Non-parametric Models
K Nearest Neighbours

Dr. Srijith P K

PARAMETRIC SUPERVISED LEARNING

A parametric algorithm has a **fixed number of parameters**.

A parametric algorithm is **computationally faster**, but makes stronger assumptions about the data; the algorithm may work well if the assumptions turn out to be correct, but it may perform badly if the assumptions are wrong.

A learning model that summarises data with a set of parameters of fixed size (predefined mapped function) (independent of the number of training examples). No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs.

A common example of a parametric algorithm is **Linear Regression, Linear Support Vector Machines, Perceptron, Logistic Regression**.

Parametric methods

- Assume some functional form (Gaussian, Bernoulli, Multinomial, logistic, Linear, Quadratic) for
  - $P(X_i|Y)$ and $P(Y)$ as in Naïve Bayes
  - $P(Y|X)$ as in Logistic, Linear and Nonlinear regression, SVM

- Estimate parameters ($\mu,\sigma^2,\theta,w,\beta$) using MLE/MAP and plug in

- **Pro** – need few data points to learn parameters
- **Con** – Strong distributional assumptions, not satisfied in practice

Parametric vs Nonparametric Models

- *Parametric models* assume some finite set of parameters θ. Given the parameters, future predictions, x, are independent of the observed data, $\mathcal{D}$:

$$P(x|\theta, \mathcal{D}) = P(x|\theta)$$

  therefore θ capture everything there is to know about the data.

- So the complexity of the model is bounded even if the amount of data is unbounded. This makes them not very flexible.

- *Non-parametric models* assume that the data distribution cannot be defined in terms of such a finite set of parameters. But they can often be defined by assuming an *infinite dimensional* θ. Usually we think of θ as a *function*.

- The amount of information that θ can capture about the data $\mathcal{D}$ can grow as the amount of data grows. This makes them more flexible.

Non-Parametric methods

- Typically don't make any distributional assumptions
- As we have more data, we should be able to learn more complex models
- Let number of parameters scale with number of training data

- Today, we will see some nonparametric methods for
  - Density estimation
  - Classification
  - Regression

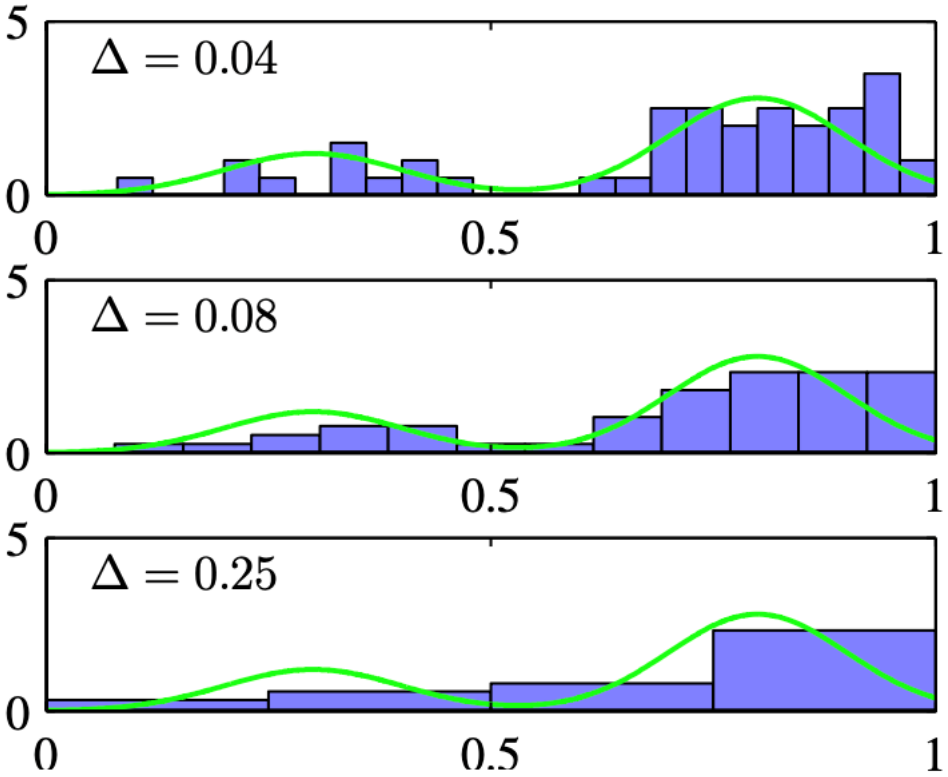
A common example of a non-parametric algorithm is **K-nearest neighbour, Decision Trees, Artificial Neural Networks, Support Vector Machines with Gaussian Kernels.**

Density Estimation

- *Nonparametric* approaches to density estimation that make few assumptions about the form of the distribution
- Process that generates the data is multimodal, Gaussian distribution cannot capture this
- Histogram density models
- Count the number ni of observations of x falling in bin i
- Probability values for each bin

$$p_i = \frac{n_i}{N\Delta_i}$$

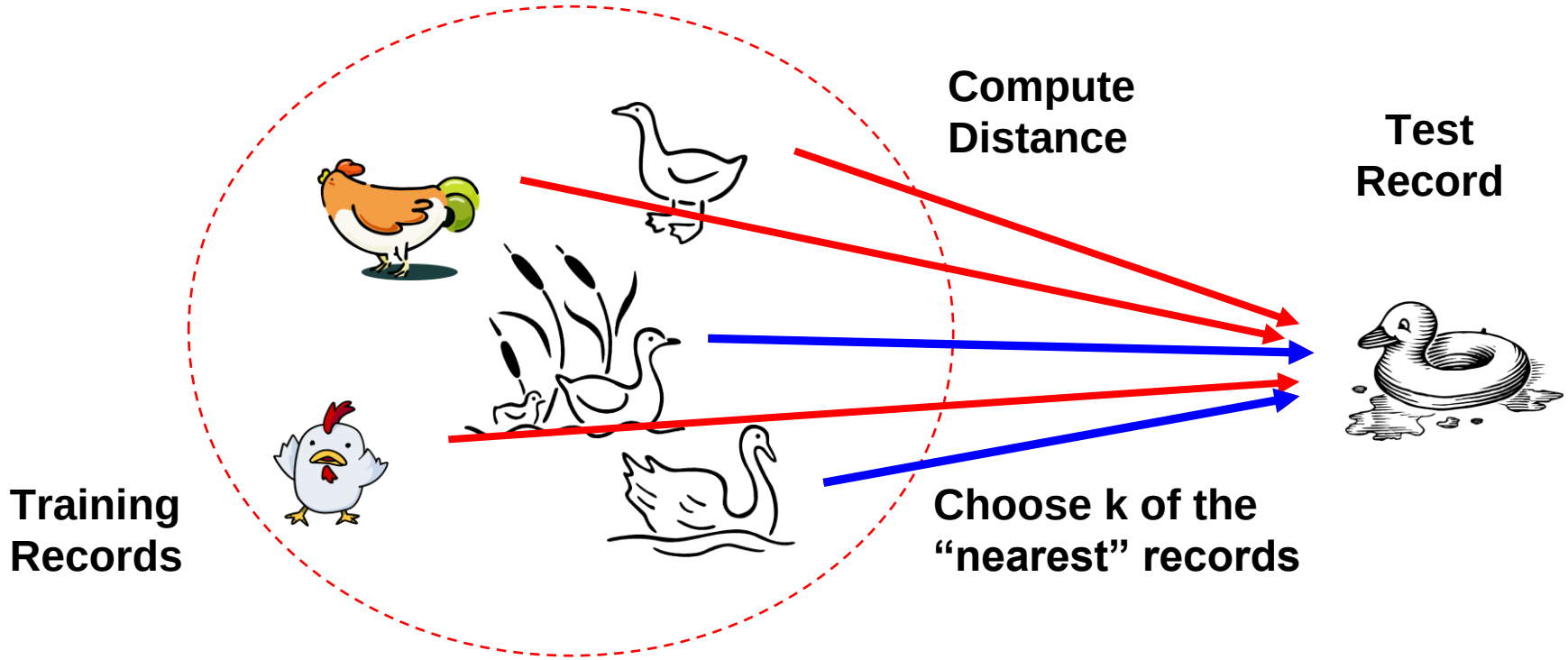
- Δ is very small (top figure), the resulting density model is very spiky
- Δ is too large (bottom figure) then the result is a model that is too smooth
- Model complexity determined by the bin size



Classifiers: kNN

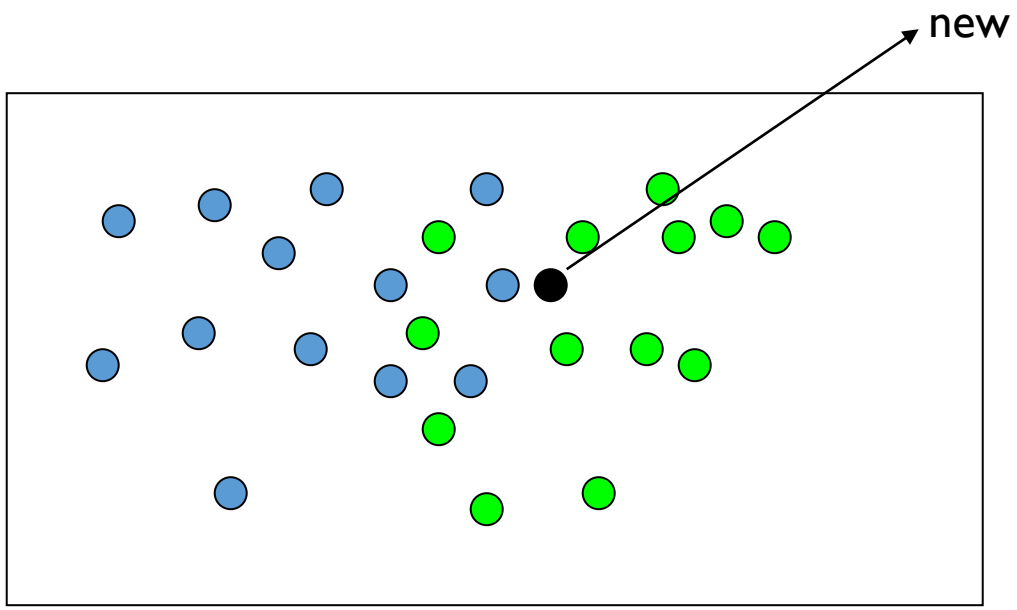
k-Nearest Neighbors

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it's probably a duck



k-Nearest Neighbors

- Majority vote within the k nearest neighbors



K= 1: blue
K= 3: green

k-Nearest Neighbors

- An arbitrary instance is represented by **$(a_1(x), a_2(x), a_3(x),.., a_n(x))$**
  - **$a_i(x)$** denotes features
- Euclidean distance between two instances
  - **$d(x_i, x_j)$=sqrt (sum for r=1 to n $(a_r(x_i) - a_r(x_j))^2$)**
- L_p distance
  - p=2: Euclidean distance
  - p=1: Manhattan distance
  - p =∞: Max distance
  - p= 0: Count non-zero distance

$$\underbrace{\|x\|_p}_{p-Norm} = (\sum_{i}^{n} |x_i|^p)^{\frac{1}{p}} = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

- In case of continuous-valued target function
  - Mean value of k nearest training examples

Other Distance Metrics

- Cosine Distance Metric

$$\rho(\vec{x}_1, \vec{x}_2) = \cos(\angle(\vec{x}_1, \vec{x}_2)) = \frac{\vec{x}_1 \cdot \vec{x}_2}{\|\vec{x}_1\|_2 \ \|\vec{x}_2\|_2}$$

- Edit Distance

x_1 = AAATCCCGTAA

x_2 = AATCGCGTAA

Minimum number of insertions, deletions and mutations needed

$$\rho(x_1, x_2) = 2$$

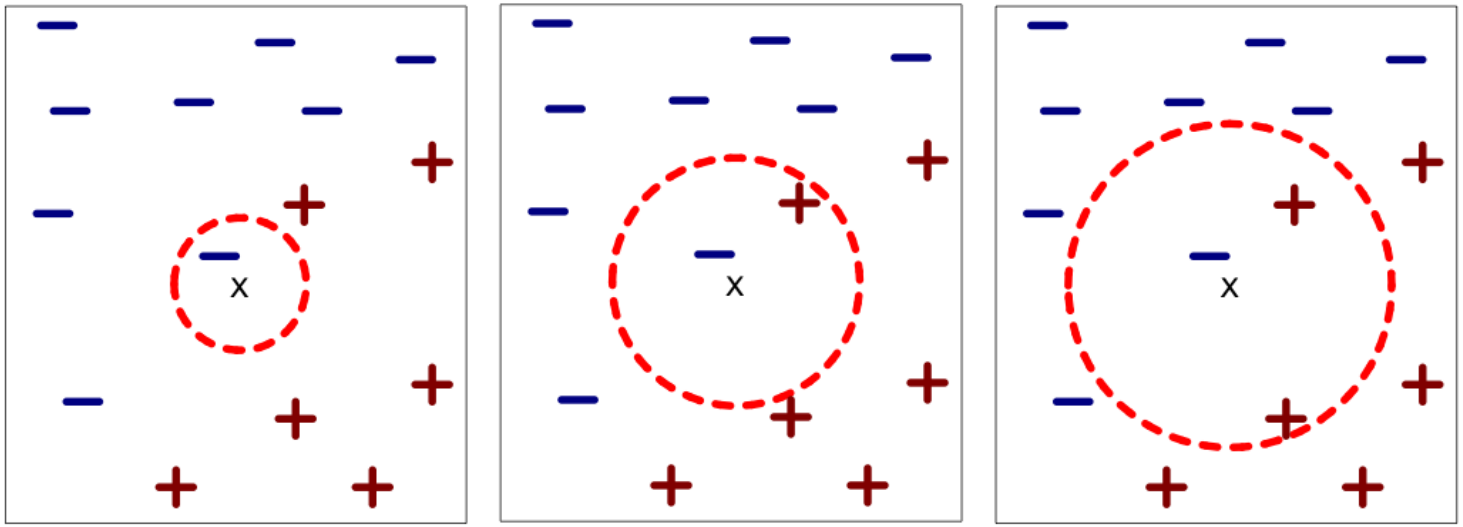
- Hamming distance is a metric for comparing two binary data strings

d(11011001, 10011101) = 2.

"On the Surprising Behavior of Distance Metrics in High Dimensional Space", by Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Kiem. " for a given problem with a fixed (high) value of the dimensionality d, it may be preferable to use lower values of p. This means that the L1 distance metric (Manhattan Distance metric) is the most preferable for high dimensional applications."

k-Nearest Neighbors

- Choosing k is important
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

How to determine k

- Determined experimentally (think cross-validation!)
  - Start with k=1 and use a test set to validate the error rate of the classifier
  - Repeat with k=k+2
  - Choose the value of k for which the error rate is minimum
  - Note: k typically an odd number to avoid ties in binary classification

Pros and Cons

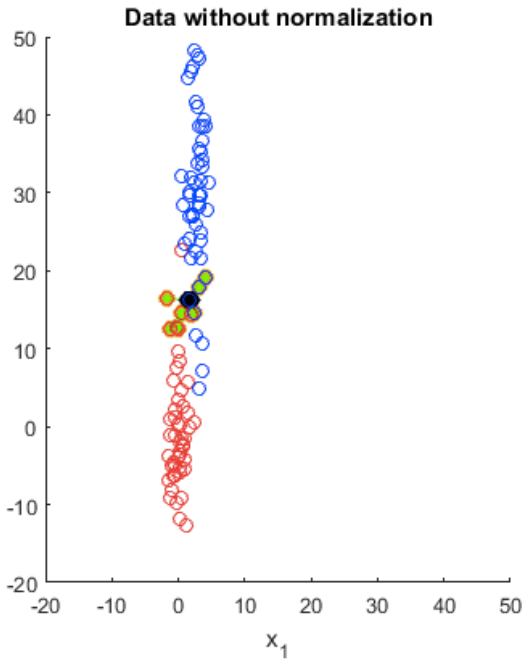
- Pros
  - Highly effective and simple method
  - Trains very fast ("Lazy" learner)
- Cons
  - Curse of dimensionality
    - In higher dimensions, all data points lie on the surface of the unit hypersphere!
  - Closeness in raw measurement space may not be good for the task
  - Storage: all training examples are saved in memory
    - A decision tree or linear classifier is much smaller
  - Slow at query time
    - Can be overcome and presorting and indexing training samples

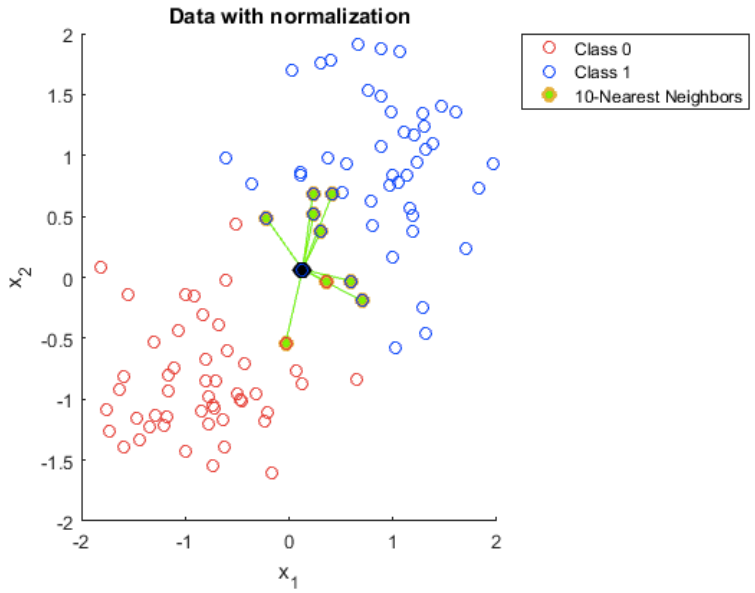
all the volume of the cube is in these spikes!

Improvements

- Distance-Weighted Nearest Neighbors
  - Assign weights to the neighbors based on their 'distance' from the query point (E.g., weight 'may' be inverse square of the distances)

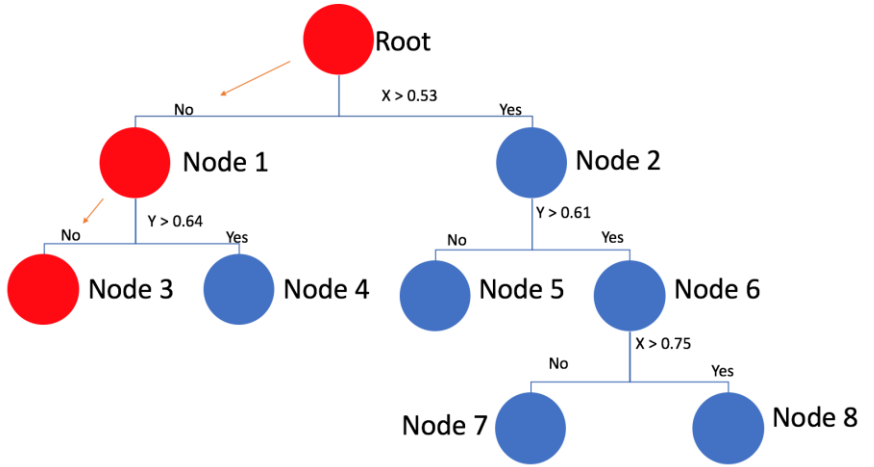
- Scaling (**normalization**) attributes for fair computation of distances

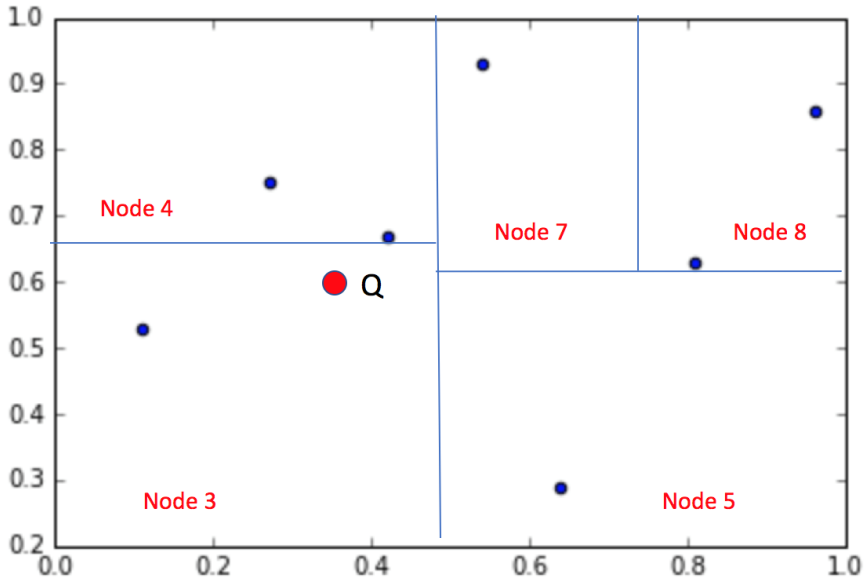




Improvements

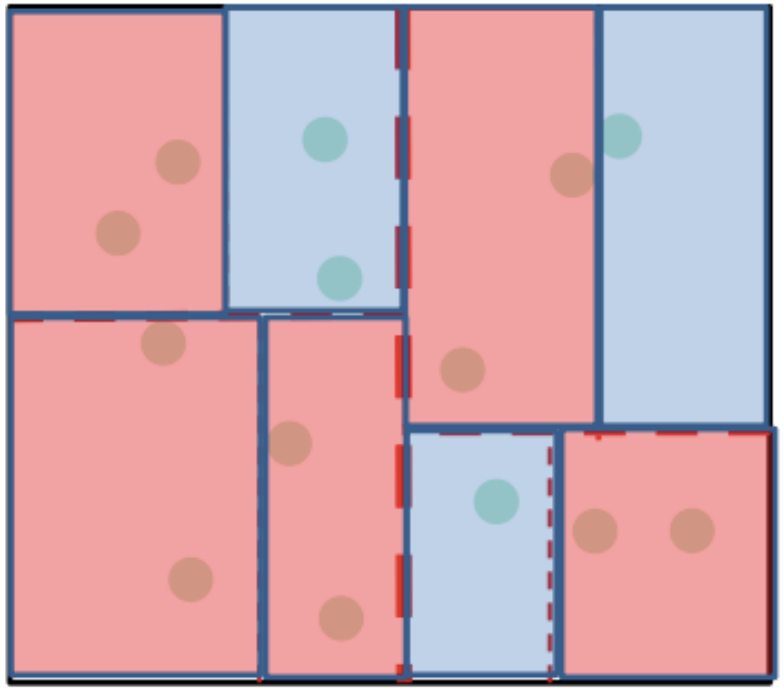
- Finding "close" examples in a large training set quickly
  - E.g. Efficient memory indexing using kd-trees
    - In 1-dimension, can reduce complexity from O(n) to O(log n) – assuming data is sorted
  - Other methods
    - Locality-Sensitive Hashing, Clustering-based methods





Improvements

- Not storing all examples
  - We can label each cell instead and discard the training data



Readings

- Chapters 8, 9, EA Introduction to ML 2nd Edn
- Chapter 14 (Sec 14. 4) + Chapter 2 (Sec 2. 5), Bishop, PRML