Versatile Document Image Content Extraction

Henry S. Baird, Michael A. Moll, Jean Nonnemaker, Matthew R. Casey, Don L. Delorenzo

Computer Science & Engineering Dept
Lehigh University
19 Memorial Dr West
Bethlehem, PA 18017 USA

E-mail: baird@cse.lehigh.edu
URL: www.cse.lehigh.edu/~baird

ABSTRACT

We offer a preliminary report on a research program to investigate versatile algorithms for *document image content extraction*, that is locating regions containing handwriting, machine-print text, graphics, line-art, logos, photographs, noise, etc. To solve this problem in its full generality requires coping with a vast diversity of document and image types. Automatically trainable methods are highly desirable, as well as extremely high speed in order to process large collections. Significant obstacles include the expense of preparing correctly labeled ("ground-truthed") samples, unresolved methodological questions in specifying the domain (*e.g.* what is a representative collection of document images?), and a lack of consensus among researchers on how to evaluate content-extraction performance. Our research strategy emphasizes *versatility first*: that is, we concentrate at the outset on designing methods that promise to work across the broadest possible range of cases. This strategy has several important implications: the classifiers must be trainable in reasonable time on vast data sets; and expensive ground-truthed data sets must be complemented by amplification using generative models. These and other design and architectural issues are discussed. We propose a trainable classification methodology that marries k-d trees and hash-driven table lookup and describe preliminary experiments.

Keywords: *Bayes decision theory, classification, k Nearest Neighbors, k-d trees, CART, spatial data structures, computational geometry, hashing*

1. INTRODUCTION

We are investigating unusually versatile algorithms for the **document image content extraction** problem:

> *Given* an image of a document,
> *find* regions containing handwriting, machine-print text, graphics, line-art, logos, photographs, noise, etc.

To solve this problem in full generality—given the great diversity of document and image types—we have decided to emphasize, in our research strategy:

1. *versatility first*: concentrate first on designing methods that work across the broadest possible variety of cases (document and image types);
2. *voracious classifiers*: trainable on billions of samples in reasonable time;
3. *extremely high speed classification*: ideally, nearly at I/O rates (as fast as the images can be read);
4. *amplification*: use real ground-truthed training samples as 'seeds' for massive synthetic generation of pseudorandomly perturbed samples for use in supplementary training;
5. *confidence before accuracy*: good estimates of the confidence of decisions is important; high accuracies are desirable but even modest accuracy can be useful;
6. *near-infinite space*: *i.e.* design for best performance in a near-term future when main memory will be orders of magnitude larger and faster than today; and
7. *data-driven design*: we won't invest much effort in making what are, ultimately, arbitrary engineering decisions such as choice of preprocessors and features; instead, we'll try to allow the training data to determine these automatically as far we can;

To appear in *Proc., SPIE/IS&T Document Recognition & Retrieval XIII Conf.*, San Jose, CA, January 18-19, 2006.

2. DOCUMENT IMAGE CONTENT

Types of document images of interest include color, grey-level, and black-and-white; also, any size or resolution (digitizing spatial sampling rate); and in any of a wide range of file formats (TIFF, JPEG, PNG, etc). We have chosen to convert all image file formats in a three-channel color PNG file in the HSL (Hue, Saturation, and Luminance) color space; Black-and-white or greylevel images convert into HSL images with fixed values for Hue and Saturation.

We are gathering a large collection of electronic images of pages containing the following types of content: handwriting (HW), machine Print (MP), line Art (LA), photos (PH), math notation (MT), maps (MA), engineering drawings (ED), chemical diagrams (CD), "junk" (e.g. margin and gutter noise, JK),and blank (BL). These content classes are intended to capture the most commonly occuring types of document image regions. We attempted to collect document images represented as bitonal (black and white), greyscale, and color (although some combinations are unlikely to be found, such as Mathematics notation in color). It is of course technically possible to convert color images to greyscale, and grayscale to bitonal, but we have eschewed this in order not to introduce arbitrary image processing decisions of our own. Sources for these include: the Univ. Washington CD-ROM Data Bases ('UW'), the Univ. Nevada, Las Vegas Data Sets ('UNLV'), various Library of Congress digital library sites such as the American Memory Project ('AMem'), the ISI Kolkata mathematics notation collection ('ISI'), Lehigh University digital library collections ('LU'), the National Institute of Standards ('NIST'), NASA Astrophysics Data System ('ADS'), the New York Public Library ('NYPL'), and the National Library of Medicine ('Med'). Document images that have been carefully "zoned," and the zones labeled with their content type appear to be a rare commodity: we found them principally in the UW, UNLV, and ISI collections. Our collections to date are summarized below.

	Bitonal	**Greyscale**	**Color**
HW	AMem UNLV NIST	AMem UNLV LU	AMem NYPL
MP	UW AMem UNLV NIST Med	UW UNLV AMem LU ADS NYPL	AMem NYPL
LA	AMem DL UNLV	AMem LU ISRI	AMem NYPL
PH	UNLV	UNLV AMem LU	NYPL AMem DL
MT	ISI UW	ADS LU	(none)
MA	NYPL AMem	NYPL AMem	NYPL AMem
ED	UW	LU	(none)
CD	UW	(none)	(none)
JK	all	all	all
BL	all	all	all

We have collected 7123 page images total from these sources. We are aware of other doc-image databases which we plan to include soon. Our software can automatically incorporate existing ground-truth in the form of rectangular zones; for other images we have developed an interactive ground-truthing tool.

3. STATISTICAL FRAMEWORK FOR CLASSIFICATION

To motivate our technical approach we briefly review a few ideas well documented in the statistical pattern recognition literature. Each training and test sample will be a pixel in a document image, and each pixel owns d features which are scalar properties extracted by image processing in the vicinity of that pixel. Note that we have chosen to classify *pixels* not *regions* as most previous R&D projects have done; this is to avoid the arbitrariness and restrictiveness inevitably connected with the choice of a limited class of region shapes. (The most popular region shape in the literature is the *isothetic rectangular zone*, which offers many engineering advantages but is often a bad fit to page layouts.) Our statistical sample space (the "universe") $U = \mathbf{R}^d$, the multidimensional reals. We expect the dimensionality $d \approx 100$. A member of the sample space is called $\mathbf{x}, \mathbf{y}, \mathbf{z}$, etc.

All samples belong to one of m classes $\mathbf{C} = \{c_j\}_{j=1,\ldots,m}$. The number of classes m will, we expect, be ≈ 10.

The components $\{x_i\}_{i=1,\ldots,d}$ of an observed sample $\mathbf{x}$ are of course measurable *features*. Within the $\mathbf{R}^d$ space we choose to use the Infinity (L_∞) norm $\|\mathbf{x}-\mathbf{y}\|_\infty \equiv max_{i=1}^{d}\{|x_i - y_i|\}$. Training data consists of a set of samples $\mathbf{T} \subset \mathbf{R}^d$ labeled with their classes. Since a single page image can contain 8-10M pixels, the number of training samples $n \equiv |\mathbf{T}|$ can easily exceed 1 billion. The true class of each training sample is given by a *ground-truth* function $\mathbf{G}: \mathbf{T} \rightarrow \mathbf{C}$.

Many pattern recognition researchers believe that the choice of metric has less influence on the accuracy of a classifier than properties of the training data,[1] such as large size and representativeness. It is therefore interesting to explore classification methods which can make use of very large training sets. Note that in our context $c \ll d \ll n$.

4. A NEARLY IDEAL CLASSIFIER (FOR OUR PURPOSES)

Our classification problem can be stated as

Given $\mathbf{T}$, $\mathbf{G}$, and a previously unseen sample $\mathbf{x}$,
find the most probable class c for $\mathbf{x}$.

Adopting a naive Bayesian approach, we choose $c_{max} = argmax_{c_j \in \mathbf{C}}\{P(c_j|\mathbf{x}, \mathbf{T}, \mathbf{G})\}$ where $P(c_j|\mathbf{x}, \mathbf{T}, \mathbf{G})$ denotes the posterior probability of class c_j given the new observation $\mathbf{x}$ and the prior knowledge expressed in training data $\mathbf{T}$ and $\mathbf{G}$. One algorithm for the Classification problem is *k-nearest neighbors* (kNN):

K Nearest Neighbors algorithm

INPUT: $\mathbf{T}$, $\mathbf{G}$, $k \in \mathbf{N}^+$, and $\mathbf{x}$
OUTPUT: $c \in \mathbf{C}$
Step 1. within $\mathbf{T}$, find k nearest neighbors of $\mathbf{x}$
that is, find a set $Y \subset \mathbf{T}$, $|Y| = k$, s.t.
$\forall \mathbf{y} \in Y \;\; \forall \mathbf{z} \in T - Y \;\; \|\mathbf{x} - \mathbf{y}\| \leq \|\mathbf{x} - \mathbf{z}\|$
Step 2. within Y, use $\mathbf{G}$ to find the most frequently occurring class, breaking ties at random.

For any particular set of features, assuming that the training data is representative, the kNN algorithm enjoys a valuable theoretical property: as the size of the training set increases, the error rate of kNN approaches no worse than twice the Bayes error.[2] It also generalizes directly to more than two classes (unlike several competing methods such as neural nets and classification trees). Finally, it has has often been competitive in accuracy (if not always in speed) with more recently developed methodologies such as neural nets and support-vector machines. For these reasons, we consider kNN to be a nearly ideal classifier.

There is of course a straightforward exhaustive-search algorithm for Step 1 of time complexity $\Theta(ndk)$. A slightly refined algorithm using heaps reduces this to $\Theta(\; n \; d \; \log k \;)$. Since both c and $k \ll n$, the runtime of Step 2 is negligible; so, hereafter, we will speak as if kNN consists only of Step 1.

We have implemented this (for $k = 5$) and, in early small scale experiments, have seen it yield pixel classification rates greater than 98% correct.

Many techniques for speeding up kNN have been studied[3] including partial distance,[4] editing (using, *e.g.*, the triangle inequality property of metrics), tree search, locality-sensitive hashing[5, 6] etc—for reasons of space we can not discuss them in detail here, except to mention that, as n increases: partial-distance and editing generally improve runtimes by fixed factors only; tree search promises runtimes sublinear in n but at the cost of a blowup in space that is exponential in d; and locality sensitive hashing is sometimes fast but is known to exhibit costly asymptotic behavior on the simple parametric distributions that we know how to analyze. Thus although many of these *may* perform well in practice, we do not know, as yet, how to guarantee good performance across a broad range of cases. We are interested by highly nonuniform distributions of $\mathbf{T}$ within $\mathbf{R}^d$ for which *hashing* techniques perhaps combined with *geometric tree search*, may offer a way forward. To explore this, we next review approximation algorithms for kNN.

5. APPROXIMATIONS TO KNN

Radius Search problem

Given $\mathbf{T}$, $r \in \mathbf{R}$, and $\mathbf{x}$,
find all points of $\mathbf{T}$ within radius r of $\mathbf{x}$: that is, the set $Y_r \equiv \{ \mathbf{y} \in \mathbf{T} \;\; s.t. \; \|\mathbf{x} - \mathbf{y}\| \leq r \}$

If $r \ll s$, we can expect that finding Y_r will be easier than finding Y_s. (Of course, generally $|Y_r| < |Y_s|$ also, in which case reporting the result of the search will take *longer*. But we choose to neglect runtimes that depend on the size of the output, for reasons that will become clear later.) If $|Y_r| = k$, then radius search is equivalent to kNN. If $|Y_r| \approx k$, then we will say that radius search *approximates* kNN. (Of course, this notion of approximation does not address the key issue, which is how much less accurate radius search classification is than kNN classification.)

It might be cheaper to compute approximations to radius search.

Nested Radius Search problem

Given $\mathbf{T}$, $\{r_i\}_{i=1,\ldots,s} \in \mathbf{R}$, $r_1 < r_2 < \ldots < r_s$, and $\mathbf{x}$,
find sets $\subset \mathbf{T}$ within radii r_i of $\mathbf{x}$: that is,
$Y_{r_i} \equiv \{ \mathbf{y} \in \mathbf{T} \;\; s.t. \; \|\mathbf{x} - \mathbf{y}\| \leq r_i \}$

Note that $Y_{r_1} \subseteq Y_{r_2} \subseteq \ldots \subseteq Y_{r_s}$. If $r_k < r_l$, then it may be possible, through judicious choice of data structures and algorithms, to compute Y_{r_l} faster than Y_{r_k}.

Approximate Radius Search problem

Given $\mathbf{T}$, $r \in \mathbf{R}$, $\epsilon \in \mathbf{R}$ and $\mathbf{x}$,
find sets Y^L and $Y^U \subset \mathbf{T}$ such that
$$Y_{r-\epsilon} \subseteq Y^L \subseteq Y_r \subseteq Y^U \subseteq Y_{r+\epsilon}$$

That is, Y^L and Y^U approximate Y_r within ϵ. As $\epsilon \to 0$, $Y^L \to Y_r$ (from below) and $Y^U \to Y_r$ (from above), and so Approximate Radius Search converges to Radius Search. When ϵ is large, it should be easier to solve Approximate Radius Search than to solve Radius Search.

How can Approximate Radius Search be used for classification? Well, if it happens that $|Y^L| = |Y^U|$, then $Y^L = Y^U = Y_r$ and, exactly as in kNN, we can choose the most frequently occurring class in that set. Lets call the frequency of occurrence of each class c_j in Y^L "f_j^L" and similarly, in Y^U, "f_j^U". Let the most frequently occurring classes in these sets be $maxc^L$ and $maxc^U$; if these happen to be the same class, we will of course choose that class. But if they differ, then we have several policy choices:

- we can compute average frequencies for each c_j, *e.g.* $f_j^{aver} \equiv (f_j^L + f_j^U)/2$, and then choose the class with the highest average frequency;
- we can use the value k from kNN to assist in the decision, for example, if $|Y^L|$ is closer to k than Y^U, we choose the most frequently occurring class in Y^L (and vice versa).

Thus Approximate Radius Search can be used as a basis for classification and offers a range of engineering tradeoffs between accuracy and speed.

6. APPROXIMATE RADIUS SEARCH UNDER THE INFINITY NORM

With the Infinity Norm as our metric, then Radius Search becomes a multidimensional range search (or "region query") in which the search regions are hypercubes of radius r centered on the query sample $\mathbf{x}$. Multidimensional range searching is an intensively explored topic[7][8][9][10] by researchers in the computational geometry, image processing, pattern recognition, and graphics research communities. However, the special requirements of pattern classification may lead us in fresh directions. For example, in the literature, range searching is commonly discussed as a generalization of *point searching*, which in turn is characterized as a generalization of dictionaries to multidimensional data. Dictionaries are classically designed to support three basic operations: *insert* a single data item, *find* an item (returning its associated metadata, or returning 'not found'), and *delete* an item. Data structures and algorithms for dictionaries are thus *dynamic*: designed to process an arbitrary sequence of these three operations efficiently.

By contrast, in classification, insertions are performed all at once in an offline batch operation (the "training stage") in which all the data (training samples) are simultaneously available. Deletion never occurs (although we may choose to summarize sets of data points statistically rather than storing all of them explicitly). Thus the data structure is *static*. Finding is the only operation which is online and must run fast; and it may also be possible to batch a large set of find operations. Thus much of the literature on dynamic multidimensional search is not directly relevant to classification.

Let us review techniques for multidimensional search that may assist us in choosing, adapting, or crafting new data structures and algorithms suitable for classification.

6.1. Adaptive k-d Trees

One multidimensional search technique with broad applicability is Bentley's k-d trees.[11] The variant of k-d trees most relevant to classification seems to be the *adaptive* k-d tree.[12] Briefly, these partition the set of points recursively in stages: at each stage one of the partitions is divided into two subpartitions; we will assume that it is possible to choose cuts that achieve *balance*, that is, to divide into subpartitions containing roughly the same number of points. Division is by cutting along one of the dimensions $i \in \{1, \ldots, d\}$, *i.e.* by choosing a threshold value and assigning each of the partition's points $\mathbf{x}$ to subpartition (a) or (b) according to whether its $\mathbf{x}_i$ component value is (a) less than, or (b) greater than or equal to the threshold. (In some implementations, the threshold corresponds to a component value for one of the points, which is then stored in the interior node of the search tree; but will we assume here that all points are stored in leaf nodes). Generally, at each stage a different dimension is cut: one simple strategy is to cycle (and if necessary recyle) through the dimensions in a fixed order; another strategy is to cut the currently most populous partition. Cutting proceeds until all partitions

contain few enough points to invite a final fast sequential search. The final partitions are generally hyperrectangles (not always hypercubes) with orthogonal sides (parallel to the coordinate axes of $\mathbf{R}^d$).

Balancing each cut ensures that find operations execute in $\Theta(\log n)$ time in the worst case. Consistent with a guarantee of such logarithmic-time finds, Bentley's k-d tree construction achieves a minimum number of cuts and thus a minimum number of partitions (close to n/p, where p is the number of points in the final partitions; in our context, assuming $p = 10$, this is still huge, $\approx 10^8$). The threshold value chosen for each cut depends on the distribution of points within the partition to be cut, so the thresholds are not *independently predictable*: that is, none (after the first) can be computed without knowledge of the cut thresholds in some earlier stages. Further, given a previously unseen $\mathbf{x}$ it is not possible to compute the d upper and lower bounds of its k-d hyperrectangle (within which it lies) without traversing the k-d tree. Thus locating $\mathbf{x}$'s k-d hyperrectangle *requires* $\Theta(\ \log\ n\ )$ time.

The pruning power of k-d trees speeds up range searches. Given a query point $\mathbf{x}$ and a radius r, defining a search hypercube, it is straightforward to generalize the find algorithm to explore all k-d tree nodes whose subtrees overlap the search hypercube. The asymptotic runtime of such variants has been studied:[13] reports that the worst-case number of tree nodes explored is $\Theta(\ d\ n^{1-1/d}\ )$. In our context, this may (or may not) promise much improvement over brute-force search, since (neglecting the multiplicative constant) $d\ n^{1-1/d}\ \approx\ 100(10^9)^{0.99}\ = 100(10^{8.91})\ =\ 10^{10.91}\ \gg\ 10^8\ \approx n$ Of course this may be a pessimistic bound for several reasons. Whether or not k-d tree range searches are efficient for classification may ultimately be decidable only by experiment.

6.2. Multidimensional Tries

An alternative data structure that assists multidimensional search is a recursive partitioning of the space using *fixed cuts*. Suppose that for each dimension $i \in \{1, \ldots, d\}$, lower and upper bounds L_i and U_i on the values of the components $\mathbf{x}_i$ are known. Then one may choose to place cuts at midpoints of these ranges, *i.e.* at $(L_i + U_i)/2$, and later, when cutting those partitions, recursively cut at the midpoint of the (now smaller) ranges. A search trie can be constructed in a manner exactly analogous to k-d trees with the exception that the distribution of the data is ignored in choosing cut thresholds. It will no longer be possible to guarantee balanced cuts and thus most of the time and space optimality properties of k-d trees are lost. However, there are gains: for example, the values of the cut thresholds can be predicted (they all are of course predetermined by $\{L_i, U_i\}_{i=1,\ldots,d}$). As a consequence, if the total number of cuts r is known, the hyperrectangle within which any query (test) sample $\mathbf{x}$ lies can be computed in $\Theta(r)$ time. We will call these recursive midpoint-cut trie hyperrectangles *partitions* (they are often called 'bins' or 'cells' in the kNN literature).

6.2.1. Bit-Interleaving

Partitions resulting from tries as above can be addressed using *bit-interleaving*. Let $< d_k, m_k >_{k=1,\ldots,r}$ be a sequence of cuts, where, at cut k, d_k is the dimension chosen to be cut and m_k is the midpoint of the partition chosen for that cut. Among the partitions of feature space that result, a test sample x will fall in a partition that can be described by a sequence of decisions $b_k\ =\ (x_{d_k} > m_k)$ taking on the value False ('0') or True ('1') as x lies below or above the cut respectively. Equivalently, any bit-sequence $< b_k >_k = 1, \ldots, r$ of length r bits determines the boundaries of some partition, and so can be thought of as an address for it. To summarize: given a test sample x and a sequence of r cuts, we can compute in $\Theta(r)$ time the bit-interleaved address of the partition within which x lies. In practice this calculation is far faster than the metric calculation and is lost in the noise.

Now in this context—as in most if not all naturally occurring image pattern recognition problems—we expect that training and test data are nonuniformly distributed ("skewed") in feature space, and specifically when r is large that only a small fraction of the resulting small partitions will be occupied by *any* training data. If this assumption holds true in practice, only a few distinct bit-interleaved addresses will occur in even a very large training set, and so it may be possible to use a dictionary data structure to manage them. The feasibility of this approach depends crucially of course on how many distinct values of bit-interleaved addresses occur in practice. Experiments on our document images where $n = 10,000,000$, $d = 15$, and $r < 75$ suggest that the number of occupied partitions, as a function of the length of their bit-interleaved address, is asymptotically roughly cubic. For $r = 50$ the absolute number observed was about 2,100,000, which is of course easily manageable by single-stage hashing with the hash table contained in main memory.

In another trial with $d\ =\ 15$, $n\ =\ 844{,}525$ training points, tested on 192,405 points, brute force 5NN achieved a correct classification rate of 70% but required a huge computation (163 billion distance calculations). By contrast, hashing bit-interleaved addresses of length $r\ =\ 32$ bits achieved a 148-times speed-up while still classifying 66% of the points correctly. Longer bit-interleaved addresses were faster still and only a little less accurate: at $r\ =\ 40$ bits, we saw a 3470-times speed-up and an accuracy of 58%. However, at $r\ =\ 48$ bits accuracy fell to 26% which seems to be too low to be useful.

Figure 1. A bilevel document image (on the left) and its per-pixel classification (on the right). Dark green indicates the photo (PH) content class (concentrated iaong the left edge and upper left corner), light blue indicates machine print text (MP) (most of the center of the page), magenta indicates handwriting (HW) (along the right margin and scattered), and light gray indicates unclassifiable (JK) (the outermost margins). Although the per-pixel classification correct rate is only 77%, the principal regions and their identities are clearly indicated and it may be possible to refine them reliably in post-processing stages.

We have reason to believe that a per-pixel correct classifcation rate well below 100% may still be useful: in the following Figure, the correct classification rate is 77% but the principal regions of text and photo are clearly delimited so that it seems to us it will be easy to refine them in downstream stages of processing to support a variey of uses.

These experimental results, though preliminary, encourage us to press ahead in an exploration of fast approximate kNN methods using hashing and related techniques.

7. DISCUSSION AND FUTURE WORK

However, we may soon face other theoretical and practical obstacles to building fast approximate kNN algorithms using bit-interleaved-address hashing. How can we ensure that a partition addressed by a bit-interleaved address contains **all** of the training samples that may fall within a kNN set, or an approximate-radius NN set? It is easy to see that, in the worst case, this extra offline preprocessing (and space requirements) may be exponential in d. Also, exponential space may be needed to store redundant copies of the test samples. Again these issues must be tested empirically and analyzed further.

In a personal communication Jon Bentley has suggested to us that a hybrid of classification trees (CARTs[14]) and k-d trees may provide a circumvention of these potential problems. Specifically, the first few levels of a CART could perhaps be compressed using a hashing scheme similar to bit-interleaving; but it is not yet clear how to make this work.

REFERENCES

1. T. K. Ho and H. S. Baird, "Large-scale simulation studies in image pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**, pp. 1067–1079, October 1997.

2. T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. on Information Theory* **13**, pp. 21–27, 1967.
3. R. Weber, H. J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proc., 24th Int'l Conf. on Very Large Data Bases*, August 1998.
4. R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification, 2nd Edition*, Wiley, New York, 2001.
5. A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc., 25th Int'l Conf. on Very Large Data Bases*, September 1999.
6. M. Datar, P. Indyk, N. Immorlica, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc., Symposium on Computational Geometry*, 2004.
7. H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, Massachusetts, 1990.
8. J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM* **18**(9), pp. 509–517, 1975.
9. A. Farag, T. Linder, and G. Lugosi, "Fast nearest-neighbor search in dissimilarity spaces," *IEEE Trans. Pattern Anal. Mach. Intell.* **15**(9), pp. 957–962, 1993.
10. G. Yuval, "Finding nearest neighbors.," *Inf. Process. Lett.* **5**(3), pp. 63–65, 1976.
11. J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM* **18**(9), pp. 509–517, 1975.
12. J. H. Freidman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.* **3**(3), pp. 209–226, 1977.
13. D. T. Lee and C. K. Wong, "Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees.," *Acta Inf.* **9**, pp. 23–29, 1977.
14. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth& Brooks/Cole, Pacific Grove, CA, 1984.