An Efficient Implementation of Hubness-Aware Weighting Using Cython

Krisztian Buza
buza@biointelligence.hu
BioIntelligence Group, Department of Mathematics-Informatics
Sapientia Hungarian University of Transylvania
Targu Mures, Romania

ABSTRACT

Hubness-aware classifiers are recent variants of k-nearest neighbor. When training hubness-aware classifiers, the computationally most expensive step is the calculation of hubness scores. We show that this step can be sped up by an order of magnitude or even more if it is implemented in Cython instead of Python while the accuracy is the same in both cases.

KEYWORDS

nearest neighbor, hubs, cython

1 INTRODUCTION

Nearest neighbor classifiers are simple, intuitive and popular, there are theoretical results about their accuracy and error bounds [6]. However, nearest neighbors are affected by *bad hubs*. An instance is called a bad hub, if it appears surprisingly frequently as nearest neighbor of other instances, but its class label is different from the labels of those other instances. Bad hubs were shown to be responsible for a surprisingly large fraction of the total classification error [10].

In order to reduce the detrimental effect of bad hubs, hubness-aware classifiers have been introduced, such as Hubness-Weighted k-Nearest Neighbor (HWKNN) [9], Naive Hubness Bayesian Nearest Neighbor (NHBNN) [16] and Hubness-based Fuzzy Nearest Neighbor (HFNN) [14]. Hubness has also been studied in context of collaborative filtering [8], regression [3], clustering [15], instance selection and feature selection [13]. Recently, hubness-aware ensembles have been proposed [17] and used for the classification of breast cancer subtypes [12].

Other prominent applications of hubness-aware methods include music recommendation [7], time series classification [11], drug-target prediction [4] and classification of gene expression data [2]. Last, but not least, we mention that even neural networks may benefit from hubness-aware weighting [5].

Hubness-aware classifiers may be implemented in various programming languages, one of the most prominent implementation is probably the Java-based HubMiner[1] library.

[1]https://github.com/datapoet/hubminer

Information Society 2022, 10–14 October 2022, Ljubljana, Slovenia

In case of the aforementioned hubness-aware classifiers, the computationally most expensive step of the training is to determine the *hubness scores* of training instances, i.e., how frequently they appear as (bad) nearest neighbors of other instances. In this paper, we address this issue by a Cython-based implementation. Cython [1] aims to combine the advantages of Python (rapid prototyping and clarity thanks to concise code) with the efficiency of C. In particular, we implement the computation of hubness scores in Cython. Compared with a standard implementation in Python, we observed up to 25 times speedup on the Spambase dataset[2] from the UCI repository (and the speedup is likely to be even more in case of larger datasets).

2 BACKGROUND: HUBNESS-AWARE WEIGHTING

We say that an instance x is a *bad neighbor* of another instance x' if (i) x is one of the k-nearest neighbors of x' and (ii) their class labels are different. In case of hubness-aware weighting [9], first we determine how frequently each instance x appears as bad neighbor of other instances. This is denoted as $BN_k(x)$. Subsequently, the normalized bad hubness score $h_b(x)$ of each instance x is calculated as follows:

$$h_b(x) = \frac{BN_k(x) - \mu(BN_k)}{\sigma(BN_k)} \quad (1)$$

where $\mu(BN_k)$ and $\sigma(BN_k)$ denote the mean and standard deviation of the $BN_k(x)$ values over all instances of the training data. HWKNN performs weighted k-nearest neighbor classification, the weight of each training instance is $w(x) = e^{-h_b(x)}$. For a detailed illustration of HWKNN we refer to [13].

3 CYTHON-BASED IMPLEMENTATION OF HUBNESS CALCULATIONS

Python code is usually run by an interpreter which makes the execution relatively slow. Much of the inefficiency originates from dynamic typing: for example, the actual semantic of the '+' symbol depends on the types of the operands. It may stand for addition of numbers, concatenation of strings or lists, element-wise addition of arrays, etc. Which of the operations to perform, will be determined by the interpreter at execution time.

The core idea of Cython[3] is to annotate variables according to their types and to compile the resulting code into C which will further be compiled into binary code for efficient execution. In case of computationally expensive functions, this may results in

[2]https://archive.ics.uci.edu/ml/datasets/spambase
[3]https://cython.org/

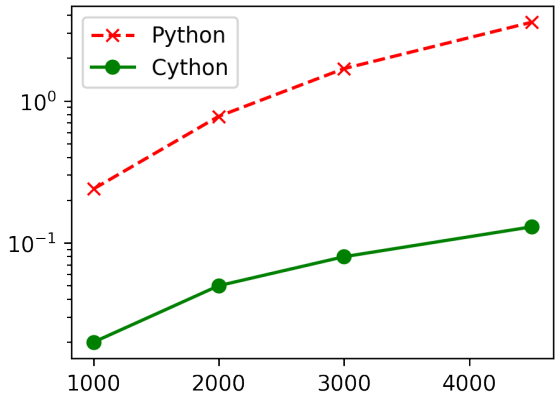

Figure 1: Runtime (in second, vertical axis) of hubness score calculation in case of Python-based (dashed line with 'x') and Cython-based (solid line with bullets) implementations for various number of instances (horizontal axis).

several orders of magnitude speedup. At the same time, functions implemented in Cython can be called from Python code just like Python functions.

We implemented the calculation of hubness scores both in Python and Cython, and made the code available in our github repository:

https://github.com/kr7/cython .

We evaluated both implementations on the Spambase dataset from the UCI repository. The dataset contains 4601 instances and 57 features (without the class label). Each instance corresponds to an e-mail. For each e-mail, the same features were extracted. The associated classification task is to decide whether the e-mail is spam or not.

We used 100 instances as test data and 4500 instances as training data. We run the experiments in Google Colab.[4] We used $k = 10$ nearest neighbors both for the calculation of hubness scores and the final classification. According to our observations, the Cython-based calculation of hubness scores was more than 20 times faster than the standard implementation in Python. Both versions produced the exactly same $BN_k(x)$ scores. As the weight of an instance x only depends on its $BN_k(x)$ score, both versions produce the same predictions. Therefore the accuracy (0.94) is equal in both cases.

We repeated the experiments with using only 1000, 2000 and 3000 instances as training data. As Fig. 1 shows, the Cython-based implementation was consistently faster than the implementation in Python. Note that logarithmic scale is used on the vertical axis. The difference showed an increasing trend when more training data was used: whereas in case of 1000 training instances, the Cython-based implementation was only about 12 times faster than the Python-based implementation, in case of 4500 training instances, the speedup factor was approximately 25. This may be attributed to the non-linear complexity of hubness score calculations. Assuming a naive implementation, determination of the nearest neighbors of an instance is linear in the size of the training data. However, in order to calculate the hubness scores, the nearest neighbors of *all* the training instances have to be determined. Thus the resulting overall complexity is quadratic.

We note that, both in case of Cython and Python, indexing techniques may be used to speed up the determination of the nearest neighbors. However, we omitted indexing in our implementation for simplicity.

[4]https://colab.research.google.com

4 DISCUSSION

In order to calculate distances effectively, we used pairwise distances from scikit-learn in our experiment. However, in case of *large* datasets, it may be necessary to calculate distances on the fly, as the distance matrix may be too large to be stored in RAM. In such cases, it may be worth considering to implement the distance calculations in Cython as well. In our previous works, we observed that the calculation of dynamic time warping distance was several orders of magnitudes faster when we implemented it in Cython instead of Python.

In case of *very large* datasets, straight forward calculation of hubness scores may be infeasible due to its quadratic complexity even if the calculations are implemented in Cython. In such cases, the aforementioned indexing techniques and/or calculation of approximate hubness scores (e.g. using a random subset of the data) may be necessary.

As future work, we plan an exhaustive evaluation of both implementations with respect to various datasets with different sizes and number of features.

ACKNOWLEDGEMENT

The author thanks to the Reviewers for their insightful comments and suggestions.

REFERENCES

[1] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. 2010. Cython: The best of both worlds. *Computing in Science & Engineering* 13, 2 (2010), 31–39.

[2] Krisztian Buza. 2016. Classification of gene expression data: A hubness-aware semi-supervised approach. *Computer methods and programs in biomedicine* 127 (2016), 105–113.

[3] Krisztian Buza, Alexandros Nanopoulos, and Gábor Nagy. 2015. Nearest neighbor regression in the presence of bad hubs. *Knowledge-Based Systems* 86 (2015), 250–260.

[4] Krisztian Buza and Ladislav Peška. 2017. Drug–target interaction prediction with Bipartite Local Models and hubness-aware regression. *Neurocomputing* 260 (2017), 284–293.

[5] Krisztian Buza and Noémi Ágnes Varga. 2016. Parkinsonet: estimation of updrs score using hubness-aware feedforward neural networks. *Applied Artificial Intelligence* 30, 6 (2016), 541–555.

[6] Luc Devroye, László Györfi, and Gábor Lugosi. 2013. *A probabilistic theory of pattern recognition*. Vol. 31. Springer Science & Business Media.

[7] Arthur Flexer, Monika Dörfler, Jan Schlüter, and Thomas Grill. 2018. Hubness as a case of technical algorithmic bias in music recommendation. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1062–1069.

[8] Peter Knees, Dominik Schnitzer, and Arthur Flexer. 2014. Improving neighborhood-based collaborative filtering by reducing hubness. In *Proceedings of International Conference on Multimedia Retrieval*. 161–168.

[9] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. 2009. Nearest neighbors in high-dimensional data: The emergence and influence of hubs. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 865–872.

[10] Milos Radovanovic, Alexandros Nanopoulos, and Mirjana Ivanovic. 2010. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research* 11, sept (2010), 2487–2531.

[11] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. 2010. Time-series classification in many intrinsic dimensions. In *Proceedings of the 2010 SIAM International Conference on Data Mining*. SIAM, 677–688.

[12] S Raja Sree and A Kunthavai. 2022. Hubness weighted svm ensemble for prediction of breast cancer subtypes. *Technology and Health Care* 30, 3 (2022), 565–578.

[13] Nenad Tomašev, Krisztian Buza, Kristóf Marussy, and Piroska B Kis. 2015. Hubness-aware classification, instance selection and feature construction: Survey and extensions to time-series. In *Feature selection for data and pattern recognition*. Springer, 231–262.

[14] Nenad Tomašev, Miloš Radovanovic, Dunja Mladenic, and Mirjana Ivanovic. 2011. A probabilistic approach to nearest-neighbor classification: Naive hubness bayesian knn. In *Proc. 20th ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 2173–2176.

[15] Nenad Tomasev, Milos Radovanovic, Dunja Mladenic, and Mirjana Ivanovic. 2013. The role of hubness in clustering high-dimensional data. *IEEE transactions on knowledge and data engineering* 26, 3 (2013), 739–751.

[16] Nenad Tomašev, Miloš Radovanović, Dunja Mladenić, and Mirjana Ivanović. 2014. Hubness-based fuzzy measures for high-dimensional k-nearest neighbor classification. *International Journal of Machine Learning and Cybernetics* 5, 3 (2014), 445–458.

[17] Qin Wu, Yaping Lin, Tuanfei Zhu, and Yue Zhang. 2020. HIBoost: A hubness-aware ensemble learning algorithm for high-dimensional imbalanced data classification. *Journal of Intelligent & Fuzzy Systems* 39, 1 (2020), 133–144.