Textual embeddings with word-type-weighted word2vec and graph neural networks

Theodor Ladin[1], Lukáš Korel[2] and Martin Holeňa[2,3]

[1]Gymnázium Nad Štolou, Prague, Czech Republic
[2]Faculty of Information Technology, CTU, Prague, Czech Republic
[3]Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic

Abstract

The increasing use of neural networks for semantic text analysis highlights the need for more efficient methods without compromising quality. We propose a lightweight approach combining traditional word embeddings with graph convolutional networks (GCNs) to improve sentence similarity recognition. By incorporating syntactic information, such as parts of speech and grammatical functions, our method reduces computational demands at least 2.5 times while maintaining accuracy, when tested statistically indifferent to larger models.

Keywords

text representation learning, text embedding, text preprocessing, word2vec,

1. Introduction

Artificial neural networks [1] allow for the modeling of complex patterns and relationships within data. However, training these models, especially large-scale architectures such as transformers, remains a major challenge due to the immense computational resources required. This limitation becomes particularly relevant when training models for smaller tasks where a high-performance infrastructure may not be available.

The present work focuses specifically on the task of identifying semantic equivalence between sentences expressed using different wording. One of the most widely used methods in text-based machine learning is *Word2Vec* [2], which enables efficient modeling of semantic relationships between words through vector representations. This technique has significantly improved the representation of meaning in large-scale text corpora and has found applications in areas such as machine translation and sentiment analysis.

However, traditional Word2Vec models have certain limitations, especially in their ability to capture semantic similarity at the sentence level. This challenge has been addressed by more recent models such as **BERT** (Bidirectional Encoder Representations from Transformers), which extend the capabilities of vector representations by incorporating a deeper contextual understanding. Despite their improved accuracy, these complex architectures come with higher computational costs and longer training times.

The focus of this study is the use of *graph neural networks* [3] (GNNs), a class of deep learning models designed for processing graph-structured data, i.e., data represented by means of nodes and edges. GNNs are particularly well-suited for capturing intricate relationships between entities, making them valuable in domains such as social networks, biological systems, and knowledge graphs. In natural language processing [4], GNNs offer a promising way to model semantic relationships between words, phrases, or entire sentences, capturing contextual dependencies more effectively. Unlike traditional approaches such as *recurrent neural networks* (RNNs) or transformers, which focus on sequential data, GNNs are capable of modeling complex structures and long-range dependencies more naturally.

The objective of this research is to develop an efficient solution that focuses on the problem of semantic similarity detection between sentences, with an emphasis on reducing computational requirements. To achieve this, the study explores the use of graph matching structures and relational graph convolutinal

ITAT'25: Information technologies – Applications and Theory, September 26–30, 2025, Telgárt, Slovakia
theodor.lagin@gmail.com (T. Ladin); lukas.korel@fit.cvut.cz (L. Korel); martin@cs.cas.cz (M. Holeňa)

networks. This differs from standard approaches by not building upon existing transformer [5, 6], but using the sole graph neural network to get best results possible.

This paper builds upon our previously published work on weighted word2vec[7].

2. Background section

2.1. Applicability of Sentence Embeddings

Text embeddings transform sentences into high-dimensional vectors, typically consisting of hundreds of dimensions, that capture their semantic meaning. These embeddings enable a range of downstream applications, including:

- **Text classification** — such as sentiment analysis or topic assignment (e.g., sports, politics).
- **Semantic similarity search** — including paraphrase detection and more accurate information retrieval.
- **Text summarization** — extracting key information for automated summaries.

One commonly used embedding method is *Word2Vec*, which maps semantically similar words to nearby points in the vector space. To compare meanings, similarity metrics such as *cosine similarity* [8] are used, where similar vectors have a value close to 1. The model is bidirectional, allowing transitions between words and their corresponding vectors. The final quality of the embeddings depends on various *hyperparameters* used during the training process.

For example, in training the **Google News Vectors Negative 300** model, the *Continuous Bag of Words* (CBOW) [9] algorithm was used. This method selects context windows, containing words both before and after the target word, and attempts to predict the target word based on these surrounding words.

2.2. Bayesian Optimization

Bayesian optimization [10] is a technique for optimizing black-box functions by modeling the probabilistic distribution of function values and selecting new evaluation points based on the expected information gain. It is widely used for *hyperparameter optimization*, where the goal is to minimize the number of function evaluations while exploring the parameter space efficiently. This approach is especially valuable in the training of neural networks, where overfitting and memorization [11] can hinder generalization, and careful tuning of hyperparameters is crucial for performance. This approach was especially useful when training the R-GCN 3.3 network, where it could affect how it either memorized or correctly learned. Bayesian optimization was therefore used on dropout, weight decay, learning rate and amount of hidden dimensions.

2.3. Graph Convolutional Networks

Graph Convolutional Networks (GCNs) extend the concept of convolution from traditional grid-structured data, such as images, to graph-structured data, enabling deep learning models to operate in non-Euclidean domains. The core idea of GCNs is to aggregate information from a node's local neighborhood, allowing the network to learn representations that capture the features of the nodes and the topology of the graph.

The original GCN model has a limited receptive field it can effectively utilize, with potential challenges such as oversmoothing in deeper layers, vanishing gradients, and information bottlenecks. Several extensions, such as graph attention networks (GAT), graph isomorphism networks (GIN), and relational graph convolutional networks (R-GCN), address these limitations by introducing attention mechanisms or more expressive aggregation functions.

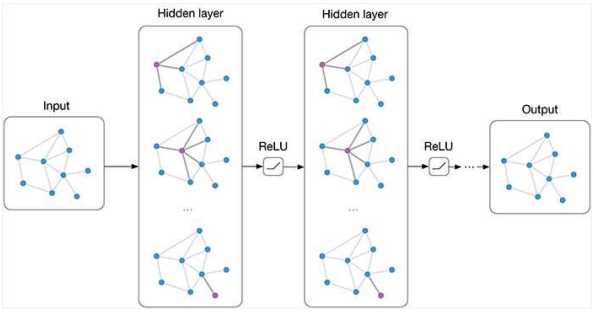

Figure 1: Overview of a graph neural network, with avtivation function ReLU and 2 hidden layers.

3. Methodology

3.1. Text Preprocessing

For basic natural language processing, the pre-trained `spaCy` model `en_core_web_sm` was employed. The extracted tokens were annotated with parts of speech and syntactic dependencies, also obtained using the `spaCy` model. During preprocessing, stop words (e.g., *"and"*, *"is"*, *"in"*) were removed, and the remaining tokens were vectorized [12].

The tokens were then transformed into a graph structure suitable for graph convolutional networks. Syntactic dependencies were used to define the edges between nodes (tokens), while parts of speech represented the node types (see Figure 2).

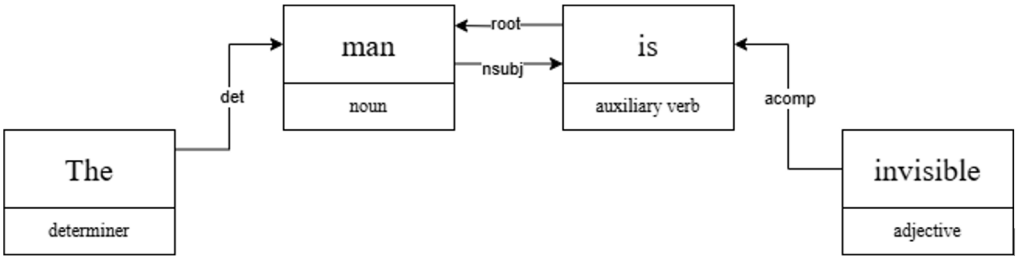

Figure 2: Graph representation of a sentence, det is determiner, nsubj refers to nominal subject, and acomp is adjectival complement (example *be, seem, appear, become*)

3.2. Graph Convolutional Network

A graph convolutional network (GCN) model was constructed with an input layer of 300 dimensions and two hidden layers consisting of 384 and 192 dimensions, respectively. These values were calculated on smaller sized datasets through an iterative algorithmic approach and then validated on larger datasets. The model also included an output layer of 300 dimensions, which aggregated the node vectors by averaging to produce the final output representation. The activation function used was Leaky ReLU, which, unlike the standard ReLU, allows small negative values to pass through.

Training was performed using 50 % of the Quora Question Pairs (QQP) dataset. Further 20 % was used as validating data and 30 % as testing data. The final training process ran on an NVIDIA GeForce RTX 3070 Laptop GPU, aided by multiprocessing [13] techniques for dataset loading and efficient conversion of individual sentences into graph structures.

Bayesian optimization was applied to tune the hyperparameters during training.

3.3. Relational Graph Convolutional Network

The relational graph convolutional network (R-GCN) differs from the standard GCN primarily in the volume of information utilized. In this model, all available graph information was leveraged. Edge types between nodes were defined based on syntactic dependencies. Each part of speech was assigned its own weight matrix for nodes with the corresponding POS tag. This weight matrix was used for information aggregation and influenced the output function (Equation 2):

$$\alpha = \text{SoftMAX}\Big(\frac{1}{\text{T}} \cdot \text{LeakyReLU}\Big(\big(H_j \cdot W \cdot N_{\text{dst}}\big) + \big(A_{\text{dst}} \cdot H_i\big),\ 0.1\Big)\Big) \tag{1}$$

$$G = \text{Mean}\left(\alpha \cdot H_j\right) \tag{2}$$

where α represents the attention coefficients, T refers to the temperature that controls how much the attention weights focus on the most relevant inputs, and G is the aggregated node representation obtained as the mean of neighbor features H_j weighted by α. H_j and H_i are the source and destination node feature vectors, W is a learnable transformation matrix, and N_{dst} and A_{dst} are relation-specific attention parameters applied to the transformed source and destination features, respectively. The LeakyReLU activation introduces a small slope for negative inputs, while the SoftMAX ensures that the attention scores for each destination node sum to one over all of its neighbors.

In the R-GCN, the message passing and aggregation functions were modified to incorporate separate weight matrices for different types of relations and nodes.

The dataset was split identically to the basic GCN setup. Bayesian optimization was also applied here, with an increased dropout rate observed due to the larger number of parameters.

The dimensionality of layers remained the same as in the original GCN model.

4. Graph Matching Network Architecture

Building upon the previously described relational graph convolutional network (R-GCN), we developed a graph matching network (GMN) designed specifically for semantic similarity tasks between sentence pairs. The GMN integrates the R-GCN as an internal component for encoding individual graph representations, while introducing additional layers to effectively compare and match these encoded graphs.

4.1. Model Architecture

The GMN consists of the following components:

- **R-GCN Encoder:** Each input sentence is first processed into a graph representation, which is passed through the pre-established R-GCN model. This R-GCN encodes node and edge features into a latent space, capturing complex relational and syntactic dependencies. The R-GCN layers maintain the same dimensionality as previously defined, with internal transformations of node features based on syntactic relations.
- **Matching Layers:** The encoded graph embeddings from the R-GCN are then fed into subsequent fully connected layers with dimensionalities 256, 128, and 1, respectively. These layers serve to learn a non-linear matching function that estimates the semantic similarity between the two input graphs. The final layer outputs a scalar similarity score.
- **Activation Functions and Regularization:** Leaky ReLU activations are applied after each hidden layer to maintain gradient flow and enable learning of complex decision boundaries. Dropout is used to reduce overfitting, particularly important due to the relatively small size of the final layers compared to the R-GCN encoder.

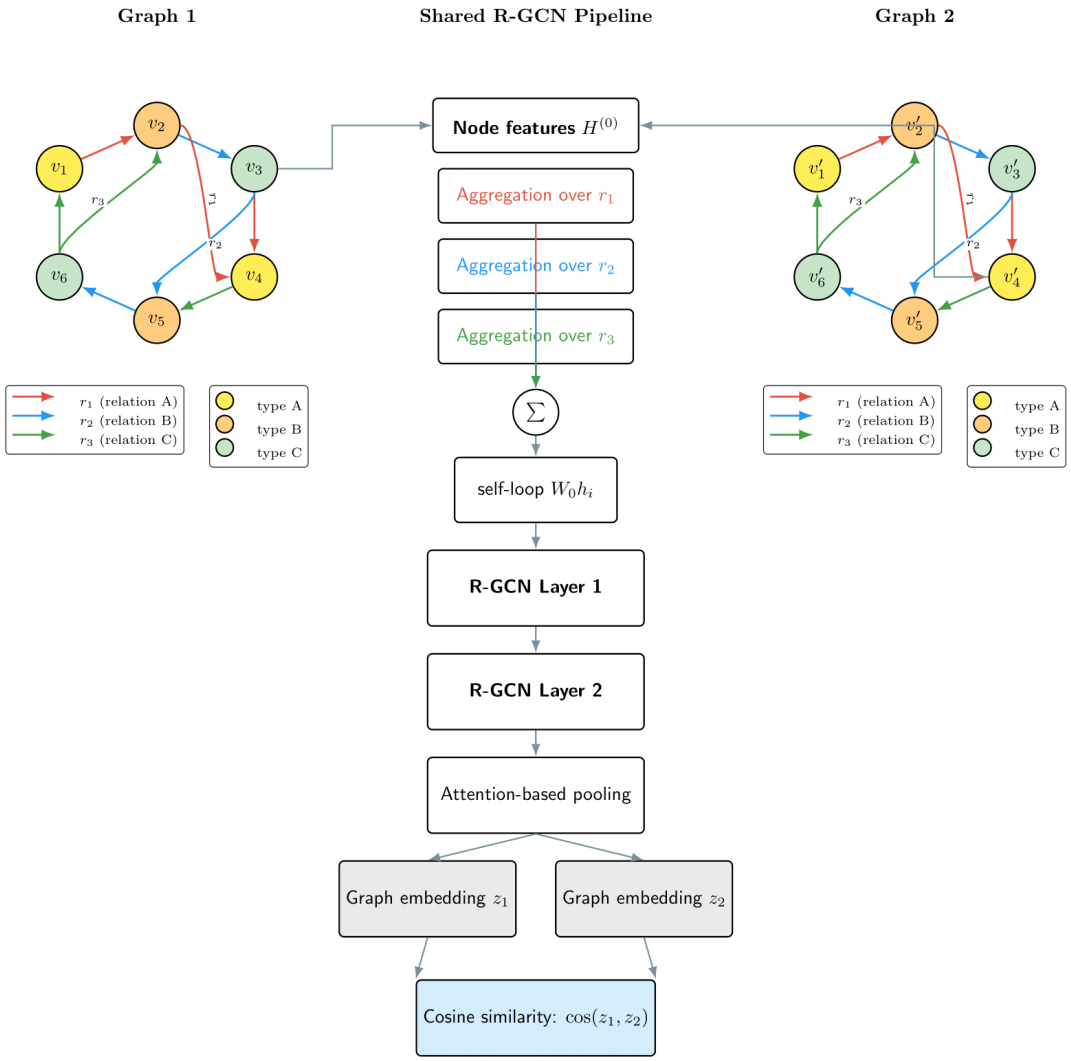

Figure 3: Pipeline of our R-GCN implementation with 2 hidden layers.

4.2. Training Procedure

The GMN was trained end-to-end on the Quora question pairs (QQP) dataset, using data proportion split 50 % training, 20 % validation, 30 % testing. We have employed Bayesian optimization to tune hyperparameters, such as learning rate, dropout rate, and layer sizes. The loss function is based on binary cross-entropy, reflecting the binary nature of semantic equivalence between sentence pairs.

4.3. Advantages of the Architecture

By leveraging the R-GCN for rich graph-structured encoding and adding dedicated matching layers with controlled dimensionality, the GMN effectively balances expressiveness and computational efficiency. The dimensionality reduction from the R-GCN output to the matching layers (256, 128, 1) enables the network to focus on the most salient features for the semantic similarity task, while reducing the risk of overfitting.

This hierarchical approach allows the GMN to capture both local relational patterns within sentences and global matching patterns between sentence pairs, resulting in improved performance compared to standalone R-GCN or traditional embedding comparison methods.

4.4. Employed Tools

For tokenization and extracting sentence structures, the `spaCy` [14] library was used. It enables precise identification of parts of speech and syntactic constituents, which is crucial for the approach used in this work.

The training of graph neural networks was conducted using `PyTorch` [15], which allowed for defining complex GNN architectures and optimizing them effectively.

To measure similarity, word vector representations from the Google News Word2Vec model were employed, enabling comparison of this work's model performance with existing embeddings such as BERT, without attributing any accuracy increase to a better-trained Word2Vec model.

Training was performed on the Quora Question Pairs (QQP) dataset, containing approximately 400,000 question pairs labeled for semantic similarity. The large size of this dataset helped to improve the model accuracy.

5. Results

5.1. Classification Performance

The R-GCN and GCN results were compared with the work [7], and a fine-tuned BERT model optimized for sentence embeddings—specifically, the `all-MiniLM-L12-v2` variant. All evaluations were conducted on an independent test dataset balanced equally across classes (semantically equivalent vs. nonequivalent sentence pairs). Performance metrics included accuracy, F1-score, and Area Under the Curve (AUC) [16], as shown in 1.

Table 1
Considered metric scores for each model in determining sentence similarity compared with results from [7]

Quality measure	Accuracy	F1 score	AUC
BERT	0.975	0.975	0.975
R-GCN	0.951	0.934	0.977
W2Vweighted	0.933	0.929	0.933
GCN	0.785	0.740	0.858
W2Vmean	0.806	0.837	0.806

Graph convolutional networks were compared against each other, the BERT model from the previous test, and the weighted Word2Vec model [7]. Again, all results were obtained from an independent test dataset. Accuracy, F1-score, and AUC metrics are reported in Figure 4.

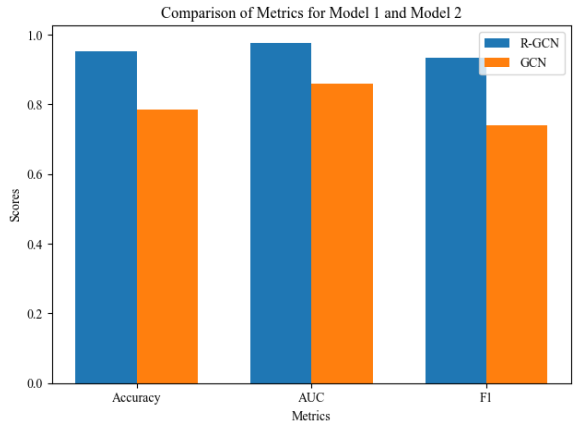

Figure 4: Difference of GCN and R-GCN shows the R-GCN outperforms the GNN in all considered metrics.

5.2. Statistical Analysis

BERT and R-GCN showed similar performance, while GCN was clearly worse than both solutions, and for sentences with different ground truth, it has brought a wider gap, so fewer sentences are marked as similar when they are actually different as seen in Figure 5. Testing was done on NVIDIA GeForce RTX 3070 Laptop GPU, where model R-GCN was 2.5 times faster than model BERT while using the same amount of memory.

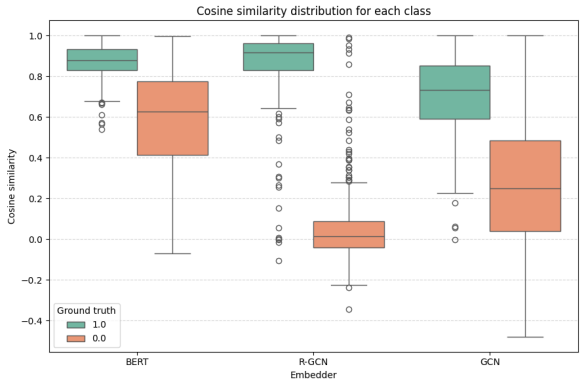

Figure 5: Difference of gap for tested sentences (testing dataset) with known ground truth in cosine similarity.

The significance of the differences between the embedding models was evaluated using the Friedman test [17]. The null hypothesis that all three embedding methods perform equally was strongly rejected with a p-value of $p = 1.39 \times 10^{-297}$. For post-hoc analysis, a Wilcoxon signed-rank test with a two-sided alternative [18] was applied pairwise between embedding methods. We adopted the arguments of [19] that, in machine learning, the Wilcoxon signed-rank test is more appropriate for this purpose than the post-hoc tests presented in [20] and [21]. The Holm method [22] was used to correct for multiple comparisons, yielding the following adjusted p-values [7]:

- BERT vs. W2V Mean: $p = 4.01 \times 10^{-156}$
- BERT vs. W2V Weighted: $p = 2.12 \times 10^{-16}$
- W2V Mean vs. W2V Weighted: $p = 1.46 \times 10^{-183}$

Similarly, to compare the two types of graph convolutional networks and model BERT, the Friedman test strongly rejected the null hypothesis of equal performance with a significance of $p = 4.71 \times 10^{-144}$. For post-hoc analysis, a Wilcoxon signed-rank test with a two-sided alternative [18] was applied pairwise between embedding methods. The Holm method [22] was used to correct for multiple comparisons, yielding the following adjusted p-values:

- R-GCN vs. GCN: $p = 9.43 \times 10^{-10}$
- BERT vs. RGCN: $p = 2.12 \times 10^{-45}$
- GCN vs. BERT: $p = 8.59 \times 10^{-38}$

6. Conclusion

The objective of this work was to develop models based on GNN capable of recognizing semantic similarity between sentences, while maintaining a level of quality comparable to the widely used and architecturally complex BERT model.

The two models of graph neural networks were developed: a basic Graph Convolutional Network (GCN) and a Relational Graph Convolutional Network (R-GCN). Their respective parameters were compared against the BERT model. The basic GCN was unable to learn sufficient information to

outperform the simple Word2Vec baseline, demonstrating its limitations. In contrast, the R-GCN effectively utilized all available syntactic information and was competitive with BERT in performance.

The results obtained suggest the feasibility of constructing models based on vector processing of entire paragraphs or even entire documents. Furthermore, these findings could be applied to enhance the quality of generative transformers that incorporate graph convolutional functions in one of their processing stages.

Overall, this research demonstrates the effectiveness of graph neural networks for text-understanding tasks.

Acknowledgments

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS23/205/OHK3/3T/18 and by the German Research Foundation (DFG) funded project 467401796.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach (4th Edition), Pearson, 2020. URL: http://aima.cs.berkeley.edu/.

[2] B. Srinivasa-Desikan, Natural Language Processing and Computational Linguistics: A Practical Guide to Text Analysis with Python, Gensim, spaCy, and Keras, Expert Insight, Packt, Birmingham, 2018.

[3] Z. Liu, J. Zhou, Introduction to Graph Neural Networks, Springer, Cham, 2022.

[4] S. Bird, E. Klein, E. Loper, Natural Language Processing with Python, O'Reilly, Beijing, 2009.

[5] Y. Kim, D. Oh, H. H. Huang, Syncse: syntax graph-based contrastive learning of sentence embeddings, Expert Systems with Applications 287 (2025) 128047. URL: https://www.sciencedirect.com/science/article/pii/S0957417425016689. doi:https://doi.org/10.1016/j.eswa.2025.128047.

[6] Y. Yang, X. Cui, Bert-enhanced text graph neural network for classification, Entropy 23 (2021). URL: https://www.mdpi.com/1099-4300/23/11/1536. doi:10.3390/e23111536.

[7] T. Ladin, L. Korel, M. Holena, Textual embeddings with word-type-weighted word2vec, in: ITAT, 2024, pp. 37–42. URL: https://ceur-ws.org/Vol-3792/paper4.pdf.

[8] I. M. Gel'fand, M. E. Saul, Trigonometry, Birkhäuser, Boston, 2001.

[9] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, OpenReview (2013) 1–12.

[10] R. Garnett, Bayesian Optimization, Cambridge University Press, Cambridge, United Kingdom, 2023.

[11] R. Lin, et al., On the over-memorization during natural, robust and catastrophic overfitting, arXiv preprint arXiv:2310.08847 (2023).

[12] D. A. Turkington, Generalized vectorization, cross-products, and matrix calculus, Cambridge University Press, New York, USA, 2013.

[13] M. Gorelick, I. Ozsvald, High Performance Python: Practical Performant Programming for Humans, Second Edition, O'Reilly, Beijing, 2020.

[14] Y. Vasiliev, Natural Language Processing with Python and spaCy: A Practical Introduction, No Starch Press, San Francisco, 2020.

[15] S. Raschka, Y. Liu, V. Mirjalili, Machine Learning with PyTorch and scikit-learn: Develop Machine Learning and Deep Learning Models with Python, Packt, Birmingham, 2022.

[16] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, New York, 2006.

[17] M. Hollander, D. A. Wolfe, E. Chicken, Nonparametric Statistical Methods, 3 ed., Wiley, Hoboken, New Jersey, 2013.
[18] E. L. Lehmann, H. J. M. D'Abrera, Nonparametrics: Statistical Methods Based on Ranks, revised 1st ed., Springer, New York, 2006.
[19] A. Benavoli, G. Corani, F. Mangili, Should we really use post-hoc tests based on mean-ranks?, Journal of Machine Learning Research 17 (2016) 1–10. URL: http://jmlr.org/papers/v17/benavoli16a.html.
[20] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.
[21] S. Garcia, F. Herrera, An extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all pairwise comparisons, Journal of Machine Learning Research 9 (2008) 2677–2694.
[22] P. H. Westfall, R. D. Tobias, R. D. Wolfinger, Multiple Comparisons and Multiple Tests Using the SAS System, SAS Institute, Cary, North Carolina, 2011.