Node Duplication in Disease Maps using Graph Neural Networks

Bachelor Thesis

submitted by

Benjamin Moser

at the

University of Konstanz

Department of Computer and Information Science

1st Supervisor: Dr. Karsten Klein

2nd Supervisor: Dr. Matthias Rupp

Konstanz, 2021

Abstract

Many diseases are thought to arise through complex interactions of biological processes or genetic and environmental factors. Knowledge on related processes and contributing factors is often scattered throughout individual publications. The set of biological entitites and the processes linking them naturally yields a network. Disease maps are large-scale digital diagrams of such networks that describe all processes relevant to a given disease in a formalised manner. They serve as a knowledge base for visual exploration and computational analysis. To date, finding a high-quality layout requires considerable manual curation by experts. A possible means for improving a preliminary layout is to duplicate an existing node and re-destribute its incident edges among the duplicates. The decision of whether to duplicate a node is commonly still left to the expertise of a human curator. Previous work suggests to employ a machine learning model trained on a set of already laid-out diagrams to predict node duplication.

In this work, we explore the usefulness of Graph Neural Networks (GNNs) for predicting node duplication. We assess which of the features suggested in previous work serve well as predictors, and explore approaches to integrate biologically meaningful information in the form of Gene Ontology term annotations. Furthermore, we consider some variants of the basic GNN model and additional techniques such as addressing class imbalance. Finally, once a prediction has been made, we propose an approach to determine the number of duplicates to be introduced and how to distribute incident edges among the duplicates.

Our results show that GNNs provide little immediate advantage compared to the Support Vector Machine approach of previous work. Further, we show that, in some tasks, using a sequence of reorganisation snapshots taken during manual curation as training data is not beneficial compared to simply using the single finished, fully laid-out diagram. We see that it makes little difference whether structural features such as node centralities are calculated based on the bipartite graph interpretation of the disease map or on its bipartite projection. Next, we show that directed node degree is already a strong indicator for node duplication in the considered data. However, using additional structural features still marginally increases classification performance. Exploiting Gene Ontology annotations remains challenging due to the complexity and ambiguity of the provided information. Our approach to determine the number of duplicates and attachment of incident edges appears promising.

Contents

1 Introduction

Deciding node duplication is an essential problem for layouting large biological process diagrams such as disease maps. Commonly, this decision is still left to manual curation by an expert. Previous work [1] suggests to train a machine learning model to suggest to the human curactor which nodes should be duplicated. In this work, we aim to extend this approach in several directions. First, we explore the applicability of Graph Neural Networks to this classification task. Second, we consider which of the previously suggested features are useful and introduce a new set of features based on Gene Ontology term annotations. Finally, we suggest an approach to distribute edges after duplication.

Our aim is to support the curation of disease map diagrams using a Machine Learning model. Given a node in a diagram, the model predicts whether that node should be duplicated. We interpret a disease map as a directed, bipartite attributed graph in which species aliases and reactions constitute the bipartite node sets. Nodes are associated with annotations providing additional domain-specific information. The graph representation also allows to compute structural characteristics such as centrality scores. For training the model, disease maps can be given in two variants: Either as a single disease map, or as a sequence of reorganisation steps. Such a sequence is made up of intermediate snapshots of a disease map taken during the curation process. For evaluation, we always consider the variant of a single disease map. Additionally, nodes of the graphs in the training dataset are associated with ground-truth labels indicating whether the node should be duplicated. Given one or several such training graphs, we aim to train a classifier for *binary node classification*, i.e. for making a prediction for nodes in previously unseen graphs. Moreover, assuming we are given the knowledge that a node should be duplicated, we seek to find the number of duplicates to introduce, as well as how to distribute the edges incident to the original node among its duplicates.

A layout can always be improved in terms of edge crossings, edge lengths or node positions by duplicating nodes. However, duplicating a node and re-attaching its incident edges potentially removes a path through that node. That path may be semantically meaningful and in that case it should be represented explicitly in the network structure. The decision of whether a node should be duplicated hinges on assessing whether such paths are meaningful or not, that is, whether the linkage via a node establishes *true* or *false connectivity* of its neighbourhood. Thus, the decision of node duplication ultimately depends on the context (neighbourhood) of the node. Due to the message-passing operations, a Graph Neural Network is able to recognise patterns of node attributes across a local neighbourhood. We hypothesize that the ability to learn such patterns may be beneficial for capturing and generalising the decisions in the training dataset.

We hypothesize that, in general, a node should be duplicated if its neighbourhood is highly heterogeneous. In this case, false connectivity is established between actually unrelated processes. Previous approaches commonly focus on structural or layout-based features to decide node duplication. However, these facets involve domain knowledge only implicitly, if at all. As a characterisation of neighbourhood heterogeneity, we seek to explicitly include domain knowledge about what biological processes a species is involved in. To this end, we consider the Gene Ontology (GO) terms linked to each species in a given disease map. In particular, we consider the terms of the *biological process* subtree of the GO graph. We represent a GO term by an embedding vector representing its position – and thus its semantics – in the GO graph.

We begin by reviewing the basic notions and definitions in Section 2. In Section 3, we review related work from the perspectives of graph drawing, machine learning and ontologies. We describe the actual methods employed for preprocessing, model training and classification in Section 4. Additionally, we give a short overview over our technical implementation of these methods. In Section 5, we describe and discuss the performance of different approaches. Finally, we conclude with a general discussion of results (Section 6) and possible directions for future work (Section 7).

2 Background

In this section, we review the basic notions and definitions relevant for this work. We introduce biological networks in general, disease maps in particular and highlight specific challenges that arise when working with such networks. Further, we provide background on the classification models used in this paper. We motivate and derive Graph Neural Networks based on the notion of neighbourhood aggregation. We proceed by informally deriving the basic formulation of Support Vector Machines and conclude with methods for evaluating the performance of classifiers. Related work is reviewed in Section 3.

2.1 Biological Networks

The behaviour of biological systems is often shaped by complex interactions. The set of entities in a system and their interactions (relationships) naturally form a network. We refer to a biological entity in such an interaction network as a *species*. Choices of what species and relationships to consider yield different kinds of biological networks. To provide an overview, we describe some prominent examples:

- *Protein-protein interaction* (PPI) networks describe the complex interactions between different proteins. Analysing the entire network of interactions is interesting because the effective biological function of proteins is rarely defined based on their identity alone, but rather on their roles as enzymes or signalling molecules in relationship to other proteins. PPI networks can be useful to infer the biological function of an unknown protein or gene, or to group functionally similar proteins.
- A *metabolic model* of some organism consists of a formally described set of chemical compounds (*metabolites*) as well as a set of chemical reactions or interactions between metabolites. Computational analysis methods on metabolic networks can be used to predict the growth of an organism under specific conditions, identify key intervention targets to alter metabolic processes, or identify relatively independent metabolic subsystems.
- *Disease maps* visually describe species and interactions that are of relevance to a specific disease. They serve as a comprehensive and coherent collection of existing knowledge that is otherwise scattered across individual publications.

Species and relationships need not necessarily have a direct physical counterpart. Several works investigate the interactions between diseases, drugs, or phenotypes, potentially connecting concrete physical entities (such as proteins) to abstract entities (such as, for instance, drug side effects) [2, 3]. Moreover, biological networks may serve as a scaffold to put data on individual species into relation with one another.

2.2 Disease Maps

Metabolic models or protein-protein interaction networks are abstract models. They are given by formally described sets of species and interactions and are thus well suited for computational analysis. However, the ability of human experts to obtain insight from network data cannot be fully replaced. To this end, it is necessary to find a representation of the abstract network that is useful to the consumer. Most commonly, this means finding a visual representation, that is, a *drawing* of the network. Moreover, metabolic models or PPIs are general models. Not all contained information may be relevant for a given task, such as investigating the mechanisms behind some particular disease. Particularly when presenting a wealth of data to human viewers, it is important to focus on on the subsets that are relevant for the task at hand.

Individual biochemical pathways can be described visually by *process description diagrams* [4] . These typically involve less than a few hundred nodes and edges. Species are represented as discs or boxes and are linked by lines representing their relationships. Additionally, reactions may also be represented by distinct graphical

elements other than lines. We refer to the visual representation of a species as a *species alias*. Such diagrams have traditionally been used to describe reaction cascades and metabolic subsystems, first in the form of hand-drawn illustrations and later as computer-generated graphics [5, 6, 7]. An example is given in Figure 2.1.

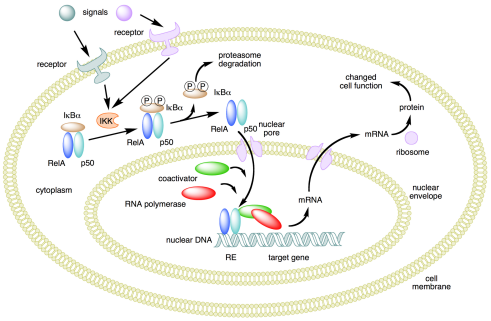
However, several diseases such Alheimer's Disease or Parkinson's Disease do not depend merely on a single pathway. Rather, they are thought to arise through complex interactions of biological processes or genetic and environmental factors. Knowledge on relevant factors is obtained incrementally and scattered across individual publications or database entries. Thus, a systematic understanding of the involved actors and their relationships is essential [8, 9]. For several diseases, experts have assembled *disease maps*, comprehensive visual diagrams combining all known mechanisms relevant for a given disease. These diagrams are particularly suited for visual, interactive exploration. An example is given in Figure 2.1. A disease map can be seen as a kind of biological process diagram. Note that a disease map may well focus on describing metabolic processes or interactions between proteins. The distuingishing property between disease maps and other, abstract biological networks is that disease maps additionally provide an informative visualisation of the network structure. To summarise, disease maps differ in nature from other types of biological networks in the following aspects:

- Disease maps are assembled based on the judgement of their curators. Only processes that are deemed relevant or informative to the given objective are included.
- While other biological networks such as PPI networks or metabolic networks are defined mainly by their abstract network structure, a disease map is an actual visual diagram. Species and relationships have been laid out to optimally present the included information.
- Recent disease maps contain up to several thousands of species and reactions and are very rich in information beyond the mere enumeration of species and their pairwise relationships. For example, a disease map may contain different types of species such as proteins, genes, phenotypes *etc.*. Species may have different states (e.g. "phosphorylated") and be nested in *complexes* (groups). Further, species and relationships are often annotated with links to external databases such as Entrez Gene [10] or UniProt [11]. This poses special challenges particularly for layout and interactive exploration.

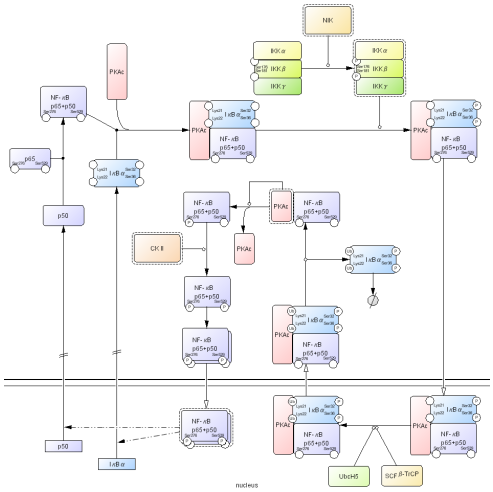
Traditionally, biological process diagrams have been drawn as pixel- or vector-based graphics. Formalised, digital representations provide several advantages:

- Creating a disease map requires a high amount of effort and domain knowledge. The extraction of knowledge from scientific publications or databases, as well as finding an adequate visual layout can be supported by computational tools.
- Entities in the diagram may be annotated with additional information such as links to research publication or database entries.
- A formalised representation enables the use of computational methods for analysis and interactive exploration (see Section 3).

Because of the complexity of the contained information, finding a drawing for a disease map that is suited for visual exploration is not trivial.



(a) Traditional process diagram [12]. The artifact is a pixel or vector graphic, containing only the information shown in the image. Species are represented as different shapes. Reactions are represented by arrows.



(b) Diagram as created with *CellDesigner* [13]. The artifact is a formal description of species and relationships, combined with layout information. Species are represented as large, coloured boxes. Biochemical reactions are represented by arrows.

Figure 2.1: Two examples for different variants of biological process diagrams.

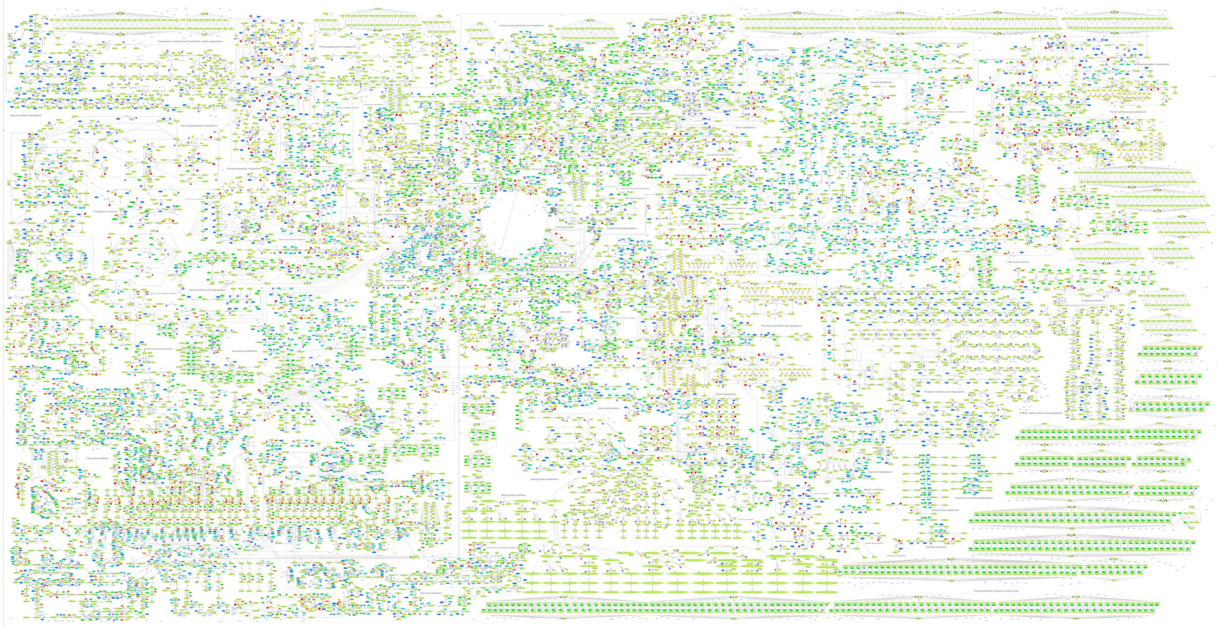


Figure 2.2: Zoomed-out view of the entire *ReconMap*, a diagram describing the human metabolism [14].

2.3 Drawing of Biological Process Diagrams

Network-structured data can be represented formally as a *graph* given by a set of entities (*nodes*) and a set of relationships(*edges*). The most widely used and intuitive visualisation paradigm is the *node-link diagram* in which nodes are represented graphically as dots, circles or boxes and edges are represented by lines. For sake of simplicity, we use *nodes* and *edges* ambiguously both in their mathematical sense and for their graphical representations. Finding a *layout* of a graph (also called *graph drawing*) means finding positions for node representations and potentially also routing edges. What makes a good layout generally depends on the specific kind of network data and the task of the consumer of the visualisation. In general, a good layout is commonly required to avoid edge crossings or overlapping nodes, to be compact and to keep euclidean distances proportional to graph distances. Additional constraints may be for example the preservation of symmetries, clear representation of hierarchical structure or preservation of a viewer's mental map when updating a dynamically changing graph layout. Automatic graph layouting is a highly studied topic and numerous different methods are available (see Section 3).

Leerstelle

Biological process diagrams, however, often contain subgraphs with particular semantics, which must be considered explicitly in the layout in order to convey the contained information as effectively as possible. Because of this, it is still common practise to draw such diagrams manually, or to obtain an initial automatic layout and adjust that manually. This process is extremely time-consuming and often requires specific domain knowledge. We outline some of the challanges in drawing biological process diagrams [15, 4, 16]:

practice?

- Large diagrams such as the disease maps considered in this work often exhibit a clear structural and visual hierarchy.
- Biochemical reactions often involve main substrates and products as well as secondary cofactors and enzymes. Substrates and products are usually placed orthogonally on opposite sides while cofactors and enzymes are placed on the side.
- Biological pathways often involve reaction cascades. These should be displayed such that the cascade is distuingishable and easy to follow. It may be preferrable to align cascades to the natural reading direction of the viewer (commonly *top-to-bottom* and *left-to-right*).
- Some biological pathways involve cyclic patterns and these should be clearly distuingishable as such.
- Biological process description diagrams may contain more information than merely species and reactions. Species can be (recursively) grouped into complexes. A complex is commonly represented as a box containing the visual representation of the species and thus its contents also need to be laid out. Further, cellular compartments are also commonly represented visually as large boxes or frames containing the biological processes therein. Since transport in and out of the compartment and other reactions involving the membranes are of high biological relevance, proper placement of nodes and edges inside, outside or on the boundary of compartments is critical.
- It may be the case that a single species often is involved in several different biological processes. It may be preferrable to represent that species by multiple, separate visual representations.

The question of if and when to represent a species by multiple different visual representations instead of a single one is exactly the focus of this work.

2.4 Node Duplication & Connectivity

If the same species S is involved in several different biological processes, S may either be represented by a single visual representation and linked to all involved processes, or it may be represented by multiple visual representations, each linked only to some processes. In any case, each path through S defines a connectivity between the involved processes.

However, it may be the case that some processes involve the same species, but that species' role is completely unrelated between the two processes. This is the case, for instance, if the processes involve different physical

instances of that species, if the species is available in such abundance that the actual physical instance is irrelevant, or if the species is merely an unimportant byproduct.

Thus, if a species S is involved in multiple processes, we need to assess between which of these processes there exists in fact a *true connectivity* via S; or which merely involve S in different, unrelated contexts (*false connectivity*). True connectivity should be represented in the network structure and the visual diagram as a path through S. There should be no edges implying false connectivity. False connectivity can be resolved by introducing another visual representation for S and re-attaching any incident edges. We refer to this procedure as *node duplication*.

These considerations are important for finding a faithful diagram layout. If the number of involved processes is large, having only a single graphical representation for S may lead to a high amount of visual noise in the diagram: there may be a large number of edges, covering a long distance and linking actually completely unrelated processes. Having many independent representations certainly makes it easier to avoid visual noise. However, connections between different subgraphs may no longer be encoded explicitly, omitting crucial information from the diagram.

The criteria for deciding between true and false connectivity, and thus node duplication are not immediately clear. In practise, the decision is made by experts case-by-case, or general heuristics are employed (see Section 3). In this work, we aim for a more precise approach to decide node duplication automatically in the context of disease maps.

2.5 Biological Databases and Ontologies

Different research projects on the same organism or disease deal in principle with the same universal sets of biological entities and relationships. For instance, if two projects were to describe the metabolic network of *E. coli*, both are likely to mention the function of Adenosine Triphosphate (ATP), a basic molecule that appears in many metabolic reactions. Both projects use the same notion of ATP and its effects, yet they may describe it differently, e.g., by its full name, abbreviation or chemical formula. This hinders knowledge transfer and -integration. We are striving towards a structured, formalised body of knowledge for representing information about physical entities such as molecules, but also on biological terms describing processes (e.g. “glycolysis”), localisation (e.g. “cytoplasm”) or functions.

There are several publicly available databases that gather information on biological entities such as proteins (*UniProt* [11]), genes (*EntrezGene* [10]) or drugs (*DrugBank* [17]). These databases often gather basic information and related research about the entity.

Likewise, biological terms describing processes, functions or cellular localisation can be represented in an *ontology*, a (directed acyclic) graph of terms and their relationships. For instance, the terms “glycolysis” and “carbohydrate metabolic process” may be connected by an *involved-in* relationship. The Gene Ontology project [18] provides three distinct ontology graphs describing molecular functions, cellular component or biological processes. Each ontology graph loosely describes a relationship, however, links between hierarchies may also exist, for instance that the DNA repair process occurs in the Mitochondrion.

2.6 Supervised Learning for Classification

Our goal is to predict whether a species alias in a disease map should be duplicated or not. For each species alias, we extract *features*, which we deem to be characteristic for the target label of the node. Additionally, we are given training data in the form of one or several disease maps and a binary label for each species alias describing whether it was duplicated during manual curation. We aim to use this training data to fit a machine learning model such that it will be able to make meaningful predictions for species aliases in other disease maps.

Thus, we are working in the setting of *supervised learning*, which we briefly introduce in the following [19, 20]: We consider n observations $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ (also called *training data* or *examples*) to be drawn from an (unknown) distribution P over $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{Y}$ is the label domain and $\mathcal{X}$ is the feature domain. For the scope of this work, we restrict ourselves to the *binary classification problem*, i.e. $\mathcal{Y} = \{0, 1\}$. We assume that $\mathcal{X} = \mathbb{R}^d$ and the number of dimensions d is large. Let us denote the set of observed *ground-truth* labels as $\mathbf{Y} := \{y_i\}_{i=1}^n$ and the set of observed feature vectors as $\mathbf{X} := \{x_i\}_{i=1}^n$. $\mathbf{X}$ can also be seen as a matrix in $\mathbb{R}^{n \times d}$. We assume that labels are generated by an unknown function f such that $y_i = f(\mathbf{x}_i)$. Further, we are given a *loss function* $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ that describes how different a prediction is from the true outcome. In general, we aim to find a (parameterised) function $\tilde{f}$ that minimises the *true risk* $\mathcal{R}(\tilde{f}) := \mathbb{E}_P\left[\mathcal{L}(y_i, \tilde{f}(\mathbf{x}_i))\right]$, where $\mathbb{E}_P$ is the expected value over P. Since the original distribution P is unknown, we instead seek to minimise the *empirical risk* on the training data $\mathcal{R}_{\text{emp}}(\tilde{f}) := 1/n \sum_{i=1}^n \mathcal{L}(y_i, \tilde{f}(\mathbf{x}_i))$ and assume the empirical risk approximates the true risk. In the following, we call a concrete choice of $\tilde{f}$ a *model* or a *classifier*. $\tilde{f}$ is commonly parameterised by a set of *model parameters* θ.

For finding the classifier $\tilde{f}$, we can choose out of a variety of existing approaches. The two model families we consider for classification in this work are graph neural networks and Support Vector Machines. We describe these in detail in the following sections.

2.7 Graph Neural Networks

A possible family of models that can be employed for classification is that of *neural networks*. A particular flavour of neural networks that we consider in this work are *graph neural networks*. We introduce basic neural networks by example of one of its simplest variants, the *multilayer perceptron*, or *fully-connected neural network*. We then introduce the notion of convolutions on grid-structured data and proceed to generalise the intuition to graph-structured data, yielding the class of graph neural network models. We proceed to describe how GNN models can be used for node classification. The derivations are based on Zhang et al. [21] and Bronstein et al. [20].

Neural Networks The most basic building block of any neural network is a single *neuron*, which is defined as the composition of a linear transformation with a nonlinear *activation function* σ. Formally, for a single input vector $\mathbf{x}_i$, the output of a single neuron is given by $\sigma(\mathbf{x}_i^T \mathbf{w} + \mathbf{b})$ where the vector $\mathbf{w}$ is said to contain the *weights* of the linear transformation and $\mathbf{b}$ is the *bias*. The stacking of multiple neurons, of which each takes $\mathbf{x}_i$ as input, constitutes a *fully-connected* or *dense layer* in a neural network. If inputs and weights are stacked in matrices $\mathbf{X}$ and $\mathbf{W}$, respectively, the output of a single layer is given by $\sigma(\mathbf{X}^T \mathbf{W} + \mathbf{b})$ where σ is applied element-wise. A neural network is obtained by stacking multiple layers sequentially, such that the output of one layer is the input to the next layer. The output of an intermediate layer is commonly called *latent* or *hidden representation*. Formally, if $\mathbf{H}^{(i)}$ is the output of the i-th layer, then the output of the $(i+1)$-th layer is given as

$$\mathbf{H}^{(i+1)} = \sigma((\mathbf{H}^{(i)})^T \mathbf{W}^{(i+1)} + \mathbf{b}^{(i+1)}). \tag{2.1}$$

Possible choices for the activation function σ are the sigmoid or tanh functions. Another common choice used in this work is the ReLU function given by $\text{ReLU}(x) := \max(0, x)$. A variant of this is to use a linear function in case the input is negative:

$$\text{pReLU}(x) := \max(0, x) + \alpha \min(0, x) \tag{2.2}$$

where α is a learnable model parameter.

Training The layer weights $\Theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(l)}\}$ are considered to be the model parameters θ in the sense of the supervised learning framework described in Section 2.6. We aim to find a set of parameters such that the empirical risk is minimised. This objective can be optimised by *Gradient Descent*: We iteratively adjust the

weights such that the model prediction is improved with respect to the empirical risk $\mathcal{R}_{\text{emp}}$. In the context of neural networks, $\mathcal{R}_{\text{emp}}$ is also sometimes called the total *loss*, not to be confused with the loss function $\mathcal{L}$. More precisely, we consider the partial derivative of $\mathcal{R}_{\text{emp}}$ with respect to each weight matrix, and iteratively update the weights for a maximum number of steps or until the risk no longer improves. We can additionally specify the step size by a given *learning rate* η. Formally, in each step, also called *epoch*, we set

$$\mathbf{W}^{(i)\prime} \leftarrow \mathbf{W}^{(i)} - \eta \frac{\partial \mathcal{R}_{\text{emp}}}{\partial \mathbf{W}^{(i)}}.$$

Hyperparameters Characteristics such as the number of layers, the number of neurons in each layer, the choice of acitvation function and the choice of loss function and learning rate determine the *model architecture* and are called *hyperparameters*. The proper choice of hyperparameters is essential to the performance of the model and often determined empirically via search techniques.

Automatic Feature Extraction and Message-Passing In practise, selecting suitable features is in itself a substantial step in the process of finding a well-performing classifier. Particularly in high-dimensional domains, feature selection is not trivial. Consider the use-case of trying to find a model that, given a grayscale pixel image of dimensions (w, h), classifies the image based on whether it depicts a dog or a cat. It is not trivial how to manually extract features (criteria) from the image that are predictive of the target class. One possible approach would be to consider the brightness value of each input pixel as a feature, i.e. $\mathbf{x}_i \in \mathbb{R}^{w \cdot h}$. However, looking at each pixel in isolation is unlikely to be useful since certainly the target class depends on the values of pixels in their context. Further, note that the hidden representation of some input (e.g. given by Equation 2.1) may be considered an alternate feature representation of this input. In this sense, neural networks can be thought to perform *automated feature extraction*.

Convolutions on Grids For sake of intuitive appeal, let us further entertain the example of classifying pixel graphics. We assume each pixel contains only one channel of information and denote an image of size (w, h) as a $w \times h$ matrix $\mathbf{X}$. Analogous formulations can be used for other kinds of structured data, including 1-dimensional grids (sequences) such as RNA and DNA sequence alignments [22, 23]. It is important to note that we have additional information on the structure of the input, namely that its pixel values are arranged in a grid. This means there is a well-defined notion of context, or *neighbourhood*. Aggregating information across a local neighbourhood may enable a model to capture not only characteristics of individual pixels but higher-order patterns. To faithfully extract local patterns, the aggregation operation should be *local* (the aggregation considers only a part of the input image) and *translation invariant* (the aggregation responds similarly to the same input patch, regardless to where the patch is positioned in the entire grid). In the context of neural networks, such an aggregation is called a *convolution* [21]. In the case of linear aggregation and grid-structured data, we can express a convolution operation centered on pixel $\mathbf{X}_{i,j}$ as

$$\mathbf{H}_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \mathbf{V}_{a,b} \mathbf{X}_{i+a,j+b} \tag{2.3}$$

where u is the bias, and Δ is the (window) *size* of the *convolution kernel* $\mathbf{V}$. Note that the kernel can be any computation adhering to the constraints of locality and translation invariance. Equation 2.3 describes a *convolutional layer* and a neural network containing such layers is called a *convolutional neural network* (CNN).

In image classification, a single input to the neural network is typically an entire image (a collection of pixels), and the convolution operator is applied to each pixel. Successive application of convolutional layers, potentially with different kernels can be thought to aggregate increasingly higher-order patterns in the input data. Additionally, the information of a local neighbourhood can be aggregated into a lower-dimensional value by *pooling* operations. The extracted patterns can be considered intermediate, higher-order feature representations and successive convolution operations extract increasingly higher-order features. Since a kernel computes a value for a given pixel based on the features of itself and its neighbours, we can interpret

the operation of a kernel to perform *message-passing*: Neighbour nodes construct and transmit messages to the target node, which are then combined with the target node's features into the final output.

Convolutions on Graphs We extend the notion of a convolution from grid-structured graphs to arbitrary simple graphs. We will see that the convolutions considered here are trivially local and translation invariant. The key differences are that the order and the size of the neighbourhood is no longer fixed. As such, a potential aggregation must be *node-order invariant*.

Let us consider an attributed graph G with node set V. Let $\mathbf{h}_i$ be the (intermediate) feature representation of vertex $v_i \in V$. Let $\mathcal{N}(v)$ be some neighbourhood of $v \in V$. Typically, $\mathcal{N}(v)$ is chosen as the 1-hop adjacency in G. However, note that different notions of neighbourhood can also be applied, so the input graph must not necessarily correspond directly to the computation graph that defines how messages are passed. or instance, GraphSAGE [24] samples a fixed number of adjacent nodes. We can describe a simple convolution operation on G as follows: the new latent representation $\mathbf{h}'_i$ of v_i is based on messages received by its neighbours. Each neighbour encodes its attributes by means of a function Msg. These messages are aggregated using a permutation-invariant aggregation function Agg. Finally, the neighbours' input and the node's own features $\mathbf{h}_i$ are coalesced via an update function Update into the new latent representation $\mathbf{h}'_i$. If we consider each node to be part of its own neighbourhood, this can be summarised as:

$$\mathbf{h}'_i \leftarrow \textsc{Update}(\textsc{Agg}(\{\textsc{Msg}(\mathbf{h}_j, \mathbf{h}_i) \mid j \in \mathcal{N}_i\})) \tag{2.4}$$

Concrete choices of Msg, Agg and Update give implementations of *graph convolution layers*. A neural network containing such layers is commonly called a *graph neural network* (GNN) or *graph convolutional network* (GCN).

In simple GNN architectures, Msg is a linear feature extraction and depends only on the sending node, i.e. $\textsc{Msg}(\mathbf{h}_j, \mathbf{h}_i) = \mathbf{W}\mathbf{h}_j =: \mathbf{msg}_j$ for some learnable weight matrix $\mathbf{W}$. Note that like the kernel parameters $\mathbf{V}$ in Equation 2.3, these weights are shared among the message-passing operations of different nodes. Update is the application of a non-linear activation function and Agg is given by

$$\textsc{Agg}(\ldots) = \bigoplus_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{msg}_j,$$

where α_{ij} is a coefficient determing the importance of $\mathbf{msg}_j$.

The GNN framework of Equation 2.4 produces a hidden feature representation for each node. This is the basis for solving node-level tasks such as the classification or clustering of individual nodes. Another common use case is to make a prediction for the entire input graph, the most prominent example in context of life sciences being molecule graphs. In this case, we want to aggregate all node features to produce a single value describing the input graph. Like in CNNs, such an aggregation is called a *pooling* layer. This can be done either by application of a simple permutation-invariant aggregation function or by iteratively coarsening the graph together with its node-level feature representations [25].

Simple GNN The *GCN layer*, as proposed by Kipf and Welling [26], defines α_{ij} as a constant depending on the degrees of v_i and v_j, namely $\alpha_{ij} := 1/\sqrt{d_i d_j}$.

GNNs with Attention The *Graph Attentional Layer* (GAT) [27] allows the importance score α_{ij} to be learnable, i.e. adjusted via backpropagation and gradient descent during network training. An attention mechanism A determines the importance e_{ij} of $\mathbf{msg}_{ij}$. If the neighbourhood $\mathcal{N}_i$ is defined as the 1-hop-neighbourhood in the input graph, this can also be interpreted as the importance of edge (v_i, v_j). In its prototypical formulation, A is a single-layer fully-connected neural network with learnable weights $\mathbf{a}$ and activation function σ_{att}. A receives both the feature representations of v_i and v_j as input, combined by concatenation:

$$e_{ij} := A(\mathbf{msg}_i, \mathbf{msg}_j) = \sigma_{\text{att}}(\mathbf{a}^T(\mathbf{msg}_i \| \mathbf{msg}_j)) \tag{2.5}$$

Importance scores e_{ij} are obtained by normalisation with the *softmax* function:

$$\alpha_{ij} := \text{softmax}_{j \in \mathcal{N}_i}(e_{ij}) := \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}. \tag{2.6}$$

Attention is a technique that has previously been used successfully in other neural network architectures. In particular, Graph Attention Networks are analogous to the Transformer architecture [28] that has achieved great popularity in areas of natural language processing such as machine translation. For translation from a source language to a target language, text input is typically given as two sequences of tokens. The key idea behind the Transformer architecture in NLP is to apply an attention mechanism to assess the importance of other words in the source or target sequence with respect to the current word. Note that while the Transformer architecture relies heavily on the attention mechanism, it also directly implements the idea of message-passing based on known relationships between atomic inputs.

Graph Neural Networks allow us to apply common Deep Learning techniques to graph-structured data. For node-level learning tasks, this allows us to consider the explicit relationships between individual nodes. For graph-level tasks such as graph classification, GNNs are able to process the entire graph without relying on a fixed-length encoding. Furthermore, the aggregations are invariant to permutation and rotation.

Additional Techniques There are some additional techniques that are commonly used for designing and training neural networks. *Dropout* [29] randomly disables a subset of neurons in each layer in a training epoch. The probability that a neural is disabled is the *dropout rate*, sometimes referred to simply as *dropout*. This is motivated by the observation that during training neurons might co-adapt to each other. *Batch normalisation* [30] is a normalisation scheme applied to each layer. *Skip connections* [31] provide the output of a layer not only to the next layer in sequence but to even later layers. For a layer with an incoming skip connection, the input of the previous layer and the input of the skip connection can be combined by either sum or concatenation.

neuron?

Neural Networks as Classifiers We now consider how graph neural networks can be employed for node classification. In each training epoch, the neural network receives as input the node feature vectors and the graph's adjacency matrix. Intermediate layers of neural networks can be interpreted to compute useful feature representations. One way to obtain a classification prediction from a neural network is to use these intermediate representations as an input to another off-the-shelf classifier (such as Support Vector Machines, for instance [32]). Another, more common approach, however, is to design the network such that the last layer contains one output neuron for each target class. Given some input $\mathbf{x}$, the output value of the i-th neuron expresses the estimated probability that $\mathbf{x}$ is of class c_i. To produce values in $[0, 1]$, the results of the final layer are normalised, e.g. with the *softmax* function (see Equation 2.6). In order to train the neural network for classification, we need a loss function $\mathcal{L}$ to assess the quality of the prediction. We introduce the *Binary Cross Entropy* loss from the viewpoint of maximum likelihood estimation (based on [21]) but note that it can also be derived via the notion of cross-entropy. Generally speaking, we seek to estimate parameters Θ of an assumed probability distribution P given some observed data X. The method of *maximum-likelihood estimation* postulates that we should pick the parameters such that the observed data is most likely (i.e. occurs with highest probability) under P. The likelihood with respect to parameters Θ is measured by a *likelihood function* $L(\Theta)$. Formally, we aim for

$$\hat{\boldsymbol{\Theta}} = \text{argmax}_{\Theta} P(X \mid \Theta) = \text{argmax}_{\Theta} L(\Theta)$$

Because of connections to entropy and computational convenience, we instead consider the negative logarithm:

$$\hat{\boldsymbol{\Theta}} = \text{argmin}_{\Theta} - \log L(\Theta)$$

Looking at the negative log-likelihood in more detail directly yields the *Binary Cross-Entropy loss*. For notational convenience, let $\mathcal{Y} = \{0, 1\}$. Let $\pi_i = P_{\Theta}(y_i = 1 \mid \mathbf{x}_i)$ be the estimated probability that $\mathbf{x}_i$ belongs to

class y_i. Since we are dealing with binary classification, we have $P_\Theta(y_i = 0 \mid \mathbf{x}_i) = 1 - P_\Theta(y_i = 1 \mid \mathbf{x}_i)$. Under the assumption that input samples are independent and identically distributed, we have

$$\begin{aligned} -\log L(\Theta) &= -\log \prod_{i=1}^{n} (\pi_i)^{y_i} \cdot (1-\pi_i)^{1-y_i} \\ &= -\sum_{i=1}^{n} \underbrace{y_i \log(\pi_i) + (1-y_i)\log(1-\pi_i)}_{:= \mathrm{CE}(f(\mathbf{x}),y)} \end{aligned} \tag{2.7}$$

where $\pi_i = f(\mathbf{x})$, f is some classifier and CE is the binary cross-entropy loss. The terms corresponding to either positive and negative class can be weighted with additional coefficients to account for class imbalance.

2.8 Support Vector Machines

Besides neural networks, another class of models that we consider for node classification is that of Support Vector Machines (SVMs). SVMs have no inherent way to explicitly consider the graph structure. Commonly, structural characteristics are encoded as node features (see Table 5.1). In this section, we present the basic derivation of SVMs with the motivation of providing intuition behind the *cost* hyperparameter and choice of kernel functions and their parameters. The derivations are based on Tibshirani, Friedman, and Hastie [33].

Support Vector Machines are linear classifiers. The basic idea is to find a hyperplane in the (possibly transformed) feature space $\mathcal{X}$ that best separates the training data with respect to its ground-truth class assignments. Note that, thus the prediction output of a SVM is binary. One can obtain a confidence score alike to the output of NNs via cross-validation.

Preliminaries For sake of notational convenience, let $\mathcal{Y} = \{-1, 1\}$. A *hyperplane* is an affine set of points $L := \{\mathbf{x} \mid \mathbf{x}^T\beta + \beta_0 = 0\}$. The signed distance from a point $\mathbf{x}$ to L is given by $d(x, L) := 1/\|\beta\|(\mathbf{x}^T\beta + \beta_0)$. Based on L, we can define a linear classifier

$$h_L(\mathbf{x}) = \mathrm{sign}(\mathbf{x}^T\beta + \beta_0). \tag{2.8}$$

$\mathbf{X}$ is *linearly separable* if there exists a hyperplane L such that $h_L(\mathbf{x}_i) = y_i$ for all $\mathbf{x}_i \in \mathbf{X}$. L is then called a *separating* hyperplane, or *decision boundary*.

Linearly separable case For now, assume that $\mathbf{X}$ is linearly separable. We wish to find a suitable separating hyperplane. One possible approach would be to minimise the distance of misclassified points to the hyperplane. Doing so by gradient descent yields the *perceptron training algorithm*. However, that does not guarantee a unique solution. Further, if $\mathbf{X}$ is not linearly separable, the algorithm will not converge at all. Another possible approach is to search for a hyperplane that maximises the *margin* of L, denoted by $\gamma(L)$ with respect to $\mathbf{X}$, i.e. the distance from the hyperplane to the closest point:

$$\gamma(L) := \min_{x \in \mathbf{X}} 1/\|\beta\| \left| \mathbf{x}^T\beta + \beta_0 \right|.$$

The maximum-margin separating hyperplane is unique. Further, it seems reasonable to assume that a maximum-margin separating hyperplane will do well in generalising to unseen points. We aim to find a hyperplane $L(\beta, \beta_0)$ that achieves:

$$\begin{aligned} &\underset{\beta, \beta_0}{\text{maximize}} && \gamma(\beta, \beta_0) \\ &\text{subject to} && y_i(\mathbf{x}_i^T\beta + \beta_0) \geq 0; i = 1, \ldots, n \end{aligned} \tag{2.9}$$

where the constraint expresses that each example $\mathbf{x}_i \in \mathbf{X}$ should lie on the side of the hyperplane according to its class.

Since L and γ are scale invariant, i.e. $\gamma(\beta, \beta_0) = \gamma(\lambda\beta, \lambda\beta_0)$ for any $\lambda \neq 0$, we can assume $\gamma(\beta, \beta_0) = 1/\|\beta\|$, i.e. $\min_{x \in X} \left|\mathbf{x}^T\beta + \beta_0\right| = 1$. Equation 2.9 can then be simplified to

$$\begin{aligned} \underset{\beta, \beta_0}{\text{minimize}} \quad & \|\beta\| \\ \text{subject to} \quad & y_i(\mathbf{x}_i^T\beta + \beta_0) \geq 1; i = 1, \ldots, n \end{aligned}$$

General case Let us now consider the case that $\mathbf{X}$ is not linearly separable. In this case, we are not guaranteed a solution to the optimisation problems given above. However, we can relax the constraints so that some data points are allowed to lie inside the margin. We achieve this by introducing *slack variables* $(\xi_1, ..., \xi_n)$ that express the allowed violation to the optimisation constraints. Additionally, the optimisation is effectively solved by first transforming it into an equivalent problem known as its *Langrangian Dual*. To this end, it is convenient to slightly reformulate the problem, resulting in

$$\begin{aligned} \underset{\beta, \beta_0}{\text{minimize}} \quad & 1/2\|\beta\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(\mathbf{x}_i^T\beta + \beta_0) \geq 1 - \xi_i, \\ & \xi_i > 0 \end{aligned} \tag{2.10}$$

The *cost* hyperparmeter C can be interpreted as a tradeoff coefficient between the cost of margin violations and simplicity of the decision boundary. For large C, margin violations will be punished more strictly. For small C, violations may be allowed to achieve a hyperplane with lower norm, i.e. smoother decision boundary.

For classification in class-imbalanced datasets, a common approach is to increase the misclassification of the minority class. We can incorporate this in the SVM formulation by specifying a *cost* parameter for the positive and negative class separately. Equation 2.10 then becomes:

$$\begin{aligned} \underset{\beta, \beta_0}{\text{minimize}} \quad & 1/2\|\beta\|^2 + C_P\sum_{i \in P} \xi_i + C_N\sum_{i \in N} \xi_i \\ \text{subject to} \quad & y_i(\mathbf{x}_i^T\beta + \beta_0) \geq 1 - \xi_i, \\ & \xi_i > 0 \end{aligned} \tag{2.11}$$

where P and N are the sets of indices of positive and negative training examples, respectively. C_P and C_N are the misclassification weights of the positive and negative training examples, respectively.

The Kernel Trick For most non-trivial classification problems, the classification function f we seek to approximate is not linear. A method to move beyond linearity but nevertheless use linear classifiers is to transform the input features $\mathbf{X}$ into a higher-dimensional space $\mathcal{X}'$ via some transformation Φ. Depending on the choice of Φ, $\mathcal{X}'$ may be of very high or even infinite dimensionality. Thus, computing Φ explicitly is sometimes not an option. However, for the computations of a Support Vector Machine, only an inner product $\langle \cdot, \cdot \rangle$ in $\mathcal{X}'$ is required. In the following, instead of some feature vector $\mathbf{x} \in \mathcal{X}$, we consider a transformed feature vector $\Phi(\mathbf{x}) \in \mathcal{X}'$. Let's inspect the most central equations for computing SVMs. First, in practise, we do not solve the optimisation problem derived above directly, but instead solve an analogous problem, namely its *Langrangian Dual*. In the dual, $\mathbf{x}_i$ and $\mathbf{x}_j$ only appear when taking their inner product. Further, the decision function h_L (see Equation 2.8) can be rewritten as

$$h_L(\Phi(\mathbf{x})) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i) \rangle \quad + \beta_0\right).$$

In this context, the particular choice of inner product is also called a *kernel function*. One popular choice for K is the *radial basis function* (RBF) kernel, also called *Gaussian* kernel. Recall that a kernel function can be interpreted as a measure of similarity between its two arguments $\mathbf{x}$ and $\mathbf{x}'$. The RBF kernel implements this notion as a decaying function of their distance. If $\mathbf{x}$ and $\mathbf{x}'$ are similar, their distance will be small and $-\gamma\|\mathbf{x} - \mathbf{x}'\|^2$ will be relatively large. The decaying characteristic is implemented by the application of the exponential function. Thus, we can define the RBF kernel as

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\left\|\mathbf{x} - \mathbf{x}'\right\|^2). \tag{2.12}$$

Since the distance is symmetric, K_{RBF} can be interpreted as a bell-shaped function. The hyperparameter γ controls the width of the bell shape, with larger values producing a more narrow shape.

3 Related Work

We directly extend the work of Nielsen et al. [1]. In this work, the authors aim to train a classifier to predict node duplication in a "human-like" manner. They focus on features based on the graph structure. For the classification model, Support Vector Machines are chosen and the authors tune the model to find optimal choices of kernel and hyperparameters. The primary use-case considered in their work is to provide a small, human-digestible number of predictions as suggestions to the curator. Likewise, a similar approach is used to suggest de-duplication operations. The SVM model is trained on a sequence of curation snapshots provided by an expert working on *AlzPathway*, a disease map describing processes related to Alzheimer's Disease (see Section 4.1). Model hyperparameters were tuned and selected based on evaluation on *PDMap*, a diagram on Parkinson's Disease. The final identified model was further validated on *ReconMap*, a large diagram describing the human metabolism (see Figure 2.2) and a number of smaller pathway graphs obtained from Reactome [34]. Finally, a user study among domain experts was conducted on smaller networks, comparing the performance of the machine learning model with the decisions of experts. We reproduce some of their main results in Section 5. In general, the identified model was able to make predictions well above random. However, the user study showed considerable variations even in expert decisions.

3.1 Drawing Biological Networks

There are numerous general-purpose methods for laying out graphs. Examples are force-directed approaches [35], stress-based approaches [36], hierarchical or layered approaches [37] or orthogonal or grid-based drawings [38], to name just a few. Drawing biological networks, however, is often associated with domain-specific additional requirements. Some of these challenges are outlined in Section 2.3. To this end, there has been work done to develop specific methods for biological networks.

Early work focussed mainly on the drawing of process diagrams that contain up to several dozens of nodes, e.g. describing only a single biological pathway [5, 39].

However, the networks we consider in this work are of the scale of several hundreds or even thousands of elements. This poses unique challenges such as respecting a given hierarchy or annotated categories (subsystems), representing semantic motifs (e.g. cyclic patterns) or keeping a uniform distribution of visual elements across the drawing. Due to this, large-scale disease maps are commonly still created manually and research on finding "human-like" layouts is ongoing.

Recent work [4, 40] has clarified the domain specific-requirements of drawing biological process diagrams and suggested specialised drawing algorithms for medium-scale diagrams. With *Metabopolis*, Wu et al. provide an approach based on the inherent hierarchies provided either by explicit subsystem annotations or related connected components [16]. Shortly thereafter, Wu, Nollenburg, and Viola work towards alleviating the uneven space space utilisation of *Metabopolis*, focussing on obtaining a layout that has balanced space utilisation at multiple levels of detail [41]. Further, they explicitly consider node duplication as a tool to improve the layout (see Section 3.2).

3.2 Node Duplication

Related Notions Several related terms appear in the literature that are relevant for the problem at hand. A *currency metabolite* [42] is a metabolite that plays only a secondary role in most of the reactions it is involved in. This role may be to merely supply energy (i.e. act as "currency" in the metabolism) or act as some other form of catalyst. Commonly, currency metabolites appear in abundance in an organism, and often it is assumed that two reactions involving the same currency metabolite are not in fact linked stoichiometrically, i.e. linking

these reactions via a common node would imply false connectivity. Currency metabolites are commonly duplicated.

In the context of decomposing a metabolic network into subsystems, Schuster et al. classify a given metabolite as *external* if it can be considered in to be present in such abundance that its production or consumption has no meaningful effect on the surrounding concentration [43]. Note that this means that the connectivity is not meaningful (*false*) in a stoichiometric sense.

The notion of true connectivity is also reflected in the concept of *key connectors* [44]. Because pathways in a metabolic network seldomly work in isolation, there necessarily will be connections between different pathways. Determining whether a bridging node between two pathways is indeed a key connector or merely describes false connectivity is not trivial and related to the problem at hand.

General Methods While node duplication (also referred to as *Vertex Splitting*) has been treated from a theoretical perspective [45, 46], to the best of our knowledge there are relatively few concrete, general-purpose algorithms that employ automatic node duplication.

Most notably, Eades and de Mendonça N introduce a general graph drawing algorithm that applies node duplication to simplify the graph structure in order to find a better layout [47]. The method is an extension of the force-directed Kamada-Kawai algorithm [48]. They define a measure of *tension* on a vertex that is based on the lengths and directions of the edges incident to it. A vertex is duplicated if its tension is above a user-defined threshold.

Node Duplication has also been applied in the area of Electronic Circuit Design, for example to avoid edge crossings [49] or paths exceeding a maximum length [50, 51].

Henr, Bezerianos, and Fekete consider how to represent duplicates in a social network visualisation in which communities are represented by adjacency heatmaps [52].

In Biological Networks We review related work which explicitly considered node duplication in biological networks. The following references mainly consider the drawing of metabolic networks, which can be considered a subclass of biological process diagrams. However, the referenced methods exclusively rely on structural criteria based on the graph structure and are thus potentially transferrable to disease maps.

A common intuition is that a node shall be duplicated if its neighbourhood is highly heterogenous with respect to some similarity measure. In other words, a node should be duplicated if it is involved in many different, unrelated processes. A possible approach to assess neighbourhood heterogeneity of a node is to consider a graph-based centrality measure. If the target node has a very high centrality score, we conclude that the node must then necessarily be involved in different, unrelated processes, i.e. its neighbourhood is heterogeneous. For a concrete choice of centrality measure, early methods simply considered the node degree [53, 43]. Further work uses the Eigenvector centrality [54]. Other approaches make the notion of neighbourhood heterogeneity more explicit by characterising communities in the given network. A node then has heterogeneous neighbourhood if it is incident to many different communities. Communities can be determined solely on the graph structure, for instance as induced by modularity maximisation [55]. Huss and Holme decide node duplication based on the contribution to overall modularity if the target node is removed [42]. Guimerà and Nunes Amaral classify nodes as different kinds of hub- or connector nodes based on their intra- and inter-community degrees, where communities are determined via modularity maximisation [56]. Communities can also be characterised by domain-specific biological knowledge, for example annotations that describe the cellular compartment [54] or the pathway [57, 34] for a given node.

Recently, in the context of visualising large-scale diagrams such as the disease maps considered herein, Wu, Nollenburg, and Viola [41] propose the following scheme for deciding whether to duplicate a node. Nodes are categorised as *important* or *unimportant* based on whether their degree exceeds a certain degree threshold. Alternatively, the categorisation may be given by an expert. Important vertices are duplicated such that they appear at most once in each cluster. Unimportant vertices are duplicated such that each duplicate has degree of exactly one.

Apart from the work of Nielsen et al., we are not aware of any other previous work that investigates using a machine learning classifier for node duplication.

VANTED [58] offers functionality to duplicate nodes if they exceed a given degree threshold. The tool introduces one duplicate per edge and can optionally lay out duplicates next to each other on a grid. Additionally, in *VANTED*, external data can be mapped to network nodes. Nodes that are assigned more than one dataset can be automatically duplicated. There are several tools that provide functionality to easily introduce duplicates once the duplication decision has been made. *Arcadia* [59] allows to split a node such that each copy has exactly unit degree. *Omix* [60] makes it easier to introduce duplicates with certain connectivity patterns by providing a *motif stamp* tool.

3.3 Disease Maps

To date, disease maps have been created for a number of diseases, including Alzheimer's Disease (*AlzPathway* [61]), Parkinson's Disease (*PDMap* [62]), and recently COVID-19 [63].

Beyond serving as a platform for integrating existing knowledge, computational methods have been applied to disease maps, for instance toidentify molecules and relations essential for the pathogenesis of Alzheimer's Disease [64].

The *Atlas of Cancer Signalling Network* (ACSN) [65] is a collection of diagrams describing signalling processes related to cancer. Recent work [66] has linked ACSN and *ReconMap* [14], a large-scale diagram describing the human metabolism, making the connections between common actors in both networks explicit. This is of particular interest since ACSN focusses on signalling processes, while *ReconMap* describes metabolic processes. Additionally, the information from *ReconMap* is used to augment the ACSN diagram.

Several specialized tools exist for the curation and exploration of large biological process diagrams, including *CellDesigner* [67], *Minerva* [68], *NaviCell* [69], *Cytoscape* [70] and *VANTED* [58]. One of the most common formats used to describe disease maps is an extension to SBML Level 2 given by *CellDesigner*. Further, SBML Level 3 now allows to attach layout information. Another prominent format is SBGN-ML [71].

Detailed information on the disease maps considered in this work can be found in Section 4.1.

3.4 Graph Neural Networks in the Life Sciences

To the best of our knowledge there is no previous work that connects GNNs to node duplication.

Tiezzi, Ciravegna, and Gori provide a method for layouting graphs using GNNs [72] . As a first approach, they train a GNN to draw graphs based on ground-truth examples obtained from other graph drawing software. As attributes, each node is assigned a positional encoding based on the Laplacian Eigenvectors of the graph. The loss function aims to minimise the distances from the produced drawing to the ground-truth drawing (modulo affine transformations). Further, inspired by optimisation-based methods such as Stress Majorisation [36], they employ a GNN to directly minimise the stress function on the predicted node coordinates. Note that the all-pairs-shortest-paths computation has to be done only during training. At inference time, the model predicts node positions based on the supplied positional encoding, which is potentially easier to compute. Additional quality measures, such as the number of edge crossings, can be included in the loss function without sacrificing the advantage that predictions only ever require the graph structure and positional encodings and no additional computation.

GNNs can generally be applied for three different types of tasks: *node-level* tasks in which the units of interest are individual nodes, *edge-level* tasks in which edges are considered as the unit of interest and *graph-level* tasks in which a statement about the entire graph is sought.

Node-level tasks You et al. [73] consider the problem of predicting the function of a protein based on its sequence. A current challenge is bioinformatics is the gap between the number of known protein sequences and the number of protein sequences annotated with a biological function (*sequence-annotation-gap*). You et al. suggest to consider each protein with respect to other proteins it is known to interact with. As such, they propose a GNN approach in which proteins are represented as nodes and connected based on information from Protein-Protein interaction databases. The function annotations here are in fact Gene Ontology terms.

in?

Once the biological function of a protein is known, another challenge is to assess the functional similarity of two proteins. One way to approach this is to compare the (sets of) Gene Ontology terms of two given proteins. Zhong, Kaalia, and Rajapakse compute an embedding for GO terms based on the graph structure of the GO ontology [74]. These embeddings can be thought of to encode the term's position in the graph, that is, its relationship to other terms. A measure of semantic similarity can be derived by comparing the (sets of) computed embeddings.

Single-cell RNA-sequencing (scRNA-seq) is a technology that provides gene expression data on the level of individual cells. In the work of Ravindra et al. [75], each cell is represented as a node and features are its gene expression data. Graph connectivity is defined on a node's k nearest neighbours. The goal is to predict whether a cell corresponds to a healthy or pathological disease state w.r.t to Multiple Sclerosis (MS). The motivation is to work towards developing a diagnostic test for MS based on scRNA-seq technology.

Graph-level tasks Graph Neural Networks have found highly successful applications for the problem of molecular property prediction. This is a graph-level task where the input is a *molecule graph* and we strive to predict specific chemical properties of this molecule. In a molecule graph, nodes represent atoms and edges represent bonds. Nodes and edges have attributes describing e.g. an atoms element, or the type of a bond. Properties to predict may include toxicity or antibacterial activity. This task is relevant particularly in the field of drug development where a large number of molecules has to be tested for potential usefulness as a drug (*virtual screening*). Traditionally, molecules were represented by *fingerprint vectors* which describe characteristics of the entire molecule such as the presence of functional groups. While these fingerprints may well serve as input to a neural network predictor, the structure of these fingerprints is designed manually and often not directly dependent on the prediction task. Graph Neural Networks provide the means to compute such fingerprints directly based on the the structure of the molecule and concrete, low level properties. Moreover, the extraction and aggregation of input features performed by graph neural networks is differentiable and thus the entire prediction pipeline can jointly be optimised end-to-end, yielding task-specific fingerprints. Stokes et al. use this approach to predict the growth inhibition against *E.coli*, eventually resulting in the discovery of experimentally verified potent antibiotics that are structurally different from known antibiotics [76]. Notably, Duvenaud et al. employ this approach well before the recent popularisation of Graph Neural Networks in the style of Equation 2.4 [77]. Focussed reviews of applications of GNNs for molecular property prediction [78] and drug development in general [79] can be found in the literature. Further, Baranwal et al. train a GNN model to compute fingerprints of molecules that are then used to predict their broad metabolic pathway class (e.g. carbohydrate metabolism, amino acid metabolism, *etc.*) [80].

Edge-level tasks GNNs have been applied for edge-level tasks particularly for the problem of link prediction in biological interaction networks. GNNs have been applied to predict interactions between diseases and drugs [81], interactions between drugs, proteins and drug side effects [82] and interactions between proteins [83]. Many other applications can be found in the literature [84].

3.5 Gene Ontology

Henry et al. extract the relationships such as reactions, links to phenotypes and genes or relationships to cellular components given in the *AlzPathway* disease map and construct an ontology of it, aiming to obtain a more formalised, precise and less ambiguous description of the contained knowledge [85].

Ruiz, Zitnik, and Leskovec make explicit use of the GO ontology graph structure by integrating it in a larger network of drugs, diseases, proteins and their known interactions [2]. The GO ontology graph thus links different proteins by a common biological function. Thus, it may be possible to predict if and how a drug affects a disease, even if no explicit relationship is known.

Work exists to develop similarity measures between two GO terms, or two sets of GO terms. [86, 87, 88] These are, by definition, pairwise similarity measures. An alternative approach is to find a embeddings (vectors describing feature representations) of the given GO term such that the similarity of their embeddings reflects the similarities in the GO graph [74].

Einzahl oder Mehrzahl?

Ostaszewski et al. [89] consider the species-level GO term annotations in *AlzPathway* to automatically identify semantic clusters in the diagram, adopting a measure of semantic similarity between GO terms.

4 Methods

In this section, we describe the concrete methods used for data preprocessing, feature engineering and model design. Further, we motivate and describe our approach for redistributing incident edges after duplication.

4.1 Preprocessing

We begin by describing how we transform a given disease map into an input suitable for a machine learning model. The main steps are to extract data from the *CellDesigner*-SBML files describing some disease maps and construct for each an attributed, directed, bipartite graph G. Further, we need to infer features and ground-truth labels for nodes in G. The basic pipeline is illustrated in Figure 4.1. An overview of the actual datasets used in this work is given in Section 5.1.

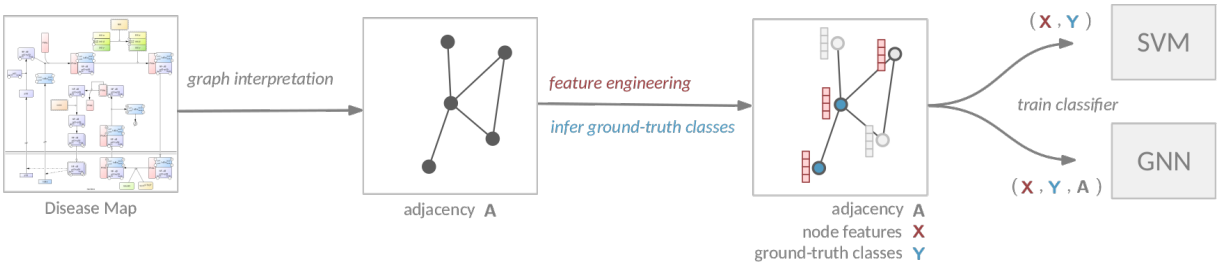


Figure 4.1: Illustration of the basic data flow up to model training. For a given disease map, we construct a corresponding graph. Additionally, for each node in the graph, we obtain node features and a ground-truth label. Some nodes are excluded from prediction. The SVM classifier acts on the node features alone, while the GNN classifier additionally performs message-passing over the graph structure.

The task at hand comes with a particular set of challenges:

- Different disease maps potentially have very different characteristics (see e.g. the difference between AlzPathwayReorg or PDMap and ReconMapin Figure 5.1), which may make it difficult for the model to generalise to unseen disease maps. We evaluate generalisation ability throughout our experiments.
- Typically, only a few nodes are duplicated (see Figure 5.1). Thus, the positive class is underrepresented in the dataset. We have to ensure that model performance does not suffer due to class imbalance. We address this issue by considering two approaches: First, we undersample the majority class. Second, we provide class-specific weights to the ML models such that an example of the minority class will be assigned more importance.
- The ground-truth labels may not be perfectly reliable. For one, the decision whether a node was duplicated or not during curation is subjective to the expertise and preference of the curator. The criteria that were relevant to the curator are most likely not perfectly reflected in the features we provide the model with. Thus, it may be the case that some training examples are contradictory in label with respect to their feature representation. Further, considering several reorganisation steps is likely to introduce contradictory examples if a node is duplicated in some step but not in an earlier one.
- The number of datapoints used for training and evaluation is relatively low compared to other common use-cases for machine learning. Care has to be taken to avoid overfitting, i.e. the model adjusting overly well to the training data at the cost of prediction performance on validation data not seen during training. Further, particularly when partitioning the available data into subsets for training and testing, we have to make sure we still have plenty of representative examples in each subset. We address this issue in part by avoiding to split a disease map internally into training and testing subsets (instead, we use separate, whole disease maps for training and testing). Further, we want to highlight that GNNs have been applied successfully on datasets of comparable size [90]. Note also that the GNN architectures we considere herein are of much smaller complexity (in terms of number of model paramaters) than

? parameters

famous neural network architectures used in computer vision or natural language processing. They may thus not require a particularly large amount of training data to fit all parameters adequately.

Graph construction

Disease maps can be interpreted as bipartite graphs in a natural manner: The bipartite node sets are the set of species and the set of interactions, respectively. The disease maps considered in this work are given in an extension to SBML defined by the *CellDesigner* tool. The format is very rich in information and leaves some room for ambiguity concerning graph construction. In the following, we describe what we take into account to construct a graph.

kleines t?

The most central elements in an SBML model are the lists of reactions and species aliases (visual representations of species). An entry in the list of reactions carries references to species taking part in that reaction. Since different occurences of a species can be visualised as separate species aliases, each entry for a participating species additionally contains a reference to a specific species alias. Participating species are distuingished by the role they play in that reaction. Herein, we consider the basic roles defined by standard SBML: products, modifiers and reactants. In fact, *CellDesigner* provides an even more fine-grained distinction of species participating in a reaction. Species can be *main* reactants or products, *modifiers* or *additional* reactants or products. We omit this for simplicity (see [91], ch. 2.4).

Each species alias is represented as a node. Complexes of species aliases (*complex species aliases*) are represented as a single node. We create directed edges for reactants and products in the direction of the reaction. A modifier is attached by two edges, one in either direction. An example is given in Figure 4.2. This yields a directed, bipartite graph, with the bipartite node sets being the set of (complex) species aliases and the set of reactions, respectively.

For computing structural node features and for message-passing, we sometimes also consider the bipartite projection onto the set of species aliases. The *bipartite projection onto A* of a bipartite graph whose node set is the disjoint union of sets A and B is defined as follows: It contains all nodes in A. Two nodes in the bipartite projection are linked if and only if they have a common neighbour in the original graph. Herein, we compute the bipartite projection based on the undirected interpretation of the graph.

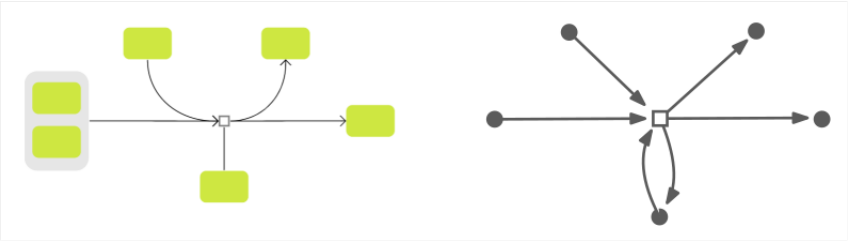


Figure 4.2: Illustration of how a rich SBML model is interpreted as bipartite, directed graph. The left graphic describes the given *CellDesigner*-SBML model. Green boxes represent species aliases, small white boxes represent reactions. Complexes of species aliases are represented by gray boxes. A species alias can be involved in a reaction through different kinds of relationship: As primary input or output (laid out horizontally to the reaction), as secondary in- or output (indicated by curved arcs) or as byproduct without directionality (laid out directly below the reaction)

A species alias can either be simple or *complex* in the sense that it represents a container for other species aliases. This is used to represent e.g. biological complexes of proteins. Simple and complex species aliases are arranged in an arbitrarily nested hierarchy. For the graph structure, we consider only top-level elements, that is, we consider complex species aliases as single nodes and omit their contents. Contents of complex species aliases will be taken into account when considering GO annotations. A reaction may involve an entire complex species alias (CSA) or only a species alias contained in the CSA. Since we interpret the CSA as a single node, an edge will be attached to it in either case.

Species aliases can also be contained in *compartments* representing biological cellular compartments or broader notions of spatial relationship. We do not consider compartments at all herein (see Section 7).

Determining ground-truth Labels

In order to train any supervised classifier, we require a set of examples for which the class is already known. Although several disease maps are publicly available, to the best of our knowledge, none are explicitly annotated with a per-alias label. Thus, in order to obtain usable training data, we first have to infer such class labels. Concretely, given a single disease map or a sequence of reorganisation steps, we need to determine which species aliases are duplicates.

The determination of ground-truth labels is based on comparing a disease map to a next curation step. Concretely, given maps D_1 and D_2, and D_2 is created from D_1 through some reorganisation, we can infer ground-truth labels for D_1 (but not for D_2).

A disease map can be given in two different variants. On the one hand, we may be given a sequence of reorganisation steps of a disease map, i.e. a series of intermediate snapshots of a disease map taken during manual curation. On the other hand, we may simply consider a single, fully laid-out disease map.

In case we consider a single disease map D and not a reorganisation sequence, we interpret D as the result of some reorganisation. Let G be the graph interpretation of D. As input for a classifier we actually require the "previous" step before reorganisation. Since this is not explicitly given, we construct a *collapsed* version G_0 of G by merging all species aliases that correspond to the same species into a single representative alias. In case we are given a reorganisation sequence $(D_1, ..., D_k)$ of disease maps, the result will be a sequence of graphs $(G_0, ..., G_{k-1})$ where G_i is the graph interpretation of D_i and additionally G_0 is the graph corresponding to the collapsed version of D_1.

Determining duplication Given two graphs G_t and G_{t+1}, we want to infer which species aliases were duplicated in the step from G_t to G_{t+1}. Note that we do not want to identify nodes in G_{t+1} that were duplicated, but nodes in G_t that will be duplicated. We refer to these as *duplication parents*. In case we are given a sequence of reorganisation steps $(G_1, ..., G_k)$, we infer node labels by comparing successive steps G_t and G_{t+1}. In case we are given only a single disease map G, we first construct a collapsed version G_0 by collapsing any species aliases corresponding to the same species into a representative node and moving any edges incident to aliases to the corresponding representative. We then proceed by comparing G_0 and G like reorganisation steps. To make our results comparable to the work of Nielsen et al. [1], we employ the same algorithm for inferring ground-truth labels. Because the algorithm has received little explicit treatment in the original publication, we motivate and describe it here in detail for clarity.

A simple approach to identify duplication parents that comes to mind is to consider nodes newly introduced in G_{t+1} and look for a subset $W \subset V(G_{t+1})$ whose neighbourhood completely covers the neighbourhood of some node in G_t, i.e. $\bigcup_{w \in W} \mathcal{N}_{t+1}(w) = \mathcal{N}_t(v)$ for some $v \in G_{t+1}$. However, this does not suffice. It is important to note that we can make no assumptions about what manipulations were made to create G_{t+1} from G_t. In particular, nodes may have been removed, added or duplicated and edges may have been added, removed or re-wired. To deal with this, instead of finding the duplicates of a given node in G_t, the algorithm of Nielsen et al. seeks to identify a possible *duplication parent* of a given node in G_{t+1}. The basic idea is that the neighbourhood of duplicates will be at least partially included in the neighbourhood of the duplication parent.

Intuitively, the algorithm works by starting at a given node $v_i \in V(G_{t+1})$, identifying its neighbour nodes in G_{t+1} and considering their neighbours in G_t. If v_i is a duplicate, we expect its duplication parent to be among these nodes.

The procedure is given in pseudocode in 13. For directed graphs, let the *positive neighbourhood* of v in G_k $\mathcal{N}_k^+(v)$ be given as $\{w \mid (v, w) \in E(G_k)\}$ and the *negative neighbourhood* $\mathcal{N}_k^-(v)$ as $\{w \mid (w, v) \in E(G_k)\}$. We abuse set notation to identify nodes solely based on their alias ID. This means that $V(G_t)$ and $V(G_{t+1})$ may potentially have nonempty intersection (indeed, the algorithm relies on it).

Line 8 states that a valid duplication parent must be reachable from both positive and negative direction. This is only relevant if there is no positive (resp. negative) neighbourhood shared between the reorganisation

Algorithm 1: Procedure to identify duplication parents. Transcribed from Nielsen et al. [1].
Data: Directed graphs G_t and G_{t+1} (reorganisation step), node $v_i \in G_{t+1}$
Result: Duplication parent of v_i or None

1 $W_+ \leftarrow \mathcal{N}_t^-(\mathcal{N}_{t+1}^+(v_i))$
2 $P_+ \leftarrow \mathcal{N}_{t+1}^-(\mathcal{N}_{t+1}^+(v_i)) \setminus W_+$
3 $W_- \leftarrow \mathcal{N}_t^+(\mathcal{N}_{t+1}^-(v_i))$
4 $P_- \leftarrow \mathcal{N}_{t+1}^+(\mathcal{N}_{t+1}^-(v_i)) \setminus W_-$
5 **if** $W_+ = \emptyset \vee W_- = \emptyset$ **then**
6 $\quad P \leftarrow P_+ \cup P_-$
7 **else**
8 $\quad P \leftarrow P_+ \cap P_-$
9 **if** $\|P\| = 1$ **then**
10 $\quad$ Let w be the single element in P
11 $\quad$ **return** w
12 **else**
13 $\quad$ **return** None

Gehört die Formel hier mitten in den Satz?

steps. This is also the case if the target node is a sink, respectively source. Line 6 takes this into account. Then, a duplication parent may still be uniquely identified if there is a single shared neighbour in the opposite direction. Line 9 handles cases when multiple candidate duplication parents exist and we cannot infer a unique single one. The algorithm is able to identify duplication parents even if edges have been removed.

Data Selection

keine Quellenangabe hier?

Like Nielsen et al., we exclude complex species aliases and nodes of degree less than two from prediction. This means they will not be considered as input examples when training or evaluating the classifier. Note that these nodes are still part of G and potentially influence the features of other nodes. Further, excluded nodes will still participate in the message-passing steps of GNN models.

If we are given a sequence of reorganisation steps and duplication events could be determined in some step, then the graph corresponding to that reorganisation step contains no nodes with positive labels. In that case, the graph is omitted from the sequence.

The set of constructed input graphs is partitioned into training and testing graphs. For the concrete choice of training and testing partitions, we explore different settings (see Section 5).

Handling contradictory examples When using reorganisation steps as training data, we have to take care to not include contradictory examples. These are data points that are very similar in features but have different ground-truth class. Consider a duplication event of node v between G_i and G_{i+1} in a given reorganisation sequence. v is assigned a positive label in G_i. However, v is assigned a negative label for any G_j for $j < i$ (if it occurs in G_j). We supply the feature vectors and labels of all graphs to the classifier for training. This is reasonable if we can assume that, in each reorganisation step from G_i to G_{i+1}, all nodes that should be duplicated are in fact being duplicated. However, in case of the *AlzPathway* reorganisation steps, this is very likely not the case. The species `Ca2+`, for instance, is not duplicated before the third step, but nothing suggests that its characteristics were different in previous steps, preventing it from being duplicated. It seems more reasonable to assume that the human curator considered and duplicated only some nodes in each reorganisation step. If a node's feature vector remains unchanged during several reorganisation steps and is not duplicated in some (negative example) steps but it is indeed duplicated in some later step (positive example), then this means that we are supplying the classification model with contradictory data. In previous

ich kenn mich ja nicht aus, aber sonst hast du immer Quellenangaben??

work, Nielsen et al. adressed this by pruning the entire training dataset of negative examples that were very similar to entries of positive examples. Note that the computation of pairwise similarities is not trivial with respect to computational cost. In this work, due to time constraints, we chose a simpler approach instead. If a node appears as a positive example for some reorganisation step, the examples corresponding to this node in previous reorganisation steps are excluded. This is based on the idea that in reality, such a node simply had not been explictily considered by the curator in previous steps.

explicitly

Feature Engineering

We are given one or several disease map diagrams and want to make a prediction for the contained species aliases whether they should be duplicated or not. Since a graph node, i.e. a species alias, is an abstract mathematical object, we need to find a representation (description) of a species alias to be used as input to the classifier. Naturally, the representation should express the characteristics of the species alias that are relevant to its class. Classifiers such as SVMs and NNs commonly expect their input in the form of a numerical vector. In the following, we define several characteristic *features* of species aliases. Such a feature can be a vector of numbers or a single number, in which case we consider it as a one-element vector. Once all features are computed, their resulting vectors are concatenated into a single *feature vector* and provided to the classifier as input.

In general, there are three aspects of a disease map based on which features can be defined. The first is the *structural* aspect in which we use graph-theoretical measures to characterise species aliases (graph nodes) based on their connectivity in the network. This was considered in detail in the feature engineering step in the work of Nielsen et al. [1] and was used in numerous approaches for metabolic networks (see Section 3).

The second aspect is *semantic*: species are annotated with biological domain knowledge, commonly in the form of links to databases that provide additional information about that species (see Section 2). In this work, we aim to explore how to exploit Gene Ontology term annotations.

The third aspect is that of *layout*: Prior to making a prediction on node duplication, the disease map may already have been laid out to some extent. If so, positions of species aliases certainly carry some meaning. Which exactly, however, is unclear since positions are most likely determined by a mixture of layout requirements, semantic arrangement and preference of the curator. For the scope of this work, we avoid using layout-based predictors since we only have one practical instance in which we have layout information and ground-truth labels available, which is when making a prediction for a AlzPathway reorganisation step and comparing it to the actual next step in the reorganisation sequence. However, this is the only disease map for which reorganisation steps are available and it is of limited size. Thus we decide to omit this approach for now, while acknowledging that considering layout information may lead to interesting approaches (see Section 7). We do consider layout information as a criterion for attaching edges to duplicates, see Section 4.3.

Structural Features The following structural features were defined and used by Nielsen et al. [1] . We re-implement them for comparability. Let $\sigma_{st}(v)$ denote the number of shortest paths from s to t passing through v. Let σ_{st} be the number of all shortest paths from s to t. Since in the following, we always consider the graph to be undirected, σ is symmetric.

- `degree`: The degree of a node, counting both incoming and outgoing edges.
- `clustering_coefficient`: The *clustering coefficient* [92] for a node v is given as

$$\frac{2\tau(v)}{\deg(v)(\deg(v)-1)}$$

where $\tau(v)$ is the number of triangles through v in the graph.

- `betweenness_centrality`: The *betweenness centrality* of v reflects the number of shortest paths that pass through v:

$$\sum_{s \neq v} \sum_{t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

- `closeness_centrality`: As used here, the *closeness centrality* for v is given by

$$\frac{\|V\| - 1}{\sum_{s \neq v \in V} d(s, v)}$$

  where $d(s, v)$ is the shortest-path distance from s to v.
- `eigenvector_centrality`: The *eigenvector* centrality is a measure of transitive importance of a node. The basic idea is that a node is important if it is linked to other important nodes. The centrality values of each node are given by the components of the principal eigenvector of the graph's adjacency matrix.

Additionally, in order to capture the characteristics of a node's direct neighbourhood, the following secondary features are computed:

- `neighbour_centrality_statistics`: For node v, this is the stacked vector of statistics over several centrality measures of neighbours of v. The statistics are mean, minimum, maximum and standard deviation. The centralities are betweenness, closeness centrality, eigenvector centrality and degree.
- `distance_set_sizes`: A vector of length k in which the k-th entry is the normalised number of nodes exactly k hops from v. Here, $k = 5$ is considered. The values are normalised by the number of nodes reachable in a grid graph via k hops.

Except for the clustering coefficient, all characteristics are also computed on the bipartite projection and included as separate features. Except for `distance_set_sizes`, all features are min-max-normalised with respect to all given training graphs.

Additionally, we consider the directed degrees (`in_degree`, `out_degree`). This is motivated by the notion that a species alias may have a different biological meaning with respect to node duplication if either in- our out-degree is higher, both are balanced or the node is a source or a sink.

Semantic Features Previous work [1] primarily used node features based on graph structure. The only feature providing domain-specific information was the one-hot encoding of species type. The *one-hot encoding* of a categorical variable with d possible values is a d-dimensional binary vector in which $d_i = 1$ if the categorical variable has value i and 0 otherwise. Possible species types are *protein*, *RNA*, *simple molecule*, *ion*, *gene*, *phenotype*, *drug*, *complex*, *degraded* and *unknown*.

Relying solely on structural features may be insufficient. For example, consider a node v of degree two whose neighbours u and w have similar structural characteristics. It may be the case that u and w appear in similar biological processes and thus v should not be duplicated. On the other hand, u and w may appear in completely unrelated processes, in which case v likely establishes false connectivity. Thus, we need to consider the biological interpretation of u and w. However, v may well also be a key connector between two pathways through u and w and, as such, should not be duplicated. Determining whether v is a key connector likely depends on the biological characteristics of the two pathways.

In the disease maps considered here, each species is annotated with a set of Gene Ontology (GO) terms. We consider the subset of the Gene Ontology that describes biological processes. We aim to assess neighbourhood heterogeneity with respect to GO term annotations. For a set of GO term annotations, we derive a fixed-length numeric *embedding* vector for each GO term that reflects its position in the GO ontology graph, and thus its semantics. We acknowledge that there is previous work developing pairwise similarity measures between GO terms (see Section 3). However, the direct, embedding-based approach has the following advantages:

- We are alleviated of the choice of a similarity measure (effectively leaving that problem to the classifier). This also means that, particularly in case of a GNN classifier, the neural network may potentially be able to capture more flexible relationships than what we would encode with some similarity measure.
- This gives us an approach for including annotation information for complex species aliases by simply combining the embeddings of contained nodes.
- Since in the end we aim to assess the heterogeneity of a node's neighbourhood, the consideration of embedding values could be incorporated in the message-passing step of a neural network (see Section 7). Further, node-level information may be useful for the task of finding an attachment of edges after duplication.

For computing an embedding vector for a given GO term, we closely follow the approach of Zhong, Kaalia, and Rajapakse [74] and encode a term's position in the GO graph using the *node2vec* algorithm.

The above considerations lead to the following concrete feature definitions:

- `GO_embedding`: The basic idea of this approach is to only provide an encoding of the term's position in the GO graph as node feature. A GNN model could then potentially capture characteristics of a target node's neighbourhood via its aggregation function.
- `GO_stddev`: We additionally try capturing neighbourhood heterogeneity by interpreting the embeddings as points in an euclidean space and deriving a measure for the spread of these points around their centroid (mean). The idea is inspired by the measure of standard deviation, i.e. the average squared distance from the mean. This can conventienly be expressed as the sum of variances over each dimension. Let $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be the embeddings of neighbour nodes, and let $\bar{\mathbf{x}}$ be the centroid (i.e. dimension-wise mean). Assume the points lie in a D-dimensional euclidean space. Let $(\mathbf{x}_i)_k$ denote the k-th entry of $\mathbf{x}_i$. We then consider $\sigma = \sqrt{\sigma^2}$ as a measure of spread, given by

$$\begin{aligned}\sigma^2 &= 1/n \sum_{i=1}^{n} \|\bar{\mathbf{x}} - \mathbf{x}_i\|_2^2 \\ &= \sum_{d=1}^{D} 1/n \sum_{i=1}^{n} \left((\bar{\mathbf{x}})_d - (\bar{\mathbf{x}_i})_d\right)^2 \\ &= \sum_{d=1}^{D} \mathrm{Var}(\left[(\mathbf{x}_1)_d, \ldots, (\mathbf{x}_n)_d\right])\end{aligned}$$

  where $\mathrm{Var}(\left[(\mathbf{x}_1)_d, \ldots, (\mathbf{x}_n)_d\right])$ is the variance along dimension d and $\|\cdot\|_2$ is the euclidean distance. Note that this approach encodes characteristics of the embeddings across a node's neighbourhood directly into a single feature vector and can thus be used with any classifier such as SVMs.

There are two cases when we are required to aggregate a set of embeddings into a single value. On the one hand, a single protein can be annotated with several GO terms. On the other hand, a complex species alias potentially contains muliple annotated aliases. In both cases, we take the mean of the embeddings associated to the GO terms.

4.2 Evaluation of classifiers

SVMs as well as GNNs are trained to optimise some specific function with respect to the training data. However, when comparing and selecting models, we need an unbiased performance measure that is based only on data and ground-truth labels.

Evaluation metrics The classifiers considered here will output a probability (or *confidence score*) that a given data point will belong to the positive class. To obtain a concrete, binary classification, we have to draw a *decision threshold* τ. If the predicted confidence score of a given data point is greater than τ, it will be assigned

the positive class, else the negative class. In Table 4.1 we introduce some basic terminology for subsets of training data based on their true and predicted class [93].

Table 4.1: Given a concrete, binary classification, the following terms describe (sizes of) subsets of the predicted data, depending on its ground-truth and predicted class.

	Predicted Positive (PP)	Predicted Negative (PN)
Actually Positive (P)	*True Positives* (TP)	*False Negatives* (FN)
Actually Negative (N)	*False Positives* (FP)	*True Negatives* (TN)

Based on the terms in Table 4.1, we define the following measures:

- *Accuracy*, given by the ratio of correctly classified examples: $\mathtt{TP+TN}/\mathtt{P+N}$. Note that for class-imbalanced data, this is not a sufficient measure, because misclassifications of the minority class will be underrepresented in the overall score.
- *True Positive Rate* (TPR), or *Recall*, given by $\mathtt{TP}/\mathtt{P}$ is the probability that a positive example will be predicted as such by the classifier.
- *False Positive Rate* (FPR), given by $\mathtt{FP}/\mathtt{N}$ is the probability that a negative example will be predicted falsely as positive.
- *Precision*, given by $\mathtt{TP}/\mathtt{PP} = 1 - \mathtt{FPR}$

A perfect classifier would yield a high True Positive Rate and a low False Positive Rate. Note that it is possible to trade-off FPR and TPR by varying the decision threshold. If the threshold is very high, only examples for which the classifier has high confidence will be actually assigned positive class. This means that the FPR will be low. However, then not all true positives may be picked up as such by the classifier, resulting in a low TPR. Lowering the decision threshold will increase TPR, but also potentially result in additionally picking up false positives, increasing the FPR.

The choice of the proper decision threshold depends on the use-case since different importance may be assigned to either Precision or Recall. The original use-case of Nielsen et al. was to use a classifier trained for predicting node duplication to provide a low number of high-confidence examples as suggestions to the user. When considering only the few examples with the highest score, Precision is more important than Recall.

To assess the trade-off between FPR and TPR with respect to possible choices of classification threshold τ, we can plot FPR and TPR as a function of τ, yielding the *Receiver Operating Characteristic* [94] (ROC) curve. The curve of a random classifier that flips an even coin for each given example would be close to the diagonal. Generally, a classifier could be considered better if its ROC curve leans towards the upper left, i.e. the area under the curve is greater. Note that the ROC curve is insensitive to class imbalance. We evaluate classifiers visually based on their ROC curves in Section 5.

As a heuristic for choosing τ, we can look for the threshold with the greatest distance between TPR and FPR at τ, i.e. $\tau_{\text{opt}} := \text{argmax}_{\tau \in (0,1)} \mathtt{TPR}(\tau) - \mathtt{FPR}(\tau)$. As an indicator for overall quality, we can compute the *Area Under Curve* (AUC) score given as

$$\text{AUC}(\tau) = 1/2(\mathtt{TPR}(\tau) - \mathtt{FPR}(\tau) + 1)$$

and define the optimal overall AUC score as $\text{AUC} := \text{AUC}(\tau_{\text{opt}})$. However, we acknowledge that reducing the performance of a classifier to a single number will hardly ever capture all characteristics and evaluation still depends heavily on the use-case.

practice? practise ist ein Verb

Training- and Validation subsets Note that, in practise, a machine learning model should be able to reliably generalise to data that was not observed during the process of finding and tuning a model. Usually, this is done by splitting the available data into three parts: one for training, one for evaluation during model tuning (such as identifying a reasonable maximum epoch like here) and, finally, a part that will only be used for validation once all decisions have been made. However, in this work we only consider a partition into two parts, a *training* and a *validation* split. This is primarily motivated by the low amount of available data. Since

we wish to evaluate generalisation performance on an unseen disease map, we cannot split disease maps internally on an unseen disease map, we cannot split disease maps internally. In the future, potentially, once a single, promising model has been identified and all decisions have been made, there should be further validation on completely unseen data.

Neural Networks are trained iteratively. It is a common phenomenon that with increasing number of training epochs, the model will be fit more and more tightly to the training dataset, sacrificing generalisation ability to the validation dataset. This is referred to as *overfitting*. Thus, the choice of the point at which to stop training the network potentially has an impact on generalisation performance. In this work, for performance evaluation, we consider the model state at the training epoch that performs best on the validation split. Note that this information is not available in a real-world use case. However, based on the results in Section 5.3, a maximum epoch can be inferred heuristically (see Section 5.3).

4.3 Attachment of Edges

Assume we are given a binary classifier that decides whether a node should be duplicated. If a node v is eligible for duplication, we next need to determine how many duplicates to introduce and how to distribute edges to and from v across the duplicates. Formally, we aim to find a partition of the neighbourhood of v. Based on the intuition that a good duplication is one that reduces the heterogeneity of the neighbourhood, we can characterise the partitions as *clusters* in the sense that intra-cluster distances are smaller than inter-cluster distances.

The choice of distance metric is open. Of particular interest are metrics that reflect semantic similarity of attached GO terms (see Section 3.5). However, because in the given datasets, most aliases are annotated with a relatively large number of GO terms and exploiting these annotations for classification was problematic (see Section 5.9), we opt for a simpler approach first and leave this open to future work. Instead, we consider the layout positions of aliases and their euclidean distances. Note that this approach is only applicable if layout information is actually given, i.e. this approach can not be used if we construct a collapsed graph from a single given disease map (unless we would infer layout information for the newly constructed, collapsed graph).

Particularly when considering euclidean distances in the layout, a suitable clustering algorithm needs to handle outliers in a sensible manner. Additionally, we do not know the number of clusters in advance. Further, the algorithm should be able to handle non-convex clusters. This eliminates basic partition-based methods such as k-Means and extensions. Further, we seek to assign all points to a cluster, i.e. we do not want to exclude any points as noise. Also, since different node neighbourhoods have potentially different scales, we aim to avoid having to specify hyperparameters like distance thresholds as they are used in density-based clustering algorithms like DBSCAN or Optics. A family of clustering algorithms that seems well suited is that of *agglomerative* clustering: Initially, each point is assigned its own cluster. Iteratively, the two clusters with the smallest inter-cluster-distance are merged until only a single cluster remains. This yiels a hierarchical clustering tree, also called *dendrogram*, as depicted in Figure 5.11.

yields?

There are several canonical choices of distance measures between two clusters C_1 and C_2. The most simple ones employed for agglomerative clustering are:

$$\begin{aligned}
\textit{single linkage:}\ & d(C_1, C_2) = \min_{p \in C_1, q \in C_2} d(p, q) \\
\textit{complete linkage:}\ & d(C_1, C_2) = \max_{p \in C_1, q \in C_2} d(p, q) \\
\textit{centroid linkage:}\ & d(C_1, C_2) = d(\text{mean}(C_1), \text{mean}(C_2))
\end{aligned}$$

where C_1 and C_2 are considered to be sets of points. We use single linkage since complete and centroid linkage are strongly affected by large in-cluster variances and do not work well with non-convex clusters.

Setting a threshold value on the maximum dissimilarity inside a cluster, we obtain a concrete clustering. This can be thought of as "cutting off" the dendrogram at a specific height. Note that here we do not have to specify the number of clusters but instead a threshold dissimilarity. This threshold can be determined automatically via a heuristic: We look for the strongest increase in the distance to the next closest cluster before each merge step. Formally, let $d = (d_1, \ldots, d_k)$ be the monotonically increasing sequence of inter-cluster distances at which a merge occured. The first discrete derivative d' of this sequence gives the step sizes while the second derivative d'' describes the step sizes between step sizes, that is, the change rate. The index of the maximum in d'' yields the number of clusters. Since a single point in a discrete derivative is based on two original points, we require $k \geq 8$ points for determining at least two values in the second discrete derivative. In case of $k < 8$, we fall back to the first derivative (step size). Examples show that this is a good approximation for a low number of points. As special cases, since the decision to duplicate is assumed to be already given by the classifier, we exclude the possibility of returning only a single cluster. If $\left\|\mathcal{N}(v)\right\| = 2$, then we always trivially split.

Note that thus we characterise the procedure to identify the number of clusters independently of the concrete scale of the data. However, we can see that this procedure strugges if intra-cluster variances are diverse an the internal variance of a cluster is close to another inter-cluster distance (see Figure 5.11). We accept this disadvantage for now and hypothesize it has little practical impact in this use-case.

struggles?

4.4 Implementation

The experiments in this work were realised mainly using Python. For deep learning on graphs, we use the library *PyTorch Geometric* [95] which provides implementations for most popular message-passing layers. It is based on the deep learning framework *PyTorch* [96]. We use the SVM implementation provided by *scikit-learn* [97].

Data preprocessing such as loading from SBML files, constructing the graph and identifying duplicates was mostly done in standard Python using core libraries. Specialised Python libraries for working with SBML data do exist. The most prominent examples are *libsbml* [98] and, building on top of it, *cobrapy* [99]. However, most of the datasets considered in this work are given in an extension of SBML as defined by the tool *CellDesigner*. The *libsbml* library does not support this extension.

The pipeline of preprocessing and model training is based on *GraphGym* [100], an open-source project that offers a modular pipeline for deep learning on graphs. It was initially developed for exploring the design space of GNNs [101]. We extend *GraphGym* to fit our needs. Some of the main extensions are

- Having different graph interpretations of a disease map (simple, bipartite, directed, undirected, ...) available simultaneously when needed, but only computing them when actually required for the current experiment.
- Handling and converting between different graph data structures such as those provided by *networkx*, *DeepSNAP* (used internally by *GraphGym*) and *igraph* (used for fast algorithms on graphs).
- Avoid splitting provided graphs internally but instead make it possible to train on some and evaluate on other (whole) graphs.
- Caching feature computations so they do not have to be recomputed on each experiment run.

In general, one of the main challenges in preprocessing was to handle tabular data (node features and labels) and graph-structured data at the same time. Moreover, there are several instances of tabular and graph-structured data for a single entitity. For example, for computing structural features based on both simple and bipartite graph, we have to construct and handle both graph representations. The result is several individual features for each node which finally have to be combined into one final feature vector.

For the computation of graph-structural features, we used either *networkx* [102] or the Python frontend of *igraph* [103]. *networkx* provides Python implementations of most common network analysis methods and is already tightly integrated into *GraphGym*. However, for the datasets considered herein, *networkx* implementations would require unfeasible computation time. *igraph* merely provides bindings in Python

while the core algorithms are implemented in `C`. The speedup is particularly relevant for characteristics of nontrivial complexity such as those relying on global information like, for example, closeness centrality.

GO term annotations were extracted and handled using the *KNIME Analytics Platform* [104] (KAP) as well as Python with libraries such as *pandas* [105] and *numpy* [106]. For this use-case, KAP has the advantage that a pipeline can be constructed visually and intermediate results can be evaluated and displayed at any step, facilitating incremental development and data exploration. This is particularly useful if the data has to be re-shaped several times as was the case here. KAP is primarily suited for tabular data and support for network-structured data is only rudimentary. For the computation of embedding vectors, we used the *node2vec* implementation provided by *PyTorch Geometric*.

The given *CellDesigner*-SBML files are essentially large XML files. Some common text or XML editors may struggle with large file sizes. Tools that have proven particularly useful were *XMLExplorer* [**_XmlExplorer_2021**] and *GNU Emacs* [107].

We deliberately refrained from using Jupyter Notebooks [108], which seem to be very common in the fields of machine learning and data science. This decision is motivated by several factors. First, the notebook approach makes it much harder to apply a debugger, which was needed here since the *GraphGym* framework had to be customised for our needs. Second, notebooks lack development ergonomics such as advanced keybindings and code navigation as it is provided by modern IDEs such as *PyCharm*. Finally, the key idea of notebooks is to keep code and output next to each other. For the scope of this project, this would have proven issues with code encapsulation, performance, analysis of results and version control. Visualisations were facilitated with *matplotlib* [109]. This means that any plots could be generated dynamically from the resulting data.

5 Experiments & Results

We motivate and describe the experiments that were carried out and discuss their results. We begin with some preliminary models identified via basic hyperparameter search. Each subsequent section is based on the best-performing model of the previous section. A discussion from a general viewpoint follows in Section 6. The applied methods and features are described in detail in Section 4.

5.1 Datasets used for training and evaluation

Since disease maps are manually created, the availability of diagrams is limited. Here, we focus on data already considered in previous work [1]. We describe the used disease maps and provide a visual overview in Figure 5.1.

The *AlzPathway* map describes signaling pathways related to Alzheimer's Disease. Since its initial publication [110], it has received several updates and further analyses [111, 61, 64]. The map has received additional curation focussing on increasing readability by means of reorganising existing network elements [112]. This includes the duplication of some species aliases. During curation, snapshots of intermediate progress (*reorganisation steps*) were saved. Note that the reorganisation steps are not atomic. Each step includes modifications to several nodes and edges. This sequence of reorganisation steps served as the basis for the work of Nielsen et al. to train an SVM classifier to predict node duplication [1]. In this work, we consider these reorganisaton steps for training data (AlzPathwayReorg). Note that from the sequence of reorganisation steps, we exclude the steps that do not correspond to any duplication events. Further, we consider the last of the reorganisation steps, i.e. the final result as a single, independent map (AlzPathwayLast). For both reorganisation steps and fully-curated map, we use the dataset published by Ostaszewski [112] at 2021-06-25.

The *PDMap* [62] describes the major pathways involved in the pathogenesis of Parkinson's Disease. The map can be explored via a hosted *Minerva* instance, reachable at `https://pdmap.uni.lu/minerva`. For this work, we consider the version of this map as exported from *Minerva* at 2021-09-07.

ReconMap [14] is a visual representation of the genome-scale metabolic model *Recon 2* [113] that aims to comprehensively model the human metabolism. It can be viewed at `https://vmh.life/minerva`. We use version `2.01` of 2015-11-20.

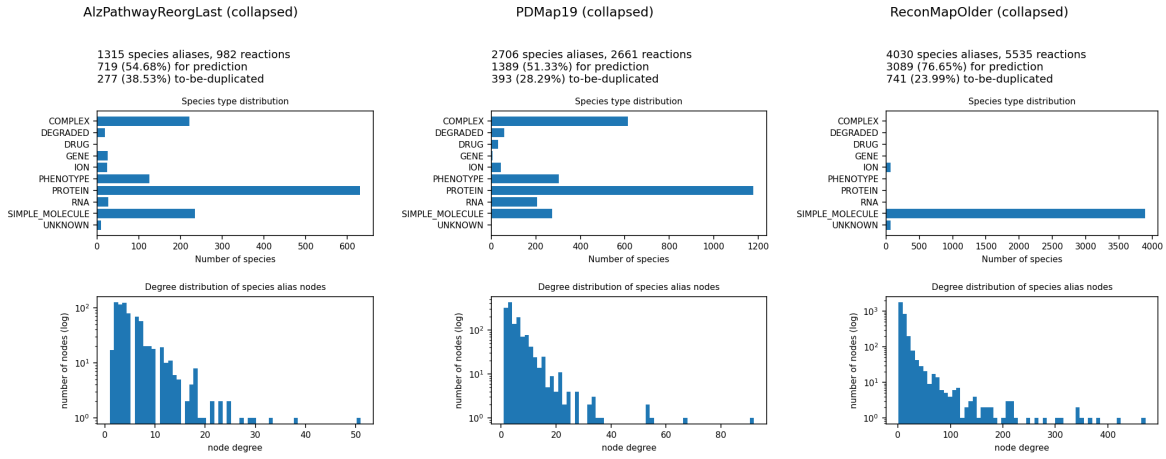


Figure 5.1: An overview of characteristics of the *collapsed* networks used for training: AlzPathwayLast, PDMap and ReconMap. The count of species aliases and reactions is effectively the count of the bipartite node sets in the constructed graphs. Networks and labels are determined as described in Section 4.1. Note that the y-axis of the node degree histogram is plotted in logarithmic scale.

In experiments, we will consider different sets of node features as input to the classifier. We outline these *feature sets* in Table 5.1. The set `basic-both` is the same set of features as used in previous work by Nielsen et al., with the exception of omitting one particular feature. For a node v, the feature would contain statistics on centrality scores of v's neighbours in the ego graph centered on v of radius k for $k \in \{3, 5\}$. This feature was not included in preliminary experiments due to computational cost. This somewhat impacts comparability to the previous work. However, note that we achieve the same performance even if not using this feature. This suggests that this feature is not essential here.

Table 5.1: Overview over the different feature sets used as input to a classification model in this work. The individual features are defined in Section 4.1.

	basic-both	basic-projection	basic-simple	degrees	degrees-basic
betweenness_centrality	✓	-	✓	-	-
...projection	✓	✓	-	-	-
closeness_centrality	✓	-	✓	-	-
...projection	✓	✓	-	-	-
eigenvector_centrality	✓	-	✓	-	-
...projection	✓	✓	-	-	-
neighbour_centrality_stat.	✓	-	✓	-	-
...projection	✓	✓	-	-	-
distance_set_size	✓	✓	✓	-	-
clustering_coefficient	✓	✓	✓	-	-
node_class_onehot	✓	✓	✓	-	-
node_degree	✓	-	✓	✓	✓
...projection	✓	✓	-	✓	✓
node_in_degree	-	-	-	✓	-
node_out_degree	-	-	-	✓	-

5.2 Basic hyperparameter search

Neural networks as well as Support Vector Machines rely on the choice of particular hyperparameters. The optimal choice is not immediately clear. A common strategy is to search the hyperparameter space by training and evaluating a candidate model for different combinations of hyperparameters. There are different search techniques of varying complexity such as Bayesian Optimisation [114] or Random Search [115]. Arguably the simplest approach is Grid Search, in which the search space is sampled by a regular grid of points. In this work, we use grid search for sake of simplicity.

The models were trained on the AlzPathway reorganisation steps (AlzPathwayReorg) and evaluated on the Parkinson's disease map (PDMap). The considered features in this and subsequent experiments are, unless stated otherwise, based on the work of Nielsen et al. (see `basic-both` in Table 5.1). We select the best-performing model with respect to AUC score.

Support Vector Machine

For the choice of kernel function, we pick the RBF kernel (see Equation 2.12) as a heuristic choice since Nielsen et al. showed that a generally best-performing kernel cannot easily be determined and that the RBF kernel generally achieves good performance. Because the source code of previous work by Nielsen et al. is not publicly available, we are likely using a different SVM implementation. Different implementations may interpret parameters slightly differently. Thus, we perform grid search over a similar range of values to determine a best-performing set of hyperparameters.

The results are given in Table 5.2. The choices of `C`, γ and weight roughly agree with the results of Nielsen et al..

Table 5.2: Considered SVM hyperparameters, value range searched via grid search and best identified combination of values. The weight hyperparameter specifies the weight of the minority class. Here, this is the positive class. It corresponds to the value C_P in Equation 2.11.

Hyperparameter	*Searched range*	*Choice*
Cost (C)	2^{-9} to 2^{3}	0.5
Gamma (γ)	2^{-3} to 2^{9}	0.1
Weight	`[1,2,3,5,7,10]`	3

Graph Neural Network

The range of searched hyperparameters is based on the work of You, Ying, and Leskovec, who systematically evaluated choices of hyperparameters across various tasks, including node classification [101]. Based on their results, they suggest a constrained design space for Graph Neural Networks for node classification.

Table 5.3: Considered hyperparameters for the GNN models. In case there are multiple possible values, the best hyperparameter combination is given in the third column.

Hyperparameter	*Searched range*	*Choice*
Dropout	`[0.0, 0.1,0.2,0.4]`	`0.0`
Aggregation function	`[add, mean, max]`	`add`
Fully-connected layers before message-passing	`[1,2]`	`2`
Fully-connected layers after message-passing	`[2,3]`	`2`
Message-passing layers	`[2,4,6,8]`	`2`
Connectivity	`[skip_sum, skip_cat]`	`skip_sum`
Activation function σ	`[PReLU]`	
Use batch normalisation?	`[yes]`	
Learning rate η	`[0.01]`	
Optimiser	`[adam]`	

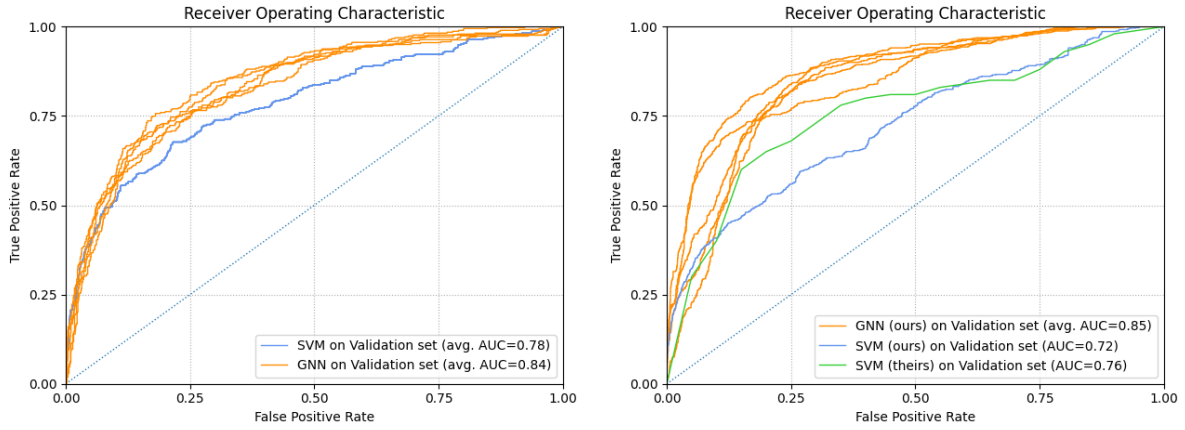
We use a hidden layer size of 256 based on previous manual experiments. For optimisation, we use the Adam optimiser [116].

The identified hyperparameters are not particularly surprising. One thing that may be of interest here is the low number of message-passing layers. There is no general rule of thumb since the proper number of layers may differ between tasks [101]. Here, the lowest considered value was deemed to perform best. Indeed, we show that, in our experiments, message-passing does not have a substantial effect at all Section 5.6. Dropout not providing a large advantage has already been hinted at in previous work [101].

5.3 Reproducing previous work & Comparison to GNN

In this section, we reproduce the original task from Nielsen et al. using an SVM classifier. Additionally, we introduce a simple GNN classifier based on the GCN layer defined in Section 2.7 and the hyperparameters identified in Section 5.2. The GNN model operates on the same input data as the SVM, using the bipartite projection of the disease map graph for message-passing (see Section 4.1). Further variants and extensions of the GNN model will be evaluated in the following sections.

Results & Discussion First, we check whether we can reproduce the results of Nielsen et al.. Note that no source code of their work was available. This means there may still be unattributed differences in data processing. We compare to the results provided in their written publication [1]. In the use-case of providing few high-confidence examples as suggestions to a human curator, the high-confidence predictions of the classifier are particularly interesting. This can be assessed visually by focussing on the left lower part of the ROC curve. Ideally, the high-confidence suggestions involve only true positives (high TPR) and no false positives (low FPR). Nielsen et al. report FPR values at specific TPR cutoffs.



(a) (AlzPathwayReorg → PDMap)

(b) (AlzPathwayReorg → ReconMap)

Figure 5.2: Direct comparison of our SVM and GNN classifiers. For multiple runs, we report the average AUC score in the legend.

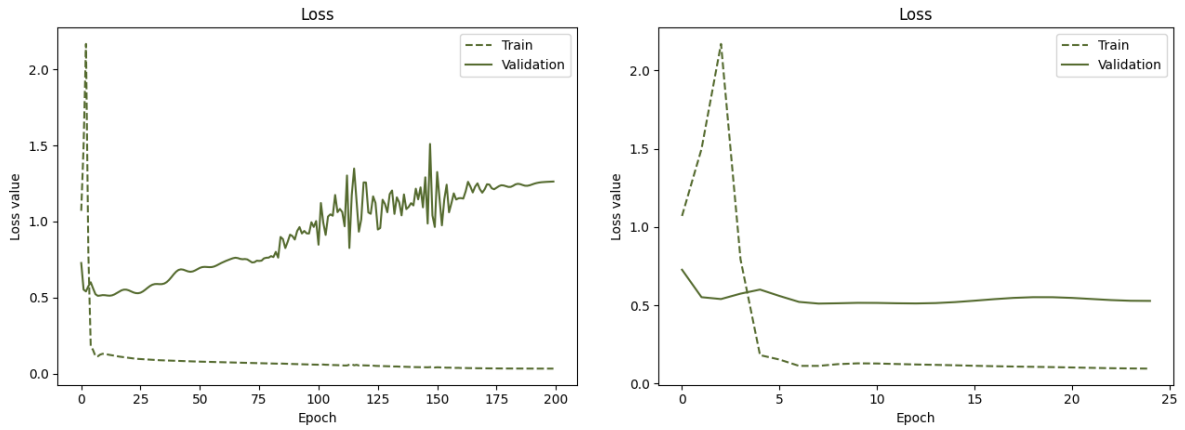


Figure 5.3: Total loss value at each training epoch of the GNN model. The right-hand-side plot a focussed view on the same data.

For evaluation on PDMap, Nielsen et al. provide no evaluation with respect to a variable classification threshold. However, they do list true and false positive rates, allowing us to calculate an AUC score. Although not explicitly stated in the paper, we assume that these rates correspond to a decision threshold that is optimal in the sense of Section 4.2. This yields an AUC score of 0.69. Our SVM model reports an AUC of 0.77. For comparison, our GNN model achieves an AUC score of 0.86.

For evaluation on ReconMap, we directly compare to the results provided by Nielsen et al. in their publication. The data is illustrated in Figure 5.2 and Figure 5.5, denoted as "theirs". The values suggest that our SVM model performs worse particularly for lower decision thresholds. This may be due to different preprocessing of the data. Namely, we handled the cleaning of contradictory examples from reorganisation steps differently (see Section 4.1). However, we show in later experiments (see Section 5.4) that we are able to find a different, simpler approach that matches the evaluation performance of Nielsen et al..

As a first sanity check for the GNN approach, we verify whether the optimisation procedure is able to overfit the model on the training data. Indeed, the overall loss value converges to a stable minimum and the AUC score for evaluation on the training set at the last epoch is 0.989, indicating a near-perfect fit (not shown in figures). For comparison, our SVM model achieves an AUC of 0.89 on the training set.

Since training a neural network ultimately means optimising the overall loss value, it is important to inspect the development of the optimisation. Ideally, the overall loss value should decrease smoothly with each training epoch and converge to a minimum. If the loss development is overly erratic, the optimisation procedure may fail to reach a local minumum. Note how, for both PDMap and ReconMap as evaluation

datasets, the validation loss reaches its minimum after just a handful of training epochs. We additionally ran the same experiment with a decreased learning rate (from 0.01 to 0.001). There, we can see a smoother decrease of the loss value, but also note that the overall achieved minimum is slightly worse than in case of a greater learning rate (see Figure 5.4). This means that the model training makes huge progress on optimising the loss (as evaluated on the training data) in the first few epochs. This progress is then also useful to some extent for generalisation to the validation data. Further, training only improves training loss by small amounts and the validation loss increases, the model optimisation is overfitting.

The initialisation of neural network weights before the first training epoch depends on random values. To check whether our training procedure is robust with respect to random initialisation, we repeat each run several times with different seed values for the random number generators. The separate runs are illustrated as multiple lines in Figure 5.2. There is noticeable variability in classification behaviour, particularly when evaluated on ReconMap. We note that for a real-world application of this model, this should be adressed. However, due to the different behaviour depending on the choice of evaluation dataset, we hypothesize that this effect may primarily be due to the quality of the training data. This idea is supported by the development of the overall loss value during optimisation. We accept this limitation for the time being. Note that successive experiments are based on the same random seeds, thus results between experiments are comparable with respect to random initialisation.

We proceed to compare the performance of our SVM and GNN models with respect to variable classification threshold. The results are illustrated in Figure 5.2. For both evaluation on PDMap and on ReconMap, the GNN's ROC curve mostly dominates the SVM's curve. This means that for a given FPR deemed acceptable, the GNN model will provide a higher ratio of true positives. This advantage grows as the decision threshold is lowered, i.e. the GNN model is better particularly when we aim to identify more than just the high-confidence examples. The difference between ROC curves is only slight on evaluation on PDMap but strong on evaluation on ReconMap. This is probably due to the quality of training data since this difference disappears when considering a slightly different set of training data (see Section 5.4).

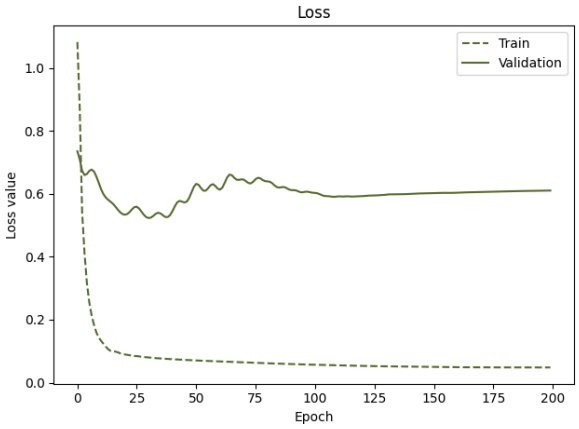
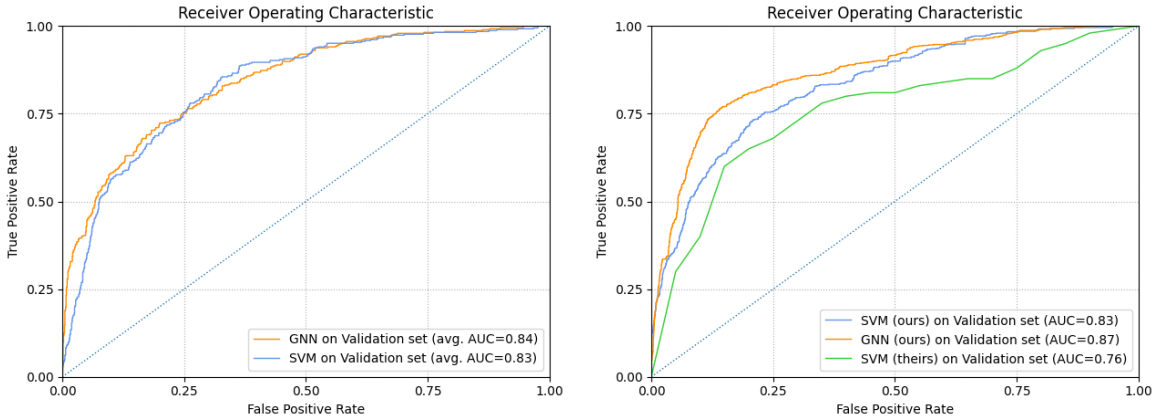


Figure 5.4: Total loss development with decreased learning rate. The plot shows the total loss value at each training epoch of the GNN model, evaluated on ReconMap.

5.4 Importance of Reorganisation Steps

Recall that in case of the Alzheimer's disease map, we are given a sequence of reorganisation steps. The original approach of Nielsen et al. was to train a model on the entire sequence of reorganisation steps such that the model would mimic the actions of a human curator as closely as possible. In fact, they additionally added an initial step corresponding to the collapsed version of the first step in the reorganisation sequence. They would then evaluate the model on single, *collapsed* versions of other disease maps. A fully collapsed map potentially has vastly different characteristics than a partially curated map. For instance, a collapsed map is likely to contain species aliases of very high degree (maybe those that appeared often as independent

aliases before collapse). In a reorganisation step, on the other hand, the map will already be in part curated and some nodes already duplicated. So, including reorganisation steps in the training data (as we did in the previous section) may actually not be beneficial because that data provides training examples for a task that is slightly different to the evaluation task. This consideration is also relevant if we consider a slightly different use-case: one of the first steps in the construction of a disease map may be the semi-automatic concatenation of different pathways, during which species appearing in several pathways will be mapped to the same species alias. This means there will be exactly one alias per species. Such a map will very closely resemble a collapsed map in the sense we construct it here. Further, approach circumvents the problem of contradictory examples introduced through reorganisation steps as described in Section 4.1. This approach has an additional practical advantage, since, in practise, reorganisation steps may rarely be available. A collapsed version, on the other hand, can be constructed from any published disease map. Thus, we deem it interesting to explore how a model will perform if trained not on the reorganisation steps but simply the collapsed version of the final map.



(a) (AlzPathwayLast⟶ PDMap)

(b) (AlzPathwayLast⟶ ReconMap)

Figure 5.5: ROC curves of models trained on collapsed version of the *AlzPathway* map (not the reorganisation steps).

Results & Discussion The results are illustrated in Figure 5.5. Note how both SVM and GNN models do not degrade noticeably in performance (compare to Figure 5.2). In fact, the SVM model performs *better* when trained only on the collapsed map. The SVM model now slightly exceeds the performance of the approach identified by Nielsen et al.. This means we effectively have at hand a simpler preprocessing approach, requiring only the final curation result instead of individual reorganisation steps that yields the same performance. Again, we need to carefully consider the use-case and what we actually train the classifier for. Collapsed disease maps are potentially different from partially curated disease maps. Herein, we select a model based on performance on collapsed disease maps. This is different from the intuitive use-case of Nielsen et al. in which the classifier will be applied to partially curated disease maps during manual reorganisation.

The results may be due to two reasons: First, not explicitly considering reorganisation steps circumvents the problem of contradictory examples described in Section 4.1. Although we do adress this issue with a simple approach, it is unlikely that this problem is solved completely. Second, this may be due to that node characteristics in later reorganisation steps become different from typical node characteristics in a collapsed map. For evaluation, we are only considering node features from a collapsed map. As such, additionally including reorganisation steps may not provide any training data that is actually useful for the evaluation task. Indeed, providing training data from reorganisation steps is detrimental to model performance, likely because this data is not well related to evaluation task.

5.5 Handling unbalanced classes in GNNs

As can be seen from Figure 5.1, we can expect to deal with unbalanced data. Class imbalance is already adressed in the SVM model of previous experiments (see Section 5.2) by means of providing class weights. We aim to assess the effect of class imbalance on the GNN model. A possible concern is that the GNN model may become biased towards predicting the majority class during training since it is simply more likely to occur when evaluating the empirical risk. We consider two simple ways to approach this problem. One is to modify the training data such that it is balanced. Another is to instruct the classifier to give more importance to examples from the minority class.

Undersampling The dataset can be made balanced either by *oversampling* the minory class (coming up with new examples for the minority class) or by *undersampling* the majority class (dropping examples from the majority class) of the majority class from the training data. For oversampling, one may simply duplicate existing examples, or generate synthetic new examples in a more advanced manner. Creating synthetic examples for graph-structured data comes with additional requirements since, contrary to conventional methods, not only a feature vector but also graph adjacency has to be inferred [117]. Although undersampling will decrease the size of the dataset even further, we stick to this approach for sake of simplicity. We undersample the majority class by considering only a random subset for prediction. Note that excluded nodes are not removed from the graph structure (see Section 4.1). We undersample the majority class to contain the same number of examples as the minority class.

Weights In case of neural networks, we can introduce an additional coefficient to the loss function that will determine the weight of prediction outcome of the positive class. We consider the *Weighted Binary Cross Entropy* loss as an extension of Equation 2.7, given by

$$\mathcal{L}_{\text{BCE weighted}} = {}^{1}/_{n} \sum_{i=1}^{n} w_i y_i \log(\pi_i) + (1 - y_i) \log(1 - \pi_i) \tag{5.1}$$

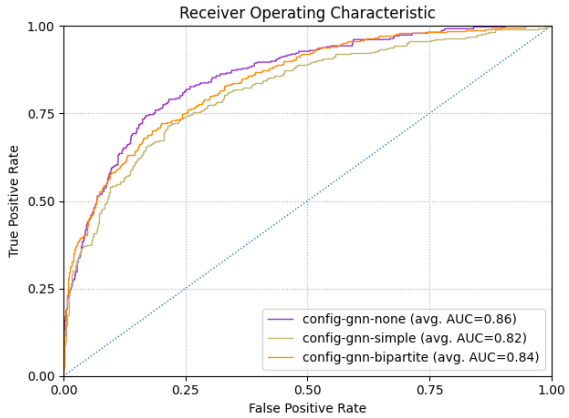
We set the weight of the minority class to ${}^{n_{\text{maj}}}/_{n_{\text{min}}}$ where n_{maj} is the number of examples in the majority class and n_{min} the number of examples in the minority class. In other words, if every example is counted according to its weight, the class counts are balanced.

Results & Discussion Experimental results show that neither undersampling nor custom class weights have substantial effect on performance of the GNN model (AlzPathwayLast→ PDMap: AUC= 0.83). We omit visualisations here due to space constraints. For sake of simplicity, we omit undersampling or weights for the GNN model in future experiments.

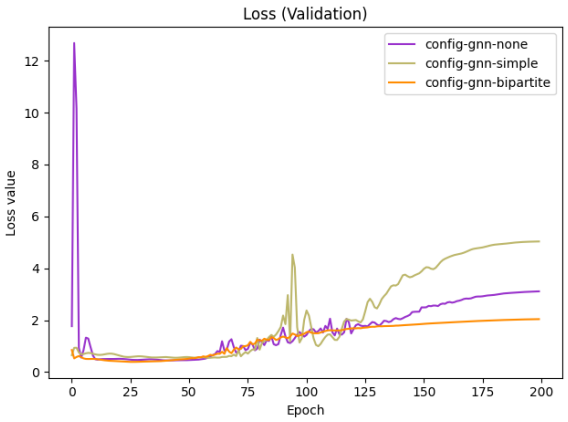
5.6 Importance of Message-Passing

We explore whether the message-passing layers of the GNN model actually provide an advantage over a neural network consisting simply of fully-connected layers. Further, we aim to compare which graph structure serves better for message-passing: We can interpret the given bipartite graph of species aliases and reactions as a simple graph, or we can consider its bipartite projection as in previous experiments.

To assess the importance of using message-passing layers at all, we compare to a model that has any message-passing layers replaced with fully-connected layers. Note that for message-passing on the simple graph interpretation, of the structural features described in Section 4.1, we can only consider those computed on the simple graph.



(a) ROC Curves for NN models with and without message-passing layers.



(b) Validation loss per training epoch for different model variants.

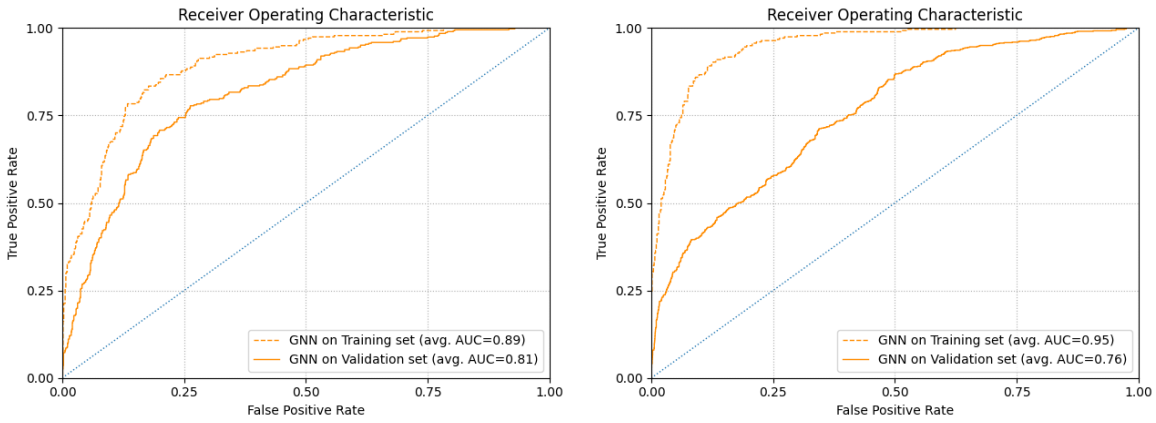
Figure 5.6: (AlzPathwayLast→ PDMap) Comparison of performance of NN models with and without message-passing layers. `config-gnn-none` has fully-connected layers instead of message-passing layers. `config-gnn-bipartite` is the same model as considered in previous experiments.

Results & Discussion Results are summarised in Figure 5.6. It is evident that on this dataset, using these features, message-passing does not provide a big advantage in overall performance. However, note that the development of the overall validation loss reaches a lower value sooner than in the fully-connected variant. With more training epochs, for all models the loss value eventually begins to fluctuate. This development is much smoother if message-passing on the bipartite graph is used. Further, the results on the choice of graph interpretation are probably not meaningful since the message-passing itself does not make a real difference.

5.7 Attention Mechanism

In previous experiments, we used a rather simple message-passing scheme. More sophisticated approaches are potentially able to capture more complex patterns, or better discern relevant from irrelevant characteristics. To this end, we explore the usefulness of another flavour of message passing, namely *Graph Attention* as defined in Equation 2.5. Moreover, the score associated with each edge may allow further interpretation.

Results & Discussion The results are illustrated in Figure 5.7. In comparison to Figure 5.6 and Figure 5.5, we seem to not gain any advantage. This and the previous results suggest that neither using more complex models nor adressing class imbalance actually improves performance. We hypothesize that, in order to improve classification performance, we have to consider the quality of the data we present to the classifier in the first place. This involves the predictive quality of the actual input features we provide to the classifier, as well as the quality of the ground-truth labels.



(a) (AlzPathwayLast→ PDMap)

(b) (AlzPathwayLast→ ReconMap)

Figure 5.7: Comparison of classification performance for GNN models using attention. Dashed lines describe performance as evaluated on the training dataset according to the model state that performs best on the evaluation dataset.

5.8 Feature Selection

In their initial publication [1], Nielsen et al. used a set of features mainly based on different node centrality measures, as well as statistics on the centralities of a node's neighbours. They remark that elaborating the usefulness of different features is left open to future work. In this section, we aim to explore this direction. While there are sophisticated approaches to automatic feature selection [118], for sake of simplicity we simply train and compare models on different subsets of the available features.

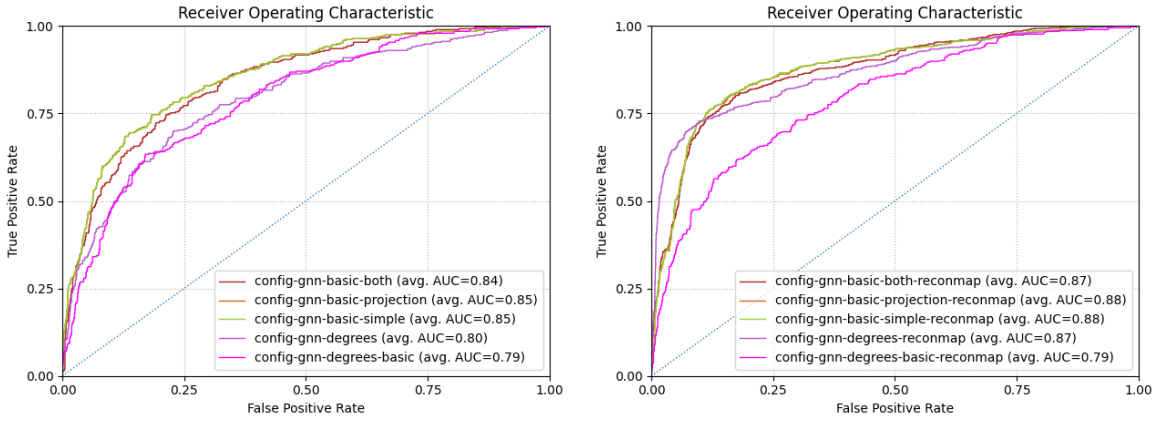
For each of the structural features defined in Section 4.1, Nielsen et al. provide two variants: one computed on the simple graph interpretation and one on the graph's bipartite projection. We aim to assess if indeed both variants are needed or if one alone works better. Further, we explore the predictive value of information on node degrees. This is interesting because node degree has been used as a heuristic for classifying currency metabolites in the context of the analysis of metabolic models (see Section 3). The basic idea is that if a node's degree is particularly large, it is likely that the connectivity it implies between its neighbours is not semantically meaningful.

Beyond the undirected degree, we additionally consider in- and out-degree. This is interesting, because directed degrees may have a biological interpretation: A species that occurs with high out-degree (resp. in-degree) may mainly appear as substrate or enabling factor (resp. product or result). A node with balanced in- and out-degree on the other hand may rather represent a central step or even a key connector.

Results & Discussion Results are given in Figure 5.8. The used feature sets are elaborated in Table 5.1. `basic-both` represents the feature set considered in earlier experiments (similar to the one considered by Nielsen et al.) and acts as a baseline to compare against.

The results indicate that it makes little difference whether structural features based on the bipartite projection, the simple graph interpretation or both are used. Using both variants even seems to slightly hurt model performance. This is potentially because then the classifier has to deal with input of much higher dimensionality but little more useful information. It seems it makes little difference which variant is used. The ROC curves for the simple and the bipartite projection variants almost coincide except for a handful of predictions.

Most interestingly, we can see that the node degree alone is already a strong indicator for node duplication. Further, the different behaviour between evaluation on PDMap and on ReconMap is striking. For evaluation on PDMap, using only degrees as features provides only a relatively small disadvantage. On ReconMap, however, using undirected degrees yields a significantly worse performance, while considering the directed



(a) (AlzPathwayLast⟶ PDMap)

(b) (AlzPathwayLast⟶ ReconMap)

Figure 5.8: Comparison of performance with different feature sets. `config-gnn-basic-both` is the feature set considered in earlier experiments and its curve shows the same data as in Figure 5.5.

degree actually matches the AUC score of the models using the full feature sets. This hints at that PDMap and ReconMap show different characteristics in network structure. We discuss this further in Section 6.

5.9 Gene Ontology Annotations

Our general intuition is to design feature representations that describe the heterogeneity of a node's neighbourhood. If a node's neighbourhood is highly hetereogeneous, that is, subsets of the neighbourhood are vastly different, we conclude that the node establishes false connectivity and should thus be duplicated. We hypothesize that a curator's decision on duplication is not based merely on structural features but also on the biological meaning of the affected processes. We seek to encode the biological processes a species is involved in. To this end, we use the Gene Ontology (GO) annotations that are provided with the disease maps considered in this work. This approach is motivated in detail in Section 4.1.

To obtain representative embeddings based on GO terms, we perform the following steps:

1. **Extract Annotations**. For each species, we extracted identifiers from the *CellDesigner*-SBML files. For *AlzPathway*, these were UniProt identifiers (as done in [89]) , for *PDMap*, we used Entrez Gene IDs. For each identifier, we obtain the associated GO terms by querying the mygene.info web service [119]. Each GO term annotation additionally comes with an *evidence code* that describes how well the annotation is supported. We allowed only some evidence codes, based on the work of Ruiz, Zitnik, and Leskovec [2]. The list of allowed evidence codes can be found in Table 5.4. The vast majority of evidence codes in both cases was either `IDA` or `IMP`. Additionally, each term annotation additionally comes with a *qualifier* that describes the relationship between the species and the term. In the vast majority of cases here, this is simply `involved_in`. We explicitly disallow `NOT involved_in`. We only consider annotations for the *Biological Process* subtree of the GO graph.
2. **Obtain GO/BP graph**. We obtain the GO ontology graph from geneontology.org (`go-basic.obo`). We consider only the *Biological Process* subtree (as identified by its root node `GO:0008150`). We consider the graph to be undirected.
3. **Obtain Embeddings for Terms**. We compute node embeddings using the `node2vec` implementation supplied by *PyTorch Geometric* with am embedding dimension of 128, walk length of 80, context size of 10 and 10 walks per node. We picked $p = 1$ and $q = 0.5$ s.t. embeddings will reflect the local positions in the graph. The optimisation was done with a learning rate of 0.01.
4. **Attach Embeddings to Graph**. For each species alias in the graph, we obtain corresponding embeddings via its species ID. In case a species is annotated with multiple GO terms, or in case of complex species aliases, we aggregate multiple embeddings by taking their mean. Of the given disease map, we only consider the induced subgraph for which embeddings could be obtained, since all nodes are required to have features for message-passing. Further, we exclude nodes with more than 40 term annotations. This threshold was chosen heuristically based on the distribution on PDMap (see Figure 5.9).

Table 5.4: Allowed evidence codes considered in the experiment of Section 5.9.

Evidence code	Description
`EXP`	Inferred from Experiment
`IDA`	Inferred from Direct Assay
`IMP`	Inferred from Mutant Phenotype
`IGI`	Inferred from Genetic Interaction
`HTP`	Inferred from High Throughput Experiment
`HDA`	Inferred from High Throughput Direct Assay
`HMP`	Inferred from High Throughput Mutant Phenotype
`HGI`	Inferred from High Throughput Genetic Interaction

We check the following combinations of features (see Table 5.1):

- As a `baseline` we use the same feature set as used in Section 5.4, i.e. `basic-both`.
- To assess the predictive value of the GO-based features in isolation, we consider `constant-stddev` and `constant-embed`: The features `GO_stddev` and `GO_embedding` as defined in Section 4.1 (the identifier "constant" comes from the fact that we additionally supply a meaningless single constant number as feature due to implementation purposes).
- Additionally, we assess the GO-based features in combination with information on node degrees: `degree-stddev` and `degree-embed` consist of the `degrees` feature set, plus `GO_stddev` or `GO_embedding`.

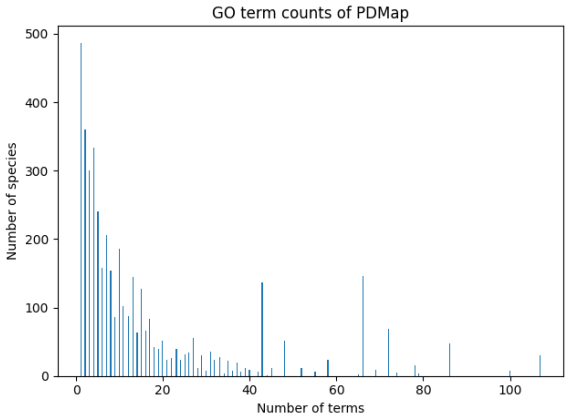
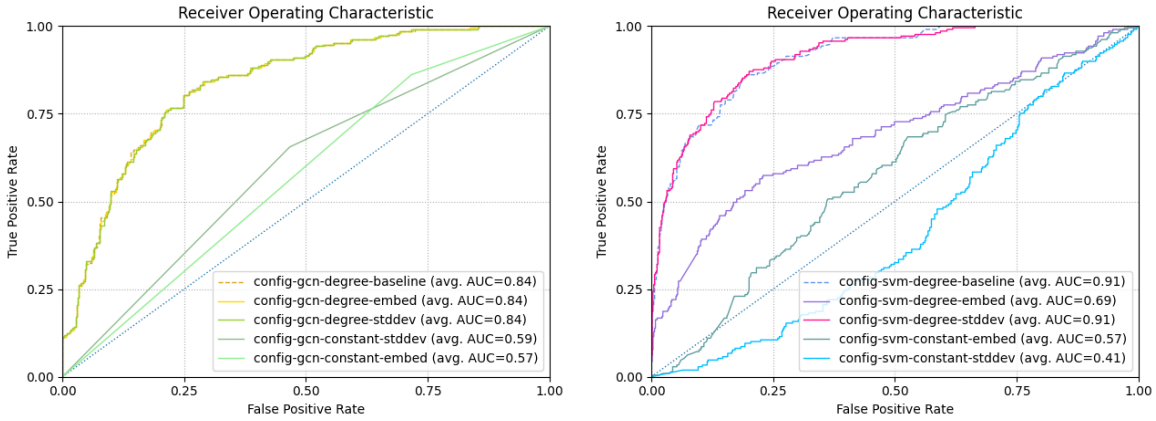


Figure 5.9: Distribution of number of annotated GO terms per species in PDMap.

Since the embeddings are of relatively large dimensionality, to avoid making the dimensionality of the final feature vector (in the sense of Section 4.1) exceedingly large, we avoid combining them with the full set of all other available features.



(a) TODO **(b)** TODO

Figure 5.10: (AlzPathwayLast→ PDMap; subsets) TODO

Results & Discussion Results are visualised in Figure 5.10. Note that since we only consider the induced subgraph or nodes for wich annotations are available, the training and validation datasets are different from previous experiments.

Neither approach seems to yield gains in classification performance. Indeed, using GO-based features only yields performance not much better than that of a random classifier. If combined with information on node degrees, model performance stays the same in the best case and drops in other cases.

The implementation of this approach given here has several shortcomings:

- It is questionable how expressive the term embeddings are. Due to time constraints, parameters for determining the embeddings have been chosen naively and there has been little evaluation of their quality.
- It is species that are annotated with GO terms, but we map this information onto species aliases. If a species has more than one alias that appears in different biological processes, both aliases can be expected to carry annotations for all biological processes the species is involved in. This potentially leads to ambiguity.

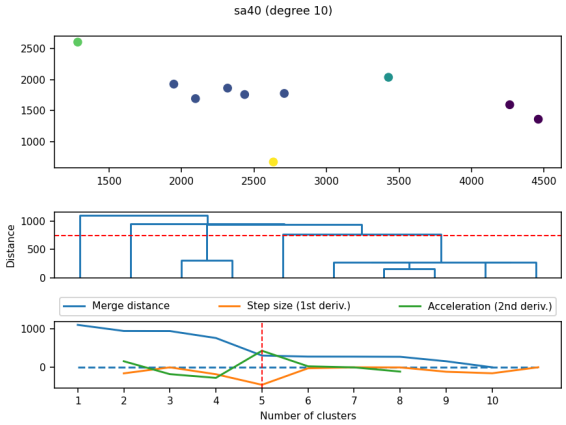
- Many species are in fact annotated with more than just a few GO terms (see Figure 5.9). It is unclear how specific or expressive many of these annotations are.
- Aggregating several embedding vectors naively by taking their mean may be problematic.
- Considering only species aliases for which annotations could be obtained further reduces the size of the dataset. This is potentially problematic for training the models. After subtracting excluded species, for AlzPathwayLast, there are 204 aliases for prediction, 83 of these are of positive class. For PDMap, there are 802 aliases for prediction, 209 of these are of positive class.

One interesting observation is that, for the GNN model, the curves for `degree-embed` and `degree-stddev` almost coincide, while for the SVM model, there is a noticeable difference in the sense that additionally including `GO_embed` hurts model performance while providing `GO_stddev` does not.

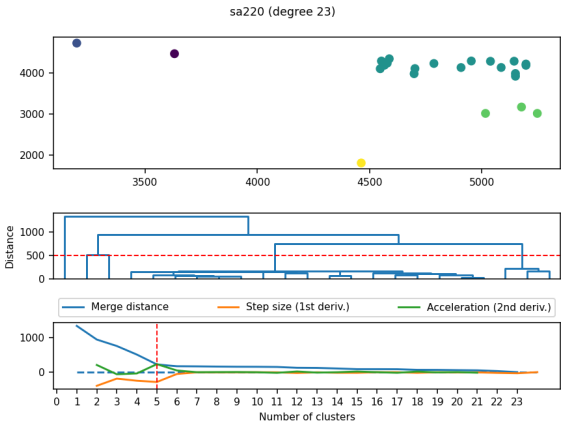
5.10 Attachment of Edges

Once the decision whether a given node should be duplicated has been made, the next step is to determine the number of duplicates to be introduced and how to distribute the edges of the duplicated node among its duplicates.

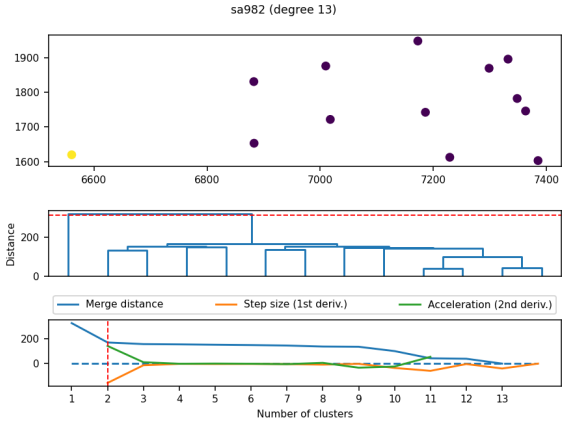
We apply the method to determine the number of duplicates and re-distribute edges described in Section 4.3 to the *AlzPathway* map. From manual inspection, the vast majority of results seemed to be reasonable clusterings, and in rare cases, ambiguous situations like illustrated here occured. We provide some examples for successful and problematic cases in Figure 5.11. All nodes considered have positive ground-truth class (i.e. are real duplication parents).



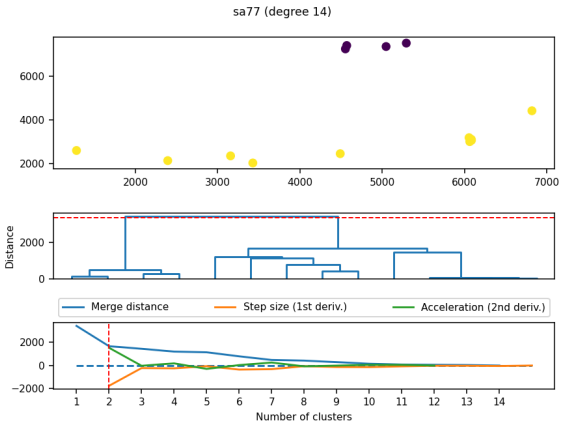
(a) Example for which the heuristic yields an intuitive clustering. Note that outliers do not distort the clustering result and are assigned their own cluster.



(b) Example in which using the second derivative yields a clustering that intuitively seems more reasonable. The first derivative has its minimum at 2 clusters.



(c) Distances between the purple points are not much different from the distance between the yellow point and the purple cluster (measured by *single linkage*). While the heuristic yields only two clusters, it may have also been feasible to create many clusters. Indeed, the second derivative curve shows another local maximum at 11 clusters.



(d) Another example where large intra-cluster variance is problematic. Contrary to the heuristic's output, it may be preferrable to further split the yellow cluster.

Figure 5.11: Examples for the clustering procedure described in Section 4.3. The topmost scatterplot shows the positions of neighbours of a given node in the layout (given node and edges are not shown). Below, a clustering dendrogram describes the distances between any two clusters as they were merged. Finally, the sequence of merge distances as well as its first and second derivative. Taking the minumum, resp. maximum yields a hint for the number of clusters to choose. The red line indicates the suggested choice for the number of clusters and thus a concrete clustering, determing the merge node in the dendrogram where the "cut" should be made. All examples are from AlzPathwayLast with positive ground-truth label.

6 Discussion

7 Outlook & Future Work

Appendix

1 Notation & Abbreviations

- A *species* is an actor in a biological system. Examples for species are proteins, smaller molecules, but also drugs or phenotypes. A *species alias* is a visual representation of a species in a drawing. If a disease map is interpreted as a graph, a species alias can be thought of as a node in the graph, see Section 4.1. There can be multiple species aliases corresponding to the same species. A *complex species alias* represents a collection of species aliases, commonly representing protein complexes.
- Vectors and matrices are usually set in bold letters, e.g. $\mathbf{x}$ or $\mathbf{A}$. Vectors are assumed to column vectors unless otherwise specified. $\cdot^T$ denotes the transpose.
- $\|\cdot\|$ denotes the l_2-norm or set cardinality, depending on context.
- G commonly denotes a graph, $V(G)$ is its vertex set and $E(G)$ its edge set. Whether we consider a directed, undirected, bipartite or simple graph will be made clear from context.
- $\mathcal{N}(v)$ denotes the graph neighbourhood of node v. Unless otherwise specified, this is the direct 1-hop neighbourhood, that is, the nodes adjacent to v.
- For a bipartite graph with node set $V = A \cup B$, that is, the disjoint union of species aliases A and reactions B, the *bipartite projection onto* A is the simple graph with node set A in which $v_i, v_j \in A$ are connected if and only if in the bipartite they have a common neighbour in B.

2 Acknowledgements

Special thanks go to Sune S. Nielsen and Marek Ostaszewski for kindly answering questions and being available for discussion. I also thank Karsten Klein, Matthias Rupp and particularly Michael Aichem for providing me with valuable feedback, guiding me through the process and simply being great people to work with.

References

[1] Sune S. Nielsen et al. 'Machine Learning to Support the Presentation of Complex Pathway Graphs'. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2019), pp. 1–1. DOI: 10.1109/TCBB.2019.2938501 (cited on pages 1, 15, 17, 23–26, 28, 33–38, 41).

[2] Camilo Ruiz, Marinka Zitnik, and Jure Leskovec. 'Identification of Disease Treatment Mechanisms through the Multiscale Interactome'. In: *Nature Communications* 12.1 (1 Mar. 19, 2021), p. 1796. DOI: 10.1038/s41467-021-21770-8. (Visited on 03/22/2021) (cited on pages 3, 19, 43).

[3] Albert-László Barabási, Natali Gulbahce, and Joseph Loscalzo. 'Network Medicine: A Network-Based Approach to Human Disease'. In: *Nature Reviews Genetics* 12.1 (1 Jan. 2011), pp. 56–68. DOI: 10.1038/nrg2918. (Visited on 10/04/2021) (cited on page 3).

[4] Martin Siebenhaller et al. 'Human-like Layout Algorithms for Signalling Hypergraphs: Outlining Requirements'. In: *Briefings in Bioinformatics* 21.1 (Jan. 17, 2020), pp. 62–72. DOI: 10.1093/bib/bby099. (Visited on 06/08/2021) (cited on pages 3, 6, 15).

[5] M. Y. Becker and I. Rojas. 'A Graph Layout Algorithm for Drawing Metabolic Pathways'. In: *Bioinformatics* 17.5 (May 1, 2001), pp. 461–467. DOI: 10.1093/bioinformatics/17.5.461. (Visited on 10/25/2021) (cited on pages 4, 15).

[6] Suzanne M. Paley and Peter D. Karp. 'The Pathway Tools Cellular Overview Diagram and Omics Viewer'. In: *Nucleic Acids Research* 34.13 (Aug. 1, 2006), pp. 3771–3778. DOI: 10.1093/nar/gkl334. (Visited on 10/13/2021) (cited on page 4).

[7] Peter Droste, Wolfgang Wiechert, and Katharina Nöh. 'Semi-Automatic Drawing of Metabolic Networks'. In: *Information Visualization* 11.3 (July 1, 2012), pp. 171–187. DOI: 10.1177/1473871611413565. (Visited on 10/04/2021) (cited on page 4).

[8] Marek Ostaszewski et al. 'Community-Driven Roadmap for Integrated Disease Maps'. In: *Briefings in Bioinformatics* 20.2 (Mar. 25, 2019), pp. 659–670. DOI: 10.1093/bib/bby024. (Visited on 06/08/2021) (cited on page 4).

[9] Alexander Mazein et al. 'Systems Medicine Disease Maps: Community-Driven Comprehensive Representation of Disease Mechanisms'. In: *NPJ systems biology and applications* 4 (2018), p. 21. DOI: 10.1038/s41540-018-0059-y (cited on page 4).

[10] Donna Maglott et al. 'Entrez Gene: Gene-Centered Information at NCBI'. In: *Nucleic Acids Research* 33 (Database issue Jan. 1, 2005), pp. D54–58. DOI: 10.1093/nar/gki031 (cited on pages 4, 7).

[11] The UniProt Consortium. 'UniProt: The Universal Protein Knowledgebase in 2021'. In: *Nucleic Acids Research* 49.D1 (Jan. 8, 2021), pp. D480–D489. DOI: 10.1093/nar/gkaa1100. (Visited on 10/04/2021) (cited on pages 4, 7).

[12] *NF-kB Mechanism of Action*. In collab. with Boghog on English Wikipedia. 3 October 2007 (original upload date). (Visited on 11/08/2021) (cited on page 5).

[13] *CellDesigner: The Process Diagram*. URL: http://www.celldesigner.org/documents/ProcessDiagram.html (visited on 11/08/2021) (cited on page 5).

[14] Alberto Noronha et al. 'ReconMap: An Interactive Visualization of Human Metabolism'. In: *Bioinformatics* 33.4 (Feb. 15, 2017), pp. 605–607. DOI: 10.1093/bioinformatics/btw667. (Visited on 07/09/2021) (cited on pages 5, 17, 33).

[15] Romain Bourqui et al. 'Metabolic Network Visualization Eliminating Node Redundance and Preserving Metabolic Pathways'. In: *BMC Systems Biology* 1.1 (July 3, 2007), p. 29. DOI: 10.1186/1752-0509-1-29. (Visited on 09/23/2021) (cited on page 6).

[16] Hsiang-Yun Wu et al. 'Metabopolis: Scalable Network Layout for Biological Pathway Diagrams in Urban Map Style'. In: *BMC Bioinformatics* 20.1 (Apr. 15, 2019), p. 187. DOI: 10.1186/s12859-019-2779-4. (Visited on 09/23/2021) (cited on pages 6, 15).

[17] David S. Wishart et al. 'DrugBank: A Knowledgebase for Drugs, Drug Actions and Drug Targets'. In: *Nucleic Acids Research* 36 (suppl_1 Jan. 1, 2008), pp. D901–D906. DOI: 10.1093/nar/gkm958. (Visited on 10/05/2021) (cited on page 7).

[18] Michael Ashburner et al. 'Gene Ontology: Tool for the Unification of Biology'. In: *Nature Genetics* 25.1 (1 May 2000), pp. 25–29. DOI: 10.1038/75556. (Visited on 10/05/2021) (cited on page 7).

[19] Vladimir Vapnik. 'Principles of Risk Minimization for Learning Theory'. In: (), p. 8 (cited on page 8).

[20] Michael M. Bronstein et al. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. May 2, 2021. URL: http://arxiv.org/abs/2104.13478 (visited on 05/06/2021) (cited on page 8).

[21] Aston Zhang et al. 'Dive into Deep Learning'. In: (), p. 1021 (cited on pages 8, 9, 11).

[22] Lex Flagel, Yaniv Brandvain, and Daniel R. Schrider. 'The Unreasonable Effectiveness of Convolutional Neural Networks in Population Genetic Inference'. In: *Molecular Biology and Evolution* 36.2 (Feb. 1, 2019), pp. 220–238. DOI: 10.1093/molbev/msy224 (cited on page 9).

[23] Genta Aoki and Yasubumi Sakakibara. 'Convolutional Neural Networks for Classification of Alignments of Non-Coding RNA Sequences'. In: *Bioinformatics* 34.13 (July 1, 2018), pp. i237–i244. DOI: 10.1093/bioinformatics/bty228. (Visited on 10/06/2021) (cited on page 9).

[24] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. Sept. 10, 2018. URL: http://arxiv.org/abs/1706.02216 (visited on 06/16/2021) (cited on page 10).

[25] Rex Ying et al. *Hierarchical Graph Representation Learning with Differentiable Pooling*. Feb. 20, 2019. URL: http://arxiv.org/abs/1806.08804 (visited on 02/04/2021) (cited on page 10).

[26] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. Feb. 22, 2017. URL: http://arxiv.org/abs/1609.02907 (visited on 12/07/2020) (cited on page 10).

[27] Petar Veličković et al. *Graph Attention Networks*. Feb. 4, 2018. URL: http://arxiv.org/abs/1710.10903 (visited on 01/05/2021) (cited on page 10).

[28] Ashish Vaswani et al. *Attention Is All You Need*. Dec. 5, 2017. URL: http://arxiv.org/abs/1706.03762 (visited on 10/10/2021) (cited on page 11).

[29] Nitish Srivastava et al. 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: (), p. 30 (cited on page 11).

[30] Sergey Ioffe and Christian Szegedy. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: (), p. 9 (cited on page 11).

[31] Kaiming He et al. *Deep Residual Learning for Image Recognition*. Dec. 10, 2015. URL: http://arxiv.org/abs/1512.03385 (visited on 05/11/2021) (cited on page 11).

[32] Taocheng Liu, Xiangbin Ye, and Bei Sun. 'Combining Convolutional Neural Network and Support Vector Machine for Gait-Based Gender Recognition'. In: *2018 Chinese Automation Congress (CAC)*. 2018 Chinese Automation Congress (CAC). Nov. 2018, pp. 3477–3481. DOI: 10.1109/CAC.2018.8623118 (cited on page 11).

[33] Sami Tibshirani, Harry Friedman, and Trevor Hastie. 'The Elements of Statistical Learning'. In: (2017), p. 764 (cited on page 12).

[34] G. Joshi-Tope et al. 'Reactome: A Knowledgebase of Biological Pathways'. In: *Nucleic Acids Research* 33 (suppl_1 Jan. 1, 2005), pp. D428–D432. DOI: 10.1093/nar/gki072. (Visited on 10/13/2021) (cited on pages 15, 16).

[35] Stephen G Kobourov. 'Force-Directed Drawing Algorithms'. In: *Handbook of Graph Drawing and Visualization* (2013), p. 26. DOI: 10.1201/b15385-15 (cited on page 15).

[36] Emden R. Gansner, Yehuda Koren, and Stephen North. 'Graph Drawing by Stress Majorization'. In: *Graph Drawing*. Ed. by János Pach. Red. by David Hutchison et al. Vol. 3383. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 239–250. DOI: 10.1007/978-3-540-31843-9_25. (Visited on 03/31/2020) (cited on pages 15, 17).

[37] Patrick Healy and Nikola S Nikolov. 'Hierarchical Drawing Algorithms'. In: *Handbook of Graph Drawing and Visualization* (2013), p. 46 (cited on page 15).

[38] Christian A Duncan and Michael T Goodrich. 'Planar Orthogonal and Polyline Drawing Algorithms'. In: *Handbook of Graph Drawing and Visualization* (2013), p. 24 (cited on page 15).

[39] Falk Schreiber. 'Comparison of Metabolic Pathways Using Constraint Graph Drawing'. In: (2003), p. 6 (cited on page 15).

[40] Steve Kieffer et al. 'HOLA: Human-like Orthogonal Network Layout'. In: *IEEE Transactions on Visualization and Computer Graphics* 22.1 (Jan. 31, 2016), pp. 349–358. DOI: 10.1109/TVCG.2015.2467451. (Visited on 06/08/2021) (cited on page 15).

[41] Hsiang-Yun Wu, Martin Nollenburg, and Ivan Viola. 'Multi-Level Area Balancing of Clustered Graphs'. In: *IEEE Transactions on Visualization and Computer Graphics* (2020), pp. 1–1. DOI: 10.1109/TVCG.2020.3038154. (Visited on 10/25/2021) (cited on pages 15, 16).

[42] Mikael Huss and Petter Holme. 'Currency and Commodity Metabolites: Their Identification and Relation to the Modularity of Metabolic Networks'. In: *IET Systems Biology* 1.5 (Sept. 1, 2007), pp. 280–285. DOI: 10.1049/iet-syb:20060077. (Visited on 06/09/2021) (cited on pages 15, 16).

[43] S. Schuster et al. 'Exploring the Pathway Structure of Metabolism: Decomposition into Subnetworks and Application to Mycoplasma Pneumoniae'. In: *Bioinformatics* 18.2 (Feb. 1, 2002), pp. 351–361. DOI: 10.1093/bioinformatics/18.2.351. (Visited on 11/20/2020) (cited on page 16).

[44] Eun-Youn Kim, Daniel Ashlock, and Sung Ho Yoon. 'Identification of Critical Connectors in the Directed Reaction-Centric Graphs of Microbial Metabolic Networks'. In: *BMC bioinformatics* 20.1 (June 13, 2019), p. 328. DOI: 10.1186/s12859-019-2897-z (cited on page 16).

[45] Annegret Liebers. 'Planarizing Graphs — A Survey and Annotated Bibliography'. In: (2001), p. 74 (cited on page 16).

[46] Faisal N. Abu-Khzam et al. 'Cluster Editing with Vertex Splitting'. In: *Combinatorial Optimization*. Ed. by Jon Lee, Giovanni Rinaldi, and A. Ridha Mahjoub. Vol. 10856. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 1–13. DOI: 10.1007/978-3-319-96151-4_1. (Visited on 10/13/2021) (cited on page 16).

[47] P. Eades and C. F. X. de Mendonça N. 'Vertex Splitting and Tension-Free Layout'. In: *Graph Drawing*. Ed. by Franz J. Brandenburg. Red. by G. Goos, J. Hartmanis, and J. van Leeuwen. Vol. 1027. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 202–211. DOI: 10.1007/BFb0021804. (Visited on 06/16/2021) (cited on page 16).

[48] Tomihisa Kamada and Satoru Kawai. 'An Algorithm for Drawing General Undirected Graphs'. In: *Information Processing Letters* 31.1 (1989), p. 9 (cited on page 16).

[49] J. Li. *Eliminate Wire Crossings by Node Duplication*. 2008. URL: https://www.semanticscholar.org/paper/Eliminate-Wire-Crossings-by-Node-Duplication-Li/f15e6d3a207ca83339ee1901047a6d7905cc4d39 (visited on 09/23/2021) (cited on page 16).

[50] Doo-Kwon Paik, S. Reddy, and S. Sahni. 'Vertex Splitting in Dags and Applications to Partial Scan Designs and Lossy Circuits'. In: *Int. J. Found. Comput. Sci.* (1998). DOI: 10.1142/S0129054198000301 (cited on page 16).

[51] M. Mayer and F. Erçal. 'Genetic Algorithms for Vertex Splitting in DAGs'. In: *undefined* (1993). (Visited on 10/13/2021) (cited on page 16).

[52] N. Henr, A. Bezerianos, and J.-D. Fekete. 'Improving the Readability of Clustered Social Networks Using Node Duplication'. In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (Nov. 2008), pp. 1317–1324. DOI: 10.1109/TVCG.2008.141. (Visited on 06/08/2021) (cited on page 16).

[53] H. Ma and A. Zeng. 'Reconstruction of Metabolic Networks from Genome Data and Analysis of Their Global Structure for Various Organisms'. In: *undefined* (2003). (Visited on 05/28/2021) (cited on page 16).

[54] Ichcha Manipur et al. 'Clustering Analysis of Tumor Metabolic Networks'. In: *BMC Bioinformatics* 21.10 (Aug. 25, 2020), p. 349. DOI: 10.1186/s12859-020-03564-9. (Visited on 01/06/2021) (cited on page 16).

[55] M. E. J. Newman. 'Modularity and Community Structure in Networks'. In: *Proceedings of the National Academy of Sciences* 103.23 (June 6, 2006), pp. 8577–8582. DOI: 10.1073/pnas.0601602103. (Visited on 04/12/2019) (cited on page 16).

[56] Roger Guimerà and Luís A. Nunes Amaral. 'Functional Cartography of Complex Metabolic Networks'. In: *Nature* 433.7028 (7028 Feb. 2005), pp. 895–900. DOI: 10.1038/nature03288. (Visited on 06/09/2021) (cited on page 16).

[57] Markus Rohrschneider et al. 'A Novel Grid-Based Visualization Approach for Metabolic Networks with Advanced Focus&Context View'. In: *Graph Drawing*. Ed. by David Eppstein and Emden R. Gansner. Red. by David Hutchison et al. Vol. 5849. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 268–279. DOI: 10.1007/978-3-642-11805-0_26. (Visited on 06/11/2021) (cited on page 16).

[58] Hendrik Rohn et al. 'VANTED v2: A Framework for Systems Biology Applications'. In: *BMC Systems Biology* 6.1 (2012), p. 139. DOI: 10.1186/1752-0509-6-139. (Visited on 04/24/2019) (cited on page 17).

[59] Alice C. Villéger, Stephen R. Pettifer, and Douglas B. Kell. 'Arcadia: A Visualization Tool for Metabolic Pathways'. In: *Bioinformatics* 26.11 (June 1, 2010), pp. 1470–1471. DOI: 10.1093/bioinformatics/btq154. (Visited on 10/04/2021) (cited on page 17).

[60] Peter Droste, Katharina Nöh, and Wolfgang Wiechert. 'Omix - A Visualization Tool for Metabolic Networks with Highest Usability and Customizability in Focus'. In: *Chemie Ingenieur Technik* 85.6 (June 2013), pp. 849–862. DOI: 10.1002/cite.201200234. (Visited on 10/04/2021) (cited on page 17).

[61] Soichi Ogishima et al. 'AlzPathway, an Updated Map of Curated Signaling Pathways: Towards Deciphering Alzheimer's Disease Pathogenesis'. In: *Methods in Molecular Biology (Clifton, N.J.)* 1303 (2016), pp. 423–432. DOI: 10.1007/978-1-4939-2627-5_25 (cited on pages 17, 33).

[62] Kazuhiro A. Fujita et al. 'Integrating Pathways of Parkinson's Disease in a Molecular Interaction Map'. In: *Molecular Neurobiology* 49.1 (Feb. 1, 2014), pp. 88–102. DOI: 10.1007/s12035-013-8489-4. (Visited on 06/14/2021) (cited on pages 17, 33).

[63] Marek Ostaszewski et al. 'COVID-19 Disease Map, Building a Computational Repository of SARS-CoV-2 Virus-Host Interaction Mechanisms'. In: *Scientific Data* 7.1 (May 5, 2020), p. 136. DOI: 10.1038/s41597-020-0477-8 (cited on page 17).

[64] Satoshi Mizuno et al. 'Network Analysis of a Comprehensive Knowledge Repository Reveals a Dual Role for Ceramide in Alzheimer's Disease'. In: *PloS One* 11.2 (2016), e0148431. DOI: 10.1371/journal.pone.0148431 (cited on pages 17, 33).

[65] I. Kuperstein et al. 'Atlas of Cancer Signalling Network: A Systems Biology Resource for Integrative Analysis of Cancer Data with Google Maps'. In: *Oncogenesis* 4.7 (7 July 2015), e160–e160. DOI: 10.1038/oncsis.2015.19. (Visited on 10/25/2021) (cited on page 17).

[66] Nicolas Sompairac et al. 'Metabolic and Signalling Network Maps Integration: Application to Cross-Talk Studies and Omics Data Analysis in Cancer'. In: *BMC Bioinformatics* 20 (Suppl 4 Apr. 18, 2019). DOI: 10.1186/s12859-019-2682-z. (Visited on 04/14/2021) (cited on page 17).

[67] Akira Funahashi et al. 'CellDesigner 3.5: A Versatile Modeling Tool for Biochemical Networks'. In: *Proceedings of the IEEE* 96.8 (Aug. 2008), pp. 1254–1265. DOI: 10.1109/JPROC.2008.925458 (cited on page 17).

[68] Piotr Gawron et al. 'MINERVA—a Platform for Visualization and Curation of Molecular Interaction Networks'. In: *npj Systems Biology and Applications* 2.1 (1 Sept. 22, 2016), pp. 1–6. DOI: 10.1038/npjsba.2016.20. (Visited on 10/04/2021) (cited on page 17).

[69] Inna Kuperstein et al. 'NaviCell: A Web-Based Environment for Navigation, Curation and Maintenance of Large Molecular Interaction Maps'. In: *BMC Systems Biology* 7.1 (Oct. 7, 2013), p. 100. DOI: 10.1186/1752-0509-7-100. (Visited on 10/25/2021) (cited on page 17).

[70] Paul Shannon et al. 'Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks'. In: *Genome Research* 13.11 (Jan. 11, 2003), pp. 2498–2504. DOI: 10.1101/gr.1239303. (Visited on 05/04/2021) (cited on page 17).

[71] Frank T. Bergmann et al. 'Systems Biology Graphical Notation Markup Language (SBGNML) Version 0.3'. In: *Journal of Integrative Bioinformatics* 17.2-3 (Aug. 24, 2020), p. 20200016. DOI: 10.1515/jib-2020-0016. (Visited on 11/08/2021) (cited on page 17).

[72] Matteo Tiezzi, Gabriele Ciravegna, and Marco Gori. *Graph Neural Networks for Graph Drawing*. Sept. 21, 2021. URL: http://arxiv.org/abs/2109.10061 (visited on 10/13/2021) (cited on page 17).

[73] Ronghui You et al. 'DeepGraphGO: Graph Neural Network for Large-Scale, Multispecies Protein Function Prediction'. In: *Bioinformatics* 37 (Supplement_1 July 1, 2021), pp. i262–i271. DOI: 10.1093/bioinformatics/btab270. (Visited on 10/13/2021) (cited on page 18).

[74] Xiaoshi Zhong, Rama Kaalia, and Jagath C. Rajapakse. 'GO2Vec: Transforming GO Terms and Proteins to Vector Representations via Graph Embeddings'. In: *BMC Genomics* 20.9 (Feb. 18, 2020), p. 918. DOI: 10.1186/s12864-019-6272-2. (Visited on 09/06/2021) (cited on pages 18, 19, 27).

[75] Neal Ravindra et al. 'Disease State Prediction from Single-Cell Data Using Graph Attention Networks'. In: *Proceedings of the ACM Conference on Health, Inference, and Learning*. CHIL '20. New York, NY, USA: Association for Computing Machinery, Apr. 2, 2020, pp. 121–130. DOI: 10.1145/3368555.3384449. (Visited on 04/21/2021) (cited on page 18).

[76] Jonathan M. Stokes et al. 'A Deep Learning Approach to Antibiotic Discovery'. In: *Cell* 180.4 (Feb. 2020), 688–702.e13. DOI: 10.1016/j.cell.2020.01.021. (Visited on 10/14/2021) (cited on page 18).

[77] David Duvenaud et al. *Convolutional Networks on Graphs for Learning Molecular Fingerprints*. Nov. 3, 2015. URL: http://arxiv.org/abs/1509.09292 (visited on 01/21/2021) (cited on page 18).

[78] Oliver Wieder et al. 'A Compact Review of Molecular Property Prediction with Graph Neural Networks'. In: *Drug Discovery Today: Technologies* (Dec. 17, 2020). DOI: 10.1016/j.ddtec.2020.11.009. (Visited on 10/14/2021) (cited on page 18).

[79] Thomas Gaudelet et al. *Utilising Graph Machine Learning within Drug Discovery and Development*. Dec. 9, 2020. URL: http://arxiv.org/abs/2012.05716 (visited on 01/14/2021) (cited on page 18).

[80] Mayank Baranwal et al. 'A Deep Learning Architecture for Metabolic Pathway Prediction'. In: *Bioinformatics (Oxford, England)* 36.8 (Apr. 15, 2020), pp. 2547–2553. DOI: 10.1093/bioinformatics/btz954 (cited on page 18).

[81] Bajaj, Heereguppe, and Sumath. *Graph Convolutional Networks to Explore Drug and Disease Relationships in Biological Networks*. 2017. (Visited on 10/14/2021) (cited on page 18).

[82] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 'Modeling Polypharmacy Side Effects with Graph Convolutional Networks'. In: *Bioinformatics* 34.13 (July 1, 2018), pp. i457–i466. DOI: 10.1093/bioinformatics/bty294. (Visited on 12/14/2020) (cited on page 18).

[83] Hryhorii Chereda et al. 'Explaining Decisions of Graph Convolutional Neural Networks: Patient-Specific Molecular Subnetworks Responsible for Metastasis Prediction in Breast Cancer'. In: *Genome Medicine* 13 (Mar. 11, 2021), p. 42. DOI: 10.1186/s13073-021-00845-7. (Visited on 10/14/2021) (cited on page 18).

[84] Xiao-Meng Zhang et al. 'Graph Neural Networks and Their Current Applications in Bioinformatics'. In: *Frontiers in Genetics* 12 (July 29, 2021), p. 690049. DOI: 10.3389/fgene.2021.690049. (Visited on 10/14/2021) (cited on page 18).

[85] Vincent Henry et al. 'Converting Disease Maps into Heavyweight Ontologies: General Methodology and Application to Alzheimer's Disease'. In: *Database: The Journal of Biological Databases and Curation* 2021 (Feb. 16, 2021), baab004. DOI: 10.1093/database/baab004 (cited on page 18).

[86] Chenguang Zhao and Zheng Wang. 'GOGO: An Improved Algorithm to Measure the Semantic Similarity between Gene Ontology Terms'. In: *Scientific Reports* 8.1 (1 Oct. 10, 2018), p. 15107. DOI: 10.1038/s41598-018-33219-y. (Visited on 10/03/2021) (cited on page 19).

[87] Guangchuang Yu et al. 'GOSemSim: An R Package for Measuring Semantic Similarity among GO Terms and Gene Products'. In: *Bioinformatics* 26.7 (Apr. 1, 2010), pp. 976–978. DOI: 10.1093/bioinformatics/btq064. (Visited on 09/06/2021) (cited on page 19).

[88] J. Z. Wang et al. 'A New Method to Measure the Semantic Similarity of GO Terms'. In: *Bioinformatics* 23.10 (May 15, 2007), pp. 1274–1281. DOI: 10.1093/bioinformatics/btm087. (Visited on 09/06/2021) (cited on page 19).

[89] Marek Ostaszewski et al. 'Clustering Approaches for Visual Knowledge Exploration in Molecular Interaction Networks'. In: *BMC Bioinformatics* 19.1 (Aug. 29, 2018), p. 308. DOI: 10.1186/s12859-018-2314-z. (Visited on 06/07/2021) (cited on pages 19, 43).

[90] Roman Schulte-Sasse et al. 'Integration of Multiomics Data with Graph Convolutional Networks to Identify New Cancer Genes and Their Associated Molecular Mechanisms'. In: *Nature Machine Intelligence* 3.6 (6 June 2021), pp. 513–526. DOI: 10.1038/s42256-021-00325-y. (Visited on 06/14/2021) (cited on page 21).

[91] *CellDesigner 4 Extension Tag Specification Document*. 2010. URL: http://www.celldesigner.org/documents/CellDesigner4ExtensionTagSpecificationE.pdf (visited on 08/16/2021) (cited on page 22).

[92] Ulrik Brandes and Thomas Erlebach, eds. *Network Analysis: Methodological Foundations*. LCNS, Tutorial 3418. Berlin ; New York: Springer, 2005. 471 pp. (cited on page 25).

[93] Alaa Tharwat. 'Classification Assessment Methods'. In: *Applied Computing and Informatics* 17.1 (Jan. 4, 2021), pp. 168–192. DOI: 10.1016/j.aci.2018.08.003. (Visited on 11/01/2021) (cited on page 28).

[94] Tom Fawcett. 'An Introduction to ROC Analysis'. In: *Pattern Recognition Letters* 27.8 (June 2006), pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010. (Visited on 11/08/2021) (cited on page 28).

[95] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. Apr. 25, 2019. URL: http://arxiv.org/abs/1903.02428 (visited on 11/01/2021) (cited on page 30).

[96] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. (Visited on 11/01/2021) (cited on page 30).

[97] Fabian Pedregosa et al. 'Scikit-Learn: Machine Learning in Python'. In: *MACHINE LEARNING IN PYTHON* (), p. 6 (cited on page 30).

[98] Benjamin J. Bornstein et al. 'LibSBML: An API Library for SBML'. In: *Bioinformatics (Oxford, England)* 24.6 (Mar. 15, 2008), p. 880. DOI: 10.1093/bioinformatics/btn051. (Visited on 11/06/2021) (cited on page 30).

[99] Ali Ebrahim et al. 'COBRApy: COnstraints-Based Reconstruction and Analysis for Python'. In: *BMC Systems Biology* 7.1 (Aug. 8, 2013), p. 74. DOI: 10.1186/1752-0509-7-74. (Visited on 11/06/2021) (cited on page 30).

[100] *Snap-Stanford/GraphGym*. snap-stanford, Feb. 22, 2021. (Visited on 02/22/2021) (cited on page 30).

[101] Jiaxuan You, Rex Ying, and Jure Leskovec. *Design Space for Graph Neural Networks*. Nov. 17, 2020. URL: http://arxiv.org/abs/2011.08843 (visited on 02/26/2021) (cited on pages 30, 35).

[102] Aric A Hagberg, Daniel A Schult, and Pieter J Swart. 'Exploring Network Structure, Dynamics, and Function Using NetworkX'. In: (2008), p. 5 (cited on page 30).

[103] Gabor Csardi and Tamas Nepusz. 'The Igraph Software Package for Complex Network Research'. In: (), p. 10 (cited on page 30).

[104] Gesellschaft für Klassifikation and Christine Preisach, eds. *Data Analysis, Machine Learning and Applications: Proceedings of the 31st Annual Conference of the Gesellschaft Für Klassifikation e.V., Albert-Ludwigs-Universität Freiburg, March 7-9, 2007*. Berlin: Springer, 2008. 719 pp. (cited on page 31).

[105] Jeff Reback et al. *Pandas-Dev/Pandas: Pandas 1.3.4*. Version v1.3.4. Zenodo, Oct. 17, 2021. DOI: 10.5281/ZENODO.3509134. (Visited on 11/01/2021) (cited on page 31).

[106] Charles R. Harris et al. 'Array Programming with NumPy'. In: *Nature* 585.7825 (Sept. 17, 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. (Visited on 11/01/2021) (cited on page 31).

[107] *GNU Emacs - GNU Project*. URL: https://www.gnu.org/software/emacs/ (visited on 11/06/2021) (cited on page 31).

[108] Thomas Kluyver et al. 'Jupyter Notebooks – a Publishing Format for Reproducible Computational Workflows'. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87. (Visited on 11/06/2021) (cited on page 31).

[109] Thomas A Caswell et al. *Matplotlib/Matplotlib: REL: V3.5.0rc1*. Version v3.5.0rc1. Zenodo, Oct. 2, 2021. DOI: 10.5281/ZENODO.592536. (Visited on 11/06/2021) (cited on page 31).

[110] Satoshi Mizuno et al. 'AlzPathway: A Comprehensive Map of Signaling Pathways of Alzheimer's Disease'. In: *BMC Systems Biology* 6.1 (May 30, 2012), p. 52. DOI: 10.1186/1752-0509-6-52. (Visited on 06/02/2021) (cited on page 33).

[111] S. Ogishima et al. 'A Map of Alzheimer's Disease-Signaling Pathways: A Hope for Drug Target Discovery'. In: *Clinical Pharmacology and Therapeutics* 93.5 (May 2013), pp. 399–401. DOI: 10.1038/clpt.2013.37 (cited on page 33).

[112] Marek Ostaszewski. *AlzPathway Regorganisation Steps for Increased Modularity*. Zenodo, June 25, 2021. DOI: 10.5281/zenodo.5032278. (Visited on 06/28/2021) (cited on page 33).

[113] Ines Thiele et al. 'A Community-Driven Global Reconstruction of Human Metabolism'. In: *Nature Biotechnology* 31.5 (5 May 2013), pp. 419–425. DOI: 10.1038/nbt.2488. (Visited on 10/18/2021) (cited on page 33).

[114] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 'Practical Bayesian Optimization of Machine Learning Algorithms'. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. (Visited on 11/06/2021) (cited on page 34).

[115] James Bergstra and Yoshua Bengio. 'Random Search for Hyper-Parameter Optimization'. In: (), p. 25 (cited on page 34).

[116] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 29, 2017. URL: http://arxiv.org/abs/1412.6980 (visited on 11/08/2021) (cited on page 35).

[117] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 'GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks'. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21: The Fourteenth ACM International Conference on Web Search and Data Mining. Virtual Event Israel: ACM, Mar. 8, 2021, pp. 833–841. DOI: 10.1145/3437963.3441720. (Visited on 08/29/2021) (cited on page 39).

[118] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. 'A Review of Feature Selection Techniques in Bioinformatics'. In: *Bioinformatics* 23.19 (Oct. 1, 2007), pp. 2507–2517. DOI: 10.1093/bioinformatics/btm344. (Visited on 10/25/2021) (cited on page 41).

[119] Jiwen Xin et al. 'High-Performance Web Services for Querying Gene and Variant Annotation'. In: *Genome Biology* 17.1 (May 6, 2016), p. 91. DOI: 10.1186/s13059-016-0953-9 (cited on page 43).