Project Report: Paper Importance Prediction
Web Mining Project - Team 7

Roman Bogdanov, Nathanael Stelzner, Victoria Zevallos, Vitor Faria De Souza, and Sharan Shyamsundar

School of Business Informatics and Mathematics, University of Mannheim

1 Introduction

Scientific research is a highly volatile field, with 1.8 million articles published each year in about 28 000 journals [17]. Researchers are usually focused on some subsection of the field, but keeping updated with the recent research is still often a challenge. It is also a challenge for the scientific journals, who not only need to keep an eye on the novel research, but also need to evaluate the papers submitted to them for publication. We believe that data analysis and machine learning can be of help in this situation. Driven by this belief, in this work, we apply diverse supervised learning methods to estimate the future importance of a scientific paper using only the information available at the time of its publication. Specifically, we leverage the properties of the paper's citation network and features of the paper itself. We do so using data from High-Energy Physics Theory Citation Network dataset. The methods we apply are divided into two groups: classical approaches which rely on manually generated features to predict the target variable, namely Decision Tree, Random Forest, Gradient Boosting, Support Vector Machine, Naive Bayes, and Multi-Layer Perceptron, and innovative Graph Neural Network (GNN) approaches, which do not require features to be generated manually and instead leverage the overall state of citation network in an automated manner. Therefore, we are driven by the following questions: (1) Can machine learning methods provide a reliable estimation of future importance of scientific papers? (2) Which models are best suited for this task? (3) How do GNN models perform compared to the classical approaches?

Supporting materials for this paper, i.e. python code, datasets, etc. are available in a GitHub repository: https://github.com/Nathanael210/WebMining. The repository follows Cookiecutter template [7].

2 Methodology

2.1 Dataset Description

The data used in this project is the High-Energy Physics Theory Citation Network, provided by the Stanford Network Analysis Platform (SNAP) [13], where each of the 27 770 scientific papers has a submission date and can cite other papers that have been previously submitted. In terms of a network, each node

is a paper and each citation is a directed edge pointing to the cited paper. The network consists of 352 807 directed edges between its 27 770 nodes, and submission dates ranging between 1993 and 2003. The meta information available for each paper are its submission date, title, author, journal, number of pages and figures, and abstract.

2.2 Target Definition

The dataset used in this project was first released as part of the 2003 KDD Cup [9], in which one of the proposed tasks was to predict the number of citations each paper would receive during the last three months of 2003. For our project, the target label that represents the paper's relevance is defined as the number of citations each paper would receive during the first year (365 days) after its submission date, binned in 4 groups: 0 citation (C1), 1-10 citations (C2), 11-20 (C3) and more than 20 (C4). Therefore, this problem could be assessed as a classification task for papers submitted until 2002, with citations received until 2003.

2.3 Data Preprocessing

Raw Data Original data is available from three sources: (1) a text file "Cit-HepTh.txt" containing a list of edges - citations - between the nodes of the graph - papers; (2) a text file "Cit-HepTh-dates.txt" containing dates of publication of all papers in the database; (3) 12 yearly folders from year 1992 to year 2003, containing text files for each of the papers published during the respective year, the text files containing metadata for the papers. Our first preprocessing task is to convert this multitude of text files into a more appropriate data format. This results in two intermediary data sources: a graph data structure from networkx python library and a pandas dataframe with metadata. We discuss them in more detail next.

Graph Creation The dataset contains a file describing the graph. Each line consists of the paper IDs of source and target of an edge. To be able to access the graph fast, we created a DiGraph object using the python module NetworkX. We wrote a function that reads in the file containing the edges and adds all of them to the graph. In the lines of another file, there is the paper ID and the corresponding submission date for all papers. To have the dates available with the graph, we added the date in the node-dictionary for every paper.

We do a few tests on the graph to check some properties. In doing so, we realize that 1 044 edges (which correspond to the citations) go into the future. That means, the submission date of the cited paper is after the date of the paper that cites it. Since this is not possible, we decide to delete these edges.

Target generation While generating the target variable (number of citations received by paper 365 days after publication), we look at a variety of alternative targets. This includes number of citations a paper has received from the date of its publication until the end of the dataset's timeline (V1), number of citation a paper has received within 90 days of its publication (V2), within 180 days of its publication (V3), within 365 days (V4), and within 730 days (V5). We cannot use V1 as our target, since it could not be uniformly computed for all papers: for some papers this time period will be longer, for some - shorter. This variable would, however, be the most representative of the entire dataset, as there would be no loss of data. We then compute correlations between the considered variables. The longer the time period, the higher the correlation to V1. Following this logic, we would choose V5 as our target, but there is a complication: V5 has a highly unbalanced frequency distribution. Therefore, we have to compromise between two factors: high correlation to V1 and relatively uniform distribution of frequencies. The best compromise is V4, as it is distributed more uniformly, but still has a high correlation to V1. We then apply a binning procedure with four groups: C1, C2, C3, C4 (see section 2.2). The number and the borders of these bins are decided on while trying to distribute papers among the bins more uniformly. We also consider not applying any binning to the target variable at all, but decide against doing so due to the unbalanced distribution of our data.

Metadata cleaning The submitter, submission date, title, abstract, authors, comments, report number, and journal reference are all included in the metadata for each publication. These attributes are used to extract some additional features that assist us in achieving our primary goal of prediction. There are a few data cleaning steps taken. We utilize regex to tidy up the text because the majority of the features are string types. This is done for the submitter, the title, the abstract, the authors, the comments, the report_no, and the journal_ref. The submitter feature is then used to get the submitter's name and email address. This is tidied up much more to produce distinct submitter identities. When a paper was revised, the submission date has multiple dates showing when the paper was submitted and revised. We extract each paper's first submission date and generate a variable that represented how many times it was edited. The number of pages and the format in which the paper was written are both listed in the comments section. After that, the journal reference attribute is cleaned to create a unique format that could be used to calculate the number of publications in a journal. Finally, the citations each submitter has received up to the date they publish a new paper are computed using the submitter details, date, and citations data.

Feature generation Using the graph data structure and the metadata files, we generate a diverse set of features. Their complete list with detailed descriptions can be found in the Data Dictionary in the project's GitHub repository. The features can be roughly divided into two groups: graph-based and metadata-based. The first group includes things like in- and out-degrees of papers cited

by the paper in question, average recency of the cited papers, maximum time difference between cited papers. The second group includes features describing properties of the paper itself: its number of pages, whether and how many times it has been revised, in which file format it was uploaded, how popular its author-submitter is. We also create two features to encode the date of submission of the paper, using sine and cosine to introduce cyclical information of the week and the month.

We should note that a particular challenge for us while working with the graph data has been taking the time perspective into account. Let us take, for instance, the "sum of in degrees of cited papers" feature, i.e., how many citations the papers cited by this paper have received so far. We must look only at such citations that were received before the date of publication of the paper in question if we do not want the future data to influence our analysis. This we keep in mind while working with all graph-based features.

We also fill missing values in the newly generated metadata-based features (there are no missing values in graph-based features). In most cases, this means filling the missing values with either 0 or an empty string, except for the page count, where we use the number 10 instead.

2.4 Modeling

The overall goal of this project is to predict how relevant an article will be in the scientific community after its publication. We frame the problem as a classification task in which we predict the rank of relevance the paper will have after one year. To do this, we leverage the citation network to generate features from its structure and combine them with features generated from the paper's metadata to train a classifier. In this context, we implement two separate frameworks: the first is based on a classical approach, in which we train Machine Learning Classifiers with the features described above, and the second uses Graph Neural Networks that extract the graph structure and perform the classification task all at once.

Classical Classifiers When applying classical classifiers, we use the standard toolset of data analysis: scikit-learn [14] pipelines with standard scaler and feature selection in them, as well as grid search for hyperparameter tuning with a 5-fold cross-validation. We also apply class weights to mitigate the imbalanced nature of our dataset. They are calculated as inversely proportional to class sizes. We apply class weights instead of resampling, because we want to avoid loss of data in case of undersampling and increased computation costs and overfitting in case of oversampling. In every grid search we use the number of features, the feature selection step is selecting as hyperparameter.

Naïve Bayes is a supervised learning method based on Bayes' theorem, with the assumption of conditional independence between every pair of features given the value of the class variable. We apply the version of the method called Gaussian

Naïve Bayes, which assumes the probability of features to follow the Gaussian distribution. It is possible to tune the var_smoothing parameter of the model, which represents the portion of the largest variance of all features that is added to variances for calculation stability. We try 10 values for this parameter: 1.e+00, 1.e-01, 1.e-02, going until 1.e-09. We get the best result at 1.e+00. [1] [18]

Multi-Layer Perceptron Classifier Multi-layer perceptron is also a supervised learning model. It implements a neural network with one or more hidden layers. The neurons in the hidden layers transform input values with a weighted linear summation and apply a non-linear activation function to the results. We fix the number of hidden layers at one and conduct hyperparameter tuning for the number of neurons in it. [2]

Decision Tree Classifier The Decision Tree Classifier [3] uses a function to measure the impurity of the data. It creates splits using one or more attributes such that the impurity decreases. In this way, a tree is built that partions the whole dataset. The leaves contain the most probable class, which is used as prediction. The hyperparameters we are tuning are min_samples_leaf, max_depth and criterion.

Random Forest Classifier Using some randomness, the Random Forest Classifier [4] builds multiple Decision Trees. The most common decision of all the Trees is chosen as prediction. The hyperparameters we are tuning are min_samples_leaf, n_estimators, max_depth and criterion.

Gradient Boosting Classifier Similar to the Random Forest Classifier, the Gradient Boosting Classifier [8] is building multiple Trees. To improve the prediction, the negative gradient is used to update the trees. These optimization steps are possible since a differentiable loss function is used. The hyperparameters we are tuning are min_samples_leaf, max_depth and criterion.

Support Vector Machine Classifier The Support Vector Machine Classifier [5] divides the data using straight lines into areas belonging to different classes. The use of kernelization makes it possible to do the division also with curves. The hyperparameters we are tuning are kernel and max_iter.

Graph Neural Networks Our second framework is based on Graph Neural Networks (GNN), which is a deep learning technique with the ability to analyze graph structure data. The reason to implement this approach is that the complex structure of graphs often hampers the capability of gaining the true insights underlying the graphs [19]. The main idea behind using neural networks with graphs is to learn the nodes, edges, and graph representations in a low-dimensional Euclidean space via embedding techniques [15]. In our case we are interested in the nodes as our task is node classification, therefore we want to map the nodes into an embedding space based on its location in the network [20].

We expect that in this space, the similarity of the nodes embeddings approximates the real similarity on the graph. Once the embeddings are learned, we can use them to classify the papers in one of our four categories of relevance. It is important to notice that as many deep learning approaches, this architecture learns the embeddings of the nodes and the parameters for the classification at the same time, end-to-end training. [12]

The domain of GNN covers a variety of different algorithms and not just a single architecture [20]. The most common GNN architectures can be divided into three groups: spectral methods, spatial methods and sampling methods. The first ones, work with graph signal processing and define the convolution operator in the spectral domain using a Fourier transformation [11]. Graph Convolutional Networks [12] (GCN) are part of this group. The second group define convolutions directly on the graph based on the graph topology, they transform the node's features using a projection, then the representation of the neighborhood is aggregated to update the representation of the node [11]. Graph Attention Networks [16] (GAT) belongs to this group. The two previous groups use all the neighborhood to build the feature vector of a node, in contrast, the last group just uses a subset of them for the propagation[11]. The implementation of GraphSage follows this conception.

After evaluating the implementation of different architectures, we decide to use GraphSage[10], as it is a more efficient and scalable algorithm, and it trains with inductive learning. In inductive learning, the model sees only the training data and then is able to create embeddings and predict class labels for unseen data. This property ensures that our model can be used to predict the relevance of papers just released without having to train the model again. It is important to mention that the majority of implementations of GNNs like the ones mention above, do not use inductive learning, instead they use transductive learning and can only generate embeddings for a single fixed graph [10].

In a nutshell, GraphSage samples uniformly a set of nodes from the neighborhood, then aggregates the feature information from the sample neighbors to create a vector representation of the node [10]. During the forward propagation, on each layer, we extend the neighborhood depth. Then it uses this embedding to perform node classification. An illustration of this process can be seen in Figure1.

To implement our architecture, we use StellarGraph [6] library built on TensorFlow. The structure of our architecture can be divided in two parts, the first one is the embeddings generation, or forward propagation algorithm, and the second one is the classification head. The embedder has two layers of 32 neurons each of them and one mean aggregator function per layer. The mean aggregator simply takes the element-wise mean of the vectors [10]. The activation function for both layers is RELU. The classification head consists of a fully connected layer with 4 neurons, as we have 4 classes, with a softmax function for activation.

To train the GraphSage architecture we use 100 epochs with a learning rate of 0.0001, as we are fine-tuning a pretrained model, and the Adam optimizer

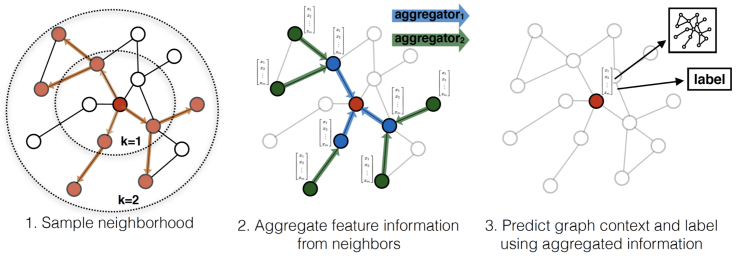

Fig. 1. Visual illustration of the GraphSAGE sample and aggregate approach adopted from [10]

and a batch size of 50. During the training, 1 796 parameters were optimized. These include the train of the aggregation function alongside with our learnable weight matrices. In consequence, the network also learns how to aggregate the features from the sampled nodes. To overcome the problem of working with an unbalanced class label, we use a value inversely proportional to the class weights to punish the mistakes that the model makes in the minority classes.

We train 3 different configurations of the model, all of them with date features as node attributes. The first (GraphSageU) ignores the implicit directionality of the network, and therefore treats the graph as undirected. The second (GraphSageD) recognizes the directionality of the edges. The third (GraphSageD meta) is also a directed network, but includes other metadata features as node attributes rather than just dates.

Such attributes included in the third configuration are the same ones used in classical classifiers, except for those derived directly from the graph structure. For instance: the average recency of cited papers, the maximum time difference between paper's citations, number of pages, format of publication, how many times it was revised, journal popularity and author popularity at the time of submission. The reason for ignoring graph-based features was that the GNN architecture is expected to learn this information from the graph itself.

To create training instances suitable for the model, we use a generator function that iterates over the nodes of the training set to create sub-graphs for each node in the batch. For the first configuration, we use an undirected generator. It samples 10 nodes in the first layer, and 4 nodes in the second layer from the neighborhood a node. For the second and third configuration, we use a directed generator that selects 10 out nodes in the first layer and 4 out nodes in the second layer.

3 Experimental Setting

3.1 Structure and statistics

We have 27 750 unique papers and 39 characteristics including the paper_id identifier and the target variable after the network data and metadata have

been preprocessed. There are 23 numeric variables, 13 text variables and 3 date format variables in the dataset. Each year, the number of articles published has progressively climbed, from around 1 200 papers in 1992 to around 3 200 papers in 2002. There are 9 904 submissions totaling 27 750 papers with a total of 9 904 submitters.

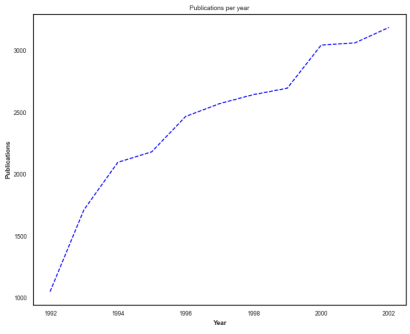

Fig. 2. Publications Trend

It is observed that the features obtained from the graph have a moderate to high correlation. The metadata-generated traits, on the other hand, have a lower correlation among themselves.

Target Variable When we look at the correlations with our target variable we see that the graph produced features have a strong relationship with it, whilst the metadata features have a weak relationship.

As previously stated, the target variable is calculated based on the number of citations a paper receives within the first 365 days of publication. The majority of the papers have between one and ten citations. There are very few papers with more than ten citations, indicating that the classes are unequally distributed.

The distribution of citations in the first year follows the power law, as it does in most network problems. As previously stated, citations in the first year are extremely skewed, with the vast majority receiving no citation in the first year. It's more tough to predict the number of citations in the first year, so we break it into bins.

3.2 Evaluation Framework

As the target label consists of 4 imbalanced classes (the vast majority of papers receive either 0 or 1-10 citations in their first year), so macro average F1-score was the metric used to evaluate and compare all models against the baseline of predicting the majority class for all papers.

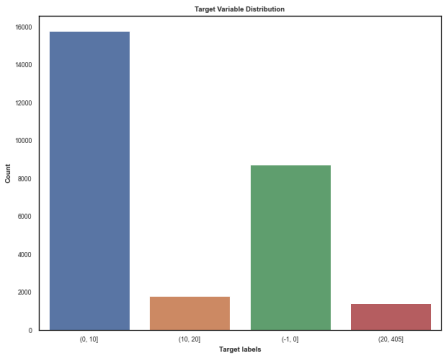

Fig. 3. Class Distribution

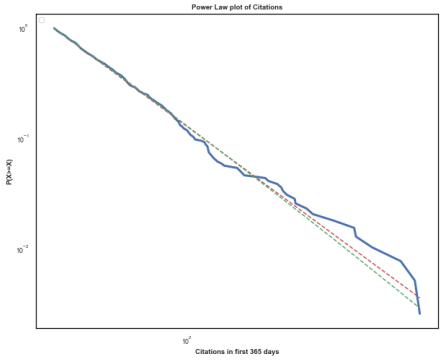

Fig. 4. Power Law of Citations

The split between training and testing set is based on submission dates. In different models three approaches to doing the split were taken. (1) In Naive Bayes and Multi-Layer Perceptron approaches papers submitted within one year before the last date of the dataset, which is 01.05.2003, were discarded: as their labels are not completely available: one can not compute the number of citations received in a year if a paper has been published for less than a year. After that a cutoff was made on a fixed date of 01.01.2002. Every non-discarded paper after it is was used for testing and every paper before - for training. This resulted in 23567 papers in the train set and 985 in the test set (2) In GNN approaches the last year was not discarded and the cutoff was made on the same date of 01.01.2002. This resulted in 23567 papers in the train set and 4183 in the test set. (3) In Decision Tree, Gradient Boosting, Random Forest, and SVM approaches last year was discarded, but the cutoff was made not on the fixed

date of 01.01.2002, but on 01.05.2001, which is exactly two years before the last date of the dataset. This has resulted in 21 464 papers in the train set and 3088 in the test set.

4 Results

4.1 Evaluation

To be able to compare the results of our different classifiers, we computed two simple baselines, which are shown in table 1. MINORITY refers to predicting every instance in the test set as the minority class of the training set, which is *C3*. MAJORITY does the same, using the majority class *C1*.

Since we are interested in the overall capacity of the model to classify the papers in the correct relevance class, we do not only compare our classifiers on their Macro Average F1-Score, but also take a closer look the to F1-Scores for each class in the test set. Those results are presented in table 2

Baseline	F1 *C0*	F1 *C1*	F1 *C2*	F1 *C3*	F1 Macro Avg
MAJORITY CLASS	0.00	**0.69**	0.00	0.00	**0.17**
MINORITY CLASS	0.00	0.00	0.00	**0.09**	**0.02**

Table 1. Baselines used for evaluation.

Looking at the classical models, we can see that Naive Bayes performs better than the other models with respect to F1-macro metric, followed by Multi-Layer Perceptron and Decision Tree. This is not a surprising result, since simpler models like Naive Bayes or Decision Tree are known to sometimes outperform the more sofisticated approaches like SVM or Gradient boosting.

Performance results for separate classes differ from the overall performance of the models. For C0 Naive Bayes performs best with F1 score of 0.5. For C1, Gradient Boosting performs best with F1 score of 0.74. C2 is overall the class where all the models perform poorly, but Decision Tree performs better than the others with F1 of 0.18. For C3, Naive bayes performs best with a score of 0.28.

Among the three GraphSage implementations we tested, the one with the worst performance is the case where we ignore the directionality of the graph. This classifier performs just like the baseline, as it classifies all articles as part of the majority class, C1 (1-10 citations). These results reveal that directionality is a very important dimension for this task, and the model should be able to incorporate this information to improve its predictions. We do so in the other two implementations, where we train the model with a directed graph and capture this information. The resulting performance of these models increase significantly, ranking as the best models so far with a F1-macro of 0.49, for the version that include just the date features and a F1-macro of 0.48 in the version that include additional metadata features. In addition, the F1-score of the class C0

Classifiers	F1 *C0*	F1 *C1*	F1 *C2*	F1 *C3*	F1 Macro Avg
Multi-Layer Perceptron	0.39	0.77	0.00	0.27	**0.36**
Decision Tree	0.42	0.64	**0.18**	0.15	**0.34**
Gradient Boosting	0.37	**0.74**	0.06	0.10	**0.32**
Random Forest	0.24	0.76	0.0	0.0	**0.25**
Support Vector Machine	0.26	0.71	0.09	0.13	**0.30**
Naïve Bayes	**0.50**	0.67	0.16	**0.28**	**0.40**
GraphSageU (date)	0.00	**0.69**	0.00	0.00	**0.17**
GraphSageD (date)	**0.99**	0.66	0.11	0.22	**0.49**
GraphSageD (date+meta)	0.91	0.59	**0.14**	**0.26**	**0.48**

Table 2. F1 scores of Classifiers

(0 citation) is 0.99, so the model is optimal predicting this class. We also see that including additional metadata features don't improve the performance of the model. From this result, we deduce that the directionality and the aggregated embeddings of the structure of the out-nodes in the neighborhood have a positive impact on the performance of the model.

All models perform better than the baseline. Out of the classical models, Naive Bayes achieves the best result. The GraphSage (directed case) with date features provide clearly better predictions than the classical approaches. So we can conclude that the more sophisticated way of the GNNs to create embeddings resulted in more useful representations than the manual feature generation we did for the classical models. This can be explained by the fact that the embeddings get updated during the training of the GNN. In this way, representations of the papers are computed such that they contain information being useful for the target prediction. In the manual feature generation, one only thinks of possible features ex-ante with no continuous feedback.

4.2 Error Analysis

Now we analyze briefly the errors of the model, to do so we present its confusion matrix in Figure 5. When looking at the predictions of the model with the best performance, it is possible to see that the most common mistake it does is to predict class C3 (more than 20 citations) for papers which belong to the majority class C1 (1-10 citations). The model can not differentiate well between these ranges. Particularly the model has the poorest performance classifying class C2 (11-20 citations), where the precision is 0.13, and the recall is 0.09, and the F1-score of 0.11. This might be a limitation of assessing this problem as classification instead of regression. Class C3 is closer to class C2 than to classes C0 and C1, but in a classification approach each class is treated independently.

5 Conclusion

Predicting future importance of a paper is a complex task, many academic journal editors struggle to do it throughout their careers. However, the data from a

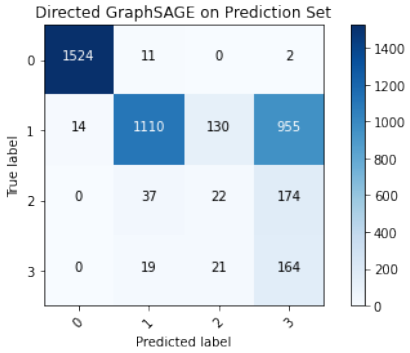

Fig. 5. Confusion Matrix in the Test set for the GNN GraphSAGE D (data)

paper's citation network and the various properties of the paper itself make it possible to at least partially solve this task by applying machine learning methods. During our project we have applied multiple diverse methods of machine learning and have achieved results that are significantly better that our baseline. The best results we archived with Graph Sage - a Graph Neural Network method. With its F1 macro score of 0.49, compared to a baseline F1 macro score of 0.17, it is the best suited model for paper importance prediction. The results of the classical models also top the baseline, but do not keep up with GNN. The dynamically updated embeddings Graph Sage creates appear to be better predictors for our task than the static human-designed features that we generate ourselves.

Going forward, one venue of improvement would be to expand the set of manually generated features and see if this might make the classical approaches perform better than GNNs. These features could be graph-based, for instance various metrics of prominence that were too expensive to compute within the scope of this project, or metadata-based. For example, one could use text analytics methods to get vector representations for the abstracts of the papers. In this way, the models could also use the content of the papers to predict importance. Another venue of improvement would be to use a different dataset. The papers of our dataset deal exclusively with High-Energy Physics Theory. A more inclusive citation network may improve the prediction quality and external validity of our results.

To summarize, we would answer our research questions as follows: (1) Machine learning methods do provide an estimate of future paper importance. Judging by our dataset, the quality of these predictions is fair. We can imagine a model like ours being used by a journal editor to pre-assess a submitted paper before sending it to reviewers. However, more research is needed, preferably of empirical nature. (2) and (3) GNN models outperform classical models and are better suited for the task.

References

1. Scikit-Learn Naive Bayes. https://scikit-learn.org/stable/modules/naive_bayes.html, accessed: 2022.05.26
2. Scikit-Learn Neural network models (supervised). https://scikit-learn.org/stable/modules/neural_networks_supervised.html, accessed: 2022.05.26
3. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth International Group, Belmont, CA (1984)
4. Breiman, L.: Random Forests. Machine Learning **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324, http://dx.doi.org/10.1023/A%3A1010933404324
5. Chang, C.C., Lin, C.J.: LIBSVM: A Library for Support Vector Machines. ACM Transactions on Intelligent Systems and Technology **2**, 27:1–27:27 (2011), software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
6. Data61, C.: StellarGraph Machine Learning Library. https://github.com/stellargraph/stellargraph (2018)
7. DrivenData Inc.: Cookiecutter Data Science. https://drivendata.github.io/cookiecutter-data-science/, accessed: 2022.05.26
8. Friedman, J.H., (y X)-values, O.K.: Stochastic Gradient Boosting. Computational Statistics and Data Analysis **38**, 367–378 (1999)
9. Gehrke, Johannes, Ginsparg, Paul, Kleinberg, Jon: Overview of the 2003 KDD cup. SIGKDD Explor. Newsl. **5**, 149– (12 2003). https://doi.org/10.1145/980972.980992
10. Hamilton, W., Ying, Z., Leskovec, J.: Inductive Representation Learning on Large Graphs. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf
11. Karagiannakos, S.: Best Graph Neural Networks architectures: GCN, GAT, MPNN and more. https://theaisummer.com/ (2021)
12. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: Proceedings of the 5th International Conference on Learning Representations. ICLR '17 (2017), https://openreview.net/forum?id=SJU4ayYgl
13. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations (08 2005). https://doi.org/10.1145/1081870.1081893
14. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
15. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM (aug 2014). https://doi.org/10.1145/2623330.2623732, https://doi.org/10.1145%2F2623330.2623732
16. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. 6th International Conference on Learning Representations (2017)
17. Ware, M., Mabe, M.: The STM report: An overview of scientific and scholarly journal publishing (2015)
18. Zhang, H.: The optimality of Naive Bayes. Aa **1**(2), 3 (2004)

19. Zhang, S., Tong, H., Xu, J., Maciejewski, R.: Graph convolutional networks: a comprehensive review. Computational Social Networks **6** (11 2019). https://doi.org/10.1186/s40649-019-0069-y
20. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. AI Open **1**, 57–81 (2020). https://doi.org/https://doi.org/10.1016/j.aiopen.2021.01.001, `https://www.sciencedirect.com/science/article/pii/S2666651021000012`