

# The Nearest Neighbor Algorithm

Sîrbu Matei-Dan

Universitatea Transilvania din Braşov  
Facultatea de Matematică şi Informatică

17 decembrie 2020

# The Nearest Neighbor Algorithm

## ■ Hypothesis Space

- variable size
- deterministic
- continuous parameters

## ■ Learning Algorithm

- direct computation
- lazy

# Nearest Neighbor Algorithm

- Store all of the training examples
- Classify a new example  $\mathbf{x}$  by finding the training example  $\langle \mathbf{x}_i, y_i \rangle$  that is nearest to  $\mathbf{x}$  according to Euclidean distance:

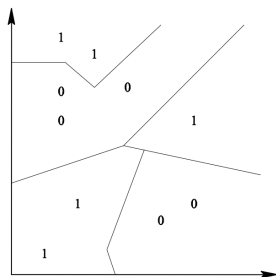
$$\|\mathbf{x} - \mathbf{x}_i\| = \sqrt{\sum_j (x_j - x_{ij})^2}$$

guess the class  $\hat{y} = y_i$ .

- Efficiency trick: squared Euclidean distance gives the same answer but avoids the square root computation

$$\|\mathbf{x} - \mathbf{x}_i\|^2 = \sum_j (x_j - x_{ij})^2$$

# Decision Boundaries: The Voronoi Diagram



- Nearest Neighbor does not explicitly compute decision boundaries. However, the boundaries form a subset of the Voronoi diagram of the training data
- Each line segment is equidistant between two points of opposite class. The more examples that are stored, the more complex the decision boundaries can become.

# Nearest Neighbor depends critically on the distance metric

## ■ Normalize Feature Values:

- All features should have the same range of values (e.g.  $[-1, +1]$ ). Otherwise, features with larger ranges will be treated as more important

## ■ Remove Irrelevant Features:

- Irrelevant or noisy features add random perturbations to the distance measure and hurt performance

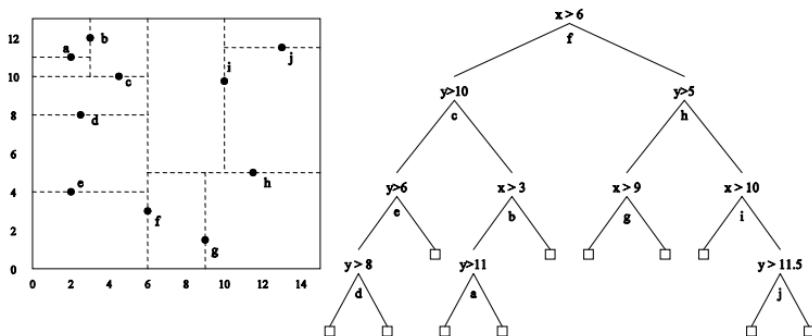
## ■ Learn a Distance Metric:

- One approach: weight each feature by its mutual information with the class. Let  $w_j = l(x_j; y)$ . Then  $d(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^n w_j (x_j - x'_j)^2$
- Another approach: use the Mahalanobis distance:  
$$D_M(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^\top \Sigma^{-1} (\mathbf{x} - \mathbf{x}')$$

## ■ Smoothing:

- Find the  $k$  nearest neighbors and have them vote. This is especially good when there is noise in the class labels.

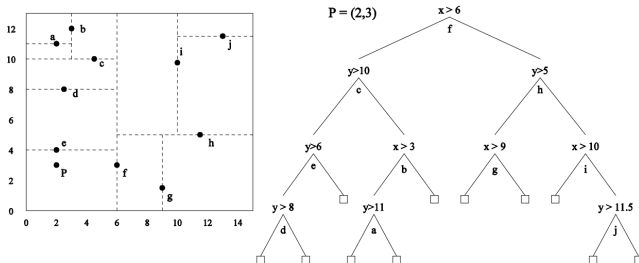
- A  $k$ -d tree is similar to a decision tree except that we split using the median value along the dimension having the highest variance. Every internal node stores one data point, and the leaves are empty



# log time queries with $k$ -d trees

```
KDTree root;
Node NearestNeighbor(Point P)
{
    PriorityQueue PQ; // minimizing queue
    float bestDist = infinity; // smallest distance seen so far
    Node bestNode; // nearest neighbor so far
    PQ.push(root, 0);
    while (!PQ.empty()) {
        (node, bound) = PQ.pop();
        if (bound >= bestDist) return bestNode.p;
        float dist = distance(P, node.p);
        if (dist < bestDist) {bestDist = dist; bestNode = node;}
        if (node.test(P)) {
            PQ.push(node.left, P[node.feats] - node.thresh);
            PQ.push(node.right, 0);
        }
        else {
            PQ.push(node.left, 0);
            PQ.push(node.right, node.thresh - P[node.feats]);
        }
    } // while
    return bestNode.p;
} // NearestNeighbor
```

# Example



New Distance	Best Distance	Best node	Priority Queue
none	$\infty$	none	$(f, 0)$
4.00	4.00	$f$	$(c, 0)(h, 4)$
7.61	4.00	$f$	$(e, 0)(h, 4)(b, 7)$
1.00	1.00	$e$	$(d, 1)(h, 4)(b, 7)$

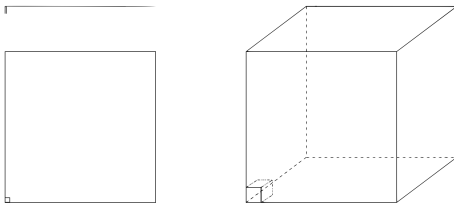
- This is a form of A\* search using the minimum distance to a node as an underestimate of the true distance



- Consider several distance measures:  $D_1, D_2, \dots, D_n$  where  $D_{i+1}$  is more expensive to compute than  $D_i$
- Calibrate a threshold  $N_i$  for each filter using the training data
- Apply the nearest neighbor rule with  $D_i$  to compute the  $N_i$  nearest neighbors
- Then apply filter  $D_{i+1}$  to those neighbors and keep the  $N_{i+1}$  nearest, and so on

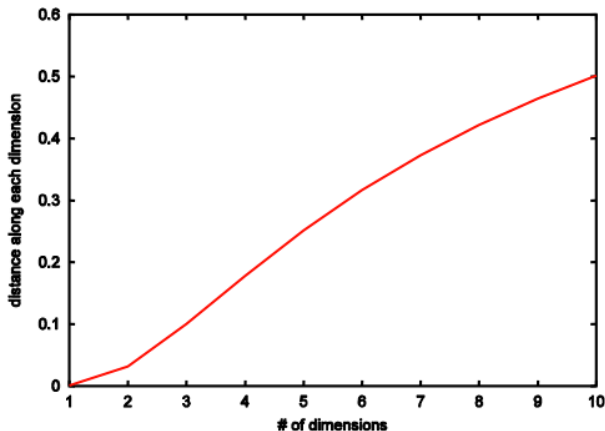
# The Curse of Dimensionality

- Nearest neighbor breaks down in high-dimensional spaces, because the “neighborhood” becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm. Suppose our query point is at the origin.
- Then on the 1-dimensional line, we must go a distance of  $5/5000 = 0.001$  on the average to capture the 5 nearest neighbors
- In 2 dimensions, we must go  $\sqrt{0.001}$  to get a square that contains 0.001 of the volume.
- In  $D$  dimensions, we must go  $(0.001)^{1/d}$



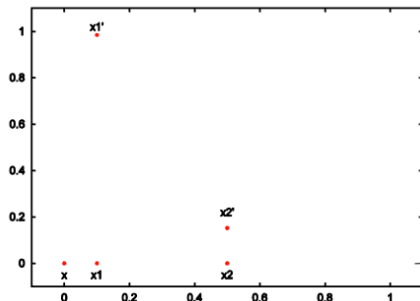
# The Curse of Dimensionality (2)

- With 5000 points in 10 dimensions, we must go 0.501 distance along each attribute in order to find the 5 nearest neighbors



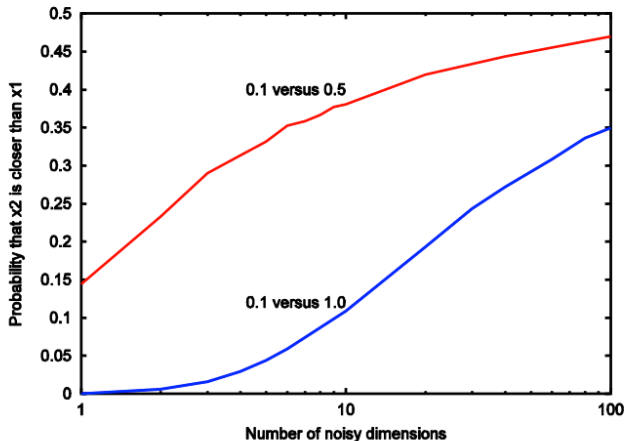
# The Curse of Noisy/Irrelevant Features

- NNbr also breaks down when the data contains irrelevant, noisy features.
- Consider a 1D problem where our query  $x$  is at the origin, our nearest neighbor is  $x_1$  at 0.1, and our second nearest neighbor is  $x_2$  at 0.5.
- Now add a uniformly random noisy feature. What is the probability that  $x'_2$  will now be closer to  $x$  and  $x'_1$ ? Approximately 0.15.



# The Curse of Noise (2)

## Location of $x_1$ versus $x_2$



# Nearest Neighbor Evaluation

Criterion	Perc	Logistic	LDA	Trees	Nets	NNbr
Mixed data	no	no	no	yes	no	no
Missing values	no	no	yes	yes	no	somewhat
Outliers	no	yes	no	yes	yes	yes
Monotone transformations	no	no	no	yes	somewhat	no
Scalability	yes	yes	yes	yes	yes	no
Irrelevant inputs	no	no	no	somewhat	no	no
Linear combinations	yes	yes	yes	no	yes	somewhat
Interpretable	yes	yes	yes	yes	no	no
Accurate	yes	yes	yes	no	yes	no

# Nearest Neighbor Summary

## ■ Advantages

- variable-sized hypothesis space
- learning is extremely efficient and can be online or batch
  - However, growing a good  $k$ -d tree can be expensive
- Very flexible boundaries

## ■ Disadvantages

- distance function must be carefully chosen
- irrelevant or correlated features must be eliminated
- typically cannot handle more than 30 features
- computational costs: memory and classification-time computation