Link Prediction in Ecological Networks using Latent Space Representation of Network Graphs

Authored by
github.com/skiand

***Abstract*—This paper evaluates a method of implementing link prediction on food webs. A diverse set of food webs from different parts of the world is used as the dataset. The dataset contains pairs of predator-prey species, which are used as pairs of source and destination nodes respectively. This project aims to find future links or uncover previously hidden connections among species. The nodes are first processed using StellarGraph, a Python library that is used for graph machine learning, and then passed through DeepWalk, an algorithm for learning latent representations of vertices in a network graph. Furthermore, the resulting embeddings are fed through a classifier for link prediction. The resulting prediction is then evaluated using the ROC AUC as the metric. The results indicate that this technique has a moderate level of success.**

Keywords—link prediction, food web, latent space embedding, DeepWalk

I. Introduction

A food web is a type of ecological network that, in simple words, "maps which species eat other species" [1]. The trophic interaction between species can be defined as nodes and edges, where each node can be one of predator (consumer) or prey (resource) and the consumption of the former by the latter forms the edge between the nodes. Foods webs tend to be very densely connected, with even the most distant species being two or three links away from each other [1].

Of particular interest is the study carried out by [2], which found that the average body-mass ratios of pairings consisting of predators and preys are high in the cases of small vertebrates, large swimming predators and flying predators. Furthermore, it introduced the GlobAl daTabasE of traits and food Web Architecture (GATEWAy), one of the biggest food web datasets to date, comprising over 290 individual food webs, across 28 different locations and 33 study sites across the world. Each food web is characterised by one or more different ecosystem types: marine environments, terrestrial above-ground environments, terrestrial below-ground environments, stream environment and lake environments.

The aforementioned food web database can be represented in the form of a network graph and is used in this paper to implement the process of link prediction. Link prediction can be defined as "the task of predicting the presence of an edge between two nodes in a network based on latent characteristics of the graph" [3]. The main task of this paper involves doing link prediction on graphs generated by a selected number of food webs. The procedure of doing link prediction is essentially a binary classification, where the algorithm randomly cuts the existing edge between pairs of nodes and then, using this edited graph, tries to predict whether an edge existed between pairs of nodes, using the original graph as ground truth.

For the purpose of predicting the links between nodes, an algorithm called DeepWalk is used. DeepWalk encodes relations among nodes and produces embeddings that present a latent space representation of the network graph. These embeddings can then be used to train the link prediction model and perform the classification.

The aim of this process is to successfully train a link prediction algorithm that can be used to accurately predict future relationships among the nodes i.e. to predict consumptions of preys by predators. The ultimate benefit is to be able to predict future relationships among species and find a connection between a predator and a prey that does not yet exist but is likely to materialise in the future.

Such a task will also be able to find links between species that did indeed exist at the time of the sampling process but were missed by the field researcher. Thus, this algorithm can be used on existing datasets, to uncover or correct missed organism connections, leading to the creation of more accurate food webs. Moreover, it is likely that this process can discover previously hidden connections between organisms that would be extremely hard or impossible for a researcher to discover in the field e.g if a species' foraging range is too far-flung.

Furthermore, this opens up the possibility that future food web dataset creations can be structured in such a way as to omit sampling certain species' connections which can instead be filled in by the algorithm. This saves costs on both labour and time, allowing institutions to focus on obtaining higher number of datasets with increasingly higher web resolutions.

II. Literature Review

A. Ecological food webs

The nodes and edges of a food web are essentially representations of species composition, and of the distribution of trophic interactions, respectively. Environmental gradients such as altitude, depth, temperature, soil humidity and precipitation can affect how energy is transferred among trophic levels. This can change our ability to predict the future state of food webs. Features of a food web can be used to learn and predict the ecological consequences of global environmental changes. Such features can include information about species' size, temperature, species-area relationships species distribution-environment associations and others [4].

The ecological food webs that exist in literature today vary in the level of resolution that they record. They often resort to tabulating organisms according to their taxonomic group rather than the down to the species level and with a varying number of features per species. The actual documentation of trophic interactions involves using methods such as collecting stable isotopes, analysing gut content or using statistical associations between different characteristics of interacting

organisms. Furthermore, it should be noted that food web datasets represent a snapshot of trophic interactions in a particular time and/or space [4].

The geographical coverage of a food web can vary widely, from studying microcosms under controlled conditions to examining whole continents, with the web resolution typically decreasing as the spatial scale increases. Moreover, the collection of web data typically occurs over a period of time, which indicates the need for accounting temporal variability by, for instance, sampling the same web at different seasons [4].

B. *Link prediction overview*

Link prediction is a method that is concerned with the study of a graph of a network, particularly social networks. A network can be static, where nodes and edges remain the same over time. However, it is more typical for the network to be dynamic, with new nodes and edges being added to the graph continuously. The understanding of the factors that lead to the evolution of a social network is a multifaceted issue due to the large number of associated parameters. Thus, an attempt can be made to instead reduce the problem to the easier problem of understanding the relationship between two nodes. The question, then, arises: how is this relationship influenced by neighbour nodes? And how can the network be modelled using the topology of the network itself? Accordingly, the problem is now shaped into the form of analysing the prediction of the possibility of a future relationship between a pair of nodes, while knowing or assuming that there is no connection between the two nodes in the present state of the graph [5].

According to [6], the problem of link prediction can be formally defined as follows: we are given a network as the "graph G = ⟨V, E⟩ in which each edge e∈E represents an interaction between its endpoints at a particular time t(e). We record multiple interactions by parallel edges with different time-stamps. For times t < t', let G[t, t'] denote the subgraph of G restricted to edges with time-stamps between t and t'. To formulate the link prediction problem, we choose a training interval [t_0, t_0'] and a test interval [t_1, t_1'] where $t_0 < t_1$, and give an algorithm access to the network G[t_0, t_0']; it must then output a list of edges, not present in G[t_0, t_0'], that are predicted to appear in the network G[t_1, t_1']" [6].

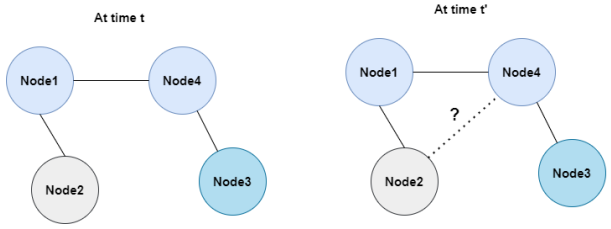


Fig. 1. Example of a simple network graph where link prediction can be implemented. Adapted from [7]

As suggested by [6], one of the first link prediction models was created for it to work exclusively on social webs. Every node can represent a person and an edge between two persons can represent the interaction between them. This multitude of interactions can be modelled by exclusively allowing parallel edges, or with a similar system using weights for the edges. This process produces the similarity value between two nodes using various measurements, and uses that similarity value to predict a link between them.

The authors of [8] continued the above research in a couple of ways. First, they showcased that using data external to the graph can improve the prediction results. Furthermore, they used similarity measurements as features in a model of supervised learning in which the problem of link prediction is defined as a binary classification process.

C. *Node embedding methods*

The domain of machine learning has exploded in recent years, owning largely due to higher processing power available to researchers at an increasingly smaller cost as well as the creation of bigger datasets, that allow these algorithms to produce better results. The subdomain of machine learning on graphs has also seen similar progress. Much attention has been given to the process known as graph representation, where the embeddings of nodes are created in a latent vector space.

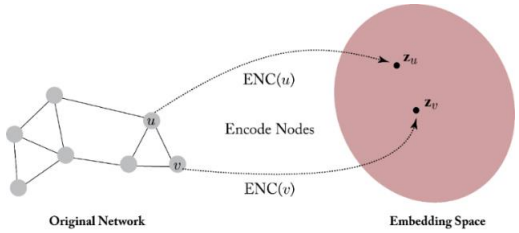


Fig. 2. Illustration of the node embedding problem. Adapted from [9]

These embedding techniques can be divided into two main categories: shallow and deep. Shallow embeddings utilise an encoder function, which generates a look-up table based on the node ID, without taking into account any features other than the existence of nodes and edges in the graph. Shallow methods also use the decoder function, whose role is to reconstruct particular graph statistics from the previously generated embeddings e.g if a pair of nodes are neighbours in the graph. Afterwards, the two functions are used to reconstruct the relationship between the pair of nodes by minimising the reconstruction loss function so that:

$$L = \sum_{(u,v)\in D} \ell\,(DEC(z_u, z_v), S[u,v] \qquad (1)$$

where:

- (z_u,z_v) is a pair of embeddings
- S[u,v] is a graph based similarity measure between nodes and
- ℓ is the loss function measuring the discrepancy between the estimated values DEC(z_u,z_v) and the true values S[u,v] [10].

Examples of these algorithms include DeepWalk by [11] and Node2Vec by [12].

In contrast, deep methods consider both the structure of the network graph as well as any associated features of the nodes, external to the graph itself, with these features differing, depending on the type of network examined. In cases where no explicit node features exist, one can use node statistics such as degree, centrality or the clustering coefficient [10].

The most general framework for deep embeddings is commonly referred to as Graphic Neural Networks (GNNs). The basic Graphic Neural Network model can be modelled as follows:

$$H^{(t)} = \sigma(AH^{(k-1)}W_{neigh}^{(k)} + H^{(k-1)}W_{self}^{(k)}) \quad (2)$$

where:

- $H^{(t)}$ denotes the matrices of node representations at layer t
- A is the graph adjacency matrix
- $W_{neigh}^{(k)}$ and $W_{self}^{(k)}$ are trainable parameter matrices
- σ is the activation function (typically ReLU) [10].

According to [13], Graphic Neural Networks can be divided into four main categories:

- Recurrent Graph Neural Networks (RecGNNs): these algorithms make the assumption that each node continuously exchanges information with its neighbour nodes up to the point that an equilibrium is reached.
- Convolutional Neural Networks (ConvCNNs): this architecture tries to create the representation of a node by aggregating both the node's features as well as the features of its neighbour nodes. ConvCNNs typically involve multiple stacked layers to extract the representations of the nodes.
- Graph autoencoders: these are unsupervised machine learning techniques that encode node representations into a latent space and then reconstruct the graph using the information generated by the encoding.
- Spatial-Temporal Graph Neural Networks: these attempt to learn hidden patterns from spatial-temporal network graphs, taking into account both spatial and temporal dependence at the same time.

D. Link prediction implementations

The authors of [14] created a link prediction process as part of a supervised machine learning environment. They introduced two new measures called *Friends measure* and *Same community* that can capture the structural features of a network graph. These measures can then be used in conjunction with nodes features and fed into a classifier such as RandomForest or Adaboost. This method was successful in finding missing links as well as in uncovering hidden links when examining datasets from large social networks such as Flickr and Facebook.

Additionally, the authors of [15] examined a subset of the Twitter and Sina Weibo networks and used the network structure as well as user text as features for their link prediction model. They used the Node2Vec algorithm to generate embeddings and an attention-based CNN for the features. Moreover, they use a time decay function to model the users' change of interests over time. In order to simplify the computations, only the users with the highest similarity scores were examined, based on the network's node topology. The authors conclude that a method that uses a combination of node embeddings and non-structural node features is ideal and is most effective on improving the performance of link prediction.

Finally, a recently published paper [16] deals with the topic of transfer learning in link prediction. The authors of [16] introduce a transfer learning model called DNformer, which can be used for the task of link prediction in dynamic networks. This algorithm takes into account the network structure as well as link similarity. The model was trained and evaluated on a variety of networks, including email networks, human contact networks and online community forums. When evaluating its performance, DNformer is found to have superior performance to its counterparts, with a difference of 1.5% to 5.1%, compared to other state of the art models such as EvolveGCN, Transformer and TGAT.

For the purposes of this project, a clear distinction is made between the definitions of node embeddings and node features. We define node embeddings as the latent space representations of a graph's nodes, which are generated by an algorithm using only the network's structure as input. Meanwhile, node features are defined as the characteristics that a node can have such as the age of a person in a social network or the size of a predator in an ecological network and which can be used in the task of link prediction [9].

From the previously mentioned papers, we can conclude that most of the research in this field has used node features, or the combination of node embeddings and node features rather than node embeddings alone for the modelling of link prediction algorithms. Instead, this paper attempts to create a link prediction model using only node embeddings. The precise methodology is discussed in the following section.

III. METHODOLOGY

A. Tools used for data processing and data modelling and sources of the implemented methods

The entirety of data pre-processing, processing and modelling was written in programming code using Python. The code was written in Jupyter notebook files (.ipynb), utilising the online Google Colab platform. The handling of the data was carried out using the Pandas library [17] while the subsequent processing and modelling was primarily done using the Numpy [18], StellarGraph [19], DeepWalk (via KarateClub) [20], NetworkX [21] and scikit-learn libraries [22], with part of the project's code being sourced from the aforementioned libraries. For data visualisation, the Matplotlib [23] and the Geopandas [24] libraries were used. The original papers of DeepWalk [11] and Node2Vec [12] were consulted on the proper implementation of node embeddings. Finally, the papers [6, 8, 16] were also consulted for the creation of the model.

B. Dataset overview

The dataset used for the purposes of this research paper is the GATEWAy, made publicly available and compiled by [2]. It presents several different food webs, each with their own unique set of species [2].

The dataset consists of 222151 rows (i.e. samples) as well as 46 columns (i.e. features). In particular, each row has two

columns, *con_taxonomy* and *res_taxonomy*, which concern a pair of two different organisms: the consumer and the resource, respectively, where the former organism is the predator and consumes the latter organism which is the prey. These two columns together represent a pair of nodes and form an edge. Within each food web, these nodes and edges form a directed network graph. Using properties from graph theory as well as more recent advances in machine learning, this project aims to use these graphs to provide insights about the relationships among the different species [2].

The rest of the columns describe different features of the consumer and the resource organisms. Some of these properties are: minimum/average/maximum length, minimum/average/maximum mass, lifestage, metabolic type, movement type, taxonomy level, geographic location, ecosystem type, foodweb name and others [2].

For the purposes of this paper, out of the available columns (i.e. features), the ones that were used for the modelling were: *con_taxonomy*, *res_taxonomy* and *foodweb_name*. The first two features were used as nodes to form a directed network graph while *foodweb_name* was utilised to split the full dataset into individual datasets according to location. The remaining 41 columns were not used [2].

From the larger dataset, a number of different food webs were selected, resulting in the creation of 7 different datasets. Each dataset can straddle across one or two different types of ecosystems. The resulting datasets and their characteristics are summarised in Table I [2].

TABLE I. SUMMARY OF THE EXAMINED FOOD WEBS [2]

Food web	Ecosystem types	Number of nodes	Number of samples
Weddell Sea	Marine	490	16041
Chesapeake Bay	Marine	521	15821
Lough Hyne	marine	346	5068
Carpinteria	marine, terrestrial aboveground	160	3960
FloridaIslandE3	terrestrial aboveground	211	3671
FloridaIslandE1	terrestrial aboveground	210	3560
Caribbean Reef	marine	247	3313

Fig. 3 below illustrates the location of the examined food webs across the world.

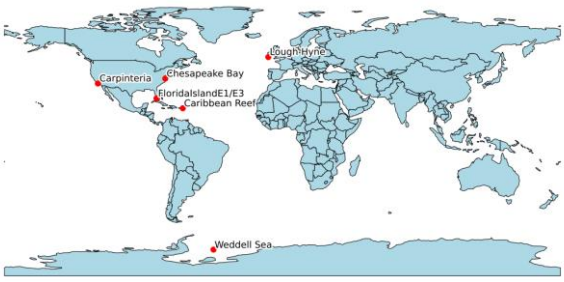


Fig. 3. Generated map of the locations of the examined food webs across the world. Data collected from [2].

The primary reason for choosing these datasets is their size. The advantage of a larger dataset is the presence of a higher number of samples and nodes from which to split the datasets into particular training and test datasets. This results in potentially better performing machine learning algorithms. In addition, a higher number of nodes and samples helps ameliorate the class imbalance problem. Alternatively, there is the possibility of grouping together multiple food webs according to geographic location and this process would also yield individual datasets with a higher number of nodes and samples. These approaches were not chosen for reasons that are detailed in the *Discussions & Conclusions* and *Defining the training and test datasets* sections.

The slicing of the larger dataset into individual ones was done using the Pandas library. Consequently, each dataset was formed into a graph using the NetworkX library before finally being turned into a StellarGraph graph object using the StellarGraph library. The final StellarGraph object is a Directed Multigraph which means that multiple edges between any pair of nodes can exist at the same time and that each edge has a direction towards one of the nodes in each pair of nodes [17, 19, 21].

C. DeepWalk

DeepWalk is an open-source Python library that "uses a randomized path traversing technique to provide insights into localized structures within networks" [11]. It utilises the SkipGram model, an algorithm used in language training models, in order to encode relations among nodes in a continuous vector space. The output of the algorithm is a latent space representation of the network. This latent space can then go through a principal component analysis and be plotted on a 2D diagram. The desired result is that the plot represents the original nodes as points that are close together if they are similar to each other [11].

The DeepWalk algorithm allows the user to choose different initial values for the number of walks and the number of dimensions. Furthermore, according to the original research paper, the relative performance between dimensions is mostly stable across different values of the number of dimensions. In addition, the paper also states that most of the benefit is accomplished by starting with 30 walks and that the effect of increasing the walk length quickly slows for values higher than 10. Consequently, for the purposes of this paper, the number of walks that was chosen was 10 and the number of dimensions chosen was 16. The process of determining the optimal values of these variables is explored in more detail in the *Determining optimal values* section. These values were kept constant across all individual datasets examined [11].

For the purpose of creating the representations of edges, binary operators are used to develop the learned feature representations of each node. Four different kinds of binary operators are examined: Average, Hadamard, Weighted-L1, and Weighted-L2. Once the operator is chosen and then used, the output is subsequently fed into the classifier.

TABLE II. BINARY OPERATORS USED FOR LEARNING EDGE FEATURES. ADAPTED FROM [12].

Operator	Definition
Average	$\frac{f_i(a) + f_i(b)}{2}$
Hadamard	$f_i(a) * f_i(b)$
Weighted-L1	$\|f_i(a) - f_i b$
Weighted-L2	$\|f_i(a) - f_i(b)\|^2$

D. Classifier

The LogisticRegressionCV classifier, as implemented in the scikit-learn library, was picked for the classification task. It uses a cross-validation technique, specifically Stratified K-Folds, to automatically select the best hyper-parameters [25]. As its hyperparameters, we use 10 folds and 200 iterations. The process of determining the optimal values of these variables is explored in more detail in the *Determining optimal values* section. Using larger values for these hyperparameters did not have any significant impact on the model.

The sensitivity and specificity of each iteration is then calculated, with the desirable results being high sensitivity and high specificity. In simple terms, with high sensitivity the model is better at correctly predicting the edges between nodes that were present in the original graph while with high specificity the model is better at correctly predicting the non-existence of edges between nodes in the original graph.

Using the sensitivity and specificity metrics, an ROC plot is drawn, using sensitivity in the vertical axis and 1- specificity in the horizontal axis. Finally, the area-under-the-curve (AUC) is calculated which serves as our metric to evaluate the performance of the link prediction algorithm. This metric has become the standard metric in the task of link prediction [26].

E. Defining the training and test datasets

For the purposes of this paper, each dataset must be split into at least two different datasets: the training dataset, where the machine learning algorithm is trained on, and the testing dataset, where the already trained algorithm has its performance evaluated according to specified metrics. This distinction is done so that we can be certain that the algorithm will perform adequately when presented with new, future data. Moreover, a high performance on the training dataset but a low performance of the test dataset indicates overfitting. Much care must be made to ensure that there is no cross contamination between training and test data as that would defeat the purpose of splitting datasets.

Furthermore, in the case processing network graphs, extra care must be placed when splitting the dataset. This is because nodes can be interconnected in a network and splitting a dataset necessarily involves cutting edges between pairs of nodes, a process that destroys part of the dataset's information.

In addition, when examining the case of link prediction, the problem of class imbalance emerges. In the link prediction problem, there are two classes: the positive class where a link exists between a pair of nodes, and the negative class where no link exists between a pair of nodes. In the majority of cases, it is more likely that any one node is only linked to a minority of other nodes and thus not linked to most of them. Consequently, this means that there exist far more negative examples than positive examples. This creates a potential problem for the classifying algorithm, as it can potentially have much better performance in the negative class than the positive class. This can be more obvious in the evaluating metrics that one chooses to use, where using a metric such as accuracy can be biased towards the negative class, and a high accuracy score won't necessarily mean that our algorithm performs as desired. In order to solve this problem, the metric of Receiver Operating Characteristic Curve – Area Under the Curve or ROC AUC is used, which takes equally into account the algorithm's predictions of both classes, using the calculated sensitivity and specificity scores that formed the ROC curve [26].

For the aforementioned reasons, the splitting of each dataset was done using the EdgeSplitter class from the Stellargraph library. This class makes sure that a true positive edge is not sampled as a negative edge. The EdgeSplitter class also makes sure that there are enough positive as well as negative samples to subsequently use in the link prediction classifier [27].

Each individual dataset is divided into three different kinds of sub-datasets:

- The Training Set, which is used for training the model. This is further divided into the Training Graph (which is used to create the embeddings from the DeepWalk algorithm) and the Training Samples. Both the embeddings and the Training Samples are used to train the model.
- The Testing Set, which is used for the final evaluation of the model. This is further divided into the Testing Graph (which is used to create the embedding from the DeepWalk algorithm) and the Testing Samples. Afterwards, both the embeddings and the Testing Samples are fed into the trained classifier and the performance of the model is evaluated.
- The Binary Operator Test Set, which is used exclusively in order to select the best binary operator function. This is further divided into the Binary Operator Testing Graph (which is used to create the embedding from the DeepWalk algorithm) and the Binary Operator Testing Samples. Out of a total of 4 binary operators (Average, Hadamard, Weighted-L1 and Weighted-L2) [11] the best performing one is selected. This process is explained in more detail in the Methodology section.

The percentage of samples split, for each dataset, is as follows: For the Testing Set, 10% of the full dataset was sampled. For the Training Set, the first step consisted of excluding the samples used in the testing data out of the full dataset and then using 10% of the remaining samples. Afterwards, the second step consisted of keeping 75% of this data for training the algorithm and using 25% for the Binary Operator Test Set. This process ensures that no class imbalance occurs at any point when using our link prediction pipeline.

F. Modelling Procedure

In the previous section, each individual part of the modelling process was described in detail. Below, we describe how the individual parts were pieced together in order:

1) Loading the necessary libraries and preparing the dataset

First, the necessary libraries were loaded. The GATEWAy dataset was then loaded from a .csv file into a Pandas dataframe. The dots contained in the columns' names are then converted to underscores for easier handling.

2) Pre-processing the dataset

The dataframe is sliced so that only the one food web is selected, forming a new, individual dataset. The individual dataset is loaded into NetworkX as a directed graph, with the columns *con_taxonomy* and *res_taxonomy* representing the pairs of nodes. The resulting NetworkX object then has its nodes converted into integers and is then loaded into the StellarGraph library as a directed multigraph.

3) Splitting the data

The EdgeSplitter function is used on the individual dataset, splitting it into training the Training Set, the Binary Operator Test Set, and the Testing Set.

4) Training and Binary Operator Selection Phase

The Training Set is the first set to go through the modelling process. The training graph goes through the DeepWalk algorithm, returning the nodes embeddings. The embeddings are then linked with the Training Test samples using a binary operator. Subsequently, the model is trained using the LogisticRegressionCV classifier. This process is done for all four operators.

The trained model is now tested on the Binary Operator Test Set. Its graph first goes through the DeepWalk algorithm, returning the nodes embeddings. The embeddings are then linked with the Training Test samples using a binary operator. The trained model is now evaluated on the processed Binary Operator Test Set. Subsequently, the ROC AUC score is computed. This process is repeated for all four operators. Finally, their scores are compared, and the highest performing operator is selected.

5) Testing Set Evaluation Phase

The Testing Set now goes through the model. Its testing graph first goes through the DeepWalk algorithm, returning the nodes embeddings. The embeddings are then linked with the Testing Set samples using a highest performing binary operator used in the previous step. The trained model is now evaluated on the processed Testing Set. Subsequently, the ROC AUC score is computed, and the ROC curve is drawn. The results are displayed in the Jupyter Notebook and recorded for later usage.

6) Repeating steps for each individual dataset

Steps 2 through 5 are repeated for all the individual datasets examined. A new model is trained for each web.

7) Displaying the resutls and writing to file

The names of the examined individual datasets, along with their ROC AUC scores and the binary operator that they used are displayed in a Pandas dataframe and are then written into a .csv file.

G. Determining optimal values

In order to determine the optimal input values for the various variables used, the model was tested with different initial values. Four variables (i.e. hyperparameters) were chosen for the analysis: number of folds and number of iterations for the classifier and number of walks and number of dimensions for the DeepWalk algorithm. The values tested are presented in Table III below.

TABLE III. HYPERPARAMETERS AND THEIR TESTED VALUES

Hyperparameter	Part of	Values tested
Number of folds	Classifier	5, 10, 20
Number of iterations	Classifier	2000, 2500, 5000
Number of walks	DeepWalk	5, 10, 30, 90
Number of dimensions	DeepWalk	16, 64, 128, 256

A nested loop was used to feed the various initial values of the hyperparameters to the model, using the average of the ROC AUC score of the food webs of each run as the desired metric. The results of each run were then recorded in a .csv file. Moreover, as stated in a previous section, because the random training-testing splitting process can sometimes result in errors, the state of whether an error occurred in a run was recorded. The output was then used to create a correlation coefficient matrix, which can be seen in Fig. 4 below.

	Average of ROC AUC scores	Folds	Iterations	Walks	Dimensions	Error occurred
Average of ROC AUC scores	1.000000	-0.017470	-0.102931	-0.273358	-0.589225	0.095380
Folds	-0.017470	1.000000	-0.000000	0.000000	-0.000000	-0.094491
Iterations	-0.102931	-0.000000	1.000000	-0.000000	-0.000000	-0.058926
Walks	-0.273358	0.000000	-0.000000	1.000000	-0.000000	0.008439
Dimensions	-0.589225	-0.000000	-0.000000	-0.000000	1.000000	0.113024
Error occurred	0.095380	-0.094491	-0.058926	0.008439	0.113024	1.000000

Fig. 4. Correlation coefficient matrix of the tested variables.

From the above figure, we can determine that the number of walks and the number of dimensions are moderately negatively correlated on the ROC AUC metric, with the other variables having negligible impact. This suggests that we should use a low number of walks and dimensions. Similar results occur if we limit ourselves to runs where an error didn't occur.

Based on the above analysis, the values presented in Table IV were selected for all subsequent model trainings:

TABLE IV. HYPERPARAMETERS AND THEIR CHOSEN VALUES

Hyperparameter	Part of	Value used
Number of folds	Classifier	10
Number of iterations	Classifier	2000
Number of walks	DeepWalk	10
Number of dimensions	DeepWalk	16

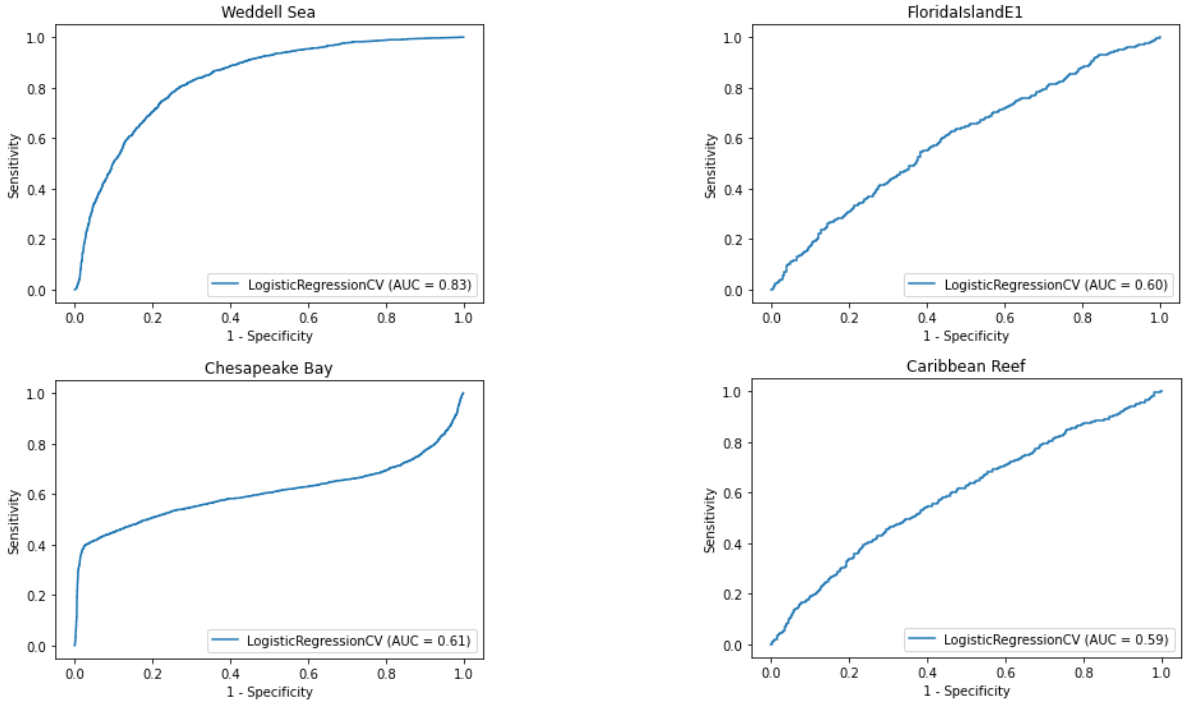
H. Results Reporting

The results of the link prediction for the various food webs that were examined are summarised in Table V below.

TABLE V. ROC AUC SCORES AND FINAL OPERATORS USED FOR THE EXAMINED WEBS

Food web	ROC AUC score	Operator used
Weddell Sea	0.83	Hadamard
Chesapeake Bay	0.61	Hadamard
Lough Hyne	0.77	Hadamard
Carpinteria	0.65	Hadamard
FloridaIslandE3	0.65	Hadamard
FloridaIslandE1	0.60	Average
Caribbean Reef	0.59	Average

The ROC curves drawn for the examined food webs are presented in Fig. 5 below.



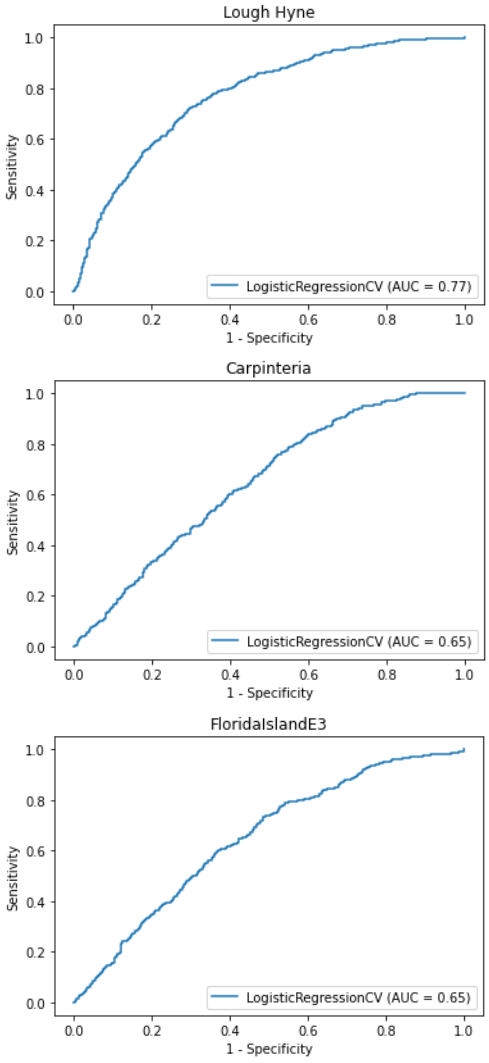


Fig. 5. The ROC curves of the examined food webs.

Overall, the link prediction algorithm appears to perform moderately well. The overall score ranges from 59% to 83% depending on the food web examined. It should be noted that an ROC AUC score of 50% is no better than random and a score lower than 50% is worse than random. We note that most of the link prediction tests used Hadamard as the optimal operator. Finally, we note that webs with higher number of samples and nodes tend to have higher ROC AUC scores.

I. Discussions & Conclusions

The lack of high scores is likely to be due to two factors. First, the size of the datasets is relatively small, with only 490 nodes and 16041 samples in the biggest data set. This means that there wasn't nearly enough data to sample a high percentage of the nodes for both the positive and negative classes, exacerbating the class imbalance problem. Indeed, some food webs couldn't be sampled due to this and thus weren't considered for training the model. Moreover, some of the largest food webs did also exhibit this problem on some trial runs, even though they could normally be run with the same setup in most runs, due to the random nature of the sampling done by the EdgeSplitter command.

Second, the link prediction is done without using any features from the full dataset but rather using only the nodes and the embeddings of the nodes as given by the DeepWalk algorithm. As discussed previously, most research in the field does use node features in order to achieve higher accuracies.

Moreover, an additional option regarding the datasets examined would be to select groups of food webs instead of single food webs as individual datasets. During the exploratory process, this approach was initially considered. However, a number of difficulties emerged. First, the rationale that individual food webs have individual species as well as pairs of prey-predator species in common can be used to link different food webs together. But this approach can link webs that are far away from each other, even when the distances

involved are vast. For example, the Lough Hyne food web and the Carpinteria food web have the *Mytilus galloprovincialis* predator species in common. However, it's unlikely that in reality the food webs interact, whether at present or in the future, since they are located over 10000 kilometres apart, with the first located in Ireland and the latter located in the eastern US seaboard. Similarly, combining all 290 webs into one composite web was also not was considered.

Alternatively, the column *geographical_location* could be used to group food webs that share the same value i.e. that share the same general geographic location. This can be useful since it would create individual datasets with larger sample sizes. However, this feature is not sufficient. For example, the resulting Germany food webs grouping would end up connecting food webs all over Germany, whereas the foods webs tend to cluster in three different locations, each with a different type of environment as can be seen in Fig. 6. Therefore, one cannot rely on the *geographic_location* value alone.

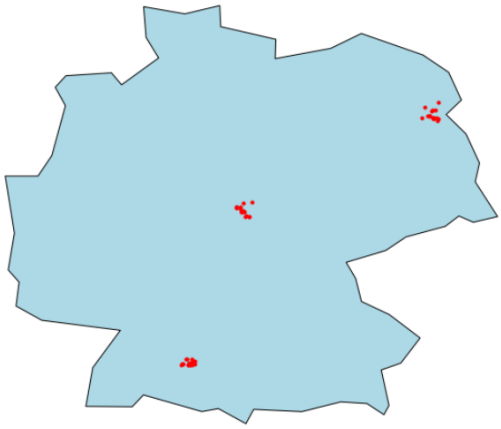


Fig. 6. Generated map of the food webs located in Germany. Data collected from [2].

This leaves the option of accounting for the distances among the food webs and grouping the closest clustering ones together, provided that they still have species in common. Using this approach leaves us with fewer food webs to choose from. This yields groupings with fewer samples, with these grouping having fewer samples than most of the biggest individual food webs. Using the column *study_site* instead of *geographic_location* results in similar difficulties.

Nevertheless, an attempt was made to test such groupings of food webs even if they might not actually be connected in reality. Regardless, this did not improve the ROC AUC scores. Therefore, the decision was made to use the largest individual food webs.

The process followed in this paper can be compared and contrasted with another approach followed by [28] which used the same GATEWay dataset. In his approach, the author primarily examined a composite graph, where he combined a high number of individual food webs into a single web. Additionally, the author used a graph autoencoder to train his model, using a Convolutional Neural Network model. Moreover, he also made use of the dataset's features, both numerical and categorical, such as mean mass, mean length, life stage among others as well as food web level features such as altitude, ecosystem types and others. The results indicate that this method is promising, with final ROC AUC scores ranging between 85% to 90%.

The aforementioned method achieves better results over the model implemented in this paper. But there remain a few things to note. First, the model is fed a lot more information beyond just the structure of the resulting network graph. It is, therefore, unsurprising that a model that has been fed more information, using node features, can achieve better performance. However, the approach is much more resource heavy, taking over 8 hours in the author's machine compared to 10 to 45 minutes in our model, depending on the parameters chosen.

Additionally, the number of samples in the examined dataset is much larger, owning to the fact that it combines multiple webs into one. As discussed earlier, it is this author's opinion that this approach might not be adequate. Nevertheless, one should note that having a larger dataset often results in a better performing model, even compared to an ideal model that is trained on fewer data [29, 30].

In conclusion, it is the author's opinion that this project accomplishes many of its stated goals. The trained model could potentially be used to uncover previously unknown pairs of predator-prey species, thus updating the known biodiversity of a food web. However, the model is not accurate enough to replace part of the food web collection process and thus being of limited utility to field researchers.

J. *Future work*

This paper examined the training of a model that does link prediction on ecological networks, with the resulting algorithm performing moderately well on the given task. Based on the experience acquired from conducting this project as well as the insights gained from the results, we note the following:

It is of high importance that when a new ecological dataset is created, one should focus on gathering as many samples as possible so that a larger dataset can be created. At the risk of mentioning the obvious, there might not exist enough pairs of prey-predator species in a given environment to accumulate more data nor is it a given that current data gathering methods are sufficient to collect all possible samples, even in theory. Nevertheless, in many cases, the number of samples could be increased by expanding the size of the area examined, at the cost of increased labour.

Moreover, out of the four binary operators used, the Hadamard operator is clearly the one with the superior performance in most training scenarios. More research is needed to understand why this operator seems to work the best.

Furthermore, in order to achieve a highly accurate link prediction model, it is not always enough to rely on just network graph statistics or their latent space representations. One should also include any additional numerical or categorical features that are available. However, using merely the pairs of nodes can be enough to create a moderately accurate model. Future research on this approach could potentially yield a training pipeline that achieves even better gains in the task of link prediction using just the structure of the network graph, saving on computing time.

ACKNOWLEDGMENTS

This dissertation project could not have been carried out without the encouragement and advocacy of my supervisor Dr. Athen Ma. Her advice and insights played an essential role to the completion of this project.

Additionally, the present paper was made almost entirely using free and open-source Python libraries. The author thanks the contributors of these projects for their support of the open-source-software movement, with the fields of data science and data analytics being indebted to their tireless work.

REFERENCES

[1] Montoya, J.M., Pimm, S.L. & Solé, R.V., 2006. Ecological Networks and their fragility. Nature, 442(7100), pp.259–264.

[2] Brose, U. et al., 2019. Predator traits determine food-web architecture across ecosystems. *Nature Ecology & Evolution*, 3(6), pp.919–927.

[3] Kumar, M. et al., 2022. CFLP: A new cost based feature for link prediction in Dynamic Networks. *Journal of Computational Science*, 62, p.101726.

[4] Mestre, F. et al., 2022. Disentangling food-web environment relationships: A review with guidelines. *Basic and Applied Ecology*, 61, pp.102–115.

[5] Hasan, M.A. & Zaki, M.J., 2011. A survey of link prediction in Social Networks. *Social Network Data Analytics*, pp.243–275.

[6] Liben-Nowell, D. & Kleinberg, J., 2003. The link prediction problem for Social Networks. Proceedings of the twelfth international conference on Information and knowledge management - CIKM '03.

[7] Daud, N.N. et al., 2020. Applications of link prediction in Social Networks: A Review. *Journal of Network and Computer Applications*, 166, p.102716.

[8] Al Hasan, M. et al., 2006. Link Prediction Using Supervised Learning. *SDM06: workshop on link analysis, counter-terrorism and security*, pp.798–805.

[9] Bianconi, G., Pin, P., & Marsili, M. (2009). Assessing the relevance of node features for network structure. *Proceedings of the National Academy of Sciences*, *106*(28), 11433–11438. https://doi.org/10.1073/pnas.0811511106

[10] Hamilton, W.L., 2020. *Graph representation learning*, San Rafael, California: Morgan and Claypool Publishers.

[11] Perozzi, B., Al-Rfou, R. & Skiena, S., 2014. Deepwalk. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.

[12] Grover, A. & Leskovec, J., 2016. node2vec: Scalable Feature Learning for Networks. *arXiv*.

[13] Wu, Z. et al., 2021. A comprehensive survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1), pp.4–24.

[14] Fire, M. et al., 2013. Computationally efficient link prediction in a variety of social networks. *ACM Transactions on Intelligent Systems and Technology*, 5(1), pp.1–25.

[15] Xiao, Y. et al., 2021. Link prediction based on feature representation and Fusion. *Information Sciences*, 548, pp.1–17.

[16] Jiang, X. et al., 2022. DNFORMER: Temporal link prediction with transfer learning in Dynamic Networks. *ACM Transactions on Knowledge Discovery from Data*.

[17] McKinney, W., 2022. Pandas documentation. *pandas documentation - pandas 1.4.3 documentation*. Available at: https://pandas.pydata.org/docs/ [Accessed August 12, 2022].

[18] Oliphant, T., 2022. NumPy documentation. *NumPy documentation - NumPy v1.23 Manual*. Available at: https://numpy.org/doc/stable/ [Accessed August 12, 2022].

[19] Data61, 2022. Stellargraph documentation. *Welcome to StellarGraph's documentation! - StellarGraph 1.2.1 documentation*. Available at: https://stellargraph.readthedocs.io/en/stable/ [Accessed August 12, 2022].

[20] Rozemberczki, B., 2022. Karate Club documentation. *Karate Club Documentation - karateclub documentation*. Available at: https://karateclub.readthedocs.io/en/latest/ [Accessed August 12, 2022].

[21] Hagberg, A., Swart, P. & Schult, D., 2014. NetworkX documentation. *NetworkX documentation - NetworkX 1.9 documentation*. Available at: https://networkx.org/documentation/networkx-1.9/ [Accessed August 12, 2022].

[22] Cournapeau, D., 2022. scikit-learn Machine Learning in Python. *scikit*. Available at: https://scikit-learn.org/stable/ [Accessed August 12, 2022].

[23] Hunter, J.D., 2022. Matplotlib 3.5.3 documentation. *Matplotlib documentation - Matplotlib 3.5.3 documentation*. Available at: https://matplotlib.org/stable/index.html [Accessed August 12, 2022].

[24] Jordahl, K., 2022. Documentation. *Documentation - GeoPandas*. Available at: https://geopandas.org/en/stable/docs.html [Accessed August 12, 2022].

[25] Cournapeau, D., 2022. sklearn.linear_model.LogisticRegressionCV. *scikit*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html [Accessed July 3, 2022].

[26] Menon, A.K. & Elkan, C., 2011. Link prediction via matrix factorization. *Machine Learning and Knowledge Discovery in Databases*, pp.437–452.

[27] Data61, CSIRO, 2017. Source code for stellargraph.data.edge_splitter. *StellarGraph*. Available at: https://stellargraph.readthedocs.io/en/latest/_modules/stellargraph/data/edge_splitter.html [Accessed June 25, 2022].

[28] Egbelo, T., 2021. Link Prediction in Food Webs Using Graph Neural Networks. dissertation.

[29] Hoffmann, J. et al., 2022. Training compute-optimal large language models. *arXiv.org*. Available at: https://arxiv.org/abs/2203.15556.

[30] Abu-Mostafa, Y.S., Magdon-Ismail, M. &; Lin, H.-T., 2012. *Learning from data: A short course*, s. l.: AML.