SAPIENZA
UNIVERSITÀ DI ROMA

Improving Bot Detection with Language Model Embeddings and Graph Representations

Faculty: Ingegneria dell'Informazione, Informatica e Statistica
Bachelor's Degree in Applied Computer Science and Artificial Intelligence

Antonio Cordeiro
ID number 1999975

Advisor
Prof. Angelo Spognardi

Co-Advisor
Prof. Dorjan Hitaj

Academic Year 2023/2024

Improving Bot Detection with Language ModelEmbeddings and Graph Representations

Bachelor's Degree Thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: cordeiro.1999975@studenti.uniroma1.it

Contents

Chapter 1

Introduction

Every day, when we use social media, we encounter fake and automated accounts, commonly known as bots. Bots have been present online from the early days of social networks, and over the years, a true "bot pandemic" has been observed. In 2017, the proportion of bots on Twitter[1] was estimated to range between 9% and 15% [2]. Many studies have shown that these bots engage in malicious activities, such as spreading misinformation and manipulating public opinion. For example, during the COVID-19 pandemic, social bots were used to propagate false information [3] and promote political conspiracy theories [4]. Similarly, bots played an important role in political events like the 2017 French presidential election [5].
These factors highlight why bot detection is an essential challenge. Research in this area has been ongoing for over a decade, with an increasing number of publications addressing social bots each year [6]. However, advancements in bot detection have led bot developers to design countermeasures. Current bots have more sophisticated features, making detection increasingly difficult and necessitating new detection techniques. Today's bots often mimic human behaviour, making it difficult to identify them singularly. As a result, detection strategies have shifted toward group-based approaches, leveraging the coordinated actions of bots, known as botnets [7, 8]. Botnets do not necessarily imply social network connections among the accounts. Instead, they represent coordinated activity managed by a single entity and targeting shared objectives.
Recent advancements have explored the use of graph neural networks (GNNs) to analyze user interaction networks for bot detection. While these graph-based methods perform well, they tend to disregard the rich content of tweets. This thesis introduces a novel graph-based approach to Twitter bot detection that uses contextual tweet embeddings rather than the user interaction network. This method creates a graph representation for each user, formulating the problem as graph classification. The hypothesis is that common subgraph patterns may appear among bots. Tweets are encoded using a large language model to generate contextual tweet representations, which are then clustered. The clustering information is employed to construct user graphs. These graphs are subsequently classified using GNNs.

[1]After Elon Musk acquired Twitter[1], the platform was rebranded as X. However, for clarity and consistency, it will be referred to by its original name, Twitter, throughout this work.

Contributions. The contributions of this work are the following:

- The development of a novel bot detection method based on user graph representations, leveraging large language models to compute contextual tweet embeddings and graph neural networks for graph classification.
- A comprehensive evaluation of the method across five benchmark datasets, including a comparison with various bot detection techniques. The proposed approach achieves performance on par with state-of-the-art methods on some datasets and demonstrates a significant improvement on the Cresci18 dataset [9].
- An analysis of the method's limitations on newer datasets, highlighting the potential importance of user relations for effective bot detection. The results also emphasize the necessity of incorporating tweet context to ensure robust detection capabilities.

Although the proposed method does not perform optimally on more recent datasets, it establishes a solid foundation for further improvements. Future enhancements could focus on integrating user-following relationships, which have yet to be considered in the proposed approach. Specifically, incorporating such relationships could significantly enhance the method's ability to detect bots by capturing more subtle and complex user interaction patterns.

Chapter 2

Related Works

The foundation of this thesis involved studying various bot detection approaches, which have evolved significantly over time. This section reviews critical methodologies that provided a basis for this study, categorized by their underlying techniques and innovations.

The first type of bot detection relies on extracting features from user metadata or tweets. Lee et al. [10] and LOBO [11] utilized metadata features such as follower and following counts, the number of links, and hashtags per tweet. These were classified using Random Forests. Similarly, Knauth et al. [12] employed metadata features but used an AdaBoost classifier instead. Yang et al. [13] extended these approaches by computing dynamic features, such as the growth rates of followers and favorites or analyzing username characteristics. Bot-Hunter [14] incorporated content-based features, such as the alignment of language between users and their tweets or the presence of the word "bot" in account descriptions, and applied Random Forests for classification.

Beskow and Carley [15] proposed an unconventional approach, relying uniquely on usernames for bot detection. Using TF-IDF techniques, features like the number of uppercase and lowercase letters were extracted and classified using SVM, logistic regression, and Naive Bayes. Abreu et al. [16] demonstrated the efficacy of a minimal selection of features, using just five account features and comparing various machine learning algorithms. Hayawi et al. [17] introduced DeeProBot, a more complex model integrating numerical and binary features with GloVe embeddings of user descriptions, processed by LSTM neural networks. The embeddings were then concatenated with the other features and a couple of dense neural network layers were used for classification.

Another class of methods utilizes natural language processing (NLP) on tweets and user descriptions. BGSRD [18] combines BERT with graph convolutional networks. It constructs a heterogeneous graph where words and user descriptions serve as nodes, with BERT embedding outputs as node features, processed by Graph Convolutional Networks (GCNs) for prediction.

Methods that combine user metadata with tweet analysis through NLP techniques have also shown promise. Efthimion et al. [19] merged user features such as username length, the friends-to-followers ratio, and presence of profile picture with tweet similarity measures through the Levenshtein distance. The classification was done

using logistic regression and SVMs. Similar studies [20, 2, 21] integrated user and tweet features, including sentiment analysis and tweet content similarity, often using models like SVMs and Random Forests.

Another direction in bot detection focuses on analyzing collective behaviours rather than individual account characteristics. Cresci et al. [22] proposed the Digital DNA technique, where each account is assigned a sequence of characters, similar to biological DNA, based on the tweets it produces. By identifying the longest common substrings, groups of accounts with significant similarities are classified as bots. Subsequent advancements expanded on this concept. For example, Chawla et al. [23] utilized BERT to analyze tweet sentiments (positive, neutral, or negative), constructing DNA sequences from the sentiment labels of tweets. One more variation by Di Paolo et al. [24] transformed these Digital DNA sequences into images, leveraging Convolutional Neural Networks (CNNs) for classification.

Recently, thanks to the advancements in geometric deep learning, graph-based approaches have emerged as powerful tools for bot detection. For instance, BotRGCN [25] builds graph representations using features such as user descriptions, tweets, and account-specific data. These features are concatenated to serve as node attributes, while the edges represent the following relationships. The resulting graphs are processed using Relational Graph Convolutional Networks (R-GCNs) for classification. Another approach [26] combines social graphs with contrastive learning and GNNs to classify accounts. The study by Lyu et al. [27] integrates a novel feature. It observes that social bots prefer sporadic activity bursts, unlike genuine users. These bursts denote sudden and intense activity after prolonged intervals of inactivity. A dual-channel GNN is employed to incorporate this feature in the model.

Twitter bot detection research is continually evolving. As bots become increasingly sophisticated, new, innovative techniques are needed to keep up with these advancements. The techniques highlighted above demonstrate diverse strategies, from simple feature extraction to advanced graph-based models, addressing various challenges in bot detection. This thesis builds upon these methodologies, introducing a novel approach that combines the strengths of graph neural networks with contextual tweet embeddings.

Chapter 3

Background

3.1 Large Language Models and Sentence Embeddings

The proposed approach in this thesis leverages Sentence-BERT (SBERT), a pre-trained large language model (LLM) capable of generating vector representations of sentences, to compute embeddings for tweets. These embeddings can later be used for efficient semantic similarity calculations, which are crucial for various tasks such as text clustering. The following section provides an overview of how this model works from the ground up.

3.1.1 Transformer Architecture

SBERT is based on the Transformer architecture, which revolutionized the world of Natural Language Processing (NLP) thanks to its ability to retain contextual information. They were first introduced in the well-renowned paper "Attention is all you need" ([28]), and nowadays, they are applied in various NLP tasks like text generation, language translation, and question answering. Fundamentally, Transformers perform next-word prediction: for instance, given an input such as "I was driving a" the output is likely to be "car", as the model predicts the most probable word to follow the input. Transformers can write text that makes sense with respect to the input because they can keep track of the context of what is being given as input. This capability is possible because these models are trained with a massive amount of data, so what it gives as output is just the best word according to all the text that they were trained on.

The architecture of a transformer (fig. 3.1) includes the following components:

- **Tokenization**: the input text is broken down into smaller units, or tokens;
- **Embedding**: each token is mapped to a dense vector representation;
- **Positional Encoding**: information about the relative or absolute position of the input sequence's tokens is added to the embeddings;
- **Transformer blocks**: each composed of a self-attention layer and a feed-forward neural network;
- **Softmax layer**: the final layer that generates the model's output.

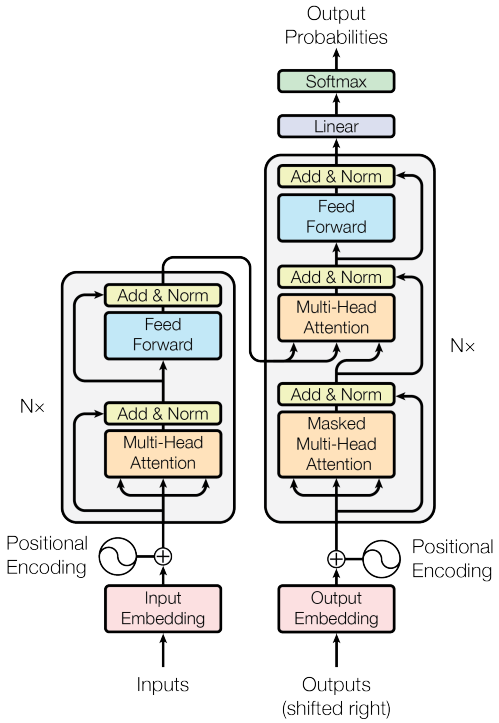


Figure 3.1. The transformer architecture - Source: [28]

Tokenization

Tokenization is the first step, and it involves segmenting text into smaller elements called tokens. Depending on the chosen tokenization strategy, tokens may include individual words, punctuation marks, or subwords. This process transforms raw, unstructured text, which is otherwise challenging to analyze, into a structured format. Tokenization facilitates the conversion of textual data into a numerical representation that machine learning models can efficiently process. It serves various NLP tasks, such as text classification, sentiment analysis or next-word prediction.

Figure 3.2. Source: [29]

Embedding

The embedding process transforms tokens into numerical representations by mapping each token to a vector of numbers. The key idea is that words (or tokens) with similar meaning or within the same semantic field will be represented by vectors that are close to each other in the vector space. Conversely, tokens with different meanings will be represented by vectors positioned further apart. Thus, the proximity in the

vector space reflects semantic similarity or dissimilarity between tokens.

Positional Encoding

After the embedding step, each token is represented by a vector. However, to represent entire sequences of tokens, such as sentences or paragraphs, a method must meaningfully combine these individual token embeddings into a single representation. A simple approach could involve summing the token embeddings componentwise. However, this method fails to account for the positional order of words within the sequence, which is crucial for capturing meaning. For instance, sentences like "I am not sad, I am happy" and "I am not happy, I am sad" would have identical representations through simple addition despite having opposite meanings. Positional encoding addresses this issue by incorporating information about the position of each token within a sequence into the embeddings. This technique involves adding a sequence of predefined vectors, corresponding to each token's position, to the embedding vectors. These positional encodings ensure that tokens in different positions are represented differently, even if they have the same lexical meaning. As a result, the positional encoding differentiates sentences with the same tokens in varying orders, ensuring unique representations that reflect the sentence's meaning.

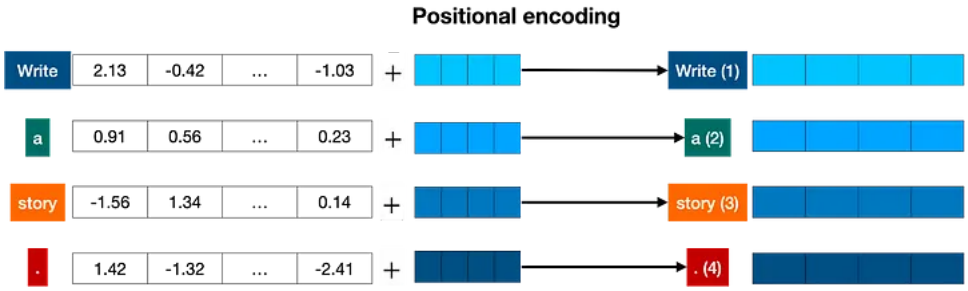


Figure 3.3. A vector representing the position of the token in the sentence is added to the token embedding. Source: [29]

Transformer Blocks

Once tokens have been embedded and their positional information encoded, they are ready to be processed by a machine learning model. A common approach involves training a very large neural network to predict the next word in a sentence. To enhance the performance of these neural networks, an attention mechanism is introduced. This is the core of transformer models and it's one of the reasons they work so well. Attention functions as a way to add contextual information to each word from surrounding text, thereby enhancing the model's understanding of language structures.. Transformer models typically consist of multiple blocks, each composed of two main components: an attention layer and a feed-forward neural network layer. The attention layer allows the model to weigh the importance of different input tokens, while the feed-forward layer generates the final output.

Attention

In natural language, the same word can assume different meanings based on context, challenging language models. Traditional word embeddings map words to vector representations without accounting for contextual differences, leading to ambiguities in meaning. The attention mechanism enables language models to address this issue by helping the model capture context-dependent meanings of words, allowing it to differentiate between meanings based on surrounding words.
For example, in the sentences "An Apple computer" and "I ate an apple" the word "apple" has distinct meanings—referring to a tech company in the first sentence and a fruit in the second. Ideally, a language model should produce different embeddings for "apple" in each context. Attention achieves this by examining the neighbouring words. In the first sentence, "apple" will be associated more closely with the word "computer," emphasizing its technological connotation. Meanwhile, in the second sentence, "apple" will be associated more closely with the word "ate," highlighting its culinary meaning. Through attention, words within a sentence are positioned closer in the embedding space to relevant context words, making their vector representations more meaningful. For instance, the embedding of "apple" in "An Apple computer" will be shifted closer to "computer," while in "I ate an apple," it will be closer to "ate." This contextual alignment enables embeddings to incorporate information from neighbouring words, improving their specificity and relevance.
Transformers use a more advanced form of attention known as multi-head attention. In this mechanism, multiple distinct attention layers, or "heads," modify embeddings in parallel so that the model varies perspectives on context. This multi-head structure enables Transformers to capture more complex relationships in text, significantly improving language understanding and representation.

Softmax Layer

The output of the Transformer blocks provides scores, representing the likelihood of a particular word being the following word in the sequence for all possible words. The highest scores are given to words most likely to follow the input sentence. These scores are processed through a softmax layer, which converts them into a probability distribution, where the words with the highest scores are assigned the highest probabilities, representing the model's prediction of the next word.
The next word in the sequence can be sampled from this distribution, and the input sentence can be updated to include this predicted word. By iteratively feeding the model its output and repeating the sampling process, generating text of any desired length is possible, with each new word informed by the preceding text.

3.1.2 Sentence-BERT

Sentence-BERT [30], or SBERT, is a framework built on the BERT model to compute sentence embeddings in a way that is both computationally efficient and adaptable for various downstream tasks, such as similarity comparisons, semantic search, clustering, and sentence classification.

BERT

BERT [31], which stands for Bidirectional Encoder Representations from Transformers, is a language representation model based on a pre-trained transformer encoder stack. Initially, BERT undergoes pre-training to gain a broad understanding of language, followed by fine-tuning to adapt to specific tasks. The pre-training phase includes two primary tasks:

1. Masked Language Modeling (MLM): Random tokens within the input sequence are masked, and the objective is to predict the original tokens based only on their surrounding context. For instance, given a sentence such as "The [MASK] brown fox jumped over the [MASK] dog," the model attempts to predict the masked words using context from both directions correctly. This bidirectional training enables a fusion of the left and right contexts.

2. Next Sentence Prediction (NSP): The model receives pairs of sentences, and its task is to predict whether the second sentence follows the first. This binary classification task requires the model to distinguish between coherent and unrelated sentence pairs.

In BERT, each input token is represented by an embedding vector formed by summing its token embedding, segment embedding, and positional encoding. The segment embedding assigns each token an identifier that reflects which segment (or sentence) it belongs to. These learnable segment and positional embeddings enable the model to understand the token order and sentence boundaries, allowing it to process relationships between sentences effectively.

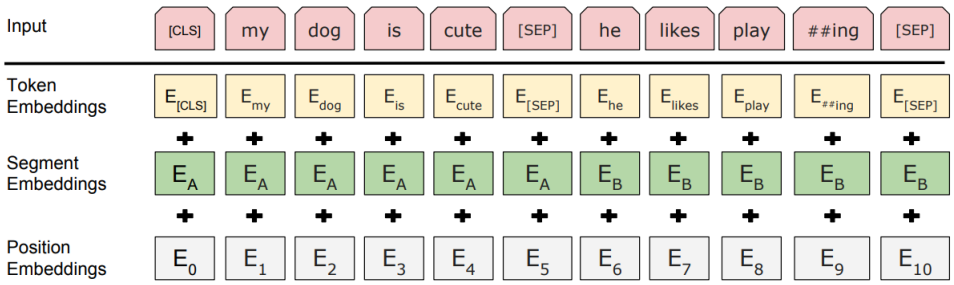
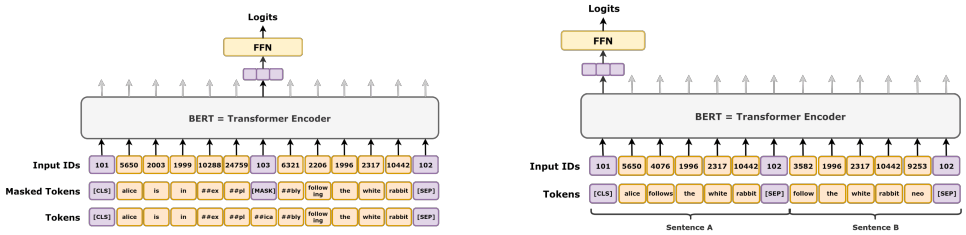


Figure 3.4. Bert input represtation. The input is made by the sum of token embeddings, segment embeddings and position embeddings. Source: [31]

For the Masked Language Modeling (MLM) task, the model's output consists of sentence vectors with masked tokens replaced by predictable tokens. Each predicted token undergoes a softmax operation over all words in the vocabulary, and cross-entropy loss is computed against the actual tokens for the masked positions. In NSP, the output is based on the representation of the [CLS] token, which is used for binary classification to indicate whether the second sentence follows the first one. During fine-tuning, BERT adapts to specific tasks by adding task-related inputs and outputs. For sentence or sentence-pair classification tasks, the probability of the

[CLS] token is used as the output. In question-answering, BERT generates logits for each token, showing its likelihood of being the start or end of an answer. For token classification, each token outputs a probability distribution across classes. Fine-tuning adds task-specific layers and adjusts all model parameters, making it faster and less costly than pre-training, which enables BERT to handle a wide range of language tasks effectively.



(a) MLM **(b)** NSP

Figure 3.5. In Masked Language Modeling, BERT predicts the original tokens, which are masked in the input. In Next Sentence Prediction, given two input sentences, BERT hether the second sentence follows the firs. Source: [32]

Siamese Network

Sentence-BERT uses siamese networks, a type of architecture with two or more identical subnetworks that generate and compare feature vectors for each input. This architecture is commonly used in face recognition and anomaly detection applications. In a typical setup, the model is provided with three items: an anchor sample, a positive sample (similar to the anchor), and a negative sample (unrelated to the anchor). The objective is to train the model to estimate similarity by minimizing the distance between similar items and increasing the distance between dissimilar ones. This is achieved through a loss function known as Triplet Loss. The Triplet Loss function is defined as follows:

$$L(A, P, N) = max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + margin, 0)$$

where A is the anchor input, P is the positive input from the same class as A, N is a negative input from a different class, and f represents the embedding function. The idea is that the distance between the anchor and negative sample embeddings should be at least a specified "margin" greater than the distance between the anchor and positive samples, effectively guiding the model in distinguishing similar and dissimilar pairs.

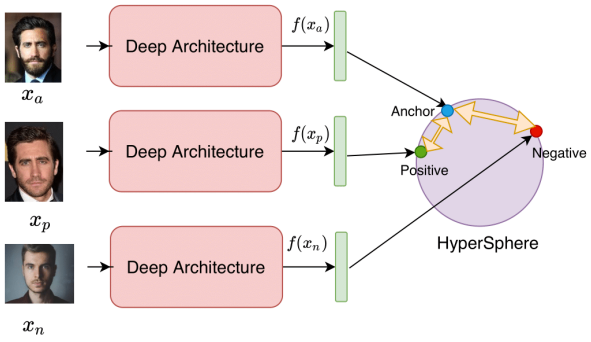
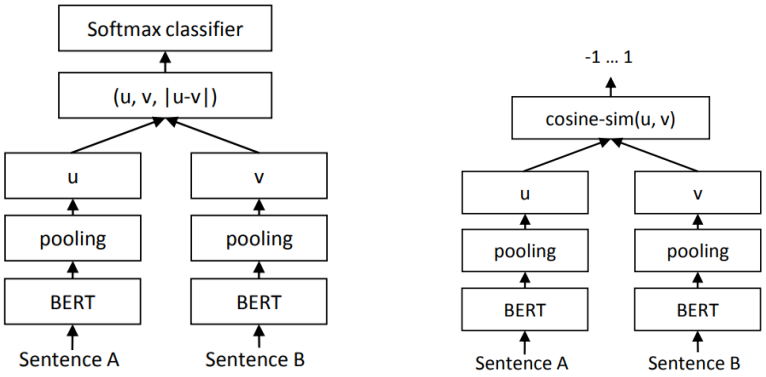


Figure 3.6. An example of a Siamese Network with images. Source: [33]

SBERT

SBERT is particularly effective for semantic text similarity, semantic search, and natural language inference tasks. While BERT can handle these tasks, its structure works best as a token embedder, making it computationally costly for sentence similarity calculations. Using BERT to compute the similarity between two sentences, each sentence pair must be processed through the transformer model, resulting in high computational overhead. An alternative approach of averaging token embeddings for similarity scoring has been attempted but has generally yielded suboptimal results. SBERT addresses this limitation by integrating BERT into a Siamese network architecture and fine-tuning it on sentence pairs. This setup has two identical BERT models running in parallel with shared weights to compute sentence embeddings and significantly reduce the computational cost directly. SBERT also includes a pooling layer, typically applying a mean pooling strategy, to create a fixed-size sentence embedding from BERT's output.



(a) Classification **(b)** Similarity

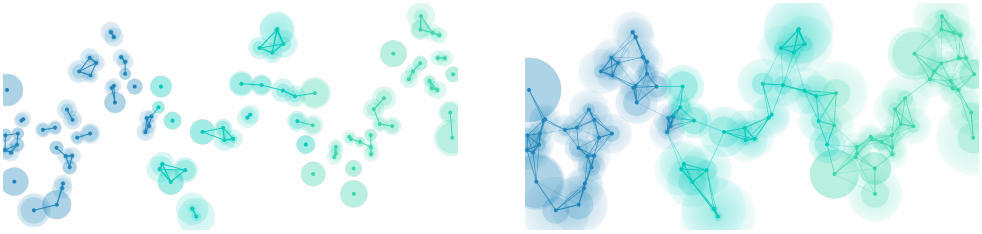
Figure 3.7. For both the classification and similarity task, two siamese BERT networks are used. Then, in the first case a Softmax classifier produces the output, while in the second one the embeddings are compared with cosine similarity. Source: [34]

During training, SBERT fine-tunes BERT, covering a range of written text genres and around 1 million annotated sentence pairs. The resulting model has achieved superior performance on benchmark tasks compared to previous approaches. SBERT is optimized for generating high-quality sentence embeddings that apply to various downstream tasks, such as calculating similarity scores via cosine similarity or performing more complex sentence-level tasks. This combination of efficiency and versatility makes SBERT highly suitable for applications requiring fast and accurate sentence comparisons.

3.2 Dimensionality Reduction

3.2.1 Uniform Manifold Approximation and Projection

When dealing with high dimensional data, dimensionality reduction is essential to face the curse of dimensionality. The most widely used algorithm for this purpose is Principal Component Analysis (PCA), a linear method that, while effective, can lose significant information when reducing from very high to very low dimensions. This limitation is addressed by manifold learning, which is a non-linear type of dimensionality reduction that can preserve more of the data's intrinsic structure. Uniform Manifold Approximation and Projection (UMAP), developed by McInnes et al. [35], is a fast and scalable manifold learning algorithm that performs well even with large datasets and high dimensionalities. UMAP is particularly effective at preserving the data's global and local structures.



(a) 30% Extent **(b)** 60% Extent

Figure 3.8. A radius extends outwards from each data point, computed by the distance to its n-th nearest neighbour. After the intersection with the first neighbor, the radius begins to get fuzzy. Source: [36]

The algorithm uses graph layout techniques to arrange data points in a low-dimensional space. This means that it starts by constructing a high-dimensional graph representation of the data, which it then optimizes to produce a structurally similar low-dimensional graph. The initial graph construction in UMAP relies on a "fuzzy simplicial complex", i.e. a weighted graph in which edge weights represent the likelihood of a connection between points. To determine these weights, UMAP extends a radius from each data point, connecting points when their radii overlap. If the radius is too small, it will form small isolated clusters; if too large, everything will be connected. Thus, choosing the right radius is crucial. UMAP addresses this

challenge by choosing a radius locally based on each point's n-th nearest neighbour distance. The graph is then made "fuzzy" by reducing connection likelihood as the radius grows, ensuring that each point connects with at least its closest neighbour to preserve a balance between local and global structure. Once this high-dimensional graph is constructed, UMAP builds a low-dimensional analogue that maintains structural similarity to the original.

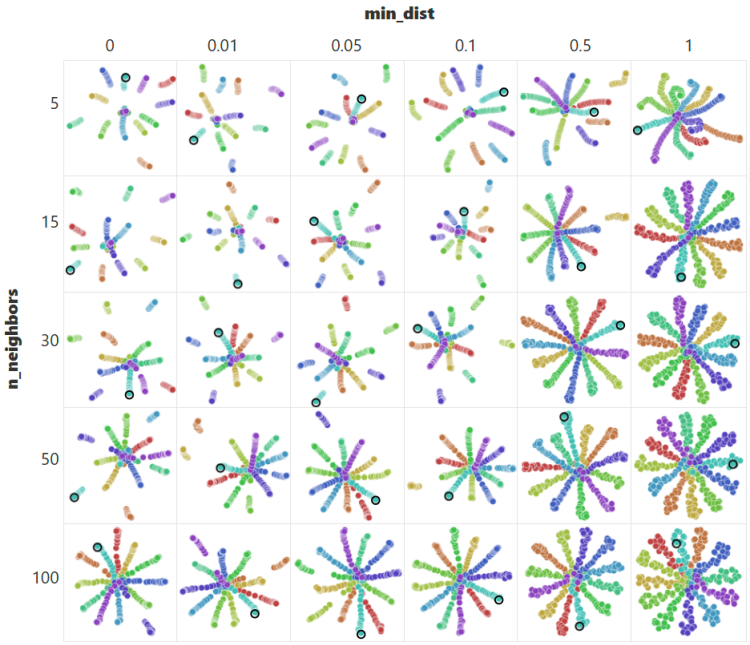


Figure 3.9. A projection of a toy dataset using different UMAP parameters combinations. Source: [36]

UMAP Algorithm Parameters

The primary parameters governing UMAP are *n_neighbors* and *min_dist*. The first one sets the number of approximate nearest neighbours used to construct the initial high-dimensional graph and controls the balance between local and global structure. Lower values emphasize local structure by reducing the number of neighbouring points considered when analyzing the data in high dimensions, while higher values emphasize the global structure. The *min_dist* parameter specifies the minimum distance between points in the low-dimensional space, controlling the compactness of the points. Lower values produce more tightly packed embeddings, while higher values allow a looser configuration.

3.2.2 PCA vs UMAP

Principal Component Analysis (PCA) is a widely used dimensionality reduction method that operates linearly. It is simpler than manifold learning methods like UMAP, which employ non-linear techniques to uncover deeper structures within the data. PCA reduces the dimensionality by finding the directions of maximum variance—principal components—within the data, as these capture the most significant

information. Mathematically, PCA achieves this by computing the eigenvalues and eigenvectors of the data's covariance matrix. Eigenvectors indicate the directions of maximum variance, and eigenvalues quantify the amount of variance along these directions. This process allows PCA to retain only the most important components, discarding those contributing less to data variability.
The strengths of PCA lie in its simplicity and computational efficiency, which makes it ideal for large datasets. As a linear method, PCA works best when relationships within the data are primarily linear, though it may oversimplify data with complex, non-linear patterns, potentially losing important information. PCA aims to maximize data variance, while manifold learning methods, such as UMAP, prioritize preserving relationships between data points.
Linear methods like PCA identify the main directions in data by fitting straight lines, whereas manifold learning captures non-linear patterns, revealing hidden structures or curves that PCA might overlook. PCA has a clear advantage in interpretability as the principal components generated can be readily interpreted and related to the original features. In contrast, manifold learning makes interpreting what each component represents challenging, as they do not directly correlate with the original feature space.
Scalability further differentiates these methods. PCA is highly efficient, scaling well to large datasets and processing thousands of features with minimal computational requirements. Instead, manifold learning can be computationally intensive, especially with high-dimensional data. These distinctions often inform the choice between methods. For linearly separable data, PCA is ideal. However, manifold learning techniques are often the better choice for complex data with non-linear relationships, as they tend to preserve these intricate patterns more effectively.

3.3 Clustering

3.3.1 Hierarchical Density-Based Spatial Clustering of Applications with Noise

HDBSCAN [37] is a hierarchical clustering algorithm that expands DBSCAN by converting it into a hierarchical structure and then extracting a flat clustering based on cluster stability. The algorithm operates in several steps:

- Density-Based Space Transformation: The algorithm first transforms the space based on data density or sparsity to identify dense regions in a dataset.
- Minimum Spanning Tree Construction: A minimum spanning tree (MST) is created from a distance-weighted graph, where each point is connected to others with the shortest possible total distance, preserving the essential structure.
- Cluster Hierarchy Construction: A hierarchical structure of clusters is built from the connected components of this MST.
- Hierarchy Condensation: This hierarchy is condensed based on user-defined minimum cluster size, simplifying the tree and filtering out smaller clusters.

- Stable Cluster Extraction: Stable clusters are then selected from this condensed tree, forming a flat clustering output.

The algorithm aims to identify dense regions within data points, which are often noisy and contain outliers. The HDBSCAN algorithm's core is single-linkage clustering, which is sensitive to noise, so a single noisy point could falsely merge two distinct clusters. To address this, HDBSCAN estimates density, interpreting lower-density areas as noise. These density estimations are usually made using the distance to the k-th nearest neighbour, called core distance, denoted as $core_k(x)$ for a point x. HDBSCAN then defines a new metric, the mutual reachability distance, defined as:

$$d_{reach} - k(a,b) = max\{core_k(a), core_k(b), d(a,b)\}$$

where $d(a,b)$ is the original metric distance between a and b. This metric adjusts distances based on density, pushing low-density (sparser) points further from each other to be at least their core distance away from any other point while high-density points remain close. The mutual reachability distance depends on the choice of k; larger k values interpret more points as having lower density.

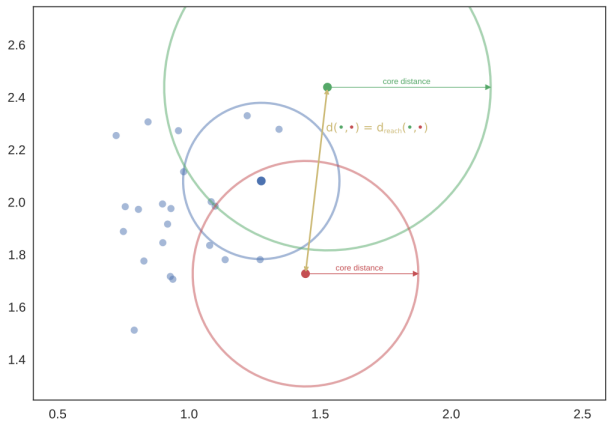


Figure 3.10. The mutual reachablity distance from red to green is simply the distance between them, since that distance is greater than either core distance. Source: [38]

The process continues by considering dense regions relative to varying density thresholds. A weighted graph is formed with data points as vertices and edges weighted by the mutual reachability distance between two points. Starting with a high threshold and iteratively lowering it, edges above the threshold value are removed, gradually fragmenting the graph into connected components. In the end, there will be a hierarchy of connected components at varying threshold levels. To streamline this process, HDBSCAN finds a minimum spanning tree (MST) using Prim's algorithm, selecting edges in increasing order of weights to form clusters at each connection.

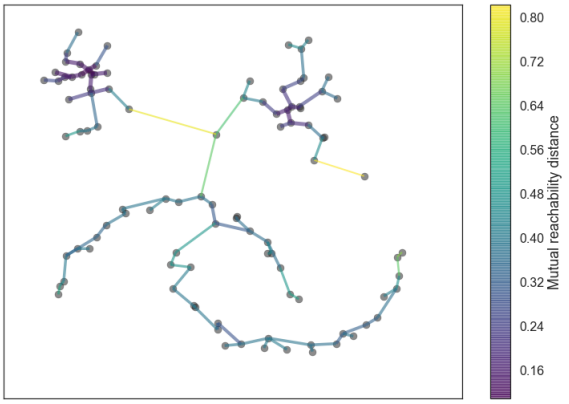
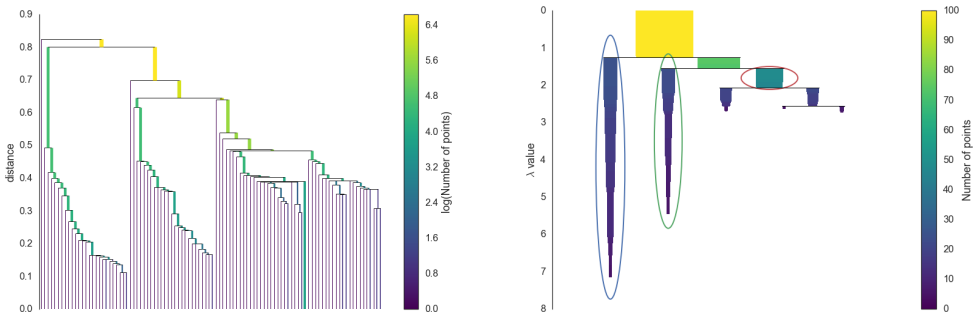


Figure 3.11. An example of the minimum spanning tree for mutual reachability distance computed by HDBSCAN. Source: [38]

Sorting the MST's edges by increasing distance allows constructing a hierarchy of connected components. As edges are added in this order, new clusters are formed by merging points or groups of points connected by these edges. This process incrementally reveals clusters, with each newly added edge connecting points or clusters progressively closer to each other in the graph.
Now, a single density threshold across all clusters would be imposed if a simple horizontal cut was made through the hierarchy. However, HDBSCAN aims to extract clusters that reflect varying densities, so the hierarchy is condensed into a more simplified tree structure that filters out smaller, less stable clusters. This condensed tree allows for the extraction of only the most significant clusters, which better represent the underlying structure of the data across varying densities.
The condensation step in HDBSCAN reduces the complex cluster hierarchy into a more manageable tree structure. Some clusters may break up when a few points detach from the main cluster body, forming tiny groups. However, suppose these tiny groups are automatically treated as new clusters. In that case, the results become overly fragmented, meaning that there will be too many small, unimportant clusters instead of a few meaningful ones. To avoid this, HDBSCAN introduces a minimum cluster size parameter, ensuring that small splits do not automatically form distinct clusters. This parameter allows clusters with fewer points than the minimum size to merge back into their parent cluster, preserving the parent's cluster identity and marking where individual points diverge from it over distance. Only splits resulting in two clusters, both meeting or exceeding the minimum size, are considered true cluster divisions, remaining in the hierarchy.
Once these criteria condense the hierarchy, it produces a reduced tree, where each node contains information on how the cluster size at that node decreases over varying distances. This reduced tree aids in flat cluster extraction, as clusters spanning a larger distance interval in the dendrogram indicate the greatest stability. For stable cluster selection, HDBSCAN also calculates cluster persistence using a metric λ, where λ is defined as the inverse of the distance ($\lambda = \frac{1}{distance}$). Each cluster is assigned a λ value at its formation (λ_{birth}) and dissolution (λ_{death}). Moreover, each point in the cluster is assigned a specific λ_p value indicating the distance at which it left the cluster, which will be a value between λ_{birth} and λ_{death} since the point

either falls out of the cluster during the cluster's lifetime or leaves the cluster when the cluster splits into two smaller clusters. This process enables the identification of clusters with longer lifespans, favouring stable and significant clusters for final selection.



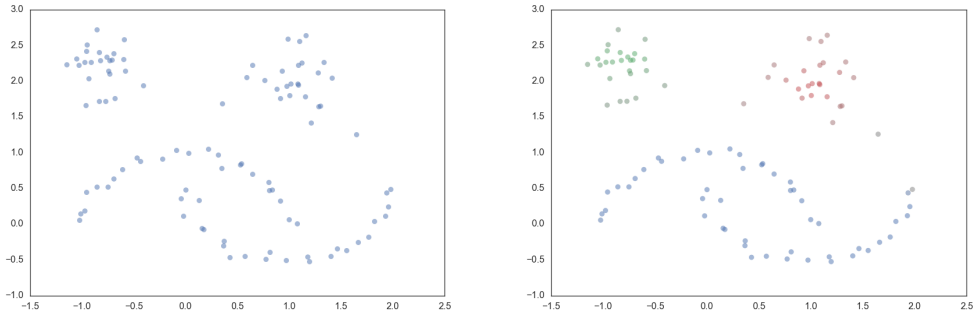
(a) **(b)**

Figure 3.12. On the left the conversion into the hierarchy of connected components of the MST visualized as a dendrogram. On the right the condensed cluster tree. Source: [38]

To avoid redundant cluster selection and ensure a meaningful flat clustering of the data, HDBSCAN enforces a hierarchical approach. In this process, the stability of each cluster is calculated using the formula:

$$\sum_{p \in cluter}(\lambda_p - \lambda_{birth})$$

All leaf nodes are initially selected as clusters. At higher levels of the hierarchy, the stability of each cluster is compared to the combined stability of its child clusters. If the sum of the child clusters' stabilities exceeds the parent cluster's stability, its stability is updated to reflect the sum of its children's stabilities. Conversely, if a cluster's stability is greater than the sum of its descendants, it is selected as a distinct cluster, and its descendants are excluded. This hierarchical selection process continues till the root node, resulting in a flat data clustering. Points not included in the selected clusters are designated as noise.



(a) Initial data **(b)** Clustered data

Figure 3.13. An example of clustering computed by HDBSCAN. Source: [38]

3.4 Graph Neural Networks

Graphs provide an effective structure for representing objects and their connections, making them valuable for modelling complex relationships. Graph Neural Networks (GNNs) have been developed to harness this structure and are applied across various domains, including antibacterial discovery, fake news detection, recommendation systems, and bot detection. The primary objective of a GNN is to predict specific features or properties within a graph, which can occur at three levels: graph classification, node classification, and edge prediction.
In graph classification, the goal is to predict a property of the entire graph. For instance, predictions could relate to characteristics like toxicity or scent in a molecule represented as a graph. Node classification aims to identify the role or category of individual nodes within the graph. Edge prediction focuses on forecasting relationships between nodes, determining which nodes are likely to share an edge or predicting the value of that edge.

Graphs Encoding

Machine learning models usually process inputs in matrix form, so an effective encoding of graph structures is required to utilize them in such models. Graphs generally contain up to four features relevant to predictions: nodes, edges, global context, and connectivity. Node features are commonly represented in a node feature matrix, N, where each node i is assigned an index, and the feature set for node_i is stored in N. For connectivity representation, an adjacency matrix is often impractical. Graphs with millions of nodes and variable numbers of edges per node lead to highly sparse adjacency matrices, which are inefficient in terms of memory. Moreover, multiple adjacency matrices can represent identical connectivity structures, and there is no guarantee that different matrices representing the same data would produce the same result in a deep neural network.
A more memory-efficient approach uses adjacency lists, which describe connectivity by listing edges (fig. 3.14). Specifically, an edge e_k connecting nodes n_i and n_j is recorded as a tuple (i, j) in the k-th entry of the adjacency list. Given that the number of edges is typically much lower than n^2_{nodes}, adjacency lists conserve memory and computation by focusing on the connected parts of the graph. For tensor representations in practice, graph attributes are often represented with vectors. Instead of a node tensor of size $[n_{nodes}]$, a node tensor of size $[n_{nodes}, node_{dim}]$ is employed, with similar structures applied for other graph attributes.

Graph Neural Network

With graph data encoded in adjacency lists, which is a permutation-invariant format, graph neural networks (GNNs) can be utilized for prediction tasks. A GNN performs an optimizable transformation on all graph attributes—nodes, edges, and global context—while preserving graph symmetries. GNNs operate within the "message passing neural network" framework, structured according to the Graph Nets architecture. They follow a "graph-in, graph-out" model, meaning they accept a graph as input, with data embedded into nodes, edges, and the global context. They progressively transform these embeddings while retaining the graph's connectivity.

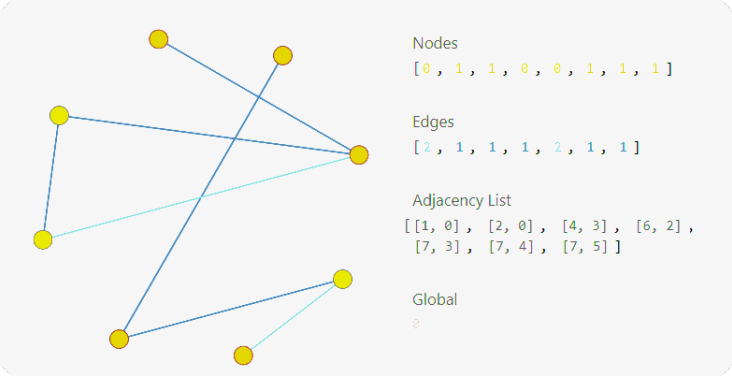


Figure 3.14. An example of a graph and its representation as an adjancecy list. The first two arrays contain the features of nodes and edges respectively. Source: [39]

The simplest GNN architecture learns new embeddings for all graph attributes (nodes, edges, global context) without leveraging graph connectivity. In this approach, a multilayer perceptron (MLP), called the GNN layer, is applied independently to each graph component. Each node's feature vector is processed with the MLP to produce a learned node vector. Similarly, the MLP computes an independent embedding for each edge and the global context. These GNN layers can be stacked together, preserving the original adjacency list and the same feature vector structure in the output graph. However, the GNN updates embeddings for each node, edge, and global context, transforming the graph attributes.
To make predictions, consider a binary classification task as an example. A linear classifier can be applied to each node embedding if binary predictions are required on nodes and node features are already available. However, more complex cases may arise. For instance, if the graph only includes edge-level information but requires node-level predictions, transferring information from edges to nodes becomes necessary. This transfer of information is achieved through a pooling mechanism, which collects information from edges and gives it to the nodes. This process has two steps:

1. Embeddings are gathered for each item to be pooled and concatenated into a matrix.

2. The embeddings are aggregated, typically through a summation.

The pooling operation is denoted by ρ, where the gathering of information from edges to nodes is represented as $p_{E_n} \rightarrow V_n$. Thus, pooling enables information to be routed to nodes for prediction in a scenario where only edge features are present. Conversely, node information can be gathered and aggregated similarly if node-level features are only available, but a binary global prediction is required. The same approach applies to edges. Pooling is a foundational operation for more advanced GNN models, allowing for flexible information transfer between graph attributes.

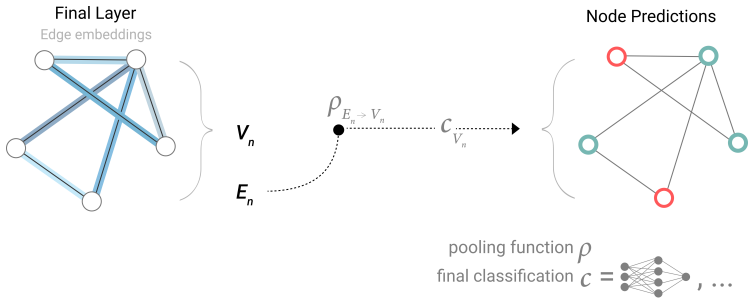


Figure 3.15. A visualization of binary node classification, having only edge-level features and using pooling to route information. Source: [39]

In this basic GNN model, graph connectivity is not utilized within the GNN layer itself. Nodes, edges, and the global context are processed independently. Graph connectivity is only considered during pooling when aggregating information for prediction, enabling the construction of models that preserve the structural integrity of the graph data.

Message Passing

Pooling within the GNN layer enables learned embeddings to incorporate graph connectivity through message passing, where neighbouring nodes or edges exchange information to influence each other's updated embeddings. Message passing consists of three key steps:

1. For each node in the graph, gather all embeddings (or "messages") from its neighbouring nodes.
2. Aggregate these messages using an aggregation function, such as summation.
3. Pass the pooled messages through an update function, typically a learned neural network.

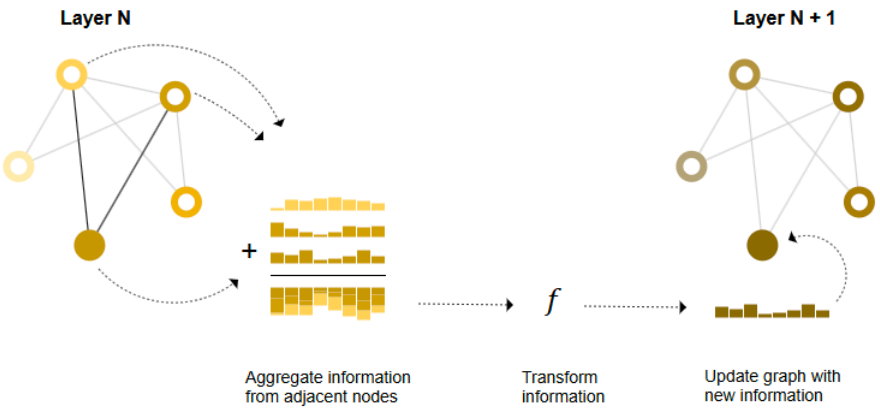


Figure 3.16. Message passing. Source: [39]

Message passing can be performed between nodes or edges, allowing GNNs to leverage graph connectivity effectively. These steps form the foundation for constructing

increasingly sophisticated message-passing mechanisms within GNN layers, enhancing model expressiveness and capability. By stacking multiple message-passing layers, a node can progressively incorporate information from a more significant portion, and ultimately the entirety, of the graph.

Edges Representation

When predictions are needed on nodes but the dataset contains only edge-level information, pooling can route information from edges to nodes in the final prediction stage of the model. However, message passing allows nodes and edges to exchange information within each GNN layer. This is achieved by pooling the edge information, transforming it with an update function, and then storing the transformed information. Since node and edge features may differ in size or shape, integrating them as they are can be challenging. One approach is to learn a linear mapping to project edge features into the node feature space and vice versa or to concatenate both before applying the update function. Decisions regarding which graph attributes to update first and in which sequence forms part of the architectural design of GNNs, such as deciding whether to update node embeddings before edge embeddings or in reverse order.

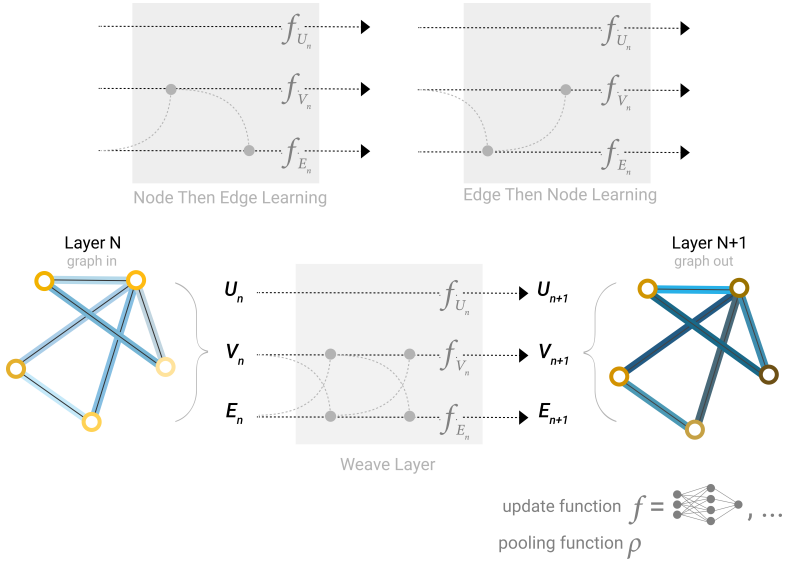


Figure 3.17. Some examples of different methods to combine edge and node representations in a GNN layer. Source: [39]

A limitation of traditional GNN architectures is that distant nodes may struggle to exchange information efficiently, even with multiple message-passing layers. For a node within a k-layer network, information is typically propagated at most k steps away. This can pose challenges when prediction tasks depend on nodes or groups of nodes located far apart within the graph. A potential solution is to enable all nodes to communicate, but this approach becomes computationally prohibitive in large graphs. Introducing a global representation, often called the "master node" or "context vector," can mitigate this issue. This global context vector is connected to

all nodes and edges within the network. It serves as a bridge, facilitating information flow across the graph to create a richer, more comprehensive representation of the graph.
All graph attributes—nodes, edges, and global context—are represented by learned embeddings, which can be combined during pooling by conditioning the target attribute's information with respect to other graph attributes. For instance, the new embedding for a node can incorporate information from neighbouring nodes, connected edges, and the global context vector. This conditioning can be achieved by concatenating the respective embeddings, mapping them to a shared space via a linear map, or applying a feature-wise modulation layer, which functions similarly to a feature-wise attention mechanism.

Chapter 4

Datasets

This study employs multiple datasets to evaluate the proposed approach, each containing both genuine human accounts and bot accounts and a selection of tweets and certain profile characteristics for each user. While these datasets share a common foundation, they vary in the specific features they provide for each account.
The datasets used in this work were selected for their diversity and representation of various real-world bots. Instead of focusing on the specific bot detection methods applied in the original research papers associated with these datasets, the emphasis here is placed on their structure. The subsequent sections provide an overview of each dataset's unique features and structure.

4.1 Cresci15

The Cresci15 dataset was developed to detect fake followers, which are accounts specifically created to increase the follower count of a target account artificially [40]. Such accounts are problematic as they can distort perceived popularity and influence, affecting sectors such as the economy, politics, and society. The dataset comprises several sub-datasets, each collected through a distinct approach to capture human and bot accounts.
The first component, the Fake Project, was an initiative by researchers from the IIT-CNR in Pisa, Italy. This experiment involved creating a Twitter account with the description: "Follow me only if you are NOT a fake". Promoted by other researchers and journalists, this account amassed 574 followers over a short period. Using Twitter's API, researchers gathered information on these followers. Subsequently, a CAPTCHA verification was sent to each account, and those who successfully responded were labelled as certified human accounts. These verified accounts were added to the human subset of the dataset.
The second component, the #elezioni2013 Dataset, was compiled as part of a research project examining political trends in Italy between 2013 and 2015. Researchers collected accounts that used the #elezioni2013 hashtag, then categorized these accounts by analyzing usernames, biographies, and specific keywords like "blogger," "journalist," "social media strategist/analyst," and "congressperson." Political party names were also searched to filter out politically involved accounts, which were excluded to focus on citizen accounts. The resulting dataset contained approximately

40,000 accounts, which were then sampled to retain around 1,500 accounts. These were manually verified to ensure they represented genuine human users, removing non-human accounts or accounts with insufficient profile data. Together, these two subsets form the human portion of the Cresci15 dataset.
For the bot subset, researchers acquired bot accounts through online marketplaces offering fake Twitter followers. Initially, 3,000 bot accounts were purchased; however, a portion of these accounts had to be excluded from the dataset due to Twitter's rapid suspension. The remaining accounts are the malicious bot accounts within the Cresci15 dataset.

Table 4.1. List of user features provided by the Cresci15 dataset.

Features
followers_count, friends_count, listed_count, favourites_count, statuses_count, profile_use_background_image, protected, verified

4.2 Cresci17

The Cresci17 dataset is structured into various subgroups representing genuine human accounts and different types of spambots.
The Genuine Accounts subgroup was created by contacting random Twitter users with a straightforward question. The responses were manually verified, and accounts that replied were classified as genuine human accounts. Accounts that did not respond were excluded to ensure that only verified human profiles were included in this set. This subset offers a baseline of authentic user behaviour on Twitter, capturing normal patterns of activity against which bot behaviour can be contrasted.
The Social Spambot #1 subgroup was created by collecting social bot accounts associated with a runner-up candidate from the 2014 mayoral elections in Rome. These bots were employed to amplify the candidate's reach, sharing messages that promoted the electoral campaign and helped extend its visibility across Twitter. This subgroup highlights a strategic, politically motivated use of social bots to influence public opinion and increase engagement during an election.
The Social Spambot #2 subgroup comprises spambots that engaged in the promotion of the #TALENTS hashtag. This hashtag was used by a mobile application company seeking to connect with and hire artists across various fields, including photography, writing, and music. The persistent promotion of this hashtag by social bots provides insights into the marketing-oriented use of bots to boost engagement around a specific branding initiative.
Social Spambot #3 represents a set of social bots that spammed links to discounted Amazon products. These accounts focused on product promotion by continuously tweeting URLs leading to sales pages on Amazon, demonstrating another type of bot behaviour oriented toward affiliate marketing or sales.
The Traditional Spambot #1 subgroup consists of accounts distributing URLs that direct users to malicious content. This dataset was originally used as a training set in a separate study [41] on spam detection and includes bots that aim to compromise the privacy and security of Twitter users by leading them to phishing sites or malicious downloads.

Additionally, the Cresci17 dataset contains subgroups that include only profile-level features without associated tweets. These subgroups were excluded from this study, as the focus here relies on tweets to examine user behaviour.

Table 4.2. List of user features provided by the Cresci17 dataset.

Features
followers_count, friends_count, listed_count, favourites_count, statuses_count, geo_enabled, verified, is_translator, profile_use_background_image, default_profile, default_profile_image, protected, notifications

4.3 Cresci18

The Cresci18 dataset was developed to study Twitter microblogs and their influence on the prediction of stock prices and traded volumes within financial markets [9]. Twitter has become a significant hub for microblogs related to economics and financial markets, with many users and automated bots posting content related to stock performance, financial trends, and market analysis. The dataset includes a collection of stock -related microblogs from Twitter, as well as supplementary financial information sourced from Google Finance. However, for this thesis work, only the Twitter-based data component is of interest.
A distinguishing feature of stock-related microblogs on Twitter is the use of cashtags, which are specific symbols that represent individual companies. A cashtag typically consists of a dollar sign followed by a company's stock ticker (for example, $AAPL for Apple Inc.). Cashtags are analogous to hashtags in that they enable efficient filtering and organization of content on Twitter, allowing users to locate relevant posts quickly. Given this functionality, cashtags were used as a primary filtering criterion for data collection in this dataset.
The initial step involved gathering a comprehensive list of stocks actively traded on prominent U.S. stock exchanges, such as NASDAQ and the New York Stock Exchange (NYSE). This list, obtained from the official NASDAQ website, provided the set of stock tickers to guide tweet collection. Using Twitter's APIs, all tweets published between May and September 2017 that included at least one cashtag from this list were extracted.

Table 4.3. List of user features provided by the Cresci18 dataset.

Features
followers_count, friends_count, listed_count, favourites_count, statuses_count, verified

4.4 TwiBot-20

The TwiBot-20 dataset was developed to address several limitations found in earlier bot detection datasets, including issues related to user diversity, limited user information, and data scarcity [42]. Unlike previous datasets that tend to focus on a narrow subset of users, often limiting their scope to a specific category or type of

account, TwiBot-20 aims to capture a broader spectrum of bot activity on Twitter. The dataset's construction utilized a breadth-first search (BFS) strategy, starting from a set of root nodes and expanding based on users' follow relationships. The BFS method enables community-based bot detection approaches, as it inherently reflects the interconnectedness of users within Twitter's social network.
The TwiBot-20 dataset includes accounts from four major interest domains: politics, business, entertainment, and sports. By covering a wide range of topics, the dataset is designed to capture users' activity across various areas of interest. For each user included in the dataset, the most recent 200 tweets are retrieved. Each user account in the dataset is manually annotated as either a bot or a genuine user.

Table 4.4. List of user features provided by the TwiBot-20 dataset.

Features
followers_count, friends_count, listed_count, favourites_count, statuses_count, geo_enabled, verified, contributors_enabled, is_translator, is_translation_enabled, profile_use_background_image, has_extended_profile, default_profile, default_profile_image

4.5 TwiBot-22

TwiBot-22 is a large-scale, graph-based dataset for bot detection on Twitter [43]. The dataset was constructed using a breadth-first search (BFS) method, further augmented by two distinct sampling strategies. The first one is based on distribution diversity, which utilizes user metadata to build a distribution that reflects the diversity of user types. This distribution is then used to sample users to ensure a broad representation of different user categories. The second sampling strategy relies on value diversity and focuses on ensuring the diversity of metadata values across the sampled users. This strategy prioritizes the inclusion of users in the dataset whose metadata values differ significantly from those of their neighbours, thereby capturing a more varied and representative network of users. Once the user network is established, breadth-first search is employed to expand through the network. During each neighbourhood expansion, one of the two sampling strategies—distribution diversity or value diversity—is randomly selected. This process continues until the desired number of users is collected.
Given the size of the TwiBot-22 dataset, manual annotation of all users would not be feasible. Therefore, a three-step annotation process was employed:

- Expert Annotation: A random sample of 1,000 users from the dataset was provided to five bot detection experts, who labelled the accounts as either bots or genuine users. The labels from this expert annotation process were split into two sets, with 80% used for training and 20% reserved for testing purposes in the subsequent steps.

- Generate Noisy Labels: Several bot detection models were trained on the expert-labeled training set. These models were then used to predict labels for the remaining users in the TwiBot-22 dataset. The uncertainty of the model

predictions was measured, and those predictions with high uncertainty were removed to reduce label noise and improve the reliability of the dataset.

- Majority Voting: Finally, the labels generated by the bot detection models were assessed for plausibility using Snorkel, a weak supervision framework. Snorkel outputs probabilistic labels and uses them to train a multilayer perceptron (MLP) classifier to generate the final annotations for the entire dataset.

Table 4.5. List of user features provided by the TwiBot-22 paper.

Features
followers_count, following_count, tweet_count, listed_count, verified

Chapter 5

Methodology

Figure 5.1 shows the entire pipeline of this method. The whole process can be summarized in the following steps:

- **Embeddings**: Tweets contain many elements that do not contribute to their semantical meaning, e.g. links, mentions, and retweet symbols. These elements are removed to capture the core content of the tweets accurately. SBERT [34], a large language model, is utilized to compute high-dimensional embeddings that effectively capture the rich context of the tweets.
- **Dimensionality reduction**: The high-dimensional embeddings are reduced to 15 dimensions using UMAP [35], a manifold learning technique. UMAP captures non-linear patterns within the data while mapping the embeddings into a lower-dimensional space, preserving as much information as possible. This step enhances computational efficiency while retaining essential features.
- **Clustering**: The reduced embeddings are clustered using HDBSCAN [37]. This algorithm identifies semantically similar groups of tweets while labelling outliers as noise points. The idea is that a user's behaviour can be characterized based on the interactions between their tweets and these semantic groups.
- **Graph representations**: A heterogeneous graph representation is constructed for each user to model user behaviour systematically. These graphs capture the relationships between the tweets of different semantic clusters. Each graph consists of tweet nodes representing individual tweets and a user node. Tweet nodes are enriched with the previously computed reduced embeddings as features, while user nodes incorporate account metadata (e.g., follower count, verified status). Edges between tweet nodes are determined based on cosine similarity and cluster membership. At the same time, the user node is connected to every tweet node.
- **Classification**: With each user represented as a graph, a GNN is employed for classification. The GNN aims to identify patterns in the graph structure that indicate specific behaviours and infer user characteristics. Specifically, the GNN architecture integrates GATv2 [44] and SAGEConv [45] frameworks. The final linear layer outputs the predicted label for the user graph, indicating whether the user is a bot or a human.

The subsequent sections provide a detailed explanation of each step.

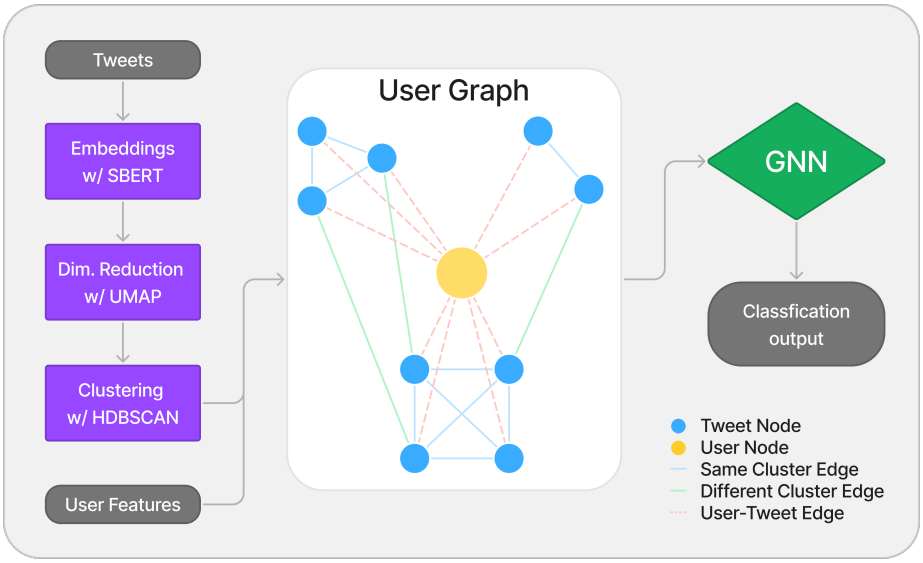


Figure 5.1. Overview of the proposed approach.

5.1 Data Preprocessing

This work aims to capture the semantic meaning of tweets using a language model. The primary objective is to detect patterns in content, as bot accounts typically share tweets centred around specific themes. For instance, a spam bot may predominantly post promotional messages for products or services. To accurately determine tweet content, removing extraneous information that does not contribute to a tweet's semantic meaning is essential. Elements such as user tags, external links, and retweet symbols (e.g., "RT") often add noise rather than clarity and thus are systematically removed in the preprocessing phase. This initial step is crucial in standardizing the text input for effective analysis.

During preprocessing, links, mentions to other users (e.g., "@username"), and retweet markers are removed from each tweet to eliminate distractions from the core content. Hashtags can sometimes carry relevant information, so they are modified by simply stripping away the "#" symbol, preserving the words themselves (e.g., "#sunny" becomes "sunny"). Handling emojis presents additional considerations, as they may or may not enhance the quality of the model. Experiments were conducted with both the inclusion and exclusion of emojis. However, the observed performance variation due to emoji inclusion or exclusion is relatively small, indicating that the impact may be dataset-specific rather than universally significant. After this stage, all empty tweets resulting from the filtering process were removed to maintain a dataset of meaningful entries.

The datasets used in this analysis vary in the number of tweets available per user,

mainly due to changing Twitter API policies, affecting data retrieval. Only a user's 200 most recent tweets were retained, providing a uniform sampling approach to ensure consistency. Additionally, users with fewer than ten tweets were excluded, resulting in a dataset where each user has between 10 and 200 tweets. This approach mitigates discrepancies in tweet availability per user, enhancing subsequent analyses' reliability.

Table 5.1. Number of accounts used per dataset. # Users indicate the total number of accounts used for the given dataset.

	Cresci15	**Cresci17**	**Cresci18**	**TwiBot-20**	**TwiBot-22**
# Bots	2,769	8,240	10,818	6,151	17,166
# Humans	1,909	1,082	1,202	4,952	20,126
# Users	4,678	9,322	12,020	11,103	37,292

Table 5.1 shows the number of accounts used per dataset. TwiBot-22 originally included 1 million users, but only a sample was considered due to hardware limitations. The other datasets have been used entirely. The counts of users do not match the original datasets because users with less than ten tweets were discarded. Every provided user feature was used as a node feature in the graph representations. Specifically the features used are indicate in tables 4.1, 4.2, 4.3, 4.4 and 4.5. For TwiBot-22, additionally the profile's description and the username were used as additional features, as the dataset provided just five of them.

5.2 Tweets clustering

After text preprocessing, the next step is to map each tweet to a label representing its semantic essence. For this purpose, Sentence-BERT (SBERT) is employed to generate embeddings for the tweets, specifically using the multilingual model *paraphrase-multilingual-mpnet-base-v2*[1] from the *sentence-transformers* Python library. This model, which outputs embeddings in 768-dimensional space, was chosen to accommodate the multilingual nature of the datasets, as tweets are written in multiple languages beyond English. By embedding each tweet into this high-dimensional space, SBERT enables more nuanced analysis, providing a foundational representation for clustering.
Direct clustering of embeddings in high-dimensional spaces is challenging due to the curse of dimensionality, which can lead to inefficiencies and unreliable results. To address this, UMAP is applied to reduce the embedding space from 768 to 15 dimensions. This reduction facilitates effective clustering while preserving the essential semantic relationships between tweets. UMAP was chosen for its ability to retain global and local structure in the data, and cosine similarity was selected as the distance metric for this process. The choice of the distance metric was informed by the documented compatibility of cosine similarity with the SBERT embeddings, which have been optimized for this similarity measure.

[1] https://huggingface.co/sentence-transformers/paraphrase-multilingual-mpnet-base-v2

Once the dimensionality has been reduced, HDBSCAN clusters the tweets. HDBSCAN identifies clusters of tweets based on density, assigning each tweet either to a specific cluster or marking it as noise. We cannot assume that clusters represent topics, as the exact nature of each cluster is not known.
The specific parameters chosen for HDBSCAN influence the clustering outcome, affecting the number of clusters formed and the proportion of tweets categorized as noise. After clustering, each tweet is assigned either a cluster or noise label.

5.2.1 Clustering observations

The clustering of tweets across datasets provided significant insights into the structure and distribution of tweet data. These observations, derived from the HDBSCAN clustering process, show the complexity of categorizing tweet data into well-defined groups. Key findings include a high proportion of noise, diversity in cluster sizes, and imbalances in bot and human representation within clusters.

Noise and Cluster Characteristics

A primary observation was the substantial proportion of tweets categorized as noise across all datasets, regardless of the clustering parameters. Adjusting the minimum cluster size had a notable impact: decreasing it reduced noise while increasing it led to higher noise levels. Larger minimum cluster sizes cause smaller clusters to be classified as noise. A lower minimum cluster size was selected to minimize the proportion of noise points. Ultimately, the noise accounted for approximately 50% of data points across all datasets.
The number of clusters identified across the datasets ranged between 1,500 and 2,500, with the minimum cluster size and minimum sample parameters both set to 15. Testing indicated that variations in the number of clusters did not significantly affect the pipeline's overall performance.

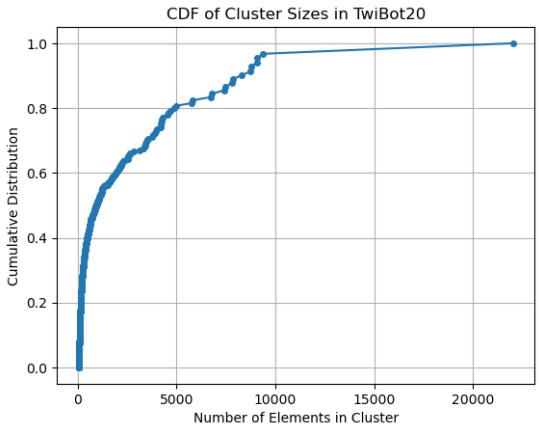


Figure 5.2

Cluster Sizes and Distribution

The clustering process did not produce any vast clusters. Most clusters were relatively small and narrowly focused. A closer analysis of the TwiBot-20 dataset revealed an imbalance in cluster size distribution, which can be observed across all datasets. Excluding the noise cluster, the majority of clusters (around 2,400) fell below the 80th percentile of the cumulative distribution function (CDF), while only a tiny fraction (around 15) represented the most significant clusters (fig. 5.2). This disparity indicates a concentration of tweets in numerous small clusters, with relatively few larger clusters containing a significant proportion of data.

Imbalance in Bot and Human Tweet Distribution

A notable observation relates to the distribution of bot and human tweets within clusters, especially in the Cresci datasets. Significant imbalances were observed, with some clusters dominated by bot tweets and others primarily consisting of human tweets. This imbalance reflects behavioural differences between bots and humans. Notably, higher levels of imbalance, as observed in the Cresci datasets, corresponded to better model performance. In contrast, more balanced clusters, such as those in the TwiBot datasets (especially TwiBot-22), were associated with lower performance. Figures 5.3 and 5.4 show the plots for the number of bots and humans in the ten biggest clusters.

Similarity Within Clusters

Cosine similarity, used in the graph representations, was analyzed within the top clusters. These clusters exhibited high average cosine similarity coupled with low standard deviation. This behaviour suggests that the clustering algorithm effectively grouped related points into the same clusters.

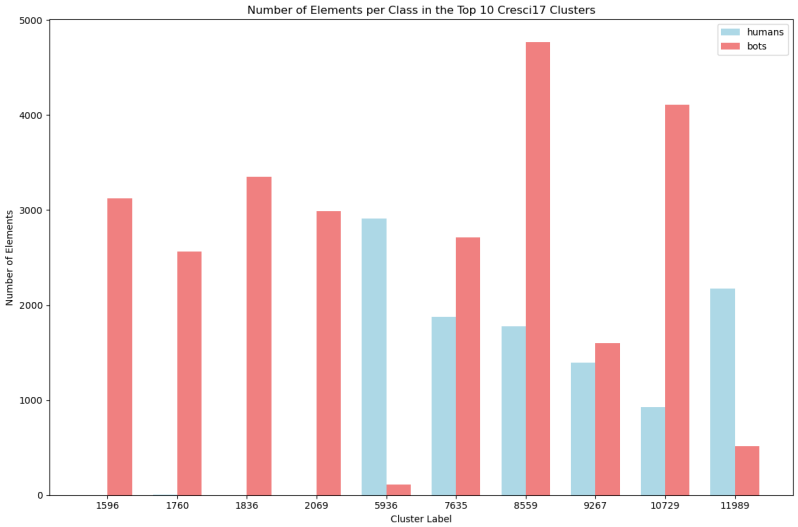


Figure 5.3. Hunan vs bot accounts distribution in the top 10 biggest clusters of Cresci17.

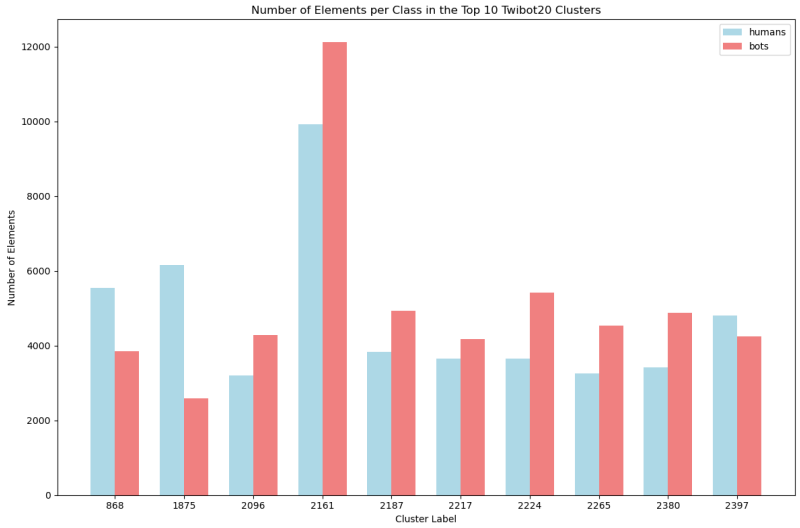


Figure 5.4. Hunan vs bot accounts distribution in the top 10 biggest clusters of Twibot-22.

5.3 Graph Representaions and GNNs

In this stage, a unique graph is constructed for each user, which is then processed in a GNN for graph classification. Two approaches were explored when constructing these user graphs. In the first approach, graphs consisted solely of tweet nodes, each representing an individual tweet, with edges capturing relationships between tweets based on clustering and similarity metrics. Here, tweet nodes hold 15-dimensional embeddings derived by reducing the original tweet embeddings with UMAP. However, testing this structure revealed limitations in capturing a complete view of user behaviour, as account metadata was not included.
A second approach was developed to address these limitations, incorporating tweet nodes and a user node representing account-related information. This heterogeneous graph structure allowed for a richer representation by including user-specific features. In this design, tweet nodes retained the 15-dimensional UMAP-reduced embeddings, while the user node held a range of account attributes. These attributes included count-based features (e.g., follower and following counts), normalized using z-score, and boolean features (e.g., account verification, profile picture presence) represented as binary values (0 or 1). By integrating the user node, this approach enriched the graph structure, embedding both tweet-level characteristics and account-level features. This allowed for a more comprehensive view of each user in the classification process.
Three distinct types of edges are employed to capture relationships within each user's graph, facilitating connectivity between tweet nodes and between tweet nodes and the user node. In the initial tweet-only graph structure, the first edge type connects all tweets within the same cluster identified by HDBSCAN, forming fully connected subregions. The second edge type links tweets from different clusters based on cosine similarity, with a threshold set at 0.6, and only tweet pairs that meet this threshold are connected. Cosine similarity is used as an edge feature for both edge types, enhancing connectivity within clusters and establishing similarity-based links across

clusters. Noise points found by HDBSCAN were considered as a normal cluster.
With the introduction of the user node with account-specific features, a third type of edge was included in the graph. This is an undirected edge linking the user node to each tweet node in the graph. While this third edge type does not include additional features, it is still useful to allow message passing among every region of the graph. This comprehensive graph representation—integrating tweet-level and account-level information—allows the GNN model to leverage tweet content and user metadata, providing a more informative foundation for classification.
A graph neural network (GNN) is employed for the classification task. The architecture is structured as follows:

- Initial Linear Layers: A linear layer is applied to the tweet embeddings, mapping each tweet node feature vector to a new dimensionality. In parallel, another linear layer is applied to the feature vector of the user node, mapping it to the same dimensionality as the tweet nodes to facilitate homogeneous processing in later stages.

- Graph Convolutional Layers: Two graph convolutional layers are utilized to propagate information within the graph. A GATv2Conv (Graph Attention Network v2) layer is used for edges between tweet nodes, which applies attention-based message passing, enabling each tweet node to focus on significant neighbouring nodes selectively. For edges between the user node and tweet nodes, SAGEConv is employed, which aggregates information from a node's local neighbourhood. The GNN adopts max pooling as the aggregation method, allowing each node to retain only the most significant features during message passing. Meanwhile, global mean pooling is used to concatenate node and edge embeddings.

- A final linear layer is employed for binary classification, with a single output node that categorizes the user as either a human or a bot based on the entire graph structure.

Chapter 6

Experiments and Results

PyTorch Geometric [1] is used for the experiments. The network structure is described in section 5.3. The weights of the network are initialized using a Xavier uniform distribution [46], while biases are initialized to zero. The loss function used is Binary Cross Entropy (BCE), defined as:

$$l(x, y) = L = l_1, ..., l_N, \quad l_n = -w_n[y_n \cdot logx_n + (1 - y_n) \cdot log(1 - x_n)]$$

The performance of the model was evaluated on five different datasets (described in chapter 4). Table 6.1 shows the results of the model on each dataset.

Table 6.1. Performance of the model on different datasets.

Dataset	Accuracy	Precision	Recall	F1-Score
Cresci15	0.987	0.991	0.987	0.989
Cresci17	0.989	0.993	0.993	0.993
Cresci18	0.956	0.959	0.993	0.975
Twibot-20	0.854	0.831	0.920	0.873
Twibot-22	0.686	0.677	0.616	0.645

The model demonstrated excellent performance on the Cresci15 and Cresci17 datasets, with an accuracy of 0.98. The Cresci18 dataset presented a slightly different scenario. The model achieved an accuracy of 0.95. As highlighted in section 5.2.1, the clustering of these datasets exhibit a higher imbalance between bot and human accounts. This imbalance is likely why the model performs better on these datasets. The model achieved a test accuracy of 0.85 for the TwiBot-20 dataset. As expected, the performance was lower compared to the Cresci datasets due to the more complex nature of the latter. The higher recall with Twibot-20 indicates that the model is effective at detecting bot accounts, although the slightly lower precision suggests some misclassification of human accounts as bots. The model's performance on TwiBot-22 was the weakest, with a test accuracy of 0.68. Here, the performance is lower, and the tweet clusters already showed less imbalance, indicating that bot and human tweets are more similar, making classification more difficult.

[1] https://pytorch-geometric.readthedocs.io/en/latest/

Table 6.2. Performance of the ablation study on TwiBot20.

Ablation Type	Accuracy	Precision	Recall	F1 Score
w/o User Node	0.6709	0.6753	0.7681	0.7187
w/ User Node	0.8541	0.8314	0.9202	0.8735

The TwiBot-20 dataset was also evaluated without including the user node and the results are shown in table 6.2. The test accuracy dropped by 18 points, highlighting the importance of the user node. This performance drop is likely because, in the TwiBot-20 dataset, all users with a verified account were annotated as humans. Therefore, incorporating this information into the graph representation it will be easier for the model to find which graphs do not represent a bot. For the Cresci17 dataset, removing the user node did not affect performance. This result suggests that the inclusion of the user node may be less critical for more straightforward datasets, where classification can already be effectively performed using tweet-level features, as the clusters in these datasets are already more imbalanced. When comparing the results from this study with those reported in the TwiBot-22 paper [43], which evaluated 35 bot detection models across multiple datasets, the performance of this approach was competitive.

Table 6.3 reports the results of the comparison. On the Cresci15 and Cresci17 dataset, the test accuracy achieved state-of-the-art performance. On Cresci18, this approach outperformed the best models, with a significant improvement in accuracy. This improvement is likely due to the specific nature of the dataset, where bots are focused on topics related to stocks and financial themes. The similarity in the topics of bot tweets may have resulted in similar embeddings in the vector space, which the clustering algorithm was able to capture effectively.

On the TwiBot-20 dataset, the accuracy achieved by this approach is almost in line with the best models reported in the TwiBot-22 paper. In contrast, the performance of the proposed model on TwiBot-22 highlighted the limitations of this approach, with a significant drop in accuracy to below 0.70. This performance drop can likely be attributed to the lack of following relationships between users. The absence of these relationships in the graph representation may have hurt the model's ability to capture user interactions, leading to lower performance. This could have been expected as Twibot-22 was proposed as a benchmark dataset for social graph-based bot detection approaches.

Tables 6.4, 6.5, and 6.6 present a comparison between the proposed technique and the results reported by Feng et al. [43]. Notably, the model achieved the highest F1-scores across all datasets except for TwiBot-20, where its performance was only marginally lower. This is a strong result because achieving the highest F1-scores across multiple datasets demonstrates the model's robustness and effectiveness in balancing precision and recall.

Table 6.3. Accuracy of the proposed method against the 35 models tested and reported in the TwiBot-22 paper[43]. **Bold** indicates the highest score and underline indicates the highest score reported in the TwiBot-22 paper[43] "/" indicates that the dataset is not supported and could not be evaluated by the method. "-" indicates that the baseline is not scalable to the size of the TwiBot-22 dataset.

Method	Cresci15	Cresci17	Cresci18	TwiBot-20	TwiBot-22
Abreu *et al.* [16]	0.757	0.927	0.754	0.734	0.707
Alhosseini *et al.* [47]	0.896	/	/	0.599	0.477
BGSRD [18]	0.878	0.759	0.507	0.664	0.719
BotHunter [14]	0.965	0.881	0.812	0.752	0.728
Botometer [48]	0.579	0.942	0.726	0.531	0.499
BotRGCN [25]	0.965	/	/	0.858	**0.797**
Cresci *et al.* [22]	0.370	0.335	/	0.478	-
Dehgan *et al.* [49]	0.621	/	/	0.867	-
Efthimion *et al.* [19]	0.925	0.880	0.708	0.628	0.741
EvolveBot [50]	0.922	/	/	0.658	0.711
FriendBot [51]	0.969	0.780	/	0.759	-
GCN [52]	0.964	/	/	0.775	0.784
GAT [53]	0.969	/	/	0.833	0.795
GraphHist [54]	0.774	/	/	0.513	-
Hayawy *et al.* [17]	0.843	0.908	0.500	0.731	0.765
HGT [55]	0.960	/	/	**0.869**	0.749
SimpleHGN [56]	0.967	/	/	0.867	0.767
Kantepe *et al.* [20]	0.975	0.982	/	0.803	0.764
Knauth *et al.* [12]	0.859	0.902	0.887	0.819	0.713
Kouvela *et al.* [57]	0.978	0.984	0.793	0.840	0.764
Kudugunta *et al.* [58]	0.753	0.883	0.775	0.596	0.659
Lee *et al.* [10]	0.982	0.988	0.815	0.774	0.763
LOBO [11]	0.984	0.966	/	0.774	0.757
Miller *et al.* [59]	0.755	0.771	0.525	0.645	0.304
Moghaddam *et al.* [60]	0.736	/	/	0.740	0.738
NameBot [15]	0.770	0.768	0.558	0.591	0.706
RGT [61]	0.972	/	/	0.866	0.765
RoBERTa [62]	0.970	0.972	/	0.755	0.721
Rodrifeuz-Ruiz *et al.* [63]	0.824	0.764	/	0.660	0.494
Santos *et al.* [21]	0.708	0.738	0.625	0.587	-
SATAR [64]	0.934	/	/	0.840	-
SGBot [13]	0.771	0.921	0.813	0.816	0.751
T5 [65]	0.923	0.964	/	0.735	0.721
Varol *et al.* [2]	0.932	/	/	0.787	0.739
Wei *et al.* [66]	0.961	0.893	/	0.713	0.702
Proposed Method	**0.987**	**0.989**	**0.956**	0.854	0.686

Table 6.4. Precision of the proposed method against the 35 models tested and reported by Feng et al. [43]. **Bold** indicates the highest score and underline indicates the highest score reported by Feng et al. [43]. "-" indicates that the score was missing.

Method	Cresci15	Cresci17	Cresci18	TwiBot-20	TwiBot-22
Abreu *et al.* [16]	0.990	0.983	0.754	0.722	0.509
Alhosseini *et al.* [47]	0.876	-	-	0.578	0.299
BGSRD [18]	0.865	0.758	0.527	0.676	0.225
BotHunter [14]	0.985	0.986	0.842	0.727	0.680
Botometer [48]	0.505	0.933	0.685	0.556	0.308
BotRGCN [25]	0.955	-	-	0.845	0.748
Cresci [22]	0.59	0.129	-	0.076	-
Dehgan [49]	0.961	-	-	0.947	-
Efthimion *et al.* [19]	0.938	0.945	0.827	0.642	0.777
EvolveBot [50]	0.850	-	-	0.669	0.563
FriendBot [51]	0.952	0.775	-	0.726	-
GCN [52]	0.955	-	-	0.752	0.711
GAT [53]	0.961	-	-	0.813	0.762
GraphHist [54]	0.731	-	-	0.512	-
Hayawi *et al.* [17]	0.929	0.954	0.507	0.716	**0.800**
HGT [55]	0.948	-	-	0.855	0.682
SimpleHGN [56]	0.956	-	-	0.847	0.725
Kantepe *et al.* [20]	0.813	0.830	-	0.634	0.786
Knauth *et al.* [12]	0.857	0.915	**0.998**	**0.965**	-
Kouvela *et al.* [57]	0.995	0.992	0.821	0.793	0.693
Kudugunta *et al.* [58]	**1.000**	0.985	0.548	0.804	0.443
Lee *et al.* [10]	0.986	**0.995**	0.847	0.766	0.672
LOBO [11]	0.984	0.993	-	0.748	0.754
Miller *et al.* [59]	0.720	0.772	0.547	0.607	0.294
Moghaddam *et al.* [60]	0.983	-	-	0.722	0.676
NameBot [15]	0.768	0.803	0.583	0.587	0.677
RGT [61]	0.963	-	-	0.851	0.750
RoBERTa [62]	0.975	0.924	-	0.738	0.632
Rodriguez-Ruiz *et al.* [63]	0.786	0.794	-	0.616	0.332
Santos*et al.* [21]	0.728	0.817	0.653	0.627	-
SATAR [64]	0.906	-	-	0.815	-
SGBot [13]	0.994	0.982	0.839	0.764	0.731
T5 [65]	0.910	0.944	-	0.721	0.632
Varol *et al.* [2]	0.922	-	-	0.780	0.757
Wei *et al.* [66]	0.917	0.859	-	0.610	0.627
Proposed Method	0.991	0.993	0.959	0.831	0.677

Table 6.5. Recall of the proposed method against the 35 models tested and reported by Feng et al. [43]. **Bold** indicates the highest score and underline indicates the highest score reported by Feng et al. [43]. "-" indicates that the score was missing.

Method	Cresci15	Cresci17	Cresci18	TwiBot-20	TwiBot-22
Abreu *et al.* [16]	0.621	0.919	0.756	0.828	0.117
Alhosseini *et al.* [47]	0.971	-	-	0.956	0.567
BGSRD [18]	0.955	**1.000**	0.704	0.731	0.199
BotHunter [14]	0.914	0.854	0.799	0.867	0.140
Botometer [48]	0.989	0.996	0.949	0.508	0.698
BotRGCN [25]	0.991	-	-	0.901	0.468
Cresci [22]	0.666	0.953	-	0.674	-
Dehgan [49]	0.838	-	-	0.821	-
Efthimion *et al.* [19]	0.943	0.892	0.580	0.706	0.167
EvolveBot [50]	0.958	-	-	0.728	0.080
FriendBot [51]	**1.000**	**1.000**	-	0.889	-
GCN [52]	0.988	-	-	0.876	0.448
GAT [53]	0.991	-	-	0.895	0.441
GraphHist [54]	**1.000**	-	-	**0.990**	-
Hayawi *et al.* [17]	0.793	0.921	0.711	0.835	0.149
HGT [55]	0.991	-	-	0.910	0.280
SimpleHGN [56]	0.992	-	-	0.920	0.329
Kantepe *et al.* [20]	0.753	0.761	-	0.610	0.468
Knauth *et al.* [12]	0.974	0.953	0.888	0.763	-
Kouvela *et al.* [57]	0.967	0.989	0.787	0.951	0.191
Kudugunta *et al.* [58]	0.609	0.858	0.475	0.334	0.619
Lee *et al.* [10]	0.984	0.991	0.803	0.836	0.196
LOBO [11]	0.990	0.961	-	0.878	0.259
Miller *et al.* [59]	**1.000**	0.991	0.588	0.974	**0.978**
Moghaddam *et al.* [60]	0.592	-	-	0.843	0.210
NameBot [15]	0.911	0.917	0.641	0.704	0.030
RGT [61]	0.992	-	-	0.910	0.301
RoBERTa it [62]	0.941	0.962	-	0.723	0.122
Rodriguez-Ruiz *et al.* [63]	0.991	0.928	-	0.987	0.813
Santos*et al.* [21]	0.858	0.844	0.649	0.581	-
SATAR [64]	0.998	-	-	0.912	-
SGBot [13]	0.636	0.908	0.810	0.949	0.243
T5 [65]	0.877	0.902	-	0.690	0.120
Varol *et al.* [2]	0.974	-	-	0.843	0.168
Wei *et al.* [66]	0.753	0.721	-	0.540	0.468
Proposed Method	0.987	0.993	**0.993**	0.920	0.645

Table 6.6. F1-score of the proposed method against the 35 models tested and reported by Feng et al. [43]. **Bold** indicates the highest score and underline indicates the highest score reported by Feng et al. [43]. "-" indicates that the score was missing.

Method	Cresci15	Cresci17	Cresci18	TwiBot-20	TwiBot-22
Abreu *et al.* [16]	0.763	0.950	0.769	0.771	0.534
Alhosseini *et al.* [47]	0.921	-	-	0.720	0.381
BGSRD [18]	0.908	0.862	0.581	0.700	0.211
BotHunter [14]	0.972	0.916	0.821	0.790	0.234
Botometer [48]	0.669	0.961	0.795	0.531	0.427
BotRGCN [25]	0.973	-	-	0.872	0.575
Cresci [22]	0.117	0.228	-	0.136	-
Dehgan [49]	0.883	-	-	0.762	-
Efthimion *et al.* [19]	0.941	0.918	0.682	0.672	0.275
EvolveBot [50]	0.900	-	-	0.697	0.140
FriendBot [51]	0.975	0.873	-	0.799	-
GCN [52]	0.971	-	-	0.808	0.549
GAT [53]	0.975	-	-	0.852	0.558
GraphHist [54]	0.844	-	-	0.675	-
Hayawi *et al.* [17]	0.855	0.937	0.607	0.770	0.247
HGT [55]	0.969	-	-	0.881	0.396
SimpleHGN [56]	0.972	-	-	**0.882**	0.454
Kantepe *et al.* [20]	0.781	0.794	-	0.622	0.587
Knauth **et al.** [12]	0.911	0.934	0.940	0.852	0.370
Kouvela *et al.* [57]	0.981	0.991	0.804	0.865	0.300
Kudugunta *et al.* [58]	0.757	0.917	0.509	0.472	0.516
Lee *et al.* [10]	0.985	**0.993**	0.824	0.799	0.304
LOBO [11]	0.987	0.976	-	0.808	0.385
Miller *et al.* [59]	0.837	0.868	0.567	0.748	0.452
Moghaddam *et al.* [60]	0.739	-	-	0.778	0.320
NameBot [15]	0.833	0.857	0.611	0.650	0.50
RGT [61]	0.977	-	-	0.880	0.429
RoBERTa [62]	0.958	0.943	-	0.730	0.205
Rodriguez-Ruiz *et al.* [63]	0.877	0.856	-	0.631	0.565
Santos*et al.* [21]	0.788	0.830	0.651	0.603	-
SATAR [64]	0.950	-	-	0.860	-
SGBot [13]	0.779	0.946	0.823	0.849	0.365
T5 [65]	0.893	0.923	-	0.705	0.202
Varol *et al.* [2]	0.947	-	-	0.810	0.275
Wei *et al.* [66]	0.826	0.784	-	0.573	0.536
Proposed Method	**0.989**	**0.993**	**0.975**	0.873	**0.645**

Chapter 7

Conclusion

This work presents a novel approach for bot detection on Twitter data by combining contextual tweet embeddings and graph-based user representations. More specifically, the methodology begins with preprocessing tweet data and removing hashtags, tags, and other noise characters to clean the content. Tweets are then encoded using SBERT embeddings, providing contextual representations in a high-dimensional space. To reduce the dimensionality of these embeddings without significant information loss, UMAP is employed. This allows to preserve the data structure while making it suitable for clustering. Following dimensionality reduction, HDBSCAN is used to identify clusters of tweets. The user graph-representation includes tweet nodes and a user features node. Finally, the graphs representing users were classified using a GNN.

The proposed approach demonstrated varying performance across the datasets. The model reached state-of-the-art performance on the Cresci datasets. On the TwiBot-20 dataset a good result was achieved, but not quite as good as other techniques. The TwiBot-22 dataset yielded the lowest performance, underscoring the limitations of this approach, particularly its lack of reliance on following relationships between users.

Interestingly, he model outperformed other state-of-the-art approaches on Cresci18, likely due to the alignment between bot tweets on financial topics and the semantic similarity captured by SBERT. Despite these strong results, this methodology has notable limitations. The absence of user interaction features in the graph representation, such as following relationships, significantly contributes to the performance gap on more complex datasets like TwiBot-22.

Future work should aim to incorporate these relationships to enhance the model's capability to capture user interactions, a critical aspect of bot behaviour. In conclusion, this work demonstrates the potential of combining tweet clustering and graph neural networks for bot detection on Twitter. The results indicate that further improvements can be achieved by refining the graph representation.

Bibliography

[1] K. Conger and L. Hirsch, "Elon Musk Completes $44 Billion Deal to Own Twitter." `https://www.nytimes.com/2022/10/27/technology/elon-musk-twitter-deal-complete.html`, October 27 2022. Accessed: 15-11-2024.

[2] O. Varol, E. Ferrara, C. Davis, F. Menczer, and A. Flammini, "Online human-bot interactions: Detection, estimation, and characterization," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 11, pp. 280–289, May 2017.

[3] K.-C. Yang, C. Torres-Lugo, and F. Menczer, "Prevalence of low-credibility information on twitter during the covid-19 outbreak," *arXiv preprint arXiv:2004.14484*, 2020.

[4] E. Ferrara, "What types of covid-19 conspiracies are populated by twitter bots?," *arXiv preprint arXiv:2004.09531*, 2020.

[5] E. Ferrara, "Disinformation and social bot operations in the run up to the 2017 french presidential election," *arXiv preprint arXiv:1707.00086*, 2017.

[6] S. Cresci, "A decade of social bot detection," *Commun. ACM*, vol. 63, p. 72–83, Sept. 2020.

[7] J. Zhang, R. Zhang, Y. Zhang, and G. Yan, "The rise of social botnets: Attacks and countermeasures," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, pp. 1068–1082, Nov. 2018. Publisher Copyright: © 2004-2012 IEEE.

[8] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race," in *Proceedings of the 26th international conference on world wide web companion*, pp. 963–972, 2017.

[9] S. Cresci, F. Lillo, D. Regoli, S. Tardelli, and M. Tesconi, "$fake: Evidence of spam and bot activity in stock microblogs on twitter," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 12, Jun. 2018.

[10] K. Lee, B. Eoff, and J. Caverlee, "Seven months with the devils: A long-term study of content polluters on twitter," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 5, pp. 185–192, Aug. 2021.

[11] J. Echeverrï£¡a, E. De Cristofaro, N. Kourtellis, I. Leontiadis, G. Stringhini, and S. Zhou, "Lobo: Evaluation of generalization deficiencies in twitter bot classifiers," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, (New York, NY, USA), p. 137–146, Association for Computing Machinery, 2018.

[12] J. Knauth, "Language-agnostic Twitter-bot detection," in *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)* (R. Mitkov and G. Angelova, eds.), (Varna, Bulgaria), pp. 550–558, INCOMA Ltd., Sept. 2019.

[13] K.-C. Yang, O. Varol, P.-M. Hui, and F. Menczer, "Scalable and generalizable social bot detection through data selection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 1096–1103, Apr. 2020.

[14] D. M. Beskow and K. M. Carley, "Bot-hunter: A tiered approach to detecting & characterizing automated activity on twitter," 2018.

[15] D. M. Beskow and K. M. Carley, "Its all in a name: detecting and labeling bots by their name," *Computational and mathematical organization theory*, vol. 25, pp. 24–35, 2019.

[16] J. V. Fonseca Abreu, C. Ghedini Ralha, and J. a. J. Costa Gondim, "Twitter bot detection with reduced feature set," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, p. 1–6, IEEE Press, 2020.

[17] K. Hayawi, S. Mathew, N. Venugopal, M. M. Masud, and P.-H. Ho, "Deeprobot: a hybrid deep neural network model for social bot detection based on user profile data," *Social Network Analysis and Mining*, vol. 12, p. 43, Mar 2022.

[18] Q. Guo, H. Xie, Y. Li, W. Ma, and C. Zhang, "Social bots detection via fusing bert and graph convolutional networks," *Symmetry*, vol. 14, no. 1, 2022.

[19] P. G. Efthimion, S. Payne, and N. Proferes, "Supervised machine learning bot detection techniques to identify social twitter bots," 2018.

[20] M. Kantepe and M. C. Ganiz, "Preprocessing framework for twitter bot detection," *2017 International Conference on Computer Science and Engineering (UBMK)*, pp. 630–634, 2017.

[21] E. Ferreira Dos Santos, D. Carvalho, L. Ruback, and J. Oliveira, "Uncovering social media bots: a transparency-focused approach," in *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, (New York, NY, USA), p. 545–552, Association for Computing Machinery, 2019.

[22] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "Dna-inspired online behavioral modeling and its application to spambot detection," *IEEE Intelligent Systems*, vol. 31, no. 5, pp. 58–64, 2016.

[23] V. Chawla and Y. Kapoor, "A hybrid framework for bot detection on twitter: Fusing digital dna with bert," *Multimedia Tools and Applications*, vol. 82, pp. 30831–30854, Aug 2023.

[24] E. Di Paolo, M. Petrocchi, and A. Spognardi, "From online behaviours to images: A novel approach to social bot detection," in *International Conference on Computational Science*, pp. 593–607, Springer, 2023.

[25] S. Feng, H. Wan, N. Wang, and M. Luo, "Botrgcn: Twitter bot detection with relational graph convolutional networks," in *Proceedings of the 2021 IEEE/ACM international conference on advances in social networks analysis and mining*, pp. 236–239, 2021.

[26] M. Zhou, D. Zhang, Y. Wang, Y.-A. Geng, and J. Tang, "Detecting social bot on the fly using contrastive learning," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, (New York, NY, USA), p. 4995–5001, Association for Computing Machinery, 2023.

[27] N. Lyu, B. Xu, F. Guo, and H. Shen, "Dcgnn: Dual-channel graph neural network for social bot detection," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, (New York, NY, USA), p. 4155–4159, Association for Computing Machinery, 2023.

[28] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[29] A. Ullah, "Transformer Architecture Explained." `https://medium.com/@amanatulla1606/transformer-architecture-explained-2c49e2257b4c`, 2023. Accessed: 15-11-2024.

[30] A. Sen, "SBERT: How to Use Sentence Embeddings to Solve Real-World Problems." `https://anirbansen2709.medium.com/sbert-how-to-use-sentence-embeddings-to-solve-real-world-problems-f950aa300c72`, 2023. Accessed: 15-11-2024.

[31] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[32] D. V. Godoy, "BERT Visualization." `https://dvgodoy.github.io/dl-visuals/BERT/`, 2024. Accessed: 15-11-2024.

[33] S. Chandhok, "Triplet loss with Keras and TensorFlow," in *PyImageSearch* (P. Chugh, A. R. Gosthipaty, S. Huot, K. Kidriavsteva, R. Raha, and A. Thanki, eds.), 2023.

[34] N. Reimers, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[35] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[36] A. Coenen and A. Pearce, "Understanding UMAP." `https://pair-code.github.io/understanding-umap/`, 2024. Accessed: 15-11-2024.

[37] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining* (J. Pei, V. S. Tseng, L. Cao, H. Motoda, and G. Xu, eds.), (Berlin, Heidelberg), pp. 160–172, Springer Berlin Heidelberg, 2013.

[38] "How HDBSCAN Works." `https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html`, 2024. Accessed: 15-11-2024.

[39] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, "A gentle introduction to graph neural networks," *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[40] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "Fame for sale: Efficient detection of fake twitter followers," *Decision Support Systems*, vol. 80, pp. 56–71, 2015.

[41] C. Yang, R. Harkreader, and G. Gu, "Empirical evaluation and new design for fighting evolving twitter spammers," *Trans. Info. For. Sec.*, vol. 8, p. 1280–1293, Aug. 2013.

[42] S. Feng, H. Wan, N. Wang, J. Li, and M. Luo, "Twibot-20: A comprehensive twitter bot detection benchmark," in *Proceedings of the 30th ACM international conference on information & knowledge management*, pp. 4485–4494, 2021.

[43] S. Feng, Z. Tan, H. Wan, N. Wang, Z. Chen, B. Zhang, Q. Zheng, W. Zhang, Z. Lei, S. Yang, *et al.*, "Twibot-22: Towards graph-based twitter bot detection," *Advances in Neural Information Processing Systems*, vol. 35, pp. 35254–35269, 2022.

[44] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?," *arXiv preprint arXiv:2105.14491*, 2021.

[45] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[46] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[47] S. Ali Alhosseini, R. Bin Tareaf, P. Najafi, and C. Meinel, "Detect me if you can: Spam bot detection using inductive representation learning," in *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, (New York, NY, USA), p. 148–153, Association for Computing Machinery, 2019.

[48] K.-C. Yang, E. Ferrara, and F. Menczer, "Botometer 101: Social bot practicum for computational social scientists," *Journal of computational social science*, vol. 5, no. 2, pp. 1511–1528, 2022.

[49] A. Dehghan, K. Siuta, A. Skorupka, A. Dubey, A. Betlen, D. Miller, W. Xu, B. Kamiński, and P. Prałat, "Detecting bots in social-networks using node and structural embeddings," *Journal of Big Data*, vol. 10, p. 119, Jul 2023.

[50] C. Yang, R. Harkreader, and G. Gu, "Empirical evaluation and new design for fighting evolving twitter spammers," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 8, pp. 1280–1293, 2013.

[51] D. M. Beskow and K. M. Carley, "You are known by your friends: Leveraging network metrics for bot detection in twitter," 2020.

[52] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[53] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[54] T. Magelinski, D. Beskow, and K. M. Carley, "Graph-hist: Graph classification from latent feature histograms with application to bot detection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5134–5141, Apr. 2020.

[55] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proceedings of the web conference 2020*, pp. 2704–2710, 2020.

[56] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang, "Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks," in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 1150–1160, 2021.

[57] M. Kouvela, I. Dimitriadis, and A. Vakali, "Bot-detective: An explainable twitter bot detection service with crowdsourcing functionalities," in *Proceedings of the 12th International Conference on Management of Digital EcoSystems*, MEDES '20, (New York, NY, USA), p. 55–63, Association for Computing Machinery, 2020.

[58] S. Kudugunta and E. Ferrara, "Deep neural networks for bot detection," *Information Sciences*, vol. 467, pp. 312–322, 2018.

[59] Z. Miller, B. Dickinson, W. Deitrick, W. Hu, and A. H. Wang, "Twitter spammer detection using data stream clustering," *Information Sciences*, vol. 260, pp. 64–73, 2014.

[60] S. H. Moghaddam and M. Abbaspour, "Friendship preference: Scalable and robust category of features for social bot detection," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, p. 1516–1528, Mar. 2023.

[61] S. Feng, Z. Tan, R. Li, and M. Luo, "Heterogeneity-aware twitter bot detection with relational graph transformers," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 3977–3985, Jun. 2022.

[62] Y. Liu, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, vol. 364, 2019.

[63] J. Rodríguez-Ruiz, J. I. Mata-Sánchez, R. Monroy, O. Loyola-González, and A. López-Cuevas, "A one-class classification approach for bot detection on twitter," *Computers & Security*, vol. 91, p. 101715, 2020.

[64] S. Feng, H. Wan, N. Wang, J. Li, and M. Luo, "Satar: A self-supervised approach to twitter account representation learning and its application in bot detection," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3808–3817, 2021.

[65] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[66] F. Wei and U. T. Nguyen, "Twitter bot detection using bidirectional long short-term memory neural networks and word embeddings," in *2019 First IEEE International conference on trust, privacy and security in intelligent systems and applications (TPS-ISA)*, pp. 101–109, IEEE, 2019.