NN-Descent on High-Dimensional Data

Brankica Bratić
University of Novi Sad
Faculty of Sciences
Novi Sad, Serbia
brankica.bratic@dmi.uns.ac.rs

Michael E. Houle
National Institute of Informatics
Tokyo, Japan
meh@nii.ac.jp

Vladimir Kurbalija
University of Novi Sad
Faculty of Sciences
Novi Sad, Serbia
kurba@dmi.uns.ac.rs

Vincent Oria
New Jersey Inst. of Technology
Dept. of Computer Science
Newark, New Jersey, USA
vincent.oria@njit.edu

Miloš Radovanović
University of Novi Sad
Faculty of Sciences
Novi Sad, Serbia
radacha@dmi.uns.ac.rs

ABSTRACT

K-nearest neighbor graphs (K-NNGs) are used in many data-mining and machine-learning algorithms. Naive construction of K-NNGs has a complexity of $O(n^2)$, which could be a problem for large-scale data sets. In order to achieve higher efficiency, many exact and approximate algorithms have been developed, including the NN-Descent algorithm of Dong, Charikar and Li. Empirical evidence suggests that the practical complexity of this algorithm is in $\tilde{O}(n^{1.14})$, which is a significant improvement over brute force construction. However, NN-Descent has a major drawback — it produces good results only on data of low intrinsic dimensionality. This paper presents an experimental analysis of this behavior, and investigates possible solutions. We link the quality of performance of NN-Descent with the phenomenon of *hubness*, defined as the tendency of intrinsically high-dimensional data to contain *hubs* — points with high in-degrees in the K-NNG. We propose two approaches to alleviate the observed negative influence of hubs on NN-Descent performance.

KEYWORDS

NN-Descent, k-nearest neighbor graph, hubness

ACM Reference Format:
Brankica Bratić, Michael E. Houle, Vladimir Kurbalija, Vincent Oria, and Miloš Radovanović. 2018. NN-Descent on High-Dimensional Data. In *Proceedings of 8th International Conference on Web Intelligence, Mining and Semantics (WIMS'18).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A K-nearest neighbor graph (K-NNG) is a directed graph whose vertices represent elements upon which some distance function is defined. Vertex v is unidirectionally connected to vertex u only if vertex u is one of the K elements that are nearest to point v with respect to the defined distance function. K-NNGs are used in many machine-learning and data-mining algorithms [8], including classification [7], similarity search [11], clustering and outlier detection problems [2, 3, 12], manifold learning [1, 20, 22]. They are also used in other areas, such as robot motion planning [5] and computer graphics [21].

Naive brute-force computation of a K-NNG entails $n \cdot (n-1)$ distance computations, which leads to quadratic time complexity. Numerous methods for K-NNG construction have been developed in order to decrease the computational cost, but many of them introduce certain restrictions and therefore do not apply to the general case. For example, many efficient methods are developed for Euclidean or other L_p distance metrics [4, 6], whereas others can be applied to more general metric spaces [14]. In order to bypass all those restrictions while still preserving efficiency, Dong, Charikar and Li created the NN-Descent [10] algorithm, that efficiently produces highly accurate K-NNG approximations independently of the underlying distance function. As reported by the authors, empirical complexity of NN-Descent is in $\tilde{O}(n^{1.14})$. Although the results produced by this algorithm are mostly excellent, there is still one major drawback — NN-Descent often produces inaccurate results when it is used on data of high intrinsic dimensionality.

Park et al. [15] developed a modification of NN-Descent based on a greedy filtering method, with which they obtained improved results for high-dimensional data together with the cosine similarity measure. Houle et al. [13] introduced NNF-Descent (Nearest Neighbor Feature Descent), an NN-Descent-based feature sparsification method optimized for image databases. It uses the Laplacian score in order to identify noisy values, which are then replaced with zeros. Debatty et al. [9] presented a method for constructing a K-NNG for large text data sets. This method applies NN-Descent only to smaller buckets of data, leading to a lower execution time.

Although several improvements have been proposed for NN-Descent, so far none of them have given an explanation for the variability of its performance, nor have they given a universal solution to the problem. In this paper we will provide an explanation for this variation in terms of *hubness*, a measure of the variation in nearest neighbor relationships [16]. We also propose two approaches that improve the quality of NN-Descent approximations for high-dimensional datasets.

WIMS'18, June 2018, Novi Sad, Serbia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Section 2 will describe the necessary background by providing an overview of the NN-Descent algorithm and the phenomenon of hubness. In Section 3 we show how a minor modification of the NN-Descent algorithm can produce estimates of hubness. Experimental results presented in Section 4 demonstrate the empirical effect of hubness on NN-Descent. Section 5 proposes methods that to some extent overcome the problems of NN-Descent on high-dimensional data. Finally, Section 6 gives an overview of the conclusions and results, and indicates directions for future work.

2 BACKGROUND

In this section we will review the NN-descent algorithm (Section 2.1) and the hubness phenomenon (Section 2.2), which, as we will demonstrate later, has a significant impact on NN-Descent.

2.1 NN-Descent

The main purpose of the NN-Descent algorithm is to create a good approximation of the true K-NNG, and to create it as fast as possible. The basic assumption made by NN-Descent can be summarized as "my neighbor's neighbor is also likely to be my neighbor." The algorithm starts by creating a random K-NNG, and then iteratively improves the neighbor lists by using the current neighborhood relationships to suggest new neighbor candidates. During the execution of NN-Descent, for a data point u, we will refer to the current list of its K nearest neighbors as the *NN list* of u. A reverse neighbor of u is a data point which has u in its own NN list; we will refer to the current list of reverse neighbors of u as its *RNN list*.

NN-Descent is organized into stages, during each of which a single improvement iteration is applied to each of the data points. The algorithm iteration for u examines its NN and RNN lists to see whether any points in these lists should be added to the NN lists of any of the others. If this is the case, the algorithm makes the necessary changes, and increases the number of total updates. After every stage has completed, the algorithm tests a termination condition based on the number of updates that were applied during the stage. If the number of updates is smaller than a supplied threshold, the algorithm terminates; otherwise, it continues with the next stage. An outline of NN-Descent is shown as Algorithm 1.

The speed of NN-Descent is strongly influenced by the K value. As K becomes larger, the algorithm becomes slower. To be more precise, the time complexity has a quadratic dependence on K, since during each iteration, points appearing in NN or RNN lists of the same point are evaluated as candidates for each others NN lists. As a way to reduce the number of combinations of points considered, the authors introduced sampling [10]. Sampling reduces the number of evaluations by taking a random selection of NN and RNN lists, and then operates only on the points from those lists. Sampling is controlled by a parameter ρ that takes a value from 0 to 1. The algorithm takes $\rho \cdot K$ points from both NN and RNN lists.

The second drawback of NN-Descent is that quality of approximation it produces is highly influenced by the intrinsic dimensionality of the data set, in that the quality of the approximation decreases as the intrinsic dimensionality increases. The reason for this has not been adequately explained. Although low-dimensional embeddings have often been used as a general method of reducing the complexity of data indexing, no solutions have yet been proposed that allow the NN-Descent strategy to work more effectively with high-dimensional data. In this paper we will explain the influence of high dimensionality on NN-Descent, and will also propose two approaches that are designed to overcome this challenge to some extent.

Algorithm 1: Outline of NN-Descent algorithm.

input : dataset D, distance function *dist*, neighborhood size K
output: K-NNG G

```
foreach data point u ∈ D do
    Initialize G by randomly generating a tentative K-NN list
      for u with an assigned distance of +∞;
end
repeat
    foreach data point u ∈ D do
        Check different pairs of u's neighbors (v, w) in u's
          K-NN and R-NN (reverse nearest neighbor) lists, and
          compute dist(v, w);
        Use ⟨v, dist(v, w)⟩ to update w's K-NN list, and use
          ⟨w, dist(v, w)⟩ to update v's K-NN list;
    end
until G converges;
return G.
```

2.2 Hubness

Hubness is an aspect of the curse of dimensionality pertaining to nearest neighbors which has come to the attention of the research community only relatively recently [16]. Let $D \subset \mathbb{R}^d$ be a set of data points and let the *k-occurrences* of point $x \in D$ be the number of times x occurs in k-nearest-neighbor lists of other points from D. $N_k(x)$ is then the number of *k-occurrences* of point $x \in D$ — that is, the in-degree of node x in the K-NNG. As the dimensionality of the data increases, the distribution of $N_k(x)$ becomes considerably skewed. As a consequence, some data points, which we will refer to as *hubs*, are included in many more k-nearest-neighbor lists than other points.

In the remainder of the discussion we will refer to the number of k-occurrences of point $x \in D$ as its *hubness value*. If a data set contains many points that are hubs — that is, if the distribution of hubness values is highly skewed — it can be said that the hubness phenomenon is present therein. It has been shown that hubness, as a phenomenon, appears in high-dimensional data as an inherent property of intrinsically high dimensionality, and is not an artifact of finite samples nor a peculiarity of specific data sets [16]. It was shown that hubness influences various data-mining and machine-learning algorithms [16–19, 23], often in a negative way. Practical computation of (exact) hubness values entails the creation of the K-NNG, from which hubness values can be easily obtained by extracting the in-degrees of nodes.

3 HUBNESS ESTIMATION USING NN-DESCENT

Approximate hubness values for each data point could be very easily calculated during NN-Descent, with minimal impact on algorithm performance. At the very beginning, we initialize the hubness values of each data set point to zero. Then, during algorithm execution, we increase the hubness value of a given point by one if that point is added to the NN list of some other point, and analogously, we decrease the hubness value by one if the point is removed from some NN list.

Some of the results that we obtained after this small upgrade of NN-Descent algorithm are illustrated in Figure 1, which shows the correlation between estimated and exact hubness values of data points. For the purposes of the experimentation in this paper, we created several synthetic data sets drawn from a Gaussian distribution with standard deviation 1 and mean $(0, 0, \ldots, 0)$. The data sets each have $n = 10000$ points, but differ in their dimensionality d. During the analysis we also varied the size of the nearest neighbor lists, represented by the parameter K. We considered the values 10, 20 and 50 for K, and the values 2, 10, 20 and 100 for parameter d. As a distance measure we used the Euclidean distance. In Figure 1 we show results only for the data set of highest dimensionality ($d = 100$), since the hubness phenomenon is known to strengthen as the dimensionality of the data increases. As can be seen, the correlation between real and estimated hubness values is very high. A shortcoming of NN-Descent is that estimated hubness values are lower then they should be for points with low true hubness values, and greater then they should be for points with high true hubness. The precision improves as K increases, but even for small K values, the algorithm produces strong correlations.

4 INFLUENCE OF HUBNESS ON NN-DESCENT

NN-Descent is known to produce large amount of incorrect K nearest neighbors when applied upon high-dimensional data. Since hubness is a phenomenon that appears in high-dimensional data, and at the same time was shown to influence many other data-mining algorithms, in this section we investigate whether it also influences the performance of NN-Descent.

For a given data set point, we will define the hit count to be the number of K-nearest neighbors produced by NN-Descent that are at the same time one of the point's K-nearest neighbors in the exact K-NNG. In other words, the hit count tells us how many correct selections NN-Descent has made for the neighborhood list of a given data point. The first part of our analysis will be to examine the correlation between hit count values and hubness values of data points. In order to do that, we first calculated the exact K-NNGs, as well as hubness values for each data set point. We then ran NN-Descent without sampling and early termination, and calculated hit count values based on the previously calculated K-NNG and NN-Descent graph approximation. We did this for all generated data sets.

Some of the results can be seen in Figure 2. Let us now say that for the points of fixed hubness value, hit count values are distributed between $\mathrm{hc}_{\mathrm{min}}$ and $\mathrm{hc}_{\mathrm{max}}$. Results are showing us that both values are directly proportional to the hubness value, with this behavior being much more evident in the case of the $\mathrm{hc}_{\mathrm{min}}$ value.

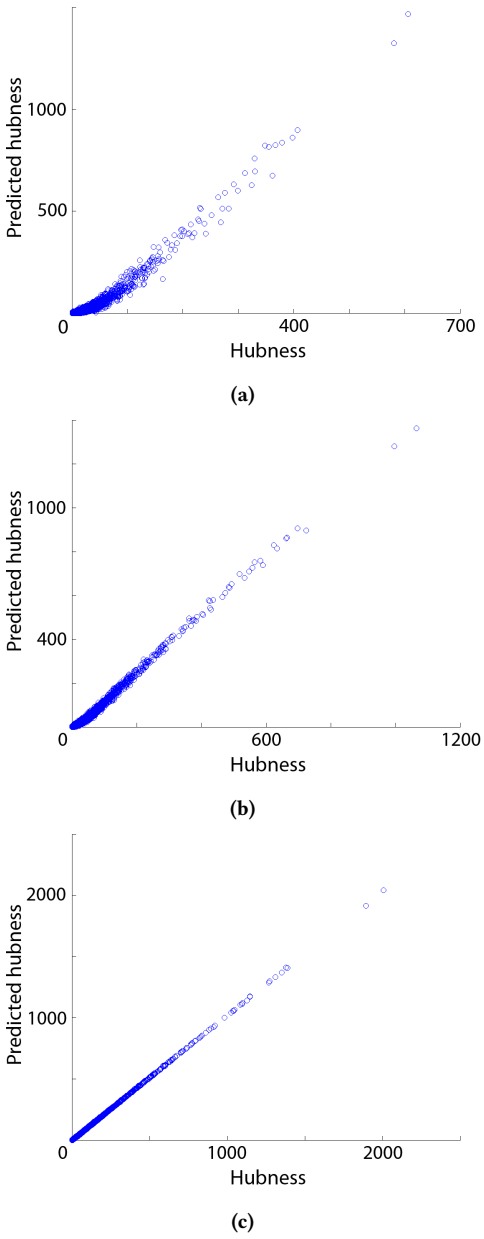

Figure 1: Correlation between estimated and exact hubness values of data set points. (a) $K = 10, d = 100$**; (b)** $K = 20, d = 100$**; (c)** $K = 50, d = 100$**. Here,** K **is the size of the neighbor lists, and** d **is the dimensionality of the data set.**

In the case of $\mathrm{hc}_{\mathrm{max}}$, the phenomenon is less evident (Figure 2d). As a consequence, points with extremely high hubness values have $\mathrm{hc}_{\mathrm{min}} \simeq \mathrm{hc}_{\mathrm{max}} \simeq K$, which means that their hit counts usually attain the greatest possible value. For the lowest hubness values, it holds that $\mathrm{hc}_{\mathrm{min}}$ is near zero, while the $\mathrm{hc}_{\mathrm{max}}$ depends on K and d; however, for all tested data sets, this value is K or very nearly K. In other words, some points with extremely low hubness values have a high hit count, and some have hit counts that are very low, which

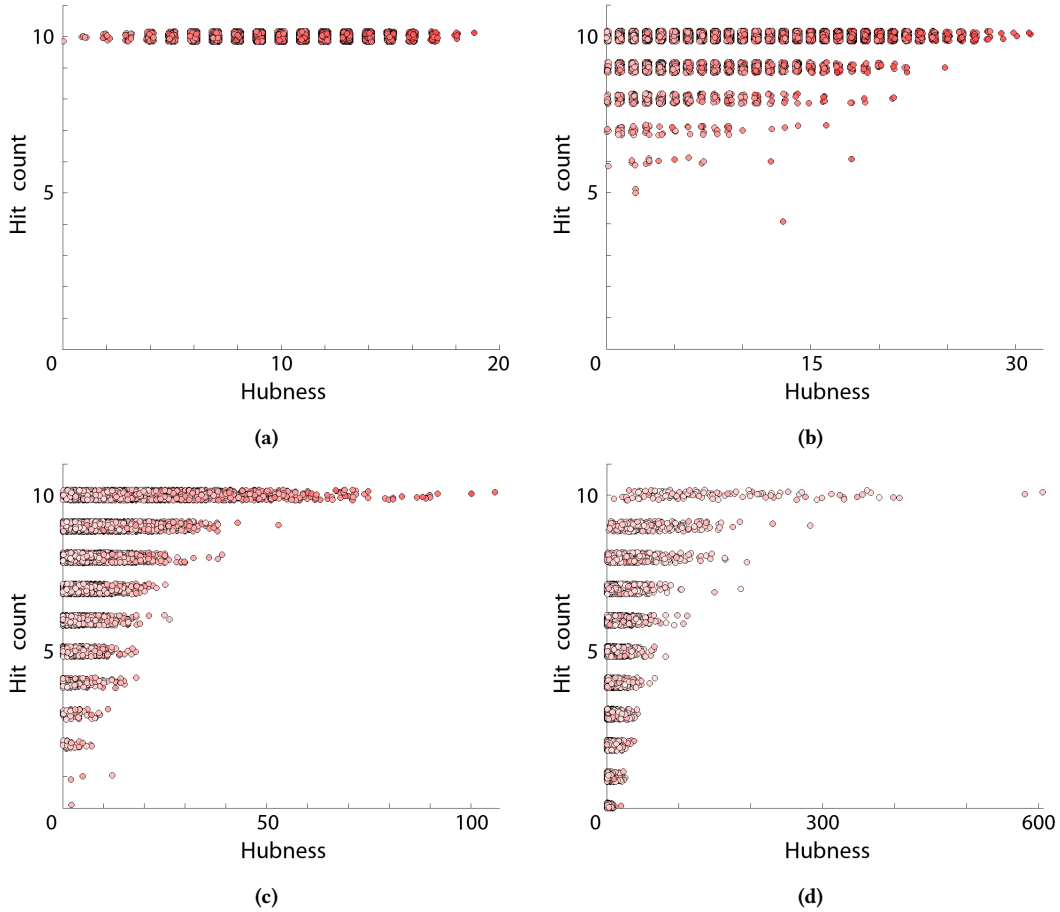

Figure 2: Hubness and hit count values of data set points. (a) $K = 10, d = 2$**; (b)** $K = 10, d = 10$**; (c)** $K = 10, d = 20$**; (d)** $K = 10, d = 100$**. Here,** K **is the size of the neighbor lists, and** d **is the dimensionality of the data set.**

means that the probability of error when determining the K nearest neighbors of point x is inversely proportional to the hubness of x. The described phenomenon is more evident in data sets with lower K values and greater dimensionality d. If K is sufficiently large, and d is sufficiently small, the phenomenon is less evident, as the hubness values of the points are more uniform. In these cases, NN-Descent produces a very good approximation of K-NNG. In further research we will concentrate on settings for which the performance of NN-Descent is poor, as we saw for the case where $d = 100$.

Now that the influence of hubness on NN-Descent has been established, we examine the cause of this phenomenon. As was already described in Section 2.1, the NN-Descent algorithm improves the K-NN graph approximation in each iteration. If the hubness of the data set is high, then in the first few stages hubs will be placed among the K nearest neighbors of a large number of data points. This implies that the NN lists of a majority of points will contain hubs. Points with lower hubness values will quickly be expelled from NN lists of other points. This expulsion of non-hubs from NN lists implies that the neighborhoods of those points, according to the nature of the algorithm, will not be updated as often — in order for the NN list of a given point to be updated, that point must be present in the NN or RNN lists of other data points. If a point is rapidly expelled from NN lists, then it will be updated only from RNN lists of the points that are very unlikely to be their true neighbors, since the initial approximation of the K-NN graph is essentially random. For some points of low hubness value, this produces relatively poor results.

Another issue to consider is that some points with low hubness values may still have high hit count values, while other low-hubness points may have very low hit counts. In order to quantify the extent to which the variation in hit count values depends on hubness, we calculated hit count values and their standard deviations for each data point, over 100 runs of NN-Descent, each time generating the random graph with a different random seed. The results are shown in Figure 3, where the intensity of the red color represents the mean of hit count values among different runs — points with greater intensity have greater average hit count values. As can be seen, points with low hubness values have hit counts with a relatively high standard deviation, while points with high hubness values have hit count standard deviations that tend toward zero. This implies that hit count values vary significantly across different runs, while the only difference between runs is the initial random

graph. All of this leads us to the conclusion that the main factor that affects the hit count of points with low hubness is the initial random K-NNG: if the point is in a high-quality neighborhood in the initial random graph, then NN-Descent is more likely to produce a high hit count. Otherwise, the point will be quickly expelled from NN lists, and will stay trapped in the RNN lists of points that are not its true neighbors, leading to a relatively low probability that the point's NN list will be updated with true neighbors.

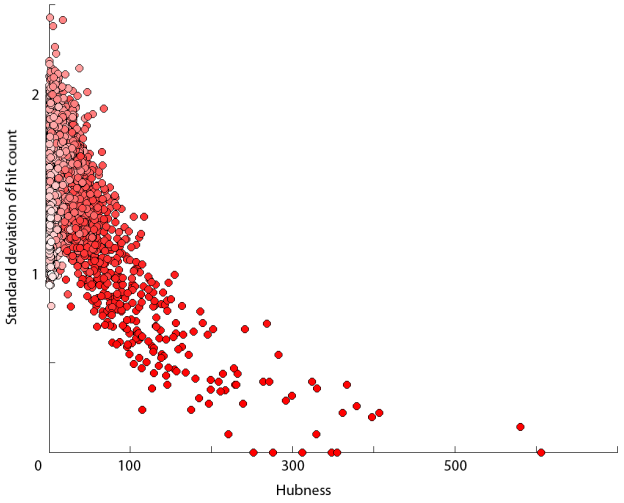

Figure 3: Results with $K = 10$ and $d = 100$; the y-axis shows standard deviations of hit count values for 100 runs of NN-Descent, while the x-axis shows hubness values of the corresponding data points.

5 PROPOSED METHODS FOR IMPROVING NN-DESCENT ON HIGH-DIMENSIONAL DATA

In order to overcome the problem described in Section 4, we implemented two different approaches. The first one is based on the idea that NN lists should not be updated strictly according to the NN and RNN lists of other points. The goal is to integrate the information about hubness values into the choice of points to compare with the current approximation of the NN list. A high hubness value indicates that a certain point already has a reasonably stable NN list, and vice versa — a low hubness value suggests that the point has a greater probability of being assigned an incorrect NN list. The second approach is based on the observation that for greater K values the algorithm tends to be more accurate. If we run the NN-Descent algorithm with a greater K value, and then reduce the resulting K-NNG to the K we actually need, the precision of final graph is expected to be better.

In order to be able to compare the results of the original NN-Descent with the results of the proposed improvements, we show the performance of the original algorithm in Figure 4, while the following two subsections will present results of the two improvements. Besides presenting the final results, those two subsections also give a more detailed overview of both approaches.

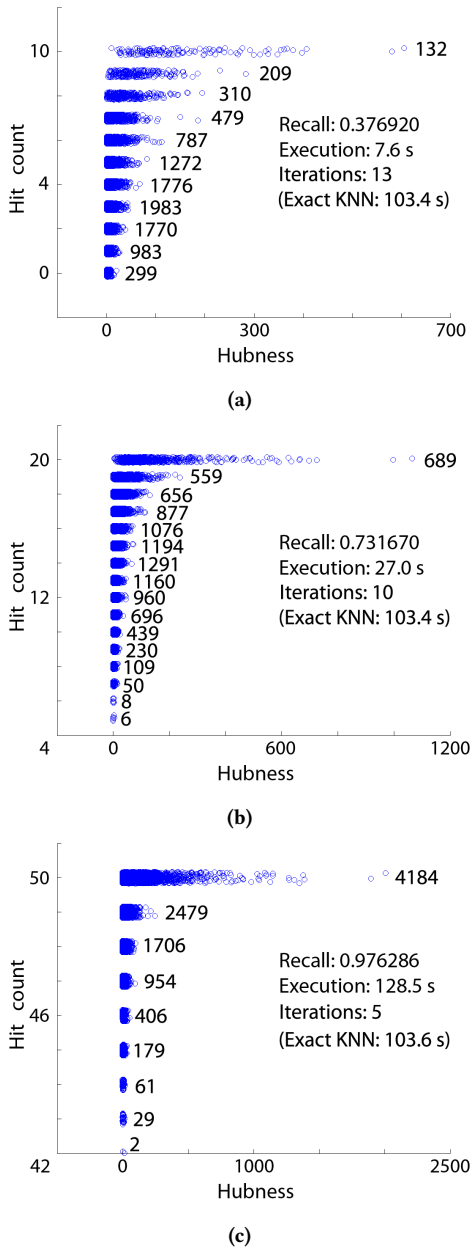

Figure 4: Hubness and hit count values of data points for the original NN-Descent algorithm. The figures show results for data sets with the following parameters: (a) $K = 10, d = 100$; (b) $K = 20, d = 100$; (c) $K = 50, d = 100$.

Before describing the new approaches, let us present a brief overview of the results generated by the original NN-Descent algorithm. The measure used to express the effectiveness of NN-Descent is *recall* 4. Recall can be calculated as the sum of all hit count values divided by the product of data-set size and K (and, in this case, is actually equivalent to precision and accuracy).

The results of the original algorithm, as shown in Figure 4, depend strongly on the size of the NN lists (K). In the case where $d = 100$ and $K = 10$ (Figure 4a), the recall is only 0.377, whereas for $K = 20$ the recall is 0.732 (Figure 4b), and for $K = 50$ the recall is 0.976 (Figure 4c).

As the motivation for NN-Descent is the speed-up of the generation of K-NNGs, the speed of execution must also be taken into account. For the case where $K = 10$ and $K = 20$, NN-Descent computes an approximate K-NNG much faster than what would be required to compute an exact K-NNG. However, for $K = 50$, the NN-Descent algorithm required a computation time even greater than that of exact K-NNG construction. Therefore, even though the graph produced by $K = 50$ is very accurate, for the relatively low data set size considered in our experimentation, the extremely high execution time makes NN-Descent impractical to use for this large choice of neighborhood size. (Note that for larger data sets execution time could still be in favor of NN-Descent.)

5.1 Hubness-Aware Variant

We now show how hubness values can be used to help guide the choice of candidate points for inclusion in the NN list of a given data point. After implementing the simple hubness estimator described in Section 3, up-to-date hubness estimates become available at any given iteration of the algorithm execution. After each stage, these values become more accurate; they are reasonably precise even after the very first few stages, as a consequence of NN-Descent itself having the same property.

Let us now describe in detail how this strategy works. In each iteration, we check if a given point has hubs in its NN list. Let h be a threshold on the number of hub points to be considered from the NN list of the given point. The idea is to replace h points of high hubness value in the current NN list with h new points chosen at random. The intuition behind this modification of NN-Descent is to diminish the impact of hubs on the update of NN lists. By adding one random point in place of each hub, we attempt to increase the probability of undiscovered neighbor points to associate themselves with the given point.

What remains to be clarified is the precise mechanism by which these h points are determined. Rather than simply choosing the h points with highest hubness values, in order to allow for the possibility of other improvements, we let the hubness value of a given point to determine the probability (between 0 and 1) of the point being included among the h points to be replaced. To generate a valid probability, we employed a linear transformation of the raw hubness values into the interval $[0, 1]$. For the purpose of the transformation, we introduce values $h_{\min}$ and $h_{\max}$ representing the maximum and minimum hubness values for which the linear transformation is applied; if the hubness value is greater than $h_{\max}$ or smaller than $h_{\min}$, the probabilities assigned would be 1 or 0, respectively. The probability that a point is selected for discard is shown in Equation 1, where x denotes the given data point and h_x is the current hubness value of point x.

$$\Pr[discard(x)] = \begin{cases} 0, & \text{if } h_x < h_{\min} \\ 1, & \text{if } h_x > h_{\max} \\ \frac{h_x - h_{\min}}{h_{\max} - h_{\min}}, & \text{otherwise.} \end{cases} \quad (1)$$

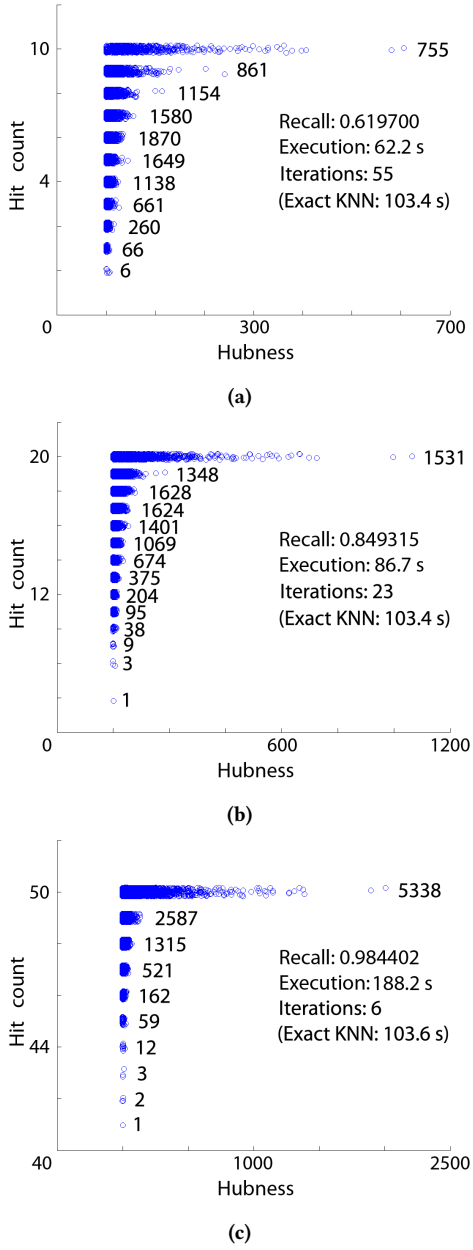

Figure 5: Hubness and hit count values for the hubness-aware NN-Descent variant, as described in Section 5.1. The figures show the results for data sets with the following parameter choices: (a) $K = 10, d = 100$; (b) $K = 20, d = 100$; (c) $K = 50, d = 100$.

Experimental results for this approach are presented in Figure 5, where the algorithm was executed for the choices $h_{\min} = 2K$ and $h_{\max} = 20K$. The experiments confirm the effectiveness of the choice of values for $h_{\min}$ and $h_{\max}$. For the case where $K = 10$, we obtained a recall of 0.620, which is almost twice that of the original recall. For $K = 20$ a recall of 0.849 was obtained, while for

$K = 50$ the recall was 0.984. In comparison with the original NN-Descent, smaller changes in recall were observed as K increases, due to the higher accuracy achieved for lower K values. Even though the recall rate is significantly improved, this came at the cost of higher execution times than the original NN-Descent. Nevertheless, the hubness-aware approach achieves better execution times as compared to the construction time for the exact K-NNG, except for the case when $K = 50$, which proved to be the same as for the original NN-Descent. For large K, the large computation time required by the hubness-aware variant was likely due to an increase in the number of iterations required to converge, due to the added variability introduced by the substitution of hubs with other data points.

5.2 Oversized NN List Variant

The second NN-Descent variant considered involved the execution of NN-Descent with some larger choice of neighborhood size $K' > K$, followed by a truncation of the NN lists to the target size K. We analyzed this approach with several different choices of K':

- Running NN-Descent with $K' = 2K$,
- Running NN-Descent with $K' = 2K$ and a sampling rate of 0.5,
- Running NN-Descent with $K' = 4K$ and sampling rate of 0.25.

The experimental results obtained for these three settings are given in Table 1, from which we can observe a trade-off between execution time and recall, in that settings that produce better recall require more time to execute. As with the the original NN-Descent and its hubness-aware variant, the execution time for $K = 50$ is longer than the time needed for the calculation of the exact K-NNG, which renders the algorithm impractical despite the extremely accuracy achieved. For $K = 20$ as well, the oversized NN-list variant could not outperform brute force K-NNG construction for the largest oversized list choice $K' = 4K$, despite the use of sampling at a rate of 0.25. All other parameter combinations considered produced a satisfactory recall score in a reasonably short time.

Table 1: Recall values and execution times of all parameter combinations. The data dimensionality is 100.

		$K = 10$	$K = 20$	$K = 50$
Exact K-NNG	Recall	1	1	1
	Execution time	103.4	103.4	103.6
	Iterations count	13	10	5
$2K$, sampling = 1	Recall	0.760	0.960	0.999
	Execution time	26.9	88.3	420.3
	Iterations count	10	6	3
$2K$, sampling = 0.5	Recall	0.630	0.906	0.998
	Execution time	17.6	57.5	284.9
	Iterations count	15	10	5
$4K$, sampling = 0.25	Recall	0.859	0.985	0.999
	Execution time	46.9	151.5	967.5
	Iterations count	17	10	6

The best trade-off between recall and execution time is obtained by the setting with $K' = 2K$ and sampling of 0.5. Figure 6 shows the relation between hubness values and hit count for the mentioned setting. Even though this method produces quite satisfying results, it has an obvious downside in its increased sensitivity (in terms of execution time) to the chosen K value.

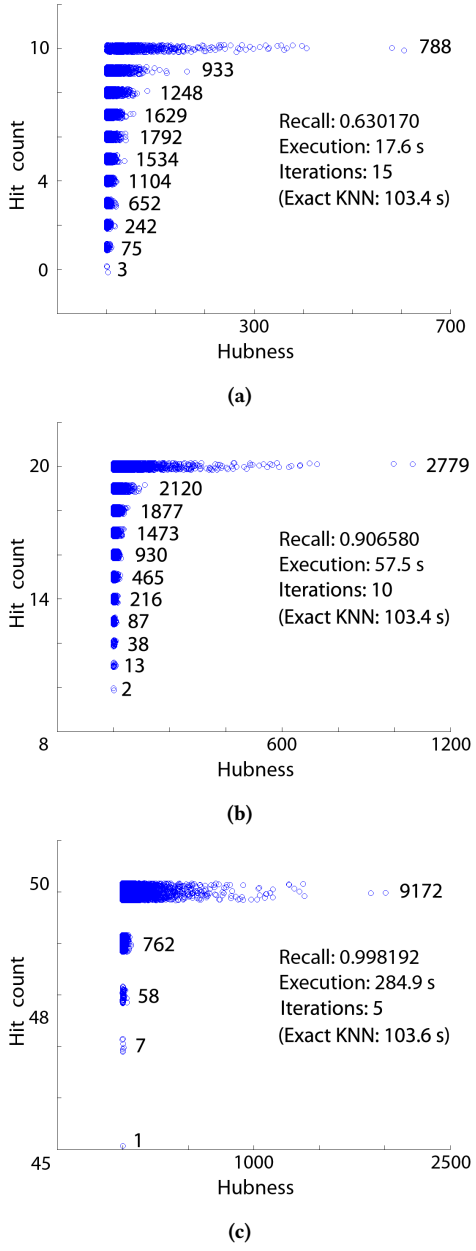

Figure 6: Hubness and hit count values for the oversized NN list variant of NN-Descent described in Section 5.2. The figures show the results for data sets with the following parameter choices: (a) $K = 10, d = 100$; (b) $K = 20, d = 100$; (c) $K = 50, d = 100$.

6 CONCLUSION

Poor performance of the NN-Descent algorithm on high-dimensional data has been observed in the past, but until now has lacked a satisfactory explanation. In this paper, we provided an experimental analysis that reveals a connection between performance and the well-studied hubness characteristics of data sets, in that the original formulation of the NN-Descent algorithm does not perform well when the data contains many hubs. In order to address this shortcoming, we introduced two different variants of the original algorithm. Our experimental results show that the new NN-Descent variants achieve better recall at the expense of an increase in execution time.

In the future, a more detailed evaluation of the described phenomenon and proposed methods will be conducted. The evaluation should, among other things, include an analysis on real data sets. Future work could also target the fine tuning of both of NN-Descent variants. In particular, the hubness-aware variant could be re-examined with the objective of decreasing its iteration count; with the oversized NN list variant, the main research target should be the improvement of sampling strategies. Conceivably, better performance may be achieved if the points chosen for NN list improvement were selected according to a carefully-managed non-uniform strategy.

ACKNOWLEDGMENTS

M. E. Houle gratefully acknowledges the financial support of JSPS Kakenhi Kiban (B) Research Grants 15H02753 and 18H03296. V. Kurbalija and M. Radovanović thank the Serbian Ministry of Education, Science and Technological Development for support through project no. OI174023, "Intelligent Techniques and their Integration into Wide-Spectrum Decision Support." V. Oria acknowledges the financial support of NSF Research Grants DGE 1565478 and ICER 1639683.

REFERENCES

[1] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation* 15, 6 (2003), 1373–1396.
[2] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying density-based local outliers. In *ACM Sigmod Record*, Vol. 29. ACM, 93–104.
[3] M. R. Brito, E. L. Chavez, A. J. Quiroz, and J. E. Yukich. 1997. Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. *Statistics & Probability Letters* 35, 1 (1997), 33–42.
[4] Jie Chen, Haw-ren Fang, and Yousef Saad. 2009. Fast approximate kNN graph construction for high dimensional data via recursive Lanczos bisection. *Journal of Machine Learning Research* 10 (2009), 1989–2012.
[5] Howie M. Choset. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementation.* MIT press.
[6] Michael Connor and Piyush Kumar. 2010. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics* 16, 4 (2010), 599–608.
[7] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13, 1 (1967), 21–27.
[8] Belur V. Dasarathy. 2002. Data mining tasks and methods: Classification: Nearest-neighbor approaches. In *Handbook of Data Mining and Knowledge Discovery*. Oxford University Press, 288–298.
[9] Thibault Debatty, Pietro Michiardi, Olivier Thonnard, and Wim Mees. 2014. Building k-nn graphs from large text data. In *Proc. 2014 IEEE Int. Conf. on Big Data*. IEEE, 573–578.
[10] Wei Dong, Moses Charikar, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proc. 20th Int. Conf. on the World Wide Web (WWW)*. ACM, 577–586.
[11] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI)*, Vol. 22. 1312.
[12] Ville Hautamaki, Ismo Karkkainen, and Pasi Franti. 2004. Outlier detection using k-nearest neighbour graph. In *Proc. 17th Int. Conf. on Pattern Recognition (ICPR)*, Vol. 3. IEEE, 430–433.
[13] Michael E. Houle, Xiguo Ma, Vincent Oria, and Jichao Sun. 2014. Improving the quality of K-NN graphs for image databases through vector sparsification. In *Proc. 4th ACM Int. Conf. on Multimedia Retrieval (ICMR)*. ACM, 89.
[14] Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. 2006. Practical construction of k-nearest neighbor graphs in metric spaces. In *International Workshop on Experimental and Efficient Algorithms*. Springer, 85–97.
[15] Youngki Park, Sungchan Park, Sang-goo Lee, and Woosung Jung. 2013. Scalable k-nearest neighbor graph construction based on greedy filtering. In *Proc. 22nd Int. Conf. on the World Wide Web (WWW)*. ACM, 227–228.
[16] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. 2010. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research* 11 (2010), 2487–2531.
[17] Milos Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. 2010. On the existence of obstinate results in vector space models. In *Proc. 33rd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*. ACM, 186–193.
[18] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. 2010. Time-series classification in many intrinsic dimensions. In *Proc. 2010 SIAM Int. Conf. on Data Mining (SDM)*. SIAM, 677–688.
[19] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. 2015. Reverse nearest neighbors in unsupervised distance-based outlier detection. *IEEE Transactions on Knowledge and Data Engineering* 27, 5 (2015), 1369–1382.
[20] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000), 2323–2326.
[21] Jagan Sankaranarayanan, Hanan Samet, and Amitabh Varshney. 2007. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics* 31, 2 (2007), 157–174.
[22] Lawrence K. Saul and Sam T. Roweis. 2003. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research* 4 (2003), 119–155.
[23] Nenad Tomašev, Miloš Radovanović, Dunja Mladenić, and Mirjana Ivanović. 2011. The role of hubness in clustering high-dimensional data. In *Proc. 15th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD)*. Springer, 183–195.