Graph-based knowledge representation and extraction from unstructured textual data using machine learning algorithms

Nikos Kanakaris

A thesis submitted for the degree of
Doctor of Philosophy

Department of Mechanical Engineering and Aeronautics
University of Patras, Greece
July 2023

Graph-based knowledge representation and extraction from unstructured textual data using machine learning algorithms

by

Nikos Kanakaris

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

DISSERTATION COMMITTEE

Nikos Karacapilidis	Dept. Mechanical Engineering and Aeronautics, University of Patras
Spyros Sioutas	Dept. Computer Engineering & Informatics, University of Patras
Dimitrios Michail	Dept. Informatics & Telematics, Harokopio University of Athens
Iraklis Varlamis	Dept. Informatics & Telematics, Harokopio University of Athens
Sotiris Kotsiantis	Dept. Mathematics, University of Patras
Christos Makris	Dept. Computer Engineering & Informatics, University of Patras
Panagiotis Stavropoulos	Dept. Mechanical Engineering and Aeronautics, University of Patras

Department of Mechanical Engineering and Aeronautics
University of Patras, Greece

July 2023

Abstract

In this dissertation, we build upon existing graph-based text representation models and introduce a novel one to represent multiple textual documents as a single graph, namely, the graph-of-docs text representation. As opposed to existing graph-based text representations, graph-of-docs enables the investigation of the importance of a term into a whole corpus of documents, masks the overall complexity by reducing each graph of words to a 'document' node, and supports the inclusion of relationship edges between documents. Hence, it enables the calculation of important metrics with respect to multiple documents and allows heterogeneous data to co-exist in a single graph. Along with the introduction of the graph-of-docs model, this dissertation proposes and empirically evaluates the combination of word embeddings, graph-based text representations and graph neural networks to advance classical machine learning tasks, including text classification, regression, feature engineering and feature selection. We conduct several experiments on diverse datasets from different domains, settings and applications, aiming to evaluate the proposed approach. Among others, these datasets are related to (i) personnel selection, (ii) the identification of software bugs, (iii) the prediction of future research collaborations, (iv) the price prediction for Airbnb listings. The evaluation results demonstrate a significant improvement in terms of accuracy, precision and recall of the proposed models compared to various classical and graph-based counterparts.

Περίληψη

Η παρούσα διατριβή επεκτείνει υπάρχοντα μοντέλα αναπαράστασης κειμένου που βασίζονται σε γράφους και προτείνει ένα νέο μοντέλο, το 'graph-of-docs', για την αναπαράσταση πολλαπλών εγγράφων κειμένου σε έναν ενιαίο γράφο. Σε αντίθεση με τις υπάρχουσες προσεγγίσεις, το graph-of-docs επιτρέπει τη διερεύνηση της σημασίας ενός όρου στο σύνολο των εγγράφων, μειώνει τη συνολική πολυπλοκότητα ανάγοντας κάθε γράφο λέξεων σε έναν κόμβο, και υποστηρίζει την εισαγωγή ακμών σχέσης μεταξύ των εγγράφων. Ως εκ τούτου, επιτρέπει τον υπολογισμό σημαντικών μετρικών σε σχέση με πολλαπλά έγγραφα και επιτρέπει τη συνύπαρξη ετερογενών δεδομένων στον ίδιο γράφο. Επιπλέον, η διατριβή προτείνει και αξιολογεί εμπειρικά το συνδυασμό των διανυσμάτων λέξεων (word embeddings), των αναπαραστάσεων κειμένου και των νευρωνικών δικτύων που βασίζονται σε γράφους για την βελτίωση κλασικών εφαρμογών της μηχανικής μάθησης, όπως η ταξινόμηση κειμένου, η παλινδρόμηση, και η επιλογή διακριτικών χαρακτηριστικών (feature selection). Στο πλαίσιο της διατριβής έλαβε χώρα μια σειρά πειραμάτων και μετρήσεων, αξιοποιώντας σύνολα δεδομένων από διαφορετικούς τομείς και εφαρμογές, με στόχο την αξιολόγηση της προτεινόμενης προσέγγισης. Τα δεδομένα αυτά αφορούν θέματα επιλογής προσωπικού, εντοπισμού σφαλμάτων λογισμικού, πρόβλεψης μελλοντικών ερευνητικών συνεργασιών και πρόβλεψης τιμών για καταχωρίσεις στην πλατφόρμα της Airbnb. Τα αποτελέσματα της αξιολόγησης επιδεικνύουν μια στατιστικά σημαντική βελτίωση μιας σειράς μετρικών (accuracy, precision, recall) των προτεινόμενων λύσεων σε σύγκριση με τα αντίστοιχα κλασικά και βασισμένα σε γράφους μοντέλα.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Nikos Karacapilidis for his tireless guidance throughout my Ph.D. journey. His ability to seamlessly combine existing literature from diverse fields and propose novel ideas has always been and will continue to be, a great inspiration to me.

Secondly, I would like to thank Prof. Dimitrios Michail who has been there for me since the first year of my undergraduate studies, teaching and providing me with valuable knowledge in the field of computer science. His remarkable ability to grasp technical concepts effortessly has been another great inspiration to me.

I would like to thank Prof. Iraklis Varlamis, who taught me how to effectively manage multiple projects and jobs throughout a day. I extend my appreciation to the members of my doctoral committee, Prof. Spyros Sioutas, Prof. Sotiris Kotsiantis, Prof. Christos Makris and Prof. Panagiotis Stavropoulos for their valuable feedback. I cannot forget to mention Dr. Alexis Lazanas, who has been my Shaman. He provided me invaluable support by attentively listening to my concerns and encouraging me to persist and not give up.

A big thank you goes to my colleagues and friends. In particular, I would like to thank my comrade Ilias Siachos for his assistance in conducting the experiments and writing many of the passages of the papers together. His company has been truly invaluable. I would like to thank Nikos Giarelis and George Kournetas, with whom I had also the pleasure of working and co-authoring several papers. I would like to thank Dr. Giannis Nikolentzos. Our short conversations, along with his significant contributions to graph mining, inspired me to conceptualize the basic idea of graph-of-docs. I would like to thank one of my closest friends, Dr. Stamatis Karlos. Since 2018, when we first met, his suggestions and wisdom have greatly assisted me in managing my Ph.D. research. I would like to extend my gratitude to the following officemates who made me feel comfortable, especially during my first days at the university: Dr. Elias Spyropoulos, Stavros Dimopoulos, Dr. Spyros Christodoulou, Vicky Rigatou, Prof. Paraskevas Georgiou and Dr. Yannis Mouzakitis.

Besides my friends in the academic world, I would like to thank Giorgos Andrikogiannopoulos. Our countless late-night conversations about various philosophical topics will forever hold a special place in my heart. I would like to thank Thodoris Varouxis who inducted me into the department of Mechanical Engineering and Aeronautics. Without his assistance my path would be totally different.

Lastly, I want to express my deepest gratitude to my family: Haroula Baliaka who puts up with my whining on a daily basis. She has been there for me during both my good and bad days. I cannot stress enough how thankful I am for her constant support and help; my parents, Tzortzis and Theoni, and my brother, Ioakim, who also supported me both mentally and financially from the very beginning of my journey in our beautiful world.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past few years, an increase in the adoption of graph-based representations for modeling unstructured textual data has been witnessed (Blanco & Lioma, 2012; Sonawane & Kulkarni, 2014). In general, graph-based representations are capable of storing relationships between the words of a document and information about each individual word, at the same time. Furthermore, the adoption of graph-based text representations enables the utilization of algorithms and methods from the domain of graphs, ranging from classical (e.g. node centrality measures and indices) to more recent ones, including graph neural networks and learning graph representations (for instance, using algorithms for embeddings such as node2vec).

In most cases, graph-based text representations overcome the limitations of the classical bag-of-words representation (Aggarwal, 2018). As opposed to bag-of-words, graph-based text representations (i) capture structural and semantic information of a text, (ii) mitigate the effects of the "curse-of-dimensionality" phenomenon, (iii) identify the most important terms of a text, (iv) seamlessly incorporate information coming from external knowledge sources and (v) most importantly, produce dense vector representations by relying on representation learning methods for graphs; these vector representations can be given as input to machine learning models to perform other downstream tasks, to name but a few, text classification, regression and question-answering.

Despite the recent advancements in the domain of graph-based text representation, the existing approaches concentrate on modeling each document of a corpus as a graph, leaving room for further research as far as the management of multiple textual documents in a single graph is concerned. The main weaknesses of the existing approaches are that (i) they are incapable of assessing the importance of a word for the whole set of documents, (ii) they do not allow for representing similarities between these documents, (iii) they cannot easily operate with heterogeneous data due to their homogeneous nature and (iv) they cannot easily facilitate the analysis of groups of documents, for instance, documents from discussion threads and conversations.

1.1 Motivation, scope and contribution

Towards mitigating the above weaknesses, this dissertation builds upon existing graph-based text representation models (Rousseau et al., 2015; Rousseau & Vazirgiannis, 2013) and introduces a novel approach to represent multiple documents as

a single graph, namely, the graph-of-docs text representation. Contrary to existing approaches, the one introduced in this dissertation (i) enables the investigation of the importance of a term into a whole corpus of documents, (ii) masks the overall complexity by reducing each graph of words to a 'document' node, (iii) supports the inclusion of relationship edges between documents, thus enabling the calculation of important metrics as far as documents are concerned, and (iv) allows heterogeneous data to co-exist in a single graph (e.g. words and document entities).

Along with the introduction of the graph-of-docs model, this dissertation proposes and empirically evaluates the combination of word embeddings, graph-based text representations and graph neural networks to advance classical machine learning tasks, including text classification, feature engineering and feature selection.

To evaluate the proposed approaches, we conduct several experiments on diverse datasets from different domains, settings and applications. Among others, these datasets are related to (i) personnel selection, (ii) the identification of software bugs, (iii) the prediction of future research collaborations, (iv) the price prediction for Airbnb listings.

To sum up, the main contributions of this dissertation from a theoretical point of view are the following (see also Figure 1.1):

- we propose a novel approach for representing multiple textual documents as a single graph;
- we introduce a new graph-based feature selection method for multiple textual documents;
- we investigate whether the integration of unstructured textual data into a single knowledge graph affects the performance of a link prediction model;
- we investigate whether the combination of graph neural networks and document embeddings benefits predictions of numbers (regression);
- we study the effect of previously proposed graph kernels-based approaches on the performance of an ML model, as far as the link prediction problem is concerned;
- we propose a three-phase pipeline that enables the exploitation of structural and textual information, as well as pre-trained word embeddings;
- we provide the research community with an approach that improves existing ones by exploiting word embeddings, neural networks and graph representation learning techniques in an integrated way.

Finally, from an application point of view, the main contributions of the dissertation are (see also Figure 1.1):

- we investigate whether the analysis of unstructured textual data benefits the personnel selection process or not;
- we propose a four-phase pipeline that assists project managers in the personnel selection process;

- we provide the research community with a rich dataset containing tasks of the projects of an organization that can be utilized as a baseline in similar research directions;
- we propose a new approach for predicting future research collaboration using knowledge graphs;
- we meaningfully introduce techniques from the area of GNNs and graph-based text representations to the task of identifying software bugs;
- we provide the research community with a new approach that improves the quality of software bug predictions;
- we investigate whether the representation of the text of a GitHub issue as a graph-of-words graph benefits the software bug identification process;
- we meaningfully introduce techniques from the area of GNNs and graph representations in the field of price prediction for Airbnb listings;
- we provide the research community with a new approach that improves the quality of price predictions, especially in the case of small touristic destinations with a big diversity of prices;
- we investigate whether the representation of a touristic area as a graph benefits the price prediction process or not.

Most of the code for this dissertation has been developed in Python, using the PyTorch, TensorFlow, PyTorch Geometric (PyG), NetworkX and scikit-learn libraries. In addition, Neo4j has been utilized as a database. All the code accompanied by the datasets is available on GitHub.

1.2 Outline

The remainder of this dissertation is organized as follows. Basic concepts and preliminaries are introduced in Chapter 2. Chapter 3 describes a novel graph-based approach for representing multiple textual documents as a single graph, namely graph-of-docs. In Chapter 4, a graph-based feature selection approach is presented, which utilizes the graph-of-docs representation. In Chapter 5, techniques from the area of knowledge graphs are employed to perform link prediction. Chapter 6 elaborates on a meaningful integration of word embeddings and graph-based text representations. In Chapter 7, we showcase the combination of word embeddings, graph-based text representation and graph attention networks. Chapter 8 demonstrates the adoption of graph neural networks and document embeddings to make numerical predictions. Concluding remarks, lessons learned as well as future work directions are summarized and discussed in Chapter 9.

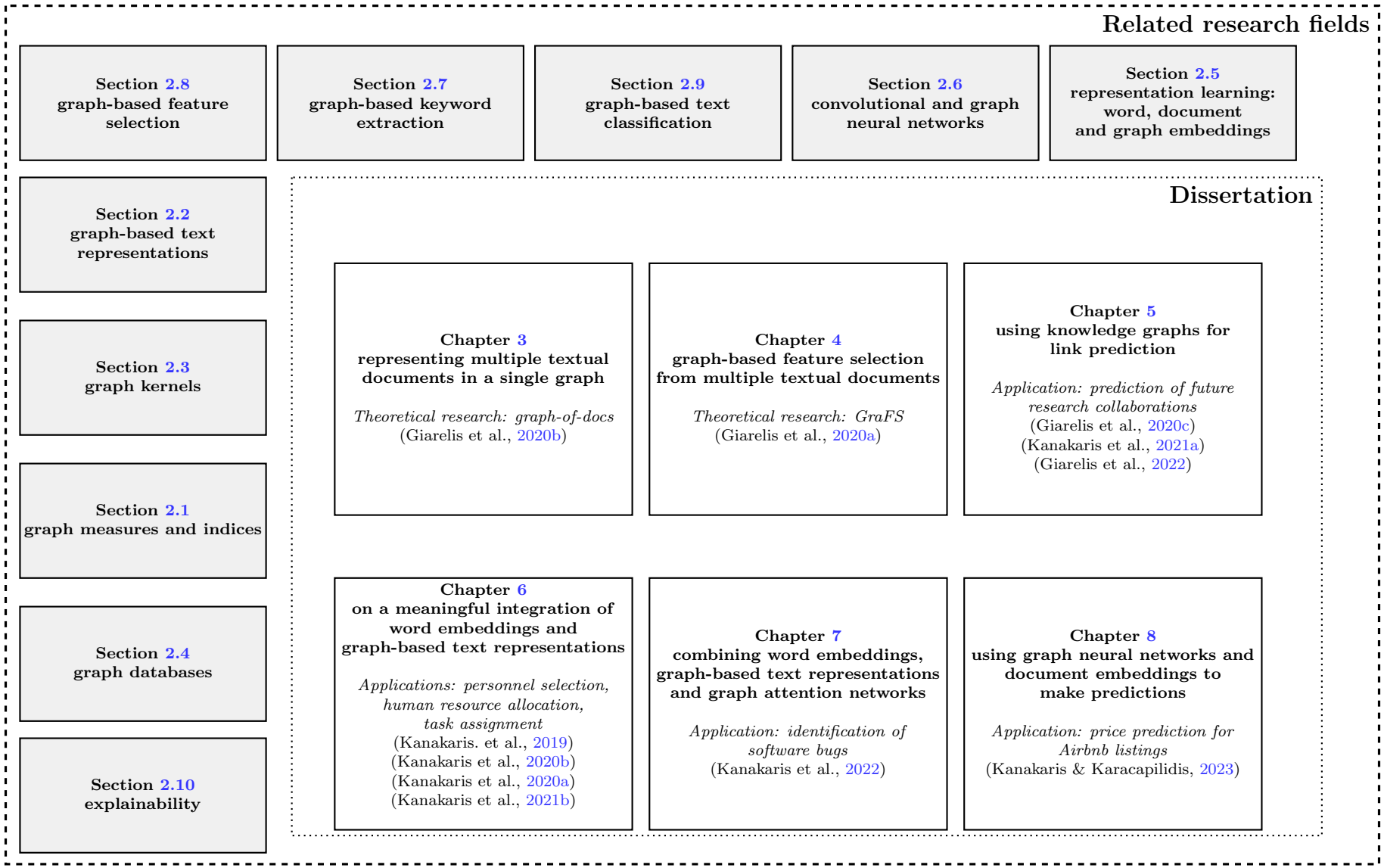


Figure 1.1: An overview of the contributions (in terms of theory and applications) presented in this dissertation, along with the related existing literature. For simplicity, the dependencies between the various components of the figure are omitted.

Chapter 2

Basic concepts and preliminaries

2.1 Graph-related concepts, measures and indices

In graph theory, numerous graph types have been developed, which are primarily differentiated on the types of vertices and edges, the features that their nodes or edges may incorporate and their general architecture (West et al., 2001). Furthermore, a variety of graph measures and indices which extract knowledge based on the structural characteristics of a graph have been proposed in the literature (Vathy-Fogarassy & Abonyi, 2013). Below, a small subset of them is presented, which are applied in the several approaches described in the next chapters.

Definition 1 (Graph). *Let a graph $G = (V, E)$ be defined as a tuple consisting of a set of vertices (or nodes) V and a set of edges $E \subseteq V \times V$.*

A vertex, denoted as v_i belongs to the graph G if $v_i \in V$. Similarly, an edge $e_i = (v_x, v_y)$, connecting two nodes $v_x, v_y \in V$, is part of the graph G if $e_i \in E$. The size of the graph is defined by the number of vertices $|V|$, while the number of the edges is defined as $|E|$.

Definition 2 (Directed Graph). *A directed graph is a graph $G = (V, E)$, where the edges are directed by arrows. Similarly, if the edges are not directed, the graph is called undirected.*

Definition 3 (Heterogeneous Graph). *A heterogeneous graph is a graph $G = (V, E)$, in which there exists a node type mapping function $l : V \rightarrow L$, which assigns types/labels to its vertices and a link type mapping function $\hat{l} : E \rightarrow \hat{L}$ which assigns types/labels to its edges from a discrete set of types L and $\hat{L}$, respectively.*

If only the vertices are labeled, the graph is called node-labeled. Similarly, in the case that only the edges are labeled, the graph is called edge-labeled, while a fully-labeled graph is considered to be a graph with both vertices and edges labelled.

Definition 4 (Neighbor Nodes). *A pair of vertices $v_i, v_j \in V$ of a graph $G = (V, E)$ are called neigbor nodes, if and only if the edge $e = (v_i, v_j) \in E$*

Throughout the next chapters, the notation $N(v)$ will denote the set of neighbors for a node v.

Definition 5 (Weighted Graph). *A weighted graph is a graph $G = (V, E)$, in which a function $w : E \to \mathbb{R}$ assigns edges with some weights or numbers.*

Definition 6 (Labeled Graph). *A labeled graph is a graph $G = (V, E)$, in which a function $l : V \cup E \to \Sigma$ assigns labels to its vertices and edges from a discrete set of values Σ. Depending on the application at hand, there may exist a necessity to label only the vertices, in which case the graph is called node-labeled graph. Accordingly, if only the edges are labeled, the graph is called edge-labeled graph. Finally, a graph with both vertices and edges labelled is called fully-labeled.*

Definition 7 (Attributed Graph). *A graph $G = (V, E)$ is an attributed graph if there exists a function $f : V \cup E \to \mathbb{R}^d$ that assigns real-valued vectors to its vertices and edges.*

Definition 8 (Adjacency Matrix). *Adjacency matrix offers a systematic mapping of the graph structure. Let A_{ij} be the element in the i^{th} th row and j^{th} th column of the matrix A. Then, the adjacency matrix A can be defined as:*

$$A_{ij} = \begin{cases} 1, & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Definition 9 (Degree Matrix). *Given a graph $G = (V, E)$, the degree matrix is defined as the diagonal matrix $D = diag(d_{ii})$ where:*

$$d_{ii} = \sum_j A_{ij} \tag{2.2}$$

Definition 10 (Walk, Path). *A walk in a graph $G = (V, E)$ is a series of vertices $v_1, v_2, ..., v_{k+1}$ where $v_i \in V$ for all $1 \leq i \leq k + 1$ and edges $\{v_i, v_{i+1}\} \in E$ for all $1 \leq i \leq k$. The number of the edges in this sequence is called the length of the walk. If all the vertices in the walk are different from each other, the walk is called a path.*

Definition 11 (Shortest Path). *A path from node v_i to node v_j is defined as the shortest path between these two nodes, if there exist no other path between these two nodes with smaller length.*

Definition 12 (Graph Isomorphism). *A graph isomorphism between two labeled/attributed graphs $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$ is a bijection $\phi : V_i \to V_j$, that preserves adjacencies, i.e., $\forall v, u \in V_i : (v, u) \in E_i \Leftrightarrow (\phi(v), \phi(u)) \in E_j$, and labels, i.e., if $\psi \in V_i \times V_i \to V_j \times V_j$ is the mapping of vertex pairs implicated by the bijection ϕ such that $\psi((v, u)) = (\phi(v), \phi(u))$, then, the conditions $\forall v \in V_i : \ell(v) \equiv \ell(\phi(v))$ and $\forall e \in E_j : \ell(e) \equiv \ell(\psi(e))$ must hold, where $\equiv$ denotes that two labels are considered equivalent.*

Definition 13 (Jaccard Coefficient). *The Jaccard Coefficient index for a pair of vertices $v_i, v_j \in V$ of a graph $G = (V, E)$, denoted by $J(v_i, v_j)$, is a statistical metric used for calculating the similarity or diversity of two nodes. Specifically, it considers the amount of the intersection of their neighbor nodes (the set of common neighbors) over the union of them (the set of nodes that are neighbors with either v_i or v_j) (Jaccard, 1901). It is defined as:*

$$J(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$$

Definition 14 (Cosine Similarity). *The Cosine Similarity measure for a pair of vectors* $\vec{x}, \vec{y} \in \mathbb{R}^D$*, denoted by* $cos(\vec{x}, \vec{y})$*, is a similarity metric calculated between two non-zero vectors, which is commonly calculated as the Euclidean dot product between vectors x, y. It is defined as:*

$$cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \, \|\vec{y}\|} = \frac{\sum_{n=1}^{D} x_i \cdot y_i}{\sqrt{\sum_{n=1}^{D} x_i^2} \cdot \sqrt{\sum_{n=1}^{D} y_i^2}}$$

In the next chapters, we utilize the cosine similarity to calculate the similarity between two word embeddings (see Section 2.5 for more information about word embeddings).

Definition 15 (Common Neighbors). *The Common Neighbors measure for two nodes a and b, denoted by CN(a, b), calculates the number of common neighbor nodes (i.e., nodes that are connected with both a and b) (Li et al., 2018). It is defined as:*

$$CN\,(a,\ b) = |N\,(a)\ \cap\ N\,(b)|$$

where N(x) *denotes the set of neighbors for a node* x.

Definition 16 (Total Neighbors). *The Total Neighbors measure for two nodes* a *and* b*, denoted by* TN(a, b)*, takes into consideration all neighbors of these two nodes (contrary to the Common Neighbors measure which deals with only the neighbor nodes). It is defined as:*

$$TN\,(a,\ b) = |N\,(a)\ \cup\ N\,(b)|$$

Definition 17 (Preferential Attachment). *The Preferential Attachment measure for a pair of nodes* a *and* b*, denoted by* PA(a, b)*, is defined as the product of the in-degree values of the two nodes (Albert & Barabasi, 2001). The assumption behind this measure is that the likelihood of two nodes being connected in the future is far greater for two highly connected nodes, in contrast to two loosely connected ones. This measure is defined as:*

$$PA\,(a,\ b) = |N\,(a)|\ \times\ |N\,(b)|$$

Definition 18 (Adamic Adar). *The Adamic Adar measure for two nodes* a *and* b*, denoted by* AA(a, b)*, calculates the sum of the inverse logarithm of the degree of the neighbor nodes shared by the nodes* a *and* b *(Adamic & Adar, 2003). This measure assumes that the likelihood of a neighbor node to be influential in the future depends on how low its degree may be. It is defined as:*

$$AA\,(a,\ b) = \sum\nolimits_{c\ \epsilon\ N(a) \cap N(b)} \left(\frac{1}{\log |c|} \right)$$

2.2 Graph-based text representations

The graph-of-words textual representation is similar to the bag-of-words representation that is widely used in the NLP field. It enables a more sophisticated keyword

extraction and feature engineering process. In a graph of words, each node represents a unique term (i.e., word) of a document, and each edge represents the co-occurrence between two terms within a sliding window of text. Nikolentzos et al. (2017a) propose the utilization of small sliding window size, due to the fact that the larger ones produce heavily interconnected graphs where the valuable information is cluttered with noise; Rousseau et al. (2015) suggest that a window size of four is generally considered to be the appropriate value, since it does not sacrifice either the performance or the accuracy of their approach.

In the graph-of-words textual representation (Rousseau & Vazirgiannis, 2013) each document of a corpus is represented as a single graph. Specifically, every unique word of a document is depicted as a graph node and the co-occurrence between two words (i.e., if two words appear simultaneously within a sliding window of text) is denoted with an edge connecting the corresponding nodes. As far as the size of the sliding window is concerned, it is suggested in (Rousseau et al., 2015) that a window of four words seems to be the most appropriate value, as the impact on the performance and the accuracy of the ML models is negligible. Taking into consideration the co-occurrence between terms results in feature engineering being more sophisticated in the graph-of-words representation, compared to the bag-of-words one. Nevertheless, the limitations of the graph-of-words text representation are that: (i) the importance of a word for a whole set of documents cannot be assessed; (ii) it is not possible for multiple documents to be represented in a single graph, and (iii) the expendability of the representation in order to support more complex data architectures is not intuitive.

To address the drawbacks of the graph-of-words representation, the authors in (Giarelis et al., 2020b) have proposed the graph-of-docs representation, where multiple textual documents are depicted in a single graph. In this way: (i) the investigation of the importance of a term into a whole corpus of documents is easily calculated, and (ii) the co-existence of heterogeneous nodes in the same graph renders the representation easily expandable and adaptable to more complicated data. In this chapter, the graph-of-docs model is utilized to represent the textual data of a knowledge graph.

2.3 Graph kernels

Kernels have gained much attention as a generalization technique in machine learning applications, aiming to model and compute the similarity among objects. Specifically, graph kernels have proven to be the dominant approach for learning on graph-structured data (Nikolentzos et al., 2021). Machine learning tasks with graph-based data are directly related to graph comparison, which presents many complex difficulties, as the nature of graph-structured data tends to be very different from the usual representations (vectors, matrices etc.). To overcome these problems, graph kernels capture the semantics and the latent characteristics that are inherent in the graph in a computationally efficient time, thus achieving state-of-the-art results on several datasets.

Graph kernels compute the similarity between two graphs, based on the common substructures they share. In the literature, a variety of substructures have been proposed, including random walks (Gärtner et al., 2003; Vishwanathan et al., 2008), shortest paths (Borgwardt & Kriegel, 2005) and subtrees (Ramon & Gaertner, 2003).

Generally, we can express a graph kernel as the inner product defined in some Hilbert space, e.g., given a kernel k, we can define a mapping function $\varphi : \mathcal{G} \to \mathcal{H}$ into a Hilbert space $\mathcal{H}$ such that $k(G_1, G_2) = \langle \varphi(G_1), \varphi(G_2) \rangle$ for all graphs $G_1, G_2 \in \mathcal{G}$. Depending on the substructures in hand, graph kernels achieve to compute the similarity among graphs with respect to these substructures. Therefore, for each application under consideration, one must carefully choose which kernels should be utilized.

Depending solely on substructures of graphs, however, can create the downside of ignoring global structure and characteristics, which may be valuable. For graphs with small size, such local approaches may be appropriate; however, for larger graphs, graph kernels may fail to perform equivalently well. As a result, the problem of incorporating global properties of graphs to graph kernels has gained some attraction and a recent work attempts to tackle this issue (Nikolentzos et al., 2017b). We refer to (Nikolentzos et al., 2021) for an in-depth review of graph kernels.

2.3.1 Pyramid match graph kernel

The pyramid match graph kernel makes use of mappings of nodes of a graph to a low-dimensional vector space (embeddings) (Nikolentzos et al., 2017b). The most profound way of producing node embeddings is by using the eigenvectors of the d largest in magnitude eigenvalues of the adjacency matrix A of the graph. By creating a d-dimensional vector for each node, we define a d-dimensional hyperplane (hypercube), where each node embedding is represented as a point. Then, the kernel partitions the hypercube into regions of increasingly larger size and for each region takes a weighted sum of the matches, i.e., sum all the co-occurrences of points into the same region. As the regions expand, the matches are less and less weighted.

More specifically, the kernel repeatedly fits a grid with cells of increasing size to the d-dimensional hypercube. Each cell is related only to a specific dimension and its size along that dimension is doubled at each iteration, whereas it remains constant for the rest of the dimensions. Given a sequence of iterations (levels) from 0 to R, at level r the d-dimensional kernel region has 2^r cells along each dimension and $D = 2^r \cdot d$ cells in total. Then, given two graphs G_i, G_j, let $H^r_{G_i}$, $H^r_{G_j}$ denote the histograms of G_i and G_j at iteration r, and $H^r_{G_i}(k)$, $H^r_{G_j}(k)$ the number of nodes that lie in the k^{th} cell for each graph. The number of nodes (matches) found in the same region at iteration r is computed as follows:

$$I\left(H^r_{G_i}, H^r_{G_j}\right) = \sum_{k=1}^{D} \min\left(H^r_{G_i}(k), H^r_{G_j}(k)\right)$$

However, there is no need to compute the matches found in every iteration as a large number of these matches should have been computed in a previous iteration. Instead, we want to compute the number of new matches found at each level which is given by $I\left(H^r_{G_i}, H^r_{G_j}\right) - I\left(H^{r+1}_{G_i}, H^{r+1}_{G_j}\right)$ for $r = 0, \ldots, R-1$. Then, the pyramid match kernel is defined as follows:

$$k(G_i, G_j) = I\left(H^R_{G_i}, H^R_{G_j}\right) + \sum_{r=0}^{R-1} \frac{1}{2^{R-r}} \left(I\left(H^r_{G_i}, H^r_{G_j}\right) - I\left(H^{r+1}_{G_i}, H^{r+1}_{G_j}\right)\right)$$

where $\frac{1}{2^{R-r}}$ is the decreasing weight appointed to each level.

2.3.2 Propagation kernel

The basic idea behind the propagation kernels is the propagation of label information between nodes of the graph, based on the overall graph structure. The general framework was introduced in (Neumann et al., 2016), where the graph is considered to have attributes on nodes (attributed graph). Propagation kernel defines a probability distribution P of size $|V| \times d$, where d is the size of attributes in the feature space. This probability distribution is then updated for a given number of iterations t_{MAX} by concatenating the neighbors' attributes, on the basis of the following substitution rule:

$$P^{t+1} \rightarrow D^{-1} \cdot A \cdot P^t$$

where D is the degree matrix and A is the adjacency matrix for a graph. Then, the probability matrix P is utilized to bin nodes, through a hash function producing a hash vector $\varphi\left(G^t\right)$. Finally, we compute the kernel $K\left(G_i^t, G_j^t\right)$ at iteration t between two graphs G_i, G_j as follows:

$$K\left(G_i^t, G_j^t\right) = \sum_{v_i \in V_i} \sum_{v_j \in V_j} k\left(v_i, v_j\right) \;=\; \left\langle \varphi\left(G_i^t\right), \varphi(G_j^t) \right\rangle$$

This procedure is repeated for t_{MAX} iterations, until the kernel converges to a local minimum.

2.4 Graph databases

Compared to conventional relational databases, graph databases provide a more convenient and efficient way to natively represent and store highly interlinked data. In addition, they allow the retrieval of multiple relationships and entities with a single operation, avoiding the utilization of rigid joint operations which are heavily used in relational databases (Miller, 2013). An in-depth review of graph databases appears in (Rawat & Kashyap, 2017). The majority of the implementations of the proposed approaches of the next chapters builds on top of the Neo4j graph database (https://neo4j.com), a broadly adopted graph database system that uses the highly expressive Cypher Graph Query Language to query and manage data.

2.5 Representation learning

2.5.1 Word and document embeddings

In general, word embeddings map a corpus of words into a vector space where similar words have similar vector representations. A list of techniques for learning word embeddings has been proposed in the literature (Almeida & Xexéo, 2019), the most popular ones including *Word2Vec* (Mikolov et al., 2013c), *GloVe* (Pennington et al., 2014) and *fastText* (Joulin et al., 2016). Word embeddings are broadly used in many NLP tasks ranging from text classification and sentiment analysis to more sophisticated ones such as spam detection and question-answering. They improve the accuracy of an ML model, prevent overfitting and assist in terms of generalization (Andreas & Klein, 2014; Kholghi et al., 2016). In majority, word embeddings are trained on large datasets, which are usually language or domain specific. Hence,

they enable the learning techniques to capture statistical correlations between the words and, subsequently, produce word embeddings for a specific NLP task (e.g., finding cars similar to a Jaguar or finding animals similar to a jaguar). A common practice is to start with pre-trained word embeddings and adjust them to a specific domain or NLP task. Popular pre-trained word embeddings include *GoogleNews* using Word2Vec and *Common Crawl* using GloVe. We refer to (van der Heijden et al., 2020) for an in-depth review of the available pre-trained word embeddings.

Similar to word embeddings, documents can be represented as vectors. Some of the most utilized techniques for creating document embeddings include *bag-of-words* and *TF-IDF* (Smeaton, 1996). Although these methods provide a simple and rapid way to produce document representations, they fail to capture the importance of grammar and ordering information, which is hidden in the sequence of the words. The need to incorporate this valuable information into the embeddings, as well as the recent advancements made as far as word representations are concerned, led to the utilization of word embeddings techniques as core elements for producing document embeddings. For example, *Doc2Vec* (Le & Mikolov, 2014) depends on *Word2Vec* to map each document of a corpus as a vector.

2.5.2 Graph embeddings

Feature extraction and selection in graph-based ML techniques, such as node classification (i.e. predicting the labels of nodes in graphs), have always been a challenging task. The nature of graph representations makes it impossible to create features automatically, as they usually depend on specific graph properties. However, graph embeddings seek to remedy this shortcoming by mapping graphs to a vector or a set of vectors, thus reducing the feature extraction task to a representation learning one. Graph embeddings can be applied on node level (i.e. producing a vector representation for each node of a graph) or graph level (i.e. mapping a graph to a vector representation). The basic concept behind node embeddings is to create a vector representation for a node by capturing high-dimensional information regarding its neighborhood and utilizing dimensionality reduction techniques to compress this information into a low-dimensional feature space.

A variety of techniques for producing node embeddings has been proposed in the literature. The majority of these methods suggest a two-step process, as it was first introduced by *DeepWalk* (Perozzi et al., 2014). *DeepWalk* is inspired by language modeling techniques and proposes the construction of a set of random walks on a graph. These random walks are fed to a *SkipGram* model (Mikolov et al., 2013a) to produce a matrix of vector representations (node embeddings), in a similar fashion to how a *SkipGram* model maximizes the co-occurrence probabilities among the words that appear within a window.

Node2Vec (Grover & Leskovec, 2016) is a semi-supervised representation learning graph algorithm that maximizes the probability of maintaining the structure of neighborhoods of nodes. It bears a great resemblance to *DeepWalk*, with the process of generating random walks being the factor that differentiates each other. Specifically, *Node2Vec* establishes second order random walks, which can be perceived as random walks that each step relies on input parameters. These parameters introduce a bias, as far as different graph traversal strategies are concerned (e.g. Breadth-first Sampling, Depth-first Sampling). *Node2Vec* manages to learn feature space repre-

sentations for the nodes, based on the graph properties and the neighborhoods they share.

Graph Embedding techniques that utilize *SkipGram* or other deep neural techniques, such as *DeepWalk*, *Node2Vec* and *LINE* (J. Tang et al., 2015), usually suffer from a performance bottleneck, because of their computationally expensive nature, which results in an increased execution time. *FastRP* (Chen et al., 2019) aims to alleviate this downside without downgrading the quality of the produced node embeddings. This is achieved by capturing the implicit relationships among nodes and then applying normalization of the similarity between each node in the graph. The normalization procedure uses the node degrees and a scalable dimension reduction algorithm, i.e. Very Sparse Random Projection to produce node embeddings in a time efficient way, compared to *SkipGram* and *SVD* (Mikolov et al., 2013b).

GraphSAGE (Hamilton et al., 2017a) is an inductive algorithm for producing node embeddings. Unlike the abovementioned algorithms that inherently calculate the node embeddings for a fixed graph (and thus producing poor results for newly added vertices, that did not participate in the training phase), *GraphSAGE* learns an embedding function, which can be generalized to include newly added nodes. The embedding function depends on node features, such as textual properties and node degrees, although the adaptability of this technique enables users to apply *GraphSAGE* to graphs that may or may not contain node attributes, as well as to fixed or dynamically expanding graphs.

2.6 Convolutional and graph neural networks

Convolutional Neural Networks (CNNs) have been successfully applied to classical ML problems, where multiple rows and columns of an input matrix are processed simultaneously. These problems include image classification, pattern recognition and natural language process tasks (He et al., 2015; Kanakaris et al., 2021b). However, they are incapable of generalizing to tasks where data with arbitrary structure exist (e.g. graphs). For instance, such data can be found in social, telecommunication and biological networks.

To remedy the above shortcoming of CNNs, Gori et al. (2005) first introduced GNNs as a generalization of recursive neural networks. Each GNN has a first iterative phase, where it disseminates the attributes of a node to its neighborhood. The second phase consists of a neural network that generates the output for each node of a graph (a number or a class for regression or classification problems, respectively). The architecture of GNNs models inherit characteristics by its predecessor, i.e. gated recurrent units (Cho et al., 2014).

Some of the most popular GNNs include Graph Convolutional Networks (*GCN*) and Graph Attention Networks (*GAT*). *GCN* (Kipf & Welling, 2017) advance the well-established Convolutional Neural Networks to create a propagation rule that utilizes the adjacency A and degree D matrices of a graph. These matrices are modified slightly with a symmetric normalization to form a hidden state, which can be repeated as many times as seems necessary. On the other hand, *GAT* (Veličković et al., 2018) introduce the concept of self-attention, where it computes the hidden states of each node by attending to its neighbors (for more details see Section 7.3.2). For an in-depth analysis of the different types of GNNs, as well as their advantages and limitations compared to each other, we refer to (Zhou et al., 2018).

2.7 Graph-based keyword extraction

A set of existing approaches performing classical NLP tasks builds on the graph-of-words textual representation. Ohsawa et al. (1998) were the first that use the graph-of-words representation in the keyword extraction and text summarization tasks. Their approach segments a graph of words into clusters aiming to identify frequent co-occurred terms.Adopting a similar research direction, the TextRank model implements a graph-based ranking measure to find the most prestigious nodes of a graph (i.e., the nodes with the highest in-degree value) and utilizes them for the tasks of keyword and sentence extraction (Mihalcea & Tarau, 2004). The utilization of node centrality measures to the keyword and key-phrase extraction tasks can also be found in the literature (Boudin, 2013); these measures include the "degree" centrality, the "closeness" centrality, the "betweenness" centrality, and the "eigenvector" centrality. Bougouin et al. (2013) propose a novel graph-based unsupervised topic extraction method, namely, TopicRank. TopicRank clusters key phrases into topics and identifies the most representative ones using a graph-based ranking measure (e.g., TextRank). Finally, Tixier et al. (2016) focus on the task of unsupervised single document keyword extraction, arguing that the most important keywords correspond to the nodes of the k-core subgraph (Seidman, 1983).

2.8 Graph-based feature selection

Several interesting graph-based feature selection approaches have been already proposed in the literature. For instance, (Rousseau et al., 2015) proposes various combinations and configurations of popular frequent subgraph mining techniques - such as gSpan (Yan & Han, 2002), Gaston (Nijssen & Kok, 2004) and gBoost (Saigo et al., 2009) - to perform unsupervised feature selection exploiting the k-core subgraph. In particular, aiming to increase performance, Rousseau and his colleagues rely on the concept of k-core subgraph to reduce the graph representation to its densest part. The experimental results show a significant increment of the accuracy compared to common classification approaches. The work reported in (Henni et al., 2018) applies centrality algorithms (such as PageRank) to calculate the centrality score of a graph's features and accordingly identify the most important ones. The approach presented in (Fakhraei et al., 2015) builds on combinations of several types of graph algorithms to discover highly connected features of a graph. Such algorithms include the Lou-vain Algorithm for community detection and the PageRank algorithm to discover influential nodes and other user-defined graph measures. This last approach combines PageRank and Coloring algorithms with the custom graph measures of in-degree and out-degree.

Other already proposed approaches rely on the recursive filtering of the existing feature space; for instance, one of them re-applies PageRank to find the most influential features (Ienco et al., 2008). These approaches use graph-connected features to include contextual information, as modeled implicitly by a graph structure, using relationships that describe connections among real data. They aim to reduce ambiguity in feature selection and improve accuracy in traditional Machine Learning methods.

2.9 Graph-based text classification

As far as graph-based text classification is concerned, several interesting approaches have been already proposed in the literature. Depending on their methodology, we can classify them into three basic categories: (i) approaches that employ frequent subgraph mining techniques during the feature extraction step; (ii) approaches that rely on graph kernels; and (iii) approaches that adopt recent techniques, including representation learning and graph neural networks.

2.9.1 Frequent subgraph mining for text classification

Popular frequent subgraph mining techniques include gSpan (Yan & Han, 2002), Gaston (Nijssen & Kok, 2004), and gBoost (Saigo et al., 2009). Rousseau et al. (2015) propose various combinations and configurations of these techniques, ranging from unsupervised feature mining using gSpan to unsupervised feature selection exploiting the k-core subgraph. In particular, aiming to increase performance, they rely on the concept of k-core subgraph to reduce the graph representation to its densest part. The experimental results show a significant increment in the accuracy compared to common classification approaches.

2.9.2 Graph kernels for text classification

Graph kernel algorithms contribute significantly to recent approaches for graph-based text categorization (Nikolentzos et al., 2021). A graph kernel is a measure that calculates the similarity between two graphs. For instance, a document similarity algorithm based on shortest path graph kernels has been proposed in (Nikolentzos et al., 2021); this algorithm can be used as a distance metric for common ML classifiers such as SVM and k-NN. The experimental results show that classifiers that are based on graph kernel algorithms outperform several classical approaches. It is noted that the GraKeL Python library collects and unifies widely used graph kernel libraries into a single framework (Siglidis et al., 2020), providing an easily understandable interface (similar to that of scikit-learn) that enables the user to develop new graph kernels.

2.9.3 Graph neural networks for text classification

While graph embeddings offer an intuitive way to extract features for the nodes of a graph, the process of feeding this information alongside the structure of the graph to an ML model remains a challenging task. The complex and arbitrary form of the graphs (i.e. they can not be mapped to a Euclidean space) is the main reason why classical ML modes cannot be applied to graphs. Graph Neural Networks (GNNs) can be applied directly to a graph, hence providing a convenient way as far as node, edge and graph prediction tasks are concerned.

Many GNN models have been proposed to address this challenge. The vast majority of these (e.g. *HeteGCN* (Ragesh et al., 2021), *STGCN* (Ye et al., 2020) *Text GCN* (Yao et al., 2019)) utilizes *GCNs* to reduce the text classification issue to either a node classification one or a graph classification one. Nevertheless, there have been also approaches that utilize graph theory concepts to create a GNN. For

example, *TextGraphTransformer* (Zhang & Zhang, 2020) create sub-graph batches for every word of the graph, which are then fed into a GNN.

2.10 Explainability

Explainability refers to the ability of interpreting the results of a (machine learning) model, as far as identifying the importance of the input features to the output is concerned (Arrieta et al., 2020). Explainable Artificial Intelligence (XAI) has gained attraction the recent years, as AI and ML tasks have been incorporated to many industry and daily activities. Many approaches handle ML models as "black boxes", in a way that they regard only the input and the output of the model as means to interpret the overall pipeline, without prior knowledge to the inner structure. *LIME* (Local Interpretable Model-Agnostic Explanation) (Ribeiro et al., 2016) can generate explanations for any classifier by creating locally perturbations across every prediction. Specifically, *LIME* create local samples around an instance x' by using a proximity score π_x and measuring the alterations of the classifier's behavior due to these samples. The explanations produced by *LIME* capture the importance of the features, which are then provided to the users in an understandable and interpetable manner. *SHAP* (SHapley Additive exPlanation) (Lundberg & Lee, 2017), on the other hand, utilizes game theory concepts to measure each feature's contribution to the prediction. Particularly, it assigns a value to each feature, which depicts the change in the expected model prediction when conditioning on that feature. These values explain how to get from the base value that would be predicted without these features to the current output of the model at hand. The calculation of these values depends on linear LIME models accompanied with a shapley kernel, as shown in (Lundberg & Lee, 2017).

While many approaches stemming from the field of XAI have been proposed and adopted towards interpreting the results of classical ML models, the nature of deep neural networks makes this process more challenging (Joshi et al., 2021). Recently, several approaches are proposed to explain the predictions of deep graph models. The basic principle behind these approaches is the extraction of the input elements (i.e., edges, neighbor nodes and the important features of each node) that contribute to each prediction. Depending on whether we require explanations for each instance of the dataset or a general understanding with high-level insights are preferred, these methods are categorized in two main groups: instance-level methods and model-level ones (Yuan et al., 2022). Instance-level methods generate individual explanations for each example. This explanations are usually more understandable as they are provided by real input instances but these approaches require more human oversight due to the individuality of the explanation generated. *SA* (Baldassarre & Azizpour, 2019) utilizes the gradients of the deep graph model to create salience maps that highlight which node features are crucial to the ML task at hand. The salience map is obtained by back-propagation in a similar fashion as the deep graph model training. The difference, however, is that the gradients are calculated for the input features rather than the model parameters. Note that the input features can be graph nodes, edges, or node features. *GNNExplainer* (Ying et al., 2019a) is a perturbation-based technique in that the importance of the input features (i.e., nodes, node features or edges) is estimated by examining if the output of the network alters in the absence of these features. In order to isolate the most crucial features, *GNNExplainer*

learns soft masks for edges and node features to explain the predictions via mask optimization. The learning procedure is optimized by employing concepts of information theory. Specifically, the mutual information between the predictions of the deep networks and the calculation graph that is obtained with the filtering mask is maximized.

On the other hand, model-level methods extract graph patterns to examine the functionality of the GNNs. They do not rely on the specific inputs to provide interpretable explanations but instead they utilize the total of the training instances to optimize a target prediction. The most known model-level approach for deep graph models is *XGNN* (Yuan et al., 2020), which uses graph generation that contain discriminative graph patterns as explanations. To that end, reinforcement learning is employed, where the graph generated is fed to the GNN to decide if an edge should be added to the explanation. In addition, several graph rules are incorporated to assure that the explanations will be valid. *XGNN* has shown promising results as far as graph classification models are concerned, whereas node-level tasks haven't, yet, been tested, to the best of our knowledge.

Chapter 3

Graph-of-docs: Representing multiple textual documents in a single graph

In this chapter, we introduce a novel approach to represent multiple documents as a single graph, namely, the graph-of-docs model, together with an associated novel algorithm for text categorization. The proposed approach enables the investigation of the importance of a term into a whole corpus of documents and supports the inclusion of relationship edges between documents, thus enabling the calculation of important metrics as far as documents are concerned. Compared to well-tried existing solutions, our initial experimentations demonstrate a significant improvement in the accuracy of the text categorization process. For the experimentations reported in this chapter, we used a well-known dataset containing about 19,000 documents organized in various subjects.

3.1 Introduction

In recent years, we have witnessed an increase in the adoption of graph-based approaches for the representation of textual documents (Blanco & Lioma, 2012; Sonawane & Kulkarni, 2014). Generally speaking, graph-based text representations exploit properties inherited from graph theory (e.g., node centrality and subgraph frequency) to overcome the limitations of the classical bag-of-words representation (Aggarwal, 2018). Specifically, graph-based models (contrary to the bag-of-words ones) are able to (i) capture structural and semantic information of a text, (ii) mitigate the effects of the "curse-of-dimensionality" phenomenon, (iii) identify the most important terms of a text, and (iv) seamlessly incorporate information coming from external knowledge sources.

However, in cases where a corpus of documents needs to be considered and analyzed, existing graph-based approaches represent each document of the corpus as a single graph. In such cases, the main weaknesses of these approaches are that (i) they are incapable of assessing the importance of a word for the whole set of documents and (ii) they do not allow for representing similarities between these documents.

To remedy the above weaknesses, this chapter expands the graph-based text representation model proposed in (Rousseau et al., 2015; Rousseau & Vazirgiannis,

2013), i.e., the graph-of-words model, and introduces a novel approach to represent multiple documents as a single graph, namely, the graph-of-docs model, as well as an associated novel algorithm for text categorization. Contrary to existing approaches, the one introduced in this chapter (i) enables the investigation of the importance of a term into a whole corpus of documents, (ii) masks the overall complexity by reducing each graph of words to a "document" node, and (iii) supports the inclusion of relationship edges between documents, thus enabling the calculation of important metrics as far as documents are concerned. The proposed approach uses the Neo4j graph database (https://neo4j.com) for the representation of the graph-of-docs model. For the implementation of our experiments, we use the Python programming language and the scikit-learn ML library (https://scikit-learn.org). Compared to well-tried existing solutions, our initial experimental results show a significant improvement in the accuracy of the text categorization process.

The remainder of the chapter is organized as follows. Our approach, i.e., graph of docs, is analytically presented in Section 3.2. Section 3.3 reports on the experiments carried out to evaluate the proposed approach. Finally, limitations of our approach, future work directions, and concluding remarks are outlined in Section 3.4.

3.2 The proposed approach: Graph of docs

In this chapter, we expand the "graph-of-words" model proposed by Rousseau and Vazirgiannis (2013) to introduce a "graph-of-docs" model. Contrary to the former model, where a graph corresponds to a single document, the proposed model represents multiple documents in a single graph. Our approach allows diverse types of nodes and edges to co-exist in a graph, ranging from types of nodes such as "document" and "word" to types of edges such as "is_similar," "connects," and "includes" (see Figure 3.1). This enables us to investigate the importance of a term not only within a single document but also within a whole corpus of documents. Furthermore, the proposed graph-of-docs representation adds an abstraction layer by assigning each graph of words to a document node. Finally, it supports relationship edges between documents, thus enabling the calculation of important metrics as far as the documents are concerned (e.g., identifying cliques or neighborhoods of similar documents, identifying important documents, generating communities of documents without any prior knowledge, etc.).

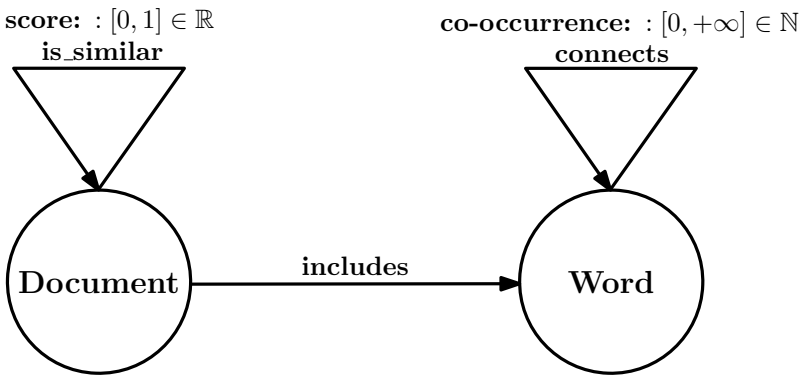


Figure 3.1: The schema of the graph-of-docs representation model.

The graph-of-docs representation produces a directed dense graph that contains

all the connections between the documents and the words of a corpus (see Figure 3.2). Each unique word node is connected to all the document nodes where it belongs to using edges of the "includes" type; edges of "connects" type are only applicable between two word nodes and denote their co-occurrence within a specific sliding text window; finally, edges of the "is_similar" type link a pair of document nodes and indicate their contextual similarity.

The above transformation of a set of documents into a graph assists in the reduction of diverse NLP problems to problems that have been well-studied through graph theory techniques (Rousseau et al., 2015). Such techniques explore important characteristics of a graph, such as node centrality and frequent subgraphs, which in turn are applied to identify meaningful keywords and find similar documents.

We argue that the accuracy of common NLP and text mining tasks can be improved by adopting the proposed graph-of-docs representation. Below, we describe how three key NLP tasks (namely, "Keyword Extraction," "Document Similarity," and "Text Categorization") can be carried out using our approach.

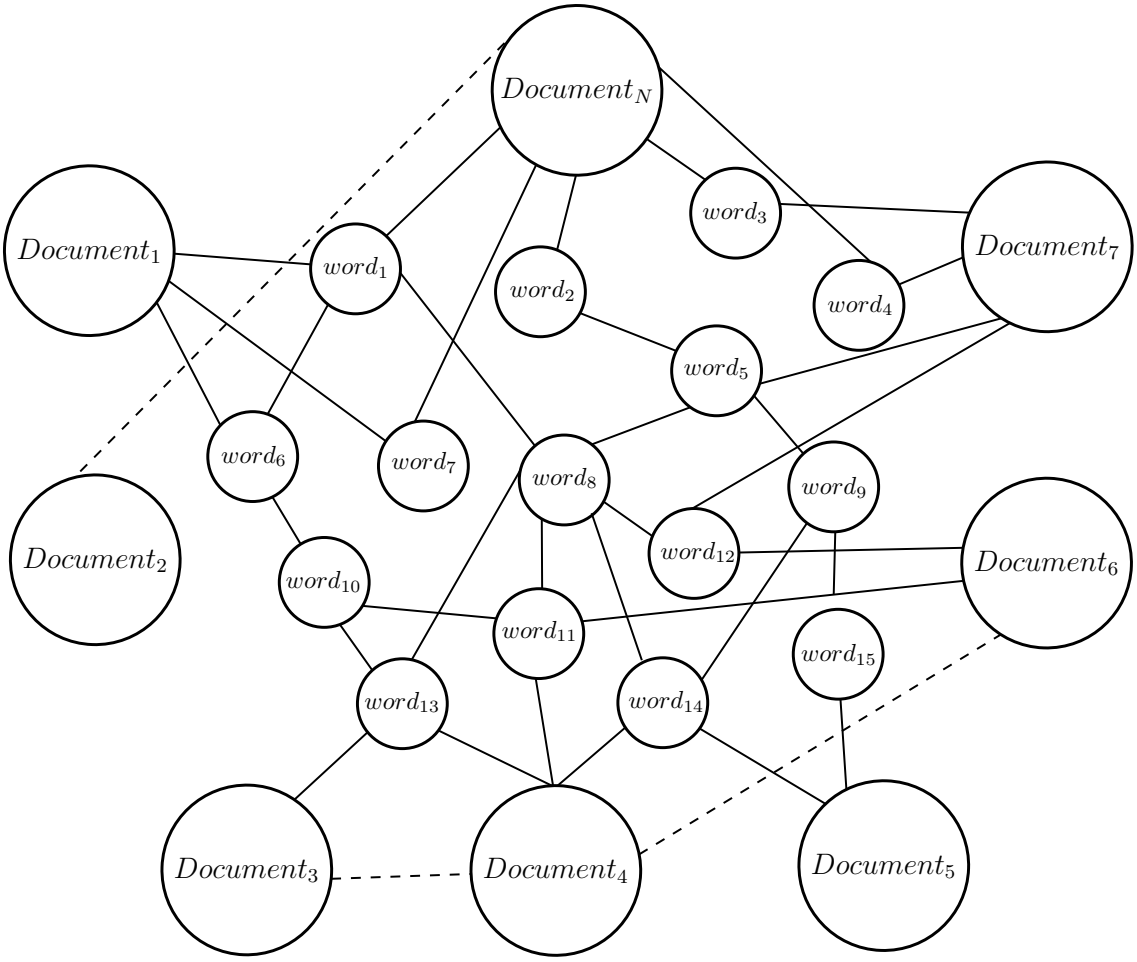


Figure 3.2: The graph-of-docs representation model (relationships between documents are denoted with dotted lines).

3.2.1 Keyword extraction

To extract the most representative keywords from each document, we apply centrality measures (an in-depth review of them appears in (Landherr et al., 2010)). In general, these measures identify the most influential nodes of a graph, i.e., those that usually have an indegree score higher than a predefined threshold. The main idea is that the words that correspond to the top-N ranked nodes can be considered as semantically more important than others in a specific document. Recent algorithms to calculate the centrality of a graph include PageRank, ArticleRank, Betweenness Centrality, Closeness Centrality, Degree Centrality, and Harmonic Centrality. While also utilizing the above algorithms to calculate centrality measures, our approach differs from the existing ones in that it considers the whole corpus of documents instead of each document separately; hence, we are able to detect a holistic perspective of the importance of each term.

3.2.2 Document similarity subgraph

Typically, graph-of-words derived from similar documents share common word nodes as well as similar structural characteristics. This enables us to calculate the similarity between two documents either by using typical data mining similarity measures (e.g., the Jaccard or the cosine similarity), or by employing frequent subgraph mining techniques (see Section 2.9). In our approach, we produce a similarity subgraph, which consists of document nodes and edges of "is_similar" type (we aim to extend the set of supported edge types in the future). It is clear that the creation of such a subgraph is not feasible in approaches that represent each document as a single graph.

3.2.3 Text categorization

By exploiting the aforementioned document similarity subgraph, we detect communities of contextually similar documents using the "score" property of the "is_similar" type edges as a distance value. A plethora of community detection algorithms can be found in the literature, including Louvain (Lu et al., 2014), Label Propagation (Raghavan et al., 2007), and Weakly Connected Components (Monge & Elkan, 1997); an in-depth review of them can be found in (Fortunato, 2009; Yang et al., 2016).

Since the documents that belong to the same graph community are similar, as far as their context is concerned, we assume that it is more likely to also share the same class when it comes to performing a text categorization task. Therefore, we can easily decide the class of a document either by using the most frequent class in its community of documents or by running a nearest neighbor algorithm (such as the k-nearest neighbors) using as input the documents of its community.

3.3 Experimental evaluation

3.3.1 Dataset

We have tested the proposed model by utilizing an already preprocessed version of the well-known 20 Newsgroups dataset, and specifically, the version containing

18,828 documents organized in various subjects (this dataset can be retrieved from http://qwone.com/~jason/20Newsgroups/20news-18828.tar.gz). This version does not contain unnecessary headers or duplicate texts that would require additional work as far as data cleansing is concerned. It is noted that this dataset is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. It has become a popular dataset for experiments in text applications of ML techniques, such as text classification and text clustering. We claim that this dataset fits well to the purposes of our experimentations (i.e., multi-class classification), given the large volume of different documents on the same subjects.

3.3.2 Implementation

The Neo4j graph database has been utilized for the representation of the proposed graph-of-docs model. Furthermore, we used the Python programming language and the scikit-learn ML library for the implementation of our experiments. The full code and documentation of our approach is freely available at https://github.com/NC0DER/GraphOfDocs.

Our approach consists of four major steps that are described in the sequel. Firstly, we execute a preprocessing function that (i) removes stopwords and punctuation marks from the texts and (ii) produces a list of terms for each document. Secondly, we execute a function that creates a graph of words by using the aforementioned terms. More specifically, this function creates unique word nodes from the list of terms and then links them (if needed), while also calculating the co-occurrence score. In this step, the graph of docs is created through the progressive synthesis of the graphs of words produced for each document. It is noted that by loading the 20 Newsgroups dataset, we generated a graph of docs with 174,347 unique nodes and 4,934,175 unique edges. Thirdly, we execute the PageRank algorithm aiming to identify the most important word nodes in the entire graph. Finally, we implement a function that calculates the Jaccard similarity measure for all document nodes; this function builds the document similarity subgraph and forms communities of similar documents using the Louvain algorithm. Our implementation is sketched in Algorithm 1.

Algorithm 1: The graph-of-docs representation.

```
database ← connect_to_the_database();
dataset ← read_dataset();
for each document_label, document in dataset do
    document ← clean_data(document);
    terms ← generate_terms(document);
    create_graph_of_words(terms, document_label, database);
end
run_word_centrality_measure_algorithm('Pagerank', database);
create_document_similarity_subgraph('Jaccard', database);
form_communities_of_similar_documents('Louvain', database);
conduct_classification_experiments();
disconnect_from_the_database();
```

3.3.3 Evaluation

Aiming to evaluate the performance of our approach, we benchmark the accuracy score of the text classifier described in Section 3.2.3 against those of common domain agnostic classifiers that use the bag-of-words model for their text representation (see Table 3.1). Considering the accuracy of each text classifier, we conclude that the proposed graph-of-docs representation significantly increases the accuracy of text classifiers (accuracy: 97.5%).

Text classifier	**Accuracy**
5-NN	54.8%
2-NN	61.0%
1-NN	76.0%
naive Bayes	93.7%
logistic regression	93.9%
neural network (100x50)	95.5%
neural network (1000x500)	95.9%
graph-of-docs classifier	**97.5%**

Table 3.1: Accuracy scores for the existing and the proposed text classifiers.

3.4 Discussion and conclusions

In this chapter, we introduced a novel approach for representing multiple textual documents in a single graph, namely, "graph of docs." To test our approach, we benchmarked the proposed "graph-of-docs"-based classifier against classical text classifiers that use the "bag-of-words" model for text representation. The evaluation outcome was very promising; an accuracy score of 97.5% was achieved, while the second best result was 95.9%. However, our approach has a set of limitations, in that (i) it does not perform equally well with outlier documents (i.e., documents that are not similar to any other document) and (ii) it has performance issues since the generation of a graph of documents requires significant time in a disk-based graph database such as Neo4j (W. Wang et al., 2005).

Aiming to address the above limitations as well as to integrate our approach into existing works on knowledge management systems, future work directions include: (i) the experimentation with alternative centrality measures, as well as diverse community detection and graph partitioning algorithms (Armenatzoglou et al., 2015); (ii) the utilization and assessment of an in-memory graph database in combination with Neo4j; (iii) the enrichment of the existing textual corpus through the exploitation of external domain agnostic knowledge graphs (e.g., DBPedia and Wikipedia knowledge); and (iv) the integration of our approach into collaborative environments where the underlying knowledge is structured through semantically rich discourse graphs (e.g., integration with the approaches described in (Kanterakis et al., 2019; Karacapilidis et al., 1997; Karacapilidis et al., 2009)).

Chapter 4

Graph-based feature selection from multiple textual documents

In this chapter, we introduce a novel graph-based approach to select features from multiple textual documents. The proposed solution enables the investigation of the importance of a term into a whole corpus of documents by utilizing contemporary graph theory methods, such as community detection algorithms and node centrality measures. Compared to well-tried existing solutions, evaluation results show that the proposed approach increases the accuracy of most text classifiers employed and decreases the number of features required to achieve 'state-of-the-art' accuracy. Well-known datasets used for the experimentations reported in this chapter include 20Newsgroups, LingSpam, Amazon Reviews and Reuters.

4.1 Introduction

Graph-based text representations are widely used in various Natural Language Processing, Text Mining and Information Retrieval tasks (Vazirgiannis et al., 2018). These representations exploit concepts and techniques inherited from graph theory (e.g. node centrality and subgraph frequency) to address limitations of the classical bag-of-words representation (Aggarwal, 2018); in this way, they are able to capture structural and semantic information of a text, mitigate the effects of the 'curse-of-dimensionality' phenomenon, identify the most important terms of a text, and seamlessly incorporate information coming from external knowledge sources. However, existing graph-based representations concern a single document each time. In cases where one needs to analyze a corpus of documents, these approaches demonstrate a series of weaknesses, the main of them being that they are incapable to assess the importance of a word for the whole set of documents.

Recently, graph-based text representations have been used to facilitate and augment the feature selection process, i.e. the process of selecting a subset of relevant features when constructing a model. These approaches combine statistical tests and graph algorithms to uncover hidden correlations between terms and document classes. However, while they take into account the co-occurrences between terms to identify the most representative features of a single document (something that is not the case in traditional statistical methods), they are not able to assess the importance of a term in a corpus of documents. To remedy the above weakness, this chapter builds on a graph-based text representation model to introduce a novel approach

to feature selection from multiple textual documents, namely GraFS. Contrary to existing approaches, the one introduced in this chapter (i) enables the investigation of the importance of a term into a whole corpus of documents, (ii) incorporates the relationships between terms (co-occurrences) into the feature selection process, (iii) achieves state-of-the-art accuracy in ML tasks such as text classification using fewer features, and (iv) mitigates the effects of the 'curse-of-dimensionality' phenomenon. GraFS has been evaluated by using five datasets and five classifiers. Compared to four well-tried existing feature selection approaches, our initial experimental results show that GraFS increases the accuracy of most text classifiers and decreases the number of features required to achieve 'state-of-the-art' accuracy.

The remainder of the chapter is organized as follows. The proposed feature selection approach is presented in Section 4.2. Section 4.3 reports on the experiments carried out to assess the proposed approach against previous ones. Limitations of our approach, future work directions and concluding remarks are outlined in Section 4.4.

4.2 The proposed approach: Graph-based feature selection

4.2.1 Graph-of-docs text representation

To select the most representative features of a corpus of documents, we build on the graph-of-docs text representation (see Chapter 3). Aiming to facilitate the feature selection process we adapt the graph-of-docs representation to allow a new type of edges to co-exist in a graph, i.e. edges with the 'feature' type (see Figure 4.1). In the expaned version of graph-of-docs, each unique word node selected as a feature is connected to document nodes using edges of the 'feature' type.

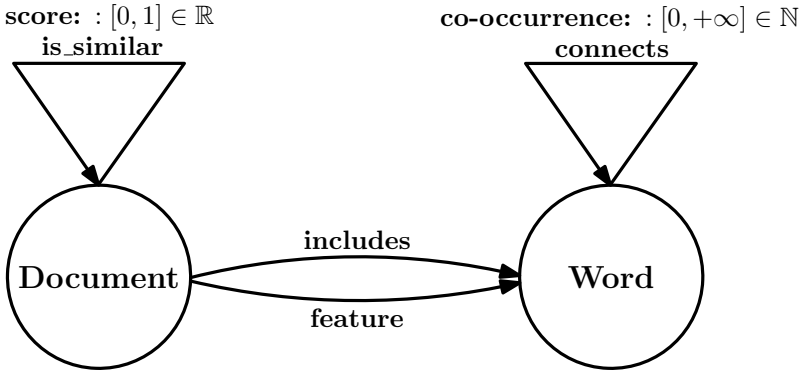


Figure 4.1: The schema of the expanded version of graph-of-docs representation model.

Graph-of-docs enables us to investigate the importance of a term not only within a single document but also within a whole corpus of documents, which in turn augments the quality of the overall feature selection process.

4.2.2 Feature selection

Our approach consists of four steps: (i) creation of a document similarity subgraph; (ii) detection of document communities; (iii) feature selection for each community,

and (iv) feature selection for the whole corpus of documents.

Creation of a document similarity subgraph

We argue that subgraphs from the graph-of-docs graph describing similar documents share common word nodes as well as similar structural characteristics. This enables us to calculate the similarity between two documents by using typical data mining similarity measures, which in turn facilitates the production of a similarity subgraph. The similarity subgraph consists of document nodes and edges of the 'is_similar' type.

Detection of document communities

By exploiting the document similarity subgraph, we detect communities of contextually similar documents using the 'score' property of the 'is_similar' type edges as a distance value. A plethora of community detection algorithms can be found in the literature, including Louvain, Label Propagation and Weakly Connected Components.

Feature selection for each community

Since documents that are in the same community are contextually similar, we assume that it is also more likely that they share common features (see Figure 4.2). Aiming to find the top-N most representative features of each community, GraFS ranks the terms of each community by their document frequency and their PageRank score.

Feature selection for the whole corpus of documents

The final step defines the feature space by merging the top-N features of each community. This reduces the number of the candidate features, something that (i) accelerates the feature selection process, (ii) mitigates the effects of the 'curse-of-dimensionality' phenomenon, and (iii) enables the training of more reliable ML models.

4.3 Experimental evaluation

For the implementation and evaluation of our approach, we used the Python programming language and the scikit-learn ML library (https://scikit-learn.org). The Neo4j graph database (https://neo4j.com) has been utilized for the needs of the graph-of-docs representation. The full code, the documentation and the evaluation results of our experiments are freely available at https://github.com/NC0DER/GraphOfDocs.

4.3.1 Baseline methods

This subsection presents the benchmarks used to evaluate the performance of GraFS. For the implementation of these methods, we used the scikit-learn ML library (implementation details can be found at https://scikit-learn.org/stable/modules/feature_selection.html).

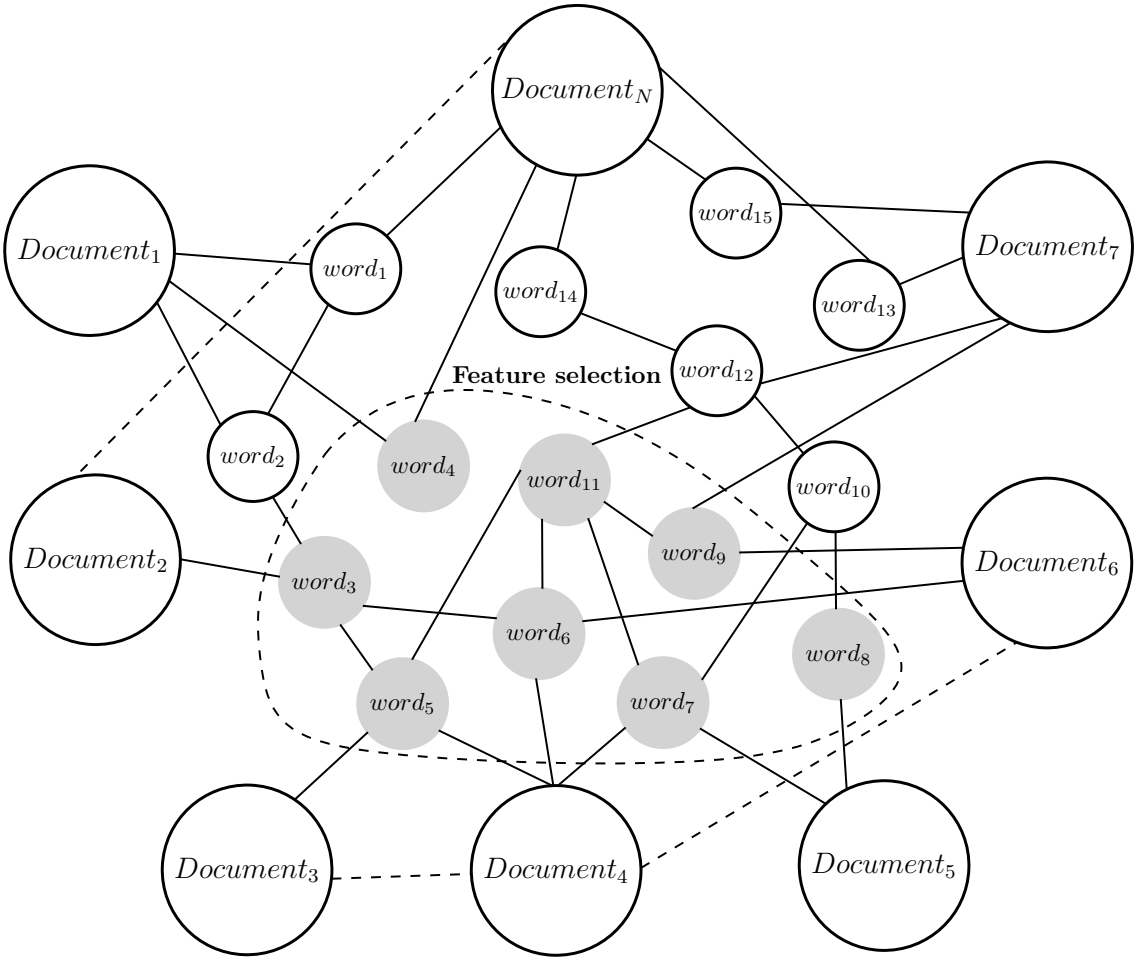


Figure 4.2: Feature selection using the graph-of-docs text representation model. The selected features, shown within the circle, are linked to documents with edges of 'feature' type. Relationships between documents are denoted with dotted lines.

Low Variance Feature Selection (LVAR)

The first benchmark removes the features that do not meet a predefined variance threshold (Aggarwal, 2018). This method is referred to as LVAR in the remainder of this chapter (scikit-learn library, class: sklearn.feature_selection.VarianceThreshold).

Univariate Feature Selection (KBEST)

The second benchmark relies on univariate statistical tests to select the k-best features (Aggarwal, 2018). In particular, it attempts to find correlations between an individual feature and a document class. In this chapter, we adopt the x2 test as our main statistical test. This method is referred to as KBEST in the remainder of this chapter (scikit-learn library, classes: sklearn.feature_selection.SelectKBest and sklearn.feature_sele-ction.chi2).

Feature Selection using a meta-transformer model (META)

The third benchmark uses a meta-transformer model to retain only the features with significant importance. It is assumed that a statistical model (e.g. logistic regression) provides importance metrics for each feature to be considered as a candidate meta-transformer model. Available meta-transformer models include logistic regression, linear SVM and neural networks, as well as more sophisticated methods such as word embeddings (e.g. word2vec (Mikolov et al., 2013c)). In this chapter, we use the linear SVM model, since it performs well regardless of the number of samples or the number of unique features of a dataset. In the remainder of the chapter, this method is referred to as META (scikit-learn library, classes: sklearn.feature_selection.SelectFromModel and sklearn.svm.LinearSVC).

4.3.2 Datasets

This subsection describes the datasets used in our experiments to evaluate the performance of GraFS. These datasets are available at https://github.com/imis-lab/aiai-2020-datasets.

20Newsgroups

We tested the proposed model by utilizing an already preprocessed version of the well-known 20Newsgroups dataset, which is a collection of approximately 20,000 newsgroup documents, partitioned evenly across 20 different news-groups. As far as the multi-class text classification task is concerned, this dataset fits well to the purposes of our experimentations since it provides a large volume of different documents on the same subjects.

Reuters

We tested the proposed model on a preprocessed version of the Reuters dataset, which includes 21,578 news stories; since almost half of them lack the class field, we used only the ones that came along with their class (i.e. 10,377). For each news story, certain attributes were retained; for instance, the 'title' attribute that contains the title of the story and the 'body' attribute that contains the main text of the

news story. In this chapter, we used this dataset to execute experiments related to the multi-class text classification task.

Amazon Reviews

We also tested the proposed model on a preprocessed version of the Amazon Reviews dataset, which contains labeled (positive or negative) reviews of products belonging to different categories (e.g. automotive, electronics, grocery etc.). We picked four product categories (i.e. books, DVD, electronics, kitchen), each having 1000 positive and 1000 negative reviews. We utilized this dataset to conduct experiments related to the opinion mining task.

LingSpam

The LingSpam dataset (Androutsopoulos et al., 2000) contains 2,893 email messages, which are classified either as 'spam' or 'not spam'. We utilized this dataset to conduct experiments related to the spam detection task.

JiraIssues

The JiraIssues dataset concerns the development of 168 software projects including 'Hadoop', 'Spark' and 'Airflow'. It contains information related to 228,969 Jira issues. Each Jira issue in this dataset has the attributes 'description', and 'assignee'. The set of the document classes of the dataset corresponds to the names of the available employees ('assignee' attribute). This dataset was retrieved from the publicly accessible Jira instance of Apache Software Foundation (https://issues.apache.org/jira). We utilized it to execute experiments related to the multi-class text classification task.

Method	Dataset	Hyper-parameter	Values
LVAR	20Newsgroups, Reuters, Amazon, LingSpam, JiraIssues	variance threshold	[0.0005, 0.001, 0.0015, 0.002, 0.003, 0.004, 0.005, 0.01]
GraFS	20Newsgroups, Reuters, Amazon, LingSpam, JiraIssues	top-N	[5, 10, 15, 20, 25, 50, 100, 250, 500]
KBEST	20Newsgroups	k	[1000, 2000, 3000, 5000, 10000, 15000, 20000, 25000, 30000]
KBEST	Reuters, Amazon, JiraIssues	k	[1000, 2000, 3000, 5000, 6000, 7000, 8000, 10000, 14000]
KBEST	LingSpam	k	[250, 500, 1000, 1500, 2000, 2500, 3000, 4000, 5000]

Table 4.1: The hyper-parameters of each feature selection method per dataset.

4.3.3 Experimental setup

To identify the most important words in the entire corpus of documents, we selected to use the PageRank algorithm, since it performs well regardless of the topics of the documents. To identify similar documents needed for the generation of the

document similarity subgraph, we used the Jaccard similarity measure since it deals only with the absence or the presence of a word, ignoring its document frequency. To form communities of similar documents, we used the Louvain community detection algorithm. Finally, we executed several experiments with different hyperparameter values for the LVAR, KBEST and GraFS feature selection methods. Table 4.1 summarizes the values given to these hyperparameters per dataset.

4.3.4 Evaluation

To evaluate the effectiveness of our approach, we assess the contribution of GraFS in the accuracy of widely used text classifiers against the bag-of-words (BOW) text representation and the three domain-agnostic feature selection techniques described in Section 4.1 (see Table 4.2 The text classifiers considered are: naive Bayes (NB), k-nearest neighbors (5NN), logistic regression (LR), neural networks (NN100x50) and linear support vector machines (LSVM). It is noted that in the case of BOW, none of the feature selection techniques has been applied to the specific experiment. Results obtained show that GraFS (i) increases the accuracy in most cases, and (ii) decreases the number of features required to achieve 'state-of-the-art' accuracy (Figure 4.3 – right part). Figure 4.3 illustrates the accuracy of the LSVM classifier per number of selected features for the GraFS, KBEST and LVAR feature selection techniques (additional comparisons can be retrieved from https://github.com/imis-lab/aiai-2020-datasets).

Our approach differs from the existing ones in that it considers the whole corpus of documents (instead of each document separately) and the associated relationships between the words. Thus, the feature set selected using GraFS contains the most influential features of a document corpus. Hence, GraFS reduces the number of the selected features, which in turn mitigates the effects of the 'curse-of-dimensionality' phenomenon, i.e. the production of over-fitted ML models and sparse feature vectors. Contrary to our approach, common feature selection methods that are based on statistics ignore the interconnections between the terms (both within a single document and across the documents of a corpus), which has as effect that more features are required from the text classifiers to perform equally well (see the left graph in Figure 4.3).

4.4 Discussion and conclusions

This chapter introduces a new approach for graph-based feature selection, namely GraFS. To test the proposed approach, we benchmarked GraFS against classical feature selection techniques. The evaluation outcome was very promising; state-of-the-art accuracy has been achieved in the classification of five well-known datasets using fewer features. In any case, our approach demonstrates two limitations: (i) it is unable to select features for outlier documents, i.e. documents that are not similar to any other document, and (ii) it requires significant time to generate the corresponding graph of documents in a disk-based graph database.

Aiming to address the above limitations as well as to integrate our approach into existing works on knowledge management systems, future work directions include: (i) the utilization and assessment of an in-memory graph database in combination

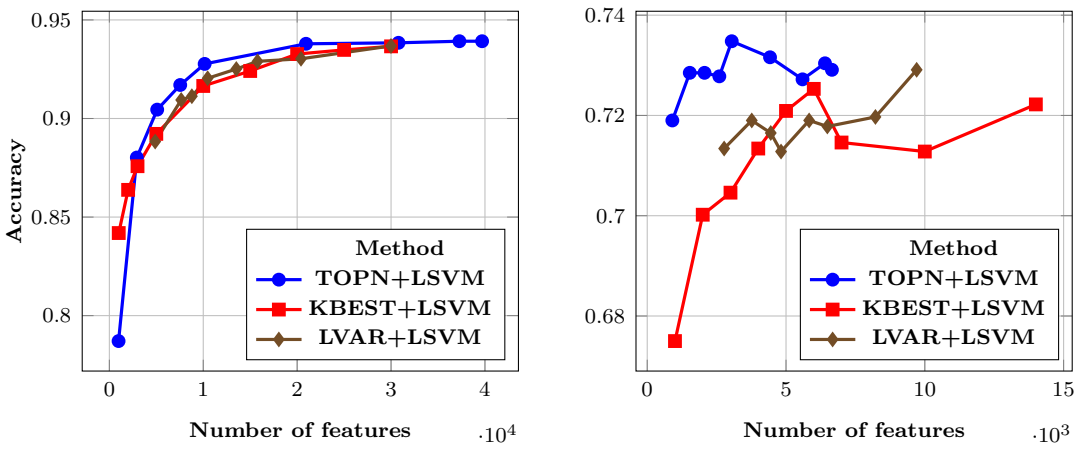


Figure 4.3: Accuracy of the LSVM classifier per number of selected features for the GraFS (TOPN), KBEST and LVAR feature selection techniques on 20Newsgroups (left) and JiraIssues (right) datasets.

	20Newsgroups		**Reuters**		**Amazon**		**LingSpam**		**JiraIssues**	
Method	**acc**	**|f|**	**acc**	**|f|**	**acc**	**|f|**	**acc**	**|f|**	**acc**	**|f|**
GraFS+5NN	0.7192	20942	0.8272	809	0.6786	1065	0.9926	120	0.682	905
GraFS+NB	0.9421	20942	0.8399	7171	0.7273	1940	0.9963	2274	0.6958	6404
GraFS+LR	0.9402	37281	**0.8782***	7171	0.776	4897	**1.0***	120	0.7517	2598
GraFS+NN100x50	**0.9575***	30793	0.8733	8187	0.7403	1940	1.0	758	**0.7542***	6404
GraFS+LSVM	0.9392	39694	0.8737	7171	0.763	4897	0.9963	120	0.7348	3045
BOW+5NN	0.643	62384	0.7582	15514	0.6169	9771	0.8333	16695	0.6486	14539
BOW+NB	0.9361	62384	0.8191	15514	0.7208	9771	0.9963	16695	0.6989	14539
BOW+LR	0.9387	62384	0.8756	15514	0.763	9771	1.0	16695	0.7461	14539
BOW+NN100X50	0.9546	62384	0.8656	15514	0.7273	9771	0.9963	16695	0.741	14539
BOW+LSVM	0.9408	62384	0.8742	15514	0.763	9771	1.0	16695	0.7304	14539
LVAR+5NN	0.6942	4880	0.8209	1482	0.7013	1719	0.8926	5464	0.6493	2771
LVAR+NB	0.94	29992	0.8403	3356	0.737	2906	0.9963	8234	0.7373	5833
LVAR+LR	0.9384	29992	0.8755	7624	0.7792	3637	1.0	8234	0.7373	5833
LVAR+NN100X50	0.9541	29992	0.8724	4870	0.75	1719	1.0	11058	0.7398	6489
LVAR+LSVM	0.9368	29992	0.8746	7624	0.7825	1719	1.0	16695	0.7291	9706
KBEST+5NN	0.721	5000	0.7957	6000	0.724	350	0.9778	5000	0.6644	4000
KBEST+NB	0.9374	25000	0.8354	7000	0.75	500	0.9963	3000	0.6952	14000
KBEST+LR	0.9389	30000	0.8764	10000	0.7727	3000	1.0	1000	0.7461	6000
KBEST+NN100X50	0.9564	30000	0.8705	14000	0.7575	1000	1.0	1000	0.7423	10000
KBEST+LSVM	0.9366	30000	0.8737	14000	0.763	6000	1.0	1000	0.7253	6000
META+5NN	0.6542	14907	0.8002	2494	0.6623	2731	0.8704	2509	0.6329	2942
META+NB	0.9387	14907	0.8376	2494	0.737	2731	0.9963	2509	0.6989	2942
META+LR	0.9376	14907	0.876	2494	**0.789***	2731	1.0	2509	0.7461	2942
META+NN100X50	0.952	14907	0.8701	2494	0.75	2731	1.0	2509	0.7467	2942
META+LSVM	0.9408	14907	0.8746	2494	0.7727	2731	1.0	2509	0.731	2942

Table 4.2: Accuracy score (acc) and number of features (|f|) per text classifier for each feature selection technique on the five selected textual datasets. * and bold font highlights the best method for a specific dataset as far as the accuracy score and the number of features are concerned.

with Neo4j; (ii) the exploitation of link prediction algorithms to deal with outlier documents; (iii) the application of graph and word embedding techniques, and (iv) the integration of our approach into collaborative argumentation environments where the underlying knowledge is structured through semantically-rich discourse graphs (e.g. integration with the approaches described in (Kanterakis et al., 2019) and (Karacapilidis et al., 2009)).

Chapter 5

Using knowledge graphs for link prediction

We consider the prediction of future research collaborations as a link prediction problem applied on a scientific knowledge graph. To the best of our knowledge, this is the first work on the prediction of future research collaborations that combines structural and textual information of a scientific knowledge graph through a purposeful integration of graph algorithms and natural language processing techniques. Our work: (i) investigates whether the integration of unstructured textual data into a single knowledge graph affects the performance of a link prediction model, (ii) studies the effect of previously proposed graph kernels-based approaches on the performance of an ML model, as far as the link prediction problem is concerned, and (iii) proposes a three-phase pipeline that enables the exploitation of structural and textual information, as well as of pre-trained word embeddings. We benchmark the proposed approach against classical link prediction algorithms using accuracy, recall, and precision as our performance metrics. Finally, we empirically test our approach through various feature combinations with respect to the link prediction problem. Our experimentations with the new COVID-19 Open Research Dataset demonstrate a significant improvement of the abovementioned performance metrics in the prediction of future research collaboration.

5.1 Introduction

The development of knowledge graph-based approaches for predicting future research collaborations has received increasing attention in recent years (Nathani et al., 2019; Vahdati et al., 2018). In these approaches, a scientific article that has been written by two researchers denotes implicitly a collaboration between them (Ponomariov & Boardman, 2016). In general, the majority of the existing knowledge graph based approaches builds on concepts and methods from graph theory to infer knowledge that is not explicitly provided, exploiting the structural characteristics of the corresponding research graph (Q. Wang et al., 2017). However, these approaches ignore the extremely useful (but unstructured) textual data that are in most cases available in documents such as scientific articles and reports (Veira et al., 2019); as a consequence, they are not able to meaningfully incorporate both structural and textual information into their knowledge graph.

Aiming to overcome the above issues, this chapter proposes the development and

deployment of a novel scientific knowledge graph that enables the co-existence and joint utilization of structured as well as unstructured data, such as author, document and word nodes. Expanding on past work, the documents of a scientific graph are represented as a graph of documents (graph-of-docs approach) (Giarelis et al., 2020a, 2020b; Giarelis et al., 2020c), which facilitates the discovery of future research collaborations by using prominent link prediction algorithms. In addition, recent advances in graph mining enable our approach to calculate the similarity between two graph-of-docs representations, using graph similarity techniques (e.g., graph kernels and graph neural networks). For the implementation of our approach, we use the Neo4j graph database (https://neo4j.com, accessed on 25 April 2021) as far as the implementation and storage of the proposed knowledge graph is concerned, the Python programming language for coding purposes, and the TensorFlow and scikit-learn Machine Learning (ML) libraries for the training of our prediction models.

To evaluate the approach described in this chapter, we benchmark it against different well-established combinations of graph-related measures. For the ML models proposed, we include the accuracy, precision, and recall metrics as they are considered the most robust and popular metrics for performance assessment. In the experiments included in this work, the COVID-19 Open Research Dataset (CORD-19) is utilized. Having in mind that the size of the dataset could affect our approach in a variety of ways (e.g., overfits or underfits), we consider ten (10) different datasets, extracted from the original dataset. The experimental results lead to a significant improvement of the link prediction accuracy, compared to the ML models considered. The final version of code, datasets, and evaluation results of our work are openly accessible on GitHub (https://github.com/nkanak/cordkel, accessed on 25 April 2021). The main contributions of this chapter are the following:

- we investigate whether the integration of unstructured textual data into a single knowledge graph affects the performance of a link prediction model;
- we study the effect of previously proposed graph kernels-based approaches on the performance of an ML model, as far as the link prediction problem is concerned;
- we propose a three-phase pipeline that enables the exploitation of structural and textual information, as well as of pre-trained word embeddings;
- we empirically test our approach through various feature combinations with respect to the link prediction problem.

To the best of our knowledge, this is the first work on the prediction of future research collaborations that combines structural and textual information of a scientific knowledge graph through a purposeful integration of graph algorithms and natural language processing (NLP) techniques. The remainder of this chapter is organized as follows. Background concepts and related work are introduced in Section 5.2. In Section 5.3, our approach is presented in a thorough and elaborated fashion, while the experiments carried out to evaluate our approach are reported in Section 5.4. The novelty, future work directions as well as the limitations of the proposed approach are discussed in Section 5.5.

5.2 Background and Related Work

Our approach in predicting future research collaborations makes use of a set of graph theory, graph-based text representation, NLP, and knowledge graph techniques.

5.2.1 Predicting Future Research Collaborations

As far as link prediction techniques for the discovery of future research collaborations are concerned, works closest to ours are proposed in (Guns & Rousseau, 2014; Huang et al., 2008; Liben-Nowell & Kleinberg, 2007; Sun et al., 2011; Yu et al., 2014). In particular, in (Liben-Nowell & Kleinberg, 2007), the authors rely only on a co-authors network's topology aspects, and the proximity of a pair of nodes in order to predict future research collaborations between them. In (Sun et al., 2011), the authors recommend an approach, where structural properties are used to calculate the probability of future research collaborations in heterogeneous bibliographic networks, with varying types of nodes (e.g., papers, authors, venues, topics) and edges (e.g., publish, write, mention, cite, contain). They leverage the multiple relationships among the papers to improve the overall accuracy of their link prediction algorithm.

In (Guns & Rousseau, 2014), the authors predict potential research collaborations by combining link prediction techniques with a random forest classifier. For each pair of nodes of a co-authorship network, a wide range of topology-based measures such as Adamic Adar and Common Neighbors, are calculated and, then, combined with location-based characteristics related to the authors. Hence, the position of the authors in the co-authorship network and their location are used to generate future collaboration proposals. In (Huang et al., 2008), the authors construct a co-authorship network that represents research collaborations from 1980 to 2005 in the field of computer science. In this case, a variety of graph theory algorithms and classical statistical techniques are employed in order to extract the co-authorship network properties. The dataset used is composed of 451,305 papers from 283,174 authors.

In (Yu et al., 2014), the authors use medical co-authorship networks along with link prediction algorithms to predict future research collaborations. For a given author, potential collaborators are identified, as long as they complement her skillset. For each pair of author nodes common topological and structural measures are extracted, such as Adamic Adar, Common Neighbors and Preferential Attachment, and multiple ML models are utilized for the prediction of possible future collaborations.

Adopting a broader link prediction perspective, additional works (e.g., (Fire et al., 2011; Julian & Lu, 2016; Panagopoulos et al., 2017)) describe the task of predicting possible relationship types between nodes of a particular network from various and interesting scopes (such as social networks and friendship suggestions).

5.3 The Proposed Approach

We first compose a scientific knowledge graph containing both structured and unstructured textual data (Section 5.3.1). The unstructured textual data are integrated into the knowledge graph via graph-of-docs text representation (see Sec-

tion 2.2). Then, a series of graph measures and graph kernels (see Section 2.3) are employed for feature extraction associated to both textual and structural information concerning the entities of the knowledge graph (Section 5.3.2). Finally, we map the whole problem of predicting future research collaborations to a link prediction task, by utilizing the features produced in the previous step for the deployment of an ML model (Section 5.3.2). To sum up, our approach is divided in three phases, namely Knowledge Graph Construction, Feature Extraction and Link Prediction. Figure 5.1 illustrates the phases of the proposed approach.

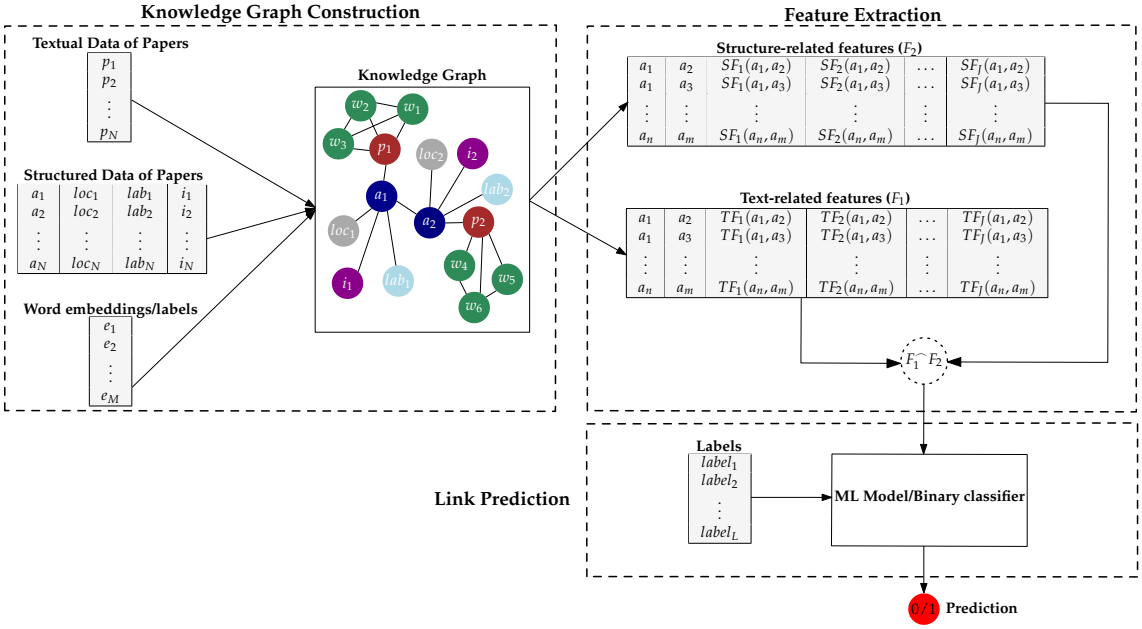


Figure 5.1: The phases of the proposed approach. p_x denotes nodes of the '*Paper*' type. w_x denotes nodes of the '*Word*' type. a_x denotes nodes of the '*Author*' type. loc_x denotes nodes of the '*Location*' type. i_x denotes nodes of the '*Institution*' type. lab_x denotes nodes of the '*Laboratory*' type. The word embedding of a word (w_x) is denoted by e_x. SF_x and TF_x denote structure-related and text-related features, respectively. $label_x$ denotes the label (0 or 1) that corresponds to the sample x of the given dataset. $F_1 \hat{} F_2$ denotes the concatenation of the structure-related and text-related features, aiming to generate the feature vector of the sample x of the given dataset.

5.3.1 Knowledge Graph Construction

The nature of knowledge graphs enables the co-existence of multiple types of entities and relationships in the same data schema. Specifically, our scientific knowledge graph includes entity nodes with types such as '*Paper*', '*Author*', '*Laboratory*', '*Location*', '*Institution*' and '*Word*', as well as types of relationship edges such as '*is_similar*', '*cites*', '*writes*', '*includes*', '*connects*', '*co_authors*' and '*affiliates_with*' (see Figure 5.2).

As far as entity nodes are concerned, nodes that are labeled as '*Paper*' express scientific documents; the '*Author*' entity represents an author of a scientific paper or document the laboratory of an author and its location are represented as a 'Laboratory' and 'Location' entity, respectively; each '*Institution*' entity corresponds to

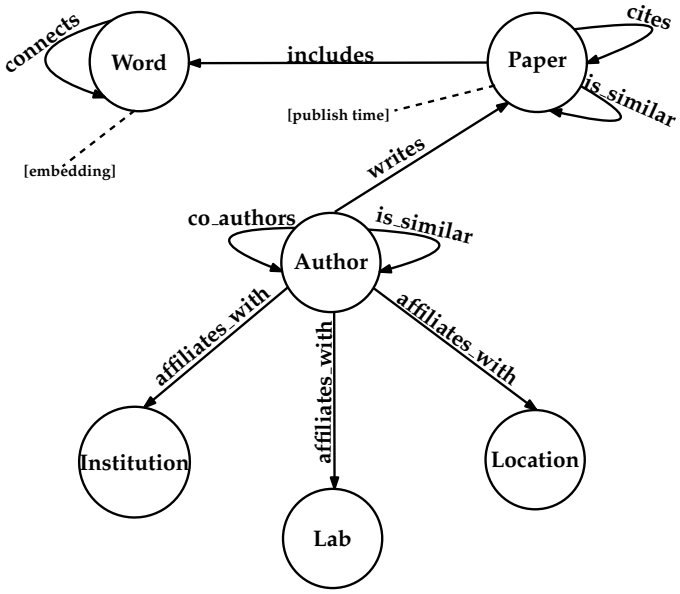


Figure 5.2: The data schema of the proposed scientific knowledge graph. Dotted lines connect properties associated with the entities of the knowledge graph.

the institution of an author; finally, for the representation of a unique word of a scientific paper a '*Word*' entity is utilized.

With respect to relationship edges, the existence of a specific word to a certain paper is depicted with the '*includes*' relationship, which connects a '*Paper*' with a '*Word*' entity. Similarly, the co-occurrence of a pair of words within a predefined sliding text window is modeled through a '*connects*' relationship, which is only applicable between two '*Word*' entities. The graph-of-docs representation of the textual data of the available papers can be defined as the subgraph constructed by the '*Word*' and '*Paper*' entities, as well as the '*includes*', '*connects*' and '*is_similar*' relationships

Furthermore, the '*is_similar*' relationship links pairs of '*Paper*' or '*Author*' nodes. In the former case, it denotes the similarity between two papers represented as graph-of-docs. In the latter, it denotes the similarity between the documents associated to the two authors. We can produce an author similarity subgraph by considering the subgraph that is composed of the '*Author*' entities and the '*is_similar*' relationships.

Finally, the citation of a paper from another one is depicted as a '*cites*' relationship between two '*Paper*' nodes; the writer of a scientific document is represented with a '*writes*' relationship between an '*Author*' with a '*Paper*' entity; accordingly, the relationship of an author with an institution and its location and laboratories are depicted with an '*affiliates_with*' relationship connecting an '*Author*' entity with an '*Institution*', '*Location*' or '*Laboratory*' entity, respectively; The '*co_authors*' relationship denotes a research collaboration between the connected '*Author*' entities. Similarly to the the graph-of-docs and authors similarity subgraphs, we can construct a co-authors' subgraph, by isolating the available '*Author*' entities and the '*co_authors*' relationships from the original scientific knowledge graph.

The produced knowledge graph enables the user to gain insights about a variety of tasks through the employment of well-studied graph algorithms. Such tasks would be recommending similar research work, finding nearby experts based on the '*Location*' entities, and discovering future research collaborations; this chapter focuses

on the last of these tasks.

5.3.2 Feature Extraction

The proposed knowledge graph enables the extraction of informative features for each entity. Each feature of an entity encapsulates either structural or textual information related to the whole knowledge graph; in this chapter, we concentrate on the extraction of features that describe the relationship between two '*Author*' nodes. To extract structure-related features, we can use graph measures or indices such as common neighbors, preferential attachment and Adamic Adar. To extract text-related features, we can use graph similarity techniques including graph neural networks and graph kernels (see Section 2.3); for the experiments reported in this chapter, we employ graph kernels. It is also noted here that text-related features can be further enriched, either by attaching labels or word embeddings to the '*Word*' nodes of each paper of the knowledge graph (see Section 2.5.1). Finally, for the generation of the feature vector of each sample of a given dataset, we concatenate the structure-related and text-related feature vectors (Figure 5.1).

5.3.3 Link Prediction

For the discovery of future research collaborations, a wide range of link prediction and ML techniques are employed. Specifically, in our approach, the issue of suggesting future research collaborations is reduced to the common binary classification problem. Such an approach enables us to construct a link prediction algorithm by predicting the presence (or absence) of a '*co_authors*' relationship for a pair of '*Author*' entities. Various binary classifiers, including logistic regression, k-nearest neighbors, linear support vector machines, decision tree, and neural networks have been considered in the related literature (Aggarwal, 2018). In the experiments reported in this chapter, we have used the logistic regression and neural network ones.

5.4 Experimental Evaluation

To implement and evaluate our approach, we utilized the Python programming language, the TensorFlow (Abadi et al., 2016) and scikit-learn (Pedregosa et al., 2011) ML libraries, as well as the *GraKeL* (Siglidis et al., 2020) library for graph kernels. In addition, we used the word embeddings from the *CORD-19 Swivel text embedding* module of the TensorFlow Hub (available online: https://tfhub.dev/tensorflow/cord-19/swivel-128d/3, accessed on 25 April 2021). Furthermore, we utilized the Neo4j database (https://neo4j.com, accessed on 25 April 2021), aiming to store the instance of the graph-of-docs representation and the knowledge graph. All the material related to our experiments and the evaluation can be also found at https://github.com/nkanak/cordkel (accessed on 25 April 2021).

5.4.1 Evaluation Metrics

To evaluate our approach, we use three widely used metrics, namely *accuracy*, *precision* and *recall*. These metrics rely on the number of true positives ($|TP|$), true

negatives ($|TN|$), false positives ($|FP|$), and false negatives ($|FN|$) predictions of each classification model. In particular, accuracy is the percentage of correct predictions. Recall estimates the ability of the classifier to find all the positive samples. Precision estimates the ability of the classifier not to label as positive a negative sample. The accuracy metric is defined as:

$$accuracy = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|}$$

The precision metric is defined as:

$$precision = \frac{|TP|}{|TP| + |FP|}$$

The recall metric is defined as:

$$recall = \frac{|TP|}{|TP| + |FN|}$$

To further evaluate our approach, we also take into account the values of the *average cross-entropy* metric, which we utilize as a loss function, whenever it is possible. As opposed to the accuracy, precision and recall metrics, cross-entropy is not affected by the order of the training data. In particular, binary cross-entropy measures the dissimilarity between the distribution of the real value y and the predicted value $\hat{y}$ of a model. The binary cross-entropy is defined as follows:

$$H\left(y, \hat{y}\right) = -y \cdot \log\left(\hat{y}\right) - (1 - y) \cdot \log\left(1 - \hat{y}\right)$$

Accordingly, we can define the average cross-entropy among N samples as:

$$J = \frac{1}{N} \sum_{n=1}^{N} H\left(y_n, \widehat{y_n}\right) = -\frac{1}{N} \sum_{n=1}^{N} y_n \cdot \log\left(\widehat{y_n}\right) + (1 - y_n) \cdot \log\left(1 - \widehat{y_n}\right)$$

Finally, we use the micro sign test to investigate whether there is any statistical significance in improvement ($p < 0.05$) of each metric between the baselines and the proposed models.

5.4.2 The CORD-19 Dataset

The COVID-19 Open Research Dataset (CORD-19) (L. L. Wang et al., 2020) includes 63,000 scientific articles, related to COVID-19 and other coronaviruses. It is available from the Allen Institute for AI and Semantic Scholar. The scientific papers in CORD-19 have been retrieved from several famous medical-related repositories, such as PubMed Central (PMC), bioRxiv, Elsevier, World Health Organization (WHO) and medRxiv. Each scientific paper in CORD-19 is associated with an array of attributes, including '*publish time*', '*citations*', '*title*', '*abstract*' and '*authors*'. Most of the scientific papers (i.e., 51,000) also has a '*full text*' attribute. It is apparent that this dataset benefits the research community as far as the COVID-19 related research is concerned. However, due to the unstructured nature of the textual data included in the dataset, many shortcomings exist in terms of gaining useful insights. As already mentioned in the literature, by combining a graph-based text representation and a knowledge graph, we are able to generate a semi-structured representation

of the data, which alleviates the existing drawbacks (Veira et al., 2019; Q. Wang et al., 2017; Z. Wang & Li, 2016). To construct the scientific knowledge graph, we use the '*publish time*', '*abstract*' and '*authors*' attributes of the scientific papers. Due to hardware limitations, we do not utilize the *full text* of each scientific paper. Figure 5.3 illustrates different snapshots of the knowledge graph that corresponds to the CORD-19 dataset. The different node and edge colors highlight the heterogeneity of the produced graph. For illustration purposes, we limit the depicted nodes to 1000, 2000, 3000 and, respectively. As Figure 5.3 shows, the number of relations between the nodes increases radically in proportion to the number of nodes.

Generation of Datasets for Predicting Future Research Collaborations

We generate ten (10) new balanced datasets using the CORD-19, aiming to investigate whether our approach performs well, irrespective of the properties of a dataset. To create a sample of each dataset, we use the '*authors*' similarity subgraph an the '*co_authors*' subgraph, which is composed of the '*co_authors*' edges. We note that edges also have as a property the first collaboration year of two authors.

Table 5.1 reports the number of samples of the training and testing subset for each one of the produced datasets. The samples of each dataset have been randomly chosen. Table 5.2 describes the features of each sample of the dataset. We conduct a list of experiments using different combinations of features, aiming to investigate whether the performance of a ML model is affected by the features under consideration (Table 5.3). We note that each training sample is selected from an earlier instance of the '*co_authors*' subgraph, which is generated from the '*co_authors*' edges that first appeared within or before the year 2013. Accordingly, the testing subset incorporates '*co_authors*' edges generated after 2013. We separate the dataset by using the time to ensure that any data leakage is avoided among the testing and the training subset (Liben-Nowell & Kleinberg, 2007).

Dataset ID	**|Training Subset Samples |**	**|Testing Subset Samples |**
D1	1000	330
D2	1000	330
D3	1000	330
D4	1000	330
D5	1000	330
D6	1000	330
D7	6000	1890
D8		9900
D9	1000	330
D10	1000	330

Table 5.1: Number of training samples (|Training subset samples|) and number of testing subset samples (|Testing subset samples|) of each dataset.

5.4.3 Baseline Feature Combinations

We benchmark the feature combinations extracted by our approach against three baseline combinations, namely AA_J, AA and J (see Table 5.3 for a detailed expla-

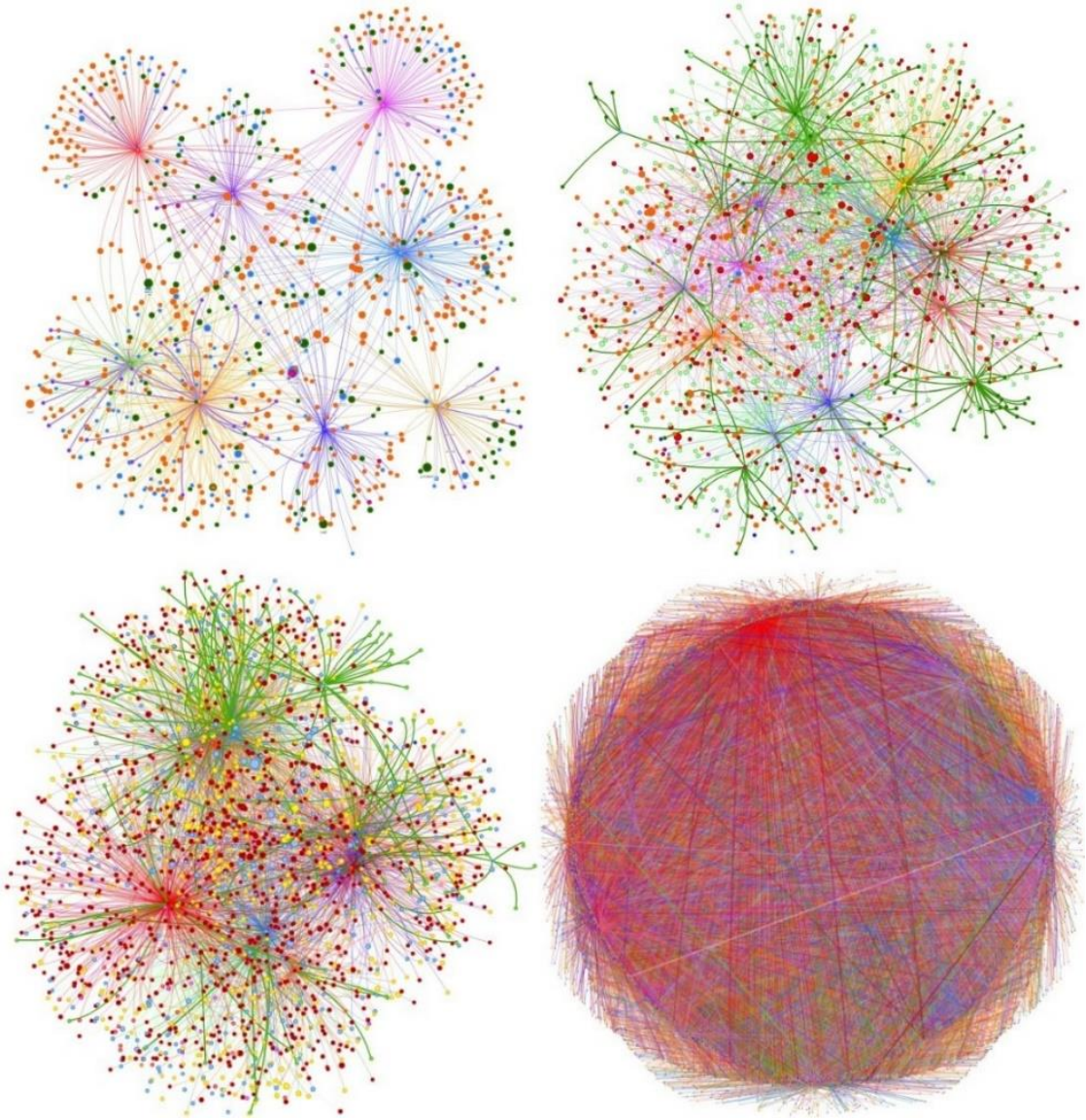


Figure 5.3: Snapshots of the knowledge graph that is generated from the CORD-19 dataset: limited to 1000 (**upper-left**), 2000 (**upper-right**), 3000 (**bottom-left**) and 30,000 (**bottom-right**) nodes. The different node and edge colors highlight the heterogeneity of the produced graph.

Feature	Description	Type
adamic adar	The sum of the inverse logarithm of the degree of the set of common neighbor 'Author' nodes shared by a pair of nodes.	Structural (SF)
common neighbors	The number of neighbor 'Author' nodes that are common for a pair of 'Author' nodes.	Structural (SF)
preferential attachment	The product of the in-degree values of a pair of 'Author' nodes.	Structural (SF)
total neighbors	The product of the in-degree values of a pair of 'Author' nodes.	Structural (SF)
pyramid match	The similarity of the text of the graph-of-docs graphs of two nodes of 'Author' type using the Pyramid match graph kernel. The Propagation graph kernel incorporates the terms, the corresponding label of each term and the structure of the text into its formula, aiming to calculate the similarity between two texts.	Textual (TF)
propagation	The similarity of the text of the graph-of-docs graphs of two nodes of 'Author' type using the Propagation graph kernel. The Propagation graph kernel incorporates the terms, the corresponding word embedding of each term and the structure of the text into its formula, aiming to calculate the similarity between two texts.	Textual (TF)
weisfeiler pyramid match	The similarity of the text of the graph-of-docs graphs of two nodes of 'Author' type using the Weisfeiler Lehman framework and the Pyramid match graph kernel. The Weisfeiler Pyramid match graph kernel incorporates the terms, the corresponding label of each term and the structure of the text into its formula, aiming to calculate the similarity between two texts.	Textual (TF)
jaccard	The similarity of the text of the graph-of-docs graphs of two nodes of 'Author' type using the Jaccard coefficient. The Jaccard index deals only with the absence or the presence of a term into a text.	Structural and Textual (SF and TF)
Label	It denotes an edge of the 'co_authors' type between two nodes of the 'Author' type. A positive value (1) represents the existence, while a negative value (0) represents the absence of the edge.	Class

Table 5.2: The features of each sample of the extracted datasets. A feature is associated with either a textual or a structural relationship of two authors.

Feature Combination Name	Features Included	Proposed In
ALL	Adamic Adar, Common Neighbors, Preferential attachment, Total Neighbors, Pyramid match, Weisfeiler Pyramid match, Jaccard, Propagation	(Giarelis et al., 2020c)
PM	Pyramid Match	(Nikolentzos et al., 2017b)
WPM	Weisfeiler Pyramid match	(Nikolentzos et al., 2017b)
AA_J (baseline)	Adamic Adar, Jaccard	(Giarelis et al., 2020c)
AA (baseline)	Adamic Adar	(Adamic & Adar, 2003)
p	Propagation	(Neumann et al., 2016)
J (baseline)	Jaccard	(Jaccard, 1901; Yu et al., 2014)
AA_WPM	Adamic Adar, Weisfeiler Pyramid match	
AA_P	Adamic Adar, Propagation	
AA_PM	Adamic Adar, Pyramid match	

Table 5.3: The various features combinations in order to test how the different combinations affect the performance of the ML models in link prediction.

nation):

- **AA_J** (Giarelis et al., 2020c): It combines the structural information of the knowledge graph using the Adamic Adar measure as well as the textual similarity of the papers of two authors using the Jaccard index.
- **AA** (Adamic & Adar, 2003): It utilizes the structural information of the knowledge graph using the Adamic Adar measure.
- **J** (Jaccard, 1901; Yu et al., 2014): It utilizes the textual similarity of the papers of two authors using the Jaccard index.

5.4.4 Evaluation Results

The evaluation of our approach is achieved by assessing the sensitivity of the ML models towards the various feature combinations, regarding the binary classification problem. The binary classifiers used in our experiments are logistic regression (LR) and neural network (NN); for the training of the NN model, we use five epochs, the Adam optimizer, two hidden layers with 50 and 25 units, respectively, while we utilize the average binary cross-entropy as a loss function. Further experimentations with a plethora of classifiers and various hyperparameter configurations is available online: https://github.com/nkanak/cordkel (accessed on 25 April 2021). As already mentioned in Section 5.4.1, our performance metrics include the accuracy, precision and recall of the binary classifiers. In addition, we test for statistical significance against the AA_J feature combination, since it has the best performance across all the baseline combinations. The experimental results are summarized in Table 5.4 and Table 5.5.

Feature Combination	Accuracy	Recall	Precision
ALL	0.6588	0.9963*	0.6345
J	0.5093	0.0233	**1.0**
AA	0.9818	0.9643	0.9995
AA_J	0.9834	0.9671	0.9998
p	0.6669	0.5589	0.8157
PM	0.8380	0.6965	0.9752
WPM	0.9476	0.9044	0.9905
AA_P	0.9652	0.9923*	0.9625
AA_PM	0.9980*	0.9966*	0.9995
AA_WPM	**0.9986***	**0.9977***	0.9995

Table 5.4: Performance of the logistic regression classifier for each feature combination. * indicates statistical significance in improvement ($p < 0.05$) for each evaluation metric using the micro sign test against the AA_J baseline.

The obtained results indicate that in many cases (e.g., AA_PM or AA_WPM) the inclusion of a text-related feature increases the average accuracy, precision and recall scores, while still remaining unaffected by the nature of the binary classifier. Additionally, it is worth mentioning that our feature combinations correctly predict the majority of the future collaborations, by observing the average precision score compared to the average recall score. The best average accuracy and recall scores

Feature Combination	Accuracy	Recall	Precision	Train Los	Test Loss	Abs Loss Difference
ALL	0.9908	**0.9931***	0.9886	0.1020	0.0499	0.0521
J	0.5093	0.0233	**1.0**	0.6647	0.6858	**0.0211**
AA	0.9922	0.9850	0.9995	0.1303	0.0497	0.0806
AA_J	0.9925	0.9856	0.9995	0.1097	0.0413	0.0684
p	0.6954	0.5045	0.8624	0.6289	0.6057	0.0232
PM	0.8452	0.7085	0.9816	0.3219	0.3990	0.0771
WPM	0.9248	0.8590	0.9905	0.2612	0.2390	0.0222
AA_P	0.9923	0.9851	0.9995	0.1311	0.0464	0.0847
AA_PM	**0.9940***	0.9886*	0.9995	0.1281	0.0395	0.0886
AA_WPM	0.9923	0.9870	0.9995	0.1108	0.0372	0.0736

Table 5.5: Performance of the neural network classifier for each feature combination. * indicates statistical significance in improvement ($p < 0.05$) for each evaluation metric using the micro sign test against the AA_J baseline. Average binary cross-entropy between real and predicted label value is considered as the train and test loss.

are achieved by the LR classifier, using the AA_WPM feature combination. Thus, by combining these two features we get the optimal outcome. The best average precision score is achieved by both classifiers, using the J feature combination. However, by using the J feature combination, both classifiers have a low recall score, which means that they always classify all the samples of a test dataset as candidate future collaborations, something that is totally wrong.

In addition, it is revealed that features including '*total_neighbors*', '*preferential_attachment*' and '*common_neighbors*' (appearing in the ALL feature combination) add noise to the LR classifier, but not to the NN classifier; this is due to the fact that NNs are able to capture more complex and non-linear connections between the given features and the predicted values than LR models. Specifically, we note that LR models are built on the assumption that the independent variables are linearly related to the log of probabilities of features. However, in most real-world applications, this relationship tends to be non-linear, thus making LR models behave poorly. Moreover, logistic regression requires average or no collinearity between features, which may not always be the case (e.g., '*common_neighbors*' and '*jaccard*' features may be highly correlated). On the other hand, LR models offer greater interpretability, as we can estimate the importance of the features by examining the corresponding model coefficients. Such a feature importance estimation can lead to fine-tuned systems, by iteratively disregarding less important features until a local optimum is reached.

It is also observed that metrics or kernels for calculating textual similarities, which incorporate into their formula the terms with their corresponding attributes as well as the structure of the text (e.g., WPM), perform better compared to metrics that deal only with the absence or the presence of a term (e.g., J). By considering the abovementioned remarks, we conclude that using both textual and structural characteristics of a scientific knowledge graph leads to more accurate ML models and less prone to overfitting. Moreover, the NNs are optimized towards minimizing average binary cross-entropy, which can lead to more dynamic and robust models as cross-entropy measures information content and is not as much sensitive to the order of the training data as accuracy-based metrics (see Figure 5.4).

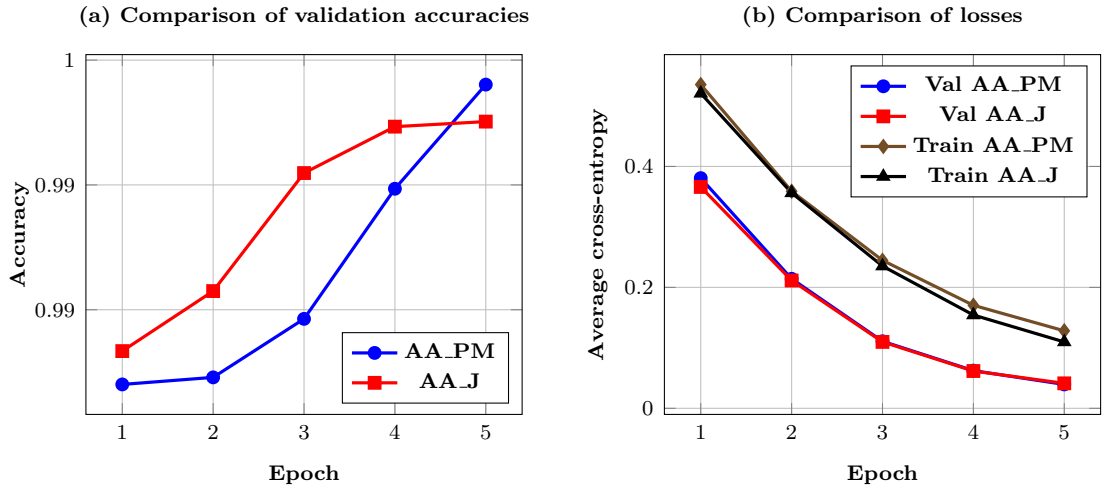


Figure 5.4: (**a**) Comparison of validation accuracies of the NN model using the AA_PM and the AA_J feature combinations; (**b**) Comparison of cross-entropy of the NN model using the AA_PM and the AA_J feature combinations.

5.5 Discussion and conclusions

In this chapter, we apply link prediction techniques on scientific knowledge graphs for recommending future research collaborations. Our three-phase approach incorporates both structured and unstructured textual data in every knowledge graph. To do so, we leverage the graph-of-docs text representation. For our experiments, we construct ten datasets using the CORD-19 dataset. We benchmark our approach against several feature combinations. Our approach demonstrates a significant improvement of the average accuracy, precision and recall of the task of predicting future collaborations, while, also, minimizing the average cross-entropy, as far as the NN models are concerned.

The proposed approach diverges from the state-of-the-art ones since it takes into consideration both the textual similarity of the papers of each set of authors as well as the associated structural information. The approaches that deal solely with unstructured textual data rely on NLP methods and text representations, which require the construction of huge feature spaces. As a result, the 'curse-of-dimensionality' phenomenon re-emerges.

Key enablers that drive the development of the proposed approach are the availability of large computing power, the existence of big volumes of unstructured textual and social network data (e.g., data from a research graph), as well as the availability of a range of well-tried and powerful ML software libraries for analyzing complex graph-structured data (Fey & Lenssen, 2019; Maiya, 2020). Building on a meaningful and flexible combination of textual and structural information related to an author and his papers, our approach enables ML models to better identify future research collaborations. Simply put, our approach improves significantly the quality of the prediction, while enabling ML models to interpret more complex data patterns. However, diverse problems and limitations still exist; these refer to: (i) the time to construct the knowledge graph, (ii) the ability of the graph-based models to add new nodes (e.g., author or paper nodes) to an existing scientific knowledge graph, and (iii) the performance of the models on graphs with rich node attribute

information (e.g., nodes having high-dimensional embeddings).

Future work directions include: (i) the experimentation with graph mining architectures such as graph neural networks and inductive representation learning methods (e.g., GraphSAGE (Hamilton et al., 2017a)); (ii) the embedment of explainability features in the predictions provided by the proposed approach (Linardatos et al., 2020; Lundberg & Lee, 2017; Ying et al., 2019a); (iii) the exploitation of the rest of the information available in the scientific knowledge graph (e.g., the location, the whole text and the institution of the authors) (Molokwu & Kobti, 2019; Pho & Mantzaris, 2020; P. V. Tran, 2018).

Chapter 6

On a meaningful integration of word embeddings and graph-based text representations

This chapter reports on a meaningful integration of techniques and algorithms from the fields of natural language processing, graph representation learning and word embeddings to assist project managers in the task of personnel selection. To do so, our approach initially represents multiple textual documents as a single graph. Then, it computes word embeddings through representation learning on graphs and performs feature selection. Finally, it builds a classification model that is able to estimate how qualified a candidate employee is to work on a given task, taking as input only the descriptions of the tasks and a list of word embeddings. Our approach differs from the existing ones in that it does not require the calculation of key performance indicators or any other form of structured data in order to operate properly. For our experiments, we retrieved data from the Jira issue tracking system of the Apache Software Foundation. The evaluation results show, in most cases, an increase of 0.43% in the accuracy of the proposed classification models when compared against a widely-adopted baseline method, while their validation loss is significantly decreased by 65.54%.

6.1 Introduction

Nowadays, artificial intelligence (AI) has changed our life dramatically by improving user experience, recommending products, making predictions, automating repetitive tasks and so on. Entertainment, transportation, e-commerce, fraud detection, online customer support and medicine are just a few examples of the fields where artificial intelligence has been successfully applied on (Abduljabbar et al., 2019; Awoyemi et al., 2017; Kanakaris et al., 2021a; Powell et al., 2020; Sezer & Altan, 2021; Vanneschi et al., 2018). Furthermore, modern Machine Learning (ML) models enable the forecasting of various ever-changing phenomena using time series data and feature selection techniques. For example, ML models may assist in predicting (i) the future price of crude oil (Karasu et al., 2020), (ii) the speed of wind (Altan et al., 2021), and (iii) the economic trend of a country using time series analysis (Altan & Karasu, 2019). Despite the radical changes and improvements that AI and ML offer to the aforementioned sectors, there has not been any significant progress as far as project

management and issue tracking process are concerned (Dam et al., 2019).

So far, the majority of existing project management systems (e.g. Jira, Wrike) assist users in mainly planning and monitoring their projects and the corresponding tasks. However, they unintentionally hide important information concerning the company itself and do not fully take advantage of the complex multidimensional data found in the hosted projects (Haidabrus et al., 2021). Moreover, the existence of numerous departments within a company and the diversity of viewpoints among employees make it difficult to identify and absorb the related information (Bjorvatn & Wald, 2018).

By utilizing well-tested and broadly accepted ML techniques, project management systems are able to detect valuable information and provide insights (Mahdi et al., 2021). Hence, new opportunities can be created ranging from the ability to recommend solutions and automate simple tasks (e.g. reporting) to solving more complex problems such as finding candidate employees for a list of tasks or even automatically assigning the employees to the available tasks (Azzini et al., 2018).

As far as the process of task assignment is concerned, several approaches can be found in the literature (Fatima et al., 2020). These approaches are divided into two distinct categories, namely the 'operations research'-based and the ML-based ones. The former view the problem of task assignment as a time-based one, which is tackled through various classical mathematical techniques (e.g. combinatorics, optimizations etc.) or resource scheduling algorithms (e.g. genetic algorithms). The latter focus on extracting important text features, which are then fed into supervised ML algorithms in order to (i) estimate the probability for each given employee to work on a given set of tasks and (ii) assign employees to software bugs (Sajedi-Badashian & Stroulia, 2020; H. M. Tran et al., 2019). Nonetheless, the above ML-based approaches can work with any type of textual 'project management'-related data.

There is a number of advantages and disadvantages for both of the abovementioned categories. On the one hand, the task scheduling approaches can estimate the maximum number of tasks that can be concurrently processed and predict the minimum amount of time required to complete a project. However, they fail to match employees to tasks relevant to their skills, since they assume that all tasks or employees are similar to one another. On the other hand, ML-based approaches take into account the skill relevance in different tasks but require large amounts of textual data for predicting accurately the most suitable employee for each task.

In this work, we present a novel four-phase approach that assists organizations in the personnel selection process. Our approach is a ML-based one in that it utilizes textual data to assign people to tasks. The outcome of our approach is an ML model that is able to estimate how relevant or qualified a candidate employee is to work on a given task. Our ML model takes as an input a list of documents that describes completed tasks and the list of the assignees of the tasks. Our approach differs from the existing ones in that it does not require the calculation of Key Performance Indicators (KPIs) or any form of structured data in order to operate. Instead, it relies on techniques from Natural Language Processing (NLP), graph representation learning and word embeddings to reveal hidden knowledge that exists in unstructured textual data. It is domain agnostic since no additional information about the employees (e.g. their curricula or profiles) is needed to operate.

For the implementation of our approach, we use the Neo4j graph database, the

Python programming language and the TensorFlow deep learning library. To evaluate our approach, we benchmark it against a widely used classification model on a dataset retrieved from the official Jira instance of the Apache Software Foundation. This dataset concerns the development of 168 open source software projects. The experimental results demonstrate a significant improvement in accuracy of the classification models generated by following our approach. The related code, dataset and evaluation results are openly accessible on the GitHub [1] platform.

Generally, project management systems encompass large amounts of unstructured textual data. Our motivation is to build a novel approach that analyzes the aforementioned data and makes recommendations related to task assignment and personnel selection by exploiting information extracted from already completed tasks.

The main contribution of this chapter is four-fold:

- we provide the research community with an approach that improves existing ones by exploiting word embeddings, neural networks and graph representation learning techniques in an integrated way;
- we investigate whether the analysis of unstructured textual data benefits the personnel selection process or not;
- we propose a four-phase pipeline that assists project managers in the personnel selection process;
- we provide the research community with a rich dataset containing tasks of the projects of an organization that can be utilized as a baseline in similar research directions.

The remainder of this chapter is organized as follows. Background concepts and related work are introduced in Section 6.2. In Section 6.3, our approach is presented in a thorough and elaborated fashion, while the experiments carried out to evaluate our approach are reported in Section 6.4. The novelty, future work directions as well as the limitations of the proposed approach are discussed in Section 6.5.

6.2 Related work

6.2.1 Human Resource Allocation

Human resource allocation (i.e. assigning the most qualifying personnel to the most appropriate task) is admittedly a difficult and complex process, with bad selections resulting in major time and cost repercussions for a company. In the recent years, the concept of automating human resource allocation has gained interest, leading to a variety of approaches originating from diverse research fields.

A wide range of existing human resource management tools rely on concepts and methods from the area of recommendation systems. For instance, Alkhraisat (2016) proposes the utilization of document similarity techniques and ontology, and semantic similarity measures for the generation of an automated issue tracking system. The proposed approach discovers similar issues and recommends candidate

[1] https://github.com/imis-lab/personnel-selection

experts as well as related materials to address each issue. Heyn and Paschke (2013) describe an extension for the Jira platform, which works in a similar fashion as the abovementioned work, enabling the re-usability of similar work and the proper dissemination of work to the right developers. Another expertise recommendation engine can be found in (Truica & Barnoschi, 2019). This work utilizes text mining and NLP techniques to infer knowledge from resumes. In the next phase, the system produces a recommendation score for each candidate by semantically matching the retrieved information with job skills and requirements that are stored in a database.

The employment of appropriate Machine Learning algorithms has gained interest lately for manufacturing innovative personnel selection applications. For instance, the authors in (McDonald & Ackerman, 2000) propose a novel approach for evaluating job candidates by utilizing ML algorithms to extract a given job's requirements and the skills of the applicants by their LinkedIn profile. Having extracted these features, a ranking of the applicants is created based on the semantic similarity of the candidates' skills with the job prerequisites. In addition, the authors in (Azzini et al., 2018) describe a knowledge extraction process about academic candidates from a semi-structured interview. To this end, they employ various ML classifiers in order to discover the soft skills of individuals and match these skills with job requirements. Their empirical results demonstrated that Support Vector Machine (SVM) and naive Bayes (NB) approaches produce better results than k-Nearest Neighbors (k-NN), decision trees and random forests. Another application of ML classifiers can be found in (Wowczko, 2015), where job advertisements in Ireland in 2014 were processed to extract job titles and descriptions, which then served as input data. A categorization of jobs was eventually developed, affiliating each job categorization with a set of skills.

6.2.2 Task assignment

Helming et al. (2011) built an automatic assignment ML system for work items (e.g. tasks and bug reports). Their system models relationships between work items as functional requirements (e.g. work A needs to be implemented before work B), extracts significant textual features from work items using TF-IDF, while the combined relationships and features are used for training various supervised learning classifiers such as NB, SVM and neural networks. Their implementation relies on known ML-oriented java libraries and WEKA.

Jonsson et al. (2016) built an automated bug assignment system that follows a similar approach as the one proposed in (Helming et al., 2011). Their work differs in that they only extract textual features from successful bug reports, using TF-IDF as provided by the WEKA Java library. Then, they train multiple classifiers that assign bug reports to developer teams. After the classifiers are trained, they are merged into a generalized classifier, which takes into account predictions from all former classifiers. Overall, they report an accuracy ranging from 50% to 89% with higher accuracy scores correlated to more training data.

Mo et al. (2020) built an automated staff assignment ML system for building maintenance workers using NLP techniques. Particularly, they utilize a bag-of-words implementation to extract meaningful features from textual work descriptions and then predict which work group of employees is assigned for this work. The unigrams extracted from the bag-of-words implementation are fed into three supervised

learning classifiers, namely Linear Regression (LR), NB, and SVM. Overall, their experimental results indicate an accuracy of 77% and 88% for predicting correctly the assigned workforce and work priority, respectively.

Hassanien and Elragal (2021) utilize Word2Vec embeddings that are fed to a siamese long short-term memory neural network, which calculates the semantic similarity between texts of work descriptions found in enterprise resource planning systems. Although this work is not strictly related to the task assignment problem, it is the most similar to our approach since it utilizes word embeddings to capture the contextual information between important text features.

6.3 The proposed approach

In this section, we first define the preliminary concepts and the mathematical notation needed to describe our methodology. Then, we analytically present the proposed approach.

6.3.1 Preliminary concepts

Let $I = [i_1, i_2, \ldots, i_N]$ be a list of textual descriptions of issues or tasks of an organization, where $N \in \mathbb{N}$ (terms *issue* and *task* are used interchangeably throughout this chapter). Let $A = [a_1, a_2, \ldots, a_K]$ be a list of assignees of an organization, where $K \in \mathbb{N}$. Let $W = [w_1, w_2, \ldots, w_M]$ be the list of unique words from the textual descriptions I, where $M \in \mathbb{N}$. Let $E = [\vec{e_1}, \vec{e_2}, \ldots, \vec{e_M}]$ be a list of word embeddings, where $\vec{e_x} \in \mathbb{R}^D$ is a vector of D dimensions that corresponds to the vector representation of word w_x. Let $F = [f_1, f_2, \ldots, f_n] \subseteq W$ be a feature space, where f_x denotes that the word w_x has been selected as a feature and the number of features $n \leq M$.

Let $GD = (V, E)$ be a heterogeneous *graph of docs*, where V is the set of vertices and E is the set of edges of the graph. An edge $e_i = (v_x, v_y)$ links a vertex v_x to a vertex v_y. Each vertex v_i is associated with a label $l_i \in \{word, issue\}$ and each edge e_i is associated with a label $\hat{l}_i \in \{includes, co\_occurs\}$. An edge $e_i = (v_x, v_y)$ with a label $\hat{l}_i = co\_occurs$ can only connect two vertices when $l_x = l_y = word$. Similarly, an edge $e_i = (v_x, v_y)$ with a label $\hat{l}_i = includes$ can only connect two vertices when $l_x = issue$ and $l_y = word$. Let $WSG = (V', E')$ be a homogeneous word similarity graph, where $V' \subset V$ is the set of vertices with a label $l'_i = word$ for each vertex $v'_i \in V'$ and E' the set of edges of the graph with a label $\hat{l_i}' = is\_similar$ for each edge $e'_i \in E'$. An edge $e'_i = (v'_x, v'_y)$ links a vertex v'_x to a vertex v'_y and denotes a similarity greater than a predefined threshold between the words w_x, w_y as far as the *cosine similarity* of their embeddings $\vec{e_x}, \vec{e_y}$ is concerned. Let $ISG = (V'', E'')$ be a homogeneous issue similarity graph, where $V'' \subset V$ is the set of vertices with a label $l''_i = issue$ for each node $v''_i \in V''$ and E'' the set of edges of the graph with a label $\hat{l_i}'' = is\_similar$ for each edge $e''_i \in E''$. An edge $e''_i = (v''_x, v''_y)$ links a vertex v''_x to a vertex v''_y and denotes a *Jaccard index score* greater than a predefined threshold between the textual descriptions i_x, i_y.

6.3.2 Methodology

The aim of our work is to build an ML model that is able to estimate how relevant or qualified a candidate employee is to work on a given task i_x. Descriptions of past completed tasks and the names of assignees are required as input. Our approach consists of four main phases which are described in depth in the following sections and can be summarized as follows (see also Figure 6.1 for a more visualized summarization):

- **Representing assignees and issues as a graph-of-docs** (Section 6.3.2). Transforming a list of issues I and a list of assignees A into a graph-of-docs GD (see Chapter 3).
- **Calculating graph-based word embeddings** (Section 6.3.2). Training a Word2Vec model on the list of descriptions of the issues I in order to construct a similarity subgraph between words appearing as nodes in the GD. In the next step, we employ algorithms for representation learning on graphs (e.g. Node2Vec and GraphSAGE) to produce a list of word embeddings for each word using the similarity graph GD as input. The abovementioned process enables us to fine-tune the trained list of word embeddings E.
- **Feature selection** (Section 6.3.2). Defining a feature space F by identifying important terms in communities of textually similar issues, as proposed in Chapter 4.
- **Estimating the relevance of an employee** (Section 6.3.2). Training a neural network, where the input is the description of an issue i_x and the output is the probability of how capable an assignee a_x is in completing the given issue.

Representing issues as a graph-of-docs

The first phase consists of integrating the list of textual descriptions I and the list of assignees A into a single graph, according to the graph-of-docs representation (Giarelis et al., 2020b). The construction of a heterogeneous graph-of-docs representation allows nodes with different labels (i.e. '*issue*', '*word*' and '*author*' labels) to co-exist into a single graph. Similarly, the graph incorporates relationship edges with various labels (i.e. '*co_occurs*', '*includes*', '*is_similar*' labels). As mentioned above, the graph-of-docs representation facilitates the investigation of the global importance of a word, as a word can be linked to multiple documents. Finally, the existence of similarity edges between issues and words enables us to calculate well-established metrics, as far as their similarity is concerned. Before the construction of the GD, a training step is performed. Specifically, the descriptions are used to train a Word2Vec model in order to calculate a word embedding for each word that appears in the graph.

The constructed heterogeneous graph GD contains all the connections between the assignees, issues and the words of a corpus. Edges with an '*includes*' label are utilized to link each unique word of the corpus to every issue node which contains this particular word. Two word nodes are connected through a relationship edge with a

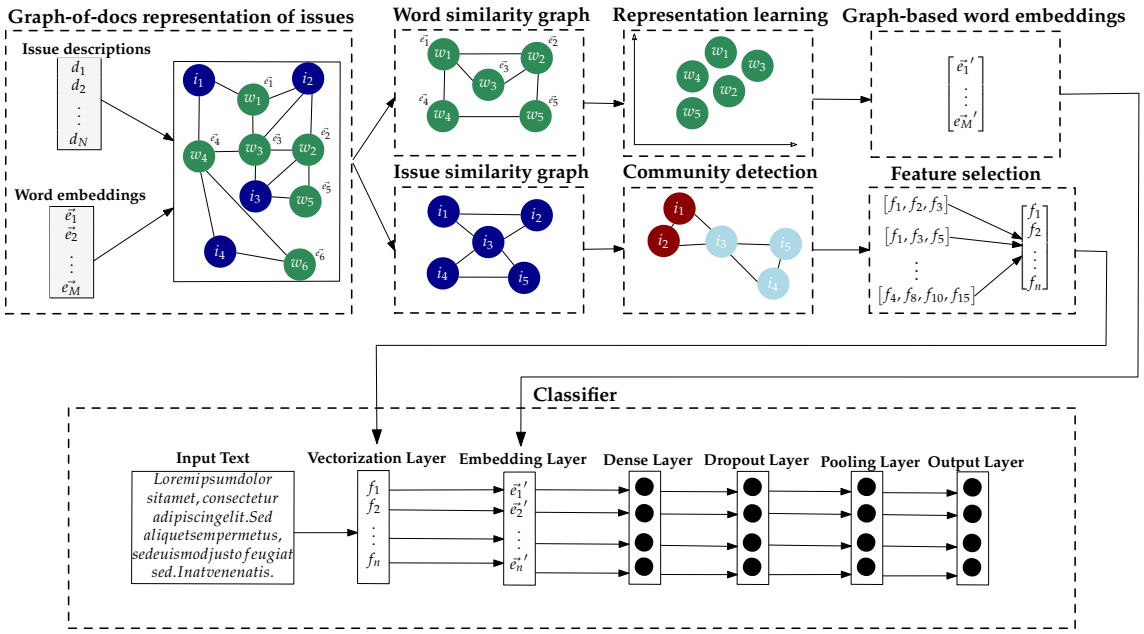


Figure 6.1: An overview of the proposed approach. i_x denotes a node that corresponds to an issue or task. w_x denotes a node that corresponds to a word from the descriptions of the issues or tasks. The word embedding of a word (w_x) is denoted by $\vec{e_x}$, while the enhanced version for the same word is denoted by $\vec{e_x}'$. f_x denotes a word that is selected as a feature for the final neural network classification model. The color of the nodes in the community detection step denotes different groups of textually similar issues.

label '*co_occurs*', which captures their co-existence within a sliding text window of fixed length. An assignee is connected with an issue node through a relationship edge with a label '*is_assigned_to*'. Last but not least, the word similarity subgraph WSG connects two highly similar word vertices based on their pairwise cosine similarity score, which is calculated by their trained Word2Vec embeddings.

The above representation of a corpus of issues as a heterogeneous graph enables us to reduce the human resource management problem to well-known and already studied graph tasks. By exploring important structural characteristics of a graph, such as node centrality and frequent subgraphs, these methods could identify important features, discover similar issues, generate issue clusters based on their similarity and even enrich existing word embeddings, through graph representation learning on the associated similarity subgraph.

Calculating graph-based word embeddings

This phase takes as an input the generated instance of the similarity subgraph WSG, which captures the similarity between words through the list of word embeddings E. The aim of this phase is to produce the enriched word embeddings list E' by fine-tuning the list E. To do so, we employ graph representation learning algorithms, which consider neighboring nodes as semantically similar. This enables us to utilize the existing word embeddings by translating their semantic similarity into a graph structural similarity. The graph representation algorithms take as input the word similarity subgraph WSG. Each edge of the graph also has a '*score*' $\in [0.0, 1.0]$ prop-

erty, which reflects the pairwise cosine similarity percent based on context provided by the initial list of word embeddings E. For instance, for two synonym words w_x, w_y the score property is ≈ 1, whereas for two antonym words w_x, w_y the score property is ≈ 0. Edges with a score value less than a predefined threshold are removed, aiming to avoid any information exchange between dissimilar nodes during the representation learning step. In this chapter, we employ *Node2Vec* and *GraphSAGE* as the main graph representation learning algorithms. Section 6.4.3 provides a detailed explanation of how we utilize the aforementioned algorithms. Finally, an in-depth review of graph representation learning algorithms can be found in (Hamilton et al., 2017b).

Feature selection

The task of the feature selection step of our pipeline is to produce a feature space F from the produced instance of graph-of-docs representation GD. This phase can be distinguished into four steps: (i) creating an issue similarity graph ISG, (ii) clustering (i.e. discovering communities) textually similar documents, (iii) performing feature selection for every one of the discovered clusters and (iv) producing an overall feature space F by combining the separate feature spaces created in the previous step (see also Algorithm 2).

The core idea behind the creation of an issue similarity graph ISG is that issues with textual similarities are expected to share a fair amount of common word nodes and structural attributes. Thus, by employing typical data mining similarity measures, we are able to recognize the underlying patterns of the graph and calculate the (textual) similarity between two issues i_x, i_y. Having calculated the similarity for each pair of issue nodes, the construction of the similarity graph ISG is a straightforward process. The issue similarity graph consists of issue nodes and edges with an '*is_similar*' label. The edges are also weighted with a '*score*' $\in [0, 1]$, which denotes how similar two nodes i_x, i_y are as far as their textual descriptions are concerned.

In the second step, the issue similarity graph facilitates the discovery of communities of textually similar issues. To this end, we define a distance value for every pair of issues, which is equal to the '*score*' property of the edge connecting the corresponding issue nodes. As far as clustering is concerned, a plethora of community detection algorithms can be found in the literature, such as Louvain (Blondel et al., 2008), Weakly Connected Components (Levorato & Petermann, 2011) and Label Propagation (Zhu & Ghahramani, 2002). In this chapter, we employ Louvain as the main community detection algorithm.

In the next step, we assume that issues that are part of the same community are also likely to share common features. To extract the most common features from every community, we prioritize the *top-N* terms of each cluster, firstly by their descending score of document frequency in the community, and secondly by their descending *PageRank* score. These *top-N* words serve as the feature space for every discovered community.

Finally, by merging the *top-N* terms of each cluster, we derive the global feature space F. The final feature space derived from the above process is bound to be significantly smaller than the original feature space. This dimensionality reduction (i) accelerates the feature extraction and selection process, (ii) restricts the repercussions of the '*curse-of-dimensionality*' phenomenon, and (iii) increases the overall

robustness and reliability of the ML models.

Algorithm 2: Feature selection

```
Input: an instance of the graph-of-docs representation GD = (V, E)
Result: a feature space F
// Step 1
IV ← get_all_issue_nodes(GD);
ISG ← initialize_graph();
for each node v in IV do
    for each node u in IV do
        similarity_score ← calculate_similarity(v, u, Jaccard);
        add_nodes_to_graph(ISG, v, u, similarity_score);
    end
end
// Step 2
communities ← [];
for each node v in ISG do
    add_node_to_a_community(communities, v, Louvain);
end
// Step 3
communities_topN ← [];
for each community c in communities do
    WV ← get_all_word_nodes_of_community(c);
    for each node v in WV do
        scores.add(calculate_centrality_score(v, PageRank));
    end
    topN ← find_topN_words(WV, scores, N);
    communities_topN.add(topN);
end
// Step 4
F ← [];
for each topN in communities_topN do
    F ← unique(F ∪ topN);
end
return F;
```

Estimating the relevance of an employee

This phase takes as an input the list of the selected features F, the produced graph-based word embeddings E' and the list of the textual descriptions of the issues I. It trains a neural network that is capable of predicting how suitable an employee is to undertake a specific issue. The proposed neural network is composed of 7 layers: (i) an *input* layer; (ii) a *vectorization* layer, where the input text is transformed into a list of features, retaining only the terms that are included in the feature space F; (iii) an *embedding* layer, where the terms are converted into their equivalent embedding representations inferred from the list E'; (iv) a *dense* layer; (v) a *dropout* layer, which mitigates overfitting; (vi) a *pooling* layer, where the embedding for the whole

text of an issue is calculated by aggregating the embedding representations of the features, using a predefined pooling function such as *max* or *average* pooling; (vii) an *output* layer, where the final relevance scores are calculated by an activation function such as *softmax*.

At this point of our pipeline process, the relevance classification problem has been reduced to a text classification one with multiple classes. The names of the candidate employees serve as the set of the model classes. A classification algorithm is utilized to accurately predict the class, which will be assigned to every issue (based on the probability scores produced by the classifier). Despite the fact that the algorithm assigns only one class to every issue, we can handle this limitation by examining the probabilities generated for the rest of the classes and thus inferring how relevant each candidate employee is with a given issue. In other words, the relevance of each employee with a task can be quantified as the probability score calculated by the ML classifier for the corresponding class.

6.3.3 Comparative assessment

Contrary to the work of Helming et al. (2011) and Jonsson et al. (2016) which use a statistical approach, namely TF-IDF, we propose a graph-based approach along with word embeddings to extract the most significant terms. The statistical approach of the aforementioned approaches measures the significance of terms across multiple documents; however, it fails to capture relationships between the candidate terms in the feature extraction phase. These relationships are captured by graph-based approaches that rely on co-occurrences such as the one proposed in this chapter. TF-IDF also fails to capture context between terms, as it does not produce word embeddings, which generally capture the similarity between terms.

Moreover, compared to the work described in (Mo et al., 2020), where simpler classifiers such as LR, SVM, and NB are used, our approach utilizes a neural network classifier. This results in an improvement in the overall performance, regardless of the selected text representation. Finally, we further expand the approach presented in (Hassanien & Elragal, 2021), which utilizes Word2Vec embeddings to calculate the similarity between two different tasks. In particular, we utilize the semantic similarity of our trained Word2Vec embeddings as a preparatory step, for training our classifier for the task assignment process. The utilization of the aforementioned state-of-the-art techniques enables the generation of ML models that are capable of deeply understanding unstructured textual data, which can be found in project management systems. We argue that the proposed approach is highly suitable for the problem of task assignment since it (i) captures complex relationships between the words of a document, (ii) identifies synonym terms and words, and (iii) utilizes neural networks, which are essential for building robust ML models for the analysis of multi-dimensional textual data.

6.4 Experimental evaluation

As far as the evaluation of our approach is concerned, the Python programming language and specifically, the TensorFlow deep learning library (Martın Abadi et al., 2015) were utilized to build our classification models. In addition, the initial word embeddings list represents each word as a one-hot vector. Storing the graph data,

as well as the procedure of enhancing the initial word embeddings was accomplished through the Neo4j graph database [2] using representation learning techniques. The full code of the experiments is freely available on the GitHub repository [3] of the chapter.

6.4.1 Dataset

The original dataset used for the evaluation of this chapter consists of *168* software projects, including *Spark*, *Hadoop* and *Airflow*. The information contained to this dataset concerns *228,969* Jira issues, with every issue incorporating the attributes *description* and *assignee* (see Table 6.1). The *assignee* attributes act as the class labels for the issues of the dataset. The dataset is available on Google Drive [4] and was fetched from the publicly accessible Jira instance of *Apache Software Foundation* [5].

Concerning the analysis of how each classification model reacts with respect to the number of samples as well as unique employees (classes), we created four subsets of the original dataset, where we conducted 10-fold cross-validation classification experiments. The details (number of classes, number of samples and evaluation results) for every subset are provided in Table 6.2.

Attribute	Description	Values	Attribute type
Assignee	The name of the assignee of the issue	e.g.{john_doe, jane_doe}	String
Description	The description of the issue	unstructured text	String

Table 6.1: The attributes of each sample of the dataset.

Dataset	Number of unique classes	Number of samples	Evaluation results
Dataset 1	3 (most frequent)	37,695	Table 6.3
Dataset 2	4 (most frequent)	39,259	Table 6.4
Dataset 3	21 (most frequent)	57,798	Table 6.5
Dataset 4	300 (all)	228,969	Table 6.6

Table 6.2: Descriptive statistics of each subset of the dataset.

6.4.2 Evaluation metrics

Two well-established metrics, namely *accuracy* and F_1 score are used to evaluate our approach. These metrics are defined as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$F_1 = \frac{2 \times precision \times recall}{precision + recall}$$

[2] https://neo4j.com/
[3] https://github.com/imis-lab/personnel-selection
[4] https://drive.google.com/file/d/19PnIT-hJtGk60ntdymo3vsw8o2Qs4Zsn/view?usp=sharing
[5] https://issues.apache.org/jira, retrieved at: 20 December 2019

where TP stands for true positives, TN for true negatives, FP for false positives, FN for false negatives predictions of each classification, and finally *precision* and *recall* are defined as follows:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Specifically, accuracy calculates the percentage of correct classifications, whereas F_1 expresses the harmonic mean of the *precision* and *recall*.

For a more elaborate evaluation of our model, we also calculate the average difference between the training and test loss of every neural network, where the sparse categorical cross-entropy was utilized as the loss function. The fact that this metric remains unaffected from the order of the data used to train the classifier is the main benefit of taking into consideration such a metric. In particular, sparse categorical cross-entropy captures how dissimilar the distribution of the real values of class labels is compared to the predicted values of the model, which in turn indicates whether the given classifier generalizes well or not.

Finally, to examine whether the improvement in the abovementioned metrics of our approach is statistically significant, we perform a micro sign test with a *p-value*$= 0.05$ for every classifier.

6.4.3 Classification models

In this section, we present the classification models used to evaluate the proposed approach. We consider the *bow_100x50* classifier as our baseline model, since it is the simplest and the most used model as far as the task of text classification is concerned.

- **bow_100x50**: A neural network model with 2 hidden dense layers with 100 and 50 units, respectively, and a dropout layer of 0.5. It also utilizes the bag-of-words model as a text representation. During the training phase, we use 15 epochs, a batch size of 128, the sparse categorical cross entropy as a loss function, the ReLU activation function for the 2 hidden layers, the softmax activation function for the output layer and the Adam optimizer.

- **graphsage_{100, 200, 300}**: A neural network model with a trainable embedding layer with 100, 200 or 300 dimensions, a hidden dense layer with 100 units, a dropout layer of 0.5 and a global max pooling layer. The initial weights of the embedding layer have been calculated by running the *GraphSAGE* algorithm on the corresponding word similarity graph *WSG* for 5 epochs. During the training phase of the neural network, we use 15 epochs, a batch size of 128, the sparse categorical cross entropy as a loss function, the ReLU activation function for the hidden layer, the softmax activation function for the output layer and the Adam optimizer.

- **node2vec_{100, 200, 300}**: A neural network model with a trainable embedding layer with 100, 200 or 300 dimensions, a hidden dense layer with 100 units, a dropout layer of 0.5 and a global max pooling layer. The initial

weights of the embedding layer have been calculated by running the *Node2Vec* algorithm on the corresponding word similarity graph *WSG* for 5 iterations. During the training phase of the neural network, we use 15 epochs, a batch size of 128, the sparse categorical cross entropy as a loss function, the ReLU activation function for the hidden layer, the softmax activation function for the output layer and the Adam optimizer.

6.4.4 Evaluation results

To evaluate our approach, we benchmark the classification models presented in Section 6.4.3 against the *bow_100x50* classifier on a list of datasets with different numbers of samples and classes. The obtained results (Table 6.3, Table 6.4, Table 6.5 and Table 6.6) indicate that classifiers, which are based on embeddings produced by the Node2Vec algorithm, are the ones that generally perform better (concerning accuracy and F_1 metrics), regardless of the properties of the given dataset. In addition, a statistically significant improvement in accuracy has been recorded for the majority of the proposed classifiers on *Dataset 4*.

It is also observed that the dimension of the embeddings does not affect significantly the performance of the classifier, since models relying on the same representation learning algorithm produce similar results. However, we notice that the dimension of the embeddings affects the generalization process of the classifier, since all models have statistically significant loss improvement against the *bow_100x500* classifier on each one of the four datasets (see Figure 6.2). Moreover, among all models, *node2vec_100* achieves the lowest absolute loss difference on each dataset. Finally, another aspect that affects the performance of a classifier is the size and the number of the unique classes of a dataset. This happens mostly due to the fact that the datasets are not balanced and, in many cases, there is a limited number of samples for a specific class, which in turn does not allow a classification model to generalize easily. This last observation is also noted in (Jonsson et al., 2016).

To provide an illustrative evaluation of our approach, we visualize the embeddings generated from (i) the pooling layer of the proposed approach, and (ii) the last hidden layer of the baseline model (see Figure 6.3). To do so, we first reduce the dimensions of the embeddings to 20 using the principal component analysis process; then, we use the t-SNE tool (Maaten & Hinton, 2008) to visualize the final embeddings in a two-dimensional space. For the sake of clarity, we only visualize the embeddings learned from *Dataset 2* (Table 6.4) through the *node2vec_100* (highest accuracy) and *bow_100x50* (baseline) models. As illustrated in Figure 6.3, the proposed approach produces embeddings that results in a better separation of the documents with respect to the associated class. In other words, documents with the same label are close to each other and the different groups of documents are better separated using the proposed approach. This explains why the majority of the proposed models achieves an improvement in accuracy and F_1 scores.

Considering the above observations, we conclude that the proposed approach is more likely to produce efficient classifiers with a better generalization process and better document or word embeddings.

Method	Accuracy	F_1	Train Loss	Validation Loss	Abs Loss Difference
bow_100x50	95.11(±0.52)	95.11(±0.52)	**0.0023**	0.4029	0.4005
graphsage_100	94.12(±0.46)	94.12(±0.46)	0.0037	0.2767	0.2730
graphsage_200	94.10(±0.48)	94.10(±0.48)	0.0040	0.2839	0.2799
graphsage_300	94.04(±0.46)	94.04(±0.46)	0.0048	0.3040	0.2991
node2vec_100	95.20(±0.32)	95.20(±0.32)	0.0218	**0.1252**	**0.1034**
node2vec_200	95.16(±0.27)	95.16(±0.27)	0.0186	0.1359	0.1173
node2vec_300	**95.28(±0.25)**	**95.28(±0.25)**	0.0166	0.1426	0.1260

Table 6.3: Classification *accuracy* and F_1 score (± standard deviation) of each model on the subset of the dataset concerning the 3 most frequent assignees (Dataset 1). All the models have statistically significant loss (last column) improvement against the *bow_100x50* baseline. Bold font indicates the best score value for each column.

Method	Accuracy	F_1	Train Loss	Validation Loss	Abs Loss Difference
bow_100x50	94.42(±0.38)	94.42(±0.38)	**0.0039**	0.4487	0.4448
graphsage_100	93.50(±0.44)	93.50(±0.44)	0.0048	0.2788	0.2740
graphsage_200	93.33(±0.36)	93.33(±0.36)	0.0058	0.3028	0.2970
graphsage_300	93.15(±0.37)	93.15(±0.37)	0.0068	0.3233	0.3165
node2vec_100	**94.49(±0.32)**	**94.49(±0.32)**	0.0297	**0.1446**	**0.1149**
node2vec_200	94.42(±0.45)	94.42(±0.45)	0.0243	0.1542	0.1299
node2vec_300	94.43(±0.38)	94.43(±0.38)	0.0223	0.1624	0.1402

Table 6.4: Classification *accuracy* and F_1 score (± standard deviation) of each model on the subset of the dataset concerning the 4 most frequent assignees (Dataset 2). All the models have statistically significant loss (last column) improvement against the *bow_100x50* baseline. Bold font indicates the best score value for each column.

Method	Accuracy	F_1	Train Loss	Validation Loss	Abs Loss Difference
bow_100x50	**87.11(±0.28)**	**87.11(±0.28)**	0.0638	0.7501	0.6863
graphsage_100	85.70(±0.48)	85.70(±0.48)	0.0595	0.4885	0.4289
graphsage_200	85.59(±0.53)	85.59(±0.53)	**0.0537**	0.5165	0.4628
graphsage_300	85.27(±0.62)	85.27(±0.62)	0.0565	0.5421	0.4856
node2vec_100	86.95(±0.61)	86.95(±0.61)	0.1607	**0.3858**	**0.2251**
node2vec_200	86.96(±0.54)	86.96(±0.54)	0.1354	0.3874	0.2520
node2vec_300	86.78(±0.30)	86.78(±0.30)	0.1281	0.4000	0.2719

Table 6.5: Classification *accuracy* and F_1 score (± standard deviation) of each model on the subset of the dataset concerning the 21 most frequent assignees (Dataset 3). All the models have statistically significant loss (last column) improvement against the *bow_100x50* baseline. Bold font indicates the best score value for each column.

Method	Accuracy	F_1	Train Loss	Validation Loss	Abs Loss Difference
bow_100x50	50.59(±0.38)	50.59(±0.38)	1.4043	2.5318	1.1275
graphsage_100	51.00(±0.53)	51.00(±0.53)	1.3063	2.0776	0.7714
graphsage_200	50.91(±0.30)∗	50.91(±0.30)	**1.2925**	2.0836	0.7912
graphsage_300	50.91(±0.36)∗	50.91(±0.36)	1.3114	2.0940	0.7826
node2vec_100	**51.64(±0.27)**∗	**51.64(±0.27)**	1.5557	**2.0287**	**0.4729**
node2vec_200	51.32(±0.29)∗	51.32(±0.29)	1.5184	2.0350	0.5166
node2vec_300	51.25(±0.60)∗	51.25(±0.60)	1.5131	2.0429	0.5298

Table 6.6: Classification *accuracy* and F_1 score (± standard deviation) of each model on the dataset concerning all the available (300) assignees (Dataset 4). * indicates statistical significance in accuracy improvement against the *bow_100x50* baseline. All the models have statistically significant loss (last column) improvement against the *bow_100x50* baseline. Bold font indicates the best score value for each column.

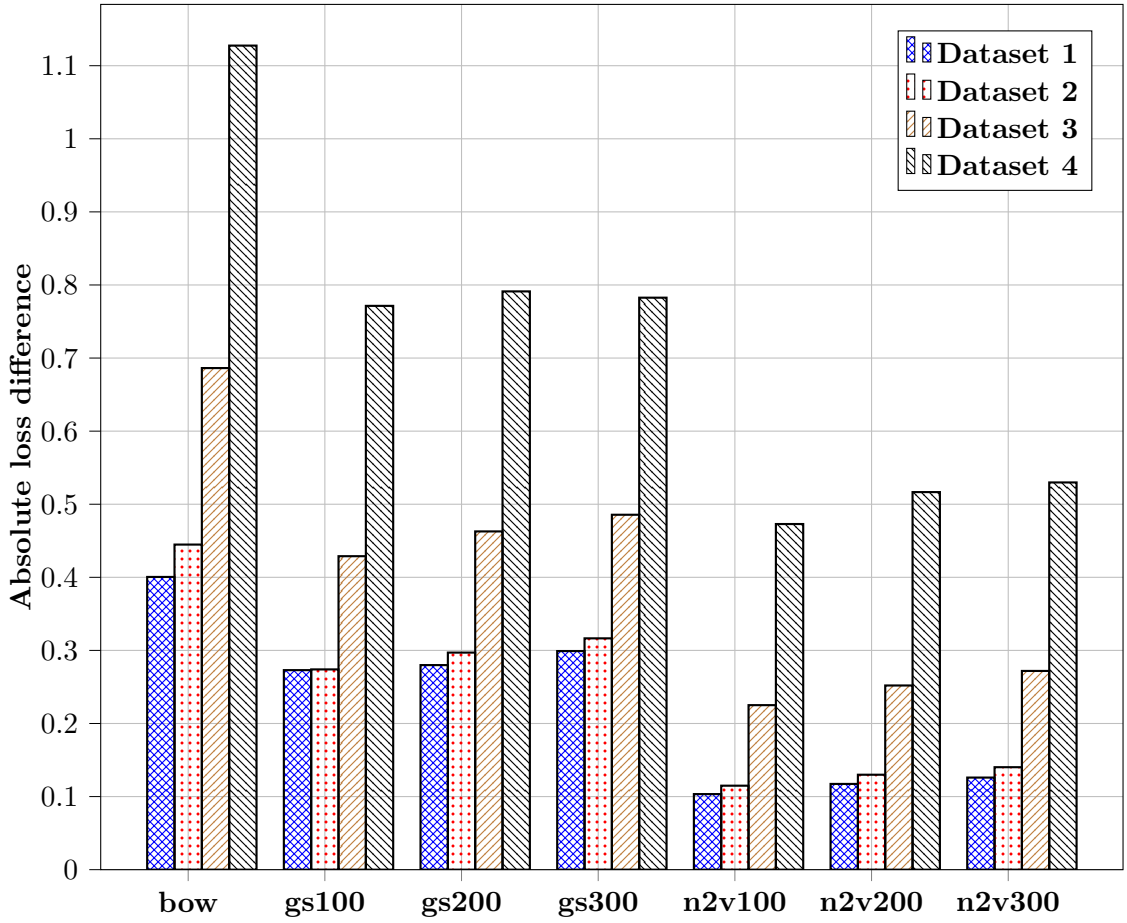


Figure 6.2: Absolute loss difference of the classification models on each subset of the original dataset concerning the 3 (**Dataset 1**), 4 (**Dataset 2**), 21 (**Dataset 3**) or 300 (**Dataset 4**) most frequent employees.

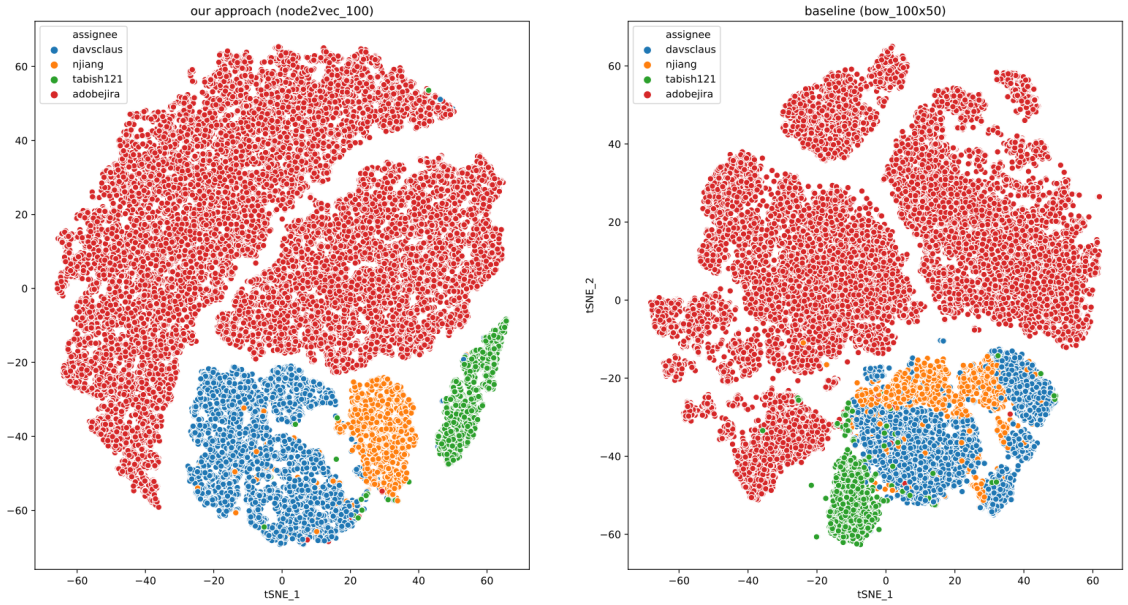


Figure 6.3: The t-SNE visualization of the embeddings learned by the proposed approach (**left**) and the baseline model (**right**) on *Dataset 4*.

6.5 Discussion and conclusions

In this chapter, we propose a novel approach that exploits techniques from various research fields, such as natural language processing, graph representation learning and word embeddings, to assist project managers in making smarter and evidence-based decisions in the personnel selection process. By properly assigning the most qualified personnel to the appropriate tasks, the chance of success of a project is expected to significantly increase. To this end, our approach estimates a relevance score between a given task X and an employee Y, by estimating the probability that the employee Y possess the skills required by task X. The neural networks developed for the calculation of this probability reveal hidden knowledge residing in unstructured textual data, which otherwise would remain untapped. In addition, our approach is domain agnostic, as it does not require additional information about the tasks (e.g. keywords that reflect the necessary skills and competences to perform a given task) or the employees (e.g. one's curriculum or social media profile).

For our experiments, we retrieved data from the Jira issue tracking system of the Apache Foundation, which we enriched with information from word embeddings. As demonstrated by the evaluation results, the proposed approach contributes to the increment of the classification accuracy (compared to a widely-adopted baseline method), which in turn denotes that it is able to calculate the relevance score precisely. It is also noted that the performance of our approach is not affected by the majority of the characteristics of the dataset. On the contrary, it is only affected by the size and the number of the unique classes of the given dataset, something that occurs as a result of the imbalance of the dataset and the limited number of samples for specific document classes.

The examples given in Section 6.4.4 demonstrate that the proposed approach produces classification models for predicting the best fit for each employee. From a practical perspective, our approach enables project managers to assign employees to tasks without any human bias and alleviates the need for calculating KPIs or other time-consuming metrics.

The main contributions of our chapter are: (i) we implemented and evaluated an approach that meaningfully integrates concepts and techniques from the fields of word embeddings, neural networks and graph mining; (ii) we investigated whether the analysis of textual data is able to assist in the personnel selection process; (iii) we proposed a novel ML-based pipeline that assists project managers in the personnel selection process; (iv) we provided the research community with a rich dataset containing tasks of the projects of an organization that can be utilized as a baseline in similar research directions.

A list of remarks has been collected during the analysis of the evaluation results, which can be summarized as follows: (i) the utilized technologies (i.e. word embeddings, neural networks and graph representation learning) can be used for efficiently analyzing textual data related to personnel selection; (ii) the combination of the aforementioned technologies advances the document classification process and assists in producing well-generalized models; (iii) the utilization of graph-based text representations is essential for capturing relationships between the words of a document; (iv) graph representation learning techniques produce better word embeddings as far as the task of identifying synonym words is concerned; (v) neural networks is of great value when dealing with the analysis of high-dimensional textual data.

Aiming to further advance the performance of our approach, future work will focus on improving its classification phase through graph mining techniques such as node classification with graph neural networks and graph attention networks (Veličković et al., 2018). We also plan to further enrich the initial graph-of-docs representation by introducing new edge types among '*issue*' nodes (e.g. '*dependencies*', '*priorities*' etc.). Another future work direction is to expand the proposed pipeline with an additional phase that utilizes techniques from the field of operations research; this will allow the inclusion of constraints and optimization rules, enabling an organization to optimize the allocation of tasks to the available employees (Gaspars-Wieloch, 2021; Kanakaris et al., 2020a).

Chapter 7

Combining word embeddings, graph-based text representations and graph attention networks

This chapter proposes a combination of word embeddings, graph-based text representations and graph attention networks. The particular application concerns the identification of software bugs. Existing approaches aim to advance each of the above components individually, without considering an integrative approach. As a result, they ignore (i) information about the structure of a given text, or (ii) information related to an individual word of the text. Instead, our approach seamlessly incorporates both semantic and structural characteristics into a graph, which are then fed to a graph attention network in order to classify GitHub issues as bugs or features. Our experimental results demonstrate a significant improvement in terms of accuracy, precision and recall of the proposed approach compared to a list of classical and graph-based machine learning models. The dataset for the experiments reported in this paper has been retrieved from the kaggle.com platform and concerns GitHub issues with short-text attributes.

7.1 Introduction

The demand for software engineers has been on the rise during the last years, and is projected to grow much faster than the average for all other occupations. One of the most time-consuming tasks software engineers have to deal with is the identification and correction of software bugs. While the utilization of experts' knowledge and experience plays a crucial role in the minimization of bugs, the idea of bug-free software development is unreachable. In this context, various platforms have been developed where the engineers can disseminate, expand and discuss their work in a peer-to-peer style, with GitHub being one of the most popular ones. Through publicizing their software, users can deliberate about the best ways to dissolve their bugs, find faults they may have missed and come up with features they could incorporate into their work.

While such platforms offer users an easy way to post and elaborate software-related issues, the information overload can be heavy, as the topic of an issue is not always clear. Very often, users need to go through a plethora of posts to distinguish if an issue relates to a bug or a feature of the corresponding software. Many

research works (Bharadwaj & Kadam, 2022; Cabot et al., 2015; Kallis et al., 2021) have shifted their attention towards reducing the time needed for such processes by utilizing Natural Language Processing (NLP) techniques, word embeddings and Machine Learning (ML) models to classify GitHub issues. Such approaches exploit the semantic information that word embeddings provide to create sophisticated ML classifiers that automate issue classification tasks. However, word embeddings are limited to capturing polysemy (i.e. the same word may have different meanings in different contexts) and word inversion. In this context, researchers have begun to experiment with graph-based text representations and Graph Convolutional Networks (*GCNs*), aiming to incorporate contextual information into their data representations (Ragesh et al., 2021; Yao et al., 2019; Ye et al., 2020; Zhang & Zhang, 2020). Nevertheless, there is no existing approach that deals with an integrated configuration of word embeddings, text representations and graph-based models.

In this paper, we propose a novel approach that leverages the short-text body attributes of a list of GitHub issues to create a software bug identification model. To this end, we create a graph-based text representation for each issue and utilize pre-trained GloVe (Pennington et al., 2014) embeddings as node features towards combining contextual and semantic information. Then, a Graph Attention Network (*GAT*) is developed for graph-level prediction. *GAT* differs from *GCN* models in that it captures local neighbor node features rather than the global structure of the graph. *GAT* propagates information through neighboring nodes to create node embeddings, which are then gathered to a global mean pooling layer to create a graph representation. In this way, we reduce the software bug identification problem to a graph classification one.

7.1.1 Research objectives and contribution

Overall, the contribution of this paper is as follows:

- we meaningfully introduce techniques from the area of GNNs and graph-based text representations to the task of identifying software bugs;
- we provide the research community with a new approach that improves the quality of software bug predictions;
- we investigate whether the representation of the text of a GitHub issue as a graph-of-words graph benefits the software bug identification process;
- we propose a novel approach that combines properly word embeddings, graph-based text representations and graph.

The remainder of this paper is organized as follows: in Section 7.3 we present the proposed GAT-based approach, while the experiments conducted and the associated results are reported in Section 7.4; finally, concluding remarks and future work directions are discussed in Section 7.5.

7.2 Identification of software bugs

There has been a wide variety of approaches derived from different research directions, as far as the identification and prediction of software bugs are concerned. For

example, in (Sahu & Srivastava, 2021) the authors utilize a neuro-fuzzy framework to predict reliable software. Specifically, they employ a neural network with a sigmoid activation function and fuzzy logic to compensate for imprecise or incomplete data. In a similar manner, the authors in (Xia et al., 2015) deal with reopened bugs. They identify three different types of data in their datasets (i.e. descriptive, commentary and metadata), which can then be combined to extract features. For each set of features, a classifier is developed and the output of each of them is taken into consideration for the final prediction with an ensemble model. In general, there have been many research approaches that utilize ML models for the task of identifying software bugs; we refer to (Khleel & Nehéz, 2021) for a more comprehensive demonstration. To the best of our knowledge, there is no approach so far that combines word embeddings, graph-based text representations and GNNs for the identification of software bugs.

7.3 The proposed approach

In this section, we first introduce some preliminary concepts. Then, we provide a detailed description of the components of the proposed approach. Our approach consists of two main components: (i) construction of graphs, and (ii) graph classification (see Figure 7.1).

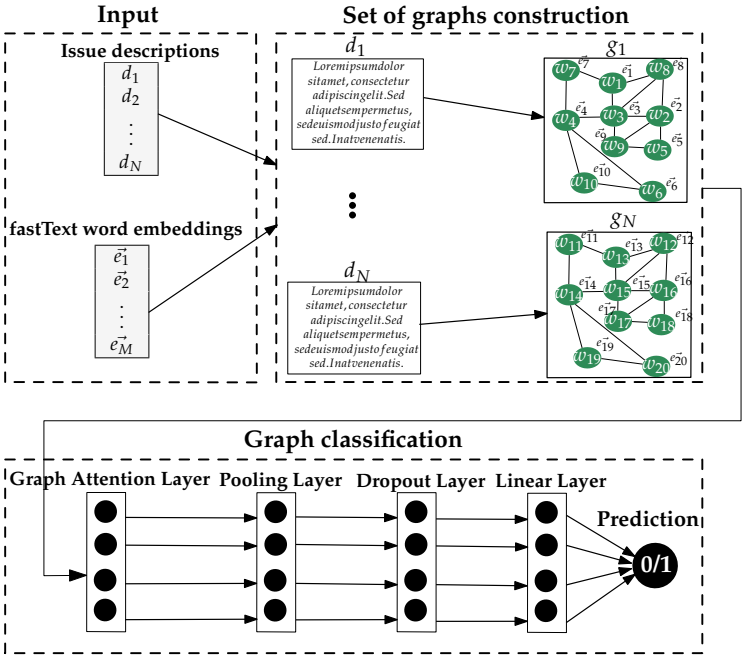


Figure 7.1: The architecture of the proposed approach.

7.3.1 Construction of graphs

The approach presented in this paper seeks to exploit the structural features that a graph provides alongside the semantically important information that derives from word embeddings. The first component of the proposed pipeline is the construction of a set of graphs, where each graph represents a document. To this end, we utilize

the *graph-of-words* text representation (Rousseau & Vazirgiannis, 2013), where every word is mapped to a vertex of the graph. Each graph is homogeneous, weighted and directed. An edge links two word nodes and its weight denotes the concurrent coexistence of these words in a sliding window of text. As this window 'slides' through the text, it essentially creates word-level n-grams (e.g. from the sentence *'currently the test suites for xl and fs backend exists'* the following 3-grams are produced: <currently the test>, <the test suites>, <test suites for> etc.), which are then used to calculate the co-occurrence weight for each pair of words. Research has shown that the length of the sliding window should be proportional to the average number of words of each sample of the given dataset (Rousseau et al., 2015). Before the construction of the graphs, each text is preprocessed in a standard way, including tokenization, punctuation and stopword removal (Blanco & Lioma, 2012; Rousseau et al., 2015).

By constructing a set of graphs (one for every document in the corpus), we capture the structural details of our corpus. However, the contextual information that is hidden in the words still remains untapped. To incorporate these features into our graph, we employ pre-trained *FastText* embeddings for every word in our corpus. *FastText* is preferred over *Word2Vec* and *GloVe* as it takes into account the internal structure of words by dissecting a word in character-level n-grams, in a similar fashion as *graph-of-docs*. This is useful for words that occur rarely. The node features are initialized with these word embeddings, denoted as $\mathbf{H} \in R^{|\mathcal{V}| \times d}$ where d is the embedding dimension. By leveraging word embeddings as node features, the graph-based representation of the corpus manages to merge both structural and semantic information lying underneath a document in a set of graphs. Figure 7.2 illustrates an example of a graph-of-words graph.

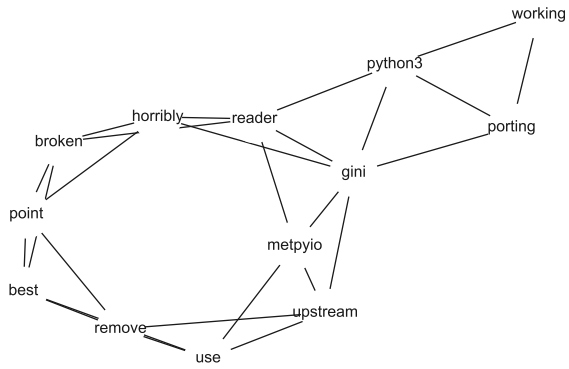


Figure 7.2: An example of a graph-of-words graph of the text of a GitHub issue using a sliding window size of 3.

7.3.2 Graph classification

As far as the graph classification is concerned, a *GAT* network is utilized. *GAT* depends on the importance of self-attention in a graph, where each node relies on its neighbor to format a hidden state, which in turn emphasizes on the statistical correlations of the words in a document. The hidden state for every node is calculated

by:

$$\mathbf{h}_i^{t+1} = \rho\left(\sum_{v_j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\mathbf{h}_j^t\right) \tag{7.1}$$

where ρ is an activation function, $\mathbf{W}$ is the weight matrix of the hidden state, $\mathbf{h}_j^t$ is the feature vector of node j produced in the previous layer. Moreover, each neighbor v_j of node v_i is assigned to an attention parameter α_{ij}, which indicates the importance of node j for node i. These attention parameters are obtained by utilizing the features of the previous layer with a Feed Forward Neural Network, using a softmax activation function as:

$$\alpha_{ij} = \frac{exp\left(LeakyRELU\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j\right]\right)\right)}{\sum_{k \in N_i} exp\left(LeakyRELU\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k\right]\right)\right)} \tag{7.2}$$

where ^{T} represents transposition and $\|$ is the concatenation operator. Note that the weight matrix is exactly the same trained in the GAT layer.

Self-attention provides a means to introduce contextual information in our model. However, research has shown that it may be unstable, especially in cases of large text (Vaswani et al., 2017). To remedy this shortcoming, we utilize multi-head attention by running K parallel GAT layers, which are then concatenated for each node in a single hidden state:

$$\mathbf{h}_i^{t+1} = \Big\Vert_{k=1}^{K} \rho\left(\sum_{v_j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \mathbf{h}_j^t\right) \tag{7.3}$$

In the case of the last GAT layer, the output for every node is the average of the multi-head outputs, such as:

$$\mathbf{h}_i^{t+1} = \frac{1}{K}\sum_{k=1}^{K}\sum_{v_j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \mathbf{h}_j^t \tag{7.4}$$

For the output layer of the neural network, the node features are concatenated in a feature matrix $H^o \in \mathbb{R}^{|\mathcal{V}| \times f}$, where f denotes the dimension of the output features and a global mean pooling layer is used so that we can perform graph-level classification with a sigmoid activation function.

7.4 Experiments

7.4.1 Dataset

For the experiments of this paper, we retrieved data from the kaggle.com platform. The original dataset contains 450,000 software issues extracted from a list of code repositories from the GitHub platform. Each sample contains three attributes, namely title, body and label; for simplicity, we use only the body attribute to classify the software issues. The samples of the dataset are divided into 3 classes (i.e., *bug*, *feature* and *question*). However, we opted to discard the *question* samples in

order to reduce the problem of the identification of software bugs to a binary graph classification one. Descriptive statistics about the dataset can be found in Table 7.1.

Descriptive statistics for the textual attributes of each sample are summarized in Table 7.2. As illustrated in Figure 7.3, the vast majority of the samples of the dataset has a small number of words that ranges from 3 to 6 and from 5 to 15 with regards to the title and body attribute, respectively. In addition, 25% of the samples have a number of words of the body attribute that is less than 12. The above observations indicate that the samples contain short-texted attributes, something that requires the selection of a small sliding window size between 5 and 15 (Rousseau et al., 2015).

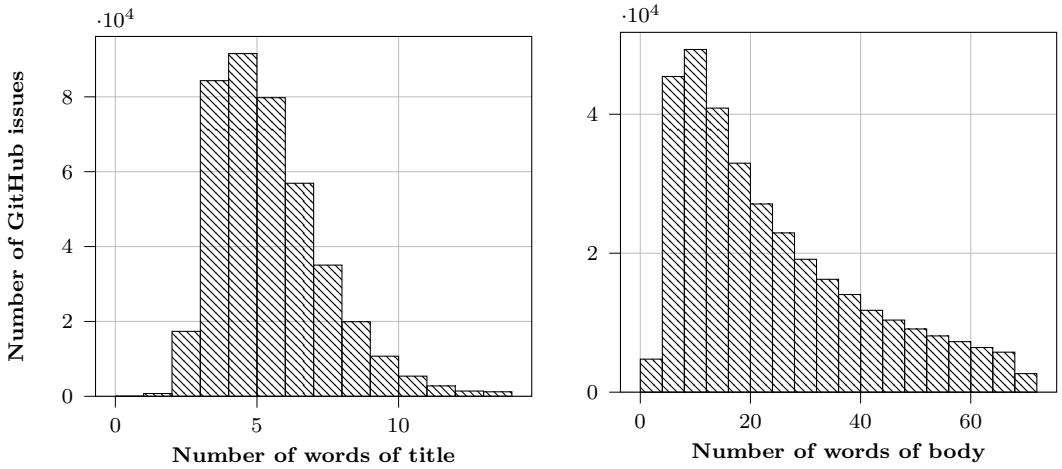


Figure 7.3: Distribution of the GitHub issues in terms of (i) the number of words of the title (left), (ii) the number of words of the body (right). For clarity, we included only the GitHub issues whose title has less than 15 words and their body has less than 70 words.

# documents	407799
# training samples	273225
# test samples	134574
# bugs	207318
# features	200481

Table 7.1: Statistics of the dataset

# words / **Feature**	**Min**	**Max**	**Average**	**STD**	**Median**	**25%**	**75%**
title	0	43	4.97	2	5	3	6
body	0	977	45.94	63.55	24	12	52

Table 7.2: Statistics of the textual features of the dataset

7.4.2 Experimental setup

To implement and evaluate our approach, we use the Python programming language, the PyTorch, PyTorch Geometric and scikit-learn libraries. To generate the word

and sentence embeddings for the baseline models, we utilize the GloVe embeddings, whereas for the proposed method we utilize the fastText embeddings. We note that in both cases we use as an input the body of each GitHub issue. The produced embeddings have a vector size of 100. For the production of the sentence embeddings, we average the embeddings of their words. To create the graph-of-words graph for each sample, we use a window size of 7 and a min count of 2 (i.e. we remove the words appearing less than 2 times in the dataset). The full code and evaluation results of our experiments are available on GitHub (https://github.com/nkanak/bug-or-feature).

The proposed model, namely *FastGATConv*, consists of a hidden layer that utilizes the graph attentional operator (Veličković et al., 2018). The attention layer has 10 output units and 3 heads. Then, we employ a global mean pooling layer and a dropout layer with a dropout of 0.5. We train the model for 10 epochs with a batch size of 300, the binary cross entropy as a loss function, the ELU activation function for the attention layer, and the Adam optimizer with a learning rate of 0.0001. We use as an input a vector of pre-trained fastText embeddings with a size of 100.

The performance of our approach is assessed by benchmarking it against the following list of classification models:

- **GloVe + LR**: A logistic regression model with a 'liblinear' solver.
- **GloVe + k-NN**: A k-nearest neighbors model with a k of 10.
- **fastText**: A model that is provided by the official fastText library (Joulin et al., 2016). The model utilizes a list of word embeddings to infer the embeddings both for words and sentences (averaging the vectors of the words of the given sentence). A linear classifier is employed for the final classification.
- **GloVe + MLP**: A neural network model with a hidden dense layer with 250 units and a dropout layer of 0.5. We train the model for 50 epochs with a batch size of 255, the binary cross entropy as a loss function, the ReLU activation function for the hidden layer, and the Adam optimizer with a learning rate of 0.0001.
- **GloVe + GATConv**: A graph convolutional network with a hidden layer that utilizes the graph attentional operator (Veličković et al., 2018). The attention layer has 10 output units and 3 heads. Furthermore, we employ a global mean pooling layer and a dropout layer with a dropout of 0.5. We train the model for 10 epochs with a batch size of 300, the binary cross entropy as a loss function, the ELU activation function for the attention layer, and the Adam optimizer with a learning rate of 0.0001. This model differs from FastGATConv in that it uses as an input the pre-trained GloVe embeddings (Pennington et al., 2014).
- **GloVe + GCNConv**: A graph neural network with a convolutional hidden layer (Kipf & Welling, 2017) with 10 output units, a global mean pooling layer and a dropout layer with a dropout of 0.5. We train the model for 10 epochs with a batch size of 300, the binary cross entropy as a loss function, the ELU activation function for the convolutional layer, and the Adam optimizer with a learning rate of 0.001. We use as an input the pre-trained GloVe embeddings (Pennington et al., 2014) with a size of 100.

- **GloVe + GraphConv**: A graph neural network with a convolutional hidden layer with 10 output units, a global mean pooling layer and a dropout layer with a dropout of 0.5. We train the model for 10 epochs with a batch size of 300, the binary cross entropy as a loss function, the ELU activation function for the convolutional layer, and the Adam optimizer with a learning rate of 0.001. We use as an input the pre-trained GloVe embeddings (Pennington et al., 2014) with a size of 100. This model differs from the GloVe + GCNConv model in that the hidden layer performs a mean aggregation of its output and utilizes the Weisfeiler-Leman Algorithm, as proposed in (Morris et al., 2018).
- **GloVe + SAGEConv**: A graph convolutional network with a hidden layer that utilizes the graph GraphSAGE operator (Hamilton et al., 2017a). The convolutional layer has 10 output units. We employ a global mean pooling layer and a dropout layer with a dropout of 0.5. We train the model for 10 epochs with a batch size of 300, the binary cross entropy as a loss function, the ELU activation function for the attention layer, and the Adam optimizer with a learning rate of 0.0001. We use as an input the pre-trained GloVe embeddings (Pennington et al., 2014) with a size of 100.

For the evaluation of our approach we use the accuracy, precision and recall metrics. Finally, we perform a micro sign test with a p_{value} of 0.05 for every baseline method, aiming to identify whether an improvement in the accuracy of our approach is statistically significant.

7.4.3 Evaluation results

Table 7.3 presents the mean accuracy, precision and recall scores (± standard deviation) of the proposed method as well as that of the baseline models. The mean scores are calculated by running each experiment 10 times. According to the micro sign statistical test, our approach performs significantly better in comparison to all of the baseline models as far as the accuracy score is concerned. In addition, the high precision and recall scores of FastGATConv denote that our approach is able to identify software bugs with a great success, without mistakenly classifying many software bugs as features (or the opposite).

By analyzing the evaluation results, we come up with the following observations:

- in most cases, the graph-based models perform better than the linear and classical counterparts;
- fastText model outperforms all the graph-based models except than the proposed one;
- GloVe + GATConv demonstrates better results compared to the rest of the GNN baselines;
- GloVe + MLP performs similar (or even better) to most of the GNN models on datasets with short texts;
- poor word embeddings inhibit the performance of the GNN models;
- the utilization of either GNN models or graph-based text representations does not always result in better performance on datasets with short texts.

The above observations indicate that the mindful selection and combination of (i) pre-trained word embeddings, (ii) textual representations, and (iii) classification models are of much importance for the identification of software bugs. Our experimentations demonstrate that the success of FastGATConv on short text datasets relies on (i) the fact that the fastText model produces better embeddings than GloVe, (ii) the proper configuration and selection of a small sliding window size for the graph-of-words representation, (iii) the employment of a convolutional network (e.g. an attention layer) that emphasizes on local-structure propagation rules instead of global-based ones.

Method \ Metric	Accuracy	Precision	Recall
GloVe + LR	0.7227 ± 0.0000	0.7408 ± 0.0000	0.7014 ± 0.0000
GloVe + k-NN	0.6918 ± 0.0000	0.7341 ± 0.0000	0.6199 ± 0.0000
fastText	0.7968 ± 0.0002	0.8107 ± 0.0009	0.7845 ± 0.0017
GloVe + MLP	0.7401 ± 0.0002	0.7544 ± 0.0006	0.7268 ± 0.0013
GloVe + GATConv	0.7651 ± 0.0006	0.7776 ± 0.0022	0.7537 ± 0.0034
GloVe + GCNConv	0.7295 ± 0.0001	0.7475 ± 0.0023	0.7068 ± 0.0043
GloVe + GraphConv	0.7493 ± 0.0006	0.7642 ± 0.0025	0.7331 ± 0.0049
GloVe + SAGEConv	0.7493 ± 0.0006	0.7642 ± 0.0025	0.7331 ± 0.0049
FastGATConv[a]	0.8022 ± 0.0002	0.8125 ± 0.0024	0.7943 ± 0.0036

[a]proposed method

Table 7.3: Evaluation results

7.4.4 Parameter sensitivity

FastGATConv has two main parameters, i.e. the size of the sliding window and the learning rate. Figure 7.4 exhibits how the sliding window size affects the performance of FastGATConv. It is shown that the accuracy of the model improves as the size of the sliding window increases. We note that a sliding window size greater than 10 does not improve the accuracy significantly. Instead, it increases the computation time radically. The above behavior is caused by the fact that the given dataset contains short texts. As a result, a large sliding window size adds redundant and noisy edges between the nodes of a graph-of-words graph, leading to poor classification performance.

In Figure 7.5, we examine how the learning rate of the GAT network influences the performance of FastGATConv. As expected, an increase in the learning rate eventually results in a decrease in accuracy. In any case, a learning rate that ranges from 0.0001 to 0.1 can be considered as a good choice.

7.4.5 Impact of the dataset size on classification performance

To assess how the size of the dataset affects the performance of our approach, we conduct several experiments using different proportions of the initial dataset. Figure 7.6 reports the test accuracy using 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%,

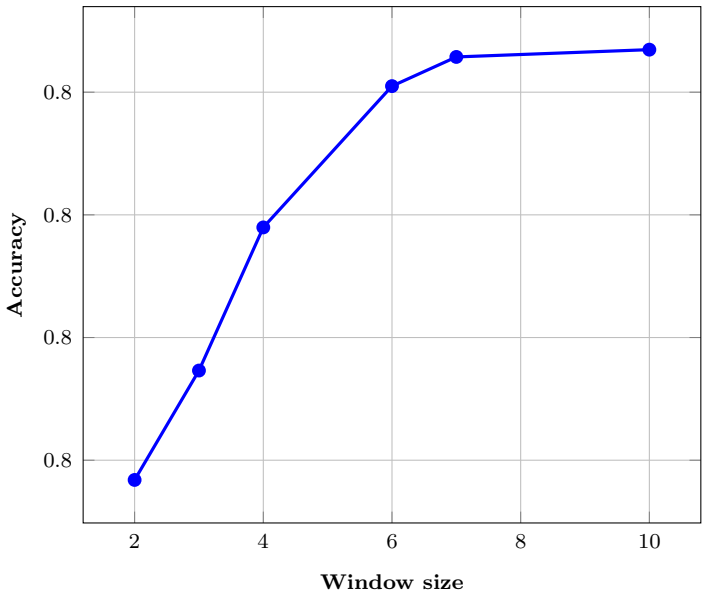


Figure 7.4: Accuracy score of the proposed approach with respect to a different sliding window size ranging from 2 (bigrams) to 10.

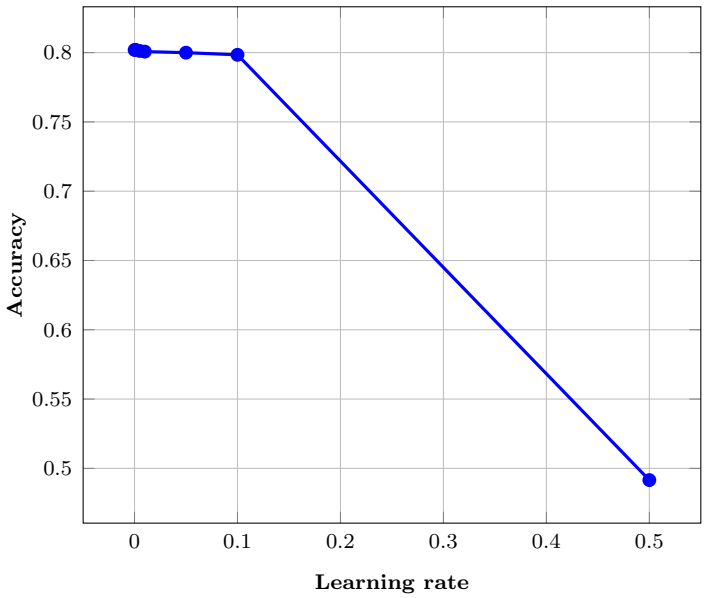


Figure 7.5: Accuracy score of the proposed approach with respect to a different learning rate ranging from 0.0001 to 0.5

90% and 100% of the original dataset. It is revealed that FastGATConv performs better with smaller proportions of the dataset. For instance, it achieves an accuracy score of 0.8510 and 0.8024 on the 10% and 100% dataset, respectively. The reason is that the underlying GNN model manages to learn better a smaller dataset in the limited number of 10 epochs. This indicates that FastGATConv may benefit from the increment of the number of epochs. Another reason that contributes to this improvement is the removal of outliers existing in the initial dataset.

To sum up, we conclude that a small number of epochs and the existence of outlier samples can possibly affect negatively the performance of FastGATConv. Thus, the proper configuration of the number of epochs along with the removal of candidate outliers may drastically improve the performance of our approach.

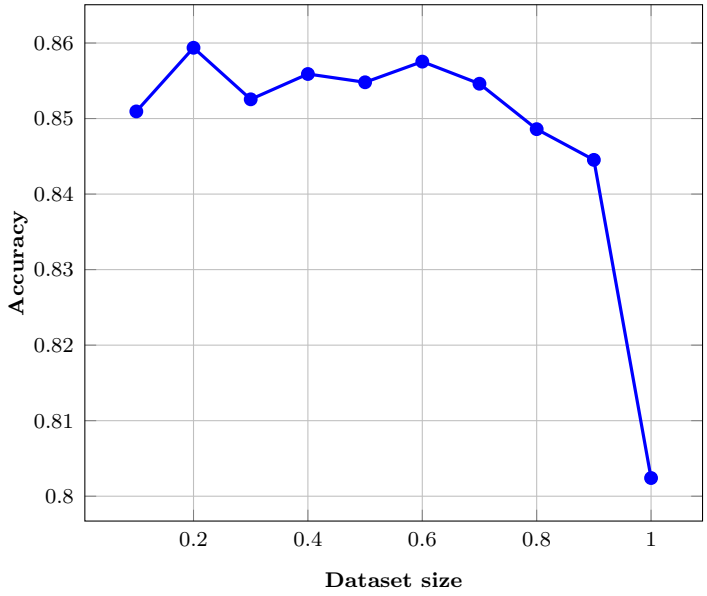


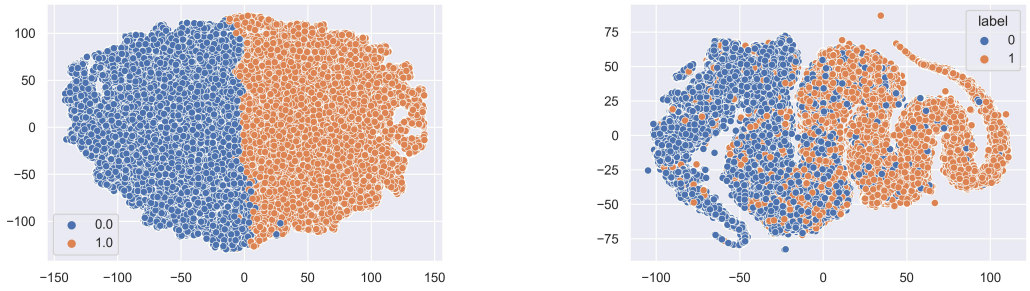
Figure 7.6: Accuracy score of the proposed approach with different proportions of the original dataset ranging from 0.1 (inclusion of 10 percent of the samples) to 1 (inclusion of all of the samples).

7.4.6 Document visualization

Aiming to evaluate our approach in a more visual manner, we compare the document embeddings learned by FastGATConv to the ones produced by the best baseline model (i.e. fastText). To do so, we use the t-SNE tool. Figure 7.7 shows the visualization of the document embeddings for the proposed approach and fastText. As it is expected, the proposed approach learns more meaningful document embeddings, which assist the t-SNE tool in splitting the samples better, as far as the labels of the documents are concerned.

7.4.7 Model explainability

Due to the complex nature of graph classification, explainability and interpretability is a challenging task. Nevertheless, providing insights about the results of a deep graph model is beneficial both to the developers so that they can pinpoint any drawbacks of the model and to the users so that they can understand the predictions



(a) FastGATConv (b) fastText

Figure 7.7: The t-SNE visualizations of document embeddings of the dataset using the proposed approach (FastGATConv) and the best baseline model (fastText).

generated. To this end, we utilize GNNExplainer in order to extract the edges and the accompanying nodes that are important for the model's prediction. In other words, GNNExplainer produces a subgraph $G_S \subseteq G$ that maximizes the mutual information between model's prediction and a given subgraph (see Figures 7.9a, 7.11a).

While, the advantages of GNNExplainer can be easily detected in small graphs/texts, the interpretability of larger graphs is not so intuitive. As a general rule, as more nodes and edges appear on a graph, the larger the final subgraph will be, hence making the explanation more obscure to the users. To remedy this shortcoming, we have employed LIME, which treats the model as a black box and detects the importance of each node/word for the final prediction. We extract the top-K words from LIME (see Figures 7.8, 7.10) which are then used to remove redundant nodes from the subgraph. Specifically, the nodes which are not included in the top-K LIME words are removed from the final subgraph, thus reducing its size and producing a far more understanable result for the user (see Figures 7.9b, 7.11b). The GNNExplainer was constructed using the built-in class of Pytorch Geometric and LIME using the lime (https://lime-ml.readthedocs.io/en/latest/) Python package.

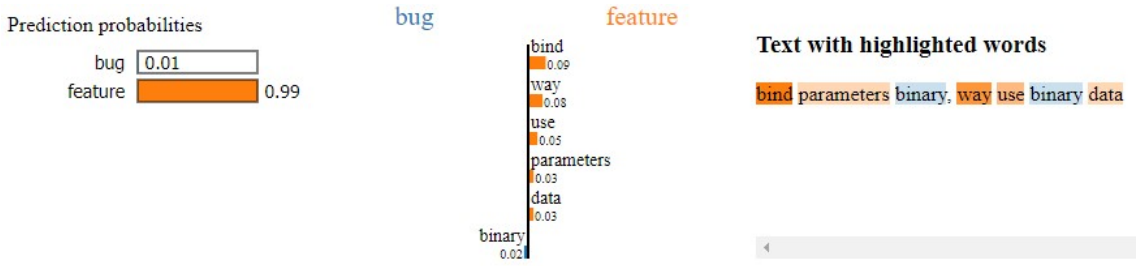
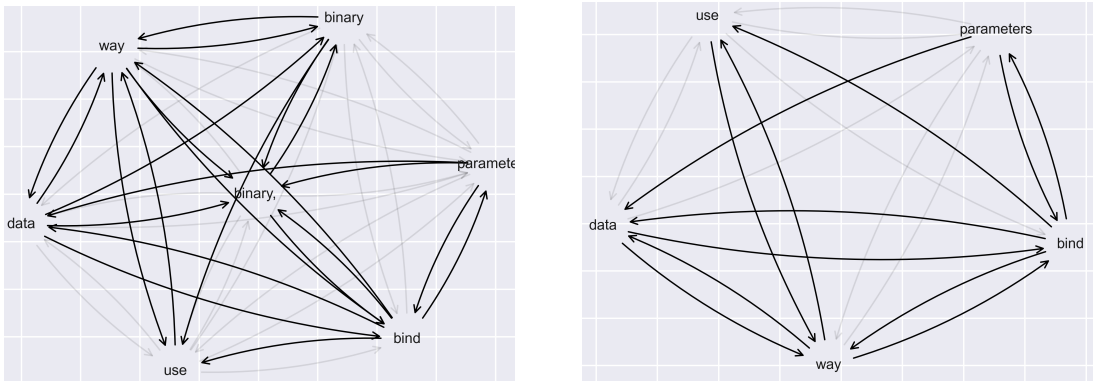


Figure 7.8: LIME example for a test instance. LIME produces a list with the importance scores for each word of the instances, where each score denotes the participation of the word to the final model prediction

7.5 Discussion and conclusions

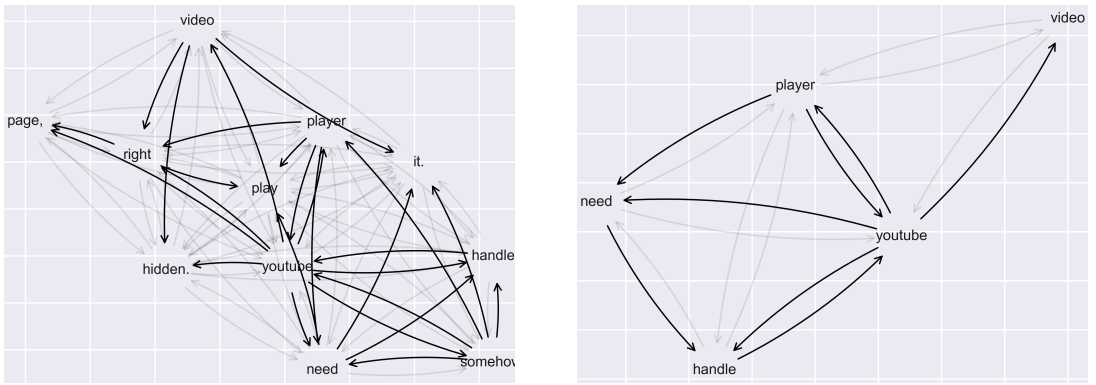
In this paper, the *graph-of-words* text representation is utilized to create a textual graph for the identification of software bugs. Each graph is then enhanced by pre-



(a) GNNExplainer (b) GNNExplainer + LIME

Figure 7.9: The subgraph produced by GNNExplainer for a test instance and the final subgraph after the nodes that are not included in the LIME list are removed. The black edges denote the edges that GNNExplainer considers as important for the model's prediction.

Figure 7.10: LIME example for a test instance. LIME produces a list with the importance scores for each word of the instances, where each score denotes the participation of the word to the final model prediction



(a) GNNExplainer (b) GNNExplainer + LIME

Figure 7.11: The subgraph produced by GNNExplainer for a test instance and the final subgraph after the nodes that are not included in the LIME list are removed. The increase in the number of nodes render the GNNExplainer subgraph incomprehensible. However, the final subgraph is much more understandable for the users due to the elimination of many redundant information.

trained *FastText* embeddings, which are utilized as node features towards improving the semantic background of the overall model. To the best of our knowledge, our approach is the first one that combines word embeddings, graph-based text representations and GNNs for the task under consideration. The experimental results show that our approach outperforms classical and graph-based machine learning models. The adopted form of text representation, accompanied by a *GAT* network, shows promising results as far as the accuracy, precision and recall metrics are concerned.

As a concluding remark, we note that GNNs that rely on local-structure propagation rules (rather than global-based ones) seem to yield better results for short textual data, suggesting that the co-occurrence of words in a document has a bigger impact on the performance of the model compared to the overall structure of the graph.

Future work directions include: (i) the inclusion of explainability features (Ying et al., 2019a); (ii) the exploitation of both the title and body attributes of a GitHub issue; (iii) the experimentation with multiple convolutional layers; and (iv) the removal of outlier GitHub issues.

Chapter 8

Using graph neural networks and document embeddings to make predictions

We propose a new approach that builds on the combination of graph neural networks and document embeddings for making predictions. The particular application concerns the prediction of prices of Airbnb listings for a specific touristic destination, namely the island of Santorini. The already existing methods rely only on the features of each individual listing, ignoring any topological or neighborhood properties. Our approach represents the listings of a given area as a graph, where each node corresponds to a listing and each edge connects two similar neighboring listings. This enables us to not only exploit the features of each individual listing, but also to take into consideration information related to its neighborhood. Our preliminary experiments demonstrate that the proposed approach outperforms a list of classical regression models as far as the coefficient of determination (R^2) is concerned and decreases the Mean Squared Error (MSE). The data of the experimentations reported in this chapter have been retrieved from the insideairbnb.com platform and describe the Airbnb listings of the island of Santorini.

8.1 Introduction

With the advent of the sharing economy era, a plethora of companies emerged in the hospitality industry (Molina-Collado et al., 2022). Among them, Airbnb is the leading one, which thrives as a peer-to-peer accommodation platform that manages more than 5 million listings worldwide. Mainly due to the ever-increasing supply of listings on the Airbnb platform, the hosts (i.e. the owners of the listings) continuously attempt to provide the candidate guests with an attractive but also highly profitable price for their listings. Generally speaking, hosts empirically adapt the prices of their listings by taking into consideration several parameters, such as the current demand, the prices of the other listings of the neighborhood, the attractiveness of the location and the characteristics of each individual listing (e.g. number of accommodates, number of bathrooms etc.). This seems to work in an efficient way in most cases. However, it requires each host to have the needed domain expertise and experience, which are only obtained in the course of time. As a result, hosts (especially the new ones) struggle and very often fail to set the appropriate

prices for their listings, which in turn results in revenue loss (Sigala, 2020).

In an attempt to assist hosts in assigning reasonable prices to their listings, various pricing models have been proposed in the literature (Kalehbasti et al., 2019; Luo et al., 2019; Tommaso et al., 2017). The existing approaches focus on popular price determinants related to the characteristics of each individual listing. They often try to incorporate information about the neighborhood of a listing by including the zip code in their formula. Although the inclusion of the zip code improves the existing models, the following issues arise: (i) it makes models biased against specific areas; (ii) it is not applicable to destinations where different price ranges exist within the same zip code area, such as the island of Santorini; (iii) the size of each neighborhood is fixed, predefined and rigidly coupled with a specific zip code; (iv) two nearby listings with different zip codes cannot be considered as neighbors.

To remedy the above shortcomings, this chapter proposes a new approach that leverages state-of-the-art techniques from the field of graph theory, Graph Neural Networks (GNNs) and document embeddings, aiming to accurately predict the price of Airbnb listings. Contrary to existing approaches, the one introduced in this chapter: (i) defines a unique neighborhood for each listing; (ii) is applicable to destinations with intense price variance within the same zip code area (e.g. small islandic destinations); (iii) allows the user to define the size of each neighborhood; (iv) works without any area bias; (v) takes into consideration both the features of each individual listing and the information regarding its neighborhood; (vi) takes into account unstructured textual data (and structured data referring to the amenities and overall characteristics of each listing) by generating and consuming document embeddings.

Overall, the main contribution of this chapter is threefold, in that:

- we meaningfully introduce techniques from the area of GNNs and graph representations in the field of price prediction for Airbnb listings;
- we provide the research community with a new approach that improves the quality of price predictions, especially in the case of small touristic destinations with a big diversity of prices;
- we investigate whether the representation of a touristic area as a graph benefits the price prediction process or not.

For the implementation of our approach, we use the Python programming language, PyTorch, PyTorch Geometric and scikit-learn libraries. To evaluate our approach, we benchmark it against a list of classical regression models (including neural networks, linear regression, random forests and decision trees) on a dataset describing the Airbnb listings of the island of Santorini. The initial experimentations demonstrate that the proposed approach overcomes the selected baseline models. Related code and data are available on GitHub (https://github.com/nkanak/predicting-prices-of-airbnb-listings).

The remainder of this chapter is organized as follows: related work is introduced in Section 8.2; Section 8.3 presents the proposed GNN-based approach, while preliminary evaluation results are reported in Section 8.4; finally, the novelty of the proposed approach, together with its limitations and future work directions, are discussed in Section 8.5.

8.2 Predicting prices of Airbnb listings

A plethora of similar solutions to predict prices for Airbnb listing exists in the literature. For instance, the authors in (E. Tang, 2015) use data from the Airbnb platform to predict prices for Airbnb listings in San Francisco. They develop a classification model that predicts the neighborhood and the price of a listing. Their ultimate goal is to build a recommender system that groups listings into discrete categories of price ranges. They use as input textual data, images, and numerical data related to each listing to train a Support Vector Machines (SVM) classifier.

Another price prediction model that aims to assist property owners and guests in evaluation prices of selected Airbnb listings is described in (Kalehbasti et al., 2019). This model is based on deep learning and NLP techniques, and takes as input numerical features of each listing, characteristics of the hosts and customer reviews. The experiments of the proposed approach demonstrate promising results with a high R^2 value.

Another approach that predicts Airbnb listing prices in the cities of New York City, Paris and Berlin using various machine learning models has been proposed in (Luo et al., 2019). The experimentation results show that the neural network based model demonstrates the better results compared to the other counterparts (e.g. linear regression, decision tree and random forest) regardless of the given city.

8.3 The proposed approach

The aim of our work is to build a GNN model to recommend an appealing and profitable price for a given listing. Our approach consists of three main steps (see Figure 8.1):

- Data preprocessing: Preprocessing original data and generating document embeddings using the description of each listing as an input;
- Graph construction: Transforming an array of Airbnb listings into a homogeneous graph;
- Node regression: Training an ML model to perform price prediction.

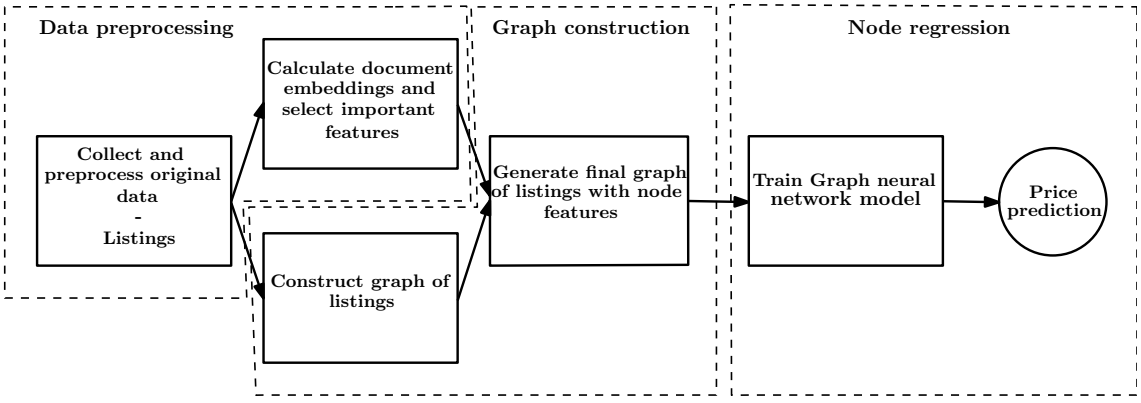


Figure 8.1: The proposed approach.

8.3.1 Data preprocessing

The first step normalizes the numerical values of the dataset and performs data cleansing. In particular, we set to zero any missing value of each sample of the dataset. Furthermore, to clean up the textual data, we utilize techniques from the field of Natural Language Processing (NLP). These techniques include text normalization, tokenization, lemmatization, removal of stop words and removal of frequently occurred words. Finally, we generate the document embeddings for each sample of the dataset using the Doc2vec algorithm (Le & Mikolov, 2014) and as an input the description of each listing.

8.3.2 Graph construction

The second step takes as input the preprocessed dataset and constructs a graph of listings. Each node of the graph corresponds to a listing. Each edge of the graph connects two nodes that their corresponding listings are close enough and have a similar textual description. Two predefined numbers are used as a distance and text similarity threshold. To calculate the distance between two listings, we utilize the haversine formula (Robusto, 1957), while to determine whether two listings have similar descriptions or not, we use the cosine similarity and the vector representation of the texts produced by the Doc2vec algorithm. The nodes of the produced graph have as attributes the features described in Table 8.1, i.e. the number of accommodates, latitude, longitude, number of reviews, calculated host listings count, number of bedrooms, number of beds, review scores rating, review scores accuracy, review scores cleanliness, review scores check-in, review scores communication, review scores location and review scores value. Finally, we enrich the feature vector by also attaching the vectors generated from the Doc2vec algorithm.

8.3.3 Node regression

In the final step, we train a Graph Convolutional Network (GCN) model to perform price prediction. This GCN model is able to exploit both structured and unstructured data related to the listings themselves (nodes of the graph) as well as characteristics with regards to the neighborhood of listings (edges of the graph). Figure 8.2 shows the architecture of the proposed GCN model. The GCN model takes as an input a graph of listings along with the attributes of each node. Furthermore, it consists of a normalization layer and a hidden convolutional layer, which extracts information with respect to the nodes and edges of the aforementioned graph. In addition, we use ReLU as an activation function for the hidden convolutional layer. The final layer is a fully connected one where its output is the desired price prediction.

8.4 Experimental evaluation

8.4.1 Dataset

For the experiments of this chapter, we retrieved data from the insideairbnb.com platform (http://insideairbnb.com/get-the-data.html). The downloaded data concern the broader area of South Aegean, Greece. This area consists of two groups of

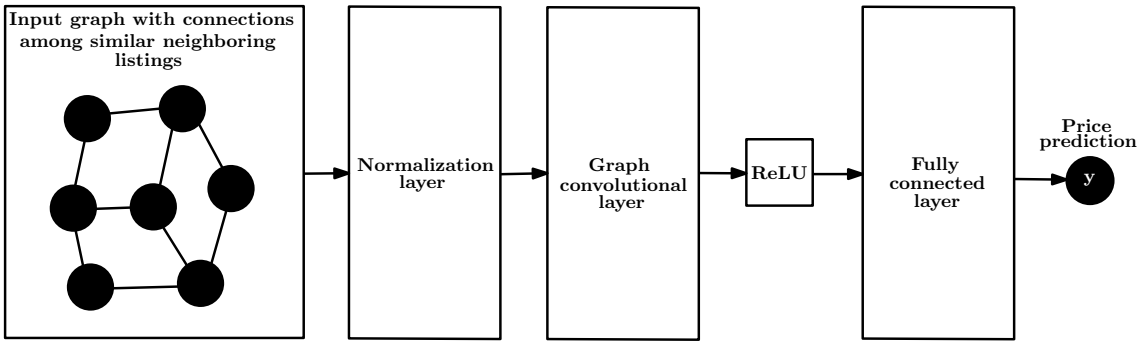


Figure 8.2: The architecture of the proposed GCN model.

Greek islands namely, Cyclades and Dodecanese. To generate our dataset, we kept only the listings of the Municipality of Santorini. The final dataset is composed of three CSV files (i.e. listings.csv, calendar.csv and reviews.csv) and describes 4540 listings of the island of Santorini. Descriptive statistics for the selected features of the dataset can be found in Table 8.1. In particular, each record of the listings.csv file corresponds to a listing and has several features; Table 8.1 presents only the list of selected features. Each record of the reviews.csv file corresponds to a review of a guest for a specific listing. Finally, calendar.csv file describes the price and the availability of each listing for each day of a year.

Name of feature	Min	Max	Average	Standard deviation
Accommodates	1	16	3.880837	2.258300
Latitude	36.333650	36.481210	36.408083	0.038471
Longitude	25.335220	25.486700	25.430520	0.031906
Number of reviews	0	861	27.475330	48.262524
Calculated host listings count	1	268	20.820044	51.783843
Bedrooms	0	43	2.220485	1.788454
Beds	0	43	2.220485	1.788454
Review scores rating	0	5	3.535092511	2.129615326
Review scores accuracy	0	5	3.52195815	2.141496003
Review scores cleanliness	0	5	3.557037445	2.156828219
Review scores check-in	0	5	3.569057269	2.166857661
Review scores communication	0	5	3.56272	2.163615
Review scores location	0	5	3.473053	2.114032
Review scores value	0	5	3.448588	2.100067

Table 8.1: Descriptive statistics for the selected features of the dataset.

The left side of Figure 8.3 shows the distribution of the Airbnb listings across the island, while the right side of the figure highlights with different color contrast the price distribution (the darker the color, the highest the price of a listing). It is observed that the majority of the listings is concentrated on the coastline and the area of the caldera. The most expensive listings are mainly located in the villages of Oia (northernmost part of the island), Imerovigli and Firostefani (neighboring villages, located in the center of the island). This is easily explained by the unique view and scenery of these highly demanding areas. Apparently, it is obvious that the price of a listing is affected not only by the provided amenities but also by its location.

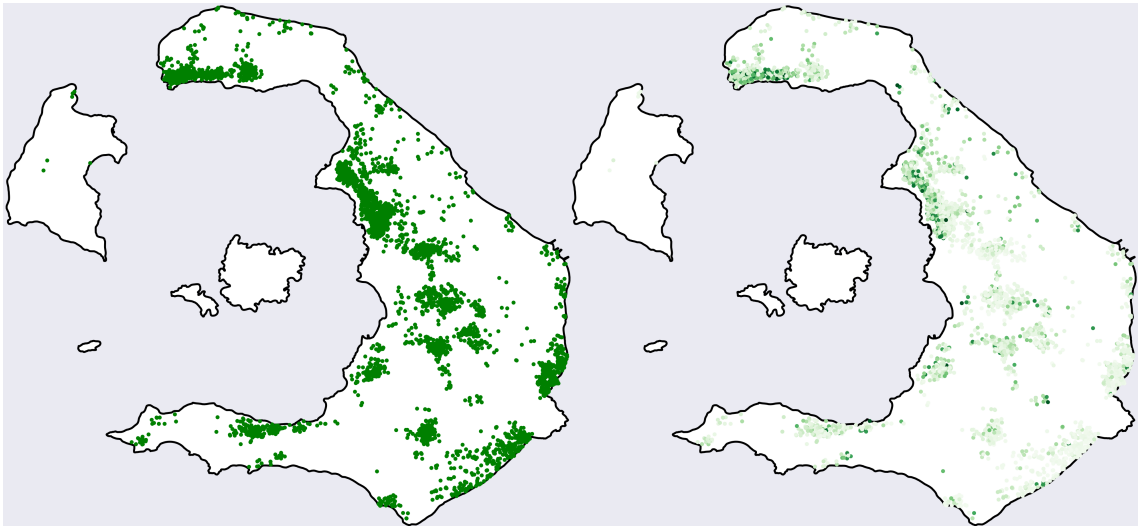


Figure 8.3: (left) The distribution of the Airbnb listings across the island, (right) The price distribution of the Airbnb listings (the darker the color, the highest the price of a listing).

8.4.2 Experimental setup

For the implementation and evaluation of our approach, we used the Python programming language, as well as the PyTorch, PyTorch Geometric and the scikit-learn ML libraries. Additionally, we trained the document embeddings using as an input the description of each listing. For the generation of the document embeddings, we utilize the genism library and the Doc2vec algorithm with a vector size of 100, a window size of 3 and a min count of 2. The full code and evaluation results of our experiments are available on GitHub (https://github.com/nkanak/predicting-prices-of-airbnb-listings).

We benchmark the proposed approach (GCNBnB) against classical regression models to investigate whether the inclusion of the information about the neighborhood of a listing improves the prediction of its price. The list of the regression models considered in this chapter includes: linear regression (LR), random forest (RF), decision tree (DT), and neural network (NN). Details about the hyperparameters and the properties of these models are summarized in Table 8.2. The features described in Table 8.1 and the document embedding produced by the Doc2vec algorithm are given as an input to the proposed and baseline models. The most notable difference between the proposed and a baseline model is that the former is able to distill extra information from the structure of the given graph of listings, which is also utilized during the training and final prediction process. An extensive list of experiments using various classifiers along with different hyperparameter configurations can be found on the GitHub repository of this chapter. The list of our evaluation metrics includes the coefficient of determination (R^2) and the Mean Squared Error (MSE) of the models.

Aiming to observe how the inclusion of the information related to the neighborhood of a listing affects the performance of GCNBnB, we created four graphs of listings from the original dataset namely, G@0, G@0.75, G@0.9 and G@09.5. Each graph of listings is created by using a different document similarity threshold (i.e. 0, 0.75, 0.9 and 0.95, respectively), resulting in an array of graphs with different

Name	Description
GCNBnB	A graph convolutional neural network with a convolutional hidden layer (PyTorch Geometric, class: GCNConv) with 100 output units. During the training phase, we use 500 epochs, a batch size of 50, the mean absolute error (MAE) as a loss function (PyTorch class: L1Loss), the ReLU activation function for the hidden layer, the Adam optimizer with a learning rate of 0.01 and a weight decay of 0.0005.
LR	A linear regression model (scikit-learn, Python class: sklearn.linear_model.LinearRegression). Whilst the implementation of linear regression is simple and easily interpretable, it frequently performs well on datasets with linear correlation between their features.
RF	A random forest model (scikit-learn, Python class: sklearn.ensemble.RandomForestRegressor) with 100 estimators and a max depth of 5.
DT	A decision tree model (scikit-learn, Python class: sklearn.tree.DecisionTreeRegressor) with a max depth of 5.
NN	A neural network model with a hidden dense layer with 100 units and a dropout layer of 0.5. During the training phase, we use 500 epochs, a batch size of 50, the MAE as a loss function (PyTorch class: L1Loss), the ReLU activation function for the hidden layer, the Adam optimizer with a learning rate of 0.0001 and a weight decay of 0.0005.

Table 8.2: Hyperparameters and properties of the models used in this chapter.

number of edges and average node degree. The characteristics (number of nodes, number of edges, etc.) of these graphs are summarized in Table 8.3.

Dataset / Feature	**G@0**	**G@0.75**	**G@0.9**	**G@0.95**
Number of nodes	4540	4540	4540	4540
Number of edges	159283	16370	7914	5426
Average node degree	70.2	7.211	3.4863	2.39030
Number of attributes of a node	114	114	114	114
Distance threshold	200	200	200	200
Document similarity threshold	0	0.75	0.9	0.95

Table 8.3: Characteristics of the G@0, G@0.75, G@0.9 and G@09.5 produced graphs.

8.4.3 Evaluation results

The evaluation results show that the consideration of the information related to the neighborhood of a listing increases the R^2 coefficient and decreases the MSE in all graphs produced in our approach (Table 8.4). The increment of the R^2 coefficient in all graphs indicates that our approach is reliable regardless of the size and the degree of the given graph. On the contrary, the baseline models produce the same evaluation results no matter the nature of the graph, as a result of their incapability to take into account features with respect to the structure and the neighborhoods of the graph of listings. It is also noted that, in all cases, GCNBnB outperforms all the baseline models.

Considering the above observations, we conclude that the inclusion of the information about the structure of the graph along with the adoption of a modern GNN-based approach is more suitable, compared to the classical ones, for the prediction of the prices of a network of Airbnb listing, since it improves both R^2 and MSE metrics in all cases considered in our work.

8.5 Discussion and conclusions

Admittedly, one of the most crucial price determinants of an Airbnb listing is its location along with its neighborhood. The already existing price prediction methods, both linear and non-linear ones, take into consideration only the features of each individual listing, while they partially ignore valuable topological and neighborhood

Method	R^2@0	MSE@0	R^2@0.75	MSE@0.75	R^2@0.9	MSE@0.9	R^2@0.95	MSE@0.95
Our approach (GCNBnB)	**0.208**	0.704	**0.281**	0.640	**0.359**	0.570	**0.4630**	0.478
Neural network (NN)	-0.171	1.043	-0.171	1.043	-0.171	1.043	-0.171	1.043
Linear regression (LR)	-0.034	0.921	-0.034	0.921	-0.034	0.921	-0.034	0.921
Random forest (RF)	0.001	0.889	0.001	0.889	0.001	0.889	0.001	0.889
Decision tree (DT)	-0.039	0.925	-0.039	0.925	-0.039	0.925	-0.039	0.925

Table 8.4: Coefficient of determination (R^2) and Mean Squared Error (MSE) metrics per regression method on the four different graphs. Bold font indicates the best score value for each column.

aspects that are available, by simply representing a list of listings as a graph. Aiming to enhance the liability and the accuracy of the predictions of the price of such listings, and focusing on touristic destinations that have a high variance of different prices in a small area, our approach builds on prominent techniques and tools from the research fields of GNNs and document embeddings. This representation shift enables one to fully exploit structured and unstructured data that exists either explicitly (e.g. the features of a listing) or implicitly (e.g. the neighborhood of a listing) in the graph. Our evaluation results show that a graph representation accompanied by a graph GNN model enhances the ability of modern ML-based pipelines to suggest accurately attractive (for the guests) and highly profitable (for the hosts) prices for a given Airbnb listing.

Furthermore, the evaluation results reveal that the addition of the postal code to the feature space of a listing as a technique to incorporate information about the neighborhood of a listing is not efficient and even leads to poor performance models. Instead, it is suggested that each listing has its own unique neighborhood that consists of other nearby similar listings. Finally, it is shown that the proposed approach outperforms the classical regression models with respect to the R^2 and MSE metrics.

Future work directions include: (i) the addition of explainability features in the proposed approach, by incorporating state-of-the-art techniques such as GNNExplainer (Ying et al., 2019b); (ii) the improvement of the document embeddings phase by relying on graph-based text representations, such as graph-of-words (Rousseau & Vazirgiannis, 2013) or graph-of-docs (Giarelis et al., 2020b); (iii) the enhancement of the preprocessing and feature selection step by taking into consideration alternative feature importance metrics; and (iv) the removal of outlier or abnormal listings (e.g. listings with many beds or bathrooms), aiming to remove noisy entries from the dataset.

Chapter 9

Conclusions

9.1 Summary and contributions

The first goal of this dissertation was to introduce a novel approach to represent multiple textual documents in a single graph, i.e. the graph-of-docs text representation. The second goal was the proposal and empirical evaluation of the combination of word embeddings, graph-based text representations and graph neural networks in a way that an improvement in the performance of the ML models is achieved, with respect to classical tasks such as text classification, feature engineering and feature selection. The final goal of the dissertation was to assess the suitability of the proposed technological combinations to specific domains and applications, that is to say, personnel selection, identification of software bugs, prediction of future research collaborations and prediction of prices for Airbnb listings; to do so, the evaluation was conducted using a variety of experiments on diverse datasets with regard to the abovementioned applications and domains.

Admittedly, the graph-of-docs text representation alleviates the effects of the drawbacks of the existing approaches, in that (i) it enables the evaluation of the importance of a word for a corpus of documents; (ii) it facilitates the representation of the relationships between a group of documents; (iii) it can easily operate with heterogeneous data; (iv) it permits the analysis of groups of documents and their interconnections (e.g. discussion threads and documents from question-answering models).

As mentioned in Chapter 1, the main contributions of this dissertation in terms of theory are the following (see Section 1.1 for a more extensive list of contributions):

- the proposition of a novel graph-based technique to represent multiple textual documents in a single graph;
- the introduction of a new graph-based feature selection method for multiple textual documents;
- the empirical evaluation of the combination of embeddings, graph-based text representations and graph neural networks to:
  - perform link prediction using unstructured textual data and knowledge graphs;
  - regression models (i.e. prediction of numerical values);

  - exploit structural and textual information as well as pre-trained word embeddings, in an integrated way.

In terms of applications, the main contributions of this dissertation are (see also Section 1.1):

- the introduction of a new ML-based personnel selection process;
- the provision of a dataset concerning the tasks of the projects of an organization, which can be used to evaluate new potential methods in similar research directions;
- the proposal of a new approach for predicting future research collaborations;
- the integration of techniques from the area of graph neural networks and graph-based text representations to the task of identifying software bugs;
- the introduction of techniques from the area of graph neural networks and graph representations in the field of price prediction for Airbnb listings.

9.2 Research findings and lessons learned

The main research findings of the dissertation can be summarized as follows. Although some of the research findings may fit in many areas, below they are grouped into 4 distinct categories.

Feature engineering

- the usage of graph-based feature selection methods, in most cases, increases the accuracy and decreases the number of features required to achieve state-of-the-art accuracy;
- graph-based feature selection methods mitigate the effects of the 'curse-of-dimensionality' phenomenon since they tend to select the most influential features of a corpus of domains;
- the inclusion of a text-related feature increases the average performance scores as far as the task of link prediction is concerned, regardless of the nature of the classification model;
- using both textual and structural characteristics of a scientific knowledge graph leads to more accurate ML models and is less prone to overfitting;
- the consideration of the information related to the neighborhood of a listing increases the R^2 coefficient and decreases the MSE as far as the node regression problem is concerned. The increment of the R^2 coefficient in all graphs indicates that our approach is reliable regardless of the size and the degree of the given graph;

- the inclusion of the information about the structure of a graph along with the adoption of a modern GNN-based approach is more suitable compared to the classical ones for the prediction of the prices of a network of Airbnb listings since it improves both R^2 and MSE metrics in all cases considered in our work;

- the addition of the postal code to the feature space of a listing as a technique to incorporate information about the neighborhood of a listing is not efficient and even leads to poor performance models. Instead, it is suggested that each listing has its own unique neighborhood that consists of other nearby similar listings.

Graph measures and indices

- including many graph measures or indices as features for a simple classification model (e.g. logistic regression) adds redundant noise, which in turns reduces the performance of the model. On the contrary, due to their non-linear natures, neural networks can benefit from the addition of such features;

- metrics or kernels for calculating textual similarities, which incorporate into their formula the terms with their corresponding attributes as well as the structure of the graph, perform better compared to metrics that deal only with the absence or the presence of a term (e.g. Jaccard similarity).

Representation learning

- classifiers that are based on embeddings produced by the node2vec algorithm are the ones that generally perform better;

- the dimension of the embeddings does not affect significantly the performance of a classifier since models relying on the same representation learning algorithms produce similar results;

- word embeddings, neural networks and graph representation learning can be used for efficiently analyzing textual data related to personnel selection;

- the utilization of graph-based text representations is essential for capturing relationships between the words of a document;

- GNNs that rely on local-structure propagation rules (rather than global-based ones) seem to yield better results for short textual data, suggesting that the co-occurrence of words in a document has a bigger impact on the performance of the model compared to the overall structure of the graph.

Model selection and hyper-parameter tuning

- neural networks are of great value when dealing with the analysis of high-dimensional textual data.

- in most cases, the graph-based models perform better than the linear and classical counterparts;

- GloVe + MLP performs similarly to most of the GNN models on datasets with short texts;
- poor word embeddings inhibit the performance of the GNN models;
- the utilzation of either GNN models or graph-based text representation does not always result in better performance on datasets with short texts;
- the mindful selection and combination of pre-trained word embeddings, textual representations and classification models are of much importance for the identification of software bugs;
- the success of the combination of fastText and GAT networks on short text datasets relies on the fact that the fastText model produces better embeddings than GloVe;
- the proper configuration and selection of a small sliding window size for the graph-of-words representation plays a critical role;
- the employment of a convolutional network that emphasizes on local-structure propagation rules instead of global-based ones is of great importance;
- a large sliding window size adds redundant and noisy edges between the nodes of a graph-of-words graph, leading to poor classification performance.

9.3 Future work directions

In the context of this dissertation, many potential future work directions have been noted, which could possibly advance the proposed approaches in different ways, for instance, by enhancing the accuracy and robustness, reducing execution time and improving interpretability. They can be grouped into three categories, namely feature engineering and model selection, explainability, time and code optimization. Below the most important general work directions are listed. For an extensive list of prominent research paths related to each individual proposed approach, see the last section of each of the previous chapters.

- experimenting with alternative centrality measures, as well as diverse community detection and graph partitioning algorithms;
- enriching the existing textual corpus through the exploitation of external domain agnostic knowledge graphs;
- integrating the proposed approaches into collaborative environments where the underlying knowledge is structured through semantically rich discourse graphs;
- incorporating techniques from the field of operations research, which allows for the inclusion of constraints and optimization rules, enabling an organization to optimize the allocation of tasks to the available employees;
- utilizing an in-memory graph database in combination with Neo4j.

Appendix A

Publications

The following papers have been published throughout the duration of my Ph.D. studies:

1. Kanakaris., N., Karacapilidis., N., & Lazanas., A. On the advancement of project management through a flexible integration of machine learning and operations research tools. In: In *Proceedings of the 8th international conference on operations research and enterprise systems - icores.* INSTICC. SciTePress, 2019, 362–369. ISBN: 978-989-758-352-0. https://doi.org/10.5220/0007387103620369

2. Kanterakis, A., Iatraki, G., Pityanou, K., Koumakis, L., Kanakaris, N., Karacapilidis, N., & Potamias, G. (2019). Towards reproducible bioinformatics: The openbio-c scientific workflow environment. *2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE)*, 221–226. https://doi.org/10.1109/BIBE.2019.00047

3. Kanakaris, N., Karacapilidis, N., Kournetas, G., & Lazanas, A. (2020b). Combining machine learning and operations research methods to advance the project management practice. In G. H. Parlier, F. Liberatore, & M. Demange (Eds.), *Operations research and enterprise systems* (pp. 135–155). Springer International Publishing

4. Kanakaris, N., Karacapilidis, N., & Kournetas, G. On the exploitation of textual descriptions for a better-informed task assignment process. In: In *Proceedings of the 9th international conference on operations research and enterprise systems - icores,* INSTICC. SciTePress, 2020, 304–310. ISBN: 978-989-758-396-4. https://doi.org/10.5220/0009151603040310

5. Giarelis, N., Kanakaris, N., & Karacapilidis, N. (2020b). On a novel representation of multiple textual documents in a single graph. In I. Czarnowski, R. J. Howlett, & L. C. Jain (Eds.), *Intelligent decision technologies* (pp. 105–115). Springer Singapore

6. Giarelis, N., Kanakaris, N., & Karacapilidis, N. (2020a). An innovative graph-based approach to advance feature selection from multiple textual documents. In I. Maglogiannis, L. Iliadis, & E. Pimenidis (Eds.), *Artificial intelligence applications and innovations* (pp. 96–106). Springer International Publishing

7. Giarelis, N., Kanakaris, N., & Karacapilidis, N. I. (2020c). On the utilization of structural and textual information of a scientific knowledge graph to discover future research collaborations: A link prediction perspective. *IFIP Working Conference on Database Semantics*

8. Kanakaris, N., Giarelis, N., Siachos, I., & Karacapilidis, N. (2021a). Shall i work with them? a knowledge graph-based approach for predicting future research collaborations. *Entropy*, *23*(6), 664

9. Giarelis, N., Kanakaris, N., & Karacapilidis, N. (2021). A comparative assessment of state-of-the-art methods for multilingual unsupervised keyphrase extraction. In I. Maglogiannis, J. Macintyre, & L. Iliadis (Eds.), *Artificial intelligence applications and innovations* (pp. 635–645). Springer International Publishing

10. Kanterakis, A., Kanakaris, N., Koutoulakis, M., Pitianou, K., Karacapilidis, N., Koumakis, L., & Potamias, G. (2021). Converting biomedical text annotated resources into fair research objects with an open science platform. *Applied Sciences*, *11*(20). https://doi.org/10.3390/app11209648

11. Kanakaris, N., Giarelis, N., Siachos, I., & Karacapilidis, N. I. (2021b). Making personnel selection smarter through word embeddings: A graph-based approach. *Machine Learning with Applications*

12. Giarelis, N., Kanakaris, N., & Karacapilidis, N. (2022). Medical knowledge graphs in the discovery of future research collaborations. In C.-P. Lim, Y.-W. Chen, A. Vaidya, C. Mahorkar, & L. C. Jain (Eds.), *Handbook of artificial intelligence in healthcare: Vol 2: Practicalities and prospects* (pp. 371–391). Springer International Publishing. https://doi.org/10.1007/978-3-030-83620-7_16

13. Kanakaris, N., Siachos, I., & Karacapilidis, N. (2022). Is it a bug or a feature? identifying software bugs using graph attention networks. *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, 1425–1429. https://doi.org/10.1109/ICTAI56018.2022.00215

14. Michail, D., Kanakaris, N., & Varlamis, I. (2022). Detection of fake news campaigns using graph convolutional networks. *International Journal of Information Management Data Insights*, *2*(2), 100104. https://doi.org/https://doi.org/10.1016/j.jjimei.2022.100104

15. Kanakaris, N., & Karacapilidis, N. (2023). Predicting prices of airbnb listings via graph neural networks and document embeddings: The case of the island of santorini [CENTERIS – International Conference on ENTERprise Information Systems / ProjMAN – International Conference on Project MANagement / HCist – International Conference on Health and Social Care Information Systems and Technologies 2022]. *Procedia Computer Science*, *219*, 705–712. https://doi.org/https://doi.org/10.1016/j.procs.2023.01.342

16. Adamides, E., Giarelis, N., Kanakaris, N., Karacapilidis, N., Konstantinopoulos, K., & Siachos, I. (2023). Leveraging open innovation practices through

a novel ict platform. In A. Zimmermann, R. Howlett, & L. C. Jain (Eds.), *Human centred intelligent systems* (pp. 3–12). Springer Nature Singapore

17. To appear: Kanakaris, N., Michail, D., Varlamis, I. A Comparative Survey of Graph Databases and Software for Social Network Analytics: The Link Prediction Perspective. Book name: Graph Databases and their use in social media and smart cities

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P. A., Vasudevan, V., Warden, P., ... Zhang, X. (2016). Tensorflow: A system for large-scale machine learning. *ArXiv*, *abs/1605.08695*.

Abduljabbar, R., Dia, H., Liyanage, S., & Bagloee, S. A. (2019). Applications of artificial intelligence in transport: An overview. *Sustainability*, *11*(1). https://doi.org/10.3390/su11010189

Adamic, L. A., & Adar, E. (2003). Friends and neighbors on the web. *Soc. Networks*, *25*, 211–230.

Adamides, E., Giarelis, N., Kanakaris, N., Karacapilidis, N., Konstantinopoulos, K., & Siachos, I. (2023). Leveraging open innovation practices through a novel ict platform. In A. Zimmermann, R. Howlett, & L. C. Jain (Eds.), *Human centred intelligent systems* (pp. 3–12). Springer Nature Singapore.

Aggarwal, C. C. (2018). *Machine learning for text* (1st). Springer Publishing Company, Incorporated.

Albert, R., & Barabasi, A. L. (2001). Statistical mechanics of complex networks. *ArXiv*, *cond-mat/0106096*.

Alkhraisat, H. (2016). Issue tracking system based on ontology and semantic similarity computation. *International Journal of Advanced Computer Science and Applications*, *7*(11). https://doi.org/10.14569/IJACSA.2016.071132

Almeida, F., & Xexéo, G. (2019). Word embeddings: A survey. *ArXiv*, *abs/1901.09069*.

Altan, A., & Karasu, S. (2019). The effect of kernel values in support vector machine to forecasting performance of financial time series and cognitive decision making. *4*, 17–21.

Altan, A., Karasu, S., & Zio, E. (2021). A new hybrid model for wind speed forecasting combining long short-term memory neural network, decomposition methods and grey wolf optimizer. *Applied Soft Computing*, *100*, 106996. https://doi.org/https://doi.org/10.1016/j.asoc.2020.106996

Andreas, J., & Klein, D. (2014). How much do word embeddings encode about syntax? *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 822–827. https://doi.org/10.3115/v1/P14-2133

Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., Paliouras, G., & Spyropoulos, C. D. (2000). An evaluation of naive bayesian anti-spam filtering. *ArXiv*, *cs.CL/0006013*.

Armenatzoglou, N., Pham, H., Ntranos, V., Papadias, D., & Shahabi, C. (2015). Real-time multi-criteria social graph partitioning: A game theoretic approach.

Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 1617–1628. https://doi.org/10.1145/2723372.2749450

Arrieta, A. B., Diaz-Rodriguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, *58*, 82–115.

Awoyemi, J. O., Adetunmbi, A. O., & Oluwadare, S. A. (2017). Credit card fraud detection using machine learning techniques: A comparative analysis. *2017 International Conference on Computing Networking and Informatics (ICCNI)*, 1–9. https://doi.org/10.1109/ICCNI.2017.8123782

Azzini, A., Galimberti, A., Marrara, S., & Ratti, E. (2018). A classifier to identify soft skills in a researcher textual description. *EvoApplications*.

Baldassarre, F., & Azizpour, H. (2019). Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686*.

Bharadwaj, S., & Kadam, T. (2022). Github issue classification using bert-style models. *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*, 40–43.

Bjorvatn, T., & Wald, A. (2018). Project complexity and team-level absorptive capacity as drivers of project management performance. *International Journal of Project Management*, *36*(6), 876–888. https://doi.org/https://doi.org/10.1016/j.ijproman.2018.05.003

Blanco, R., & Lioma, C. (2012). Graph-based term weighting for information retrieval. *Information Retrieval*, *15*(1), 54–92. https://doi.org/10.1007/s10791-011-9172-x

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, *2008*(10), P10008. https://doi.org/10.1088/1742-5468/2008/10/p10008

Borgwardt, K. M., & Kriegel, H.-P. (2005). Shortest-path kernels on graphs. *Fifth IEEE International Conference on Data Mining (ICDM'05)*, 8 pp.-.

Boudin, F. (2013). A comparison of centrality measures for graph-based keyphrase extraction. *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, 834–838. https://aclanthology.org/I13-1102

Bougouin, A., Boudin, F., & Daille, B. (2013). TopicRank: Graph-based topic ranking for keyphrase extraction. *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, 543–551. https://aclanthology.org/I13-1062

Cabot, J., Izquierdo, J. L. C., Cosentino, V., & Rolandi, B. (2015). Exploring the use of labels to categorize issues in open-source software projects. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 550–554.

Chen, H., Sultan, S. F., Tian, Y., Chen, M., & Skiena, S. (2019). Fast and accurate network embeddings via very sparse random projection. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 399–408. https://doi.org/10.1145/3357384.3357879

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn en-

coder–decoder for statistical machine translation. *Conference on Empirical Methods in Natural Language Processing*.

Dam, H. K., Tran, T., Grundy, J., Ghose, A., & Kamei, Y. (2019). Towards effective ai-powered agile project management. *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 41–44. https://doi.org/10.1109/ICSE-NIER.2019.00019

Fakhraei, S., Foulds, J. R., Shashanka, M., & Getoor, L. (2015). Collective spammer detection in evolving multi-relational social networks. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Fatima, T., Azam, F., Anwar, M. W., & Rasheed, Y. (2020). A systematic review on software project scheduling and task assignment approaches. *Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence*, 369–373. https://doi.org/10.1145/3404555.3404588

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. *ArXiv*, *abs/1903.02428*.

Fire, M., Tenenboim-Chekina, L., Lesser, O., Puzis, R., Rokach, L., & Elovici, Y. (2011). Link prediction in social networks using computationally efficient topological features. *2011 IEEE Third Int'l Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third Int'l Conference on Social Computing*, 73–80.

Fortunato, S. (2009). Community detection in graphs. *ArXiv*, *abs/0906.0612*.

Gärtner, T., Flach, P. A., & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. *Annual Conference Computational Learning Theory*.

Gaspars-Wieloch, H. (2021). The assignment problem in human resource project management under uncertainty. *Risks*, *9*(1). https://doi.org/10.3390/risks9010025

Giarelis, N., Kanakaris, N., & Karacapilidis, N. (2020a). An innovative graph-based approach to advance feature selection from multiple textual documents. In I. Maglogiannis, L. Iliadis, & E. Pimenidis (Eds.), *Artificial intelligence applications and innovations* (pp. 96–106). Springer International Publishing.

Giarelis, N., Kanakaris, N., & Karacapilidis, N. (2020b). On a novel representation of multiple textual documents in a single graph. In I. Czarnowski, R. J. Howlett, & L. C. Jain (Eds.), *Intelligent decision technologies* (pp. 105–115). Springer Singapore.

Giarelis, N., Kanakaris, N., & Karacapilidis, N. (2021). A comparative assessment of state-of-the-art methods for multilingual unsupervised keyphrase extraction. In I. Maglogiannis, J. Macintyre, & L. Iliadis (Eds.), *Artificial intelligence applications and innovations* (pp. 635–645). Springer International Publishing.

Giarelis, N., Kanakaris, N., & Karacapilidis, N. (2022). Medical knowledge graphs in the discovery of future research collaborations. In C.-P. Lim, Y.-W. Chen, A. Vaidya, C. Mahorkar, & L. C. Jain (Eds.), *Handbook of artificial intelligence in healthcare: Vol 2: Practicalities and prospects* (pp. 371–391). Springer International Publishing. https://doi.org/10.1007/978-3-030-83620-7_16

Giarelis, N., Kanakaris, N., & Karacapilidis, N. I. (2020c). On the utilization of structural and textual information of a scientific knowledge graph to discover

future research collaborations: A link prediction perspective. *IFIP Working Conference on Database Semantics*.

Gori, M., Monfardini, G., & Scarselli, F. (2005). A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, *2*, 729–734 vol. 2.

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864.

Guns, R., & Rousseau, R. (2014). Recommending research collaborations using link prediction and random forest classifiers. *Scientometrics*, *101*, 1461–1473.

Haidabrus, B., Grabis, J., & Protsenko, S. (2021). Agile project management based on data analysis for information management systems. *Design, Simulation, Manufacturing: The Innovation Exchange*, 174–182.

Hamilton, W. L., Ying, R., & Leskovec, J. (2017a). Inductive representation learning on large graphs. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 1025–1035.

Hamilton, W. L., Ying, R., & Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *CoRR*, *abs/1709.05584*. http://arxiv.org/abs/1709.05584

Hassanien, H. E.-D., & Elragal, A. (2021). Deep learning for enterprise systems implementation lifecycle challenges: Research directions. *Informatics*, *8*(1). https://doi.org/10.3390/informatics8010011

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Helming, J., Arndt, H., Hodaie, Z., Koegel, M., & Narayan, N. (2011). Automatic assignment of work items. In L. A. Maciaszek & P. Loucopoulos (Eds.), *Evaluation of novel approaches to software engineering* (pp. 236–250). Springer Berlin Heidelberg.

Henni, K., Mezghani, N., & Gouin-Vallerand, C. (2018). Unsupervised graph-based feature selection via subspace and pagerank centrality. *Expert Syst. Appl.*, *114*, 46–53.

Heyn, V., & Paschke, A. (2013). Semantic jira - semantic expert finder in the bug tracking tool jira. *CoRR*, *abs/1312.5150*. http://arxiv.org/abs/1312.5150

Huang, J., Zhuang, Z., Li, J., & Giles, C. L. (2008). Collaboration over time: Characterizing and modeling network evolution. *Web Search and Data Mining*.

Ienco, D., Meo, R., & Botta, M. (2008). Using pagerank in feature selection. *Sistemi Evoluti per Basi di Dati*.

Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, *37*, 547–579.

Jonsson, L., Borg, M., Broman, D., Sandahl, K., Eldh, S., & Runeson, P. (2016). Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering*, *21*(4), 1533–1578. https://doi.org/10.1007/s10664-015-9401-9

Joshi, G., Walambe, R., & Kotecha, K. (2021). A review on explainability in multimodal deep neural nets. *IEEE Access*, *9*, 59800–59821.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. *CoRR*, *abs/1612.03651*. http://arxiv.org/abs/1612.03651

Julian, K. D., & Lu, W. (2016). Application of machine learning to link prediction.

Kalehbasti, P. R., Nikolenko, L., & Rezaei, H. (2019). Airbnb price prediction using machine learning and sentiment analysis. *International Cross-Domain Conference on Machine Learning and Knowledge Extraction*.

Kallis, R., Di Sorbo, A., Canfora, G., & Panichella, S. (2021). Predicting issue types on github. *Science of Computer Programming*, *205*, 102598.

Kanakaris, N., Giarelis, N., Siachos, I., & Karacapilidis, N. (2021a). Shall i work with them? a knowledge graph-based approach for predicting future research collaborations. *Entropy*, *23*(6), 664.

Kanakaris, N., Giarelis, N., Siachos, I., & Karacapilidis, N. I. (2021b). Making personnel selection smarter through word embeddings: A graph-based approach. *Machine Learning with Applications*.

Kanakaris, N., & Karacapilidis, N. (2023). Predicting prices of airbnb listings via graph neural networks and document embeddings: The case of the island of santorini [CENTERIS – International Conference on ENTERprise Information Systems / ProjMAN – International Conference on Project MANagement / HCist – International Conference on Health and Social Care Information Systems and Technologies 2022]. *Procedia Computer Science*, *219*, 705–712. https://doi.org/https://doi.org/10.1016/j.procs.2023.01.342

Kanakaris, N., Karacapilidis, N., & Kournetas, G. On the exploitation of textual descriptions for a better-informed task assignment process. In: In *Proceedings of the 9th international conference on operations research and enterprise systems - icores,* INSTICC. SciTePress, 2020, 304–310. ISBN: 978-989-758-396-4. https://doi.org/10.5220/0009151603040310.

Kanakaris, N., Karacapilidis, N., Kournetas, G., & Lazanas, A. (2020b). Combining machine learning and operations research methods to advance the project management practice. In G. H. Parlier, F. Liberatore, & M. Demange (Eds.), *Operations research and enterprise systems* (pp. 135–155). Springer International Publishing.

Kanakaris., N., Karacapilidis., N., & Lazanas., A. On the advancement of project management through a flexible integration of machine learning and operations research tools. In: In *Proceedings of the 8th international conference on operations research and enterprise systems - icores*. INSTICC. SciTePress, 2019, 362–369. ISBN: 978-989-758-352-0. https://doi.org/10.5220/0007387103620369.

Kanakaris, N., Siachos, I., & Karacapilidis, N. (2022). Is it a bug or a feature? identifying software bugs using graph attention networks. *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, 1425–1429. https://doi.org/10.1109/ICTAI56018.2022.00215

Kanterakis, A., Iatraki, G., Pityanou, K., Koumakis, L., Kanakaris, N., Karacapilidis, N., & Potamias, G. (2019). Towards reproducible bioinformatics: The openbio-c scientific workflow environment. *2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE)*, 221–226. https://doi.org/10.1109/BIBE.2019.00047

Kanterakis, A., Kanakaris, N., Koutoulakis, M., Pitianou, K., Karacapilidis, N., Koumakis, L., & Potamias, G. (2021). Converting biomedical text annotated

resources into fair research objects with an open science platform. *Applied Sciences*, *11*(20). https://doi.org/10.3390/app11209648

Karacapilidis, N., Papadias, D., Gordon, T., & Voss, H. (1997). Collaborative environmental planning with geomed. *European Journal of Operational Research*, *102*(2), 335–346. https://doi.org/https://doi.org/10.1016/S0377-2217(97)00113-6

Karacapilidis, N., Tzagarakis, M., Karousos, N., Gkotsis, G., Kallistros, V., Christodoulou, S., & Mettouris, C. (2009). Tackling cognitively-complex collaboration with cope_it! *International Journal of Web-Based Learning and Teaching Technologies (IJWLTT)*, *4*(3), 22–38.

Karasu, S., Altan, A., Bekiros, S., & Ahmad, W. (2020). A new forecasting model with wrapper-based feature selection approach using multi-objective optimization technique for chaotic crude oil time series. *Energy*, *212*, 118750. https://doi.org/https://doi.org/10.1016/j.energy.2020.118750

Khleel, N. A. A., & Nehéz, K. (2021). Comprehensive study on machine learning techniques for software bug prediction. *International Journal of Advanced Computer Science and Applications*, *12*(8).

Kholghi, M., De Vine, L., Sitbon, L., Zuccon, G., & Nguyen, A. (2016). The benefits of word embeddings features for active learning in clinical information extraction. *Proceedings of the Australasian Language Technology Association Workshop 2016*, 25–34. https://www.aclweb.org/anthology/U16-1003

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. 2017. *ArXiv abs/1609.02907*.

Landherr, A., Friedl, B., & Heidemann, J. (2010). A critical review of centrality measures in social networks. *Business & Information Systems Engineering*, *2*(6), 371–385. https://doi.org/10.1007/s12599-010-0127-3

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, II–1188–II–1196.

Levorato, V., & Petermann, C. (2011). Detection of communities in directed networks based on strongly p-connected components. *2011 International Conference on Computational Aspects of Social Networks (CASoN)*, 211–216. https://doi.org/10.1109/CASON.2011.6085946

Li, S., Huang, J., Zhang, Z., Liu, J., Huang, T., & Chen, H. (2018). Similarity-based future common neighbors model for link prediction in complex networks. *Scientific reports*, *8*(1), 1–11.

Liben-Nowell, D., & Kleinberg, J. M. (2007). The link-prediction problem for social networks. *J. Assoc. Inf. Sci. Technol.*, *58*, 1019–1031.

Linardatos, P., Papastefanopoulos, V., & Kotsiantis, S. B. (2020). Explainable ai: A review of machine learning interpretability methods. *Entropy*, *23*.

Lu, H., Halappanavar, M., Kalyanaraman, A., & Choudhury, S. (2014). Parallel heuristics for scalable community detection. *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, 1374–1385. https://doi.org/10.1109/IPDPSW.2014.155

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *NIPS*.

Luo, Y., Zhou, X., & Zhou, Y. (2019). Predicting airbnb listing price across different cities.

Maaten, L., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, *9*(86), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

Mahdi, M. N., Mohamed Zabil, M. H., Ahmad, A. R., Ismail, R., Yusoff, Y., Cheng, L. K., Azmi, M. S. B. M., Natiq, H., & Happala Naidu, H. (2021). Software project management using machine learning technique—a review. *Applied Sciences*, *11*(11), 5183.

Maiya, A. S. (2020). Ktrain: A low-code library for augmented machine learning. *ArXiv*, *abs/2004.10703*.

Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems [Software available from tensorflow.org]. https://www.tensorflow.org/

McDonald, D. W., & Ackerman, M. S. (2000). Expertise recommender: A flexible recommendation system and architecture. *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, 231–240. https://doi.org/10.1145/358916.358994

Michail, D., Kanakaris, N., & Varlamis, I. (2022). Detection of fake news campaigns using graph convolutional networks. *International Journal of Information Management Data Insights*, *2*(2), 100104. https://doi.org/https://doi.org/10.1016/j.jjimei.2022.100104

Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing order into text. *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, 404–411. https://aclanthology.org/W04-3252

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, *abs/1301.3781*. http://dblp.uni-trier.de/db/journals/corr/corr1301.html#abs-1301-3781

Mikolov, T., Chen, K., Corrado, G. S., & Dean, J. (2013b). Efficient estimation of word representations in vector space. http://arxiv.org/abs/1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013c). Distributed representations of words and phrases and their compositionality. *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, 3111–3119.

Miller, J. J. (2013). Graph database applications and concepts with neo4j.

Mo, Y., Zhao, D., Du, J., Syal, M., Aziz, A., & Li, H. (2020). Automated staff assignment for building maintenance using natural language processing. *Automation in Construction*, *113*, 103150. https://doi.org/https://doi.org/10.1016/j.autcon.2020.103150

Molina-Collado, A., Gómez-Rico, M. F., Sigala, M., Molina, M. V., Aranda, E., & Salinero, Y. (2022). Mapping tourism and hospitality research on information and communication technology: A bibliometric and scientific approach. *Information Technology & Tourism*, *24*, 299–340.

Molokwu, B. C., & Kobti, Z. (2019). Event prediction in complex social graphs via feature learning of vertex embeddings. *International Conference on Neural Information Processing*.

Monge, A. E., & Elkan, C. P. (1997). An efficient domain-independent algorithm for detecting approximately duplicate database records. *Workshop on Research Issues on Data Mining and Knowledge Discovery*.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2018). Weisfeiler and leman go neural: Higher-order graph neural networks. https://doi.org/10.48550/ARXIV.1810.02244

Nathani, D., Chauhan, J., Sharma, C., & Kaul, M. (2019). Learning attention-based embeddings for relation prediction in knowledge graphs. *Annual Meeting of the Association for Computational Linguistics*.

Neumann, M., Garnett, R., Bauckhage, C., & Kersting, K. (2016). Propagation kernels: Efficient graph kernels from propagated information. *Machine Learning*, *102*, 209–245.

Nijssen, S., & Kok, J. N. (2004). A quickstart in frequent structure mining can make a difference. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 647–652. https://doi.org/10.1145/1014052.1014134

Nikolentzos, G., Meladianos, P., Rousseau, F., Stavrakas, Y., & Vazirgiannis, M. (2017a). Shortest-path graph kernels for document similarity. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 1890–1900. https://doi.org/10.18653/v1/D17-1202

Nikolentzos, G., Meladianos, P., & Vazirgiannis, M. (2017b). Matching node embeddings for graph similarity. *AAAI Conference on Artificial Intelligence*.

Nikolentzos, G., Siglidis, G., & Vazirgiannis, M. (2021). Graph kernels: A survey. *J. Artif. Intell. Res.*, *72*, 943–1027.

Ohsawa, Y., Benson, N., & Yachida, M. (1998). Keygraph: Automatic indexing by co-occurrence graph based on building construction metaphor. *Proceedings IEEE International Forum on Research and Technology Advances in Digital Libraries -ADL'98-*, 12–18. https://doi.org/10.1109/ADL.1998.670375

Panagopoulos, G., Tsatsaronis, G., & Varlamis, I. (2017). Detecting rising stars in dynamic collaborative networks. *J. Informetrics*, *11*, 198–222.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Louppe, G., Prettenhofer, P., Weiss, R., Weiss, R. J., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, *12*, 2825–2830.

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. https://doi.org/10.3115/v1/D14-1162

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710. https://doi.org/10.1145/2623330.2623732

Pho, P. D., & Mantzaris, A. V. (2020). Regularized simple graph convolution (sgc) for improved interpretability of large datasets. *Journal of Big Data*, *7*, 1–17.

Ponomariov, B. L., & Boardman, C. (2016). What is co-authorship? *Scientometrics*, *109*, 1939–1963.

Powell, M., Rotz, J. A., & D O'Malley, K. (2020). How machine learning is improving us navy customer support. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(08), 13188–13195.

Ragesh, R., Sellamanickam, S., Iyer, A., Bairi, R., & Lingam, V. (2021). Hetegcn: Heterogeneous graph convolutional networks for text classification. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 860–868.

Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, *76*, 036106. https://doi.org/10.1103/PhysRevE.76.036106

Ramon, J., & Gaertner, T. (2003). Expressivity versus efficiency of graph kernels.

Rawat, D. S., & Kashyap, N. K. (2017). Graph database: A complete gdbms survey.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 1135–1144.

Robusto, C. (1957). The cosine-haversine formula. *American Mathematical Monthly*, *64*, 38.

Rousseau, F., Kiagias, E., & Vazirgiannis, M. (2015). Text categorization as a graph classification problem. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 1702–1712. https://doi.org/10.3115/v1/P15-1164

Rousseau, F., & Vazirgiannis, M. (2013). Graph-of-word and tw-idf: New approach to ad hoc ir. *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, 59–68. https://doi.org/10.1145/2505515.2505671

Sahu, K., & Srivastava, R. (2021). Predicting software bugs of newly and large datasets through a unified neuro-fuzzy approach: Reliability perspective. *Advances in Mathematics: Scientific Journal*, *10*(1), 543–555.

Saigo, H., Nowozin, S., Kadowaki, T., Kudo, T., & Tsuda, K. (2009). Gboost: A mathematical programming approach to graph classification and regression. *Machine Learning*, *75*(1), 69–89. https://doi.org/10.1007/s10994-008-5089-z

Sajedi-Badashian, A., & Stroulia, E. (2020). Guidelines for evaluating bug-assignment research [e2250 smr.2250]. *Journal of Software: Evolution and Process*, *32*(9), e2250. https://doi.org/https://doi.org/10.1002/smr.2250

Seidman, S. B. (1983). Network structure and minimum degree. *Social Networks*, *5*(3), 269–287. https://doi.org/https://doi.org/10.1016/0378-8733(83)90028-X

Sezer, A., & Altan, A. (2021). Detection of solder paste defects with an optimization-based deep learning model using image processing techniques. *Soldering & Surface Mount Technology*.

Sigala, M. (2020). Tourism and covid-19: Impacts and implications for advancing and resetting industry and research. *Journal of Business Research*, *117*, 312–321.

Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., & Vazirgiannis, M. (2020). Grakel: A graph kernel library in python. *J. Mach. Learn. Res.*, *21*, 54:1–54:5.

Smeaton, A. F. (1996). An overview of information retrieval. *Information Retrieval and Hypertext*, 3–25.

Sonawane, S. S., & Kulkarni, P. A. (2014). Graph based representation and analysis of text document: A survey of techniques. *International Journal of Computer Applications*, *96*, 1–8.

Sun, Y., Barber, R., Gupta, M., Aggarwal, C. C., & Han, J. (2011). Co-author relationship prediction in heterogeneous bibliographic networks. *2011 International Conference on Advances in Social Networks Analysis and Mining*, 121–128.

Tang, E. (2015). Neighborhood and price prediction for san francisco airbnb listings.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). Line: Large-scale information network embedding. *Proceedings of the 24th International Conference on World Wide Web*, 1067–1077. https://doi.org/10.1145/2736277.2741093

Tixier, A., Malliaros, F., & Vazirgiannis, M. (2016). A graph degeneracy-based approach to keyword extraction. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 1860–1870. https://doi.org/10.18653/v1/D16-1191

Tommaso, P. D., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., & Notredame, C. (2017). Nextflow enables reproducible computational workflows. *Nature Biotechnology*, *35*, 316–319.

Tran, H. M., Le, S. T., Nguyen, S. V., & Ho, P. T. (2019). An analysis of software bug reports using machine learning techniques. *SN Computer Science*, *1*(1), 4. https://doi.org/10.1007/s42979-019-0004-1

Tran, P. V. (2018). Learning to make predictions on graphs with autoencoders. *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, 237–245.

Truica, C., & Barnoschi, A. (2019). Innovating HR using an expert system for recruiting IT specialists - ESRIT. *CoRR*, *abs/1906.04915*. http://arxiv.org/abs/1906.04915

Vahdati, S., Palma, G., Nath, R. J., Lange, C., Auer, S., & Vidal, M.-E. (2018). Unveiling scholarly communities over knowledge graphs. *International Conference on Theory and Practice of Digital Libraries*.

van der Heijden, N., Abnar, S., & Shutova, E. (2020). A comparison of architectures and pretraining methods for contextualized multilingual word embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(05), 9090–9097. https://doi.org/10.1609/aaai.v34i05.6443

Vanneschi, L., Horn, D. M., Castelli, M., & Popovič, A. (2018). An artificial intelligence system for predicting customer default in e-commerce. *Expert Systems with Applications*, *104*, 1–21. https://doi.org/https://doi.org/10.1016/j.eswa.2018.03.025

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Vathy-Fogarassy, Á., & Abonyi, J. (2013). Graph-based clustering and data visualization algorithms. *SpringerBriefs in Computer Science*.

Vazirgiannis, M., Malliaros, F. D., & Nikolentzos, G. (2018). Graphrep: Boosting text mining, nlp and information retrieval with graphs. *Proceedings of the*

27th ACM International Conference on Information and Knowledge Management.

Veira, N., Keng, B., Padmanabhan, K., & Veneris, A. G. (2019). Unsupervised embedding enhancements of knowledge graphs using textual associations. *International Joint Conference on Artificial Intelligence*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., & Borgwardt, K. M. (2008). Graph kernels. *ArXiv*, *abs/0807.0093*.

Wang, L. L., Lo, K., Chandrasekhar, Y., Reas, R., Yang, J., Eide, D., Funk, K., Kinney, R. M., Liu, Z., Merrill, W. C., Mooney, P., Murdick, D. A., Rishi, D., Sheehan, J., Shen, Z., Stilson, B., Wade, A. D., Wang, K., Wilhelm, C., ... Kohlmeier, S. (2020). Cord-19: The covid-19 open research dataset. *ArXiv*.

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, *29*, 2724–2743.

Wang, W., Wang, C., Zhu, Y., Shi, B., Pei, J., Yan, X., & Han, J. (2005). Graphminer: A structural pattern-mining system for large disk-based graph databases and its applications. *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 879–881. https://doi.org/10.1145/1066157.1066273

Wang, Z., & Li, J.-Z. (2016). Text-enhanced representation learning for knowledge graph. *International Joint Conference on Artificial Intelligence*.

West, D. B., et al. (2001). *Introduction to graph theory* (Vol. 2). Prentice hall Upper Saddle River.

Wowczko, I. A. (2015). Skills and vacancy analysis with data mining techniques. *Informatics*, *2*(4), 31–49. https://doi.org/10.3390/informatics2040031

Xia, X., Lo, D., Shihab, E., Wang, X., & Zhou, B. (2015). Automatic, high accuracy prediction of reopened bugs. *Automated Software Engineering*, *22*(1), 75–109.

Yan, X., & Han, J. (2002). Gspan: Graph-based substructure pattern mining. *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, 721–724. https://doi.org/10.1109/ICDM.2002.1184038

Yang, Z., Algesheimer, R., & Tessone, C. J. (2016). A comparative analysis of community detection algorithms on artificial networks. *Scientific Reports*, *6*(1), 30750. https://doi.org/10.1038/srep30750

Yao, L., Mao, C., & Luo, Y. (2019). Graph convolutional networks for text classification. *Proceedings of the AAAI conference on artificial intelligence*, *33*(01), 7370–7377.

Ye, Z., Jiang, G., Liu, Y., Li, Z., & Yuan, J. (2020). Document and word representations generated by graph convolutional network and bert for short text classification. In *Ecai 2020* (pp. 2275–2281). IOS Press.

Ying, R., Bourgeois, D., You, J., Zitnik, M., & Leskovec, J. (2019a). Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, *32*, 9240–9251.

Ying, R., Bourgeois, D., You, J., Zitnik, M., & Leskovec, J. (2019b). Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, *32*, 9240–9251.

Yu, Q., Long, C., Lv, Y., Shao, H., He, P., & Duan, Z. (2014). Predicting co-author relationship in medical co-authorship networks. *PLoS ONE*, *9*.

Yuan, H., Tang, J., Hu, X., & Ji, S. (2020). Xgnn: Towards model-level explanations of graph neural networks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 430–438.

Yuan, H., Yu, H., Gui, S., & Ji, S. (2022). Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zhang, H., & Zhang, J. (2020). Text graph transformer for document classification. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., & Sun, M. (2018). Graph neural networks: A review of methods and applications. *ArXiv*, *abs/1812.08434*.

Zhu, X., & Ghahramani, Z. (2002). *Learning from labeled and unlabeled data with label propagation* (tech. rep.).