Fast Nearest Neighbour Computation[1]

David Barber

University College London

[1] These slides accompany the book *Bayesian Reasoning and Machine Learning*. The book and demos can be downloaded from `www.cs.ucl.ac.uk/staff/D.Barber/brml`. Feedback and corrections are also available on the site. Feel free to adapt these slides for your own purposes, but please include a link the above website.

Finding your nearest neighbour quickly

- Consider that we have a set of datapoints $\mathbf{x}^1, \ldots, \mathbf{x}^N$ and a new query vector $\mathbf{q}$.
- Our task is to find the nearest neighbour to the query $n^* = \underset{n}{\operatorname{argmin}}\, d(\mathbf{q}, \mathbf{x}^n)$, $n = 1, \ldots, N$.
- For the Euclidean distance

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^2} = \sqrt{\sum_{i=1}^{D} (x_i - y_i)^2}$$

  and D dimensional vectors, it takes $O(D)$ operations to compute this distance.
- For a set of N vectors, computing the nearest neighbour to $\mathbf{q}$ would take then $O(DN)$ operations. For large datasets this can be prohibitively expensive. Is there a way to avoid calculating all the expensive distances?
- Useful for nearest neighbour classification and k-means clustering.

General Approaches

Orchard We'll consider first the situation in which the calculation of a distance $d(\mathbf{x}, \vec{y})$ is expensive. For example, the distance between very high dimensional vectors. We'll make use of the Triangle Inequality to reduce the number of such calculations at the cost of storing a large interpoint distance matrix.

AESA This can be used to eliminate datapoints so that $d(\vec{q}, \mathbf{x})$ need not be computed for datapoint $\vec{x}$. However, this will still require a cost proportional to the number of datapoints. Both AESA and Orchard rely on the distance being a metric.

KD Tree An alternative approach is to try to partition the space in such a way that we do not need to use many distance calculations. In this case the number of calculations may be less than N. The distance does not need to be a metric.

Note that in the worst case, all methods will have complexity equal or worse than simply calculating all distances and finding the smallest.

Metric Distances

Using the triangle inequality to speed up search

- For the squared Euclidean distance we have

$$(\mathbf{x}-\mathbf{y})^2 = (\mathbf{x}-\mathbf{z}+\mathbf{z}-\mathbf{y})^2 = (\mathbf{x}-\mathbf{z})^2 + (\mathbf{z}-\mathbf{y})^2 + 2(\mathbf{x}-\mathbf{z})^{\mathsf{T}}(\mathbf{z}-\mathbf{y})$$

- Since the scalar product between two vectors $\mathbf{a}$ and $\mathbf{b}$ relates the lengths of the vectors $|\mathbf{a}|$, $|\mathbf{b}|$ and the angle θ between them by $\mathbf{a}^{\mathsf{T}}\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos(\theta)$ we have (using $|\mathbf{x}| \equiv \sqrt{\mathbf{x}^{\mathsf{T}}\mathbf{x}}$)

$$|\mathbf{x}-\mathbf{y}|^2 = |\mathbf{x}-\mathbf{z}|^2 + |\mathbf{z}-\mathbf{y}|^2 + 2\cos(\theta)|\mathbf{x}-\mathbf{z}||\mathbf{z}-\mathbf{y}|$$

- Using $\cos(\theta) \leq 1$ we obtain the triangle inequality

$$|\mathbf{x}-\mathbf{y}| \leq |\mathbf{x}-\mathbf{z}| + |\mathbf{z}-\mathbf{y}|$$

- Geometrically this simply says that for a triangle formed by the points $\mathbf{x}, \mathbf{y}, \mathbf{z}$, it is shorter to go from $\mathbf{x}$ to $\mathbf{y}$ than to go from $\mathbf{x}$ to an intermediate point $\mathbf{z}$ and then from that point to $\mathbf{y}$.

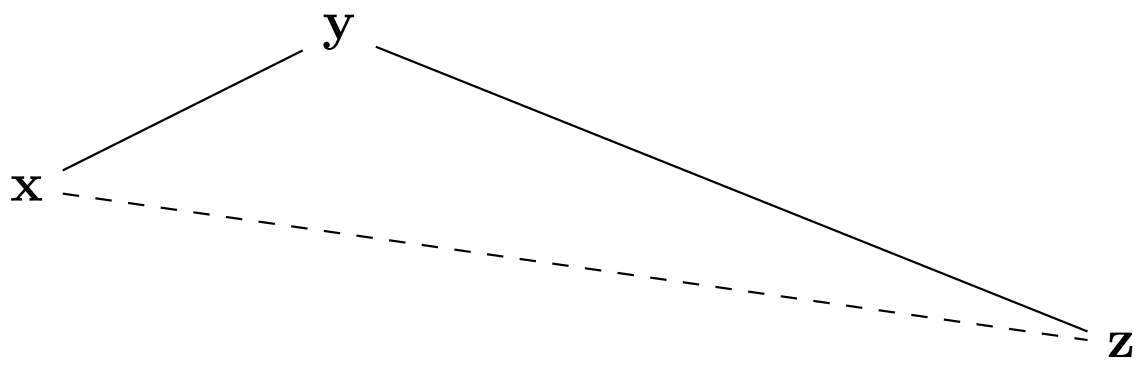
Metric Distances

- More generally a distance $d(\mathbf{x}, \mathbf{y})$ satisfies the triangle inequality if it is of the form

$$d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{y}, \mathbf{z})$$

- Formally the distance is a metric if it is symmetric $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$, non negative, $d(\mathbf{x}, \mathbf{y}) \geq 0$ and $d(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$.
- Note that the Euclidean distance $|\mathbf{x} - \mathbf{y}|$ is a metric – the squared Euclidean distance $|\mathbf{x} - \mathbf{y}|^2$ is not a metric (fails triangle inequality).
- The term 'metric' is often abused in the literature, referring to any 'distance' measure – be aware!

Exploiting the triangle inequality



- If $d(\mathbf{x}, \mathbf{y}) \leq \frac{1}{2} d(\mathbf{z}, \mathbf{y})$, then $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z})$, meaning that we do not need to compute $d(\mathbf{x}, \mathbf{z})$.
- To see this consider

$$d(\mathbf{y}, \mathbf{z}) \leq d(\mathbf{y}, \mathbf{x}) + d(\mathbf{x}, \mathbf{z})$$

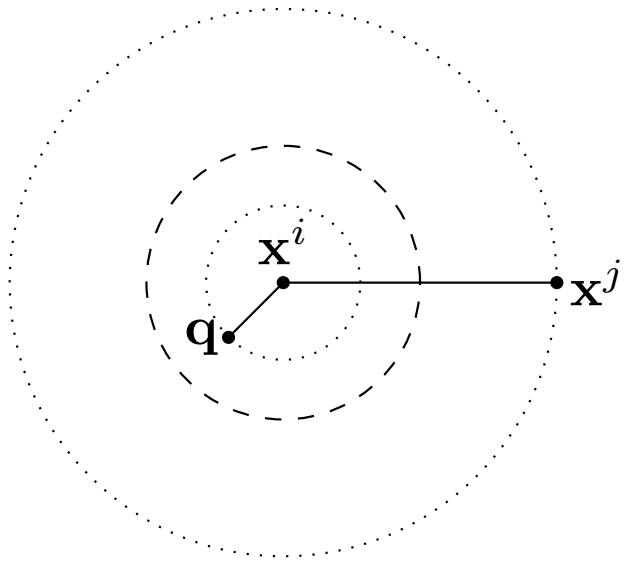
- If we are in the situation that $d(\mathbf{x}, \mathbf{y}) \leq \frac{1}{2} d(\mathbf{z}, \mathbf{y})$, then we can write

$$2d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{y}, \mathbf{x}) + d(\mathbf{x}, \mathbf{z})$$

and $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z})$.

Exploiting the triangle inequality

In the nearest neighbour context, we can infer that if $d(\mathbf{q}, \mathbf{x}^i) \leq \frac{1}{2} d(\mathbf{x}^i, \mathbf{x}^j)$ then $d(\mathbf{q}, \mathbf{x}^i) \leq d(\mathbf{q}, \mathbf{x}^j)$:



The dashed circle represents points $\mathbf{q}'$ for which $d(\mathbf{q}', \mathbf{x}^i) = \frac{1}{2} d(\mathbf{x}^i, \mathbf{x}^j)$. If we know that $\mathbf{x}^i$ is close to $\mathbf{q}$, but that $\mathbf{x}^i$ and $\mathbf{x}^j$ are not close, namely $d(\mathbf{q}, \mathbf{x}^i) \leq \frac{1}{2} d(\mathbf{x}^i, \mathbf{x}^j)$, then we can infer that $\mathbf{x}^j$ will not be closer to $\mathbf{q}$ than $\mathbf{x}^i$, *i.e.* $d(\mathbf{q}, \mathbf{x}^i) \leq d(\mathbf{q}, \mathbf{x}^j)$.

Orchard's Algorithm

- Precompute all the distance pairs $d_{ij} \equiv d(\mathbf{x}^i, \mathbf{x}^j)$ in the dataset.
- Given these distances, for each i we can then compute an ordered list $\mathcal{L}^i = \left\{j_1^i, j_2^i, \ldots, j_{N-1}^i\right\}$ of those vectors $\mathbf{x}^j$ that are closest to $\mathbf{x}^i$, with $d(\mathbf{x}^i, \mathbf{x}^{j_1^i}) \leq d(\mathbf{x}^i, \mathbf{x}^{j_2^i}) \leq d(\mathbf{x}^i, \mathbf{x}^{j_3^i}) \ldots$.
- We then start with some vector $\mathbf{x}^i$ as our current best guess for the nearest neighbour to $\mathbf{q}$ and compute $d(\mathbf{q}, \mathbf{x}^i)$. We then examine the first element of the list $\mathcal{L}^i$ and consider the following cases:
- BOUND CHECK: If $d(\mathbf{q}, \mathbf{x}^i) \leq \frac{1}{2} d_{i,j_1^i}$ then j_1^i cannot be closer than $\mathbf{x}^i$ to $\mathbf{q}$; furthermore, neither can any of the other members of this list since they automatically satisfy this bound as well. In this fortunate situation, $\mathbf{x}^i$ must be the nearest neighbour to $\mathbf{q}$.
- If $d(\mathbf{q}, \mathbf{x}^i) \not\leq \frac{1}{2} d_{i,j_1^i}$ then we compute $d(\mathbf{q}, \mathbf{x}^{j_1^i})$. If $d(\mathbf{q}, \mathbf{x}^{j_1^i}) < d(\mathbf{q}, \mathbf{x}^i)$ we have found a better candidate $i' \equiv j_1^i$ than our current best guess, and we jump to the start of the new list $\mathcal{L}^{i'}$. Otherwise we move down the current list and consider j_2^i in the BOUND CHECK step above.

Orchard's Algorithm

Cost

- Precomputation of distance matrix: $O\left(DN^2\right)$
- Evaluating each member in the list: $O\left(D\right)$
- Need to examine $M < N$ members in the lists.

Comments

- Orchard's algorithm can work well in low dimensional cases, avoiding the calculation of many distances.
- It requires however a potentially very time consuming one-time calculation of all point to point distances.
- The storage of this inter-point distance matrix can be prohibitive.

Approximating and Eliminating Search Algorithm (AESA)

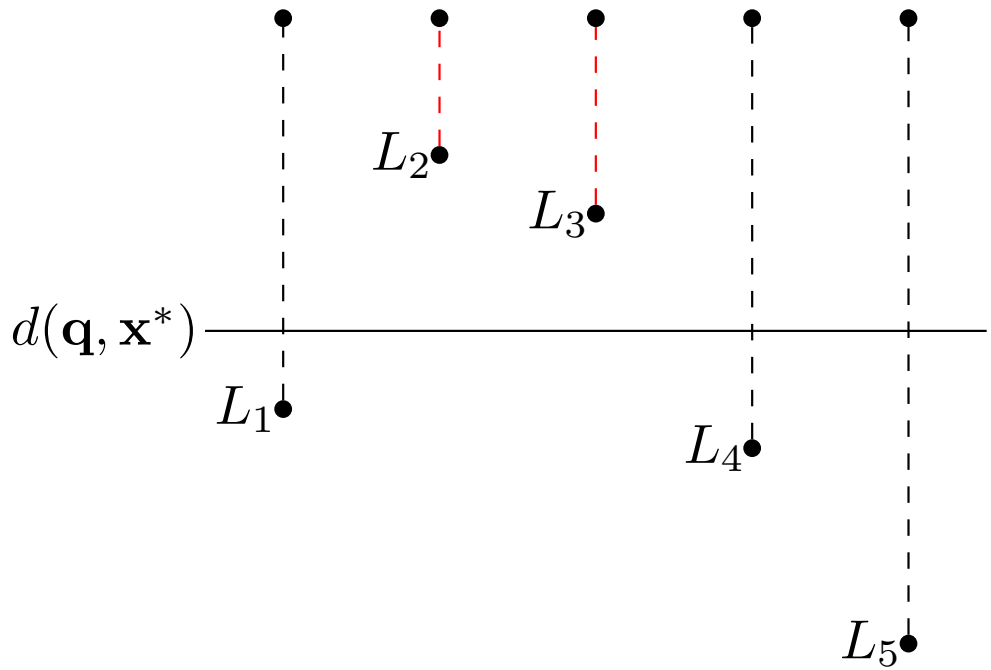
- The triangle inequality can be used to form a lower bound

$$d(\mathbf{q}, \mathbf{x}^j) \geq d(\mathbf{q}, \mathbf{x}^i) - d(\mathbf{x}^i, \mathbf{x}^j)$$

- Define $\mathcal{I}$ to be the set of datapoints for which $d(\mathbf{q}, \mathbf{x}^i)$, $i \in \mathcal{I}$ has already been computed. One can then maximise the lower bounds to find the tightest lower bound on all other $d(\mathbf{q}, \mathbf{x}^j)$

$$d(\mathbf{q}, \mathbf{x}^j) \geq \max_{i \in \mathcal{I}} \left\{ d(\mathbf{q}, \mathbf{x}^i) - d(\mathbf{x}^i, \mathbf{x}^j) \right\}$$

- All datapoints $\mathbf{x}^j$ whose lower bound is greater than the current best nearest neighbour distance can then be eliminated. One may then select the next (non-eliminated) candidate datapoint $\mathbf{x}^j$ corresponding to the lowest bound and continue, updating the bound and eliminating.



We can eliminate datapoints $\mathbf{x}^2$ and $\mathbf{x}^3$ since their distance to the query is greater than the current best candidate distance $d(\mathbf{q}, \mathbf{x}^*)$. After eliminating these points, we use the datapoint with the lowest bound to suggest the next candidate, in this case $\mathbf{x}^5$.

AESA Algorithm: Cost

- Precomputation of distance matrix: $O\left(DN^2\right)$
- Evaluating the bound for all M remaining datapoints: $O\left(M(N-M)\right)$
- Need to compute $M < N$ bounds.

Using 'buoys'

- Both Orchard's algorithm and AESA can significantly reduce the number of distance calculations required.
- However, we pay an $O\left(N^2\right)$ storage cost. For very large datasets, this storage cost is likely to be prohibitive.
- (More strictly, we can actually compute the distances $d(\mathbf{x}^i, \mathbf{x}^j)$ 'on the fly' when they are required. For a single query this may be effective; however for many different queries, we would end up recomputing these distances – hence we may as well store them.)
- Given the difficulty in storing $d_{i,j}$, an alternative is to consider the distances between the training points and a smaller number of strategically placed 'buoys' (also called 'pivots' or 'basis vectors'), $\mathbf{b}^1, \ldots, \mathbf{b}^B$, $B < N$.
- These buoys can be either a subset of the original datapoints, or new positions.

Pre-elimination using Buoys

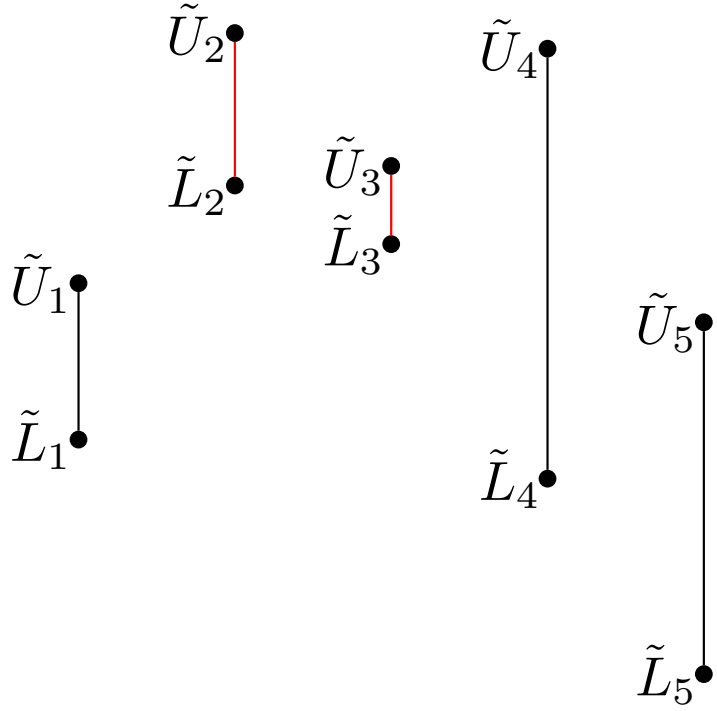
- Given the buoys, the triangle inequality gives the following upper and lower bounds on the distance from the query to each datapoint:

$$d(\mathbf{q}, \mathbf{x}^n) \geq \max_m \{d(\mathbf{q}, \mathbf{b}^m) - d(\mathbf{b}^m, \mathbf{x}^n)\} \equiv \tilde{L}_n$$

$$d(\mathbf{q}, \mathbf{x}^n) \leq \min_m \{d(\mathbf{q}, \mathbf{b}^m) + d(\mathbf{b}^m, \mathbf{x}^n)\} \equiv \tilde{U}_n$$

- We can then immediately eliminate any datapoint whose lower bound is greater than the upper bound of some other datapoint.
- This enables one to 'pre-eliminate' datapoints, at a cost of B distance calculations. (Still takes $O(N)$ operations to do preelimination.)
- The remaining candidates can then be used in the Orchard or AESA algorithms.

Pre-elimination using bounds

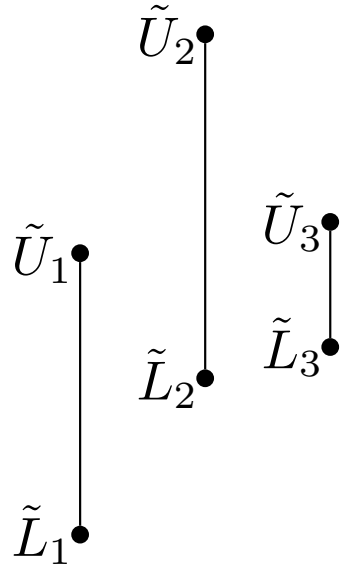


We can eliminate datapoint $\mathbf{x}^2$ since there is another datapoint (either $\mathbf{x}^1$ or $\mathbf{x}^5$) that has an upper bound U that is lower than $\tilde{L}_2$. We can similarly eliminate $\mathbf{x}^3$.

AESA with buoys

- In place of the exact distances to the datapoints, an alternative is to relabel the datapoints according to $\tilde{L}_n$, with lowest distance first $\tilde{L}_1 \leq \tilde{L}_2, \ldots \leq \tilde{L}_N$.
- We can then compute the distance $d(\mathbf{q}, \mathbf{x}^1)$ and compare this to $\tilde{L}_2$. If $d(\mathbf{q}, \mathbf{x}^1) \leq \tilde{L}_2$ then $\mathbf{x}^1$ must be the nearest neighbour, and the algorithm terminates. Otherwise we move on to the next candidate $\mathbf{x}^2$.
- If this datapoint has a lower distance than our current best guess, we update our current best guess accordingly. We then move on to the next candidate in the list and continue. If we reach a candidate in the list for which $d(\mathbf{q}, \mathbf{x}^{best}) \leq \tilde{L}_m$ the algorithm terminates. This algorithm is also called 'linear' AESA.
- The gain here is that the storage costs are reduced to $O(NB)$ since we only now need to pre-compute the distances between the buoys and the dataset vectors. By choosing $B \ll N$, this can be a significant saving. The loss is that, because we are now not using the true distance, but a bound, that we may need more distance calculations $d(\mathbf{q}, \mathbf{x}^i)$.

Linear AESA

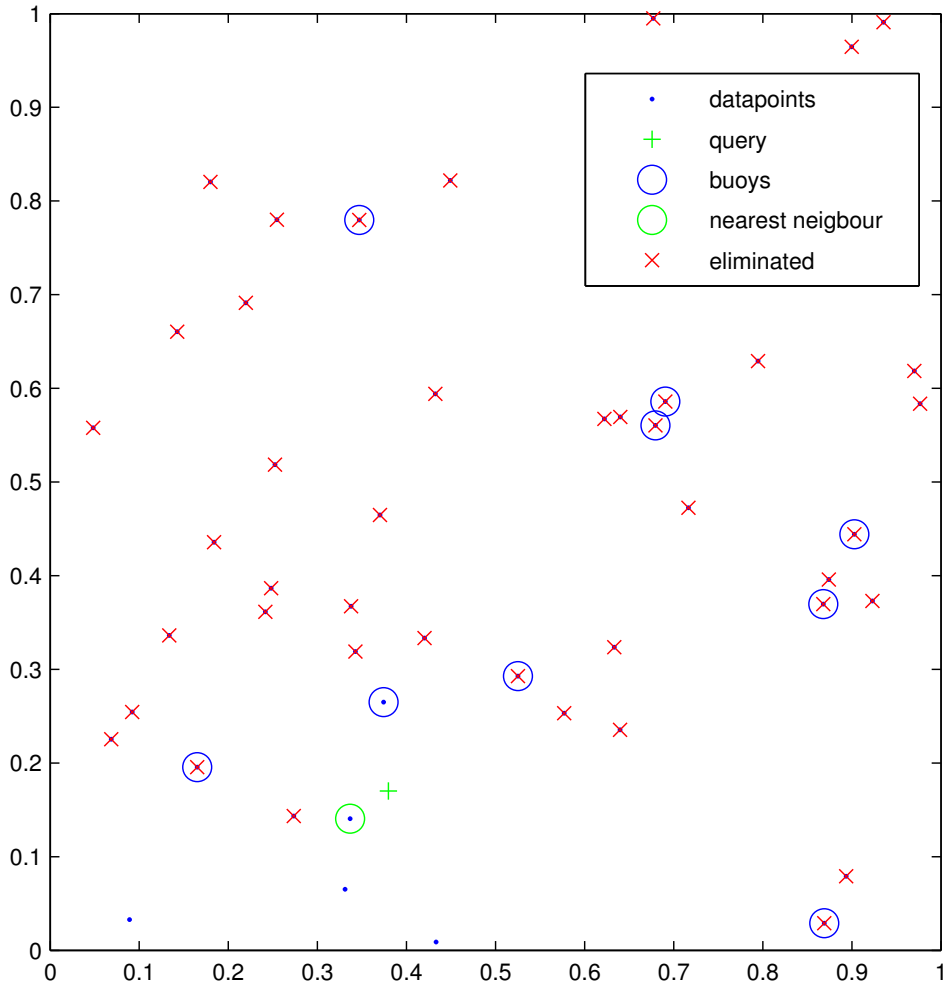


- The lower bounds of non-eliminated datapoints relabelled such that $\tilde{L}_1 \leq \tilde{L}_2 \leq \ldots$.
- In 'linear' AESA we use these lower bounds to order the search for the nearest neighbour, starting with $\mathbf{x}^1$.
- If we get to a bound where $\tilde{L}_m$ is greater than our current best distance, then all remaining distances must be greater than our current best distance, and the algorithm terminates.

Orchard with buoys

- One can also use buoys to replace the exact distance $d(\mathbf{x}^i, \mathbf{x}^j)$ in the Orchard algorithm with an upper bound $d(\mathbf{x}^i, \mathbf{b}) - d(\mathbf{x}^j, \mathbf{b})$.
- However, one can show that AESA-buoys (linear AESA) dominates Orchard-buoys in terms of the number of distance calculations $d(\mathbf{q}, \mathbf{x}^i)$ required (see main text).
- No obvious advantage of this approach since computing the upper bounds on the distance requires an $O(N)$ computation.

Elimination using buoys



- All points except for the query are datapoints.
- Using buoys, we can pre-eliminate (crossed datapoints) a large number of the datapoints from further consideration.

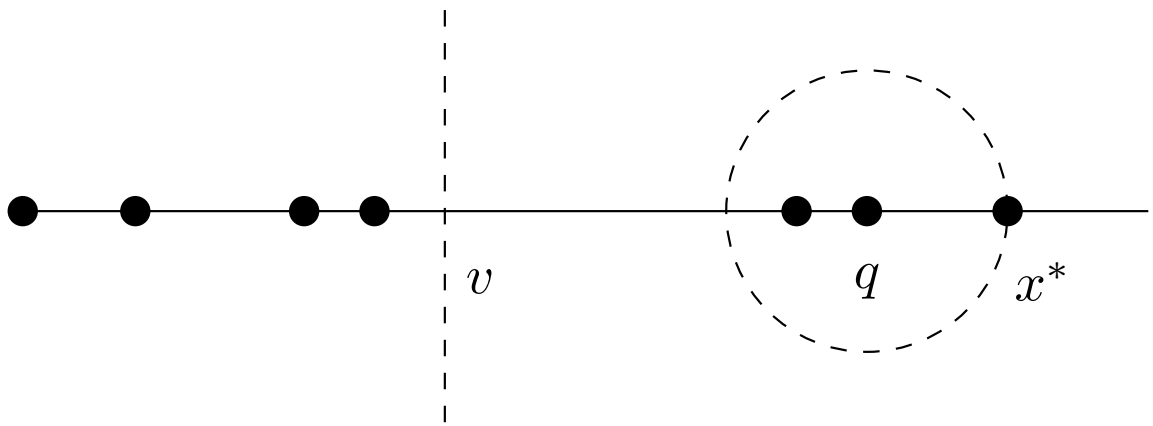
KD Tree

KD trees

- K-dimensional trees are a way to form a hierarchical partition of the space that can be used to help speed up search.
- This is a form of 'spatial data structure'
- An advantage over the Orchard and AESA approaches is that the storage cost of the tree is only $O(N)$.
- Also, only in the worst case are $O(N)$ operations required for a new query.
- Before introducing the tree, we'll discuss the basic idea on which the potential speed-up is based.

Basic idea in one-dimension

- If we consider first one-dimensional data $x^n, n = 1, \ldots, N$ we can partition the data into points that have value less than a chosen value v, and those with a value greater than this.
- If the distance of the current best candidate x^* to the query point q is smaller than the distance of the query to v, then points to the left of v cannot be the nearest neighbour.



One dimension case

Consider a query point that is in the right space, $q > v$ and a candidate x that is in the left space, $x < v$, then

$$(x-q)^2 = (x-v+v-q)^2 = (x-v)^2 + 2\underbrace{(x-v)}_{\leq 0}\underbrace{(v-q)}_{\leq 0} + (v-q)^2 \geq (v-q)^2$$

- Let the distance of the current best candidate to the query be $\delta^2 \equiv (x^* - q)^2$.
- Then if $(v-q)^2 \geq \delta^2$ it follows that all points in the left space are further from q than x^*.

K dimensional case

- In the more general K dimensional case, consider a query vector $\mathbf{q}$.
- Let's imagine that we have partitioned the datapoints into those with first dimension x_1 less than a defined value v (to its 'left'), and those with a value greater or equal to v (to its 'right'):

$$\mathcal{L} = \{\mathbf{x}^n : x_1^n < v\}, \qquad \mathcal{R} = \{\mathbf{x}^n : x_1^n \geq v\}$$

- Let's also say that our current best nearest neighbour candidate is $\mathbf{x}^i \in \mathcal{R}$ and that this point has squared Euclidean distance $\delta^2 = \left(\mathbf{q} - \mathbf{x}^i\right)^2$ from $\mathbf{q}$.
- The squared Euclidean distance of any datapoint $\mathbf{x} \in \mathcal{L}$ to the query is

$$(\mathbf{x} - \mathbf{q})^2 = \sum_k (x_k - q_k)^2 \geq (x_1 - q_1)^2 \geq (v - q_1)^2$$

- If $(v - q_1)^2 \geq \delta^2$, then $(\mathbf{x} - \mathbf{q})^2 > \delta^2$.
- That is, all points in $\mathcal{L}$ must be further from $\mathbf{q}$ than the current best point $\mathbf{x}^i$.
- On the other hand, if $(v - q_1)^2 < \delta^2$, then it is possible that some point in $\mathcal{L}$ might be closer to $\mathbf{q}$ than our current best nearest neighbour candidate, and we need to check these points.

Constructing the tree

- For N datapoints, the tree consists of N nodes. Each node contains a datapoint, along with the axis along which the data is split.
- We first need to define a routine

```
[middata leftdata rightdata]=splitdata(x,axis)
```

that splits data along a specified dimension (axis).

`middata` We first form the set of scalar values that correspond to the `axis` components of `x`. These are sorted and `middata` is the datapoint closest to the median of the data.

`leftdata` These are the datapoints 'to the left' of the `middata` datapoint.

`rightdata` These are the datapoints 'to the right' of the `middata` datapoint.

Splitting example

If the datapoints are $(2,3),(5,4),(9,6),(4,7),(8,1),(7,2)$ and we split along dimension 1, then we would have:

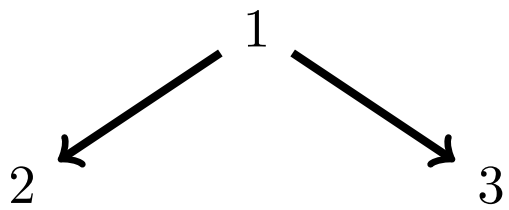
```
>>x =

     2     5     9     4     8     7
     3     4     6     7     1     2


>> [middata leftdata rightdata]=splitdata(x,1)
middata =

     7
     2

leftdata =
     2     4     5
     3     7     4

rightdata =
     8     9
     1     6
```

Tree construction (top layer)

- We start with `node(1)` and create a temporary storage `node(1).data` that contains the complete dataset.
- We then call

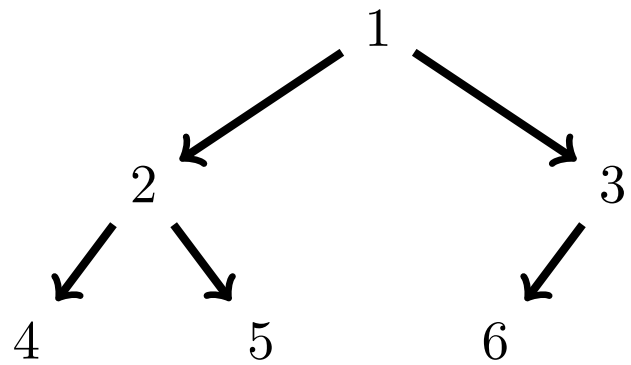
  ```
  [middata leftdata rightdata]=splitdata(node(1).data,1)
  ```

  forming `node(1).x=middata`.
- We now form two child nodes and populate them with data `node(2).data=leftdata; node(3).data=rightdata`.
- We store also in node(1).axis which axis was used to split the data.
- The temporary data `node(1).data` can now be removed.



Tree construction (next layer)

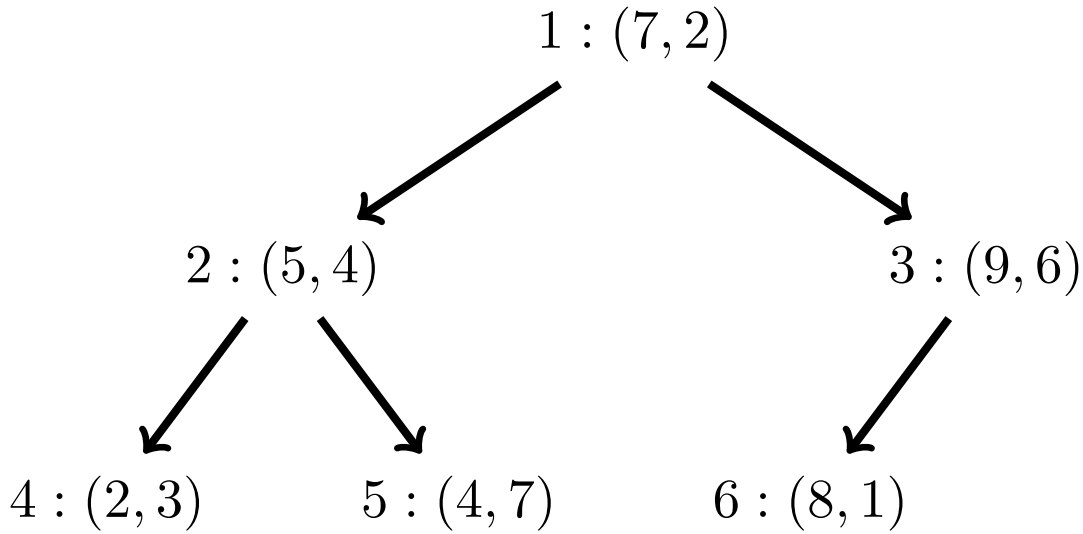
- We now move down to the second layer, and split along the next dimension for nodes in this layer.
- We then go through each of the nodes and split the corresponding data, forming a layer beneath.
- If `leftdata` is empty, then the corresponding child node is not formed, and similarly if `rightdata` is empty.



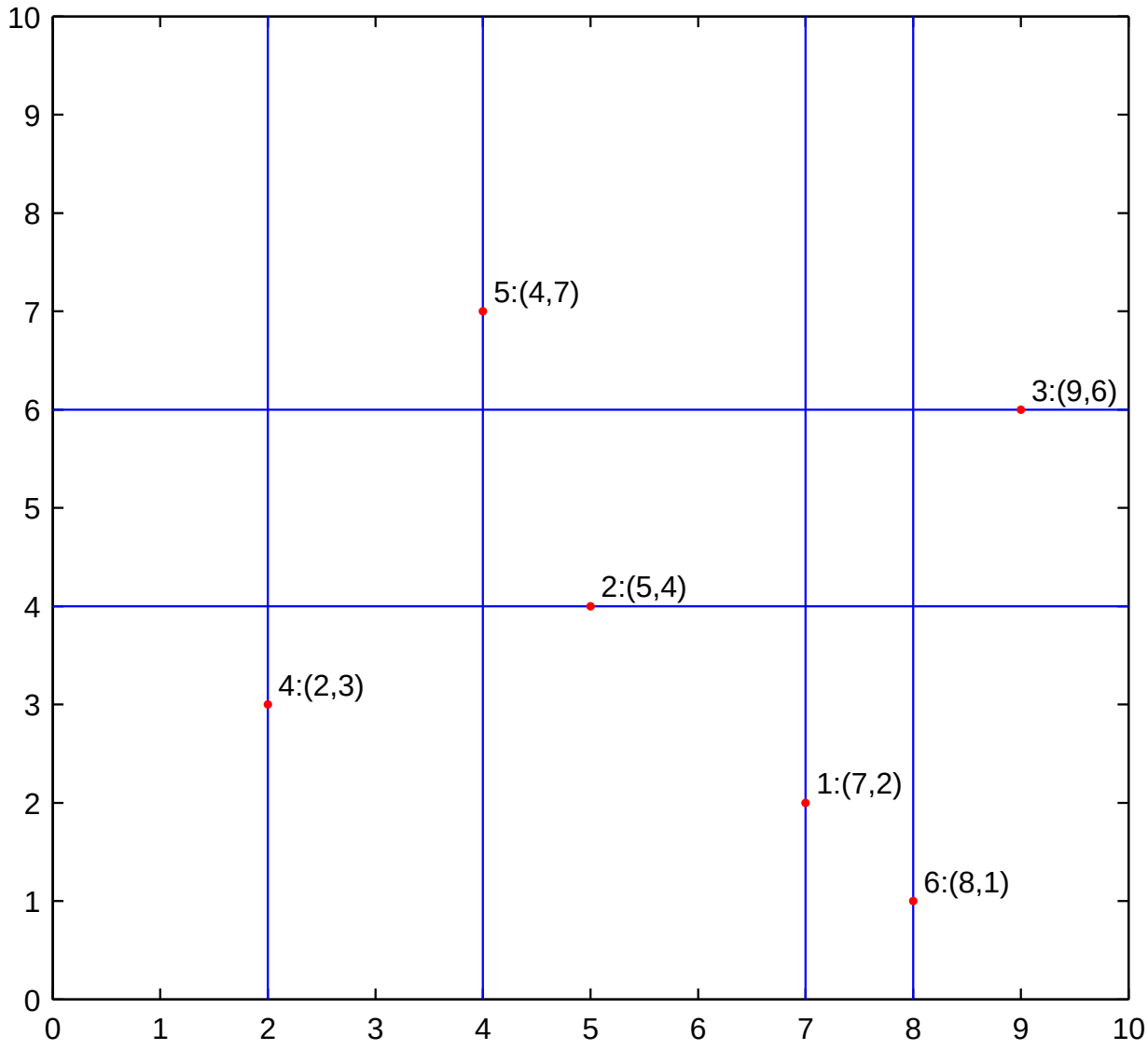
In this way, the top layer splits the data along axis 1, and then the nodes 2 and 3 in the second layer split the data along dimension 2.

- The nodes in the third layer don't require any more splitting since they contain single datapoints.

Final KD Tree



Visualisation



The KD tree partitions the space into hyperrectangles.

Nearest Neighbour search $q = (9, 2)$

- For a query point $\mathbf{q}$ we first find the leaf node of the tree by traversing the tree from the root, and seeing if the corresponding components of $\mathbf{q}$ is 'to the left' or 'to the right' of the current tree node.
- For the above tree, for the query $\mathbf{q} = (9, 2)$, we would first consider the value 9 (since the first layer splits along dimension 1).
- Since 9 is 'to the right' of 7 (the first dimension of node 1), we go now to node 3.
- In this case, there is a unique child of node 3, so we continue to the leaf, node 6.

Nearest Neighbour search $q = (9, 2)$

- Node 6 now represents our first guess for the nearest neighbour.
- This has distance $(9-8)^2 + (2-1)^2 = 2 \equiv \delta^2$ from the query. We set this to our current best guess of the nearest neighbour and corresponding distance.
- We then go up the tree, to the parent, node 3.
- We check if this is a better neighbour. It has distance $(9-9)^2 + (2-6)^2 = 16$, so this is worse.
- Since there are no other children to consider, we move up another level, this time to node 1. This has distance $(9-7)^2 + (2-2)^2 = 4$, which is also worse than our current best.
- We now need to consider if there are any nodes in the other branch of the tree containing nodes 2,4,5, that could be better than our current best.
- We know that for all nodes 2,4,5 they have first dimensions with value less than 7.
- We can then check if the corresponding datapoints are necessarily further than our current best guess by checking if $(v - q_1)^2 > \delta^2$, namely if $(7-9)^2 > 2$. Since this is true, it must be that all the points in nodes 2,4,5 are further away from the query than our current best guess.
- At this point the algorithm terminates, having examined only 3 datapoints in which to find the nearest neighbour, rather than all 6.

$q = (6, 5)$

- With this query, we go down the tree to node (4,7)
- This gives our initial $\delta^2 = (6-4)^2 + (5-7)^2 = 8$
- Go up to (5,4). This has distance from the query $(6-5)^2 + (5-4)^2 = 2$. This is therefore a better neighbour and forms our new best guess for the nearest neighbour.
- What about (2,3)? Here we have $v = 4$ and check $v - q_2)^2 = (4-5)^2 = 1$. This is less than δ^2 so we have to check (2,3). This has distance $(6-2)^2 + (5-3)^2 = 20$. This is bigger than our current δ^2 (2) so we reject this and move up the tree to (7,2).
- This has distance $(7-6)^2 + (2-5)^2 = 10$ which is larger than our current δ^2 (2) so we reject this.
- Do we need to check the right branch of the tree? Here $v = 7$ and $(v - q_1)^2 = (7-6)^2 = 1$. This is less than δ^2 so we have to check the right branch.
- Continuing in this way, we actually end up having to explicitly check all nodes.
- This is an example of a worst-case situation in which the computation is as bad (even slightly worse) than just calculating all the distances.