ALTEGRAD Report

Ikhlass YAYA-OYÉ,[1] Valerii MARKIN,[1] and César LEBLANC[1]

[1]*Institut Polytechnique de Paris*
M2 Data Science

The goal of this project is to study and apply machine learning/artificial intelligence techniques to a link prediction problem. Link prediction is the problem of predicting the existence of a link between two entities in a network. The problem has recently attracted a lot of attention in many domains. For instance, in social networks, one may be interested in predicting friendship links among users, while in biology, predicting interactions between genes and proteins is of paramount importance. In this challenge, we dealt with the problem of predicting whether a research paper cites another research paper. More specifically, we were given a citation network consisting of several thousands of research papers, along with their abstracts and their lists of authors. The pipeline that is typically followed to deal with the problem is similar to the one applied in any classification problem; the goal is to use edge information to learn the parameters of a classifier and then to use the classifier to predict whether two nodes are linked by an edge or not.

I. INTRODUCTION

The full code of our project is available on Ikhlass' GitHub (https://github.com/ikhlo) and everything can be launched on a basic computer (there is no need of an extra large amount of memory nor a brand-new GPU). For your information, we launched everything on a Windows 10 Asus laptop with an Intel i7-7500 processor, dual-core, 8go of RAM and Nvidia GeForce 940MX graphic card.

II. DATASETS

In order to do that, we have at our disposal four different datasets:

1. edgelists.txt: a citation network created from papers published at machine learning, artificial intelligence, data mining and natural language processing venues. Nodes correspond to papers, while edges represent citation relationships. The graph is undirected. Therefore, if there is the line u, v in the file, then either paper u cites paper v or paper v cites paper u. The graph consists of 138 499 vertices and 1 091 955 edges in total. Let's see in Fig.1 how are the distributed the degrees of the nodes.

   We can see that a large majority of the papers either cite or are cited by less than 20 other papers present in the file. The most cited paper (or paper that cites the most other papers) creates a node with 3037 edges, but the mean is 16 degrees and the median is 9.

2. abstracts.txt: the abstracts of the 138 499 papers. Each row of the file contains the ID of a paper and its abstract, separated by the string $|--|$. Let's look at how many words the abstracts contain in Fig.2.

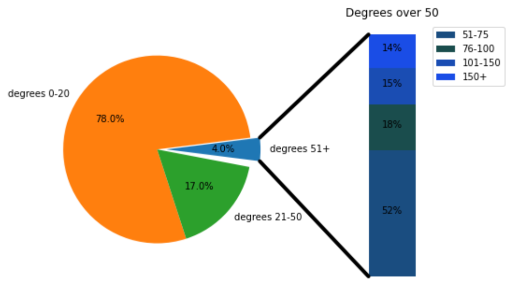

FIG. 1. Distribution of the degree of the nodes

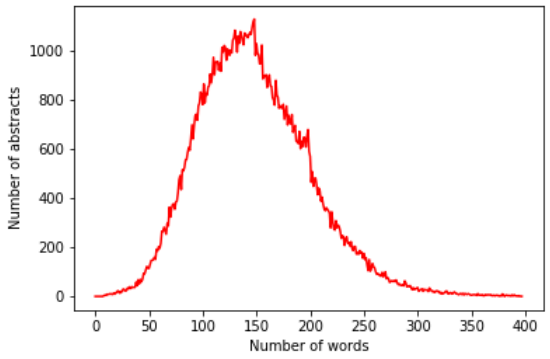

FIG. 2. Number of words per abstract

We can see that most of the papers have an abstract of between 100 and 200 words. In fact, we removed them while making this graph, but we have a lot of papers with an empty extract (7249 to be precise). If we count them, the mean is 143 words per abstract and the median is 142, and if we remove them we have a mean of 150 and a median of 145.

3. authors.txt: the authors of the 138 499 papers. Each row of the file contains the ID of a paper and its authors, separated by the string $|--|$. For papers that have more than one author, the comma character ”,” is used to separate the different authors. Let’s check the distribution of the number of authors in the file, showed in Fig.3:

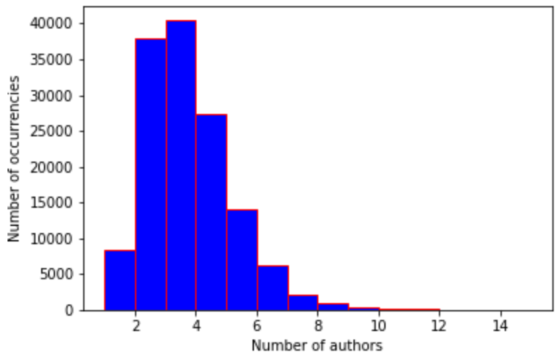

FIG. 3. Number of occurrences of the number of authors

In reality, there are academic papers with more than 100 authors (the paper with the most authors in the file has 155 authors), but as we can see in Fig.3, more than half the papers have between 2 and 3 authors and more than $\frac{3}{4}$ of them have between 2 and 4 authors. In fact, the mean number of authors per paper is 3.3 and the median is 3.

4. test.txt: 106 692 ordered node pairs. Each row of the file contains the ID of the source node and the ID of the target node. The final evaluation of our methods will be done on these node pairs and the goal will be to predict if there is an edge between the two elements of each pair or not.

III. EVALUATION

The performance of our models will be assessed using the logarithmic loss measure, also known as binary cross entropy loss. This metric is defined as the negative log-likelihood of the true class labels given a probabilistic classifier’s predictions. Specifically, the binary log loss is defined as:

$$L = -\frac{1}{n}\sum_{i=1}^{N}\big(y_i\log(p_i) + (1-y_i)\log(1-p_i)\big)$$

where N is the number of samples (i.e., node pairs), y_i is 1 if there is an edge between the two nodes and 0 otherwise, and p_i is the predicted probability that there is an edge between the two nodes.

IV. ABSTRACTS PRE-PROCESSING

Before working with the abstracts in order to create features with them using Natural Language Processing techniques, they have been pre-processed. Each time we discuss a feature that needs the abstracts, just know that we are talking about the pre-processed abstracts. About the pre-processing of the text, for one abstract we first set all its characters as lower characters and split it into a list of tokens. Then we use a tagger from the NLTK library and define a function to create a list of tuples, with each tuple containing a token and its tag. We applied a filter operation to only retrieve tokens defined as noun, adjective, verb and adverb and took care to remove English stop-words with NLTK stop-words list. To finish, we stemmed each token and finally end with a list of stemmed tokens. The same process was applied on all abstracts efficiently.

V. FEATURE ENGINEERING

We have 20 different features (plus two embeddings), that we use between nodes that we know for sure are connected (taken from the edgelists.txt dataset) and between nodes that we suppose are not connected (two nodes taken randomly and that are not linked in the edgelists.txt dataset but that might still be connected, since some of the edges have been removed on purpose for the challenge).

A. Sum of number of unique terms of the two nodes’ abstracts

For this feature, we calculate the sum of unique words of the abstract of the first node, and we add it to the sum of unique words of the abstract of the second node (if these two abstracts have words in common, they will be counted only once). The bigger this number gets, the more different words the abstracts have.

B. Absolute value of difference of number of unique terms of the two nodes’ abstracts

For this feature, we calculate the sum of unique words of the abstract of the first node, and we subtract it to the sum of unique words of the abstract of the second node, and we take the absolute value of this number. The lower this number gets, the more these two abstracts have the same number of unique words (the two abstracts can have totally different words from one another and this number can still be very low).

C. Number of common terms between the abstracts of the two nodes

We calculate the intersection of the two abstracts (meaning the number of words that they have in common, but if they have the same word in common twice it will only count once) and we take the length of this intersection. The bigger this number gets, the most common terms the two abstracts have.

D. Sum of degrees of the two nodes

We calculate the degree of the first node, and we add it to the degree of the second node. It assumes that the more a paper A cites or is cited, the more chance it cites (or is cited by) another paper B.

E. Absolute value of difference of degrees of the two nodes

Same thing than above, but we subtract the degrees instead of adding them, and we take the absolute value. The smaller this number gets, the more the two nodes have the same degree.

F. Number of common authors of the two nodes

We calculate the length of the intersection of the authors from the two papers. The bigger this number is, the more authors in common the nodes have.

G. Jaccard similarity between the authors of the two nodes

We divide the number from the feature above by the length of the union of the authors from the two papers. The bigger the number gets (it can go from 0 to 1), the biggest the ratio of common authors between the two nodes is. Here is the formula to calculate the Jaccard coefficient between two sets A and B:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

H. Number of common linked papers of the two nodes

We calculate the length of the intersection of the neighbors from the two papers (a neighbor of paper A is whether a paper cited by paper A or a paper that cites paper A). The bigger this number is, the more neighbors in common the nodes have.

I. Jaccard similarity between the authors of the two nodes

We divide the number from the feature above by the length of the union of the neighbors from the two papers. The bigger the number gets (it can go from 0 to 1), the biggest the ratio of common neighbors between the two nodes is.

J. Resource allocation index of the two nodes

We calculate the sum of the inverse degree centrality of the neighbors shared by the two nodes. Considering that $N(u)$ (respectively $N(x)$ and $N(y)$) are the neighbours of u (respectively x and y), the Resource allocation index of the nodes x and y is defined by:

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{|N(u)|}$$

K. Adamic/Adar index of the two nodes

We calculate the sum of the inverse logarithmic degree centrality of the neighbors shared by the two nodes. Considering that $N(u)$ (respectively $N(x)$ and $N(y)$) are the neighbours of u (respectively x and y), the Adamic/Adar index of the nodes x and y is defined in the same way than the precedent feature but with a logarithmic smoothing:

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log|N(u)|}$$

L. Jaccard similarity between the abstracts of the two nodes

We calculate the intersection of the two abstracts (meaning the number of words that they have in common, but if they have the same word in common twice it will only count once), we take the length of this intersection and we divide it by the length of the union of the two abstracts (the total number of words). The bigger the number gets (it can go from 0 to 1), the biggest the ratio of common words between the two nodes is.

M. Sum of authors of the two nodes

We take the number of authors of the first node and we add it to the number of authors of the second node. The bigger this number is, the most authors in the two papers there are.

N. Number of paths of length 3 between the two nodes

We count the number of paths from the first node to the second node that go through exactly 3 edges. The bigger this number is, the more ways to go from paper A to paper B through only 2 citations there are, meaning that the two papers are probably similar.

O. Dispersion between the two nodes

We calculate the dispersion between the two nodes (a link between two nodes has a high dispersion when their mutual ties are not well connected with each other). So if we want to calculate the dispersion between u and v, we note C_{uv} the set of their common neighbors and d_v a distance function and we calculate:

$$disp(u, v) = \sum_{s,t \in C_{uv}} d_v(s, t)$$

P. Whether the shortest path between the two nodes is less than 11 or not

We calculate the shortest path between the two nodes, and if the result we get is 10 or less then this feature takes the value ”True”, otherwise it takes the value ”False”.

Q. Preferential attachment score between the two nodes

We compute the preferential attachment score of the two nodes. If we note $\Gamma(u)$ the set of neighbors of the first node and $\Gamma(v)$ the set of neighbors of the second note, then we calculate:

$$|\Gamma(u)||\Gamma(v)|$$

R. Salton index between the two nodes

We measure the cosine of the angle between columns of the adjacency matrix, corresponding to given vertices. In other words, for each nodes pair u and v, if we note k_u and k_v their degrees and $\Gamma(u)$ and $\Gamma(v)$ their sets of neighbors, we calculate:

$$s_{uv} = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{k_u \times k_v}}$$

S. PageRank between the two nodes

We calculate the absolute value of the difference of the PageRank of the two nodes. PageRank computes a ranking of the nodes in the graph based on the structure of the incoming links.

T. Eigenvector centrality between the two nodes

We calculate the absolute value of the difference of the eigenvector centrality of the two nodes. Eigenvector centrality computes the centrality for a node based on the centrality of its neighbors. For a node u, if we note A the adjacency matrix of the of the graph and λ its eigenvalue, the eigenvector centrality of this node is:

$$A \times x = \lambda \times x$$

U. Text embeddings using BERT

To create the representation of the abstract we used the pre-trained BERT model, described in [1] .

V. Author embeddings using Node2Vec

To obtain the representation of each author using the giving data, we construct two graphs of authors G_1 and G_2. The nodes of each graph represent authors. In G_1, the edge (a_1, a_2) exits if at least one paper of a_1 cites paper of a_2. The weight of this edge is the number of these citations. In G_2 we use the same philosophy but for cooperation between authors.
To create node representations of these graphs we use Deepwalk algorithm where transition probability is proportional to the weight of the edge.

W. Feature importances

Let’s look at the feature importances (impurity-based) in Fig.4.

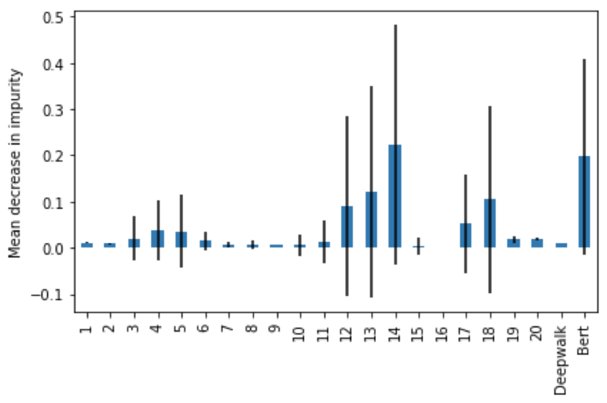

FIG. 4. Feature importances using MDI

The higher, the more important the feature. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature (also known as the Gini importance). As we can see, some features, such as 13, 14 or the Bert embeddings are more important than other features like 15, 16 or the Deepwalk embeddings. This graph was created using a Random Forest Classifier model.

VI. MODEL

A. Stacking model

We used a stacking model, which is an ensemble machine learning algorithm, which used a meta-learning algorithm (in our case, Logistic Regression) to learn how to best combine the predictions from several base machine learning algorithms (in our case, Random Forest, Gradient Boosting, Linear Support Vector Classification, Gaussian Naives Bayes and Logistic Regression). In particular, we used a K-fold stacking model [2], which works like in Fig.5:

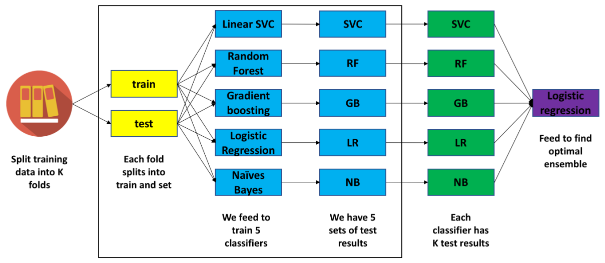
Split training data into K folds
train
test
Each fold splits into train and set
Linear SVC
Random Forest
Gradient boosting
Logistic Regression
Naïves Bayes
We feed to train 5 classifiers
SVC
RF
GB
LR
NB
We have 5 sets of test results
SVC
RF
GB
LR
NB
Each classifier has K test results
Logistic regression
Feed to find optimal ensemble

FIG. 5. How our K-Fold stacking model works

First, we split the training set in K parts. For each fold k ($k \in [1, K]$), the k^{th} part of data will be used as the test data, while the rest of the parts will be used as the training data (a set of individual classifiers will be trained over it). We chose 5 classifiers because we took the most famous algorithms for supervised classification and didn't want the training time to be too long. For each of these basic classifiers, the trained model is tested on the k^{th} part of data. After the K folds training, each classifier has K sets of test results where the union is the whole training set. We feed the test results of all the basic classifiers to a Logistic Regression, to find the best ensemble of the set of classifiers. The benefit of stacking is that it can harness the capabilities of a range of well-performing models on our classification task and make predictions that have better performance than any single model in the ensemble. After running a Grid Search algorithm with a default 5-fold cross validation, here are the hyperparameters that achieve the best performance on our dataset for all the models of the stacked ensemble:

Penalty	L1	L2
Loss	Hinge	Squared-hinge
C	0.1	1

TABLE I. Hyperparameter tuning for Linear SVC

Max features	sqrt	log2
Number of estimators	100	1000

TABLE II. Hyperparameter tuning for Random Forest

Learning rate	0.01	0.1
Number of estimators	100	1000
Loss	Deviance	Exponential

TABLE III. Hyperparameter tuning for Gradient Boosting

Penalty	None	L1	L2	elasticnet
C	0.01	0.1	1	10

TABLE IV. Hyperparameter tuning for Logistic Regression

Variance smoothing	1e-12	1e-9	1e-6

TABLE V. Hyperparameter tuning for Naive Bayes

B. GNN

We propose a GNN-based architecture for citation prediction. The diagram of the model is presented on Figure 7. First, the feature vector for each node of the citation graph is created by concatenating abstract embeddings(Section V U) and author embeddings (Section V V). Next, the citation graph with features is passed to a SAGEConvNet [3]. The SAGEConvNet is a convolutional graph neural network developed for handling large graphs. As the output of a GNN we get a node representation $\mathbf{h}_i$ for each node. To obtain a score for the pair (i, j) there are three main options:

1. $y_{ij} = \sigma(\mathbf{h}_i^\top \mathbf{h}_j)$. This approach encounters the similarity between two nodes to predict the edge, however such classifier has no trainable parameters and thus it has poor generalization.

2. $y_{ij} = f_{\mathrm{MLP}}(\mathbf{h}_{ij})$ where $\mathbf{h}_{ij} = [\mathbf{h}_i, \mathbf{h}_j]$. This classifier has trainable parameters and we can expect better performance here, however it is easy to see that in such approach $\mathbf{h}_{ij} \neq \mathbf{h}_{ji}$ and this leads to $y_{ij} \neq y_{ji}$. This does not match with the general idea of the problem.

3. $y_{ij} = f_{\mathrm{MLP}}(\mathbf{h}_{ij})$ where $\mathbf{h}_{ij} = (\mathbf{h}_i - \mathbf{h}_j)^2$. This classifier combines best of two models above: it has trainbalble parameters, it is invariant to permutations ($y_{ij} = y_{ji}$ and encounters the similarity between two nodes We use the third approach in our final model since it significantly outperforms two other approaches.

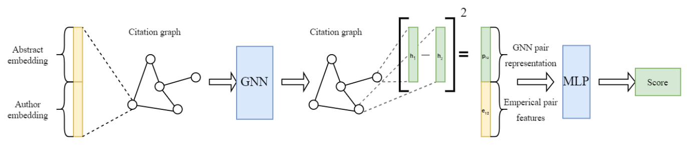

FIG. 6. GNN architecture

C. XGBoost

The final model we used was the XGBoost one [4], which is a popular and efficient open-source implementation of the gradient boosted trees algorithm.

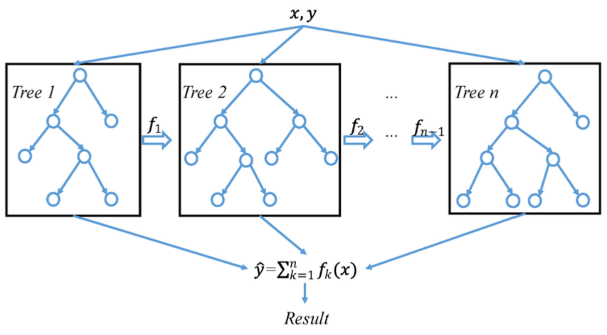

FIG. 7. XGB architecture

Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models. XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity. The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It's called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

D. Results

Let's compare the F1 score of the 5 algorithms individually, of the stacked ensemble, of the GNN and of XGBoost in Fig.9.

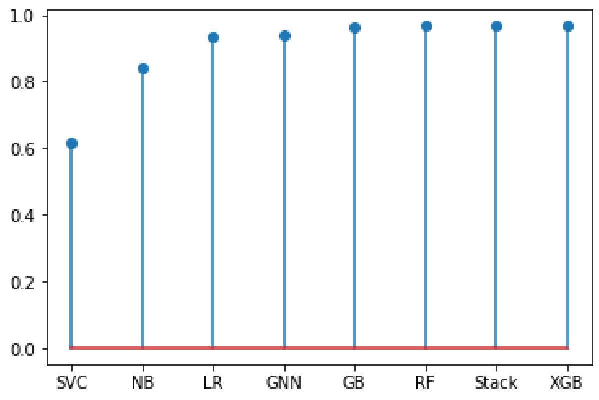

FIG. 8. F1 score for each model

As we can see, except for SVC that has a F1 score of around 0.25 lower and Naive Bayes that has a F1 score of around 0.1 lower, all the other models (including the stacked one and the GNN) have an F1 score of around 0.95. We can see that we obtain our best F1 score with the XGB model. This score was calculated while putting 20% of the training set of size 200 000 (nodes pair taken randomly from the given dataset) into a validation set on which we tested the models previously fitted on the training set. The F1 score is the harmonic mean of the precision and recall [5] and is calculated as follow:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$$

where TP means True Positive (pair nodes which we predicted as linked and are indeed linked), FP means False Positive (pair nodes which we predicted as linked but are in fact not linked) and FN means False Negative (pair nodes which we predicted as not linked but are in fact linked).

We can also compare the different losses, which will be more adequate because we can have a F1 score of 1 and still have a huge loss. Moreover, the binary cross entropy is how we will be evaluated, so we should select the method that gives us the best results.

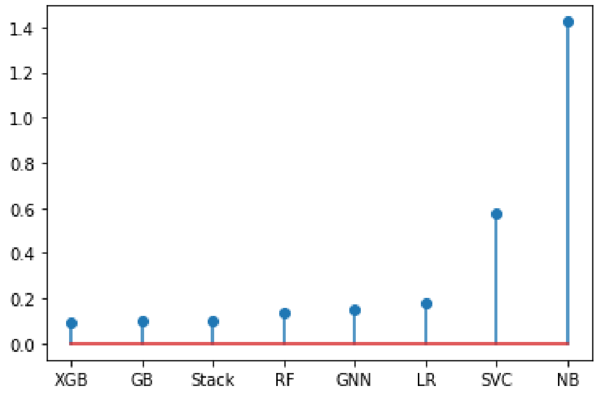

FIG. 9. Loss for each model

We can see that, as for the F1 score, we achieve our best loss with the XGB model. Then, there is not much of a difference with the other algorithms (Gradient Boosting, Stacked Ensemble, Random Forest, GNN and Logistic Regression) who all have a loss around 0.15, except for Linear SVC that has around 0.7 and Naive Bayes that doubles it with a loss around 1.4.

VII. CONCLUSION

As a conclusion, we arrived at a final loss of 0.09366, which allowed us to be 4th in the public Kaggle leaderboard. To obtain this score, we used a training set of only 200 000 data points (instead of more than 2 millions), taken randomly from the edgelists file (100 000 connected pairs and 100 000 unconnected pairs). During the entire challenge, we felt that the crucial step was really more about the feature engineering than the model, because even though we put a lot of time in training an adequate stack of estimators, a GNN and an XGBoost, it didn't improve our score by a lot, and we could even have used a simple classifier like Logistic Regression and have more or less the same loss. Moreover, some of the techniques we used, whether they were algorithms (like the SGD or the AdaBoost classifiers that didn't improve the F1 score of previous supervised algorithms for classification), neural networks (we tried different architectures involving different numbers of layers and neurons but every time the loss was bigger than the one of our XGBoost) or features (for example we calculated for each pair of nodes the length of the intersection of the number of common words between the 10 000 less commons words from the whole file, because we felt like if two papers have some words that are very rare in common, they might cite each other, but the feature was ignored by our algorithms and its importance was almost null) were removed from our project for different reasons, whether they were too long to compute or didn't improve our loss when we submitted our project (or both).

[1] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," (2019), arXiv:1908.10084 [cs.CL].

[2] Y. Zhou and A. Sharma, in *Proceedings of the 2017 11th joint meeting on foundations of software engineering* (2017) pp. 914–919.

[3] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," (2018), arXiv:1706.02216 [cs.SI].

[4] T. Chen and C. Guestrin, in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016) pp. 785–794.

[5] Y. Sasaki *et al.*, URL: https://www. cs. odu. edu/~ mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07. pdf [accessed 2021-05-26] (2007).