Machine Learning: Chenhao Tan

University of Colorado Boulder

LECTURE 5

Slides adapted from Chris Ketelsen, Jordan Boyd-Graber

Logistics

- Interest and teammates match
- TA/GSS office hours

Learning objectives

- Understand bias-variance tradeoff
- Understand K-nearest neighbor classifiers

Outline

Bias-variance tradeoff

K-nearest neighbors

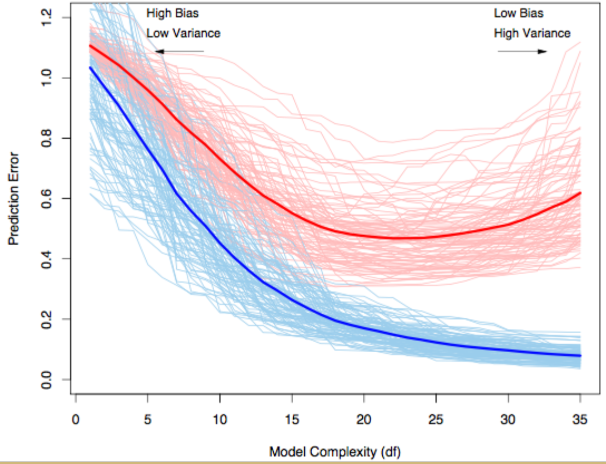
- Overview
- Weighted KNN
- Performance Guarantee
- Curse of Dimensionality

Outline

Bias-variance tradeoff

K-nearest neighbors
- Overview
- Weighted KNN
- Performance Guarantee
- Curse of Dimensionality

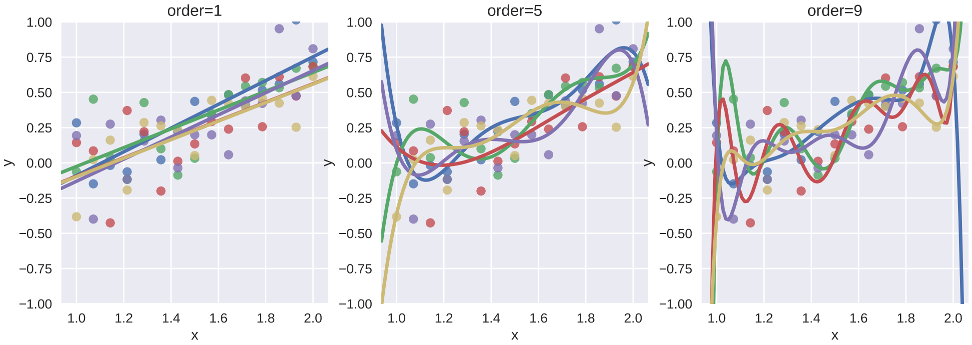
A more general view of overfitting



[Friedman et al., 2001]

Example

$$f(x) = 0.5x^2 - 0.8x + 0.3 + \epsilon, \epsilon \sim \mathcal{N}(0,1)$$



Bias-Variance Tradeoff

Assume a simple model $y = f(x) + \epsilon$, $\mathrm{E}(\epsilon) = 0, \mathrm{Var}(\epsilon) = \sigma_\epsilon^2$,

Bias-Variance Tradeoff

Assume a simple model $y = f(x) + \epsilon$, $\mathrm{E}(\epsilon) = 0, \mathrm{Var}(\epsilon) = \sigma_\epsilon^2$,

$$\mathrm{Err}(x_0) = \mathrm{E}[(y - h(x_0))^2]$$

Bias-Variance Tradeoff

Assume a simple model $y = f(x) + \epsilon$, $\mathrm{E}(\epsilon) = 0, \mathrm{Var}(\epsilon) = \sigma_\epsilon^2$,

$$\begin{aligned} \mathrm{Err}(x_0) &= \mathrm{E}[(y - h(x_0))^2] \\ &= \sigma_\epsilon^2 + [\mathrm{E}h(x_0) - f(x_0)]^2 + \mathrm{E}[h(x_0) - \mathrm{E}h(x_0)]^2 \\ &= \sigma_\epsilon^2 + \mathrm{Bias}^2(h(x_0)) + \mathrm{Var}(h(x_0)) \\ &= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance} \end{aligned}$$

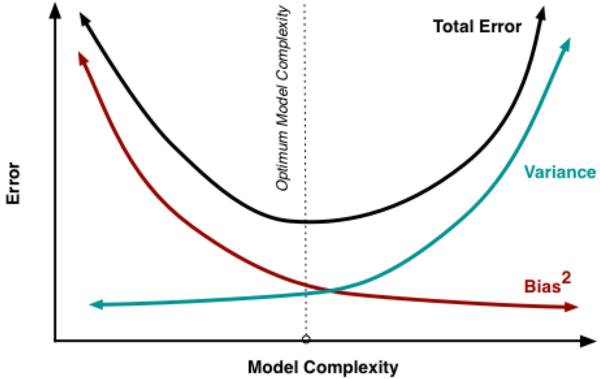
Bias-Variance Tradeoff

$$\text{Generailization error} \quad = \quad \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}$$

- $[\mathrm{E}h(x_0) - f(x_0)]^2$, high bias means that even with all training data, the error is still high. Model is not flexible enough to model the true function.
- $\mathrm{E}[h(x_0) - \mathrm{E}h(x_0)]^2$, high variance means that a small variation of training data leads to a great change in the learned model. Model is very sensitive to training data.

Revisit Overfitting

$$\text{Generailization error} \quad = \quad \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}$$



http://scott.fortmann-roe.com/docs/BiasVariance.html

Outline

Bias-variance tradeoff

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

Continuous

Euclidean distance

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

Discrete

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

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

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

Manhattan distance

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

Hyperparameters in the algorithm

- Distance function

Discrete

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

Continuous

Euclidean distance

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

Manhattan distance

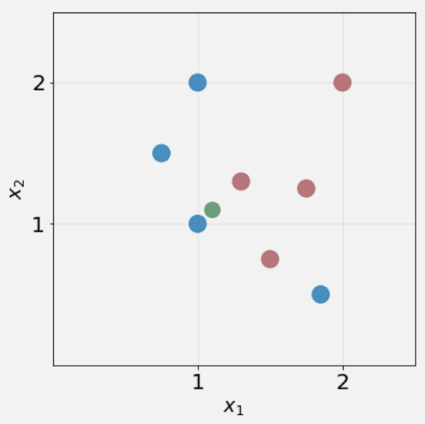
$$d(x_1, x_2) = \|\vec{x}_1 - \vec{x}_2\|_1 \quad (3)$$

- 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.

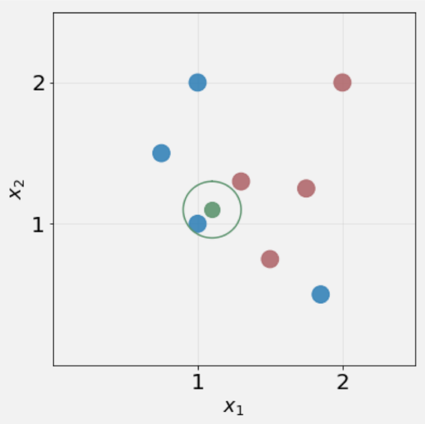
KNN Classification



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

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

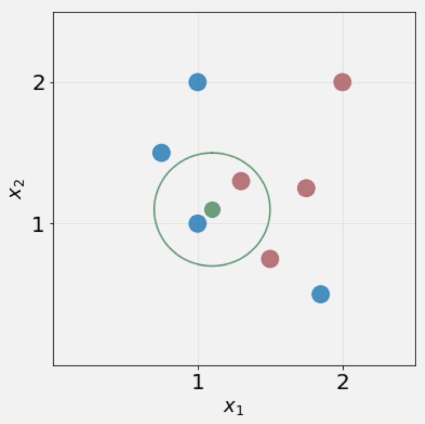
KNN Classification



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

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

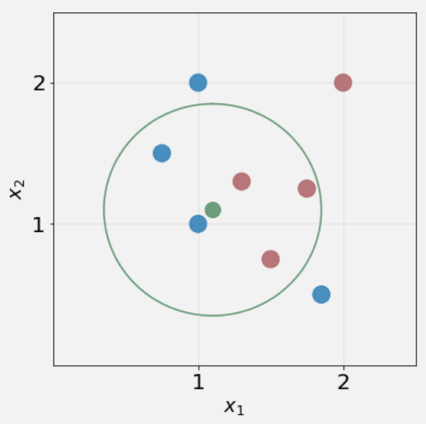
KNN Classification



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

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

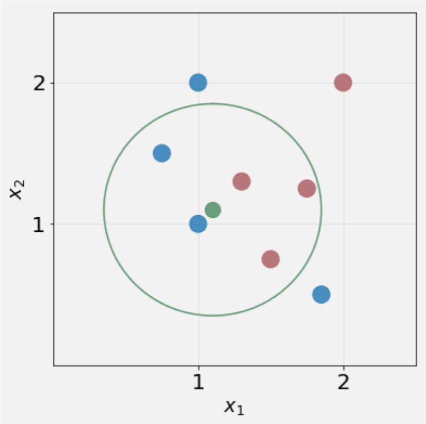
KNN Classification



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

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

KNN Classification

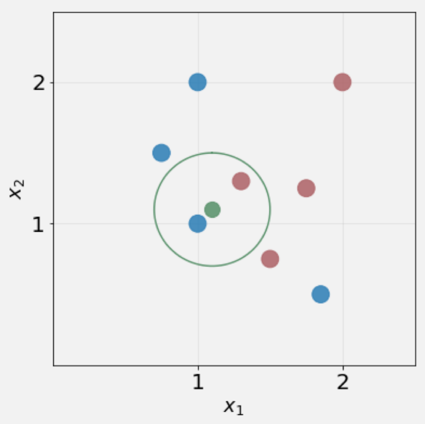


Euclidean distance: $||\vec{x}_1 - \vec{x}_2||_2$

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

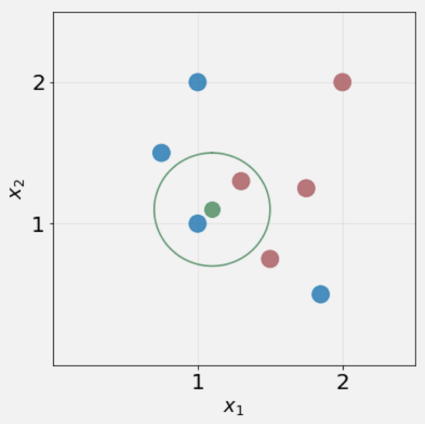
KNN Classification

What about ties?



KNN Classification

What about ties?

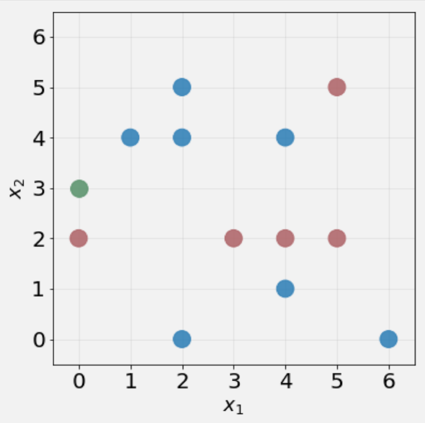


One reasonable strategy:

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

KNN Classification

Another example

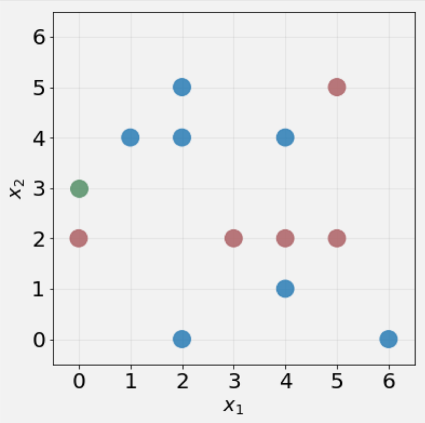


Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

$k = 1$

KNN Classification

Another example

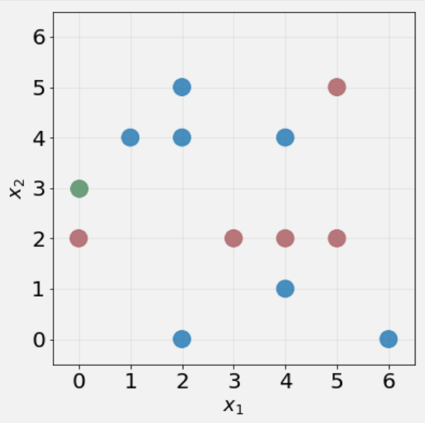


Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

$k = 2$

KNN Classification

Another example

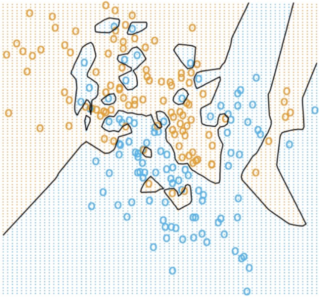


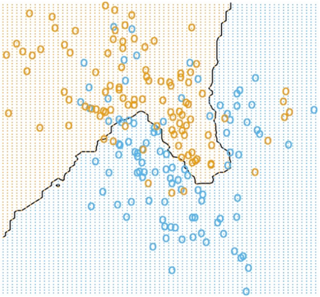
Euclidean distance: $\|\vec{x}_1 - \vec{x}_2\|_2$

$k = 3$

K-nearest neighbors

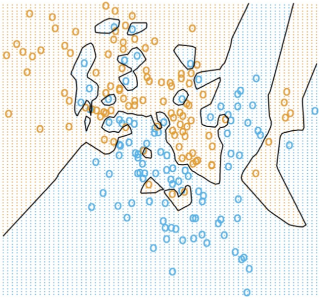
Recall the danger of overfitting



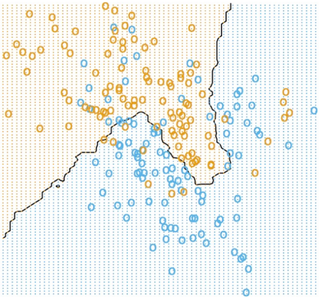


K-nearest neighbors

Recall the danger of overfitting



$k = 1$

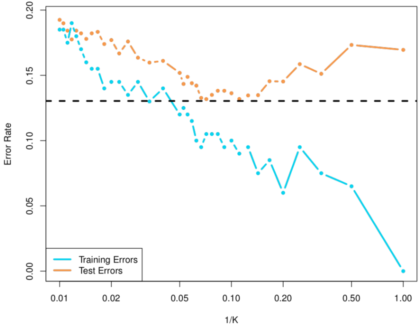


$k = 15$

K-nearest neighbors

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

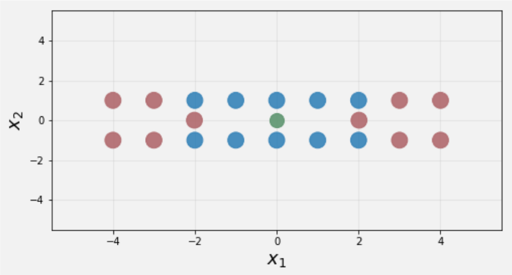
- Plot vs. $\frac{1}{k}$
- Lower k leads to more flexibility



Feature normalization

Practical tips:

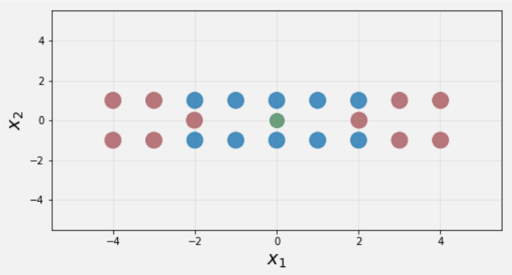
- Important to normalize features to similar sizes
- Consider doing 2-NN on unscaled and scaled data

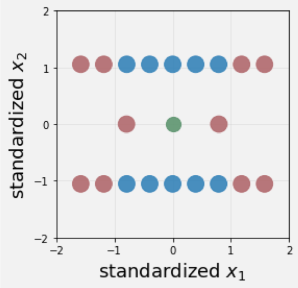


Feature normalization

Practical tips:

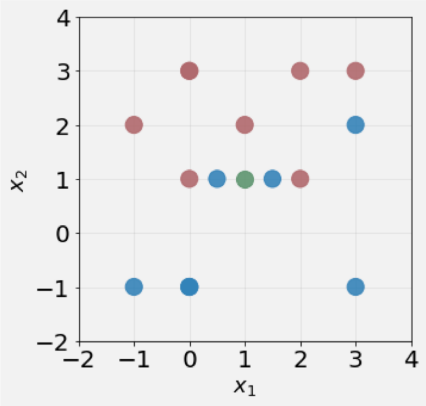
- Important to normalize features to similar sizes
- Consider doing 2-NN on unscaled and scaled data





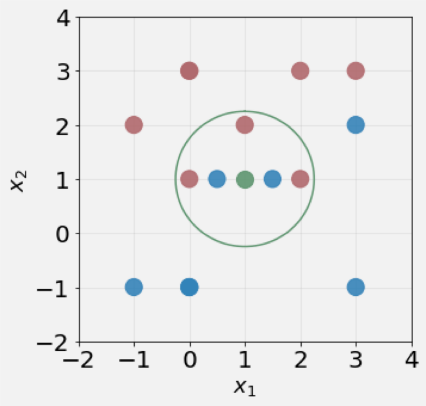
Weighted KNN

What should 5-NN predict in the following figure?



Weighted KNN

What should 5-NN predict in the following figure?



Weighted KNN

Weighted-KNN:

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

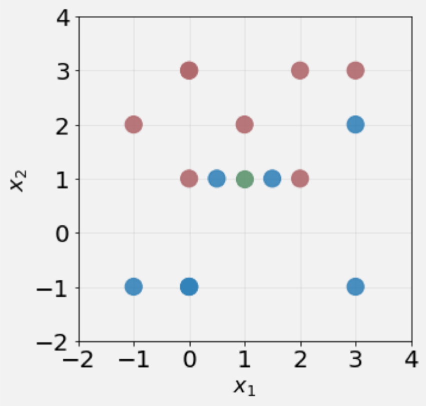
- Find $\mathrm{NN}(\mathrm{x}, \mathrm{S}_{\mathrm{train}}, \mathrm{k})$: the set of K training examples nearest to
- Predict $\hat{y}$ to be weighted-majority label in $\mathrm{NN}(\mathrm{x}, \mathrm{S}_{\mathrm{train}}, \mathrm{k})$, weighted by inverse-distance

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:
- Blue distance:
- Red weighted-Majority vote:
- Blue weighted-majority vote:
- Prediction:

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 ($n \rightarrow \infty$), 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?

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

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 $P(y^*|x_{\text{NN}}) \to P(y^*|x)$.

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 $P(y^*|x_{\text{NN}}) \to P(y^*|x)$.

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

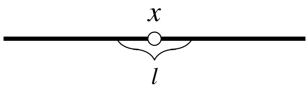
How does the algorithm scale? (Curse of Dimensionality)

Curse of Dimensionality

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

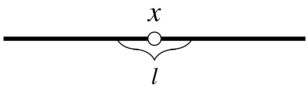
Curse of Dimensionality

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



Curse of Dimensionality

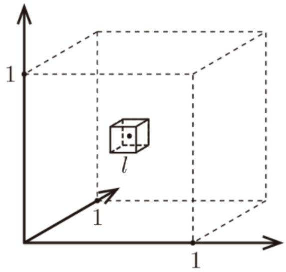
Given N points in $[0, 1]$, what is the size of the smallest interval to contain k-nearest neighbor for x?



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

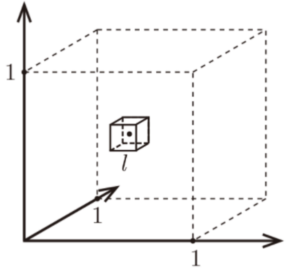
Curse of Dimensionality

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



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.

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

How does the algorithm scale? (Memory and Efficiency of the naive implementation)
Training: N/A
Testing

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

Summary

- Show bias-variance tradeoff using a simple model
- Learned about KNN and weighted KNN

References

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.