CEE 697M: Probabilistic Machine Learning

M4 Nonparametric Methods: L4a: Exemplar-based methods

Jimi Oke

UMass Amherst
College of Engineering

Wed, Apr 19, 2023

Outline

Nonparametric modeling

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

- $\boldsymbol{\theta}$ is a fixed-dimensinoal vector of **parameters**

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

- $\boldsymbol{\theta}$ is a fixed-dimensinoal vector of **parameters**
- Estimation is performed using a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

- $\boldsymbol{\theta}$ is a fixed-dimensinoal vector of **parameters**
- Estimation is performed using a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$
- There is an assumed functional form: $\boldsymbol{y} \sim f_{\boldsymbol{\theta}}(\boldsymbol{x})$

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

- $\boldsymbol{\theta}$ is a fixed-dimensinoal vector of **parameters**
- Estimation is performed using a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$
- There is an assumed functional form: $\boldsymbol{y} \sim f_{\boldsymbol{\theta}}(\boldsymbol{x})$

Nonparametric models are defined based on similarity between a test input $\boldsymbol{x}$ at each training input $\boldsymbol{x}_n$: $d(\boldsymbol{x}, \boldsymbol{x}_n)$

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

- $\boldsymbol{\theta}$ is a fixed-dimensinoal vector of **parameters**
- Estimation is performed using a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$
- There is an assumed functional form: $\boldsymbol{y} \sim f_{\boldsymbol{\theta}}(\boldsymbol{x})$

Nonparametric models are defined based on similarity between a test input $\boldsymbol{x}$ at each training input $\boldsymbol{x}_n$: $d(\boldsymbol{x}, \boldsymbol{x}_n)$

- No assumption of functional form on model parameters

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

- $\boldsymbol{\theta}$ is a fixed-dimensinoal vector of **parameters**
- Estimation is performed using a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$
- There is an assumed functional form: $\boldsymbol{y} \sim f_{\boldsymbol{\theta}}(\boldsymbol{x})$

Nonparametric models are defined based on similarity between a test input $\boldsymbol{x}$ at each training input $\boldsymbol{x}_n$: $d(\boldsymbol{x}, \boldsymbol{x}_n)$

- No assumption of functional form on model parameters
- Effective number of parameters can grow with size of dataset $|\mathcal{D}|$

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

- $\boldsymbol{\theta}$ is a fixed-dimensinoal vector of **parameters**
- Estimation is performed using a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$
- There is an assumed functional form: $\boldsymbol{y} \sim f_{\boldsymbol{\theta}}(\boldsymbol{x})$

Nonparametric models are defined based on similarity between a test input $\boldsymbol{x}$ at each training input $\boldsymbol{x}_n$: $d(\boldsymbol{x}, \boldsymbol{x}_n)$

- No assumption of functional form on model parameters
- Effective number of parameters can grow with size of dataset $|\mathcal{D}|$
- Known as **exemplar-based models** (as training samples are used to make each future prediction)

Nonparametric modeling

Parametric models seek to estimate $p(\boldsymbol{y}|\boldsymbol{\theta})$ (unconditional case) or $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ (conditional case).

- $\boldsymbol{\theta}$ is a fixed-dimensinoal vector of **parameters**
- Estimation is performed using a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) : n = 1 : N\}$
- There is an assumed functional form: $\boldsymbol{y} \sim f_{\boldsymbol{\theta}}(\boldsymbol{x})$

Nonparametric models are defined based on similarity between a test input $\boldsymbol{x}$ at each training input $\boldsymbol{x}_n$: $d(\boldsymbol{x}, \boldsymbol{x}_n)$

- No assumption of functional form on model parameters
- Effective number of parameters can grow with size of dataset $|\mathcal{D}|$
- Known as **exemplar-based models** (as training samples are used to make each future prediction)
- Other names: instance-based learning, memory-based learning

Exemplar-based models

Exemplar-based models

We will consider the following exemplar approaches:

Exemplar-based models

We will consider the following exemplar approaches:

- K-nearest neighbors (KNN)
- Kernel density estimation
- Kernel [local] regression

K nearest neighbor classifier

K nearest neighbor classifier

Basic idea: classify new/test input $\boldsymbol{x}$ by assigning to most probable (majority) label in the neighborhood of $\boldsymbol{x}$ (closest examples) from the training set.

K nearest neighbor classifier

Basic idea: classify new/test input $\boldsymbol{x}$ by assigning to most probable (majority) label in the neighborhood of $\boldsymbol{x}$ (closest examples) from the training set.
Thus, we estimate:

K nearest neighbor classifier

Basic idea: classify new/test input $\boldsymbol{x}$ by assigning to most probable (majority) label in the neighborhood of $\boldsymbol{x}$ (closest examples) from the training set.

Thus, we estimate:

$$p(y = c|\boldsymbol{x}, \mathcal{D}) = \frac{1}{K} \sum_{n \in N_K(\boldsymbol{x}, \mathcal{D})} \mathbb{I}(y_n = c) \tag{1}$$

K nearest neighbor classifier

Basic idea: classify new/test input $\boldsymbol{x}$ by assigning to most probable (majority) label in the neighborhood of $\boldsymbol{x}$ (closest examples) from the training set.
Thus, we estimate:

$$p(y = c|\boldsymbol{x}, \mathcal{D}) = \frac{1}{K} \sum_{n \in N_K(\boldsymbol{x}, \mathcal{D})} \mathbb{I}(y_n = c) \tag{1}$$

- c class label

K nearest neighbor classifier

Basic idea: classify new/test input $\boldsymbol{x}$ by assigning to most probable (majority) label in the neighborhood of $\boldsymbol{x}$ (closest examples) from the training set.
Thus, we estimate:

$$p(y = c|\boldsymbol{x}, \mathcal{D}) = \frac{1}{K} \sum_{n \in N_K(\boldsymbol{x}, \mathcal{D})} \mathbb{I}(y_n = c) \tag{1}$$

- c class label
- K: number of training samples in neighborhood

K nearest neighbor classifier

Basic idea: classify new/test input $\boldsymbol{x}$ by assigning to most probable (majority) label in the neighborhood of $\boldsymbol{x}$ (closest examples) from the training set.
Thus, we estimate:

$$p(y = c|\boldsymbol{x}, \mathcal{D}) = \frac{1}{K} \sum_{n \in N_K(\boldsymbol{x}, \mathcal{D})} \mathbb{I}(y_n = c) \tag{1}$$

- c class label
- K: number of training samples in neighborhood
- $N_K(\boldsymbol{x}, \mathcal{D})$: neighborhood of $\boldsymbol{x}$ (size K) based on dataset $\mathcal{D}$

Illustration of KNN

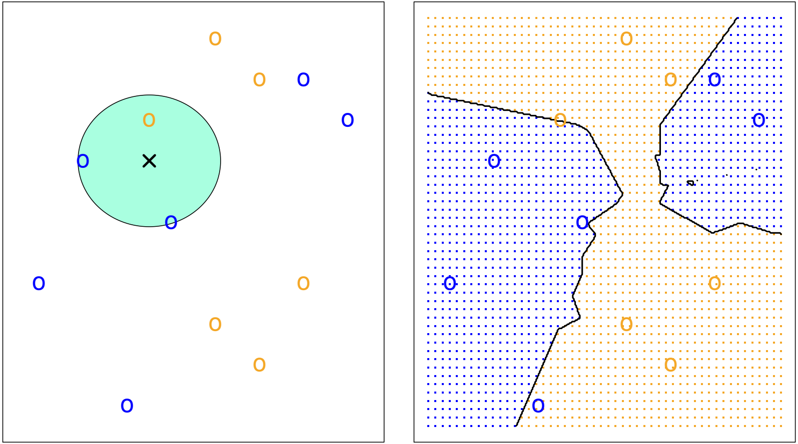


Figure: Illustration of the KNN approach on a training set of 12 observations and the resulting decision boundary. (ESL Fig 2.14)

Bias-variance trade-off in KNN

Bias-variance trade-off in KNN

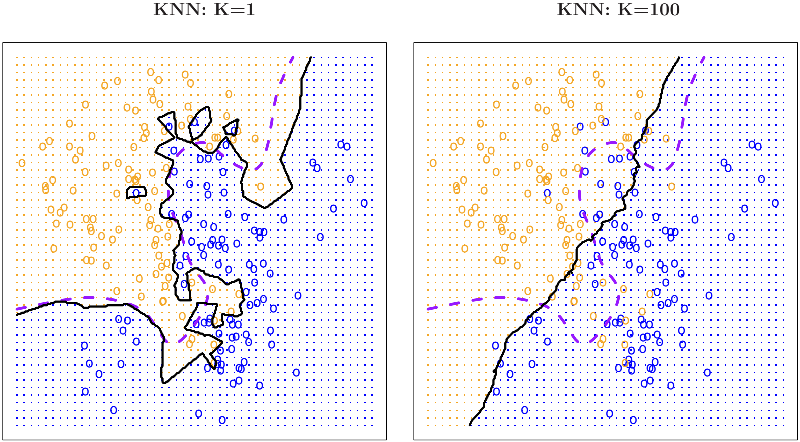


Figure: Comparing decision boundaries $K = 1$ and $K = 100$ for a dataset of 100 observations. Which model has lower bias? Which one gives a higher variance? The Bayes decision boundary is the purple dashed line (ESL Fig 2.16)

Approximating Bayes decision boundary with KNN

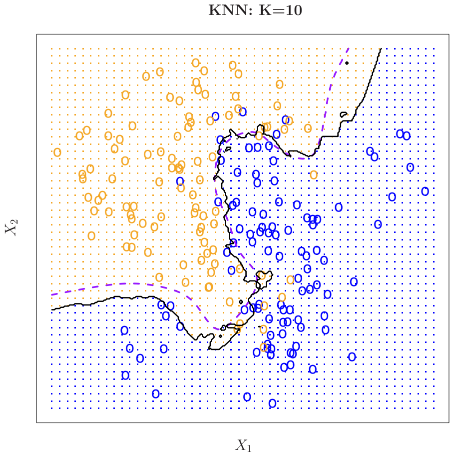


Figure: KNN decision boundary with $K = 10$ on the same training data set. (ESL Fig 2.15)

Training and test error rates for KNN

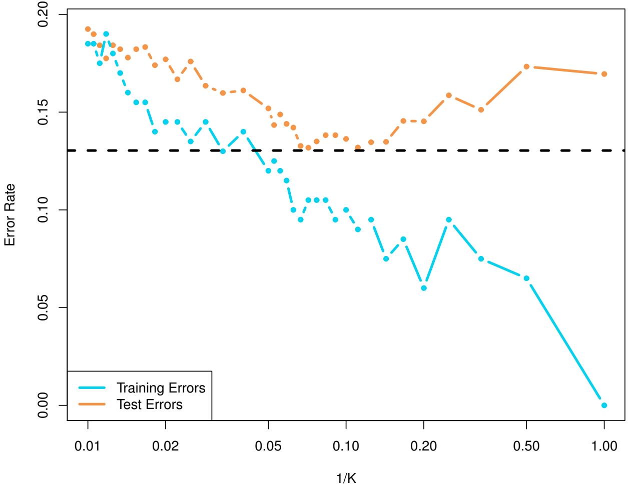


Figure: KNN training error rate (blue, 200 observations) and test error rate (orange, 5000 observations). Flexibility increases as K decreases. Which K should you choose? (ESL Fig 2.17)

KNN considerations

KNN considerations

- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$.

KNN considerations

- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$. This is done via a specified distance metric:

KNN considerations

- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$. This is done via a specified distance metric:$d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$

KNN considerations

- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$. This is done via a specified distance metric:$d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$ (e.g. Euclidean, Mahalanobis)

KNN considerations

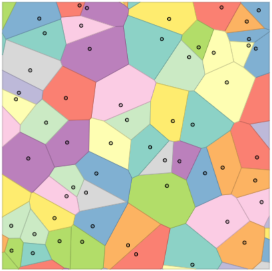
- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$. This is done via a specified distance metric:$d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$ (e.g. Euclidean, Mahalanobis)
- $K = 1$ induces a Voronoi tessellation: partitioning of input sapce such that all points $\boldsymbol{x} \in V(\boldsymbol{x}_n)$ are closer to $\boldsymbol{x}_n$ than to any other point

KNN considerations

- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$. This is done via a specified distance metric:$d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$ (e.g. Euclidean, Mahalanobis)
- $K = 1$ induces a Voronoi tessellation: partitioning of input sapce such that all points $\boldsymbol{x} \in V(\boldsymbol{x}_n)$ are closer to $\boldsymbol{x}_n$ than to any other point (From a modeling perspective, this is overfitting)

KNN considerations

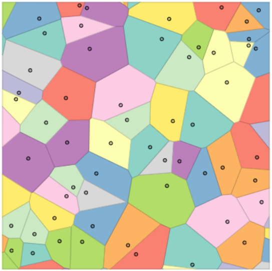
- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$. This is done via a specified distance metric:$d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$ (e.g. Euclidean, Mahalanobis)
- $K = 1$ induces a Voronoi tessellation: partitioning of input sapce such that all points $\boldsymbol{x} \in V(\boldsymbol{x}_n)$ are closer to $\boldsymbol{x}_n$ than to any other point (From a modeling perspective, this is overfitting)



Source: https://package.elm-lang.org/packages/ianmackenzie/elm-geometry/latest/VoronoiDiagram2d

KNN considerations

- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$. This is done via a specified distance metric:$d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$ (e.g. Euclidean, Mahalanobis)
- $K = 1$ induces a Voronoi tessellation: partitioning of input sapce such that all points $\boldsymbol{x} \in V(\boldsymbol{x}_n)$ are closer to $\boldsymbol{x}_n$ than to any other point (From a modeling perspective, this is overfitting)

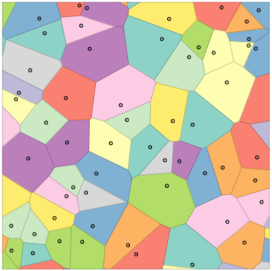


Source: https://package.elm-lang.org/packages/ianmackenzie/elm-geometry/latest/VoronoiDiagram2d

- Suffers under high dimensionality

KNN considerations

- To find the points in the neighborhood N_K, we need to determine the K-closest points to input $\boldsymbol{x}$. This is done via a specified distance metric:$d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$ (e.g. Euclidean, Mahalanobis)
- $K = 1$ induces a Voronoi tessellation: partitioning of input sapce such that all points $\boldsymbol{x} \in V(\boldsymbol{x}_n)$ are closer to $\boldsymbol{x}_n$ than to any other point (From a modeling perspective, this is overfitting)



Source: https://package.elm-lang.org/packages/ianmackenzie/elm-geometry/latest/VoronoiDiagram2d

- Suffers under high dimensionality
- Memory intensive

KNN extension: open set recognition

KNN extension: open set recognition

KNN is readily applicable to open set recognition (set of classes $\mathcal{C}$ not fixed).

KNN extension: open set recognition

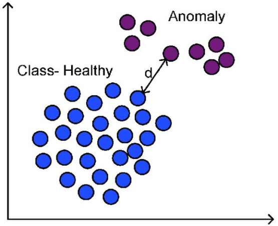
KNN is readily applicable to open set recognition (set of classes $\mathcal{C}$ not fixed).

- Novelty/out-of-distribution/anomaly detection

KNN extension: open set recognition

KNN is readily applicable to open set recognition (set of classes $\mathcal{C}$ not fixed).

- Novelty/out-of-distribution/anomaly detection



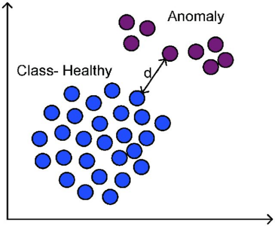
Source: https://www.intechopen.com/chapters/74393

- Incremental/online/life-long/continual learning: any potentially new label is added to new class C_{t+1}; dataset augmented

KNN extension: open set recognition

KNN is readily applicable to open set recognition (set of classes $\mathcal{C}$ not fixed).

- Novelty/out-of-distribution/anomaly detection



Source: https://www.intechopen.com/chapters/74393

- Incremental/online/life-long/continual learning: any potentially new label is added to new class C_{t+1}; dataset augmented
- Few-shot classification (for person re-identification or face verification)

Distance metrics

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed
- Distance/similarity required for KNN, unsupervised learning (e.g. clustering) among other tasks

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed
- Distance/similarity required for KNN, unsupervised learning (e.g. clustering) among other tasks
- Common metrics:

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed
- Distance/similarity required for KNN, unsupervised learning (e.g. clustering) among other tasks
- Common metrics:
  - Euclidean distance:

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed
- Distance/similarity required for KNN, unsupervised learning (e.g. clustering) among other tasks
- Common metrics:
    - Euclidean distance:

$$d_{\boldsymbol{E}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^\top (\boldsymbol{x} - \boldsymbol{x}')} \quad (2)$$

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^{+}$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed
- Distance/similarity required for KNN, unsupervised learning (e.g. clustering) among other tasks
- Common metrics:
  - Euclidean distance:

$$d_{\boldsymbol{E}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^{\top}(\boldsymbol{x} - \boldsymbol{x}')} \tag{2}$$

  - Mahalanobis distance:

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed
- Distance/similarity required for KNN, unsupervised learning (e.g. clustering) among other tasks
- Common metrics:
  - Euclidean distance:

$$d_{\boldsymbol{E}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^\top (\boldsymbol{x} - \boldsymbol{x}')} \tag{2}$$

  - Mahalanobis distance:

$$d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{M} (\boldsymbol{x} - \boldsymbol{x}')} \tag{3}$$

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed
- Distance/similarity required for KNN, unsupervised learning (e.g. clustering) among other tasks
- Common metrics:
  - Euclidean distance:

$$d_{\boldsymbol{E}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^\top (\boldsymbol{x} - \boldsymbol{x}')} \tag{2}$$

  - Mahalanobis distance:

$$d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{M} (\boldsymbol{x} - \boldsymbol{x}')} \tag{3}$$

  where $\boldsymbol{M}$ is the Mahalanobis distance matrix

Distance metrics

The [semantic] distance between points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is specified by a distance metric $d(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^+$.

- Alternately, the similarity $s(\boldsymbol{x}, \boldsymbol{x}')$ can be computed
- Distance/similarity required for KNN, unsupervised learning (e.g. clustering) among other tasks
- Common metrics:
  - Euclidean distance:

$$d_{\boldsymbol{E}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^\top (\boldsymbol{x} - \boldsymbol{x}')} \tag{2}$$

  - Mahalanobis distance:

$$d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{M} (\boldsymbol{x} - \boldsymbol{x}')} \tag{3}$$

where $\boldsymbol{M}$ is the Mahalanobis distance matrix

Distance metrics

The process of finding the optimal $\boldsymbol{M}$ is called **metric learning**

Distance metrics

The process of finding the optimal $\boldsymbol{M}$ is called **metric learning**

- When D is large, we typically learn an embedding (mapping): $\boldsymbol{e} = f(\boldsymbol{x})$ and then compute $d_{\boldsymbol{M}}(\boldsymbol{e}, \boldsymbol{e}')$ instead.

Distance metrics

The process of finding the optimal $\boldsymbol{M}$ is called **metric learning**

- When D is large, we typically learn an embedding (mapping): $\boldsymbol{e} = f(\boldsymbol{x})$ and then compute $d_{\boldsymbol{M}}(\boldsymbol{e}, \boldsymbol{e}')$ instead.
- When f is a deep neural network, this process is termed **deep metric learning**

Methods for estimating M

Methods for estimating M

- Large margin nearest neighbors (LMNN)

Methods for estimating M

- Large margin nearest neighbors (LMNN)
- Neighborhood component analysis (NCA)

Methods for estimating M

- Large margin nearest neighbors (LMNN)
- Neighborhood component analysis (NCA)
- Latent coincidence analysis (LCA)

Methods for estimating M

- Large margin nearest neighbors (LMNN)
- Neighborhood component analysis (NCA)
- Latent coincidence analysis (LCA)
- Minimization of classification and ranking losses (with mining techniques and proxy methods):

Methods for estimating M

- Large margin nearest neighbors (LMNN)
- Neighborhood component analysis (NCA)
- Latent coincidence analysis (LCA)
- Minimization of classification and ranking losses (with mining techniques and proxy methods):
    - Pairwise/contrastive loss

Methods for estimating M

- Large margin nearest neighbors (LMNN)
- Neighborhood component analysis (NCA)
- Latent coincidence analysis (LCA)
- Minimization of classification and ranking losses (with mining techniques and proxy methods):
  - Pairwise/contrastive loss
  - Triplet loss

Methods for estimating M

- Large margin nearest neighbors (LMNN)
- Neighborhood component analysis (NCA)
- Latent coincidence analysis (LCA)
- Minimization of classification and ranking losses (with mining techniques and proxy methods):
  - Pairwise/contrastive loss
  - Triplet loss
  - N-pairs loss

Density kernel

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.
Density kernels have two important properties:

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.
Density kernels have two important properties:

- **Normalization**:

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.

Density kernels have two important properties:

- **Normalization**:

$$\int x\mathcal{K}(x)dx = 0 \tag{4}$$

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.
Density kernels have two important properties:

- **Normalization**:

$$\int x\mathcal{K}(x)dx = 0 \tag{4}$$

- **Symmetry**:

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.
Density kernels have two important properties:

- **Normalization**:

$$\int x\mathcal{K}(x)dx = 0 \tag{4}$$

- **Symmetry**:

$$\mathcal{K}(-x) = \mathcal{K}x \tag{5}$$

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \rightarrow \mathbb{R}_{+}$.
Density kernels have two important properties:

- **Normalization**:

$$\int x\mathcal{K}(x)dx = 0 \tag{4}$$

- **Symmetry**:

$$\mathcal{K}(-x) = \mathcal{K}x \tag{5}$$

Kernels have several uses, e.g.

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.
Density kernels have two important properties:

- **Normalization**:

$$\int x\mathcal{K}(x)dx = 0 \tag{4}$$

- **Symmetry**:

$$\mathcal{K}(-x) = \mathcal{K}x \tag{5}$$

Kernels have several uses, e.g.

- density function estimation

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.
Density kernels have two important properties:

- **Normalization**:

$$\int x\mathcal{K}(x)dx = 0 \tag{4}$$

- **Symmetry**:

$$\mathcal{K}(-x) = \mathcal{K}x \tag{5}$$

Kernels have several uses, e.g.

- density function estimation
- local regression (our focus)

Density kernel

A density kernel $\mathcal{K}(x)$ is a weighting function that specifies a mapping or transformation of an input x: $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$.
Density kernels have two important properties:

- **Normalization**:

$$\int x\mathcal{K}(x)dx = 0 \tag{4}$$

- **Symmetry**:

$$\mathcal{K}(-x) = \mathcal{K}x \tag{5}$$

Kernels have several uses, e.g.

- density function estimation
- local regression (our focus)
- smoothing time series

Kernel functions

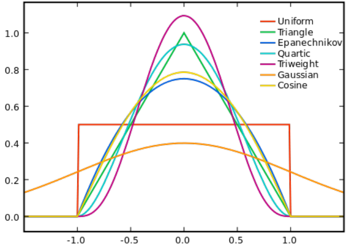


Figure: Popular kernel functions

Kernel functions

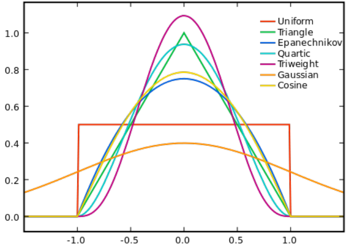


Figure: Popular kernel functions

Kernel functions

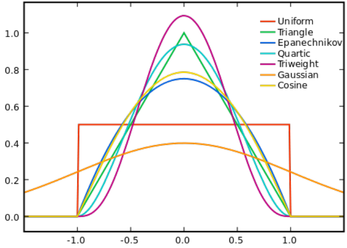


Figure: Popular kernel functions

- Triangular: $\mathcal{K}(x) = (1 - |x|)$
- Epanechnikov: $\mathcal{K}(x) = \frac{3}{4}(1 - x^2)$
- Triweight: $\mathcal{K}(x) = \frac{35}{32}(1 - x^2)^3$
- Tricube: $\mathcal{K}(x) = \frac{70}{81}(1 - |x|^3)^3$
- Gaussian: $\mathcal{K}(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$

Radial basis function

Radial basis function

The radial basis function (RBF) kernel is a generalization of a density kernel to an input vector $\boldsymbol{x}$:

Radial basis function

The radial basis function (RBF) kernel is a generalization of a density kernel to an input vector $\boldsymbol{x}$:

$$\mathcal{K}_h(\boldsymbol{x}) \propto \mathcal{K}_h(||\boldsymbol{x}||) \tag{6}$$

Radial basis function

The radial basis function (RBF) kernel is a generalization of a density kernel to an input vector $\boldsymbol{x}$:

$$\mathcal{K}_h(\boldsymbol{x}) \propto \mathcal{K}_h(||\boldsymbol{x}||) \tag{6}$$

where h is the bandwidth parameter:

Radial basis function

The radial basis function (RBF) kernel is a generalization of a density kernel to an input vector $\boldsymbol{x}$:

$$\mathcal{K}_h(\boldsymbol{x}) \propto \mathcal{K}_h(||\boldsymbol{x}||) \tag{6}$$

where h is the bandwidth parameter:
The RBF Gaussian kernel is thus given by:

Radial basis function

The radial basis function (RBF) kernel is a generalization of a density kernel to an input vector $\boldsymbol{x}$:

$$\mathcal{K}_h(\boldsymbol{x}) \propto \mathcal{K}_h(||\boldsymbol{x}||) \tag{6}$$

where h is the bandwidth parameter:
The RBF Gaussian kernel is thus given by:

$$\mathcal{K}_h(\boldsymbol{x}) = \frac{1}{h^D(2\pi)^{D/2}} \prod_{d=1}^{D} \exp\left[-\frac{1}{2h^2} x_d^2\right] \tag{7}$$

Radial basis function

The radial basis function (RBF) kernel is a generalization of a density kernel to an input vector $\boldsymbol{x}$:

$$\mathcal{K}_h(\boldsymbol{x}) \propto \mathcal{K}_h(||\boldsymbol{x}||) \tag{6}$$

where h is the bandwidth parameter:
The RBF Gaussian kernel is thus given by:

$$\mathcal{K}_h(\boldsymbol{x}) = \frac{1}{h^D(2\pi)^{D/2}} \prod_{d=1}^{D} \exp\left[-\frac{1}{2h^2} x_d^2\right] \tag{7}$$

Bandwidth parameter

Radial basis function

The radial basis function (RBF) kernel is a generalization of a density kernel to an input vector $\boldsymbol{x}$:

$$\mathcal{K}_h(\boldsymbol{x}) \propto \mathcal{K}_h(||\boldsymbol{x}||) \tag{6}$$

where h is the bandwidth parameter:
The RBF Gaussian kernel is thus given by:

$$\mathcal{K}_h(\boldsymbol{x}) = \frac{1}{h^D(2\pi)^{D/2}} \prod_{d=1}^{D} \exp\left[-\frac{1}{2h^2}x_d^2\right] \tag{7}$$

Bandwidth parameter

Specifies the width of the kernel:

Radial basis function

The radial basis function (RBF) kernel is a generalization of a density kernel to an input vector $\boldsymbol{x}$:

$$\mathcal{K}_h(\boldsymbol{x}) \propto \mathcal{K}_h(||\boldsymbol{x}||) \tag{6}$$

where h is the bandwidth parameter:
The RBF Gaussian kernel is thus given by:

$$\mathcal{K}_h(\boldsymbol{x}) = \frac{1}{h^D(2\pi)^{D/2}} \prod_{d=1}^{D} \exp\left[-\frac{1}{2h^2} x_d^2\right] \tag{7}$$

Bandwidth parameter

Specifies the width of the kernel:

$$\mathcal{K}_h := \frac{1}{h}\mathcal{K}\left(\frac{x}{h}\right) \tag{8}$$

K-nearest neighbor smoother

K-nearest neighbor smoother

In the simple case of the KNN kernel, we use the neighborhood average:

K-nearest neighbor smoother

In the simple case of the KNN kernel, we use the neighborhood average:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in N_K(x_0)} y_i \tag{9}$$

K-nearest neighbor smoother

In the simple case of the KNN kernel, we use the neighborhood average:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in N_K(x_0)} y_i \tag{9}$$

where $N_K(x_0)$ is the K-nearest neighborhood of x_0.

K-nearest neighbor smoother

In the simple case of the KNN kernel, we use the neighborhood average:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in N_K(x_0)} y_i \tag{9}$$

where $N_K(x_0)$ is the K-nearest neighborhood of x_0.

- All points in the neighborhood are equally weighted

K-nearest neighbor smoother

In the simple case of the KNN kernel, we use the neighborhood average:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in N_K(x_0)} y_i \tag{9}$$

where $N_K(x_0)$ is the K-nearest neighborhood of x_0.

- All points in the neighborhood are equally weighted
- The resulting $\hat{f}(x)$ is not smooth

K-nearest neighbor smoother

In the simple case of the KNN kernel, we use the neighborhood average:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in N_K(x_0)} y_i \tag{9}$$

where $N_K(x_0)$ is the K-nearest neighborhood of x_0.

- All points in the neighborhood are equally weighted
- The resulting $\hat{f}(x)$ is not smooth
- To achieve smoothness, we weight observations by distance to the target point

K-nearest neighbor smoother

In the simple case of the KNN kernel, we use the neighborhood average:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in N_K(x_0)} y_i \tag{9}$$

where $N_K(x_0)$ is the K-nearest neighborhood of x_0.

- All points in the neighborhood are equally weighted
- The resulting $\hat{f}(x)$ is not smooth
- To achieve smoothness, we weight observations by distance to the target point

K-nearest neighbor smoother

In the simple case of the KNN kernel, we use the neighborhood average:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in N_K(x_0)} y_i \tag{9}$$

where $N_K(x_0)$ is the K-nearest neighborhood of x_0.

- All points in the neighborhood are equally weighted
- The resulting $\hat{f}(x)$ is not smooth
- To achieve smoothness, we weight observations by distance to the target point

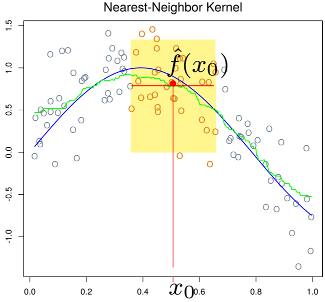


Figure: KNN equally-weighted kernel

Nadaraya-Watson smoother

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] =$$

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0)$$

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

where K_λ can be any kernel function.

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

where K_λ can be any kernel function.

If we use the popular **Epanechnikov** (quadratic) kernel, then:

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

where K_λ can be any kernel function.

If we use the popular **Epanechnikov** (quadratic) kernel, then:

$$K_\lambda(x_o, x_i)$$

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

where K_λ can be any kernel function.

If we use the popular **Epanechnikov** (quadratic) kernel, then:

$$K_\lambda(x_o, x_i) = D\left(\frac{|x_i - x_0|}{\lambda}\right) \tag{11}$$

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

where K_λ can be any kernel function.

If we use the popular **Epanechnikov** (quadratic) kernel, then:

$$K_\lambda(x_o, x_i) = D\left(\frac{|x_i - x_0|}{\lambda}\right) \tag{11}$$

where

$$D(t) =$$

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

where K_λ can be any kernel function.

If we use the popular **Epanechnikov** (quadratic) kernel, then:

$$K_\lambda(x_o, x_i) = D\left(\frac{|x_i - x_0|}{\lambda}\right) \tag{11}$$

where

$$D(t) = \begin{cases} \frac{3}{4}\left(1 - t^2\right) & |t| \leq 1 \\ \end{cases}$$

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

where K_λ can be any kernel function.

If we use the popular **Epanechnikov** (quadratic) kernel, then:

$$K_\lambda(x_o, x_i) = D\left(\frac{|x_i - x_0|}{\lambda}\right) \tag{11}$$

where

$$D(t) = \begin{cases} \frac{3}{4}\left(1 - t^2\right) & |t| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Nadaraya-Watson smoother

Nadaraya-Watson kernel-weighted average implements distance-based weighting:

$$\mathbb{E}[y|\boldsymbol{x}, \mathcal{D}] = \hat{f}(x_0) = \frac{\sum_{i=1}^{n} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} \tag{10}$$

where K_λ can be any kernel function.

If we use the popular **Epanechnikov** (quadratic) kernel, then:

$$K_\lambda(x_o, x_i) = D\left(\frac{|x_i - x_0|}{\lambda}\right) \tag{11}$$

where

$$D(t) = \begin{cases} \frac{3}{4}\left(1 - t^2\right) & |t| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

In D, the half-width (or *bandwidth*) of the neighborhood is given by λ.

Nadaraya-Watson smoother (Epanechnikov kernel)

Nadaraya-Watson smoother (Epanechnikov kernel)

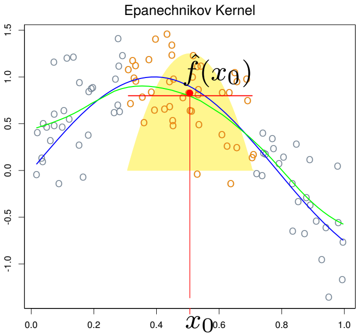


Figure: Nadaraya-Watson estimate of $\hat{f}(x)$ using the Epanechnikov kernel. Half-width is fixed at $\lambda = 0.2$

Nadaraya-Watson smoother (Epanechnikov kernel)

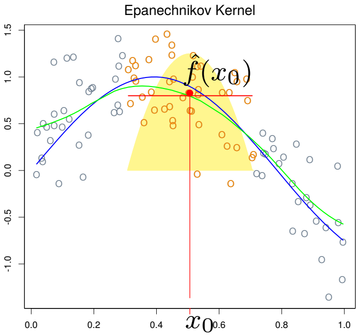


Figure: Nadaraya-Watson estimate of $\hat{f}(x)$ using the Epanechnikov kernel. Half-width is fixed at $\lambda = 0.2$

Nadaraya-Watson smoother (Epanechnikov kernel)

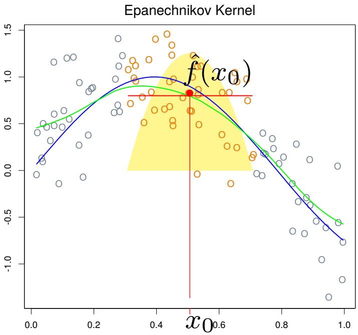


Figure: Nadaraya-Watson estimate of $\hat{f}(x)$ using the Epanechnikov kernel. Half-width is fixed at $\lambda = 0.2$

The half-width can be generalized as any function h_λ of the target point x_0:

Nadaraya-Watson smoother (Epanechnikov kernel)

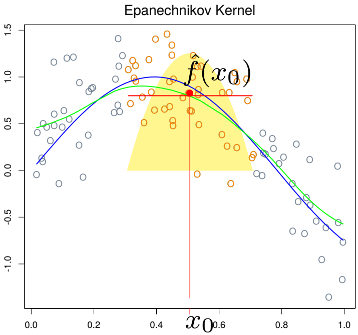


Figure: Nadaraya-Watson estimate of $\hat{f}(x)$ using the Epanechnikov kernel. Half-width is fixed at $\lambda = 0.2$

The half-width can be generalized as any function h_λ of the target point x_0:

$$K_\lambda(x_0, x_i)$$

Nadaraya-Watson smoother (Epanechnikov kernel)

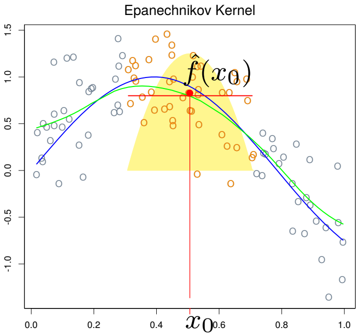


Figure: Nadaraya-Watson estimate of $\hat{f}(x)$ using the Epanechnikov kernel. Half-width is fixed at $\lambda = 0.2$

The half-width can be generalized as any function h_λ of the target point x_0:

$$K_\lambda(x_0, x_i) = D\left(\frac{|x_i - x_0|}{h_\lambda(x_0)}\right) \tag{13}$$

Kernels as a localization device

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point
- the function defining how the points in the neighborhood of x_0 are weighted is called a **kernel**:

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point
- the function defining how the points in the neighborhood of x_0 are weighted is called a **kernel**:

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point
- the function defining how the points in the neighborhood of x_0 are weighted is called a **kernel**: $K_\lambda(x_0, x_i)$ where λ controls the width of the neighborhood
- this is a localized/memory-based approach

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point
- the function defining how the points in the neighborhood of x_0 are weighted is called a **kernel**: $K_\lambda(x_0, x_i)$ where λ controls the width of the neighborhood
- this is a localized/memory-based approach
- the resulting $\hat{f}(X)$ is smooth in $\mathbb{R}^D$

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point
- the function defining how the points in the neighborhood of x_0 are weighted is called a **kernel**: $K_\lambda(x_0, x_i)$ where λ controls the width of the neighborhood
- this is a localized/memory-based approach
- the resulting $\hat{f}(X)$ is smooth in $\mathbb{R}^D$

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point
- the function defining how the points in the neighborhood of x_0 are weighted is called a **kernel**: $K_\lambda(x_0, x_i)$ where λ controls the width of the neighborhood
- this is a localized/memory-based approach
- the resulting $\hat{f}(X)$ is smooth in $\mathbb{R}^D$

Using kernel functions, we can estimate $\hat{f}(X)$ in two ways:

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point
- the function defining how the points in the neighborhood of x_0 are weighted is called a **kernel**: $K_\lambda(x_0, x_i)$ where λ controls the width of the neighborhood
- this is a localized/memory-based approach
- the resulting $\hat{f}(X)$ is smooth in $\mathbb{R}^D$

Using kernel functions, we can estimate $\hat{f}(X)$ in two ways:

1. **Nonparametric**: define an averaging function (kernel) to estimate y_0 for each point x_0

Kernels as a localization device

Rather than estimate a regression function $f(X)$ over the entire $\mathbb{R}^D$, we can estimate the response at each training point using a weighted average:

- only observations close a target point x_0 are used in estimating f at that point
- the function defining how the points in the neighborhood of x_0 are weighted is called a **kernel**: $K_\lambda(x_0, x_i)$ where λ controls the width of the neighborhood
- this is a localized/memory-based approach
- the resulting $\hat{f}(X)$ is smooth in $\mathbb{R}^D$

Using kernel functions, we can estimate $\hat{f}(X)$ in two ways:

1. **Nonparametric**: define an averaging function (kernel) to estimate y_0 for each point x_0
2. **Parametric**: estimate a linear model for each point x_0

Local linear regression

Local linear regression

Using a nonparametric average function produces biased estimates at the boundaries.

Local linear regression

Using a nonparametric average function produces biased estimates at the boundaries.

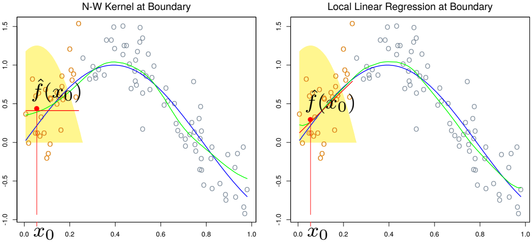


Figure: Locally weighted average (Nadaraya-Watson) versus local linear regression.

To correct this, we can estimate a linear model at each point x_0 by solving a weighted least squares:

Local linear regression

Using a nonparametric average function produces biased estimates at the boundaries.

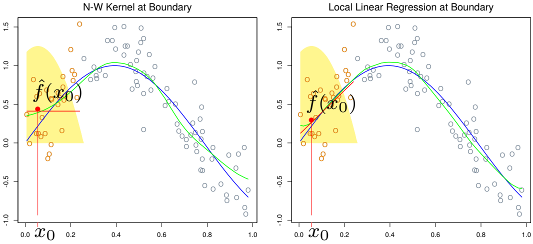


Figure: Locally weighted average (Nadaraya-Watson) versus local linear regression.

To correct this, we can estimate a linear model at each point x_0 by solving a weighted least squares:

Local linear regression

Using a nonparametric average function produces biased estimates at the boundaries.

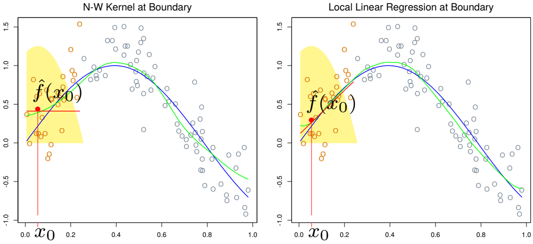


Figure: Locally weighted average (Nadaraya-Watson) versus local linear regression.

To correct this, we can estimate a linear model at each point x_0 by solving a weighted least squares:

$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{n} K_\lambda(x_0, x_i)[y_i - \alpha(x_0) - \beta(x_0)x_i]^2 \tag{14}$$

Local linear regression estimates

Local linear regression estimates

Let $b(x)^T = (1, x)$ and:

$$\boldsymbol{B} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad \boldsymbol{W}(x_0) = \begin{pmatrix} K_\lambda(x_0, x_1) & 0 & \cdots & 0 \\ 0 & K_\lambda(x_0, x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_\lambda(x_0, x_n) \end{pmatrix} \tag{15}$$

Then the solution to the locally weighted regression problem is:

Local linear regression estimates

Let $b(x)^T = (1, x)$ and:

$$\boldsymbol{B} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad \boldsymbol{W}(x_0) = \begin{pmatrix} K_\lambda(x_0, x_1) & 0 & \cdots & 0 \\ 0 & K_\lambda(x_0, x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_\lambda(x_0, x_n) \end{pmatrix} \tag{15}$$

Then the solution to the locally weighted regression problem is:

$$\hat{f}(x_0) \quad =$$

Local linear regression estimates

Let $b(x)^T = (1, x)$ and:

$$\boldsymbol{B} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad \boldsymbol{W}(x_0) = \begin{pmatrix} K_\lambda(x_0, x_1) & 0 & \cdots & 0 \\ 0 & K_\lambda(x_0, x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_\lambda(x_0, x_n) \end{pmatrix} \tag{15}$$

Then the solution to the locally weighted regression problem is:

$$\hat{f}(x_0) \quad = \quad b(x_0)^T (\boldsymbol{B}^T \boldsymbol{W}(x_0) \boldsymbol{B})^{-1} \boldsymbol{B}^T \boldsymbol{W}(x_0) \boldsymbol{y} \tag{16}$$

Local linear regression estimates

Let $b(x)^T = (1, x)$ and:

$$\boldsymbol{B} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad \boldsymbol{W}(x_0) = \begin{pmatrix} K_\lambda(x_0, x_1) & 0 & \cdots & 0 \\ 0 & K_\lambda(x_0, x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_\lambda(x_0, x_n) \end{pmatrix} \tag{15}$$

Then the solution to the locally weighted regression problem is:

$$\hat{f}(x_0) = b(x_0)^T(\boldsymbol{B}^T\boldsymbol{W}(x_0)\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{W}(x_0)\boldsymbol{y} \tag{16}$$

$$\hat{f}(x_0) =$$

Local linear regression estimates

Let $b(x)^T = (1, x)$ and:

$$\boldsymbol{B} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad \boldsymbol{W}(x_0) = \begin{pmatrix} K_\lambda(x_0, x_1) & 0 & \cdots & 0 \\ 0 & K_\lambda(x_0, x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_\lambda(x_0, x_n) \end{pmatrix} \tag{15}$$

Then the solution to the locally weighted regression problem is:

$$\hat{f}(x_0) = b(x_0)^T(\boldsymbol{B}^T\boldsymbol{W}(x_0)\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{W}(x_0)\boldsymbol{y} \tag{16}$$

$$\hat{f}(x_0) = \sum_{i=1}^{n} \ell_i(x_0)y_i \tag{17}$$

Local linear regression estimates

Let $b(x)^T = (1, x)$ and:

$$\boldsymbol{B} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad \boldsymbol{W}(x_0) = \begin{pmatrix} K_\lambda(x_0, x_1) & 0 & \cdots & 0 \\ 0 & K_\lambda(x_0, x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_\lambda(x_0, x_n) \end{pmatrix} \tag{15}$$

Then the solution to the locally weighted regression problem is:

$$\hat{f}(x_0) = b(x_0)^T(\boldsymbol{B}^T\boldsymbol{W}(x_0)\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{W}(x_0)\boldsymbol{y} \tag{16}$$

$$\hat{f}(x_0) = \sum_{i=1}^{n} \ell_i(x_0) y_i \tag{17}$$

The weights $\ell_i(x_0)$ are called the **equivalent kernel**

Effect of equivalent kernel

Effect of equivalent kernel

The equivalent kernel (local regression) corrects the bias from local average kernel methods to the first order.

Effect of equivalent kernel

The equivalent kernel (local regression) corrects the bias from local average kernel methods to the first order.

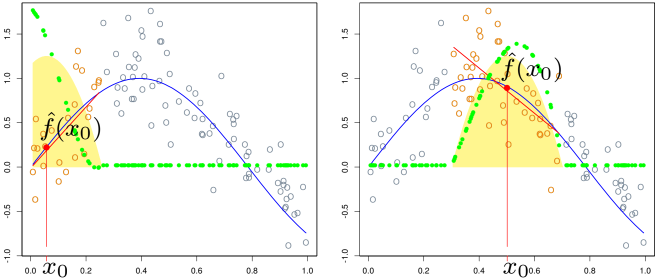


Figure:

Effect of equivalent kernel

The equivalent kernel (local regression) corrects the bias from local average kernel methods to the first order.

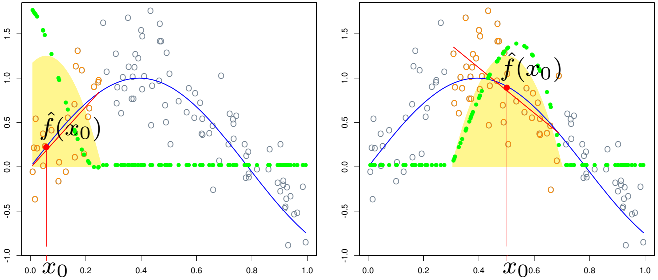


Figure:

Effect of equivalent kernel

The equivalent kernel (local regression) corrects the bias from local average kernel methods to the first order.

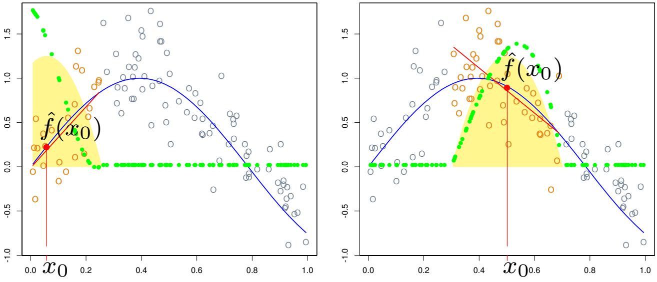


Figure:

Effect of equivalent kernel

The equivalent kernel (local regression) corrects the bias from local average kernel methods to the first order.

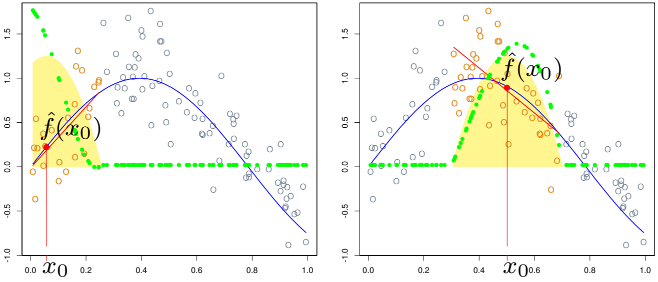


Figure: Green points show the weights $\ell_i(x_0)$ (rescaled for display purposes) along with those for the Nadaraya-Watson (N-W) local average (yellow shaded region; also rescaled).

Effect of equivalent kernel

The equivalent kernel (local regression) corrects the bias from local average kernel methods to the first order.

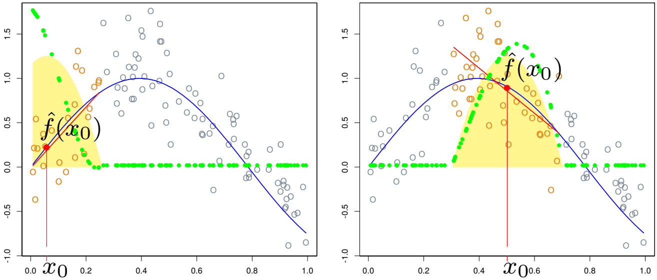


Figure: Green points show the weights $\ell_i(x_0)$ (rescaled for display purposes) along with those for the Nadaraya-Watson (N-W) local average (yellow shaded region; also rescaled). The correction effect of local regression can be observed.

Local polynomial regression

Local polynomial regression

- Higher-order terms in $\hat{f}(x)$ are required to reduce bias in curved regions

Local polynomial regression

- Higher-order terms in $\hat{f}(x)$ are required to reduce bias in curved regions
- Local polynomial regression can correct this at the cost of higher variance

Local polynomial regression

- Higher-order terms in $\hat{f}(x)$ are required to reduce bias in curved regions
- Local polynomial regression can correct this at the cost of higher variance
- Given by:

Local polynomial regression

- Higher-order terms in $\hat{f}(x)$ are required to reduce bias in curved regions
- Local polynomial regression can correct this at the cost of higher variance
- Given by:

Local polynomial regression

- Higher-order terms in $\hat{f}(x)$ are required to reduce bias in curved regions
- Local polynomial regression can correct this at the cost of higher variance
- Given by:

$$\hat{f}(x_0) = \hat{\alpha}(x_0) + \sum_{j=1}^{d} \hat{\beta}_j(x_0)x_0^j \tag{18}$$

Local polynomial regression

- Higher-order terms in $\hat{f}(x)$ are required to reduce bias in curved regions
- Local polynomial regression can correct this at the cost of higher variance
- Given by:

$$\hat{f}(x_0) = \hat{\alpha}(x_0) + \sum_{j=1}^{d} \hat{\beta}_j(x_0) x_0^j \tag{18}$$

where $\hat{f}(x_0)$ is the solution to:

Local polynomial regression

- Higher-order terms in $\hat{f}(x)$ are required to reduce bias in curved regions
- Local polynomial regression can correct this at the cost of higher variance
- Given by:

$$\hat{f}(x_0) = \hat{\alpha}(x_0) + \sum_{j=1}^{d} \hat{\beta}_j(x_0)x_0^j \tag{18}$$

where $\hat{f}(x_0)$ is the solution to:

$$\min_{\alpha(x_0),\beta(x_0),j=1,\ldots,d} \sum_{i=1}^{n} K_\lambda(x_0, x_i)\left[y_i - \alpha(x_0) - \sum_{j=1}^{d} \beta_j(x_0)x_i^j\right]^2 \tag{19}$$

Bias correction of local quadratic regression

Bias correction of local quadratic regression

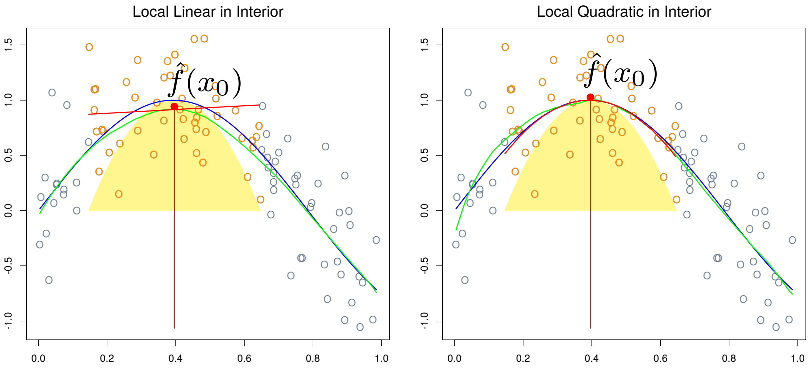


Figure: The local linear regression estimator is biased in curved regions. A higher-order local fit (in this case, quadratic) can eliminate this.

Local regression algorithms

Local regression algorithms

- LOESS: locally estimated scatterplot smoothing

Local regression algorithms

- LOESS: locally estimated scatterplot smoothing
  - Identical approached earlier developed in 1964 by Savitzky and Golay for smoothing noisy data (known as the Savitzky-Golay filter)
  - "Rediscovered" by William Cleveland in 1979
- LOWESS: locally weighted scatterplot smoothing
  - Extension of LOESS by Cleveland and Susan Devlin (1988)
- LOESS can be considered a generalization of LOWESS, as it fits multivariate data, while LOWESS is for univariate cases

Summary

Summary

- Kernel smoothing methods can be used for flexible functional fitting

Summary

- Kernel smoothing methods can be used for flexible functional fitting
- Local regression generates a linear/polynomial fit at each target point using a kernel weighted loss function

Summary

- Kernel smoothing methods can be used for flexible functional fitting
- Local regression generates a linear/polynomial fit at each target point using a kernel weighted loss function

Summary

- Kernel smoothing methods can be used for flexible functional fitting
- Local regression generates a linear/polynomial fit at each target point using a kernel weighted loss function

Reading:

- **PMLI** 16

Summary

- Kernel smoothing methods can be used for flexible functional fitting
- Local regression generates a linear/polynomial fit at each target point using a kernel weighted loss function

Reading:

- **PMLI** 16
- **ESL 6.1** One-dimensional Kernel Smoothers (pp. 191–199)

Summary

- Kernel smoothing methods can be used for flexible functional fitting
- Local regression generates a linear/polynomial fit at each target point using a kernel weighted loss function

Reading:

- **PMLI** 16
- **ESL 6.1** One-dimensional Kernel Smoothers (pp. 191–199)
- **ISLR 7.6:** Local Regression (pp. 280–282)