Lecture 22: k-Nearest Neighbors

Preview

- k-Nearest Neighbors is a learning algorithm that can be used for **regression or classification**
- It is powerful, interpretable, and has O(1) training time
- The tradeoffs are inference time and scaling to higher dimensions

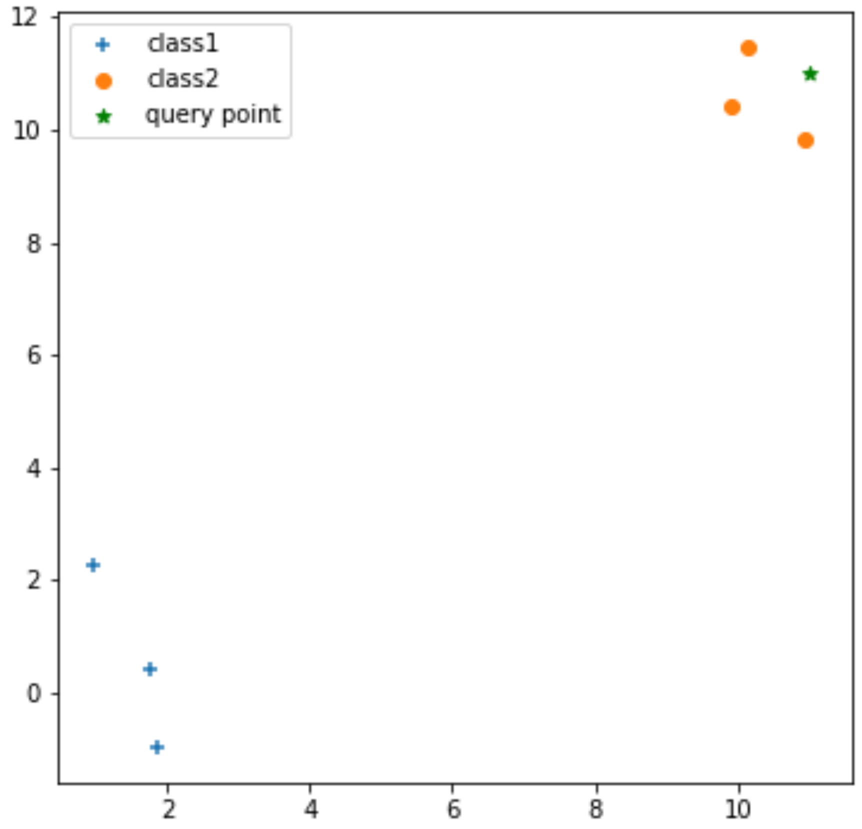
Key idea

- Similar data points should have similar outputs

Outline

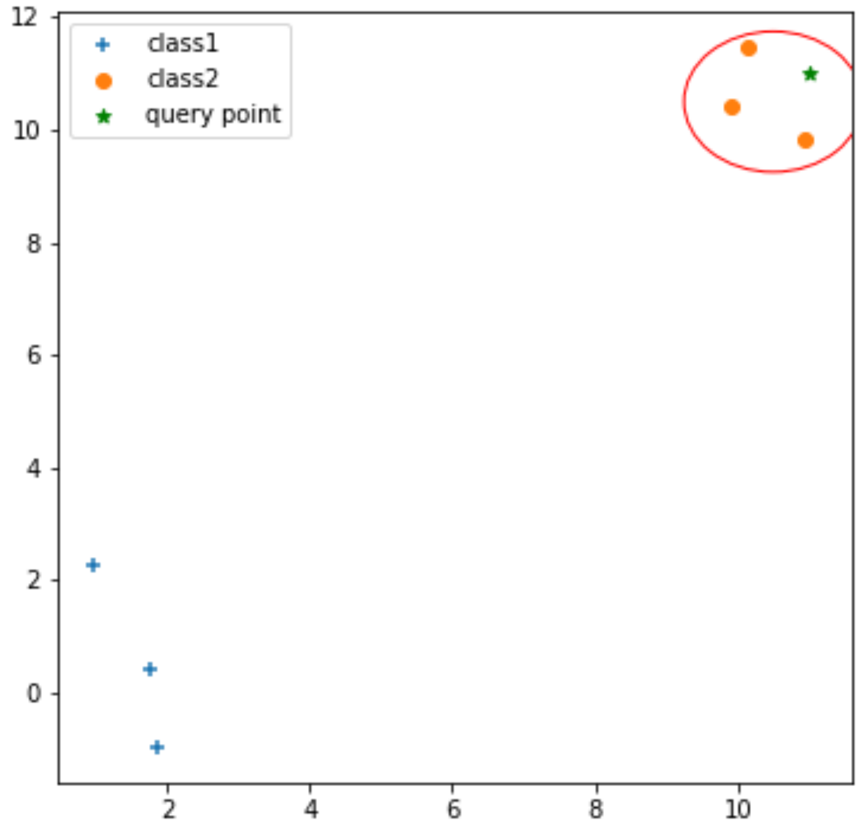
1. **An example of k-NN**
2. Pseudocode for k-NN
3. k-NN and regression
4. How to choose k
5. Computational complexity
6. Behavior in higher dimensions
7. Improving k-NN

k-Nearest Neighbors: an example



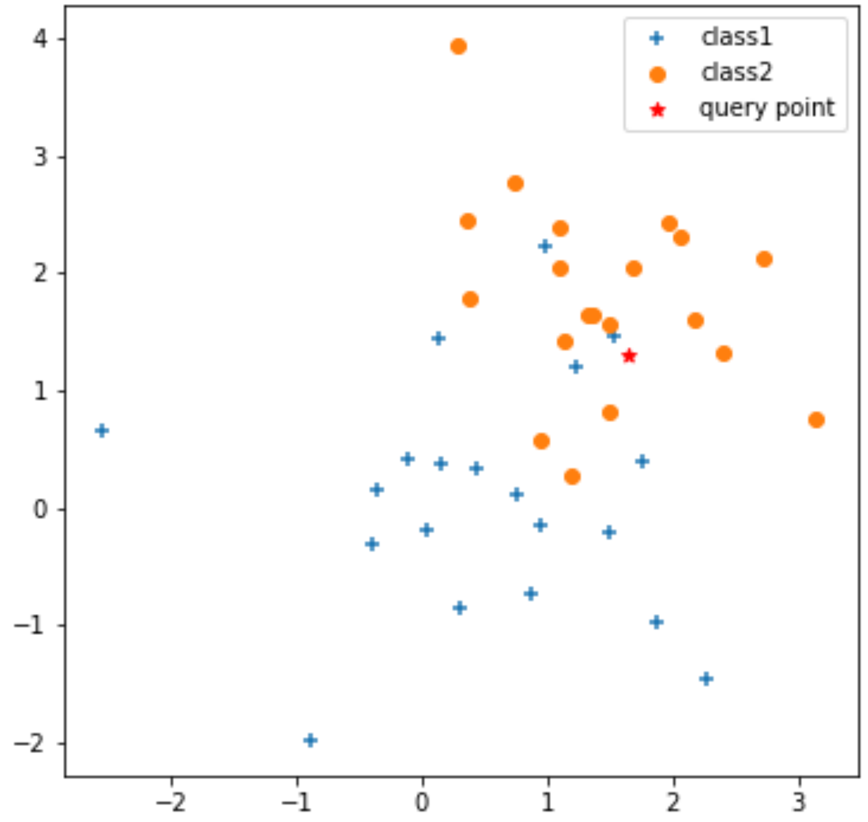
What class does query example have?

k-Nearest Neighbors: an example



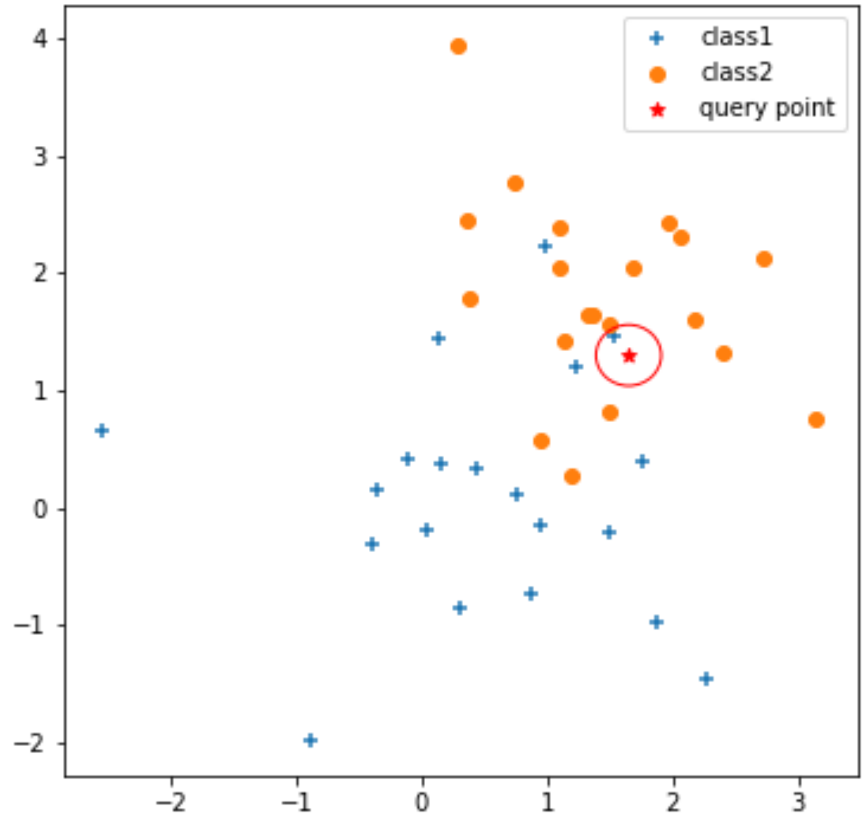
Closest to class-2 training examples, therefore class 2

k-Nearest Neighbors: another example



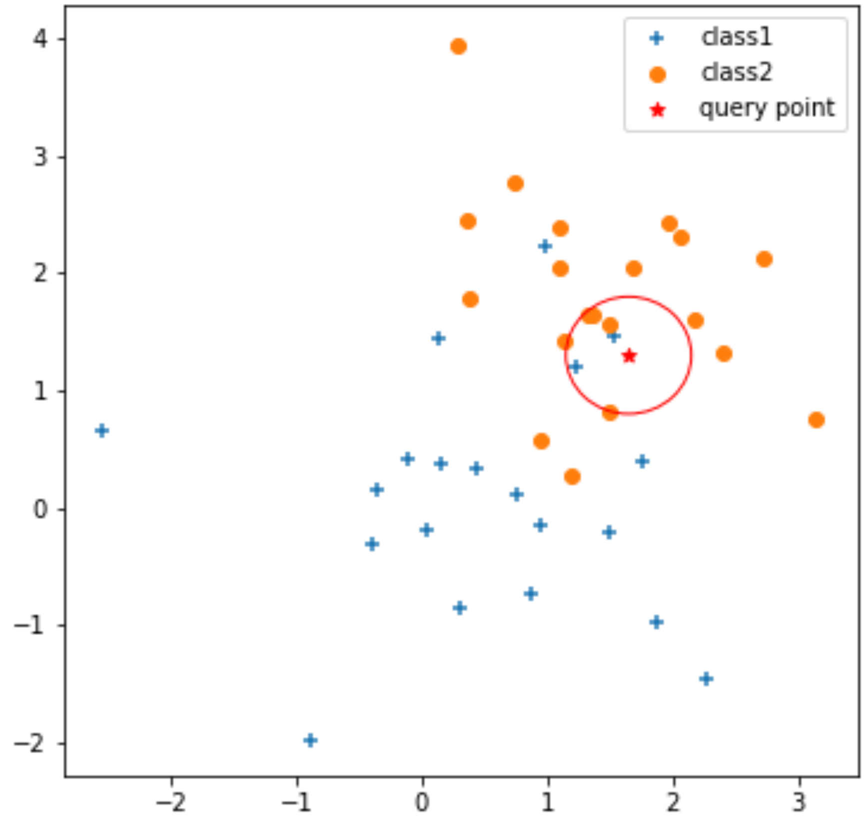
What class does query example have?

k-Nearest Neighbors: another example



Closest neighbor is class-1, so is it class-1?

k-Nearest Neighbors: another example



But most of its neighbors are class-2, so maybe it's class-2?

What does k do?

- k = number of neighbors to consider
- If the task is classification, the k neighbors **vote** on which class the query point should correspond to

Outline

1. An example of k-NN
2. **Pseudocode for k-NN**
3. k-NN and regression
4. How to choose k
5. Computational complexity
6. Behavior in higher dimensions
7. Improving k-NN

k-NN: The Algorithm

```
def kNN(query_point, data, k=1):
    features, labels = data
    N = features.shape[0]
    # Calculate distance between query_point and all X
    distances = distance_fn(features, query_point)
    # Find indices of k closest points
    closest_point_indices = argsort(distances)[:k]
    # Find the labels of the neighbors
    neighbor_labels = labels[closest_point_indices]
    predicted_label = mode(neighbor_labels)

    return predicted_label
```

k-NN: The Algorithm

```
def kNN(query_point, data, k=1):
    features, labels = data
    N = features.shape[0]
    # Calculate distance between query_point and all X
    distances = distance_fn(features, query_point)
    # Find indices of k closest points
    closest_point_indices = argsort(distances)[:k]
    # Find the labels of the neighbors
    neighbor_labels = labels[closest_point_indices]
    predicted_label = mode(neighbor_labels)

    return predicted_label
```

k-NN: The Algorithm

```
def kNN(query_point, data, k=1):
    features, labels = data
    N = features.shape[0]
    # Calculate distance between query_point and all X
    distances = distance_fn(features, query_point)
    # Find indices of k closest points
    closest_point_indices = argsort(distances)[:k]
    # Find the labels of the neighbors
    neighbor_labels = labels[closest_point_indices]
    predicted_label = mode(neighbor_labels)

    return predicted_label
```

k-NN: The Algorithm

```
def kNN(query_point, data, k=1):
    features, labels = data
    N = features.shape[0]
    # Calculate distance between query_point and all X
    distances = distance_fn(features, query_point)
    # Find indices of k closest points
    closest_point_indices = argsort(distances)[:k]
    # Find the labels of the neighbors
    neighbor_labels = labels[closest_point_indices]
    predicted_label = mode(neighbor_labels)

    return predicted_label
```

k-NN: The Algorithm

```
def kNN(query_point, data, k=1):
    features, labels = data
    N = features.shape[0]
    # Calculate distance between query_point and all X
    distances = distance_fn(features, query_point)
    # Find indices of k closest points
    closest_point_indices = argsort(distances)[:k]
    # Find the labels of the neighbors
    neighbor_labels = labels[closest_point_indices]
    predicted_label = mode(neighbor_labels)

    return predicted_label
```

k-NN: The Algorithm

```
def kNN(query_point, data, k=1):
    features, labels = data
    N = features.shape[0]
    # Calculate distance between query_point and all X
    distances = distance_fn(features, query_point)
    # Find indices of k closest points
    closest_point_indices = argsort(distances)[:k]
    # Find the labels of the neighbors
    neighbor_labels = labels[closest_point_indices]
    predicted_label = mode(neighbor_labels)

    return predicted_label
```

k-NN: The Algorithm

```
def kNN(query_point, data, k=1):
    features, labels = data
    N = features.shape[0]
    # Calculate distance between query_point and all X
    distances = distance_fn(features, query_point)
    # Find indices of k closest points
    closest_point_indices = argsort(distances)[:k]
    # Find the labels of the neighbors
    neighbor_labels = labels[closest_point_indices]
    predicted_label = mode(neighbor_labels)

    return predicted_label
```

k-NN: The Algorithm

```
def kNN(query_point, data, k=1):
    features, labels = data
    N = features.shape[0]
    # Calculate distance between query_point and all X
    distances = distance_fn(features, query_point)
    # Find indices of k closest points
    closest_point_indices = argsort(distances)[:k]
    # Find the labels of the neighbors
    neighbor_labels = labels[closest_point_indices]
    predicted_label = mode(neighbor_labels)

    return predicted_label
```

Outline

1. An example of k-NN
2. Pseudocode for k-NN
3. **k-NN and regression**
4. How to choose k
5. Computational complexity
6. Behavior in higher dimensions
7. Improving k-NN

How to use k-NN for regression

- Find k closest points
- Instead of voting on class, simply take the average
- Can also try other schemes, like a weighted average

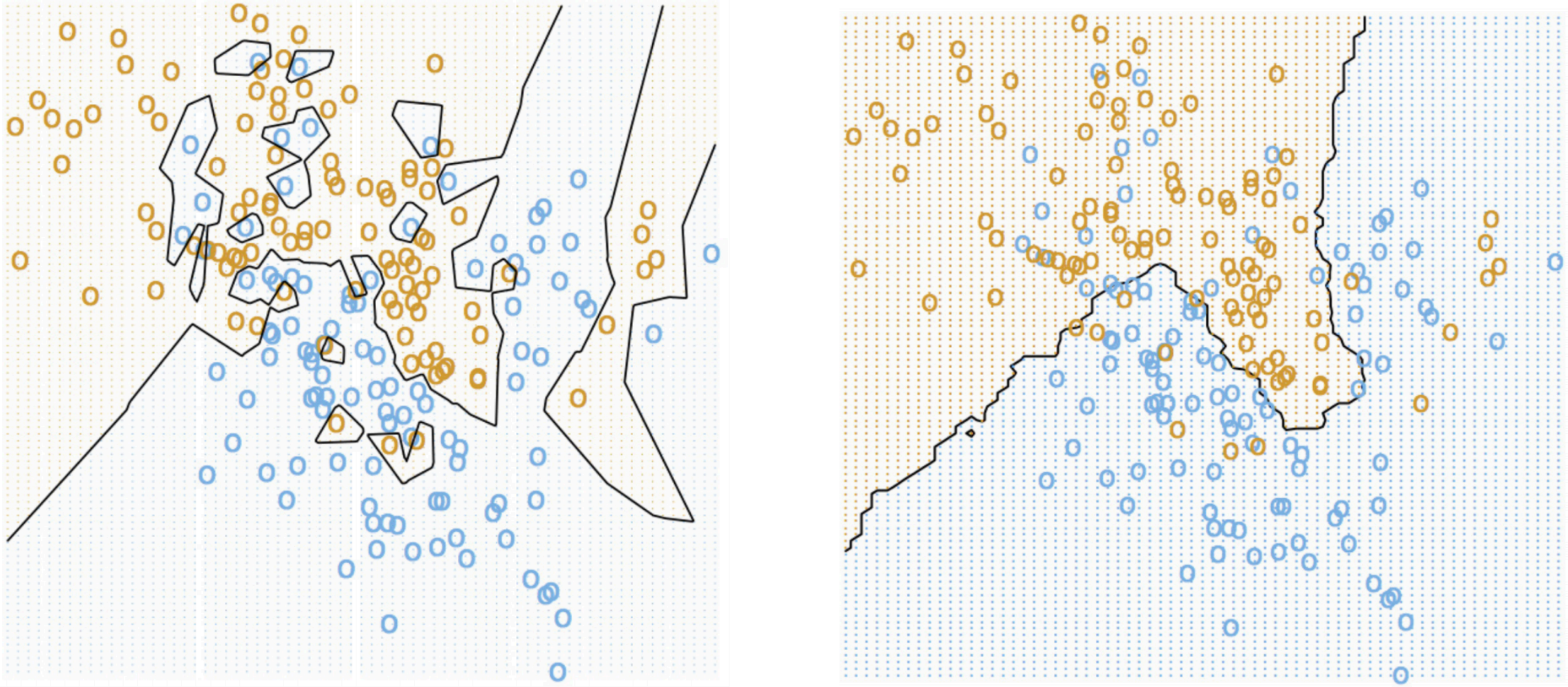
Outline

1. An example of k-NN
2. Pseudocode for k-NN
3. k-NN and regression
4. **How to choose k**
5. Computational complexity
6. Behavior in higher dimensions

How to choose k?

- k is a hyper parameter, choose it through cross-validation!

What effect does k have on the decision function?



(a) $k = 1$ (b) $k = 15$

Figure 2: **Voronoi diagram** for $k = 1$ vs. $k = 15$. Figure from Introduction to Statistical Learning.

What effect does k have on the decision function?

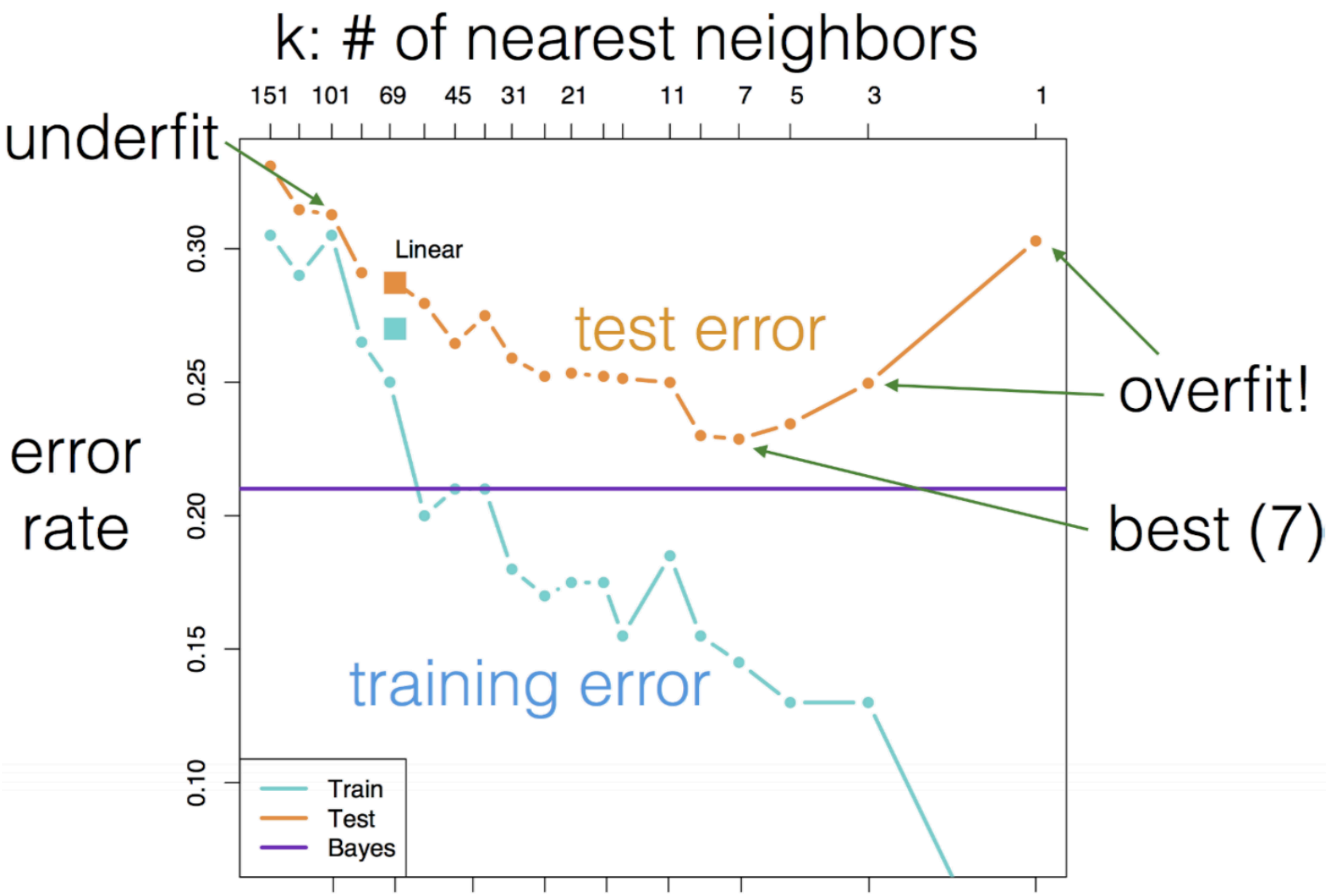


Figure 3: Training and Testing error as a function of k. Figure from Introduction to Statistical Learning.

What effect does k have on the decision function?

Takeaways

- k-NN can model some highly complex decision functions
- Performance of the classifier is super sensitive to k
- Increasing k increases bias and decreases variance
- As always, a good way to choose k is through cross-validation

Outline

1. An example of k-NN
2. Pseudocode for k-NN
3. k-NN and regression
4. How to choose k
5. **Computational complexity**
6. Behavior in higher dimensions
7. Improving k-NN

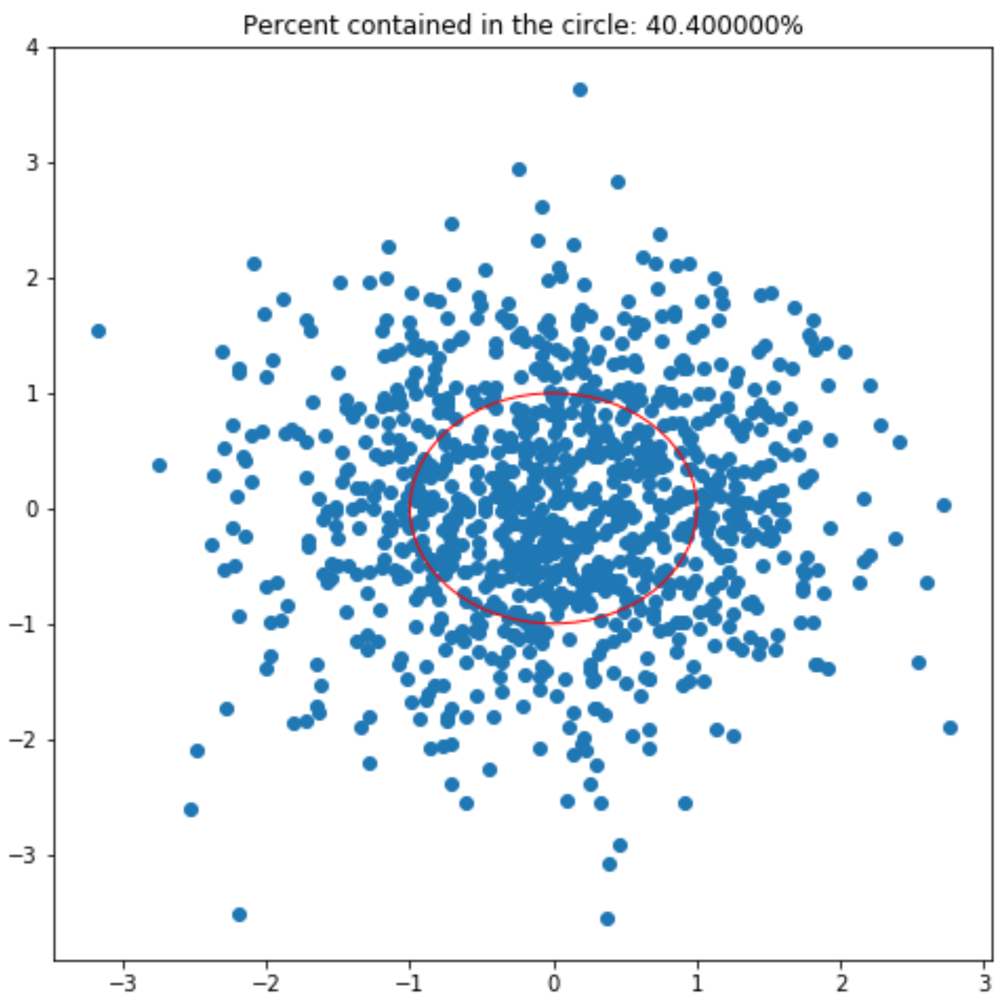
Computational complexity of k-NN

- Memory requirement
  - O(dN) [d = dimensionality of data, N = number of data points]
- Training time
  - O(1)
- Inference time
  - O(dN)
- Approximate nearest neighbor algorithms exist that try to reduce this inference time (e.g., Locality Sensitive Hashing)

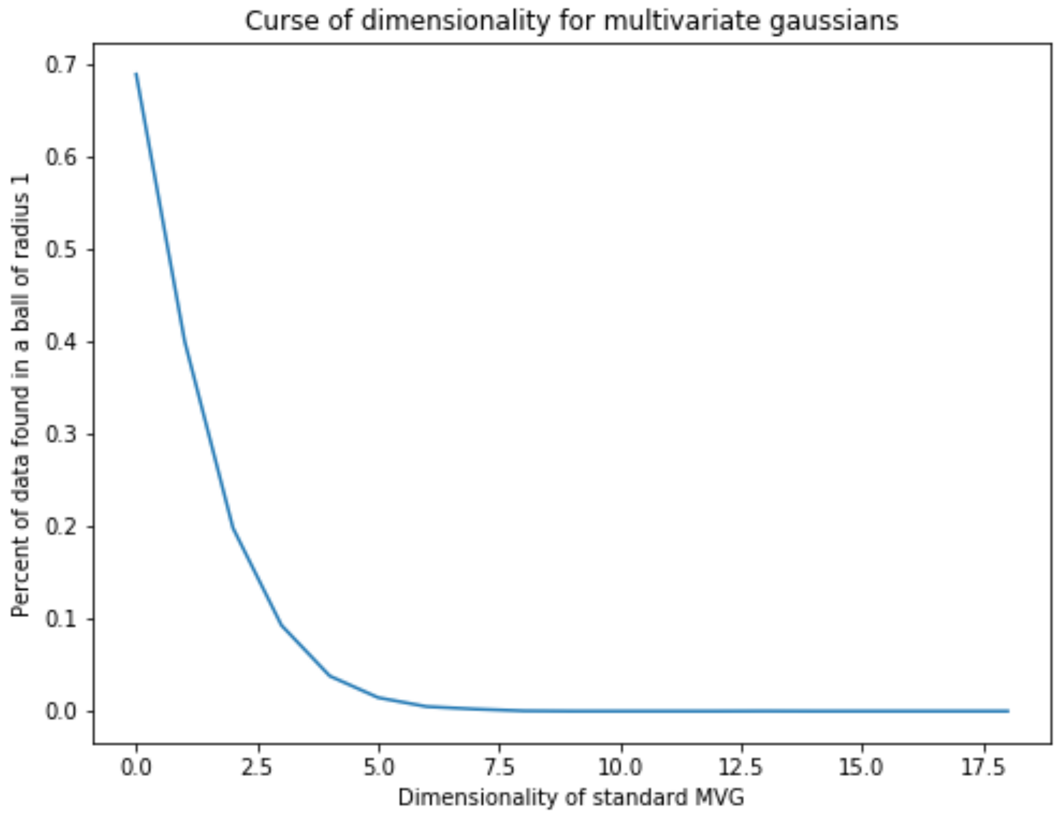
Outline

1. An example of k-NN
2. Pseudocode for k-NN
3. k-NN and regression
4. How to choose k
5. Computational complexity
6. **Behavior in higher dimensions**
7. Improving k-NN

Our intuition about Gaussians: Most of data < 1std away



Behavior in high dimension



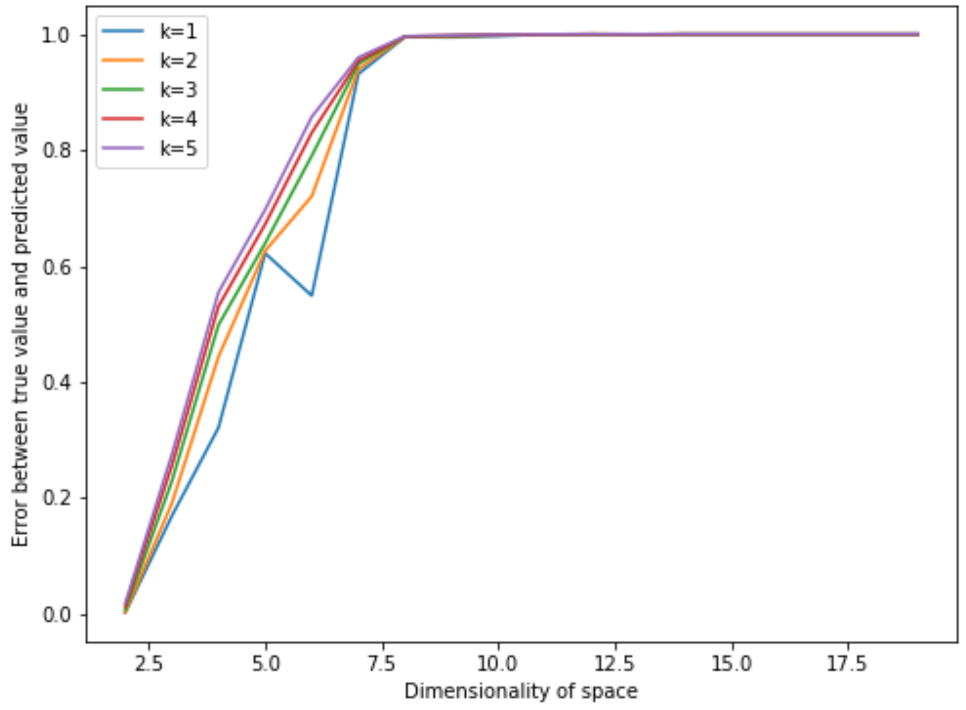
- Let's look at what percentage of a standard MVG data falls within ball of radius 1
- As the dimensionality of the MVG increases, the percentage of data distance <= 1 goes to zero!

How does two-class k-NN scale to n dimensions?

- Sample 1,000 datapoints x_i from $[-1, 1]^n$
- Ground truth relationship between x_i and y_i: $y_i = \exp(-8||x_i||^2)$
- What is the error of the k-NN regressor for point $x_i = \vec{0}$ as we increase n?

How does two-class k-NN scale to n dimensions?

- Sample 1,000 datapoints x_i from $[-1, 1]^n$
- Ground truth relationship between x_i and y_i: $y_i = \exp(-8||x_i||^2)$
- What is the error of the k-NN regressor for point $x_i = \vec{0}$ as we increase n?



Outline

1. An example of k-NN
2. Pseudocode for k-NN
3. k-NN and regression
4. How to choose k
5. Computational complexity
6. Behavior in higher dimensions
7. **Improving k-NN**

How to improve k-NN?

- Obtain more training data
- Reduce the dimensionality of data
- Consider other distance functions
  - E.g., Minkowski distances $D_p(x, z) = \left(\sum_{i=1}^{d} |x_i - z_i|^p \right)^{1/p}$
  - E.g., Using Kernels