Indian Institute of Information Technology, Allahabad

Introduction to Machine Learning

k-Nearest Neighbors

By

Dr. Shiv Ram Dubey
Associate Professor
Computer Vision and Biometrics Lab
Department of Information Technology
Indian Institute of Information Technology, Allahabad
Web: https://profile.iiita.ac.in/srdubey/

DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Parametric vs. Non-Parametric Learning

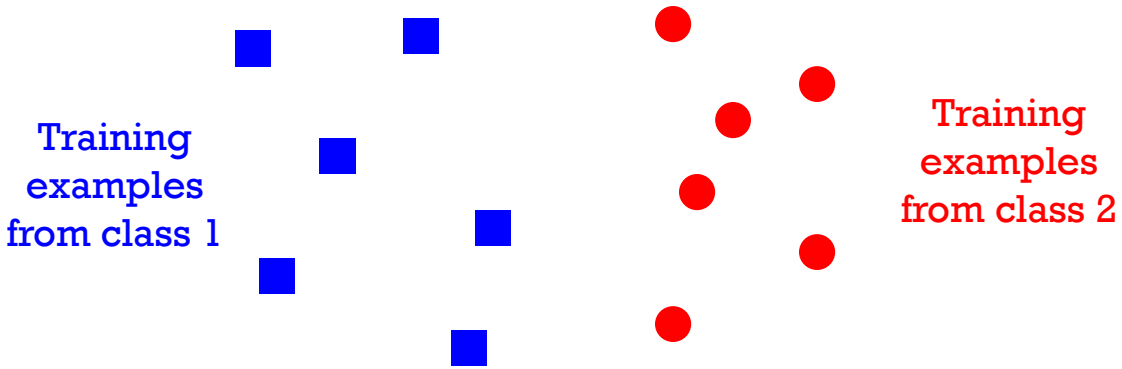
- The algorithms we have seen so far are **parametric**
  - Assume the model family has the form $\{ f_{\mathrm{w}} \mid \mathrm{w} \in \mathbb{R}^d \}$
- Not all model families have this form!
- **Non-parametric models**
  - Very high capacity model families that can fit "arbitrary" functions

Parametric vs. Non-Parametric Learning

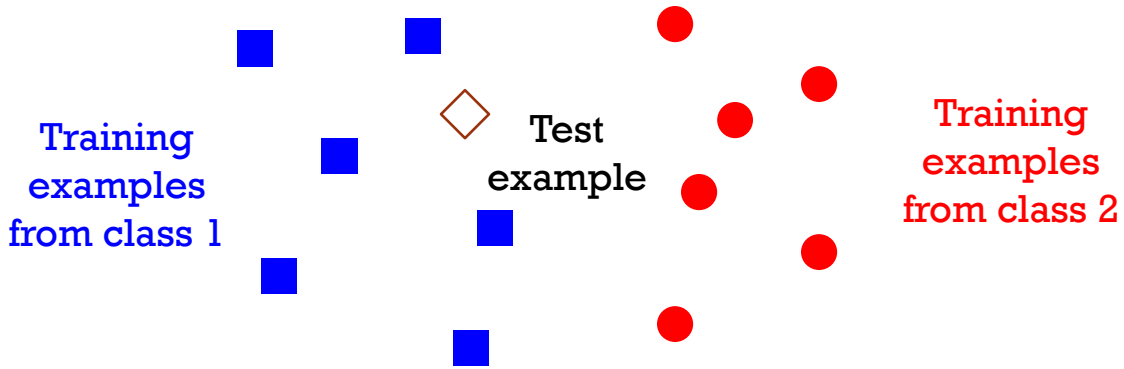
- Parametric
  - The number of parameters is fixed (weights)
  - Data is compressed into weights
  - Fixed complexity irrespective of number of training sample
  - Example: Linear Regression, Logistic Regression, …

- Non-Parametric
  - Model complexity increases when N increases
  - No fixed parametric form
  - Example: K-Nearest Neighbor, Decision Tree, …

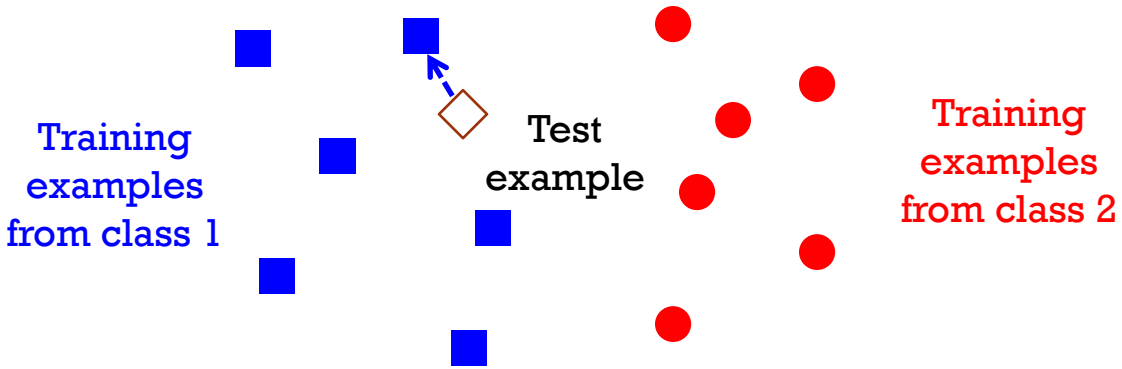
Classifiers: Nearest Neighbor



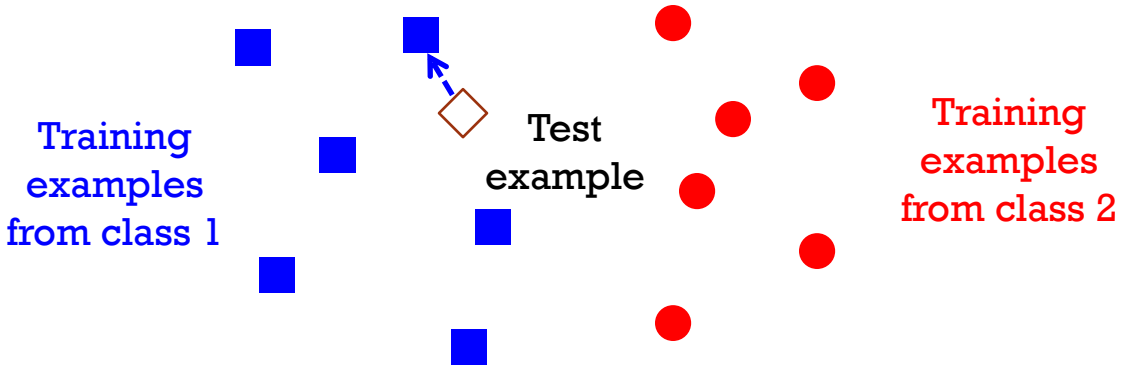
Classifiers: Nearest Neighbor



Classifiers: Nearest Neighbor



Classifiers: Nearest Neighbor

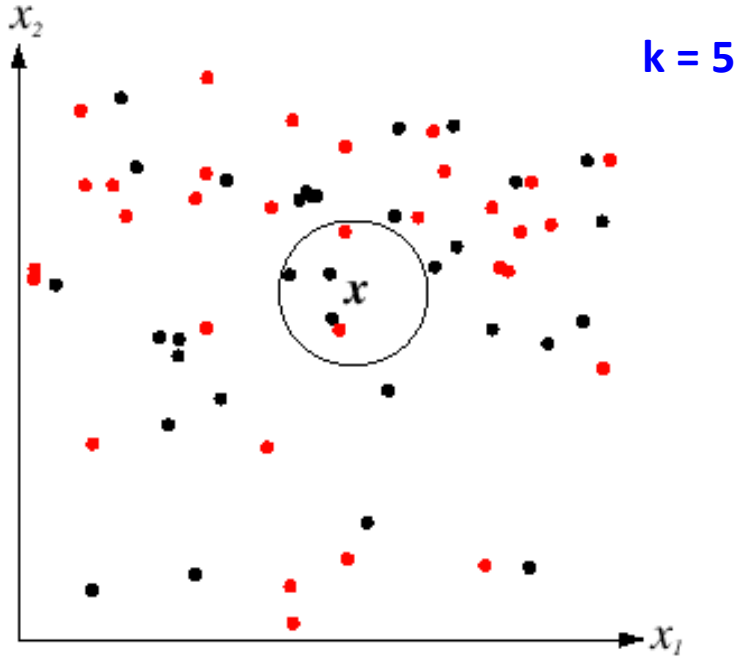


f(**x**) = label of the training example nearest to **x**

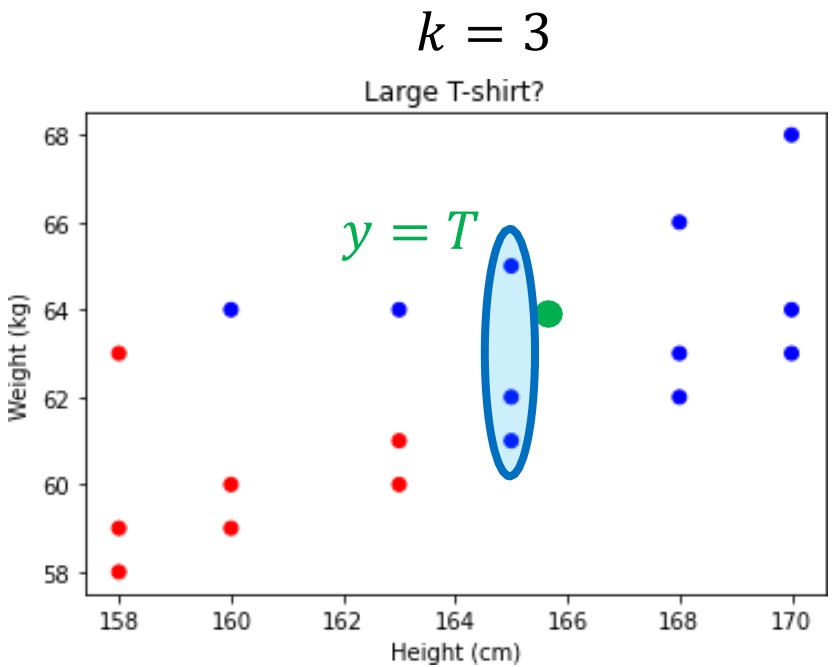
- All we need is a distance function for our inputs
- No training required!

k-Nearest Neighbor Classifier

- For a new point, find the k closest points from training data
- Vote for class label with labels of the k points



kNN Example: T-Shirt Size

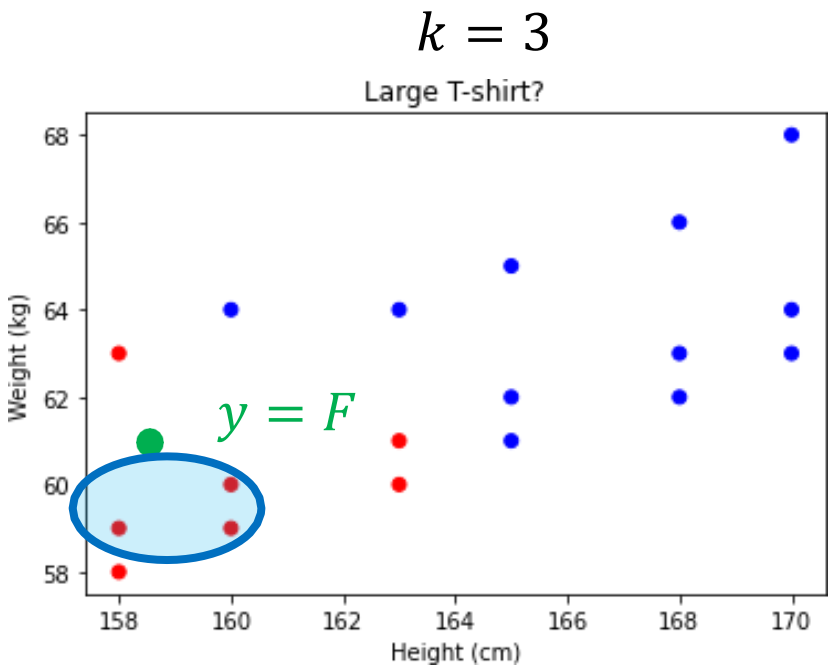


Height (cm)	Weight (kg)	Large (vs Medium) t-shirt?
158	58	F
158	59	F
158	63	F
160	59	F
160	60	F
163	60	F
163	61	F
160	64	T
163	64	T
165	61	T
165	62	T
165	65	T
168	62	T
168	63	T
168	66	T
170	63	T
170	64	T
170	68	T

Inputs x_i Labels y_i

Based on data from https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html

kNN Example: T-Shirt Size

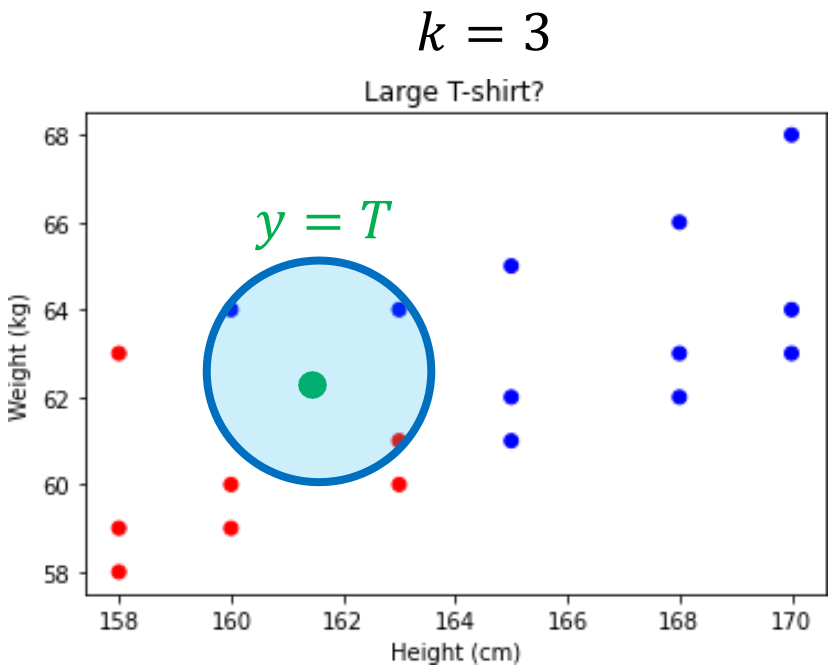


Height (cm)	Weight (kg)	Large (vs Medium) t-shirt?
158	58	F
158	59	F
158	63	F
160	59	F
160	60	F
163	60	F
163	61	F
160	64	T
163	64	T
165	61	T
165	62	T
165	65	T
168	62	T
168	63	T
168	66	T
170	63	T
170	64	T
170	68	T

Inputs x_i Labels y_i

Based on data from https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html

kNN Example: T-Shirt Size

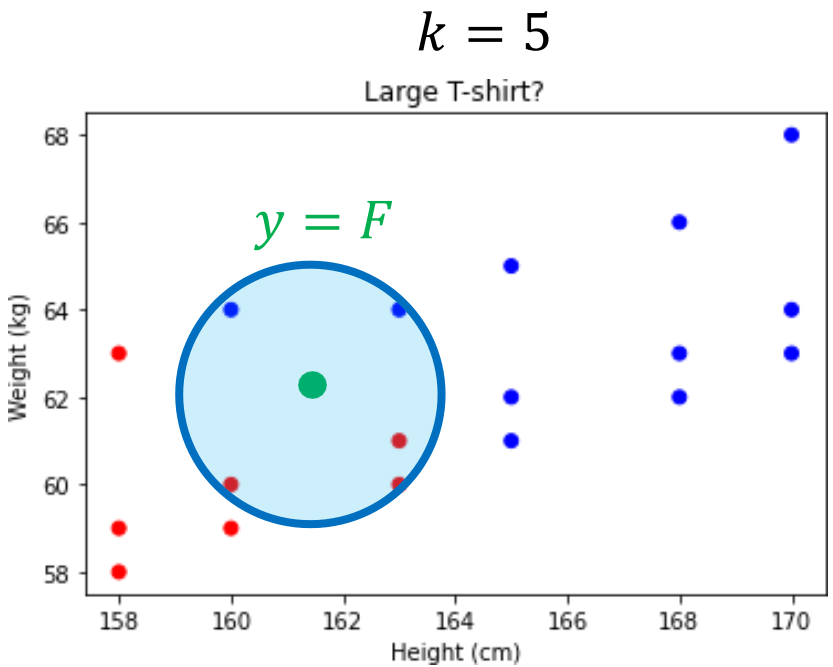


Height (cm)	Weight (kg)	Large (vs Medium) t-shirt?
158	58	F
158	59	F
158	63	F
160	59	F
160	60	F
163	60	F
163	61	F
160	64	T
163	64	T
165	61	T
165	62	T
165	65	T
168	62	T
168	63	T
168	66	T
170	63	T
170	64	T
170	68	T

Inputs x_i Labels y_i

Based on data from https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html

kNN Example: T-Shirt Size

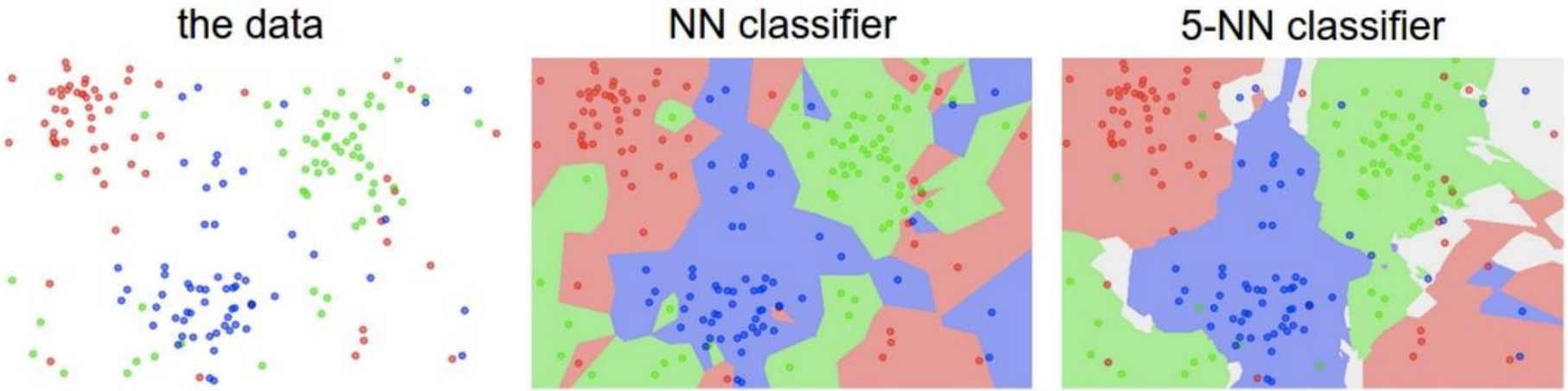


Height (cm)	Weight (kg)	Large (vs Medium) t-shirt?
158	58	F
158	59	F
158	63	F
160	59	F
160	60	F
163	60	F
163	61	F
160	64	T
163	64	T
165	61	T
165	62	T
165	65	T
168	62	T
168	63	T
168	66	T
170	63	T
170	64	T
170	68	T

Inputs x_i Labels y_i

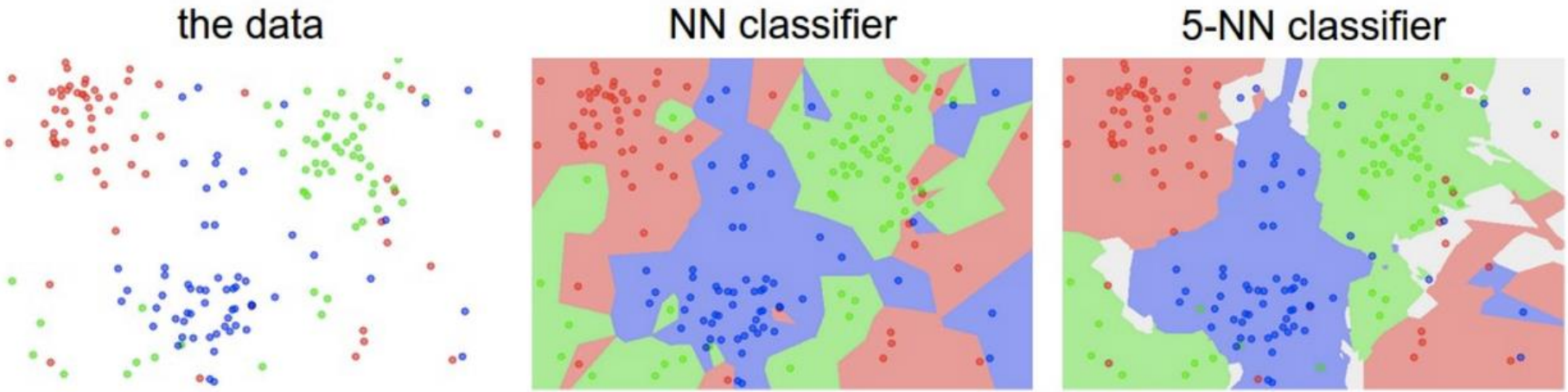
Based on data from https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html

k-Nearest Neighbor Classifier



Credit: cs231n, http://cs231n.github.io/classification/

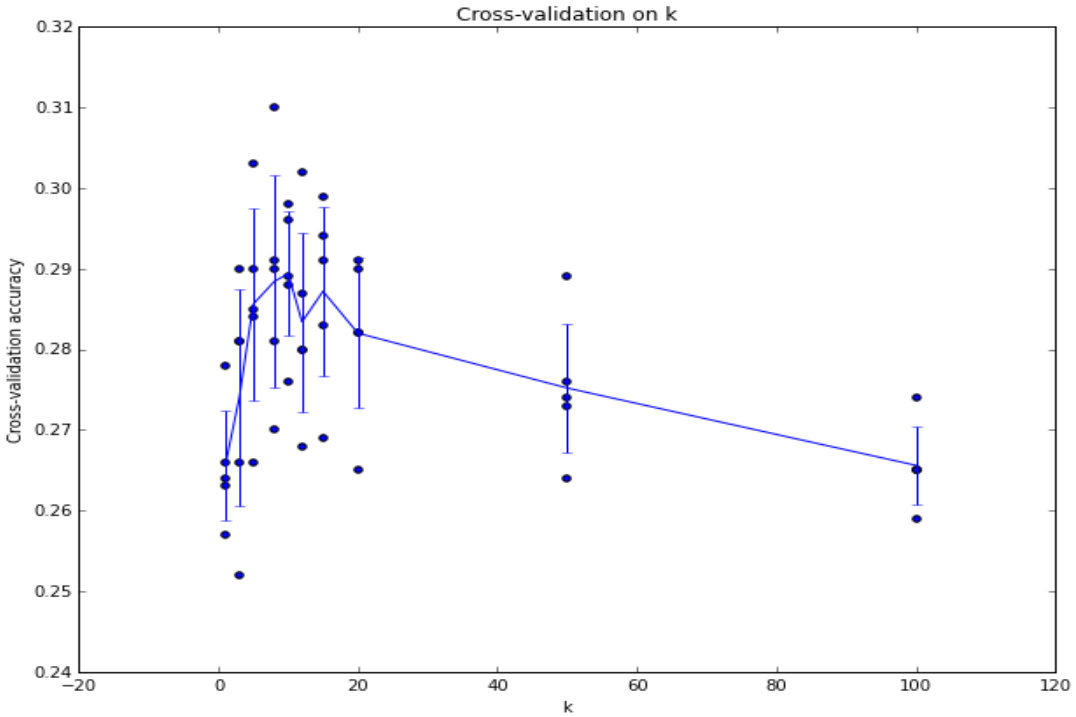
k-Nearest Neighbor Classifier



- Which classifier is more robust to *outliers*?

Credit: cs231n, http://cs231n.github.io/classification/

Choice of k in kNN Classifier



Example of a 5-fold cross-validation run for the parameter **k**. Note that in this particular case, the cross-validation suggests that a value of about **k** = 7 works best on this particular CIFAR10 dataset (corresponding to the peak in the plot).

Credit: cs231n, http://cs231n.github.io/classification/

k-Nearest Neighbors (kNN)

- **Regression:** Given a new input x:
  - **Step 1:** Find k nearest neighbors $\{i_1, \dots, i_k\}$ in the training dataset

$$i_1, \dots, i_k = k\text{ArgMin}_{i \in \{1,\dots,n\}} \text{dist}(x, x_i)$$

  - **Step 2:** Return the average label (i.e., label that occurs most frequently):

$$y = \text{Average } \{y_{i_1}, \dots, y_{k_k}\}$$

k-Nearest Neighbors (kNN)

- **Regression:** Given a new input x:
  - **Step 1:** Find k nearest neighbors $\{i_1, \dots, i_k\}$ in the training dataset

$$i_1, \dots, i_k = k\text{ArgMin}_{i \in \{1,\dots,n\}} \text{dist}(x, x_i)$$

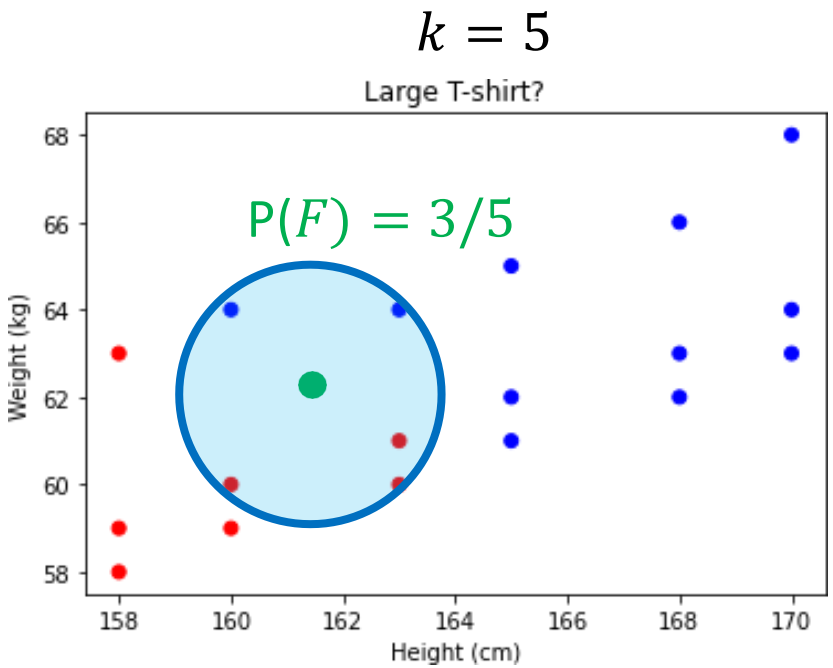
  - **Step 2:** Return the average label (i.e., label that occurs most frequently):

$$y = \text{Average } \{y_{i_1}, \dots, y_{k_k}\}$$

- We can use this approach to get probabilities for classification

$$P(y = c|x) = \frac{1}{k} \sum_{i \in N_k(x)} \mathbf{1}(y_i = c)$$

Example: T-Shirt Size



Height (cm)	Weight (kg)	Large (vs Medium) t-shirt?
158	58	F
158	59	F
158	63	F
160	59	F
160	60	F
163	60	F
163	61	F
160	64	T
163	64	T
165	61	T
165	62	T
165	65	T
168	62	T
168	63	T
168	66	T
170	63	T
170	64	T
170	68	T

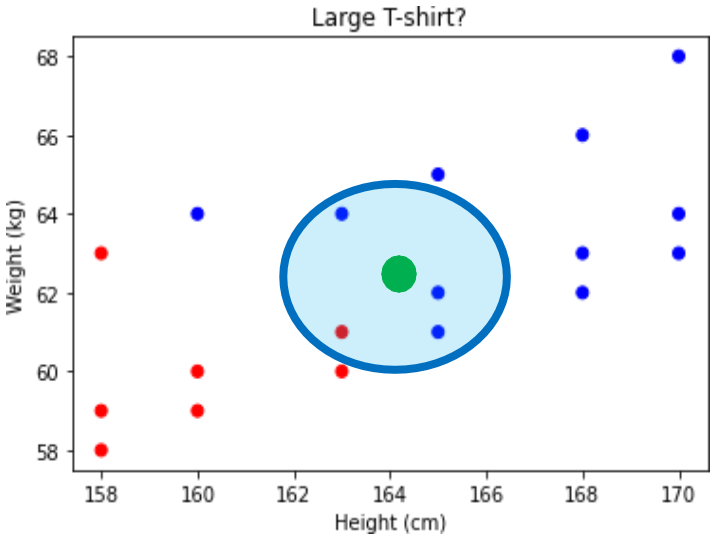
Inputs x_i Labels y_i

Based on data from https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html

Weighted k-Nearest Neighbors (kNN)

- kNN treats all neighbors equally, but maybe closer neighbors are usually more informative?
- We can assign weights to each neighbor

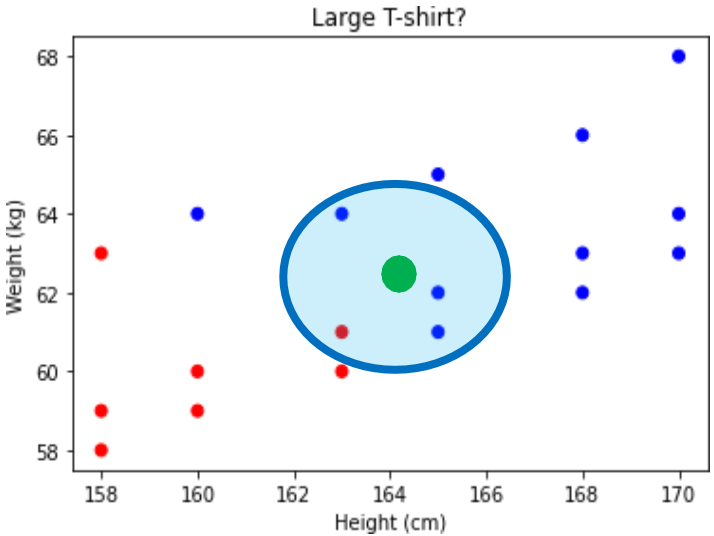
$$P(y = c|x) = \frac{\sum_{i \in N_k(x)} w_i . \mathbf{1}(y_i = c)}{\sum_{i \in N_k(x)} w_i}$$



Weighted k-Nearest Neighbors (kNN)

- kNN treats all neighbors equally, but maybe closer neighbors are usually more informative?
- We can assign weights to each neighbor

$$P(y = c|x) = \frac{\sum_{i \in N_k(x)} w_i . \mathbf{1}(y_i = c)}{\sum_{i \in N_k(x)} w_i}$$



- Kernel Regression: w_i is computed based on a **kernel** function:

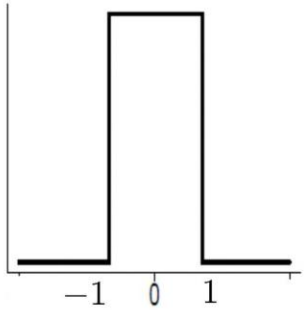
$$w_i = K(d(x, x_i))$$

- (weighted) kNN can be seen as a special case of kernel regression

Kernel Regression

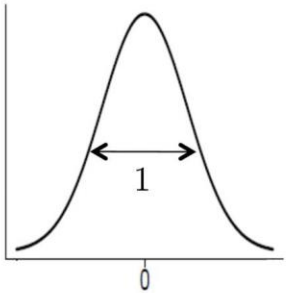
boxcar kernel :

$$K(x) = \frac{1}{2} I(x),$$



Gaussian kernel :

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$



Any kernel function that satisfies

$$K(x) \geq 0,$$

$$\int K(x)dx = 1$$

k-Nearest Neighbors (kNN)

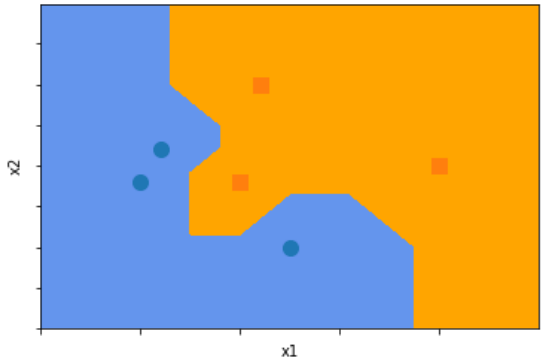
General framework

- **Design decision 1:** What notion of "distance" to use? (e.g., L_2 distance)
- **Design decision 2:** How to aggregate labels? (e.g., majority or average)

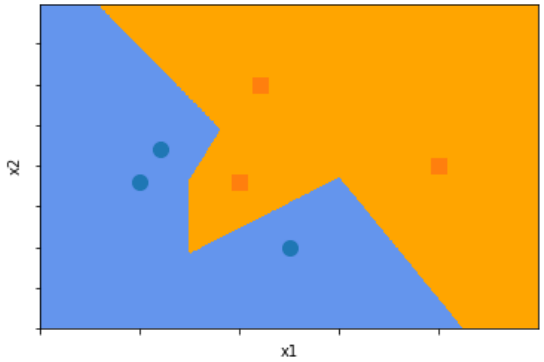
Choice of Distance Function

- **L_1 distance:** $\text{dist}(x, x') = \|x - x'\|_1 = \sum_{j=1}^{d} |x_j - x'_j|$
- **L_2 distance:** $\text{dist}(x, x') = \|x - x'\|_2 = \left(\sum_{j=1}^{d} (x_j - x'_j)^2\right)^{\frac{1}{2}}$
- **L_∞ distance:** $\text{dist}(x, x') = \|x - x'\|_\infty = \max_{j \in \{1,\ldots,d\}} |x_j - x'_j|$
- **L_p distance:** $\text{dist}(x, x') = \|x - x'\|_p = \left(\sum_{j=1}^{d} |x_j - x'_j|^p\right)^{\frac{1}{p}}$

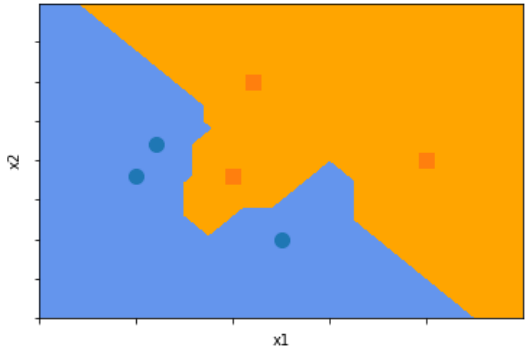
Choice of Distance Function



L_1 distance



L_2 distance



L_∞ distance

The distance affects the decision boundary a lot in high-dimensional space.

Distances for Strings

- **Hamming distance:** Number of characters that are different
  - **Example:** ABCDE vs. AGDDF → Hamming distance $= 3$
  - Assumes strings have the same length

- **Edit distance:** Number of insert/delete/replace operations needed to transform one string into the other
  - **Example:** ROBOT vs. BOT → Edit distance $= 2$
  - Can be computed using dynamic programming

Distances for Strings

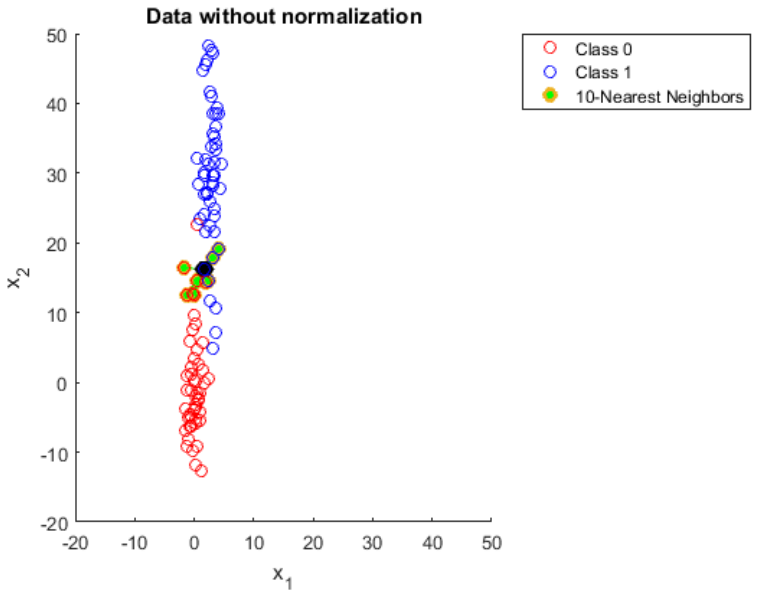
- **Jaccard distance** between sets $1 - \frac{|A \cap B|}{|A \cup B|}$
  - Apply to set of ***n*-grams** (i.e., n-character substrings)
  - **Example:** ROBOT vs. BOT yields

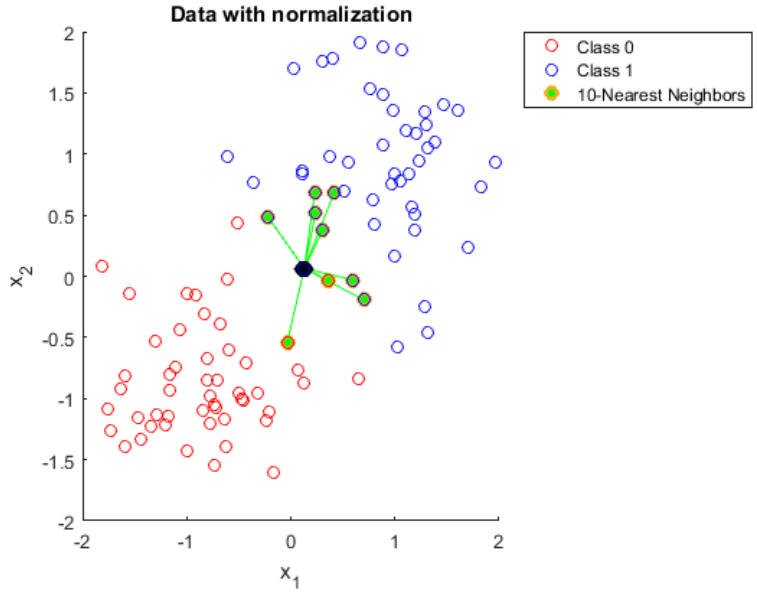
$$A = \{R, RO, ROB, OBO, BOT, OT, T\}$$
$$B = \{B, BO, BOT, OT, T\}$$

  - → Jaccard distance $= 1 - \frac{3}{9} = \frac{2}{3}$

Feature Standardization

- It is very important for kNN!



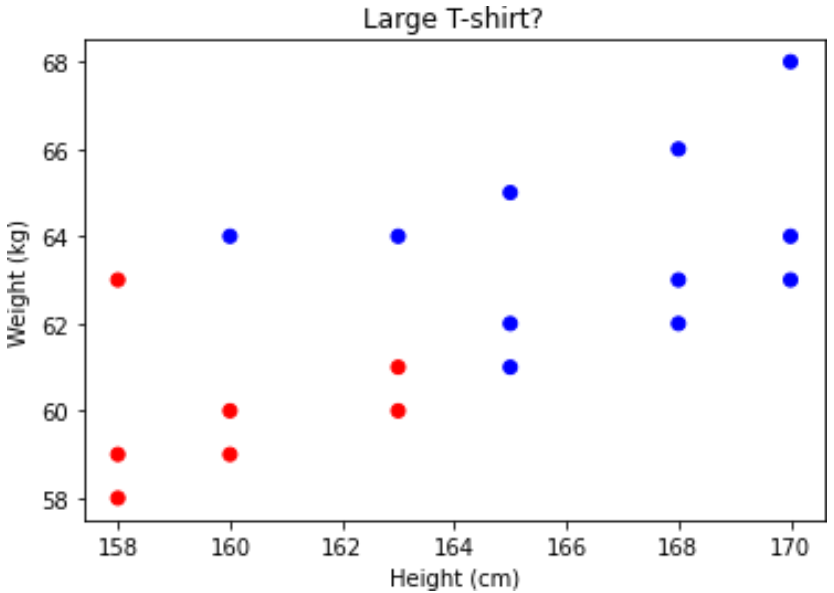


https://stats.stackexchange.com/questions/287425/why-do-you-need-to-scale-data-in-knn

Feature Standardization

- It is very important for kNN!
  - kNN is purely based on distance

Changing cm to m, weight will dominate the distance.



Curse of Dimensionality

- **Example:** Predict acceleration of an object being pushed by a robot
- Features:
  - $x_1 =$ mass
  - $x_2 =$ Force
  - $x_3 =$ color of object
  - $x_4 =$ what the operator ate for breakfast that morning
- When more irrelevant variables, distance function becomes *dominated by irrelevant dimensions* in x
  - Amount of data needed by kNN scales exponentially in dimension

Curse of Dimensionality

- **Example:** Predict acceleration of an object being pushed by a robot
- Features:
  - $x_1 =$ mass
  - $x_2 =$ Force
  - $x_3 =$ color of object
  - $x_4 =$ what the operator ate for breakfast that morning
- When more irrelevant variables, distance function becomes *dominated by irrelevant dimensions* in x
  - Amount of data needed by kNN scales exponentially in dimension
- Adding more dimensions makes a lot of things counterintuitive
- For kNN, nearest neighbors become very far apart, and of similar distance, making it an **unreliable predictor**

Scalability

Scaling: Naively, must compute n distances between pairs of d-dimensional vectors to compute kNN

Complexity:

$$O(nd)$$

- n = 1B
- d = 1024
- …

This is very different from parametric models!

Scalability

- **Indexing**
  - Use kd-trees and other multidimensional indices to capture the training data
  - Each lookup is O(log *n*) but on disk
- **Parallelism**
  - Use multiple cores, and compare against in-memory data
- **Approximation**
  - Compare against a sample, not all of the training data
  - See, e.g., https://www.kaggle.com/code/pawanbhandarkar/knn-vs-approximate-knn-what-s-the-difference/notebook

Classifiers: K-Nearest Neighbor

“Non-parametric” classifier: the entire training set is essentially the model parameters.

Classifiers: K-Nearest Neighbor

"Non-parametric" classifier: the entire training set is essentially the model parameters.

Pros:

- **Very fast at training** time
- **Flexible**: all it requires is a way to compute similarity or distances between pairs of features. Applies to many different kinds of features.
- Works with **any number of classes**.
- Works well in practice for large datasets (but see cons)
- **Modern usage:** Look up nearest examples according to "learned" features (in the context of deep learning)

Classifiers: K-Nearest Neighbor

"Non-parametric" classifier: the entire training set is essentially the model parameters.

Cons:

- The classifier must ***remember*** **all of the training data** and store it for future comparisons with the test data.
- This is **space inefficient** because datasets may easily be gigabytes in size.
- **Slow at test time** (need to compute distances between test example and every training example)
- Optimum value of **K is not known**.

Nearest Neighbor vs. Linear Classifiers

- NN pros:
  - Simple to implement
  - Decision boundaries not necessarily linear
  - Works for any number of classes
  - *Nonparametric* method
- NN cons:
  - Need good distance function
  - Slow at test time, Memory in-efficient
- Linear pros:
  - Low-dimensional *parametric* representation
  - Very fast at test time
- Linear cons:
  - How to train the linear function?
  - What if data is not linearly separable?

Acknowledgement

Most of the slides are based on the ML course of:

- CS231n, Stanford University
- Mingmin Zhao and Jiatao Gu, University of Pennsylvania

THANK YOU