Applied Statistical Methods

Moving Beyond Linearity

Xuemao Zhang
East Stroudsburg University

April 7, 2023

Outline

Linearity assumption is not always good enough.

- Polynomial Regression
- Logistic Regression
- Polynomial Regressions with Python
- Splines
  - Cubic Splines
  - Natural Cubic Splines
  - Smoothing Splines
- Local regression

Polynomial Regression

- We already used polynomial regression models, which essentially are multiple linear regression models.
- The standard way to extend linear regression to nonlinear is to replace the standard linear model with a polynomial function

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d + \varepsilon_i$$

- Logistic regression follows naturally.

$$Pr(y_i > 250|x_i) = \frac{\exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d)}{1 + \exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d)}$$

Logistic Regression

- We explain the concept of logistic regression models before we continue.
- Example: Risk Factors Associated with Discharge Disposition Following Rehabilitation (n=100)
    - Response variable: `Discharge`
        - ★ code 0 for going home (reference), and
        - ★ 1 for going to LTC (target).

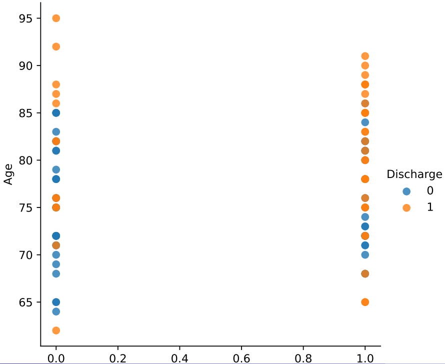
```
import pandas as pd
risk=pd.read_csv("../data/Risk_Factors.csv")
print(risk.columns)
```

```
## Index(['ADL', 'Age', 'Marital', 'Gender', 'Discharge'], dtype='object')
```

Logistic Regression

- Plot of ADL, Age and Discharge
- The `regplot()` and `lmplot()` functions are closely related, but the former is an axes-level function while the latter is a figure-level function that combines regplot() and FacetGrid.
  - https://seaborn.pydata.org/generated/seaborn.lmplot.html

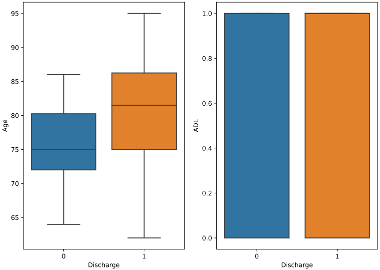
```
import matplotlib.pyplot as plt
import seaborn as sns
#plt.scatter(risk['ADL'], risk['Age'], color = 'lightblue')
sns.lmplot(data=risk, x='ADL', y='Age',ci=None, fit_reg=False,
hue = 'Discharge')
```



Logistic Regression

- Plot of Age vs. Discharge, and ADL vs. Discharge

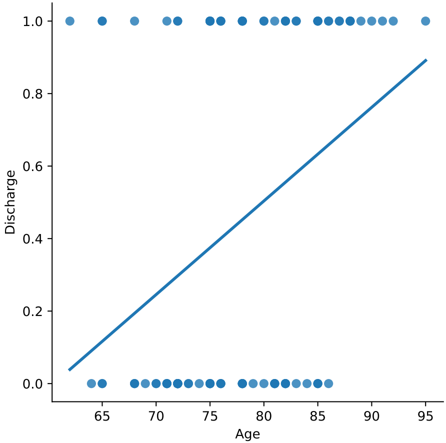
```
fig, ax = plt.subplots(1, 2)
sns.boxplot(x='Discharge', y='Age', data=risk, ax=ax[0])
sns.boxplot(x='Discharge', y='ADL', data=risk, ax=ax[1])
plt.show()
```



Logistic Regression

- Can we use Linear Regression?

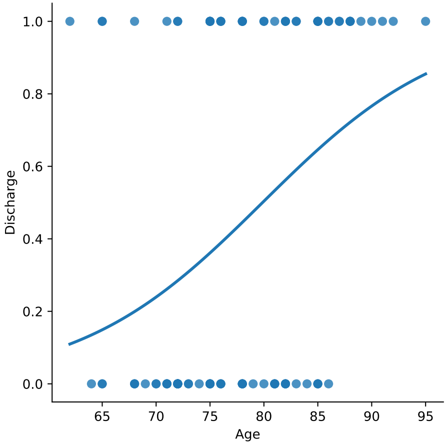
```
sns.lmplot(data=risk, x='Age', y='Discharge',ci=None, fit_reg=True)
```



Logistic Regression

- In this case of a binary outcome, linear regression might produce probabilities less than zero or bigger than one. So it can not give a good estimate of $E(Y|X = x) = Pr(Y = 1|X = x)$. Logistic regression is more appropriate.

```
sns.lmplot(data=risk, x='Age', y='Discharge',ci=None,
logistic=True, fit_reg=True)
```



Logistic Regression

- Logistic regression is the straightforward extension of linear regression to the binay responses setting.
- We consider the case that $y \in \{0, 1\}$
- Let $p(X) = Pr(Y = 1|X)$
  - For example, we want to use biomarker level to predict probability of cancer.
- Logistic regression uses the form

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

  - $p(X)$ will lie between 0 and 1.
- Furthermore,

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X.$$

  - This function of $p(X)$ is called the logit or log odds (by log we mean natural log : ln).

Logistic Regression

- We use **maximum likelihood** to estimate the parameters
- Most statistical packages can fit linear logistic regression models by maximum likelihood. For example, in `sklearn.linear_model`

```
from sklearn.linear_model import LogisticRegression
log_regression = LogisticRegression()
fit1=log_regression.fit(risk['Age'].values.reshape(-1, 1), risk['Discharge'])
fit1.coef_
```

```
## array([[0.11702883]])
```

```
fit1.intercept_
```

```
## array([-9.34722785])
```

Logistic Regression

- Function `logit()` in `statsmodels`

```
import statsmodels.formula.api as smf
fit2=smf.logit("Discharge~Age", data=risk).fit()
```

```
## Optimization terminated successfully.
##          Current function value: 0.615875
##          Iterations 5
```

```
fit2.summary()
```

```
## <class 'statsmodels.iolib.summary.Summary'>
## """
##                            Logit Regression Results
## ==============================================================================
## Dep. Variable:              Discharge   No. Observations:                  100
## Model:                          Logit   Df Residuals:                       98
## Method:                           MLE   Df Model:                            1
## Date:                Fri, 07 Apr 2023   Pseudo R-squ.:                  0.1021
## Time:                        21:46:50   Log-Likelihood:                -61.588
## converged:                       True   LL-Null:                       -68.593
## Covariance Type:            nonrobust   LLR p-value:                 0.0001818
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept     -9.3578      2.661     -3.517      0.000     -14.573      -4.143
```

Logistic Regression

- Function glm() in statsmodels

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
fit3=smf.glm("Discharge~Age", data=risk,family = sm.families.Binomial()).fit()
fit3.summary()
```

```
## <class 'statsmodels.iolib.summary.Summary'>
## """
##                  Generalized Linear Model Regression Results
## ==============================================================================
## Dep. Variable:              Discharge   No. Observations:                  100
## Model:                            GLM   Df Residuals:                       98
## Model Family:                Binomial   Df Model:                            1
## Link Function:                  Logit   Scale:                          1.0000
## Method:                          IRLS   Log-Likelihood:                -61.588
## Date:            Fri, 07 Apr 2023   Deviance:                       123.18
## Time:                        21:46:52   Pearson chi2:                     104.
## No. Iterations:                     4   Pseudo R-squ. (CS):             0.1307
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept     -9.3578      2.661     -3.517      0.000     -14.573      -4.143
## Age            0.1172      0.034      3.447      0.001       0.051       0.184
## ==============================================================================
```

Logistic Regression

- Logistic Regression with Several Variables: Suppose that there are p predictors: $X_1, \ldots, X_p$.
- Just like the logistic regression models with one predictor

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

- And just like before

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$

- We'll discuss more about logistic regression models later.

Logistic Regression

```
fit4=smf.logit("Discharge~ADL+Age+Marital+Gender", data=risk).fit()
```

```
## Optimization terminated successfully.
##          Current function value: 0.551655
##          Iterations 6
```

```
fit4.summary()
```

```
## <class 'statsmodels.iolib.summary.Summary'>
## """
##                            Logit Regression Results
## ==============================================================================
## Dep. Variable:              Discharge   No. Observations:                  100
## Model:                          Logit   Df Residuals:                       95
## Method:                           MLE   Df Model:                            4
## Date:                Fri, 07 Apr 2023   Pseudo R-squ.:                  0.1958
## Time:                        21:46:53   Log-Likelihood:                -55.166
## converged:                       True   LL-Null:                       -68.593
## Covariance Type:            nonrobust   LLR p-value:                 2.127e-05
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept    -10.2740      2.935     -3.500      0.000     -16.027      -4.521
## ADL            0.9652      0.481      2.008      0.045       0.023       1.907
## Age            0.1175      0.037      3.179      0.001       0.045       0.190
## Marital        1.0189      0.480      2.121      0.034       0.077       1.960
```

Python: Polynomial Regression

- For the `Wage` data, we first fit a simple linear regression model.

```
import pandas as pd
from statsmodels.formula.api import ols
Wage=pd.read_csv('../data/Wage.csv')
fit0=ols('wage~age', Wage).fit()
print(fit0.params)
```

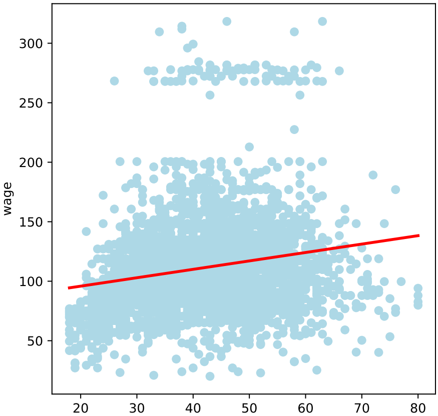
```
## Intercept    81.704735
## age           0.707276
## dtype: float64
```

```
print(fit0.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                   wage   R-squared:                       0.038
## Model:                            OLS   Adj. R-squared:                  0.038
## Method:                 Least Squares   F-statistic:                     119.3
## Date:                Fri, 07 Apr 2023   Prob (F-statistic):           2.90e-27
## Time:                        21:46:55   Log-Likelihood:                -15391.
## No. Observations:                3000   AIC:                         3.079e+04
## Df Residuals:                    2998   BIC:                         3.080e+04
## Df Model:                           1
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept     81.7047      2.846     28.706      0.000      76.124      87.286
## age            0.7073      0.065     10.923      0.000       0.580       0.834
## ==============================================================================
## Omnibus:                     1065.217   Durbin-Watson:                   1.952
## Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4518.235
## Skew:                           1.690   Prob(JB):                         0.00
## Kurtosis:                       7.972   Cond. No.                         168.
## ==============================================================================
##
## Notes:
## [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Python: Polynomial Regression

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.scatter(Wage['age'], Wage['wage'], color = 'lightblue')
sns.regplot(data=Wage,x='age', y='wage',ci=None,
   scatter = False, color = 'red')
plt.show()
```



Python: Polynomial Regression

- For the Wage data, we fit a simple polynomial regression model with degree 4.

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 4)
X_poly = poly.fit_transform(Wage['age'].values.reshape(-1, 1))
fit1 = LinearRegression(fit_intercept=False)
fit1.fit(X_poly, Wage['wage'])
```

```
## LinearRegression(fit_intercept=False)
```

```
fit1.intercept_
```

```
## 0.0
```

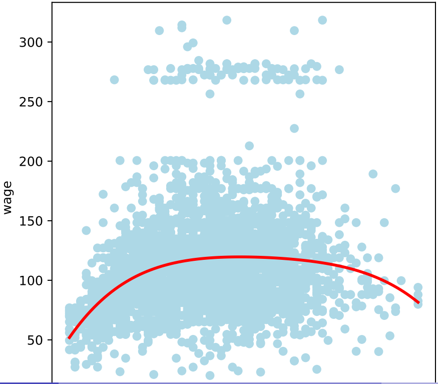
```
fit1.coef_
```

```
## array([-1.84154180e+02,  2.12455205e+01, -5.63859312e-01,  6.81068771e-0
##        -3.20383037e-05])
```

Python: Polynomial Regression

- Plot

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.scatter(Wage['age'], Wage['wage'], color = 'lightblue')
sns.regplot(data=Wage,x='age', y='wage',ci=None, order=4,
scatter = False, color='red')
plt.show()
```



Python: Polynomial Regression

- There are other ways to fit polynomial regression models. For example, use `statsmodels` or `numpy`
  - https://www.statsmodels.org/dev/generated/statsmodels.formula.api.gls.html
  - https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html
  - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html#sklearn.preprocessing.PolynomialFeatures.fit_transform

```
from statsmodels.formula.api import ols
fit2a=ols('wage~age + I(age**2)+I(age**3)+I(age**4)', data = Wage).fit()
#fit2a.summary()
```

```
import statsmodels.api as sm
X4 = PolynomialFeatures(4).fit_transform(Wage['age'].values.reshape(-1,1))
fit2b=sm.GLS(Wage['wage'], X4).fit() #GLS can be replaced by GLM
#fit2b.summary()
```

```
import numpy as np
fit2c=np.poly1d(np.polyfit(Wage['age'],  Wage['wage'], 4))
print(fit2c)  # the order of the coefficients is reversed
```

```
##             4            3          2
## -3.204e-05 x + 0.006811 x - 0.5639 x + 21.25 x - 184.2
```

Python: Polynomial Regression

- In performing a polynomial regression we must decide on the degree of the polynomial to use. One way to do this is by using hypothesis tests.
    - We use the `statsmodels.api.stats.anova_lm()` function, which performs an analysis of variance in order to test the null hypothesis that a model $\mathcal{M}_1$ is sufficient to explain the data against the alternative hypothesis that a more complex model $\mathcal{M}_2$ is required. In order to use the `anova()` function, $\mathcal{M}_1$ and $\mathcal{M}_2$ must be **nested models**.
- Suppose we choose among 5 models

```
X1 = PolynomialFeatures(1).fit_transform(Wage['age'].values.reshape(-1,1))
X2 = PolynomialFeatures(2).fit_transform(Wage['age'].values.reshape(-1,1))
X3 = PolynomialFeatures(3).fit_transform(Wage['age'].values.reshape(-1,1))
X4 = PolynomialFeatures(4).fit_transform(Wage['age'].values.reshape(-1,1))
X5 = PolynomialFeatures(5).fit_transform(Wage['age'].values.reshape(-1,1))
```

Python: Polynomial Regression

- Either a cubic or a quadratic polynomial appear to provide a reasonable fit to the data, but lower- or higher-order models are not justified.

```
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm

fit_1 = sm.GLS(Wage['wage'], X1).fit()
fit_2 = sm.GLS(Wage['wage'], X2).fit()
fit_3 = sm.GLS(Wage['wage'], X3).fit()
fit_4 = sm.GLS(Wage['wage'], X4).fit()
fit_5 = sm.GLS(Wage['wage'], X5).fit()

print(sm.stats.anova_lm(fit_1, fit_2, fit_3, fit_4, fit_5, typ=1))
```

```
##    df_resid           ssr  df_diff        ss_diff           F       Pr(
## 0    2998.0  5.022216e+06      0.0            NaN         NaN
## 1    2997.0  4.793430e+06      1.0  228786.010101  143.593107  2.363850e
## 2    2996.0  4.777674e+06      1.0   15755.693660    9.888756  1.679202e
## 3    2995.0  4.771604e+06      1.0    6070.152118    3.809813  5.104620e
## 4    2994.0  4.770322e+06      1.0    1282.563016    0.804976  3.696820e
```

Python: Polynomial Regression

- The `anova_lm` method can compare models when there are other terms in the model.
- Join two arrarys: https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html

```
import numpy as np
import pandas as pd
Wage['education'].unique()
```

```
## array(['1. < HS Grad', '4. College Grad', '3. Some College', '2. HS Grad',
##        '5. Advanced Degree'], dtype=object)
```

```
dummies = pd.get_dummies(Wage['education'])
#dummies.columns
education=dummies[['1. < HS Grad','2. HS Grad','3. Some College',
'4. College Grad']]
X_1=np.concatenate((X1,education.values.reshape(-1,4)), axis=1)
X_2=np.concatenate((X2,education.values.reshape(-1,4)), axis=1)
X_3=np.concatenate((X3,education.values.reshape(-1,4)), axis=1)
fit_11 = sm.GLS(Wage['wage'], X_1).fit()
fit_21 = sm.GLS(Wage['wage'], X_2).fit()
fit_31 = sm.GLS(Wage['wage'], X_3).fit()
#print(fit_31.summary())
```

Python: Polynomial Regression

```
print(sm.stats.anova_lm(fit_11, fit_21, fit_31, typ=1))
```

```
##    df_resid           ssr  df_diff         ss_diff           F        Pr(
## 0    2994.0  3.867992e+06      0.0             NaN         NaN
## 1    2993.0  3.725395e+06      1.0   142597.096993  114.696898  2.728001e
## 2    2992.0  3.719809e+06      1.0     5586.660320    4.493588  3.410431e
```

Python: Polynomial Regression

- Next we consider the task of predicting whether an individual earns more than \$250,000 per year. We fit a polynomial logistic regression model with response `(Wage['wage']>250)`.
- Scikit-learn implements a regularized logistic regression model particularly suitable for high dimensional data. Since we just have one feature (age) we use the GLM model from `statsmodels`.
  - https://www.statsmodels.org/dev/generated/statsmodels.genmod.generalized_linear_model.GLM.html
  - https://www.statsmodels.org/dev/glm.html#module-statsmodels.genmod.families.family

```
import statsmodels.api as sm
X4 = PolynomialFeatures(4).fit_transform(Wage['age'].values.reshape(-1,1))
y=(Wage['wage']> 250).map({False:0, True:1}).values
logistic_model = sm.GLM(y, X4, family=sm.families.Binomial())
logistic_fit = logistic_model.fit()
print(logistic_fit.summary())
```

```
##                  Generalized Linear Model Regression Results
## ==============================================================================
## Dep. Variable:                      y   No. Observations:                 3000
## Model:                            GLM   Df Residuals:                     2995
## Model Family:                Binomial   Df Model:                            4
## Link Function:                  Logit   Scale:                          1.0000
```

Python: Polynomial Regression

- We first create a grid of values for age at which we want predictions.

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
age_grid = np.arange(Wage.age.min(), Wage.age.max()).reshape(-1,1)
X_test = PolynomialFeatures(4).fit_transform(age_grid)
preds=logistic_fit.predict(X_test)
```

- The predictions are for the logit. That is, the predictions are of the form $\hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p$

$$\log\left(\frac{Pr(Y=1|X)}{1-Pr(Y=1|X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$

Python: Polynomial Regression

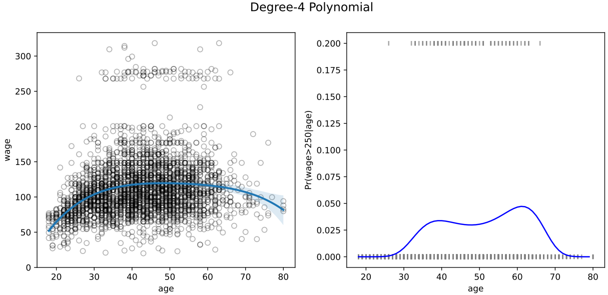
- Now we plot the fitted probabilities.

```
import matplotlib.pyplot as plt
import seaborn as sns
# creating plots
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))
fig.suptitle('Degree-4 Polynomial', fontsize=14)

# Scatter plot with polynomial regression line
ax1.scatter(Wage.age, Wage.wage, facecolor='None', edgecolor='k', alpha=0.3)
sns.regplot(x=Wage.age, y=Wage.wage, order = 4, logistic=False,
truncate=True, scatter=False, ax=ax1)
ax1.set_ylim(ymin=0)

# Logistic regression showing Pr(wage>250) for the age range.
ax2.plot(age_grid, preds, color='b')

# Rug plot showing the distribution of wage>250 in the training data.
# 'True' on the top, 'False' on the bottom.
ax2.scatter(Wage.age, y/5, s=30, c='grey', marker='|', alpha=0.7)

ax2.set_ylim(-0.01,0.21)
ax2.set_xlabel('age')
ax2.set_ylabel('Pr(wage>250|age)')
plt.show()
```

Python: Polynomial Regression

```
## (0.0, 333.25527475900003)
```

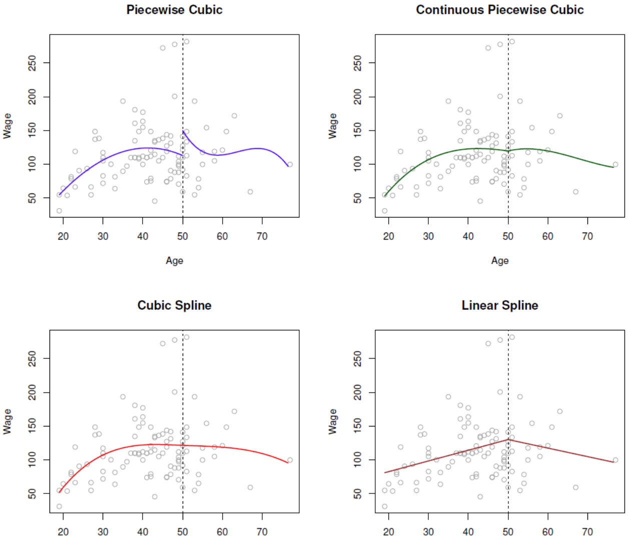
```
## (-0.01, 0.21)
```



- We can use `lmplot()` in `seaborn` to plot logistic regression fit. But we the data must be data frame form.

Piecewise Polynomials

- Instead of a single polynomial in X over its whole domain, we can rather use different polynomials in regions defined by **knots**.



Linear Splines

- Better to add constraints to the polynomials, e.g. continuity.
- **Splines** have the "maximum'' amount of continuity.
- Suppose the knot is $\xi = 50$, then the model for the linear spline is

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \varepsilon_i,$$

where $b_1(x_i)$ and $b_2(x_i)$ are basis functions:

$$\begin{aligned} b_1(x_i) =& x_i \\ b_2(x_i) =& (x_i - 50)_+ = \begin{cases} x_i - 50, & \text{if } x_i > 50 \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

- The construction guarantees that the linear spline is continuous at the knot 50.

Linear Splines

- A linear spline with knots at $\xi_k, k = 1, \ldots, K$ is a piecewise linear polynomial continuous at each knot. We can represent this model as

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+1} b_{K+1}(x_i) + \varepsilon_i,$$

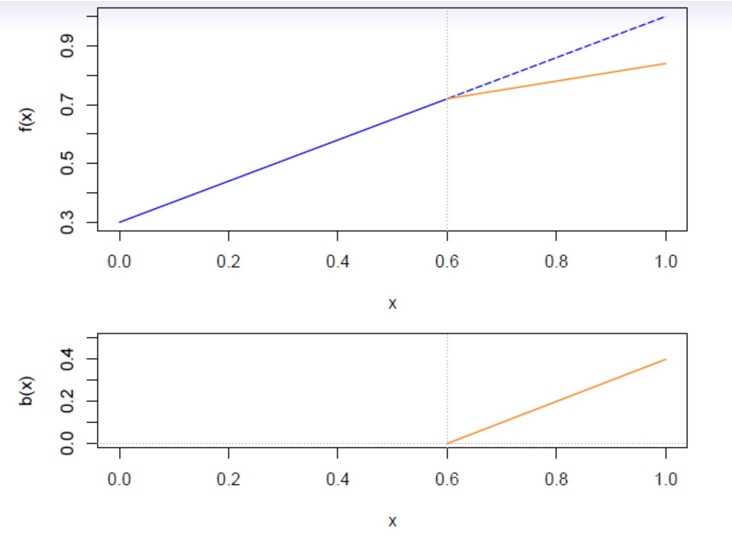
where $b_k(x_i)$ are basis functions:

$$\begin{aligned} b_1(x_i) =& x_i \\ b_{k+1}(x_i) =& (x_i - \xi_k)_+, k = 1, \ldots, K. \end{aligned}$$

Here $()_+$ means positive part:

$$(x_i - \xi_k)_+ = \begin{cases} x_i - \xi_k, & \text{if } x_i > \xi_k \\ 0, & \text{otherwise} \end{cases}$$

Linear Splines



Cubic Splines

- A cubic spline with knots at $\xi_k, k = 1, \ldots, K$ is a piecewise cubic polynomial with continuous derivatives up to order 2 at each knot. We can represent this model as

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \varepsilon_i,$$

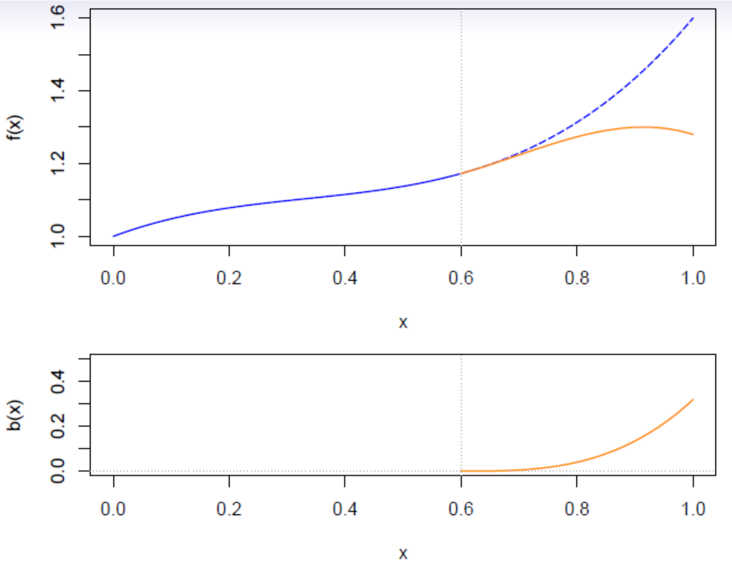
where $b_k(x_i)$ are basis functions:

$$\begin{aligned} b_1(x_i) &= x_i \\ b_2(x_i) &= x_i^2 \\ b_3(x_i) &= x_i^3 \\ b_{k+1}(x_i) &= (x_i - \xi_k)_+^3, k = 1, \ldots, K. \end{aligned}$$

where

$$(x_i - \xi_k)_+^3 = \begin{cases} (x_i - \xi_k)^3, & \text{if } x_i > \xi_k \\ 0, & \text{otherwise} \end{cases}$$

Cubic Splines



Python: Cubic Splines

- In Python, we need to first generate the basis function matrix for splines, and then fit with the linear regression model.
  - Using patsy to create non-linear transformations of the input data. See http://patsy.readthedocs.org/en/latest/
  - The bs() function generates the entire matrix of basis functions for splines with the specified set of knots.
  - Here we have prespecified knots at ages 25, 40, and 60. This produces a spline with six basis functions. (Recall that a cubic spline with three knots has seven degrees of freedom; these degrees of freedom are used up by an intercept, plus six basis functions.)

```
import statsmodels.api as sm
from patsy import dmatrix
# Specifying 3 knots:
transformed_x = dmatrix("bs(Wage.age, knots=(25,40,60), degree=3, include_intercept=False)",
{"df.age": Wage.age}, return_type='dataframe')
# Build a regular linear model from the splines:
fit3 = sm.GLM(Wage.wage, transformed_x).fit()
print(fit3.params)
```

```
## Intercept                                                              60.493714
## bs(Wage.age, knots=(25, 40, 60), degree=3, include_intercept=False)[0]    3.980500
## bs(Wage.age, knots=(25, 40, 60), degree=3, include_intercept=False)[1]   44.630980
## bs(Wage.age, knots=(25, 40, 60), degree=3, include_intercept=False)[2]   62.838788
## bs(Wage.age, knots=(25, 40, 60), degree=3, include_intercept=False)[3]   55.990830
## bs(Wage.age, knots=(25, 40, 60), degree=3, include_intercept=False)[4]   50.688098
## bs(Wage.age, knots=(25, 40, 60), degree=3, include_intercept=False)[5]   16.606142
## dtype: float64
```

Python: Cubic Splines

- Instead of using knots, we could also use the `df` option to produce a spline with knots at uniform quantiles of the data:

```
transformed_x2 = dmatrix("bs(Wage.age, df=6, include_intercept=False)",
{"df.age": Wage.age}, return_type='dataframe')
# Build a regular linear model from the splines:
fit4 = sm.GLM(Wage.wage, transformed_x2).fit()
print(fit4.params)
```

```
## Intercept                                       56.313841
## bs(Wage.age, df=6, include_intercept=False)[0]    27.824002
## bs(Wage.age, df=6, include_intercept=False)[1]    54.062546
## bs(Wage.age, df=6, include_intercept=False)[2]    65.828391
## bs(Wage.age, df=6, include_intercept=False)[3]    55.812734
## bs(Wage.age, df=6, include_intercept=False)[4]    72.131473
## bs(Wage.age, df=6, include_intercept=False)[5]    14.750876
## dtype: float64
```

Natural Splines

- Unfortunately, splines can have high variance at the outer range of the predictors—that is, when X takes on either a very small or very large value.
- A natural spline is a regression spline with additional boundary constraints: the function is required to be linear at the boundary (in the region where X is smaller than the smallest knot, or larger than the largest knot).
  - This additional constraint means that natural splines generally produce more stable estimates at the boundaries.

Python: Natural Splines

- In order to instead fit a natural spline, we use the cr() function. Here we fit a natural spline with four degrees of freedom:

```
# Specifying 4 degrees of freedom
transformed_x3 = dmatrix("cr(Wage.age, df=4)", {"df.age": Wage.age}, return_type='dataframe')
fit5 = sm.GLM(Wage.wage, transformed_x3).fit()
print(fit5.params)
```

```
## Intercept                 79.642095
## cr(Wage.age, df=4)[0]    -14.667784
## cr(Wage.age, df=4)[1]     36.811142
## cr(Wage.age, df=4)[2]     35.934874
## cr(Wage.age, df=4)[3]     21.563863
## dtype: float64
```

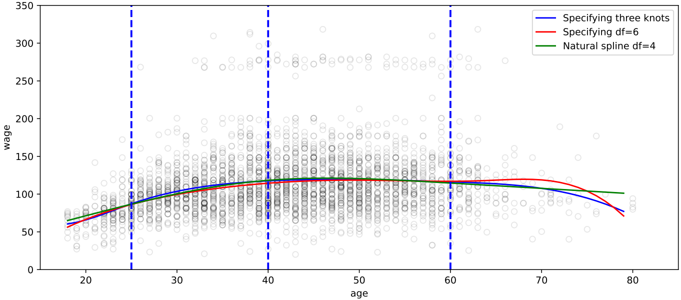
Python: Cubic Splines and Natural Splines

- Plot: Let's see how these three models stack up

```
# Generate a sequence of age values spanning the range
age_grid = np.arange(Wage.age.min(), Wage.age.max()).reshape(-1,1)
pred3 = fit3.predict(dmatrix("bs(age_grid, knots=(25,40,60), include_intercept=False)",
{"age_grid": age_grid}, return_type='dataframe'))
pred4 = fit4.predict(dmatrix("bs(age_grid, df=6, include_intercept=False)",
{"age_grid": age_grid}, return_type='dataframe'))
pred5 = fit5.predict(dmatrix("cr(age_grid, df=4)", {"age_grid": age_grid}, return_type='datafran
plt.scatter(Wage.age, Wage.wage, facecolor='None', edgecolor='k', alpha=0.1)
plt.plot(age_grid, pred3, color='b', label='Specifying three knots')
plt.plot(age_grid, pred4, color='r', label='Specifying df=6')
plt.plot(age_grid, pred5, color='g', label='Natural spline df=4')
[plt.vlines(i , 0, 350, linestyles='dashed', lw=2, colors='b') for i in [25,40,60]]
plt.legend()
plt.xlim(15,85)
plt.ylim(0,350)
plt.xlabel('age')
plt.ylabel('wage')
plt.show()
```

Python: Cubic Splines and Natural Splines

```
## (15.0, 85.0)

## (0.0, 350.0)
```



Smoothing Splines

- In fitting a smooth curve to a set of data, what we really want to do is find some function, say $g(x)$, that fits the observed data well.
- Consider this criterion for fitting a smooth function $g(x)$ to some data:

$$\text{minimize}_{g \in \mathcal{S}} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int_{\text{range of data}} [g^{''}(t)]^2 \; dt$$

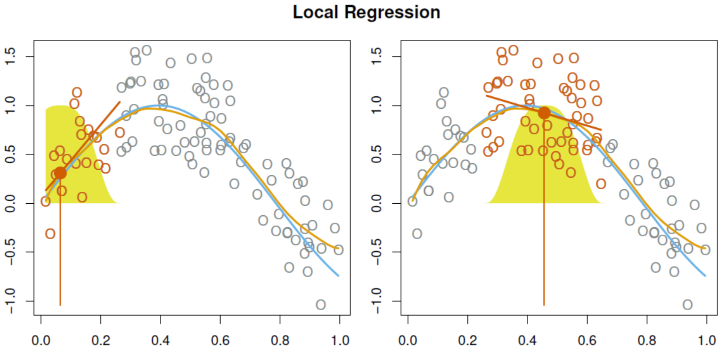
  - The equation takes the "Loss+Penalty" formulation that we encounter in the context of ridge regression and the lasso regression.
  - The first term is RSS, and tries to make $g(x)$ match the data at each x_i.
  - The second term is a roughness penalty and controls how wiggly $g(x)$ is. It is modulated by the tuning parameter $\lambda \geq 0$.
    - ★ The smaller λ, the more wiggly the function, eventually interpolating y_i when $\lambda = 0$.
    - ★ As $\lambda \to \infty$, the function g(x) becomes linear.

Smoothing Splines

- The solution is a natural cubic spline, with a knot at every unique value of x_i. The roughness penalty still controls the roughness via λ.
- However, it is not the same natural cubic spline that one would get if one applied the basis function approach.
  - It is a **shrunken version of such a natural cubic spline**, where the value of the tuning parameter λ controls the level of shrinkage.
- In R, the function *smooth.spline*() will fit a smoothing spline.
  - We can use R functions in Python with package `rpy2`. In order to fit a smoothing spline, we use the `smooth.spline()` function in R.
    - https://rpy2.github.io/doc/v2.9.x/html/introduction.html
  - We can specify df rather than λ.
- Check this link for an example https://stackoverflow.com/questions/51321100/python-natural-smoothing-splines

Local Regression

- Local regression is a different approach for fitting flexible non-linear functions, which involves computing the fit at a target point x_0 using only the nearby training observations.



Local Regression

- With a sliding weight function, we fit separate linear fits over the range of X by weighted least squares.
- Algorithm: Local Regression At $X = x_0$
  - 1. Gather the fraction $s = k/n$ of training points whose x_i are closest to x_0.
  - 2. Assign a weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.
  - 3. Fit a weighted least squares regression of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_0$ that minimize

$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$$

  - 4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

Python: Local Regression

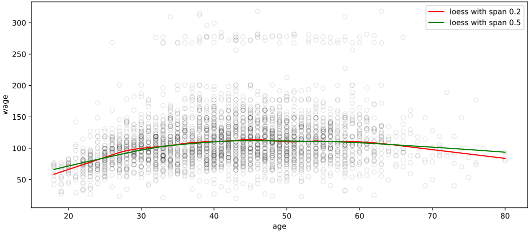
- In order to perform local regression, we use the `loess()` function in `statsmodels.api`
  - https://www.statsmodels.org/dev/generated/statsmodels.nonparametric.smoothers_lowess.lowess.html

```
import numpy as np
import statsmodels.api as sm
lowess = sm.nonparametric.lowess
fit1=lowess(Wage.wage, Wage.age,frac=.2) #fitted values
fit2=lowess(Wage.wage, Wage.age,frac=.5)
```

Python: Local Regression

- Here we have performed local linear regression using spans of 0.2 and 0.5: that is, each neighborhood consists of 20% or 50% of the observations. The larger the span, the smoother the fit.

```
import matplotlib.pyplot as plt
plt.scatter(Wage.age, Wage.wage, facecolor='None', edgecolor='k', alpha=0.1)
plt.plot(fit1[:,0], fit1[:,1], color='r', label='loess with span 0.2')
plt.plot(fit2[:,0], fit2[:,1], color='g', label='loess with span 0.5')
plt.legend()
plt.xlabel('age')
plt.ylabel('wage')
plt.show()
```



License