Distributed Link Prediction in Large Scale Graphs using Apache Spark

Master’s thesis of Anastasios Theodosiou

Supervisor: Associate Prof. Apostolos N. Papadopoulos

Thessaloniki, February 2019
Abstract

Online social networks (OSNs) such as Facebook, Instagram, Twitter, and LinkedIn are aware of high acceptance since they enable users to share digital content (links, pictures, videos), express or share opinions and expand their social circle, by making new friends. All these kinds of interactions that users participate in lead to the evolution and expansion of social networks over time. OSNs support users, providing them with friend recommendations based on the existing explicit friendship network but also to their preferences resulting from their interaction with the net which they gradually build. Link prediction methods try to predict the likelihood of a future connection between two nodes in a given network. This can be used in biological networks to infer protein-protein interactions or to suggest possible friends to a user in an OSN. In e-commerce it can help to build recommendation systems such as the "people who bought this may also buy" feature, e.g., on Amazon and in the security domain link prediction can help to identify hidden groups of people who use potential violence and criminals. Due to the massive amounts of data that is collected today, the need for scalable approaches arises to this problem. The purpose of this diploma thesis is to experiment and use various techniques of machine learning, both supervised and unsupervised, to predict links to a network of academic papers using document similarity metrics based on the characteristics of the nodes but also other structural features, based on the network. Experimentation and implementation of the application took place using Apache Spark to manage the large data volume using the Scala programming language.

**Keywords:** Link prediction, Data mining, Machine Learning, Apache Spark, Graphs, Online Social Networks, Recommender Systems
Περίληψη

Τα κοινωνικά δίκτυα όπως το Facebook, Instagram, Twitter αλλά και το LinkedIn, γνωρίζουν μεγάλη ανάπτυξη και αποδοχή από τους χρήστες, καθώς τους δίνουν την δυνατότητα να διαμοιράζονται ψηφιακό περιεχόμενο (συνδέσμους, φωτογραφίες, βίντεο), να εκφράζουν ή να μοιράζονται τη γνώμη τους και να επεκτείνουν τον κοινωνικό τους κύκλο, κάνοντας νέους φίλους. Όλα αυτά τα είδη αλληλεπιδράσεων που οι χρήστες συμμετέχουν, οδηγούν στην εξέλιξη και επέκταση των κοινωνικών δικτύων, με την πάροδο του χρόνου. Τα κοινωνικά δίκτυα, υποστηρίζουν τους χρήστες παρέχοντάς τους προτάσεις για νέους συνδέσμους φιλίας, βασισμένες στο υπάρχον δίκτυό τους αλλά και σε προτιμήσεις τους που προκύπτουν από την αλληλεπίδρασή τους με το δίκτυο που χτίζουν σταδιακά. Οι μέθοδοι πρόβλεψης συνδέσμων, προσπαθούν να προβλέψουν την πιθανότητα μιας μελλοντικής σύνδεσης μεταξύ δύο κόμβων σε ένα δεδομένο δίκτυο. Οι προβλέψεις αυτές μπορούν να χρησιμοποιηθούν σε βιολογικά δίκτυα για να συναχθούν αλληλεπιδράσεις πρωτεϊνής-πρωτεϊνής ή να υποδειχθούν πιθανοί φίλοι σε ένα χρήστη ενός κοινωνικού δικτύου. Στο ηλεκτρονικό εμπόριο ακόμα, μπορούν να βοηθήσουν στην ανάπτυξη συστημάτων σύστασης σε χρήστες που πραγματοποιούν μια ηλεκτρονική αγορά και να τους υποδείξουν ένα παρόμοιο προϊόν είτε ακόμα και ένα παρόμοιο προϊόν είτε ακόμα και μια συνδυαστική πώληση όπως για παράδειγμα γίνεται στο Amazon. Ακόμα και στην περιοχή της ασφάλειας, η πρόβλεψη συνδέσμων μπορεί να βοηθήσει στην αναγνώριση κρυφών ομάδων ανθρώπων, οι οποίοι χρησιμοποιούν πιθανή βία ή και εγκληματίες. Λόγω του τεράστιου όγκου δεδομένων που συλλέγεται σήμερα, δημιουργείται η ανάγκη για κλιμακωμένες προσεγγίσεις στο πρόβλημα της πρόβλεψης συνδέσμων. Αντικείμενο αυτής της διπλωματικής, είναι η πειραματισμός και η χρήση διαφόρων τεχνικών μηχανικής μάθησης, τόσο επιβεβλήμενες όσο και μη, με στόχο την πρόβλεψη συνδέσμων σε ένα δίκτυο ακαδημαϊκών εγγράφων (paper) με την χρήση του Apache Spark για την διαχείριση του μεγάλου όγκου δεδομένων και την Scala, ως γλώσσα προγραμματισμού.

Λέξεις κλειδιά: Πρόβλεψη συνδέσμων, εξόρυξη δεδομένων, μηχανική μάθηση, γράφοι, κοινωνικά δίκτυα, συστήματα συστάσεων, Apache Spark, Scala
Acknowledgements

With great satisfaction and pride, I present my thesis on *link prediction for large scale graphs*, for partial fulfillment of my Master of science degree in Internet and Web Computing from the Aristotle University of Thessaloniki and the Department of Informatics.

I am thankful to Prof. Apostolos N. Papadopoulos for being the best guide and advisor for this work in every field I have taken to complete my requirements. His ideas and inspirations have helped and gave me the opportunity to complete this work. Without the presence of him, I may never have tasted the flavor of Big Data’s and Artificial Intelligence’s world.

I would also like to dedicate this project to my family, who always stood by me in every point of my life.

So, I am thankful again to all who are being a part of my final project.

*Anastasios Theodosiou*
# Table of contents

## CHAPTER 1: INTRODUCTION .................................................................................................................. 8

## CHAPTER 2: THE LINK PREDICTION ................................................................................................... 9

2.1 Problem definition .............................................................................................................................. 9
2.2 Why it is a hard problem ................................................................................................................... 10
2.3 Link prediction in social networks .................................................................................................. 10
2.4 The dataset used for this work ......................................................................................................... 11

## CHAPTER 3: GRAPHS FOR SOCIAL NETWORKS .................................................................................. 12

3.1 Graphs and their applications ......................................................................................................... 12
3.2 Why graphs ..................................................................................................................................... 13
3.3 Graph terminologies ....................................................................................................................... 14
3.4 Link prediction methods from graph theory .................................................................................... 15
   3.4.1 Node-based methods ................................................................................................................. 15
   3.4.2 Path-based methods .................................................................................................................. 17

## CHAPTER 4: APACHE SPARK ............................................................................................................. 18

4.1 Introduction ..................................................................................................................................... 18
4.2 A brief history of Spark .................................................................................................................... 18
4.3 A short comparison between Spark and Hadoop MapReduce ....................................................... 19
4.4 The essential difference between Hadoop MapReduce and Spark ............................................... 19
   Task Hadoop MapReduce is good for ................................................................................................. 19
   Tasks Spark is good for ...................................................................................................................... 19
4.5 Apache Spark Architecture ............................................................................................................. 20
   4.5.1 The role of the driver in Spark architecture ............................................................................. 21
   4.5.2 The role of the executor in Spark architecture ......................................................................... 21
   4.5.3 The role of cluster manager in Spark architecture ................................................................. 21
4.6 Spark’s Components ....................................................................................................................... 22
   4.6.1 Spark Core ............................................................................................................................... 22
   4.6.2 Spark SQL ............................................................................................................................... 22
   4.6.3 Spark Streaming ....................................................................................................................... 22
   4.6.4 MLLib ..................................................................................................................................... 23
   4.6.5 GraphX .................................................................................................................................. 23
   4.6.6 Cluster Managers .................................................................................................................... 23
4.7 The runtime architecture .................................................................................................................. 23
   4.7.1 Resilient Distributed Datasets (RDD) ..................................................................................... 24
   4.7.2 Directed Acyclic Graph (DAG) ............................................................................................... 24
   4.7.3 DataFrames ............................................................................................................................ 24
4.8 The Spark program lifecycle ........................................................................................................... 25
   4.8.1 Launching a Spark program ..................................................................................................... 25

## CHAPTER 5: MACHINE LEARNING WITH APACHE SPARK ............................................................. 26

5.1 Introduction ..................................................................................................................................... 26
5.2 How Spark enhances machine learning .......................................................................................... 26
5.3 Supervised machine learning ......................................................................................................... 27
   5.3.1 Naïve Bayes Classifier ............................................................................................................. 28
   5.3.2 Logistic Regression Classifier ................................................................................................. 28
Table of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>How a social network changed by the time</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Network represented by a graph</td>
<td>12</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Example of networks and their components. Source: TheMathCompany</td>
<td>13</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Differences between Hadoop Map Reduce and Spark</td>
<td>20</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Spark Architecture Diagram – Overview of Apache Spark Cluster</td>
<td>21</td>
</tr>
<tr>
<td>Figure 6</td>
<td>The Spark Stack</td>
<td>22</td>
</tr>
<tr>
<td>Figure 7</td>
<td>An example of a multilayer perceptron classifier execution process.</td>
<td>28</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors. Source: Wikipedia</td>
<td>29</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Tokenization and sparse matrix encoding</td>
<td>31</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Pipeline workflow. Source: spark.apache.org</td>
<td>32</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Workflow of a pipeline to produce predictions. Source: spark.apache.org</td>
<td>33</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Process of k-folds cross-validation</td>
<td>34</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Confusion matrix</td>
<td>35</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Example of type</td>
<td>and type</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Example of a ROC curve</td>
<td>37</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Comparison of general and locality sensitive hashing techniques.</td>
<td>38</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Intersection of sets A and B</td>
<td>39</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Band partitioning</td>
<td>41</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Comparison of execution time for ANN and ASL instead of a full scan</td>
<td>43</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Logistic Regression - Accuracy based on the number of iterations.</td>
<td>49</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Logistic Regression - F1 score based on the number of iterations.</td>
<td>50</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Logistic Regression - Accuracy and F1 for the feature set with structural properties</td>
<td>51</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Comparison of accuracy and F1 score per classifier</td>
<td>53</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Execution Time per Classifier</td>
<td>53</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Brute Force Execution Time based on Execution Cores</td>
<td>56</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Execution Time of Brute Force based on Dataset Volume</td>
<td>57</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Increase of Accuracy based on Dataset Volume</td>
<td>57</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Execution Time of Brute Force based on Accuracy</td>
<td>58</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Functionality of LSH</td>
<td>59</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Execution Time of MinHashLSH based on Execution Cores</td>
<td>60</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Accuracy of MinHashLSH based on Hash Tables</td>
<td>61</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Precision and Recall of MinHashLSH based on Hash Tables</td>
<td>61</td>
</tr>
<tr>
<td>Figure 33</td>
<td>F1 score based on Hash Tables</td>
<td>62</td>
</tr>
<tr>
<td>Figure 34</td>
<td>TP of MinHashLSH according to Hash Tables</td>
<td>62</td>
</tr>
<tr>
<td>Figure 35</td>
<td>TN of MinHashLSH according to Hash Tables</td>
<td>63</td>
</tr>
<tr>
<td>Figure 36</td>
<td>FP of MinHashLSH according to Hash Tables</td>
<td>63</td>
</tr>
<tr>
<td>Figure 37</td>
<td>FN of MinHashLSH according to Hash Tables</td>
<td>63</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Execution Time for Brute Force and MinHashLSH based on nodes number</td>
<td>65</td>
</tr>
<tr>
<td>Figure 39</td>
<td>At the left is the initial graph and a right it will be the clustered graph after filtering it with a threshold of θ = 0.5</td>
<td>67</td>
</tr>
</tbody>
</table>
Table of tables

TABLE 1: NODES DATASET STRUCTURE .......................................................................................................................... 11
TABLE 2: DESCRIPTION OF THE DATA SET ATTRIBUTES ................................................................................................. 44
TABLE 3: NAIVE BAYES CLASSIFIER RESULTS .................................................................................................................. 49
TABLE 4: ACCURACY AND F1 VS. MAXIMUM ITERATIONS .................................................................................................. 49
TABLE 5: LOGISTIC REGRESSION CLASSIFIER RESULTS .................................................................................................... 50
TABLE 6: LINEAR SVM CLASSIFIER RESULTS ..................................................................................................................... 51
TABLE 7: MULTICLASS PERCEPTRON CLASSIFIER RESULTS .................................................................................................. 52
TABLE 8: DECISION TREE CLASSIFIER RESULTS ............................................................................................................... 52
TABLE 9: RANDOM FOREST CLASSIFIER RESULTS ................................................................................................................. 52
TABLE 10: COMPARISON OF CLASSIFIERS BASED ON ACCURACY, F1 AND EXECUTION TIME ........................................... 54
TABLE 11: EXECUTION TIME FOR Brute Force algorithm per execution core number ............................................................. 55
TABLE 12: TOTAL CHECKS, CANDIDATE PAIRS, ACCURACY AND EXECUTION TIME OF Brute Force ..................................... 56
TABLE 13: STATISTIC RESULTS OF MinHashLSH .................................................................................................................. 60
TABLE 14: EXECUTION TIMES FOR DIFFERENT NODE NUMBERS .......................................................................................... 64
Chapter 1: Introduction

Networks can describe many biological, social and technological phenomena. In such systems, nodes represent entities, and links represent relationships or interactions between objects. Social interactions, for example, can be expressed through a system where a node is a person, and a link shows a friendship between the two individuals. Link predictions are the problem of assessing the likelihood of a relationship between two currently unconnected nodes based on observed nodes and links. Typical applications that require future link predictions are like Facebook, LinkedIn, Twitter, Google+, dating sites or even a co-authorship social network where link prediction can be used to propose a new friendship relationship to the first four social network types mentioned above or to suggest a co-operation in an original article to the last class of social network. Nowadays, where technology is overgrowing, social networks and their applications are fully integrated into our everyday lives. Therefore, more and more of us use them and enjoy their benefits. Some not so apparent forms of link prediction find usage in recommender systems and epidemiology. Recommender systems are broad, and in this case, there are (at least) two classes that would benefit from link prediction. First, the kind that suggests new friends/followers on social networks like them mentioned above, etc. Second, the type that suggests interests that are probably common, like Amazon and Netflix. In the latter case, link prediction is particularly beneficial for people who have just joined the system and do not yet have an extensive, robust network in place or are introverted or isolated and might never grow a big social network. Link prediction can give them an implied interface that will significantly improve the recommender system's performance since they tend to do worse with small data sets. Another application could use it for epidemiology; trying to predict the spread of disease through communities, and planning interventions to try to curtail it. In a similar vein, it could be used for planning political campaigns to understand how to optimize our canvassing to influence most people. Maybe even transportation planning to predict how to add or improve routes based on how people are trying to connect up. Consider this motivating example. People in the real world meet new friends. The relationship is represented by the appearance of a new connection in his or her social graph. Through the new relationship, both people’s social circle enlarges. Predicting these relationships before they are formed is vital to the success of social networking service. Link prediction attempts to address the issue of discovering future connections. As social networks become more and more common in our daily lives, the number of users is overgrowing, and so the data have imposed the need of new scalable technologies for extracting, processing and analyzing them. On this thesis, we will use Apache Spark which is ideal for this purpose. Apache Spark is also known as a fast, easy-to-use and general engine for significant data processing that has built-in modules for streaming, SQL, Machine Learning (ML) and graph processing. This technology is an in-demand skill for data engineers, but also data scientists can benefit from learning Spark when doing Exploratory Data Analysis (EDA), feature extraction and, of course, machine learning. On the next chapter we will discuss the link prediction problem and how can we deal with it. We will also look at different techniques which are used to solve this problem approximately.
Chapter 2: The link prediction

2.1 Problem definition

In this chapter, we will discuss the link prediction problem. Let us introduce what this problem is about. Think that we have a social network whose nodes represent individuals, and the edges between nodes represent relationships or interactions. So, given a social network with these characteristics, can we predict how the future network would look like? Alternatively, what new edges are likely to be formed in the future? In other words, to predict the likelihood of formation of new edges between any two nodes currently not connected. When we mention the “in the future” we take into account the concept of time. On figure 1 below we can see how a network changes by the time.

![Diagram of social network evolution over time]

*Figure 1: How a social network changed by the time*

By studying a social’s network evolution over time; we can understand how a node interacts to another node. To do this study, we need a lot of different snapshots of the network structure, so the volume of data which we have to compute enlarges rapidly, and we need to find different approaches for parallel computing. The link prediction problem can also be extended to inferring missing (erased or broken) edges in the network. Let us see another example. In real-world three persons will go for a meeting. If one of them is missing; his name can be predicted if we already know the other two persons and their past meetings form the network. Some other real-world examples of link prediction are suggesting friends and followers on a social networking site, suggesting relevant products to customers, providing suggestions to professionals to work as a team based on their fields or interests. The same could happen with organizations as well. Concluding; we can assume that the link prediction problem is the likelihood to predict the future association between two nodes.
2.2 Why it is a hard problem

Let us imagine a (social) network $G (V, E)$, then the number of possible edges in the network, to choose from, are $[V*(V-1)]/2$. If the networks are dense, then $E \approx V^2 - b$ where $b$ is some constant between 1 and $V$. If $G$ is sparse, then $E \approx V$ so, we have a $V^2$ edge of choosing from, and $O(1/V^2)$ probability of choosing correctly at random. However social networks are sparse, so there is no need to pick an edge at random a try to predict it in our network. On this work, we are experimenting with a big data set file which consists of rows that represent “papers.” The records are 27770 papers long, and each document represents a node (vertex) or our co-authorship network. If we had to check all these papers one to one to find similar documents based on a similarity metric, we should have check $[27770 *(27770 -1)]/2 = 385.572.565$ different edges manually. We understand that to check all these edges is time-consuming, so we need to find different, more efficient approaches for link prediction.

2.3 Link prediction in social networks

Interactions between users are the central element of social networks. When we referred to social networks in the past, we usually associate it with the classical study of sociology that often required a great deal of effort to measure and collect interactions between people. What makes it even more difficult is that social networks are very dynamic objects since new people (vertex) and new relationships (edges) are added to the network over time. With the development of the internet and web applications, we are in the information era. The last decade is the time when more and more people get access to the internet. Affordable technology products that have access to the internet, such as computers, laptops, tablets, mobile phones, and smartwatches, have helped the rapid evolution of social networks. The internet has solved all geographical constraints on communication issues. More and more people are becoming members of a social network with the need to share information. Major social networks such as Facebook, Twitter, Instagram and the social networking site for professionals, LinkedIn, have as their basic service the suggestion of friends or information that their users may be interested. If we can accurately predict the possible edges that will be added to the social network in the future, we will be able to gain a better understanding of what is the factor that leads to the development of the social networks to provide recommendations of higher accuracy and importance. This problem is called link prediction and has a significant impact on social networks as well as on many other areas. Co-authorship networks have been studied a lot in the past. A classic method for proposing collaboration in such a social network is through the bibliography system. However, the result of a new proposal based on this system could not be accurate enough. We need to extend and enrich this method with more data or new techniques so that we can achieve greater accuracy in our recommendations. For example, we can use methods based on the content and structure of documents.
2.4 The dataset used for this work

As we mentioned above, we will use a co-authorship network with 27770 nodes (which represents a paper) and 352,807 edges. Each edge represents a citation from one document to another document. The network graph is undirected. To use some supervised machine learning algorithms, we need to have training and a test set. For this study, we use a training set with 615,512 rows and a test set which is 32,618 entries long. Each row of the training set consists of 3 columns with the first two columns to represent a node id and the third column which holds the label. The label is a binary value while 0 means that the two nodes are not connected and 1 for the opposite case. The test set has the same structure as the training set while the label column is missing. This is the value which we will try to predict with classification. One the table 1 below we can see the structure of the node dataset which is used on this thesis.

<table>
<thead>
<tr>
<th>ID</th>
<th>Publication Year</th>
<th>Title</th>
<th>Authors</th>
<th>Journal</th>
<th>Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1995</td>
<td>compactification geometry and duality</td>
<td>Paul S. Aspinwall</td>
<td>Nucl.Phys</td>
<td>These are notes based...</td>
</tr>
</tbody>
</table>

Table 1: Nodes dataset structure

We will discuss the different approaches which are used on this word on chapters below. Both supervised and unsupervised methods or techniques we used, needs an introduction to Apache Spark and mllib, the library for machine learning.
Chapter 3: Graphs for social networks

On this chapter, we will discuss for some fundamentals about graphs and how a system, especially a social network, can be described by a graph. A graph is a visual representation of data. This form of visualization helps us gain actionable insights and make better data-driven decisions based on them. To understand what graphs are and why they are used, we need to know a concept called *Graph Theory*.

![Individual graph example](image)

*Figure 2: Network represented by a graph*

Graphs came from Mathematics and is a whole science field. Are structures used to study pairwise relationships between objects and entities? We use them in computer science because they help us to visualize data and understand more of them. So, in these sections we will not go deep in graph theory but, we will mention some basic about graphs and concepts which are used in network analysis. The Data Science and Analytics field have also used Graphs to model various structures and problems. As Data Scientists, we should be able to solve problems efficiently, and Graphs provide a mechanism to do that in cases where the data is arranged in a specific way. Generally speaking, a graph is a pair of sets, $G = (V, E)$, where:

- $G$: stands for Graph
- $V$ stands for Vertex. $V$ is a set of vertices and
- $E$: stands for Edge. $E$ is a set of Edges and is composed of pairs of elements of $V$.

3.1 Graphs and their applications

As we mentioned above, a network can be described as a graph, with vertices and edges. However, every network differs from other networks. Vertices and edges could have a different meaning. On figure 3 below, we can see these differences while the type of network changed.
As we can understand from the above picture, it is clear that the applications of Graphs in Data Analytics are numerous and vast. A few use cases of them could be:

- **Marketing Analytics** – Graphs can be used to figure out the most influential people in a Social Network.
- **Advertisers and Marketers** can estimate the biggest bang for the marketing buck by routing their message through the most influential people in a Social Network.
- **Banking Transactions** – Graphs can be used to find unusual patterns helping in mitigating Fraudulent transactions. There have been examples where Terrorist activity has been detected by analyzing the flow of money across interconnected Banking networks.
- **Supply Chain** – Graphs help in identifying optimum routes for your delivery trucks and in determining locations for warehouses and delivery centers.
- **Pharma** – Pharma companies can optimize the paths of the salesman using Graph theory. This helps in cutting costs and reducing the travel time for a salesman.
- **Telecom** – Telecom companies, typically use Graphs (Voronoi diagrams) to understand the quantity and location of Cell towers to ensure maximum coverage.

The advantages we gain from using graphs for analyzing data are numerous. On the next sections, we will dive into graphs to explore more uses of them.

### 3.2 Why graphs

Graphs provide a better way of dealing with abstract concepts like relationships and interactions. They also offer an intuitively visual way of thinking about these concepts. Graphs also form a natural basis for analyzing relationships in a Social Context. By the time, more and more are using graphs. Graph databases have become common computational tools and alternatives to SQL and
NoSQL databases. They are also used to model analytics workflows in the form of DAGs\(^1\) (Directed acyclic graphs). Neural networks\(^2\) are also utilized DAGs to model the various operations in different layers. Graph Theory concepts are used to study and model Social Networks, Fraud patterns, Power consumption patterns, Virality and Influence in Social Network. Social Network Analysis (SNA)\(^3\) is probably the best-known application of Graph Theory for Data Science. Graphs are also used in clustering algorithms, specifically on K-Means\(^4\). So, there are many reasons of using graphs but also so many other application fields. From a computer science perspective, graphs are offering computational efficiency. The Big O complexity for some algorithms is better for data arranged in the form of Graphs (compared to tabular data).

3.3 Graph terminologies

Before we go further, it is recommended that we should get familiar with these terminologies. We will not stop above that in addition to their enumeration.

- The vertices \(u\) and \(v\) are called the end vertices of the edge \((u, v)\)
- If two edges have the same end vertices, they are Parallel
- An edge of the form \((v, v)\) is a loop
- A Graph is simple if it has no parallel edges and loops
- A Graph is said to be Empty if it has no edges. Meaning \(E\) is empty
- A Graph is a Null Graph if it has no vertices. Meaning \(V\) and \(E\) is empty
- A Graph with only 1 Vertex is a Trivial graph
- Edges are Adjacent if they have a common vertex. Vertices are Adjacent if they have a common edge
- The degree of the vertex \(v\), written as \(d(v)\), is the number of edges with \(v\) as an end vertex. By convention, we count a loop twice, and parallel edges contribute separately
- Isolated Vertices are vertices with degree 1. \(d(1)\) vertices are isolated
- A Graph is Complete if its edge set contains every possible edge between ALL of the vertices
- A Walk in a Graph \(G = (V, E)\) is a finite, alternating sequence of the form \(V_iE_iV_iE_i\) consisting of vertices and edges of the graph \(G\)

\(^1\) In computer science and mathematics, a directed acyclic graph (DAG) is a graph that is directed and without cycles connecting the other edges.
\(^2\) A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes.
\(^3\) Social network analysis is the process of investigating social structures through the use of networks and graph theory.
\(^5\) \(V_iE_iV_iE_i\) : Stands for Vertex - Edge - Vertex - Edge
- A Walk in Open if the initial and final vertices are different. A Walk is Closed if the initial and final vertices are the same.
- A Walk is a Trail if ANY edge is traversed at most once.
- A Trail is a Path if ANY vertex is traversed at most once (Except for a closed walk).
- A Closed Path is a Circuit – Analogous to electrical circuits.

3.4 Link prediction methods from graph theory

In this section, we will look at some of the standard techniques which are used in link prediction and came from graph theory. The ways which we will survey are assigning a weighted score on edge for a pair of nodes \((x, y)\), based on the input graph \(G\) and finally, they sort the results in decreasing order. As we already said that methods are adapted from graph theory\(^6\) and social network analysis\(^7\), we should also point that, these techniques were not designed to measure node-to-node similarity, and need to be modified for this purpose. Below, we discuss these methods in more details. Perhaps the most basic approach is to rank a pair of nodes, is by the length of their shortest path\(^8\) in graph \(G\). We will categorize these methods into two different groups. In methods which are based on the node’s neighborhoods and in methods which are based on the ensemble of all paths.

3.4.1 Node-based methods

Approaches which are based on nodes information are addressing the link prediction problem from the view of nodes similarity by proximity. On this subsection, we will analyze some widely used measures including common neighbors, Jaccard similarity, Katz status index, Adamic / Adar and preferential attachment.

- **Common Neighbors index**

  The common-neighbors index captures the notion that two strangers who have a common friend may be introduced by that friend. This introduction has the effect of “closing a triangle” in the graph and feels like a common mechanism in real life. Newman [6] has computed this quantity in the meaning of collaboration networks, resulting in that a positive correlation between the number of common neighbors of \(x\) and \(y\) at time \(t\), and the probability that \(x\) and \(y\) will collaborate at some time after \(t\). The score function of Common Neighbors can be calculated from the function below:

---

\(^6\) In mathematics, graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of vertices, nodes, or points which are connected by edges, arcs, or lines. Source: Wikipedia

\(^7\) Social network analysis is the process of investigating social structures through the use of networks and graph theory. It characterizes networked structures in terms of nodes and the ties, edges, or links that connect them. Source: Wikipedia

\(^8\) In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. Source: Wikipedia
\[ \text{score} \ (V_x, V_y) = |\Gamma(x)| \cap |\Gamma(y)| \]

- **Jaccard Similarity**

The Jaccard’s coefficient is a similarity metric that has been used in the past in the information retrieval field. It measures the probability that both \( V_x \) and \( V_y \) have a common feature \( f \). Now, it can be assumed that feature \( f \) is the neighbors in graph \( G \) [7]. The score function of Jaccard’s index is given by the equation below:

\[ \text{score} \ (V_x, V_y) = \frac{|\Gamma(V_x) \cap \Gamma(V_y)|}{|\Gamma(V_x) \cup \Gamma(V_y)|} \]

- **Adamic / Adar**

The Adamic/Adar index is a measure which was introduced by Lada Adamic and Eytan Adar to predict links in a social network, according to the number of shared links between two nodes. The Adamic/Adar measure could be described as a weighted version of common neighbors. The \( \Gamma(z) \) is defined as the section between two related web pages or ONS\(^9\) users, counting the frequency of common features and weighting rarer features with a bigger score. Adamic/Adar score function is described below:

\[ \text{score} \ (V_x, V_y) = \frac{1}{\log(|z|)} \text{, } z \in \Gamma(V_x) \cap \Gamma(V_y) \]

- **Preferential attachment**

It has been proposed by Newman [6], who have been based on empirical evidence, that the probability of future co-authorship between nodes \( V_x \) and \( V_y \) is highly related with the product of the number of the existing collaborations between these nodes. In terms of a social network is that users with many friends tend to create more connections in the future. In some networks, like in finance, the rich get richer. We estimate how “rich” our two vertices are by calculating the multiplication between the number of friends (\( |\Gamma(x)| \)) or followers each node has. There is no need for node neighborhood information; therefore, this similarity index has the lowest computational complexity. The score function is shown below:

\[ \text{score} \ (V_x, V_y) = |\Gamma(V_x)| \cdot |\Gamma(V_y)| \]

\(^9\) Online Social Networks
3.4.2 Path-based methods

Methods based on paths are also available. These methods take into account the hole network structure, considering all different paths of different length between two nodes. Two of them, as are widely used on link prediction, are described below.

- **Shortest Path - Graph Length**

  The algorithm of the shortest path between two nodes in a graph. It is a commonly used similarity measure on social network analysis (SNA). There are a plethora of well-known algorithms for calculating the shortest path, such as Dijkstra one.

- **Katz Status Index**

  Leo Katz [8] introduced a new status index combining sociometric analysis with basic linear algebra back in 1953. Katz’s method computes influential nodes in a social network by summing over the collection of paths of different length \( l \), that connect two nodes \( Vx \) and \( Vy \). The idea behind the Katz-measure is that the more paths there are between two nodes and the shorter these paths are, the stronger the connection. The Katz status index can be described by the following equation:

  \[
  Katz_\beta = \sum_{\ell=1}^{\infty} \beta^\ell |paths_{Vx, Vy}^\ell|_{10}
  \]

  There are also some other interesting path-based methods such as Hitting time and PageRank, SimRank, Random Walk with Restart (RWR) and more but, they are not the subject for the study of this diploma thesis.

---

\(^{10}\) *Equation for Katz’s status index. Source: Katz centrality, Wikipedia*
Chapter 4: Apache Spark

4.1 Introduction

Apache Spark\(^{11}\) is a cluster computing platform designed to be fast and general-purpose. At its core, Spark is a “computational engine” that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines or a computing cluster. Spark extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. One of the main features Spark offers for speed is the ability to run computations in memory. On the generality side, Spark is designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming which is often necessary for production data analysis pipelines. Spark is also designed to be highly accessible, offering simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries. It also integrates tightly with other Big Data tools. In particular, Spark can run in Hadoop clusters and access any of its data source, any file stored in the Hadoop distributed file system (HDFS) or other storage systems supported by the Hadoop APIs (including your local file system, Amazon S3, Cassandra, Hive, HBase, etc.).

4.2 A brief history of Spark

Spark started in 2010 as a research project in the UC Berkeley RAD Lab, later to become the AMPLab. The researchers in the lab had previously been working on Hadoop MapReduce and observed that MapReduce was inefficient for iterative and interactive computing jobs. Thus, from the beginning, Spark was designed to be fast for interactive queries and iterative algorithms, bringing in ideas like support for in-memory storage and efficient fault recovery. Soon after its creation, it was already 1× faster than MapReduce for specific jobs. Some of Spark’s first users were other groups inside UC Berkeley, including machine learning researchers such as the Mobile Millennium project, which used Spark to monitor and predict traffic congestion in the San Francisco Bay Area. Spark was first to open source on March 2 and was transferred to the Apache Software Foundation on June 2 where it is now a top-level project. It is an open source project that has been built and is maintained by a thriving and diverse community of developers. In addition to UC Berkeley, significant contributors to Spark include Databricks\(^{12}\), Yahoo!, and Intel. Internet powerhouses such as Netflix, Yahoo, and eBay have deployed Spark at massive scale, collectively processing

\(^{11}\) https://spark.apache.org/

\(^{12}\) Databricks is a company founded by the creators of Apache Spark, that aims to help clients with cloud-based big data processing using Spark. Databricks grew out of the AMPLab project at University of California, Berkeley that was involved in making Apache Spark, a distributed computing framework built atop Scala.
multiple petabytes of data on clusters of over 8, nodes. It has quickly become the largest open source community in big data, with over 1000+ contributors from 250+ organizations.

4.3 A short comparison between Spark and Hadoop MapReduce

Both Hadoop and Spark are open source works which are offered by Apache Software. The bidg data market has been leaded from Hadoop for more than a decade. According to ScienceSoft market research, Hadoop’s installed base amounts to 50+ customers, while Spark boasts 10+ installations only. However, Spark’s popularity skyrocketed in 2to overcome Hadoop in only a year. A new installation growth rate shows that the trend is still ongoing. Spark is outperforming Hadoop with 47% vs. 14% correspondingly.

4.4 The essential difference between Hadoop MapReduce and Spark

To be fair with the comparison, we will contrast Spark with Hadoop MapReduce, as both are responsible for significant data processing. The key difference between them is in processing. Spark can do in-memory while Hadoop has to read and write to a disk. This is a big difference which occurs in data processing. As a result, Spark can be 100 times faster, but there is a difference in the volume of data which processed. Hadoop MapReduce can work with far bigger files than Spark.

Task Hadoop MapReduce is good for

- **Linear processing of massive datasets.** Hadoop MapReduce allows the parallel processing of vast amounts of data. It breaks a large chunk into smaller ones to be processed separately on different data nodes and automatically gathers the results across the multiple nodes to return a single result. In case the resulting dataset is more substantial than the available RAM, Hadoop MapReduce may outperform Spark.
- **An economical solution, if no immediate results are expected.** MapReduce is a good solution if the speed of processing is not critical. For instance, if data processing can be done during night hours, it makes sense to consider using Hadoop MapReduce.

Tasks Spark is good for

- **Fast data processing.** In-memory processing makes Spark faster than Hadoop MapReduce – up to 100 times for data in RAM and up to 10 times for data in storage.
- **Iterative processing.** If the task is to process data again and again – Spark defeats Hadoop MapReduce. Spark’s Resilient Distributed Datasets (RDDs) enable multiple map operations in memory, while Hadoop MapReduce has to write temporary results to a disk.
- **Near real-time processing.** If a business needs immediate insights, then they should opt for Spark and its in-memory processing.
- **Graph processing.** Spark’s computational model is suitable for iterative computations that are typical in graph processing. Moreover, Apache Spark has GraphX, an API for graph computation.

- **Machine learning.** Spark has MLlib, a built-in machine learning library, while Hadoop needs a third-party to provide it. MLlib has out-of-the-box algorithms that also run in memory.

- **It is joining datasets.** Because of its speed, Spark can create all combinations faster, though Hadoop may be better if joining of massive datasets that require a lot of shuffling and sorting is needed.

Concluding, Hadoop Map Reduce is written in Java while is written in Scala. On figure 4 below we can see the differences between them.

![Figure 4: Differences between Hadoop Map Reduce and Spark](image)

### 4.5 Apache Spark architecture

On this chapter, we will discuss Apache Spark architecture. Apache Spark has a well-defined and layered architecture where all of its components and layers are loosely coupled and integrated with various extensions and libraries. The basic structure of Spark architecture is shown below in Figure 5.
Apache Spark is following a master/slave architecture with two main daemons and a cluster manager.

- Master Daemon – (Master/Driver Process)
- Worker Daemon – (Slave Process)

A spark cluster has a single Master and any number of Slaves/Workers. The driver and the executors run their Java processes, and users can run them on the same horizontal spark cluster or separate machines, i.e., in a vertical spark cluster or mixed machine configuration.

4.5.1 The role of the driver in Spark architecture

It is the central and the entry point of the Spark Shell (Scala, Python, and R). The driver program runs the primary function of the application and is the place where the Spark Context is created. Spark Driver contains various components – DAG Scheduler, Task Scheduler, Backend Scheduler and Block Manager responsible for the translation of spark user code into an actual spark jobs which are executed on the cluster.

4.5.2 The role of the executor in Spark architecture

The executor is a distributed agent responsible for the execution of tasks. Every spark application has its executor process. Executors usually run for the whole lifetime of a Spark application and this phenomenon is known as “Static Allocation of Executors.” However, we (users) can also opt for dynamic allocations of them wherein they can add or remove spark executors dynamically to match with the overall workload.

4.5.3 The role of cluster manager in Spark architecture

A cluster manager is an external service responsible for acquiring resources on the spark cluster and allocating them to a spark job. There are three different types of cluster managers which a Spark application can leverage for the allocation and deallocation of various physical resources such as memory for client spark jobs, CPU memory, etc. Hadoop YARN, Apache Mesos or the simple standalone spark cluster manager either of them can be launched on-premise or in the cloud for a spark application to run. In order to choose a cluster manager for any spark application
depends on the application’s prerequisites because all cluster managers provide a different set of scheduling capabilities.

4.6 Spark’s components

In this section, we will briefly introduce each of Spark’s components, shown in Figure 6 below.

4.6.1 Spark Core

Spark Core contains the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage systems, and more. Spark Core is also home to the API that defines resilient distributed datasets (RDDs), which are Spark’s central programming abstraction. RDDs represent a collection of items distributed across many compute nodes that can be manipulated in parallel. Spark Core provides many APIs for building and manipulating these collections.

4.6.2 Spark SQL

Spark SQL is Spark’s package for working with structured data. It allows querying data via SQL as well as the Apache Hive variant of SQL—called the Hive Query Language (HQL)—and it supports many sources of data, including Hive tables, Parquet, and JSON. Beyond providing a SQL interface to Spark, Spark SQL allows developers to intermix SQL queries with the programmatic data manipulations supported by RDDs in Python, Java, and Scala, all within a single application, thus combining SQL with sophisticated analytics.

4.6.3 Spark Streaming

Spark Streaming is a Spark component that enables the processing of live streams of data. It offers an API for manipulating data streams that closely matches the Spark Core’s RDD API, making it easy for programmers to learn the project and move between applications that manipulate data.
stored in memory, on disk, or arriving in real time. Underneath its API, Spark Streaming was designed to provide the same degree of fault tolerance, throughput, and scalability as Spark Core.

4.6.4 MLlib
Spark comes with a library containing everyday machine learning (ML) functionality, called MLlib. MLlib provides multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering, as well as supporting procedures such as model evaluation and data import. It also provides some lower-level ML primitives, including a generic gradient descent optimization algorithm. All of these methods are designed to scale out across a cluster.

4.6.5 GraphX
GraphX is a library for manipulating graphs (e.g., a social network’s friend graph) and performing graph-parallel computations. Like Spark Streaming and Spark SQL, GraphX extends the Spark RDD API, allowing users to create a directed graph with arbitrary properties attached to each vertex and edge. It also provides various operators for manipulating graphs (e.g., subgraph and map Vertices) and a library of common graph algorithms (e.g., PageRank and triangle counting).

4.6.6 Cluster Managers
Under the hood, Spark is designed to efficiently scale up from one to many thousands of compute nodes. To achieve this while maximizing flexibility, Spark can run over a variety of cluster managers, including Hadoop YARN, Apache Mesos, and a simple cluster manager included in Spark itself called the Standalone Scheduler.

4.7 The runtime architecture
When a user submits a spark application code, the driver implicitly converts the code containing transformations and actions into a logical directed acyclic graph (DAG). At this stage, the driver program also performs specific optimizations like pipelining transformations and then it converts the logical DAG into a physical execution plan with a set of stages. After creating the physical execution plan, it creates small physical execution units referred to as tasks under each stage. After that tasks are bundled to be sent to the Spark Cluster. After that, the driver program talks to the cluster manager and negotiates for resources. After that, the cluster manager launches executors on the worker nodes on behalf of the driver. At this point, the driver sends tasks to the cluster manager based on data placement. Before executors begin execution, they register themselves with the driver program so that the driver has a holistic view of all the executors. Then executors start executing the different tasks assigned by the driver program. At any time when the spark application is running, the driver program will monitor the set of executors that run. Driver program in the spark architecture also schedules future tasks based on data placement by tracking
the location of cached data. When driver programs main () method exits or when it calls the stop () method of the Spark Context. This function will terminate all the executors and release the resources from the cluster manager. The structure of a Spark program at a higher level is - RDD’s are created from the input data and new RDD’s are derived from the existing RDD’s using another type of transformations, after which an action is performed on the data. In any spark program, the DAG operations are created by default and whenever the driver runs the Spark DAG will be converted into a physical execution plan. The whole spark architecture is based on two main abstractions:

- Resilient Distributed Datasets (RDD)
- Directed Acyclic Graph (DAG)

4.7.1 Resilient Distributed Datasets (RDD)

RDD’s are a collection of data items that are split into partitions and can be stored in-memory on workers nodes of the spark cluster. In terms of datasets, apache spark supports two types of RDD’s – Hadoop Datasets which are created from the files stored on HDFS and parallelized collections which are based on existing Scala collections. Spark RDD’s support two different types of operations – Transformations and Actions.

4.7.2 Directed Acyclic Graph (DAG)

Direct - Transformation is an action which transitions data partition state from A to B. Acyclic - Transformation cannot return to the older partition. DAG is a sequence of computations performed on data where each node is an RDD partition, and the edge is a transformation on top of data. The DAG abstraction helps eliminate the Hadoop MapReduce multistage execution model and provides performance enhancements over Hadoop.

4.7.3 DataFrames

DataFrames does not belong on the two main abstractions of Spark but is the easiest way to work with structured data. DataFrame is a Dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as structured data files, tables in Hive, external databases, or existing RDDs.

---

13 Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Hive gives a SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop. Source: Wikipedia
4.8 The Spark program lifecycle

As we already know, when the driver program is launched (this is our Spark application) it waits for the launching of the application’s master task YARN or Mesos or Spark resource manager (it is up to us which manager we will use, and it can be configured by the config params of spark-submit). Application’s master allocates tasks in the cluster according to locality, resource requirements availability (CPU, RAM, etc.). After that, the worker nodes (slaves) receives a task which they have to execute, and finally, they write the results. If the main () method completes or invokes the SparkContext.stop () method, executors terminate and releases resources.

4.8.1 Launching a Spark program

There is only one script which is used to submit a spark program, and that is spark-submit. It launches the application on the cluster. There are multiple options through which spark-submit script can connect with different cluster managers and control on the number of resources the application gets. We can submit our Spark application to a Spark deployment environment for execution, kill or request the status of Spark applications. The spark-submit script is located in the bin directory of the Spark distribution. Bellow, we can see an example of a spark submit script.

```bash
$ ./bin/spark-submit
Usage: spark-submit [options] <app jar | python file> [app arguments]
Usage: spark-submit --kill [submission ID] --master [spark://...]
Usage: spark-submit --status [submission ID] --master [spark://...]
Usage: spark-submit run-example [options] example-class [example args]
...
Chapter 5: Machine learning with Apache Spark

Apache Spark includes MLlib to bring machine learning on big data. Its goal is to make practical machine learning scalable and secure. At a high level, it provides tools such as ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering.

5.1 Introduction

In this section, we will discuss Machine learning (ML), how we can use ML algorithms along with Apache spark and enumerate some of the algorithms which have been used on this work. Generally speaking, ML is the scientific study of algorithms and statistical models that computer systems use to improve their performance on a specific task progressively. Before we go, we need to know what they are and who they work. Machine learning algorithms build a mathematical model of sample data, known as "training data," in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics\(^\text{14}\), which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining\(^\text{15}\) is a field of study within machine learning and focuses on exploratory data analysis (EDA) through unsupervised learning. As more of the ML algorithms works better with a significant amount of data in order to make more accurate predictions, there raises the need for parallelization and scalability of the tasks. To help solve this problem, Apache Spark provides a general machine learning library MLlib that is designed for simplicity, scalability, and easy integration with other tools. With the scalability, language compatibility, and speed of Spark, data scientists can solve and iterate through their data problems faster. On the next section, we will mention briefly why Spark enhances machine learning.

5.2 How Spark enhances machine learning

Data scientist due in no small number of packages and libraries that are already available, are more active and accustomed with Python and R Spaks APIs than the Scala’s one. These can help them to solve their data problems. However, traditional uses of these tools are often limiting, as they process data on a single machine where the transferring procedure of data becomes time-

\(^{14}\) Computational statistics, or statistical computing, is the interface between statistics and computer science. It is the area of computational science (or scientific computing) specific to the mathematical science of statistics. This area is also developing rapidly, leading to calls that a broader concept of computing should be taught as part of general statistical education. Source: Wikipedia

\(^{15}\) In statistics, exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. Source: Wikipedia
consuming, the analysis requires sampling (which often does not accurately represent the data), and moving from development to production environments requires extensive redesign. Spark’s solution on this problem is to provide to data engineers and data scientists with a powerful, unified engine that is both fast (100x faster than Hadoop for large-scale data processing) and easy to use. This enables data practitioners to solve their machine learning problems (as well as graph computation, streaming, and real-time interactive query processing) interactively and at a much higher scale. Spark also provides many language choices, including Scala, Java, Python, and R. The latest while writing this work (2 Spark Survey) that polled the Spark community shows unusually rapid growth in Python and R. Specifically, 62 percent of respondents were using Python (a 4 percent increase over 2 and 20 percent were already using the R API (a 2 percent increase over 2. With more than 1, code contributors in 2 Apache Spark are the most actively developed open source project among data tools, big or small. Much of the focus is on Spark’s machine learning library, MLlib, with 38 percent increase in 2 The importance of machine learning has not gone unnoticed, with 40 percent of the 2Spark Survey respondents using Spark for recommendation engine products, 45 percent for real-time streaming products and 29 percent developed fraud detection/security products. These are sophisticated users. Forty-one percent of the survey respondents identified themselves as data engineers, while 23 percent identified themselves as data scientists. As a result, Spark is an ideal framework for implementing Machine Learning on Big Data with a lot’s of already implemented algorithms which are scalable and made to work cross-platform. On the section below we will discuss Supervised and Unsupervised Machine learning, and we will briefly introduce some of the basic concepts we used on this work.

5.3 Supervised machine learning

With the nature of the problem and the advantages of supervised learning, we can encounter it as a binary classification problem. Then we will try to predict if a node is connected with another node. With other words, we are trying to predict edges a time t for the time t+1. In this section, we will only discuss Classification algorithms which are used on this work. It will be a short introduction to classification machine learning with Spark. MLlib provides a variety of different types of classifiers and regressors, but we will introduce only some of the used classifiers. On this work we experiment with six different classifiers which are:

- Naive Bayes Classifier
- Logistic Regression Classifier

---

16 Source: 2Spark Survey
17 Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. Source: Wikipedia
18 Unsupervised Learning is a class of Machine Learning techniques to find the patterns in data. The data given to unsupervised algorithm are not labelled, which means only the input variables(X) are given with no corresponding output variables. Source: towardsdatascience.com
19 Statistical classification is a type of statistical or machine learning problem. The character of these problems is such that a new observation should be assigned to mutually exclusive categories. Source: Quora
5.3.1 Naive Bayes Classifier

It is a classification technique based on Bayes’ Theorem\(^\text{20}\) with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability. Naive Bayes model is easy to build and particularly useful for substantial data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Spark provides a Naive Bayes Classifier too.

5.3.2 Logistic Regression Classifier

It is a statistical method for analyzing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables.

5.3.3 Decision Tree Classifier

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision

\(^{20}\) In probability theory and statistics, Bayes’ theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Source: Wikipedia
node has two or more branches, and a leaf node represents a classification or decision. The first decision node in a tree which corresponds to the best predictor called the root node. Decision trees can handle both categorical and numerical data. As Naive Bayes and Logistic Regression, Decision Tree classifier is also included on Spark.

5.3.4 Random Forest Classifier
Random forests or random decision forests are an ensemble learning\(^\text{21}\) method for classification, regression, and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or means prediction (regression) of the individual trees. Random decision forests correct for decision trees’ habit of overfitting to their training set.

5.3.5 Linear Support Vector Machine
Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is some features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiates the two classes very well. Support Vectors are merely the coordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). Below in figure 8, we can see the hyperplane and the line between points as well as the margin between them.

\[ w \cdot x - b = 1 \]
\[ w \cdot x - b = 0 \]
\[ w \cdot x - b = -1 \]

*Figure 8: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors. Source: Wikipedia.*

\(^{21}\) Ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem. In contrast to ordinary machine learning approaches which try to learn one hypothesis from training data, ensemble methods try to construct a set of hypotheses and combine them to use. Source: Wikipedia
5.3.6 Multilayer Perceptron Classifier

Multilayer perceptron classifier (MLPC) is a classifier based on the feedforward artificial neural network. MLPC consists of multiple layers of nodes. Each layer is fully connected to the next layer in the network. Multilayer Perceptron (MLP) is a supervised learning algorithm that learns a function $\mathbb{R}^m \rightarrow \mathbb{R}^0$ by training on a dataset, where $m$ is the number of dimensions for input and is the number of dimensions for output. Given a set of features $X = x_1, x_2, x_3, x_m$, and a target $y$, it can learn a nonlinear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more nonlinear layers, called hidden layers.

5.4 Feature engineering

In this section, we will enumerate some of the most common text processing methods which are used on text classification. The text is a type of data that can come in different formats. There are so many text processing methods, but they are not the subject of this work because they are not all of them, used in our implementation. Before working with text, one must tokenize it. Tokenization implies splitting the text into units (hence, tokens). Most simply, tokens are just the words. However, splitting by word can lose some of the meaning, "Santa Barbara" is one token, not two, but "rock'n'roll" should not be split into two tokens. There are ready-to-use tokenizers that take into account peculiarities of the language, but they make mistakes as well, mainly when you work with specific sources of text (newspapers, slang, misspellings, typos), or in our case, academic papers. Spark provides a variety of feature transformer\textsuperscript{22}. In Scala, it is pretty easy to work with Tokenizers, and we can create an instance of them as shown in the figure below.

```scala
import org.apache.spark.ml.feature.Tokenizer
def tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")
```

After tokenization, we will normalize the data. For text, this is about stemming and lemmatization; these are similar processes used to process different forms of a word. Another crucial thing that we have to consider is the stop words. It is crucial to remove those words because they can drive our machine learning model in less accurate predictions. Words like those could be the word ‘I’, am, and, at, the ‘Scala also provides a StopWordRemover which is as easy as the tokenization process. So, now that we have turned the document into a sequence of words, we can represent it with vectors. The most straightforward approach is called Bag of Words: we create a vector with the length of the dictionary, compute the number of occurrences of each word in the text, and place

\textsuperscript{22} Transformers. A Transformer is an abstraction that includes feature transformers and learned models. Technically, a Transformer implements a method transform() , which converts one DataFrame into another, generally by appending one or more columns. Source: spark.apache.org
that number of occurrences in the appropriate position in the vector. Figure 9 below describes the above scenario. It is important to mention that this is an incredibly naive implementation.

“This is how you get ants.”

tokenizer

['this','is','how','you','get', 'ants']

Build a vocabulary over all docum

['aardvak','amsterdam','ants', ...'you','your', 'zyxst']

Sparse matrix encoding

aardvak ants get you zyxst

[0, ..., 0, 1, 0, ..., 0, 1, 0, ..., 0, 1, 0, ..., 0 ]

*Figure 9: Tokenization and sparse matrix encoding*

The next thing we could do, always based on our scenario, is to transform our Bag of Words with N-grams. The reason that we need this process is why we are using algorithms like Bag of Words, and we lose the order of the words in the text, which means that the texts "I have no cows" and "no, I have cows" will appear identical after vectorization when, in fact, they have the opposite meaning. To avoid this problem, we can revisit our tokenization step and use N-grams (the sequence of N consecutive tokens) instead. Apache Spark is also providing this feature. Using these algorithms, it is possible to obtain a working solution for a simple problem, which can serve as a baseline. However, for those who do not like the classics, there are new approaches. The most popular method in the new wave is Word2Vec, but there are a few alternatives as well (GloVe, Fasttext, etc.). Word2Vec is a particular case of the word embedding algorithms of Spark. Using Word2Vec and similar models, we can not only vectorize words in a high-dimensional space (typically a few hundred dimensions) but also compare their semantic similarity. This is a classic example of operations that can be performed on vectorized concepts: king - man + woman = queen.

It is worth noting that this model does not comprehend the meaning of the words but tries to position the vectors such that words used in a familiar context are close to each other. Such models need to be trained on extensive datasets in order for the vector coordinates to capture the semantics.
5.5 ML Pipelines

ML Pipelines provide a uniform set of high-level APIs built on top of DataFrames that help users create and tune practical machine learning pipelines. A pipeline in machine learning, it is common to run a sequence of algorithms to process and learn from data. E.g., a simple text document processing workflow might include several stages:

- Split each document’s text into words.
- Convert each document’s words into a numerical feature vector.
- Learn a prediction model using the feature vectors and labels.

So how it works? A Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator. These stages are run in order, and the input DataFrame is transformed as it passes through each stage. For Transformer stages, the transform () method is called on the DataFrame. For Estimator stages, the fit () method is called to produce a Transformer (which becomes part of the Pipeline Model, or fitted Pipeline), and that Transformer’s transform () method is called on the DataFrame. We illustrate this for the simple text document workflow. The figure below (figure 10) is for the training time usage of a Pipeline.

![Pipeline workflow. Source: spark.apache.org](image)

Above, the top row represents a Pipeline with three stages. The first two (Tokenizer and HashingTF) are Transformers (blue), and the third (Logistic Regression) is an Estimator (red). The bottom row represents data flowing through the pipeline, where cylinders indicate DataFrames. The Pipeline.fit() method is called on the original DataFrame, which has raw text documents and labels. The Tokenizer.transform() method splits the raw text documents into words, adding a new column with words to the DataFrame. The HashingTF.transform() method converts the words column into feature vectors, adding a new column with those vectors to the DataFrame. Now, since LogisticRegression is an Estimator, the Pipeline first calls LogisticRegression.fit() to produce a logistic regression model. If the Pipeline had more Estimators, it would call the LogisticRegressionModel’s transform() method on the DataFrame before passing the DataFrame to the next stage. A Pipeline is an Estimator. Thus, after a Pipeline’s fit() method runs, it produces
a Pipeline Model, which is a Transformer. This Pipeline Model is used at test time and the figure below (figure 11) illustrates this usage.

In the figure above, the Pipeline Model has the same number of stages as the original Pipeline, but all Estimators in the original Pipeline have become Transformers. When the Pipeline Model’s `transform()` method is called on a test dataset, the data are passed through the fitted pipeline in order. Each stage’s `transform()` method updates the dataset and passes it to the next stage. Pipelines and Pipeline Models help to ensure that training and test data go through identical feature processing steps.

5.6 Evaluating a classifier

5.6.1 Holdout method

After training the model, the most important part is to evaluate the classifier to verify its applicability. There are several methods that exist, and the most common method is the holdout method. In this method, the given data set is divided into two partitions as a test and train 20% and 80% respectively. The train set will be used to train the model, and the unseen test data will be used to test its predictive power.

- **Pros:** Train and test split is still useful because of its flexibility and speed
- **Cons:** Provides a high-variance estimate of out-of-sample accuracy

5.6.2 K-Fold Cross Validation

Overfitting is a common problem in machine learning which can occur in most models. k-fold cross-validation can be conducted to verify that the model is not overfitted. This method splits the data set into K equal partitions (“folds”), then uses 1-fold as for testing purposes and the union of the other folds as the training set. The process will follow the above steps K times, using different folds as the testing set each time. The average testing accuracy of the process is testing accuracy. We can see the procedure in figure 12 below.
Figure 12: Process of k-folds cross-validation

- **Pros**: A more accurate estimate of out-of-sample accuracy. More “efficient” use of data (every observation is used for both training and testing)
- **Cons**: Much slower than Train/Test split.

5.7 Model Evaluation Metrics

A module evaluation metric is a criterium by which the performance or the accuracy of a model is measured. In the upcoming subsections, we will discuss the evaluation metrics in detail.

5.7.1 Classification Accuracy

Classification accuracy is by far the most common model evaluation metric used for classification problems. Classification accuracy is the percentage of correct predictions. Even though classification is a useful metric, when the class distribution is imbalanced, it can give a false sense of high accuracy. Apache spark provides a separate class to evaluate the accuracy, and not only this, which is “Multiclass Classification Evaluator MCE.” Also, the “accuracy” estimator is built in as a parameter in “MCE.” The classification accuracy metric works better if there is an equal number of samples in each class. For example, if there are 90% class A samples and 10% of class B, and trained a model, the model would have a 90% training accuracy just by predicting every sample as class A. However, if the same model is applied to a dataset with a different class distribution, (60% samples is class A, and 40% in class B), the test accuracy score would drop to 60%. So, we do not have to trust only this metric. We should look and compare the results with different approaches.
5.7.2 Confusion Matrix

Confusion Matrix can be interpreted loosely as a table that describes the performance of a classification model on a set of test data for which the actual values are known. The confusion matrix is highly interpretative and can be used to estimate the number of other metrics. Spark provides a method to perform the confusion matrix on the testing data set. The “confusion matrix” metric which is included in the Multiclass Classification class requires the actual response class values and the predicted values to determine the matrix. Since our problem (Link Prediction Problem) has only two response classes (edge -> existence /nonexistence), it can be categorized as a binary classification problem. Therefore, the confusion matrix is a 2 X 2 grid. The confusion matrix is interpreted differently in different implementations.

<table>
<thead>
<tr>
<th>Predicted Classes</th>
<th>Observed Classes (Reference)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Negative</td>
</tr>
<tr>
<td>Negative</td>
<td>a</td>
</tr>
<tr>
<td>Positive</td>
<td>c</td>
</tr>
</tbody>
</table>

Figure 13: Confusion matrix

The basic terminology related to the confusion matrix is as follows. We will interpret with regards to our problem.

- **True Positives (TP):** Correct prediction as the existence of an edge
- **True Negatives (TN):** Correct prediction as the nonexistence of an edge
- **False Positives (FP):** Incorrect prediction as the existence of an edge (‘Type I error’)
- **False Negatives (FN):** Incorrect prediction as the nonexistence of an edge (‘Type II error’)

There is no better example to explain what we mean with ‘Type I error’ and ‘Type II error’ of the following example as shown in figure 14.

Figure 14: Example of type I and type II errors. Source: Google
5.7.3 Metrics computed from Confusion Matrix

The following metrics can be computed from the values of the confusion matrix. First, we have to parse the obtained confusion matrix into True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

- **Classification accuracy**: Classification accuracy is the ratio of correct predictions to the total no. of predictions. Alternatively, more simply, how often is the classifier correct. We can calculate the accuracy using the confusion matrix. Following is the equation to calculate the accuracy using the confusion matrix:

  \[
  \text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}
  \]

- **Sensitivity/Recall**: Sensitivity or recall is the ratio of correct positive predictions to the total no. of positive predictions. Alternatively, more simply, how sensitive the classifier is for detecting positive instances. This is also called “True Positive Rate.”

  \[
  \text{Recall} = \frac{TP}{TP+FN}
  \]

- **Specificity**: Specificity is the ratio of correct negative predictions to the total number of negative predictions. This determines how specific the classifier is in predicting positive instances.

  \[
  \text{Specificity} = \frac{TP}{TN+FP}
  \]

- **False Positive Rate (FPR)**: The false positive rate is the ratio of negative predictions that were determined to be positive to the total no. negative predictions. Alternatively, when the actual value is negative, how often is the prediction incorrect.

  \[
  \text{FPR} = \frac{FP}{TN+FP}
  \]

- **Precision**: Precision is the ratio of correct predictions to the total no. of predicted correct predictions. This measures how precise the classifier is when predicting positive instances.

  \[
  \text{Precision} = \frac{TP}{TP+FP}
  \]

- **ROC curve**: Receiver Operating Characteristics. Is used for visual comparison of classification models which shows the tradeoff between the true positive rate and the false positive rate. The area under the ROC curve is a measure of the accuracy of the model. When a model is closer to the diagonal, it is less accurate, and the model with perfect accuracy will have an area of 1.0. An example of a ROC curve can be shown in figure 15 below.
There is a variety of metrics which can be derived from the confusion matrix and is useful for multi-class problems as well.

*Figure 15: Example of a ROC curve*
Chapter 6: Locality Sensitive Hashing – LSH

Locality-sensitive hashing (LSH) reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same “buckets” with high probability (the number of buckets being much smaller than the universe of possible input items).\(^{23}\)

6.1 Introduction

Locality sensitive hashing (LSH) reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same “buckets” with high probability (the number of buckets being much smaller than the universe of possible input items). Locality-sensitive hashing has much in common with data clustering and nearest neighbor search. LSH was initially introduced by Alexandr Andoni in 2005. We can think of applications like finding duplicate or similar documents, audio/video search. Although using brute force\(^{24}\) to check for all possible combinations will give you the exact nearest neighbor, but it is not scalable at all. Approximate algorithms to accomplish this task has been an area of active research. Although these algorithms do not guarantee to give us the exact answer, more often than not they will provide a good approximation. These algorithms are faster and scalable.

![Comparison of general and locality sensitive hashing techniques.](image)

LSH has much application, including:

- Near-duplicate detection: LSH is commonly used to deduplicate large quantities of documents, web pages, and other files.
- Genome-wide association study: Biologists often use LSH to identify similar gene expressions in genome databases.
- Large-scale image search: Google used LSH along with PageRank to build its image search technology VisualRank.
- Audio/video fingerprinting: In multimedia technologies, LSH is widely used as a fingerprinting technique A/V data.

Another application of LSH is that it can help in a Link Prediction Problem like does on this work. LSH can be used in order to find similar documents and suggest an edge between two nodes if the similarity of the two documents is over a given threshold. In order to understand how this algorithm

---


\(^{24}\) Brute force: The method for checking item per item on a list of k items. It has a time complexity \(O(k^2)\).
works; we need to understand first how the Jaccard index\(^{25}\) is calculated for two sets. We will speak about this topic in the next subsection.

### 6.2 Jaccard Index

The Jaccard index is the intersection over the union. We count the number of common items from two sets and divide by the number of elements that belong to either the first set, the second set or both. Let’s imagine that we have two set of words, set A and set B. The Jaccard Index for these two sets is calculated by the following equation:

\[
\text{Jaccard Index} = \frac{|A \cap B|}{|A \cup B|}
\]

We can conclude that the bigger the Jaccard index, the more probable it is that two sets A and B are similar. However, in order to be able to say something like this for those sets, we need to define a similarity threshold and only if the Jaccard Index of these two sets over the similarity threshold, the sets are identified as similar. Suppose A: “Nadal” and B: “Nadia”, then 2-shingles\(^{26}\) representation will be:

A: \{Na, ad, da, al\} and B: \{Na, ad, di, ia\}

We can calculate the Jaccard Index for those two sets which are shown on image 14, and we get the result of:

\[
\text{Jaccard Index} = \frac{2}{6} \Rightarrow \text{Jaccard Index} = 0.3333
\]

---

\(^{25}\) The Jaccard index, also known as Intersection over Union and the Jaccard similarity coefficient (originally coined coefficient de communauté by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. Source: Wikipedia

\(^{26}\) Shingling: The process in which, we convert each document into a set of characters of length \(k\) (also known as \(k\)-shingles or \(k\)-grams).
Some common shingles will result in more significant Jaccard Index and hence more likely that the documents are similar.

6.3 General idea

LSH refers to a family of functions (known as LSH families) to hash data points into buckets so that data points near each other are located in the same buckets with high probability, while data points far from each other are likely to be in different buckets. This makes it easier to identify observations with various degrees of similarity. The general idea of LSH is to find an algorithm such that if we input signatures of 2 documents, it tells us that those two documents form a candidate pair or not, i.e. their similarity is greater than a threshold t.

6.3.1 Specifically, for min-hash signature matrix

There are two necessary steps. First, we hash the columns of the signature matrix N using several hash functions and second, we have to check if two documents hash into the same bucket for at least one of the hash functions, we can take the two documents as a candidate pair. In our situation, for link prediction between two papers, if the Jaccard index of papers is bigger of a given threshold, we can conclude that this pair of documents (papers), is a candidate pair. So, we can add the edge between the papers. A question that arises now is how we can create different hash functions. The solution to this question is described in the next subsection, and it is called “Band Partitioning.”

6.3.2 Band partitioning for LSH

There are a few steps to create those hash functions. In the first step, we divide the signature matrix into b bands with each of this band having r rows. To continue, we have to hash each portion of each column to a hash table of k buckets for each band. Candidate column pairs are those that hash to the same bucket for at least one band. Then if we need to find the most similar pairs but few in similar pairs, we must tune the band r parameters. The following image (figure 18), is describing the above logic.
After all that, there are a few considerations which we have to resolve. Impeccable, for each band, we need to take $k$ to be equal to all possible combinations of values that a column can take within a zone. This will be equivalent to identity matching. However, in this way $k$ will be a huge enough number which is computationally time-consuming. We have to stay to the idea which says that if two documents are similar, then they will appear as candidate pair in at least one of the bands.

Selection of bands and rows for band partitioning

Choosing correct numbers for $b$ (bands) and $r$ (rows) is crucial for our results. If we choose a large number for $b$, it is the same as we are choosing more hash functions and we reduce the $r$ by this way because the $b*r$ is a constant value which is the number of rows in a signature matrix. This selection means that we are increasing the probability of finding a candidate pair and it is the same as choosing a smaller $t$ (similarity threshold). We will give a short example to understand this. Let us consider that our signature matrix has 100 rows. Imagine two different cases:

- $b_1 = 10 \Rightarrow r = 10$
- $b_2 = 20 \Rightarrow r = 5$

On the second case, it is more possible for the two documents to hash in the same bucket at least one time as they have more choices (20 over 10) and fewer items of the signature are getting compared (5 over 10). In other words, we can conclude that a higher $b$ implies a lower similarity threshold (higher false positives) and a lower lower $b$ implies to higher similarity threshold (more false positives). LSH is a powerful algorithm and reduces the search time while increases the memory efficiency but we have to consider the collisions which it can produce in case of similar items.

---

27 Image 15 source: https://util-forever.tistory.com/entry/Chapter-3-Finding-Similar-Items
6.4 Spark’s implementation for LSH

Apache Spark has implemented two algorithms for Locality Sensitive Hashing (LSH) which are commonly used in clustering\(^ {28}\), approximate nearest neighbor\(^ {29}\) search and outlier detection\(^ {30}\) with large datasets. Spark provides different LSH families in separate classes (MinHash for example), and APIs are also provided for features transformation in each class.

6.4.1 LSH operations

As is described in Spark’s documentation, LSH has three major types of operations:

- **Feature Transformation**: This is the necessary functionality to add hashed values as a new column. It can also be used for dimensionality reduction. Sparks LSH implementation supports multiple hash tables by setting the ‘numHashTables.’ The output column of a feature transformer as a type of Seq[Vector] where the dimension of the array (Seq) equals the ‘numHashTables’ value and the dimension of the vectors are set to one.

- **Approximate Similarity Join**: It is a method which takes to datasets and approximately returns pairs of rows in the datasets whose distance is smaller than a user-defined threshold. This operation also supports similarity joins for different datasets and a self-join.

- **Approximate Nearest Neighbor Search**: This is an operation whose function takes a dataset and a key, and it approximately returns a specified number of rows from the dataset that is closest to the principal vector.

On figure 19 below, we can see the difference in the execution file for both Approximate Nearest Neighbor (ANN) and Approximate Similarity Join (ASJ) compared with a full scan of a dataset.

\(^{28}\) Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). Source: Wikipedia

\(^{29}\) Nearest neighbor search, as a form of proximity search, is the optimization problem of finding the point in a given set that is closest to a given point. Source: Wikipedia

\(^{30}\) Novelty and Outlier Detection. Many applications require being able to decide whether a new observation belongs to the same distribution as existing observations (it is an inlier), or should be considered as different (it is an outlier). Often, this ability is used to clean real data sets. Source: scikit-learn.org
6.4.2 LSH Algorithms

Sparks implementation for LHS provides two different algorithms for two different distance metrics:

- **Bucketed Random Projection for Euclidean Distance:** We can run an instance of this algorithm by setting the following parameters as of our choice.

  ```scala
  val brp = new BucketedRandomProjectionLSH()
  .setBucketLength(2.0)
  .setNumHashTables(3)
  .setInputCol("keys")
  .setOutputCol("values")
  ```

- **MinHash for Jaccard Distance:** It is an LSH family for Jaccard distance where input features are sets of natural numbers. We can create an instance of the Minhas LSH algorithm by setting the following parameters as of our choice.

  ```scala
  val mh = new MinHashLSH()
  .setNumHashTables(3)
  .setInputCol("keys")
  .setOutputCol("values")
  ```

---

31 Image 19 source: Detecting Abuse at Scale: Locality Sensitive Hashing at Uber Engineering
Chapter 7: Experimental setup and results

In this section, we will describe our experimental methods which we used and the upcoming results both for supervised and unsupervised machine learning implementations for the Link Prediction Problem. We will go through the code, to explain each step of our implementation while we will discuss the results.

7.1 Explanation of the data sets

In order to understand our process and how we are going to work with this data, we must first understand its structure. We can imagine our problem as a network described by a graph in which the nodes (vertices) are the actual papers with its original attributes and the links (edges) between the nodes will exist only if two papers are referencing each other, irrespectively of the network direction. So, in table 2, we can see the paper’s attributes and a short description of them.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_id</td>
<td>A unique id for each paper</td>
</tr>
<tr>
<td>year</td>
<td>The publication year of the paper</td>
</tr>
<tr>
<td>title</td>
<td>The title of the paper</td>
</tr>
<tr>
<td>authors</td>
<td>The authors of the paper</td>
</tr>
<tr>
<td>journal</td>
<td>The journal of the paper</td>
</tr>
<tr>
<td>abstract</td>
<td>The abstract of the paper</td>
</tr>
</tbody>
</table>

*Table 2: Description of the data set attributes*

As we observe, each paper has six attributes which can be used for the feature engineering process. The paper’s dataset is a CSV file (comma separated), and each row of it represents a node. It is worth mentioning that the authors could have more than one authors and, in this file, papers with multiple authors have on this attribute the authors enclosed inside two double quotes, separating them with a comma. This particularity needs more attention to the feature extraction process. One other file which was used is the edge list file. This file is a .txt file and holds two ids of papers on each row. A row represents an edge between the two papers which are identified by their ids. These two ids are tab separated. This file is our ground truth, and we will need it in unsupervised methods. Below, we can see the file structure.

```
1001 1002
1001 1003
1020 76583
...
```

For the supervised methods, we used two other files. The first one is the training set and the second one is the test set. Those two files have almost the same structure with the edge list file (ground
truth) except that they have one more column with the label filled with zeros or ones based on the papers co-citation as it was already mentioned above.

7.2 Addressing the Link Prediction Problem as a classification problem

Our target is to predict a citation from a paper to another paper. As we already have a data set of papers with its attributes and a data set which represents an edge list of the paper network, we now need to predict future links at time t which will be created at the time t+1. This is a Link Prediction Problem, and it has only two labels which need to be predicted. The labels will be an integer number of 0 or 1.

- 0: The papers will not cite each other
- 1: The papers will cite each other

This problem can be treated as a binary classification problem. In the next sections, we will describe our datasets and the machine learning models which were used in our implementation as long as we will discuss the final results of our experiments.

7.2.1 Supervised the proposed approach

In these sections, we will discuss the supervised machine learning approach which we used to solve the Link Prediction Problem. As we mention in previous sections, we have to predict if a paper is cited or if it will cite another paper. There is a lot of different approaches and models which we can use to solve this problem. In our implementation, we experimented with various machine learning classifiers in order to predict an edge between two different papers based on node attributes and some structural attributes. In the next subsections, we analyze the whole process for each different model while we will describe the various parameters with which we experimented.

7.2.2 Environment setup

The whole application was built in Scala 2.11.8 with Apache Spark 2.4.0. The Java JDK was version 1.8. In order to accomplish the prerequisites of our work, we used four library dependencies from Spark (spark-core, spark-graphx, spark-SQL, and spark-mllib) and we tried two different external libraries which were open sourced in GitHub which offers different string metrics. Those two libraries are offered from vickumar1981 named string distance\(^{32}\) and from rockymadden named string metric\(^{33}\). The first step was to read out our data which process will be described in the next section.

---

\(^{32}\) String distance library from vickmar1981: https://github.com/vickumar1981/stringdistance

\(^{33}\) String metric library from rocky madden: https://github.com/rockymadden/stringmetric
7.2.3 Reading the dataset

The first thing we faced was to read the dataset. As we already mentioned in previous chapters, we used a CSV file (text file, comma separated) from which we had to create our nodes. One node is a single paper with its different attributes. We have already described those attributes in table 1, above. The second files we used were the training and test sets. We read those 3 data sets, and we created three separate data frames one for each of them. The Scala code of this process for the node data frame is shown below.

```scala
println("Reading node_information.csv as dataframe with name nodeDf...")
val nodeDf = spark.read.option("header", false).schema(schemaStruct).csv("resources/data/node_information.csv")
```

The next file was the training set which is a txt file separated by tab characters. So for that reason, we read this file with another easy way.

```scala
println("Reading training_set.txt...")
val data = sc.textFile("resources/data/training_set.txt").map(line =>
  val fields = line.split(" ")
  (fields(0),fields(1), fields(2).toInt)
)
```

As we see on the code block above, this file holds three values. First, the node’s one id, second the node’s two ids and last, the label. We will not say more about this file as we already explained it on the previous section.

7.2.4 Feature engineering

In these sections, we will discuss and describe the whole process of our feature extraction and selection. The next step after we read the necessary files was to make a self-join of the data frame of nodes (papers). This step was required in order to able us in creating a new column into the joined data frame, with the Jaccard similarities of each paper with each other. This operation (self-join) was too costly because we had to calculate tuples for each paper with all other papers. In other words, we had a data frame of papers with 27770 nodes and the total number of the generated tuples was calculated by the following equation:

\[
Total \text{ tuples} = n! / k! \ast (n-k)!
\]

Where n, in our case was 27770 and k = 2. The total number of the calculated tuples is substantial, so it needs time to be calculated. After the self-join of the nodes data frame, was time for feature
The first feature was to calculate the time difference of publication between the first and the second paper for each row. This is a simple feature, and it is not so time-consuming. Another feature was to calculate the PageRank for each node of the graph. This feature is more expensive than the time difference. In the end, it was not used in the final features set as its contribution was found to be near to zero percent. One other feature which we added in our features set, was the Jaccard Index for the two papers. As we already know, the bigger this index is, the more similar are the two documents. The next feature belongs to the structural properties of the graph. It is called common neighbors. Spark’s GraphX library has already implemented an algorithm for finding common neighbors in a given graph. We used this implementation to create a new column in our features data frame. It is worth to say that this feature has increased the accuracy and F1 score of our model as much as possible. In numbers, our models gained an accuracy boost from 0.1 to 0.3 percent and F1 score from 0.05 to 0.12 percent. One more feature from structural graph properties was to find in how many triangles the two nodes of a table, belongs. There is also a Spark implementation in GraphX for finding triangles and count them. Its name is TriangleCount. After that, it was time for some preprocessing. As we mentioned on previous sections, on papers, expect its id holds some more information. For every paper, we know that it has a title, publication year, authors, journal and it is abstract. From these attributes, we can add four more features. Overlap for titles, authors, journal and abstract. Such as the structural features, these four overlapping features add to the final results but less than them. In order to find the overlap between two different papers attributes, we had to split this column into words and find the sum of its intersection. This process achieved via creating and applying a user-defined function (UDF35) in which we use the split functions from SQL. We define how to split the input string by passing a parameter as a character or as a Regular Expression. In the code block below we can see how we add a new column in our data frame with those split words.

```java
joinedDf.withColumn("journal_from_words", functions.split(col("journal from"), \"\s+\"))
```

With this line of code, we add a new column via withColumn() function in joinedDf with the name “journal_from_words,” and we assign how the split will occur. Now, we remove the stopwords in orders to clean up our data. We do not new words like, “me, not, it, etc.” It is useful to do this step because we are using a term frequency, so we do not want to increase those frequencies by a big number of stopwords. In order to remove stopwords, we used feature transformer from Spark which called StopwrodsRemover, and it is aimed to do the pre-mentioned work. On the code block below we can see how to use it:

---

34 Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive. Source: Wikipedia

35 User-Defined Functions (aka UDF) is a feature of Spark SQL to define new Column-based functions that extend the vocabulary of Spark SQL’s DSL for transforming Datasets. ... You define a new UDF by defining a Scala function as an input parameter of udf function. It accepts Scala functions of up to 10 input parameters.

---
After we have prepared our dataset, we need to select which of our features we will keep in order to do that we need to run a Squared test of independence\textsuperscript{36}. Spark also provides a feature selector for this test, called. In the following code lines, we can see a short example of how to use this selector.

```scala
val remover2 = new StopWordsRemover().setInputCol("title_to_words").setOutputCol("title_to_words_f")
joinedDf = remover2.transform(joinedDf)
```

```
val df = spark.createDataset(data).toDF("id", "features", "clicked")
val selector = new ChiSqSelector()
  .setNumTopFeatures(1)
  .setFeaturesCol("features")
  .setLabelCol("clicked")
  .setOutputCol("selectedFeatures")
val result = selector.fit(df).transform(df)
```

It supports five choosing methods: numTopFeatures, percentile, fpr, fdr, fwe. numTopFeatures chooses a fixed number of top features according to a chi-squared test. This is akin to yielding the features with the most predictive power. Percentile is similar to numTopFeatures but chooses a fraction of all features instead of a fixed number. Fpr chooses all features whose p-values are below a threshold, thus controlling the false positive rate of selection. Fdr uses the Benjamini-Hochberg procedure to choose all features whose false discovery rate is below a threshold. Fwe chooses all features whose p-values are below a threshold. The threshold is scaled by 1/numFeatures, thus controlling the family-wise error rate of selection. By default, the selection method is numTopFeatures, with the default number of top features set to 50. We can choose a selection method using setSelectorType.

### 7.2.5 Machine learning models

In this section, we describe the different machine learning modes and its parameters which we used in our work. All of our used models were described in the sections above. Here we will describe the properties with which we experimented. Our first model was the Naive Bayes classifier. On this classifier, we experimented with the threshold parameter. After a few evaluations, we decide to keep the threshold values to 50%. In this test, we split the training and test set to 70% and 30% respectively. The results for this split of the dataset and with the full features above are shown in table 3 below:

\textsuperscript{36} A chi-squared test, also written as $\chi^2$ test, is any statistical hypothesis test where the sampling distribution of the test statistic is a chi-squared distribution when the null hypothesis is true. Source: Wikipedia
**Naïve Bayes Classifier**

<table>
<thead>
<tr>
<th>Dataset Split</th>
<th>Evaluator</th>
<th>Accuracy</th>
<th>F1</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>70/30</td>
<td>Multiclass Classification Evaluator</td>
<td>0.58614</td>
<td>0.58876</td>
<td>1090.06</td>
</tr>
</tbody>
</table>

*Table 3: Naïve Bayes Classifier results*

For the evaluation of the machine learning models, we used the Multiclass Classification Evaluator which provides different metrics by passing just the metric name as a parameter. The score of accuracy and F1 was near to 60%, and it was quite slow. It was not very good, so we needed to try another model to increase our results. So, the next model was the Logistic Regression Classifier. With this classifier, we “played” with more attributes but the most useful was the Max Iterations. We experimented from 10 to 10000 iterations and with different feature sets. Our first try was with just four features:

- Titles overlap
- Titles Jaccard index
- Authors overlap
- Journal overlap
- Abstract overlap and
- Time diff.

The results for those features and the different max iteration values are shown in table 4 below:

<table>
<thead>
<tr>
<th>Dataset Split</th>
<th>Max. Iterations</th>
<th>Accuracy</th>
<th>F1</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>70/30</td>
<td>10</td>
<td>0.79890</td>
<td>0.79812</td>
<td>1694.76</td>
</tr>
<tr>
<td>70/30</td>
<td>100</td>
<td>0.79890</td>
<td>0.79947</td>
<td>1654.78</td>
</tr>
<tr>
<td>70/30</td>
<td>1000</td>
<td>0.79713</td>
<td>0.79957</td>
<td>1628.16</td>
</tr>
<tr>
<td>70/30</td>
<td>10000</td>
<td>0.79723</td>
<td>0.79778</td>
<td>1807.94</td>
</tr>
</tbody>
</table>

*Table 4: Accuracy and F1 vs. maximum iterations*

The best result from the above experiments was for the 100 max iterations, based on execution time, with an Accuracy of 79.89% and an F1 score of about 79.94%.

![Accuracy based on iterations](image)
Figure 21 shows the F1 score for the same values. As we can observe, we gain a more significant F1 score for the 1000 iterations and in almost the same execution time.

![F1 score based on iterations](image)

*Figure 21: Logistic Regression - F1 score based on the number of iterations*

Those results are better than the Naive Bayes one, but it can be much better if we add structural features. So, we rerun this model, now with all our available features which we have already described in previous sections. The results are shown below in Table 5.

**Logistic Regression Classifier**

<table>
<thead>
<tr>
<th>Dataset Split</th>
<th>Max. Iterations</th>
<th>Accuracy</th>
<th>F1 score</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>70/30</td>
<td>10</td>
<td>0.93518</td>
<td>0.93559</td>
<td>959.72</td>
</tr>
<tr>
<td>70/30</td>
<td>100</td>
<td>0.93561</td>
<td>0.93600</td>
<td>1002.28</td>
</tr>
</tbody>
</table>

*Table 5: Logistic Regression Classifier Results*

It is clear enough that the scores are more significant both for accuracy and F1, with the whole feature set. From those results, we can conclude that the structural features improve our results. It is worth to say that the execution time becomes better with those features in comparison with the old features set. The next figure 22 shows the difference in accuracy and f1 score for the 10 and 100 iterations of the algorithm:
One other classifier which we used was the Linear SVM. It is a linear method as described in the sections above. We this classifier we experimented with two basic parameters. First, the max iterations of the algorithm and second, the regParam. It is a regularization parameter, and we can find more about that on the official documentation of Linear SVM, on Apache Spark website. The results for the different values for each of max iterations and the regParam are shown in table 6.

**Linear SVM**

<table>
<thead>
<tr>
<th>Max Iterations</th>
<th>RegParam</th>
<th>Accuracy</th>
<th>F1 score</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.1</td>
<td>0.85967</td>
<td>0.85967</td>
<td>934.15</td>
</tr>
<tr>
<td><strong>100</strong></td>
<td><strong>0.1</strong></td>
<td><strong>0.88044</strong></td>
<td><strong>0.88152</strong></td>
<td><strong>1124.26</strong></td>
</tr>
<tr>
<td>10</td>
<td>0.3</td>
<td>0.84362</td>
<td>0.84355</td>
<td>893.23</td>
</tr>
<tr>
<td>100</td>
<td>0.3</td>
<td>0.85683</td>
<td>0.85821</td>
<td>1313.11</td>
</tr>
</tbody>
</table>

*Table 6: Linear SVM Classifier Results*

We achieve the best results in Accuracy and F1 score, based on our implementations, with the combination of the values 0.1 for regParam and 100 for the maximum algorithm iterations. We can observe that for the scope of this work and our currently selected feature set, Linear SVM does not get good enough results in comparison with Logistic Regression Classifier. After support vector machines, we tried our fourth classifier which was the Multilayer Perceptron Classifier. MLPC is a classifier based on the feedforward artificial neural network. MLPC consists of multiple layers of nodes. Each layer is fully connected to the next layer in the network. With this classifier, we were expecting much better results as it uses a neural network. We tried different values, for different parameters. Some of the most important parameters with which we experimented, was the max iterations of the algorithm and the layers of the network. The results for those parameters are shown below in Table 7.

---

37 *The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. Source: Wikipedia.*
Multilayer Perceptron Classifier

<table>
<thead>
<tr>
<th>Max Iterations</th>
<th>Layers</th>
<th>Accuracy</th>
<th>F1 score</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>13,10,7,2</td>
<td>0.87953</td>
<td>0.87951</td>
<td>1007.67</td>
</tr>
<tr>
<td>200</td>
<td>13,10,7,2</td>
<td>0.94770</td>
<td>0.94776</td>
<td>1106.78</td>
</tr>
<tr>
<td>400</td>
<td>13,7,4,2</td>
<td>0.95187</td>
<td>0.95205</td>
<td>1347.12</td>
</tr>
</tbody>
</table>

Table 7: Multiclass Perceptron Classifier Results

There are some works which are explaining the different ways in a layer selection. The first layer is equal to the number of our features, and the last one is similar to the number of cases (classes) which we are trying to predict. We will not describe the operation of layers selection here as we will focus on results. We earn the best score in Accuracy and F1 score for 400 iterations and the set of layers {13, 7, 4, 2}. The Accuracy of the classifier was 95.18% while the F1 score was 95.2%. Although the results were quite good with Perceptron, we tried two more classifiers based on trees. The first was the Decision Tree Classifier and the last one, the Random Forest Classifier. For the Decision Tree Classifier, we tested one parameter to achieve the best possible results. This parameter was the maximum depth in the tree for search. Table 8 below shows the variation in its percentages in Accuracy and the F1 score.

Decision Tree Classifier

<table>
<thead>
<tr>
<th>Max Depth</th>
<th>Accuracy</th>
<th>F1 score</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.95116</td>
<td>0.95129</td>
<td>1302.87</td>
</tr>
<tr>
<td>8</td>
<td>0.95300</td>
<td>0.95314</td>
<td>1308.23</td>
</tr>
<tr>
<td>16</td>
<td>0.94262</td>
<td>0.94279</td>
<td>1177.16</td>
</tr>
<tr>
<td>30</td>
<td>0.92497</td>
<td>0.92494</td>
<td>1342.28</td>
</tr>
</tbody>
</table>

Table 8: Decision Tree Classifier Results

As we observe, from the table above, we tried a variety of different maximum depth values, but the most accurate was for the number of 8. This classifier achieved much better results from the pre-mentioned classifiers and in a reasonably satisfactory time concerning the volume of our data. Finally, the last one classifier which we experimented with, was the Random Forest Classifier. For this classifier, we used two values to parameterize them. Those parameters were, the maximum depth such as the Decision Tree Classifier and the second one was the number of trees. The final results for this classifier and with those two attributes were shown in table 9 below.

Random Forest Classifier

<table>
<thead>
<tr>
<th>Max Depth</th>
<th>Num. Trees</th>
<th>Accuracy</th>
<th>F1 score</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10</td>
<td>0.95066</td>
<td>0.95077</td>
<td>1314.01</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>0.95580</td>
<td>0.95591</td>
<td>1191.91</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>0.95058</td>
<td>0.95068</td>
<td>1262.46</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>0.95527</td>
<td>0.95538</td>
<td>1230.55</td>
</tr>
</tbody>
</table>

Table 9: Random Forest Classifier Results

Random Forest Classifier managed to achieve the highest results concerning all other classifiers with the same features in an adequate time (1191 sec). The Accuracy reached 95.58% while F1
approached marginally 96%. It is worth mentioning that the algorithms were executed on a 4-core computer and an 8-GB memory. Those times could be much better running on a cluster taking advantage of 100% of the Apache Spark’s distributed processing benefits.

7.2.6 Comparison of the classifiers

Summarizing the results from the above classifiers and based on their different parameters, we can conclude as to their accuracy, F1 score and the overall execution time. The figure below shows the change in Accuracy and F1 concerning the classifiers.

![Figure 23: Comparison of Accuracy and F1 score per Classifier](image)

Based on the results from the figure 23 above, we can conclude that Random Forest Classifier is the best choice for the Link Prediction Problem with our pre-described dataset and can achieve an Accuracy of 95.58% in predicting a new paper will cite and other paper in the future. Also, it has a big enough F1 score which means that our classifiers Precision and Recall are balanced. The next figure 24 combines the classifiers with their execution time.

![Figure 24: Execution Time per Classifier](image)
So, based on the figure above, we can see that the fastest classifier was the Linear Regression classifier with an execution time of 1002.28 sec and the slowest one was the Naïve Bayes classifier which was 652 sec slower than the first. We gain a more detailed image if we look at table 25 below.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy (%)</th>
<th>F1 score (%)</th>
<th>Exec. Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bayes</td>
<td>79.89</td>
<td>79.94</td>
<td>1654.78</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>88.04</td>
<td>88.15</td>
<td>1124.26</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>93.56</td>
<td>93.60</td>
<td>1002.28</td>
</tr>
<tr>
<td>Multi. Perceptron</td>
<td>95.18</td>
<td>95.20</td>
<td>1347.12</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>95.30</td>
<td>95.31</td>
<td>1308.23</td>
</tr>
<tr>
<td>Random Forest</td>
<td>95.58</td>
<td>95.59</td>
<td>1191.91</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>91.25</strong></td>
<td><strong>91.29</strong></td>
<td><strong>1271.43</strong></td>
</tr>
</tbody>
</table>

*Table 10: Comparison of Classifiers based on Accuracy, F1 and Execution Time*

Observing the results shows that Random forest has the highest possible accuracy and F1 score for the data and the problem as we have set it, while it is also much faster than the average. Finally, we select Random Forest for our classifiers because it is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms because it is simplicity and the fact that it can be used for both classification and regression tasks.

7.3 Treat link prediction problem as an unsupervised machine learning task

In this chapter, we will study the problem of link prediction for the same network as in the previous chapter, but from the perspective of the unsupervised machine learning. We will try to predict links without knowing their existence from the beginning. We will start with the simplest method of brute force, and we will complete our study of our problem with a more efficient algorithm based on the MinHashig technique for predicting the candidate pairs, the MinHashLSH algorithm.

7.3.1 Unsupervised proposed approach

The technique we have followed at this stage of studying the problem to predict possible links between the various records is well established in a relatively simple procedure. The key idea is to convert all the attributes of a node into words and to put them together in one set (bag of words). Then, we perform some basic text preprocessing before running our algorithms. As we said above, in order to create a bag of words, we must first break the original text into them. This can be achieved by using Tokenization as we already have done. Next, we will remove stop words so that we avoid a large Jaccard Similarity dice that will be due to these words. We want as much as possible the words that will remain in the bag to be as diverse as possible, so if a high Jaccard
Similarity for a pair of records is given, then this will probably give us a greater chance of having found a possible link. We tried two different but so commonly used techniques. The first one is the brute force pairwise check and the second, the algorithm MinHashLSH which is based on MinHashing for finding the similarity between the documents. In the next section, we will speak about the brute force technique.

7.3.2 Brute force pairwise check

Once we have finished with the pre-processing of our data, we continue with the structure of the data which will enter into the algorithms for processing. First, we calculate all the possible combinations that may arise concerning the size of our initial nodes. Suppose for example that we have N nodes, the number of all possible combinations that can occur is given by the following relation:

\[ N = \frac{N*(N-1)}{2} \]

After we have computed all the pairs, we create a data frame which will contain in each of its lines a pair. For each line, we add two new columns containing the bag of words which corresponds to the two nodes of the pair. The next step is to calculate the Jaccard Index (similarity) for each row of the data frame, based on the two bags of words. Finally, we append in our data frame one new column for the Jaccard similarity. The algorithm goes on with the selection of the candidate pairs (row) which meet the minimum similarity threshold requirements. We choose this similarity based on some experimentations, from which the candidate pairs must have a bigger value in order to be selected. Now, for our dataset which consists of 27770 nodes and 352857 edges, a brute force algorithm must compute 385572565 Jaccard Indexes. This process is too time-consuming, and it is dependent on the RAM of the system. Brute force is accurate enough but too slow for big data. While we increase the volume of the data, the execution time of brute force increases too. In our dataset, the maximum value for the Jaccard index was 0.4973. We have tried different values for the similarity threshold, and after a few experiments, we decide that the best Jaccard similarity threshold was 0.17. Based on our threshold, the accuracy of the algorithm differs. On table 11 below we can see the execution time of brute force for a different number of cores.

<table>
<thead>
<tr>
<th>Execution Cores</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>432.12</td>
</tr>
<tr>
<td>4</td>
<td>208.56</td>
</tr>
<tr>
<td>8</td>
<td>161.04</td>
</tr>
<tr>
<td>16</td>
<td>148.63</td>
</tr>
<tr>
<td>32</td>
<td>123.03</td>
</tr>
<tr>
<td>64</td>
<td>127.38</td>
</tr>
</tbody>
</table>

*Table 11: Execution time for Brute Force algorithm per execution core number*
The above results refer to a subgraph of the original graph, namely for 2000 nodes and 59000 edges. As we observe, the execution time is decreasing while we increase the number of available cores. The next figure 25 shows the change in run time of brute force check for the pre-mentioned dataset, relative to the execution cores.

The brute force technique achieves fairly high accuracy but its execution time for a large volume of data is unprofitable. The table below shows the execution time of the brute force, for a different number of data. The following results on table 12 are related to the execution of the algorithm in a computing system with 32 cores and 64 GB of RAM. The threshold value from which the algorithm will select the candidate pairs is 0.17. The maximum threshold value of the dataset is 0.4855 or 48.55%.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Total checks (n * (n-1)) / 2</th>
<th>Candidate pairs</th>
<th>Accuracy</th>
<th>Execution Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>499500</td>
<td>3916</td>
<td>0.9368</td>
<td>62.89</td>
</tr>
<tr>
<td>2000</td>
<td>1999000</td>
<td>14055</td>
<td>0.9662</td>
<td>161.04</td>
</tr>
<tr>
<td>5000</td>
<td>12497500</td>
<td>74302</td>
<td>0.9711</td>
<td>566.73</td>
</tr>
<tr>
<td>7000</td>
<td>24496500</td>
<td>106534</td>
<td>0.9789</td>
<td>1446.98</td>
</tr>
</tbody>
</table>

Table 12: Total Checks, Candidate Pairs, Accuracy and Execution Time of Brute Force

As we observe, from the above results, the brute force technique is ok for small datasets and provides high accuracy. In our case, brute force took almost 63 seconds to calculate the Jaccard similarities for our sampled network with 1000 nodes. However, while we increase the number of the dataset entries, we see a big difference in the execution time and a minimal gain in the accuracy. We need 1384 seconds more to calculate Jaccard indexes for 7000 nodes, and we only gain 0.0421 or 4.21% more accuracy. Based on our resources, brute force has a limit on 7000 nodes due to the ram limitation. Figure 26 below shows the execution time of the algorithm based on nodes which have to calculate.
As we can observe, the execution time of the brute force increases rapidly while the dataset becomes bigger. So, it is not a technique we can use when handling a large amount of data. It is suitable only for small datasets and for light procedures. On figure 27, we can see the change of the accuracy of the brute force algorithm while we are increasing the dataset volume.

As can be seen from the above figure, the change in accuracy is too small. Therefore, the nature of the problem is which allows us to use or not this algorithm. However, in order to come to the conclusions about the use of the algorithm in our problem, we need to look at it altogether. So, we must observe how long our algorithm needs to reach a satisfactory point of performance. For this reason, in figure 28 below, we can see the change in accuracy concerning the algorithm’s execution time.
So, we realize that the more accuracy we want to achieve, the more time we need. The brute force technique achieves high accuracy as we said above, but it is entirely dependent on the available resources of our computing system and is very costly in terms of time. We could use a lower similarity threshold to achieve greater accuracy, but this also entails a longer execution time as the number of candidate pairs increases and more computational calculations are required. So, we need a better approach than brute force. We need some algorithm which will need less time to proceed even if it has smaller accuracy. In the next section, we will analyze MinHashLSH and all the steps we have done in order to predict link with approximation but in a shorter time.

7.3.3 MinHashing and Locality Sensitive Hashing

Performing pairwise comparisons in a corpus, as we have shown in the previous section, is time-consuming because the number of comparisons grows geometrically with the size of the data. Most of those comparisons, furthermore, are unnecessary because they do not result in pairs. The combination of MinHash and locality-sensitive hashing (LSH) seeks to solve this problem. They make it possible to compute all the possible matches only once for each document so that the cost of computation grows linearly rather than exponentially. The key idea of Locality Sensitive Hashing is to use a group of functions (also known as LSH families) to hash data points into buckets so that data points near each other are located in the same buckets with high probability, while data points far from each other are likely in different buckets. LSH is a technique for effectively reducing the search space when looking for similar pairs and aims to reduce the running time of this procedure. The idea is to convert vectors into a fixed-size bit sequence, called a signature, such as that similar vectors are likely to have similar signatures.

Approach

At this point, we will analyze the methodology we applied with MinHashLSH for faster link prediction based on obstetric techniques. In the first step, such as in the brute force, we concatenate all of our dataframe, columns into one column which will hold all the available data that we have.
for our papers. Before that, we have replaced null values with an empty string. The next step was to create term frequency vectors from our tokenized dataset, to give them as input to the MinHashLSH. In order to create these term frequency vectors, we have used HashingTF with 10000 features. When our feature vector was ready, we gave it as an input to MinHashLSH. This algorithm very roughly functions as follows. The input is a set of documents, and the output is tables (only one shown) whose entries contain the documents. The documents in the same entry are supposed to have a similarity between themselves of at least a threshold value s. Documents that are indistinct entries are supposed to have a similarity below s. LSH is not exact. It has false positives and false negatives. False positives occur when a pair is on the same entry but is below the threshold. False negatives occur when a pair is above the threshold but on distinct entries. Figure 29 below demonstrates this functionality.

LSH is, therefore, a function that maps similar documents to the same value and unsimilar documents to different values. Where similar depends both on the metric being used and on the threshold s. We should notice that the resulting structure considerably reduces the computational complexity of the problem. Back to our implementation, when we get the output of the MinHashLSH (which was the hashed documents), we used another algorithm to create the candidate pairs (our links). This algorithm called approximateSimilarityJoin. This function takes as input, our features dataframe (the same for a self-join or two different) in order to make a join on a specified distance threshold. MinHashLSH works with Jaccard Similarity while approximateSimilarityJoin works with Jaccard distance. As in brute force, in this algorithm, we have to set a similarity threshold so that we can get pairs of papers that meet these requirements. However, if we want to get pairs of 60% same papers, i.e., having a Jaccard index of 0.6, we need to set the distance threshold if it is equal to 1 - Jaccard index. Therefore, for 60% homogeneity in papers, they should have a distance of at most 0.4 or 40%. So, the Jaccard distance was described by the equation below.

\[
\text{Jaccard Distance} = 1 - \text{Jaccard Index}
\]

The last argument in this algorithm is the column name that will contain the Jaccard distances. Once we set the threshold, the approximateSimilarityJoin algorithm returns all those pairs that
have a Jaccard distance less than the threshold we set. To prevent pairing from the same paper, we took advantage of the filter() function to give away couples that would have a Jaccard distance equal to 0. This detail is required because papers with zero Jaccard distance come from the combination as a document with itself. This is assumed to occur, as the approximateSimilarityJoin algorithm combines all of the papers with everything else. The results of the MinHashLSH algorithm in combination with the approximateSimilarityJoin algorithm are faster than the results of the brute force algorithm, as not all the possible combinations need to be calculated but are not so accurate and reliable. In order to achieve more accurate results, we need to increase the number of hash tables. The more hash tables we have, to more accurate and reliable would be our results. It is essential to emphasize that as the number of hast tables grows, the more time we run our algorithm, as we increase its complexity. The table below shows the execution results of the MinHashLSH on a cluster with 32 cores and 64 GB of RAM for a subgraph of our network with 2000 nodes and a Jaccard distance threshold of 0.8. We could not run the algorithm with the whole dataset due to the hardware limitations. The following results on table13 are fully representative of the behavior of the algorithm.

<table>
<thead>
<tr>
<th>Hash Tables</th>
<th>Candidates</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F1</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>986</td>
<td>0.7261</td>
<td>0.0120</td>
<td>0.97133</td>
<td>0.02370</td>
<td>26.19</td>
</tr>
<tr>
<td>4</td>
<td>1610</td>
<td>0.7304</td>
<td>0.0385</td>
<td>0.97147</td>
<td>0.03853</td>
<td>31.12</td>
</tr>
<tr>
<td>8</td>
<td>3026</td>
<td>0.6265</td>
<td>0.0607</td>
<td>0.97147</td>
<td>0.06072</td>
<td>70.20</td>
</tr>
<tr>
<td>16</td>
<td>3628</td>
<td>0.5975</td>
<td>0.0364</td>
<td>0.97148</td>
<td>0.06877</td>
<td>111.48</td>
</tr>
<tr>
<td>32</td>
<td>3824</td>
<td>0.5983</td>
<td>0.0385</td>
<td>0.97147</td>
<td>0.07246</td>
<td>211.95</td>
</tr>
<tr>
<td>64</td>
<td>3840</td>
<td>0.5968</td>
<td>0.0385</td>
<td>0.97149</td>
<td>0.07246</td>
<td>514.83</td>
</tr>
<tr>
<td>128</td>
<td>3840</td>
<td>0.5968</td>
<td>0.0385</td>
<td>0.97151</td>
<td>0.07251</td>
<td>1344.72</td>
</tr>
</tbody>
</table>

Table 13: Statistic Results of MinHashLSH

We can very quickly see that as the number of cores in our computer system increases, Spark takes full advantage of the resources and automatically parallelizes our algorithm. By increasing the parallelization of the process we are running, we significantly reduce its execution time. In order to understand more about the increase of the parallelism of an algorithm and the simultaneous reduction of the time of execution of the work, we have created the diagram (figure 30) below which is illustrating the above task.

![Figure 30: Execution Time of MinHashLSH based on Execution Cores](image-url)
As we observe, in the diagram above, the runtime of our algorithms is significantly reduced as our system's available resources grow. However, MinHashLSH's execution time depends not only on the number of cores but also on another crucial factor. This factor is the number of hash tables. We are well aware that by increasing the number of these tables, we can improve the efficiency of our algorithm because we are increasing the probability of the algorithm to predict new connections correctly. According to the results of the above table, the following figure 31 shows the increase in the accuracy of the algorithm as the number of hash tables increases.

![Accuracy vs Hash Tables](image)

*Figure 31: Accuracy of MinHashLSH based on Hash Tables*

We see the gradual increase in accuracy as the number of hash tables increases. The rise is also applied to F1. F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Precision is the ability of our model to return only relevant instances while Recall is the ability to identify all relevant instances. Intuitively it is not as easy to be understood as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have a similar cost. If the cost of false positives and false negatives are very different, it is better to look at both Precision and Recall. In our case, the change between Precision and Recall while we are increasing the number of hash tables is shown in figure 32 below.

![Precision and Recall vs Hash Tables](image)

*Figure 32: Precision and Recall of MinHashLSH based on Hash Tables*
As we can observe, our model has a higher precision than recall. This translates as follows. LSH in combination with MinHashing and all our parameters (threshold and number of hash tables) has a greater ability to recognize the relevant instances, but it does not go so well in getting them back and everything. This means that it produces many false positives. After studying the change of Precision and Recall from the above figure, we can now move on with F1 curve in the following diagram (figure 33). Here, we are watching the change of the F1 score while we are increasing the number of hash tables. We have a good one for the number of the 16 hash tables.

![F1 vs Hash Tables](image)

*Figure 33: F1 score based on Hash Tables*

Something else that can be traced is the change in TP, TN, FP, and FN when raising the threshold. The diagram (figure 34) below shows this change for a subgraph of our original graph for a total of 1000 nodes and eight hash tables. The total number of comparisons is 499500.

![TP vs Jaccard Distance Threshold](image)

*Figure 34: TP of MinHashLSH according to Hash Tables*
From the graphs above we deduce the following. As far as TPs are concerned, their numbers are rising as the Jaccard Distance is reached. This is perfectly normal as if we think of the definition of Jaccard distance we understand that as the distance between papers increases, the similarity between them decreases. Therefore, their smaller the similarity, the more the combination of the paper, we receive. So this is the explanation for increasing the number of FPs. That is since we are getting more and more of a combination, the more likely it is to have a pair of papers that do not really exist. Finally, TN and FN are functions of the two metrics preceded.
7.4 Compare of Brute Force and MinHashLSH

When we have to deal with a small amount of data, brute force is a fresh and reputable solution. As this volume increases, however, the run time of this algorithm increases. The temporal complexity of brute force is $O(n^2)$, a factor that is not unremitting. However, if we removed the time, and we were only interested in the accuracy of our results, then and then only a brute force would be a good choice. In contrast, MinHashLSH has a much more gentle increase in runtime as the volume of data increases. Its time complexity is almost linear, which puts it dominating as a choice in terms of speed. From the other, it is not as accurate as brute force as it produces a large number of FPs. Summing up what we have just mentioned, we gathered them in a small test by performing both of these algorithms for a variety of nodes in our network, and we recorded their execution times. The results of this test are shown in Table 14 below and are visualized in the figure.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Brute Force Execution Time</th>
<th>MinHashLSH Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>69.98</td>
<td>76.43</td>
</tr>
<tr>
<td>2000</td>
<td>200.91</td>
<td>146.39</td>
</tr>
<tr>
<td>3000</td>
<td>554.71</td>
<td>362.21</td>
</tr>
<tr>
<td>4000</td>
<td>1148.54</td>
<td>602.42</td>
</tr>
<tr>
<td>5000</td>
<td>1686.13</td>
<td>834.56</td>
</tr>
<tr>
<td>6000</td>
<td>1992.38</td>
<td>1001.87</td>
</tr>
<tr>
<td>7000</td>
<td>2345.76</td>
<td>1231.34</td>
</tr>
</tbody>
</table>

*Table 14: Execution Times for Different Node Numbers*

We can conclude from the table 14 above that the difference in the execution time of the two algorithms is significant. Interestingly, for a small number of nodes, brute force does better but as the data volume increases, its execution time proliferates. In contrast, MinHashLSH has a more stable increase in execution time and tends to grow almost linearly. In figure 38 below, the difference between the two algorithms in their execution times seems more apparent.
7.5 Evaluation

In this section, we will speak about the method we used to evaluate these two unsupervised techniques, brute force, and MinHashLSH. We will not mention in detail the whole process which has been followed, but we will describe the general picture surrounding the evaluation of these two unsupervised techniques. Since we did not have any classifiers or regressor for evaluation, we had to evaluate two non-supervised machine learning techniques, the brute force algorithm and the MinHashLSH algorithm. These two algorithms cannot be evaluated as a supervised machine learning models, by merely using an evaluator. For this purpose, we had to create our evaluator. The procedure we followed was relatively simple but generally very time-consuming. The results presented in this work come from a subset of our original data. It was not possible to experiment with the full amount of data we had at our disposal, as we have had many hardware limitations. It is worth noting that for experimental purposes, ground truth was given from the beginning so that we can evaluate our results at the end and make a cross-correlation between these two algorithms. The ground truth that was given was a file that is composed of three columns. The first two refer to the ids of the source paper and the other to the destination paper. The third column in the series is one containing 0 and one where when it is 0 it means that both of these papers are not connected while when it is one means that there is an edge between them. In essence, each ground truth line is an edge of our original network of papers. Our experiments were performed for a certain number of nodes at a time. For this reason, and because we had to evaluate the accuracy and efficiency of each of the two algorithms, in the end, we kept only a subgraph of the original graph which will contain only the nodes which have been selected to run with the algorithms. For this reason, we filtered the original ground truth file and kept the subgraph for the selected nodes. Then after we had the final subgraph and the results from our algorithms, we combined them with the number of total calculations made by each of the algorithms. Next, we found that predictions of the algorithms which were within the ground truth subgraph and based on all of the above, we calculated the TPs, TNs, FPs, and FNs. At this point, and having calculated the above, we computed Precision, Recall,
and Accuracy. Finally, through the two previous metrics, we also calculated the F1 measure. All the process mentioned above has been encapsulated in a function so that it can be used with other algorithms which produce results in the same form.

7.6 Conclusion

Imagine that we have 1 million documents in a database. For a new incoming document, we may want to know which of this million is the most similar to it or such as in our case we want to predict new links between documents based on a similarity threshold. The brute force approach would be to make 1 million comparisons. With LSH however, we would have these hash tables saved somewhere (one for each band). Their generation would have had to occur beforehand, but only once. Then, we would apply the same LSH functions to this new document. For the MinHash case, this would require using the same permutations in the same order. The new document would ultimately be mapped to a given index in the hash table. Finally, one would compare the new document only to the other documents present in that same entry of the hash table (for each hash table representing a band). Similarly, if the objective is to compare all documents between themselves, then one would now only need to compute \( \text{sim}(x,y) \) for each \((x, y)\) pair which shared an index in any of the hash tables (bands). In general, as we have seen in the previous chapters, the brute force technique is very costly in time but has the advantage of offering greater accuracy. On the other hand, the MinHashLSH algorithm roughly approximates the similarity of documents, producing less accurate results and requires almost linear runtime. There are several techniques to increase its accuracy, for example by increasing the number of hash tables, but also increasing the run time. However, there are applications where predictions against an error up to a limit are acceptable. Regarding the problem of link prediction, the brute force goes well enough but takes much time to execute it. On the other hand, the MinHashLSH algorithm is far faster than brute force and also achieves a high value of accuracy in its results. Therefore, it is a unique appeal to the problem of link prediction.
Chapter 8: Future work

In the future, we will address the link prediction problem through a different viewpoint. We will re-examine the same network but this time with a cluster-based technique. This approach uses similar nodes from a cluster and aims that nodes from the same cluster exhibit a similar connectivity pattern. More analytically, with this method, we will initially set a threshold $\theta$, and then we will subtract all the edges of our graph with an edge weight $> \theta$. After that, each connected component of the graph will correspond to one cluster. Generally, two nodes are in the same connected component if there is a path between them. Suppose that our initial graph is something such as on the left side of figure 39.

![Figure 39](image)

*Figure 39: At the left is the initial graph and a right it will be the clustered graph after filtering it with a threshold of $\theta = 0.5$*

After the filtering with the threshold of $\theta = 0.5$, our new graph would be something like the second graph of figure 39. It is a technique pervasive to solve the problem of predicting links with a reasonably high success rate.
Appendix

This work involves the prediction of links within a (social) network with machine learning techniques, both supervised and unsupervised, using Apache Spark and Scala. Throughout the completion of this diploma thesis, a source code has been created which implements the above techniques. This code of this work is free to anyone who is interested in, for personal use or for proposing corrections and/or improvements. For this purpose, a repository was created at GitHub, which is located at the following link, under the MIT license.

GitHub repository: https://github.com/atheodosiou/LinkPredicionForLargeScaleGraphs
References

[1] Charu C. Aggarwal (auth.) - Recommender Systems, 2 Springer International Publishing
[9] Hially Rodrigues S´a and Ricardo B. C. Prud’encio, Supervised Learning for Link Prediction in Weighted Networks, Center of Informatics, Federal University of Pernambuco, CEP 5-970 - Recife (PE) - Brazil
[10] Huang, Zan, Link Prediction Based on Graph Topology: The Predictive Value of Generalized Clustering Coefficient, 2010
[18] Social Media Mining - Reza Zafarani, Mohammad Ali Abbasi, Huan Liu, Cambridge University, 2014
[19] Recommender Systems: The Textbook, Charu C. AggarwalIBM T.J. Watson Research Center Yorktown Heights, NY, USA
