COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

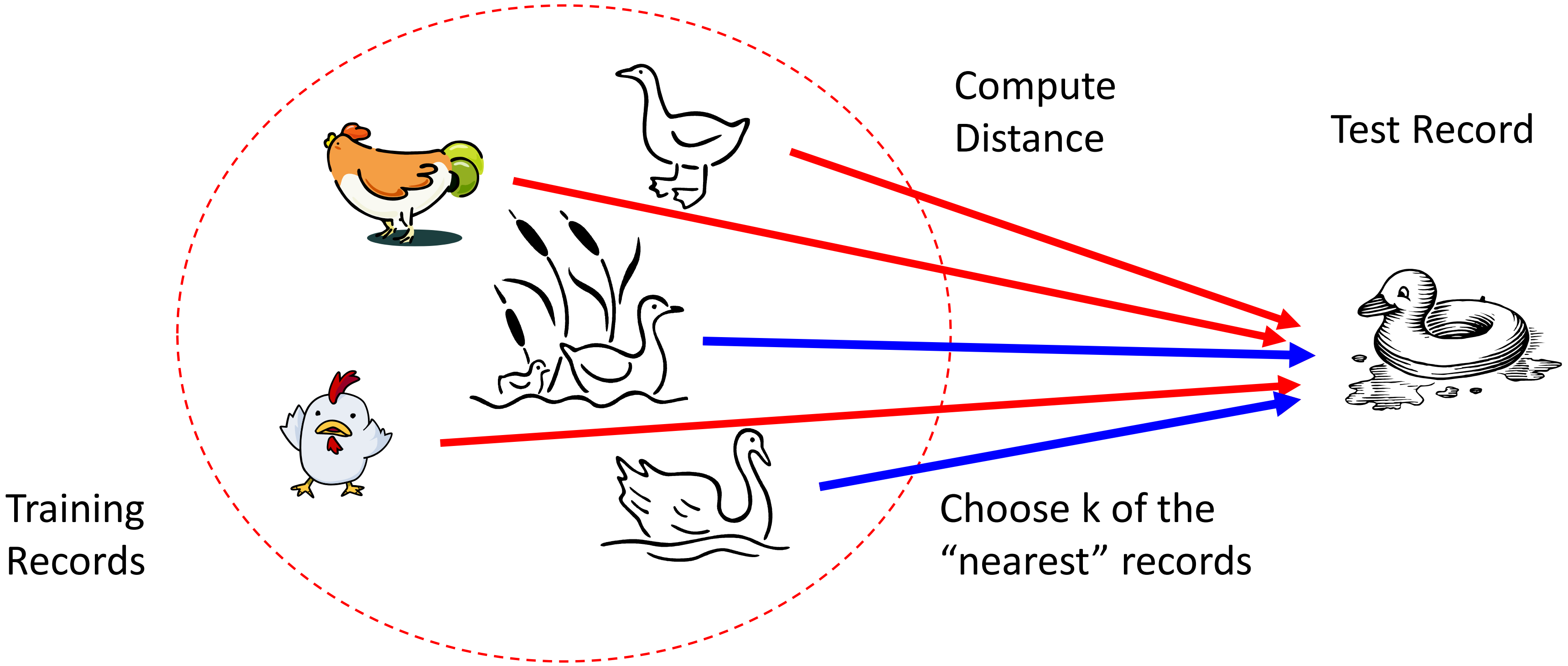
Methods to Learn

	Matrix Data	Text Data	Set Data	Sequence Data	Time Series	Graph & Network	Images
Classification	Decision Tree; Naïve Bayes; Logistic Regression **SVM; kNN**			HMM		Label Propagation	Neural Network
Clustering	K-means; hierarchical clustering; DBSCAN; Mixture Models; kernel k- means*	PLSA				SCAN; Spectral Clustering	
Frequent Pattern Mining			Apriori; FP-growth	GSP; PrefixSpan			
Prediction	Linear Regression				Autoregression	Collaborative Filtering	
Similarity Search					DTW	P-PageRank	
Ranking						PageRank	

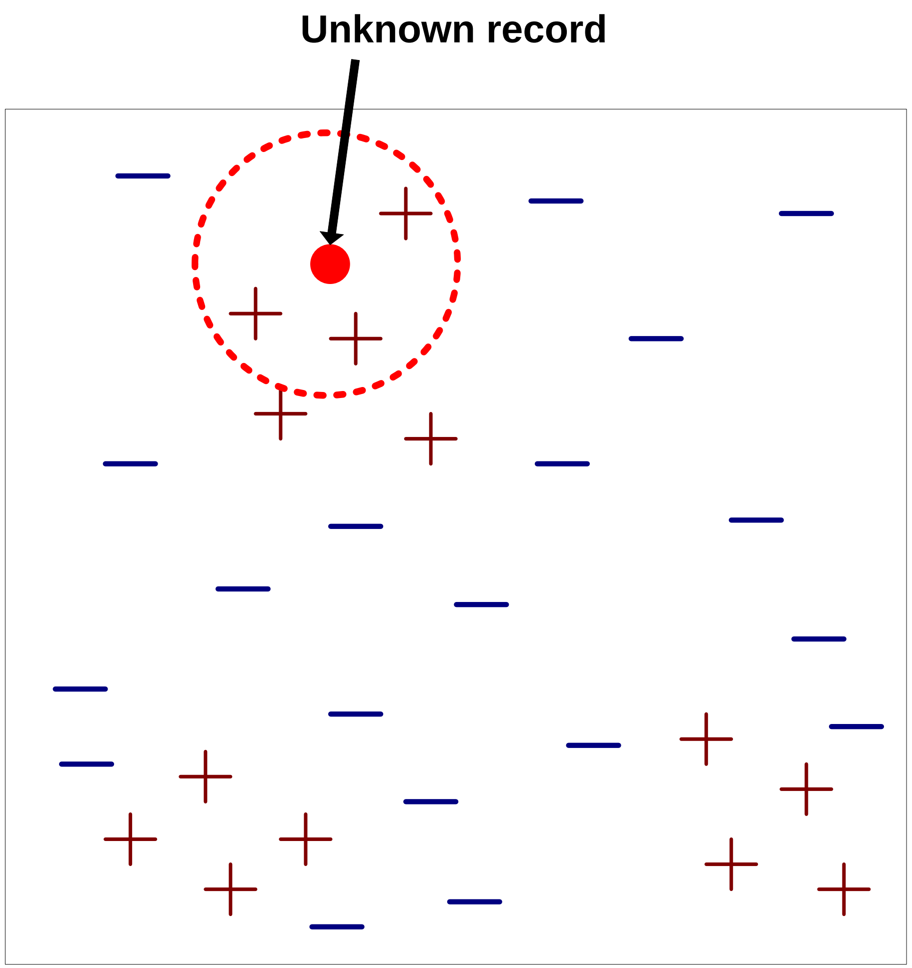
3

Nearest Neighbor Classifiers

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



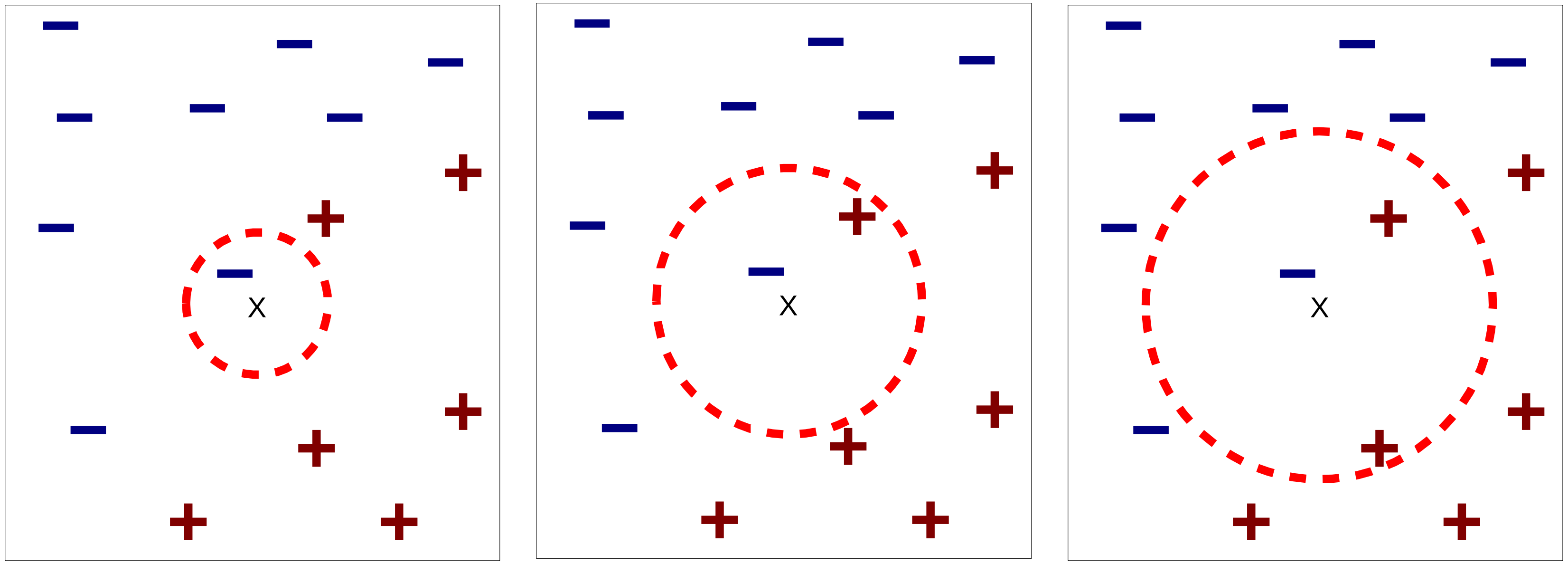
Nearest-Neighbor Classifiers



- Requires three things
  - –The set of stored records
  - –Distance Metric to compute distance between records
  - –The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record:
  - –Compute distance to other training records
  - –Identify *k* nearest neighbors
  - –Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority

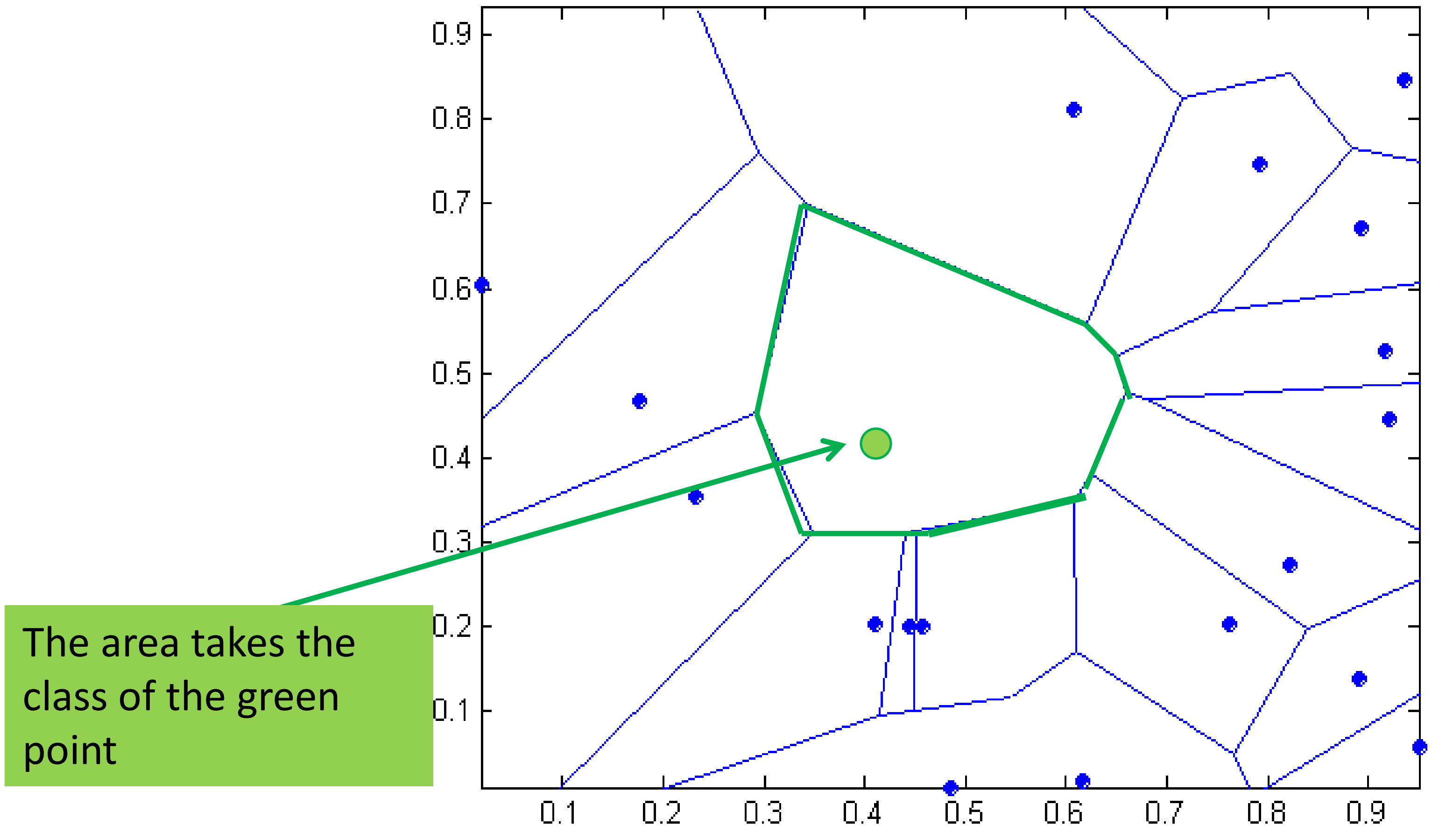
Definition of Nearest Neighbor



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

1 nearest-neighbor

Voronoi Diagram defines the classification boundary



Nearest Neighbor Classification

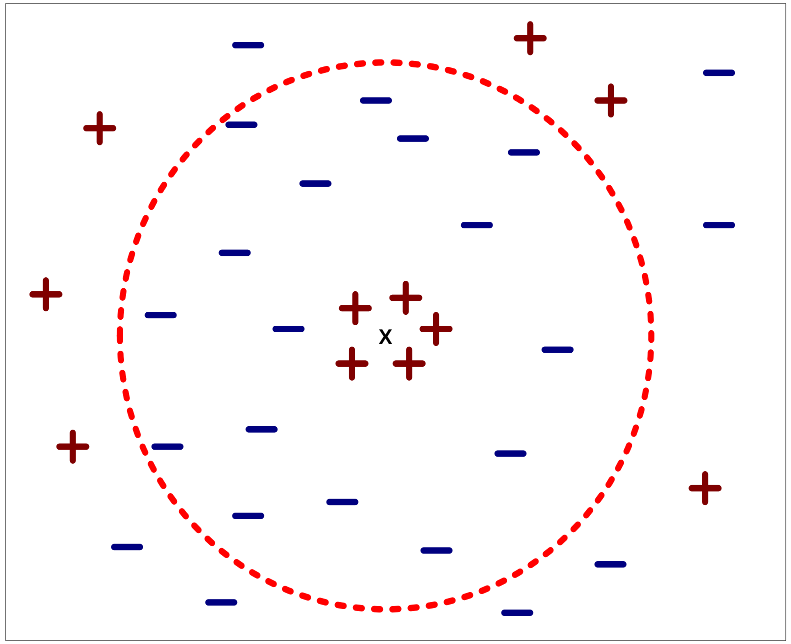
- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, $w = 1/d^2$

Nearest Neighbor Classification…

- 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

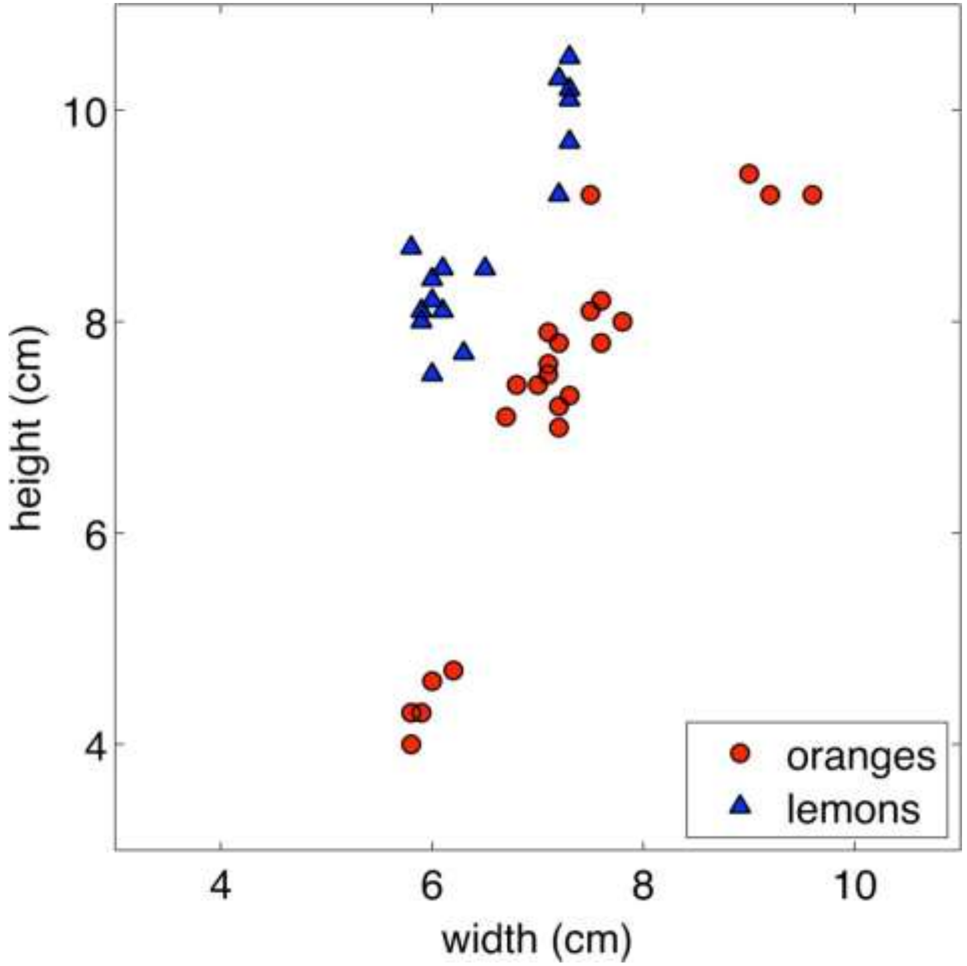


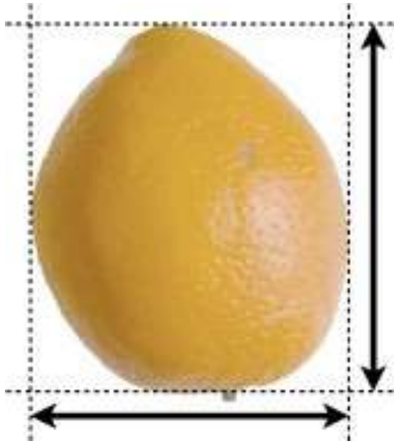
Classification :Nearest Neighbors

Non-parametric models

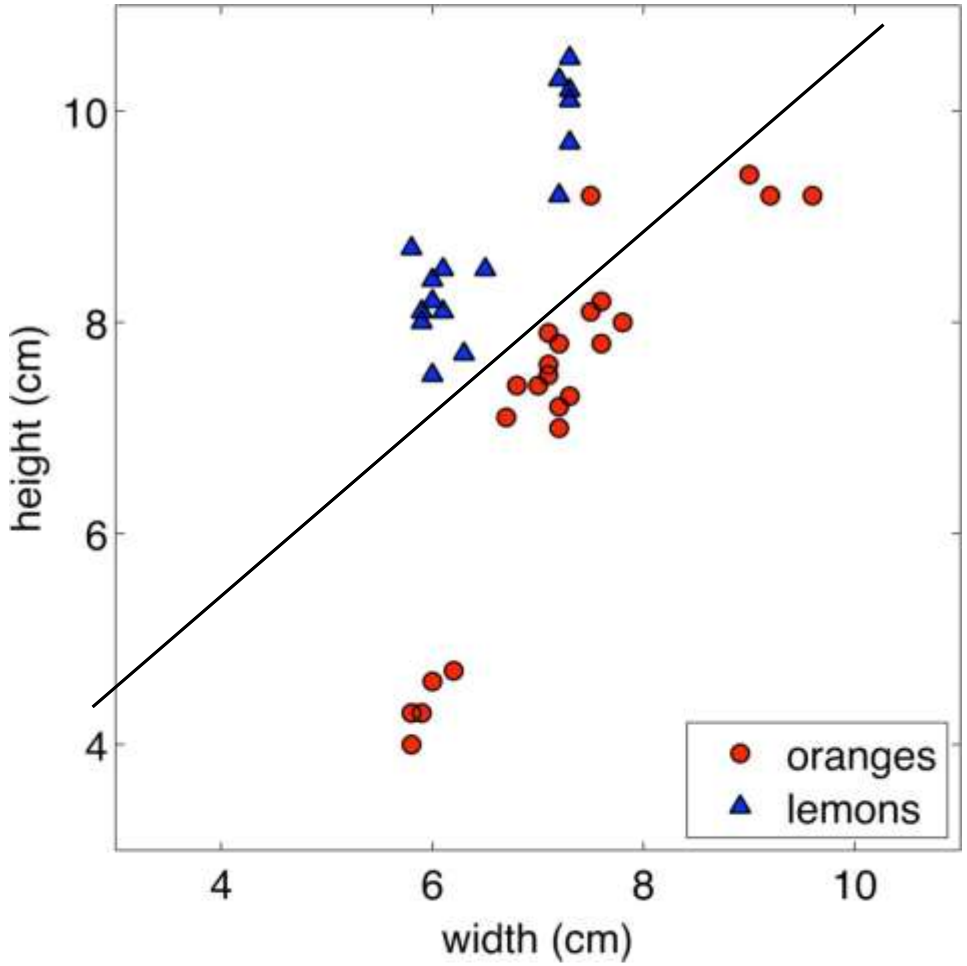
- Distance
- Non-linear decision boundaries

Classification: Oranges and Lemons



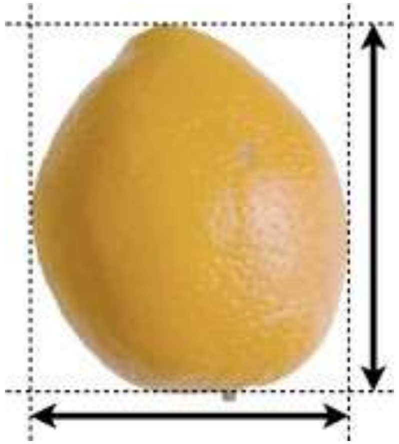


Classification: Oranges and Lemons



Can construct simple linear decision boundary:

$y = \text{sign}(w_0 + w_1 x_1 + w_2 x_2)$



Parametric models

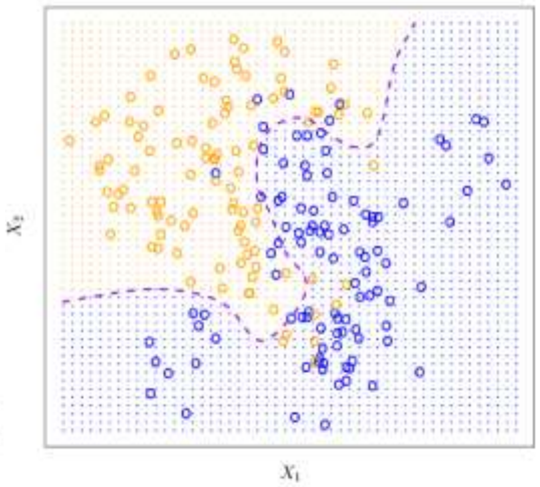
A basic approach to classification is to find a **decision boundary** in the space of the predictor variables.

The decision boundary is often a curve formed by a regression model:

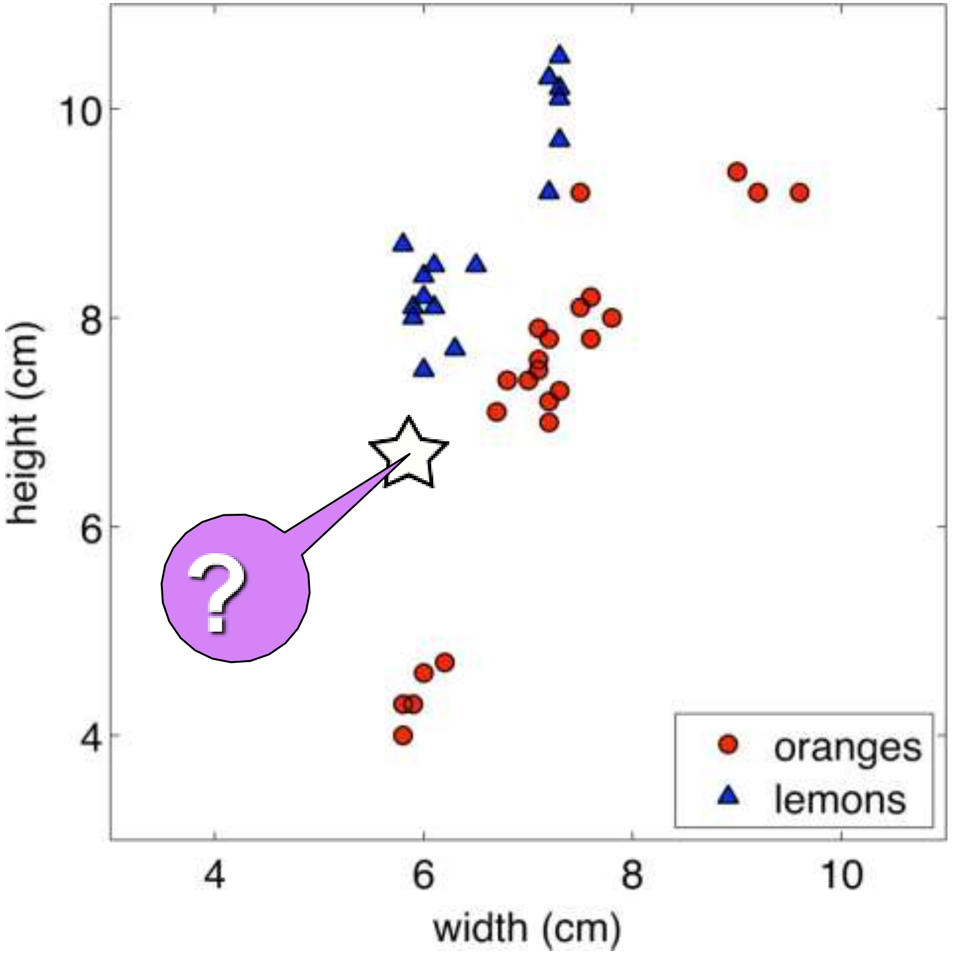
$$y_i = f(x_i) + \epsilon_i,$$

which we often take as linear:

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi} + \epsilon_i \\ &\approx \beta_0 + \beta^\top x_i. \end{aligned}$$



Classification as Induction



Instance-based Learning: Non_Parametric models

Alternative to parametric models are non-parametric models

These are typically simple methods for approximating discrete-valued or real-valued target functions (they work for classification or regression problems)

Learning amounts to simply storing training data

Test instances classified using similar training

instances Embodies often sensible underlying

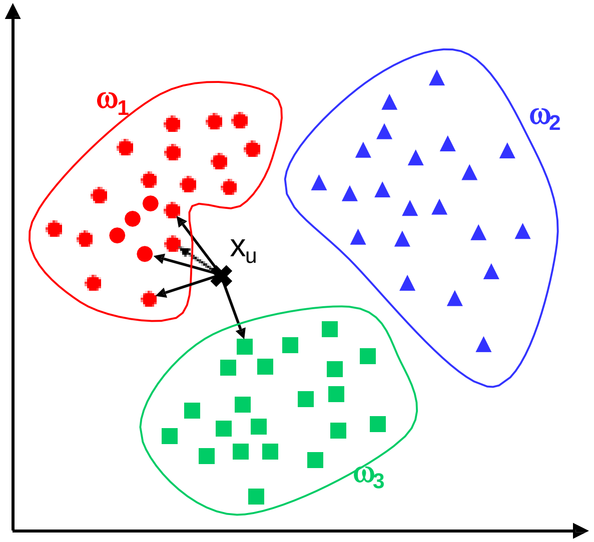
assumptions:

- Output varies smoothly with input
- Data occupies sub-space of high-dimensional input space

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 $x_u \in \Re^D$, 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 only requires
  - An integer k
  - A set of labeled examples (training data)
  - A metric to measure "closeness"

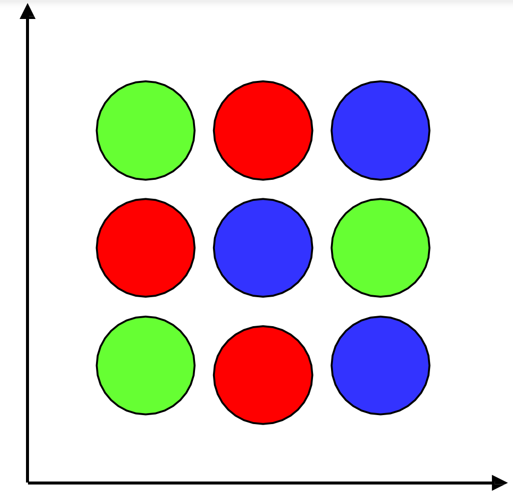


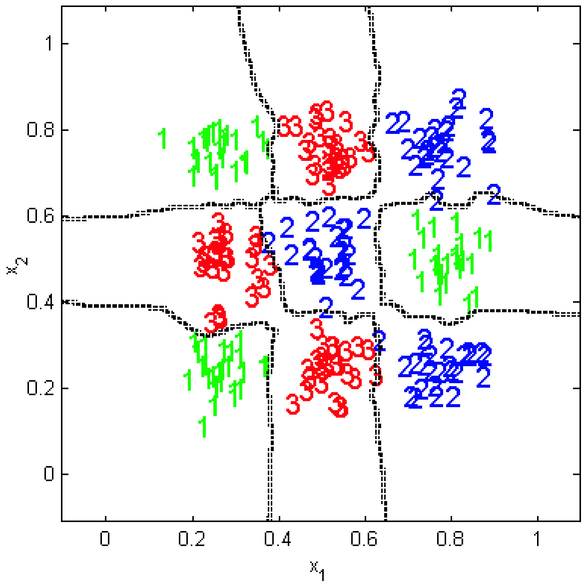
- Example
  - In the example here we have three classes and the goal is to find a class label for the unknown example x_u
  - In this case we use the Euclidean distance and a value of $k = 5$ neighbors
  - Of the 5 closest neighbors, 4 belong to ω_1 and 1 belongs to ω_3, so x_u is assigned to ω_1, the predominant class

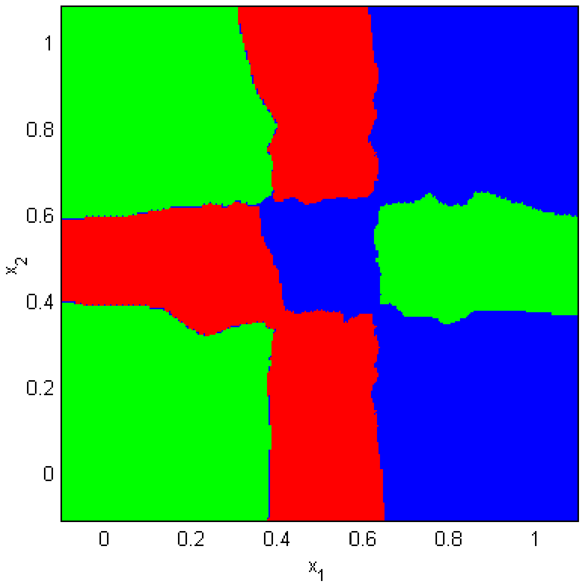
kNN in action

Example I

- 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

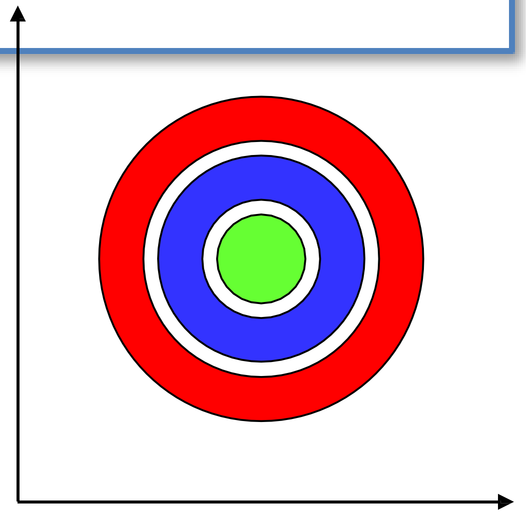


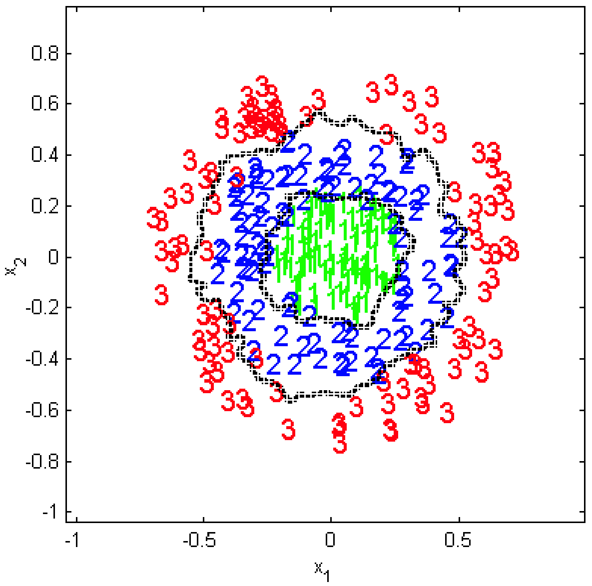


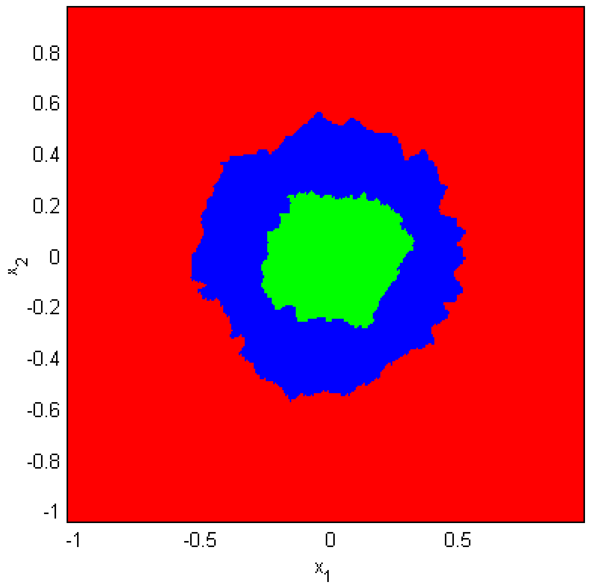


Example II

- 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 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

This strategy is opposed to an eager learning algorithm which

- Compiles its data into a compressed description or model
  - A density estimate or density parameters (statistical PR)
  - A graph structure and associated weights (neural PR)
- Discards the training data after compilation of the model
- Classifies incoming patterns using the induced model, which is retained for future requests

Tradeoffs

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

Characteristics of the kNN classifier

Advantages

- Analytically tractable
- Simple implementation
- Nearly optimal in the large sample limit ($N \rightarrow \infty$)
- Uses local information, which can yield highly adaptive behavior
- Lends itself very easily to parallel implementations

Disadvantages

- Large storage requirements
- Computationally intensive recall
- Highly susceptible to the curse of dimensionality

1NN versus kNN

- 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
  - In addition, it increases the computational burden

Optimizing storage requirements

The basic kNN algorithm stores all the examples in the training set, creating high storage requirements (and computational cost)

- However, the entire training set need not be stored since the examples may contain information that is highly redundant
  - A degenerate case is the earlier example with the multimodal classes, where each of the clusters could be replaced by its mean vector, and the decision boundaries would be practically identical
- In addition, almost all of the information that is relevant for classification purposes is located around the decision boundaries

A number of methods, called edited kNN, have been derived to take advantage of this information redundancy

- One alternative [Wilson 72] is to classify all the examples in the training set and remove those examples that are misclassified, in an attempt to separate classification regions by removing ambiguous points
- The opposite alternative [Ritter 75], is to remove training examples that are classified correctly, in an attempt to define the boundaries between classes by eliminating points in the interior of the regions

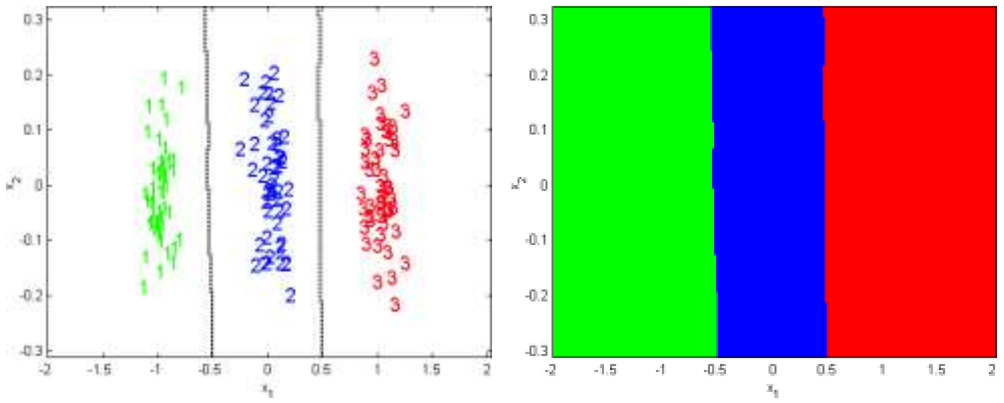
A different alternative is to reduce the training examples to a set of prototypes that are representative of the underlying data =➔Clustering

kNN and feature weighting

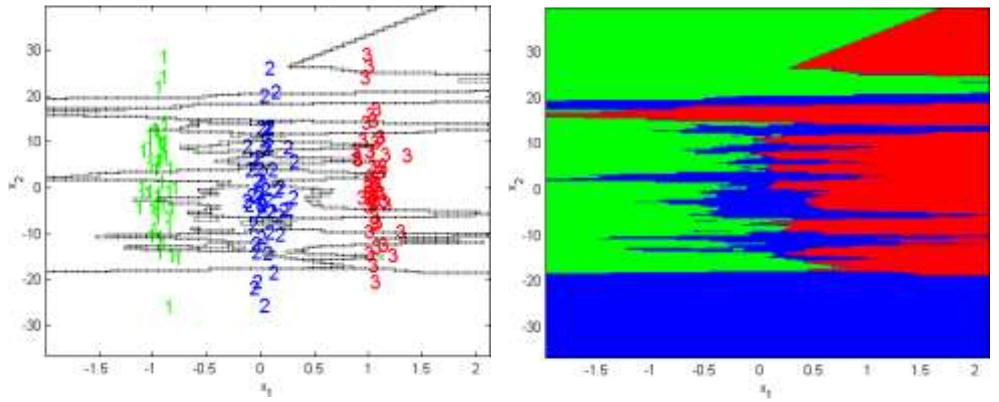
kNN is sensitive to noise since it is based on the Euclidean distance

- 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

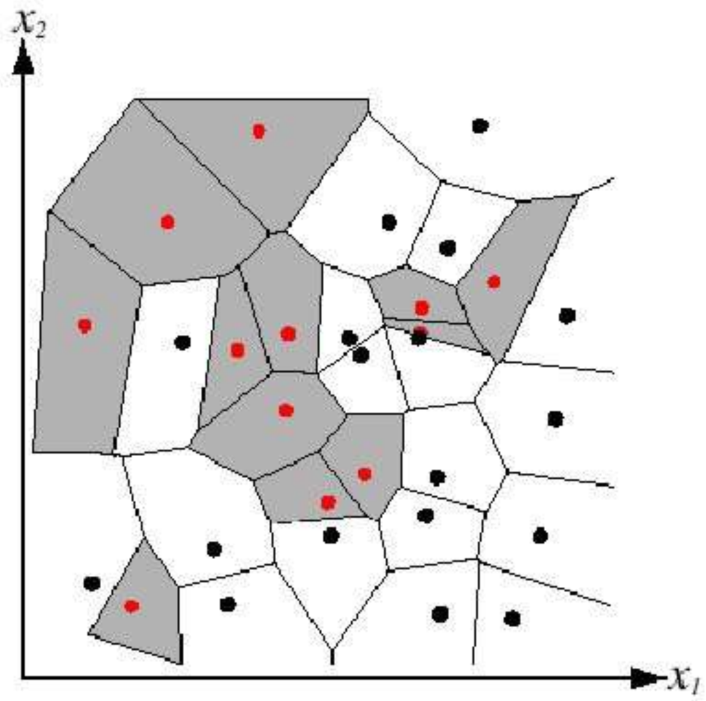


- In a second case, the scale of the second axis has been increased 100 times
  - kNN is biased by the large values of the second axis and its performance is very poor



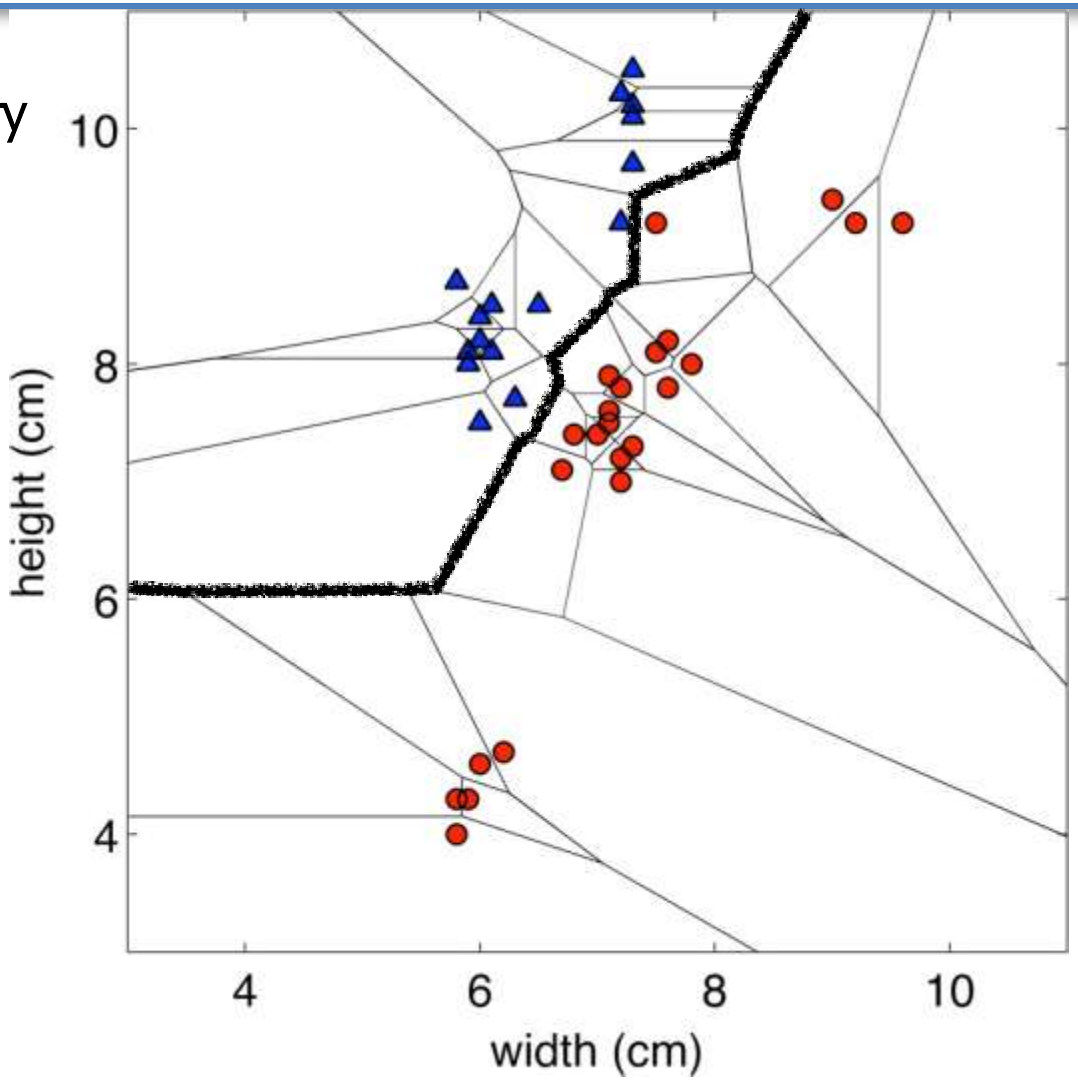
Nearest Neighbors: Decision Boundaries

- Nearest neighbor algorithm does not explicitly compute decision boundaries, but these can be inferred
- Decision boundaries: Voronoi diagram visualization
  - show how input space divided into classes
  - each line segment is equidistant between two points of opposite classes



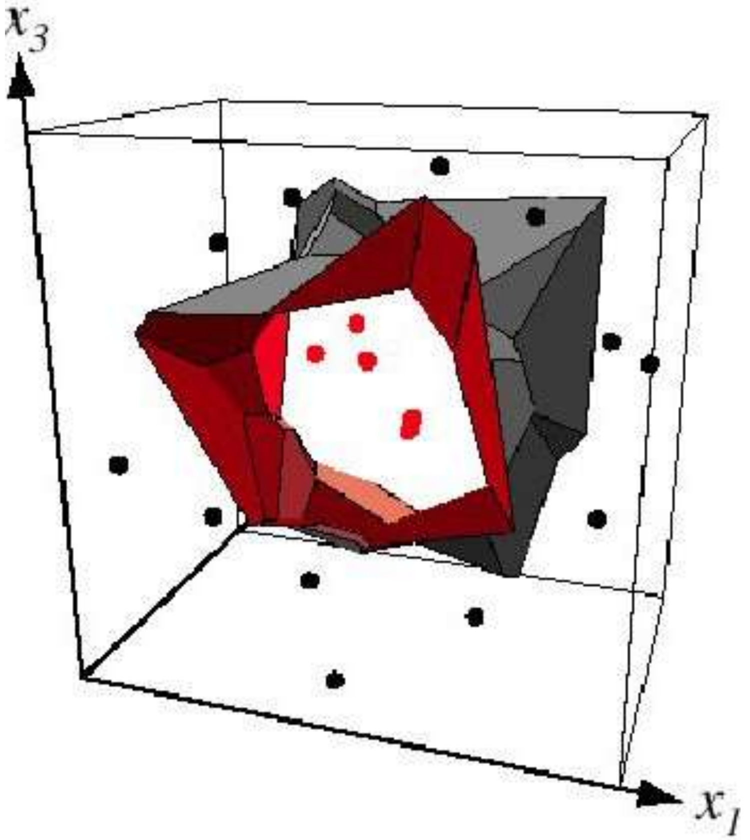
Nearest Neighbors: Decision Boundaries

Example: 2D decision boundary



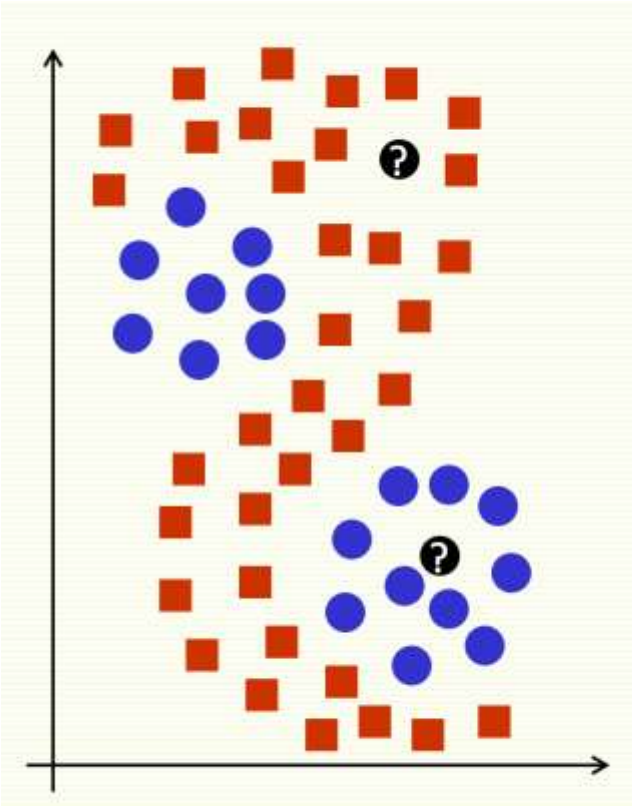
Nearest Neighbors: Decision Boundaries

Example: 3D decision boundary



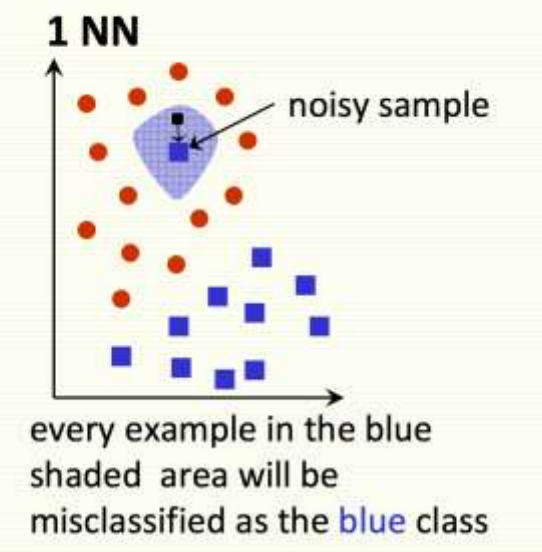
Nearest Neighbors: Multi-modal Data

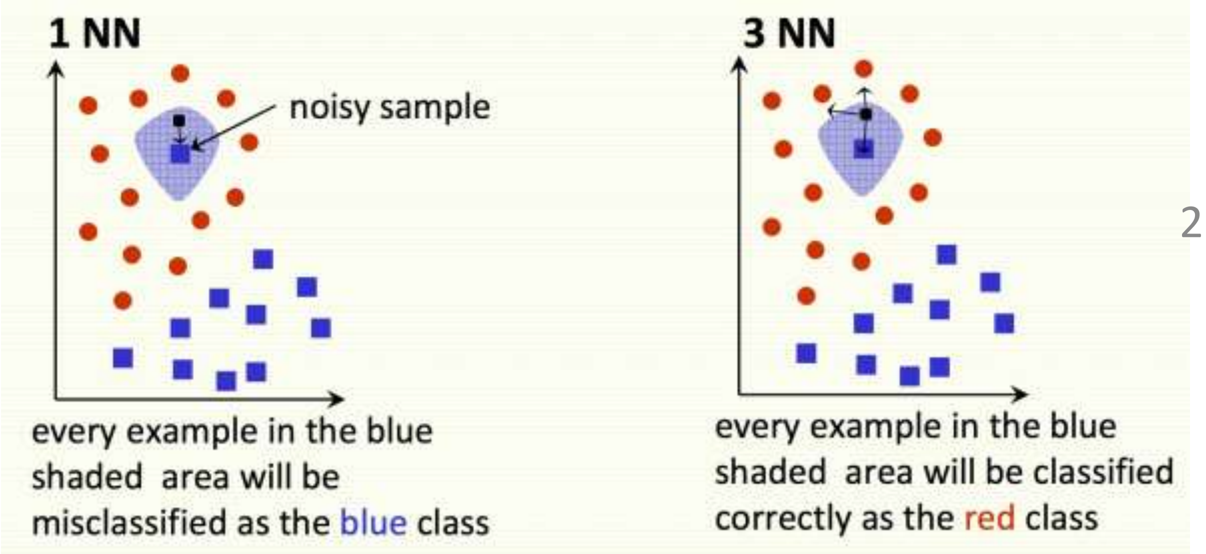
Nearest Neighbor approaches can work with multi-modal data



Nearest Neighbors

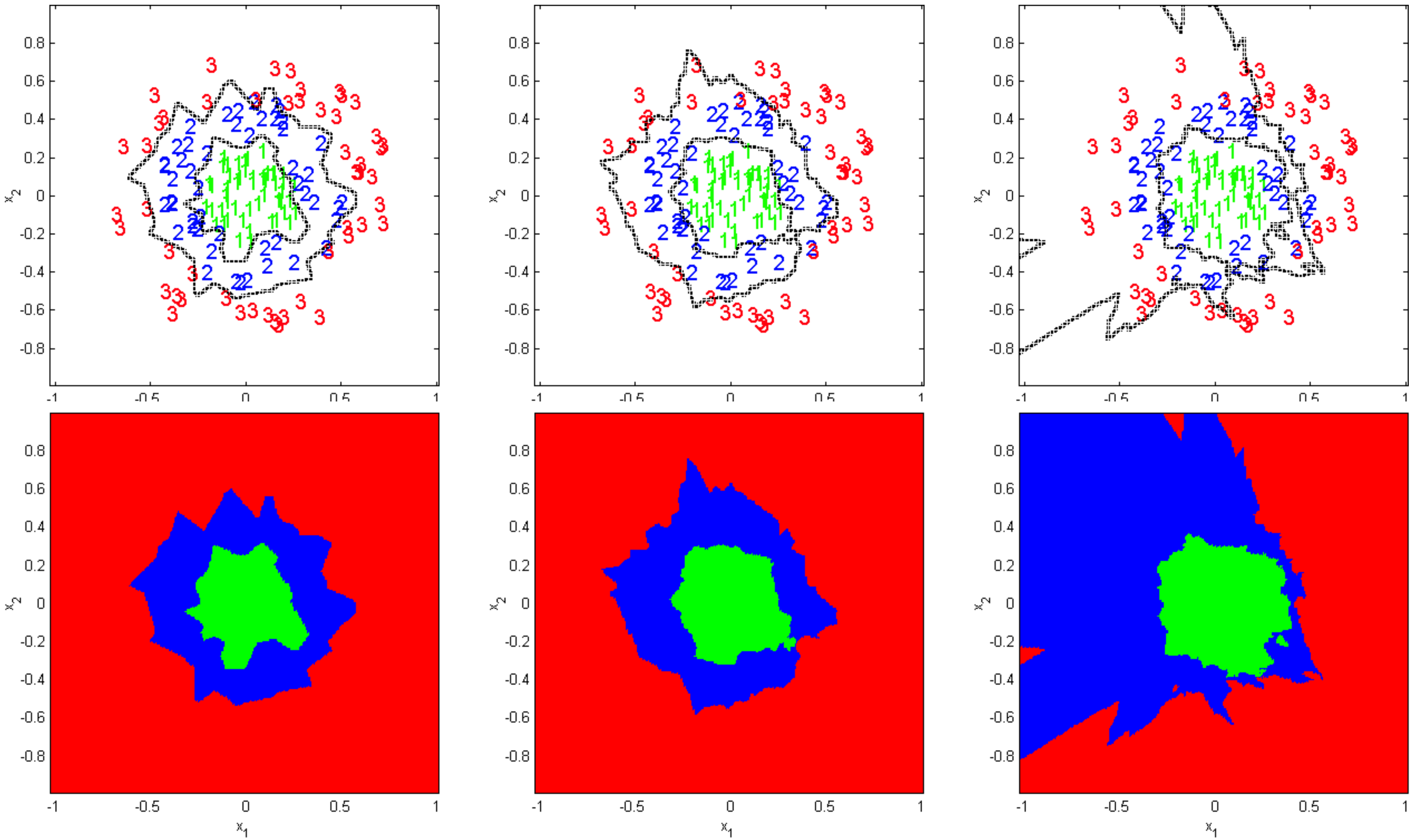
Nearest neighbors sensitive to mis-labeled data ("class noise"). Solution?





kNN versus 1NN

1-NN | 5-NN | 20-NN



k-Nearest Neighbors

How do we choose k?

- Larger k may lead to better performance
- But if we set k too large we may end up looking at samples that are not neighbors (are far away from the query)
- We can use cross-validation to find k
- Rule of thumb is $k < sqrt(n)$, where n is the number of training examples

k-Nearest Neighbors: Issues & Remedies

- If some attributes (coordinates of **x**) have larger ranges, they are treated as more impor
  - normalize scale
    - Simple option: Linearly scale the range of each feature to be, e.g., in range [0,1]
    - Linearly scale each dimension to have 0 mean and variance 1 (compute mean μ and variance σ^2 for an attribute x_j and scale: $(x_j - m)/\sigma$)
  - be careful: sometimes scale matters
- Irrelevant, correlated attributes add noise to distance measure
  - eliminate some attributes
  - or vary and possibly adapt weight of attributes

29

- Non-metric attributes (symbols)
  - Hamming distance

k-Nearest Neighbors: Issues & Remedies

Expensive at test time: To find one nearest neighbor of a query point **x**, we must compute the distance to all N training examples. Complexity: $O(kdN)$ for kNN

- Use subset of dimensions
- Pre-sort training examples into fast data structures (e.g., kd-trees)
- Compute only an approximate distance (e.g., LSH)
- Remove redundant data (e.g., condensing)

Storage Requirements: Must store all training data

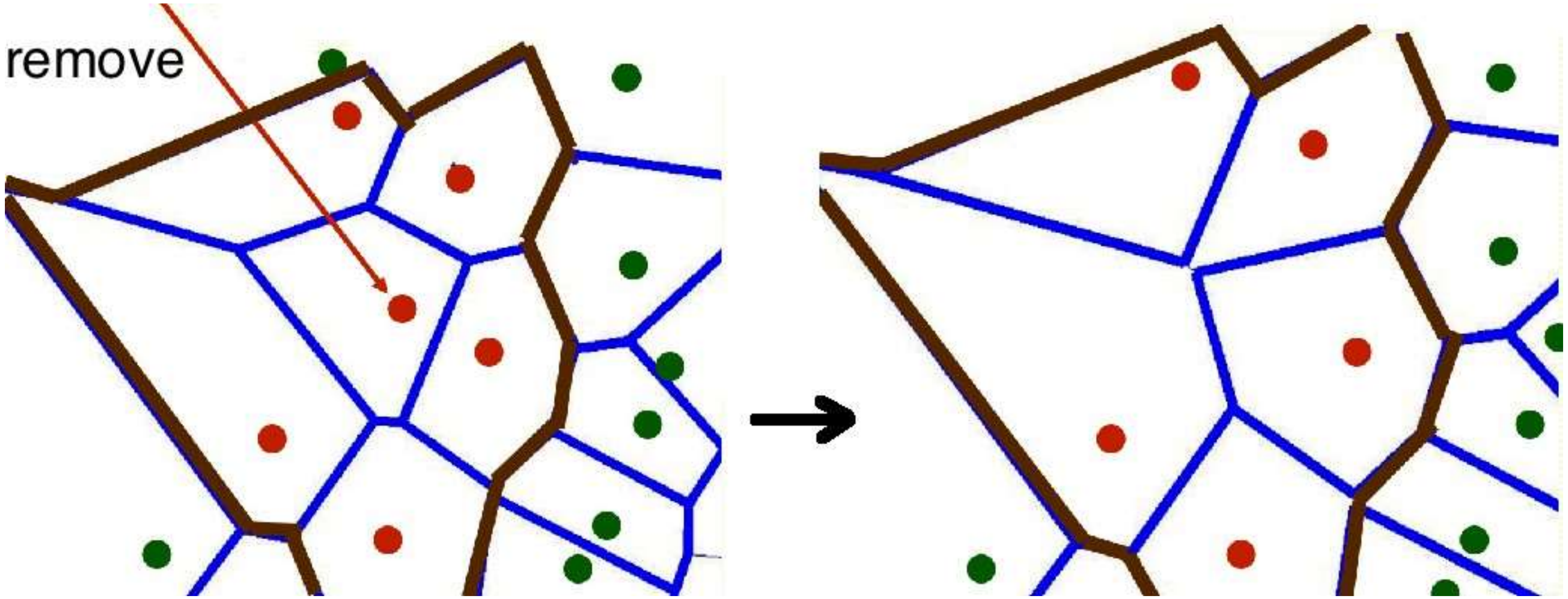
- Remove redundant data (e.g., condensing)
- Pre-sorting often increases the storage requirements

High Dimensional Data: "Curse of Dimensionality"

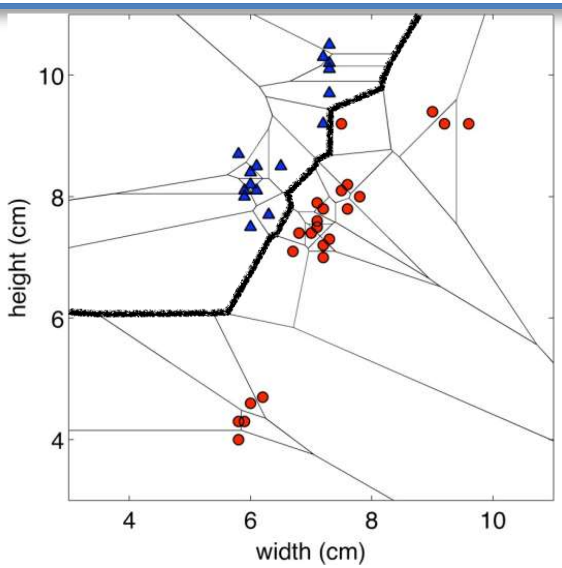
- Required amount of training data increases exponentially with dimension
- Computational cost also increases

k-Nearest Neighbors Remedies: Remove Redundancy

If all Voronoi neighbors have the same class, a sample is useless, remove it



K-NN Summary



Naturally forms complex decision boundaries; adapts to data density

If we have lots of samples, kNN typically works well

Problems:

- Sensitive to class noise
- Sensitive to scales of attributes
- Distances are less meaningful in high dimensions
- Scales linearly with number of examples

Nearest Neighbor Classification…Scaling issues

- Scaling issues
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 90lb to 300lb
    - income of a person may vary from $10K to $1M

Nearest neighbor Classification…

- k-NN classifiers are lazy learners
  - It does not build models explicitly
  - Unlike eager learners such as decision tree induction and rule-based systems

How to handle categorical variables in KNN?

Create dummy variables out of a categorical variable and include them instead of original categorical variable. Unlike regression, create k dummies instead of (k-1).

For example, a categorical variable named "Department" has 5 unique levels / categories. So we will create 5 dummy variables. Each dummy variable has 1 against its department and else 0.

How to find best K value?

Cross-validation is a smart way to find out the optimal K value. It estimates the validation error rate by holding out a subset of the training set from the model building process.

Cross-validation (let's say 10-fold validation) involves randomly dividing the training set into 10 groups, or folds, of approximately equal size. 90% data is used to train the model and remaining 10% to validate it.
The misclassification rate is then computed on the 10% validation data. This procedure repeats 10 times. Different group of observations are treated as a validation set each of the 10 times. It results to 10 estimates of the validation error which are then averaged out.

Using Euclidean distance, apply KNN to predict tomato sweet 6, crunch =4 to which category of food?

Ingredient	SWEET	CRUNCH	FOOD TYPE
GRAPE	8	5	fruit
Greenbean	3	7	vegetable
Nuts	3	6	pROTEIN
Orange	7	3	fruit

D(tomato,grape)=sqrt((6-8)^2 +(4-5)^2)=2.2
D(tomato,greenbeans)= 4.2
D(tomato,Nuts)= 3.6
D(tomato,orange)=1.4
Since d(tomato from orange is minimum therefore tomato will belong to fruit type category

Apply Manhattan distance

Suppose we have height, weight and T-shirt size of some customers and we need to predict the T-shirt size of a new customer given only height and weight information we have. Data including height, weight and T-shirt size information is shown below -

New customer named 'Monica' has height 161cm and weight 61kg.?Euclidean?Manhattan?

Euclidean :

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

Manhattan / city - block :

$$d(x, y) = \sum_{i=1}^{m} |x_i - y_i|$$

New customer named 'Monica' has height 161cm and weight 61kg using Euclidean/Manhattan predict her T Shirt size?k=5

Height (in cms)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

```
>>> X = [[0], [1], [2], [3]]

>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[0.66666667 0.33333333]]
```

getAccuracy function that sums the total correct predictions and returns the accuracy as a percentage of correct classifications.

```
testSet = [[1,1,1,'a'], [2,2,2,'a'], [3,3,3,'b']]
predictions = ['a', 'a', 'a']
accuracy = getAccuracy(testSet, predictions)
print(accuracy)
```

```
def getAccuracy(testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
          if testSet[x][-1] is predictions[x]:
                        correct += 1
        return (correct/float(len(testSet))) *
100.0
```