



## Final year thesis

---

# Multiclass Node Classification in Twitch Social Networks: A supervised task utilizing a custom DNN, the Neo4j Platform and node2vec Feature Learning Algorithmic Framework

*Lirika Sola*

*Supervisor: Mr. Pantelis Kaplanoglou  
Computer Science  
Division of Science and Technology  
American College of Thessaloniki  
Pilea, Greece*



All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owners, except for non-profit educational purposes in which the owners give the leading rights for pieces of work, and only if the owners are properly cited as the original designers of the system and the authors of the present project proposal.

Contact: *erikasholla@hotmail.com*

© 2021, Lirika Sola, Pantelis Kaplanoglou,  
American College of Thessaloniki, Pilea, Greece.

## Abstract

The emergence and vast development of online social networks (OSNs) over the last decade, has led to a major increase in the volume of information brought forth about individual users' demographic values, activities, views, beliefs, and interpersonal relationships, all of which are modelled as labels closely affiliated to the individuals who are represented as nodes of a graph-like structure. In an ideal application that employs these graph-like structures of OSN, users that are part of the network would responsibly select their labels in a manner that is not partial, ambiguous, unfit, and out of date. Real life applications however have proved that this is simply not the case. The goal of this project is to examine “the graph labelling problem” on the Twitch social network communities and build a Deep Learning model that will perform multiclass node classification of the broadcasters' streaming language based on a series of extracted attributes of their neighbor nodes and edges that represent mutual friendship connections. The algorithm used to facilitate this process is the node2vec Random Walk Graph Embedding algorithm.



## Acknowledgements

I would like to give my sincere thanks to my thesis supervisor Mr. Pantelis Kaplanoglou for his constant support, guidance, and encouragement in completing my project.

I would also like to extend my gratitude to all the members of the Division of Science and Technology for their constant support; understanding and valuable advice that helped me grow into the more mature person I am today.

Last, my thanks go to my family and close friends. Thank you for being a constant rock in my life that I always know I can rely on.

.

# 1 Table of Contents

Abstract .....	i
Acknowledgements .....	iii
Acronyms .....	vi
1 Introduction .....	7
1.1 Motivation for the project .....	8
1.1.1 Stating the problem .....	8
1.2 Project objectives .....	9
2 Literature and State of the Art Review .....	10
2.1 Literature Review .....	10
2.1.1 Graph Databases and the Neo4j Platform .....	10
2.1.2 Vertex Embeddings .....	11
2.1.3 Deep Learning and Deep Neural Networks .....	13
2.2 Existing State of the Art Solutions .....	14
3 Design Requirements and Deliverables .....	16
3.1 Project Deliverable's Contribution .....	16
3.2 List of design requirements .....	16
3.3 Development milestones .....	17
3.4 Project Timeline .....	18
3.5 Contingency planning .....	18
4 Software and Hardware Development .....	19
4.1 System Level Design .....	19
4.2 Description of data .....	21
4.2.1 Dataset description .....	21
4.2.2 Dataset adaptation .....	22
4.2.3 Dataset pre-processing .....	22
4.3 Description of Software .....	26
4.3.1 Neo4j Desktop .....	26
4.4 Algorithms .....	31
4.4.1 Custom Deep Neural Network .....	31
4.4.2 node2vec algorithm .....	33
4.4.3 Code Design: Flowcharts .....	36
4.4.4 Source code development .....	37
4.5 Description of hardware .....	41
5 Characterization, Validation and Discussion .....	42
5.1 Characterization Experiments and Results .....	43
5.1.1 DNN Characterization Experiments .....	43

5.1.2	Validation Software .....	45
5.1.3	Validation Hardware .....	45
5.2	Validation Experiments and Results .....	45
5.2.1	DNN Validation Experiments.....	46
5.3	Discussion .....	52
5.3.1	Discussion of Characterization Results .....	52
5.3.2	Discussions of Validation Results .....	52
5.4	Conclusions .....	53
6	A look into the future.....	55
6.1	Advances in the technologies relevant to the project.....	55
6.2	Future improvements (on this project) .....	55
6.3	Insights into future applications .....	57
	Appendices.....	59
	Initially Proposed Design Requirements.....	59
	Initially Proposed Timeline .....	60
	Table of figures .....	61
	Table of equations .....	62
	Table of tables .....	63
	Bibliography .....	64



## Acronyms

OSN	Online Social Network
ML	Machine Learning
DL	Deep Learning
DNN	Deep Neural Network
BFS	Breadth-First Search
DFS	Depth-First Search
CPU	Central Processing Unit
t-SNE	t-Distributed Stochastic Node Embedding
UI	User Interface
CLI	Command Line Interface
WCC	Weakly Connected Components
SNA	Social Network Analysis
ACID	Atomic Consistent Isolated Durable

# 1 Introduction

The emergence and vast development of online social networks (OSNs) over the last decade, has led to a major increase in the volume of information brought forth about individual users' demographic values, activities, views, beliefs, and interpersonal relationships, all of which are modelled as labels closely affiliated to the individuals who are represented as nodes of a graph-like structure. Additionally, it is possible to build a Deep Neural Network (DNN) that predicts people's demographic values such as their spoken language, based on their social media profiles using multiclass node classification.

Twitch is currently considered as the world's leading live streaming platform, enabling millions of users around the world to come together, connect and interact with each other based on their common interests. These common interests along with factors such as their streaming habits, language, location, preferred game genre and more serve as the main contributors to the formation of different communities within the platform. Particularly, Twitch provides its broadcasters with the option of setting their Broadcast Language by applying a representative language label to their channel. This gives to the international viewers the opportunity to find better and more personalized content recommendations of broadcasts that match their native language, while allowing the broadcasters to appear in more directories than just the standard English streaming one.

As the importance of social network studies, node classification and community centrality measures has increased, so have their applications in the different fields of sociological studies, contact and objects recommendation systems, advertising systems, medical diagnosis, and epidemiological studies. The interest in the social network data has also led to an increase of its volume, variety, and velocity, making it challenging to process and extract information from.

## 1.1 Motivation for the project

Technological developments that have taken place over the last years have become a driving force in the evolution of network technologies and online social networks. The variety of models introduced from that moment on, has contributed to the further understanding of multidisciplinary communication systems as well as the general economic and social networks. As a result of these studies, the importance of network structures and the voluminous amount of data they contain is evident. The valuable information this data hold is most depicted as a series of labels associated with individuals who are then represented as nodes of a graph-like structure, all connected to one another through edges that symbolize their interpersonal relationships. The labels come in different forms capturing aspects of an individual's behavior, preferences, beliefs, demographic information and more. This variation makes labels a key factor to a large number of applications, all of which are centered on the different ways social network data is applied to various areas of life. The importance of node labelling however comes with a series of challenges that although not overwhelmingly impossible to be solved, have proved to be prevalent impacting the effectiveness and accuracy of the methods employed in applications that make use of it.

The goal of this project is to explore the domains of Graph Machine Learning and Deep Learning for the potential applications of predicting labels on social graphs. More specifically, this project will examine “the graph labelling problem” [1][2] on the Twitch social network communities, providing a DNN model that is able to perform multiclass node classification of the broadcasters' streaming language based on a series of extracted attributes of their neighbor nodes and edges that represent mutual friendship connections.

### 1.1.1 Stating the problem

In an ideal application that employs graph-like structures of OSN, users that are part of the network would responsibly select their labels in a manner that is not partial, ambiguous, unfit, and out of date. Real life applications however have proved that this is simply not the case. Due to a number of reasons that vary from privacy concerns to purposefully including misleading information for amusement purposes, the absence of labels in specific categories leads to less efficient methods of providing personalized

content recommendations. While it was demonstrated that this problem could be solved through the engagement of several professionals, who managed to maintain the labels and data corresponding to the network's nodes via financial incentives, it goes without saying that this technique does not scale well when confronted with networks that contain up to millions of users. Alternatively, Machine Learning (ML) [3] and node classification algorithms make use of the information found in the moderately labelled graph to predict the classification of the unlabeled nodes [4]. Due to the fact that in social networks, the edges typically depict some form of association between the nodes, the information extracted from the neighboring nodes is also used for a better classification of the unclassified node labels.

Twitch platform's use of broadcasters' languages as a filtering and content recommendation method, facilitates the creation of a model trained to predict a specific attribute of the network's nodes based on datasets with a specific set features, leveraged by the graph structure that the platform's users create through their interactions. This model can later be used in a series of applications that utilize node classification and graph databases to depict analysis of behavior in social networks, perform sociological studies of different communities to find out how they are founded around specific common interests, used for marketing purposes in advertising systems and more.

## 1.2 Project objectives

- Model the datasets into graphs on the Neo4j graph database.  
Modelling the dataset into a graph database through the Neo4J framework and using Cypher-query language to access the nodes and relationships in an efficient and non-time-consuming way.
- Yield node embeddings from the OSN modeled on the graph database using the node2vec algorithm.  
The widely popular graph embedding technique node2vec will be used to

compute the vector representations of the nodes in the graph sampled by a number of random walks.

- Visualize the embedding space for a better understanding of the communities detected in the network based on their streaming languages. The embedding space will be visualized in a scatter plot, with each language speaking community represented through a particular color.
- Build a Deep Neural Network to perform a multiclass classification on the nodes' language property.  
The model will utilize the extracted node embeddings as its' dataset to predict the streaming language of the broadcasters on the Twitch network.
- Perform a series of validation techniques to evaluate the performance of the DNN model.  
Implement K-Fold Cross Validation procedure to evaluate the model's prediction skill on new data. Calculate the confusion matrix and derive from its' performance metrics such as precision, recall and F1-score.

## **2 Literature and State of the Art Review**

### **2.1 Literature Review**

#### **2.1.1 Graph Databases and the Neo4j Platform**

The development of new database technologies in the beginning of the 21<sup>st</sup> century led to the creation of a new type of database, different from the existing relational kind. Distinguished under the name of NoSQL databases, they soon became very popular and well-known for their abilities to scale and scan many records in little time. One type of these NoSQL databases are graph databases [5] which emerged as a need for a better representation of huge volumes of strongly interconnected data. Similarly, to the relational type, these graph structures can easily and intuitively be queried for a better insight on the data.

Based on Graph Theory, graph databases model the data into graph-like structures with nodes representing entities, and connecting edges depicting the relationships between them [6]. Additionally, the node and edge information which is useful for a better understanding of the graph structure is stored into properties. This simplistic representation gives to this category of databases a series of advantages such as a more optimized information search, data storage in the scale of petabytes, larger applicability to real-world cases of data connectivity and its ability to model a larger spectrum of data types. There is a large variety of reference software which implements graph databases, however one of the most used platforms is the popular open-source graph database Neo4j [7].

Known as a native graph database due to its ability to implement the graph model in an efficient manner to its storage level, Neo4j uses pointers to traverse the modelled data. Other features that have increased the popularity and widespread usability of this database platform, include the ability to make use of both vertical and horizontal scalability depending on the utilized version of the platform, the Atomic Consistent Isolated Durable (ACID) [32] guaranteed integrity, cluster support, and minimalistic user interface. Furthermore, Neo4j is popular for its Cypher query language, which although was initially created as an optimized version of the SQL query language for the Neo4j graph database, is now widely employed by other databases like Redis and SAP HANA graph. As a result of its systematic representation of records in the form of nodes and relationships, Neo4j also allows for continual depth and breadth time traversals in large graphs. Its flexible and adaptable property graph schema also allows for the addition of new relationships at later stages, as a way of speeding up the data at times of changes from the business perspective. Lastly, Neo4j is also known for its numerous drivers available for different programming languages, that allow for its implementation on different platforms, software applications and more.

### **2.1.2 Vertex Embeddings**

There are several real-life applications and use cases of graphs, however mathematical, statistical operations and Machine Learning methods are still very hard to be facilitated and implemented. A reasonable solution to these challenging situations is graph embeddings. The graph embedding strategy [8] transforms the graph into a low dimensional space where its information can be saved in a vector or set of vectors. The

transformation conserves the topologies, relationships and attributes of the graph's vertices and edges and can be performed on different levels of granularity. Divided into two main categories, Vertex Embeddings and Graph Embeddings, the first category, which is also a category of focus for the development of this project, is used to encode each node of the graph into its own vector representation. This allows for a great visualization of the embedding space into a two-dimensional plane as well as predictions on the node level based on their similarities. As a result of their practicality and favorable results vertex embedding approaches can be classified into three wide categories: Random Walk, Factorization and Deep Learning based. This project will concentrate on the first category which makes use of random walks to estimate properties such as node similarity [9] and centrality [10] on graphs that can be either partially observable or of a size that does not facilitate measurements easily.

#### 2.1.2.1 The node2vec Algorithm

The node2vec algorithmic framework is one of the first Deep Learning ventures to learn continuous feature representations from graph structured data [14]. Published as a modification of the DeepWalk [11] model, node2vec uses random walks to perform node embedding. For the graph traversal to take place, each node is represented in a vector form. Consequently, the arrangement of these vector representations within a matrix determines the graph's traversal sequences. Similar to DeepWalk, the node2vec algorithm's approach to complete this series of random walks uses the:

$$Pr(v_i | (\varphi(v_1), \varphi(v_2), \dots, \varphi(v_{i-1})))$$

Equation 1 node2vec's approach to random walks

equation, whose objective is to calculate the probability of distinguishing nodes given all the prior visited nodes of the walk's sequence [12].

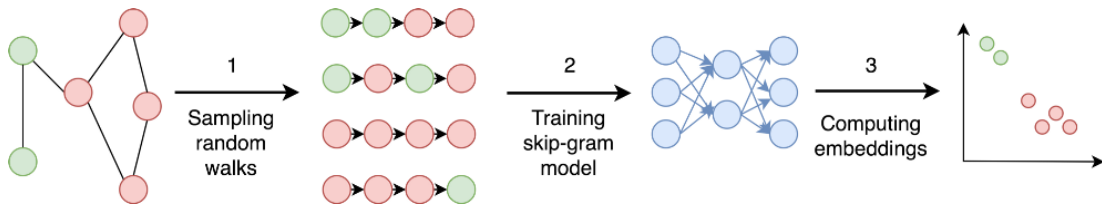


Figure 1 Phases of Random Walk approach

The latent representations of the graph's nodes mapped by the function are later used as inputs for the DNN model [13]. The later, based on the inputted nodes and the number of times they were encountered throughout the walk, will be able to predict an unlabeled feature of theirs or perform a classification.

Moreover, the additional modifications that led to node2vec becoming an improvement of the DeepWalk algorithmic framework, are connected to its intention of capturing various connectivity patterns of networks to a finer level of detail [15]. By defining a more flexible approach to a node's network neighborhood, the design of a biased random walk approach used to explore various neighborhoods is achieved. This approach enables the algorithm to learn representations that will be used to classify nodes according to their roles and/or the communities [16] they are part of. The flexible walk bias  $\alpha$ , parameterized by the  $p$  and  $q$  parameters, give to the algorithm the ability to perform both BFS and DFS procedures, the executions of which are controlled by the probabilities  $1/p$  or  $1/q$ .

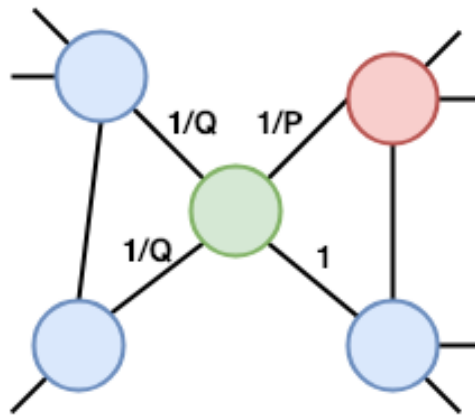


Figure 2 Random probabilities of node2vec's random walk step

Based on these parameters, node2vec can return results about local neighbors and/or global variables found in the network.

### 2.1.3 Deep Learning and Deep Neural Networks

Machine Learning (ML) technologies have majorly influenced the development of applied sciences and the services they offer to the wide public. Services such as e-



commerce websites, product recommendations, content filtering, web searches and more are becoming increasingly more prominent in consumer products such as smartphones and high-quality cameras. However, for the quality and accuracy of these products and services to be of a high level, prior conventional Machine Learning technologies had very limited data processing abilities and thus, a new category of techniques under the name of Deep Learning (DL) was used.

The main reason as to why the standard Machine Learning techniques were not considered conventional was connected to their necessity of being carefully and competently engineered as well as consistently tuned in order to be able to depict patterns and extract features out of the raw data that was retrieved as its input. Representation learning methods however, allowed the machine to which the raw data was being fed, to automatically depict the representational patterns that were later transformed into feature vectors utilized by classifier subsystems to classify the patterns detected in their input. As part of the representation learning category, Deep Learning methods can detect patterns through an automatic general-purpose learning method. The key difference from other representation learning methods is the high number of representation layers that are supported, acquired through the composition of non-linear modules, each with the ability to transform the input received in the form of raw data into continuously more abstract levels of representation patterns. This approach allows for complex functions in the likes of classification tasks, where the high number of layers is important for the amplification of key aspects of input data, to be executed easier. Consequently, many major advances in improving the already existing state of the art models were made. This was notable especially in the domains of object detection, visual object and speech recognition and other governmental, scientific, and business fields.

As the amount of available computation and the importance of big data and big data analysis has been increasing at an incredibly fast pace, the public interest in Deep Learning methodologies is higher than ever. With new algorithms and neural network architectures being constantly developed, the future of Deep Learning is looking more promising than ever. [17]

## **2.2 Existing State of the Art Solutions**

The interest for social network analysis has increased with the rise of large social websites such as LinkedIn, Twitter, and Facebook. The large amount of

accumulated data brought new challenges related to the task of node classification. There are several state-of-the-art solutions whose main purpose is predicting the missing properties, one of which is the case of Facebook [20][21]. This study concentrates on the investigation of different factors affecting the tendency of students to use Facebook for educational purposes. Based on existing factors such as student ID, gender, GPA, age, Facebook friends, active hours spent online, descriptive data and multi-linear regression analysis are used to predict people's education path. Once this classification task is completed, specialized recommendations are provided as incentives to encourage them towards that specific educational journey.

Regarding Deep Learning and DNNs, there are many existing state-of the art methods that are constantly being developed and improved in solving the existing problem of finding intricate patterns in high dimensional data. Other than improving the existing techniques of image and speech classification, DNNs are greatly assisting the process of natural language understanding and its subfields of sentiment analysis, topic classification and language translation. Additionally, DNNs are looking promising in aiding in the reconstruction of brain circuits, prediction of the interactions between different prospective drug molecules, classification of different health risks and diseases and so on. The last topic is of particular interest, with a large number of conducted studies dedicated to it, one of which is the multiclass classification of different types of chronic diseases. More specifically, the records of 110,300 patients were examined in an anonymous way to predict diseases such as hypertension, diabetes, and different combinations of theirs [18]. After splitting the dataset into the training and testing sets with a ratio of 90% to 10%, and transforming them into suitable classification datasets, different Machine Learning algorithms were combined to form Deep Learning architectures that would correctly examine the data. Some approaches were the popular support vector machine (SVM) [24], multi-layer perceptron (MLP) [25] and Random Forest algorithms. Additionally, similar to this final year project, a K-Fold Cross Validation technique was used to assess the prediction accuracy along with other metrics such as recall, precision and F1-score.

## 3 Design Requirements and Deliverables

In this chapter the design requirements of the DNN model are explained.

### 3.1 Project Deliverable's Contribution

Seeing that this project leans more towards the research spectrum rather than being a fully functional, user-friendly software application, it targets people that have adequate knowledge in the domain of data science, social network analysis (SNA) and ML. The field of Deep Representation Learning for SNA [19] is still new and in the process of attracting more attention, funding, and research. While it has many applications one of which is the classification of the semantic labels of nodes in social networks, the existing methods and approaches used are still limited and open to new solutions. The said current solutions are mostly focused on using extraction-based techniques to retrieve information about the network's node features based on their surrounding neighbors. This information is then applied to existing ML classifiers in the likes of Support Vector Machines, logistic regression, or naïve Bayes [26].

The DNN model developed in this project was intended as a potential contribution among the aforementioned solutions.

### 3.2 List of design requirements

The list of the design requirements is as following:

1. Filter the original dataset by creating subsets in cases when only specific streamers are of interest.
2. Preprocess properties of the dataset that have an inconsistent range of values for more data consistency, attribute completeness and less noise.
  - 2.1. Translate the categorical target variables to numerical.
3. Build the graph dataset in the Neo4j desktop platform both directly through the queries provided or indirectly by running the appropriate python script.
4. Produce the node embeddings by performing the node2vec algorithm.
  - 4.1. Extract embeddings and their respective language targets into a csv file.
  - 4.2. Visualize the node embeddings into a two-dimensional embedding space through the t-Distributed Stochastic Node Embedding (t-SNE) [22] tool.

5. Balance the dataset with simultaneous increase and reduction of samples per category for a non-biased classification.
6. Build a DNN model to perform multiclass classification on the dataset and display the categorized streamer labels.
7. Produce statistically descriptive data of the training, evaluation, and prediction process.

### 3.3 Development milestones

The first development milestone of the project was the modification and preprocessing of the already extracted anonymous raw dataset by the Stanford Network Analysis Project (SNAP), into csv files that are of a smaller size and thus more easily supported by the hardware the project was running on. The second development was the upload of the modified csv files into the Neo4j Desktop application followed by the initialization and configuration of a local database, as well as the modeling of the data into graphs through the Cypher-query language. Thirdly, after having installed the APOC [23] and Graph Data Science [27] libraries, the node embedding algorithm node2vec was executed on the existing graph data. Continuing to the fourth development milestone, after having run the respective query to retrieve the generated node embeddings along with the node id and the language label targets, the data was saved into a csv file that would later serve as the input dataset of the created DNN model. For a visualization of the embedding space and the created clusters, a small program was developed utilizing the t-SNE visualization tool and Matplotlib library. This program maps the dataset into a two-dimensional plot, where each of the streaming language categories were denoted with a different color. The fifth development milestone consisted of the creation of the Deep Neural Network model. For a better performance, during this development step the extracted embeddings dataset required for standardization as well as balancing of the highly unbalanced number of records each language category had. Lastly, for the evaluation of the prediction accuracy the K-Fold Cross Validation [28] technique was implemented on the existing model, along with the implementation of different accuracy metrics such as the precision, recall, F1-score all derived from the calculation of the confusion matrix. Moreover, different experiments with the parameter settings, dataset splitting, validation approaches and over sampling/under sampling techniques were made.

### 3.4 Project Timeline

Due to the fact that the project’s completion deadline is May 31<sup>st</sup>, 2021, the period available for the development, validation and thesis writing is approximately 6 months. Most of this time, more specifically 90 days, was allocated for the implementation of the graph databases, graph embedding algorithms, and DNN model. Similarly, the validation process also took place for a period of 85 days in a parallel manner with the project’s development. This, due to the nature of the project, which requires a highly functional performance of each developed aspect before being able to proceed to the following one. Lastly, the thesis document writing, poster and PowerPoint presentation were allocated a time limit of 40 days to be completed.

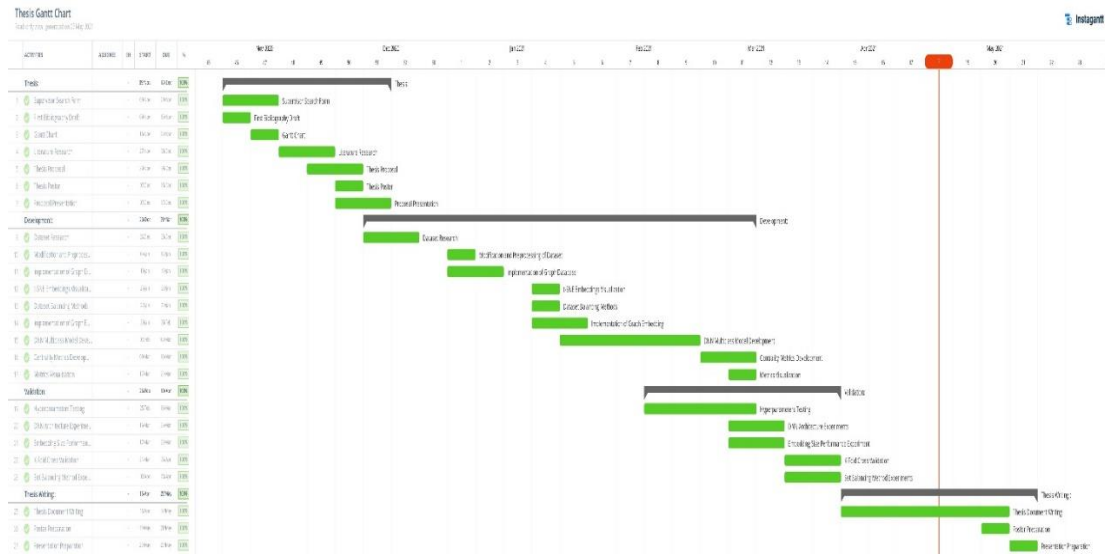


Figure 3 Project Timeline

### 3.5 Contingency planning

After thoroughly looking at the initial dataset titled Twitch Social Networks [29] selected for the purpose of this project, it resulted that the explanations provided by its’ authors regarding the semantics of its’ vertex features were not adequately descriptive to allow conducting our research. This resulted in having to select the current Twitch Gamers dataset [30], newly published by the same authors but with a better description of its’ vertex semantics. Although the social network analysis and research was still to be conducted on the same social media platform with the main interest being the streaming language of the users, the scope of the initially proposed project had to be adapted.

Instead of using existing ML models to perform binary classification of the node embeddings extracted by the node2vec and DeepWalk algorithms, a new DL model was created to perform multiclass node classification of the streaming language categories. The newly set objectives were all met, however as always, there is room for further improvement and development, all of which are elaborated in detail in section 6.2.

## 4 Software and Hardware Development

The following chapter concentrates on describing the used dataset along with the algorithms used and source code development.

### 4.1 System Level Design

The project combines three small supplementary programs and one DNN model, that when executed individually step by step create a ML pipeline.

To begin, the first auxiliary program was created solely for the purpose of modifying the existing Twitch network dataset's size and preprocessing its raw data. Because of the large number of edge records in the original dataset and lack of powerful hardware to support its translation into a graph via the Neo4j database, it was important to be able to select a subset of nodes as well as be able to filter their corresponding relationships to later model the subset into a graph structure more easily. Additionally, due to the target column containing categorical variables, it was necessary for the variables to be converted into numbers before being used as input of the Deep Neural Network model. Similarly, other properties of the dataset were pre-processed for any inconsistencies to be resolved and noise to be smoothed prior to appointing them as node properties. This program's input are the original dataset csv files while the outputs are two new csv files containing the modified node records and the filtered node edges. These files will serve as input for the second auxiliary program, responsible for running the Cypher queries needed to model the records into a Neo4j graph. Moreover, it should be noted that the program does not have a User Interface (UI) but rather a simple Command Line Interface (CLI) that simply notifies that the files are generated, and the data has been pre-processed.

The second auxiliary program serves as a substitution for manually writing and running the Cypher queries used to build the graph on the Neo4j desktop application. While it still requires for an active Neo4j local database to connect with through the Python driver, for the APOC and GDS libraries to be installed, and the modified csv files from the first program to be placed on the database's 'Import' file, the queries can also be automatically run from the Python script in cases when we are not interested in the output produced by the Cypher queries. Similar to the first one, this program does not accept any input and produces a notification of process completeness as its output in a simple CLI. Additionally, after its' execution, a csv file containing the node embeddings created by the node2vec algorithm, and node language labels will be extracted and allocated in the same 'Import' folder.

The third program was developed for the visualization of the graph nodes' embedding space created from the execution of the node2vec algorithm. This program as well requires a connection with the active local database created in the Neo4j desktop, through a Python driver object. After the connection is made, the language categories and the node embeddings of the graph are retrieved via the execution of a Cypher query. These embeddings are then visualized into a two-dimensional plot through the unsupervised, non-linear t-SNE tool. The embeddings are all categorized according to the nodes' streaming language categories and can be easily distinguished by the color assigned to each category. Moreover, this program does not have a UI as well as it does not require any input.

The fourth and main program is the DNN model written in Python, was developed to train, test, evaluate, and predict the language category of the Twitch streamers. Its' input file is the csv file extracted from the Neo4j database after the creation of the graph and execution of its' node embedding algorithm. More specifically, this dataset contains the embeddings and language targets for each data entry. However, before the execution of the program, the dataset's classes are balanced for the training of a more accurate and unbiased classification model. This program does not have a UI, but it does have a simple command line interface that displays the training, testing, evaluation process as well as accuracy metrics in the likes of precision, recall, F1-score, and confusion metrics. In addition, the program outputs two graphs, with the first one visualizing the training and testing losses and the second their respective accuracies. The confusion matrix is also

visualized for a better understanding of the model's prediction accuracy. Lastly, the model weight data is saved into a .h5 Keras file format, from where it can be loaded and accessed at any time.

## 4.2 Description of data

### 4.2.1 Dataset description

The Twitch Gamers dataset used in the studies of this project was crawled by its authors in April of 2018 using snowball sampling. Its main characteristics consist of having no missing attributes, symmetric mutual edges as well as being part of the same large, connected component. As a result of this three-step cleaning process, the undirected, single component OSN consists of a total of 168,114 nodes and 6,767,557 edges. The explanation of the node attributes is summarized:

- ID: anonymous identifier for the streamers, serving as the index of each data entry.
- Creation date: the date when the user account was first created, part of the date type variables.
- Last update date: the date when the user had last streamed, date type variable.
- Lifetime of the account: the number of days passed between the first and the last stream, a count variable.
- Number of views: the total number of stream views per channel, a count datatype.
- Dead account: whether the account is inactive or not, a categorical variable.
- Mature content: whether the account is for mature audiences or not, a categorical variable.
- Affiliate status: whether the streamer has gained their affiliate status from which they can monetize their content or not, categorical data.
- Streaming language: the language the streamers use for their broadcasting, a categorical variable.

The categorical attribute of interest for this project was that of broadcasting language which consisted of a total of twenty-one categories, suitable for building a multiclass classification model.



### 4.2.2 Dataset adaptation

The modifications of the Twitch Gamers dataset can be categorized into two stages: prior to the development of the DNN model and after. This subsection will focus on the first stage, while the subsection 4.2.3 will describe the techniques used for the normalization/standardization of the data as well as the balancing process of the dataset. As it was mentioned in the description of the first auxiliary program on subchapter 4.1, for the purpose of this project only a subset of the original dataset was used. More specifically, around 30,001 nodes and their 204,641 corresponding edges were selected out of the original records. Before being modelled into a graph, the language property of the nodes which was in the form of a categorical feature, was encoded into an integer feature form, taking values from 0 to 20.

After the creation of the graph through the Cypher queries, a series of extra steps needed to take place to ensure that there were no orphan nodes left on the graph, and that the entire set of nodes to be inspected by the DNN was one large set of connected components. The first issue was solved through the execution of a Cypher query that checks each node for any incoming and outgoing relationships, deleting them when the retrieved number is zero. Continuing, the second issue was resolved through the execution of the Weakly Connected Components (WCC) [31] algorithm used to determine the number of independent and disconnected clusters within the same set. After retrieving a total count of the number of clusters, their identification id, and the number of nodes belonging to each of the components, a joint query was used to iterate and delete all the clusters except the one with the maximum number of components. The result of this query, the largest graph of connected components would then serve as an input for the node2vec embedding algorithm with a return and in and out factor of 1.0, embedding dimension of 15, a walk length of 80 and a walk per node equal to 8, all taking place during one iteration. The embedding dimensions would then be denoted under the property name of "embeddingNode2Vec", which would be later extracted along with their respective language target into a new csv file that would serve as the dataset for the DNN, training, validation, and evaluation process.

### 4.2.3 Dataset pre-processing

#### 4.2.3.1 Dataset Balancing

The new dataset extracted from the graph database consists of 26,787 records of Twitch streamers in the form of 15-dimensional embeddings. Even so, after plotting the data to visualize the number of records per language categories, it was clearly noticeable that the dataset was highly imbalanced, with the dominant streaming language being English. This imbalance is portrayed in the following image graph:

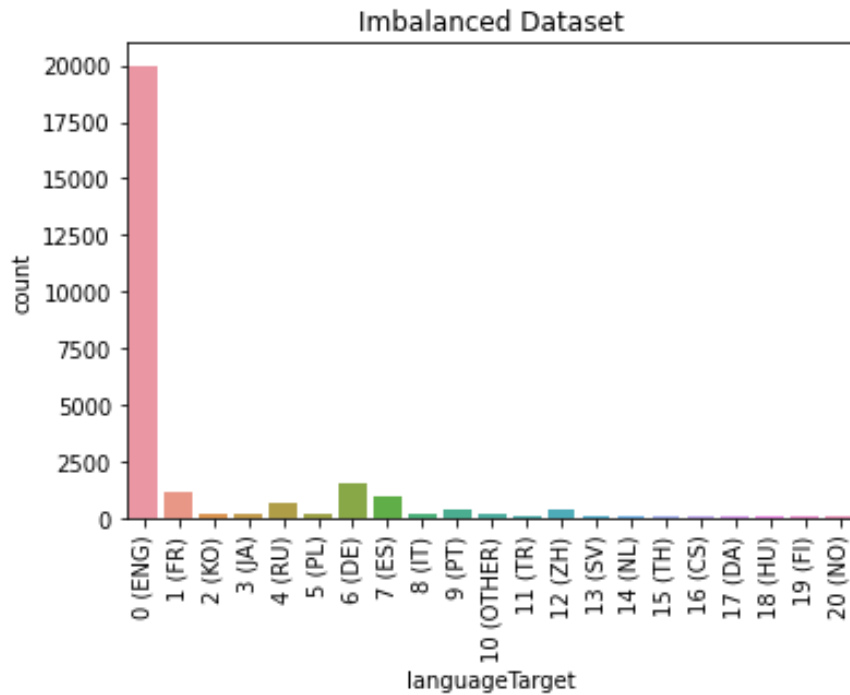


Figure 4 Unbalanced Dataset

As denoted by the graph as well, the majority class is that of English-speaking streamers, followed by the classes of German, Spanish and French speaking streamers. On the other hand, there is a considerable number of minority classes, the most predominant one being the class consisting of Norwegian streamers.

Although this highly imbalanced dataset is a representation of the real-life predominant streaming languages in the Twitch platform, this bias should not be mirrored in the prediction and classification abilities of the DNN model. Due to the model's default tendency to centre on learning the characteristics of the majority classes whose number of records is more abundant, the process of tuning the model's skills to correctly predict the minority classes' labels is of special interest.

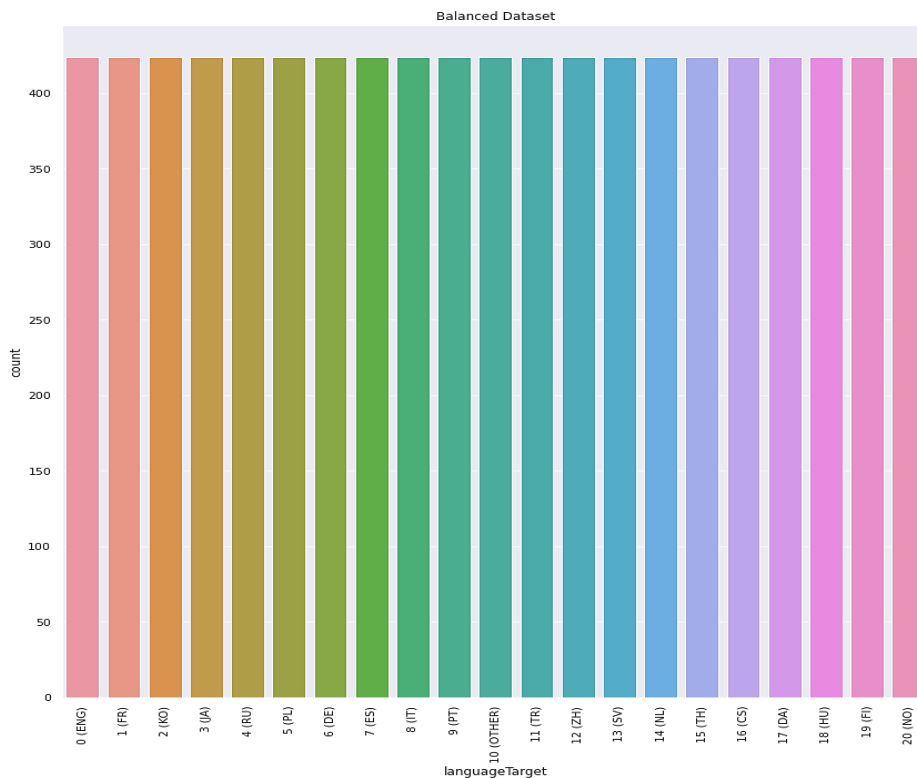
The class imbalance problem has been an open problem for decades, having to be addressed for each dataset separately. However, there have been several suggested

techniques used to make the balancing process of the dataset more approachable. For this dataset, a number of balancing techniques were tried such as synthetic oversampling with SMOTE [33] and ADASYN [34] algorithms as well as random oversampling and under sampling techniques from the Imbalanced-learn open-source Python library.

It was proven that the best approach to balance this specific dataset was a resampling technique that consisted of a combination of both increasing and reducing the number of existing records among different classes.

More precisely, the majority classes are resampled through the removal of a considerable number of their observations, while the minority classes are resampled by adding multiple copies of their observations, until all the classes are balanced out. There are several advantages and disadvantages to both of these techniques that affect the classification capability of the model, however this topic will be more elaborate in chapter 5.

The following plot visualizes the dataset after the resampling process has been completed. In this specific case, each class has been resampled to 424 observations which is equal to the number of records that belong to the 12th category – that of Chinese streamers.



**Figure 5** Balanced Dataset

#### 4.2.3.2 Feature Normalization

Machine Learning algorithms aim to find patterns in the data through the comparison of the data points' features. For each one of these features to be treated with the same importance, it is important for the data points to be of the same scale. For this project, one of the popular methods used, was the Min-Max normalization [35]. This method transforms the feature's minimum value into zero, maximum value to one, and any other value into a decimal between the scale of zero and one. The formula of this approach can be summarised as:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Equation 2 Min-Max normalization

Although this approach guarantees that all the datapoints' features are of the same scale, its major downside is that it is not optimised to deal with outliers. For this reason, the approach of feature standardization will also be used on this dataset. The impact of both these approaches on the model's validation accuracy will be discussed more thoroughly in chapter 5.

#### 4.2.3.3 Feature Standardization

Standardization, also known as Z-score normalization rescales the features so that their properties resemble a standard normal distribution: centred around zero, with a standard deviation from the mean equal to one. The formula to calculate the standard scores also known as z-scores is:

$$Z = \frac{x - \mu}{\sigma}$$

Equation 3 Feature Standardization

where  $\mu$  represents the mean and  $\sigma$  the standard deviation [35]. If a value is below the mean of all the feature values, the result of rescaling will be a negative number while if it is above the mean, it will be a positive number. In cases when the value is equal to the mean, it will be normalized to zero.

However, it should be taken into consideration that although this approach is good at handling outliers, the normalized data being produced is not of the exact same scale as it would be with the Min-Max normalization approach.

## 4.3 Description of Software

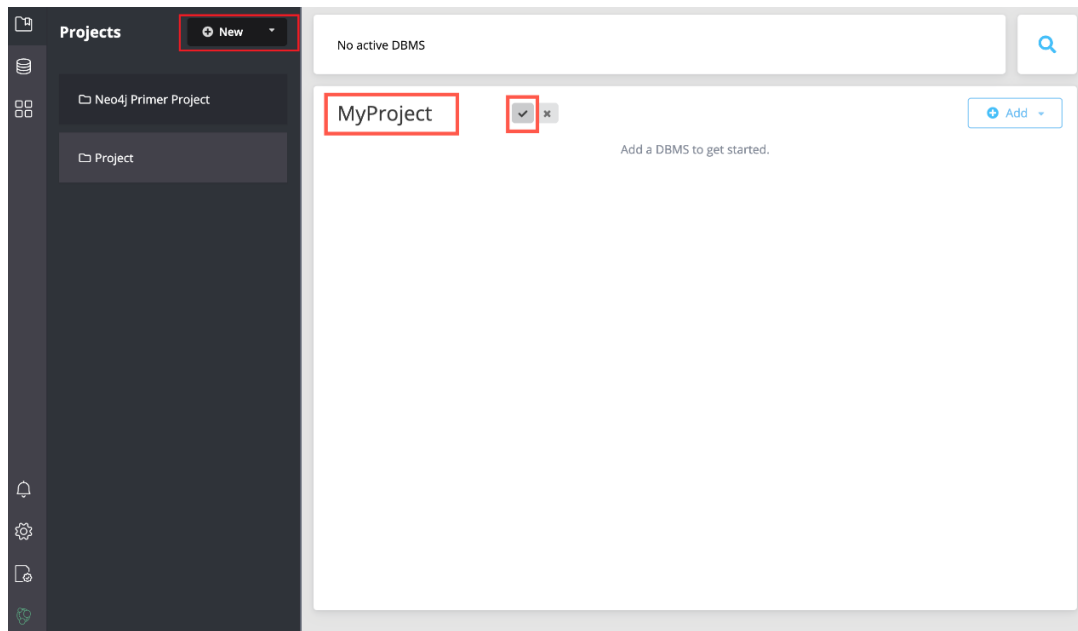
This section provides an elaborate explanation of the software used during the development of the project.

### 4.3.1 Neo4j Desktop

Used as Developer Integrated Development Environment (IDE) for Neo4j instances, the Neo4j Desktop allows its users to manage different projects as well as local and remote Neo4j database servers. Each one of these servers can be managed, upgraded, and configured through the application's UI, with no prior knowledge of command line commands needed. Additionally, this IDE allows for the installation of different Neo4j plugins such as the Graph Data Science (GDS) and APOC libraries that were utilized in this project through just a single click.

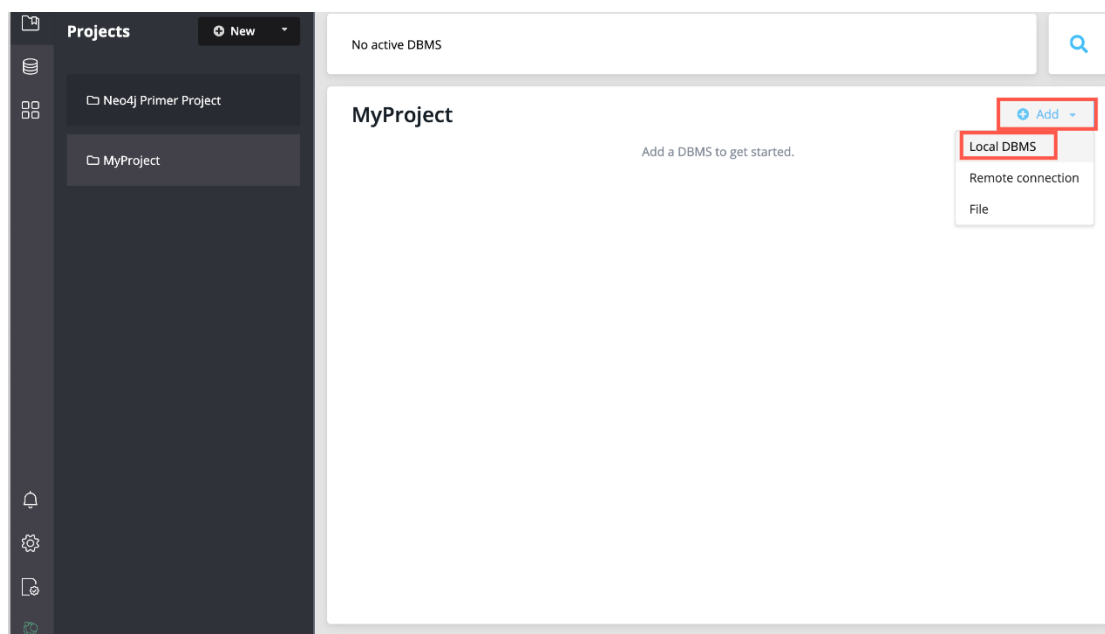
First time users, after the activation of the software, will be greeted by a number of guides that will introduce them to a sample database that has been authenticated under a randomly generated password. Opening this sample database leads to the Neo4j Browser UI, in which the users can run their Cypher queries as well as retrieve their results and visualize them in a graph form if needed. After being introduced to the basics of the Neo4j Desktop and Browser, the users can create their separate projects which can contain one or more database servers.

For a new project to be added, the 'New' button on top of the sidebar should be clicked. The newly created project will be given a default name that can be edited in accordance with the users' preferences by selecting the 'Edit' button as seen on the image below.



**Figure 6 Creating a new project.**

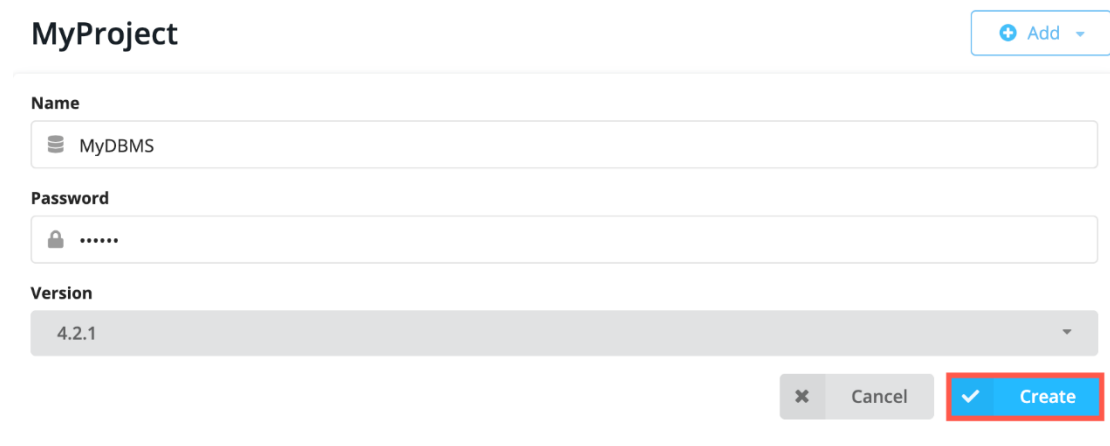
This project can contain multiple database servers, established either as local databases or remote connections to external projects. For the purpose of this project, a local database server was created following the steps provided in the screenshots below:



**Figure 7 Adding a local database.**

The added server will open a new dialog box in which the credentials of the database should be entered. While changing the default database name is optional, it is mandatory for a password to be provided. Additionally, due to new releases being deployed

frequently, the users can select and download their own database version, which is generally set by default as the version of the installed Neo4j Desktop.



The screenshot shows a form titled 'MyProject' with a '+ Add' button in the top right. The form has three main sections: 'Name' with a text input containing 'MyDBMS', 'Password' with a masked input field, and 'Version' with a dropdown menu showing '4.2.1'. At the bottom right, there are three buttons: a close button (X), a 'Cancel' button, and a 'Create' button which is highlighted with a red border.

Figure 8 Database initialization

After having successfully set-up the database server, a series of actions can be performed such as installing different plugins (libraries), change the default configuration settings to enable or disable different functionalities, import, and export csv files and more.

To install the APOC and GDS plugins and enable the functionalities of this project, the 'Manage' pane which contains a number of tabs, one of them being the 'Plugins' should be accessed as follows:

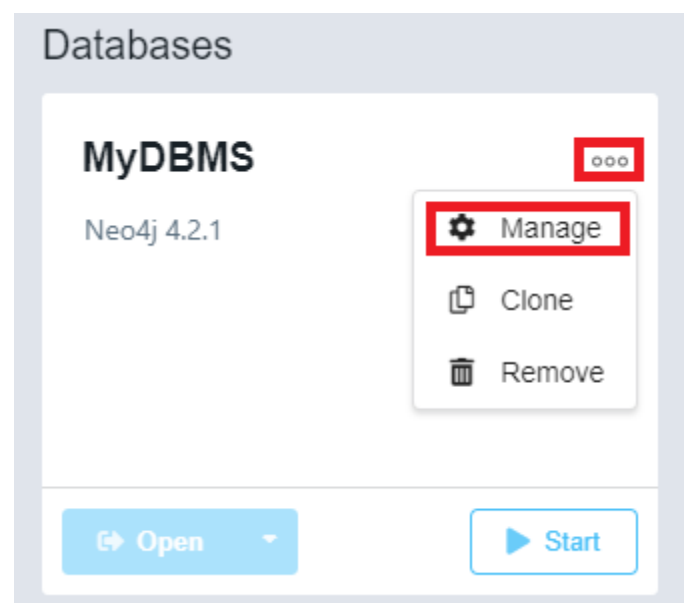


Figure 9 Accessing the managing options.

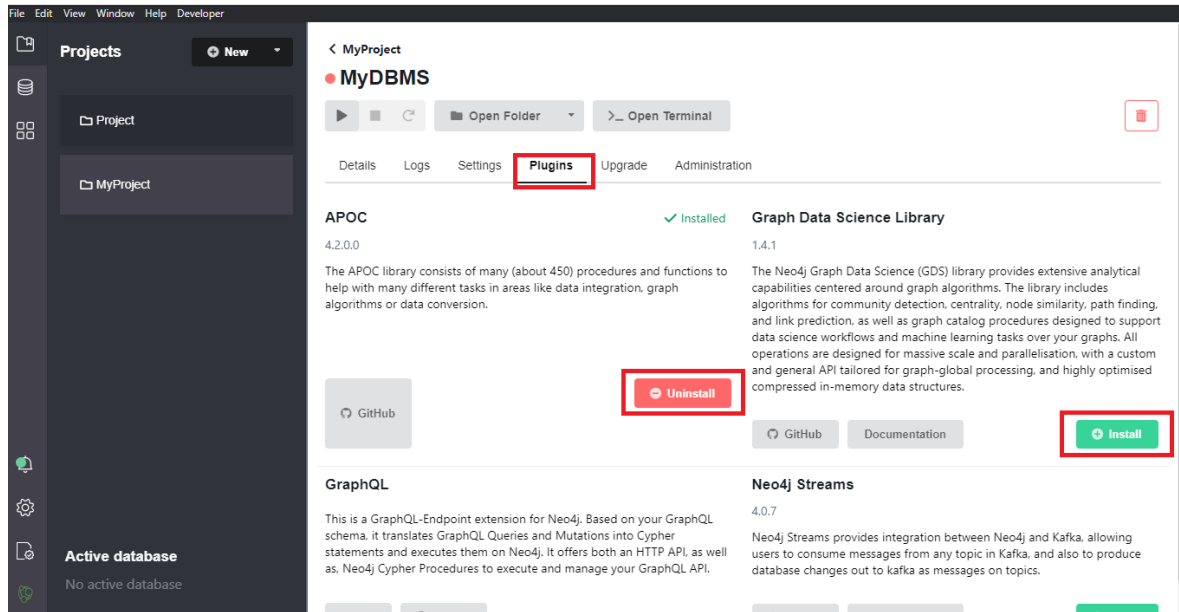


Figure 10 Plugin installation

Similarly, in order for the second auxiliary program to work smoothly with the csv file extraction functionality, changes to the database's default configuration file need to be made. This can be done by accessing the Settings tab, and adding the following changes and confirming the action by clicking the 'Apply' button:

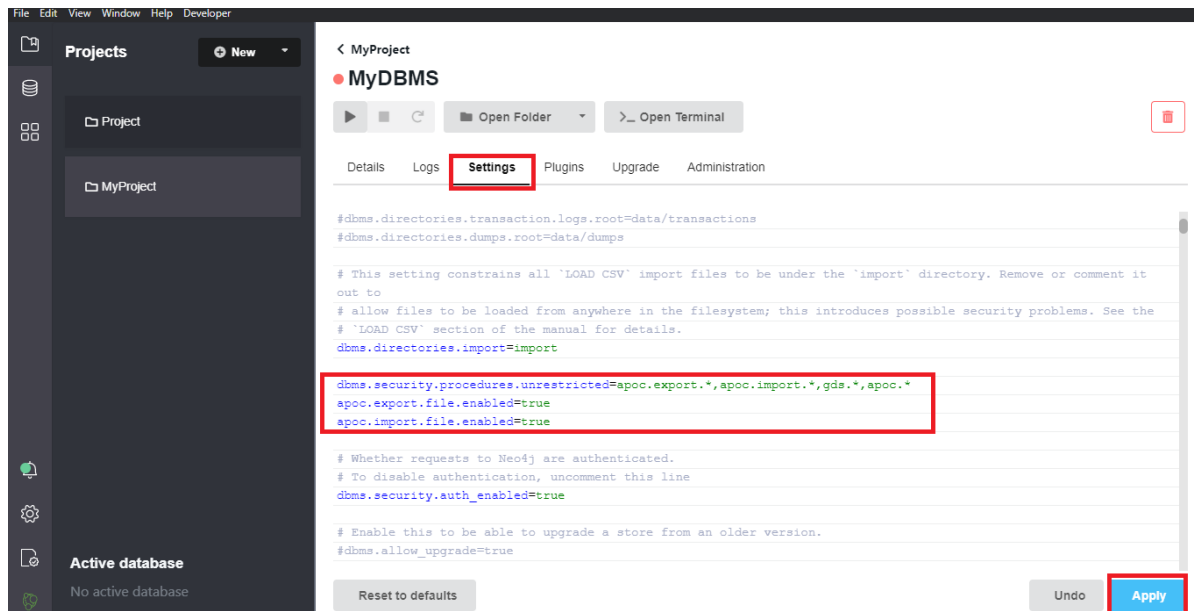


Figure 11 Configuration settings

In the same manner, other changes can be made in this tab to facilitate different functionalities. For example, different users working with big amounts of data or trying



to map large files into a graph may need to change their default Java heap size and amount of allocated memory to facilitate the size of the files they are working on. Since sizes by default are calculated by the available system resources, this configuration step varies from user to user.

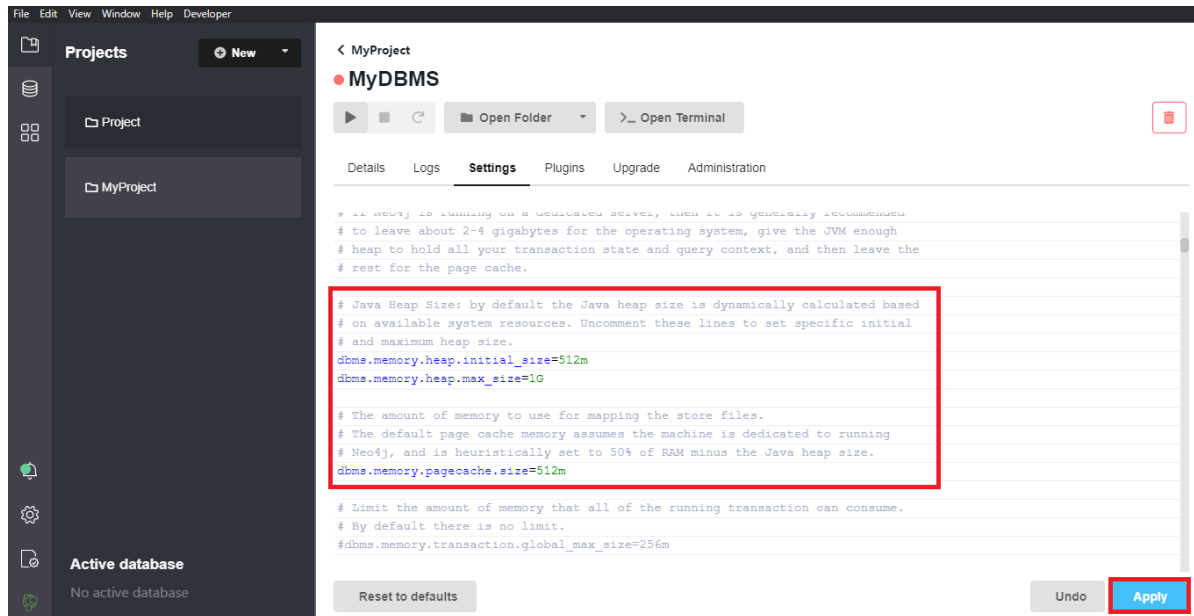
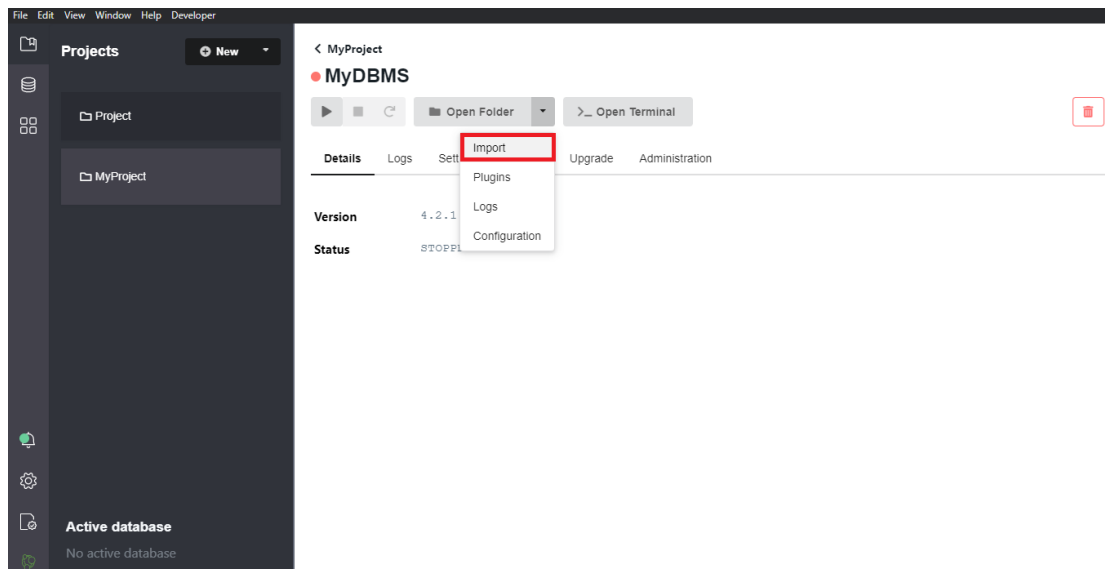


Figure 12 Configuration settings (2)

Lastly, to import the needed csv files that contain the node and edge data necessary for the creation of the graphs, the 'Import' file should be accessed following the steps as shown in the images below:



## 4.4 Algorithms

This section details the algorithms created and used in this project.

### 4.4.1 Custom Deep Neural Network

The developed and evaluated Deep Neural Network model was created with the purpose of providing a new solution to the node embeddings multiclass classification challenges. For the implementation of this model's architecture, the Tensorflow and Keras frameworks were employed to enable the stacking of the consecutive layers of the neural network. The types of layers that this DNN model is composed of are as following:

- Dense layer which is densely connected with the previous layer of the neural network. Specifically, each neuron receives input from all the neurons of the previous layer. The matrix-vector multiplication performed as a background operation in this layer makes use of actual parameters that not only can be trained but also updated through the backpropagation algorithm. As an output, this layer generates an m-dimensional vector.
- Batch Normalization layer where an additional layer is introduced to carry out a number of operations on the inputs received from the previous layer such as standardization, normalization and later scaling and shifting. This makes possible the avoidance of the effects that unstable gradients can have on the NN.

Another operation used as well within the DNN is the activation function that transforms the neuron signals to normalized output. The purpose of this function is to introduce non-linearity on the neural network, giving it bigger representational power. These aforementioned layers are then encapsulated into a number of sequential “modules”. More specifically, this DNN model contains six modules with the input layer being the first one and the output layer being the sixth. The following image represents the actual architecture of the model:

Model: "c_fully_connected_dnn"		
Layer (type)	Output Shape	Param #
=====		
dense (Dense)	multiple	1920
activation (Activation)	multiple	0
batch_normalization (BatchNo	multiple	512
dense_1 (Dense)	multiple	32768
activation_1 (Activation)	multiple	0
batch_normalization_1 (Batch	multiple	1024
dense_2 (Dense)	multiple	76800
activation_2 (Activation)	multiple	0
batch_normalization_2 (Batch	multiple	1200
dense_3 (Dense)	multiple	76800
activation_3 (Activation)	multiple	0
batch_normalization_3 (Batch	multiple	1024
dense_4 (Dense)	multiple	32768
activation_4 (Activation)	multiple	0
batch_normalization_4 (Batch	multiple	512
dense_5 (Dense)	multiple	2688
activation_5 (Activation)	multiple	0
=====		
Total params: 228,016		
Trainable params: 225,880		
Non-trainable params: 2,136		

Figure 14 DNN architecture

As it can be noticed, each module has a dense, activation and batch normalization layer. The activation function that is utilized is the Tanh Hyperbolic Tangent Function. By taking real values as input, this function outputs values that can range from -1 to 1. The larger and thus, more positive the input, the closer will the function's output be to 1 and vise-versa. The calculations are done in accordance with the following the equation:

$$\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

Equation 4 tanh activation function

Regarding the hyperparameters of the adapted baseline model the number of epochs is set to 50, the batch size to 55 and the learning rate to 0.0001. Additionally, a callback object is used to monitor the validation loss, stopping the training and validation process, once

the said loss stops decreasing. Although this directly affects the number of epochs used for the training, it serves as a great practice for ensuring that no underfitting or overfitting scenarios take place.

Moreover, due to the problem being a multi-class classification problem, the loss function utilized is the Categorical Hinge, with an Adam optimizer and Categorical Accuracy metric to calculate the percentage of the correctly predicted values for each one of the one-hot encoded ground truth labels. By penalizing both incorrect predictions and correct but unconfident predictions to different degrees, the categorical hinge loss tries to maximize the margin between the data points and the boundary threshold. This is summarized in the following equation:

$$l(y) = \sum_{y \neq t} \max(0.1 + w_y x + w_t x)$$

Equation 5 Categorical Hinge loss function

Differently from the usual Categorical Cross Entropy loss function used in multiclass classification problems that applies on probability, the Categorical Hinge applies on distance which does not have to be in the range 0 to 1. Therefore, the activation function used in the last layer is the Linear activation function, rather than the usually used Softmax.

#### 4.4.2 node2vec algorithm

By applying the Skip-gram architecture to the domain of networks, node2vec aims to optimize the function responsible for maximizing the log-probability of surveying a  $N_s(u)$  network neighborhood with  $u$  being a node of the given  $G = (V, E)$  network, given its feature representation based on the node-to-feature representations mapping function  $f: V \rightarrow \mathbb{R}^d$ . This is summarized in the following equation:

$$\max_f \sum_{u \in V} \log Pr(N_s(u) | f(u))$$

Equation 6 Calculation of maximum log-probability

By assuming symmetry in the feature space - meaning that the effect that a node and its neighbor have over each other is symmetric; and conditional independence - with the

likelihood of node observation in a network being independent of any observation of its neighborhood nodes, the above equation is optimized given the following structure:

$$\max_f \sum_{u \in V} [ -\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) ]$$

**Equation 7 Optimization of the maximum log-probability function**

Given that the notion of feature representation through the Skip-gram architecture was initially designed for different applications of the natural language processing domains, the notion of a neighborhood can be quite linear. To adapt this issue to the domain of networks, where the notion of a neighborhood is richer, a randomized sampling strategy  $S$  is proposed.

Regarding this sampling strategy, real life social networks more commonly than not, exhibit a mixture of both homophily and structural equivalence tendencies. However, the existing works such as feature hand-engineering and feature-learning through optimization problems, fail to facilitate both principles. Created as an improvement of already existing works, by obeying these two principles the node2vec algorithm provides the flexibility that is needed when it comes to learning representations of nodes in both scenarios: when the nodes are embedded closely to the rest of the network community they belong to also known as homophily, as well as when the nodes have similar embeddings as a result of playing the same important role within the network structure, known under the term of structural equivalence. This allows for a better generalization of the feature learning process across a wide diversity of prediction problems.

Building upon these observations, node2vec presents the concept of biased random walks, that allow for the exploration of the graph both as a combination of homophily and structural equivalence. The search bias is expressed as:

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

**Equation 8 Summary of search bias in random walks**

where  $\alpha_{pq}(t, x)$  is the transition probability,  $d_{tx}$  the shorted distance between nodes  $t$  and  $x$  and  $p$  and  $q$  the parameters that control the probability of re-visiting a node during the random walk, and the ability to detect which nodes are “inward” and which are “outward”.

Having said that, the node2vec algorithm used in this project from the Neo4j's GDS library has the following hyperparameter configuration:

- initial learning rate: 0.025,
- walks per node: 8,
- iterations: 1,
- return factor  $p$ : 1.0,
- walk length: 80,
- window size: 10,
- write property: embeddingNode2Vec,
- in and out factor: 1.0,
- embedding dimension: 15

The rest of the unspecified parameters are automatically set to their default value.

### 4.4.3 Code Design: Flowcharts

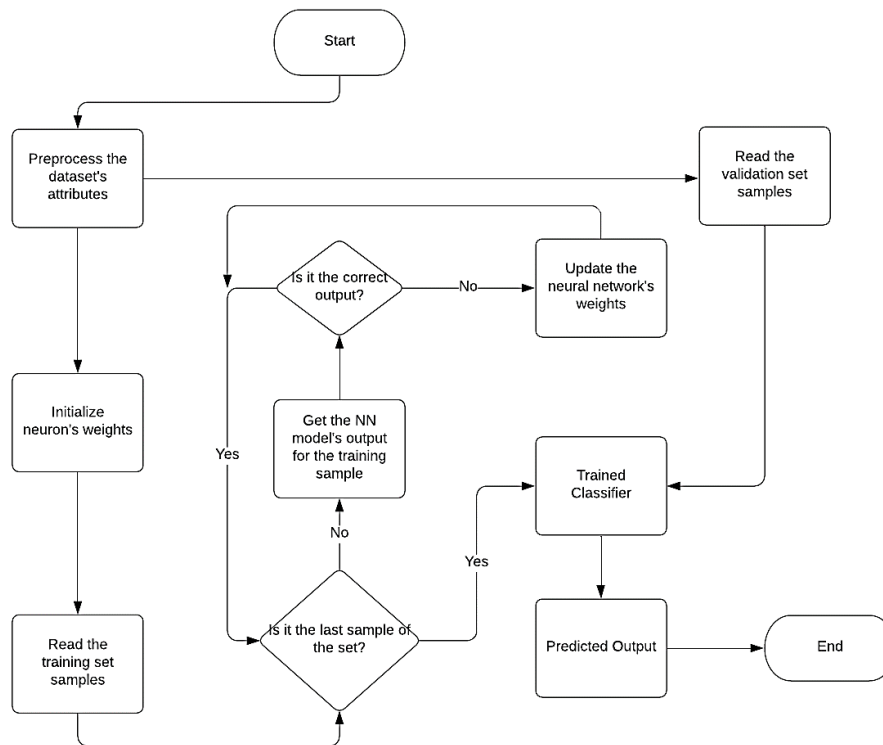


Figure 15 DNN model's flowchart

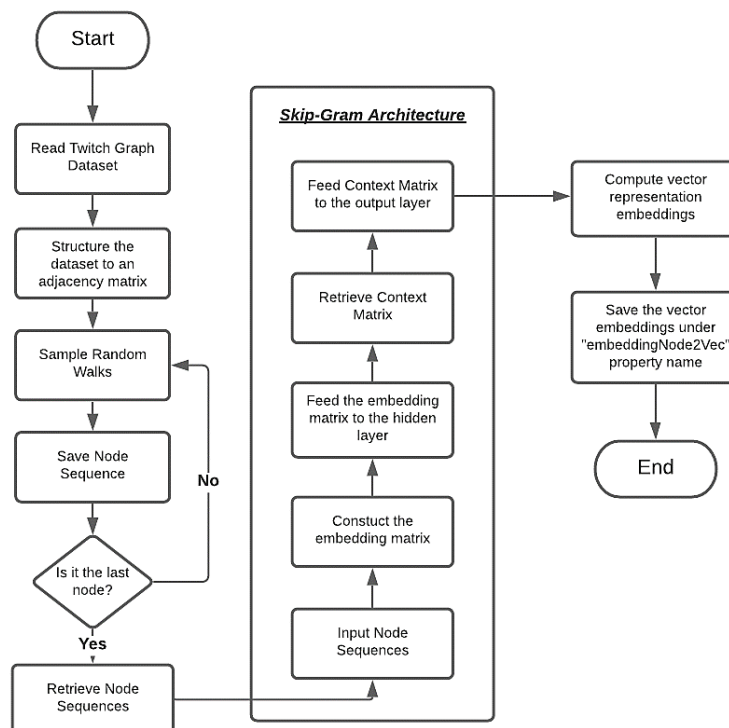


Figure 16 node2vec algorithm's flowchart

#### 4.4.4 Source code development

The source code of this project was developed using the Python programming language and the Visual Studio IDE. Each program was developed in a separate python file, allowing for them to be executed independently from each other. Several of the Python scientific libraries were used to handle different aspects of the code writing, the most used one being Pandas, NumPy, Scikit-learn, Keras and Tensorflow libraries, along with the Matplotlib and Seaborn visualization libraries and the Neo4j driver package. Additionally, an object-oriented approach was utilized, allowing for a better organization of the code and re-usability.

To begin with, the DNN classification model's development was heavily based on the Keras framework built as a simplified interface on top of the Tensorflow backend. Through their high and low-level APIs, the building, training, and validation process of the model was easier and more intuitive. After extracting the data from the csv file and loading it into a Pandas data frame, each language category of the dataset was plotted for the general skewness of the data to be displayed. As a result of a heavy imbalance between the classes, the dataset was balanced using a resampling technique through the sample function provided by the Pandas library. After concatenating all the resampled data, the data frame's column containing the node embeddings was stripped from the brackets confining each one of its rows. Furthermore, due to the node embeddings being of a high dimension, the data of each row of the embeddings' column was split at a comma delimiter and saved into a NumPy array of lists. This array was then iterated, with each record being converted into a float data type and appended to a Python list. For this list to be in the proper format that is utilizable by the DNN model, it was converted into a NumPy array of NumPy arrays. After pre-processing each sample of this array and retrieving their respective target labels, both the samples and labels were split into the training and validation sets. The labels of the training and validation sets were then transformed into one hot encoded vector that could be recognised by the model.

Continuing, the model architecture was built using the Keras framework along with the specification of its activation, loss functions and optimization algorithm. Finally, the model was trained and tested against the configuration specified in section 4.4.1, as well as evaluated using the Keras evaluate function. For a better understanding, the training and validation accuracies and losses are displayed into their plots using the Matplotlib library. Lastly, the accuracy metrics of F1-score, precision, and recall are calculated and



displayed along with a plotted heatmap version of the model's predicted confusion matrix.

#### 4.4.4.1 Google Colaboratory

For the training and evaluation process of the DNN model Google Colaboratory was employed. As a result of its ability to run Python scripts with no prior configuration required and access to GPUs the process of importing, analysing, and balancing the dataset was easier and faster than it would normally be on the computer's central processing unit (CPU). This was also made possible due to Colab's ability to make full use of the popular Python scientific libraries such as Pandas, NumPy, Matplotlib, TensorFlow, Scikit-learn and Seaborn that allowed for a better data processing and visualization. Furthermore, since the Colab notebooks make use of Google's cloud servers and hardware such as GPUs and TPUs, building the Machine Learning DNN as well as training and evaluating it only took a few lines of code and little time for the results to be produced.

The development of the first auxiliary program used for the modification of the original dataset, as explained in section 4.1, made use of the Pandas and Scikit-learn libraries. Two functions are dedicated to the process of filtering the records of the original csv files containing the edge and node information, based on specified user input. More specifically the program takes as input the names of the original csv files that should have been priorly placed into the specified directory, along with the range of streamer ids that are of interest for the study. It then proceeds to read the files and load their contents into their respective Pandas data-frame, filtering the records whose ids are not part of the specified range. Before saving the new data-frames into their new csv files, the 'views' property values are scaled into a range of 0 to 1 using Min-Max normalization, for a more consistent and smooth interpretation of the data. The original column's values are then replaced by their scaled equivalents under the new column header 'views\_scaled'. Lastly, due to the dataset's language property values that serve as a target for our classification problem, being of the categorical type, a function was created to automatically encode each category into an integer while iterating the records. A representative label of each language category was assigned as a key to a dictionary, while the dictionary's length at the time of their assignment was used as the

corresponding value of the key-value pairs. While iterating, each language label was checked against the dictionary, retrieving the value for the key it corresponded to. This value was then appended to a list, which after the completion of the iteration process gets converted into a one-dimensional array also known as Pandas' Series that is stored as an additional column to the data-frame.

The second auxiliary program serves as a script to automatically run the Cypher queries and extract the final csv file, without having to manually interact and insert the queries into the Neo4j Browser. After having imported in the right directory the modified csv files outputted by the first auxiliary program, as well as performed all the steps described in section 4.3.1 on how to correctly set up your project on Neo4j Desktop, the script will establish a connection with the local database using the Neo4j Driver Bolt protocol. Consequently, the Cypher queries will be executed in the form of transactions. The queries to be performed are:

- A Cypher query that loads the csv file containing node information and models the information into the corresponding node properties.
- A Cypher query that creates an index on the 'id' property of the established nodes. This allows for a faster establishment of the edges that will connect the nodes.
- A Cypher query that loads the content of the filtered edges csv file and creates the relationships between the nodes created in the first query. The execution speed of this query varies depending on the CPU power Neo4j Desktop is running on and the number of edges that need to be created.
- A Cypher query that checks for the existence of nodes without any incoming or outgoing edges, also known as orphan nodes, and deletes them. This is an extra step that needs to be executed in cases when working with subsets of the original dataset.
- A Cypher query that creates a temporary native in-memory projection of the existing graph, making the declarations of the following queries more flexible and boosting their performance. For the execution of this query the GDS plugin should have been priorly installed.
- A Cypher query that executes the Weakly Connected Components algorithm, in order to identify existing independent clusters, if any, within the graph. After having extracted the component ids of the identified clusters and the nodes that

belong to each one of them, the cluster sizes are calculated. While comparing the cluster sizes to one another, a correlated subquery is used to delete all the clusters except the one with the largest number of nodes. This allows for the graph we want to run the node embedding algorithm to be one connected component, giving a better representation of the embedding space. For the execution of these queries both APOC and GDS libraries are required.

- A Cypher query that will execute the node2vec embedding algorithm on the graph obtained after having performed all the aforementioned modifications.
- A Cypher query that will retrieve the node ids, embeddings and encoded language categories of all graph nodes and save them into a csv file that will be outputted in the same 'Import' folder directory, where the loaded csv files in the first query were placed in. This folder will become the dataset used to train and validate the created DNN model.

Lastly, the third auxiliary program is used as a visualization tool of the graph embedding space, after the Node2Vec embedding algorithm has been executed. Similarly, to the second program, a connection to the active database is established through the Neo4j Driver Bolt protocol. Through a Cypher query, all node embeddings along with the respective node streaming language labels are retrieved. The records are structured as a dictionary which is later converted into a Pandas data-frame consisting of three columns: node id, node embeddings and node language label. A high-dimensional data t-SNE visualization tool is then used on the data-frame's embeddings column to reduce the number of embedding dimensions from 15 to 2, so that they can be easier to plot and understand. After choosing the first dimension of the reduced embeddings for the values of x-axis, and the second dimension for the values of y-axis, the data is visualized in a scatterplot form using the Seaborn visualization library that is built on top of Matplotlib. By using this library, we can visualize the nodes of each language category through a different colour, noticing the different clusters representing the network communities created based on their streaming languages. This is done by t-SNE minimizing the relative entropy between the joint probabilities that are created by the conversion of the similarities between the retrieved data points.

## 4.5 Description of hardware

For the implementation of this project, my personal laptop was used with the following specs: AMD A10-9600P RADEON R5, 10 COMPUTE CORES 4C+6G 2.40 GHz, 12.0GB of RAM, 238GB HDD, 1TB SSD, AMD Radeon(TM) R5 Graphics GPU run on a 64-bit Windows Operating System.

## 5 Characterization, Validation and Discussion

As explained in section 3.4, due to the nature of this research it is important for a quantitative validation process to be done in a simultaneous manner with the rest of the project's development. As a result, for a more objective evaluation of the model, each training phase is followed by the calculation of the confusion matrix, out of which are derived different performance metrics such as Precision, Recall and F1-score.

Considered as one of the most intuitive evaluation metrics for finding the correctness of the classification model, the confusion matrix is a two-dimensional table that serves as reference guide in distinguishing:

- True positives (TP) - Cases when the actual class of the data point and the predicted are both 1 (True).
- True Negatives (TN) - Cases when the actual class of the data point and the predicted are both 0 (False).
- False Positives (FP) - Cases when the actual class of the data point was 0 (False) and the predicted 1 (True).
- False Negatives (FN) - Cases when the actual class of the data point was 1 (True) and the predicted is 0 (False).

Deriving from the context of the confusion matrix, precision highlights the ratio between true positive predictions and all positive predictions in total. Mathematically is it expressed as:

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

Equation 9 Calculation of the precision metric

Recall, also known as Sensitivity measures, how accurately the classifier can identify true positives. Similar to precision it can be expressed mathematically as:

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

Equation 10 Calculation of the recall metric

Lastly, F1-Score, also referred to as the Harmonic Mean, combines both the model's precision and recall while treating them with the same importance. Mathematically it can be denoted as:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Equation 11 Calculation of F1-score metric

## 5.1 Characterization Experiments and Results

In this section the implemented validation techniques used on the created DNN model will be described. Their results will be thoroughly explained and discussed on section 5.2.

### 5.1.1 DNN Characterization Experiments

The validation analysis of the model's algorithmic performance relied on the subset selected out of the balanced dataset after being split at a ratio of 70% to 30%. This subset consists of 2672 validation samples selected randomly out of all 21 categories.

#### 5.1.1.1 Orthogonalization

One of the challenges many face during the process of developing their own Machine Learning models, is finding the right configuration for their model. This configuration is directly connected to the strategy followed for the selection of the right hyperparameters, metrics, loss function and optimizer as well as the process of tuning them so that the classifier's training, testing, and evaluation performance is of a satisfactory level. Orthogonalization is considered as one of the most effective techniques used to change and tune one hyperparameter at a time while keeping the rest of the model's settings and configuration set. More specifically, the hyperparameters that were constantly changed and tested were the model's number of layers, the number of their neurons, the number of training epochs, the learning rate, and the batch size. Additionally, different loss functions and optimizers were employed, the selected ones being the ones that provided the best validation performance.

#### 5.1.1.2 Node2vec Parameter Sensitivity

This validation method focuses on evaluating how changes to the parameterization of the node2vec algorithm affect the DNN model's performance on the classification task. After having set a fixed window size and a sensible range of values for the walk length, different modifications were made to the number of walks started per vertex and the number of latent dimensions. These combinations were then studied to determine how the parameters impact the performance of the node classification process.

#### 5.1.1.3 K-Fold Cross-Validation

The ultimate goal of building a Machine Learning model is for the latter to be able to generalize and maintain its training performance on unseen data. Having said that, the popular data partitioning strategy known as K-Fold Cross-Validation was implemented to evaluate the developed DNN model's performance on the limited, unseen data samples. This method provided a less biased estimation of the model's skills, mimicking a potential real-life scenario. By splitting the existing training set into a defined number of  $k$  folds of the same size, each unique group was used as a hold out test set one iteration at a time, while the model was trained on the remaining  $k-1$  fold data. Each iteration is followed by the retrieval of the training and testing accuracy of the model before resetting the model to its initial state, for the training process of the following iteration not to be affected by the existing weights. After the last iteration, where the model is trained and tested on the last fold, the model's mean evaluation accuracy is retrieved. This evaluation is then followed by the model's final evaluation score, tested on the initial validation set created by splitting the original dataset into a ratio of 70% to 30%. An example of this process is also shown in the image below:

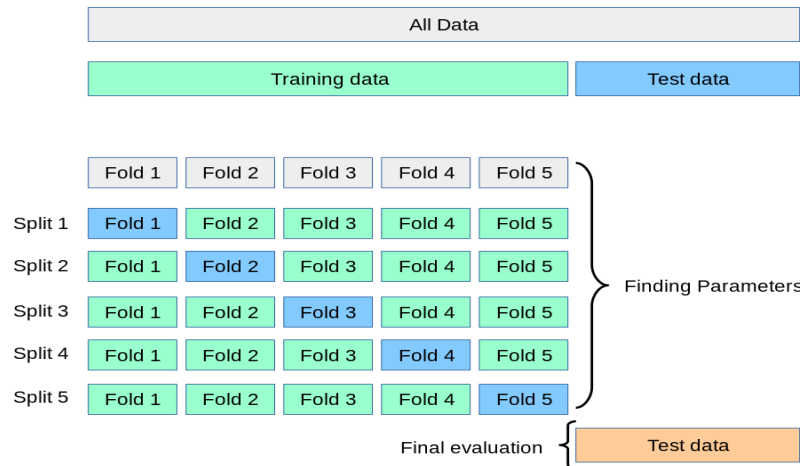


Figure 17 Example of 5-Fold Cross-Validation

### 5.1.2 Validation Software

The source code created for the evaluation of the model was developed utilizing the Google Colaboratory and the Python scientific libraries. The Keras framework was used to calculate the accuracy and loss performance metrics of the network, by keeping track of the training and validation history. Additionally, the Scikit-learn ML library was used to calculate the already mentioned classification metrics, which were later visualized through the use of the Seaborn and Matplotlib visualization libraries.

### 5.1.3 Validation Hardware

For the evaluation of the Deep Neural Network model, my personal laptop was used with the following specs: AMD A10-9600P RADEON R5, 10 COMPUTE CORES 4C+6G 2.40 GHz, 12.0GB of RAM, 238GB HDD, 1TB SSD, AMD Radeon(TM) R5 Graphics GPU run on a 64-bit Windows Operating System.

## 5.2 Validation Experiments and Results

This section will contain a detailed description of the training and validation results obtained during the process of evaluating the performance of the developed DNN model.



## 5.2.1 DNN Validation Experiments

### 5.2.1.1 Baseline model

Throughout the development and validation process of the DNN model, many experiments were conducted for the final model to have the most optimal performance and generalization skills.

To begin with, as already mentioned in section 4.4.1 the DNN model employs the Categorical Hinge loss function, Adam optimizer and Categorical Accuracy metric. The activation function of the first 5 modules was Tanh, while the Linear activation was used on the output layer. Its' configuration can be summarized as follows:

- Batch size: 55
- Epochs: 50
- Learning rate: 0.0001
- Feature Number: 15 (equal to the dataset's number of embedding dimensions per node)
- Number of neurons in the 1<sup>st</sup> module: 128
- Number of neurons in the 2<sup>nd</sup> module: 256
- Number of neurons in the 3<sup>rd</sup> module: 300
- Number of neurons in the 4<sup>th</sup> module: 256
- Number of neurons in the 5<sup>th</sup> module: 128

Moreover, as explained in sections 4.2.3.2 and 4.2.3.3, both normalization and standardization techniques were separately applied to the balanced dataset, to denote which led to a better performance. Each technique was tested twice: on the full validation set and on 20% of the training dataset, after performing a validation split. The results can have been summarized in the following table:

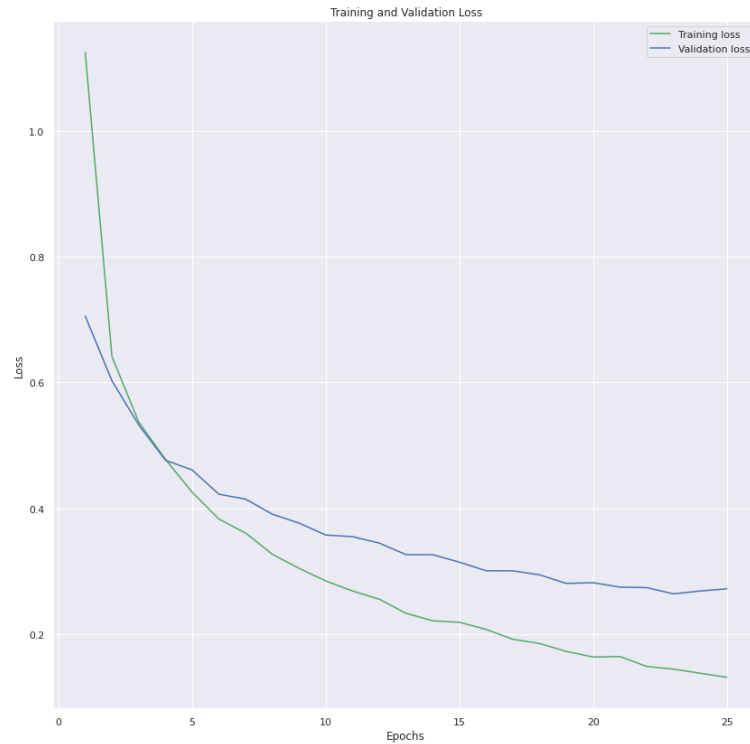
		Training		Testing		Evaluation	
		Error	Accuracy	Error	Accuracy	Error	Accuracy
Validation Set = 20%	Normalization	0.1849	0.9330	0.2763	0.8885	0.2972	0.8780
	Standardization	0.1313	0.9587	0.2721	0.8982	0.2709	0.8967
Full Validation Set	Normalization	0.1032	0.9674	0.2036	0.9233	0.2036	0.9233
	Standardization	0.1081	0.9641	0.2214	0.9109	0.2214	0.9109

**Table 1 Effects of Preprocessing Methods on DNN's Accuracy**

From the table it can be noted that in both testing scenarios of the model the standardization pre-processing method leads to better performance with a higher accuracy and smaller error.

This led to choosing standardization as the main preprocessing method, upon which the rest of the experiments would follow.

The following plots display the training and validation losses and accuracies respectively:



**Figure 18 The training and validation loss recorded during the training of the classifier.**

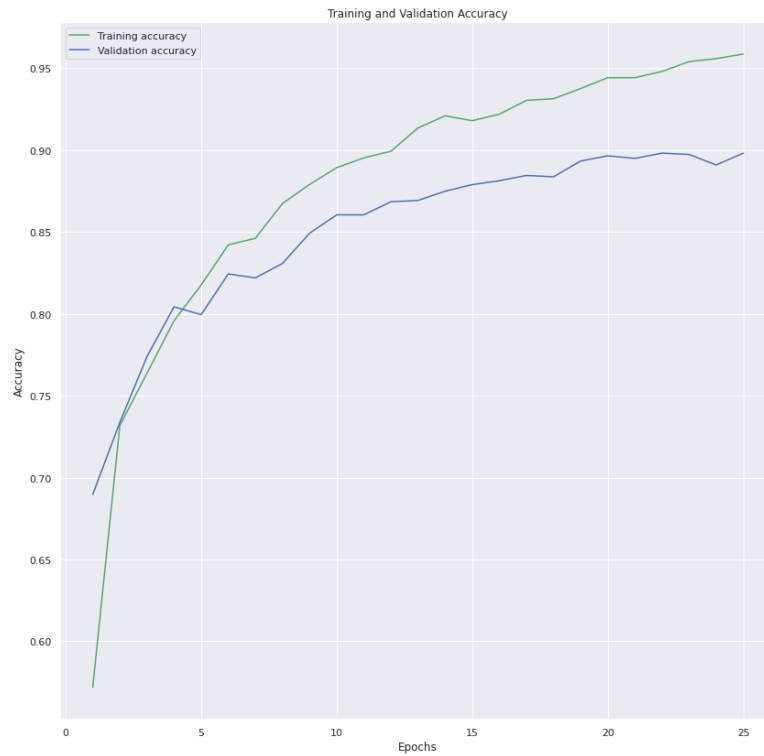


Figure 19 The training and validation accuracy recorded during the training of the classifier.

These plots were followed by the calculation of the confusion matrix as well the performance metrics derived from it:

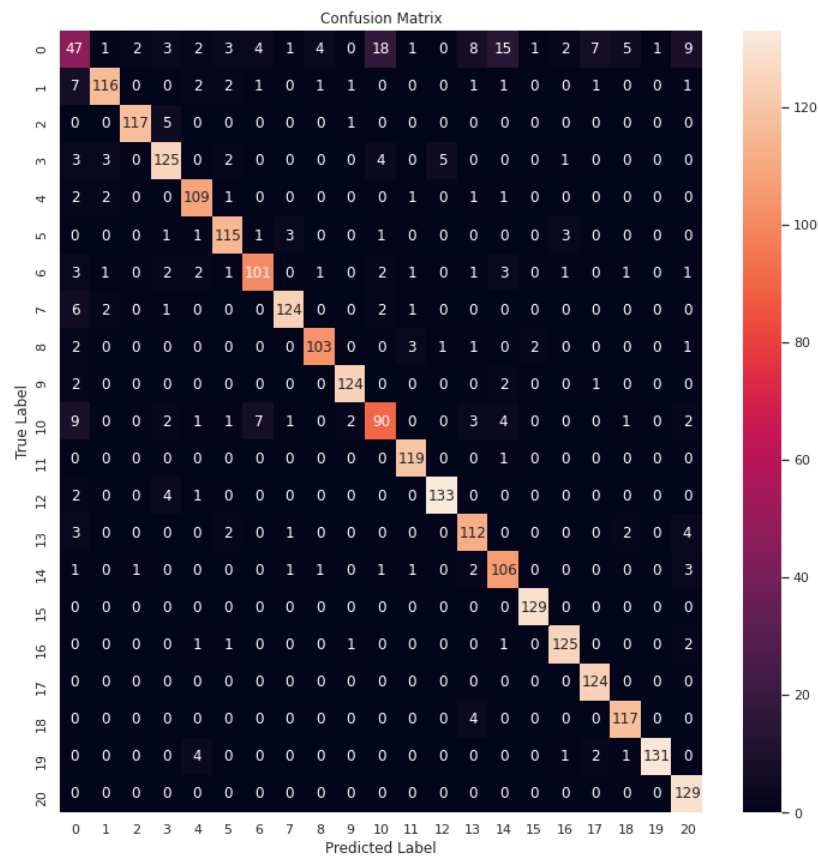


Figure 20 The calculated confusion matrix of the baseline DNN model

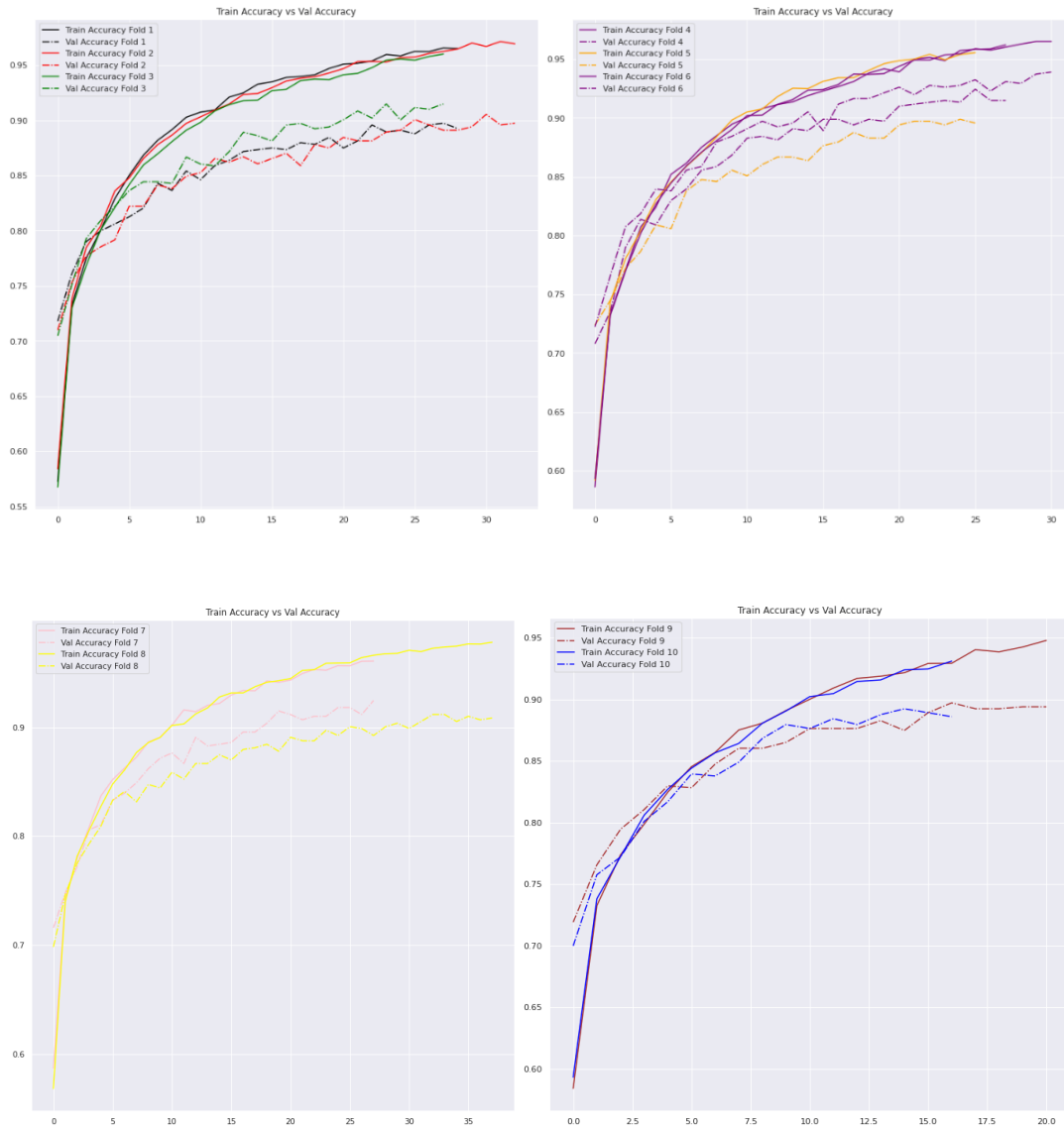
	Precision	Recall	F1-score	Support
0	0.53	0.35	0.43	134
1	0.93	0.87	0.90	134
2	0.97	0.95	0.96	123
3	0.87	0.87	0.87	143
4	0.89	0.93	0.91	117
5	0.90	0.92	0.91	125
6	0.89	0.83	0.86	121
7	0.95	0.91	0.93	136
8	0.94	0.91	0.92	113
9	0.96	0.96	0.96	129
10	0.76	0.73	0.75	123
11	0.94	0.99	0.96	120
12	0.96	0.95	0.95	140
13	0.84	0.90	0.87	124
14	0.79	0.91	0.84	117
15	0.98	1.00	0.99	129
16	0.94	0.95	0.95	131
17	0.92	1.00	0.96	124
18	0.92	0.97	0.94	121
19	0.99	0.94	0.97	139
20	0.85	1.00	0.92	129

Accuracy			0.90	2672
Macro avg	0.89	0.90	0.89	2672
Weighted avg	0.89	0.90	0.89	2672

**Table 2 Performance Metrics of the DNN**

#### 5.2.1.2 K-Fold Cross-Validation on the baseline model

For a better evaluation of the model's generalization skills a 10-Fold Cross-Validation was implemented, simulating the scenarios similar to those to be faced in the real world. In the implementation where  $k=10$ , after having split the original dataset into a training and validation set with a ratio of 70% - 30%, the later will be split into 10 equal groups of data, out of which one serves as the completely unknown validation set used to evaluate the model's performance after each iteration, while the training is conducted on the remaining 9 folds. Continuing, out of the training dataset consisting of these 9 folds, 10% is kept as validation data. The achieved evaluation accuracy with this implementation was 88.74%. The training and validation accuracies per fold, are also plotted:



**Figure 21 Training and Validation Accuracies Per Fold of the 10-Fold Cross Validation**

Lastly, the model's performance is evaluated against the actual testing set consisting of 30% of the initial dataset, split in the beginning of the process. The model's metrics were collected in two scenarios, one being when the model was validated against the entire fold that was selected as the validation fold, and the second one being when it was validated against 90% of the validation fold, keeping the last 10% as completely unknown data used for evaluation.

	<b>Evaluation</b>	
	<b>Error</b>	<b>Accuracy</b>
Validation with Validation Split = 10%	0.2778	0.8867
Validation Against Full Validation Fold	0.2635	0.8943

**Table 3 Summary of 10-Fold CV metrics**

### 5.2.1.3 Node2vec Parameter Sensitivity Validation

After having successfully established and validated the performance of the model that was to be considered as the baseline against which all the other experiments were to be made, the parameter sensitivity of the node2vec algorithm was also tested. More specifically, changes to the algorithm's walk length were made, increasing, and decreasing it in comparison to its initial length of 80 walks. The rest of the parameters are left unchanged, with an embedding dimension of 15. This parameter is extremely important in extracting useful embeddings that are rich in information and it is expected to play an important role in the model's ability to predict the labels correctly. The following table summarizes some of the collected observations:

	<b>Training</b>		<b>Validation</b>		<b>Evaluation</b>	
	<b>Error</b>	<b>Accuracy</b>	<b>Error</b>	<b>Accuracy</b>	<b>Error</b>	<b>Accuracy</b>
Walk length						
5	1.5234	0.2243	1.4807	0.2294	1.4795	0.2204
15	0.239	0.9071	0.3499	0.8581	0.3412	0.8589
40	0.1279	0.9579	0.2976	0.8829	0.3083	0.8772
80	0.1313	0.9587	0.2721	0.8982	0.2709	0.8967
90	0.1132	0.9629	0.2536	0.9046	0.2446	0.9004

**Table 4 Summary of how node2vec's walk length affects the DNN's prediction accuracy.**

## 5.3 Discussion

In this section the results of the developed DNN model will be discussed, along with the outcomes of the different validation experiments performed on it.

### 5.3.1 Discussion of Characterization Results

For the DNN to achieve a satisfactory performance, a series of iterative hyperparameter fine-tuning processes took place. In order for this process to be more organized, so as to easily notice the impact of every change on the performance of the model, the Orthogonalization technique was used. This method allowed the tuning procedure of each hyperparameter towards an optimal performance to be easier and achieved at a faster time. Furthermore, the method responsible for the preprocessing of the dataset's records is important in creating a model with good predictive performance. Thus, an iterative approach was followed for the selection of a pre-processing model that fit the dataset and model best.

The dataset's contribution to the model's accuracy however does not depend solely on the preprocessing method selected for its' normalization. In social media analysis and node classification applications, the initial parameterization of the employed algorithm for the extraction of the node embeddings plays a crucial role. Random walk algorithms such as node2vec have a series of parameters that need to be tuned for node embeddings that contain adequate information about the network's nodes, their features, space, and neighbors to be generated. In particular, the walk length parameter responsible for the number of steps completed during a random walk length was observed, for the best configuration to be selected.

Lastly, the performance of DNN models can sometimes be misleading. For a better insight of its generalization abilities, a 10-Fold Cross-Validation was implemented. This implementation was accompanied by a series of advantages such as the reduction in the model's predictive bias, and an increase in the model's predictive accuracy for unknown samples.

### 5.3.2 Discussions of Validation Results

The created Deep Neural Network achieved an evaluation accuracy of 89.67% with a loss of 0.2446 and performance metrics as follows: 89% for precision, 90% for recall and

89% for F1-score. By taking a deeper look at the respective metrics per each language category, displayed in Table 2, it was noticed that the only category with a precision less than 70% was that of the category 0, of users with English as their streaming language. This metric alone indicated that during the process of balancing the original dataset, in which English streamers were the predominant category, information was lost. By reducing the number of samples to 424, in this category a larger number of records was discarded when compared to the other categories. This was the trade-off of increasing the predictive accuracy for the minority classes.

Another factor that contributed to the good performance of the model was choosing the standardization technique as the applied pre-processing method. As it can be seen in Table 1, this approach had better generalization abilities, performing best on unknown data. This due to the standardization's ability of removing the scale dependence that the inputs have on their initial weights and biases, increasing the training time and reducing the possibility of being stuck in local optimum. Additionally, scaling the inputs to the closed interval of -1 to 1 allows for a better handling of the dataset's outliers by treating them with the same importance as the rest of the records.

Furthermore, by tuning the walk length parameter of the node2vec random walk algorithm, the DNN classifier was provided richer data to work with. Setting the parameter to a length of 80 with an in and out and return factor of 1, allowed the random walks to branch out further in the nodes' neighbourhoods, retrieving more information that was later represented in the form of node embeddings. A walk length of 40 also proved to generate embeddings rich in information, boosting the model's prediction accuracy to 87.72%. This slight decrease in accuracy when compared to the walk length of 80 can be justified by an increase in the performance of the node2vec embedding generation process. The effect that this parameter had on the increase of the classifier's performance can be seen in table 4.

Lastly, the 10-Fold Cross-Validation of the DNN's performance provided a better insight on how the model is to perform with real life data, with an average evaluation accuracy of 88.67% that has been displayed in table 3. The 1% difference in accuracy from the initial run mentioned in the beginning of this section is a strong indicator that the performance of the model is stable throughout the different folds.

## 5.4 Conclusions



After experimenting with different architectures, configuration metrics and hyperparameters it was noted that the classification accuracy of the node embeddings was the highest when the DNN model was paired to the unique combination of the Tanh and Linear activation functions along with Categorical Hinge loss function. Continuing, several iterative testing processes confirmed that the selection of the correct pre-processing technique and dataset balancing method highly impacted the performance and sensitivity of the model.

Regarding the semi-supervised node2vec embedding algorithm, experiments on its configuration proved that representations of node features were richer when extracted by random walks of long length. Being used as the input dataset for the classifier, the richness in information that these representations contained directly affected the model's predictive accuracy. Additionally, the relatively - deep architecture of the model allowed for a better detection of feature patterns of the dataset's node embeddings.

The findings collected by the combination of the semi-supervised node2vec feature learning algorithm with the developed Deep Neural Network fully supported our project thesis, proving that the correct pairing of both algorithms can assist in the existing graph labelling problem; predicting people's spoken language based on information extracted from their social media profiles.

## 6 A look into the future

### 6.1 Advances in the technologies relevant to the project

Throughout recent years, performing graph analysis through the use of Machine Learning techniques has been receiving progressively more attention. While the developments in the field of Deep Learning have created models that excel at learning from the large amounts of data constantly generated by the graph data structures that are social networks, the complexity of this data at times, requires for further research and improvement of the current state of art. The main challenge of being unable to use the existing downstream Machine Learning models in several tasks such as classification, clustering, link prediction or anomaly detection stands is extracting the key features from non-Euclidian structured networks [36]. For this difficulty to be avoided in a way that is not expensive and time-consuming, I believe that in the near future the focus will be placed into developing more efficient network feature representation methods.

One of the most promising methods seem to be Graph Convolutional Networks (GCN) [37]. Inspired by the continuous improvement in the performance of Convolutional Neural Networks (CNN) in the domain of image recognition, many successful attempts have been made into adapting these modules into learning network data representations. Due to the encoding function of the graph convolutional approaches being able to leverage along with the node's attributes, its local neighborhood information, efforts into applying the traditional CNNs to social network data have been made, assisting in the process of generating network embeddings.

The development of efficient network embedding techniques in my opinion will have a positive impact not only in the economic and social sectors, but also in different domains ranging from computer vision to bioinformatics [38]. On the other hand, by automating everything, the human economic value will be greatly depreciated as the manpower will be less sought after.

### 6.2 Future improvements (on this project)

A substantial improvement to this project would be developing and converting the completed research work into a software application that appeals to a wider audience and has a wider usability. This includes generalizing the existing DNN

model and auxiliary programs in a way that they can be easily tuned to new datasets extracted from social network platforms other than Twitch. This includes using the already developed and trained DNN model to implement transfer learning as well as converting the existing Neo4j Cypher queries into parameterized. Additionally, the development of a more interactive and aesthetically pleasing GUI can take place. This will create fewer problems of interactivity, increase user involvement and strengthen the bond between the users and the software. Moreover, since the software application's target audience will consist of professionals interested in the analysis of social networks and how they are concentrated towards specific interests, tutorials that show the users how to interact with it could be included.

A second improvement could be related to the creation of an intelligent web application [39] that will use Machine Learning and graph theory to generate real time label classifications. Since the semantic web can connect the data and information found on the Web in a close manner, better emphasis will be put on e-learning, data access and information exchange making the transfer learning and node classification algorithms more effective.

Regarding the balancing problem of the existing dataset, further attempts will be made to balance it using the already mentioned modern techniques of ADASYN and SMOTE algorithms. If this scenario proves to be unfruitful, a web crawler can also be built to extract streamer data from the Twitch streaming platform, filtering the content in a way that the created dataset is originally balanced, avoiding the issue of balancing the dataset and consequently creating a more accurate classifier. Finally, better implementation and a larger number of the random walk algorithms could be provided. This will drastically improve the algorithmic performance along with their response time and statistical description provided by the algorithms. Although the Neo4j platform does not have an alpha release of the DeepWalk random walk algorithm, it accepts community contributions in the form of libraries, tools, drivers and more. Thus, the DeepWalk algorithm can be developed and implemented by the author into the platform in the form of a JAR file plugin, and then executed in a similar fashion with the node2vec algorithm.

## 6.3 Insights into future applications

While many Deep Learning models and representation techniques for network studies have been developed in the last few years, there are still many promising topics and applications that are currently under research.

One of these areas focuses on developing Deep Learning and network embedding techniques that can be applied to dynamic networks. The existing static learning approaches fail to work correctly when tackling real-life scenarios with highly dynamic social networks. This comes due to the fact that there is almost always the assumption that the number of nodes within the network is set, with no changes in their features being made. The establishment of a method that takes into consideration the network's constantly changing features will significantly improve the ability and accuracy of existing applications such as recommendation systems, anomaly detection, link prediction or classification applications.

Continuing, I believe another area that will witness breakthrough will be the techniques used to learn embeddings by approximating the semantic similarity existing between the embeddings of heterogeneous networks. This will create applications that are more in tune with real-life networks, which deal with a diversity of data seen via the different types of nodes and edges.

Seeing that one elementary limitation that almost all Deep Learning models have is interpretability, I believe that the task of developing models which are able to thoroughly interpret their learning outcomes and can improve based on them, will receive a lot more attention in the near future. Based on their level of interpretability skills, these applications will have the ability to learn embeddings that are meaningful and task-specific, rather than trying to find all possible representation patterns that exist within the network's embedding space.

I foresee that these new applications with improved scalability, interpretability, and evaluation technique of dynamic and heterogeneous networks, will highly impact not only the already mentioned fields of applications, but also the medical sector of studies. The improved feature representation learning techniques will lead to a better identification of the genetic and protein interactions and additional discoveries in the domain of biology and genetics.

Personally, I believe that the application of DL and SNA in all these different domains will impact our society both positively and negatively. While the quality and accuracy of

the offered customer and healthcare services will increase, the job market will have to drastically change for it to adapt to the latest trends. This will result in many people losing their jobs because of a lot of services being converted to automation, not requiring human expertise. On the other hand, many promising job sectors will emerge, and others will continue to be developed, leading to the creation of several employment opportunities for people with the necessary skills.

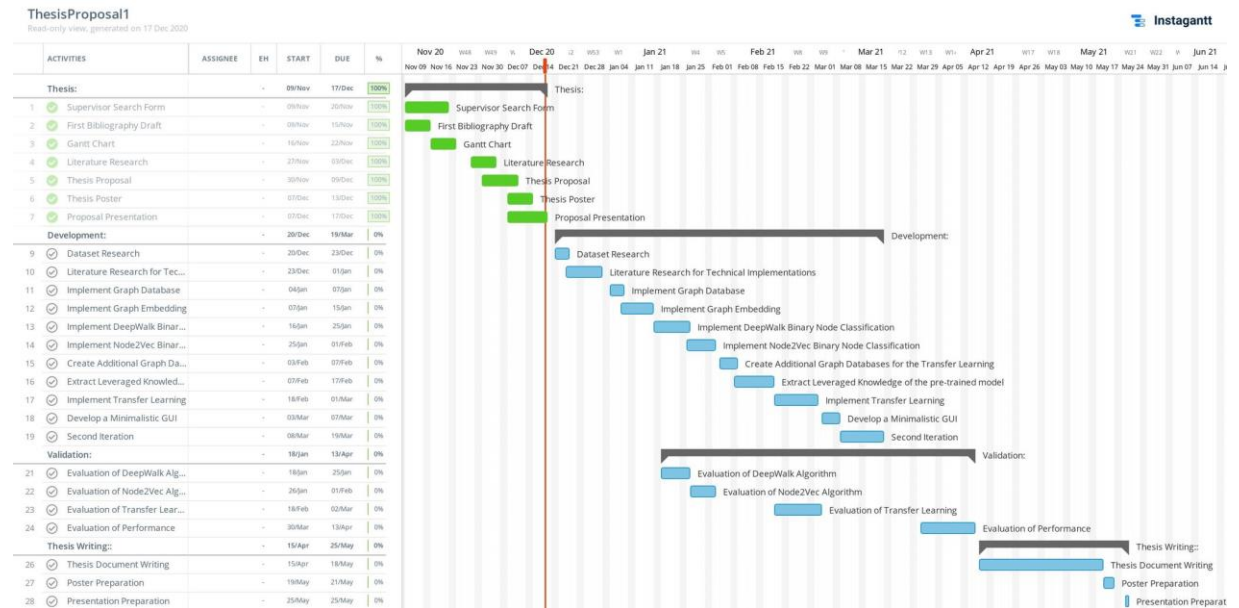
# Appendices

## Initially Proposed Design Requirements

The list of the design requirements proposed initially is as following:

- The user should be able to insert new unlabeled data in the graph database.
- The system should use the knowledge extracted from the pre-trained model on binary node classification, to categorize the unlabeled data.
- The system should visualize the classification of the entered data.
- The system should produce descriptive statistics for the data.
- The system should have a response time up to one minute.

# Initially Proposed Timeline



## Table of figures

Figure 1 Phases of Random Walk approach .....	12
Figure 2 Random probabilities of node2vec's random walk step.....	13
Figure 3 Project Timeline .....	18
Figure 4 Unbalanced Dataset .....	23
Figure 5 Balanced Dataset .....	24
Figure 6 Creating a new project.....	27
Figure 7 Adding a local database.....	27
Figure 8 Database initialization .....	28
Figure 9 Accessing the managing options. ....	28
Figure 10 Plugin installation.....	29
Figure 11 Configuration settings .....	29
Figure 12 Configuration settings (2).....	30
Figure 13 Importing csv files.....	30
Figure 14 DNN architecture.....	32
Figure 15 DNN model's flowchart.....	36
Figure 16 node2vec algorithm's flowchart.....	36
Figure 17 Example of 5-Fold Cross-Validation.....	45
Figure 18 The training and validation loss recorded during the training of the classifier. ....	47
Figure 19 The training and validation accuracy recorded during the training of the classifier. ....	48
Figure 21 The calculated confusion matrix of the baseline DNN model .....	48
Figure 20 Training and Validation Accuracies Per Fold of the 10-Fold Cross Validation.....	50



## Table of equations

Equation 1 node2vec's approach to random walks .....	12
Equation 2 Min-Max normalization.....	25
Equation 3 Feature Standardization .....	25
Equation 4 tanh activation function .....	32
Equation 5 Categorical Hinge loss function .....	33
Equation 6 Calculation of maximum log-probability .....	33
Equation 7 Optimization of the maximum log-probability function .....	34
Equation 8 Summary of search bias in random walks .....	34
Equation 9 Calculation of the precision metric.....	42
Equation 10 Calculation of the recall metric .....	42
Equation 11 Calculation of F1-score metric .....	43

## Table of tables

Table 1 Effects of Preprocessing Methods on DNN's Accuracy .....	47
Table 2 Performance Metrics of the DNN .....	49
Table 3 Summary of 10-Fold CV metrics.....	51
Table 4 Summary of how node2vec's walk length affects the DNN's prediction accuracy. ....	51

## Bibliography

- [1] S. Bhagat, G. Cormode, and S. Muthukrishnan, “Node Classification in Social Networks,” in *Social Network Data Analytics*, C. C. Aggarwal, Ed. Boston, MA: Springer US, 2011, pp. 115–148.
- [2] Z. Wang, Y. Tan, and M. Zhang, “Graph-Based Recommendation on Social Networks,” in *2010 12th International Asia-Pacific Web Conference*, Busan, Korea (South), Apr. 2010, pp. 116–122, doi: 10.1109/APWeb.2010.60.
- [3] S. Tiwari, E. Suryani, A. K. Ng, K. K. Mishra, and N. Singh, Eds., *Proceedings of International Conference on Big Data, Machine Learning and their Applications: ICBMA 2019*. Springer Singapore, 2021.
- [4] A. Lorena, A. Carvalho, and J. Gama, “A review on the combination of binary classifiers in multiclass problems,” *Artificial Intelligence Review*, vol. 30, pp. 19–37, Dec. 2008, doi: 10.1007/s10462-009-9114-9.
- [5] J. L. Larriba-Pey, N. Martínez-Bazán, and D. Domínguez-Sal, “Introduction to Graph Databases,” in *Reasoning Web. Reasoning on the Web in the Big Data Era: 10th International Summer School 2014*, Athens, Greece, September 8-13, 2014. *Proceedings*, M. Koubarakis, G. Stamou, G. Stoilos, I. Horrocks, P. Kolaitis, G. Lausen, and G. Weikum, Eds. Cham: Springer International Publishing, 2014, pp. 171–194.
- [6] “Graph Algorithms [Book].” <https://www.oreilly.com/library/view/graph-algorithms/9781492047674/> (accessed Dec. 21, 2020).
- [7] I. Robinson, J. Webber, and E. Eifrem, *Graph databases*, Second edition. Beijing: O’Reilly, 2015.
- [8] H. Cai, V. W. Zheng, and K. C.-C. Chang, “A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications,” *arXiv:1709.07604 [cs]*, Feb. 2018, Accessed: Dec. 21, 2020. [Online]. Available: <http://arxiv.org/abs/1709.07604>.
- [9] F. Fouss, A. Pirotte, J. Renders, and M. Saerens, “Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 3, pp. 355–369, Mar. 2007, doi: 10.1109/TKDE.2007.46.
- [10] M. E. Newman, “A Measure of Betweenness Centrality based on Random Walks,” *Social Networks*, vol. 27, pp. 39–54, Oct. 2003, doi: 10.1016/j.socnet.2004.11.009.
- [11] B. Perozzi, R. Al-Rfou, and S. Skiena, “DeepWalk: Online Learning of Social Representations,” *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’14*, pp. 701–710, 2014, doi: 10.1145/2623330.2623732.
- [12] P. Goyal and E. Ferrara, “Graph Embedding Techniques, Applications, and Performance: A Survey,” *Knowledge-Based Systems*, vol. 151, pp. 78–94, Jul. 2018, doi: 10.1016/j.knosys.2018.03.022.
- [13] W. Zaremba, I. Sutskever, and O. Vinyals, “Recurrent Neural Network Regularization,” *arXiv:1409.2329 [cs]*, Feb. 2015, Accessed: Dec. 21, 2020. [Online]. Available: <http://arxiv.org/abs/1409.2329>.
- [14] A. Grover and J. Leskovec, “node2vec: Scalable Feature Learning for Networks,” in *Proceedings of the 22nd ACM SIGKDD International Conference*

- on Knowledge Discovery and Data Mining, San Francisco California USA, Aug. 2016, pp. 855–864, doi: 10.1145/2939672.2939754.
- [15] D. Jungnickel, *Graphs, Networks and Algorithms*. Berlin Heidelberg: Springer Verlag, 1999.
  - [16] S.-H. Yang, B. Long, A. Smola, N. Sadagopan, Z. Zheng, and H. Zha, “Like like alike: joint friendship and interest propagation in social networks,” in *Proceedings of the 20th international conference on World wide web - WWW '11*, Hyderabad, India, 2011, p. 537, doi: 10.1145/1963405.1963481.
  - [17] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, Art. no. 7553, May 2015, doi: 10.1038/nature14539.
  - [18] A. Maxwell et al., “Deep learning architectures for multi-label classification of intelligent health risk prediction,” *BMC Bioinformatics*, vol. 18, no. 14, p. 523, Dec. 2017, doi: 10.1186/s12859-017-1898-z.
  - [19] Q. Tan, N. Liu, and X. Hu, “Deep Representation Learning for Social Network Analysis,” *Front. Big Data*, vol. 2, 2019, doi: 10.3389/fdata.2019.00002.
  - [20] P. Compagnon and K. Ollivier, “Graph Embeddings for Social Network Analysis: State of the Art,” Jan. 2017.
  - [21] “Benefits of Facebook „Friends:“ Social Capital and College Students“ Use of Online Social Network Sites | Journal of Computer-Mediated Communication | Oxford Academic.”  
<https://academic.oup.com/jcmc/article/12/4/1143/4582961?login=true> (accessed Dec. 21, 2020).
  - [22] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, Nov. 2008.
  - [23] “Neo4j APOC Library - Developer Guides,” Neo4j Graph Database Platform. <https://neo4j.com/developer/neo4j-apoc/> (accessed Jun. 07, 2021).
  - [24] Q. Wu and D.-X. Zhou, “Analysis of support vector machine classification,” *Journal of Computational Analysis and Applications*, vol. 8, Apr. 2006.
  - [25] J. A. K. Suykens and J. Vandewalle, “Training multilayer perceptron classifiers based on a modified support vector method,” *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 907–911, Jul. 1999, doi: 10.1109/72.774254.
  - [26] M.-L. Zhang, J. M. Peña, and V. Robles, “Feature selection for multi-label naive Bayes classification,” *Information Sciences*, vol. 179, no. 19, pp. 3218–3229, Sep. 2009, doi: 10.1016/j.ins.2009.06.010.
  - [27] “Neo4j Graph Data Science - Developer Guides.”  
<https://neo4j.com/developer/graph-data-science/> (accessed Jun. 07, 2021).
  - [28] D. Berrar, “Cross-Validation,” 2018. doi: 10.1016/B978-0-12-809633-8.20349-X.
  - [29] B. Rozemberczki, C. Allen, and R. Sarkar, “Multi-scale Attributed Node Embedding,” arXiv:1909.13021 [cs, stat], Mar. 2021, Accessed: Jun. 07, 2021. [Online]. Available: <http://arxiv.org/abs/1909.13021>
  - [30] B. Rozemberczki and R. Sarkar, “Twitch Gamers: a Dataset for Evaluating Proximity Preserving and Structural Role-based Node Embeddings,” arXiv:2101.03091 [cs], Feb. 2021, Accessed: Jun. 07, 2021. [Online]. Available: <http://arxiv.org/abs/2101.03091>
  - [31] “Weakly Connected Components - Neo4j Graph Data Science,” Neo4j Graph Database Platform. <https://neo4j.com/docs/graph-data->

- science/1.6/algorithms/wcc/ (accessed Jun. 07, 2021).
- [32] R. Ajayi, “Acid Properties, CAP Theorem & Mobile Databases,” Jun. 2015. doi: 10.13140/RG.2.1.1941.0084.
  - [33] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *jair*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.
  - [34] H. He, Y. Bai, E. Garcia, and S. Li, “ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning,” Jul. 2008, pp. 1322–1328. doi: 10.1109/IJCNN.2008.4633969.
  - [35] P. Muhammad Ali and R. Faraj, Data Normalization and Standardization: A Technical Report. 2014. doi: 10.13140/RG.2.2.28948.04489.
  - [36] B. R V, E. Kanaga, and S. Kundu, “Application of Machine Learning in the Social Network,” 2020, pp. 61–83. doi: 10.1002/9781119551621.ch4.
  - [37] J. Zhou et al., “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, 2020, doi: 10.1016/j.aiopen.2021.01.001.
  - [38] S. S. De, S. Dehuri, and G.-N. Wang, “Machine Learning for Social Network Analysis: A Systematic Literature Review,” *The IUP Journal of Information Technology*, vol. 8, pp. 30–51, Jan. 2012.
  - [39] “THE SEMANTIC WEB on JSTOR.” <https://www.jstor.org/stable/26059207> (accessed Dec. 21, 2020).