Sibson's formula for point classification

Andrea de las Heras-Parrilla*[1], Clemens Huemer†[1], and Dolores Lara‡[2]

[1]Universitat Politècnica de Catalunya
[2]Centro de Investigación y de Estudios Avanzados

Abstract

We present a novel method, called KNaN, for point classification inspired by a recent generalization of Sibson's interpolation formula to higher-order Voronoi diagrams. An experimental study shows that KNaN performs very well, also when compared with the widely used KNN-method. For our method, we present an algorithm to construct the region of the order-k Voronoi diagram of a data set modified by a query point. This generalizes the incremental algorithm for Voronoi diagrams of order 1 to Voronoi diagrams of order $k > 1$.

1 Introduction

Supervised learning is a key technique in machine learning, where a model is trained using a labelled dataset, with each input paired with a known output or class. From this training set, the model learns to predict or classify new data. One of the most used classification methods is k-Nearest Neighbours (KNN), developed in [4, 6]. KNN is a simple, yet effective algorithm that classifies a data point based on the majority class of its nearest neighbours in the feature space. It is commonly used in various applications such as image recognition, recommendation systems, and medical diagnosis. See e.g. [13] for a recent survey.

In this paper, we propose another method for point classification, which is based on Sibson's interpolation formula. Let S be a set of n points in general position in $\mathbb{R}^d$. The order-k Voronoi diagram of S, $V_k(S)$, is a subdivision of $\mathbb{R}^d$ into cells such that points in the same cell have the same k nearest points of S. Sibson [11] obtained a formula to express a point q of S as a convex combination of other points of S, using the ratios of the volumes of intersections between the cells of $V_2(S)$ and the cell of Q in $V_1(S)$. This was the basis for the natural neighbour interpolation [12]. Natural neighbour interpolation assigns values to unknown points by applying the same ratios of Sibson's formula to the values of known points.

In our previous work [2], we generalized Sibson's formula to Voronoi diagrams of any order. This generalization expands the applicability of natural neighbour interpolation by allowing the method to work with higher-order Voronoi diagrams instead of considering only the first-order Voronoi diagram. Given $V_k(S)$ and a point $p \in S$, the region $R_k(p)$ of p is defined as the union of cells of $V_k(S)$ that have the point p as one of their k nearest neighbours. We denote a cell of $V_k(S)$ as $f(P_k)$ where $P_k \subset S$ is the subset of the k nearest points from S to each point in the cell. Let $q_\ell \in S$. If $R_k(q_\ell)$ is a bounded region, with $1 \leq k \leq n-2$, then q_ℓ can be expressed as a convex combination of other points of S as follows [2]; σ denotes the Lebesgue measure on $\mathbb{R}^d$.

$$q_\ell = \sum_{f(P_k)\in R_k(q_\ell)} \sum_{\substack{f(P_{k+1})\in V_{k+1}(S) \\ q_j \in P_{k+1}\setminus P_k}} \frac{\sigma(f(P_{k+1}) \cap f(P_k))}{\sigma(R_k(q_\ell))} q_j.$$

Our proposal introduces a classification method using this generalization of Sibson's formula. Similarly to the KNN method, our method uses a parameter k to define the order of the Voronoi diagram considered in the analysis. However, unlike KNN, which assigns a class based on the most frequent label among the k nearest neighbours, in our approach the class assigned to the unknown point is determined by the highest cumulative value of Sibson's formula coefficients among its order-k natural neighbours of the same class. See Figure 2 for an example with $k = 1$. Although the classification described works for any dimension, here we only consider point sets in dimension $d = 2$. Our method is called KNaN, which stands for Order-K Natural Neighbours. Our method differs from other classification methods that use natural neighbours, such as ENN and its variants [14]. In KNaN, we need to calculate the region $R_k(q)$ of a query point q in the plane that we want to classify; this is described in Section 2. This also yields an algorithm to construct $V_k(S)$ that extends the classic incremental algorithm to construct $V_1(S)$. Section 3 discusses some more details of KNaN, followed by some experimental results in Section 4.

2 Point insertion in $V_k(S)$

Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in general position in the plane. Given the Voronoi diagrams

*Email: andrea.de.las.heras@upc.edu.
†clemens.huemer@upc.edu.
‡Email: dolores.lara@cinvestav.mx.

$V_{k-1}(S)$ and $V_k(S)$, and a point $q \notin S$, in this section, we describe an algorithm to construct $R_k(q)$, that is, the region of point q in $V_k(S \cup q)$.

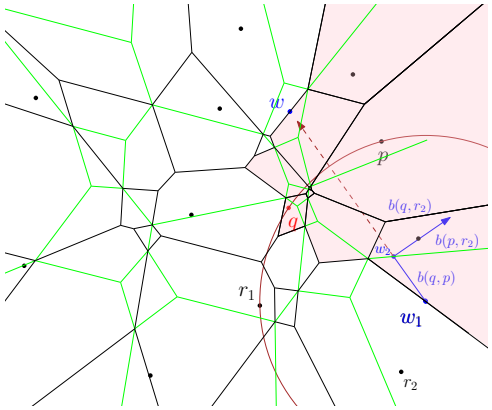
Our algorithm is a generalization of the 1978 incremental insertion algorithm by Green and Sibson [7], to construct a first-order Voronoi diagram; with complexity $O(n^2)$. Several improvements to the original algorithm have been made, including an algorithm with an expected running time $O(n)$ [9], which is the best possible. Regarding the construction of the order-k Voronoi diagram, the algorithm with the best running time is a randomized algorithm given in [1] with expected complexity $O(n \log n + nk)$. We refer the reader to [1] for a recent review of previous results on algorithms for $V_k(S)$. The algorithm we present here to construct $R_k(q)$ is a deterministic algorithm with complexity $O(k(n-k))$; this algorithm can be extended to construct $V_k(S \cup q)$.

2.1 The algorithm

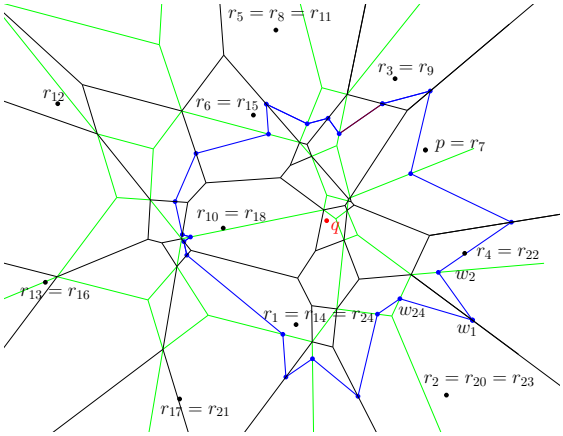
Suppose that we have already constructed $V_k(S)$, and now we want to add a new site q. First, we construct the boundary of the region $R_k(q)$ of q, then remove the substructure of $V_k(S)$ inside $R_k(q)$, and add all the vertices and edges of $V_{k-1}(S)$ that lie in the interior of $R_k(q)$.

A central part of the algorithm is the computation of the boundary of $R_k(q)$. To do this, find $f(P_k)$, the cell of $V_k(S)$ that contains q; $P_k \subset S$ is the set of sites that define the cell. Let p denote the point in P_k farthest from q. Consider $R_k(p)$, the region of p in $V_k(S)$. Since $R_k(p)$ is star-shaped, the perpendicular bisector $b(p,q)$, between p and q, crosses its boundary at at least two points; construct these two points w and w_1. These are vertices of $R_k(q)$. The circle centred on w_1 and passing through points $\{p,q\}$ also passes through some other point in S, call this point r_1. All this is illustrated in Figure 1a. Starting with w_1, we construct the boundary of the region $R_k(q)$ as follows. Draw the directed line segment $\overrightarrow{w_1 w}$ until it crosses the boundary of either $V_{k-1}(S)$ or $V_k(S)$. In either case, the intersected edge must be a bisector between p and some point in S, say r_2. The centre of the circle $\{q,p,r_2\}$ is a vertex of $R_k(q)$; let this point be w_2; refer to Figure 1a. Restart the process with the bisector between q and r_2, and repeat this procedure until we reach the starting point w_1. Let $(\overline{w_1 w_2}, \ldots, \overline{w_m w_1})$ be the constructed sequence of segments. This sequence forms the counter-clockwise boundary of $R_k(q)$; the boundary of $R_3(q)$ is shown in Figure 1b. The procedure described works only when $R_k(q)$ is bounded; the case for unbounded regions is considered in Section 3.

In order to present the pseudocode of the algorithm, we must first take care of some details. We distinguish two types of vertices [3]: those appearing in both diagrams V_{k-1} and V_k are of type I, and those only



(a) $V_2(S)$ and $V_3(S)$ are shown in green and black; $R_3(p)$ is shown in pink.



(b) The boundary of $R_3(q)$ is shown in blue.

Figure 1: Illustration of the computation of $\delta R_k(q)$.

appearing in V_k are of type II. In what follows, we assume that both diagrams are given as an input to the algorithm and that each diagram is stored in the data structure known as DCEL. We follow the definition of DCEL given in [5], with some modifications. Due to the restricted space, we refrain from describing the data structure in detail. The modifications are as follows. If $v \in V_k(S)$ is a type II vertex, then v must have a record in V_k and a record in V_{k-1}; we link these records using a pointer called $Pair(v)$. If v is of type I then $Pair(v)$ is NIL. Each site $s \in S$ is stored in the DCEL; the site record of a site s stores its coordinates and a pointer to one half-edge for each face bounded by some bisector formed by s and some other site. A half-edge $\vec{e}$ bounds the face to its left. A half-edge $\vec{e}$ that belongs to a perpendicular bisector $b(i,j)$ has the ordered pair i,j as a label if i is to the left of $\vec{e}$ and j is to its right; $Twin(\vec{e})$ has the label j,i. Each record half-edge $\vec{e}$ stores its label in a field called $Label(\vec{e})$; $Label(\vec{e}).i$ returns the first entry and $Label(\vec{e}).j$ returns the second entry.

First, we have to address how to compute the intersection of the boundary of $R_k(p)$ with the bisector $b(p,q)$. Recall that p denotes the point in P_k farthest

from q. Denote the boundary of $R_k(p)$ by $\delta R_k(p)$. By construction, the record of p contains a pointer to some half-edge in $\delta R_k(p)$; let this half-edge be $\vec{e}$. Navigate the boundary of $R_k(p)$, in the direction of $\vec{e}$, asking in each half-edge if there is an intersection with the bisector. When we encounter the special case in which the half-edge is not bounded, we return to the start vertex and navigate the boundary in the direction contrary to $\vec{e}$. The pseudocode of the algorithm is shown in the Appendix A, Algorithm1; it has complexity $O(|\delta R_k(p)|)$. Note that we are assuming that p is given as an input to the algorithm, if this is not the case, then we must add $O(n)$ time to find the kth nearest neighbour of q.

Now that we have a point on $\delta R_k(q)$ we can compute the rest. Initialize a DCEL D_p to store $V_{k-1}(S)$; the algorithm adds edges and vertices to D_p as needed. Let e_1 be the half-edge returned by Algorithm 1; recall that a half-edge bounds the face to its left. Let $w_1 = b(p,q) \cap \vec{e_1}$, since w_1 is a vertex of $R_k(q)$, we add it to D_p. We wish to choose the end point of $\vec{e_1}$ that lies inside $R_k(q)$; let us call this point v. v is the centre of the circle passing through the two points corresponding to $Label(\vec{e_1})$ and that contains the point q. During the execution of the algorithm, $\vec{e}$ always points to v and w stores the last vertex of $R_k(q)$ that was found. The algorithm works as follows: if v is type I, we navigate the current face in the direction of $\vec{e}$, checking at each edge whether there is an intersection with the current bisector $b(p, r_i)$ or not; if there is an intersection, then we update D_p and the bisector, we also change the face and direction of navigation. Now, suppose that we are navigating V_k (resp. V_{k-1}), if v is of type II then the current face contains inside vertices and edges from V_{k-1} (resp. V_k) [3]. Navigate through these edges and vertices asking at each edge whether there is an intersection with the bisector or not; if there is an intersection, we change face and direction. The pseudocode of the described algorithm is presented in detail in the Appendix A, Algorithm 2. The bisector between points p_i and p_{i+1} is represented in the pseudocode as the pair of points (p_i, p_{i+1}).

The last point we have to address is how to trim the substructure of $V_{k-1}(S)$, to keep only what lies inside $R_k(q)$. Suppose a new vertex w' of $\delta R_k(q)$ is found and let $\vec{e}$ be the edge, either of V_{k-1} or of V_k, which is currently being processed. If $\vec{e}$ is an edge of V_{k-1}, then $\vec{e}$ and $Twin(\vec{e})$ are trimmed: let $v = Origin(\vec{e})$, then $\vec{e}$ is now (v, w') and $Twin(e)$ is now (w', v). The incident face of both records remains the same. All these operations can be handled in constant time each.

Algorithm2 has complexity proportional to the number of edges of $V_{k-1} \cup V_k$ that it visits. In the worst case, this number is equal to the number of edges of V_{k-1} plus the number of edges of V_k. The number of edges, vertices and faces in $V_k(S)$ does not exceed $O(k(n-k))$ [8]. However, this upper bound for our algorithm is far from tight. We believe that its complexity is, in fact, proportional to the size of $\delta R_k(q)$, since the only edges that are visited are those of the cells that have non-empty intersection with $\delta R_k(q)$. The expected size of $\delta R_k(p)$ for p chosen randomly from S is $O(k)$.

3 KNaN: Order k Natural Neighbours Classification

The class of the new point is given by the maximum between the sums of ratios of the nearest neighbours belonging to the same class; see Figure 2, and the formula in the introduction.

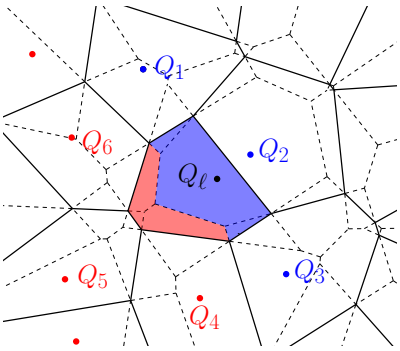


Figure 2: Coloured areas given by the intersections of $f(\{Q_\ell\})$ and the cells of $V_2(S)$ (shown in dashed). The class assigned to Q_ℓ is blue, because it is the colour that occupies the largest area, i.e., blue is the class associated with the maximum among the sums of ratios.

The number of natural neighbours of each point depends on its location within the feature space. In this sense, our method adapts to the local density and distribution of the data. This means that unlike KNN that uses a fixed number of neighbours for all points, natural neighbour classifications use a flexible neighbourhood size for each point, depending on the data set. Sibson's formula only works for bounded faces. This is a problem in the practical implementation of the classifier, since there would be points that could not be classified in this way.

To address the problem described above, we propose the following strategy for the case of dimension $d = 2$. First, we assign a fixed length to each unbounded edge of $V_k(S)$. Then, we join the new endpoints with *phantom* edges, which close the unbounded regions and allow us to apply Sibson's formula. Using a sufficiently large length, we approximate the volume ratios generated inside the face at the intersection with the faces of $V_{k+1}(S)$, as the edge lengths approach infinity.

4 Experimental Results

In this section, we present results that compare the performance of our classifier (KNaN) with the k-Nearest Neighbours (KNN) algorithm. Standard metrics such as precision and recall were used to evaluate performance in the different synthetic datasets generated. Since we have already discussed the algorithmic complexity in the previous section, in this section, we are not going to evaluate or compare execution times.

The Python module Scikit-Learn [10] provides various methods for generating synthetic data sets to train and test machine learning models. These methods create data with different distributions. The module also includes classification algorithms as well as functions for cross-validation and model evaluation. Following the example results of Scikit-Learn, we tested the method for 2-dimensional data sets consisting of 100 points, distributed between two labelled classes. For the experiments, 60 points were chosen with an equal representation of each class for the training set, and the remaining 40 points were used as the test set. Given the way these datasets are generated, adding more points increases class density and may cause overfitting. In addition, we increased the dataset noise to simulate more challenging conditions.

After evaluating the performance for different values of k in each dataset, we have selected the best results for both our KNaN method and the KNN classifier. Table 1 presents the optimal results for each approach, where the results have been chosen according to the highest accuracy among all the tested values of k. The metrics listed in the table are the following. Accuracy (Acc) is the ratio of correct predictions to the total number of cases evaluated. Precision (P) is the ratio of true positive predictions to all positive predictions. Recall (R) is the ratio of the true positive prediction to the total number of actual positive cases. F1-score (F1) can be interpreted as a harmonic mean of precision and recall.

Based on the experiments, both classifiers have similar performance. However, when noise is introduced into the data, our method seems to better preserve the accuracy of the classification. Furthermore, for the synthetic data sets tested, $k = 2$ offers the most stable performance for our method, whereas for KNN, $k = 3$ yields the best results.

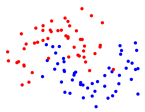
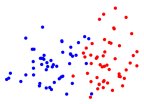
Dataset	KNaN	KNN
	Acc= 93.4 ± 4.35 P= 94.82 ± 3.52 R= 92 ± 6.78 F1= 93.3 ± 4.66	Acc= 92.5 ± 6.32 P= 93.54 ± 3 R= 91 ± 10.67 F1= 92.25 ± 4.68
	Acc= 87.5 ± 3.5 P= 88.93 ± 5.23 R= 86 ± 3.74 F1= 87.34 ± 3.46	Acc= 82.5 ± 4.18 P= 85.12 ± 4.3 R= 79 ± 7.34 F1= 81.94 ± 5.43
	Acc= 94 ± 2.54 P= 95.28 ± 4.78 R= 93 ± 5.1 F1= 93.93 ± 2.63	Acc= 92.5 ± 3.16 P= 94.5 ± 6.09 R= 91 ± 6.63 F1= 92.71 ± 6.34

Table 1: Comparison of KNaN and KNN. For each metric, the value in the table correspond to the mean of the different runs in the cross-validation, followed by the standard deviation (mean ± std).

References

[1] T. M. Chan, P. Cheng, and D. W. Zheng. An optimal algorithm for higher-order Voronoi diagrams in the plane: The usefulness of nondeterminism. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4451–4463, 2024.

[2] M. Claverol, A. de las Heras Parrilla, C. Huemer, and D. Lara. Sibson's formula for higher order Voronoi diagrams. 2024. `https://arxiv.org/abs/2404.17422`.

[3] M. Claverol, A. de las Heras Parrilla, C. Huemer, and A. Martínez-Moraian. The edge labeling of higher order Voronoi diagrams. *J. Global Optim.*, 90(2):515–549, 2024.

[4] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, 13:21–27, 1967.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry*. Springer, 2010.

[6] E. Fix and J. L. Hodges. Discriminatory analysis. Nonparametric discrimination: Consistency properties. *U.S. Air Force Sch. Aviation Medicine, Randolph Field, Texas, Project 21-49-004, Tech. Rep.*, 54, 1951.

[7] P. J. Green and R. Sibson. Computing Dirichlet tessellations in the plane. *Comput. J.*, 21:168–173, 1978.

[8] D.-T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, 31:478–487, 1982.

[9] T. Ohya. Improvements of the incremental method for the Voronoi diagram with computational comparison of various algotithms. *Journal of Operations Research Society of Japan*, 27(4):69–97, 1984.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[11] R. Sibson. A vector identity for the Dirichlet tessellation. *Math. Proc. Camb. Philos. Soc.*, 87(1):151–155, 1980.

[12] R. Sibson. A brief description of natural neighbour interpolation. In *Interpreting multivariate data*, pages 21–36. John Wiley & Sons, 1981.

[13] P. K. Syriopoulos, S. B. Kotsiantis, and M. N. Vrahatis. Survey on KNN methods in data science. In *Int. Conf. on Learning and Intelligent Optimization. LNCS*, volume 13621, pages 379–393. Springer, 2023.

[14] B. Tang and H. He. ENN: Extended nearest neighbor method for pattern recognition. *IEEE Computational Intelligence Magazine*, 10(3):52–60, 2015.

A Pseudocode

Algorithm 1 Computing $\delta R_k(p) \cap b(p,q)$

Input: $b(p,q)$, half-edge $\vec{e}$ in $\delta R_k(p)$ and with $Label(\vec{e}).i = p$
Output: E one edge in $\delta R_k(p)$ intersected by $b(p,q)$

procedure RKP($b(p,q)$, $\vec{e}$)
  $E =$ NIL; $start = Origin(v)$
  repeat
    if $b(p,q) \cap \vec{e} \neq \emptyset$ **then**
      $E = \vec{e}$
    end if
    if $Origin(Twin(\vec{e})) = \infty$ **then**
      $v = start$
    else
      $v = Origin(Twin(\vec{e}))$
    end if
    Assign $\vec{e}$ to a half-edge $\vec{g}$, with $\vec{e} \neq \vec{g}$, $Origin(\vec{g}) = v$ and $Label(\vec{g}).i = p$
  until $E \neq$ NIL
  return E
end procedure

Algorithm 2 Computing $\delta R_k(q)$

Input: w_1, $\vec{e}$, v, $bisector(p,q)$
Output: DCEL D_q containing $\delta R_k(q)$

procedure RKQ($w_1, \vec{e}, v, (p,q)$)
  Add w_1 to D_q
  $w = w_1$
  $bisector = (p,q)$
  if $v = Origin(\vec{e})$ **then**
    $\vec{e} = Twin(\vec{e})$
  end if
  repeat
    if v is of type I **then**
      $w' = Next(\vec{e}) \cap bisector$
      if $w' \neq \emptyset$ **then**
        Add w' to D_q
        Add new edge (w, w') to D_q
        $w = w'$; $w' = \emptyset$
        $bisector = (q, Label(\vec{e}).j)$
        $\vec{e} = Twin(Next((\vec{e})))$
      else
        $\vec{e} = Next(\vec{e})$
        $v = Origin(Twin(\vec{e}))$
      end if
    else
      $Newlabel = Label(\vec{e}) \triangle Label(Next(\vec{e}))$
      $v = Pair(v)$
      Let $\vec{e}$ be the edge incident in v and with label $Newlabel$
      $w' = \vec{e} \cap bisector$
      if $w' \neq \emptyset$ **then**
        Add w' to D_q
        Add new edge (w, w') to D_q
        $w = w'$; $w' = \emptyset$
        $bisector = (q, Label(\vec{e}).j)$
        $\vec{e} = Twin(\vec{e})$
      else
        $v = Origin(Twin(\vec{e}))$
      end if
    end if
  until $v = w_1$
  return D_q
end procedure