UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE

Master's thesis
(Magistrsko delo)

NETWORK BASED PREDICTIVE MODELLING

(Modeliranje napovedovanja s pomočjo omrežij)

Ime in priimek: *Lisi Qarkaxhija*
Študijski program: *Podatkovna znanost, 2. stopnja*
Mentor: *izr. prof. dr. Miklós Kreśz*
Somentor: *asist. Lászlo Hajdu*

Koper, junij 2021

Ključna dokumentacijska informacija

Ime in PRIIMEK: Lisi QARKAXHIJA
Naslov magistrskega dela: Modeliranje napovedovanja s pomočjo omrežij

Kraj: Koper
Leto: 2021

Število listov: 91 Število slik: 40 Število tabel: 15
Število prilog: 1 Število strani prilog: 2
Število referenc: 61

Mentor: izr. prof. dr. Mikós Kreśz

Somentor: asist. Lászlo Hajdu

UDK: 004.032.26(043.2)

Ključne besede: omrežja citatov, napoved števila citatov, globoke nevronske mreže, odločitvena drevesa za pospeševanje gradienta, načrtovanje in upravljanje z lastnostmi

Izvleček:
V zadnjih letih je bilo objavljenih veliko člankov z različno stopnjo kakovosti in vpliva. Število citatov je pomembna metrika za določanje vpliva in kakovosti članka. V tej nalogi preučujemo problem napovedovanja števila citatov, ki jih bo prejelo znanstveno delo. Napovedovanje števila citatov člankov je dobro znan in dobro preučen problem, za katerega so bili predlagani različni pristopi, vključno s preučevanjem citatnega omrežja, omrežja soavtorjev in uporabo informacij po objavi, kot so npr. povzetek, naslov, prizorišče in pretklo število citatov. Predlagamo nov pristop za napovedovanje dolgoročnega števila citatov, ki temelji na strukturi in razvoju časovnih omrežij citiranja. Poskusi kažejo, da naša inovativna metoda presega najsodobnejše pristope za napovedovanje števila citatov člankov. Ker ta metoda temelji izključno na časovnih omrežjih, ima široko paleto aplikacij na različnih področjih, kot so socialna omrežja, svetovni splet, biološka omrežja itd.

Key document information

Name and SURNAME: Lisi QARKAXHIJA
Title of the thesis: Network based predictive modelling

Place: Koper
Year: 2021

Number of pages: 91 Number of figures: 40 Number of tables: 15
Number of appendices: 1 Number of appendix pages: 2
Number of references: 61

Mentor: assoc. prof. Miklós Kreśz, PhD

Co-Mentor: assist. Lászlo Hajdu

UDC: 004.032.26(043.2)

Keywords: citation network, citation count prediction, deep neural networks, gradient boosted decision trees, feature engineering

Abstract:
In the recent years, a large number of papers have been published with varying degrees of quality and influence. The citation count is an important metric for determining a paper's impact and quality. In this thesis we look into the problem of forecasting the number of citations a scientific work will receive. Predicting the citation count of papers is a well-known and well-studied problem for which various approaches have been proposed, including studying the citation network, co-authorship network, and utilizing information after publication, such as abstract, title, venue, and previous citation counts, among others. We propose a novel approach for predicting long-term citation count of papers based on the structure and evolution of temporal citation networks. Experiments show that our unique method exceeds state-of-the-art approaches for predicting the citation counts of papers. Since this method is based solely on temporal networks, it has a wide range of applications in different fields, such as social networks, the World Wide Web, biological networks and so on.

List of Contents

List of Tables

List of Figures

List of Appendices

List of Abbreviations

i.e.	that is
DNN	Deep Neural Networks
GBRT	Gradient Boosting Regression Trees
$RMSE$	Root Mean Square Error

Acknowledgments

I would like to express my deepest gratitude my mentor, assoc. prof. Miklós Kreśz, PhD, and co-mentor, assist. Lászlo Hajdu, for their invaluable assistance, advice, and guidance with my master's thesis.

I also want to thank assoc. prof. Marko Tkalčič, PhD and assist. prof. Mike Burnard, PhD for their time and ongoing collaborations on and off class.

I would like to extend my gratitude to the Faculty of Mathematics, Natural Sciences, and Information Technologies, as well as their outstanding staff, for their unwavering support anytime I needed it. I am thankful to Nina and Uroš for helping me with translation.

Thanks to Rita, Lum, Kaltrina and Ema in for providing me with encouragement and entertaining discussions during this journey.

I would like to express my gratitude to my parents for always being there for me and inspiring me.

Finally, I would want to show my profound gratitude to Eda for constantly motivating, pushing, and helping me to reach my goals, even during the most trying periods.

1 INTRODUCTION

1.1 MOTIVATION

Many industries, including chemistry, physics, biology, health, business, finance, and social media, have realized the importance of *networks* [3, 18, 42]. In a broad sense, a *network* is a set of *nodes* joined by *links*, which represent relationships between nodes [58]. As a result, networks may be used to simulate complex relationships within a system, such as chemical compound interactions, protein interaction networks in biology, consumer behavior networks and economic networks in business, and social networks. Our primary focus is on citation networks and predicting citation counts for articles by invoking a network based approach.

Citations of scientific articles have become perhaps the most commonly used metric of the scientific influence of an article [11, 14, 17]. A standard argument is that the number of citations for a highly cited paper reflects its influence and contributions to the advancement of scientific knowledge. Large numbers of publication databases store information about authors, titles of articles, publishing venues, and publication year. This data can be used to create a directed network based on whether an article cites the other, which are called *citation networks*.

In academia, determining the impact of a paper is an important task. Despite its widespread use, citations can only assess the current and past scientific impact of papers, whereas in many cases, people prefer to go beyond that to forecast future scientific impact. As a result, we require not only the current citations of a paper, but also a prediction of its future citations, which can recognize its future scientific impact [1, 32, 38, 44, 56].

Price [44] showed that the number of citations that a paper receives in a network or the number of links to a paper had a heavy-tailed distribution which follows a power law. This led to the definition of *scale-free networks*, which are networks whose degree distribution sequence follow a power law [6]. The presence of highly connected nodes in a network causes the degree distribution to have a long tail, indicating the presence of nodes with significantly higher degrees than the majority of other nodes.

Interest in such networks arose by the end of the last century, when Barabási and Albert [5] investigated the structure of a number of large networks, including the Internet and a scientific coauthorship network, and concluded that they follow a scale-free power-law distribution. This was discovered to be the result of two generic processes:

1. networks develop over time by adding new vertices continuously, and

2. new vertices attach to nodes that are already highly connected (also known as *preferential attachment*).

These two processes are common features of real-world networks such as citation networks, the Internet, metabolic networks, telephone call networks, social networks etc.

The problem of predicting citation counts for articles has numerous applications in several disciplines, due to the increasing number of published articles. Therefore, researchers must identify the most influential applications ahead of time in order to plan their future research work, and so various prediction techniques have been discussed in [1,10,11,16,32,33,38,56]. Since there are so many distinct types of citation patterns, projecting citation counts is a challenging feat. Some publications go unnoticed for years before gaining a lot of attention, while others lose their relevance and progressively lose citations.

In [1], Abrishami and Aliakbary present a novel method for predicting an article's long-term citations based on the amount of citations received in the first few years after publication. They use an artificial neural network to train a citation count prediction model, and their model outperforms other methods with respect to prediction accuracy in both yearly and total citation prediction.

Weihs and Etzioni's research work [56] is another noteworthy article because of the vast amount of data used and the impressive results. They concentrate on features that may be extracted from the citation graph, coauthor graph, and paper metadata such as authors and venue. Their implementation is based on the assumption that citation counts are modeled using preferential attachments, and they performed extensive feature engineering followed by supervised learning with a regression model. The major takeaway from this article is that long-term forecasting, and not only short-term forecasting, is possible.

In this Master's thesis we consider the data set used by Weihs and Etzioni in [56]. We propose a novel feature extraction technique using properties from temporal citation networks. Two different models based on Deep Neural Networks and Gradient Boosting Decision Trees are implemented to predict citation counts up to ten years in advance by utilizing the extracted features. We reproduce the models described by Abrishami and Aliakbary [1] and Weihs and Etzioni in [56], and we apply various comparison methods to measure the performance of each model. We then show that our novel model significantly outperforms both models for the first five years. In addition, we employ the network-based features in conjunction with the Weihs and Etzioni [56] properties in a joint model to examine how it compares to the others. The latter model outperforms all the other models for the whole timespan. We conclude that features extracted from temporal networks provide more insights to better predict the target variable and thus yield a greater feature importance than features employed by Weihs and Etzioni.

Since we focus only on the structure of temporal citation networks, this innovative approach is applicable for making predictions in various networks, such as the social networks, World Wide Web, protein-protein interaction networks, airline networks, interbank payment networks, etc.

1.2 STRUCTURE OF THE THESIS

In Chapter 2, we introduce the preliminary theory, which includes the necessary definitions for networks and machine learning that are mentioned in this work. In particular, we present citation networks, scale-free networks, Deep Neural Networks and Gradient Boosted Decision Trees.

This thesis primarily focuses on predicting citation counts so we dedicate Chapter 3 to become acquainted with the concept of citation networks and related work on predicting scientific impact. We provide an overview of the most relevant papers on this topic and categorize previous work into three categories.

In Chapter 4 we provide a comprehensive summary of the research methods used in this thesis. We begin by explaining the origin and other details of the data set, such as network statistics, as well as providing some key information about the journals conferences, most cited papers and our novel feature engineering methodology. We conclude this chapter by describing the implemented models and the selection procedure.

We summarize the models in Chapter 5, where we present the results of our experiments and compare the models. The results are interpreted in detail and visualized via different measurement criteria. We perform other types of analyses on randomly chosen data points and we show the relative importance of features extracted from the citation network.

Finally, we conclude the thesis with an overview of all of the findings and we give a discussion about further research work.

2 PRELIMINARIES

This chapter describes the preliminary theory needed throughout this thesis. We begin with the theory behind networks and graphs [3, 42, 58], then continue with the theory behind machine learning [24, 41].

2.1 NETWORKS AND GRAPHS

The birth of graph theory is often associated with the problem of the bridges of Königsberg [58]. Prussia used to have a city named Königsberg (now Kaliningrad, Russia) which was located on the Pregel river. The city occupied two islands and both areas of land on each side of the river. The islands and both parts of the lands were connected by bridges. The problem arose when the citizens began to wonder whether they could leave home, cross every bridge exactly once, and return home. The graphical representation of this problem can be seen in Figure 1 where on the right side we represent the land areas with dots and the bridges with curves.

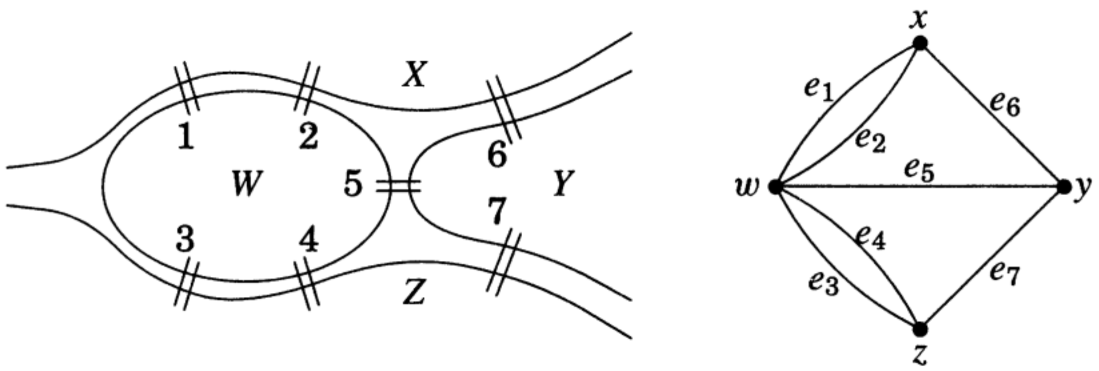


Figure 1: Königsberg's bridges.

Euler observed that if there is a path going through all the bridges, but never crossing the same bridge twice, then all points must have an even number of curves since we enter and leave each node. Based on this observation, Euler concluded that the tour desired by the citizens was not possible. This was the first case where graphs were introduced as formal abstract models. Furthermore, Euler generalized this result to detect whether any graph has a path going through all the dots and visiting each link exactly once. In his honor, this type of path was then called an *Eulerian tour* and he proved that a graph has an Eulerian tour if and only if all the dots have an even number of curves. We will now introduce graphs formally.

A *network* (or a *graph*) is a collection of elements called *nodes* (or *vertices*) and the direct interaction between them is called *links* (or *edges*). Formally, a *network* G is a pair $(V(G), E(G))$ where $V(G)$ is the *node* set and

$$E(G) = \{\{v, u\} : v, u \in V(G)\} \subseteq V(G) \times V(G)$$

is the *link* set.

This representation of the node-link structure allows us to use it for a variety of different systems that need network representation, as shown in Figure 2.

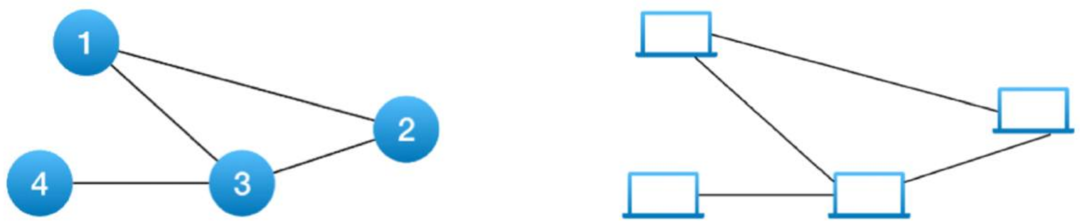


Figure 2: Different networks, same graph structure.

Let G be a network with vertex set $V(G)$ and edge set $E(G)$. The *size* of a network G is represented by $|V(G)|$, which denotes the number of nodes in the network. The number of interactions between nodes is represented by the total number of links in the network, denoted with $|E(G)|$. In Figure 2 we see that the size of both networks is 4, and the total number of interactions is 4.

If $E(G)$ is a multiset and its elements are also multisets then the graph G can have *loops* (edges joining vertices to themselves) and *multiple edges* (more than one edge between a pair of vertices). A graph without loops or multiple edges is called a *simple graph.*

Assigning an orientation to each link e of $E(G)$ we obtain a *directed network* (*directed graph* or *digraph*). Then $E(G)$ consists of ordered pairs (v, u) for $v, u \in V(G)$ which are called *directed links.* Here v is called the *source* or *initial* node and u is called the *target* or *terminal* node. If there exists a link (v, u) in G, then we say that v is a *predecessor* of u and u is a *successor* of v.

Throughout this thesis we will only consider simple graphs and digraphs with at most one edge in each direction for all nodes. If we do not specify if the graph is directed, we assume that it is undirected.

Let G be a graph with node set $V(G)$ and link set $E(G)$. Two nodes v, u in G are *adjacent* if $\{v, u\}$ is a link in G and the link is said to be *incident* to v and u. We also say that v and u are *neighbours*. The *degree* of a node v, denoted $\deg(v)$, is the number of its neighbours. We can get the total number of links $|E(G)|$ expressed as the sum of the node degrees:

$$|E(G)| = \frac{1}{2} \sum_{v \in V(G)} \deg(v).$$

We count each link twice therefore we divide the total sum by 2.

In a directed network we have two types of node degrees: the *incoming degree* and the *outgoing degree*. The *incoming degree* or *in-degree* of node $v \in G$, denoted $\operatorname{indeg}(v)$, is the number of links for which v is the target node. The *outgoing degree* or *out-degree* of node $v \in G$, denoted $\operatorname{outdeg}(v)$, is the number of links that for which v is the source node. Removing directions from the network we get the *underlying network*. In a directed network $G = (V, E)$, the degree of node v is given by

$$\deg(v) = \operatorname{indeg}(v) + \operatorname{outdeg}(v),$$

and the total number of links is

$$|E| = \sum_{v \in V(G)} \operatorname{indeg}(v) = \sum_{v \in V(G)} \operatorname{outdeg}(v),$$

and here we do not divide by 2 since we count the in-degrees and out-degrees separately.

A network is often represented through its *adjacency matrix*. The adjacency matrix A of an undirected network G with $|V(G)|$ nodes has $|V(G)|$ rows and $|V(G)|$ columns, where $A_{ij} = 1$ if nodes i and j are adjacent, otherwise $A_{ij} = 0$. The adjacency matrix of an undirected network is symmetric since there is a link between node i and node j if and only if there is a link between node j and node i, hence $A^T = A$. The adjacency matrix of a directed network has the same structure except that entry $A_{ij} = 1$ if node j is a terminal point of node i, and $A_{ij} = 0$ if node i and j are not adjacent.

Example 2.1. In Figure 3, we have one directed network and one undirected network. We will find the adjacency matrices of these networks.

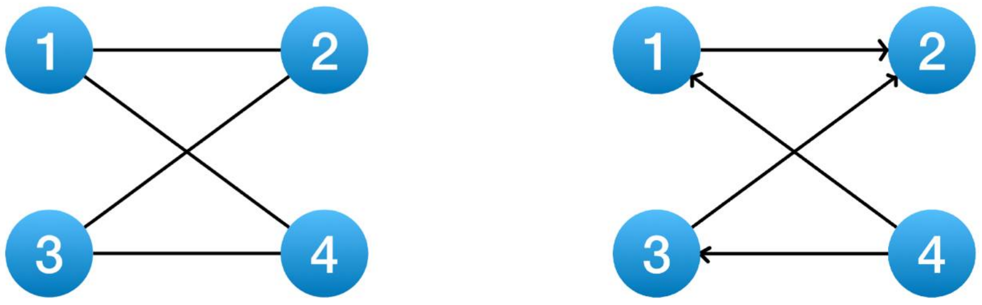


Figure 3: An undirected and a directed network.

The adjacency matrices for the undirected and directed network are:

$$A^{\text{undirected}} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad A^{\text{directed}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Let G be a network with node set $V(G)$ and link set $E(G)$, such that each node is adjacent to all the other nodes. Let $|V(G)| = n$. This type of network is called a *complete network* of size n and we denote it by K_n. The total number of links in this network is given by

$$|E(G)| = \frac{n(n-1)}{2}.$$

Often, real networks have a much smaller number of links. A network G of size $|V(G)|$ is *sparse* if $|E(G)| \lll |E(K_n)|$. The number of links in a sparse network is proportional to the number of nodes, and using the *little o* notation to indicate how fast the number of links grows in comparison to the number of nodes, we can write

$$|E(G)| = o(|V(G)|).$$

Let G be a directed network with node set $V(G)$ and link set $E(G)$, such that each node is adjacent to all the other nodes. Let $|V(G)| = n$. This type of directed network is called *complete directed network* of size n and we denote it by $\vec{K_n}$. The total number of links in G is given by

$$|E(G)| = n(n-1).$$

A sequence $v_1, v_2, \ldots, v_k$ of nodes of a network G is called a *walk* in G if v_i and v_{i+1} are adjacent for all $1 \leq i \leq k-1$. A *path* is a walk in which no node is repeated. A *cycle* is a path that starts and ends at the same node. A directed network is said to be *strongly connected* if for each pair of nodes u and v, there is a path from u to v. In the undirected case, we simply say that the network is *connected.* In a directed network, we denote the length of the shortest directed path from u to v by $d(u, v)$. For undirected networks, $d(u, v) = d(v, u)$ for any two nodes u, v and we call $d(u, v)$ the *distance* between node u and v.

2.1.1 Real world networks

Real world networks can be classified into several categories, namely social networks, information networks, technological networks, biological networks, etc. *Citation networks* are one of the most famous examples of *information networks.*

The *degree distribution* p_k gives the probability that a randomly selected node in the network has degree k. For a network with n nodes, the degree distribution is given by

$$p_k = \frac{n_k}{n},$$

where n_k denotes the number of nodes with degree k. The *degree sequence* of a network is the non-increasing sequence of its node degrees. Following the discovery of *scale-free networks*, the degree distribution has taken on a central role in network theory. A *scale-free network* is a network whose degree distribution follows a *power law.*

The distribution of a random variable x follows a *power law* if its probability distribution satisfies $P(x) \propto x^{-\alpha}$, where α is the characterizing *scaling exponent* and typically is bounded by $2 < \alpha < 3$. The number of nodes with extremely large numbers of links is substantially higher in the power-law distribution than in the normal distribution. In a normal distribution, however, well-connected nodes are more common.

It was observed that the distribution of numbers of papers written by scientists follows a power of law. In [44] Price, among other results, emphasises that both the in-degree and the out-degree distribution of the citation networks follow power laws.

One of the other aims of network science is to create models that replicate the properties of real networks. Most real networks lack the regularity (obedience to a set of principles) that we want, but they seem to have a dose of randomness. To mimic the complexity of real systems, network theory builds and characterizes networks that are truly random via placing links between the nodes randomly.

Let G be a network with n nodes, such that any two nodes are adjacent independently with probability p, where p is predefined. We call G a *random network* and denote it by $G(n, p)$. Pál Erdös and Alfréd Rényi are two mathematicians that have played a key role in the interpretation of random network properties. In their honor, a random network is called the *Erdös-Rényi network*, where edge probabilities are independent [19,20]. The degree distribution sequence of Erdös-Rényi networks follows the *Poisson distribution.* The distribution of a random variable x follows a Poisson distribution, with parameter $\lambda > 0$, if its probability distribution satisfies $P(x) \propto \frac{\lambda^k e^{-k}}{k!}$.

One interesting phenomenon that been observed in real and random networks is the *small-world phenomenon* [55]. The small-world effect follows from the well-known *six degrees of separation* phenomenon, which originated from early works of Karinthy [31] and the Small-world experiment conducted by Milgram [39]. Karinthy mentions that "it's always easier to find someone who knows a famous or popular figure than some runthe-mill, insignificant person". The small-world phenomenon extends this theory and implies that any two randomly chosen nodes in a network are at distance at most six. Small-world networks have no official definition, but some common characteristics include a high *clustering coefficient* (which is the measure of the degree to which nodes in a network tend to group together), link sparsity and small distances between any two randomly chosen nodes.

The Zachary karate club network is a very famous social network example described and used for the first time in [61]. This network has 34 members of karate club, and it documents links between members who interact outside the club. During the research, a disagreement emerged between the administrator John and the instructor Jack, resulting in the club being split in half. Half of the members created a new club around Jack, while the other half either found a new instructor or stopped doing karate altogether. Zachary successfully placed all but one member of the club to the clubs they really joined after the split based on the data he gathered.

Let us use the Zachary karate club network as a real-world example and the Erdös-Rényi network ($G(34, 0.14)$) as a randomly generated example. This example reflects the difference between a scale-free network and an Erdös-Rényi random network, which has the same size and expected number of links as the Zachary network. In Figure 4 we can see both networks.

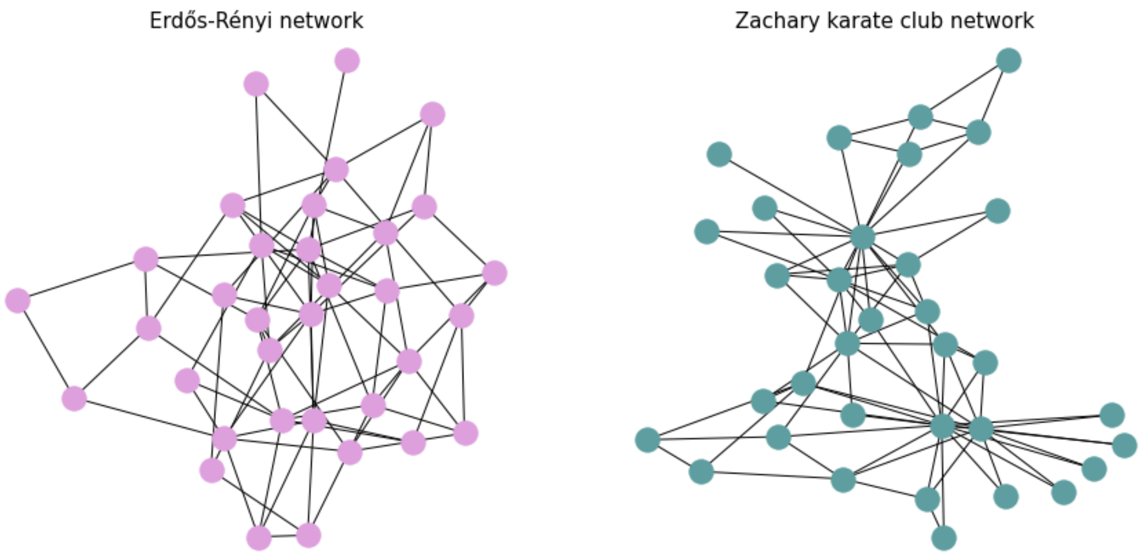


Figure 4: Zachary karate club and $G(34, 0.14)$ networks.

The parameters $n = 34$ and $p = 0.14$ for generating the Erdös-Rényi network where chosen in such a way that we have the same number of nodes and links as in the Zachary karate club network. The total number of links in a complete network is $\frac{n(n-1)}{2}$ so since we have 78 links in the Zachary karate club network then probability is $p = \frac{78}{\frac{34(34-1)}{2}}$.

As described above real world networks have a power law degree distribution sequence and randomly generated graphs have a Poisson degree distribution sequence which we can also see that this holds for these two networks in Figure 5. Even though the graphs are small with only 34 nodes and 78 links we can clearly see the degree distribution.

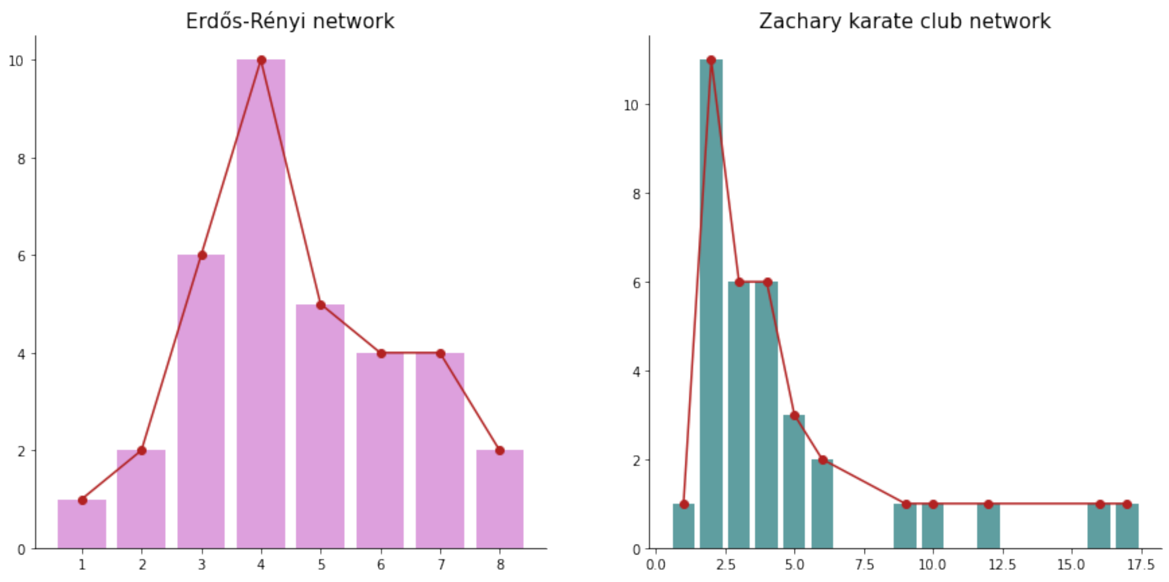


Figure 5: Zachary karate club and $G(34, 0.14)$ degree distributions.

The clustering coefficient for the Zachary karate club network is 0.57 and for $G(34, 0.14)$ is 0.08 which is expected since scale-free networks share the same properties as small-world networks, in this case a high clustering coefficient number. The average shortest path for both networks is relatively close to one another since for the Zachary karate club network is 2.41 and for the $G(43, 0.14)$ network is 2.38. Checking the same properties for bigger networks that have more nodes and links then we would see bigger differences.

2.2 MACHINE LEARNING

Machine learning is the study of computer algorithms that improve automatically through experience and by the use of data or information. Machine learning entails translating experience into expertise or knowledge in order to extract information or knowledge from data. It is also referred to as statistical learning or predictive analytics. This field is the intersection of statistics, artificial intelligence, mathematics, and computer science.

Machine learning methods are used widely today from applications like recommendation of products to buy or movies to watch, to automated self-driving cars and many more. Giant tech companies like Tesla, Facebook, Netflix, Amazon or Google rely heavily on the use of machine learning models but also spend a huge amount of their capital in research and development of new methods for machine learning. A learning algorithm's input is *training data*, which represents experience, and its output is some *expertise*, which typically takes the form of another computer program that can perform some task. We also measure the success of the learning algorithm as the probability that the algorithm does not predict the correct output on a random instance drawn from the data set. For a formal mathematical treatment of machine learning see [51].

Before machine learning advancements, "intelligent" applications were systems that required hand-coded rules of "if" and "else" statements to process data. This approach was not time efficient and required a lot of people to work on a single model, and a slight change in the task might require to rewrite the whole system. An example where this approach may not be feasible is the construction of a set of rules to detect a face in an image since every face is unique.

Generalizations from previously known examples are the most used kinds of machine learning algorithms. We distinguish four types of machine learning algorithms according to the amount and type of supervision we offer during training: *supervised learning*, *unsupervised learning*, *semisupervised learning* and *reinforcement learning*. The former three are *passive* learning techniques, while the latter is an *active* learning technique. The phrase *active* learning refers to a learning system in which the learner plays a part in deciding what data will be utilized to train it. In contrast, *passive* learning involves the learner being presented with a training set over which it has no influence.

In supervised learning, we feed the training set and the desired output to the algorithm. *Classification* problems are a typical supervised learning task. A *classification* model tries to deduce something from observed data, by predicting the value of one or more outputs given one or more inputs. Let us consider an example.

Example 2.2. Suppose that we want to build a system that know how to differentiate a spam email from legit ones.

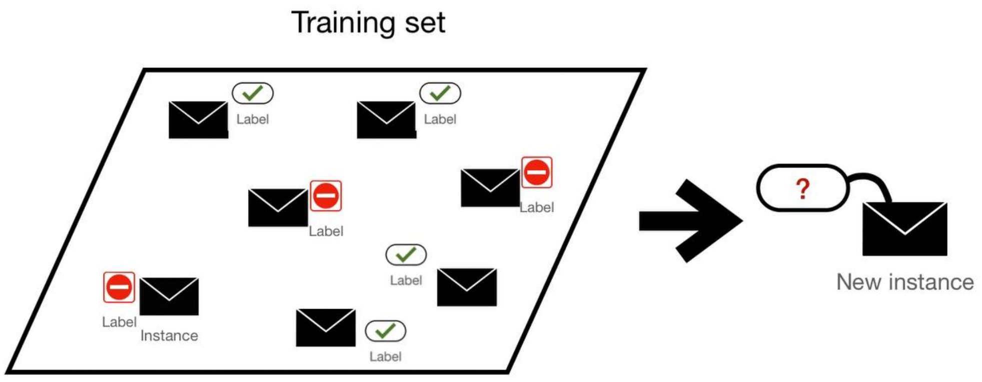


Figure 6: Spam classification problem.

The model is trained with example emails along with their *class* (spam or good), and the goal is to learn to classify new emails. Besides classifying new instances, another task is to predict a *target*, for example the temperature for the following days, given a set of *features* (temperature of the previous days, month, season, etc.) called *predictors.* Tasks like this are called *regression* problems.

Unsupervised learning does not have labeled training data, so the system tries to learn by itself. Unsupervised learning algorithms are used for *clustering*, *anomaly detection* and *novelty detection*, *dimensionality reduction*, and *association rule learning. Clustering* aims to automatically group data in such a way that items in a group are more *similar* to each other than items outside the group. The term *similarity* refers to how similar two data objects are, for example, in terms of distance (Euclidean, Manhattan, Minkowski) or other types of measures such as Cosine Similarity or Jaccard Similarity (for a detailed description of these measures see [34,51]). On the other hand, *dissimilarity* tells us how different two data items are. We illustrate these notions in Example 2.3.

Example 2.3. Suppose that you have a group of friends and you want to see which of them have similar hobbies. We run a clustering algorithm to detect similar groups.

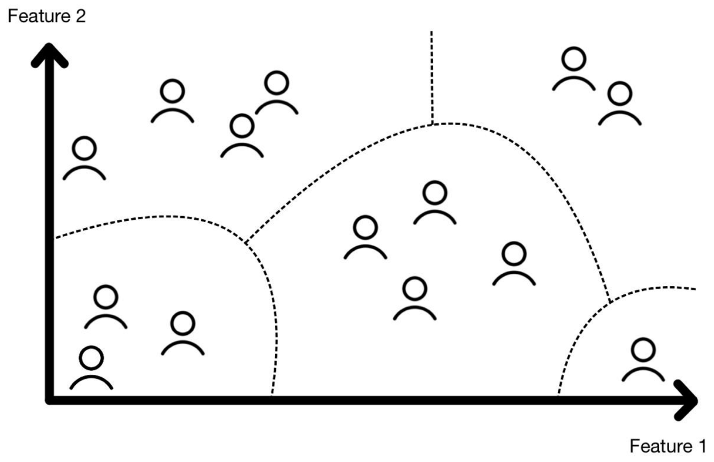


Figure 7: Clustering.

Anomaly detection is a data mining process that detects data points, events, or observations that differ from the expected behavior of a data set. *Novelty detection* identifies novel or unusual data from the data set. *Dimensionality reduction* aims to transform data from a high-dimensional space into a low-dimensional one, in such a way that it preserves some relevant properties of the original data. *Association rule learning* searches for associations among data points by examining whether one data item is dependent on another. Labeling data is often very costly and time-consuming, and we typically have a few labeled instances and a lot more unlabeled instances. This is what *semisupervised learning* algorithms deals with.

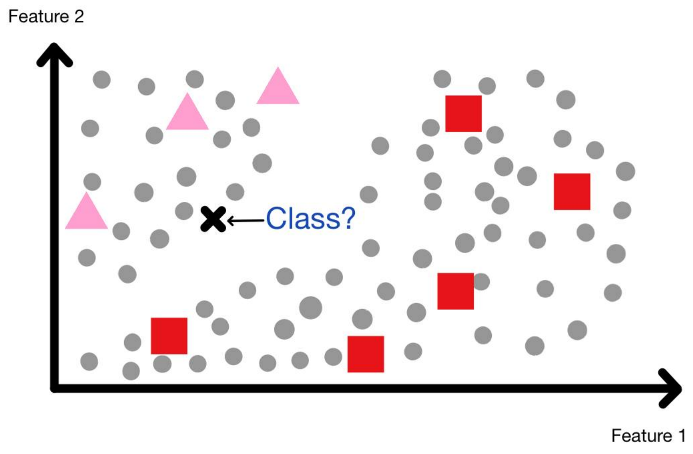


Figure 8: Semisupervised learning with two classes.

Reinforcement learning is the study of how intelligent agents should operate in a given environment to maximize the concept of cumulative reward. More specifically, in this scenario, the learning system, referred to as an *agent*, can observe the environment, select and conduct actions, and receive *rewards* (or *penalties* in the form of negative rewards). It must then learn on its own what is the best strategy, known as a *policy*, to maximize reward over time.

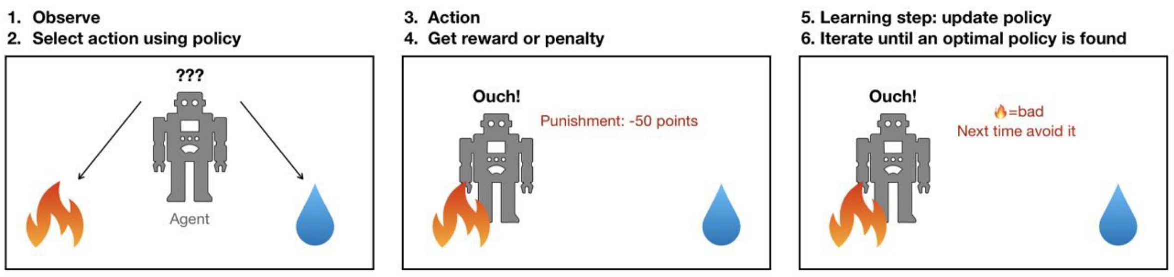


Figure 9: Reinforcement learning.

In this thesis we will focus on supervised learning techniques. There is a wide variety of supervised learning algorithms, each with its strengths and weaknesses. There can not be a single learning algorithm that performs best on all supervised learning problems (for a proof see [59]). In our model we take decision based and neural network based approaches which we describe in the next two sections.

2.2.1 Gradient Boosted Decision Trees

One of the most used models for classification and regression tasks are *decision trees*. A *decision tree* learns a hierarchy of if/else questions, leading to a decision.

Example 2.4. Let us build a model that can distinguish between types of transportation vehicles.

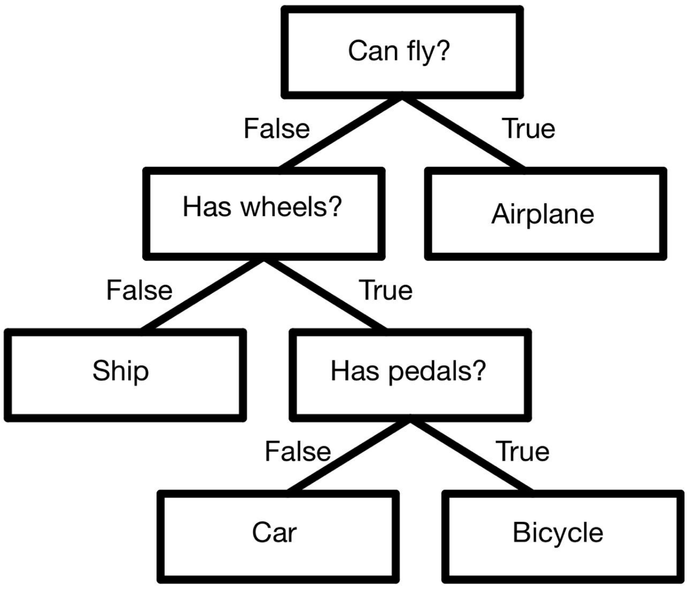


Figure 10: A decision tree to distinguish types of transportation vehicles.

Here each node represents a question or a terminal node that contains the answer. In machine learning, this models is built to distinguish between four classes of transportation vehicles (bicycle, car, ship, and airplane) using these features: “can fly”, “has wheels”, “has pedals”. Instead of checking everything by hand, we can learn them from the data using supervised learning.

Some of the most famous supervised learning approaches train a group of regressors or classifiers, each on a different random subset of the training set. The class that takes the most “votes” from the obtained predictions of the individual trees is the output of this predictor. Most of the time this approach of aggregating the prediction from a group of predictors yields a better result than using a single predictor. An *ensemble* is a group of predictors, and the technique of using a group of predictors is called *Ensemble learning*. *Ensemble methods* are algorithms from Ensemble Learning. Some famous Ensemble methods such are: *bagging*, *boosting*, *stacking*, *Gradient Boosted Decision Trees*, *Random Forest* etc. The last two methods are a group of Decision Tree regressors or classifiers.

The combination of several *weak learners* (simple models such as shallow trees) into a *strong learner* is an Ensemble method called *boosting*. The main idea of most boosting methods is to train predictors sequentially, and each trying to correct its predecessor. Let us now describe *Gradient Boosting* [22], which is a very popular boosting algorithm. The central goal of *Gradient Boosting* is to enhance an imperfect model by adding a new learner. We present the layout in Algorithm 1.

Algorithm 1: Gradient Boost

```
Input: Training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, $L(y_i, F(\vec{x}))$
Output: A function that best approximates the output variable from the
        values of input variables via minimizing the loss function $L(y_i, F(\vec{x}))$
1 $F_0(\vec{x}) = \arg\min_\rho \sum_{i=1}^{n} L(y_i, \rho)$;
2 for $m = 1$ to $M$
3     $r_i = -\left[\frac{\partial L(y_i, F(\vec{x}_i))}{\partial F(\vec{x}_i)}\right]_{F(\vec{x})=F_{m-1}(\vec{x}_i)}, i = 1, \ldots, n$;
4     Fit a weak learner $h_m$ using $\{(x_1, r_1), \ldots, (x_n, r_n)\}$;
5     $\rho_m = \arg\min_\rho \sum_{i=1}^{n} L(r_i, F_{m-1}(x_i) + \rho h_m(x_i))$;
6     $F_m(\vec{x}) = F_{m-1}(\vec{x}) + \rho_m h_m(\vec{x})$
7 return $F_M(\vec{x})$
```

The *cost* or *loss* function $L(y_i, F(\vec{x}))$ is a differentiable function which finds the difference between initial prediction of an algorithm and the ground truth. It is initialized with a model of constant value $F_0(\vec{x})$.

Then we compute the residuals r_i, which can be interpreted as the negative gradient of the predefined loss function. The next learner h_m, which is a decision tree in the Gradient Boosting Decision Tree algorithm, is trained on the residual set. In essence, the learner function h is derived by using a *gradient descent* [51, Section 14] method, which estimates the local minimum of a differentiable function. Next, we compute the weights ρ_m which minimize the loss function, and finally the latest learner $F_m(\vec{x})$ is added to the ensemble model.

Let us consider a simplified example where we use Gradient Boosting for a regression problem (from [30]). Let us try to predict house prices in London, where the data looks like in Table 1.

Table 1: House prices in London data.

House size	Garden size	Garage	True House Price
1000	700	Garage	£1m
770	580	No Garage	£0.75m
660	200	Garage	£0.72m

We make an initial prediction for each of the house prices based on an initial model f_0. Initial models are often very simple, predicting the mean of the target variable of the training data. An *error* (*residual*) e_1 is defined for each sample as $e_1 = y - f_0$, where y is the true value and f_0 is our initial prediction. In Table 2 we can see the true values, initial prediction and error.

Table 2: Initial prediction and error.

True House Price	Initial Prediction f_0	Error e_1
£1m	£0.82m	£0.18m
£0.75m	£0.82m	-£0.07m
£0.72m	£0.82m	-£0.1m

As we see from Table 2 our initial prediction is not very accurate since it is just the mean. To improve this result we introduce a new model f_1 to predict the *error* e_1 from the sample feature values. In gradient boosted decision trees, this model itself is a decision tree. Now we can predict the error e_1 for each sample using f_1.

Table 3: Predicting error e_1.

True House Price	Initial Prediction f_0	Error e_1	Predicted Error f_1
£1m	£0.82m	£0.18m	£0.17m
£0.75m	£0.82m	-£0.07m	£-0.09m
£0.72m	£0.82m	-£0.1m	£-0.1m

Looking at the first house our f_0 was £0.82m and knowing the true value we have an error e_1 of £0.18m. Then we trained a decision tree f_1 to predict e_1 for each sample. Now we could combine the two models into a new second prediction F_1 by adding the predicted error f_1 to the initial prediction f_0, shown in Table 4.

Table 4: Additive model.

True House Price	Initial Prediction f_0	Predicted Error f_1	Prediction $F_1 = f_0 + f_1$
£1m	£0.82m	£0.17m	£0.99m
£0.75m	£0.82m	£-0.09m	£0.73m
£0.72m	£0.82m	£-0.1m	£0.71m

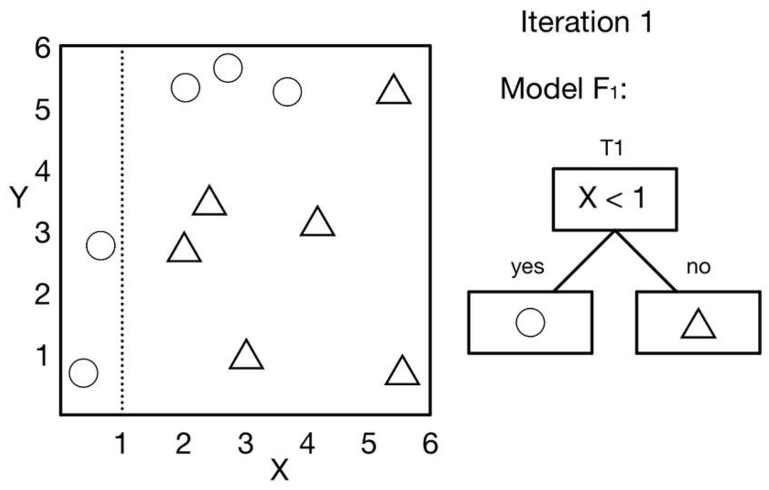
Now that we have a second prediction F_1 we can continue in a sequential manner, calculating again the error of our second prediction e_2 and training a tree f_2 to predict the second error. Again we have $F_2 = F_1 + f_2$, and we continue in this manner. This approach is known as an *additive model*, since we are summing up models. The general formula is given by

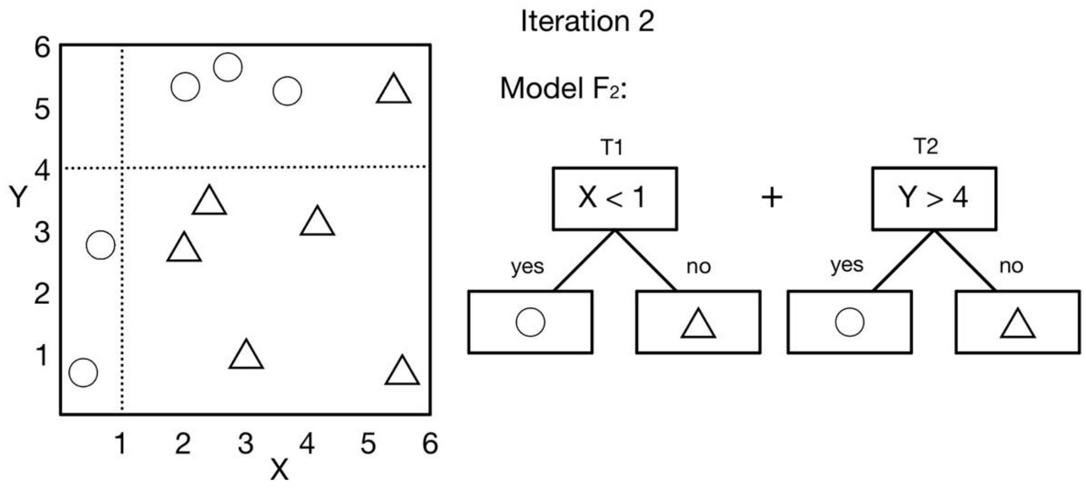
$$F_m = F_{m-1} + f_m,$$

where F_{m-1} is the current prediction F_{m-1} and the prediction of the error $f_m \sim e_m = y - F_{m-1}$.

The number of weak learners is a *hyperparameter* you have to choose. A *hyperparameter* is a parameter of a learning algorithm. We set it prior to training and it remains like that during the training, so it is not affected by the learning algorithm itself. *Tuning hyperparameters* is one of the most important parts of building a Machine Learning system. By tuning hyperparameters we want to find the right parameters that maximize the accuracy (or minimize the error).

Example 2.5. Let us consider an example of visualizing Gradient Boosting trees for a classification problem.





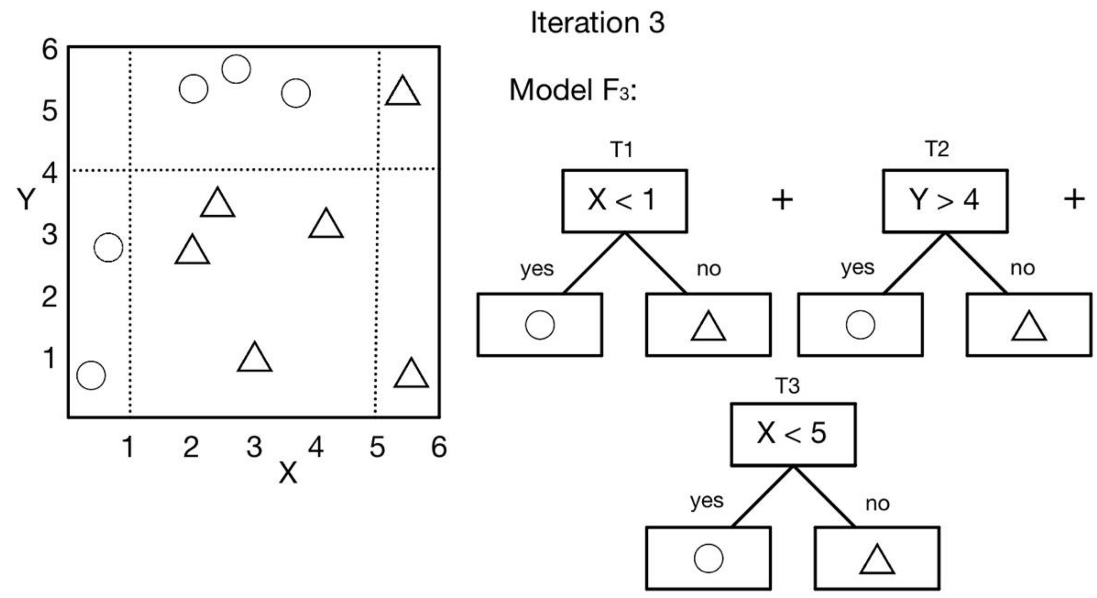


Figure 11: An example of Gradient Boosting Decision Trees

From Figure 11 we see that this algorithm needed only three iterations to learn how to classify the objects in their corresponding classes.

2.2.2 Neural Networks and Deep Learning

Nature has inspired countless innovations and one of the most important ones that we use today in machine learning was inspired by looking at our brain's architecture to build an intelligent machine. The biological network of neurons in our brains were the reason that *artificial neural networks* (ANNs) were invented. A *biological neuron* is an electrically excitable cell. These kind of cells communicate with one-another via specialized connections called *synapses.* It is the main component of nervous tissue in all animals except sponges and placozoa [52]. There are three types of neurons classified by their function. *Inter neurons* help connect neurons to the other neurons within the same region. Stimuli like touch, light, or sound is noticed by *sensory neurons* which then send the signals to the brain or spinal cord. Finally *motor neurons* control everything like muscle contractions, and the signals that they receive come directly from the brain and spinal cord. A group of connected neurons is called a *neural circuit.* Neurons consist of the *soma*, *dendrites* and a *single axon.* Dendrites and the soma help neurons receive signals and the axon forwards the signals out.

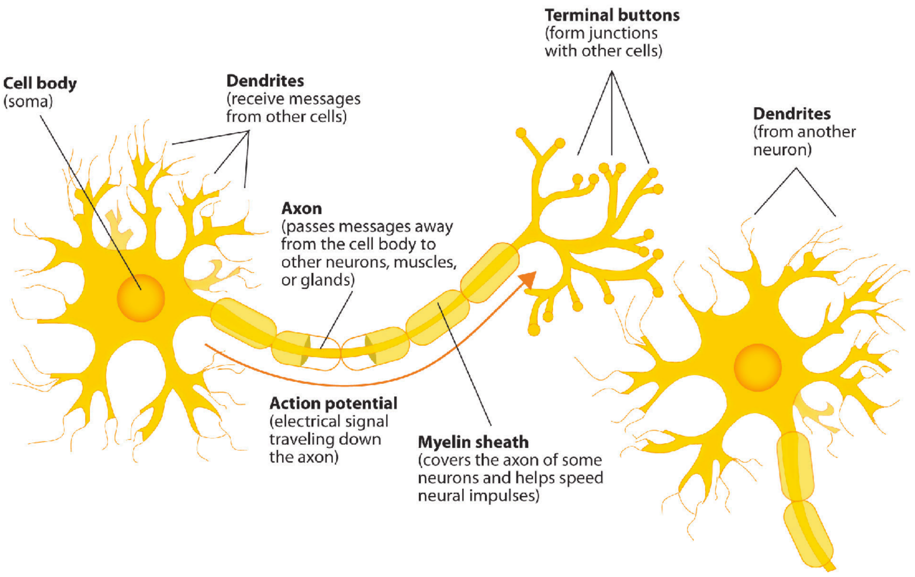

Terminal buttons
(form junctions with other cells)
Cell body
(soma)
Dendrites
(receive messages from other cells)
Dendrites
(from another neuron)
Axon
(passes messages away from the cell body to other neurons, muscles, or glands)
Action potential
(electrical signal traveling down the axon)
Myelin sheath
(covers the axon of some neurons and helps speed neural impulses)

Figure 12: Biological neuron. [53]

Although ANNs were inspired by the biological neurons, they slowly evolved to loosely emmulate them. The first ones to introduce ANNs were the neurophysiologist Warren McCulloch and the mathematician Walter Pitts in 1943 [37]. Their proposed model *artificial neuron* has one or more binary inputs and one binary output. In this paper, McCulloch and Pitts demonstrate with examples how this simple model is capable of solving any logical proposition by building a network of artificial neurons.

Example 2.6. Let us solve some logical computations by building ANNs, assuming that an artificial neuron activates when two or more inputs are active.

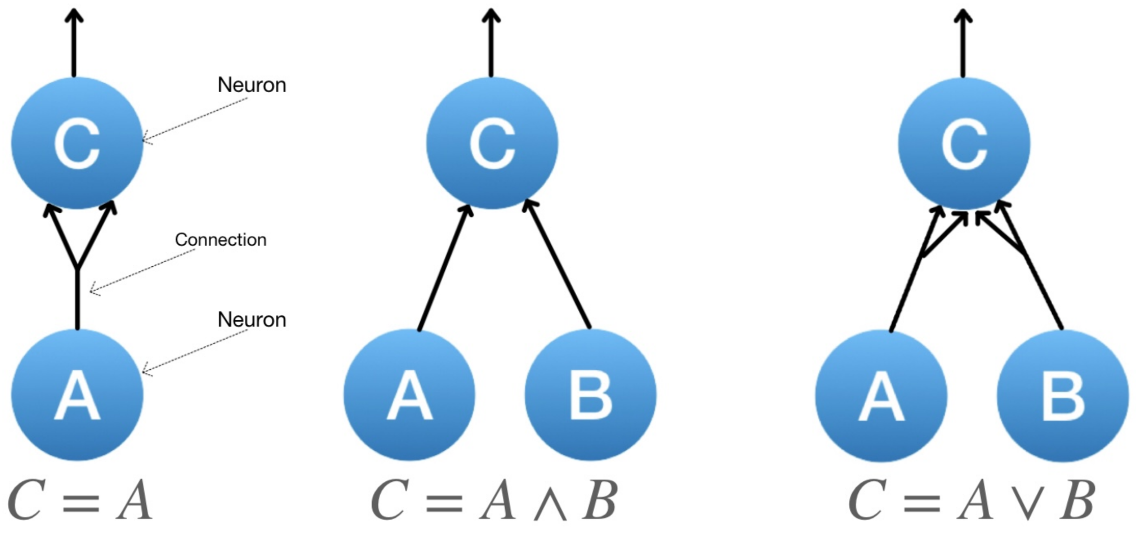


Figure 13: Logical computations with artificial neurons.

In the first network we see that if A is activated then C gets activated as well, but if A is not activated then neither is C. In the second network, we see that C gets activated if and only if both A and B are activated. The last network shows that C is activated when one of A or B are activated.

The *Threshold logic unit* (TLU) is an artificial neuron, also known as the *linear threshold unit* (LTU). The input and output are numbers and each input connection is associated with a weight. The TLU computes a weighted sum of the inputs for the occurring TLU

$$x = w_0 a_1 + w_2 a_2 + \ldots + w_n a_n = \vec{a}^T \vec{w},$$

and after that it applies a *step function* to the weighted sum and outputs the result

$$h_{\vec{w}}(\vec{a}) = step(\vec{a}^T \vec{w}).$$

Similar to Logistic Regression [23] or a linear Support Vector Machine (SVM) [13], TLU computes a linear combination of the inputs, and if the outcome passes a certain threshold, the output of the model is the positive class, otherwise the output is the negative class. Training a TLU means finding the right values for the weights.

A different artificial neuron known as the *Perceptron* was invented by Frank Rosenblatt [46], which was moderately different from TLU. If we have one layer of TLUs, where all the inputs are connected to all the TLUs, then we get the Perceptron. A *dense layer* (or *fully connected layer*) means that all the neurons in a layer are connected to all the neurons in the previous layer. Feeding data to the Precepron is done via the *input neurons* which immediately output the data that is fed without changing anything. Combining all the input neurons we form the *input layer.* After the input layer, we have a different type of neuron called the *bias neuron* which adds 1 all the time. In Figure 14, we have an example of the architecture of a Perceptron that has one input neuron, one bias neuron and two output neurons.

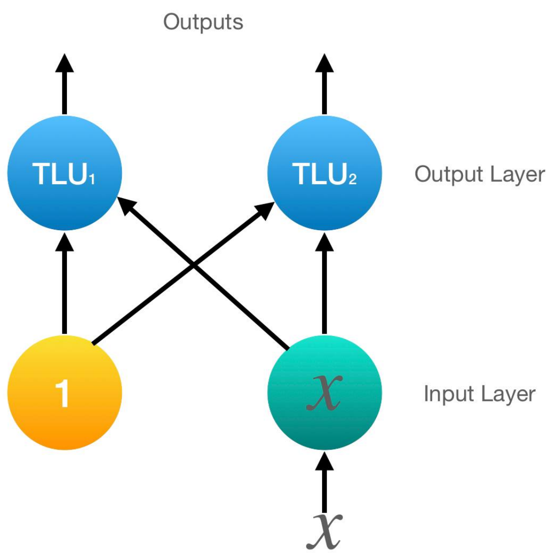


Figure 14: Perceptron with one input neuron, one bias neuron and two output neurons.

The mathematical formula for computing the outcome of a layer for several samples at once is:

$$h_{\vec{W},\vec{b}} = \phi(\vec{X}\vec{W} + \vec{b}).$$

Here $\vec{X}$ is the matrix of the input features (one row per sample and one column per feature). The matrix $\vec{W}$ has all the connection weights from the input neurons (one row per input neuron and one column per artificial neuron in the layer). The bias vector $\vec{b}$ has all the connection weights between the bias neuron and the artificial neurons (one bias term per artificial neuron). The function ϕ is called an *activation function.* When working with TLUs, this function is the same as the step function.

Training this algorithm was inspired by *Hebb's rule* [28] and was proposed by Rosenblatt. Hebb suggested that the bond between two neurons grows stronger when a neuron triggers another neuron often. We train the Perceptron by giving one instance at a time, and for that instance it makes the prediction. If we have a wrong prediction the connection weight is reinforced from the input that would have contributed to the correct prediction. The prediction learning rule is

$$w^1_{i,j} = w^0_{i,j} + \eta(y_j - \hat{y}_j)x_i,$$

where $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron, and the upper indices 0 and 1 stand for the current step and the next step respectively. The current training instance of the i^{th} input value is denoted by x_i. For the j^{th} output neuron for the current training instance $\hat{y}_j$ is the output, and y_j is the target output. The learning rate is denoted by η. They way that the Perceptron learns is very similar to *Stochastic Gradient Descent* [47].

The Perceptron has some limitations , for example, as shown by Minsky and Papert [40] it can not solve the *XOR classification problem.* Given two binary inputs, an

XOR function should return a true value if the two inputs are not equal and a false value if they are equal. A possible solution is stacking multiple Perceptrons, which is called a *Multilayer Perceptron* (MLP). An MLP has one input layer, one or more layers of TLUs, known as *hidden layers*, and the last layer of TLUs is called the *output layer*. *Deep neural networks* are ANNs that have multiple hidden layers, see Figure 15.

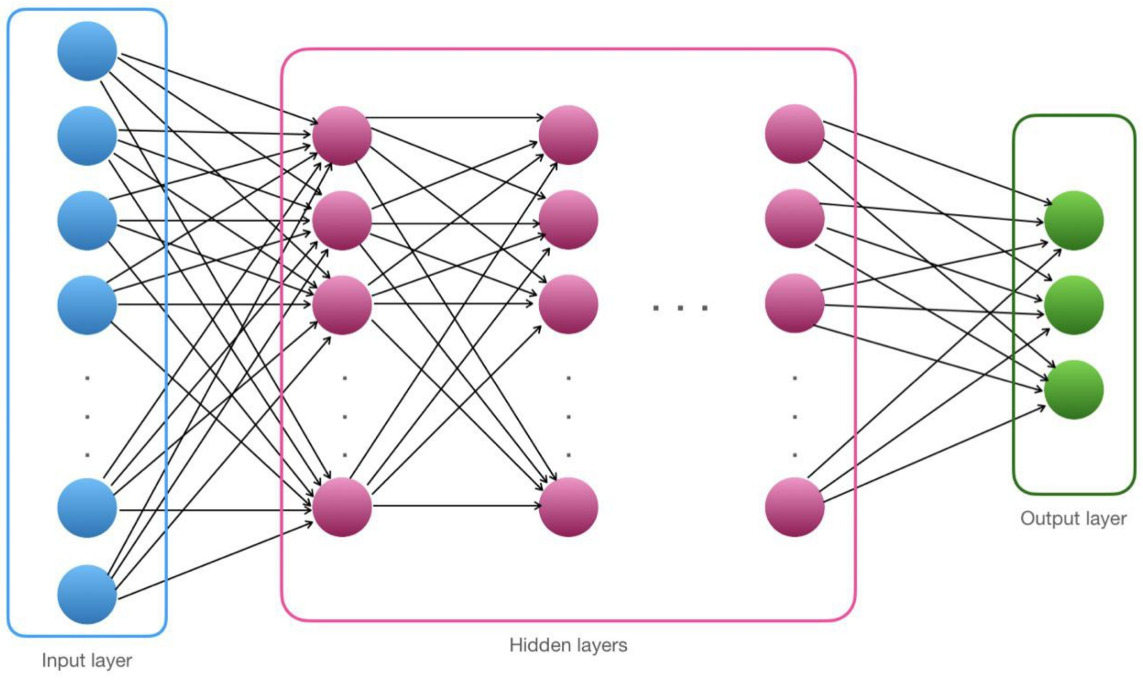


Figure 15: A deep neural network with input, hidden, and output layers.

The *backpropagation* training algorithm was introduced in [48], which we still use today. We feed the training instances to the input layer and for each instance the backpropagation algorithm makes a prediction called *forward pass*, and measures the error. Then it goes in revere through all the layers to measure the error contribution from each connection, called *reverse pass*. At the end it tweaks the connection weights and biases to reduce the error. Let us describe this algorithm in more detail.

1. We can feed one *mini-batch* at a time (a batch contains multiple instances). An *Epoch* is the number of times we go through the full training set. Both of these are hyperparameters.

2. The input layer gets the mini-batch inputs and it passes them to the first hidden layer. All the necessary computations are done in this layer and the results are then passed to the other hidden layer, and this is repeated until the last hidden layer is reached, which is the output layer. We save all the in-between results since we need them for later computation. This step is called the *forward pass*.

3. We will use a *loss function*, to compare the predicted value with the true value, and returns the error.

4. We apply the chain rule for derivatives of composite functions analytically to find the connection weights that caused this error.

5. We repeat the previous step going backward and measuring all the error gradients until we reach the input layer.

6. Gradient Descent is used in the final step of the algorithm to modify all the connection weights using the computed error gradients.

Since the step function in the original implementation of TLUs contains only flat segments, it was substituted with the *sigmoid function*

$$\phi(z) = \frac{1}{1+e^{-z}}.$$

This function has a well-defined nonzero derivative everywhere, thus Gradient Descent can learn in each step. There are also several other famous activation functions that gradually replaced sigmoid function, such as *Rectified Linear Unit function*: $\mathrm{ReLU}(z) = \max(0, z)$ or the *hyperbolic tangent* activation function defined as

$$\tan h(z) = \frac{(e^{z} – e^{-z})}{(e^{z} + e^{-z})}.$$

The number of output neurons depends on the task that we are trying to solve. If we have a binary classification problem, then you only need one output neuron. The predicted value is associated with the positive class, and one minus the predicted value is the negative class. This number can be interpreted as the estimated probability that the event will be of the positive or the negative class. An example when we would have more than one output neuron is when we have a number classification problem such as recognizing whether a picture has one of the numbers from zero to nine. In this case the there would be there would be ten output neurons, one for each of the numbers from zero to nine. The number of output neurons for regression tasks differs on the type of problem that we are trying to solve, if we want to predict a single value for example weather temperature for tomorrow then we need a single output neuron, but if we want to predict the coordinates of an object in the plane then we need two output neurons.

Now that we are familiar with the way neural networks works, we will introduce a different type of a deep learning model. *Recurrent Neural Networks* are very similar to a feedforward network. The main difference is that RNNs also have connections going backwards. The output that is produced in an RNN is then sent back to itself, as in Figure 16. We feed the input $\vec{x}_{(t)}$ and the previous output $\vec{y}_{t-1}$ to the *recurrent neuron.*

If we do not have a previous frame, then we have a vector of 0's or a vector of 1's. We can unroll the network through time and we can clearly see how at time step t, the input neuron gets the input vector and the output vector from the previous time step $t-1$.

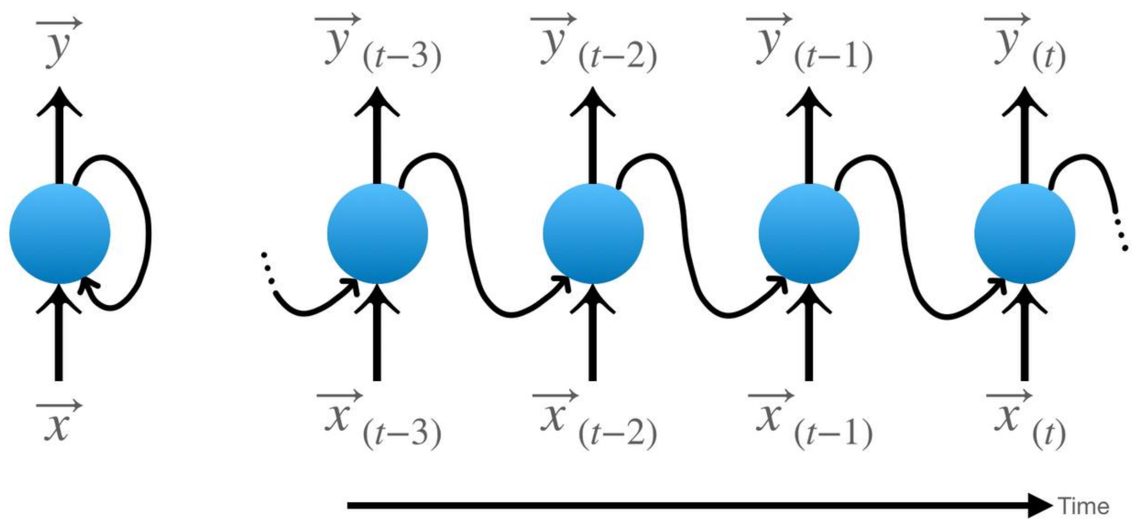

$\overrightarrow{y}$
$\overrightarrow{y}_{(t-3)}$
$\overrightarrow{y}_{(t-2)}$
$\overrightarrow{y}_{(t-1)}$
$\overrightarrow{y}_{(t)}$
$\overrightarrow{x}$
$\overrightarrow{x}_{(t-3)}$
$\overrightarrow{x}_{(t-2)}$
$\overrightarrow{x}_{(t-1)}$
$\overrightarrow{x}_{(t)}$
Time

Figure 16: A recurrent neuron, and the unrolled neuron through time.

Since we have a different architecture, the formula used to make prediction changes. The formula used to calculate the output of a recurrent layer for a single instance is

$$\vec{y}_{(t)} = \phi(\vec{W}_x^\top \vec{x}_{(t)} + \vec{W}_y^\top \vec{y}_{(t-1)} + \vec{b}),$$

where $\vec{W}_i$ for $i \in x, y$ is the weight matrix and $\vec{b}$ is the bias vector. We observe that compared to the formula of a neural network, we have an extra weight matrix. We can again compute the output of a recurrent layer for a whole mini-batch with the following formula

$$\begin{aligned} \vec{Y}_{(t)} &= \phi\left(\vec{X}_{(t)}\vec{W}_x + \vec{Y}_{t-1}\vec{W}_y + \vec{b}\right) \\ &= \phi\left(\left[\vec{X}_{(t)}\vec{Y}_{(t-1)}\right]\vec{W} + \vec{b}\right) \text{ where } \vec{W} = \begin{bmatrix} \vec{W}_x \\ \vec{W}_y \end{bmatrix}. \end{aligned}$$

Similarly as before, $\vec{b}$ is the vector containing bias term for each neuron, $\vec{X}_{(t)}$ is the matrix containing all the inputs for all the instances in the mini-batch, and $\vec{Y}_{(t)}$ is the matrix of the outputs for each instance in the mini-batch at time t. The weight matrices $\vec{W}_x$ and $\vec{W}_y$ contain the weights of the inputs at the current time stamp and the previous time stamp, respectively.

A *memory cell* is a part of a neural network that saves some state across time steps. We denote the state of a cell at time step t by $\vec{h}_{(t)}$ which is a function of an input at time step t and its state at time step $t-1$ is given by $\vec{h}_{(t)} = f(\vec{h}_{(t-1)}, \vec{x}_{(t)})$.

We have four different types of RNNs based on the desired input and output of the model. An RNN that has a sequence as an input and also produces a sequence as an output is called *sequence-to-sequence network* (see Figure 17). This specific type of RNN is useful for predicting time series, such as temperature of the weather. You feed the temperatures of the weather over the past N days as an input, and it outputs the temperature for the following days.

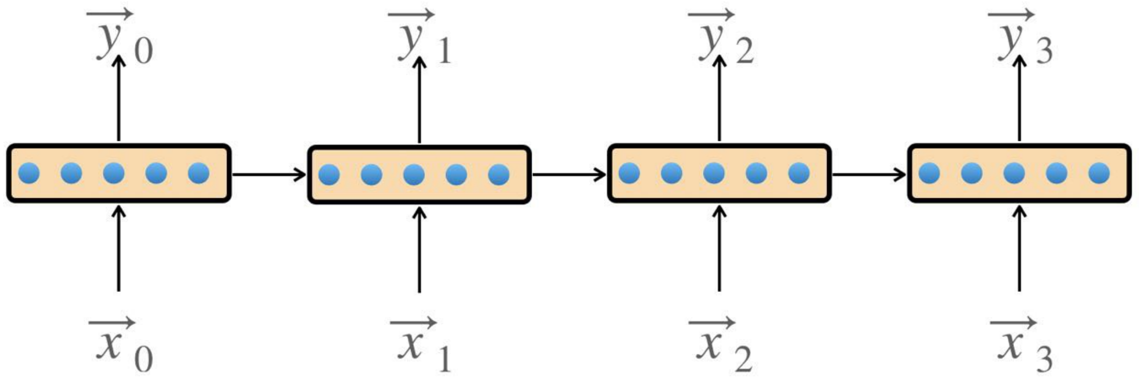


Figure 17: Sequence-to-sequence network.

The second type of RNN is a *sequence-to-vector network* (see Figure 18), where the input is a sequence and for the output, all but the last output is ignored. A typically example of such an RNN can be feeding the sequence of words (lyrics) of a song to a network and the output is a sentiment score.

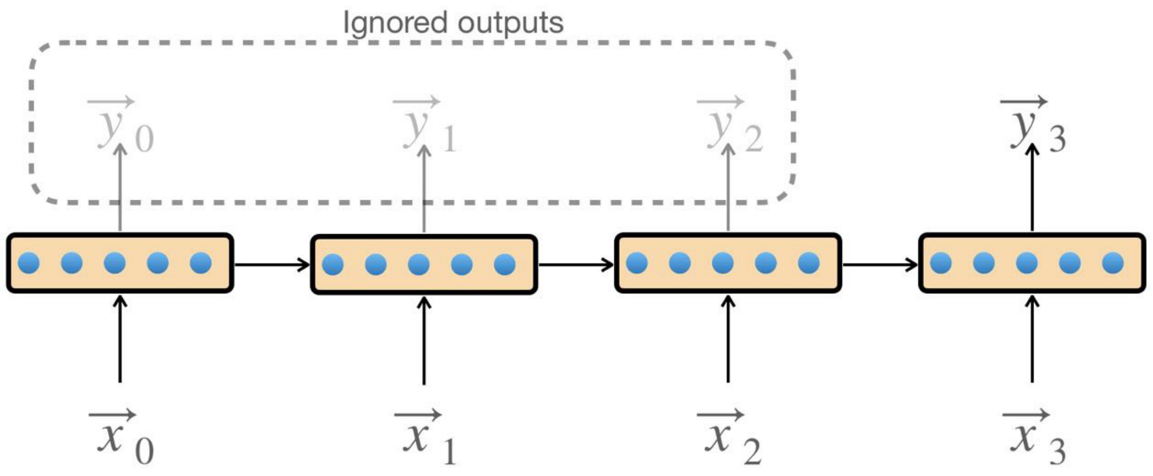


Figure 18: Sequence-to-vector network.

We can also feed a vector and get a sequence as an output. This type is called a *vector-to-sequence network* (see Figure 19). A typical example that used such a network is feeding an image to the input neurons and getting a description for that image as an output.

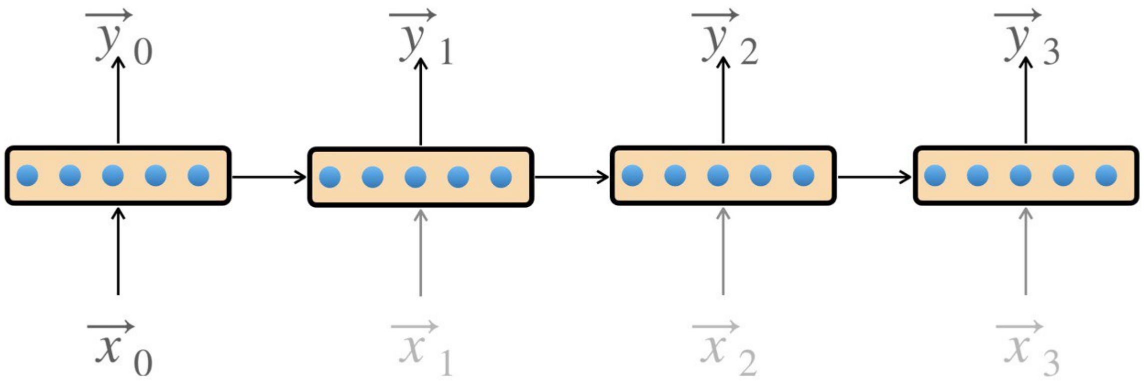


Figure 19: Vector-to-sequence network.

The last type of networks has two components: a sequence-to-vector network called an *encoder*, and a vector-to-sequence network called a *decoder*. A typical example of this type of RNN is to translate sentences from one language to another. We feed the sentence to the encoder which outputs a vector and then it feeds that vector to the decoder which outputs the sentence in another language. This model is known as *Encoder-Decoder* (see Figure 20).

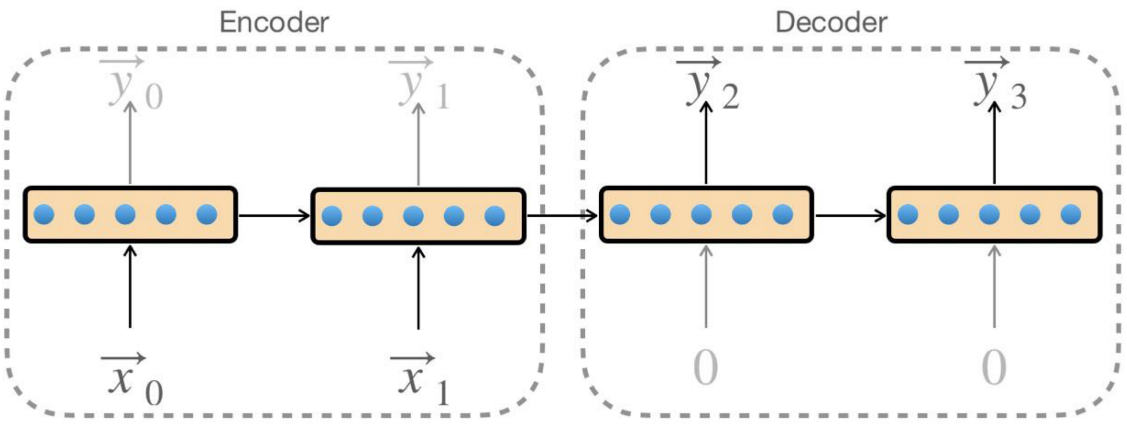


Figure 20: Encoder-Decoder network.

To train an RNN we use a strategy called *backpropagation through time*, which is a regular backpropagation but we first need to unroll the network through time. Then we evaluate the output sequence using a loss function. After this step the gradients of the loss function are propagated backwards, and the model parameters are updated using the gradients computed through the unrolled network through time.

3 PREDICTING SCIENTIFIC SUCCESS

Information networks, also known as knowledge networks, are a very significant type of network. The network of citations between academic works is a famous example of an information network. The majority of publications mention previous works on related topics by others to give credit to the ideas and work of other researchers. Citation networks consist of articles citing previous work that is relevant to their particular field of study. These citations comprise a network, with the nodes being articles, and directed links from A to B indicating that A cites B.

The citation network's structure then mirrors the structure of the information held at its nodes, hence the term *information network*. Citation networks do not contain cycles because papers can only cite previously published papers. As a result, all network edges point backwards in time, making closed paths impossible or highly rare.

Price [44] underlines the structure of networks of scientific papers and it is perhaps the first reported example of a scale-free network. It is also worth mentioning the work of Newman [42] on the structure and function of complex networks, which is the first comprehensive study of the complex networks. He discussed the small-world effect, degree distributions, clustering, random graph models, models of network growth and preferential attachment, and dynamical processes taking place on networks.

Egghe and Rousseau [17] are among the pioneers that utilize quantitative methods on *bibliometrics*, which is the application of statistical approaches to the analysis of books, papers, and other publications. They also introduce various ways for citation analysis.

Predicting the success and impact of scientific works is a well known problem and many studies with different approaches have been conducted. Most of the papers have different objectives such as predicting the h-index (it is the highest value of h for an author or journal that has produced at least h papers, each of which has been referenced at least h times) of researchers [2, 4, 16], predicting the impact factor of scientific journals [32, 45, 60], citation count prediction for scientific papers [11, 15, 16], and predicting highly cited papers [32, 43, 50].

Rocha-e-Silva [45] wanted to find out whether trends of Journal Impact Factor variation can be objectively predicted for the upcoming year. He created citation curves for publications written in the two years preceding the current year, as well as their citations in the current year. For the first and second years, separate curves were formed. He defined a parameter INDEX R and calculated it for a sample of 10 journals chosen at random with Impact Factors ranging from 1 to 3. INDEX R was observed to have a quasi-normal distribution (with a slight skew toward higher values) with a slight adherence to the Gauss distribution (central tendency) and thus he managed to show that by looking at INDEX R, it is possible to forecast the trend for the Impact Factor for the following year.

Cao, Chen and Liu [11] present a simple but efficient and robust data analytic approach for predicting potential citations of papers from various disciplines. The proposed approach can provide a reliable estimate of possible citations using relatively short-term (3 years after the paper is published) citation data, outperforming state-of-the-art prediction methods significantly. This approach is focused on the idea that future citation dynamics are linked to past citation dynamics. The proposed method is called *Gaussian Mixture Model* (GMM) which predicts several possible trends of the future citations of a paper. This method clusters matched citation dynamics into K clusters by fitting a GMM with K Gaussian components, then it predicts the K Gaussian means of the K Gaussian components as the top K possible trends of the paper's future citations.

Newman [43] proposed an alternative measure of impact. He suggests that instead of looking for papers with high total citation counts, we should look for papers with citation counts higher than expected given their date of publication. We can look at this by counting the number of citations a paper has received and compare it to the number of citations received by other articles on the same subject published at the same time. He measures the mean number of citations and its standard deviation for papers published in a window close to a paper's publication date, and also measures the number of standard deviations by which the citation count of that paper differs from the mean. The articles with higher score were assumed to be more important within the field.

In [1] Abrishami and Aliakbary categorize the existing papers based on their information sources for scientific impact prediction. According to their findings, here are three key groups that papers fall under.

1. Papers in the first category use the graph of the scientific papers as the main source of information [14]. Further, some of these papers tackle this problem as a link prediction problem in the citation network [14], some use the co-authorship network [56], and some use the paper neighborhood properties [38].

   Daud et al. [14] address the problem of predicting reciprocal connections in citation networks as a classification problem. Understanding the structure of reciprocal connections allows one to comprehend the underlying community structure of co-authorship and citations, as well as how authors interact, influence, and cite one another. The aim of this study was to use classification models to predict reciprocal links using a variety of textual and graph-based features. The main question that the authors try to answer is: "If you cite an author, will they cite you back?". The results show that features such as paper, author, and field of research are effective for reciprocal link prediction.

McNamara et al. in [38] proposed a new method for predicting a paper's future impact using features of the paper's neighbourhood in the citation network, including measures of interdisciplinarity. Predictors of high impact papers include features such as: high early citation counts of the paper, high citation counts by the paper, citations of and by highly cited papers, and interdisciplinary citations of the paper and of papers that cite it. Their findings indicate that nodes that are highly connected and span network boundaries are more likely to be influential.

2. In the second category we have papers that use the information about other papers just after the publication [16, 56]. This includes the text of the paper, conference, journal, research area and information associated with the authors and references.

   Dong, Johnson and Chawla [16] state that citations are hard to predict due to their heavily-tailed distribution. Because of this they chose to answer two questions: "How will h-index evolve over time?" and "Which of previously or newly published papers of an author will contribute to their own h-index?"

   First, they create a model that predicts potential h-indices for authors based on their current scientific impact. Second, they examine whether newly published or previously published papers cause papers to be more popular, and thus increasing future h-indices of authors. They achieve an R^2-score of 0.92 for predicting the authors' h-index in five years and a 0.99 F_1 score for predicting whether a previously (newly) published paper would lead to this future h-index.

3. In the last category, the information about the papers is extracted after publication [11, 33]. Here authors try to predict the long-term citation count based on the short-term citation count of the papers or by getting extra information about papers in the web and social networks.

   Lamb, Gilbert and Ford [33] measure the association between citation rates and the *Altmetric Attention Score*, which is an indicator of the amount and reach of the attention an article has received. Attention Score was found to be positively correlated with citation rates. They found that the passage of time (years since publication) increased citation rates, with letters and notes receiving fewer citations than traditional articles and review papers receiving more citations than both other article types. However, in recent years, they discovered that increasing media exposure did not correspond to the same number of citations as in previous years, indicating diminishing returns in recent years.

Abrishami and Aliakbary [1] proceed to claim that we do not need many types of features and only the citation count of the first three to five years are enough to have an intuition about how a paper's citation count will perform in the future. According to their categorization, their own article [1] belongs to the third category. They use Recurrent Neural Networks as a tool to make the predictions, which yields good results even with limited information. We shall reproduce their model according to their setup and further compare it with our models. We go into detail about the model in Chapter 4.

A different approach to predicting paper impact was taken by Weihs and Etzioni in [56]. They predict the author impact and paper impact 10 years into the future. The information they use is extracted from co-author networks of papers, citation networks and extra information such as the type of scientific work (a paper or a survey), number of years since the paper was published, venue of the publication etc. Their best performing algorithm is Gradient Boosted Regression Trees (GBRT) which we introduced in Section 2.2.1. Following the categorization by Abrishami and Aliakbary [1], the paper [56] belongs to the first and third category since it uses the graph of the scientific papers but also information of the paper after publication. We used the same data set as Weihs and Etzioni in [56] to develop our own methodology since their large and qualitative data collection was available.

4 METHODOLOGY

To address the problem of predicting citation counts of scientific papers, we propose a novel technique based on feature extraction from temporal citation networks. In other words, we capture snapshots of citation networks evolving through time and extract data that allow us to compute various measurements regarding nodes from each snapshot. In the framework of machine learning approaches, this task is classified as a regression problem because the model should predict the citation count as a non-negative integer value. We create a customized neural network and a Gradient Boosting Regression Tree as majors components of our proposed method since artificial neural networks and ensemble methods are one of the most powerful methods for regression learning. The prediction algorithms are trained using some papers from a data set of existing papers with known citation histories in the training phase, and then the rest of the papers are used as the test set to evaluate the accuracy of the trained neural network and GBRT in the testing phase. We utilize the data set that is made available by Weihs and Etzioni [56], which contains more than four million papers from computer science. Their approach also provides results that are an improvement over previous efforts and thus serves as a baseline against which we measure our progress. Since Abrishami and Aliakbary's work [1] is very innovative and only requires few features for training, we replicate their model and data engineering method. We show that our approach outperforms both mentioned models and this chapter is devoted to a comprehensive discussion of this framework.

We present the machine learning pipeline starting with the *exploratory data analysis* part, where we analyze the data set and summarize the main characteristics via data visualization methods and statistical tables. We then proceed to understand the model pipeline of [56] since we use their model as a baseline against which we compare our models. To replicate and test the Encoder-Decoder model from [1], we first engineer the data set employing their own data cleaning pipeline. Finally, we implement our innovative feature extraction technique and define the optimal hyperparameters for our model to achieve the best possible results. Let us describe the pipeline implementation in detail.

In Section 4.1 we explain the origin, the availability of the data set and the feature engineering process performed in [56]. We describe the data set in detail by giving statistics such as the number of papers, number of authors, number of venues, the years spanned in this data set etc. We also fit an exponential line to see the increase in the number of paper over the years, and we try to deduce expected results from the graph.

Furthermore, we give additional statistical information for the citation network including measures such as the average degree, density, connectivity, number of components etc.

We then visualize the popularity of venues in this data set. This is important because some journals have a greater impact than others, and selecting the appropriate journal to publish research work is critical for reaching the intended audience.

The most cited papers in this data collection are then represented graphically. We summarize all the features used in [56] and their descriptions via Table 7.

In Section 4.2 we explain the feature engineering process for the Encoder-Decoder model and the Network based model. We then give details of the features that we need to extract from the data set in Section 4.2.1. The programming language and the packages used for feature engineering and visualization are described and explained in this section too.

The most important section of this thesis is Section 4.2.2. Here we show our novel feature engineering approach using temporal networks to address the problem of predicting success of papers in the future. The concept of temporal networks is a modeling paradigm that extends the concept of complex networks to provide details about when nodes interact. They can be used to investigate how a network grows, improves, or evolves over time. This shift in time may represent how knowledge travels across any type of network.

We begin this section by illustrating and visualizing the feature extraction approach from the citation network. Afterwards, we present the features extracted from the citation network, as well as additional descriptions and explanations for each of them. We conclude this section with an example of a randomly generated directed network and perform our feature extraction technique on this network.

In Section 4.3 we explain the methods and measurement criteria used in [56], where we choose their best performing model as a baseline model for comparison with our approach. Furthermore, we describe the implementation details of Encoder-Decoder model in Section 4.3.1, which is known as a sequence-to-sequence model. We implement this model using the neural network framework Keras, and include additional model information for replicability purposes.

In Section 4.3.2 we introduce the pipeline of the Network based model, starting with the sizes of the train, test and validation sets, then the DNN and GBRT algorithm hyperparameters for selecting the best performing hyperparameters using the Grid Search algorithm, and then we summarize all the steps of the predictive model. For the sake of replicability, we also include additional implementation specifics and descriptions for both the DNN and the GBRT models.

4.1 DATA SET

The data set that we used is made publicly available[1] in [56], which includes paper published between 1975 and 2016. There are approximately 4 million papers that come from the field of computer science, covering paper published in over 7000 conferences and journals, which are written by around 800, 000 authors. Weihs and Etzioni [56] use information available in 2005 to predict impact in the upcoming 10 year period (2006-2015).

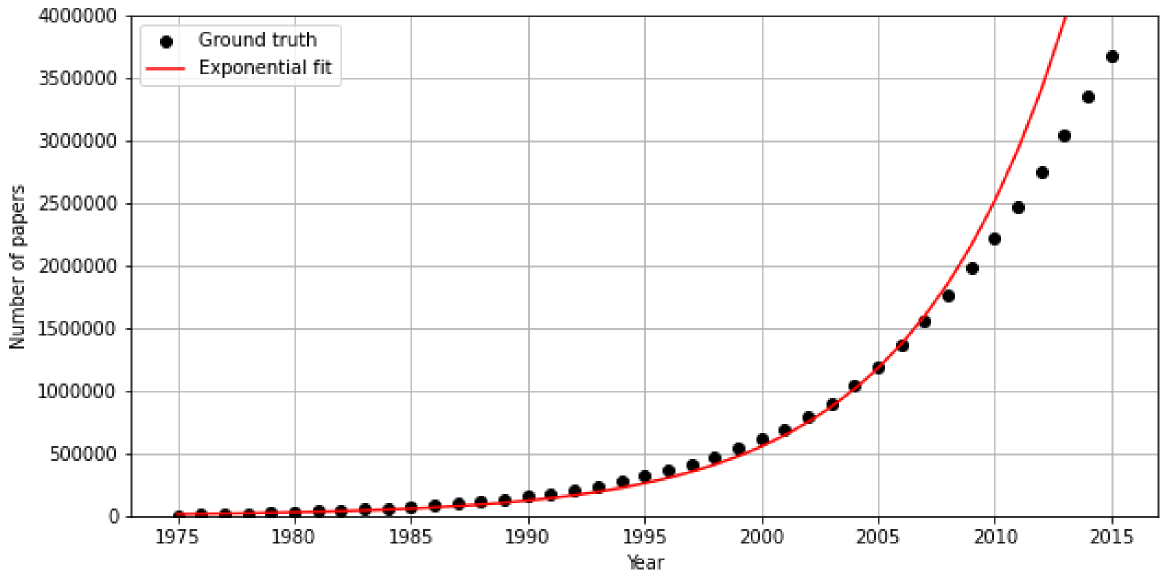


Figure 21: The exponential fit and the cumulative number of published papers in this data set over 1975 – 2015 time period.

From Figure 21, we see how the number of papers increases over time, and the exponential fit suggests that the number of papers is expected to double approximately every six years.

Table 5: Citation network statistics.

Year range	**Nodes**	**Edges**	**Average degree**	**Density**	**Is connected**	**Connected components**	**Largest component**	**Smallest component**
1975	3594	3699	2.058431	0.000573	False	2	3592	2
$\leq$ 1986	82221	126304	3.072305	0.000037	False	4	82213	2
$\leq$ 1996	363938	951772	5.230407	0.000014	False	20	363890	2
$\leq$ 2006	1366613	5827844	8.528887	0.000006	False	109	1366338	2
$\leq$ 2016	3674639	19180176	10.439216	0.000003	False	147	3674281	2

In Table 5, we present some statistics of the citation network (of the underlying

[1]Here you can find the data set together with the code to reproduce the results of [56]: `https://github.com/Lucaweihs/impact-prediction`

network) used in [56]. We observe the network in five time periods: in the first one we include papers published in 1975, in the second one we include papers published from 1975 to 1986, the third one includes papers published from 1975 to 1996, the fourth one includes papers published from 1975 to 2006, and the last group includes papers published from 2007 to year 2016. We observe that the number of nodes is growing almost exponentially, and the number of edges is growing even faster. The average degree of the citation network in 1975 is 2.06. After 11 years the average degree grows to 3.07. In the time period 1975 to 1996 the average degree stands at 5.23, then it grows to 8.53 in the time period from 1975 to 2006, and lastly the average degree of all papers in the data set from 1975 to 2016 is 10.44.

Quantitative metrics can distinguish networks, we can learn about their topologies and their properties. One of the most popular metrics is *network density.* Network density is the ratio of the edges in the network to all possible edges in the network. If our network consists of n nodes the maximum number of edges in an undirected network is $\binom{n}{2}$. If we have a density of zero, then there are no connections at all, so we have only isolated nodes. If the density is one, then we have all the possible edges in the network, otherwise known as a complete graph K_n.

Citation networks are acyclic directed networks, but in some extreme events there may be that two papers are citing each other. These situations do not influence the magnitude therefore using the formula for calculating the density of a simple directed network can be applied also here. In this citation network 0.001% of the links have an opposite direction. In the first group, we see that density is 0.000573, which is on the lower end of the scale and it means that there are very few connections. Even though the density is very small, we observe that when the network is growing the density becomes even smaller. By observing the first row of Table 5, we see that we have almost as many edges as nodes, which is not the case for the other rows, as we have way more edges than nodes. Since we have more nodes, then the number of all possible edges also increases. Thus shrinking the density even further on the density scale.

Some other important properties to check in networks are the size of the largest and smallest connected component. All of the other connected components, except the largest one, have a small number of nodes which does not exceed 10 nodes. From this statistic, we suspect that this citation network follows a power law degree distribution since the majority of the nodes is grouped together in a component. In [44], Price shows that the in-degree and the out-degree in a directed network have a power law distribution sequence. To show that this is indeed the case with this citation network, we plot the in-degree and out-degree distribution sequence of this citation network in Figure 22. Note that the majority of the nodes have a small in-degree and out-degree, and only a few nodes large in-degree and out-degree.

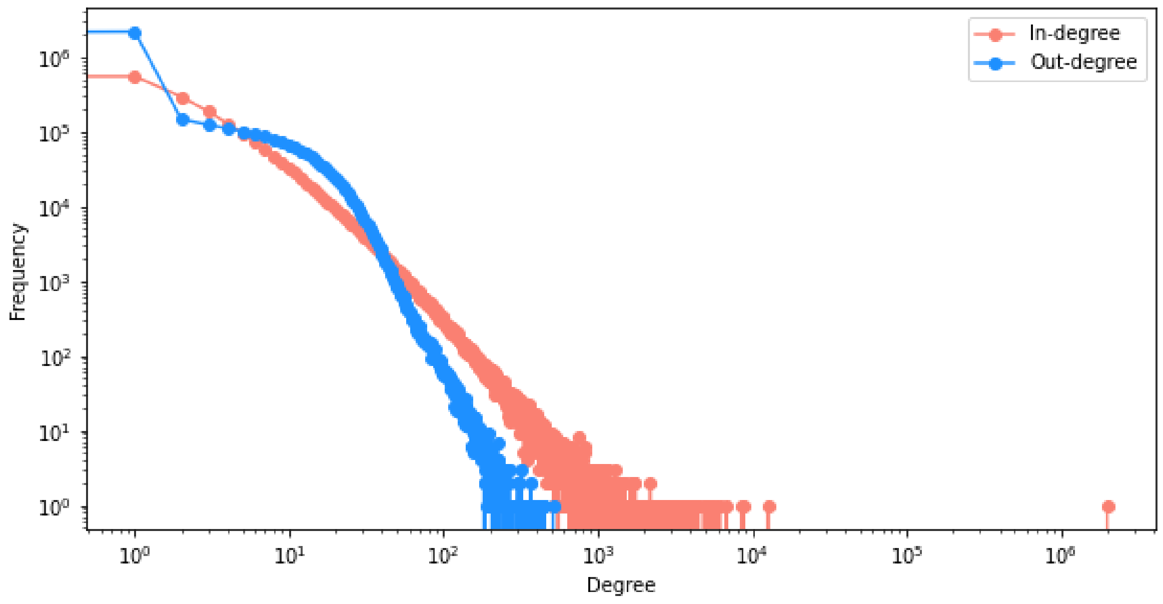


Figure 22: In-degree and out-degree distribution sequence of the citation network.

Since this data set consists of over 7000 conferences and journals, we can see which ones are the top 15 publishing venues in Figure 23.

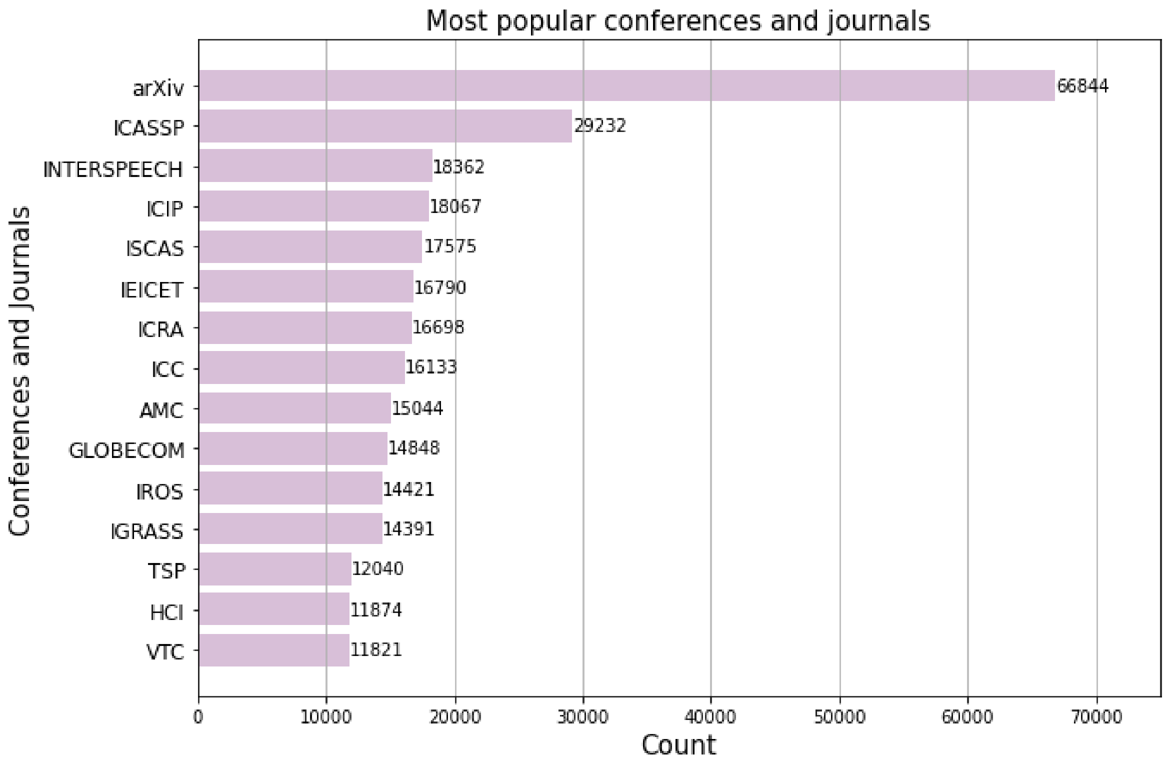


Figure 23: Most popular conferences and journals.

Most of the papers in our data set are published in arXiv, which is an open-access repository of electronic preprints and postprints, that are not peer-reviewed but are approved for posting after moderation. The second most popular venue in our data set is the IEEE International Conference on Acoustics, Speech, and Signal Processing, the third one is the Conference of the International Speech Communication Association, the fourth one is IEEE International Conference on Image Processing, which are top conferences in computer science backed by the IEEE organization. As noted before we see that all of the journals and conferences are from the field of computer science.

This data set consists of almost twenty million edges indicating that there are almost twenty million citations. In Figure 24, we see which papers (their Semantic Scholar IDs) are the most cited ones, and how many citations they received.

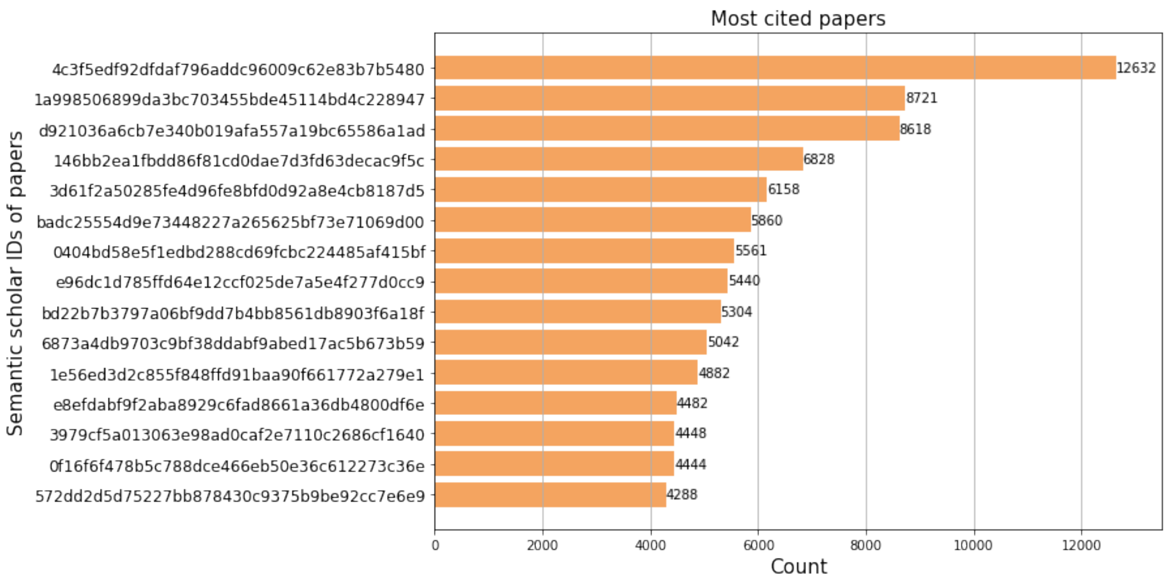


Figure 24: Most cited papers in this data set.

In Table 6, we present the titles of the aforementioned papers to see which are the exact ones with the most influence. It is important to note that the ID of "Multiple View Geometry in Computer Vision" paper was later changed to another ID[2].

[2]It was changed to "339093c7ed71919ce59a7e78979a77abd25bad0c" instead of "e8efdabf9f2aba8929c6fad8661a36db4800df6e" appearing in Figure 24 and in Table 6

Table 6: Paper titles and their IDs.

IDs	**Paper Title**
4c3f5edf92dfdaf796addc96009c62e83b7b5480	Computers and Intractability: A Guide to the Theory of NP-Completeness
1a998506899da3bc703455bde45114bd4c228947	Distinctive Image Features from Scale-Invariant Keypoints
d921036a6cb7e340b019afa557a19bc65586a1ad	Elements of information theory (2. ed.)
146bb2ea1fbdd86f81cd0dae7d3fd63decac9f5c	Genetic Algorithms in Search Optimization and Machine Learning
3d61f2a50285fe4d96fe8bfd0d92a8e4cb8187d5	The Nature Of Statistical Learning Theory
badc25554d9e73448227a265625bf73e71069d00	Probabilistic reasoning in intelligent systems - networks of plausible inference
0404bd58e5f1edbd288cd69fcbc224485af415bf	The Mathematical Theory of Communication
e96dc1d785ffd64e12ccf025de7a5e4f277d0cc9	C4.5: Programs for Machine Learning
bd22b7b3797a06bf9dd7b4bb8561db8903f6a18f	Reinforcement Learning: An Introduction
6873a4db9703c9bf38ddabf9abed17ac5b673b59	A Tutorial on Hidden Markov Models and Selected Applications
1e56ed3d2c855f848ffd91baa90f661772a279e1	Latent Dirichlet Allocation
e8efdabf9f2aba8929c6fad8661a36db4800df6e	Multiple View Geometry in Computer Vision
3979cf5a013063e98ad0caf2e7110c2686cf1640	Basic local alignment search tool.
0f16f6f478b5c788dce466eb50e36c612273c36e	LIBSVM: A library for support vector machines
572dd2d5d75227bb878430c9375b9be92cc7e6e9	Statistical learning theory

The process of extracting features from raw data is known as *feature engineering*, whose aim is to improve the accuracy and performance of machine learning algorithms. This process involves a few steps such as selecting the most useful features to train known as *feature selection*, combining existing features known as *feature extraction*, and creating new features by acquiring new data. This ensures that the data set contains enough relevant features.

The feature engineering process in [56] focused on features that can be extracted from the citation graph, coauthor graph, and paper metadata. They came up with 63 features as shown in Table 7.

In the following section we present our feature engineering pipeline for the Encoder-Decoder model and the Network based model. In our data set we will not include features from the co-authorship network because, although we have the available data, we aim to make our framework easily reproducible since data acquisition of co-authorship networks is generally difficult.

Table 7: Features used by Weihs and Etzioni.

Features	**Description**
author_hindex_{mean, min, max}	H-indices of authors
author_hindex_delta_{mean, min, max}	Change in h-indices of authors in last 2 years
author_citations_{mean, min, max}	Cumulative citations for each author
author_key_citations_{mean, min, max}	Cumulative key citations for each author
author_key_citations_delta_{mean, min, max}	Change in cumulative key citations for each author in last 2 years
author_mean_citations_{mean, min, max}	Mean citations per paper for each author
author_mean_citations_delta_{mean, min, max}	Change in mean citations per paper for each author in last 2 years
author_papers_{mean, min, max}	Number of papers published for each author
author_papers_delta_{mean, min, max}	Number of papers published for each author in last 2 years
author_unweighted_pagerank_{mean, min, max}	PageRank of each author in the unweighted coauthorship network
author_weighted_pagerank_{mean, min, max}	PageRank of each author in the weighted coauthorship network
author_mean_citation_rank_{mean, min, max}	Rank of each author among all authors in terms of mean citations per paper
author_recent_num_coauthor_{mean, min, max}	Number of coauthors each author had in last 2 years
author_max_single_paper_citations_{mean, min, max}	Maximum citations a single paper of each author has received
total_num_authors	Total number of authors for the paper
venue_hindex	H-index of the venue
venue_hindex_delta	Change in h-index of the venue in last 2 years
venue_mean_citations	Mean citations per paper published in the venue
venue_mean_citations_delta	Change in mean citations per paper published in the venue in last 2 years
venue_papers	Number of papers published in the venue
venue_papers_delta	Number of papers published in the venue in last 2 years
venue_rank	Rank of the venue among all venues in terms of mean citations per paper
venue_max_single_paper_citations	Maximum number of citations any paper published in the venue has received
paper_age	Age of the paper in years (rounded up)
paper_citations	Cumulative citation count
paper_key_citations	Cumulative key citation count
paper_mean_citations_per_year	Average number of citations received per year
is_survey	Whether or not the paper is a survey
paper_citations_delta_{0,1}	Number of citations the paper received in the last year and the year before that
paper_key_citations_delta_{0,1}	Number of key citations the paper received in the last year and the year before that

4.2 FEATURE ENGINEERING

We will use the same citation graph, coauthorship graph, and paper metadata as Weihs and Etzioni to extract features needed for the Encoder-Decoder model and the Network based model.

4.2.1 Feature engineering for Encoder-Decoder model

Abrishami and Aliakbary show in [1] that we do not need a lot of features to have good predictions. They use only these 9 features as shown in Table 8.

Table 8: Features used by Abrishami and Aliakbary.

Features
paper_id
journal_id
publication_year
citations_in_year_0
citations_in_year_1
citations_in_year_2
citations_in_year_3
citations_in_year_4
citations_in_year_5

We observe that Abrishami and Aliakbary use a totally different approach compared to Weihs and Etzioni. We use Python 3 [54] as a programming language to do all the data manipulation process. The main packages that we use in the process of feature engineering are:

- **pandas** [57], which is an open source data analysis and manipulation tool built on top of the Python.
- **NumPy** [27], which is a Python library that assists with large, multi-dimensional arrays and matrices, and includes mathematical functions to operate on these arrays.
- **NetworkX** [26], which is a Python package for the creation, manipulation, and study of structure, dynamics, and functions of complex networks.
- **json** [9], which is a data interchanged format based on JavaScript object syntax to represent structured data.

- **Matplotlib** [29], which is a plotting library for Python.

Visualization is a crucial step in analyzing data sets and getting further insights, so we use Matplotlib as a visualization tool. The networks were in json file format therefore we use the respective Python library named json to load the data sets. To manipulate with the data we use the pandas library. To replicate Abrishami and Aliakbary's feature extraction process, we select papers that have at least 16 years of history since their publication date. More precisely, we extract the citation count of papers in $2001 - 2005$ as an input variable in order to predict citation count of papers in the upcoming years $2006 - 2015$. We use NetworkX to extract this information from the citation graph and then we count the in-degree of each paper for 16 consecutive years. Each node has its unique identifier "paper_id" and the edges represent the year that a paper cited another paper. The journal or conference that the paper was published in is extracted from paper metadata which is in a *csv* (comma-separated values) format. Table 9 illustrates how the data set looks like after feature engineering for five sample papers.

Table 9: Features of five sampled papers for Encoder-Decoder model.

id	**venue**	**year**	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
2a368cc463ae66611a683dba202528a055ac3475	iccs	2001	2	7	8	9	12	14	16	16	19	19	20	21	22	22	22
91fe40b93f877df9b5230f7aa9a5f5a814fc6093	tsp	1997	4	8	9	9	10	12	12	12	12	12	12	12	12	12	12
736435d881ce14243725d746c66d3d803f9a7fa4	esa	2001	0	2	9	11	13	15	17	20	20	20	21	21	21	21	21
080a875ff8d444a7efcde0237d6edfa7136ecbd9	iva	2001	0	0	0	0	1	1	2	2	2	2	2	2	2	2	2
a67b0f5072f929ffecb062592316823f2ea5db9c	tcsv	1994	16	18	20	21	24	26	26	29	30	32	32	32	34	34	34

Since machine learning algorithms do not accept strings as input we transform the first two columns of the data set, namely "id" and "venue", to numerical variables. The columns c_i where $i \in [1, 2, \ldots, 15]$ represent citation count after i years. The first 8 columns are the predictors and the other 10 columns are the response variables.

4.2.2 Feature engineering for Network based model

In this section, we describe our novel feature engineering approach based on network properties. To implement this approach we use the same citation network, co-authorship network and paper metadata as in [56]. Up to now, we have seen two approaches that were used to handle this type of data, which have the same goal of predicting citation count up to 10 years in advance. The h-index is typically regarded as an author-level indicator that assesses a scientist's or scholar's productivity as well as the effect of his or her publications in terms of citations. The approach in [56] is to predict citation count based on h-indices of authors, number of authors, venue h-index, age of papers and many other variables that can be extracted from the co-authorship network, citation network and paper metadata. The approach in [1] is to predict citation count based on the venue where the paper was published, year it was published and the first five years after it was published.

Among many other differences these two approaches have, they also differ by the number of predictors used, since the first approach uses approximately eight times more predictors. None of these two approaches explain which are the most important features that authors need to tweak in order to maximize the citation number their papers in the future.

Our method relies strictly on the position of a paper in the network. We show that the position of a paper in the network suffices to have strong predictions. Instead of looking for variables such as h-indices of the venues, h-indices of the co-authors, or even waiting for some years to use the past years' citation count as predictors for the future, we utilize network properties of nodes that will be extracted from the citation network through time.

In Figure 25 we show an example of the evolution of a network through time known as a *temporal network*. A *temporal network* is a sequence of n networks

$$G(V_i, E_i, t_i) \text{ with } 1 \leq i \leq n$$

such that t_i is a time stamp, and V_{i+1} and E_{i+1} are constructed from V_i and E_i by adding or removing nodes and links. In the case of citation networks, we only add nodes and links so $E_i \subseteq E_{i+1}$ and $V_i \subseteq V_{i+1}$.

From here we observe that the network is growing over time as new nodes are being added at each new time stamp. The introduction of new nodes at each time step t_i changes the network structure, global network properties, and network properties of each node.

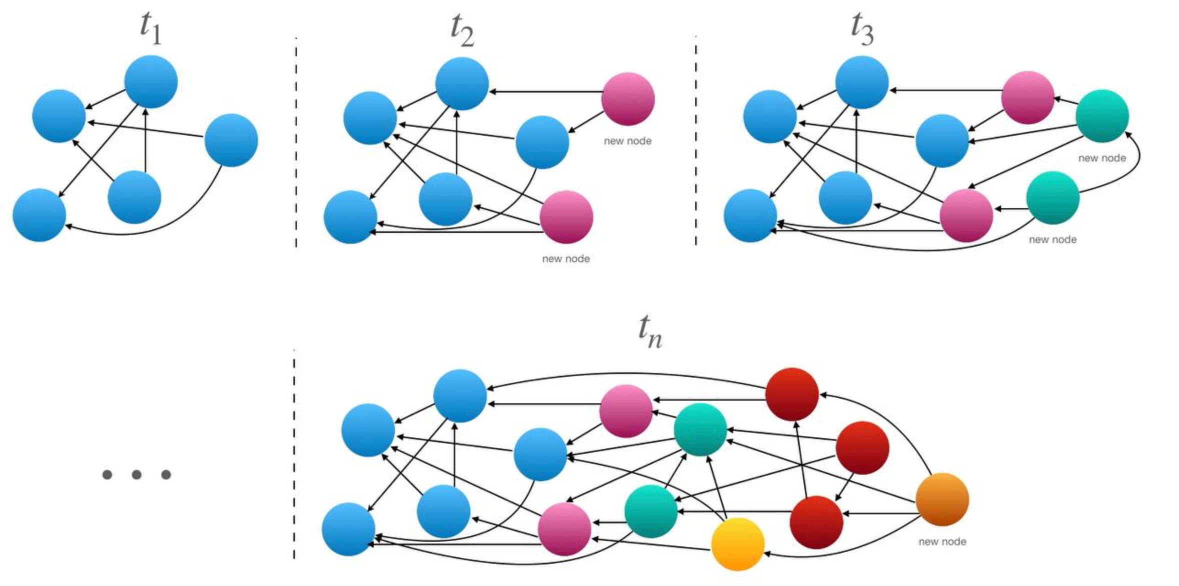


Figure 25: Evolution of network through time.

In our data set, we take advantage of these changes as we will record the network properties of nodes at each time stamp t_i and by doing this, the amount of instances (rows) increases approximately six-fold. This means that the older a paper is, the more instances of it we have in our data set. Since the network is changing at every time step, none of the features that we extract will remain the same, thus while increasing the size of our data set we are not introducing any bias.

We start this feature engineering process by first removing all the isolated nodes, which will decrease the number of papers in this data set as shown in Figure 26.

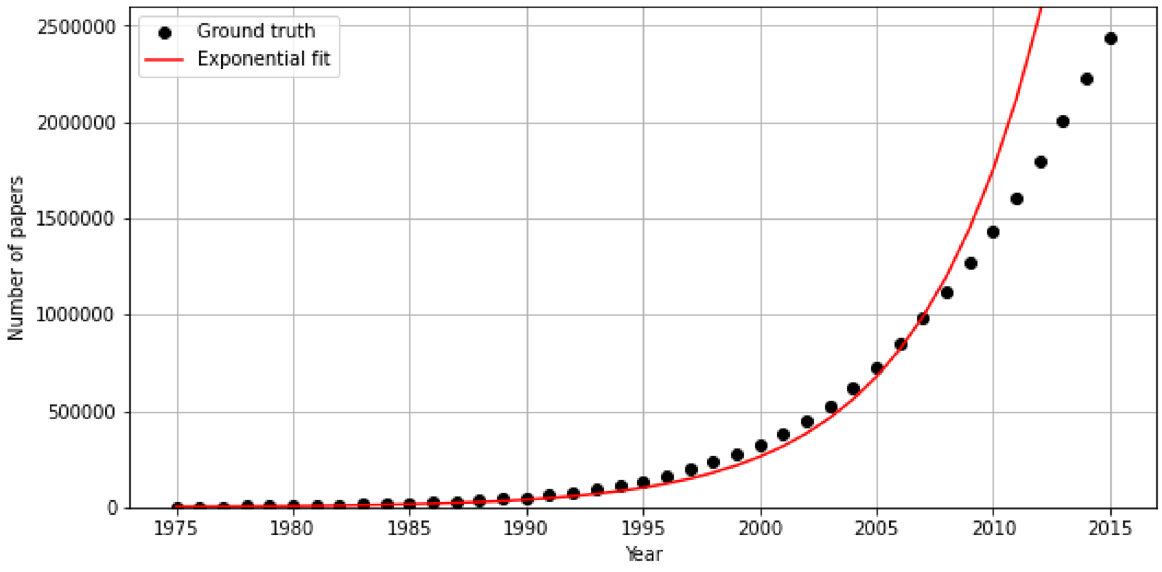


Figure 26: The exponential fit and the cumulative number of published papers after feature engineering.

After removing all the isolated nodes we start recording the properties of nodes from 1975 up to 2005. In Table 10 we display all the features that will be extracted from the network using the Python library NetworkX. Again in this step, we use pandas and NumPy to transform and manipulate with the data. For most of the features in Table 10 there are implemented functions in NetworkX. The functions that are not implemented in NetworkX are the ones that check properties of successors.

Table 10: Network based features.

Features
In-degree
Out-degree
Degree Centrality
In-degree Centrality
Out-degree Centrality
Eigenvector Centrality
Katz Centrality
Average Neighbour Degree
PageRank
Clustering Coefficient
Squares Clustering Coefficient
Minimum of successors in-degrees
Maximum of successors in-degrees
Mean of successors in-degrees
Sum of successors in-degrees
Minimum of successors PageRank
Maximum of successors PageRank
Mean of successors PageRank
Sum of successors PageRank
Age

Since we want to predict the success of each node, then an appropriate approach to take is the extraction of information about the importance of each node in the network. As mentioned before, citation networks are scale-free networks, which follow a power law degree distribution. Preferential attachment is suggested as a method for explaining power law degree distributions in real networks [5]. This implies that nodes with a higher number of citations will continue to receive additional citations, or in other words, "the rich get richer". As a result, extracting these measure of importance for nodes will provide a good summary of the expected number of citations that a scientific paper will receive in the future.

Let us describe the features from Table 10 for a comprehensive understanding of the feature engineering process. Let v be a node from the set of all nodes $V(G)$, where G is the citation network. The in-degree of node v is the number of current citations that the node has received. The out-degree of node v is the number of papers that v has cited (used as references).

The most important nodes in network analysis and graph theory are identified by indicators of *centrality* [36]. *Centrality measures* provide rankings which identify the most important nodes based on different functions. Based on how centrality is constructed, we can classify nodes in two classes: *radial* or *medial. Radial centralities* count walks which start or end from a given node, while *medial centralities* count walks which pass through a given node.

Social network analysis is the area where centrality concepts where first developed, thus applications of centrality include identifying the most influential people in social media, but also other applications such as finding the super-spreaders of a disease.

In a directed network $G = (V, E)$, the *degree centrality* of a node v is the ratio of the number of links incident to v to the maximum possible degree in a simple graph, which equals $|V| - 1$. The *in-degree centrality* for a node v is the ratio of the number of incoming links to $|V| - 1$. The *out-degree centrality* for a node v is the ratio of the number of outgoing links to $|V| - 1$. In what follows, we consider the underlying network for calculations if the network is directed, unless it is specified otherwise.

Eigenvector centrality [7] computes the centrality of a node v based on the centrality of its neighbours. The eigenvector centrality for a node v is given by $Ax = \lambda x$, where A is the adjacency matrix of the graph G, λ is a scalar and x consists of values of the centrality measure of neighbours of v. The scalar λ is called an *eigenvalue* and x is called an *eigenvector*. In general, the matrix A will have multiple eigenvalues, but the Perron-Frobenius theorem [8] guarantees that if all the entries of the eigenvector x are positive, then there is only one eigenvalue that satisfies this requirement. Therefore, we can assign a unique centrality score to each node.

Similar to eigenvector centrality, *Katz centrality* [25] is a measure which computes the centrality for a node based on the centrality of its neighbours. It generalizes the eigenvector centrality. To compute the Katz centrality for node v we have the following formula:

$$x_v = \alpha \sum_u A_{v,u} x_u + \beta,$$

where A is the adjacency matrix of graph G, the parameter α is bounded above by the reciprocal of $\lambda_{\max}$, where $\lambda_{\max}$ denotes the largest eigenvalue, β is a constant initial weight assigned to each node in order to account for nodes with zero in-degree or out-degree, and x_v and x_u are vectors. If $\alpha = \frac{1}{\lambda_{\max}}$ and $\beta = 0$, then Katz centrality is the same as eigenvector centrality. It thus extends the concept of eigenvector centrality to directed networks that are not strongly connected. From the formula, we see that Katz centrality computes the relative influence of a node by measuring the number of neighbours.

PageRank [21] is an iterative algorithm used by Google Search to rank web-pages in their search engine outputs. PageRank was originally a function that assigns a real number to each page in the web, and now it is a widely used network analytics technique. A page's rank is a value between 0 and 1, each page starts with some amount of rank and the sum of all pages' PageRanks is 1. There are a lot of varieties of PageRank algorithm. A basic PageRank formula is:

$$PR(v) = \sum_{u} \frac{PR(u)}{|\text{outdeg}(u)|},$$

where the sum runs over all predecessors u of v. We calculate the page rank of each node using the equation above repeatedly until convergence.

The *clustering coefficient* [49] for directed graphs is calculated as follows:

$$C_3(v) = \frac{T(v)}{\deg(v) \cdot (\deg(v) - 1) - \frac{2}{\deg(v)}},$$

where $T(v)$ is the number of *triangles* (a set of three nodes such that any two of them are connected by a link) through node v. This coefficient gives the probability that two neighbours of node v are adjacent.

The *squares clustering coefficient* [35] for directed graphs is calculated as follows:

$$C_4(v) = \frac{\sum_{u=1}^{k_v} \sum_{w=u+1}^{k_v} q_v(u,w)}{\sum_{u=1}^{k_v} \sum_{w=u+1}^{k_v} [a_v(u,w) + q_v(u,w)]},$$

where $q_v(u,w)$ is number of shared neighbours of u and w without considering node v, and

$$a_v(u,w) = (k_u(1 + q_v(u,w) + \theta_{u,v}))(k_w - (1 + q_v(u,w) + \theta_{u,w})),$$

where $\theta_{u,w} = 1$ if u and w are adjacent and 0 otherwise. This coefficient gives the probability that two neighbours of node v share a common neighbour different from v.

The *minimum of successors in-degrees* is calculated as:

$$\min\{\deg(u) : u \text{ is a successor of } v\}.$$

The *maximum*, *mean*, *sum* of successors in-degrees are calculated similarly, by switching to the appropriate operations.

The *minimum of successors PageRank* is calculated as:

$$\min\{PR(u) : u \text{ is a successor of } v\}.$$

The *age* is the year that indicates when a certain paper was published.

The *average neighbour degree* of a node v is the sum of degrees of all the neighbours of v divided by the number of neighbours of v. If the network is directed, we will use the sum of out-degrees of nodes that have v as a source node.

Example 4.1. Let D be the digraph from Figure 27. Let us see how the network based features would differ when we extract the aforementioned features. The digraph from Figure 27 randomly generated and drawn using the NetworkX package.

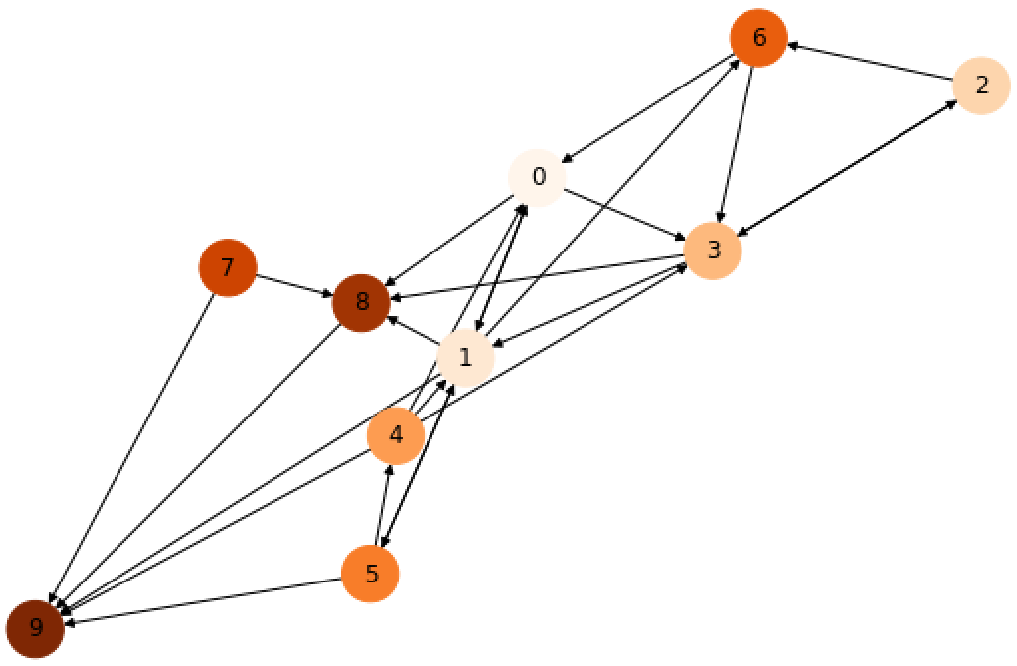


Figure 27: A randomly generated digraph with 10 nodes and 25 arcs.

In Table 11 we explore the variability of the features from Table 10 for the digraph from Figure 27.

Table 11: Network based features of the random generated graph D.

Features/Nodes	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
In-degree	3	4	1	4	1	1	2	0	4	5
Out-degree	3	5	2	3	4	3	2	2	1	0
Degree Centrality	0.666667	1	0.333333	0.777778	0.555556	0.444444	0.444444	2.222222e-01	0.555556	0.555556
In-degree Centrality	0.333333	0.444444	0.111111	0.444444	0.111111	0.111111	0.222222	0	0.444444	0.555556
Out-degree Centrality	0.333333	0.555556	0.222222	0.333333	0.444444	0.333333	0.222222	2.222222e-01	0.111111	0
Eigenvector Centrality	0.323088	0.409976	0.154782	0.352870	0.078879	0.179829	0.247723	4.198040e-10	0.476328	0.502240
Katz Centrality	0.327721	0.357245	0.271019	0.351870	0.262988	0.271556	0.298658	2.358319e-01	0.363099	0.384904
Average Neighbour Degree	3	1.8	2.5	2.666667	2.75	3	3	5.000000e-01	0	0
PageRank	0.100347	0.125821	0.071829	0.137916	0.048093	0.054142	0.084670	3.275272e-02	0.135571	0.208858
Clustering Coefficient	0.392857	0.25	0.5	0.25	0.4	0.5	0.5	5.000000e-01	0.3	0.3
Squares Clustering Coefficient	2	0.666667	0	2	4	2	0.666667	0	0	0
Minimum of successors in-degrees	4	1	2	1	3	1	3	4	5	0
Maximum of successors in-degrees	4	5	4	4	5	5	4	5	5	0
Mean of successors in-degrees	4	3	3	3	4	3.333333	3.5	4.5	5	0
Sum of successors in-degrees	12	15	6	9	16	10	7	9	5	0
Minimum of successors PageRank	0.125821	0.054142	0.084670	0.071829	0.100347	0.048093	0.100347	1.355707e-01	0.208858	0
Maximum of successors PageRank	0.137916	0.208858	0.137916	0.135571	0.208858	0.208858	0.137916	2.088577e-01	0.208858	0
Mean of successors PageRank	0.133103	0.116718	0.111293	0.111074	0.143236	0.127591	0.119132	1.722142e-01	0.208858	0
Sum of successors PageRank	0.399308	0.583588	0.222586	0.333221	0.572942	0.382772	0.238263	3.444284e-01	0.208858	0

4.3 MODEL SELECTION

After performing an exploratory data analysis and feature engineering, the next step in a data science project pipeline is feeding the data to a machine learning model. In this section we present the Encoder-Decoder model into which we feed the data from Section 4.2.1. Furthermore, we build two machine learning models into which we feed the data from our novel feature engineering technique from Section 4.2.2.

Before continuing with the next sections, we first analyze the machine learning model which Weihs and Etzioni used in [56]. There are six different machine learning algorithms that are compared with one another based on the R^2 *metric* or R^2*-score*. This metric compares the relative performance of a model against a predictor that returns the mean of the labels. The formula to calculate the R^2-score is

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{4.1}$$

where N is the total number of papers, y_i is the citation count of the ith paper, $\hat{y}_i$ is the predicted citation count for the paper, and $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ is the average citation count over all papers.

The six models used in [56] are:

1. Plus-k, which is a baseline model that adds a fixed constant to all citation counts every year, this constant is chosen by linear regression using Hubber loss (a loss function that is less sensitive to outliers in data).

2. Simple Markov, which is a linear regression model that is only given features describing citation counts in 2005 and the change in paper's citation count from 2003 to 2005.

3. Lasso, which is a regularized linear regression model that uses all features with the regularization parameter chosen by 10-fold cross validation.

4. Random forest, which is an ensemble of regression trees using randomization techniques to improve performance.

5. Gradient boosted regression trees, which is a collection of simple regression trees that are trained iteratively by a type of functional gradient descent.

6. Reinforced Poisson Process, which is a probabilistic model for predicting citation counts of individual papers.

After all the experiments that were conducted, the Gradient boosted regression trees model outperforms all the other models and Reinforced Poisson Process is a very close runner-up. We will refer to the best performing model here as a baseline model in the up-coming sections.

4.3.1 Encoder-Decoder model

In [1], Abrishami and Aliakbary build a model that learns to predict citation count of a paper in the future based on its citation history. Predicting citation count of a paper in the future is defined as a regression problem in the context of machine learning methods. Since Artificial Neural Networks are one of the most popular and powerful methods for regression learning, they propose a neural network that learns to predict $\hat{c}_{k+1}, \hat{c}_{k+2}, \dots, \hat{c}_n$ values based on $c_0, c_1, \dots, c_k$ values, where c_i for $i \in 1, 2, \dots, k$ are the past citation counts of a paper and $\hat{c}_i$ for $i \in k+1, k+2, \dots, n$ are the predicted citation count for the future of that paper. The neural network is first trained on a subset of the data and then the rest of the data is used as a test set to evaluate the accuracy of the trained neural network.

From the definition of this problem we see that both the history of a paper's citation count (input) and the future of a paper's citation count (output) form a sequence of consecutive values. Since Recurrent Neural Networks are known to be the most effective for learning sequence of the values, the authors make use of this algorithm. The data sequence that is fed to RNNs is processed in their intrinsic order and a hidden memory is built as the sequence is being processed, based on the past values of that sequence. Thus a model that has a sequence-to-vector and a vector-to-sequence architecture was designed for this problem, which is called a *many-to-many* or Encoder-Decoder RNN architecture, which we defined in Section 2.2.2. This model is composed of two independent neural networks called the encoder and the decoder.

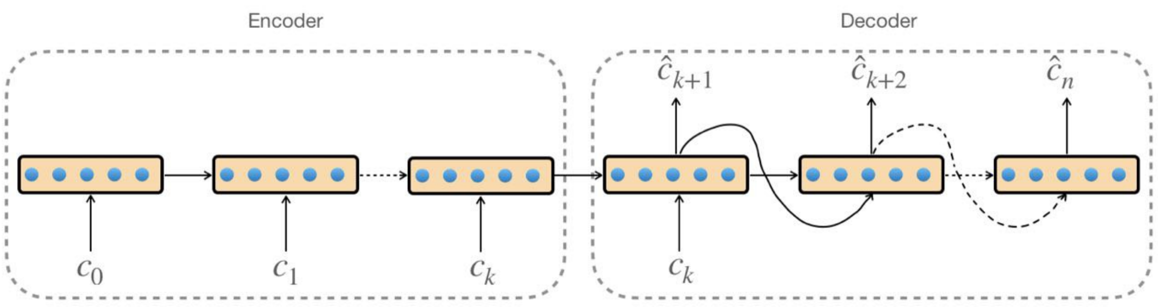


Figure 28: The proposed RNN architecture.

In Figure 28 we see that the input sequence is fed to the encoder, then the output of the encoder is fed as an input for the decoder, and finally the output of the decoder gives the actual predictions.

Keras [12] is a very famous framework that is used to implement various deep neural networks. The *SimpleRNN* module of Keras is used for implementing the proposed neural network. To make better generalizations and learn complex functions, the recurrent layers include 512 neurons, which implies that the output of the encoder will be a 512 dimensional vector. The decoder has a *Dense* layer to generate the predictions. The Rectified Linear Unit activation function is used on all layers of the neural network. Overfitting is always a problem in machine learning algorithms so the *Dropout technique* with rate 0.2 is used avoid it. *Dropout* is a very popular and one of the most used regularization techniques for DNNs. During training, some layer outputs are disregarded or *dropped out* at random. The probability of a neuron being temporarily ignored during a certain step is called the *dropout rate.*

To minimize risk of overfitting we use 100 epochs. The optimization algorithm that we use is *RMSProp* with a learning rate of 10^{-5}, which is an algorithm that accumulates only the gradients from the most recent iterations and thus ensures convergence to the global optimum. The data is fed to the network in batches of size 256. The summary of this model is presented in Table 12.

Table 12: The implementation details of the Encoder-Decoder model.

Neural network API	Keras
RNN module	SimpleRNN
Output dimensions of the encoder	512
The output layer	Dense layer
Activation function	ReLU
Overfitting prevention technique	Dropout with 0.2 rate
Epochs	100
Optimization algorithm	RMSProp
Learning rate	10^{-5}
Batch size	256

4.3.2 Network based model

In this section, we describe two different models into which we fed the data from Section 4.2.2. The first model that we present is a Gradient boosted regression trees model. The framework that we use for this model is scikit-learn, and the algorithm's name is *GradientBoostingRegressor.* We also try different parameters so we can get the best generalization. *GridSeachCV* is the module used to find the best parameters, since the data set is big (see Table 13) and finding the best parameters can take a long time.

Table 13: Data set and the number of instances.

Data set	**Size**
Train set	3705675
Validation set	463186
Test set	465621

We first train the Gradient boosted regression trees algorithm independently for each year. We need to train this model 10 times, and for each year, we need to use GridSearchCV for hyperparameter tuning. Before feeding the data to our model, we first scale the data set by using *StandardScaler*, which standardizes features by subtracting the mean and scaling to unit variance. The hyperparameters that we choose to tune are shown in Table 14, together with the ranges that the data set was trained on.

Table 14: GBRT hyperparameters.

Parameters	**Range**
n_estimators	(100, 200, 300, 400, 500, 600)
max_depth	(2, 3, 4, 5, 6, 7, 8)
loss	lad
min_samples_leaf	(2, 3, 4, 5)
min_samples_split	(2, 3, 4, 5)
subsample	(0.8, 0.9, 1)

The hyperparameter “n_estimators” is the number of boosting stages to perform. Maximum depth of the individual regression estimators is the “max_depth” parameter. The parameter “loss” represents the loss function which is used to compute the quantity that a model should seek to minimize during training. We choose “lad” for the “loss” hyperparamter which refers to least absolute deviation. It is a highly robust loss function based on order information of the input variables. The “min_sample_leaf” hyperparameter is the minimum number of samples required to be at a *leaf node*, which is a node of degree 1, and “min_sample_split” hyperparameter is the minimum number of samples required to split a non-leaf node. Finally, the hyperparameter “subsample” is the fraction of samples to be used for fitting the individual base learners. The other hyperparameters of the module *GradientBoostingRegressor* are left in their default setting. The pseudo-code for this approach to find the best model regarding the hyperparameters is shown in Algorithm 2.

The second model that we implement is a Deep Neural Network. The reason that we choose to implement a DNN is because of the amount of data that we have. DNNs often perform better than other models when they are fed a lot of data, and in this case we have more than four million instances in our train set and validation set together (see Table 13). Keras framework is used again to implement this model. As with GBRT, we use GridSearchCV from scikit-learn to find the best hyperparameters for this model. The algorithm is the same, except that we change the model to a DNN and the hyperparameter space changes.

Algorithm 2: Model selection and evaluation metric

```
   Input: train_val_set, all_possible_hyperparameters
   Output: Evaluation of the model and the model trained with the best
           hyper-parameters
 1 metric_for_all_models = [ ];
 2 for hyper_para in all_possible_hyperparameters
 3     fold_metric = [ ];
 4     for fold in cross_validation_fold(3)
 5         train_fold, test_fold = split(fold);
 6         model = train(train_fold, hyper_para);
 7         metric = evaluate(model, test_fold);
 8         fold_metric.append(metric) ;
 9     metric_for_all_models.append((mean(fold_metric), hyper_para));
10 best_hyper_para = sort(metric_for_all_models); /* Sort based on the
       metric                                                          */
11 best_model = train(train_set, best_hyper_para[0][1]); /* train the model
       with the best hyperparameters                                   */
12 test_metric = evaluate(model, test_set);
13 return test_metric, best_model
```

We again train this model for each year separately and although we use 3-fold cross-validation, we then run the best performing model three more times and average the results over all the runs. This is because the weights are randomly assigned first and they tend to play a role in the outputs as well. The hyperparameters that we chose to tune for this model are “n_hidden”, “n_neurons” and “learning_rate”. In Table 15 we can see the ranges for all hyperparameters.

Table 15: DNN hyperparameters.

Parameters	**Range**
n_hidden	(1,2,3,4,5,6)
n_neurons	(150, 200, 250, 300, 320, 360, 400)
learning_rate	(0.0003, 0.03)

Next, we create a *Sequential* model. This is the simplest kind of Keras model for neural networks, which are composed of a single stack of layers connected sequentially. Then we add an *InputLayer* as a first layer and set the input shape. After that, we add a Dense hidden layer with "n_neurons".

Each Dense layer manages its own weight matrix, containing all the connection weights between the neurons and their inputs. It also manages a vector of bias terms. The term "n_hidden" determines how many Dense hidden layers we will have. The final layer is a Dense output layer with one neuron since we we will predict only one number (citation count) per paper.

After the model is created, we must compile it and specify the loss function and the optimizer to use. The loss function that we chose is the Mean Squared Error, and the optimizer is *adaptive moment estimation*, or *Adam*, which is an optimization function that keeps track of an exponentially decaying average of past gradients. The reason we use Adam as an optimizer is that training large deep neural networks is a slow process and Adam is faster than the regular Gradient Descent.

When fitting the model we add *callbacks*, which are arguments that let us specify a list of objects that Keras will call at the start and end of training. We use *EarlyStopping* callback. It interrupts training when it measures no progress on the validation set for a number of epochs (defined by the *patience* argument, in our case we set it to 15, and it optionally rolls back to the best model). The number of epochs that we use is 600 and the batch size is 256. The ReLU activation function is not perfect since during training, some neurons stop outputting anything other than zero. A neuron dies when its weights get tweaked in such a way that the weighted sum of its inputs are negative for all instances in the training set. The dropout rate is set at 30%.

We use the *Scaled Exponential Linear Units* (SELU) activation function which is defined by:

$$f(x) = \begin{cases} \lambda x & \text{if } x \geq 0 \\ \lambda\alpha(e^x - 1) & \text{if } x < 0 \end{cases},$$

where $\alpha \sim 1.6733$ and $\lambda \sim 1.0507$. Since all hidden layers will use SELU activation function, then the network will self-normalize: the output of each layer will tend to preserve a mean of zero and standard deviation of one during training. Because of this, the SELU activation function often significantly outperforms other activation functions.

5 RESULTS

In this chapter, we will present the results of our experiments and proceed to compare the models. In Section 5.1 we define the $RMSE$ which is another measurement criterion of our models. We use the R^2-score and the $RMSE$ to monitor and measure the performance of our model. Moreover, we describe the correct interpretations of both criteria.

In Section 5.2 we analyze, interpret and visualize the results of the Encoder-Decoder model. We have given a thorough introduction of the model in Section 4.3.1 and now we utilize it for our predictions. First, we visualize the R^2-score and the $RMSE$ for each of the ten years that we are forecasting and we interpret the possible outcomes. Then, from the test set, we choose six papers at random to visualize and compare the predicted citation count trajectory to the ground truth citation trajectory. We also compare the citation count predictability of papers published in the top four venues.

In Section 5.3 we analyze and compare the results of the models introduced in Section 4.3.2, namely DNN and GBRT. A similar analysis to the Encoder-Decoder model via visualizing the R^2-score and the $RMSE$ for each of the ten years is performed for both models. Moreover, we show the relative importance of each feature (*feature importance*) over time and how that importance shifts throughout the years.

In Section 5.4, we make comparisons of all the models, namely, the Network based model using DNN, the Network based model using GBRT, the Encoder-Decoder model, the baseline model, and the Network based model combined with the baseline model. We see that our novel Network based model outperforms the baseline model by a great margin for the first five years. This shows that the amount of instances added and the potential of network features extracted from the temporal networks can produce remarkable results for predicting the short-term future. Furthermore we combine the Network based features with the baseline features and outperform state-of-the-art methods for short-term and long-term predictions. Checking also the feature importance of Network based features and baseline features we see that the Network based features are more important in describing the correlation between the predictors and response variable.

5.1 MEASUREMENT CRITERIA

To evaluate the proposed methods we make use of two popular metrics: R^2 and root mean square error ($RMSE$). We have seen the formula for the R^2-score specifically given for citation counts in the previous section. The general formula is:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}, \tag{5.1}$$

where y is the vector of observed values, $\hat{y}$ is the vector of predicted values, and $\bar{y}$ is the mean value of y_i. Note that the R^2-score is the measure of the correlation between the ground truth and the predicted values. The value of R^2 is always in the range of 0 and 1, and $R^2 = 1$ means that there is a perfect correlation between the predicted and ground truth values, while $R^2 = 0$ means there is no correlation at all. Thus, higher values of R^2 preferable. The second metric is $RMSE$, which measures the variation of the predicted values to the ground truth values, thus lower values of $RMSE$ are preferable. The formula to calculate the $RMSE$ is given by:

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{5.2}$$

where y is the vector of observed values, $\hat{y}$ is the vector of predicted values.

5.2 ENCODER-DECODER MODEL

We feed the citation counts of the first five years of the papers after they are published to the Encoder-Decoder model, and predict the citation count for the following ten years. The accuracy of this method is computed according to the criteria mentioned in Section 5.1. In Figure 29 we can see the R^2-scores over ten years of predictions. On the x-axis we present the years, on the y-axis we present the R^2-scores and the red line corresponds to the change of R^2 score over the years.

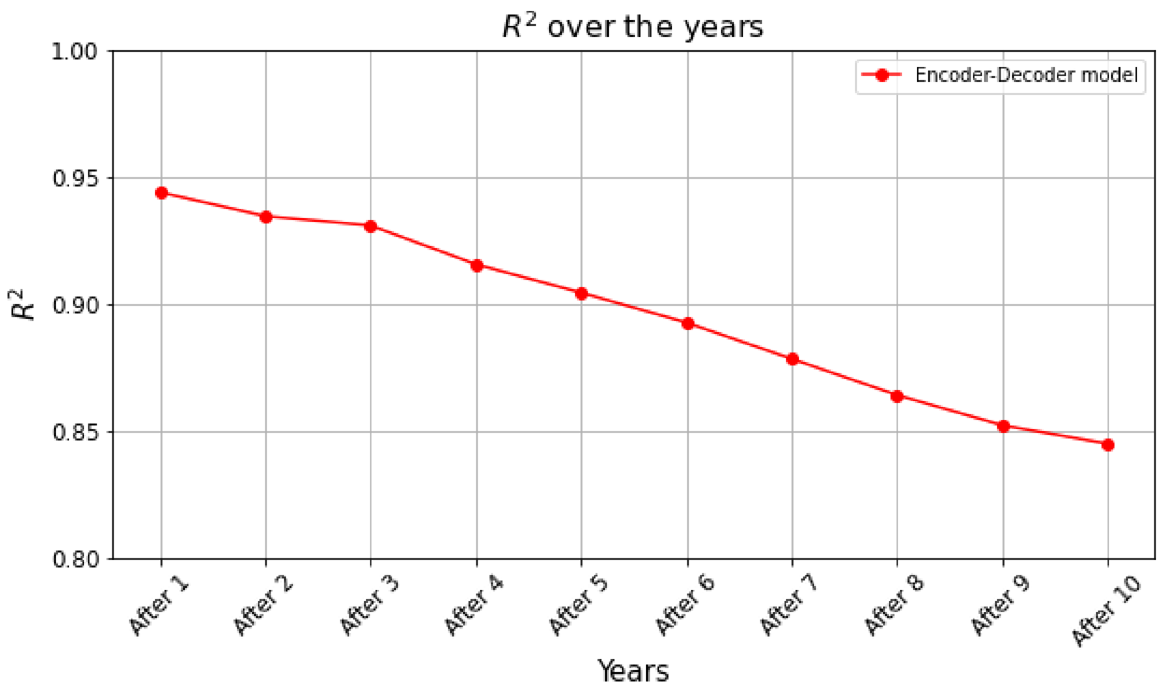


Figure 29: Encoder-Decoder R^2 scores over 10 years.

Here R^2 is the percentage of dependent variable variation that the model explains. From Figure 29 we see that this model explains almost 95% of the variation in the response variable around its mean for the first year. We observe that the highest R^2-score is recorded when predicting the first year in our set of dependent variables. This is somewhat expected since predicting short-term events is easier than predicting long-term events. Even though the R^2-score is decreases after each consecutive year, the R^2-score of the 10th predicted year explains almost 85% of the variation in the response variable around its mean. These are very promising results which can also be attributed to the internal state (memory) architecture of the RNN model.

The $RMSE$ is the square root of the variance of the residuals. It can also be interpreted as the standard deviation of the unexplained variance. It checks how close the observed data points are to the model's predicted values, and thus it stipulates the absolute fit of the model to the data. The $RMSE$ of citation counts for each predicted year is shown in Figure 30.

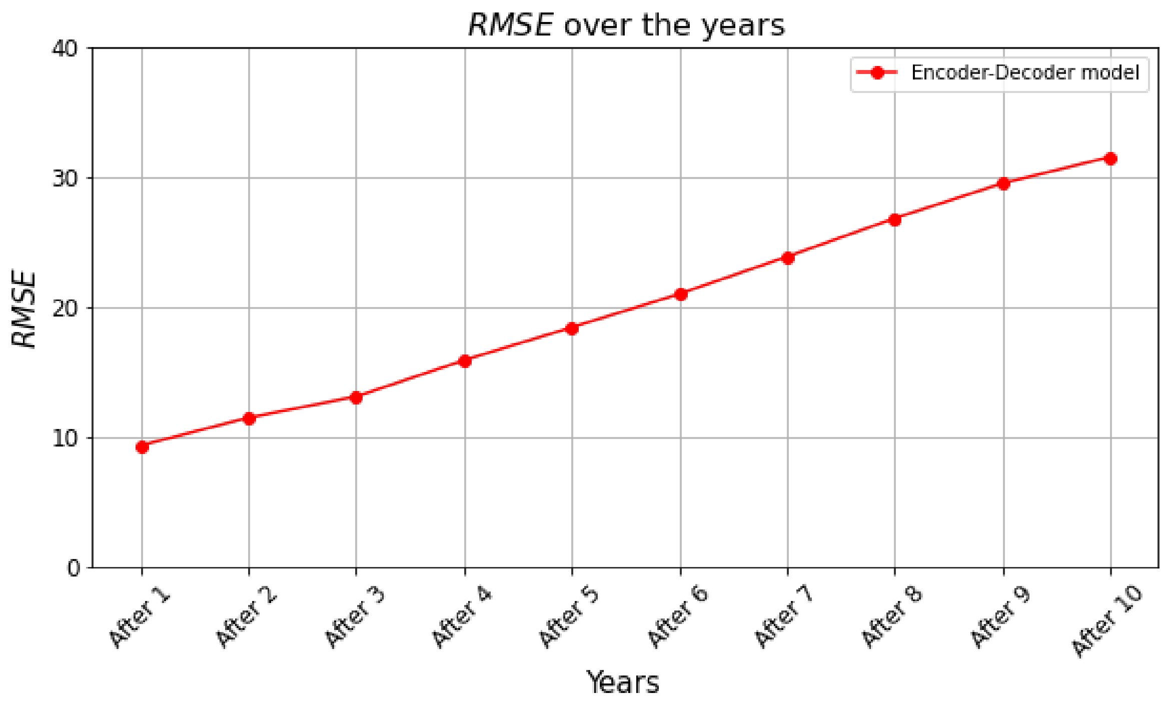


Figure 30: Encoder-Decoder $RMSE$ scores over 10 years.

Observe that the $RMSE$ is increasing after each year, which again shows that it is easier to predict the short-term events rather than long-term ones. The features that we fed to this model indicate that predicting the citation count after ten years from the citation count of the present time give worse results than when predicting the first few years. The $RMSE$ value for predicting the first year is around 10.

To see how this model makes predictions, in Figure 31 we show six randomly chosen papers from the test set and we visualize the citation count trajectory. The trajectory starts from the first five years, which are used as an input, and continues to the ten years that we predict. These citation counts are calculated from the papers that are included in our network, so note that the papers in the real world network may have a bigger citation count.

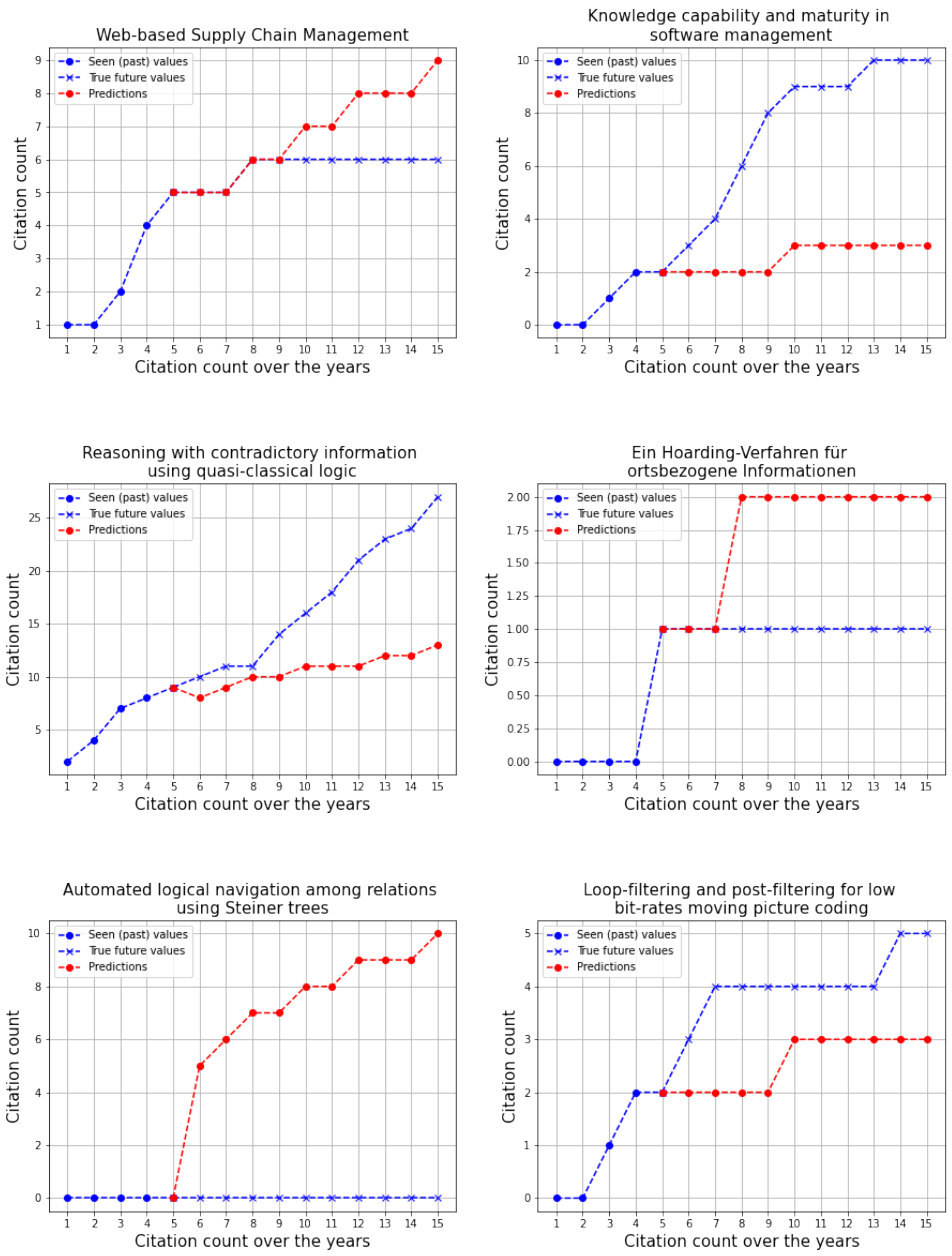


Figure 31: Predicted and ground truth citation counts for randomly selected papers using Encoder-Decoder.

The arXiv, ICASSP, INTERSPEECH, and ICIP are the most prominent publishing venues in our network. In Figure 32 we visualized the $RMSE$ for each venue, we see that we can predict citation counts of papers that were published in INTERSPEECH better than in the other three venues, whereas the predictability of citation count of papers published in ICIP is in the second place for a small margin from the first place. The citation count of papers published in ICASSP is the third most difficult to predict among these four venues, and the citation count of papers published in arXiv is the most difficult to predict.

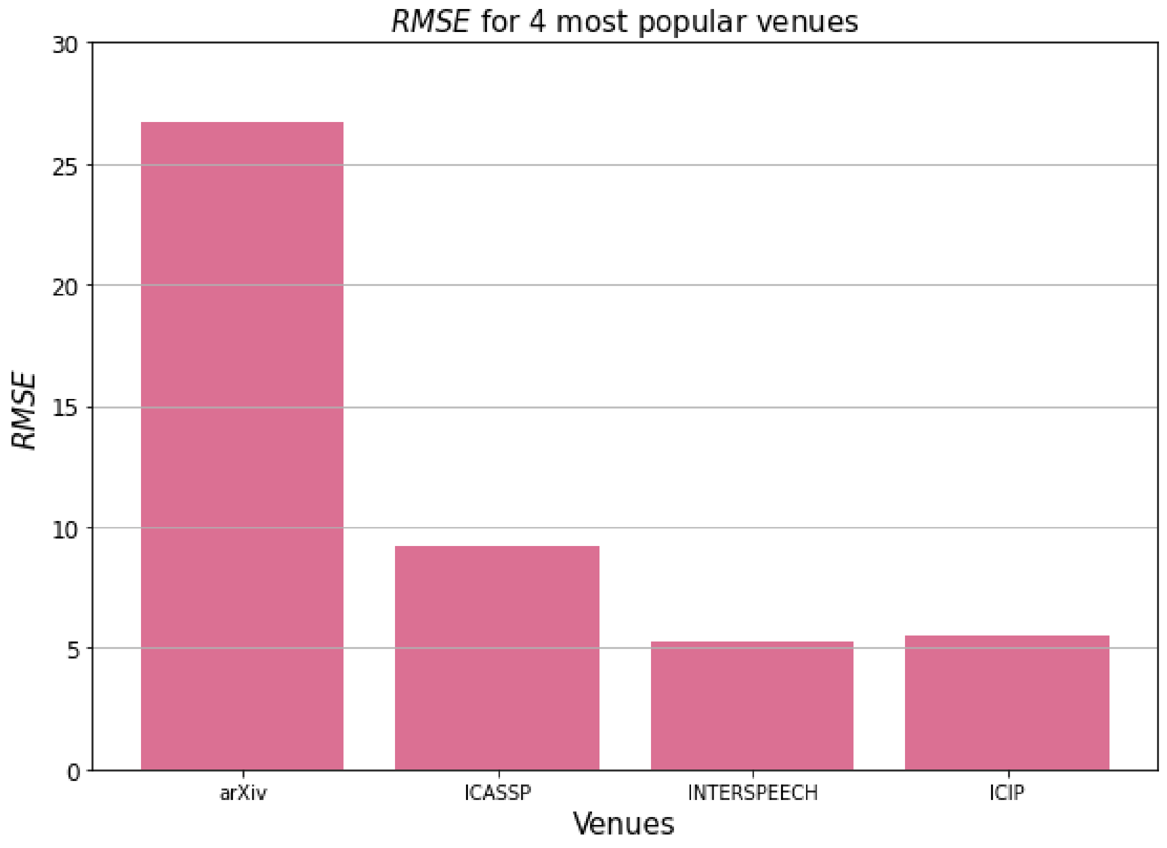


Figure 32: $RMSE$ of 4 most popular venues using Encoder-Decoder.

5.3 NETWORK BASED MODEL

As described in Section 4.3.2, we feed the network based properties to two different models: DNN and GBRT.

After training these models 10 times each to predict 10 consecutive years, we measure the accuracy of our prediction according to the measurement criteria described in Section 5.1. In Figure 33 we see the R^2-scores over ten years of predictions for both models.

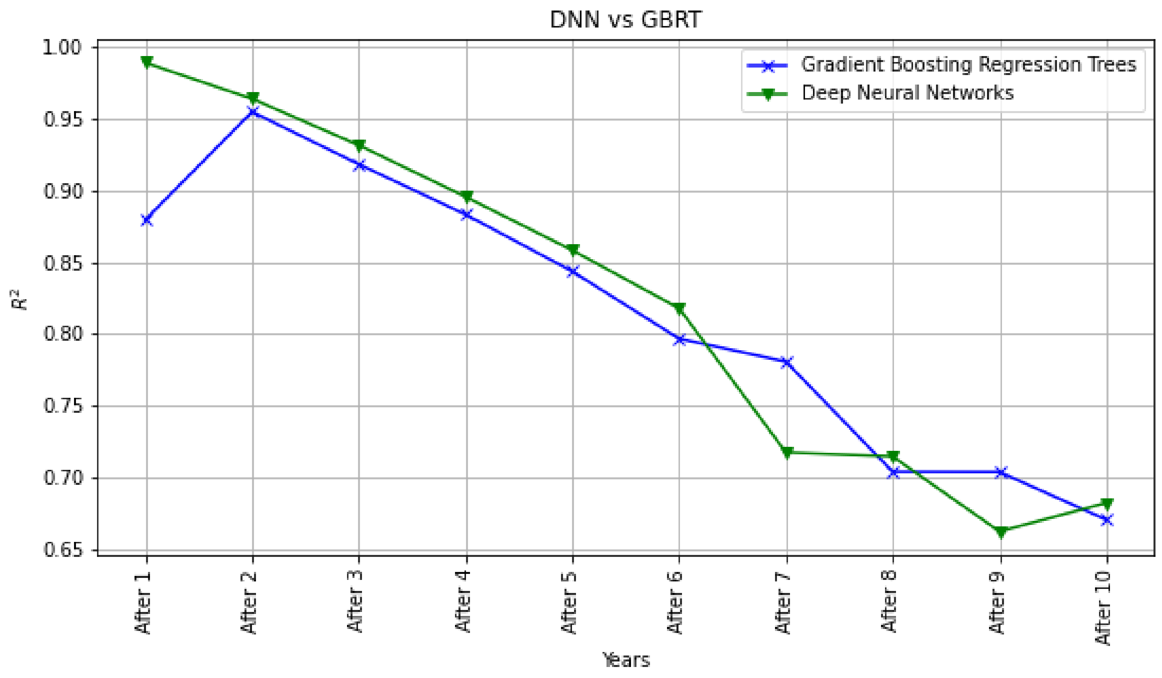


Figure 33: R^2 for DNN and GBRT models.

For all of the forecast years, the R^2-scores for both models are relatively similar to each other. DNN explain almost 98 percent of the variation in the response variable around its mean, while GBRT's R^2-score is just below 90 percent. The gap between the models narrows after the first year, and DNN marginally outperforms GBRT until the sixth year, after which R^2-scores switch positions often. Similarly to the RNN model, we see that the R^2-scores decrease over time because long-term predictions are more difficult than short-term predictions.

In Figure 34, we illustrate the $RMSE$ for both models. We can see that the R^2-score and the $RMSE$ show a similar pattern. It is easy to observe that DNN considerably outperforms GBRT in the first year, and it continues to slightly outperform it until the sixth year, after which again they interchange positions.

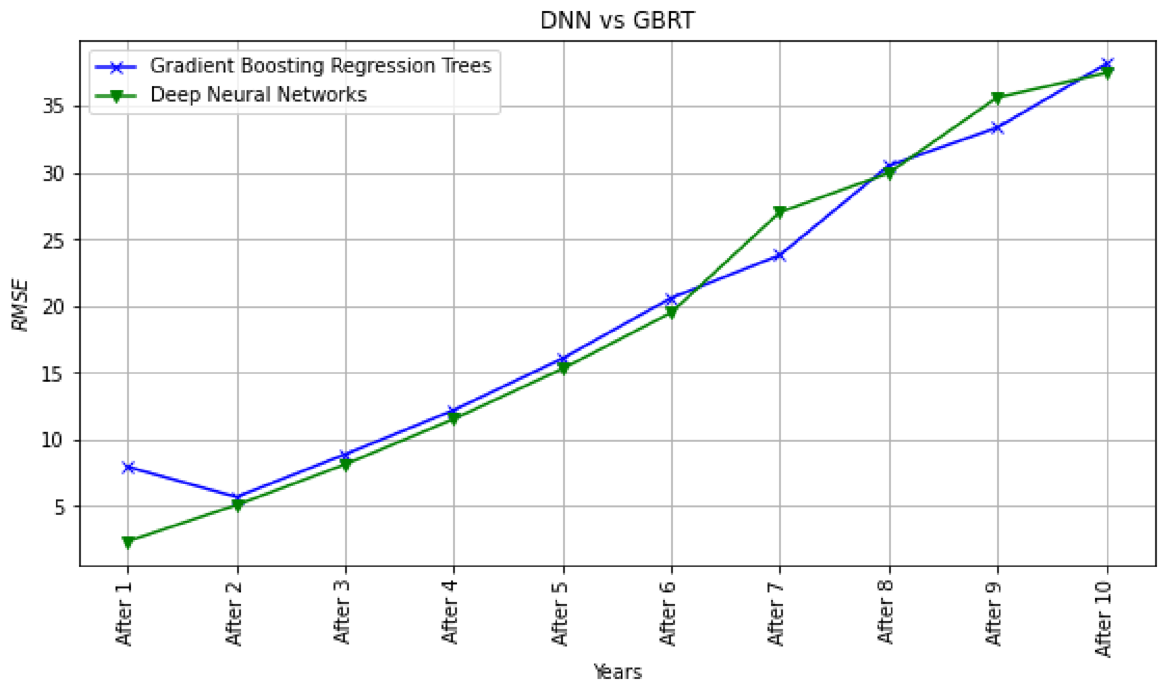
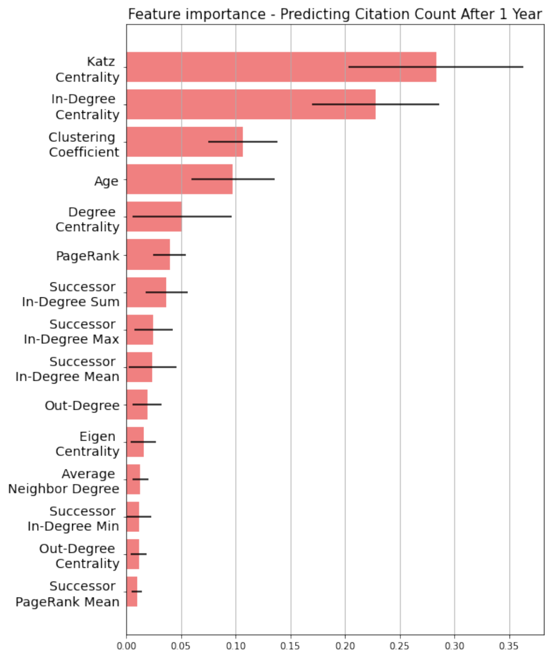


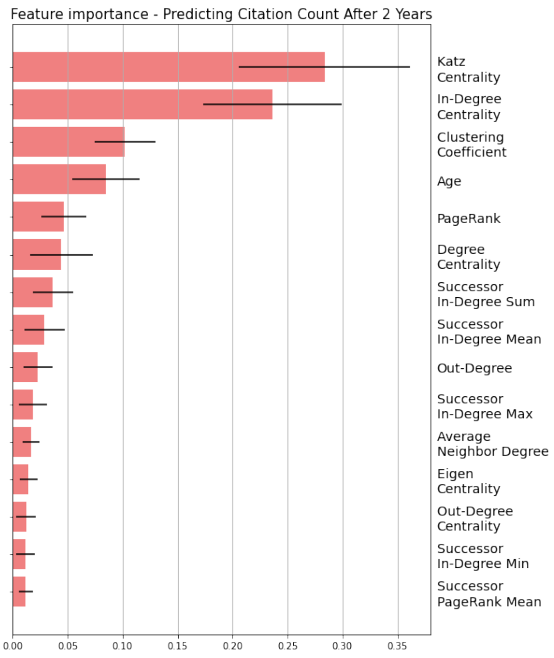
Figure 34: $RMSE$ for DNN and GBRT models.

The importance of each feature of the training data set varies. *Feature importance* is one of the most commonly used summary types for the importance of a feature. We calculate the increase in the model's prediction error after permuting a feature to determine its significance. Since the model relies on the feature for prediction, a feature is said to be *important* if shuffling its values increases the model error. This number ranges from 0 to 1, with 0 indicating that the feature is never used and 1 indicating that it perfectly predicts the response variable.

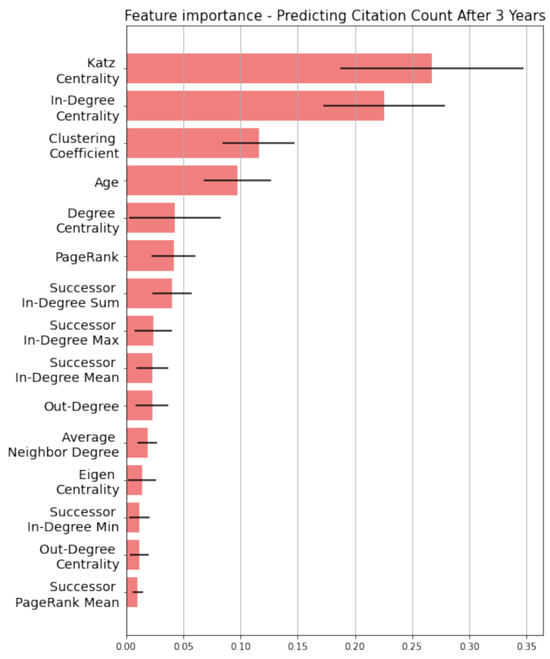
In Figures 35(a)-35(j) we present the feature importance for predicting citation count of papers throughout each training period of 10 years.



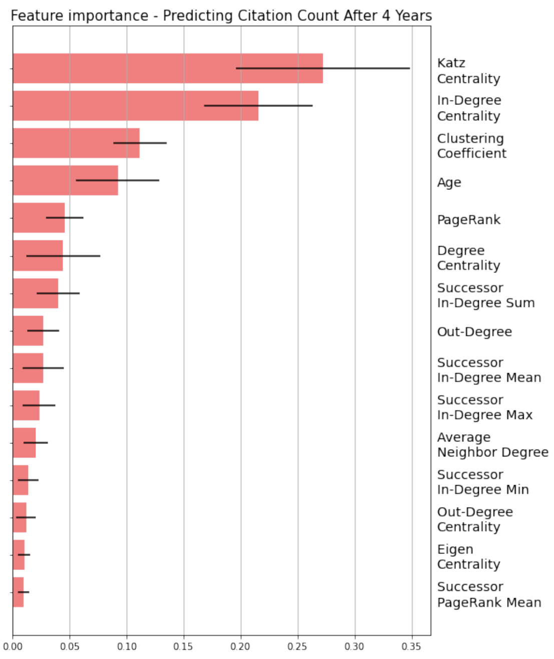
(a)



(b)

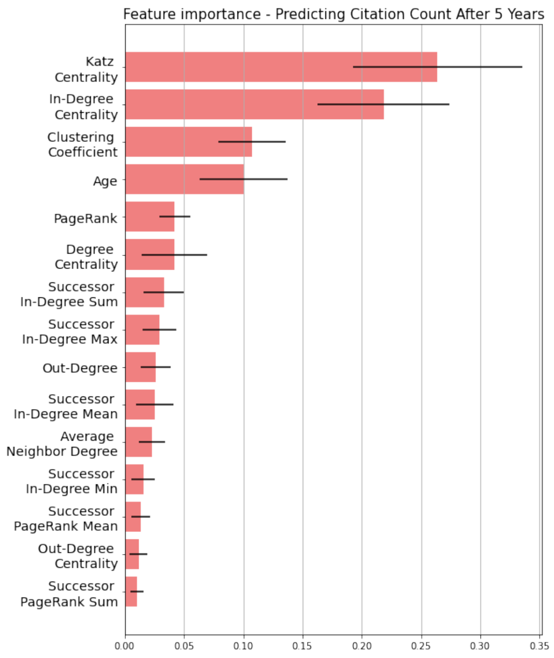


(c)

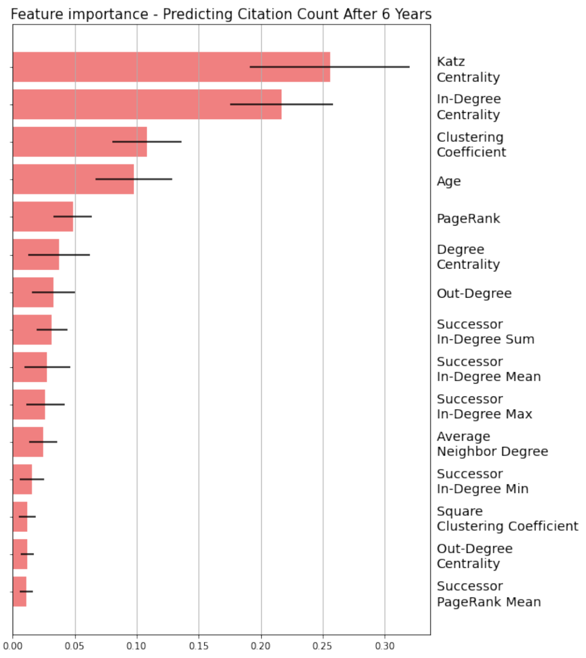


(d)

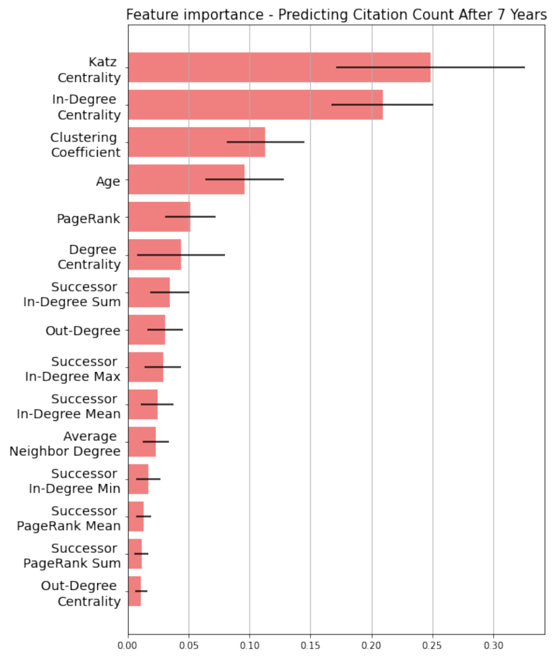
Figure 35: (a)-(d) Feature importance (without considering in-degree) for predicting citation count of years ahead.



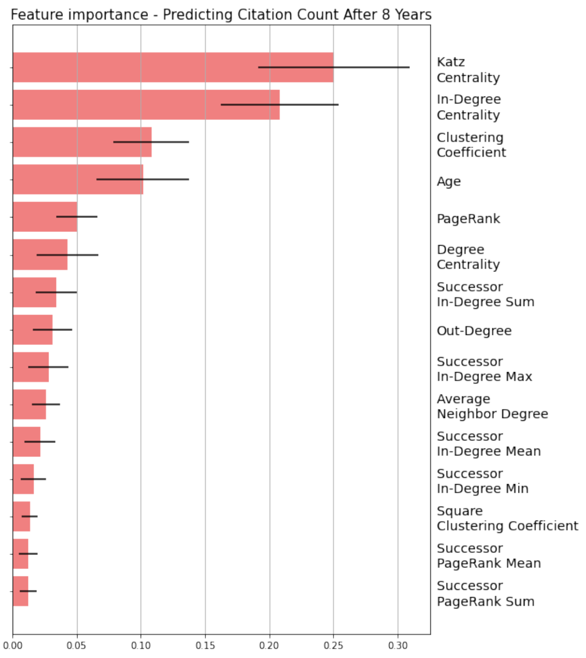
(e)



(f)



(g)



(h)

Figure 35: (e)-(h) Feature importance (without considering in-degree) for predicting citation count of years ahead.

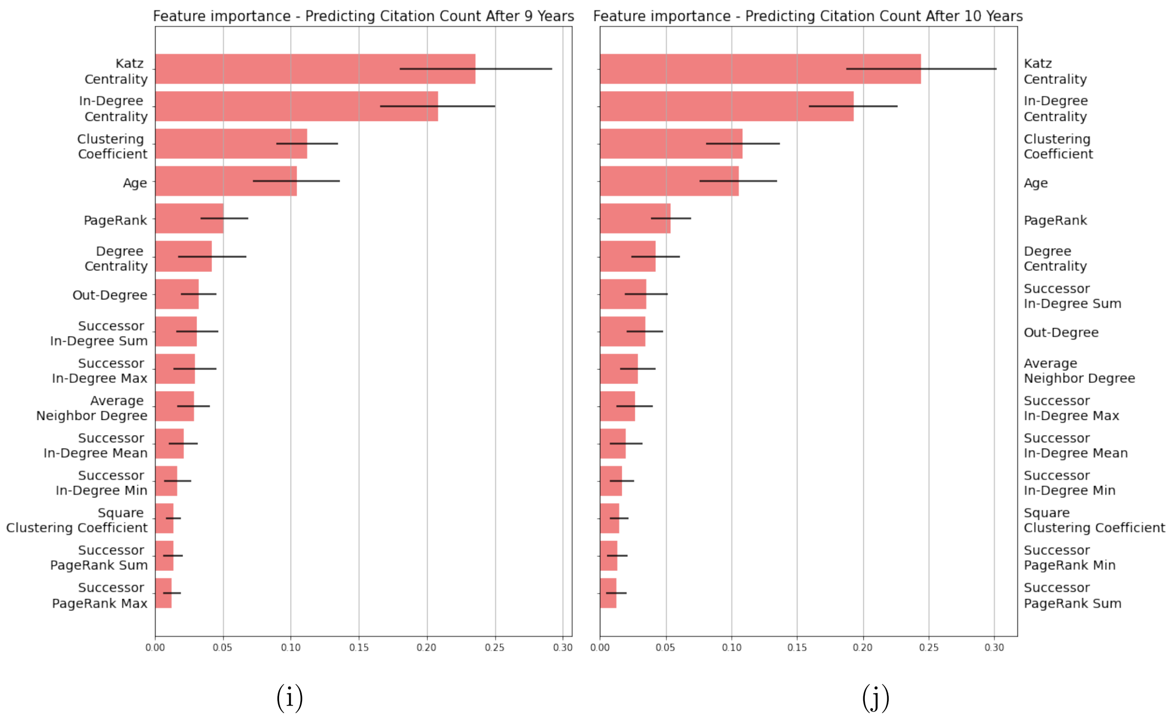


Figure 35: (i)-(j) Feature importance (without considering in-degree) for predicting citation count of years ahead.

We observe that the centrality measures such as Katz centrality, in-degree centrality, degree centrality, PageRank are some of the most important features for predicting the citation count. The clustering coefficient, age, and out-degree are also essential features over the years. Features that provide information about in-degrees of successors are also relevant, however, those that provide information on PageRank of successors are the least important out of all the other features we described above, because some of them do not appear for all years.

We forecast the citation count trajectory for the same randomly selected papers (see Figure 31) from Section 5.2 to demonstrate how our Network based predictive model performs on specific examples. The results from our model are shown in Figure 36.

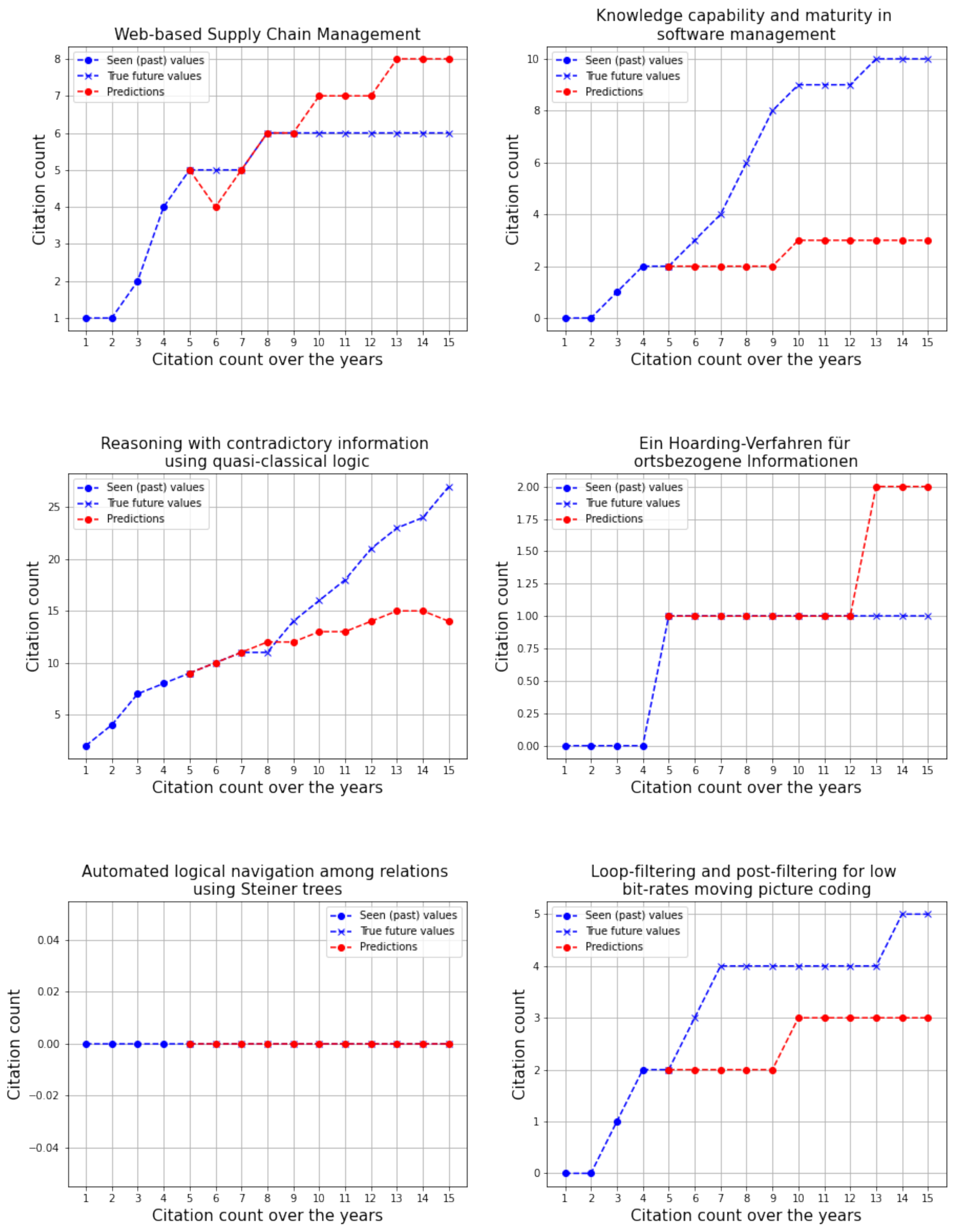


Figure 36: Predicted and ground truth citation counts for randomly selected papers using GBRT.

Figure 37 shows how well we can predict the number of citations for papers published in one of our data set's four most popular venues. The citation count of papers published in ICASSP was predicted the best, with INTERSPEECH coming in second as an immediate runner-up. The predictability of citation count of papers published in ICIP was ranked third, the predictability of citation counts of papers published in arXiv was the most difficult to predict once again. We can also observe from Figure 37 that the $RMSE$ is bigger compared to the RNN model.

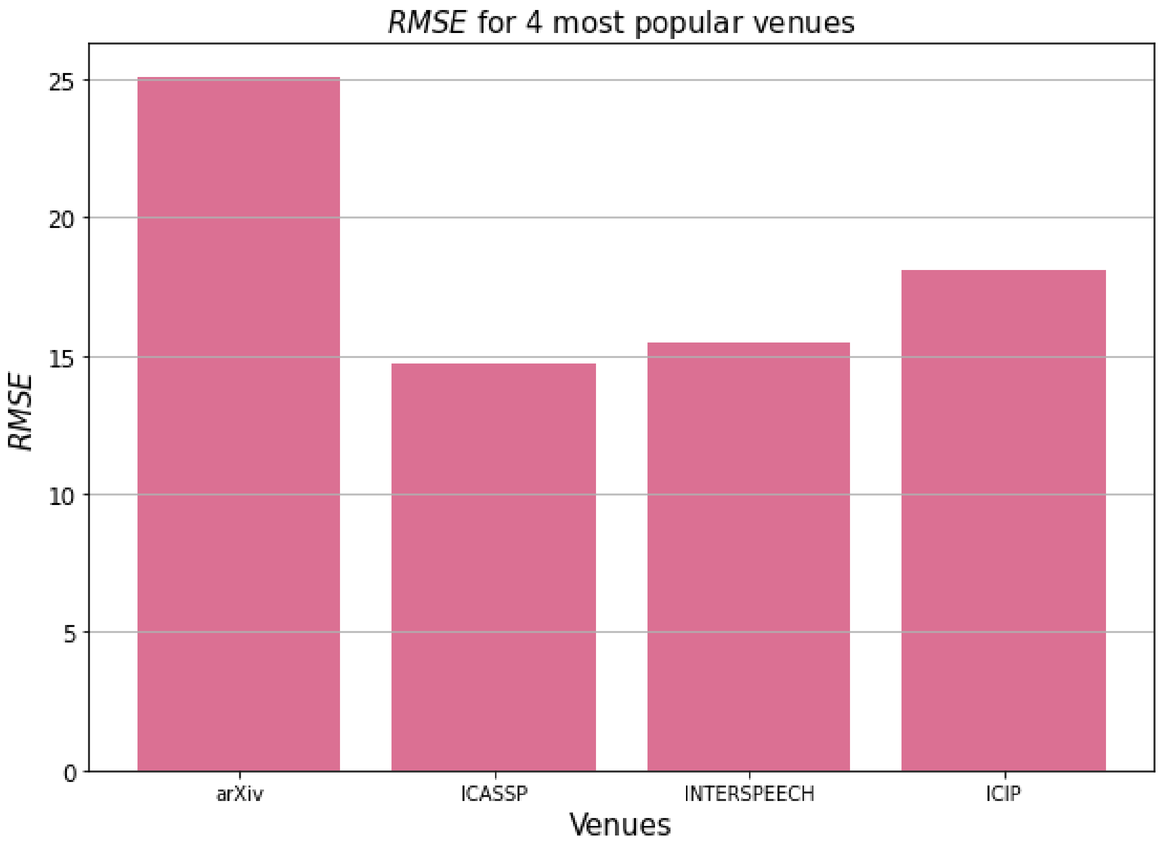


Figure 37: $RMSE$ of 4 most popular venues using GBRT.

5.4 COMPARISON OF DIFFERENT MODELS

In this section we shall compare all the different methods that we have introduced until now. In Figure 38 we have plotted the R^2-score for all the model thorough-out all the 10 years that we predict and we now proceed to discuss these results.

We note that the Network based properties perform better than properties from [56], which we consider as baseline properties, up to the fourth year, and then the baseline properties take over. This shows that Network based properties extracted from the temporal citation network for predicting the future up to 5 years are more significant than other data that needs to be gathered from other sources (i.e. author information, citation count of the previous years etc.). The latter typically requires additional work and resources and often difficult to acquire. We observe that the additional properties that can not be extracted from the citation network are necessary to predict with better accuracy after the 5th year.

Figure 38 also shows that the Encoder-Decoder model outperforms the Network based properties models and baseline model for most of the years even though it uses less features. This is due to the architecture of the model, since for each year that passes, the model also sends some information about the predicted output of the previous year. The other models are trained separately for each year and do not possess any information about the previous years.

Since Network based properties give remarkable results for the first 5 years and the baseline properties perform better for the other upcoming years, we decided to combine the Network properties with the baseline features and create another model using a GBRT as the predictive algorithm. In Figure 38 we observe that after combining these two groups of features, this model outperforms all the other models, and especially in the last three years the Encoder-Decoder model marginally exceeds it.

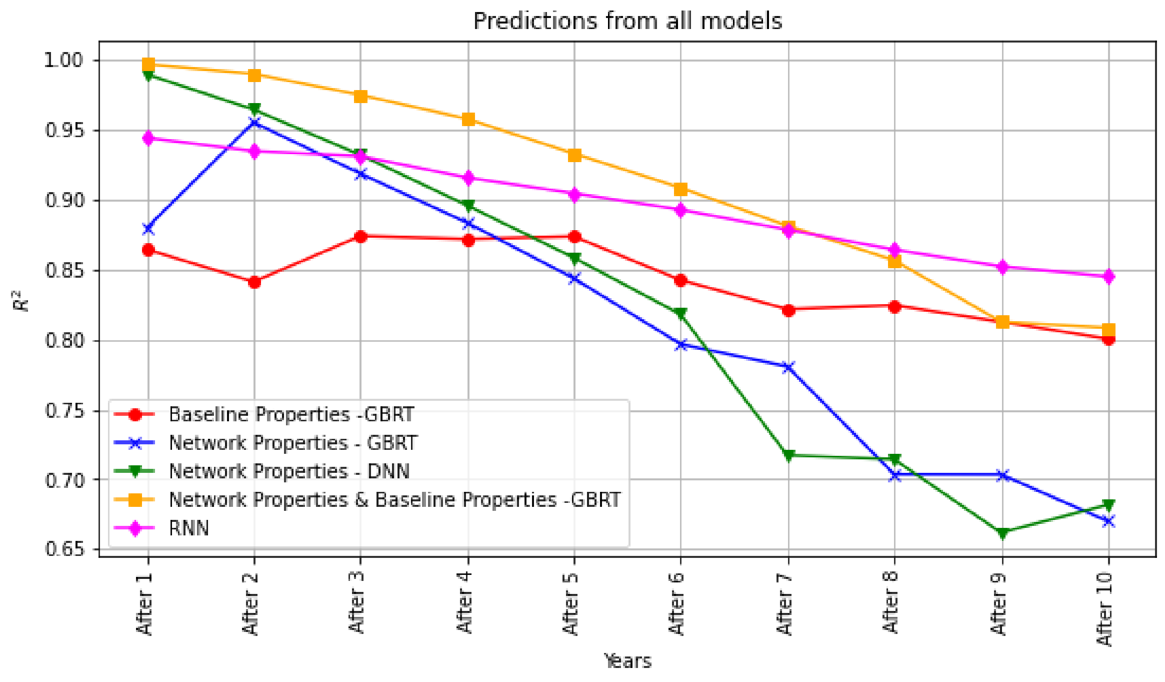


Figure 38: R^2 all the models.

Since the combined model performs better by using Network based features and baseline features, we plot the feature importance values of 15 most important features in Figure 39 and Figure 40. The baseline features are explained in Table 7, whereas the Network properties are described in Section 4.2.2. In Figure 38 we observed that Network based features perform better than the baseline features, which matches the feature importance when predicting the first five years shown in Figure 39, which lists more important features from the Network based properties. Next, by checking the feature importance when predicting the last five years shown in Figure 40, we note that we have more features from the baseline properties as expected.

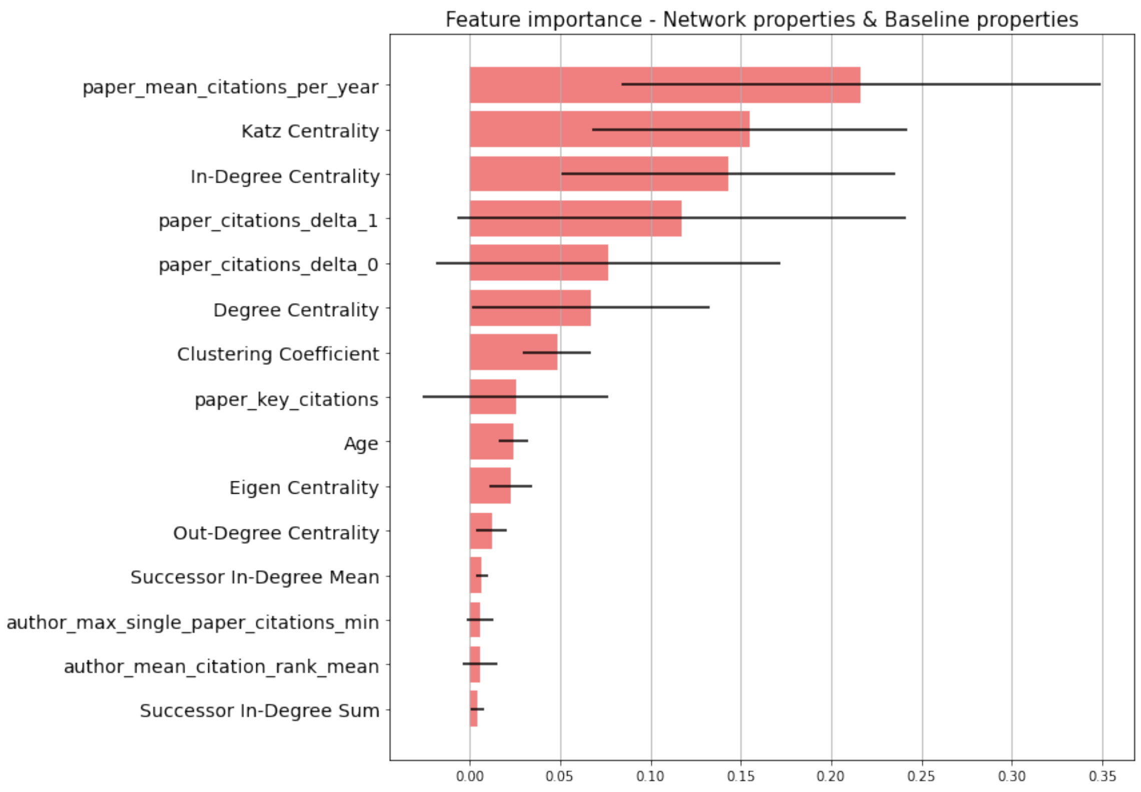


Figure 39: Feature importance of Network based and baseline properties, for predicting the first five years.

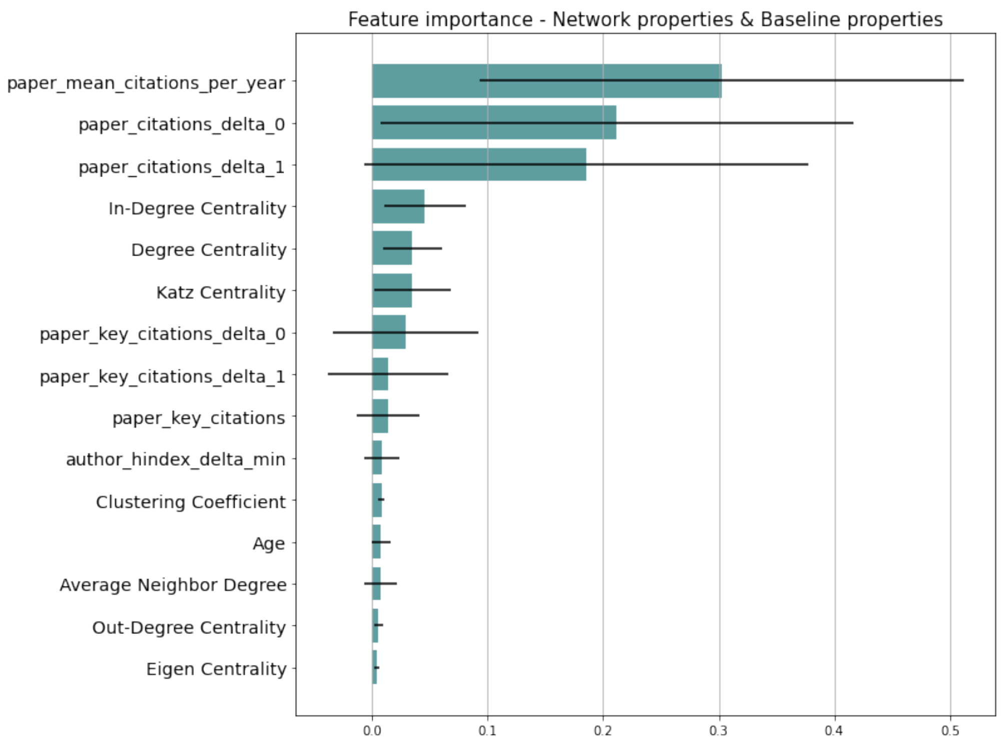


Figure 40: Feature importance of Network based and baseline properties, for predicting the last five years.

To summarize, we have demonstrated that our unique method for predicting citation counts of scientific papers outperforms state-of-the-art methods and may thus be used as a benchmark for future progress. We have shown the effectiveness of our technique by reviewing several comparison measures that compare our Network based predictive model, which employs Deep Neural Networks and Gradient Boosting Regression Trees as predictive algorithms, to the baseline model proposed by Weihs and Etzioni [56] and the Encoder-Decoder model proposed by Abrishami and Aliakbary [1].

6 CONCLUSION

Networks allow scientists to investigate patterns of connection in a variety of physical and social phenomena, from chemistry and biology to business and finance. Modern network research has revealed that most of the nodes in real networks are fairly poorly connected, while some nodes have extremely high connection [6] and many real networks have power law features. One type of such networks are citation networks.

In this thesis, we focused on predicting scientific impact of papers, which is a very important issue in academia, as shown in [1, 11, 16, 32, 38, 56]. Researchers strive for fruitful research work and thus recognizing the most impactful articles is vital.

After providing the preliminary theory regarding networks and machine learning, we analyzed important related work and addressed the problem of predicting citation counts in detail. This a well-known problem which has been studied exhaustively using various techniques. The most relevant articles for our research work have been published by Abrishami and Aliakbary [1] and Weihs and Etzioni [56]. Both studies predict citation counts, although the former utilizes short-term citation counts while the latter utilizes long-term citation counts and other features from the citation network, co-authorship network and other paper metadata.

We presented a detailed overview of the research methodology implemented in this work by delving into the data set and its relevant statistical features. We show our innovative feature engineering technique in Section 4.2.2, which is the most essential part of this thesis. We used temporal networks, which are networks that evolve over time, to tackle the topic of predicting the success of papers in the future. Moreover, we reproduced the Encoder-Decoder model and the feature engineering process proposed by Abrishami and Aliakbary [1] for this specific task.

We finally gave a thorough analysis of the results of our novel approach by comparing it with state-of-the-art methods using different measurement criteria. Furthermore, we showed that our innovative technique outperforms every other method for the first five years, and when combined with Weihs and Etzioni's [56] model, it outperforms them for all ten years.

After comparing the feature importance of the Network based features and the features used by Weihs and Etzioni, we conclude that features extracted from temporal networks produce more important features and have better correlation with the respondent variables.

As future work, we propose the implementation of a vec-to-seq Recurrent Neural

Network using the Network based properties because of the advantages that RNNs offer. RNNs process the input vector in its natural order, and as the vector is processed, a hidden memory is constructed based on the previously visited input data, effectively considering the input sequence. For longer-term predictions, this architecture may yield superior results.

We may then utilize temporal co-authorship networks to perform a similar feature extraction technique combined with node properties from temporal citation networks. This adds extra features to the data set, such as the relevance of the authors, which gives the prediction algorithm additional information and, as a result, may provide better results.

Finally, we may also extract features based on the content of the article's text to gain more insights for each publication, which will allow us to follow trending subjects and could lead to interesting outcomes.

It is noteworthy to mention that this research work is based solely on the structure of temporal networks and can thus be applied to make predictions in a variety of network-related disciplines, including but not limited to computer science, biology, and business.

7 DALJŠI POVZETEK V SLOVENSKEM JEZIKU

Številna področja, vključno s kemijo, fiziko, biologijo, zdravstvom, podjetništvom, financami in socialnimi mediji, so spoznale pomembnost *omrežij* [3, 18, 42]. V širšem smislu je *omrežje* niz *vozlišč*, katerim se pridruži *povezave*, ki predstavljajo razmerja med vozlišči [58]. Naš glavni poudarek je na omrežjih citatov in predvidevanju števila citatov člankov z uporabo omrežnega pristopa.

Magistrsko nalogo začnemo s poglavjem 2, kjer predstavimo osnovne teorije, ki vključuje potrebne definicije omrežij in strojnega učenja, ki so omenjene v tem delu. Velik poudarek namenjamo opisu omrežja za citiranje, brezobsežnega omrežja globokega nevronskega omrežja in odločitvenega drevesa, ki temelji na uporabi gradientov.

Poglavje 3 posvečamo predsavitvi koncepta citatnih omrežij in s tem povezanih del o napovedovanju znanstvenega vpliva. Navedbe znanstvenih člankov so postale morda najpogosteje uporabljena metrika znanstvenega vpliva članka [11, 14, 17]. Standardni argument je, da število citatov visoko citiranega prispevka odraža njegov vpliv in prispevek k napredku znanstvenih spoznanj.

Problem napovedovanja števila citatov za članke ima zaradi vse večjega števila objavljenih člankov številne aplikacije. Raziskovalci so zato primorani vnaprej določiti najvplivnejše aplikacije, da lahko načrtujejo svoje prihodnje raziskovalno delo. V povezavi s temo so bile obravnavane različne tehnike napovedovanja, glej [1, 11, 16, 32, 38, 56].

Po podaji onsonvih defnicij o omrežjih in strojnem učenju smo analizirali pomembna sorodna dela in podrobno obravnavali problem napovedovanja števila citatov. Najpomembnejša članka za to magistrso delo so objavili Abrishami in Aliakbary [1] ter Weihs in Etzioni [56]. V [1] Abrishami in Aliakbary predstavljata novo metodo za napovedovanje dolgoročnih citatov članka na podlagi količine citatov, prejetih v prvih nekaj letih po objavi.

Raziskovalno delo Weihsa in Etzionia [56] je za nas pomembno zaradi velike količine uporabljenih podatkov in impresivnih rezultatov. V delu se avtorja osredotočatan na značilnosti, ki jih je mogoče povzeti iz grafa zgrajenega na citatih, grafa soavtorejv in metapodatkov o članku, kot so avtorji in prizorišče.

V poglavju 4 podajamo izčrpen povzetek raziskovalnih metod, uporabljenih v tej nalogi. Upoštevamo množico podatkov, ki sta jo uporabila Weihs in Etzioni v [56]. Predlagamo novo tehniko ekstrakcije lastnosti z uporabo lastnosti iz časovnih omrežji citiranja. Našo inovativno tehniko inženirstva prikažemo v poglavju 4.2.2, ki predstavlja najpomembnejši del te magistrske naloge. Za reševanje problema napovedovanja uspeha člankov smo uporabili časovna omrežja, ki so omrežja, ki se sčasoma spreminjajo.

Implementirana sta dva različna modela, ki temeljita na globokih nevronskih omrežjih in odločitvenih drevesih, ki temeljijo na uporabi gradienta, za napovedovanje števila citatov do deset let vnaprej, z uporabo povzetih podatkov. Ponovno ustvarimo modela opisana s strani Abrishamia in Aliakbarya [1] ter Weihsa in Etzionia [56], ter uporabimo različne primerjalne metode za merjenje učinkovitosti posameznega modela. Nato pokažemo, da je naš novi model v prvih petih letih bistveno boljši od obeh že znanih. Poleg tega, v skupnem modelu, uporabimo značilnosti omrežij v povezavi z lastnostmi Weihsa in Etzionia [56], da preučimo primerjavo z drugimi modeli. Slednji model v celotnem časovnem obdobju prekaša vse ostale modele.

Ker smo se v magistrski nalogi osredotočili le na strukturo časovnih omrežij citiranja, je ta inovativen pristop uporaben za napovedovanje v različnih omrežjih, kot so socialna omrežja, svetovni splet, omrežja za interakcijo proteinov in beljakovin, letalska omrežja, medbančna plačilna omrežja itd.

8 REFERENCES

[1] A. Abrishami and S. Aliakbary, *Predicting citation counts based on deep neural network learning techniques*, Journal of Informetrics, 13 (2019), pp. 485–499. *(Cited on pages 1, 2, 28, 30, 31, 39, 40, 48, 69, 70, 72 in 73.)*

[2] D. E. Acuna and K. P. Kording, *Predicting scientific success*, Nature, 489 (2012), pp. 201–202. *(Cited on page 27.)*

[3] B. Albert-Làszlò and P. Màrton, *Network science*, Cambridge University Press, 2017. *(Cited on pages 1, 4 in 72.)*

[4] S. Ayaz, N. Masood, and M. A. Islam, *Predicting scientific impact baed on h-index*, Scientometrics, 114 (2018), pp. 993–1010. *(Cited on page 27.)*

[5] A.-L. Barabási and R. Albert, *Emergence of scaling in random networks*, Science, 286 (1999), pp. 509–512. *(Cited on pages 1 in 43.)*

[6] A.-L. Barabási and E. Bonabeau, *Scale-free networks*, 288 (2003), pp. 60–69. *(Cited on pages 1 in 70.)*

[7] P. Bonacich, *Power and centrality: A family of measures*, American Journal of Sociology, 92 (1987), p. 1170–1182. *(Cited on page 44.)*

[8] S. Bowles, *Technical change and the profit rate: a simple proof of the Okishio theorem*, Cambridge Journal of Economics, 5 (1981), pp. 183–186. *(Cited on page 44.)*

[9] T. Bray, *The javascript object notation (json) data interchange format*, (2014). *(Cited on page 39.)*

[10] C. C., D. D., and G. A., *Estimating number of citations using author reputation*, String Processing and Infromation Retrival, (2007), pp. 107–117. *(Cited on page 2.)*

[11] X. Cao, Y. Chen, and K. Ray Liu, *A data analytic approach to quantifying scientific impact*, Journal of Informetrics, 10 (2016), pp. 471–484. *(Cited on pages 1, 2, 27, 28, 29, 70 in 72.)*

[12] F. Chollet et al., *Keras*. `https://github.com/fchollet/keras`, 2015. *(Cited on page 48.)*

[13] C. Cortes and V. Vapnik, *Support-vector networks*, Machine learning, 20 (1995), pp. 273–297. *(Cited on page 20.)*

[14] A. Daud, W. Ahmed, T. Amjad, J. A. Nasir, N. R. Aljohani, R. A. Abbasi, and I. Ahmad, *Who will cite you back? Reciprocal link prediction in citation networks*, Library Hi Tech, 35 (2017), pp. 509–520. *(Cited on pages 1, 28 in 72.)*

[15] Y. Dong, R. A. Johnson, and N. V. Chawla, *Will this paper increase your h-index? scientific impact prediction*, WSDM, (2015), pp. 149–158. *(Cited on page 27.)*

[16] Y. Dong, R. A. Johnson, and N. V. Chawla, *Can scientific impact be predicted?*, IEEE Transactions on Big Data, 2 (2016), p. 18–30. *(Cited on pages 2, 27, 29, 70 in 72.)*

[17] L. Egghe and R. Rousseau, *Introduction to Informetrics: Quantitative Methods in Library, Documentation and Information Science*, vol. 61, 1990. *(Cited on pages 1, 27 in 72.)*

[18] F. Emmert-Streib and M. Dehmer, *Network science: From chemistry to digital society*, Frontiers for Young Minds, 7 (2019). *(Cited on pages 1 in 72.)*

[19] P. Erdös and A. Rènyi, *On random graphs*, I. Publicationes Mathematicae (Debrecen), 6 (1959), pp. 290–297. *(Cited on page 8.)*

[20] ——, *On evolution of random graphs*, Publ. Math. Inst. Hung. Acad. Sci., 5 (1960), pp. 17–16. *(Cited on page 8.)*

[21] M. Franceschet, *Pagerank*, Communications of the ACM, 54 (2011), pp. 92–101. *(Cited on page 44.)*

[22] J. H. Friedman, *Greedy function approximation: A gradient boosting machine.*, The Annals of Statistics, 29 (2001). *(Cited on page 14.)*

[23] F. Galton, *Regression towards mediocrity in hereditary stature.*, The Journal of the Anthropological Institute of Great Britain and Ireland, 15 (1886), p. 246. *(Cited on page 20.)*

[24] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, OReilly, 2020. *(Cited on page 4.)*

[25] K. I. Goh, B. Kahng, and D. Kim, *Universal behavior of load distribution in scale-free networks*, Physical Review Letters, 87 (2001). *(Cited on page 44.)*

[26] A. Hagberg, P. Swart, and D. S Chult, *Exploring network structure, dynamics, and function using networkx*, tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008. *(Cited on page 39.)*

[27] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'io, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, *Array programming with NumPy*, Nature, 585 (2020), pp. 357–362. *(Cited on page 39.)*

[28] D. O. Hebb, *The organization of behavior: A neuropsychological theory*, Science Education, 34 (1950), pp. 336–337. *(Cited on page 21.)*

[29] J. D. Hunter, *Matplotlib: A 2d graphics environment*, Computing in Science & Engineering, 9 (2007), pp. 90–95. *(Cited on page 40.)*

[30] S. W. Jones, *Gradient booseted decision trees*. `https://www.simonwardjones.co.uk/posts/gradient_boosted_decision_trees/`, 2020. *(Cited on page 15.)*

[31] F. Karinthy, *Minden másképpen van*, Athenaeum, 1929. *(Cited on page 8.)*

[32] C. M. Ketcham, *Predicting impact factor one year in advance*, Laboratory Investigation, 87 (2007), pp. 520–526. *(Cited on pages 1, 2, 27, 70 in 72.)*

[33] C. T. Lamb, S. L. Gilbert, and A. T. Ford, *Tweet success? scientific communication correlates with increased citations in ecology and conservation*, PeerJ, (2018), p. e4564. *(Cited on pages 2 in 29.)*

[34] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*, Cambridge University Press, 2014. *(Cited on page 12.)*

[35] P. G. Lind, M. C. González, and H. J. Herrmann, *Cycles and clustering in bipartite networks*, Physical Review E, 72 (2005). *(Cited on page 45.)*

[36] P. V. Marsden, *Network Analysis*, Elsevier, 2005. *(Cited on page 44.)*

[37] W. S. Mcculloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, The Bulletin of Mathematical Biophysics, 5 (1943), pp. 115–133. *(Cited on page 19.)*

[38] D. Mcnamara, P. Wong, P. Christen, and K. S. Ng, *Predicting high impact academic papers using citation network features*, Lecture Notes in Computer Science Trends and Applications in Knowledge Discovery and Data Mining, (2013), pp. 14–25. *(Cited on pages 1, 2, 28, 29, 70 in 72.)*

[39] S. Milgram, *The small world problem*, Psychology Today, Ziff-Davis Publishing Company, 2 (1967). *(Cited on page 8.)*

[40] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969. *(Cited on page 21.)*

[41] A. C. Müller. and S. Guido, *Introduction to machine learning with Python: a guide for data scientists*, OReilly Media, 2018. *(Cited on page 4.)*

[42] M. E. J. Newman, *The structure and function of complex networks*, SIAM Review, 45 (2003), pp. 167–256. *(Cited on pages 1, 4, 27 in 72.)*

[43] M. E. J. Newman, *Prediction of highly cited papers*, EPL (Europhysics Letters), 105 (2014), p. 28002. *(Cited on pages 27 in 28.)*

[44] D. J. D. S. Price, *Networks of scientific papers*, Science, 149 (1965), pp. 510–515. *(Cited on pages 1, 8, 27 in 34.)*

[45] M. Rocha-e Silva, *Journal Impact Factors for the year-after the next can be objectively predicted*, MedicalExpress, 3 (2016). *(Cited on page 27.)*

[46] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain.*, Psychological Review, 65 (1958), pp. 386–408. *(Cited on page 20.)*

[47] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747, (2016). *(Cited on page 21.)*

[48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, (1985). *(Cited on page 22.)*

[49] J. Saramäki, M. Kivelä, J.-P. Onnela, K. Kaski, and J. Kertész, *Generalizations of the clustering coefficient to weighted complex networks*, Physical Review E, 75 (2007). *(Cited on page 45.)*

[50] E. Sarigöl, R. Pfitzner, I. Scholtes, A. Garas, and F. Schweitzer, *Predicting scientific success based on coauthorship network*, EPJ Data Science, 3 (2014). *(Cited on page 27.)*

[51] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014. *(Cited on pages 11, 12 in 15.)*

[52] O. Sporns, *Networks of the Brain*, The MIT Press, 2010. *(Cited on page 18.)*

[53] C. Stangor and J. Walinga, *Introduction to psychology*, BCcampus, BC Open Textbook Project, 2019. *(Cited on pages VI in 19.)*

[54] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*, CreateSpace, Scotts Valley, CA, 2009. *(Cited on page 39.)*

[55] D. J. Watts and S. H. Strogatz, *Collective dynamics of 'small-world' networks*, Nature, 393 (1998), pp. 440–442. *(Cited on page 8.)*

[56] L. Weihs and O. Etzioni, *Learning to predict citation-based impact measures*, 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL), (2017). *(Cited on pages 1, 2, 3, 28, 29, 30, 31, 32, 33, 34, 37, 40, 47, 66, 69, 70, 72 in 73.)*

[57] Wes McKinney, *Data Structures for Statistical Computing in Python*, in Proceedings of the 9th Python in Science Conference, Stéfan van der Walt and Jarrod Millman, eds., 2010, pp. 56 – 61. *(Cited on page 39.)*

[58] D. B. West, *Introduction to Graph Theory*, Pearson Education, Inc., 2001. *(Cited on pages 1, 4 in 72.)*

[59] D. H. Wolpert, *The supervised learning no-free-lunch theorems*, Soft Computing and Industry, (2002), pp. 25–42. *(Cited on page 13.)*

[60] X. Wu, Q. Fu, and R. Rousseau, *On indexing in the web of science and predicting journal impact factor*, J Zhejiang Univ Sci B, 9 (2008), pp. 582–590. *(Cited on page 27.)*

[61] W. W. Zachary, *An information flow model for conflict and fission in small groups*, Journal of Anthropological Research, 33 (1977), pp. 452–473. *(Cited on page 9.)*

Appendices

APPENDIX A Network based feature extraction

```
# imported packages
import pandas as pd
import numpy as np
import networkx as nx

# imported functions
from networkx.algorithms.centrality import degree_centrality
from networkx.algorithms.centrality import in_degree_centrality
from networkx.algorithms.centrality import out_degree_centrality
from networkx.algorithms.centrality import eigenvector_centrality
from networkx.algorithms.centrality import katz_centrality
from networkx.algorithms.assortativity import average_neighbor_degree
from networkx.algorithms.link_analysis import pagerank
from networkx.algorithms.cluster import clustering
from networkx.algorithms.cluster import square_clustering

# helper function to extract in-degree and PageRank of successors
def pagerank_indeg_sum_mean_min_max(G, dataframe, paper):
    neigh_list = list(G.neighbors(paper))
    neigh_df = dataframe[dataframe.id.isin(neigh_list)]
    # PageRank of successors
    pg_sum_neigh = neigh_df.PageRank.sum()
    pg_mean_neigh = np.mean(neigh_df.PageRank)
    pg_min_neigh = np.min(neigh_df.PageRank)
    pg_max_neigh = np.max(neigh_df.PageRank)
    # InDegree of successors
    indeg_sum_neigh = neigh_df.InDegree.sum()
    indeg_mean_neigh = np.mean(neigh_df.InDegree)
    indeg_min_neigh = np.min(neigh_df.InDegree)
    indeg_max_neigh = np.max(neigh_df.InDegree)
    return pg_sum_neigh, pg_mean_neigh, pg_min_neigh, pg_max_neigh,\
        indeg_sum_neigh, indeg_mean_neigh, indeg_min_neigh, indeg_max_neigh

# generate a random direted network
G = nx.gnm_random_graph(10, 25, directed=True)

# dataframe to save the extracted features
processed_df = pd.DataFrame({'id': [i for i in range(10)]})

# Degree Centrality
DegreeCentrality = degree_centrality(G)
DegreeCentrality_df = pd.DataFrame(DegreeCentrality.items(), columns=['id', 'DegreeCentrality'])
processed_df = pd.merge(processed_df, DegreeCentrality_df, how='left', on='id')

# In Degree Centrality
InDegreeCentrality = in_degree_centrality(G)
InDegreeCentrality_df = pd.DataFrame(InDegreeCentrality.items(), columns=['id', 'InDegreeCentrality'
    ])
processed_df = pd.merge(processed_df, InDegreeCentrality_df, how='left', on='id')

# Out Degree Centrality
OutDegreeCentrality = out_degree_centrality(G)
OutDegreeCentrality_df = pd.DataFrame(OutDegreeCentrality.items(), columns=['id', '
    OutDegreeCentrality'])
processed_df = pd.merge(processed_df, OutDegreeCentrality_df, how='left', on='id')


# Eigen Centrality
EigenCentrality = eigenvector_centrality(G, max_iter=1000)
EigenCentrality_df = pd.DataFrame(EigenCentrality.items(), columns=['id', 'EigenCentrality'])
processed_df = pd.merge(processed_df, EigenCentrality_df, how='left', on='id')
```

```
# Katz Centrality
KatzCentrality = katz_centrality(G)
KatzCentrality_df = pd.DataFrame(KatzCentrality.items(), columns=['id', 'KatzCentrality'])
processed_df = pd.merge(processed_df, KatzCentrality_df, how='left', on='id')

# Average Neighbor Degree
AverageNeigborDegree = average_neighbor_degree(G)
AverageNeigborDegree_df = pd.DataFrame(AverageNeigborDegree.items(), columns=['id', '
     AverageNeigborDegree'])
processed_df = pd.merge(processed_df, AverageNeigborDegree_df, how='left', on='id')

# PageRank
PageRank = pagerank(G)
PageRank_df = pd.DataFrame(PageRank.items(), columns=['id', 'PageRank'])
processed_df = pd.merge(processed_df, PageRank_df, how='left', on='id')

# Clustering Coefficient
ClusteringCoefficient = clustering(G)
ClusteringCoefficient_df = pd.DataFrame(ClusteringCoefficient.items(), columns=['id', '
     ClusteringCoefficient'])
processed_df = pd.merge(processed_df, ClusteringCoefficient_df, how='left', on='id')

# Square Clustering Coefficient
SquareClusteringCoefficient = square_clustering(G)
SquareClusteringCoefficient_df = pd.DataFrame(SquareClusteringCoefficient.items(),
    columns=['id', 'SquareClusteringCoefficient'])
processed_df = pd.merge(processed_df, SquareClusteringCoefficient_df, how='left', on='id')

# In Degree
InDegree = dict(G.in_degree())
InDegree_df = pd.DataFrame(InDegree.items(), columns=['id', 'InDegree'])
processed_df = pd.merge(processed_df, InDegree_df, how='left', on='id')

# Out Degree
OutDegree = dict(G.out_degree())
OutDegree_df = pd.DataFrame(OutDegree.items(), columns=['id', 'OutDegree'])
processed_df = pd.merge(processed_df, OutDegree_df, how='left', on='id')

# Successors: PageRank and InDegree
processed_df['SuccesorPageRankSum'] = np.zeros(processed_df.shape[0])
processed_df['SuccesorPageRankMean'] = np.zeros(processed_df.shape[0])
processed_df['SuccesorPageRankMin'] = np.zeros(processed_df.shape[0])
processed_df['SuccesorPageRankMax'] = np.zeros(processed_df.shape[0])
processed_df['SuccesorInDegreeSum'] = np.zeros(processed_df.shape[0])
processed_df['SuccesorInDegreeMean'] = np.zeros(processed_df.shape[0])
processed_df['SuccesorInDegreeMin'] = np.zeros(processed_df.shape[0])
processed_df['SuccesorInDegreeMax'] = np.zeros(processed_df.shape[0])
for paper in range(processed_df.shape[0]):
    SuccesorPageRankSum, SuccesorPageRankMean, SuccesorPageRankMin, SuccesorPageRankMax,\
    SuccesorInDegreeSum, SuccesorInDegreeMean, SuccesorInDegreeMin, SuccesorInDegreeMax = \
                pagerank_indeg_sum_mean_min_max(G, processed_df, processed_df.id[paper])
    processed_df.loc[paper,'SuccesorPageRankSum'] = SuccesorPageRankSum
    processed_df.loc[paper,'SuccesorPageRankMean'] = SuccesorPageRankMean
    processed_df.loc[paper,'SuccesorPageRankMin'] = SuccesorPageRankMin
    processed_df.loc[paper,'SuccesorPageRankMax'] = SuccesorPageRankMax
    processed_df.loc[paper,'SuccesorInDegreeSum'] = SuccesorInDegreeSum
    processed_df.loc[paper,'SuccesorInDegreeMean'] = SuccesorInDegreeMean
    processed_df.loc[paper,'SuccesorInDegreeMin'] = SuccesorInDegreeMin
    processed_df.loc[paper,'SuccesorInDegreeMax'] = SuccesorInDegreeMax

# save the extracted data
processed_df.to_csv('extracted_features.csv', index=False)
```