UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANA PAULA CAROLINO DE OLIVEIRA MELLO

Use of embedding concatenation and ensemble to improve node classification on graphs

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dra. Mariana Recamonde
Mendoza

Porto Alegre
June 2021

ABSTRACT

Artificial intelligence (AI) is a powerful tool that can be used in several different fields to solve many problems, and its use has been increasing every year. However, traditional machine learning (ML) algorithms have a specific limitation: their input format. Since they expect the input to be in vectors and matrices, data that is best represented by graphs can not be easily used to train ML models, even though they could often be the best alternative researchers have. This hurdle inspired the creation of a set of algorithms for a process called embedding, which maps graph data to a vector space, allowing the data to be fed to ML methods with ease. Embedding, however, does not yield a perfect representation since there is an inherent trade-off in the process. Embedding algorithms have to choose to preserve one out of two characteristics of a graph: community (the neighborhood of each node) or structure (the role each node has in the graph structure). Algorithms have to focus on one aspect over the other or attempt to balance them in the representation, resulting in shallower preservation of both. This means essential aspects of a graph can be lost in translation, which can yield bad results purely because of the type of representation chosen. It can also mean that the results could improve by making the graph representation more complete. Inspired by this observation, we propose a combination of two ideas aiming at improving the representation of graph data to be used in ML algorithms. The first is a simple concatenation of three types of embeddings, each using a different embedding strategy, and the second is the use of a bootstrap aggregation ensemble for the task. To evaluate these approaches, we run experiments on six datasets comparing the performance of the proposed approaches against simple classifiers trained on each embedding separately. Our results suggest that, while the concatenation does not have the best results, it constantly gets very close to it in all tested datasets, which does not happen with individual embeddings.

Keywords: Machine learning. ensemble. embeddings. graphs. node classification.

Uso de concatenação de embeddings e ensemble para melhorar a classificação de nodos em grafos

RESUMO

Inteligência artifical (IA) é uma ferramenta poderosa que pode ser usada em diferentes áreas para resolver vários tipos de problemas, e seu uso vem aumentando a cada ano. Porém, algoritmos tradicionais de aprendizado de máquina (AM) possuem uma limitação específica: o formato de entrada dos dados. Como eles esperam que a entrada esteja na forma de vetores e matrizes, dados que são melhor representados por um grafo não podem ser facilmente utilizados para treinar modelos de AM, mesmo quando podem ser a melhor alternativa para pesquisadores. Esse obstáculo inspirou a criação de um conjunto de algoritmos para um processo chamado *embedding*, que mapeia dados de um grafo em um espaço vetorial, permitindo que esses dados sejam passados para modelos de AM com facilidade. *Embeddings*, no entanto, não geram uma representação perfeita, já que existe uma relação inversa inerente ao processo. Os algoritmos precisam escolher preservar uma de duas características de um grafo: comunidade (a vizinhança de cada nodo) ou estrutura (o papel que cada nodo tem na estrutura do grafo). Eles precisam focar em um aspecto em detrimento do outro, ou precisam tentar balanceá-los na representação, resultando em uma preservação pior de ambos. Isso significa que aspectos importantes de um grafo podem se perder, o que pode gerar resultados ruins para uma tarefa de classificação ou de predição apens por causa do tipo de representação escolida. Isso também pode significar que os resultados podem melhorar caso a representação do grafo seja mais completa. Inspirados pelo conceito, propomos a combinação de duas ideias para tentar melhorar a representação de grafos para serem usados em algoritmos de aprendizado de máquinas. A primeira é uma concatenação simples de três tipos de *embedding*, cada um focando em uma característica específica, e a segunda é o uso de um *ensemble boostrap aggregation* para a tarefa. Para avaliar as abordagens, nós rodamos experimentos com seis conjuntos de dados copmarando a performance das abordagens propostas com a de clasificadores simples treinados em cada *embedding* separadamente. Nossos resultados mostram que, apesar de a concatenação não ter os melhores resultados, ela constamentemente fica perto dos melhores em todos os datasets testados, o que não ocorre com *embeddings* individuais.

Palavras-chave: aprendizado de máquina, *embedding*, *ensemble*, grafos, classificação de nodos.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artifical intelligence
ML	Machine Learning
GCN	Graph Convolutional Networks
SDNE	Structural Deep Network Embedding
SVC	Support-vector classification
SVM	Support-vector machine
PPI	Protein-protein interaction
GAE	Graph Auto-Encoder

CONTENTS

1 INTRODUCTION

Artificial intelligence (AI), and more specifically its subset machine learning (ML), is a powerful tool that has been used to solve many complex problems in several different fields. It has been used for spam filtering (GUZELLA; CAMINHAS, 2009), speech (DENG; LI, 2013) and image recognition (PAK; KIM, 2018), personalized recommendations (CHU; PARK, 2009), medical data analysis (KONONENKO, 2001), credit scoring (WANG et al., 2011), disease prediction (JADHAV et al., 2019), among many others. Over time, it has become more and more present in our lives, and this trend shows no sign of slowing down.

For all their usefulness, however, traditional ML algorithms have a specific limitation: their input format. Almost all of them expect data to be fed to them as vectors and matrices, which makes it difficult to use these methods when data does not easily fit this format. Biological networks, such as protein-protein interaction (PPI) networks and gene regulatory networks, are good examples of this, since they are represented by graphs. The tasks to be performed on these graphs, such as classification and prediction, are extremely suited for ML algorithms, but the data itself is not, at least not in its original format.

Because of this hurdle, researchers developed algorithms for a process called embedding, which can map nodes, clusters of nodes and even whole graphs into a vector space (CAI; ZHENG; CHANG, 2017). This representation allows graph data to be used as input to traditional ML methods, not requiring big changes to existing implementations just to feed them data. This process, however, has some shortcomings. The biggest one is an inherent trade-off when defining the exact algorithm to generate the embedding: they can either preserve community (the neighborhood of each node) or structure (the structural role a node has in the graph as a whole) (HAMILTON; YING; LESKOVEC, 2017). Even algorithms that try to balance both of them have to deal with the trade-off, prioritizing one over the other or preserving both types of information in a shallower way than embeddings that focus on one of the characteristics.

The nature of this trade-off means different datasets can have much better or much worse results on node classification or link prediction tasks depending on the embedding method chosen and the specific characteristics of the graph. This can make good models have bad results simply due to a mistake in interpreting which characteristic would be more important for a specific task on a specific network. One of the state-of-the-art methods, node2vec (GROVER; LESKOVEC, 2016), attempts to work with this trade-off by

allowing some fine-tuning of parameters that control what information it preserves. Their results showed an improvement when compared to other algorithms, showing that there are interesting results when trying to enhance the vision of the graph so it can give the ML models a better, more accurate representation of the data.

Inspired by this concept, in this work we propose a combination of two ideas as means to improve the representation of graph data to be used in ML algorithms. The first one is a concatenation of three types of embedding, each focusing on preserving a specific property of the graph in order to generate a more accurate representation of the data as a whole. The second is the use of a bootstrap aggregation ensemble as the classifier, since the idea behind that ensemble is to get a better result by having a classifier that gets a better view of the data by using several samples for training.

To test this, we run four types of experiments for six datasets in order to compare the influence of the concatenation, of the ensemble and the combination of them on a classification task, using a simple classification on each separate embedding as the baseline. We also present the results of a soft vote between the predictions of each individual embedding.

The work is structured as follows. In chapter 2, we explain the theoretical background needed to understand the proposal. We show works that use similar ideas or techniques on the subject in chapter 3. In chapter 4 we explain the proposal itself, in chapter 5 we show how it was implemented and in chapter 6 we present all results in detail. Finally, we conclude with a discussion of the results in chapter 7 and some suggestions for future work.

2 THEORETICAL BACKGROUND

2.1 Machine Learning

Machine learning is the study of algorithms that are able to improve themselves automatically, without any direct interference from humans (MITCHELL, 1997). It is considered a subset of the field of artificial intelligence, which is usually defined as the study of intelligent agents: something that can perceive its environment and take actions to maximize the chance of success in its goal (RUSSELL; NORVIG, 2020). The field is closely related to computational statistics, mathematical optimization and data mining, and it aims to develop algorithms that generalize the ability to perform certain tasks on a completely new data set based on the experience of performing them on a training set.

Since the results are based on previous data, it is important to note that every ML model will have errors which can be measured by two different concepts: bias and variance. Bias refers to assumptions models make about the data, which can lead to wrong predictions, while variance is the sensitivity the model has to differences in the training data, which can lead to predictions errors due to noise in the data. It is important to observe these two characteristics, since high bias can lead to an oversimplified classifier that can not predict more complex data very well, and high variance can lead to overfitting, where the classifier is so sensitive to any small change in the data that it is unable to generalize its learning to work accurately with new inputs.

ML algorithms are used to make predictions that were not explicitly programmed, and they are commonly used in situations where a traditional algorithm would be too hard to define, usually due to size of the data or the complexity of the task itself. They can be classified by the type of feedback provided in order to improve on its results, with the supervised and the unsupervised approaches being the most used ones.

A supervised approach gives the algorithm a series of example inputs (x) and their desired outcomes (y), usually used to learn a general rule $f(x) = y$ that maps inputs to outputs. This rule is learned through constant optimization, which is run in a self feeding loop until it reaches a specific threshold, either a maximum number of iterations or a minimum improvement in the current best result from one iteration to the next. Some examples of supervised learning algorithms are linear regression, decision trees, and ensemble learning.

A common supervised learning task is classification, a process where data can be

separated into two or more classes (or labels) based on the previous classification similar input had. When the data can only be sorted into two classes, this is a binary classification problem. When there are more, it is called a multi-class classification problem. Because a lot of classification algorithms are made for binary classification, they are often adapted to work with multi-class problems by turning them into several binary classifications, where the classification is done based on whether a data can be categorized for a specific class or not.

In order to evaluate how good is the model being used, we can extract some metrics from the results. They are usually based on the concepts of true positive (when the prediction that data belongs to a class is correct), false positive (when the prediction that data belongs to a class is wrong), true negative (when the prediction that data does not belong to a class is correct) and false negative (when the prediction that data does not belong to a class is wrong). The most common metrics are accuracy (the percentage of right predictions made by the model), precision (the percentage of positive predictions that were correct), recall (the percentage of positives that were correctly predicted) and F1 score (the harmonic mean of the precision and the recall).

An unsupervised algorithm, on the other hand, does not receive any examples of input-output parings, and instead tends to look for some structure in the given data, like patterns or clustering. The input does not have any labelling or prior classification, so instead of improving through direct feedback it reacts to similarities found in the data. K-means clustering and hierarchical clustering are common examples of unsupervised models.

2.1.1 Support-vector classification

Support-vector classification (SVC) is a learning method that can be used for classification tasks based on the Support Vector Machine (SVM) algorithm (CORTES, 1995). It constructs a hyperplane (or a series of hyperplanes) in order to split the data into general classes by using it as a boundary, trying to make the smallest distance between the boundary and the elements of the classes as large as possible. This distance is called functional margin, and its size is a good indication of the generalization the method was able to achieve, since the distance between the members of the classes is as large as possible.

In order to improve its generalization capability, the data is mapped onto a much higher dimensional space, since the original dimension the data is in might not allow it

to be linearly separated. This is done using kernel functions so the algorithm does not require too much computational resource in this step.

2.1.2 Ensemble learning

Ensemble learning is a type of supervised machine learning algorithm that aims to improve upon the results of selected base algorithms (DIETTERICH, 2000). It combines several models in order to attempt to get the best parts of the prediction made by each algorithm, yielding a better model than one that uses only one predictor.

The justification for this algorithm comes from the "wisdom of the crowds", the idea that, in general, the average answer of a crowd will be closer to the real answer than what a single random person would give you (SUROWIECKI, 2004). The ensemble applies this logic to machine learning: the combination of several predictors will generally be better than a single predictor.

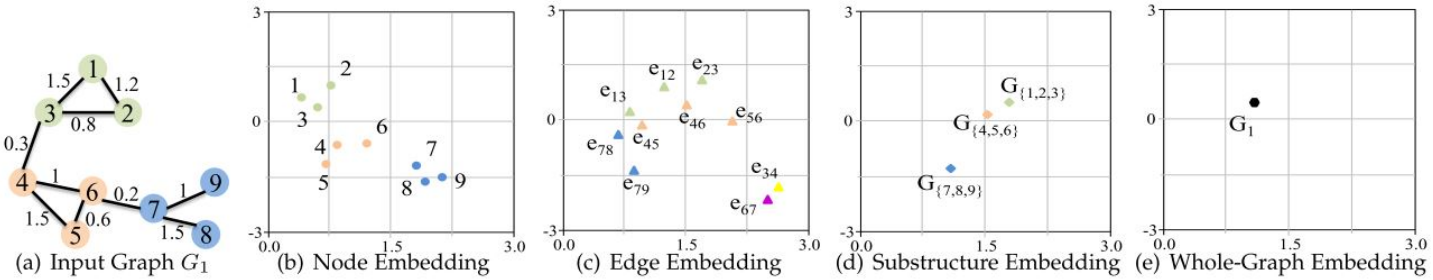
The general algorithm for an ensemble is to run a base learner several times on a dataset, with some pre-defined mechanism to generate diversity among models, combining the predictors to yield the final result. How many types of learners are used, if they run serially or concurrently, if they are run on the whole dataset or on random samples of the dataset, all of this is defined by the type of ensemble chosen. There are multiple types, but the three main ones are bootstrap aggregating, boosting, and stacking.

Bootstrap aggregating, also called bagging, is a method that focuses on getting a result with less variance than its components, using independent classifiers to create a more robust model (BBEIMAN, 1996). It starts by using a statistical technique called bootstrapping, generating samples of size n from a initial dataset by randomly drawing with replacement. Each sample is then used to train a classifier, which sees the bootstrap as an independent dataset (thus the learners can be trained at the same time). After the models are trained, their results are aggregated, yielding the final result for the ensemble. This aggregation can be done in several ways, such as simple averages (for regression tasks) or majority vote (for classification tasks).

2.2 Machine learning on graphs with node embedding

Machine learning algorithms typically receive data in the form of vectors, making it hard to use graph data as input, since they are defined by nodes and edges and can not be easily represented as vectors. However, these techniques are very useful to analyze graphs, and they are usually more efficient then traditional graph specific algorithms when the dataset is large. The great results obtained with machine learning in tasks like node classification and link prediction create a need to represent graphs in a way those algorithms can use. This is done by a process called embedding (CAI; ZHENG; CHANG, 2017), which maps graph data to a vector space of any size where every point in the vector space will represent one piece of data, as it is shown in figure 2.1.

Figure 2.1: An example of a graph embedded in a 2D space using different embedding concepts



Source: Cai, Zheng and Chang (2017)

Typically, most algorithms will embed single nodes, but it is possible to have those points representing links, node clusters and even the graph itself, which is shown in figure 2.1. The graph shown in (a) has its nodes embedded in (b), its edges embedded in (c), its substructures embedded in (d) and the entire structure embedded in (d). The choice of what will be embedded is important since algorithms will not necessarily yield embeddings that are suitable for every type of task, and this can have great effects on the final results. This work will discuss only node embedding, as it is the focus of our approach.

When it comes to node embedding, there is an important question to be asked before choosing a specific method: what does it mean when two nodes are represented by points that are close to each other in the vector space? There are two answers for this, each one representing a specific graph property that was preserved (HAMILTON; YING; LESKOVEC, 2017). The first is that two close points mean the nodes have a small minimum path between them, which means the embedding is preserving the community. The other is that those nodes have very similar structural functions in the graph, thus the

embedding preserves the structure.

The structure of a graph relates to the general shape of the graph, meaning that nodes with a similar function in the graph have similar roles (HENDERSON et al., 2012). For example, nodes that have several connections have similar roles, just as nodes with only one connection to the rest of the graph. This property tends to be useful for tasks such as link prediction or link classification.

The community, on the other hand, is about the nodes' immediate neighborhood (FORTUNATO; CASTELLANO, 2007). In general, a node is said to be another's neighbor when the minimum path to it has size one, but this can be extended to nodes that have a path of size two or even size three. Preserving this property is most useful for tasks like node classification.

There is usually a trade off between those two properties, meaning we have to choose between preserving one of them more than the other. There are some algorithms that will let you choose how much of each one you want to represent, like node2vec (GROVER; LESKOVEC, 2016), but most techniques favor one over the other. Therefore, you need to be aware of which one is more important for the problem you want to solve. Just like the type of data that will be embedded, this is something that will have a big impact, and preserving the wrong information can negatively impact your results. This work will discuss embedding techniques regardless of the property they preserve.

There are three broad categories that most embedding algorithms fall into: matrix factorization, random walk, and graph neural networks (CAI; ZHENG; CHANG, 2017). In what follows we briefly review these categories.

2.2.1 Matrix factorization based

Matrix factorization is a mathematical operation where a matrix is decomposed into two other matrices that, when multiplied, will return the original one. The first studies on node embedding focused on matrix factorization approaches (BELKIN; NIYOGI, 2001), (SHEPARD et al., 1994), inspired by dimensionality reduction techniques. The main idea was that obtaining embeddings is equivalent to a structure preserving dimensionality reduction problem, so techniques used to solve those problems could also yield good vector representations for graphs (CAI; ZHENG; CHANG, 2017). To do so, the connections between nodes are represented as a matrix, which is then factorized in such a way that one of the resulting matrices can be used as an embedding (GOYAL; FERRARA,

2017).

There are several matrices that can be used to represent those connections, such as node adjacency matrix, Laplacian matrix and node transition probability matrix. The approach to the factorization itself is informed by properties of the matrix chosen. A Laplacian matrix can use eigenvalue decomposition, for example (BELKIN; NIYOGI, 2001).

These earlier methods often required large memory and computational resources due to the use of a matrix structure, which is a major problem when using larger datasets. However, there are newer methods that are based on factorization take into account the sparsity of real-world networks (OU et al., 2016) (GOOGLE et al., 2013), which allow those techniques to scale better.

2.2.2 Random walk-based

Random walk methods are inspired by the Skip-gram, a neural network architecture commonly used for neural language processing (MIKOLOV et al., 2013). A Skip-gram is a neural network used to generate word embeddings, a similar concept to node embedding where words are represented as points in a vector space, using a sentence dictionary as context to train a model. The most notable aspect of the Skip-gram is that the embeddings are not the neural network output after training, they are the hidden layer's weights represented as vectors.

Because of this, the Skip-gram's hidden layer's definitions are highly connected to both the input and the size wanted for the embedding vector. The input layer has a node for every word that will be embedded, and it generally receives a one-hot vector (a vector set to zero in every row except the one corresponding to the word you want). The number of hidden layers is determined by the number of features an embedding will have, and each layer has the same number of nodes as the input layer. The output layer also has the same number of nodes, and each one will return the probability that a random chosen word will be a specific word from the input dictionary. Training for that means that every row in the hidden layer's weight matrix can be selected by the one-hot vector used as input to be used as an embedding for a specific word.

Random walk methods work with the same principle. The name of this category comes from how those methods turn graph data into the sentence dictionary a Skip-gram architecture needs: a series of random walks are performed on the graph, and the resulting

sequence of nodes is the equivalent of a sentence in neural language processing problems. This way, a dictionary of random walks can be used to train the network, and later it can be used to select the node embedding as if each node was a word.

Random walk algorithms mainly differ from each other in their random walk strategy, since the core Skip-gram tends to remain the same. Some algorithms, like node2vec (GROVER; LESKOVEC, 2016), allow you to determine which graph property (structure or community) you want to preserve more by configuring random walk parameters, while others, like DeepWalk (PEROZZI; AL-RFOU; SKIENA, 2014), have a more fixed approach to their strategy.

2.2.2.1 node2vec

node2vec (GROVER; LESKOVEC, 2016) is a random walk embedding algorithm that tries to capture both structure and function in its representation. The data used to train the model is obtained by a series of 2nd-order random walks, in which the nodes visited by the walk will constitute a sample. This movement is controlled by two parameters, p (return rate) and q (exploring rate). p controls the likelihood of immediately returning to a node you already visited in the walk, so the higher the p value is, the more likely it is to choose vertices that haven't been visited before. q, on the other hand, controls the likelihood of choosing to go to nodes that are further from the one previously visited than to nodes that are closer. The higher the q value is, the more it tends to choose nodes that are closer to the previous one.

These parameters are what allow node2vec to balance structure and function. Instead of being restricted to a breadth-first or a depth-first search when sampling the nodes, which respectively captures only a node's immediate neighbourhood or its position on the graph's macro-structure, this random walk strategy can interpolate between those two and therefore keep both types of information in its final representation for each node.

2.2.2.2 DeepWalk

Like node2vec, DeepWalk (PEROZZI; AL-RFOU; SKIENA, 2014) is an embedding algorithm that uses the random walk approach to generate a vector representation for graphs. Unlike node2vec, however, it leverages only the local information of a node to do so, resulting in an embedding that captures community more than the global structure of the graph.

Due to the focus on local neighborhoods, each node will generate γ short random walks as the starting node. These walks use a uniform probability distribution, so each possible node that can be reached during the walk will have an equal chance of being chosen as the next step, and there is no mechanism to prevent the return to an already visited note nor to make the current walk explore more of the graph. After being generated, each random walk will immediately update the internal Skip-gram network, from which the embeddings can be extracted after all the walks have been generated.

2.2.3 Neural network-based

Graph Convolutional Networks are a type of neural network that work directly on graphs and attempt to use their structural information in order to get better results (SCARSELLI et al., 2009). They are so powerful that even a small number of hidden layers is enough to generate useful embedding. Due to leveraging the graph's structure, they usually yield embeddings that preserve it in detriment of the community.

The design of the network is dependent on the graph, but they always receive a feature matrix and a matrix representation of the graph structure, like an adjacency matrix. These inputs are fed to an activation function that is used for the propagation through the hidden layers. Each layer can be represented as a feature matrix where each row represents a node, which is propagated to generate more abstract features. In the end, the output is a feature matrix from where the embeddings for every node can be extracted.

The main variation in the methods that fall into this category is in the activation function, since the main structure construction for the network tends to be the same. The number of nodes in the hidden layer are dependent on the size of the input matrices, and the number of layers can be decided experimentally.

Structural Deep Network Embedding (SDNE) is a semi-supervised deep learning model that aims to preserve a graph's structure, exploiting first order and second order proximity to do so (WANG; CUI; ZHU, 2016). It considers that, since graphs have a highly non-linear network structure, the embedding process needs to be able to capture non-linear information in order to preserve structural information.

The algorithm uses a deep learning model with multiple non-linear functions that capture network structure by mapping the inputs to non-linear spaces. It uses a supervised component to exploit first order proximity looking to preserve local network structure, and since many links can be missed when using only first order proximity, an unsupervised

component is used to reconstruct second order proximity to preserve global structure. Optimizing for both also helps with sparsity in data.

3 RELATED WORK

Although there is a large volume of research being done on embedding algorithms, they mostly focus on creating new algorithms to improve either the preservation of some specific characteristic or to make the process more efficient. It is also common to see attempts to work with more realistic datasets, where the network is really large or very sparse, noting that some common embedding methods have issues when dealing with graphs that have those characteristics and proposing more specific methods to work around those issues.

One thing that does not seem to be very explored in the literature is the combination of several types of information in order to improve the final result without creating a new embedding process, which keeps those solutions stuck to the community-structure trade-off. There are few works that go for the approach of combining the results of an already established embedding process with other data, and even less that combine two or more embeddings. That means there is a lot of room to explore the combination of different types of embedding to leverage the different information each of them captures.

Another thing that is not very common is the use of ensembles on embeddings. Most works proposing new algorithms use simple classifiers in order to test the performance of their method, and there are few works testing the use of ensembles with graph data, even though ensemble techniques can greatly improve the results of base classifiers. Therefore, there is a lot of room to work with ensembles on embeddings to see which methods work best and even to measure the impact they can have when compared to simple classifiers.

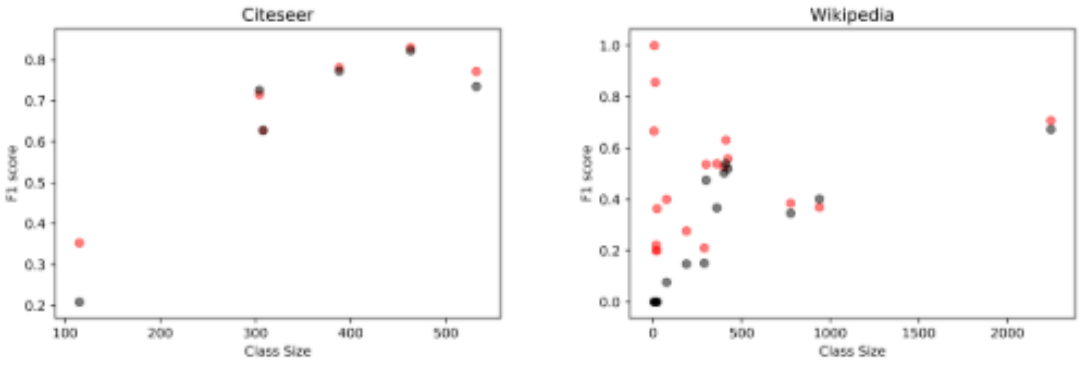
3.1 Combining more information to embeddings

Goyal et al. (2019) propose a framework using an ensemble method in order to aggregate different embedding techniques in an efficient way, arguing that graphs are not properly represented by a single method when they have complicated combinations between community and structure, since most embeddings have to preserve one property in detriment of the other. Using the idea that simple classifiers combined in an ensemble perform much better than any single one, using ensemble learning to combine diverse embeddings to generate a new one would yield similar improvements to the resulting representation.

Before choosing the methods to be used, they formally define a way to measure the diversity between any two techniques, since an ensemble requires the classifiers to be diverse in order to achieve good results. They adopt the RV coefficient, a multivariate generalization of the Pearson correlation coefficient, using the distance covariance metric to capture the possible non-linear relationship between two embeddings and calculate the distance correlation. These metrics allow them to define their embedding correlation upper bound and thus determine the label diversity prediction. In order to apply their framework, the nodes of the input graph are divided dividing the nodes into training (50 percent), validation (20 percent) and testing (30 percent). First, the validation is used to get the accuracy score for each embedding. Then the algorithm will greedily add the embedding with the next best score, evaluating the ensemble result. After, the performance is evaluated on the testing set. The greedy approach is used as a runtime optimization, diminishing the time complexity of a naive approach.

The citation graph is the only one whose result was not improved by the ensemble. Analyzing their results in figure 3.1, they note that the best improvement seems to be when representing smaller classes, a scenario where individual methods could not yield good results but the ensemble could, showing the usefulness of the ensemble in improving graph representation.

Figure 3.1: Node classification results on Citeseer and Wikipedia



(a) Node Classification on Citeseer (b) Node Classification on Wikipedia

Source: Goyal et al. (2019)

Ata et al. (2018) combines PPI networks in the form of node embedding and various biological annotations as feature representation for genes to find associations between genes and diseases . This data then is used to build binary classification models for disease gene prediction, in a model called N2VKO. It uses an adaptation of node2vec to

learn embeddings for the chosen graph, and then it integrates with biological information obtained from keywords or diseases associated with specific genes, obtained from biological datasets. The keywords are represented by a feature vector with binary values to describe all the keywords a node can have, with its size determined by the total amount of keywords selected to be used. A similar vector is used to represent the disease association. The embeddings are aggregated to those vectors by simple concatenation.

In order to improve performance, the representation learned from N2VKO passes through a feature selection before being used to build classifiers, since some features may be irrelevant and different feature subsets may be used to predict the association for specific diseases. They test four techniques to do so. They also use two oversampling techniques for imbalance correction, since the data for disease gene prediction is highly imbalanced and can lead to degraded performance. After optimization techniques are applied, they build several classifiers for disease gene prediction. First they focus their comparisons mainly in the impact the keywords and the oversampling have on the results, comparing four scenarios: a simple node2vec, node2vec with oversampling and without feature selection, N2VK with feature selection and N2VKO, which uses both oversampling and feature selection. While N2VO had similar results to node2vec, N2VK was already an improvement to both, showing that the keywords are a useful information to add when attempting to predict gene-disease associations. When comparing the full model with four state-of-the-art methods for disease gene prediction, it had better results then all of them in most datasets. Observing the poor results in two of the datasets, they note that high sparsity will result in low prediction performance, indicating that adding keywords won't be enough to compensate the lack of information.

3.2 Ensemble for tasks on graphs

Zhang, Xiang and Wang (2019) takes inspiration in the ensemble learning idea, using it to design two neural networks to learn embeddings where each one will use a different kind of data, such as network structure for one and node labels for the other. The goal is to integrate them in such a way that it satisfies the ensemble learning preconditions. The authors define two preconditions so that the ensemble learning principle can take place: the performance of the weak classifiers can not be too bad, and the errors of the classifiers must be mutually independent. The core idea of their method involves designing two separate models using different targets, satisfying the second precondition,

Table 3.1: Accuracy (%) on Citeseer dataset obtained by Zhang, Xiang and Wang (2019).

Training Ratios	10%	20%	30%	40%	50%	60%	70%	80%	90%
Deepwalk	52.99	55.78	57.20	59.16	58.33	60.41	58.16	58.96	58.41
Line(1st)	45.70	51.22	54.55	56.28	57.02	58.05	58.94	59.77	59.37
Line(2nd)	46.68	51.23	53.36	55.41	57.55	58.14	58.37	59.00	59.04
SNE	31.25	48.38	59.77	68.85	76.62	84.00	88.49	91.74	96.10
GCN	49.65	63.97	70.80	77.52	80.89	81.44	82.58	82.43	84.68

Source: Zhang, Xiang and Wang (2019)

so any two models that perform well for node classifications will be enough to have the ensemble learning effect desired when used in their framework. They combine the results of the neural networks using a simple linear combination with a parameter λ, where λ represents the influence of the embedding from one model in the final result and $1 - \lambda$ the influence of the other model. The input is the spectral transform of adjacency matrix, in order to make the models more stable. The first neural network has two layers and is based on an autoencoder like model, with the target of learning network structure, while the second has three layers and has the target of learning node labels.

As shown in Table 3.1, they find that, when the proportion of training labels is over 80 percent, the ensemble does not perform as well as the baseline methods, and that a proportion of between 10 percent and 30 percent is far too low for the node labels model to learn anything useful, performing even worst at node classification than the model for learning structure. However, when combined, even a low proportion of training labels for the node label model is enough to improve on the results of the structure model. The observed behavior when changing the proportion of the training data, combined with adjustments on the λ parameter, show that the ensemble can be used to improve upon baseline methods if the right settings are found.

Zhang, Xiang and Wang (2020) create an embedding method that uses an ensemble to combine several different embedding methods into a final embedding that can be used for tasks such as node classification and link prediction. The idea is to get similar improvement over the individual methods like an ensemble does with classifiers without running into too many scaling issues. They use a bootstrap sampling technique in order to get diverse models.

Since one of the main motivations is to improve results without losing too much efficiency, they use an ensemble that allows for parallel training for the models, more specifically, bootstrap aggregating (bagging). They train each individual model on a set obtained by bootstrap sampling, evaluate the performance of the resulting embedding and

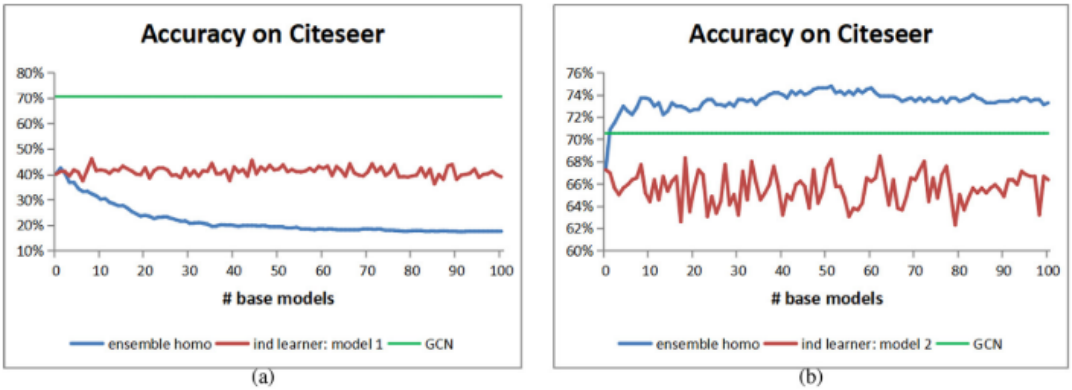
use the evaluation score to calculate the weight for the aggregation of network embeddings later.

They also use a stacking model to test heterogeneous ensemble methods. First, the original learners are trained using the original dataset, and the outputs are concatenated horizontally as a new set of features, which are then fed to the metalearner. Not only is this a serial learning approach, the experimental results for stacking performed worse than the individual learners, so they do not expand much on this.

For the ensemble, they design two models adapting GCN and Graph Auto-Encoder (GAE), two graph convolution network embedding methods, by removing the graph convolution and feeding them network structures as input. They were chosen due to their performance in both node classification and link prediction. They test their models on two paper-citation networks, comparing them to a three-layer GCN in a node classification task and to a GAE in a link prediction task.

The performance of the aggregated embeddings for the Citeseer dataset, as shown in Figure 3.2, has a boost when compared to individual baseline models, which they attribute to the repeated bootstrap sampling. Similar conclusions were extracted from experiments with the Cora dataset. They also highlight the quick converging of the model, which shows the efficiency they were looking for. Using only the GAE model as base, the results were not good for node classification, likely because the method does not use labels for training and an ensemble can only work if the base model has a strong generalization ability, although it can learn enough about the structure in order to work well for link prediction. The GCN model, however, has a strong generalization ability for both node classification and link prediction tasks, performing well on both of them.

Figure 3.2: Accuracy (%) on Citeseer dataset obtained by Zhang, Xiang and Wang (2020). Similar results were found for the Cora dataset.



Source: Zhang, Xiang and Wang (2020)

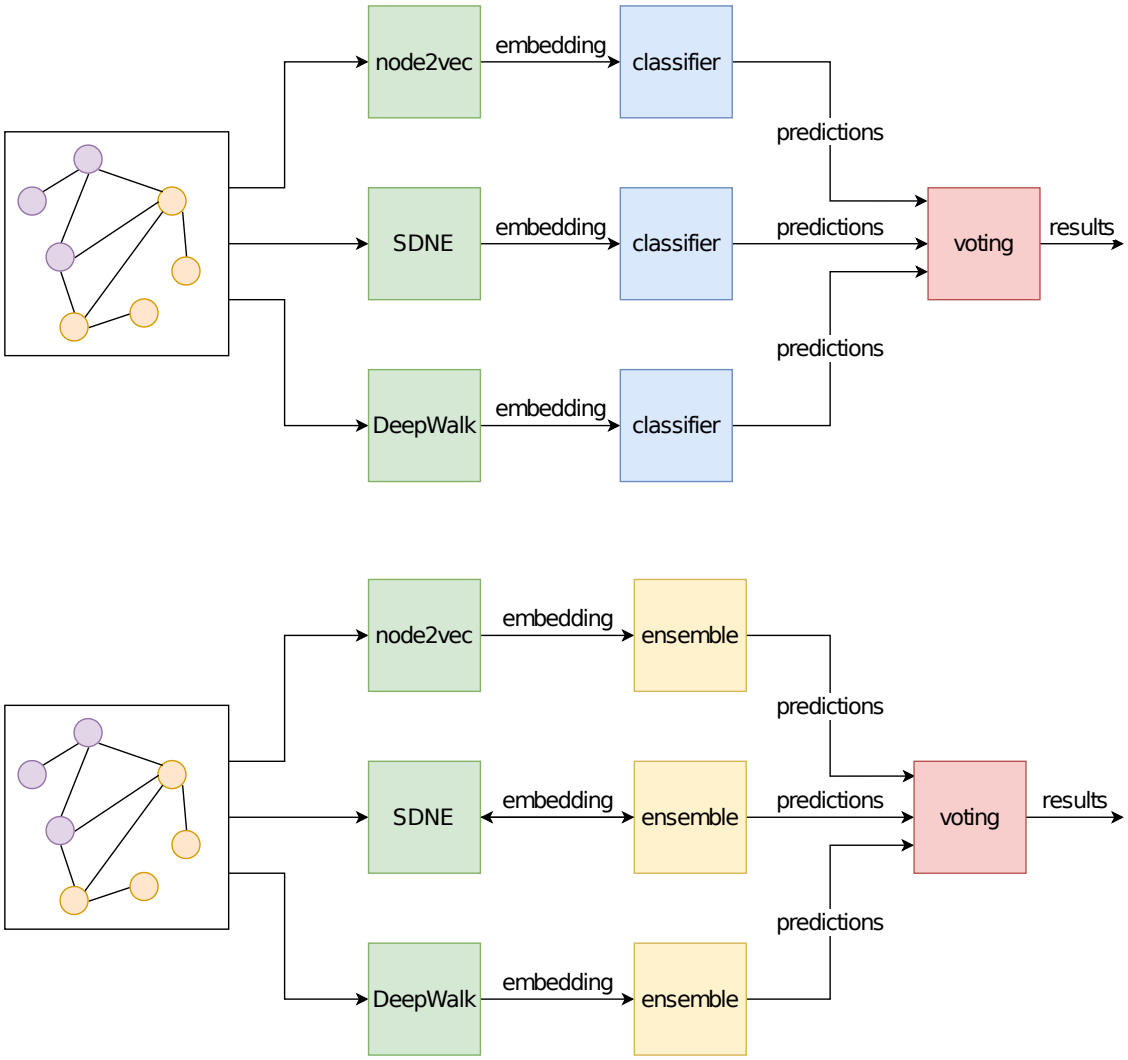
4 PROPOSAL

In order to get a more complete representation of graphs, we propose a simple concatenation of three different embeddings generated using the exact same graph data as input. Since we are working with node embedding, the concatenation is done by node too, meaning each node will be represented by the concatenation of three vectors, each one being the vector generated by an embedding algorithm for that node. By choosing each type based on the information it preserves we can feed the classifier more detailed information than if we used any single embedding, even if it was a representation that tried to balance what type of information it preserves.

Using the same reasoning, we also propose the use of a simple bootstrap aggregation ensemble instead of a base classifier, since diminishing the classifier variance by the use of representative samples is close to the idea that several representations of a graph will improve the results by giving it a better and less biased vision of the graph.

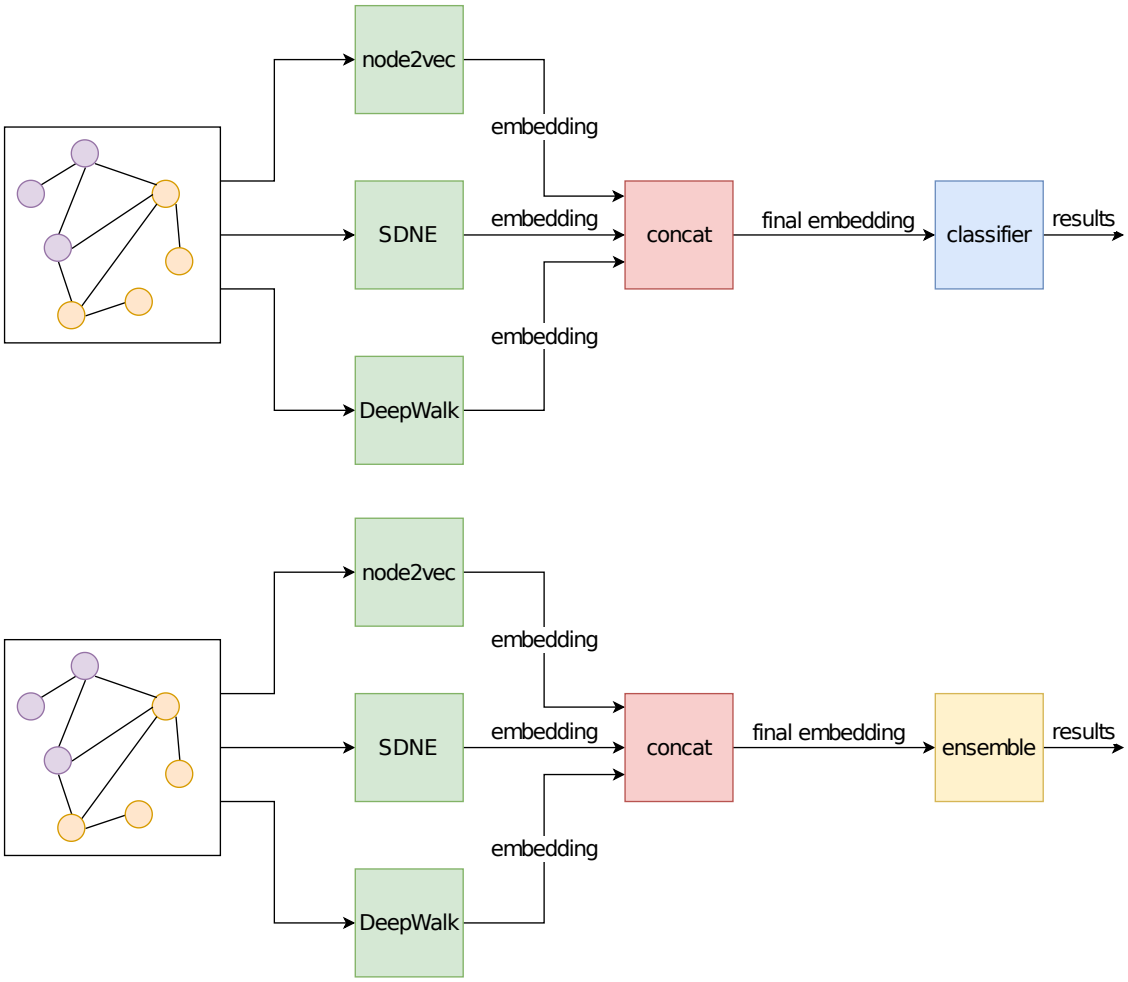
To test both proposals separately and then their combination in order to see the effect each of them have, we define four experiments. The first one is running a basic classifier on each embedding separately and having a soft vote with the three results (First flowchart in Figure 4.1). The second is running an ensemble on each embedding separately, also using a soft vote at the end to combine the results (Second flowchart in Figure 4.1). The third is running a simple classifier on the vector resulting from the concatenated embeddings (First flowchart in Figure 4.2). Finally, the fourth is running an ensemble on the vector resulting from the concatenated embeddings (Second flowchart in Figure 4.2). This way, we can verify not only if the combination of the concatenated embeddings and the ensemble yields better results than we would get when not using any of them, but also if they have better results than when using each technique on its own.

Figure 4.1: First two experiments, using a simple classifier and an ensemble on each embedding and them combining all three predictions in a final one using a soft vote.



Source: The Author

Figure 4.2: Third and fourth experiments, concatenating all three embeddings into one and using a simple classifier and an ensemble on the resulting vector.



Source: The Author

The models are evaluated twice. First, it uses a simple stratified split into training (75%) and testing (25%) sets, and after it uses that same training set in a cross-validation with five stratified folds. Due to limitations of the scikit-learn library, the predictions made using the soft vote with the results of each embedding could not be evaluated with the cross validation.

The implementation also yielded evaluations metrics by class, but those results will not be discussed at length here since they did not have any information regarding the initial proposal that was not present in the general evaluation metrics.

We run each experiment five times in each dataset, presenting the average result between them in order to avoid any outliers in training muddying the results.

5 EXPERIMENTS

5.1 Implementation

In order to test the proposal, we implemented the basic algorithm in Python 3.6.12, using the scikit-learn library for all machine learning algorithms (Bagging and SVC) and support functions for basic tasks such as splitting datasets and running a cross validation. The embedding algorithms used were implemented by the OpenNE library. The exact version of the main external libraries used are as follows:

- scikit-learn: 0.22
- pandas: 1.1.5
- numpy: 1.16.4

The implementation was built in order to make the experiment's repetition faster and more accurate, ensuring the same steps were taken every time. It also allows easy configuration for the dataset, embeddings and classifiers chosen for a specific run.

We chose to use the following three embedding algorithms: SDNE, prioritizing structural information, DeepWalk, prioritizing community information and node2vec, balancing both types of information as much as possible. Each embedding generated has the number of features set to 32, in order to prevent the final vector from being too big after concatenation. Other than that, the parameters used for all embeddings were the ones set as default by the OpenNE library.

5.2 Chosen data

The parameters for the classifiers and ensemble are chosen during the run, using the basic grid search implemented by the scikit-learn library. The SVC classifier uses grids of 0.001, 0.01, 0.1, 1 and 10 for the regularization parameter (C) and 0.001, 0.01, 0.1 and 1 for the gamma parameter, and the Bagging ensemble uses grids of 5, 10 and 15 for the number of estimator and 0.6, 0.8, 1.0 for the number of samples.

We run the experiments on six different datasets: brazilian_air_traffic (WU; HE; XU, 2019), a network that shows the existence of commercial flight between Brazilian airports, where the labels for each airport are assigned based on how busy the airport is; wikipedia (GROVER; LESKOVEC, 2016), a coocurrence network of words appear-

Table 5.1: Dataset statistics

	brazilian_air_traffic	wikipedia	ecoli	yeast	twitch_pt_br	cora
# of nodes	131	2405	3489*	5548*	1912	2708
# of links	1074	17981	178271	526392	31,299	5429
# of classes	4	17	2	2	2	7

*Number of labeled nodes is smaller than the total number of nodes in the graph.
Source: The Author

ing in the million bytes of the Wikipedia dump, where the labels are grammatical tags; ecoli (SCHAPKE et al., 2020), a PPI network for the *Escherichia coli* organism; yeast (SCHAPKE et al., 2020), a PPI network for the *Saccharomyces cerevisiae* organism; twitch_pt_br (ROZEMBERCZKI; ALLEN; SARKAR, 2019), a network of twitch users that create content using the same language where the links are mutual friendships and the labels indicate if a user uses explicit language; and cora (ORBIFOLD, 2019), a citation network where the nodes are papers and the labels indicate the subject of the paper.

Since the experiment in focused on supervised learning for node classification, the library expects that every node has a label, which is not the case for ecoli and yeast. In order to treat this, the embeddings are generated using the complete graph, but the classification task uses only the representation of the nodes that have labels.

brazilian_air_traffic and cora are relatively balanced multiclass networks, having more or less an equal distribution of nodes between their many classes. wikipedia is an unbalanced multiclass network, where the less represented class has around ten instances in a graph with over 2400 nodes. twitch_pt_br, yeast and ecoli are binary, unbalanced networks. This diversity in chosen networks allows us to better test the proposal, avoiding a conclusion biased by the characteristic of the chosen dataset. The table 5.1 shows the number of nodes, links and classes each dataset has.

6 RESULTS

The figures and tables bellow show the results from the experiments on brazilian_air_traffic (Tables 6.1 and 6.2 and Figure 6.1), cora (Tables 6.3 and 6.4 and Figure 6.2), ecoli (Tables 6.5 and 6.6 and Figure 6.3), twitch_pt_br (Tables 6.7 and 6.8 and Figure 6.4), wikipedia (Tables 6.9 and 6.10 and Figure 6.5), and yeast (Tables 6.11 and 6.12 and Figure 6.6). The tables show the average metrics obtained from the results of the five runs for each experiment, both using the basic evaluation and the cross validation, while the boxplot shows the distribution of the F1 score results of all five runs using the basic evaluation.

The results of the first experiment, as seen in the first flowchart in Figure 4.1, can be found in rows where the embedding value is sdne, deepWalk, node2vec or vote, and where the classifier value is SVM. The results of the second experiment, as seen in the second flowchart in Figure 4.1, are the ones in the rows with the same embedding values as mentioned above, but with the classifier value as Bagging.

The same logic applies for the other two experiments. Both have the embedding value as concat, but the third experiment (first flowchart in Figure 4.2) will have the classifier value as SVM and the fourth experiment (second flowchart in Figure 4.2) will have the classifier as Bagging.

6.1 Brazilian Air Traffic

The Brazilian Air Traffic graph represents the airport network in Brazil. The nodes represent airports, the edges represent the existence of flights between these two airports and the node classification indicates how busy the airport is.

For this dataset, the best results were obtained with SDNE and the concatenation, with the predictions made by the soft vote also having a good result in the basic evaluation. Meanwhile, both deepWalk and node2vec underperformed, with lower results both in the basic evaluation and the cross validation. This can be explained by the fact that the node classification for this graph is heavily related to its structure.

The use of the ensemble seems to have yielded slightly worse results for almost all embeddings. This can be seen more clearly in the results of the cross validation, where the difference between the ensemble and the simple classifier is larger.

Table 6.1: Table with the average results for the brazilian_air_traffic network obtained with basic evaluation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
SVM	sdne	0.73	0.08	0.74	0.73	0.74
Bagging	sdne	0.71	0.05	0.74	0.72	0.72
SVM	vote	0.69	0.11	0.70	0.69	0.69
SVM	concat	0.64	0.04	0.66	0.65	0.64
Bagging	concat	0.59	0.04	0.59	0.59	0.59
Bagging	vote	0.58	0.06	0.58	0.59	0.60
Bagging	deepWalk	0.53	0.09	0.53	0.54	0.55
SVM	deepWalk	0.52	0.11	0.52	0.53	0.53
SVM	node2vec	0.44	0.09	0.46	0.45	0.45
Bagging	node2vec	0.42	0.08	0.44	0.43	0.43

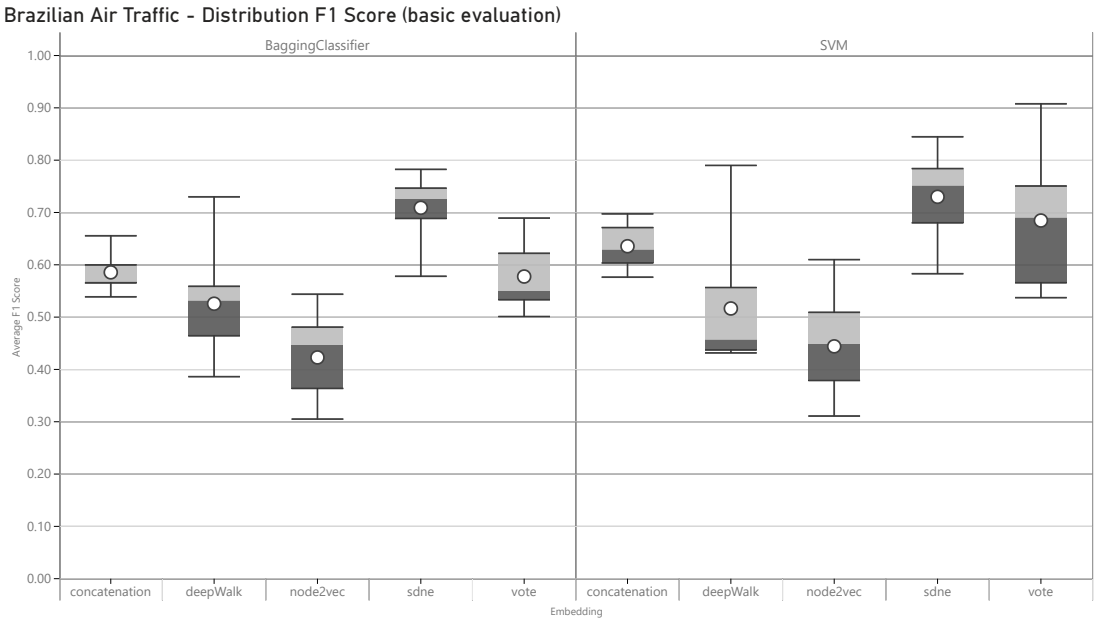
Source: The Author

Table 6.2: Table with the average results for the brazilian_air_traffic network obtained with cross validation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
SVM	concat	0.63	0.03	0.64	0.66	0.64
SVM	sdne	0.62	0.10	0.63	0.65	0.63
Bagging	concat	0.58	0.03	0.59	0.61	0.60
Bagging	sdne	0.56	0.11	0.59	0.60	0.58
SVM	deepWalk	0.53	0.04	0.55	0.56	0.55
SVM	node2vec	0.51	0.05	0.52	0.54	0.52
Bagging	deepWalk	0.47	0.05	0.49	0.50	0.49
Bagging	node2vec	0.45	0.06	0.48	0.47	0.48

Source: The Author

Figure 6.1: Boxplot showing distribution of F1 score results for brazilian_air_traffic network



Source: The Author

6.2 Cora

Cora is a citation graph. The nodes represent papers, the edges indicate that one paper has cited the other, and the node classification shows the paper's subject.

This dataset had very similar results for most experiments, with little variation between the scores of the concatenation, deepWalk and node2vec. The predictions made by the soft vote had a surprisingly good result for this graph, while SDNE greatly underperfomed. Just as mentioned in Section 6.1, this is probably due to the nature of the node labels, except in this graph they are highly related to the neighborhood instead of the structure.

Once again, the ensemble did not have a big impact in the results. This time, however, there was no slight worsening of the metrics, with most results either slightly better or the same as the ones obtained with the simple classifier.

Table 6.3: Table with the average results for the cora network obtained with basic evaluation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
Bagging	vote	0.84	0.01	0.85	0.82	0.84
SVM	vote	0.83	0.02	0.85	0.82	0.84
Bagging	concat	0.73	0.27	0.74	0.74	0.73
SVM	deepWalk	0.73	0.27	0.74	0.74	0.73
SVM	concat	0.73	0.27	0.74	0.74	0.73
Bagging	deepWalk	0.73	0.27	0.74	0.74	0.73
Bagging	node2vec	0.73	0.27	0.73	0.74	0.72
SVM	node2vec	0.73	0.27	0.73	0.74	0.72
Bagging	sdne	0.19	0.12	0.36	0.26	0.23
SVM	sdne	0.19	0.12	0.36	0.24	0.23

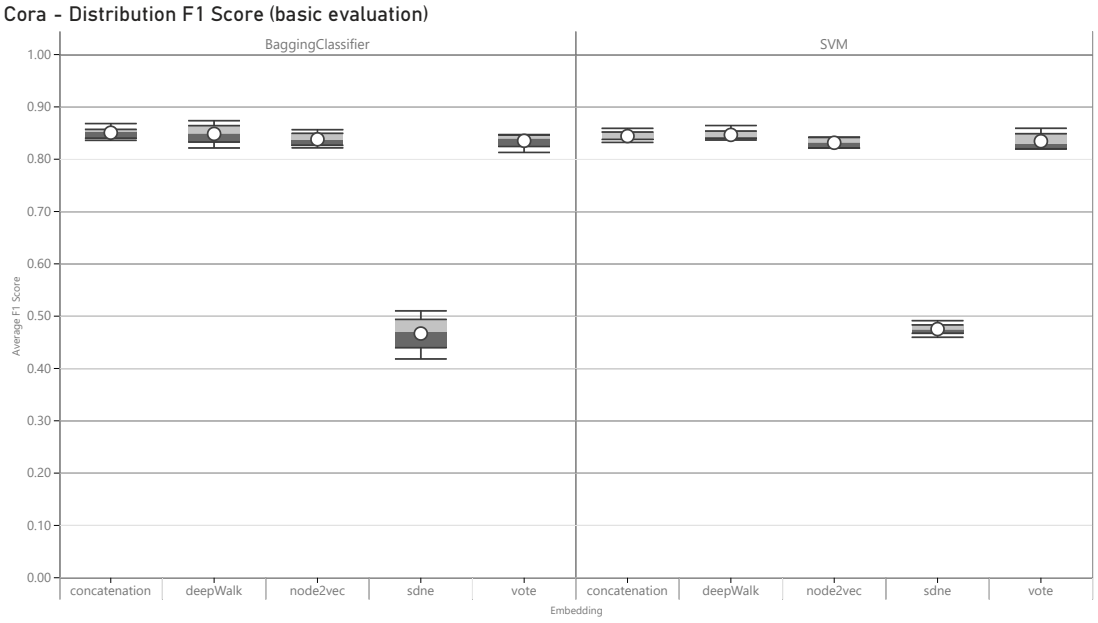
Source: The Author

Table 6.4: Table with the average results for the cora network obtained with cross validation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
Bagging	concat	0.83	0.01	0.84	0.85	0.83
SVM	deepWalk	0.83	0.00	0.84	0.85	0.83
SVM	concat	0.83	0.01	0.84	0.84	0.83
Bagging	deepWalk	0.83	0.01	0.84	0.85	0.82
Bagging	node2vec	0.83	0.01	0.83	0.84	0.81
SVM	node2vec	0.83	0.01	0.84	0.85	0.81
Bagging	sdne	0.17	0.00	0.38	0.25	0.22
SVM	sdne	0.17	0.00	0.38	0.24	0.22

Source: The Author

Figure 6.2: Boxplot showing distribution of F1 score results for cora network



Source: The Author

6.3 Ecoli

The ecoli dataset is a PPI network where nodes represent proteins, the edges represent a mathematical representation of physical interactions between proteins, and the classification indicates gene essentiality (an essential gene is one that the organism requires to survive or reproduce normally).

This dataset was the only one to see an improvement in the results when using concatenation, which can be explained by its more complex node classification that benefits from having more information from both characteristics. This can also be seen in the fact that all three individual embeddings had very similar results, both in the basic evaluation and the cross validation. It also possible to observe a big improvement when using the soft vote. This was one of the few datasets in which the concatenation was not so close to the best result, even though, as mentioned above, it was an improvement over the individual embeddings.

This was also another dataset where the ensemble was responsible for a slight improvement in results when compared to the simple classifier, although still not by much. It should be noted that in this case this was a more constant improvement, and there was

Table 6.5: Table with the average results for the ecoli network obtained with basic evaluation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
Bagging	vote	0.59	0.05	0.79	0.56	0.95
SVM	vote	0.58	0.06	0.77	0.55	0.95
Bagging	concat	0.48	0.18	0.81	0.59	0.52
SVM	concat	0.47	0.17	0.79	0.66	0.51
Bagging	deepWalk	0.46	0.18	0.81	0.59	0.50
Bagging	node2vec	0.46	0.18	0.81	0.53	0.50
SVM	deepWalk	0.45	0.18	0.80	0.51	0.50
SVM	node2vec	0.45	0.18	0.80	0.43	0.49
Bagging	sdne	0.43	0.16	0.79	0.43	0.49
SVM	sdne	0.43	0.16	0.80	0.42	0.49

Source: The Author

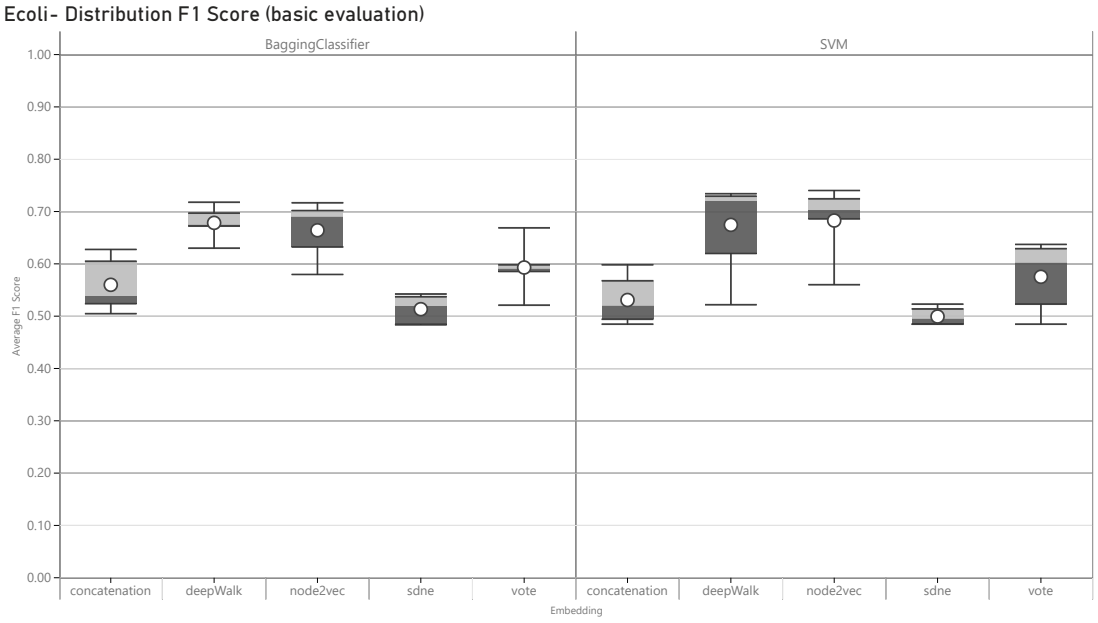
Table 6.6: Table with the average results for the ecoli network obtained with cross validation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
Bagging	concat	0.55	0.04	0.94	0.68	0.54
SVM	concat	0.53	0.02	0.94	0.77	0.53
Bagging	deepWalk	0.50	0.00	0.94	0.64	0.51
Bagging	node2vec	0.49	0.01	0.94	0.58	0.50
SVM	deepWalk	0.49	0.01	0.94	0.55	0.50
Bagging	sdne	0.49	0.00	0.94	0.48	0.50
SVM	node2vec	0.49	0.00	0.94	0.47	0.50
SVM	sdne	0.49	0.00	0.94	0.47	0.50

Source: The Author

no instance where the ensemble did not have a better result than the simple classifier.

Figure 6.3: Boxplot showing distribution of F1 score results for ecoli network



Source: The Author

6.4 Twitch PTBR

The Twitch PTBR graph represents the relationship network between Twitch (a video stream platform) users. Nodes represent users that create content in Brazilian Portuguese, the edges represent a friendship between users and the classification indicates if the user uses explicit language or not.

This dataset had some of the most similar results when comparing all experiments. The SDNE embedding had an underperfomance when compared to the rest, but not as much as it did in other datasets. All other embeddings had very similar results, with the concatenation being slightly worse than the others. Due to the nature of the classification done here (whether a user uses explicit language or not), it can be hard to pinpoint which characteristic would be more important. Although we can assume the neighborhood is more important, this is an analysis done after the fact, and it is helped by the results seen in datasets where we can derive this information before doing any experiments.

This was a graph where the use of the ensemble did not have a very consistent result, so we are not able to draw any conclusion other than the fact that it did not seem to change the results much.

Table 6.7: Table with the average results for the twitch_pt_br network obtained with basic evaluation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
SVM	vote	0.54	0.02	0.67	0.57	0.68
Bagging	vote	0.53	0.01	0.67	0.56	0.68
Bagging	deepWalk	0.52	0.19	0.60	0.57	0.53
SVM	deepWalk	0.52	0.19	0.60	0.58	0.53
SVM	node2vec	0.52	0.19	0.60	0.57	0.54
Bagging	node2vec	0.52	0.19	0.60	0.56	0.53
Bagging	concat	0.51	0.19	0.60	0.57	0.53
SVM	concat	0.50	0.18	0.60	0.56	0.52
SVM	sdne	0.41	0.15	0.57	0.52	0.47
Bagging	sdne	0.41	0.15	0.57	0.53	0.47

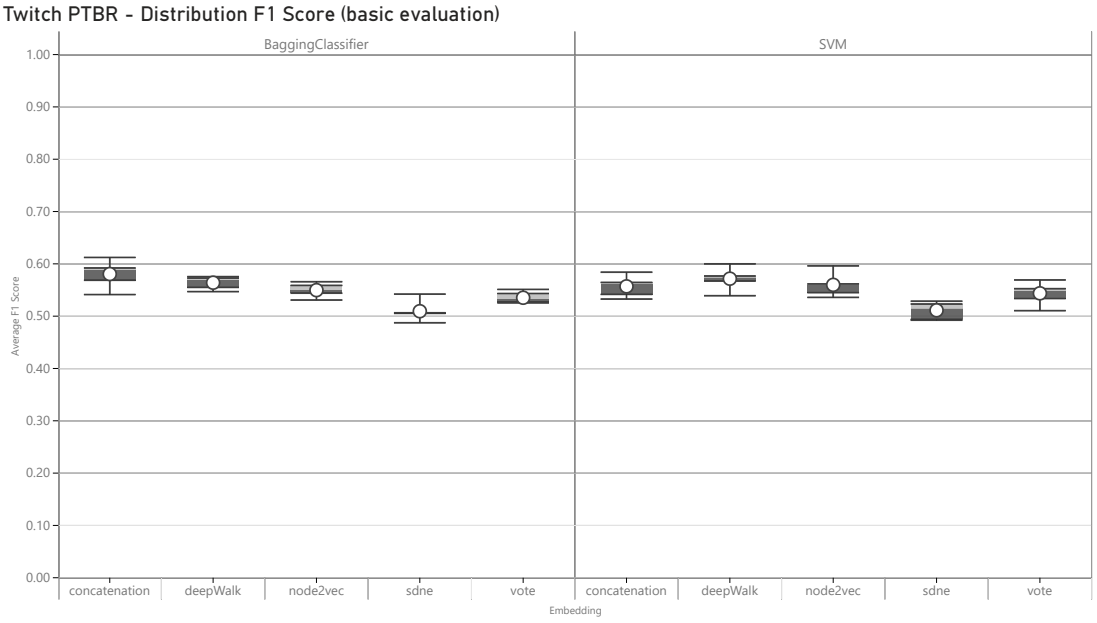
Source: The Author

Table 6.8: Table with the average results for the twitch_pt_br network obtained with cross validation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
Bagging	node2vec	0.59	0.01	0.68	0.65	0.59
SVM	node2vec	0.59	0.02	0.69	0.66	0.60
Bagging	deepWalk	0.59	0.02	0.69	0.66	0.60
SVM	deepWalk	0.59	0.02	0.69	0.66	0.60
Bagging	concat	0.58	0.01	0.69	0.66	0.59
SVM	concat	0.57	0.03	0.68	0.65	0.58
SVM	sdne	0.45	0.03	0.66	0.59	0.52
Bagging	sdne	0.45	0.02	0.66	0.61	0.52

Source: The Author

Figure 6.4: Boxplot showing distribution of F1 score results for twitch_pt_br network



Source: The Author

6.5 Wikipedia

The Wikipedia graph represents the words present in the first million bytes of a Wikipedia content dump. The nodes are words, the edges indicate that two words appeared beside each other, and the classification indicates a linguistic classification called Part of Speech.

This graph had very interesting results. It was the only one where the soft vote had a massive underperformance when compared to all other embeddings, which had similarly high F1 scores in the basic evaluation. It was also the only one where the cross validation yielded results that were much lower than the ones obtained with the basic evaluation, although the order of which embedding performed best did not change much.

This was another one where, similar to most of the others, the concatenation was close to the best results, but unlike the other graphs, there was no individual embedding that massively underperfomed.

It was also the one dataset that saw bigger improvements with the ensemble in the cross validation. The results with node2vec and deepWalk improved almost 0.1 when compared with the results obtained with the simple classifier, while the improvements

Table 6.9: Table with the average results for the wikipedia network obtained with basic evaluation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
Bagging	node2vec	0.93	0.14	0.60	0.63	0.60
Bagging	deepWalk	0.93	0.14	0.60	0.63	0.60
SVM	concat	0.92	0.15	0.55	0.61	0.55
Bagging	concat	0.92	0.15	0.54	0.60	0.55
SVM	deepWalk	0.92	0.15	0.52	0.56	0.52
Bagging	sdne	0.92	0.16	0.54	0.59	0.54
SVM	node2vec	0.92	0.15	0.52	0.56	0.52
SVM	sdne	0.91	0.16	0.52	0.55	0.52
SVM	vote	0.57	0.03	0.65	0.54	0.71
Bagging	vote	0.55	0.02	0.62	0.53	0.70

Source: The Author

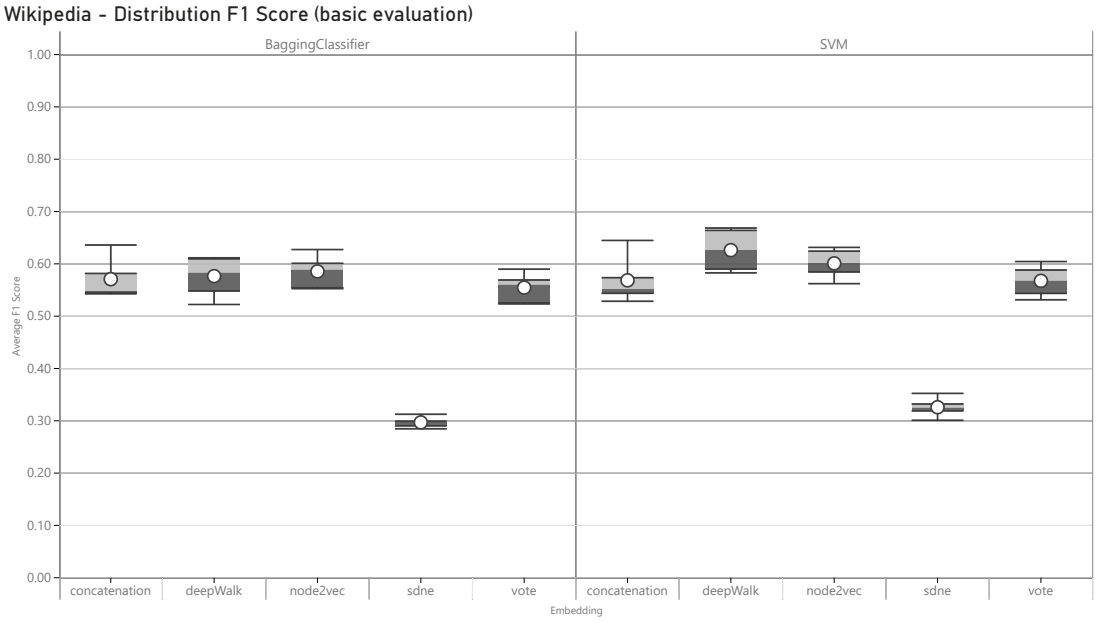
Table 6.10: Table with the average results for the wikipedia network obtained with cross validation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
Bagging	node2vec	0.61	0.02	0.71	0.64	0.60
Bagging	deepWalk	0.60	0.02	0.70	0.64	0.59
SVM	concat	0.56	0.02	0.67	0.61	0.54
Bagging	concat	0.56	0.02	0.68	0.61	0.54
Bagging	sdne	0.55	0.12	0.65	0.61	0.54
SVM	deepWalk	0.52	0.10	0.64	0.56	0.52
SVM	node2vec	0.52	0.10	0.64	0.56	0.52
SVM	sdne	0.52	0.10	0.64	0.56	0.52

Source: The Author

seen in other datasets were around 0.01.

Figure 6.5: Boxplot showing distribution of F1 score results for wikipedia network



Source: The Author

6.6 Yeast

The yeast graph is a PPI similar to the ecoli one. Nodes represent proteins, edges represent protein interactions and the classification indicates gene essentiality.

As the largest graph, and one of the most complex in the selected datasets, the results obtained by the yeast dataset were the least conclusive of all. It was the only one where the concatenation was not close to the best results, and its performance was actually closer to the worst. node2vec and deepWalk had the best results, specially in the cross validation, but they lost by a big margin to the soft vote when using the basic evaluation.

The ensemble results were also inconclusive. Like in the twitch graph, there was no consistency on whether the results were better or worse than when using the simple classifier, and even when it did change the results it was not by much.

Table 6.11: Table with the average results for the yeast network obtained with basic evaluation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
SVM	vote	0.85	0.01	0.92	0.81	0.92
Bagging	vote	0.84	0.02	0.92	0.80	0.92
Bagging	node2vec	0.77	0.29	0.81	0.79	0.76
SVM	node2vec	0.77	0.29	0.81	0.79	0.76
SVM	deepWalk	0.72	0.27	0.79	0.76	0.71
Bagging	deepWalk	0.72	0.27	0.79	0.75	0.71
Bagging	concat	0.59	0.22	0.73	0.65	0.59
SVM	concat	0.57	0.21	0.72	0.65	0.58
SVM	sdne	0.50	0.18	0.72	0.69	0.52
Bagging	sdne	0.49	0.17	0.71	0.70	0.52

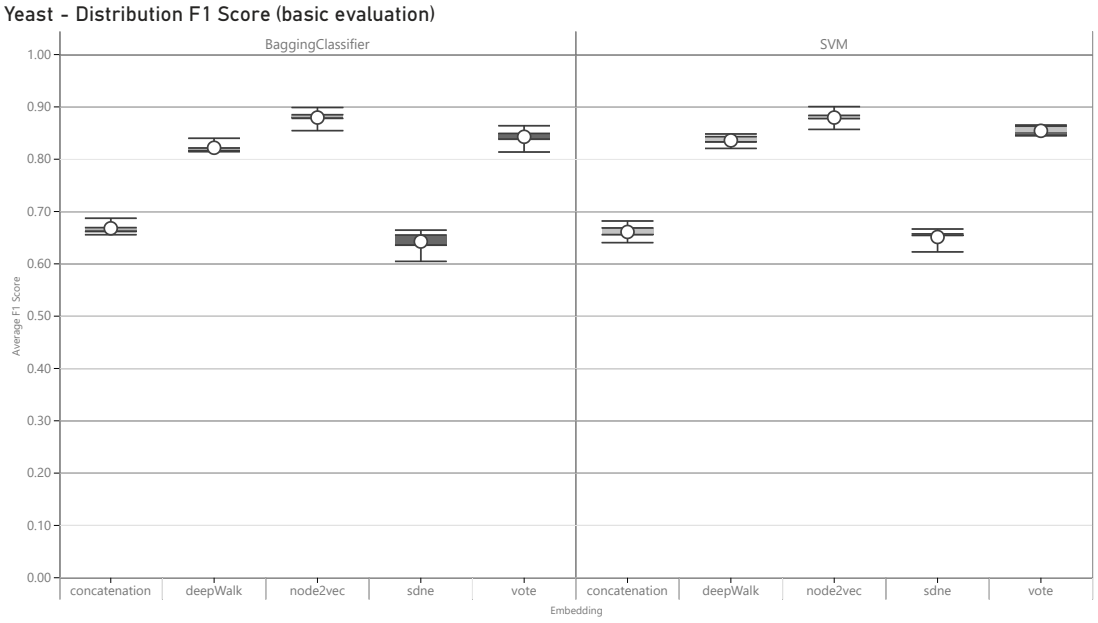
Source: The Author

Table 6.12: Table with the average results for the yeast network obtained with cross validation

Classifier	Embedding	F1 Score	Std Dev F1 Score	Precision	Recall	Accuracy
Bagging	node2vec	0.88	0.00	0.93	0.91	0.86
SVM	node2vec	0.88	0.01	0.93	0.91	0.86
SVM	deepWalk	0.82	0.00	0.90	0.87	0.79
Bagging	deepWalk	0.82	0.01	0.90	0.87	0.79
Bagging	concat	0.67	0.01	0.84	0.75	0.65
SVM	concat	0.65	0.02	0.84	0.75	0.63
SVM	sdne	0.55	0.04	0.83	0.81	0.55
Bagging	sdne	0.54	0.01	0.82	0.82	0.55

Source: The Author

Figure 6.6: Boxplot showing distribution of F1 score results for yeast network



Source: The Author

6.7 Conclusions

As seen by the results, the concatenation of embeddings does not necessarily improve upon every embedding method tested. In fact, when a network had labels that highly correlated with the structure or the neighborhood, algorithms more tailored to that specific characteristic had better results. The concatenation, however, seemed to have one advantage over any individual embedding: a very good result regardless of which characteristic mattered more for the node classification in each graph.

SDNE, for example, had very good results in the airtraffic network, as seen in Table 6.1. This graph classified its nodes using a metric (how busy the airport is) that is highly related to the number of connections each node has, so SDNE was able to capture the structural information that was very useful for the classifiers in this task. deepWalk, on the other hand, a method that focuses more on the neighborhood of each node, had better results with graphs like wikipedia, as seen in Table 6.9 which used a metric (linguistic classification) that was better informed by what was present in its neighborhood instead of just the number of links.

None of those individual embeddings, however, managed to have a good perfor-

mance in every network tested, while the concatenation was constantly amongst the best results, often having very similar metrics to the best one. This can be very useful for networks where the classification is not as clearly related to one characteristic, because it allows researchers to train only one classifier and have a satisfactory result, instead of training several with different embeddings as input, saving on time and computer processing power.

Another positive aspect of the concatenation is the possible improvement over individual embeddings in networks where the classification is not directly related to one single characteristic, often needing information from both of them for a more accurate classification. The results were less conclusive in this direction, since we can see this improvement on the ecoli graph, shown in Table 6.5, but a similarly complex network like yeast (Table 6.11) saw worst results with the concatenation than with all three individual embeddings.

An interesting point to note is that the soft voting done with the predictions of each individual embedding also had very good results in every network, including a minor improvement over the individual embeddings in the yeast networks. This could indicate that a better way to combine individual embeddings is a soft vote of predictions made by classifiers trained by each representation, since it consistently had similar or better results than the concatenation and did not show an underperfomance on any network. This method, however, still requires more time and computational resources since it needs to train multiple models with different data, so the concatenation can often be a better alternative, specially since after you train all classifiers you could just choose the best of them.

As for the use of an ensemble instead of a simple classifier, the difference seems to be negligible, both in the individual classifiers and the concatenated embeddings. It could be due to the choice of ensemble, the choice of the classifier used for the ensemble, the chosen parameters or any combination of those things. Either way, based on the results it seems that the computational cost of a basic ensemble is not worth it when applied to a node classification task.

7 CONCLUSION

From the results shown in the previous chapter, it is possible to conclude that, while concatenating different embeddings does not necessarily improve upon the best result you can obtain with a well chosen embedding algorithm, it can be a useful tool to save on time and processing power and still get very good results when the choice of the best embedding method to use is not obvious. On the other hand, using an ensemble did not produce any meaningful differences in the results when compared to a simple classifier.

Further research is needed in order to test this approach in complex, unbalanced graphs, since the results obtained with the two networks used that had both these characteristics (ecoli and yeast) were less conclusive than the rest. Since there were other unbalanced, but simpler, networks used in the evaluations that had good results, the more interesting approach seems to be focusing on the complexity of the data.

Other works could also explore the use of dimensionality reduction on the representation obtained by the concatenation process before feeding it to classification models. Since the size of the embedding vector effectively triples after the concatenation, the larger dimensionality could be one of the reasons for the slightly worse performances it had when compared to vector with less information but smaller. The use of dimensionality reduction could improve results by eliminating noise in the representation while maintaining important extra information.

Another aspect that could use further work is the optimization of the parameters for each embedding algorithm. While we used hyperparametrization for classifier parameters, no such things was used when configuring the embedding algorithms, since we used the default values set by the external implementation, and did no other tests to see how much this aspect could change the results.

Finally, there is a possibility to expand upon the proposal by exploring the use of a soft vote instead of a concatenation. As mentioned in the results, this could be a better approach to improving upon individual embeddings in more complex networks. Another approach to this vote can be the use of a stacking ensemble instead of a simple soft vote, which could use the concept of ensemble to better combine the existing predictions.

REFERENCES

ATA, S. K. et al. Integrating node embeddings and biological annotations for genes to predict disease-gene associations. **BMC Systems Biology**, BioMed Central Ltd., v. 12, dec 2018. ISSN 17520509.

BBEIMAN, L. **Bagging Predictors**. [S.l.], 1996. v. 24, 123–140 p.

BELKIN, M.; NIYOGI, P. **Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering**. [S.l.], 2001.

CAI, H.; ZHENG, V. W.; CHANG, K. C.-C. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. sep 2017. Available from Internet: <http://arxiv.org/abs/1709.07604>.

CHU, W.; PARK, S. T. Personalized recommendation on dynamic content using predictive bilinear models. **WWW'09 - Proceedings of the 18th International World Wide Web Conference**, p. 691–700, 2009.

CORTES, C. **Support-Vector Networks**. [S.l.], 1995. v. 20, 273–297 p.

DENG, L.; LI, X. Machine learning paradigms for speech recognition: An overview. **IEEE Transactions on Audio, Speech and Language Processing**, v. 21, n. 5, p. 1060–1089, 2013. ISSN 15587916.

DIETTERICH, T. G. **Ensemble Methods in Machine Learning**. [S.l.], 2000. Available from Internet: <http://www.cs.orst.edu/{~}t>.

FORTUNATO, S.; CASTELLANO, C. Community Structure in Graphs. **Computational Complexity: Theory, Techniques, and Applications**, Springer New York, v. 9781461418009, p. 490–512, dec 2007. Available from Internet: <http://arxiv.org/abs/0712.2716>.

GOOGLE, A. A. et al. Distributed Large-scale Natural Graph Factorization * Shravan Narayanamurthy. In: **22nd International World Wide Web Conference**. [S.l.: s.n.], 2013. ISBN 9781450320351.

GOYAL, P.; FERRARA, E. Graph Embedding Techniques, Applications, and Performance: A Survey. **Knowledge-Based Systems**, Elsevier B.V., v. 151, p. 78–94, may 2017. Available from Internet: <http://arxiv.org/abs/1705.02801http://dx.doi.org/10.1016/j.knosys.2018.03.022>.

GOYAL, P. et al. Graph Representation Ensemble Learning. sep 2019. Available from Internet: <http://arxiv.org/abs/1909.02811>.

GROVER, A.; LESKOVEC, J. node2vec: Scalable Feature Learning for Networks. 2016. Available from Internet: <http://dx.doi.org/10.1145/2939672.2939754>.

GUZELLA, T. S.; CAMINHAS, W. M. A review of machine learning approaches to Spam filtering. **Expert Systems with Applications**, Elsevier Ltd, v. 36, n. 7, p. 10206–10222, 2009. ISSN 09574174. Available from Internet: <http://dx.doi.org/10.1016/j.eswa.2009.02.037>.

HAMILTON, W. L.; YING, R.; LESKOVEC, J. **Representation Learning on Graphs: Methods and Applications**. [S.l.], 2017.

HENDERSON, K. et al. RolX: Structural Role Extraction Mining in Large Graphs. 2012.

JADHAV, S. et al. Disease Prediction by Machine Learning from Healthcare Communities. **International Journal of Scientific Research in Science and Technology**, p. 29–35, 2019. ISSN 2395-6011.

KONONENKO, I. Machine learning for medical diagnosis: history, state of the art and perspective. **Artificial intelligence in medicine**, v. 23, n. 1, p. 89–109, 2001. ISSN 0933-3657. Available from Internet: <http://www.ncbi.nlm.nih.gov/pubmed/11470218>.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. In: **1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings**. International Conference on Learning Representations, ICLR, 2013. Available from Internet: <http://ronan.collobert.com/senna/>.

MITCHELL, T. **Machine Learning**. [S.l.]: McGraw Hill, 1997.

ORBIFOLD. **The Cora Dataset**. 2019. Available from Internet: <https://graphsandnetworks.com/the-cora-dataset/>.

OU, M. et al. Asymmetric transitivity preserving graph embedding. In: **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.]: Association for Computing Machinery, 2016. v. 13-17-August-2016, p. 1105–1114. ISBN 9781450342322.

PAK, M.; KIM, S. A review of deep learning in image recognition. **Proceedings of the 2017 4th International Conference on Computer Applications and Information Processing Technology, CAIPT 2017**, v. 2018-January, p. 1–3, 2018.

PEROZZI, B.; AL-RFOU, R.; SKIENA, S. DeepWalk: Online learning of social representations. In: **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, New York, USA: Association for Computing Machinery, 2014. p. 701–710. ISBN 9781450329569. Available from Internet: <http://dl.acm.org/citation.cfm?doid=2623330.2623732>.

ROZEMBERCZKI, B.; ALLEN, C.; SARKAR, R. **Multi-scale Attributed Node Embedding**. 2019.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach, 4th edition**. [S.l.]: McGraw Hill, 2020.

SCARSELLI, F. et al. The graph neural network model. **IEEE Transactions on Neural Networks**, Institute of Electrical and Electronics Engineers Inc., v. 20, n. 1, p. 61–80, jan 2009. ISSN 10459227.

SCHAPKE, J. et al. **EPGAT: Gene Essentiality Prediction With Graph Attention Networks**. [S.l.], 2020.

SHEPARD, . R. N. et al. **Nonlinear Dimensionality Reduction by Locally Linear Embedding**. [S.l.], 1994. v. 1, n. 2, 50 p. Available from Internet: <www.research.att.com/yann/ocr/mnist.>

SUROWIECKI, J. **The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations**. [S.l.]: Doubleday, 2004.

WANG, D.; CUI, P.; ZHU, W. Structural Deep Network Embedding. 2016.

WANG, G. et al. A comparative assessment of ensemble learning for credit scoring. **Expert Systems with Applications**, Elsevier Ltd, v. 38, n. 1, p. 223–230, 2011. ISSN 09574174. Available from Internet: <http://dx.doi.org/10.1016/j.eswa.2010.06.048>.

WU, J.; HE, J.; XU, J. Demo-net: Degree-specific graph neural networks for node and graph classification. In: **Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining**. [S.l.: s.n.], 2019. p. 406–415.

ZHANG, B.; XIANG, J.; WANG, X. Ensemble learning for network embeddings. In: **Proceedings - 21st IEEE International Conference on High Performance Computing and Communications, 17th IEEE International Conference on Smart City and 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019**. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2019. p. 945–951. ISBN 9781728120584.

ZHANG, B.; XIANG, J.; WANG, X. Network representation learning with ensemble methods. **Neurocomputing**, Elsevier B.V., v. 380, p. 141–149, mar 2020. ISSN 18728286.