Extending Linear Regression

Ngoc Hoang Luong

University of Information Technology (UIT), VNU-HCM

April 13, 2023

Linear Regression

- Regression: To model an output value y_i in terms of one or more input features $\mathbf{x}_i$ (a p-vector).

Linear Regression

- Regression: To model an output value y_i in terms of one or more input features $\mathbf{x}_i$ (a p-vector).
- In addition, there is a noise or error term ϵ_i to indicate the imperfect nature of our model:

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

Linear Regression

- Regression: To model an output value y_i in terms of one or more input features $\mathbf{x}_i$ (a p-vector).
- In addition, there is a noise or error term ϵ_i to indicate the imperfect nature of our model:

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

- We assume the error terms ϵ_i are independent from the input features with zero-mean and constant variance:

$$\mathbb{E}(\epsilon_i) = 0; \quad Var(\epsilon_i) = \sigma^2$$

Linear Regression

- Regression: To model an output value y_i in terms of one or more input features $\mathbf{x}_i$ (a p-vector).
- In addition, there is a noise or error term ϵ_i to indicate the imperfect nature of our model:

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

- We assume the error terms ϵ_i are independent from the input features with zero-mean and constant variance:

$$\mathbb{E}(\epsilon_i) = 0; \quad Var(\epsilon_i) = \sigma^2$$

- And, there is:

Linear Regression

- Regression: To model an output value y_i in terms of one or more input features $\mathbf{x}_i$ (a p-vector).
- In addition, there is a noise or error term ϵ_i to indicate the imperfect nature of our model:

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

- We assume the error terms ϵ_i are independent from the input features with zero-mean and constant variance:

$$\mathbb{E}(\epsilon_i) = 0; \quad Var(\epsilon_i) = \sigma^2$$

- And, there is:
  - A join distribution $P(y, \mathbf{x})$

Linear Regression

- Regression: To model an output value y_i in terms of one or more input features $\mathbf{x}_i$ (a p-vector).
- In addition, there is a noise or error term ϵ_i to indicate the imperfect nature of our model:

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

- We assume the error terms ϵ_i are independent from the input features with zero-mean and constant variance:

$$\mathbb{E}(\epsilon_i) = 0; \quad Var(\epsilon_i) = \sigma^2$$

- And, there is:
  - A join distribution $P(y, \mathbf{x})$
  - A marginal distribution for the input features $P(\mathbf{x})$

Linear Regression

- Regression: To model an output value y_i in terms of one or more input features $\mathbf{x}_i$ (a p-vector).
- In addition, there is a noise or error term ϵ_i to indicate the imperfect nature of our model:

$$y_i = f(\mathbf{x}_i) + \epsilon_i$$

- We assume the error terms ϵ_i are independent from the input features with zero-mean and constant variance:

$$\mathbb{E}(\epsilon_i) = 0; \quad Var(\epsilon_i) = \sigma^2$$

- And, there is:
  - A join distribution $P(y, \mathbf{x})$
  - A marginal distribution for the input features $P(\mathbf{x})$
  - A conditional distribution $P(y \mid \mathbf{x})$

Linear Regression

- Regression: to model $f()$, called the **regression function**, as the conditional expectation of $y \mid \mathbf{x}$:

$$\mathbb{E}(y \mid \mathbf{x}) = \hat{f}(\mathbf{x})$$

Linear Regression

- Regression: to model $f()$, called the **regression function**, as the conditional expectation of $y \mid \mathbf{x}$:

$$\mathbb{E}(y \mid \mathbf{x}) = \hat{f}(\mathbf{x})$$

- The regression framework says nothing about what the target function $f()$ should look like. We can decide any form for $\hat{f}()$.

Linear Regression

- Regression: to model $f()$, called the **regression function**, as the conditional expectation of $y \mid \mathbf{x}$:

$$\mathbb{E}(y \mid \mathbf{x}) = \hat{f}(\mathbf{x})$$

- The regression framework says nothing about what the target function $f()$ should look like. We can decide any form for $\hat{f}()$.
- A standard form of $\hat{f}()$ is a **linear model**: the estimated regression is simply a linear combination of the p input features, possibly including a constant term b_0:

$$\hat{y}_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_p x_{ip} = \mathbf{b}^\top \mathbf{x}_i$$

Linear Regression

- Regression: to model $f()$, called the **regression function**, as the conditional expectation of $y \mid \mathbf{x}$:

$$\mathbb{E}(y \mid \mathbf{x}) = \hat{f}(\mathbf{x})$$

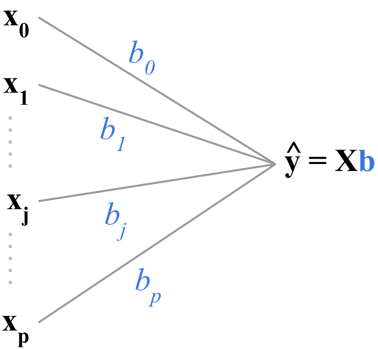
- The regression framework says nothing about what the target function $f()$ should look like. We can decide any form for $\hat{f}()$.
- A standard form of $\hat{f}()$ is a **linear model**: the estimated regression is simply a linear combination of the p input features, possibly including a constant term b_0:

$$\hat{y}_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_p x_{ip} = \mathbf{b}^\top \mathbf{x}_i$$

- In matrix-vector notation, the vector of predictions is:

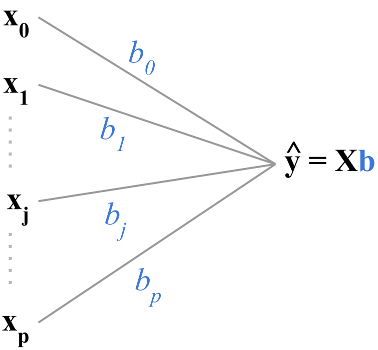
$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{b}$$

Expanding the Regression Horizon



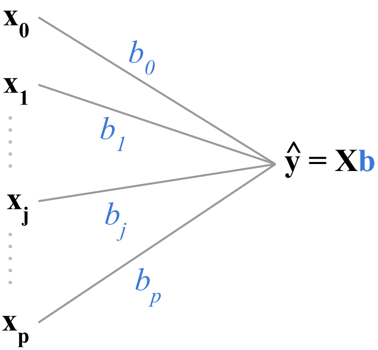
- How can we extend a linear regression model?

Expanding the Regression Horizon



- How can we extend a linear regression model?
- What do we exactly mean by "linear" in linear regression models?

Expanding the Regression Horizon



- How can we extend a linear regression model?
- What do we exactly mean by "linear" in linear regression models?
- What are "parametric" and "non-parametric" models?

Linear Regression - Linearity

- The standard linear regression model:

$$f(\mathbf{x}_i) = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_p x_{ip} + \epsilon_i$$

  is linear in two modes:

Linear Regression - Linearity

- The standard linear regression model:

$$f(\mathbf{x}_i) = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_p x_{ip} + \epsilon_i$$

  is linear in two modes:
  - It is linear in the input variables $X_1, X_2, \ldots, X_p$.

Linear Regression - Linearity

- The standard linear regression model:

$$f(\mathbf{x}_i) = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_p x_{ip} + \epsilon_i$$

  is linear in two modes:
  - It is linear in the input variables $X_1, X_2, \ldots, X_p$.
  - It is also linear in the parameters $b_0, b_1, b_2, \ldots, b_p$.

Linear Regression - Linearity

- The standard linear regression model:

$$f(\mathbf{x}_i) = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_p x_{ip} + \epsilon_i$$

  is linear in two modes:
  - It is linear in the input variables $X_1, X_2, \ldots, X_p$.
  - It is also linear in the parameters $b_0, b_1, b_2, \ldots, b_p$.
- In the regression world, we mostly concern about the linearity of the parameters (i.e., the regression coefficients) $b_0, b_1, b_2, \ldots, b_p$. For example:

$$f(\mathbf{x}_i) = b_0 + b_1 x_{i1} + b_2 x_{i2} + b_3 x_{i1}^2 + b_4 x_{i2}^2 + \epsilon_i$$

Linear Regression - Linearity

- The standard linear regression model:

$$f(\mathbf{x}_i) = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_p x_{ip} + \epsilon_i$$

  is linear in two modes:
  - It is linear in the input variables $X_1, X_2, \ldots, X_p$.
  - It is also linear in the parameters $b_0, b_1, b_2, \ldots, b_p$.
- In the regression world, we mostly concern about the linearity of the parameters (i.e., the regression coefficients) $b_0, b_1, b_2, \ldots, b_p$. For example:

$$f(\mathbf{x}_i) = b_0 + b_1 x_{i1} + b_2 x_{i2} + b_3 x_{i1}^2 + b_4 x_{i2}^2 + \epsilon_i$$

- An example model that is non-linear in both the predictors (input variables) and the parameters:

$$f(\mathbf{x}_i) = b_0 + \exp(x_{i1}^{b_1}) + \sqrt{b_2 x_{i2}} + \epsilon_i$$

Linear Regression - Parametric and Nonparametric

- In parametric models, the functional form of $\hat{f}()$ is fully described by a finite set of parameters, like in the standard linear model:

$$\hat{f}(\mathbf{x}_i) = b_0 + b_1 x_{i1} + \ldots + b_p x_{ip} + \epsilon_i$$

Linear Regression - Parametric and Nonparametric

- In parametric models, the functional form of $\hat{f}()$ is fully described by a finite set of parameters, like in the standard linear model:

$$\hat{f}(\mathbf{x}_i) = b_0 + b_1 x_{i1} + \ldots + b_p x_{ip} + \epsilon_i$$

- In non-parametric models, we can assume a more relaxed way to specify a function $\hat{f}()$ without directly imposing a known functional form. Non-parametric models can still have **hyperparameters**.

Linear Regression - Parametric and Nonparametric

- In parametric models, the functional form of $\hat{f}()$ is fully described by a finite set of parameters, like in the standard linear model:

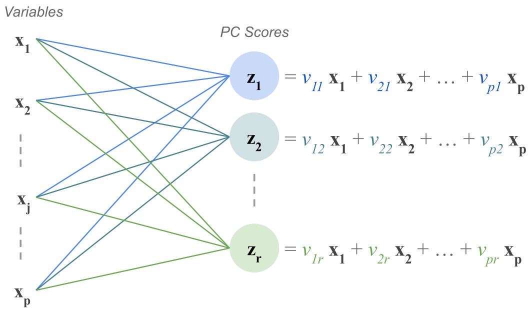
$$\hat{f}(\mathbf{x}_i) = b_0 + b_1 x_{i1} + \ldots + b_p x_{ip} + \epsilon_i$$

- In non-parametric models, we can assume a more relaxed way to specify a function $\hat{f}()$ without directly imposing a known functional form. Non-parametric models can still have **hyperparameters**.
- An example non-parametric method is the K-nearest-neighbors (KNN). We use an average of the response values y_i for the closest k points $\mathbf{x}_i$ to a query $\mathbf{x}_0$:

$$\hat{y}_0 = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\mathbf{x}_0)} y_i$$

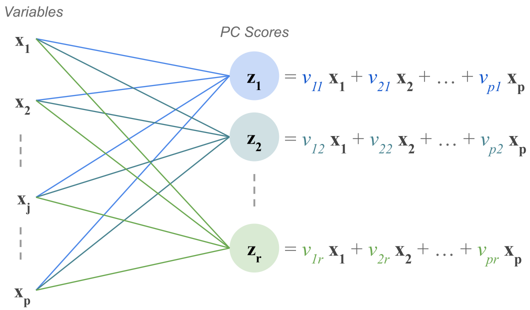
where $\mathcal{N}_k(\mathbf{x}_0)$ indicates the set of k closest neighbors to $\mathbf{x}_0$.

Transforming Features



- Consider a linear model that uses some type of dimension reduction approach: obtaining new variables by using linear combinations of the input features.

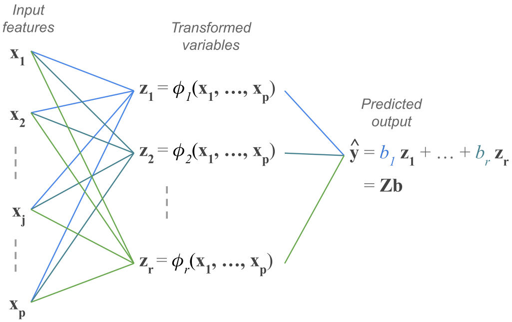
Transforming Features



- Consider a linear model that uses some type of dimension reduction approach: obtaining new variables by using linear combinations of the input features.
- Any component $\mathbf{z}_q$ is a transformation applied on all features:

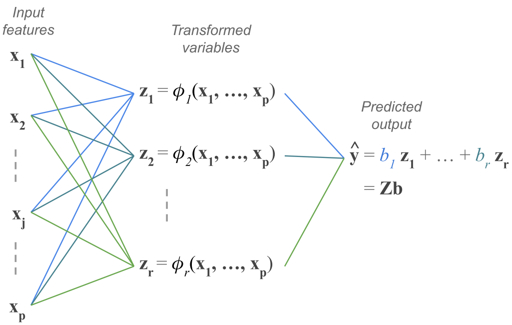
$$\mathbf{z}_q = v_{1q}\mathbf{x}_1 + \ldots + v_{pq}\mathbf{x}_p \longrightarrow Z_q = \phi_q(X_1, \ldots, X_p)$$

Transforming Features



- Assume a function $\phi_q : \mathbb{R}^p \to \mathbb{R}$ that transforms the inputs into a new synthetic feature. In Principle Component Regression (PCR), we consider $\phi_q()$ to be linear functions.

Transforming Features



- Assume a function $\phi_q : \mathbb{R}^p \rightarrow \mathbb{R}$ that transforms the inputs into a new synthetic feature. In Principle Component Regression (PCR), we consider $\phi_q()$ to be linear functions.
- In PCR, with the matrix $\mathbf{Z}$ containing the transformed features, we can still use the OLS to obtain the predicted response as:

$$\hat{\mathbf{y}} = \mathbf{Z}(\mathbf{Z}^\top\mathbf{Z})^{-1}\mathbf{Z}^\top\mathbf{y}$$

Basis Expansion - Basis Functions

- We have a dataset consisting of n data points:
  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$.

Basis Expansion - Basis Functions

- We have a dataset consisting of n data points:
  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$.
- In regression, we assume an unknown target function $f : \mathcal{X} \to \mathbb{R}$.

Basis Expansion - Basis Functions

- We have a dataset consisting of n data points: $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$.
- In regression, we assume an unknown target function $f : \mathcal{X} \to \mathbb{R}$.
  - In the simplest case, we have only one input feature: $\mathcal{X} = \mathbb{R}$

Basis Expansion - Basis Functions

- We have a dataset consisting of n data points:
  $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$.
- In regression, we assume an unknown target function $f : \mathcal{X} \to \mathbb{R}$.
  - In the simplest case, we have only one input feature: $\mathcal{X} = \mathbb{R}$
  - In the multi-dimensional case, there are p input variables: $\mathcal{X} = \mathbb{R}^p$.

Basis Expansion - Basis Functions

- We have a dataset consisting of n data points: $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$.
- In regression, we assume an unknown target function $f : \mathcal{X} \to \mathbb{R}$.
  - In the simplest case, we have only one input feature: $\mathcal{X} = \mathbb{R}$
  - In the multi-dimensional case, there are p input variables: $\mathcal{X} = \mathbb{R}^p$.
- We can look for $\hat{f}()$ in a finite dimensional space of functions spanned a given basis. We specify a set of functions $\phi_0, \phi_1, \ldots, \phi_m$ from $\mathcal{X}$ to $\mathbb{R}$, and estimate f in the form of a linear combination:

$$\hat{f}(\mathbf{x}) = \sum_{q=0}^{m} b_q \phi_q(\mathbf{x})$$

Basis Expansion - Basis Functions

- We have a dataset consisting of n data points: $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \mathbb{R}$.
- In regression, we assume an unknown target function $f : \mathcal{X} \to \mathbb{R}$.
  - In the simplest case, we have only one input feature: $\mathcal{X} = \mathbb{R}$
  - In the multi-dimensional case, there are p input variables: $\mathcal{X} = \mathbb{R}^p$.
- We can look for $\hat{f}()$ in a finite dimensional space of functions spanned a given basis. We specify a set of functions $\phi_0, \phi_1, \ldots, \phi_m$ from $\mathcal{X}$ to $\mathbb{R}$, and estimate f in the form of a linear combination:

$$\hat{f}(\mathbf{x}) = \sum_{q=0}^{m} b_q \phi_q(\mathbf{x})$$

- Perform the regression is reduced to finding the parameters $b_0, b_1, \ldots, b_m$.

Basis Expansion - Linear Regression

- In the one dimensional case, we can use $\phi_0(x) = 1$ and $\phi_1(x) = x$. This gives the simple linear regression model:

$$\hat{f}(x) = b_0\phi_0(x) + b_1\phi_1(x) = b_0 + b_1 x$$

Basis Expansion - Linear Regression

- In the one dimensional case, we can use $\phi_0(x) = 1$ and $\phi_1(x) = x$. This gives the simple linear regression model:

$$\hat{f}(x) = b_0\phi_0(x) + b_1\phi_1(x) = b_0 + b_1 x$$

- In the multi-dimensional case, we take $\phi_1(\mathbf{x}) = [\mathbf{x}]_1, \phi_2(\mathbf{x}) = [\mathbf{x}]_2, \ldots, \phi_p(\mathbf{x}) = [\mathbf{x}]_p$. Here, $[\mathbf{x}]_k$ denotes the k-th element of the input vector $\mathbf{x} \in \mathcal{X}$.

$$\begin{aligned}\hat{f}(\mathbf{x}_i) &= b_0\phi_0(\mathbf{x}_i) + b_1\phi_1(\mathbf{x}_i) + \ldots + b_p\phi_p(\mathbf{x}_i)\\ &= b_0 + b_1[\mathbf{x}_i]1 + \ldots + b_p[\mathbf{x}_i]p\\ &= b_0 + b_1 x_{i1} + \ldots + b_p x_{ip}\end{aligned}$$

Basis Expansion - Polynomial Regression

- In the one-dimensional case, we can choose $\phi_q(x) = x^q$ for $q = 1, 2, \ldots, m$.

Basis Expansion - Polynomial Regression

- In the one-dimensional case, we can choose $\phi_q(x) = x^q$ for $q = 1, 2, \ldots, m$.
- This allows us to fit f from the class of polynomial functions of degree at most m:

$$\begin{aligned}
\hat{f}(x) &= \sum_{q=0}^{m} b_q \phi_q(x) \\
&= b_0\phi_0(x) + b_1\phi_1(x) + b_2\phi_2(x) + \ldots + b_m\phi_m(x) \\
&= b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m
\end{aligned}$$

Basis Expansion - Polynomial Regression

- In the one-dimensional case, we can choose $\phi_q(x) = x^q$ for $q = 1, 2, \ldots, m$.
- This allows us to fit f from the class of polynomial functions of degree at most m:

$$\begin{aligned}\hat{f}(x) &= \sum_{q=0}^{m} b_q \phi_q(x) \\ &= b_0\phi_0(x) + b_1\phi_1(x) + b_2\phi_2(x) + \ldots + b_m\phi_m(x) \\ &= b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m\end{aligned}$$

- In the multi-dimensional case, for example $p = 2$ input features X_1 and X_2 and a polynomial of degree $m = 2$:

$$\hat{f}(X_1, X_2) = b_0 + b_1 X_1 + b_2 X_2 + b_3 X_1 X_2 + b_4 X_1^2 + b_5 X_2^2$$

Basis Expansion - Polynomial Regression

- We can index the parameters $\mathbf{b}_k$ with a multi-index $q = (q_1, q_2)$ with $q_1 + q_2 \leq m$. For example:

$$\hat{f}(X_1, X_2) = b_0 + b_{(1,0)} X_1 + b_{(0,1)} X_2 + b_{(1,1)} X_1 X_2 + b_{(2,0)} X_1^2 + b_{(0,2)} X_2^2$$

Basis Expansion - Polynomial Regression

- We can index the parameters $\mathbf{b}_k$ with a multi-index $q = (q_1, q_2)$ with $q_1 + q_2 \leq m$. For example:

$$\hat{f}(X_1, X_2) = b_0 + b_{(1,0)} X_1 + b_{(0,1)} X_2 + b_{(1,1)} X_1 X_2 + b_{(2,0)} X_1^2 + b_{(0,2)} X_2^2$$

- A model can be compactly expressed as:

$$\hat{f}(\mathbf{x}) = \sum_{(q_1, q_2)} b_q \phi_q(\mathbf{x})$$

Non-parametric Regression - Conditional Averages

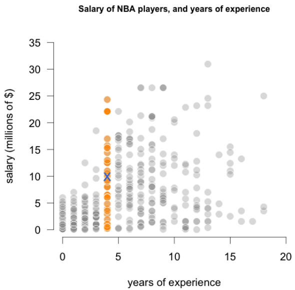
- Consider NBA players in the 2018 season. The response Y is the salary and the predictor X is the number of years of experience.

Non-parametric Regression - Conditional Averages

- Consider NBA players in the 2018 season. The response Y is the salary and the predictor X is the number of years of experience.
- If we predict the salary of a player with 4 years of experience, one approach is using a conditional mean:

$$\texttt{Predicted Salary} = Avg(\texttt{Salary} \mid \texttt{Experience} = 4 \text{ years})$$

Non-parametric Regression - Conditional Averages



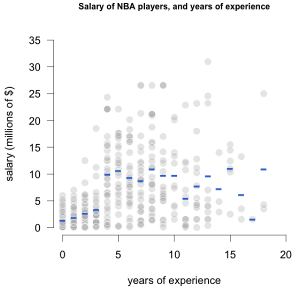
- We look at the salaries y_i of all players having 4 years of experience $x_i = 4$, and take the average of those values.

$$\hat{y}_0 = Avg(y_i \mid x_i = 4)$$

- Regression value is computed in terms of conditional expectation:

$$\hat{f}(x_0) = \mathbb{E}(y \mid X = x_0) = \hat{y}_0$$

Non-parametric Regression - Conditional Averages

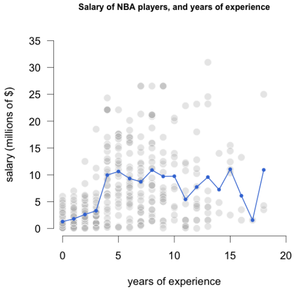


- Regression value is computed in terms of conditional expectation:

$$\hat{f}(x_0) = \mathbb{E}(y \mid X = x_0) = \hat{y}_0$$

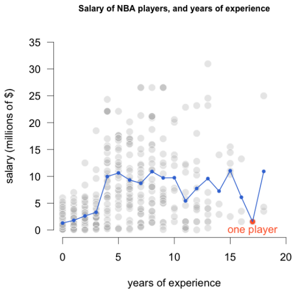
- We compute all the average salaries for each value of years of experience.

Non-parametric Regression - Conditional Averages



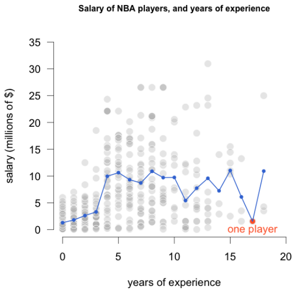
- We compute all the average salaries for each value of years of experience.
- We connect the dots of average salaries to get a **non-parametric regression line**.
- A disadvantage is that we don't have regression coefficients to interpret the model.

Non-parametric Regression - Conditional Averages



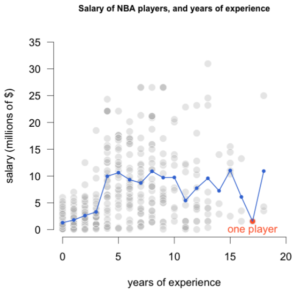
- A disadvantage is that we don't have regression coefficients to interpret the model.
- We might have certain X-values for which there's scarcity of data.
- With just a few data points, the predicted salary will be highly unreliable.

Non-parametric Regression - Looking at the Neighbors



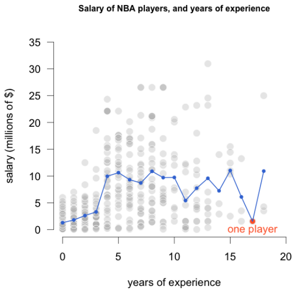
- When there's a scarcity of data, we can use neighboring data points to improve the prediction.

Non-parametric Regression - Looking at the Neighbors



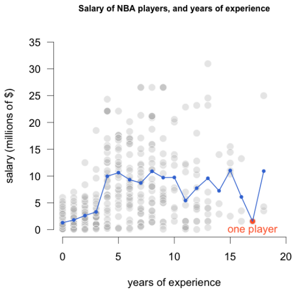
- When there's a scarcity of data, we can use neighboring data points to improve the prediction.
- Some common options to choose from:

Non-parametric Regression - Looking at the Neighbors



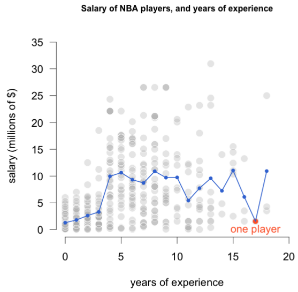
- When there's a scarcity of data, we can use neighboring data points to improve the prediction.
- Some common options to choose from:
  - Compute the arithmetic mean of the neighbors y_i's.

Non-parametric Regression - Looking at the Neighbors



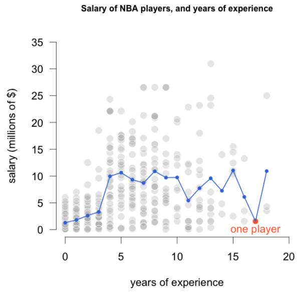
- When there's a scarcity of data, we can use neighboring data points to improve the prediction.
- Some common options to choose from:
  - Compute the arithmetic mean of the neighbors y_i's.
  - Compute a weighted average of the neighbors y_i's.

Non-parametric Regression - Looking at the Neighbors



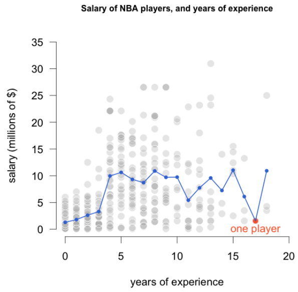
- When there's a scarcity of data, we can use neighboring data points to improve the prediction.
- Some common options to choose from:
  - Compute the arithmetic mean of the neighbors y_i's.
  - Compute a weighted average of the neighbors y_i's.
  - Compute a linear fit with the neighboring points (x_i, y_i)'s.

Non-parametric Regression - Looking at the Neighbors



- When there's a scarcity of data, we can use neighboring data points to improve the prediction.
- Some common options to choose from:
  - Compute the arithmetic mean of the neighbors y_i's.
  - Compute a weighted average of the neighbors y_i's.
  - Compute a linear fit with the neighboring points (x_i, y_i)'s.
  - Compute a polynomial fit with the neighboring points (x_i, y_i)'s.

Non-parametric Regression - Looking at the Neighbors



- **Neighborhood**: How to define a neighborhood of neighboring points x_i for a given query point x_0.
- **Local Fitting Mechanism**: How to define a local fit with the y_i's (of the neighboring points) to predict the outcome $\hat{y}_0$.

Nearest Neighbor Estimates - k Nearest Neighbors (KNN)

- In KNN, we define a neighborhood $\mathcal{N}_0 = \mathcal{N}_k(\mathbf{x}_0)$ that consists of k closest points around a query point $\mathbf{x}_0$.

Nearest Neighbor Estimates - k Nearest Neighbors (KNN)

- In KNN, we define a neighborhood $\mathcal{N}_0 = \mathcal{N}_k(\mathbf{x}_0)$ that consists of k closest points around a query point $\mathbf{x}_0$.
- These k neighboring points $(\mathbf{x}_i, y_i)$'s will be used to compute $\hat{y}_0$ with some local averaging mechanism.

Nearest Neighbor Estimates - k Nearest Neighbors (KNN)

- In KNN, we define a neighborhood $\mathcal{N}_0 = \mathcal{N}_k(\mathbf{x}_0)$ that consists of k closest points around a query point $\mathbf{x}_0$.
- These k neighboring points $(\mathbf{x}_i, y_i)$'s will be used to compute $\hat{y}_0$ with some local averaging mechanism.
- The vanilla KNN uses the arithmetic mean of the k closest points:

$$\texttt{running mean:}\ \hat{f}(\mathbf{x}_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} y_i = \hat{y}_0$$

Nearest Neighbor Estimates - k Nearest Neighbors (KNN)

- In KNN, we define a neighborhood $\mathcal{N}_0 = \mathcal{N}_k(\mathbf{x}_0)$ that consists of k closest points around a query point $\mathbf{x}_0$.
- These k neighboring points $(\mathbf{x}_i, y_i)$'s will be used to compute $\hat{y}_0$ with some local averaging mechanism.
- The vanilla KNN uses the arithmetic mean of the k closest points:

$$\texttt{running mean: } \hat{f}(\mathbf{x}_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} y_i = \hat{y}_0$$

- We can also use the median of the y_i values:

$$\texttt{running median: } \hat{f}(\mathbf{x}_0) = \texttt{median}(y_i \mid \mathbf{x}_i \in \mathcal{N}_0)$$

Nearest Neighbor Estimates - k Nearest Neighbors (KNN)

- In KNN, we define a neighborhood $\mathcal{N}_0 = \mathcal{N}_k(\mathbf{x}_0)$ that consists of k closest points around a query point $\mathbf{x}_0$.
- These k neighboring points $(\mathbf{x}_i, y_i)$'s will be used to compute $\hat{y}_0$ with some local averaging mechanism.
- The vanilla KNN uses the arithmetic mean of the k closest points:

$$\texttt{running mean: } \hat{f}(\mathbf{x}_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} y_i = \hat{y}_0$$

- We can also use the median of the y_i values:

$$\texttt{running median: } \hat{f}(\mathbf{x}_0) = \texttt{median}(y_i \mid \mathbf{x}_i \in \mathcal{N}_0)$$

- We can also use a local linear fit:

$$\texttt{running line: } \hat{f}(\mathbf{x}_0) = b_{0,x_0} + b_{1,x_0} x_0$$

where b_{0,x_0} and b_{1,x_0} are the least squares estimate using data points (x_i, y_i) with $i \in \mathcal{N}_0$ (one-dimensional case).

Nearest Neighbor Estimates - Distance Measure

- Common choices of distances measure are:
- Euclidean:

$$d(\mathbf{x}_0, \mathbf{x}_i) = \sqrt{\sum_{j=1}^{p} (x_{0j} - x_{ij})^2}$$

- Manhattan:

$$d(\mathbf{x}_0, \mathbf{x}_i) = \sum_{j=1}^{p} |x_{0j} - x_{ij}|$$

k Nearest Neighbors (KNN) - How to find k?

- We use the K-fold cross validation approach.

k Nearest Neighbors (KNN) - How to find k?

- We use the K-fold cross validation approach.
- To avoid the confusion between K and k, we temporarily name the approach Q-fold cross validation.

k Nearest Neighbors (KNN) - How to find k?

- We use the K-fold cross validation approach.
- To avoid the confusion between K and k, we temporarily name the approach Q-fold cross validation.
- We split our n data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ into Q folds:

$$\mathcal{D}_{train} = \mathcal{D}_{fold-1} \cup \mathcal{D}_{fold-2} \cup \ldots \cup \mathcal{D}_{fold-Q}$$

k Nearest Neighbors (KNN) - How to find k?

- We use the K-fold cross validation approach.
- To avoid the confusion between K and k, we temporarily name the approach Q-fold cross validation.
- We split our n data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ into Q folds:

$$\mathcal{D}_{train} = \mathcal{D}_{fold-1} \cup \mathcal{D}_{fold-2} \cup \ldots \cup \mathcal{D}_{fold-Q}$$

- Each fold set $\mathcal{D}_{fold-q}$ is an evaluation set $\mathcal{D}_{eval-q}$. We then have Q training sets:

k Nearest Neighbors (KNN) - How to find k?

- We use the K-fold cross validation approach.
- To avoid the confusion between K and k, we temporarily name the approach Q-fold cross validation.
- We split our n data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ into Q folds:

$$\mathcal{D}_{train} = \mathcal{D}_{fold-1} \cup \mathcal{D}_{fold-2} \cup \ldots \cup \mathcal{D}_{fold-Q}$$

- Each fold set $\mathcal{D}_{fold-q}$ is an evaluation set $\mathcal{D}_{eval-q}$. We then have Q training sets:
    - $\mathcal{D}_{train-1} = \mathcal{D}_{train} \Big\backslash \mathcal{D}_{fold-1}$

k Nearest Neighbors (KNN) - How to find k?

- We use the K-fold cross validation approach.
- To avoid the confusion between K and k, we temporarily name the approach Q-fold cross validation.
- We split our n data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ into Q folds:

$$\mathcal{D}_{train} = \mathcal{D}_{fold-1} \cup \mathcal{D}_{fold-2} \cup \ldots \cup \mathcal{D}_{fold-Q}$$

- Each fold set $\mathcal{D}_{fold-q}$ is an evaluation set $\mathcal{D}_{eval-q}$. We then have Q training sets:
    - $\mathcal{D}_{train-1} = \mathcal{D}_{train} \Big\backslash \mathcal{D}_{fold-1}$
    - $\mathcal{D}_{train-2} = \mathcal{D}_{train} \Big\backslash \mathcal{D}_{fold-2}$

k Nearest Neighbors (KNN) - How to find k?

- We use the K-fold cross validation approach.
- To avoid the confusion between K and k, we temporarily name the approach Q-fold cross validation.
- We split our n data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ into Q folds:

$$\mathcal{D}_{train} = \mathcal{D}_{fold-1} \cup \mathcal{D}_{fold-2} \cup \ldots \cup \mathcal{D}_{fold-Q}$$

- Each fold set $\mathcal{D}_{fold-q}$ is an evaluation set $\mathcal{D}_{eval-q}$. We then have Q training sets:
    - $\mathcal{D}_{train-1} = \mathcal{D}_{train} \Big\backslash \mathcal{D}_{fold-1}$
    - $\mathcal{D}_{train-2} = \mathcal{D}_{train} \Big\backslash \mathcal{D}_{fold-2}$
    - $\ldots$

k Nearest Neighbors (KNN) - How to find k?

- We use the K-fold cross validation approach.
- To avoid the confusion between K and k, we temporarily name the approach Q-fold cross validation.
- We split our n data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ into Q folds:

$$\mathcal{D}_{train} = \mathcal{D}_{fold-1} \cup \mathcal{D}_{fold-2} \cup \ldots \cup \mathcal{D}_{fold-Q}$$

- Each fold set $\mathcal{D}_{fold-q}$ is an evaluation set $\mathcal{D}_{eval-q}$. We then have Q training sets:
    - $\mathcal{D}_{train-1} = \mathcal{D}_{train} \Big\backslash \mathcal{D}_{fold-1}$
    - $\mathcal{D}_{train-2} = \mathcal{D}_{train} \Big\backslash \mathcal{D}_{fold-2}$
    - $\ldots$
    - $\mathcal{D}_{train-Q} = \mathcal{D}_{train} \Big\backslash \mathcal{D}_{fold-Q}$

Cross Validation

1. For $k = 1, 3, \ldots, B$ (number of neighbors)

Cross Validation

1. For $k = 1, 3, \ldots, B$ (number of neighbors)
    - For $q = 1, \ldots, Q$ (fold index number):

Cross Validation

1. For $k = 1, 3, \ldots, B$ (number of neighbors)
   - For $q = 1, \ldots, Q$ (fold index number):
     - Fit $h_{k,q}$ KNN with k_b-neighbors on $\mathcal{D}_{train-q}$

Cross Validation

1. For $k = 1, 3, \ldots, B$ (number of neighbors)
   - For $q = 1, \ldots, Q$ (fold index number):
     - Fit $h_{k,q}$ KNN with k_b-neighbors on $\mathcal{D}_{train-q}$
     - Compute and store $E_{eval-q}(h_{k,q})$ using $\mathcal{D}_{eval-q}$

Cross Validation

1. For $k = 1, 3, \ldots, B$ (number of neighbors)
   - For $q = 1, \ldots, Q$ (fold index number):
     - Fit $h_{k,q}$ KNN with k_b-neighbors on $\mathcal{D}_{train-q}$
     - Compute and store $E_{eval-q}(h_{k,q})$ using $\mathcal{D}_{eval-q}$
   - Compute and store $E_{cv_k} = \frac{1}{Q} \sum_q E_{eval-q}(h_{k,q})$

Cross Validation

1. For $k = 1, 3, \ldots, B$ (number of neighbors)
   - For $q = 1, \ldots, Q$ (fold index number):
     - Fit $h_{k,q}$ KNN with k_b-neighbors on $\mathcal{D}_{train-q}$
     - Compute and store $E_{eval-q}(h_{k,q})$ using $\mathcal{D}_{eval-q}$
   - Compute and store $E_{cv_k} = \frac{1}{Q} \sum_q E_{eval-q}(h_{k,q})$
2. Compare all cross-validation errors $E_{cv_1}, E_{cv_3}, \ldots, E_{cv_B}$ and choose the smallest of them $E_{cv_{k^*}}$.

Cross Validation

1. For $k = 1, 3, \ldots, B$ (number of neighbors)
    - For $q = 1, \ldots, Q$ (fold index number):
        - Fit $h_{k,q}$ KNN with k_b-neighbors on $\mathcal{D}_{train-q}$
        - Compute and store $E_{eval-q}(h_{k,q})$ using $\mathcal{D}_{eval-q}$
    - Compute and store $E_{cv_k} = \frac{1}{Q}\sum_q E_{eval-q}(h_{k,q})$
2. Compare all cross-validation errors $E_{cv_1}, E_{cv_3}, \ldots, E_{cv_B}$ and choose the smallest of them $E_{cv_{k^*}}$.
3. Use k^* to fit the final KNN model.