University of Colorado **Boulder**

Department of Computer Science

CSCI 5622: Machine Learning

Chenhao Tan

Lecture 4: K-Nearest Neighbors/Perceptron I

Slides adapted from Chris Ketelsen, Jordan Boyd-Graber, and Noah Smith

Administrivia

- HW 1 due on Friday
- Change in homework deadlines
- Office hour schedules
- Wednesday is a hands-on day, so bring your laptops

Learning Objectives

- Understand K-nearest neighbor classifiers
- Understand the parametric models and linear classifiers

K-Nearest Neighbors

- Overview
- Weighted KNN
- Performance Guarantee
- Curse of Dimensionality

K-nearest neighbors

Find the K-nearest neighbors of x in training data and predict the majority label of those K points.

K-nearest neighbors

Find the K-nearest neighbors of x in training data and predict the majority label of those K points.

$$h(x) = \arg \max_{y \in \{+1,-1\}} \sum_{(x',y') \in \mathrm{NN}(\mathrm{x},\mathrm{S}_{\mathrm{train}},\mathrm{k})} I(y = y')$$

K-nearest neighbors

Find the K-nearest neighbors of x in training data and predict the majority label of those K points.

$$h(x) = \arg \max_{y \in \{+1,-1\}} \sum_{(x',y') \in \mathrm{NN}(\mathrm{x},\mathrm{S}_{\mathrm{train}},\mathrm{k})} I(y = y')$$

Assumptions in the algorithm: nearby instances share similar labels.

Hyperparameters in the algorithm

- Distance function

Hyperparameters in the algorithm

- Distance function

Discrete

$$d(x_1, x_2) = 1 - \frac{|x_1 \cap x_2|}{|x_1 \cup x_2|} \quad (1)$$

Continuous

Euclidean distance

$$d(x_1, x_2) = \|\vec{x}_1 - \vec{x}_2\|_2 \quad (2)$$

Hyperparameters in the algorithm

- Distance function

Discrete

$$d(x_1, x_2) = 1 - \frac{|x_1 \cap x_2|}{|x_1 \cup x_2|} \tag{1}$$

Continuous

Euclidean distance

$$d(x_1, x_2) = \|\vec{x}_1 - \vec{x}_2\|_2 \tag{2}$$

Manhattan distance

$$d(x_1, x_2) = \|\vec{x}_1 - \vec{x}_2\|_1 \tag{3}$$

Hyperparameters in the algorithm

- Distance function

Discrete

$$d(x_1, x_2) = 1 - \frac{|x_1 \cap x_2|}{|x_1 \cup x_2|} \tag{1}$$

Continuous

Euclidean distance

$$d(x_1, x_2) = \|\vec{x}_1 - \vec{x}_2\|_2 \tag{2}$$

Manhattan distance

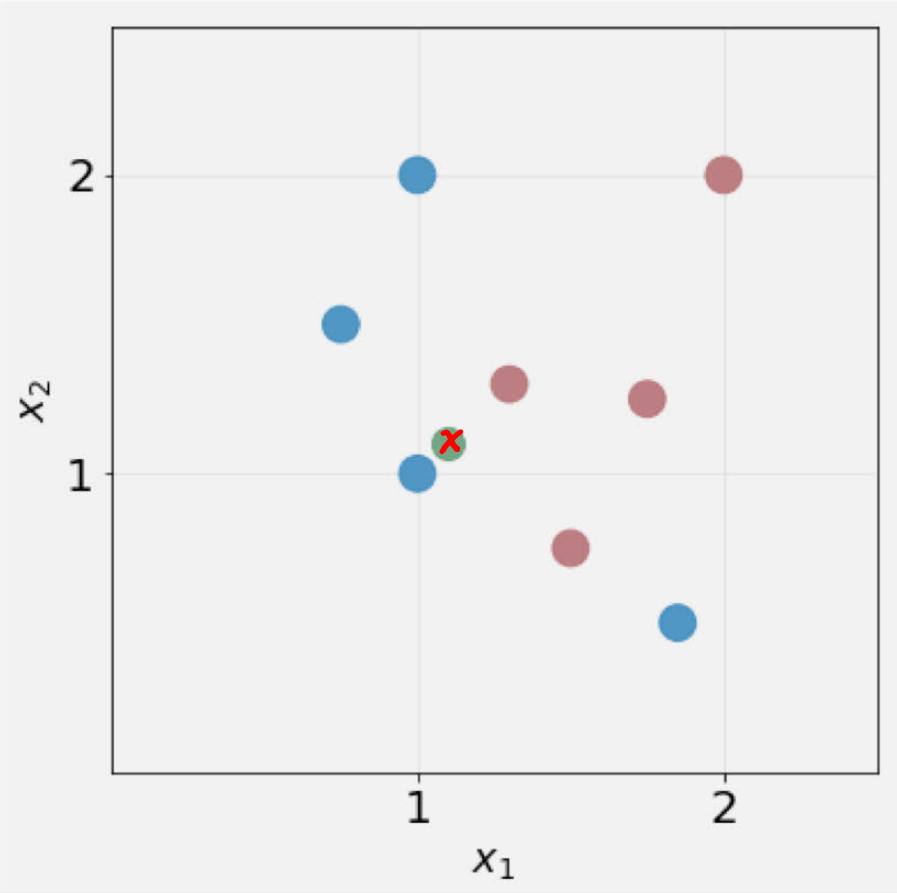
$$d(x_1, x_2) = \|\vec{x}_1 - \vec{x}_2\|_1 \tag{3}$$

- Number of nearest neighbors k

Decision tree vs. KNN

- Decision tree uses one feature at a time, and splits the data to reduce entropy.
- KNN takes a geometry perspective and quickly finds the closest points in the training data.

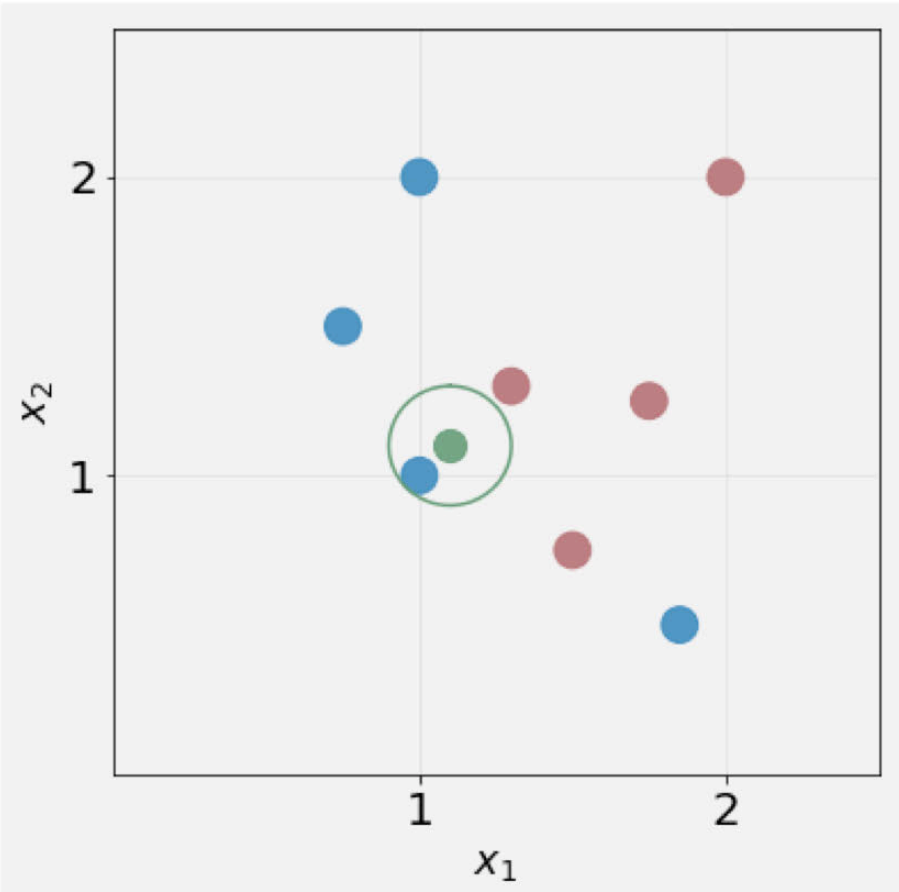
KNN Classification



Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

- 1-NN predicts: blue
- 2-NN predicts:
- 5-NN predicts:

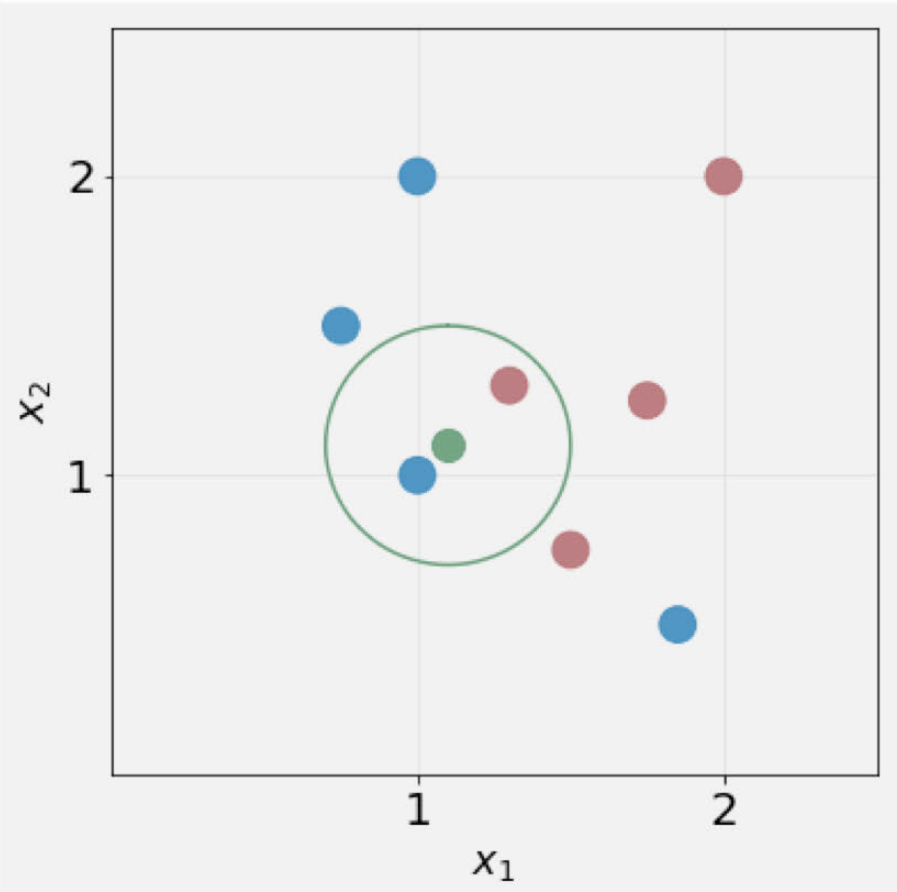
KNN Classification



Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

- 1-NN predicts:
- 2-NN predicts:
- 5-NN predicts:

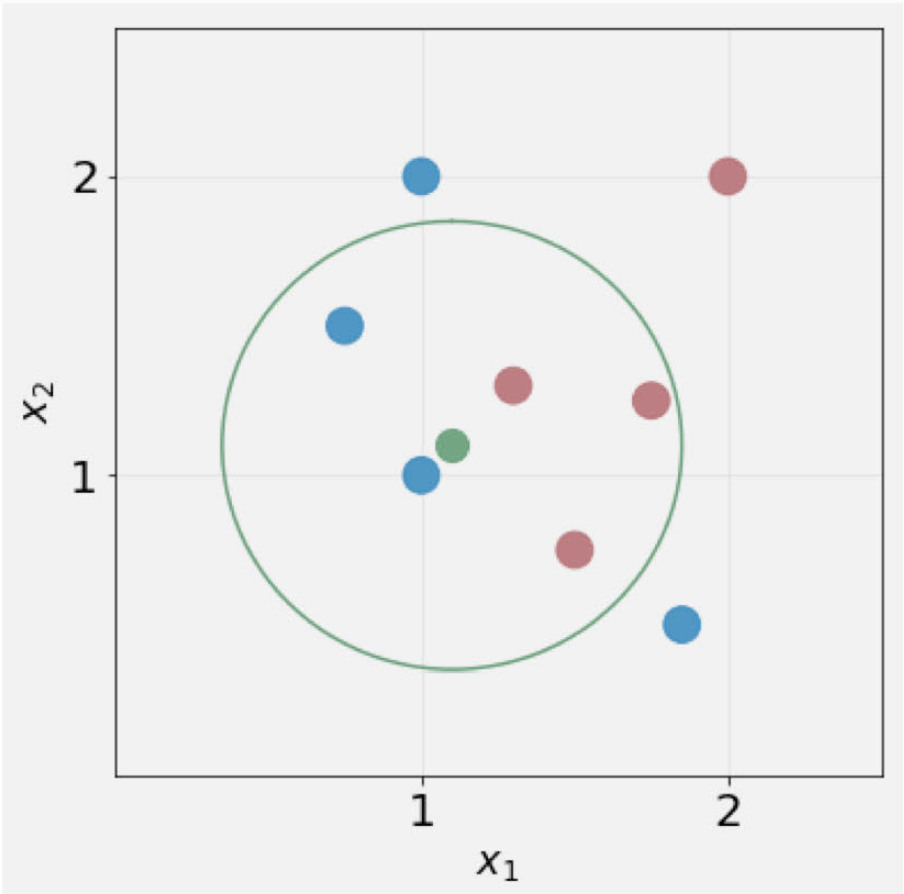
KNN Classification



Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

- 1-NN predicts: blue
- 2-NN predicts:
- 5-NN predicts:

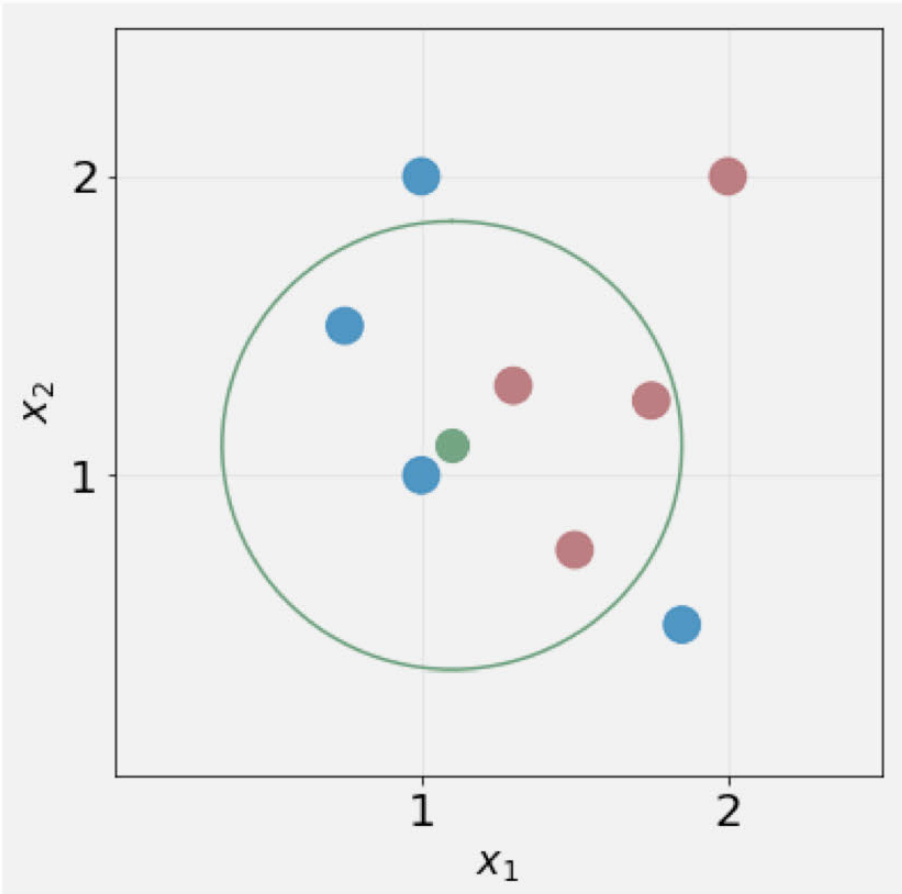
KNN Classification



Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

- 1-NN predicts: blue
- 2-NN predicts: unclear
- 5-NN predicts:

KNN Classification

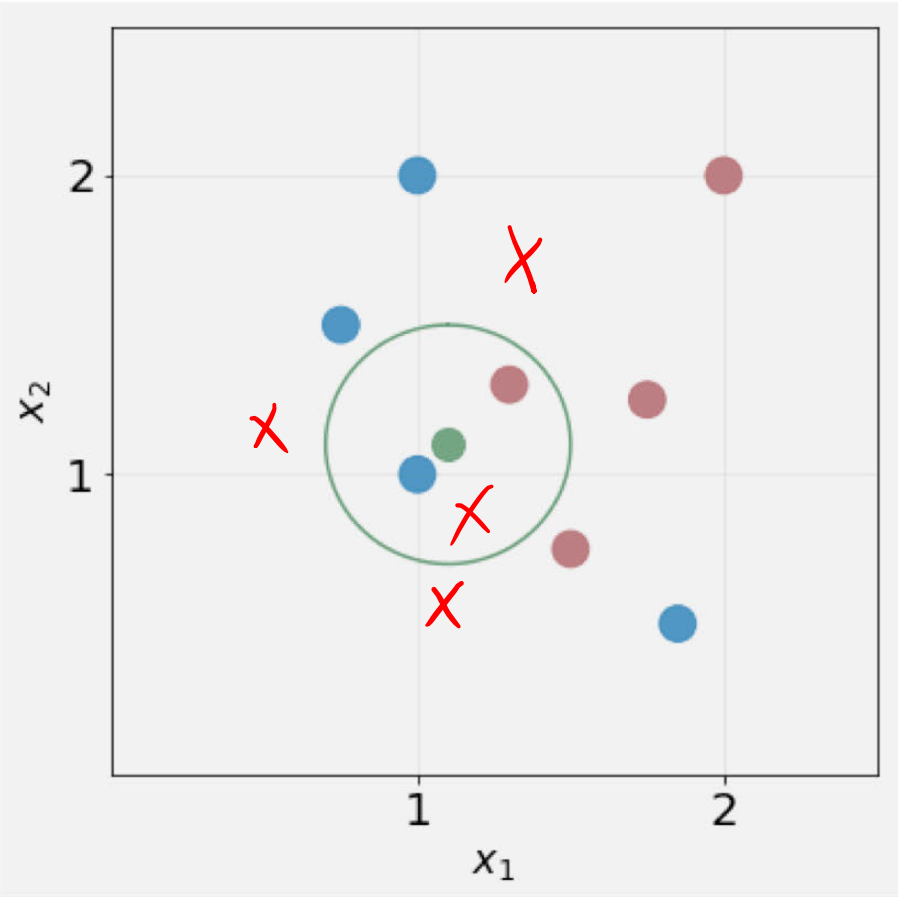


Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

- 1-NN predicts: blue
- 2-NN predicts: unclear
- 5-NN predicts: red

KNN Classification

What about ties?

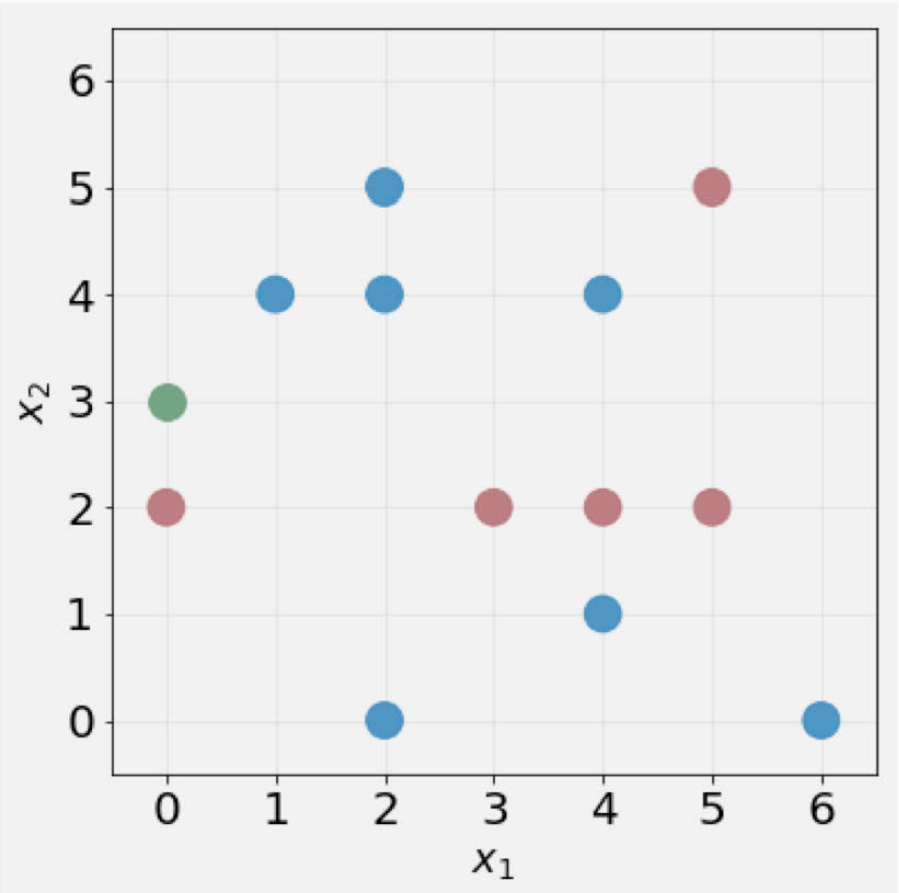


One reasonable strategy:

- reduce k by 1 and try again
- repeat until the tie is broken

KNN Classification

Another example

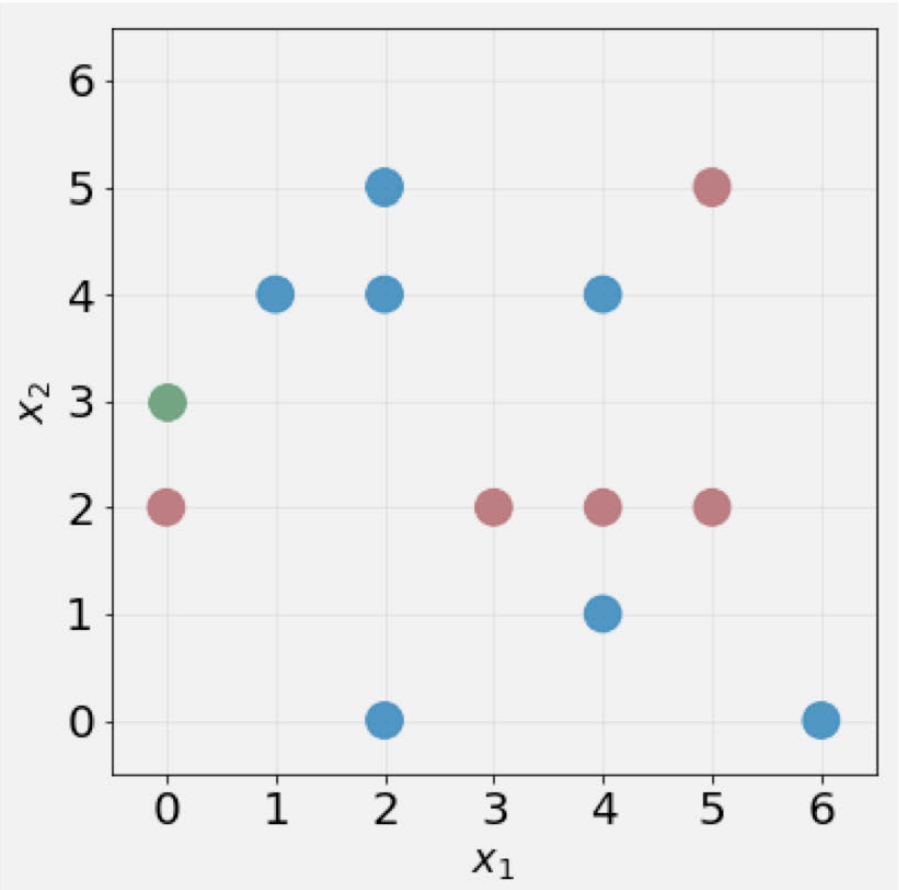


Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

$k = 1$

KNN Classification

Another example

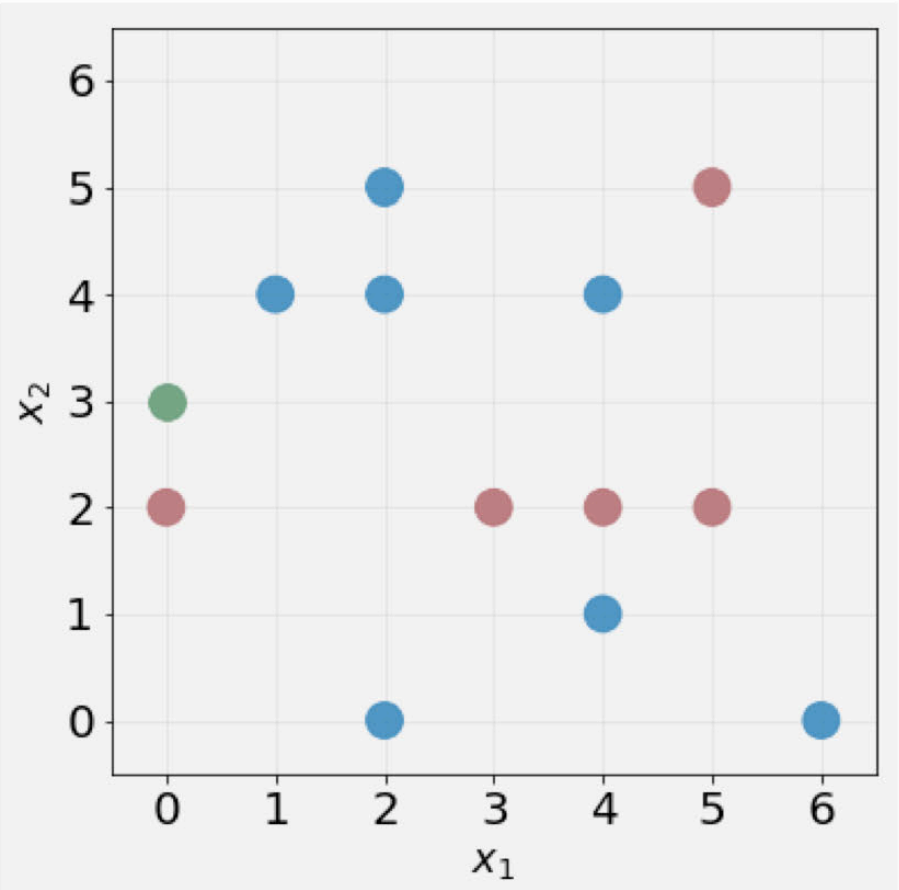


Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

$k = 2$

KNN Classification

Another example

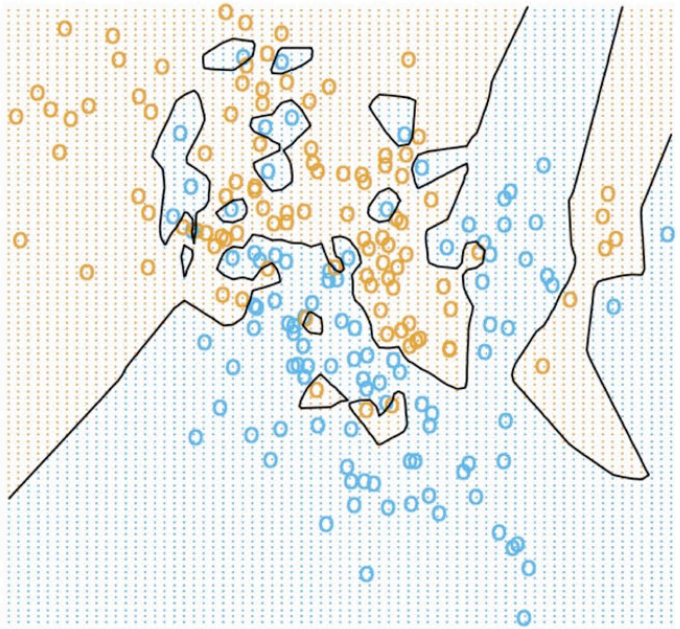


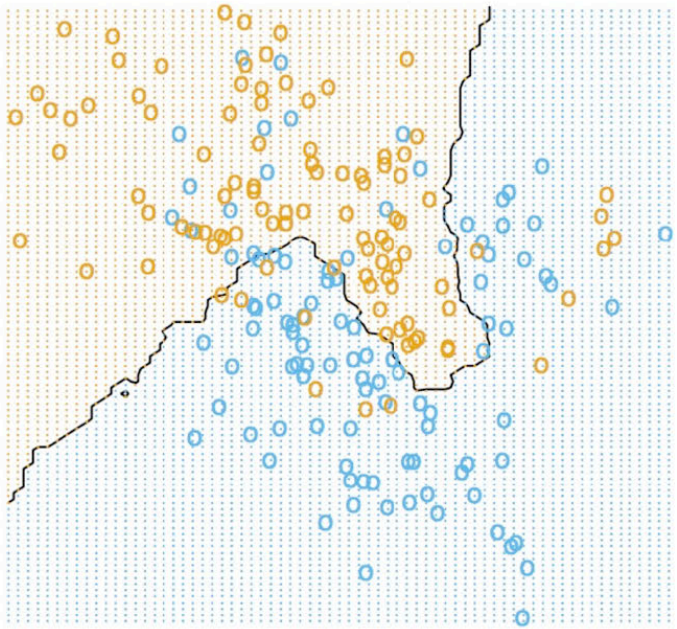
Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

$k = 3$

K-nearest neighbors

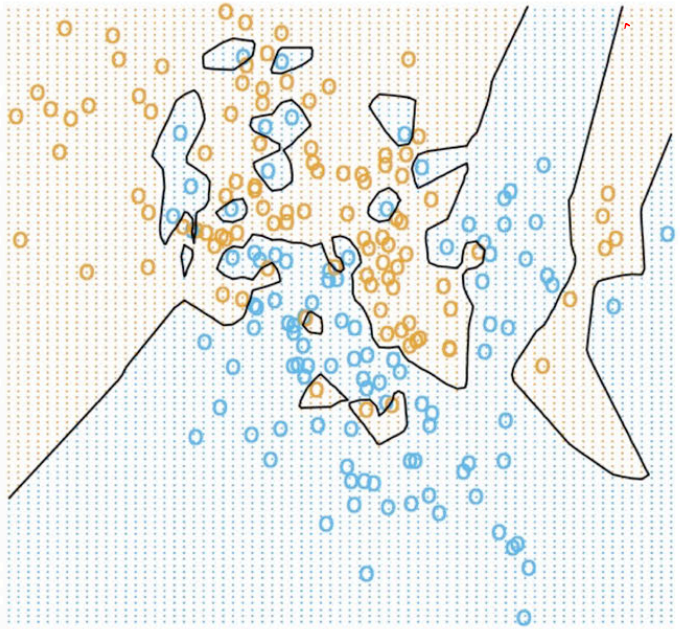
Recall the danger of overfitting



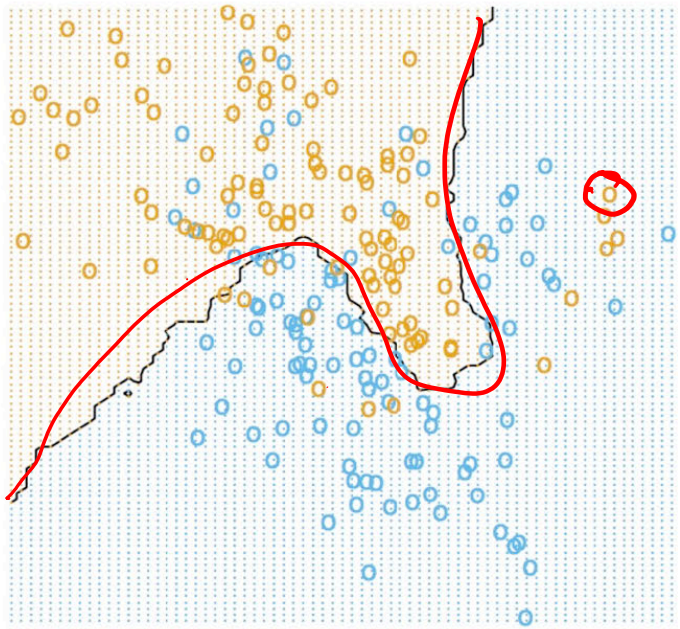


K-nearest neighbors

Recall the danger of overfitting



$k = 1$

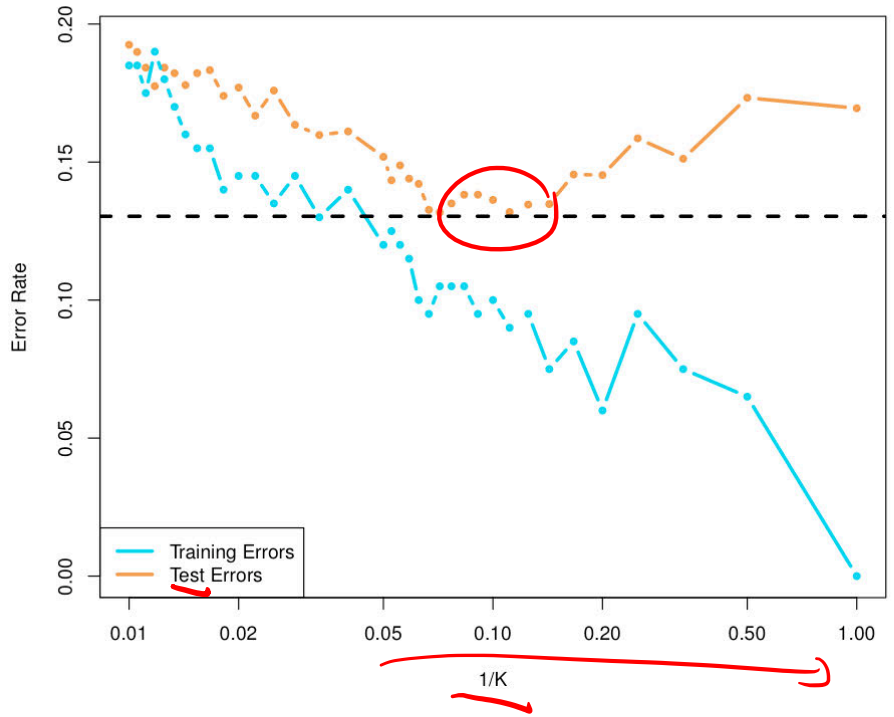


$k = 15$

K-nearest neighbors

Choose optimal k by varying k and computing error rate on the development set.

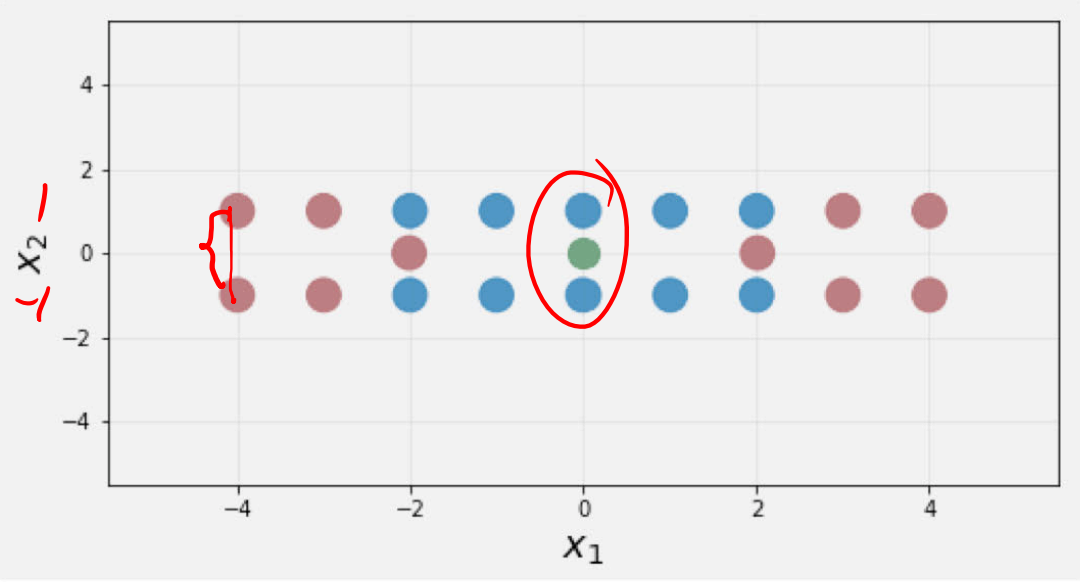
- Plot vs. $\frac{1}{k}$
- Lower k leads to more flexibility



Feature normalization

Practical tips:

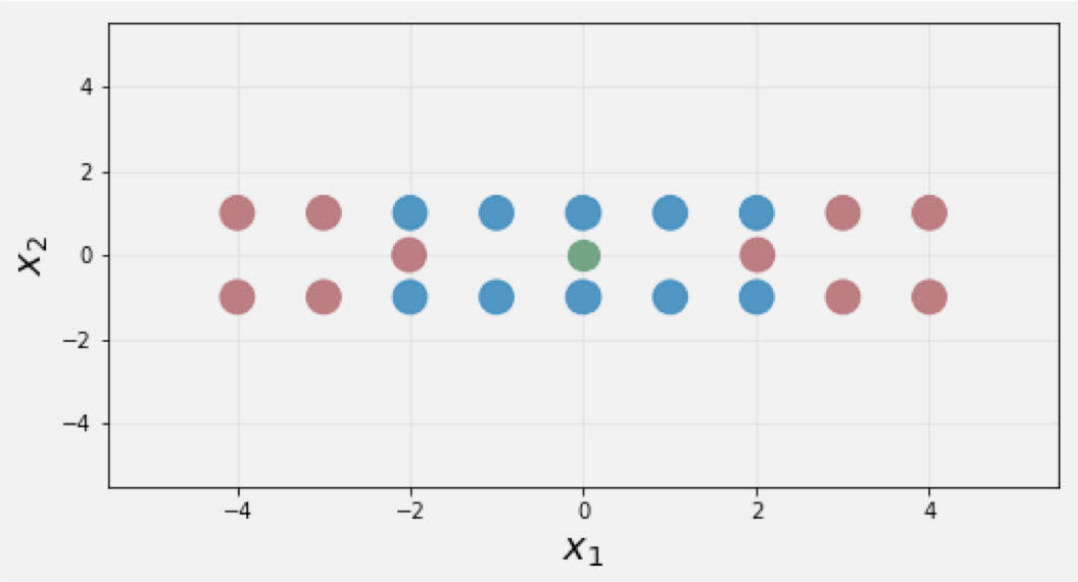
- Important to normalize features to similar sizes
- Consider doing 2-NN on unscaled and scaled data

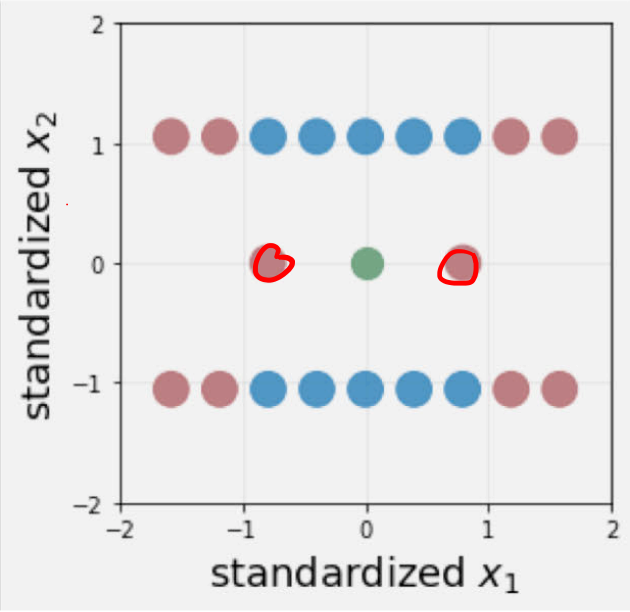


Feature normalization

Practical tips:

- Important to normalize features to similar sizes
- Consider doing 2-NN on unscaled and scaled data



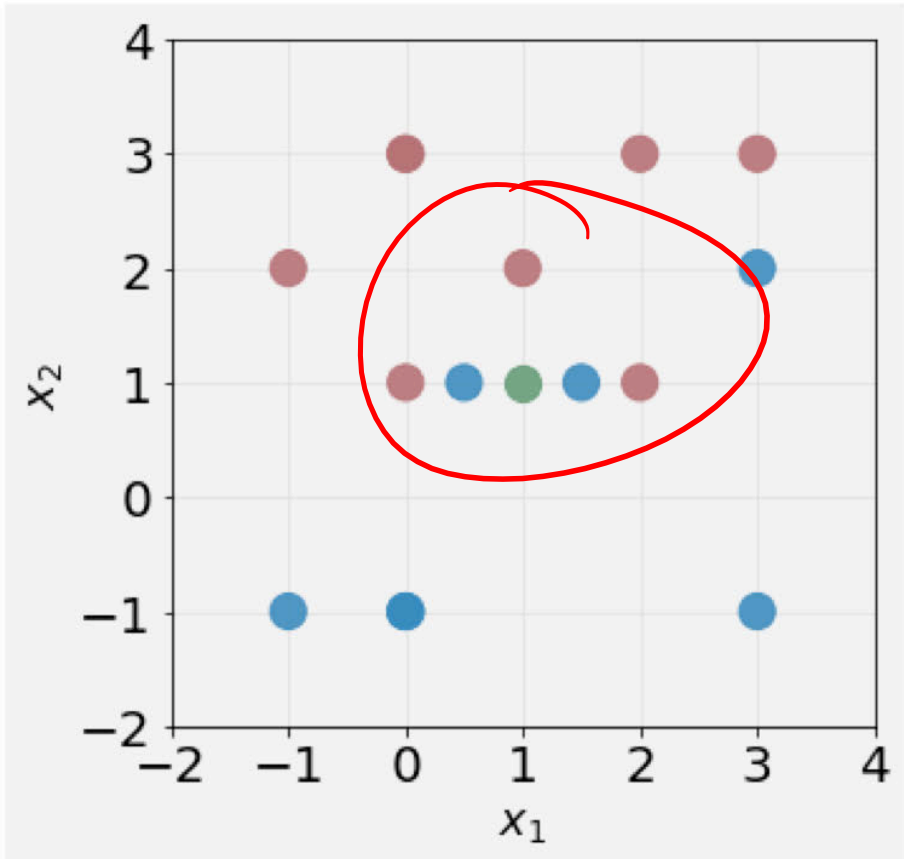


K-Nearest Neighbors

- Overview
- Weighted KNN
- Performance Guarantee
- Curse of Dimensionality

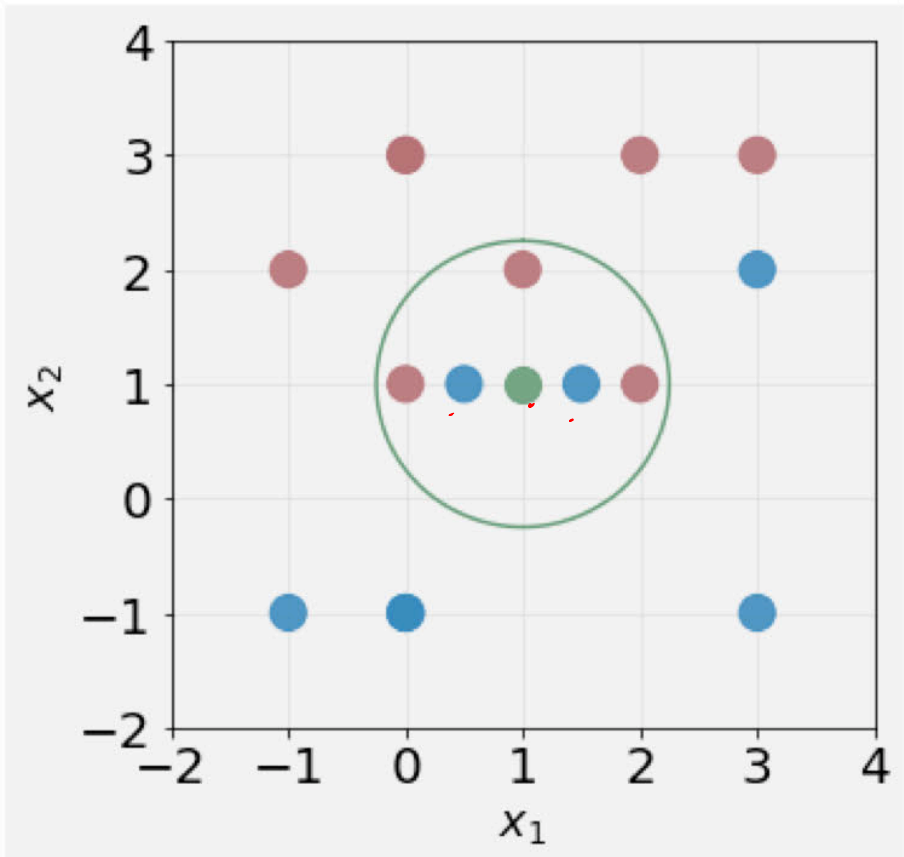
Weighted KNN

What should 5-NN predict in the following figure?



Weighted KNN

What should 5-NN predict in the following figure?



Weighted KNN

Weighted-KNN:

$$h(x) = \arg \max_{y \in \{+1,-1\}} \sum_{(x',y') \in \mathrm{NN}(\mathrm{x},\mathrm{S}_{\mathrm{train}},\mathrm{k})} \frac{1}{d(x,x')} I(y = y')$$

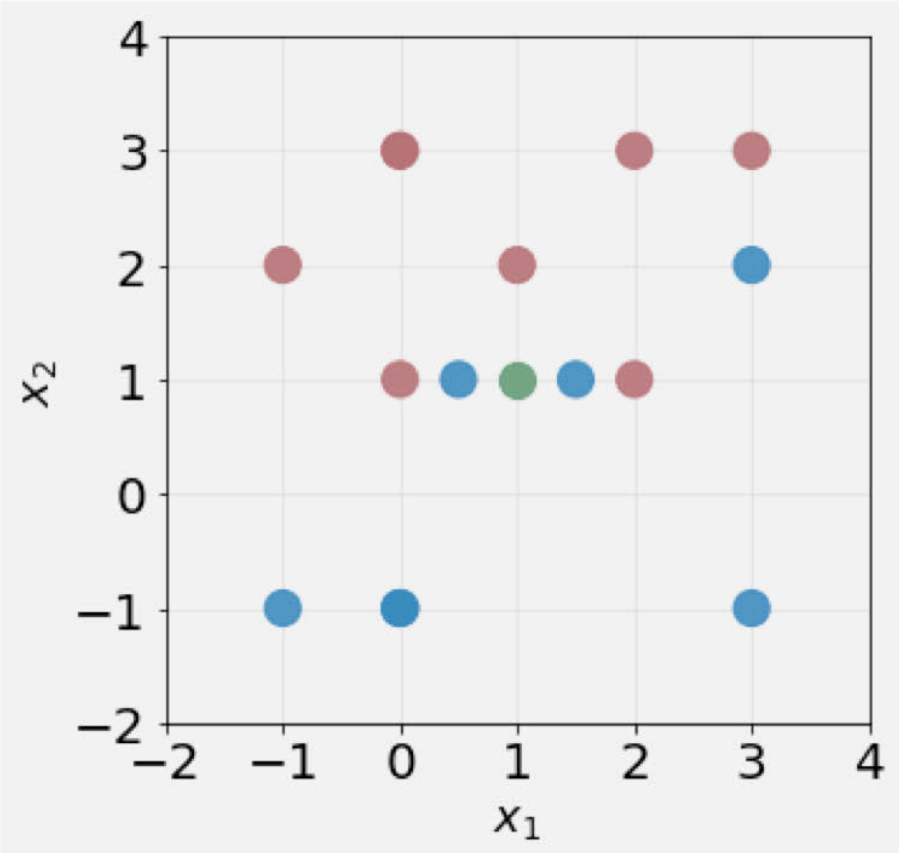
- Find $\mathrm{NN}(\mathrm{x}, \mathrm{S}_{\mathrm{train}}, \mathrm{k})$: the set of K training examples nearest to
- Predict $\hat{y}$ to be weighted-majority label in $\mathrm{NN}(\mathrm{x}, \mathrm{S}_{\mathrm{train}}, \mathrm{k})$, weighted by inverse-distance

Improvements over KNN:

- Gives more weight to examples that are very close to query point
- Less tie-breaking required

Weighted KNN

What should 5-NN predict in the following figure?



- Red distance: 1
- Blue distance: 0.5
- Red weighted-Majority vote: 3
- Blue weighted-majority vote: 4
- Prediction: Blue

K-Nearest Neighbors

- Overview
- Weighted KNN
- Performance Guarantee
- Curse of Dimensionality

How good is K-NN in theory? (Performance guarantee)

Performance Guarantee for 1-NN

What if we have close to infinite training data ($n \to \infty$), how well can 1-NN perform?

Performance Guarantee for 1-NN

What if we have close to infinite training data ($n \to \infty$), how well can 1-NN perform?

Bayes optimal classifier

Assuming that we know $P(y|x), y \in \{+1, -1\}$,

best prediction: $y^* = \arg\max_y P(y|x)$

Performance Guarantee for 1-NN

What if we have close to infinite training data ($n \to \infty$), how well can 1-NN perform?

Bayes optimal classifier

Assuming that we know $P(y|x), y \in \{+1, -1\}$,

best prediction: $y^* = \arg\max_y P(y|x)$

Example: $P(+1|x) = 0.9, P(-1|x) = 0.1$, what do you predict for x?

Performance Guarantee for 1-NN

What if we have close to infinite training data ($n \to \infty$), how well can 1-NN perform?

Bayes optimal classifier

Assuming that we know $P(y|x), y \in \{+1, -1\}$,

best prediction: $y^* = \arg\max_y P(y|x)$

Example: $P(+1|x) = 0.9, P(-1|x) = 0.1$, what do you predict for x?

$$\text{Err}_{\text{BayesOpt}} = 1 - P(y^*|x)$$

Performance Guarantee for 1-NN

Theorem

As $N = |S_{\text{train}}| \to \infty$, the 1-NN error is no more than twice the error of the Bayes Optimal Classifier. [Cover and Hart, 1967]

Proof.

Let x_{NN} be the nearest neighbor of our test point x. As $N \to \infty$, $\text{dist}(x_{\text{NN}}, x) \to 0$, thus $P(y^*|x_{\text{NN}}) \to P(y^*|x)$.

$$\begin{aligned}
\text{Err}_{\text{nn}} &= P(y_x \neq y_{x_{\text{NN}}}) \\
&= P(y^*|x)(1 - P(y^*|x_{\text{NN}})) + P(y^*|x_{\text{NN}})(1 - P(y^*|x)) \\
&\leq (1 - P(y^*|x_{\text{NN}})) + (1 - P(y^*|x)) \\
&= 2 * (1 - P(y^*|x)) = 2\,\text{Err}_{\text{BayesOpt}}
\end{aligned}$$

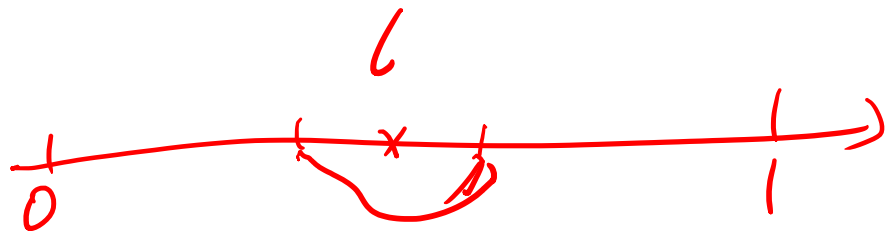
K-Nearest Neighbors

- Overview
- Weighted KNN
- Performance Guarantee
- Curse of Dimensionality

How does the algorithm scale? (Curse of Dimensionality)

Curse of Dimensionality

Given N points in $[0, 1]$, what is the size of the smallest interval to contain k-nearest neighbor for x?

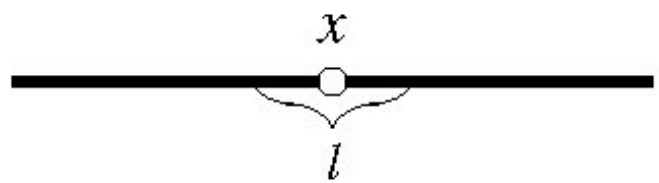


$$\frac{l}{1} \times N = k$$

$$l \sim \frac{k}{N}$$

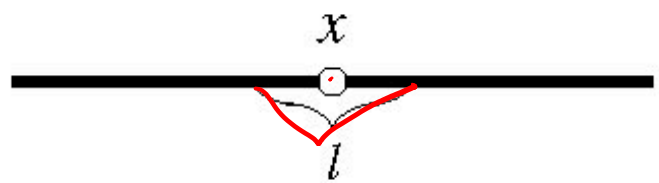
Curse of Dimensionality

Given N points in $[0, 1]$, what is the size of the smallest interval to contain k-nearest neighbor for x?



Curse of Dimensionality

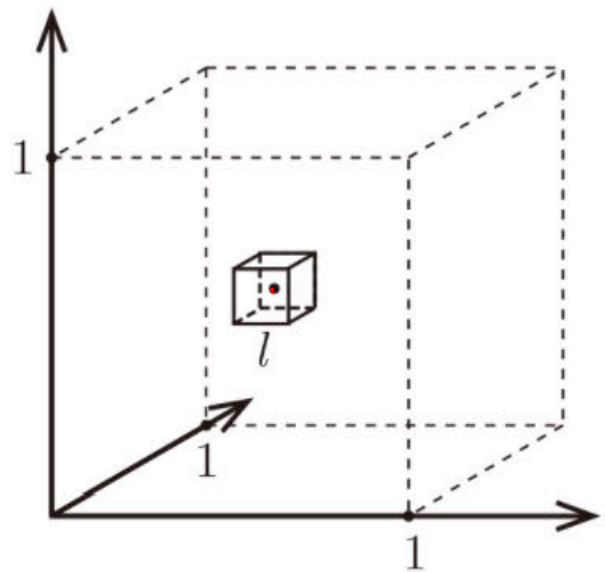
Given N points in $[0, 1]$, what is the size of the smallest interval to contain k-nearest neighbor for x?



$$N * l \approx k \Rightarrow l \approx \frac{k}{N}$$

Curse of Dimensionality

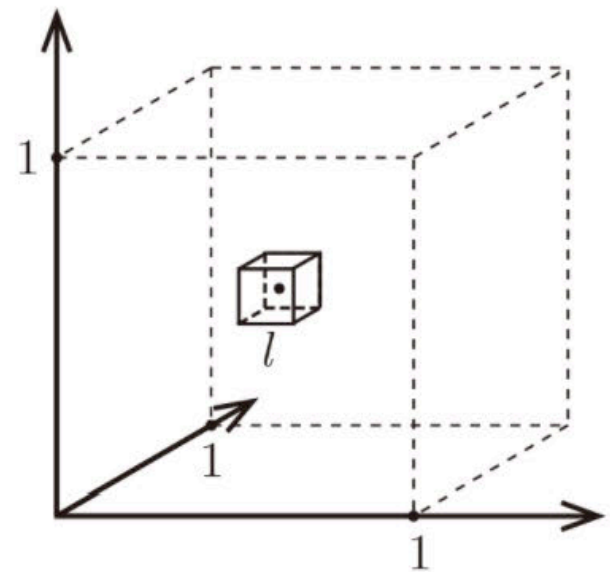
In general, for d dimensions, what is the length of the smallest hypercube to contain k-nearest neighbor for x?



$$\frac{\left(\frac{k}{N}\right)^{\frac{1}{d}}}{\frac{l^d}{1} = \frac{k}{N}}$$

Curse of Dimensionality

In general, for d dimensions, what is the length of the smallest hypercube to contain k-nearest neighbor for x?



$$N * l^d \approx k \Rightarrow l \approx \left(\frac{k}{N}\right)^{\frac{1}{d}}$$

Curse of Dimensionality

If $N = 1000, k = 10,$

d	l
2	0.1
10	0.63
100	0.955
1000	0.9954

We almost need the entire space to find 10 nearest neighbors.

n d.

How does the algorithm scale? (Memory and Efficiency of the naive implementation)

Memory: n×d

n×d

How does the algorithm scale? (Memory and Efficiency of the naive implementation)

Training: N/A

Testing

- memory: $O(Nd)$
- time: $O(Nd)$

Parametric models

- Neuron-Inspired classifier
- Linear classifiers

Happy medium

- Decision trees (that aren't too deep): use relatively few features to classify
- K-nearest neighbors: all features weighted equally
- Now: use all features but weight them

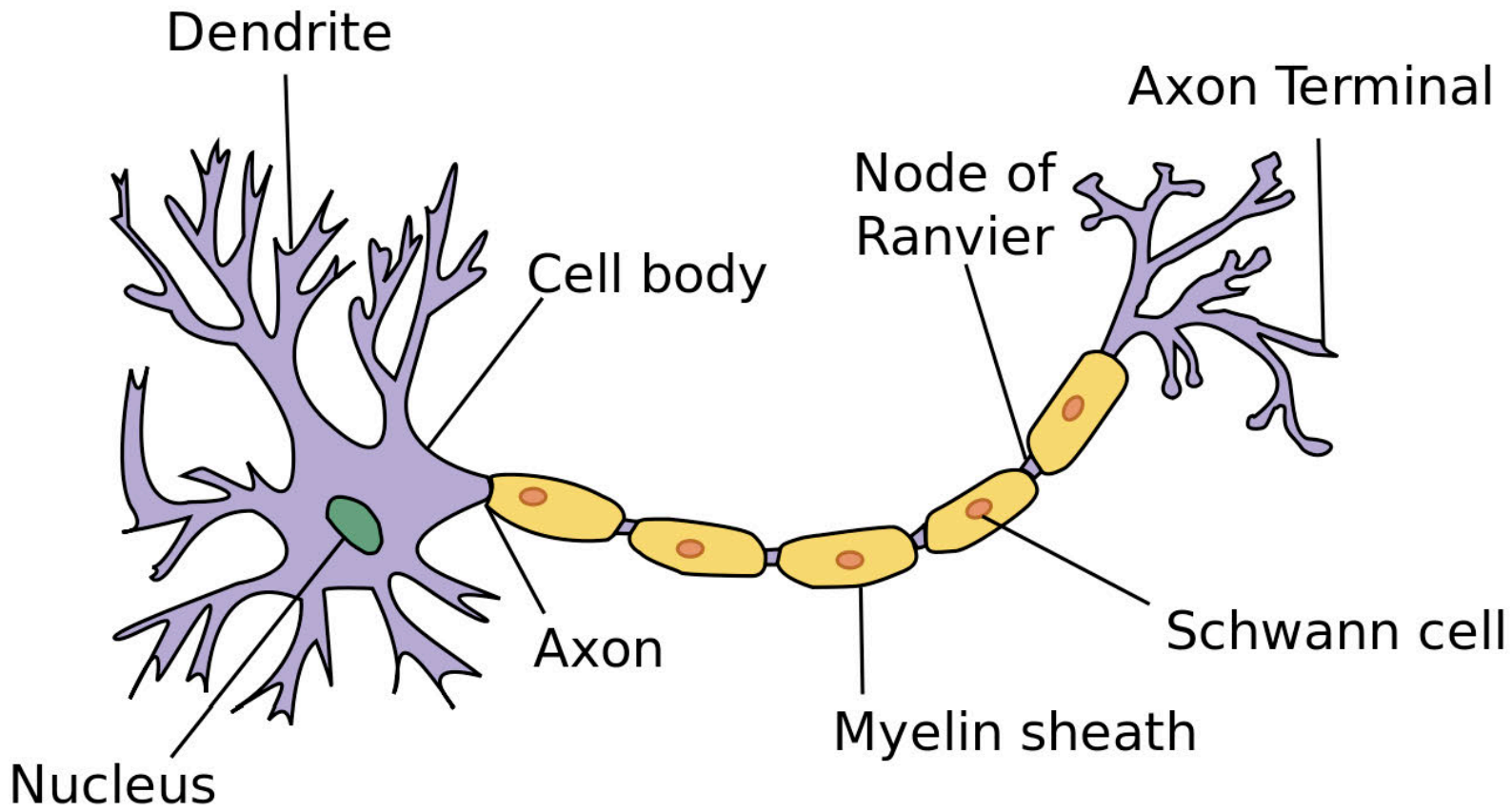
Happy medium

- Decision trees (that aren't too deep): use relatively few features to classify
- K-nearest neighbors: all features weighted equally
- Now: use all features but weight them

assume that $y \in \{-1, +1\}$ instead of $\{0, 1\}$, and that $\boldsymbol{x} \in \mathbb{R}^d$.

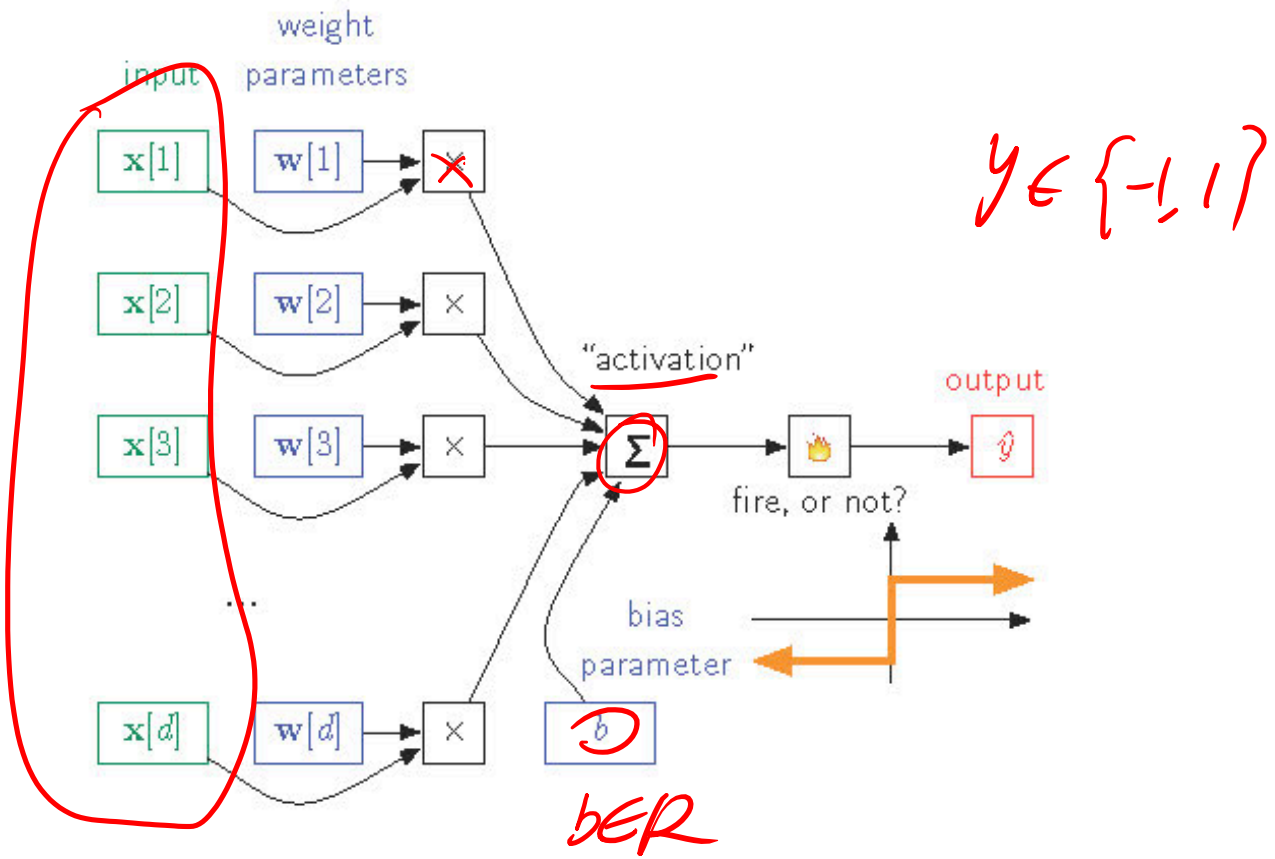
Inspiration from Neurons

Image from Wikimedia Commons.



Input signals come in through dendrites, output signal passes out through the axon.

Neuron-inspired classifier



Neuron-inspired classifier

$$f(\boldsymbol{x}) = \text{sign}\,(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

remembering that: $\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{j=1}^{d} \boldsymbol{w}[j] \cdot \boldsymbol{x}[j]$

Neuron-inspired classifier

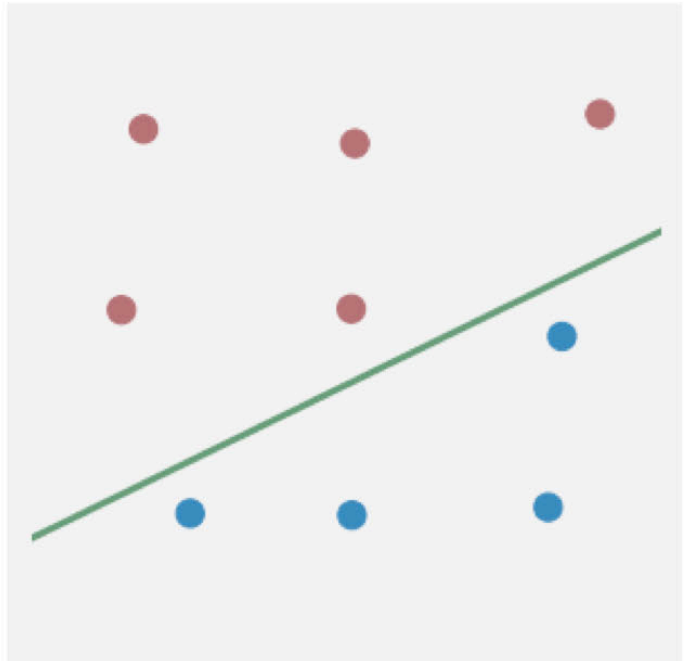
$$f(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right)$$

remembering that: $\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{j=1}^{d} \boldsymbol{w}[j] \cdot \boldsymbol{x}[j]$

Learning requires us to set the weights $\boldsymbol{w}$ and the bias b.

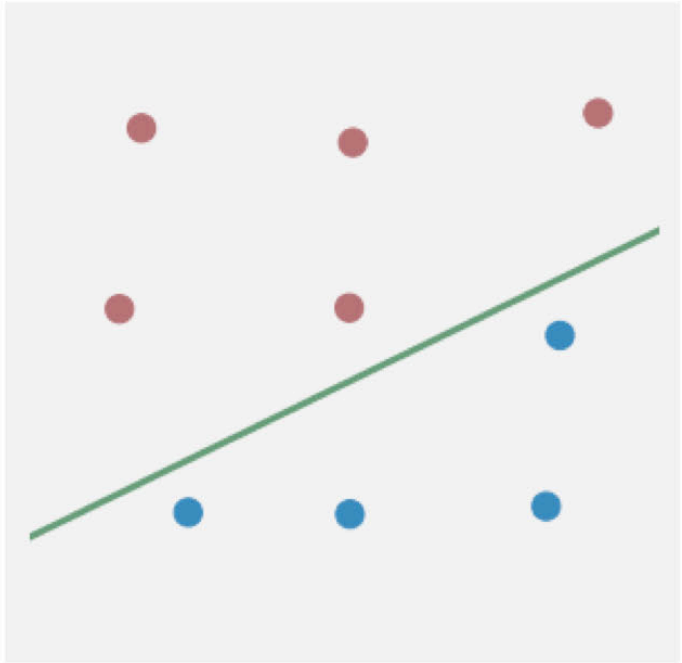
Linear classifiers

- Binary Classification: Only two classes
- A linear classifier draws a line through space separating the two classes.
- For two-features, a linear classifier takes form on the right



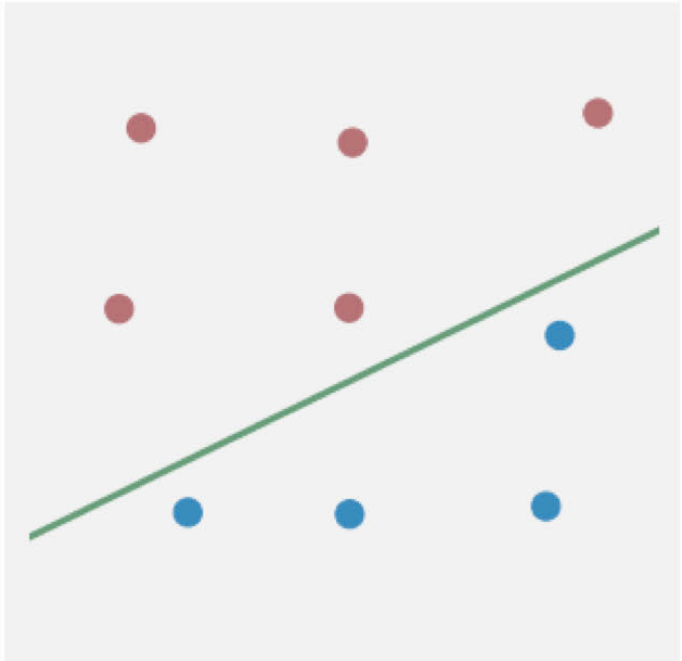
Perceptron classifiers are linear classifiers

$$y = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$



Perceptron classifiers are linear classifiers

$$y = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$
$$= \begin{cases} +1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b \geq 0 \\ -1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b < 0 \end{cases}$$



Wrap up

- K-nearest neighbor
- Perceptron/linear classifiers