Density Estimation

Nearest Neighbors (NN) Classifier

CS 6316 – Machine Learning
Fall 2017

OUTLINE

- Preface
- K-NN Classifiers
- Definition
- Choosing “k”
- K-NN as a ML algorithm
- K-NN vs 1NN
- Feature weighting
- Minimizing EPE – Expected Prediction Error
- Bias-Variance Trade-off

Preface

- Often times we are aware of the underlying densities

- In most situations, however, the true distributions are unknown and must be estimated from data

  - Two approaches are commonplace:
    - Parameter estimation
    - Non-parametric density estimation

Preface

- Parameter estimation
  - Assume a particular form for the density (e.g. Gaussian), so only the parameters (e.g. mean and variance) need to be estimated
    - Maximum Likelihood
    - Bayesian Estimation
- Non-parametric density estimation
  - Assume NO knowledge about the density
    - Kernel Density Estimation
    - K Nearest Neighbor ← *will concentrate on this!*

Preface

- We can divide the large variety of **classification approaches** into roughly three major types:
  1. Discriminative
     - Directly estimate a decision rule/boundary
     - E.g. decision tree (done), SVM
  2. Generative
     - Build a generative statistical model
     - E.g. Bayesian networks
  3. Instance based classifiers
     - Use observation directly (no models)
     - E.g. K nearest neighbors *(this lecture!)*

K Nearest Neighbors

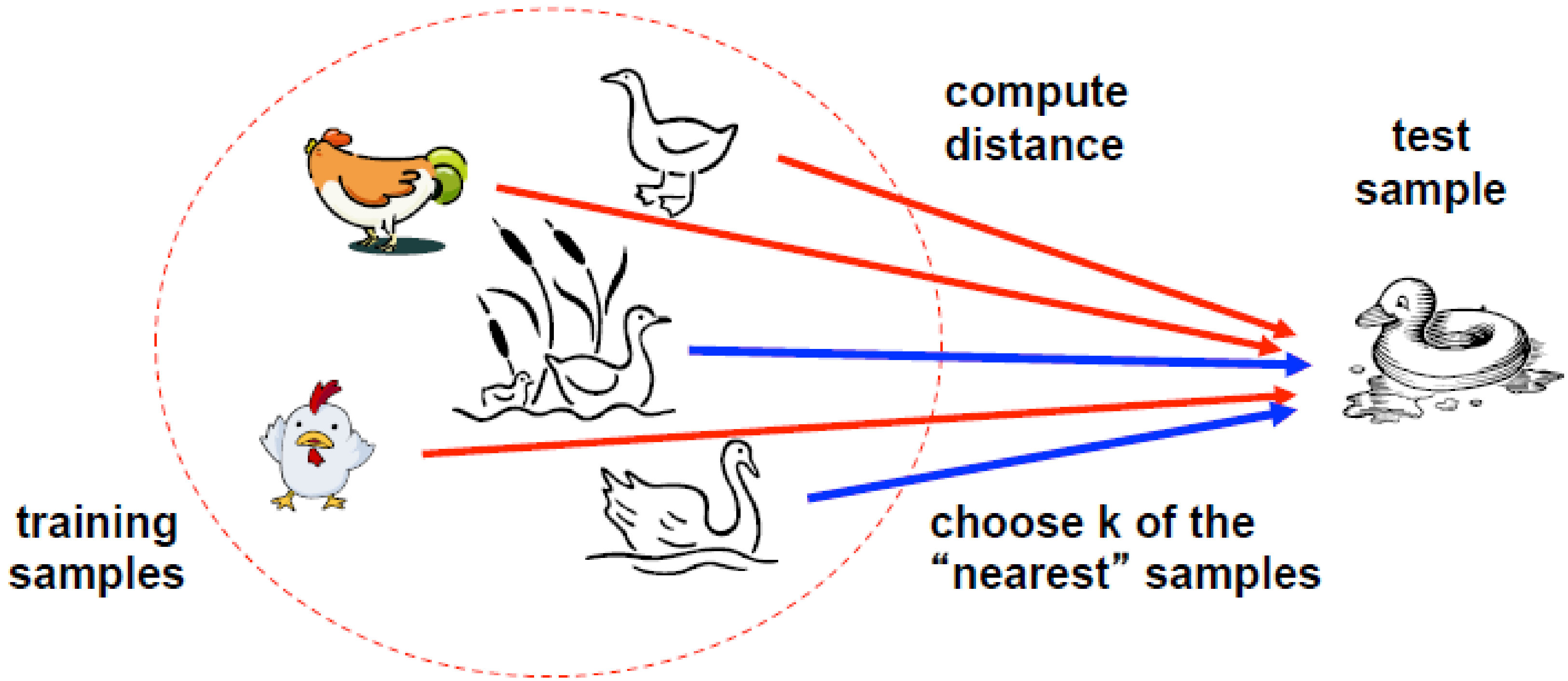
Classifier

Material adapted from Ricardo Osuna slides & Dr. Qi slides

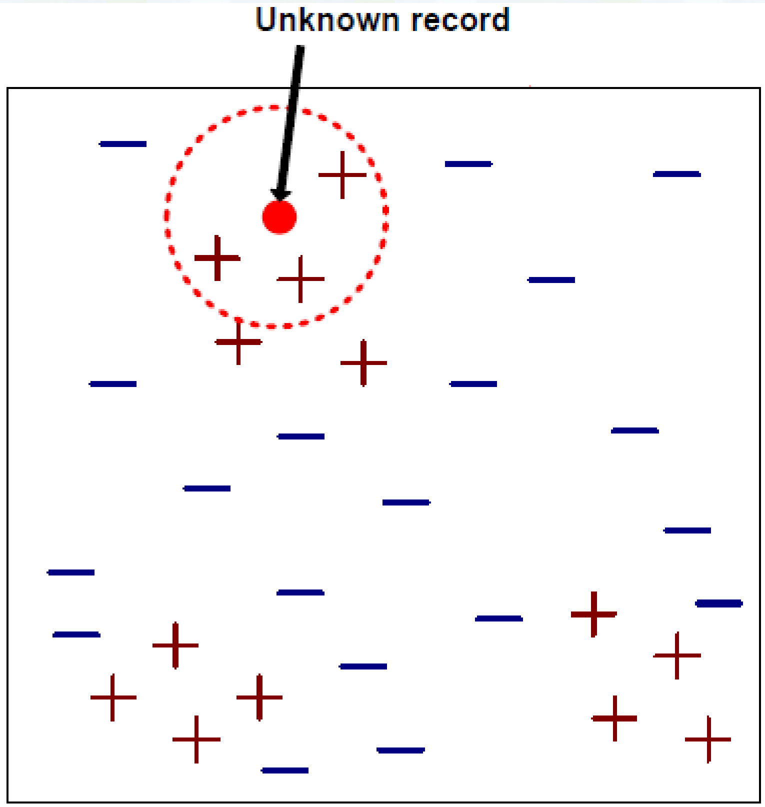
Nearest neighbor classifiers

Basic idea:

- If it walks like a duck, quacks like a duck, then it is probably a ***duck***!



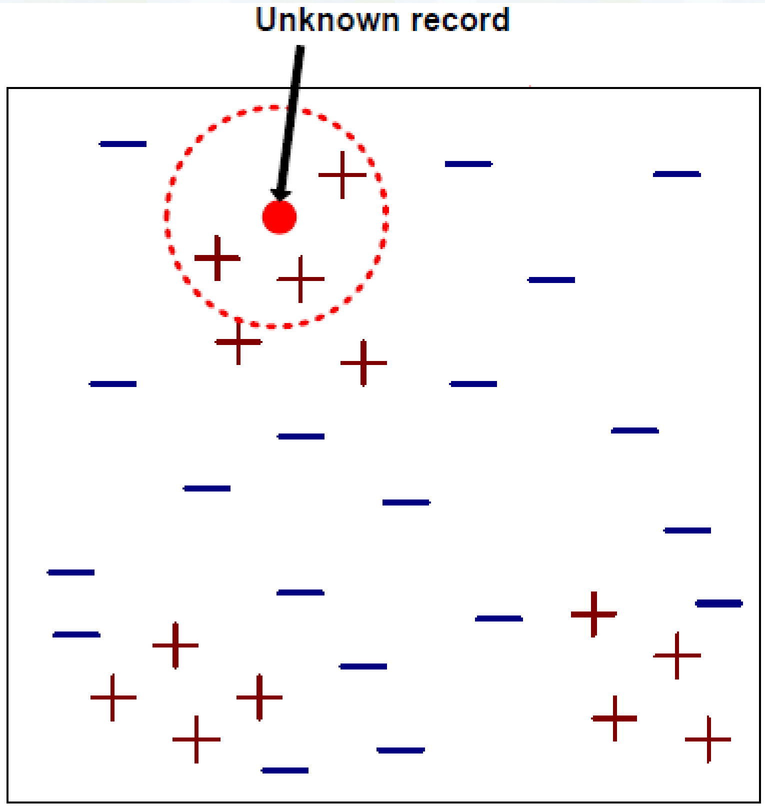
Nearest neighbor classifiers



In this example, K = 3

- Requires three inputs:
  1. The set of stored training samples
  2. Distance metric to compute distance between samples ("*closeness*")
  3. The value of ***k***, i.e., the number of nearest neighbors to retrieve

Nearest neighbor classifiers



In this example, K = 3

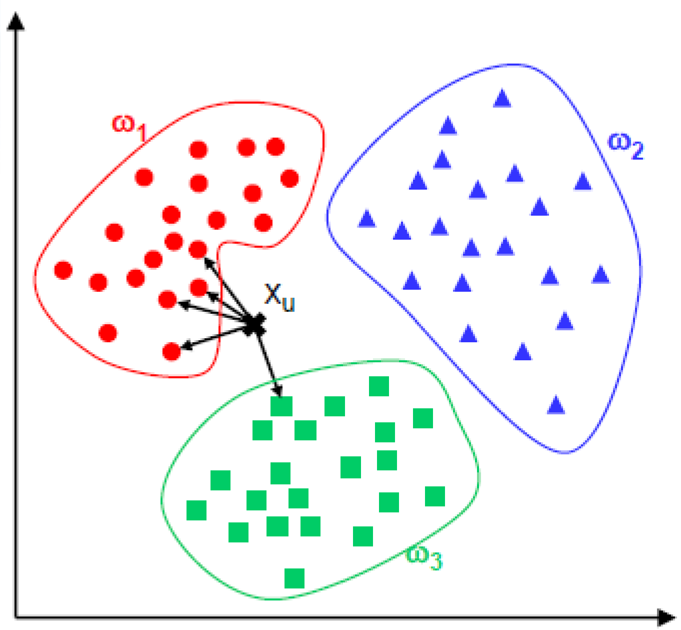
- New unknown* sample arrives… *now what?!*:
  1. Using distance metric, compute distance to other training records (“*closeness*”)
  2. Identify k nearest neighbors (*hence the name!*)
  3. Use class labels of nearest neighbors to determine the class label of unknown record (i.e. by taking a majority vote)

* Unknown means unlabeled

The kNN classifier: definition

- The kNN rule is a very intuitive method that classifies unlabeled examples based on their similarity to examples in the training set
- For a given unlabeled example x_u, find the k "closest" labeled examples in the training data set and assign x_u to the class that appears most frequently within the k-subset

The kNN classifier: definition



- In this example there are three classes and the goal is to find a class label for the unknown example x_u
- Let's use the Euclidean distance and a value of $k = 5$ neighbors
- Of the 5 closest neighbors, 4 belong to $\boldsymbol{\omega_1}$ and 1 belongs to $\boldsymbol{\omega_3}$, so x_u is assigned to $\boldsymbol{\omega_1}$, the *predominant class*

The kNN classifier: definition

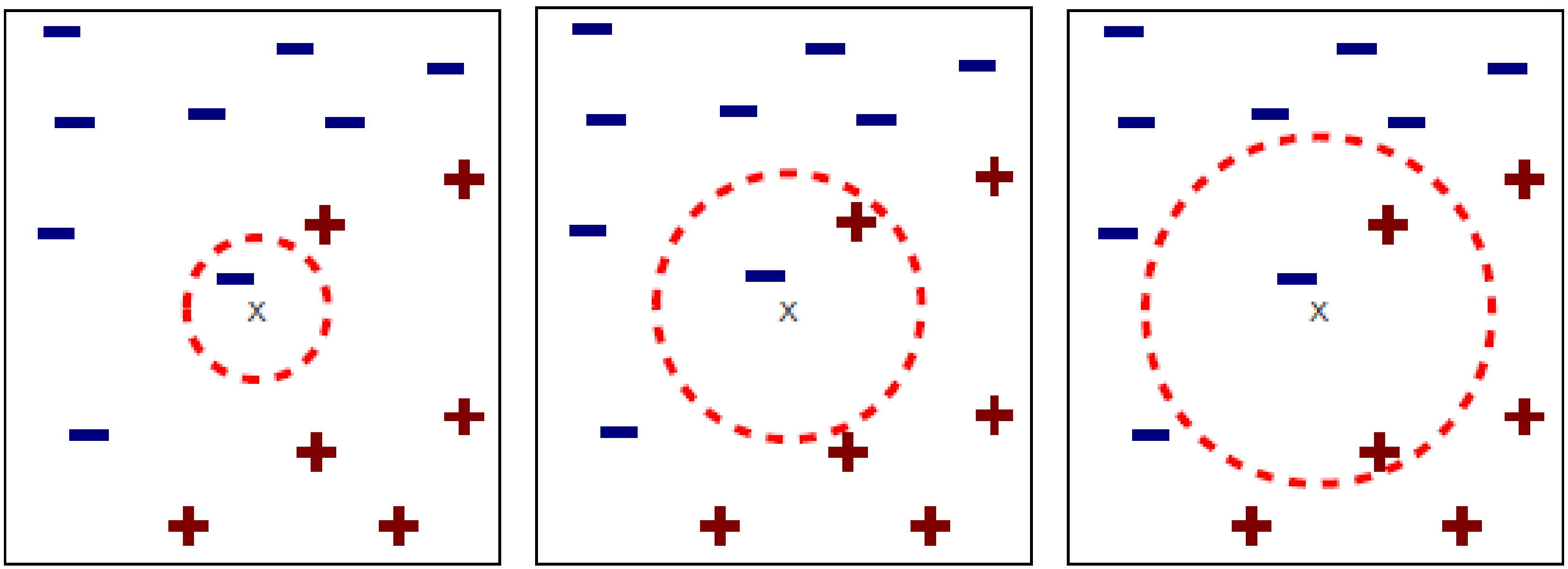
- Compute **distance** between two points:
  - For example, Euclidean distance

$$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

  - Can use Cosine distance for text
- Options for determining the class from nearest neighbor list
  - Take majority vote of class labels among the k-nearest neighbors
  - Weight the votes according to distance, example: weight factor $w = 1 / d^2$

The kNN classifier: definition

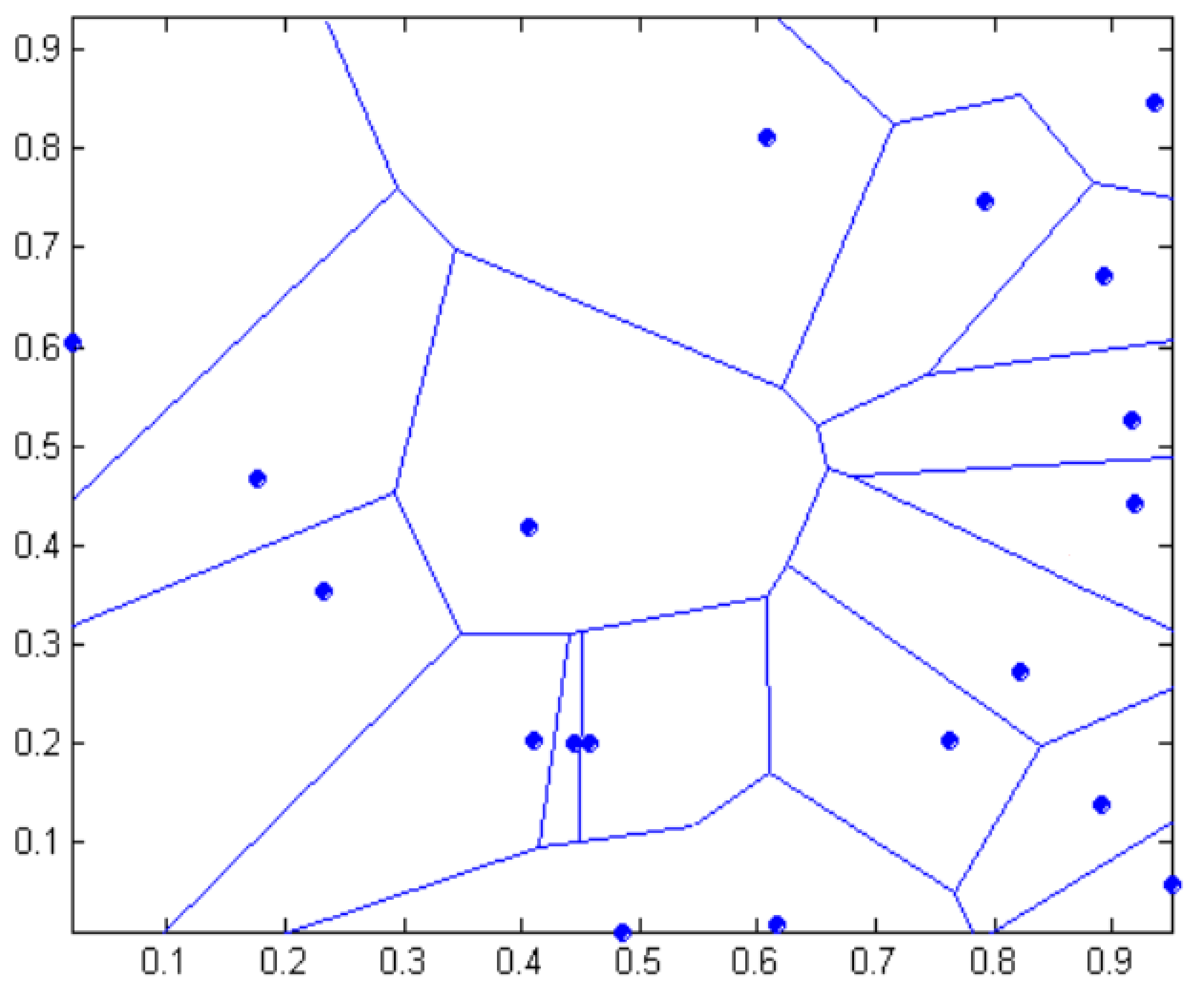
- K-nearest neighbors of a sample x are data points that have the k smallest distances to x



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

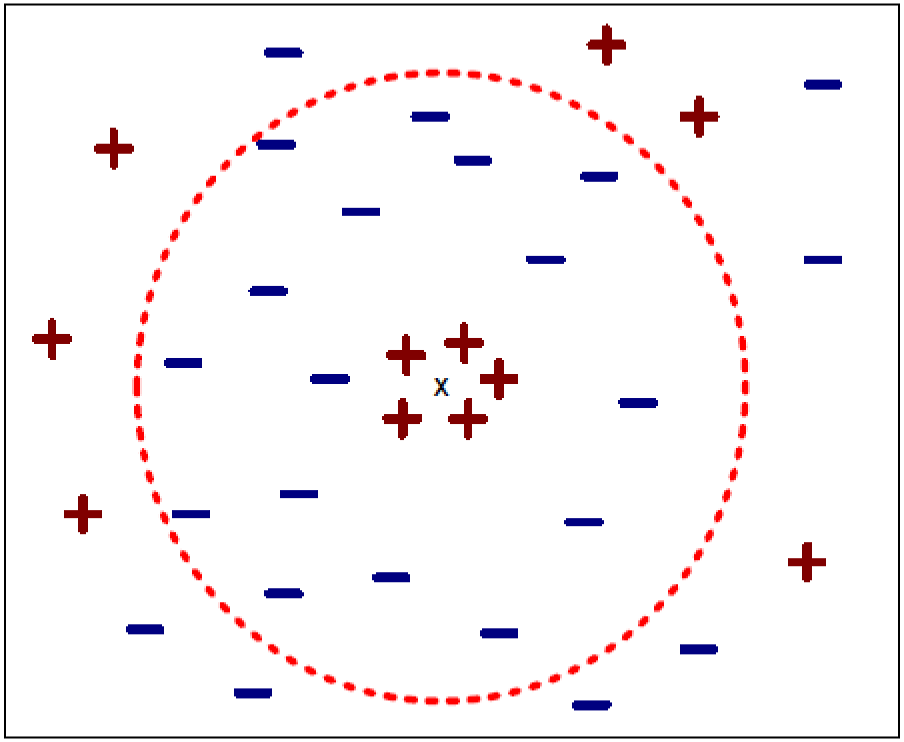
1-Nearest Neighbor

- Voronoi diagram:
- Partitioning of a plane into regions based on distance to points in a specific subset of the plane
- *If a new point falls within a region, it is clear which is the closest point (and what the label is)*



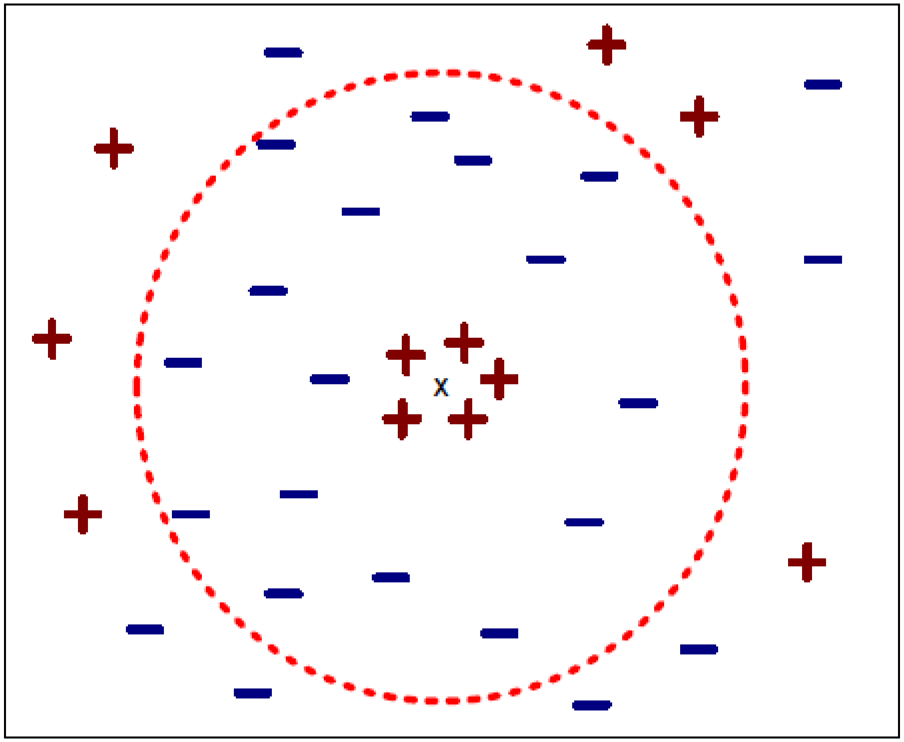
The kNN classifier: choosing *k*

- Choosing the value of k:
  - If ***k*** is too small, sensitive to noise points
  - If ***k*** is too large, neighborhood may include points from other classes



The kNN classifier: choosing *k*

- Choosing the value of k:
  - If ***k*** is too small, sensitive to noise points
  - If ***k*** is too large, neighborhood may include points from other classes
  - If ***k*** is small:
    - Flexible
    - Varies a lot
  - If ***k*** is large:
    - Smooth
    - Varies little



kNN as a Machine Learning algorithm

- kNN is considered a **lazy** learning algorithm
  - Defers data processing until it receives a request to classify unlabeled data
  - Replies to a request for information by combining its stored training data
  - Discards the constructed answer and any intermediate results
- Does not build model explicitly
- Classifying unknown samples is relatively expensive
  → kNN: all training samples;   SVM: num support vectors
- kNN is a local model (vs. global model of linear classifiers)

kNN as a Machine Learning algorithm

- Opposed to eager learning algorithm which:
  - Compiles its data into a compressed description or model
    - A density estimate or density parameters
    - A graph structure and associated weights
  - Discards training data after compilation of the model
  - Classifies incoming patterns using the induced model (which is retained for future requests)

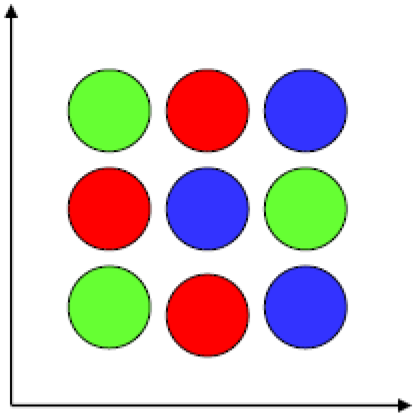
kNN as a Machine Learning algorithm

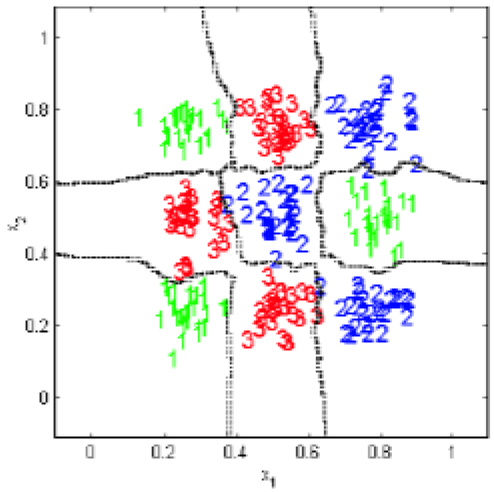
- Tradeoffs
  - Lazy algorithms have fewer computational costs than eager algorithms during training
  - Lazy algorithms have greater storage requirements and higher computational costs on recall

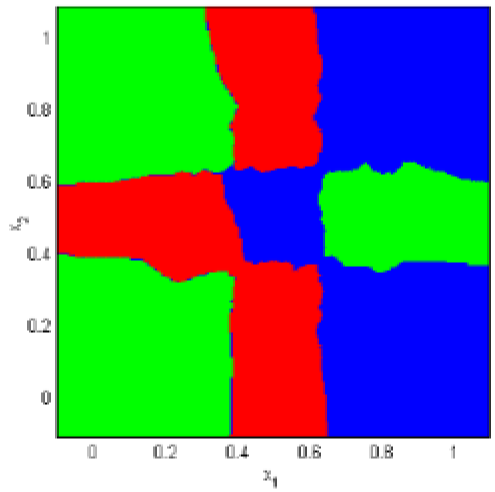
- However, has advantages…
  - Simple implementation
  - Use of local info, which can yield highly adaptive behavior
  - Lends itself very easily to parallel implementations

Some Examples

– Three-class 2D problem with non-linearly separable, multimodal likelihoods
– We use the kNN rule ($k = 5$) and the Euclidean distance
– The resulting decision boundaries and decision regions are shown below

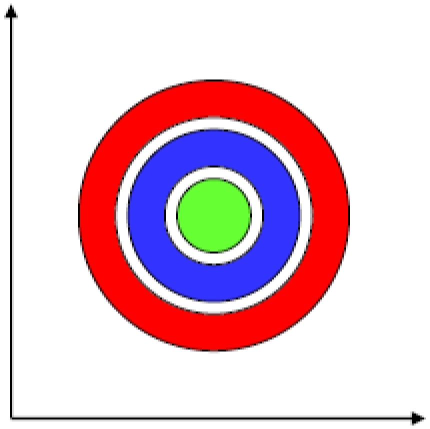


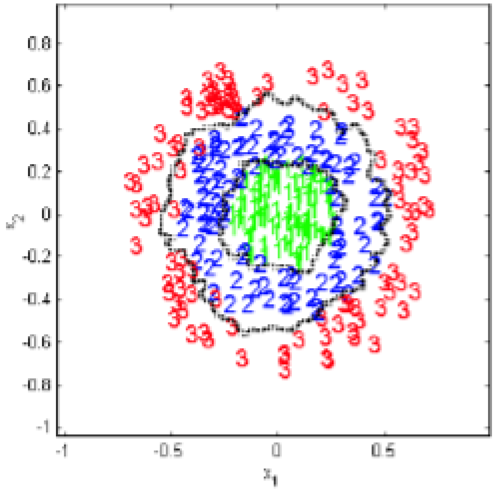


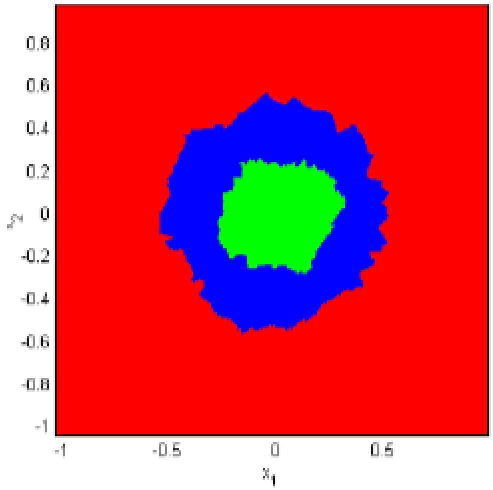


Some Examples

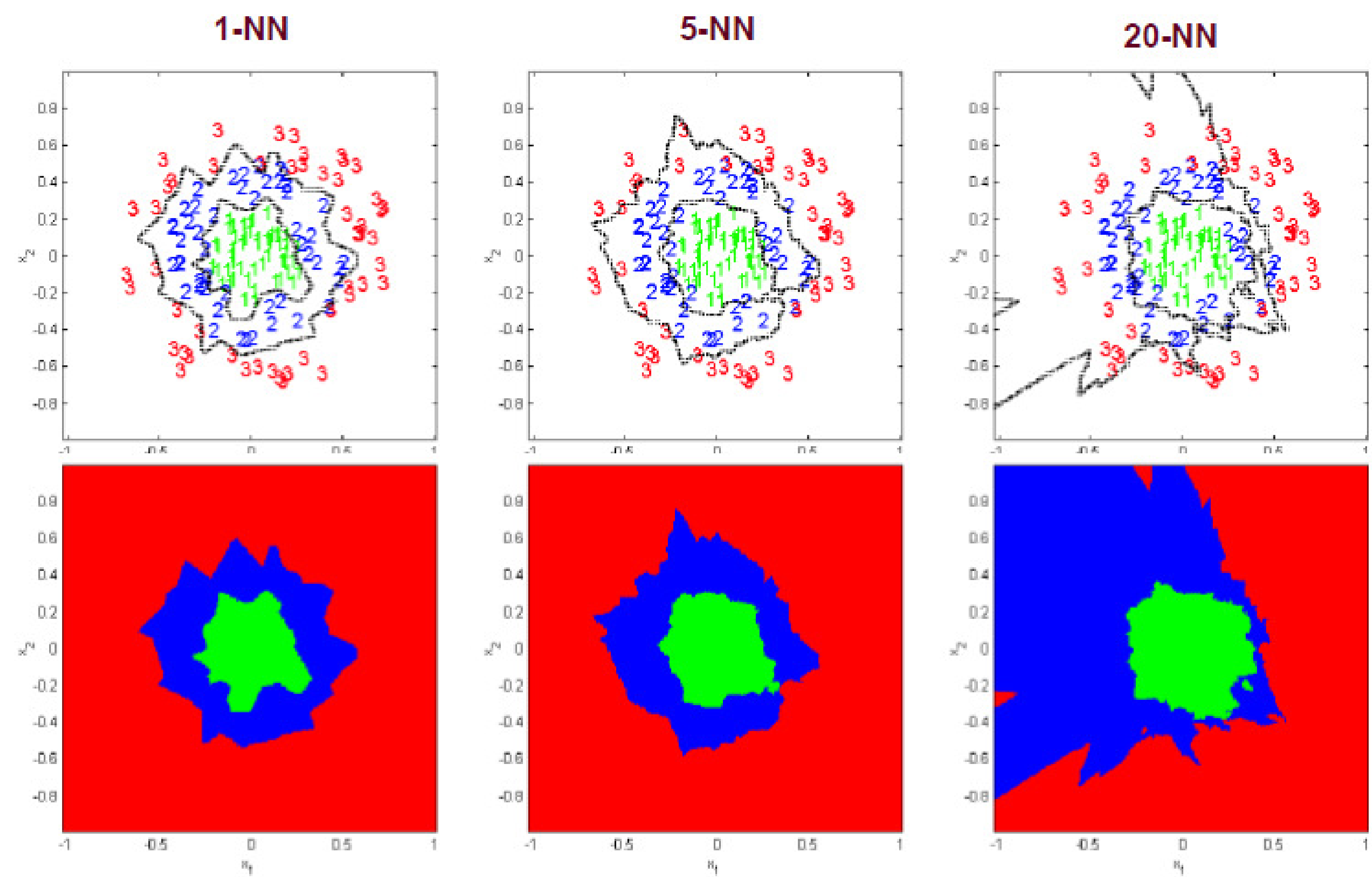
- Two-dim 3-class problem with unimodal likelihoods with a common mean; these classes are also not linearly separable
- We used the kNN rule ($k = 5$), and the Euclidean distance as a metric







kNN versus 1NN



kNN versus 1NN

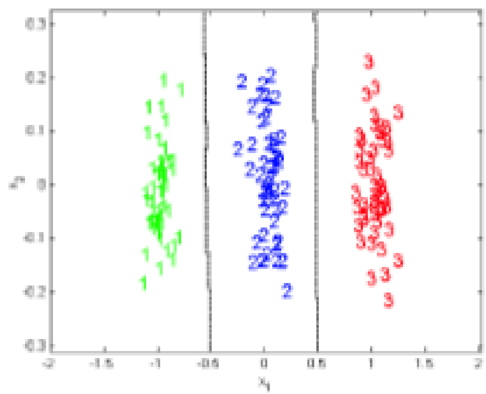
- The use of large values of k has two main advantages
  - Yields smoother decision regions
  - Provides probabilistic information, i.e., the ratio of examples for each class gives information about the ambiguity of the decision
- However, too large a value of k is detrimental
  - It destroys the locality of the estimation
    - since farther examples are taken into account
  - It increases the computational burden

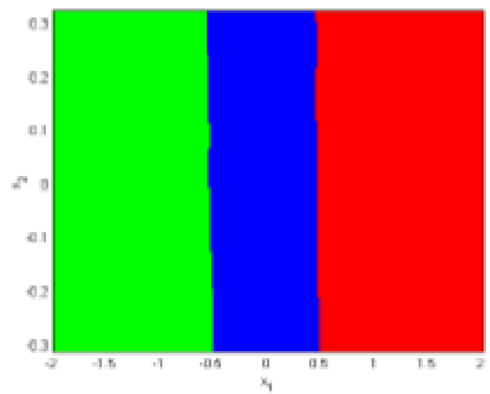
kNN and feature weighting

kNN is sensitive to noise since it is based on the Euclidean distance

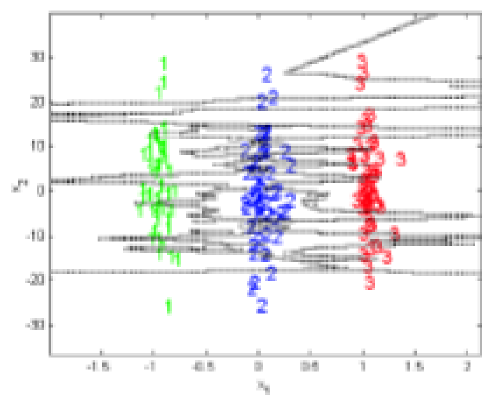
- To illustrate this point, consider the example below
  - The first axis contains all the discriminatory information
  - The second axis is white noise, and does not contain classification information

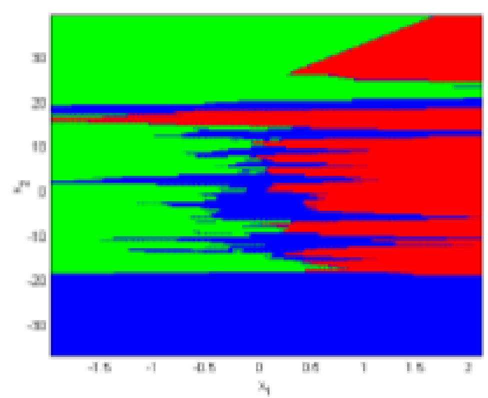
- In a first case, both axes are scaled properly
  - kNN ($k = 5$) finds decision boundaries fairly close to the optimal



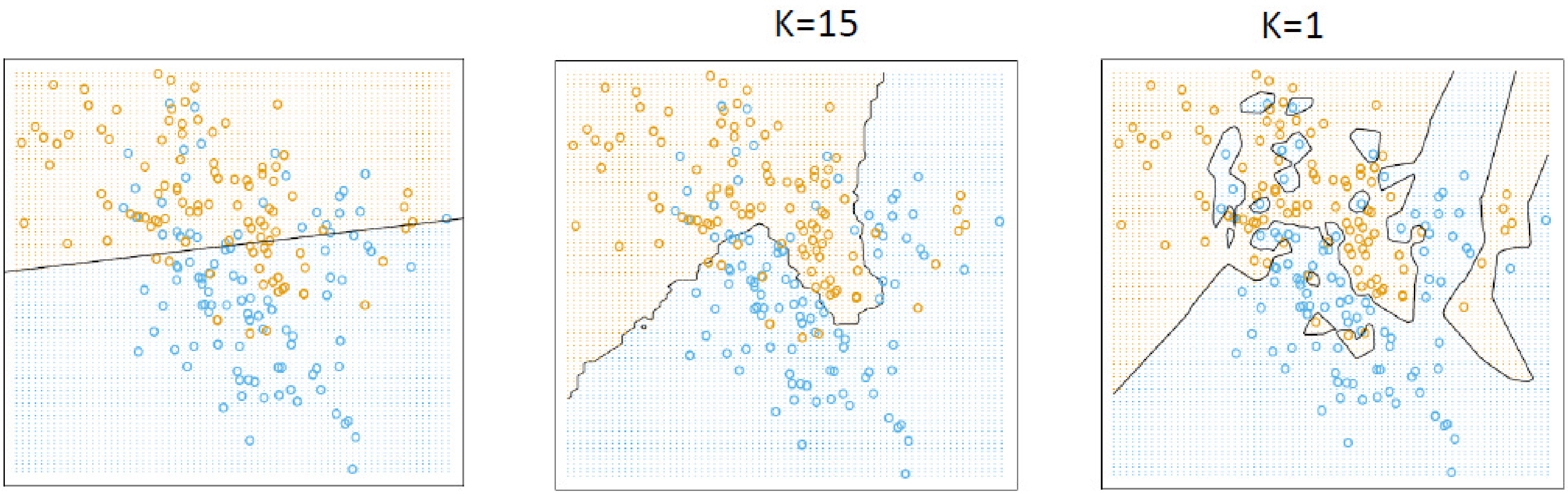


- In a second case, the scale of the second axis has been increased 100 times
  - kNN is biased by the large values of the second axis and its performance is very poor





Decision Boundaries in Global vs. Local Models



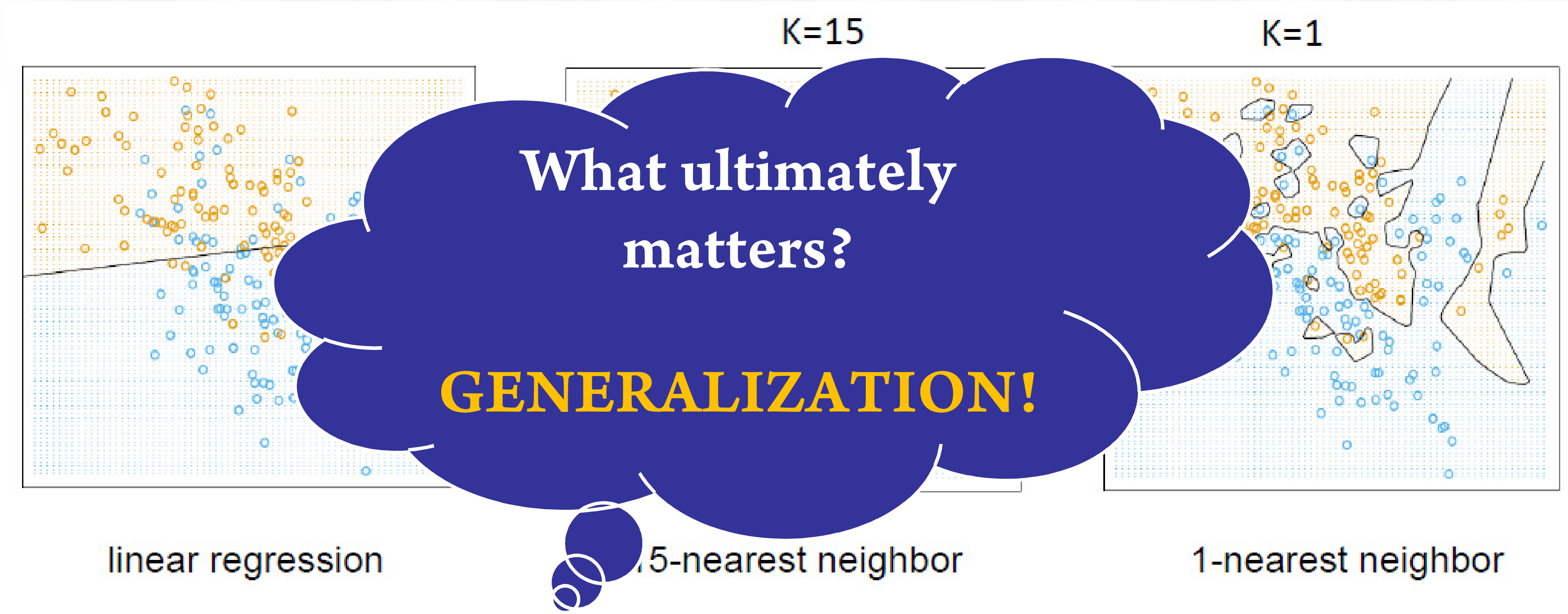
linear regression | 15-nearest neighbor | 1-nearest neighbor

- Global
- Stable
- Can be inaccurate

- K acts as a smoother

- Local
- Accurate
- Unstable

Decision Boundaries in Global vs. Local Models



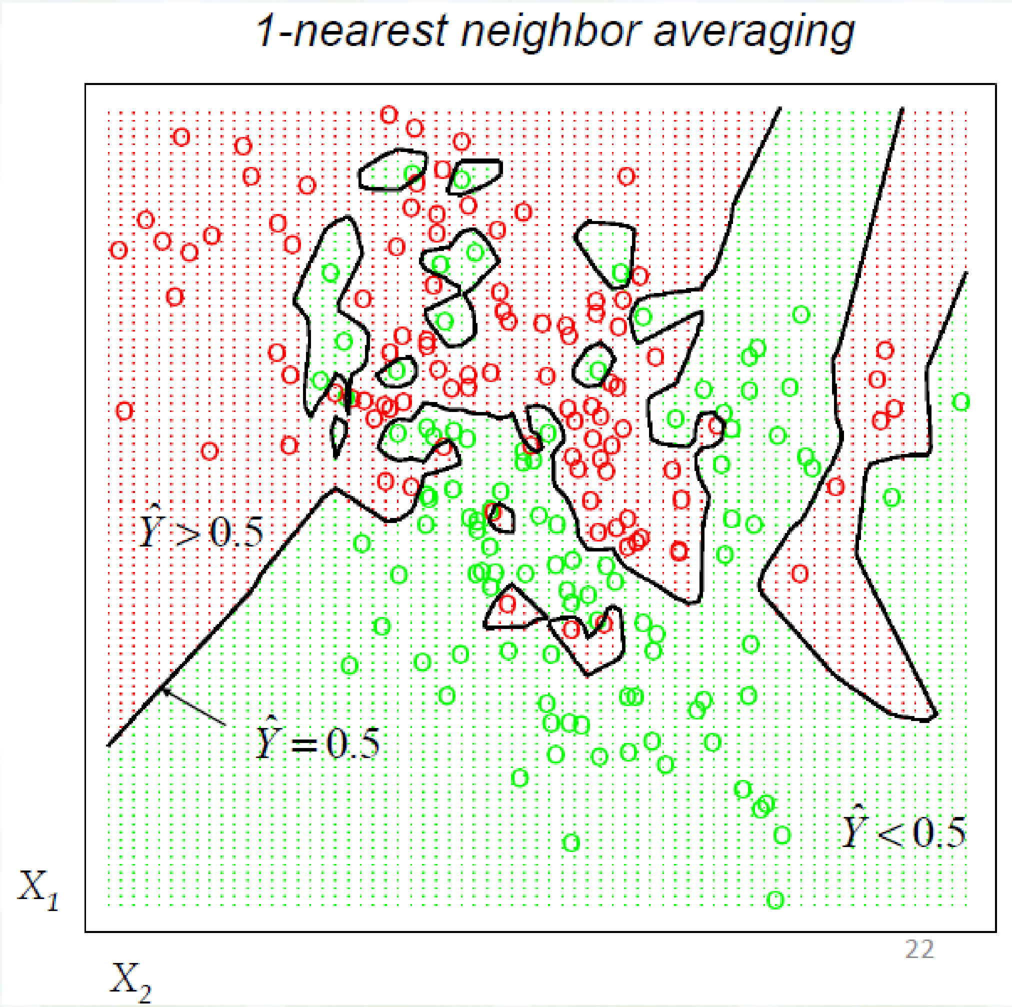
- Global
- Stable
- Can be inaccurate

- K acts as a smoother

- Local
- Accurate
- Unstable

Training Error from kNN
Lesson Learned

- When **k = 1**
- No misclassification (on training): Overtraining
- Minimizing training error is not always good (e.g. 1-NN)



Statistical Decision Theory

- Random input vector: X
- Random output variable: Y
- Joint distribution: Pr(X,Y)
- Loss function: L(Y, f(X))

Consider population distribution

- Expected prediction error (EPE):

$$EPE(f) = E\left(L\left(Y, f(X)\right)\right) = \int L\left(y, f(x)\right)\Pr(dx, dy)$$

Squared Loss

$$e.g. = \int (y - f(x))^2 \Pr(dx, dy)$$

kNN for minimizing EPE

- For squared error loss (also called L2), best estimator for EPE (theoretically) is conditional mean

$$EPE(f) = E\left(L\big(Y, f(X)\big)\right) = \int L\big(y, f(x)\big)\Pr(dx, dy)$$

Conditional mean: $\hat{f}(x) = E(Y|X = x)$

- Nearest neighbor methods are the direct implementation (approximation)
- Nearest neighbors assumes that f(x) is well approximated by a locally constant function

Bias-Variance Trade-off EPE:

$$EPE(f(x_0) = noise^2 + bias^2 + variance$$

Unavoidable error

Error due to Incorrect assumptions

Error due to Variance of training samples

Bias-Variance Trade-off EPE: *More General Setting!*

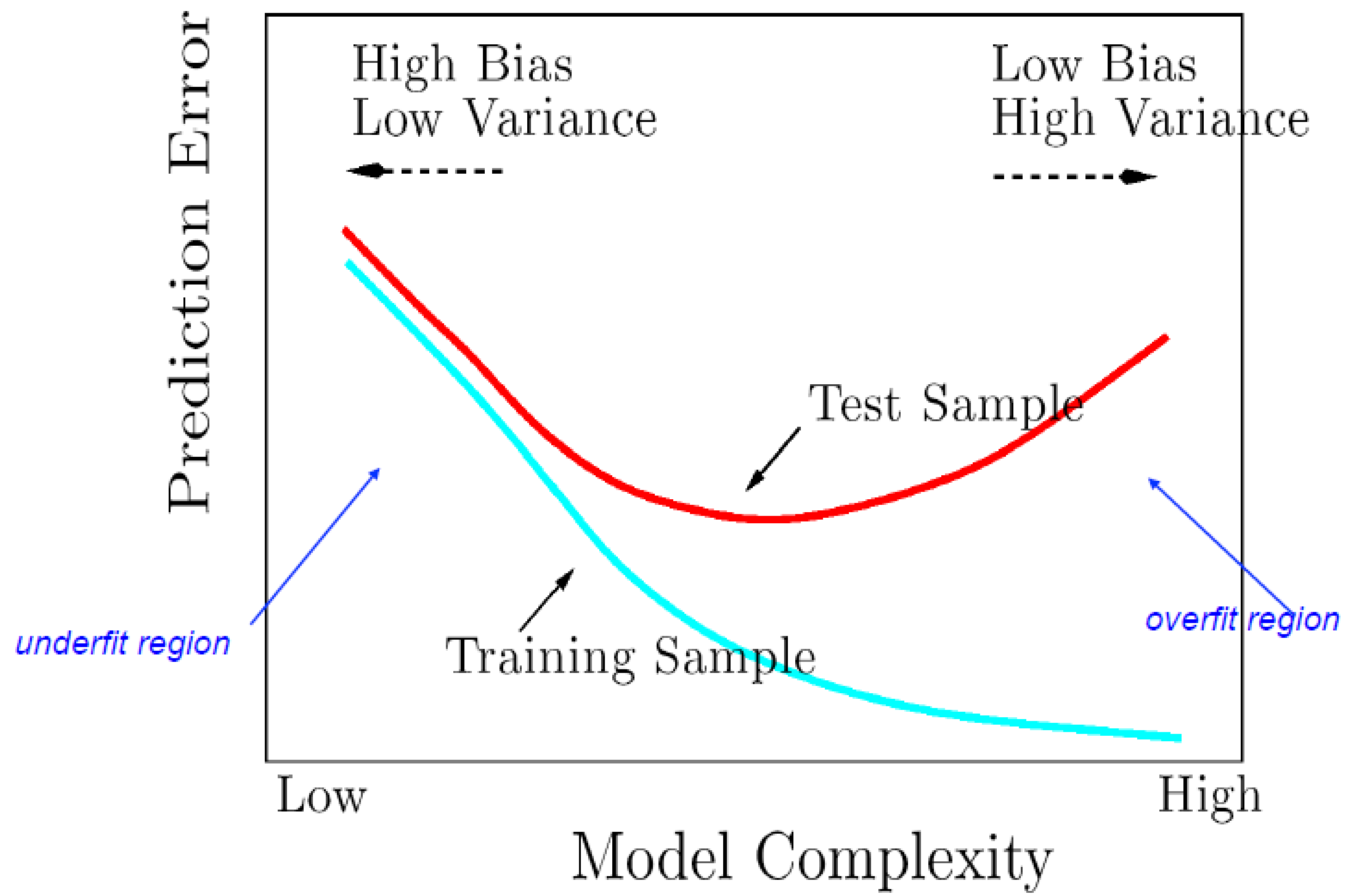
θ : true value (normally unknown)

$\hat{\theta}$: estimator

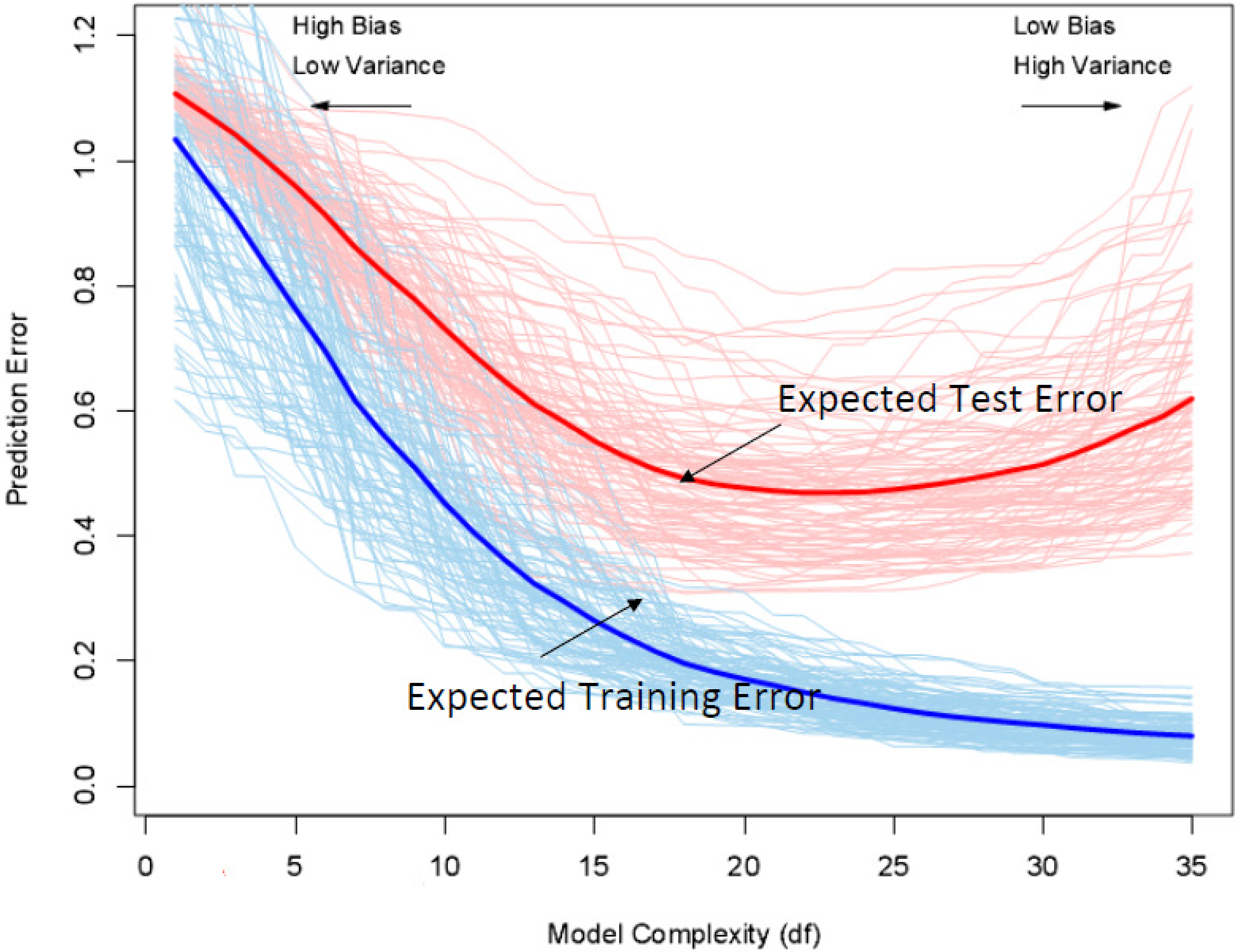
$\bar{\theta}$: $E\left[\hat{\theta}\right]$ (mean, i.e. expectation of the estimator)

- **Bias** $E[(\bar{\theta} - \theta)^2]$
  - Measures accuracy or quality of the estimator
  - Low bias implies on average we will accurately estimate true parameter or function from training data
- **Variance** $E[(\hat{\theta} - \bar{\theta})^2]$
  - Measures precision or specificity of the estimator
  - Low variance implies the estimator does not change much as the training set varies

Bias-Variance Trade-off / Model Selection



<KNN(large k) / Regression(small d) KNN(small k) / Regression(large d)>



Expected test error and CV error → good approximation of EPE