels, without using movie-content embeddings. Visualized with matplotlib.

We conduct similar experiments for all the models of table 6.2. The best version of each model, based on the validation RMSE, is summarized in table 6.4.

Table 6.4: The layers and hidden channels combination with the lowest validation RMSE for each GNN architecture on the 100K dataset, without utilizing node embeddings for the movies-content

Model	GNN Encoder layers	Edge Decoder layers	Hidden Channels	Validation RMSE
SAGE	2	6	64	0.919
GAT	2	10	64	0.926
GraphConv	2	10	128	1.018
GIN	2	6	16	0.934

6.4.3.2 Experiments utilizing movie-content embeddings

We repeat the experiments of section 6.4.3.1, but now tuning the number of hidden channels on the models of table 6.3. The experiments for GraphSAGE and GAT are visualized in figures ?? and ??. The resulting RMSE losses on the validation set for each experiment, are gathered in table 6.5.

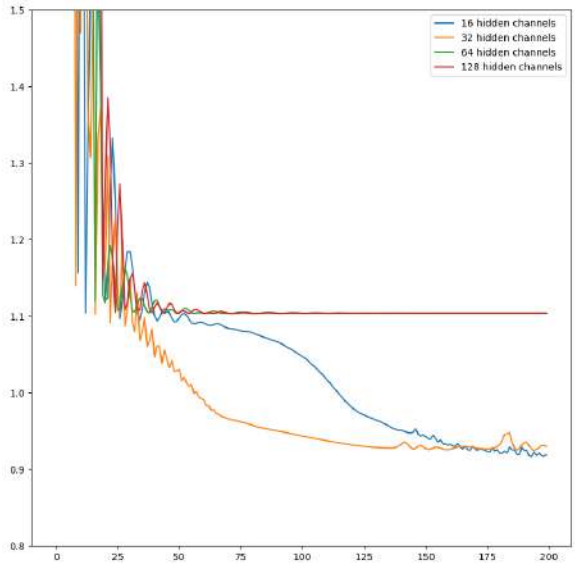

Figure 6.11: The validation RMSE loss of the GraphSAGE, with 2 GNN layers, 10 layers in the Edge Decoder, and variable number of hidden channels, using movie-content embeddings. Visualized with matplotlib.

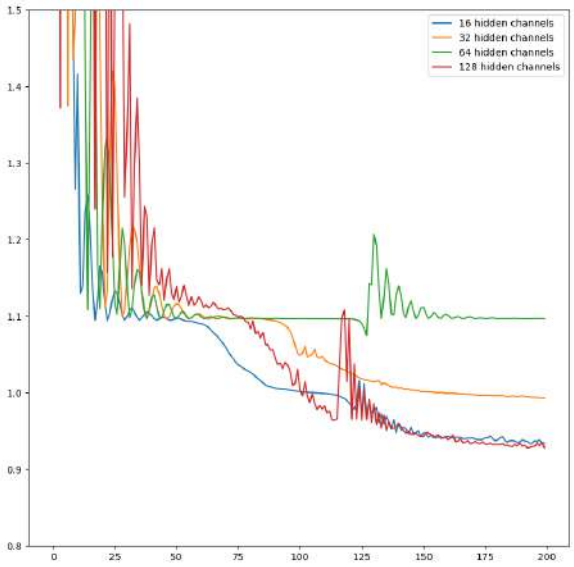

Figure 6.12: The validation RMSE loss of GAT, with 2 GNN layers, 10 layers in the Edge Decoder, and variable number of hidden channels, with using movie-content embeddings. Visualized with matplotlib.

Table 6.5: The layers and hidden channels combination with the lowest validation RMSE for each GNN architecture on the 100K dataset, utilizing FastRP node embeddings for the movies-content

Model	GNN Encoder layers	Edge Decoder layers	Hidden Channels	Validation RMSE
SAGE	**2**	**10**	**16**	**0.919**
GAT	2	10	128	0.928
GraphConv	2	10	16	0.972
GIN	2	6	32	0.930

6.4.4 Node embeddings usage

As the next step of our experimental process, we conduct experiments on node embeddings usage. More specifically, for each one of the models of table 6.4, we compare the RMSE losses on the validation set, if we provide different combinations and types of node embeddings. At first, we explore FastRP, GraphSAGE, and Node2Vec embeddings on the movies-content, using either the whole content subgraph to generate them, or a separated bipartite graph for each related entity. The validation loss on these experiments is visualized for GraphSAGE and for GAT on figures 6.13 and 6.14.

We conduct the same experiments on rest of the models of table 6.5, and we summarize the experiment that achieved the lowest validation loss for each architecture, on table 6.6. We refer to the node embeddings generated by the **FastRP** algorithm on the whole movies-content subgraph as **FastRP combined** embeddings, and to the set of node embeddings generated by the **FastRP** algorithm on the multiple bipartite subgraphs of the whole movies-content graph as **FastRP separate** embeddings.

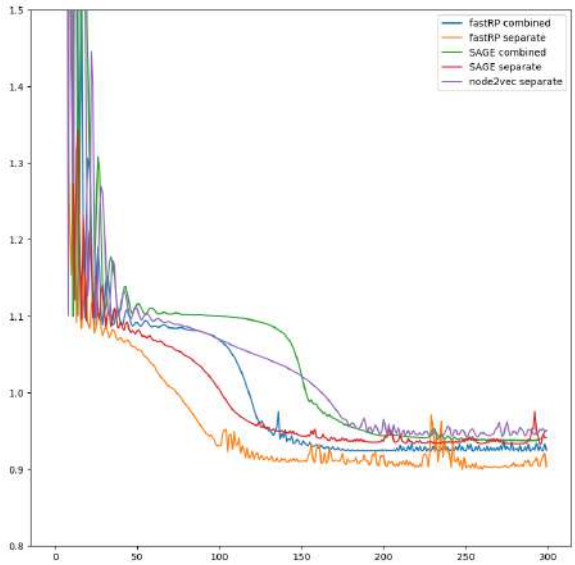

Figure 6.13: The validation RMSE loss of the GraphSAGE, with 2 GNN layers, 10 layers in the Edge Decoder, and 16 hidden channels, using a variety of movie-content embeddings. Visualized with matplotlib.

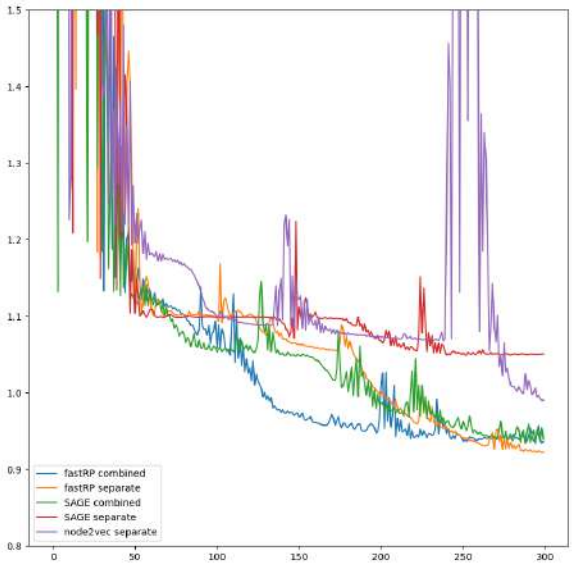

Figure 6.14: The validation RMSE loss of GAT, with 2 GNN layers, 10 layers in the Edge Decoder, and 128 hidden channels, using a variety of movie-content embeddings. Visualized with matplotlib.

Table 6.6: The layers, hidden channels, and node embeddings usage combination with the lowest validation RMSE for each GNN architecture on the 100K dataset, after 200 training epochs

Model	GNN layers	Edge Decoder layers	Hidden Channels	Node embeddings	Val. RMSE
SAGE	**2**	**10**	**16**	**FastRP separate**	**0.903**
GAT	2	10	128	FastRP separate	0.922
GraphConv	2	10	16	FastRP combined	0.972
GIN	2	6	32	FastRP combined	0.923

We observe that every GNN architecture achieved lower validation loss with the utilization of the movies-content node embeddings. The results can be summarized in figure 6.15.

6.4.5 Final comparison

The model that achieved the lowest RMSE value on the validation set consists of a two-layered GNNEncoder, utilizing layers following the GraphSAGE architecture, and an EdgeDecoder with 10 layers. The movie-content is utilized with node embeddings produced by the FastRP algorithm, and mulitple separate node embeddings are generated to encode the content of each movie. Our model achieves a RMSE of **0.904** on the test set. Although this result cannot be directly compared to the RMSE values that the previously reported graph-based state-of-the-art techniques achieve on MovieLens 100K, due to the different experimental setup, we summarize the results, for intuitive reasons, in table 6.7.

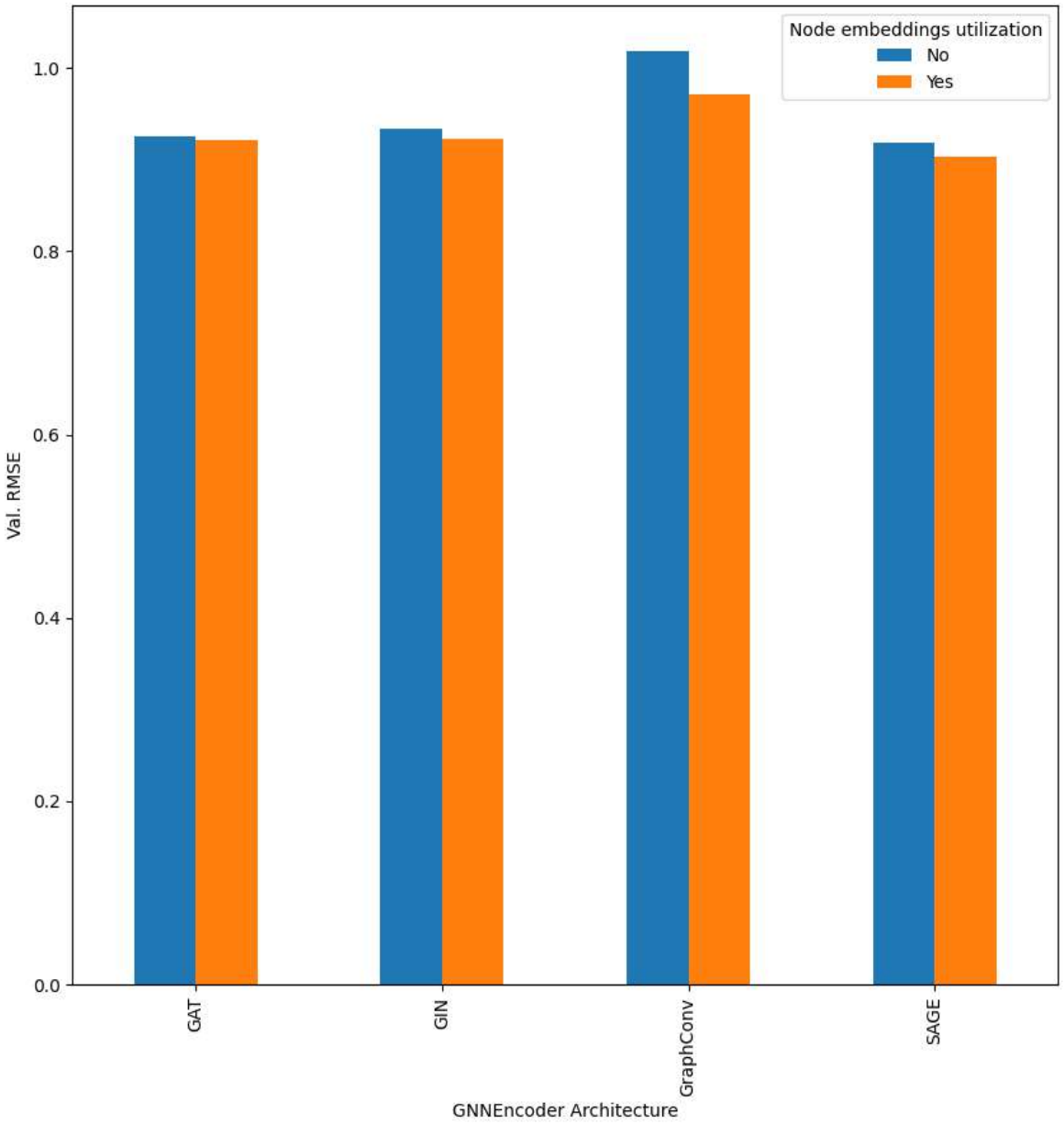

Figure 6.15: The effect of utilizing the movies-content node embeddings on the validation RMSE loss for the four GNN architectures. Visualized with matplotlib.

6.5 Predicting the ratings for a specific user

Subsequently to the process of tuning the multiple hyperparameters to achieve the wanted performance on our task, we proceed to some more practical examples, to validate that the recommender will conduct useful predictions for the platform's users. For the following experiments, we utilized the GraphSAGE model of table 6.6.

6.5.0.1 Differentiating users

The aim of the first experiment was to confirm that users whose submitted ratings follow different distributions, will receive the corresponding predicted ratings. For this purpose, we picked the users

Table 6.7: RMSE baselines and our final result on the 100K datasets. The RMSE values on the MovieLens dataset are reported in the publication of Hekmatfar et al. (2022) and are run under the same experimental setup. It is important to note that our best SAGE model cannot be directly compared to the other methods, due to the different experimental setup.

Model	Dataset (100K)	RMSE
GARec (Hekmatfar et al. (2022))	MovieLens	0.880
SAGE (under our hyperparameters)	The Movies Dataset	0.904
GC-MC (Berg et al. (2017))	MovieLens	0.905
IGMC (Zhang and Chen (2019))	MovieLens	0.927
PinSAGE (Ying et al. (2018))	MovieLens	0.962

empathicRelish3 and *puzzledSalt7* of the platform, and examined the distributions of their submitted and predicted ratings. As seen in figure 6.16, the majority of the first user's ratings tend to be in the range [2, 3.5], and the predicted ratings are lying in the same area. In contrast, as seen in figure 6.17, the second user tends to submit high ratings constantly. As expected, the predicted ratings for this user are greater than 4.0. The conclusion of this experiment is that indeed each user receives personalized recommendations, according to his/her part interaction with the movies.

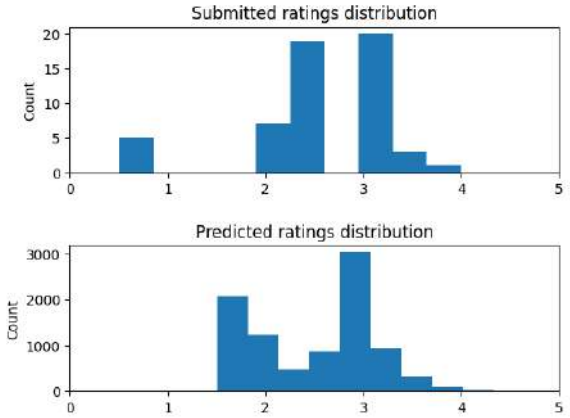

Figure 6.16: The distribution of the ratings that user *empathicRelish3* has submitted (first diagram), as long as the distribution of the predicted ratings for the specific user.

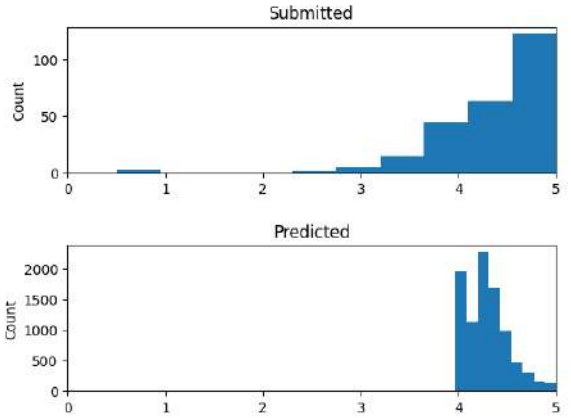

Figure 6.17: The distribution of the ratings that user *puzzledSalt7* has submitted (first diagram), as long as the distribution of the predicted ratings for the specific user.

6.5.0.2 Adjusting to changes in user behavior over time

An additional practical aspect of the platform, given its nature, is the ability to adapt to changes in user behavior over time. The current experiment demonstrates the model's capacity to keep up with evolving preferences of users, ensuring that the recommendations it provides remain relevant and useful for the platform's users.

More specifically, in the current experiment, we randomly chose a user that used to submit high ratings. The initial distribution of the user's ratings, as long as the distribution of the ratings our model predicts, is visualized in figure 6.18. As expected, the model predicts high ratings for the

specific user. However, let's suppose that the user suddenly decides to change his rating behavior, and starts submitting multiple ratings with relatively low values, mainly in the range of [0.5, 2.5]. The distribution of the user's submitted ratings has now adapted to the distribution of figure 6.19. After conducting a re-training on the model, including the new data, we observe that the distribution of the ratings for the specific user has started shifting towards lower values, adapting to his/her new behavior.

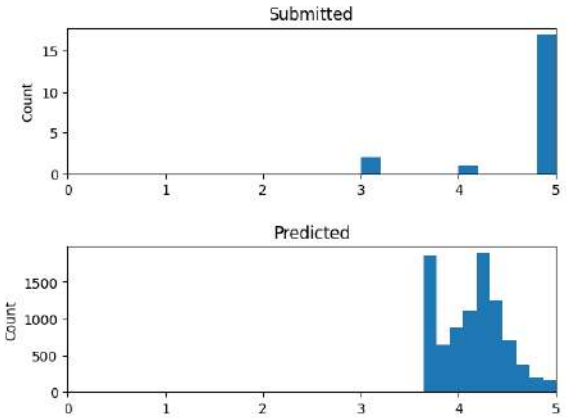

Figure 6.18: The distribution of the ratings that a specific user has submitted (1st diagram), as long as the distribution of the predicted ratings for the specific user.

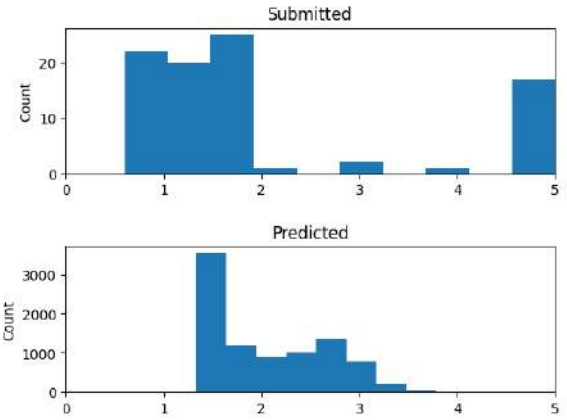

Figure 6.19: The distribution of the submitted ratings (1st diagram) after a change in his/her rating behavior, as long as the predicted ratings, after retraining the model.

6.5.0.3 Understanding more complex user tastes

In the experiments that have been performed in the previous sections, we can observe that the ratings of each user are mainly lying in a single area of values. In the current experiment, we mimic the behavior of user *dearMagpie8* that submits ratings in a wide range, rating both the movies that he/she likes, as long as the movie he/she dislikes.

More specifically, we programmatically mimic the behavior of a user that likes movies of genre *Crime* by rating them in the range [3.0, 4.5], and dislikes the movies of the genre *Adventure*, by rating them in the range [0.5, 2.0]. The distribution formed by the ratings of the certain user is more complex than the previous ones, and we expect that the model would face difficulty in predicting the correct ratings. The distributions of the total submitted and predicted ratings for *dearMagpie8* are visualized in figure 6.20. The predictions tend to be separated into two clusters, and therefore tend to follow the distribution of the submitted ratings. In figure 6.21, we visualize the predicted ratings for the movies that belong to each one of the two genres. We observe that most of the *Crime* movies are rated higher than the *Adventure* movies, and we conclude that the recommender tends to identify that the user *dearMpagpie8* prefers the first genre.

However, in general, this experiment shows that our model cannot predict the ratings of more complex users, with the same efficiency that it can predict the ratings of simpler users. For this reason, we conduct a sampling over multiple users of our dataset, and we observe that most of these users tend to have a simple rating behavior. Therefore, bearing in mind the relatively small version

of our dataset, we can reach the conclusion that our model was not trained for a sufficient amount of users with such complex rating behaviors, and as a result, it cannot predict their ratings in a satisfying way.

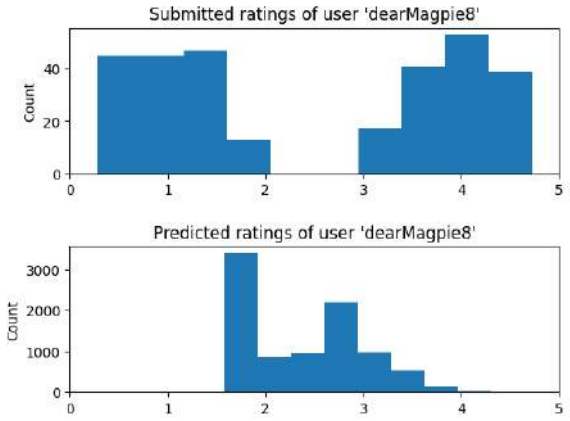

Figure 6.20: The distribution of the ratings that a specific user has submitted (1st diagram), as long as the distribution of the predicted ratings for the specific user.

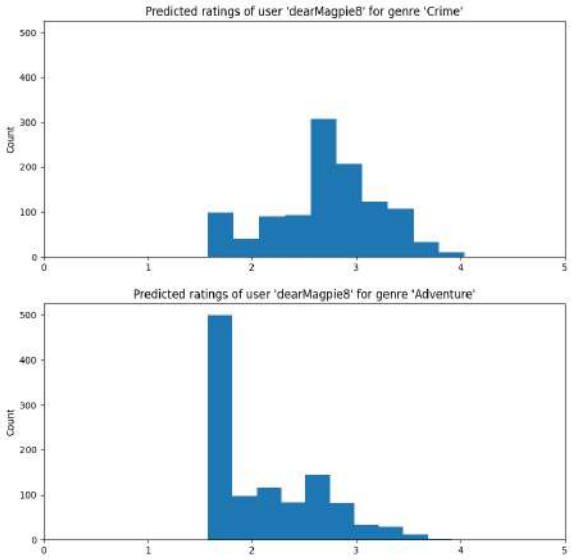

Figure 6.21: The distribution of the predicted ratings for genres Crime and Adventure. The recommender tends to understand the user's preference towards the first genre.

Chapter 7

Conclusion

7.1 General Conclusion

In this work, we explored the usage of **Graph Neural Networks** to perform **movie recommendations**. We modeled the recommendation problem as the **link weight prediction** task on the bipartite graph consisting of users and movies as nodes, and the corresponding ratings as weighted edges.

As far as the usage of Neo4j (Neo4j (2012)) as our **graph database** is concerned, we conclude that this decision proved to be a critical factor in achieving the desired results. Neo4j enabled us to store the dataset natively as a graph and perform a variety of **graph algorithms** on it, to uncover various insights into the complex connectivity of the entities, with the Neo4j Graph Data Science Library (GDS (2022)). This library allowed us to take advantage of the underlying graph structure, and extract **node embeddings**, using a variety of implemented graph algorithms. The database management process required experience in back-end development and strong scripting skills, but the high-level API of Neo4j, as long as the existing libraries, such as Py2Neo (Py2neo (2020)), made the database initialization process efficient and customizable. Therefore, it enabled us to transform the initial CSV files of the dataset into the required infrastructure to support the development of a machine learning model and a whole web app, with the minimum possible development overhead.

On the development of the recommender model, the framework Pytorch Geometric (Paszke et al. (2019)) allowed us to easily bring to life a variety of **GNN architectures**, customize them, and perform numerous **experiments** to identify the solution that performed better in our problem. We implemented a widely used architecture that decouples the process of generating node embeddings from predicting the corresponding link weights. Our best model, which utilized the GraphSAGE (Hamilton et al. (2017a)) GNN architecture, reached an RMSE of **0.903** on the 100K version of the **The Movies Dataset** (Banik (2017)), which is compared to the RMSE that multiple state-of-the-art models achieve on the corresponding MovieLens (Grouplens) dataset.

The development of the web app was a crucial step in ensuring that our model could be tested and used in a **real-world** scenario. Our focus on security and scalability made sure that the system was designed to handle the demands of a production environment. The **integration** of the recommendations model into the web app was a complex process that required expertise in both back-end and front-end development. This process not only allowed us to test our model but also opened up opportunities for expanding the dataset. Additionally, it served as a gentle introduction to the field of **MLOps**, which is an important area of research in the field of machine learning.

7.2 Future work

As expected, the research around recommender systems, as long as the development of a whole platform that encapsulates this functionality, are processes that cannot fit in a single diploma thesis.

There is a variety of future steps that could be performed in the research around utilizing GNNs for the development of movie recommender systems. In our context, we plan to perform research on the following fields:

- **Dataset versions** The experiments that were showcased were performed on the small version of the **The Movies Dataset** (Banik (2017)), which consists of 100K ratings. We consider it an interesting step to verify that the conclusions of our experiments will be similar on larger versions of the dataset.

- **GNN Architectures** In our experiments, the architectures GraphSAGE and GAT seemed to outperform GIN and GraphConv on the achieved RMSE values. We plan to perform more extensive experiments on more GNN architectures, to validate whether the are versions of our model that can achieve even better results.

- **Node embedding algorithms** In our experiments, we investigated the efficacy of three algorithms to generate node embeddings on the metadata of the movies. As a future direction, two avenues of exploration are worth considering. Firstly, an in-depth analysis could be conducted on tuning the numerous hyperparameters of these algorithms. Secondly, it would be interesting to explore alternative algorithms besides FastRP, Node2Vec, and GraphSAGE. This way, we might empower our recommendation engine, by capturing the underlying graph structure in a more expressive way.

- **Explainability** In recent years, there has been a growing interest in understanding the decisions made by machine learning models. We plan to explore the explainability of our GNN-based recommendations model, in order to understand the reasons behind its predictions and increase the trust our users show toward our recommendations.

Simultaneously, numerous actions can be performed, to result in a more robust and efficient platform. More specifically, we consider the following steps would help in this direction:

- **Deployment** We consider the deployment of our platform as an important step to verify the quality of our predictions and the system's implementation. Making the platform publicly available, will lead to an increase in active users, enriching the dataset with new ratings. Therefore, a new extended version of the dataset could be created and made publicly available as a contribution to the open-source community. Concurrently, the recommendations model could be tested in a scenario closer to the real world.

- **Model re-training** We plan to investigate the optimal frequency of re-training the recommendations model, considering the trade-off between the cost of re-training, and the need to keep the recommendations up-to-date with new users and changes in the behavior of our long-term users

- **Graph refresh** Another important aspect to explore is the frequency of refreshing the in-memory graph on the Model API. This will help ensure that the model's predictions are up-to-date with the latest changes in the user-movie interactions.

Bibliography

Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. ISSN 0022-0000. doi: https://doi.org/10.1016/S0022-0000(03)00025-4. URL https://www.sciencedirect.com/science/article/pii/S0022000003000254. Special Issue on PODS 2001.

G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005. doi: 10.1109/TKDE.2005.99.

Rabaa Alabdulrahman and Herna Viktor. Catering for unique tastes: Targeting grey-sheep users recommender systems through one-class machine learning. *Expert Systems with Applications*, 166:114061, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2020.114061. URL https://www.sciencedirect.com/science/article/pii/S0957417420308241.

Renzo Angles. The property graph database model. In *Alberto Mendelzon Workshop on Foundations of Data Management*, 2018.

Rounak Banik. The movies dataset, 2017. URL https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset.

Sami Belkacem. *Machine learning approaches to rank news feed updates on social media.* PhD thesis, 04 2021.

Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion, 2017. URL https://arxiv.org/abs/1706.02263.

S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424(4):175–308, 2006. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2005.10.009. URL https://www.sciencedirect.com/science/article/pii/S037015730500462X.

J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* Elsevier, New York, 1976.

Michael Bronstein. Expressive power of graph neural networks and the weisfeiler-lehman test.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4), 2017. doi: 10.1109/MSP.2017.2693418.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3438–3445, Apr. 2020. doi: 10.1609/aaai.v34i04.5747. URL https://ojs.aaai.org/index.php/AAAI/article/view/5747.

Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. Fast and accurate network embeddings via very sparse random projection, 2019. URL https://arxiv.org/abs/1908.11512.

Christuniversity. URL https://sites.google.com/a/cs.christuniversity.in/discrete-mathematics-lectures/graphs/directed-and-undirected-graph.

Microsoft Corporation. TypeScript, 2012. URL https://www.typescriptlang.org/.

Zahra Zamanzadeh Darban and Mohammad Hadi Valipour. GHRS: Graph-based hybrid recommendation system with application to movie recommendation. *Expert Systems with Applications*, 200: 116850, aug 2022. doi: 10.1016/j.eswa.2022.116850. URL https://doi.org/10.1016%2Fj.eswa.2022.116850.

Tommaso Di Noia and Vito Ostuni. *Recommender Systems and Linked Open Data*, volume 9203, pages 88–113. 07 2015. ISBN 978-3-319-21767-3. doi: 10.1007/978-3-319-21768-0_4.

Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg; New York, fourth edition, 2010. ISBN 9783642142789 3642142788 9783642142796 3642142796.

Minh-Phung Do, Dung Nguyen, and Loc Nguyen. Model-based approach for collaborative filtering. 08 2010.

Paula Gómez Duran, Alexandros Karatzoglou, Jordi Vitrià, Xin Xin, and Ioannis Arapakis. Graph convolutional embeddings for recommender systems, 2021. URL https://arxiv.org/abs/2103.03587.

Gintare Karolina Dziugaite and Daniel M. Roy. Neural network matrix factorization, 2015. URL https://arxiv.org/abs/1511.06443.

Brendan Eich. Javascript, 1995.

Facebook. React. https://reactjs.org/, 2013.

Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, 5 2019. URL https://github.com/pyg-team/pytorch_geometric.

The Node.js Foundation. Node.js, 2009. URL https://nodejs.org.

Neo4j GDS. Neo4j graph data science library, 2022. URL https://neo4j.com/docs/graph-data-science/current/.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017. URL https://arxiv.org/abs/1704.01212.

Jennifer Golbeck. Chapter 3 - network structure and measures. In Jennifer Golbeck, editor, *Analyzing the Social Web*, pages 25–44. Morgan Kaufmann, Boston, 2013. ISBN 978-0-12-405531-5. doi: https://doi.org/10.1016/B978-0-12-405531-5.00003-1. URL https://www.sciencedirect.com/science/article/pii/B9780124055315000031.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, jul 2018. doi: 10.1016/j.knosys.2018.03.022. URL https://doi.org/10.1016%2Fj.knosys.2018.03.022.

Grouplens. Movielens latest datasets. URL https://grouplens.org/datasets/movielens/.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016. URL https://arxiv.org/abs/1607.00653.

William Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14:1–159, 09 2020. doi: 10.2200/S01045ED1V01Y202009AIM046.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017a. URL https://arxiv.org/abs/1706.02216.

William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications, 2017b. URL https://arxiv.org/abs/1709.05584.

Soyeon Caren Han, Taejun Lim, Siqu Long, Bernd Burgstaller, and Josiah Poon. GLocal-k. In *Proceedings of the 30th ACM International Conference on Information &amp Knowledge Management*. ACM, oct 2021. doi: 10.1145/3459637.3482112. URL https://doi.org/10.1145%2F3459637.3482112.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation, 2020. URL https://arxiv.org/abs/2002.02126.

Taher Hekmatfar, Saman Haratizadeh, Parsa Razban, and Sama Goliaei. Attention-based recommendation on graphs, 2022. URL https://arxiv.org/abs/2201.05499.

Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983. ISSN 0378-8733. doi: https://doi.org/10.1016/0378-8733(83)90021-7. URL https://www.sciencedirect.com/science/article/pii/0378873383900217.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

Yuchen Hou and Lawrence B. Holder. Deep learning approach to link weight prediction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017. doi: 10.1109/IJCNN.2017.7966076.

Yuchen Hou and Lawrence B. Holder. Link weight prediction with node embeddings, 2018. URL https://openreview.net/forum?id=ryZ3KCy0W.

Binxuan Huang and Kathleen M. Carley. Residual or gate? towards deeper graph neural networks for inductive graph representation learning, 2019. URL https://arxiv.org/abs/1904.08035.

Ningyuan Teresa Huang and Soledad Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, jun 2021. doi: 10.1109/icassp39728.2021.9413523. URL https://doi.org/10.1109%2Ficassp39728.2021.9413523.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016a. URL https://arxiv.org/abs/1609.02907.

Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016b. URL https://arxiv.org/abs/1611.07308.

Adrien Leman. The reduction of a graph to canonical form and the algebra which appears therein. 2018.

Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns?, 2019. URL https://arxiv.org/abs/1904.03751.

Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 287–296, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150436. URL https://doi.org/10.1145/1150402.1150436.

Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. Semantic object parsing with graph lstm, 2016. URL https://arxiv.org/abs/1603.07063.

L. Lovász. Random walks on graphs: A survey. In D. Miklós, V. T. Sós, and T. Szőnyi, editors, *Combinatorics, Paul Erdős is Eighty*, volume 2, pages 353–398. János Bolyai Mathematical Society, Budapest, 1996.

matplotlib. Matplotlib: Python plotting. https://matplotlib.org. [Online; accessed 2022].

Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

Peter Meltzer, Marcelo Daniel Gutierrez Mallea, and Peter J. Bentley. Pinet: A permutation invariant graph neural network for graph classification, 2019. URL https://arxiv.org/abs/1905.03046.

Prem Melville and Vikas Sindhwani. *Recommender Systems*, pages 1056–1066. Springer US, Boston, MA, 2017. ISBN 978-1-4899-7687-1. doi: 10.1007/978-1-4899-7687-1_964. URL https://doi.org/10.1007/978-1-4899-7687-1_964.

Mohseni Milad, Razavi. URL https://commons.wikimedia.org/w/index.php?curid=48581824.

Federico Monti, Michael M. Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks, 2017. URL https://arxiv.org/abs/1704.06803.

Federico Monti, Alberto Bressan, Alessio Bronzin, and Dario Malchiodi. Graph attention networks for link prediction in recommender systems. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2397–2406. ACM, 2019.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2018. URL https://arxiv.org/abs/1810.02244.

Neo4j. Neo4j - the world's leading graph database, 2012. URL http://neo4j.org/.

Inc. Neo4j. Neo4j desktop, 2021. URL https://neo4j.com/download/.

Ignavier Ng, Shengyu Zhu, Zhitang Chen, and Zhuangyan Fang. A graph autoencoder approach to causal structure learning, 2019. URL https://arxiv.org/abs/1911.07420.

Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027, nov 2021. doi: 10.1613/jair.1.13225. URL https://doi.org/10.1613%2Fjair.1.13225.

Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. URL https://arxiv.org/abs/1511.08458.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, aug 2014. doi: 10.1145/2623330.2623732. URL https://doi.org/10.1145%2F2623330.2623732.

Natasa Przulj, Derek G. Corneil, and Igor Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20 18:3508–15, 2004.

Py2neo. Py2neo, 2020. URL https://pypi.org/project/py2neo/.

Software Foundation Python. Python programming language. https://www.python.org/.

Sandeep K. Raghuwanshi and R. K. Pateriya. Recommendation systems: Techniques, challenges, application, and evaluation. In Jagdish Chand Bansal, Kedar Nath Das, Atulya Nagar, Kusum Deep, and Akshay Kumar Ojha, editors, *Soft Computing for Problem Solving*, Singapore, 2019. Springer Singapore. ISBN 978-981-13-1595-4.

Ahmed Rashed, Josif Grabocka, and Lars Schmidt-Thieme. Attribute-aware non-linear co-embeddings of graph features. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 314–321, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3346999. URL https://doi.org/10.1145/3298689.3346999.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019. URL https://arxiv.org/abs/1908.10084.

Claudio Rocchini. *By Claudio Rocchini - Own work, CC BY 2.5.* URL https://commons.wikimedia.org/w/index.php?curid=1988980.

Håkon Drolsum Røkenes. Graph-based natural language processing: Graph edit distance applied to the task of detecting plagiarism. 2012.

Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. doi: 10.23915/distill.00033. https://distill.pub/2021/gnn-intro.

Anida Sarajlić, Noël Malod-Dognin, Ömer Yaveroğlu, and Natasa Przulj. Graphlet-based characterization of directed networks. *Scientific Reports*, 6:35098, 10 2016. doi: 10.1038/srep35098.

Ryoma Sato. A survey on the expressive power of graph neural networks, 2020. URL https://arxiv.org/abs/2003.04078.

Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. *arXiv preprint arXiv:1511.07939*, 2015.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011. URL http://dblp.uni-trier.de/db/journals/jmlr/jmlr12.html#ShervashidzeSLMB11.

Thiago C. Silva and Liang Zhao. Semi-supervised learning guided by the modularity measure in complex networks. *Neurocomputing*, 78, 2012.

SoftwareHelpingTest. URL https://www.softwaretestinghelp.com/java-graph-tutorial/.

University Stanford. Stanford cs224w: Machine learning with graphs, 2021. URL http://web.stanford.edu/class/cs224w/.

tannerlinsley. React-query. https://github.com/tannerlinsley/react-query, 2020.

Material-UI Team. Material-ui. https://github.com/mui-org/material-ui, 2020.

Pallets Team. Flask. https://flask.palletsprojects.com, 2010.

John W. Tukey. On the Comparative Anatomy of Transformations. *The Annals of Mathematical Statistics*, 28(3):602 – 632, 1957. doi: 10.1214/aoms/1177706875. URL https://doi.org/10.1214/aoms/1177706875.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017. URL https://arxiv.org/abs/1710.10903.

S.S. Vempala. *The Random Projection Method*. DIMACS Series. American Mathematical Soc. ISBN 9780821871072. URL https://books.google.gr/books?id=L5L2J0UEY4QC.

Lilapati Waikhom and Ripon Patgiri. Graph neural networks: Methods, applications, and opportunities, 2021. URL https://arxiv.org/abs/2108.10733.

Xiang Wang, Yifei Chen, Huan Liu, Chuan Li, Xiaoyan Zhu, and Wenwu Ma. Neural graph collaborative filtering. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983. ACM, 2019.

Wikipedia. a. URL https://commons.wikimedia.org/w/index.php?curid=1347423.

Wikipedia. b. URL https://commons.wikimedia.org/w/index.php?curid=1347424.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021. doi: 10.1109/tnnls.2020.2978386. URL https://doi.org/10.1109%2Ftnnls.2020.2978386.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2018. URL https://arxiv.org/abs/1810.00826.

Wikipedia Yepke. URL http://nl.wikipedia.org/,CCBY-SA3.0,https://commons.wikimedia.org/w/index.php?curid=18460150.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM, jul 2018. doi: 10.1145/3219819.3219890. URL https://doi.org/10.1145%2F3219819.3219890.

Wayne W. Zachary. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research*, 33(4), 1977. doi: 10.1086/jar.33.4.3629752. URL https://doi.org/10.1086/jar.33.4.3629752.

Muhan Zhang. Graph neural networks: Link prediction. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 195–223. Springer Singapore, Singapore, 2022.

Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks, 2019. URL https://arxiv.org/abs/1904.12058.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications, 2018. URL http://arxiv.org/abs/1812.08434. cite arxiv:1812.08434.

Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. Graph neural networks: Taxonomy, advances, and trends. *ACM Transactions on Intelligent Systems and Technology*, 13(1):1–54, jan 2022. doi: 10.1145/3495161. URL https://doi.org/10.1145%2F3495161.