Supervised Learning (Part I)

SYS 6018 | Spring 2023

supervised_1.pdf

Contents

1 Supervised Learning Intro

1.1 Survey

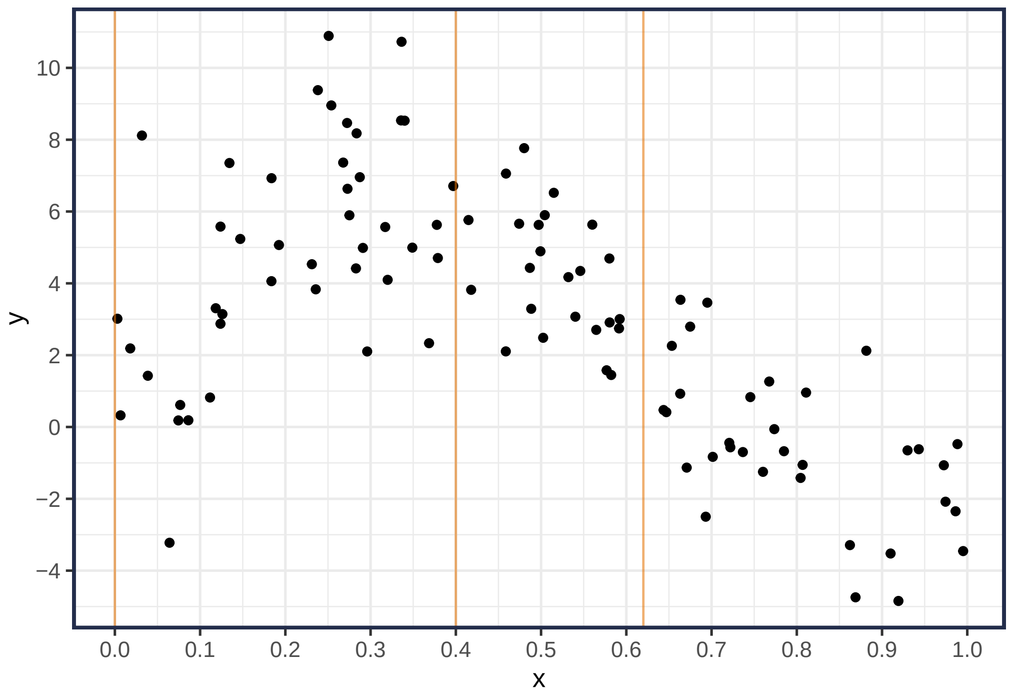
There will be a survey at end of lecture. You can access the survey from MS Teams. At the top tab of the class find "Fill | Supervised Learning I : What is the best model?". You may need to expand "More".

1.2 Supervised Learning

- In *supervised learning*, each observation can be partitioned into two sets: the predictor variables and the outcome variable(s).
  - Predictor variables are sometimes called independent/feature variables
  - Outcome variables are sometimes called target/labels/response/dependent variables.
- Usually the predictor variables are represented by X and the response variables represented by Y
- The goal in supervised learning is to find the patterns and relationships between the predictors, X, and the response, Y.
  - Usually the goal is to *predict* the value of Y given X.
- Later in the course we will explore the *unsupervised learning* topics of association analysis, network analysis, density estimation, clustering, and anomaly detection which do not have any outcomes (i.e., no Y's).

2 Example Data

Consider some data $D = \{(X_i, Y_i)\}_{i=1}^n$ with $Y_i \in \mathbb{R}$, $X_i \in [0, 1]$ and $n = 100$.



Your Turn #1

The goal is to predict new Y values if we are given the X's.

- If $x = .40$, predict Y.
- If $x = 0$, predict Y.
- If $x = .62$, predict Y.
- How should we build a *model* that will automatically predict Y for any given X?

3 Linear Models

- Linear models refer to a class of models where the output (predicted value) is a linear combination (weighted sum) of the input variables

$$f(x;\beta) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

  where $x = [x_1, \ldots, x_p]^\mathsf{T}$ is a vector of features/variables/attributes and $\hat{Y}|x = f(x;\hat{\beta})$ is the predicted response at $X = x$

- the coefficients (or weights), $\hat{\beta}$ are often selected by minimizing the squared residuals of the *training data* (may also be described as *ordinary least squares*)
  - But, there are other, and better, ways to estimate the parameters in linear regression that we will discuss later in the course. (e.g., Lasso, Ridge, Robust)

3.1 Simple Linear Regression

- single predictor variable $x \in \mathbb{R}$
- $f(x;\beta) = \beta_0 + \beta_1 x$
- Use *training data*: $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{n}$
- OLS uses the weights/coefficients that minimize the RSS loss function over the training data

$$\hat{\beta} = \underset{\beta}{\arg\min}\ \text{SSE}(\beta)$$

- where SSE is the *sum of squared errors* (also known as *residual sum of squares* (RSS))

$$\begin{aligned}\text{SSE}(\beta) &= \sum_{i}^{n} (y_i - f(x_i,\beta))^2 \\ &= \sum_{i}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \\ &= \sum_{i}^{n} \hat{\epsilon}_i^2 \qquad \text{where } \hat{\epsilon}_i = y_i - \hat{y}_i \text{ is the residual}\end{aligned}$$

- The solutions are

$$\begin{aligned}\hat{\beta}_0 &= \bar{y} - \beta_1 \bar{x} \\ \hat{\beta}_1 &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\end{aligned}$$

- Definitions:

$$\begin{aligned}\text{MSE}(\beta) &= \frac{1}{n}\text{SSE}(\beta) \\ &= \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i;\beta))^2 \\ \text{RMSE} &= \sqrt{\text{MSE}} = \sqrt{\text{SSE}}/\sqrt{n}\end{aligned}$$

3.2 OLS Linear Models in R

3.2.1 Estimation with `lm()`

In **R**, the function `lm()` fits an OLS linear model

```
data_train = tibble(x,y)       # create a data frame/tibble
m1 = lm(y~x, data=data_train) # fit simple OLS
```

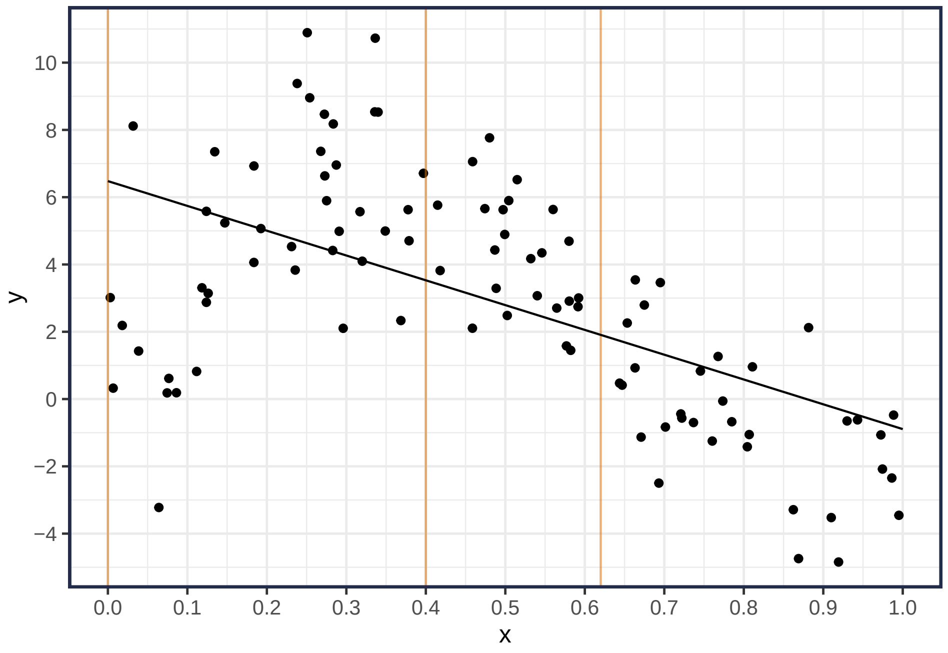
```
summary(m1)                    # summary of model
#>
#> Call:
#> lm(formula = y ~ x, data = data_train)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -9.229 -1.635  0.019  1.940  6.728
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)    6.478      0.584   11.09  < 2e-16 ***
#> x             -7.372      1.058   -6.97  3.7e-10 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.91 on 98 degrees of freedom
#> Multiple R-squared:  0.331,  Adjusted R-squared:  0.325
#> F-statistic: 48.6 on 1 and 98 DF,  p-value: 3.69e-10
broom::tidy(m1)                # model coefficients (as a data frame)
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic  p.value
#>   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)     6.48     0.584      11.1  5.39e-19
#> 2 x              -7.37     1.06      -6.97 3.69e-10
broom::glance(m1)              # model properties
#> # A tibble: 1 x 12
#>   r.squared adj.r.squa~1 sigma stati~2  p.value    df logLik   AIC   BIC devia~3
#>       <dbl>        <dbl> <dbl>   <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>   <dbl>
#> 1     0.331        0.325  2.91    48.6 3.69e-10     1  -248.  501.  509.    829.
#> # ... with 2 more variables: df.residual <int>, nobs <int>, and abbreviated
#> #   variable names 1: adj.r.squared, 2: statistic, 3: deviance
```

- `lm()` uses the `formula` interface, which includes the intercept by default.
  - Some examples of using formulas as well as getting the underlying X (model/design matrix) can be found here

3.2.2 Prediction with `predict()`

The function `predict()` is used to get the predicted values.

```
xseq = seq(0, 1, length=200)          # sequence of equally spaced values from 0 to 1
xeval = tibble(x = xseq)              # make into a tibble object
yhat1 = predict(m1, xeval)            # vector of yhat's (predictions)
```



3.2.3 Questions

Your Turn #2

1. How did we do? If X_{new} is close to 0, or close to 0.4, or close to .62?
2. How to make it better?

4 Polynomial inputs

- In the *simple* linear regression model, we had 2 parameters that we needed to estimation, β_0 and β_1. Thus, the model complexity is minimal.
  - The only thing simpler is an intercept only model.
- But the data appears to have a more *complex* structure than linear.
- A *parametric approach* to add complexity is to incorporate *polynomial terms* into the model.
  - A quadratic model is $f(x;\beta) = \beta_0 + \beta_1 x + \beta_2 x^2$

4.1 Estimation

- OLS uses the weights/coefficients that minimize the SSE loss function over the training data

$$\begin{aligned}\hat{\beta} &= \arg\min_{\beta}\ \text{SSE}(\beta) \qquad \text{Note: } \beta \text{ in this problem is a } \textit{vector}\\ &= \arg\min_{\beta}\ \sum_{i=1}^{n}(y_i - f(x_i;\beta))^2\\ &= \arg\min_{\beta}\ \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i - \beta_2 x_i^2)^2\end{aligned}$$

4.1.1 Matrix notation

- **Model**

$$f(\mathbf{x};\beta) = \mathbf{x}^{\mathsf{T}}\beta$$

$$\mathbf{x} = \begin{bmatrix}1\\ x\\ x^2\end{bmatrix} \qquad \beta = \begin{bmatrix}\beta_0\\ \beta_1\\ \beta_2\end{bmatrix}$$

Your Turn #3 : Matrix Notation

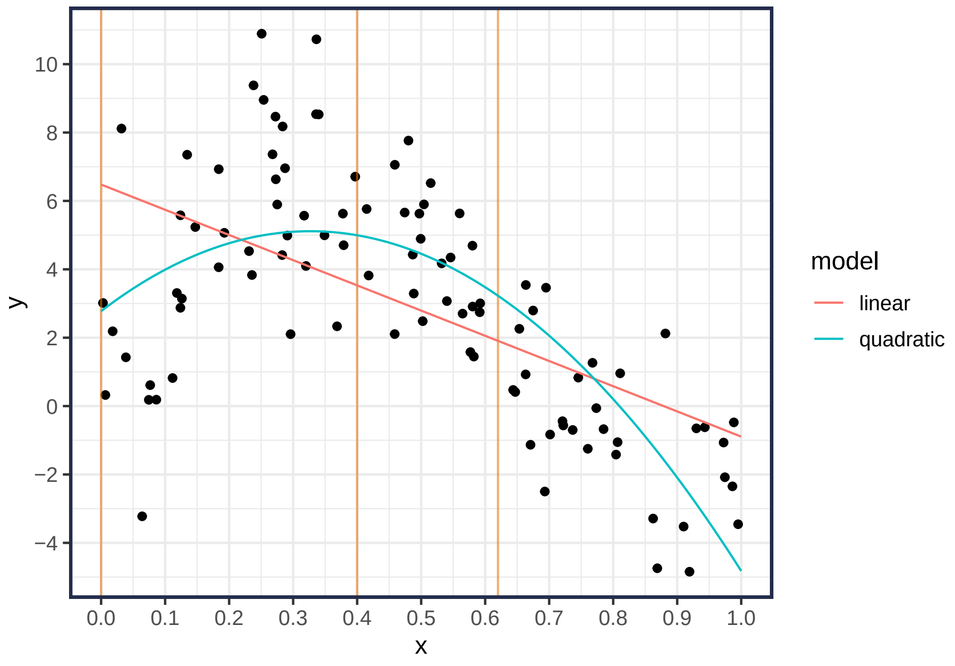
Solve for $\hat{\beta}$ using matrix notation.

$$Y = \begin{bmatrix}Y_1\\ Y_2\\ \vdots\\ Y_n\end{bmatrix} \qquad X = \begin{bmatrix}1 & X_1 & X_1^2\\ 1 & X_2 & X_2^2\\ \vdots & \vdots & \vdots\\ 1 & X_n & X_n^2\end{bmatrix} \qquad \beta = \begin{bmatrix}\beta_0\\ \beta_1\\ \beta_2\end{bmatrix}$$

4.1.2 R implementation

In **R**, the function `poly()` is a convenient way to get polynomial terms

```
m2 = lm(y~poly(x, degree=2), data=data_train)
yhat2 = predict(m2, xeval)
```



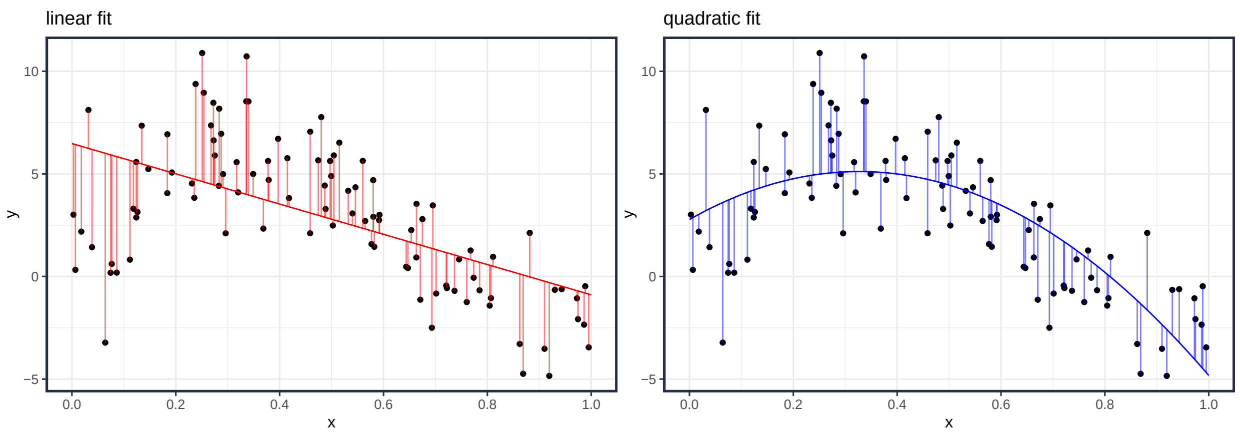
Your Turn #4

1. How did we do? If X_{new} is close to 0, or close to 0.4, or close to $.62$?
2. But does the quadratic model fit better *overall*?
3. What is the *complexity* of the quadratic model?

4.2 Performance Comparison (on Training Data)

Comparing the two models (according to MSE), the quadratic model does much better!

degree	MSE	npars
1	8.29	2
2	5.58	3



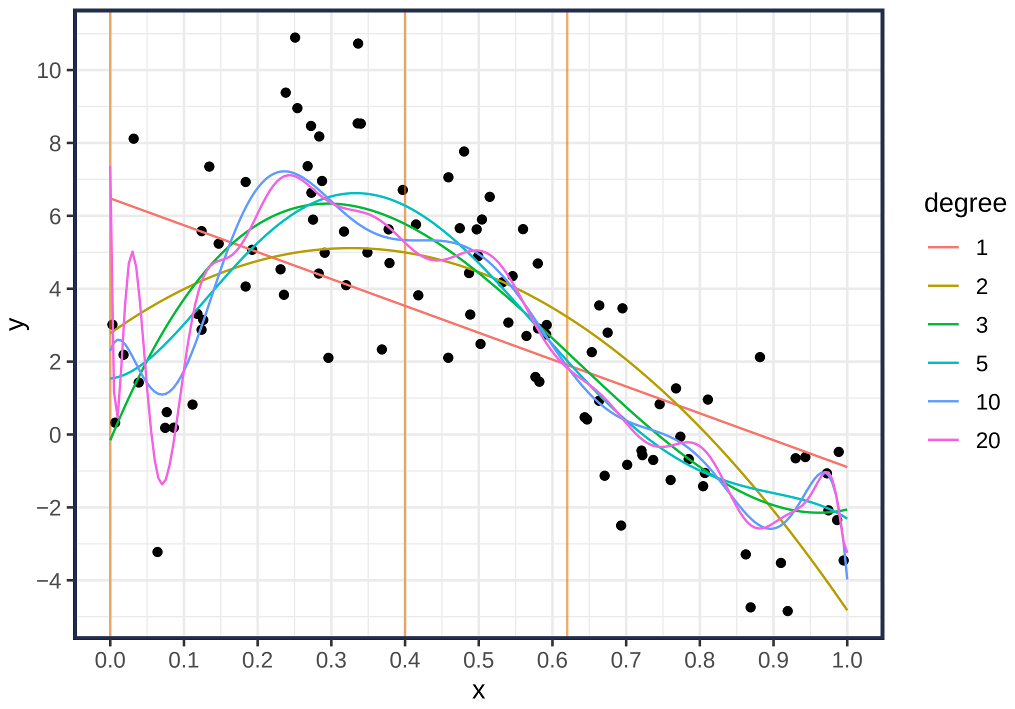
As my kids always reason, "if a little is good, than a lot must be better". So why not try more complex models by increasing the polynomial degree.

- Polynomial of degree d

$$f_{\text{poly}}(x; \beta, d) = \beta_0 + \sum_{j=1}^{d} \beta_j x^j$$

degree	MSE	npars
1	8.29	2
2	5.58	3
3	4.28	4
5	4.10	6
10	3.65	11
20	3.16	21

And its always good to observe the plot



- For `degree=20`, the behavior at the end points are a bit erratic.
- Using a higher degree would further reduce the MSE, but the fitted curve would be more "complex" and may not be as good for new data.

5 k-nearest neighbor models

- The k-NN method is a non-parametric *local* method, meaning that to make a prediction $\hat{y}|x$, it only uses the training data in the *vicinity* of x.
  - contrast with OLS linear regression, which uses all x's to get prediction.
- The model is simple to describe

$$\begin{aligned} f_{\text{knn}}(x;k) &= \frac{1}{k} \sum_{i:x_i \in N_k(x)} y_i \\ &= \text{Avg}(y_i \mid x_i \in N_k(x)) \end{aligned}$$

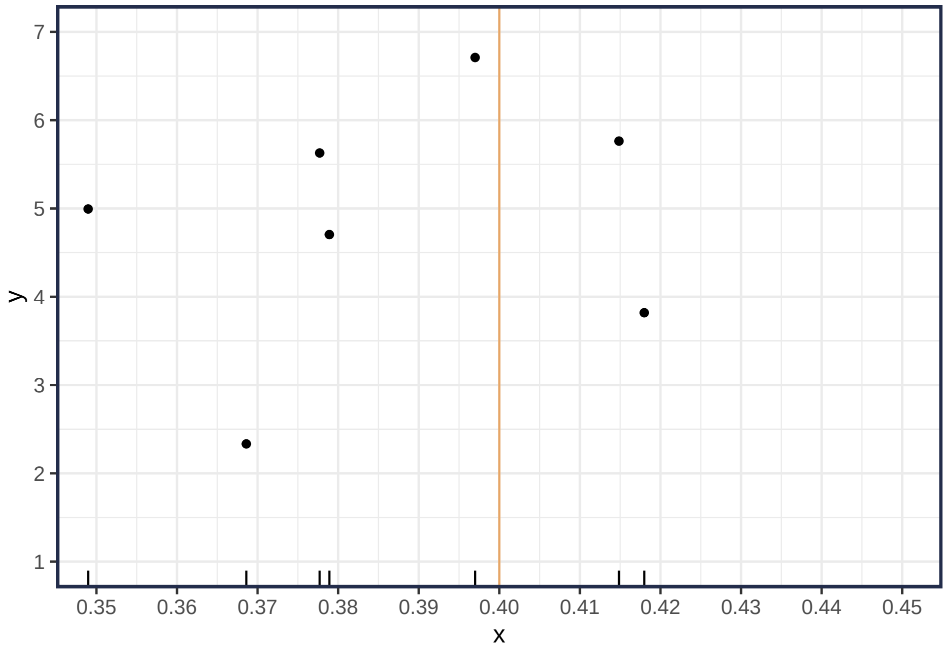
  - $N_k(x)$ are the set of k nearest neighbors to x
  - only the k closest y's are used to generate a prediction
  - it is a *simple mean* of the k nearest observations

Your Turn #5

What is the estimate $f_{\text{knn}}(x; k = n)$?

5.0.1 Example

Let's zoom in on the region around $x = 0.4$



x	y	k	D	$\hat{f}_{\text{knn}}(x;k)$
0.397	6.710	1	0.003	6.710
0.415	5.763	2	0.015	6.237
0.418	3.819	3	0.018	5.431
0.379	4.705	4	0.021	5.249
0.378	5.628	5	0.022	5.325
0.369	2.333	6	0.031	4.826
0.349	4.994	7	0.051	4.850

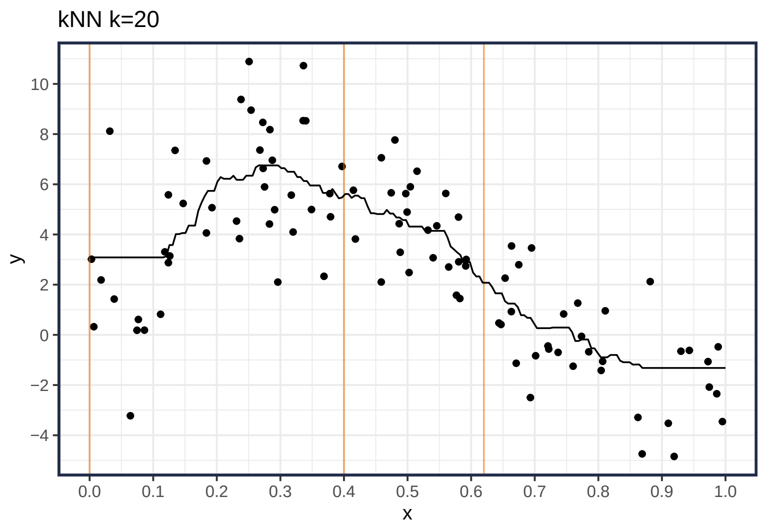
5.0.2 Notes about knn

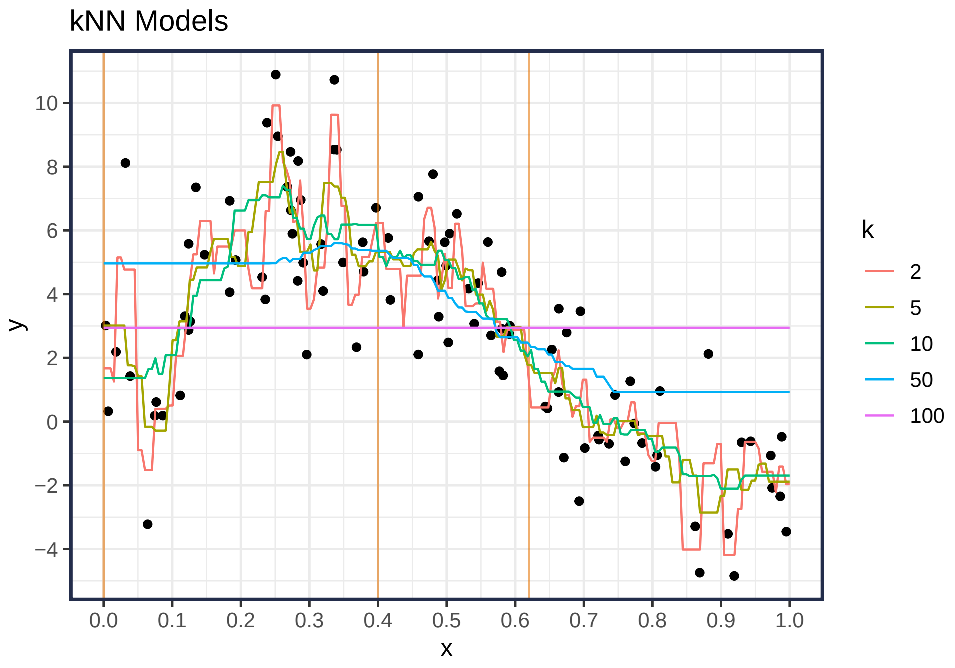
- A suitable *distance* measure (e.g. Euclidean) must be chosen.
  - And predictors are often *scaled* (same sd or range) so one variable doesn't dominate the distance calculation
- Because the distance to neighbors grows exponentially with increased dimensionality/features, the *curse of dimensionality* is often referenced with respect to knn.
  - This means that in high dimensions most *neighbors* are not very close and the method becomes less *local*
- One computational drawback of knn methods is that all the training data must be stored in order to make predictions.
  - For large training data, may need to sample (or use prototypes)
- The *complexity* of a knn model increases as k decreases.
- The least complex model, which is a constant, occurs when $k = n$
- The most complex model when $k = 1$
- The effective degrees of freedom or *edf* for a knn model is n/k
  - this is a measure of the model *complexity*. It is approximately the number of parameters that are estimated in the model (to allow comparison with parametric models)

5.1 knn in action

In **R**, the function `knn.reg()` from the `FNN` package will fit a knn regression model. Here is a $k = 20$ nearest neighbor model

```
library(FNN)                          # library() loads the package. Access to knn.reg()
knn.20 = knn.reg(select(data_train, x), test=xeval, y=data_train$y, k=20)
```





5.1.1 Performance of the knn models (on training data)

k	MSE	edf
100	12.40	1
50	6.87	2
10	3.86	10
5	3.16	20
2	1.84	50

6 Predictive Model Comparison (or how to choose the best model)

6.1 Predictive Model Evaluation

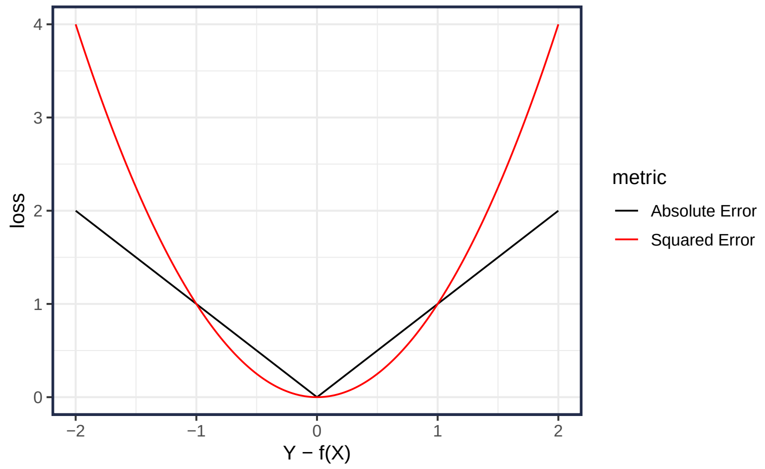
Our goal is prediction, so we should evaluate the models on their *predictive performance*.

- We need to use hold-out data (i.e., data not used to fit the model) to evaluate how well our models do in prediction
- Call these data *test data* $D_{\text{test}} = \{(X_j, Y_j)\}_{j=1}^{J}$
  - Note: assume that the test data comes from the same distribution as the training data
  - Or $P_{\text{test}}(X, Y) = P_{\text{train}}(X, Y)$
  - **both** Y and X from same distribution
- Later in the course we will cover ways to do this when we only have training data (e.g., cross-validation)
- but for today, we have an unlimited amount of *test data* at our disposal (since we know how the data were generated)

6.2 Statistical Decision Theory

- In a prediction context, we want a *point estimate* for the value of an unobserved r.v. $Y \in \mathbb{R}$ given an input feature $X \in \mathbb{R}$.
- Let $f(X)$ be the prediction of Y given X.
- Define a *loss function* $L(Y, f(X))$ that indicates how bad it is if we estimate the value Y by $f(X)$
  - E.g. Y is the number of customers complaints in a call center and X is the day of week
  - If we guess $f(X) = 500$, but there are really $Y = 2000$, how bad would that be?
- A common loss function is *squared error*

$$L(Y, f(X)) = (Y - f(X))^2$$



- The best model is the one that minimizes the *expected loss* or Risk or Expected Prediction Error (EPE)

$$\text{Risk} = \text{EPE} = \text{E[loss]}$$

- For *squared error*, the *risk* for using the model f is:

$$\begin{aligned} R(f) &= E_{XY}[L(Y, f(X))] \\ &= E_{XY}[(Y - f(X))^2] \end{aligned}$$

where the expectation is w.r.t. the *test values* of X, Y.

- Note under squared error loss, the risk is also known as the *mean squared error* (MSE)

- To simplify a bit, let's examine the risk of model f at a given fixed input $X = x$. This removes the uncertainty in X, so we only have uncertainty coming from Y.

$$\begin{aligned} R_x(f) &= E[L(Y, f(x)) \mid X = x] \\ &= \mathrm{E}[(Y - f(x))^2 \mid X = x] \qquad \text{for squared error loss} \end{aligned}$$

where the expectation is taken with respect to $Y|X = x$

- The best prediction $f^*(x)$, given $X = x$, is the value that minimizes the risk

$$\begin{aligned} f^*(x) &= \arg\min_c R_x(c) \\ &= \arg\min_c \mathrm{E}[(Y - c)^2 \mid X = x] \end{aligned}$$

Your Turn #6

What is the optimal prediction at $X = x$ under the squared error loss?

- I.e., find $f^*(x)$.

6.2.1 Squared Error Loss Functions

- **Conclusion:** If quality of prediction is measured by squared error, then the best predictor is the (conditional) expected value $f^*(x) = \mathrm{E}[Y|X = x]$.
  - And the minimum Risk/MSE is $R_x(f^*) = \mathrm{V}[Y|X = x]$

- **Summary:** Under *squared error loss* the Risk (at input x) is

$$\begin{aligned}
R_x(f) &= \mathrm{E}_Y[L(Y, f(X)) \mid X = x] \\
&= \mathrm{E}_Y[(Y - f(x))^2 \mid X = x] \qquad \text{using squared error loss} \\
&= \mathrm{V}[Y \mid X = x] + (E_Y[Y \mid X = x] - f(x))^2 \\
&= \text{Irreducible Variance} + \text{model squared error}
\end{aligned}$$

6.2.2 kNN and Polynomial Regression

- The kNN model estimates the conditional expectation by using the data in a *local region* around x

$$\hat{f}_{\text{knn}}(x; k) = \mathrm{Ave(y_i \mid x_i \in N_k(x))}$$

  This assumes that the true $f(x)$ can be well approximated by a *locally constant* function

- Polynomial (linear) regression, on the other hand, assumes that the true $f(x)$ is well approximated by a *globally polynomial* function

$$\hat{f}_{\text{poly}}(x; d) = \beta_0 + \sum_{j=1}^{d} \beta_j x^j$$

6.2.3 Empirical Risk

- The actual Risk/EPE is based on the expected error from *test data* (out-of-sample), or data that was not used to estimate $\hat{f}$

$$\begin{aligned}
R(f) &= E_{XY}[L(Y, f(X))] \\
&= E_{XY}[(Y - f(X))^2] \qquad \text{for squared error loss}
\end{aligned}$$

  where X, Y are from $\Pr(X, Y)$ (i.e., test data)

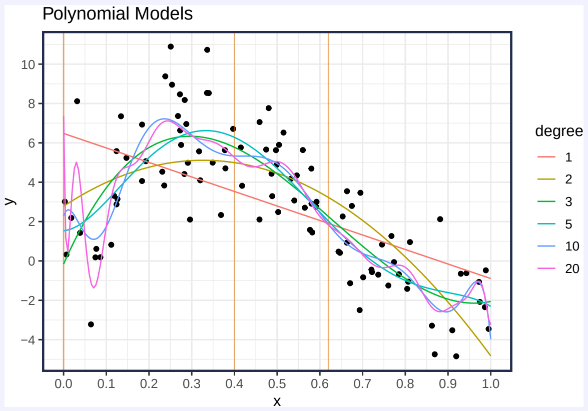
- But is it a bad idea to choose the best model according to *empirical risk* or *training error*?

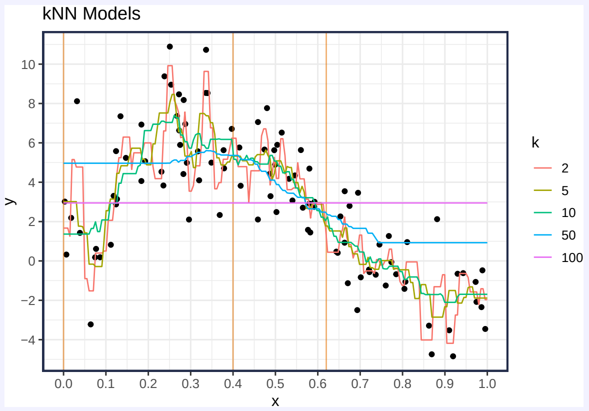
$$\begin{aligned}
\hat{R}_n(f) &= \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) \\
&= \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 \qquad \text{for squared error loss}
\end{aligned}$$

6.3 Choose the best *predictive* model

Your Turn #7

Which model will you choose?
Enter your answer on MS Teams. At the top tab of the class find “Fill | Supervised Learning I : What is the best model?”. You may need to expand “More”.





Polynomial

degree	MSE	npars
1	8.29	2
2	5.58	3
3	4.28	4
5	4.10	6
10	3.65	11
20	3.16	21

kNN

k	MSE	edf
50	6.87	2.00
30	5.06	3.33
20	4.18	5.00
15	4.13	6.67
10	3.86	10.00
5	3.16	20.00

7 Appendix: R Code

7.1 Required R Packages

We will be using the R packages of:

- `FNN` for k nearest neighbor models
- `tidyverse` for data manipulation and visualization
- `broom` for tidying model output
- `tidymodels` (for optional tidymodels approach)

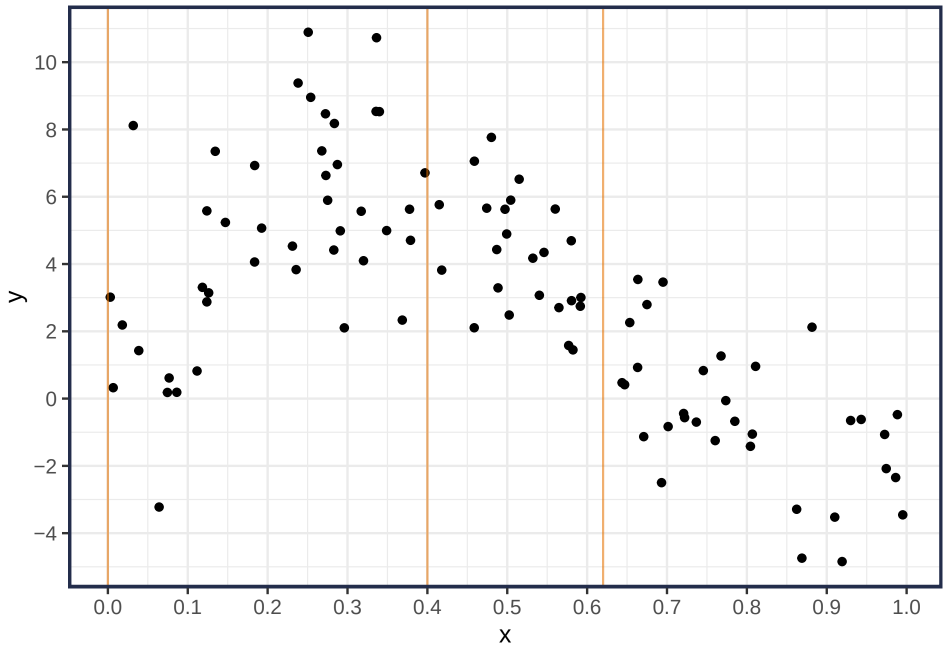
```
library(FNN)
library(broom)
library(tidyverse)
library(tidymodels)
```

7.2 Generate Data

```
#: Functions
sim_x <- function(n) runif(n)            # U[0,1]
mu <- function(x) 1 + 2*x + 5*sin(5*x)   # true mean function
sim_y <- function(x, mu, sd){            # generate Y|X from N{f(x),sd}
  n = length(x)
  noise = rnorm(n, sd=sd)
  mu(x) + noise
}

#: Generate Data
set.seed(825)                            # set seed for reproducibility
n = 100                                  # number of observations
sd = 2                                   # stdev for error
x = sim_x(n)                             # get x values
y = sim_y(x, mu, sd=sd)                  # get y values
data_train = tibble(x, y)                # put data into tibble (data frame)

#: Data for evaluation
xseq = seq(0, 1, length=200)  # sequence of 200 equally spaced values from 0 to 1
xeval = tibble(x = xseq)      # make into a tibble object
```

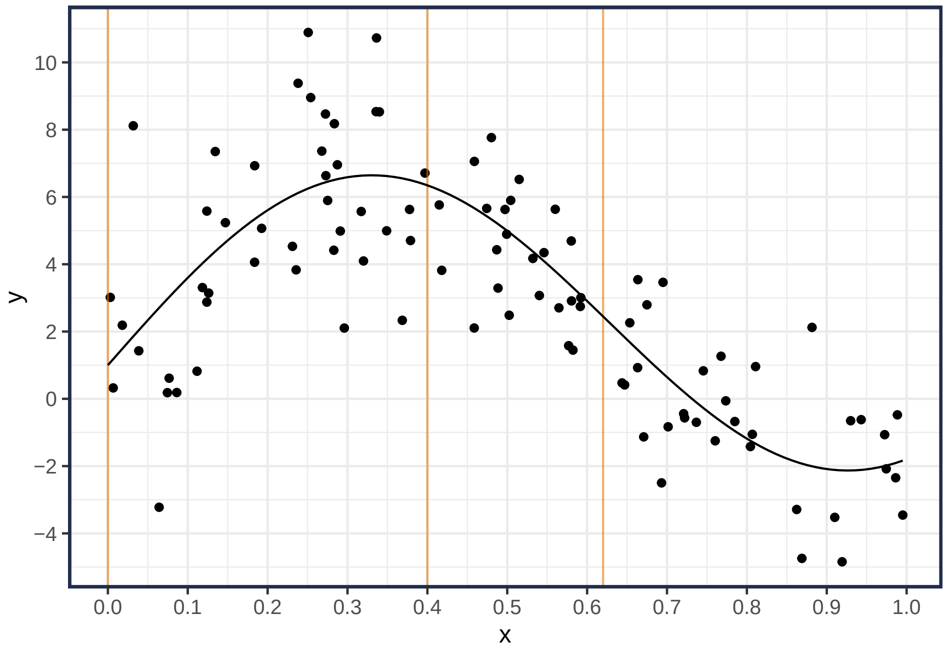
7.3 Make scatter plot

```
gg_example = ggplot(tibble(x,y), aes(x,y)) +
  geom_point() +
  geom_vline(xintercept=c(0, .4, .62), col="#E5720090") +
  scale_x_continuous(breaks=seq(0, 1, by=.10)) +
  scale_y_continuous(breaks=seq(-6, 10, by=2))
```



Add true mean line

```
gg_example +
  geom_function(fun=mu, color="black")
```



7.4 Simple Linear Model

```
#: Load Required Packages
library(broom)
library(tidyverse)
```

```
#: Fitting
m1 = lm(y~x, data=data_train) # fit simple OLS
```

```
summary(m1)                    # summary of model
```

```
#>
#> Call:
#> lm(formula = y ~ x, data = data_train)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -9.229 -1.635  0.019  1.940  6.728
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)    6.478      0.584   11.09  < 2e-16 ***
#> x             -7.372      1.058   -6.97  3.7e-10 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.91 on 98 degrees of freedom
#> Multiple R-squared:  0.331,  Adjusted R-squared:  0.325
#> F-statistic: 48.6 on 1 and 98 DF,  p-value: 3.69e-10
```

```
#: tidy output using broom
broom::tidy(m1)                # model coefficients (as a data frame)
```

```
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic  p.value
#>   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)     6.48     0.584     11.1  5.39e-19
#> 2 x              -7.37     1.06      -6.97 3.69e-10
```

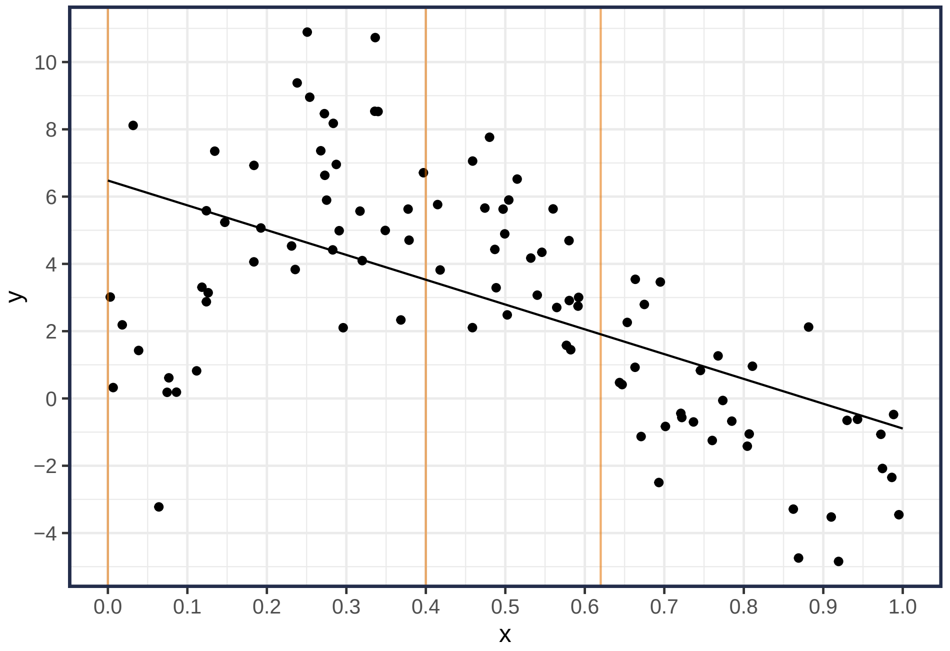
```
broom::glance(m1)              # model properties
```

```
#> # A tibble: 1 x 12
#>   r.squared adj.r.squa~1 sigma stati~2  p.value    df logLik   AIC   BIC devia~3
#>       <dbl>        <dbl> <dbl>   <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>   <dbl>
#> 1     0.331        0.325  2.91    48.6 3.69e-10     1  -248.  501.  509.    829.
#> # ... with 2 more variables: df.residual <int>, nobs <int>, and abbreviated
#> #   variable names 1: adj.r.squared, 2: statistic, 3: deviance
```

```
#: Prediction
yhat1 = predict(m1, xeval)    # vector of yhat's (predictions)

# tidy predictions using broom::augment()
broom::augment(m1, new_data = xeval)
```

```
#> # A tibble: 100 x 8
#>        y     x .fitted  .resid   .hat .sigma     .cooksd .std.resid
#>    <dbl> <dbl>   <dbl>   <dbl>  <dbl>  <dbl>       <dbl>      <dbl>
#> 1 -1.07  0.972  -0.691 -0.375  0.0422   2.92 0.000383      -0.132
#> 2  0.958 0.811   0.500  0.457  0.0245   2.92 0.000319       0.159
#> 3  4.42  0.283   4.39   0.0230 0.0151   2.92 0.000000485    0.00796
#> 4 -0.566 0.722   1.16  -1.72   0.0178   2.92 0.00323       -0.597
#> 5  9.38  0.238   4.72   4.66   0.0177   2.88 0.0235         1.62
#> 6  2.91  0.580   2.20   0.711  0.0114   2.92 0.000348       0.246
#> # ... with 94 more rows
```

```
#: Plotting
gg_example +                           # re-use base plot
  geom_line(data=tibble(x=xseq, y=yhat1), col="black")
  # geom_smooth(method="lm")                  # equivalent method
# Base R plots
# plot(x, y, las=1)                  # plot data
# lines(xseq, yhat1, col="black")    # add fitted line
```



7.4.1 Tidymodels

```
library(tidymodels)
m1_tidy = linear_reg() %>% fit(y~x, data=data_train)

# same results and interface
broom::tidy(m1_tidy)
```

```
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic  p.value
#>   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)     6.48     0.584     11.1  5.39e-19
#> 2 x              -7.37     1.06      -6.97 3.69e-10
```

```
broom::glance(m1_tidy)
```

```
#> # A tibble: 1 x 12
#>   r.squared adj.r.squa~1 sigma stati~2  p.value    df logLik   AIC   BIC devia~3
#>       <dbl>        <dbl> <dbl>   <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>   <dbl>
#> 1     0.331        0.325  2.91    48.6 3.69e-10     1  -248.  501.  509.    829.
#> # ... with 2 more variables: df.residual <int>, nobs <int>, and abbreviated
#> #   variable names 1: adj.r.squared, 2: statistic, 3: deviance
```

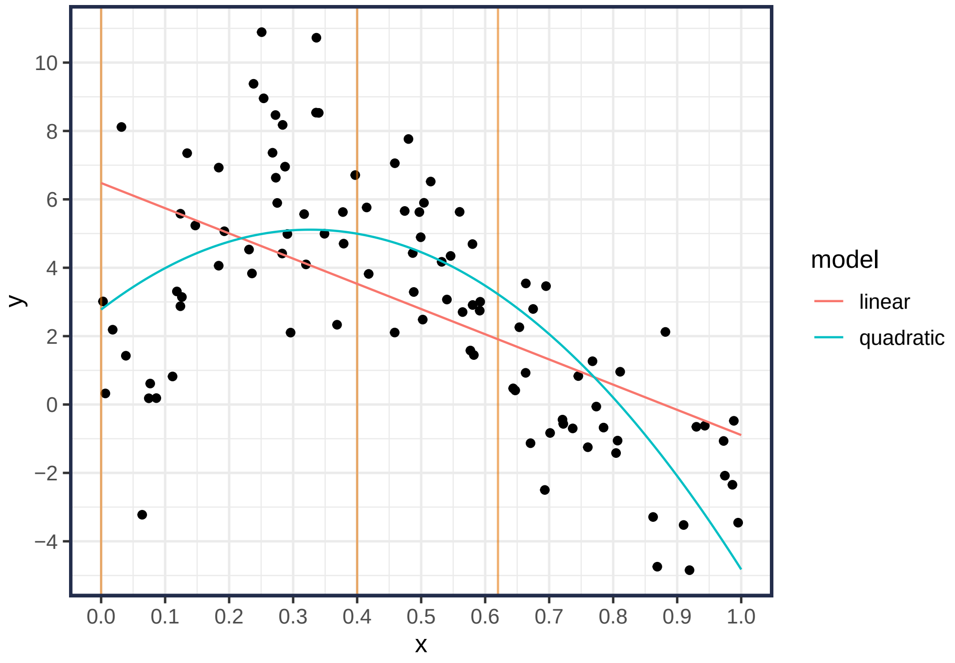
```
broom::augment(m1_tidy, new_data = xeval)
```

```
#> # A tibble: 200 x 2
#>         x .pred
#>     <dbl> <dbl>
#> 1 0        6.48
#> 2 0.00503  6.44
#> 3 0.0101   6.40
#> 4 0.0151   6.37
#> 5 0.0201   6.33
#> 6 0.0251   6.29
#> # ... with 194 more rows
```

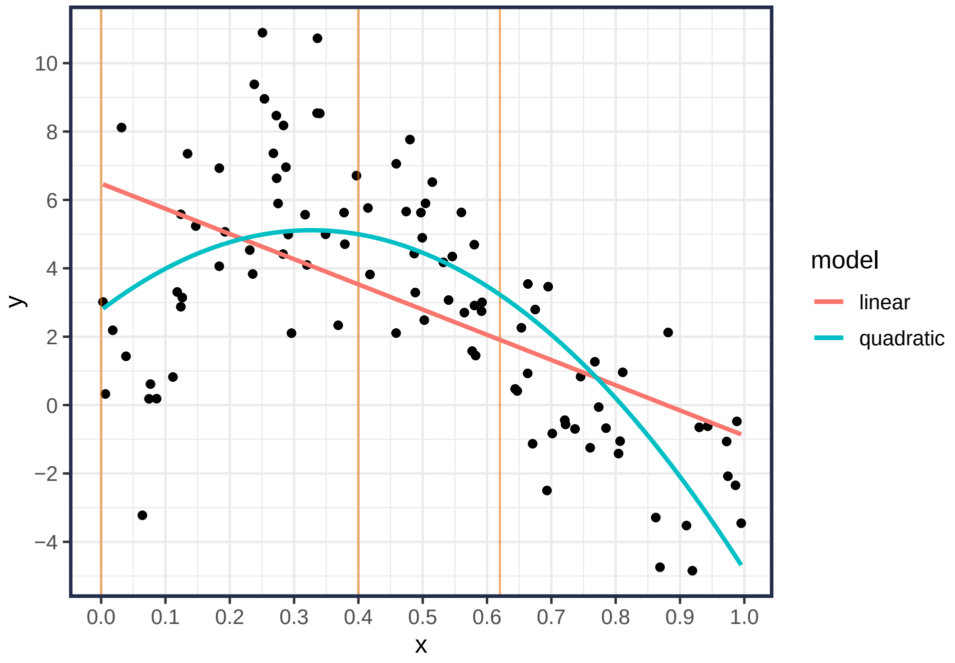
7.5 Polynomial Models

```
#: Fit Quadratic Model
m2 = lm(y~poly(x, degree=2), data=data_train)
yhat2 = predict(m2, xeval)
```

```
#: Plotting

# Option 1: specify data to draw lines
poly.data = tibble(x = xseq, linear=yhat1, quadratic=yhat2) %>%  # long data
  pivot_longer(-x, names_to="model", values_to="y")
gg_example +
  geom_line(data=poly.data, aes(color=model))
```



```
# Option 2: use geom_smooth()
gg_example +
  geom_smooth(method="lm", se=FALSE, aes(color="linear")) +
  geom_smooth(method="lm", formula="y~poly(x,2)", se=FALSE, aes(color="quadratic")) +
  scale_color_discrete(name="model")
```



```r
# Option 3: base R
plot(x, y, las=1)
lines(xseq, yhat1, col="black")
lines(xseq, yhat2, col="red")
legend("topright",
       c("linear", "quadratic"),
       col=c("black", "red"),
       lty=1, cex=.8)
```

7.5.1 Tidymodels

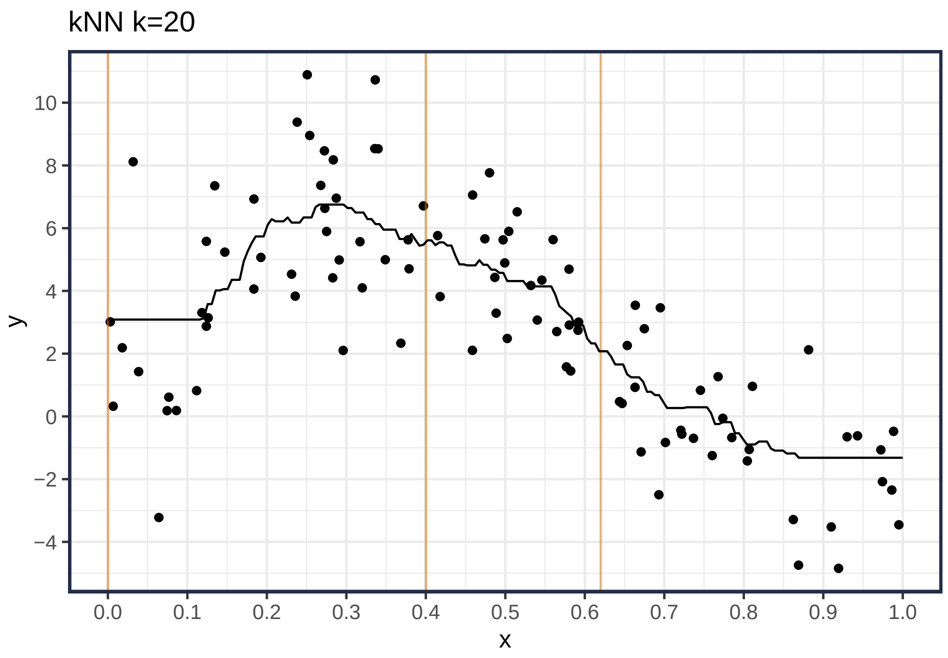
```r
library(tidymodels)

#: Option 1: use formula interface
m2_tidy = linear_reg() %>% fit(y~poly(x, degree=2), data=data_train)

#: Option 2: use a workflow
# A 'recipe' does pre-processing and the `spec` specifies the model


poly_rec = recipe(y~x, data = data_train) %>% # only use "main effects" in fmla
  step_poly(x, degree = 2)                    # use step_{} to transformations

m2_wf = workflow(
  preprocessor = poly_rec, # recipe or formula here
  spec = linear_reg()      # a model specification
) %>%
  fit(data = data_train)
```

7.6 K Nearest Neighbor Regression

```r
library(FNN) # this is just one of many KNN packages
```

```
#: fit a k=20 knn regression
knn.20 = knn.reg(select(data_train, x), test=xeval, y=data_train$y, k=20)
```

```
#-- ggplot2 plot
gg_example +
  geom_line(data=xeval %>% mutate(y=knn.20$pred)) +
  ggtitle("kNN k=20")
```

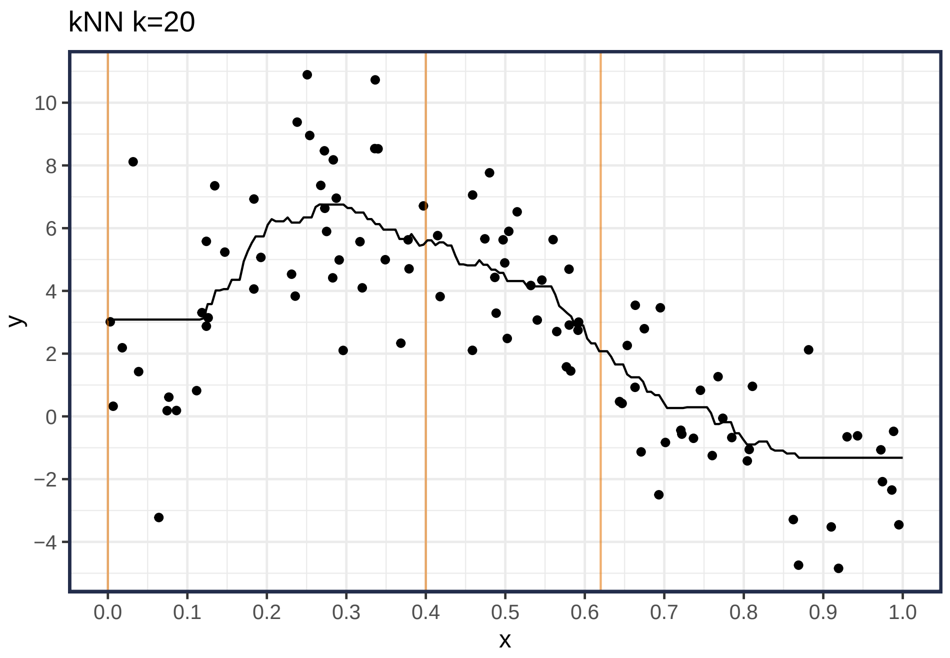


```
#-- base R plot
plot(x, y, las=1)
lines(xseq, knn.20$pred)
```

7.6.1 Tidymodels

```
library(tidymodels)

knn_tidy.20 = nearest_neighbor(
  mode = "regression",    # specify mode = regression (instead of classification)
  neighbors = 20,         # number of neighbors (k)
  weight_func = "rect"    # this function can use weighted average ("rect" = no weighting)
) %>%
  fit(y~x, data = data_train)
```

```
gg_example +
  geom_line(data=augment(knn_tidy.20, xeval), aes(y = .pred)) +
  ggtitle("kNN k=20")
```


kNN k=20
y
x
10
8
6
4
2
0
−2
−4
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0