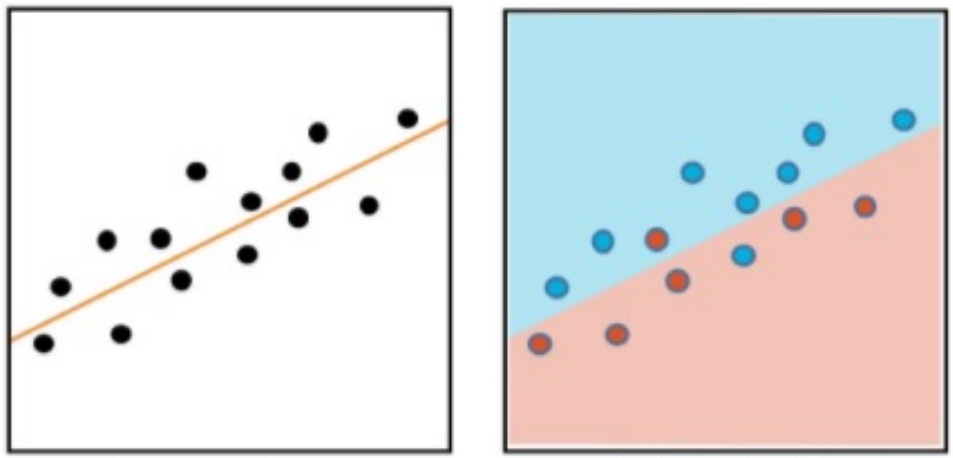
Regression Models

Ishwar K Sethi

What is Regression?

A return to a former or less developed state – Non-technical definition

A modelling technique wherein the behavior of one or more dependent variables is analyzed with respect to one or more independent variables – Technical definition



Regression involves fitting a curve/surface to multidimensional data that best describes it while classification involves finding a curve/surface that best partitions the data. Both techniques require training set and thus are examples of supervised learning.

Simple Linear Regression

- Two variable case: x – independent (predictor) variable, y – dependent (output variable)

  Given $\{x_i, y_i\}$, i= 1, N

  Find a_0 and a_1 to fit the following model

  $\hat{y}_i = a_0 + a_1 x_i$; i =1, N

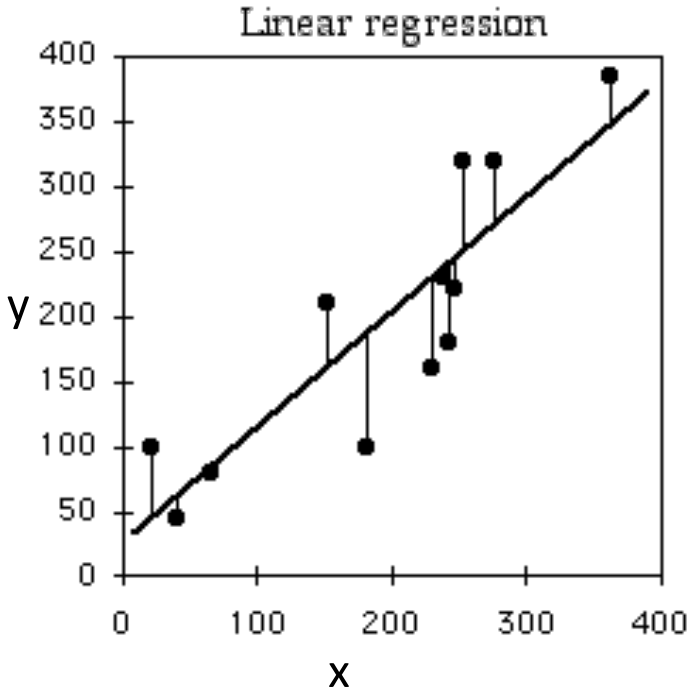
  By minimizing the prediction error, the following solution is obtained

$$Err = \sum_{i=1}^{N} (y_i - a_0 - a_1 x_i)^2$$

$$a_1 = \frac{\sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)/N}{\sum_i x_i^2 - (\sum_i x_i)^2 / N}$$

$$a_0 = \bar{y} - a_1 \bar{x}$$

Try to derive the result by taking derivatives of Err w.r.t. a_0 and a_1, set them equal to 0 and solve.



Numerical Example

Number of Employees (x)	Total contract sales (in $10,000's) y
10	13
9	11
8	12
7	10
6	8
5	6
4	5
3	3
2	4
1	3

The idea of doing regression here is to see how the sale is affected by changing the number of employees

Model $\hat{y}i = 0.79 + 1.218\ xi$

$y_i - \hat{y}_i$ Residual

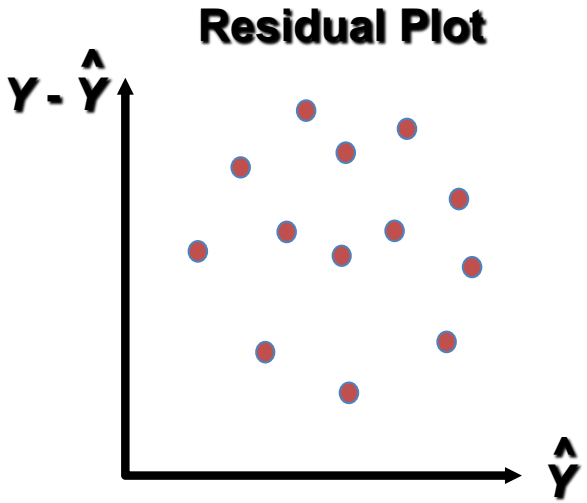
The prediction error is called the *residual sum of square error*. The *residual variance* is a measure of error fluctuation. The square root of the residual variance is called the *standard error of the estimate*.

$SSE = \Sigma(y_i - \hat{y}_i)^2$

Assessing Model Goodness

Residuals should have…

- Randomness
- Constant Variance
- Normal Distribution



SSR stands for Sum of Squares due to Regression. Measures how far are predicted values from mean of the dependent variable

$$SSR = \sum(\hat{Y}_i - \overline{Y})^2$$

$$SST = \sum(Y_i - \overline{Y})^2$$

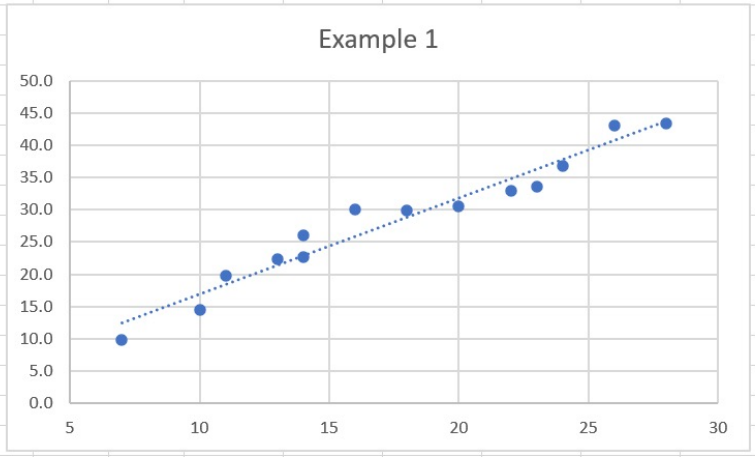
SST stands for Sum of Squares Total. Similar to variance.

$R^2 = SSR/SST$

Coefficient of Determination; it measures the proportion of variation in Y explained by (the variation in) X. A value close to 1 is considered good. Why?

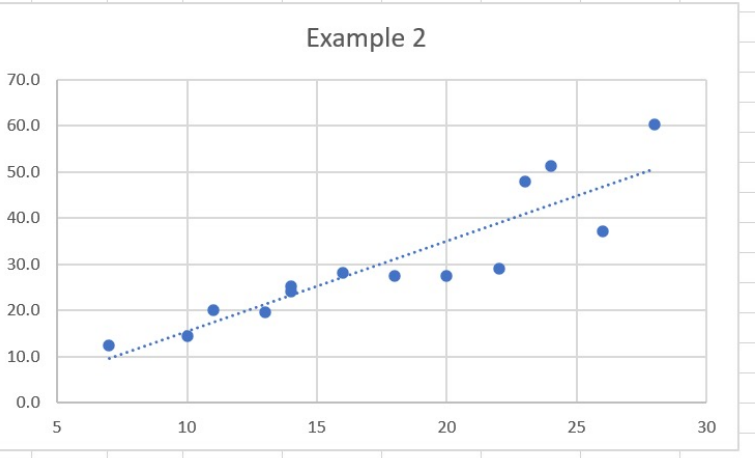
Homoskedasticity vs. Heteroskedasticity

	x	y
1		
2	x	y
3	7	9.8
4	10	14.5
5	11	19.8
6	13	22.4
7	14	26.1
8	14	22.6
9	16	30.0
10	18	29.9
11	20	30.6
12	22	32.9
13	23	33.5
14	24	36.8
15	26	43.1
16	28	43.4



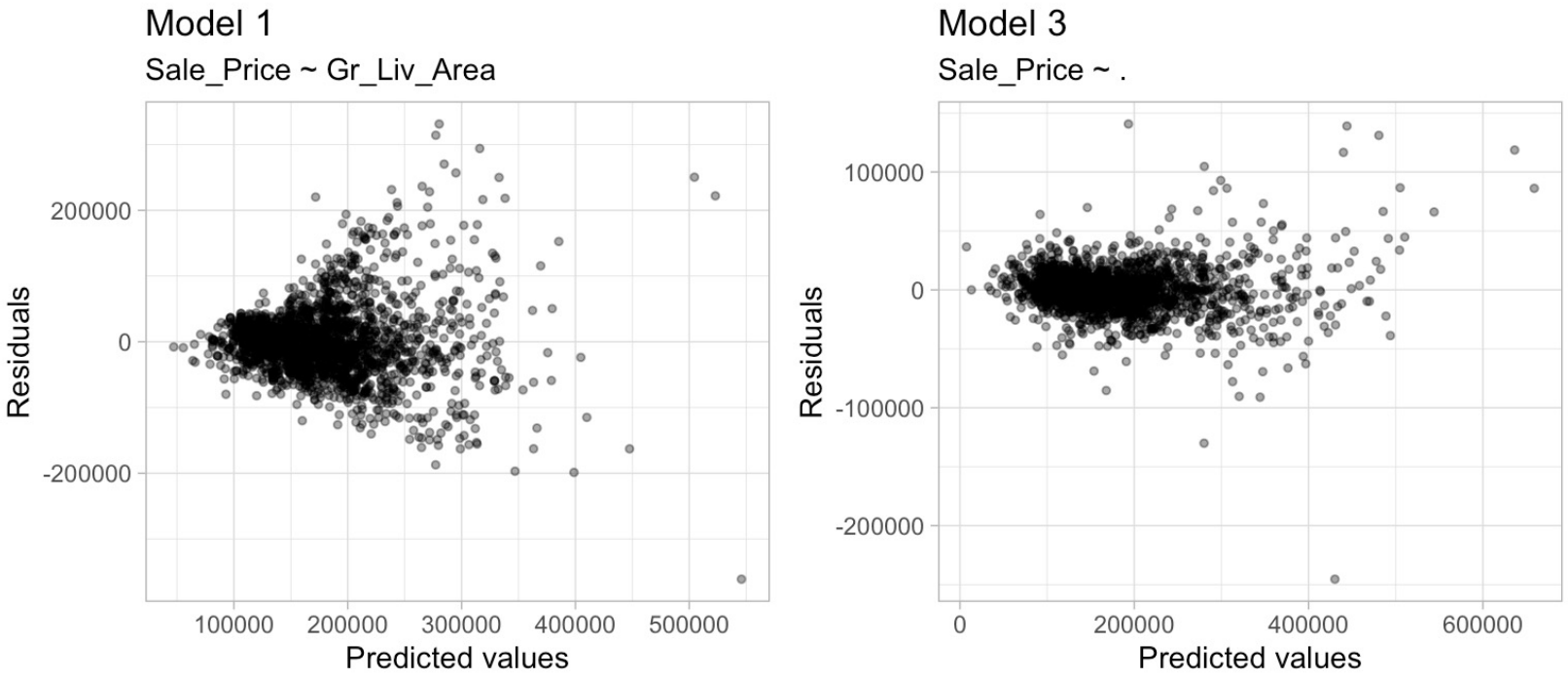
Homoskedasticity: When residuals show more or less constant variance

	x	y
17		
18	x	y
19	7	12.4
20	10	14.4
21	11	20.0
22	13	19.6
23	14	24.1
24	14	25.3
25	16	28.2
26	18	27.5
27	20	27.4
28	22	29.1
29	23	48.0
30	24	51.2
31	26	37.1
32	28	60.2



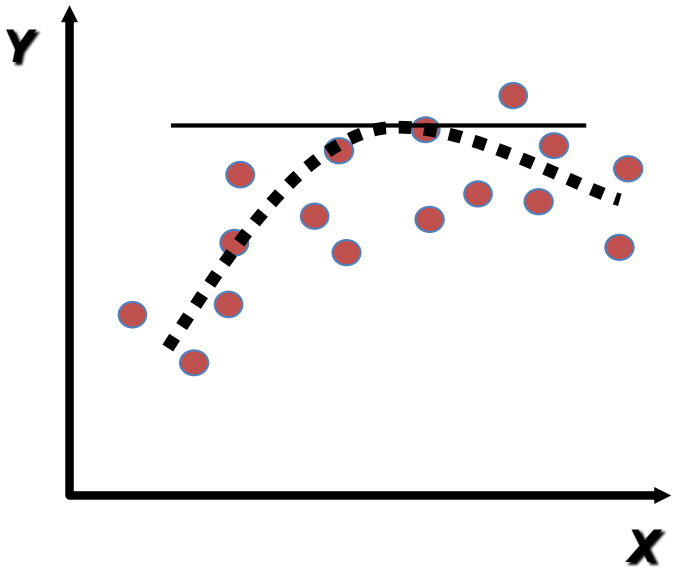
Heteroskedasticity: When residuals variance is divergent

Homoskedasticity vs. Heteroskedasticity



RHS model has additional predictors

Regression model need not be linear

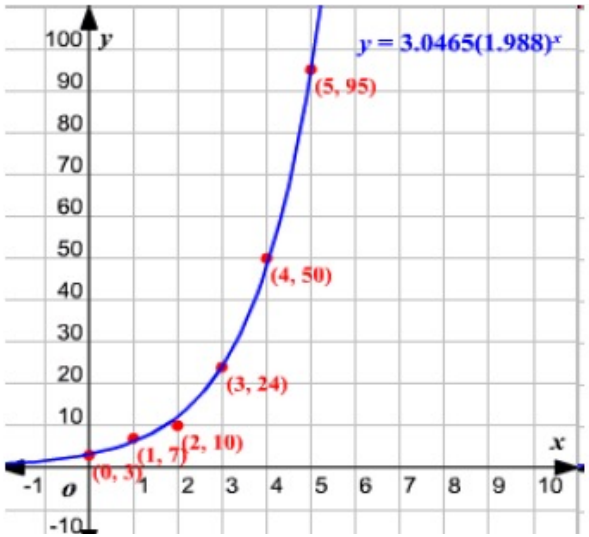


Quadratic regression model:

$$Y = b_0 + b_1 X + b_2 X^2$$

Logarithmic regression model: $Y = a + b (\ln x)$

Exponential regression model: $Y = a*b^x$



Indicator (Dummy) Variables

Dummy, or indicator, variables allow for the inclusion of qualitative variables in the model

For example: $I = \begin{cases} 1 & \text{if female} \\ 0 & \text{if male} \end{cases}$

Model with Indicator variable:

$$Y = b_0 + b_1 X + b_2 I$$

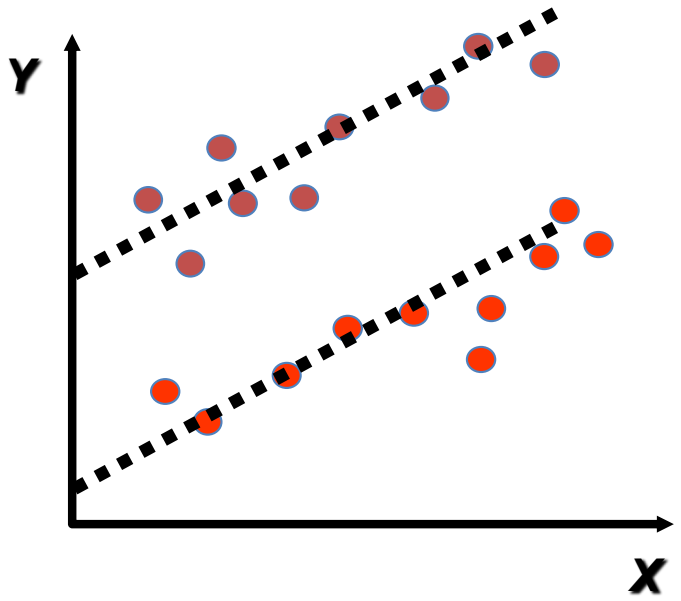
Rewrite the model as:

- For $I = 0$,

  $Y = b_0 + b_1 X$

- For $I = 1$,

$Y = (b_0 + b_2) + b_1 X$



Multiple Linear Regression

- When the number of predictor variables is more than one, then the term multiple linear regression is used
- In general, the multiple regression equation of y on x_1, x_2, …, x_k is given by:

  $y = a_0 + a_1 x_1 + a_2 x_2 + \ldots\ldots\ldots\ldots\ldots\ldots + a_k x_k$
- Representing a's and x's as vectors, and given a set of training examples, we can find **a** by minimizing

$$J(\mathbf{a}) = \left\| \mathbf{Y} - \mathbf{X}\mathbf{a} \right\|^2$$

Multiple linear regression is also known as ordinary least squares (OLS) regression.

Such a minimization can be carried out following the LMS rule discussed earlier. For small k, it can be also done using the pseudo-inverse approach

Solution Through Pseudo-inverse Matrix

$$\mathbf{Y} = \mathbf{Xa} \rightarrow \|\mathbf{Y} - \mathbf{Xa}\|^2$$

Error

Applying minimization, we get

$$2\mathbf{X}^t(\mathbf{Y} - \mathbf{Xa}) = \mathbf{0} \rightarrow \mathbf{X}^t\mathbf{Y} = \mathbf{X}^t\mathbf{Xa}$$

$$\mathbf{a} = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{Y} = \mathbf{X}^*\mathbf{Y}$$

$$\mathbf{X}^* = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t \leftarrow$$ Psuedo-inverse

Example: Predicting the Number of Credit Cards

- The table on right shows a small part of a credit card company's database
- Build a model to predict the number of credit cards held by a customer

# of CCs	Family Size (x1)	Income (x2)
1	4	2	14
2	6	2	16
3	6	4	14
4	7	4	17
5	8	5	18
6	7	5	21
7	8	6	17
8	10	6	25

Predicted # of Credit Cards = $a0 + a1x1 + a2x2$

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 14 \\ 1 & 2 & 16 \\ 1 & 4 & 14 \\ 1 & 4 & 17 \\ 1 & 5 & 18 \\ 1 & 5 & 21 \\ 1 & 6 & 17 \\ 1 & 6 & 25 \end{bmatrix}$$

$$\mathbf{X}^t = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 4 & 4 & 5 & 5 & 6 & 6 \\ 14 & 16 & 14 & 17 & 18 & 21 & 17 & 25 \end{bmatrix}$$

$$(\mathbf{X}^t\mathbf{X}) = \begin{bmatrix} 8 & 34 & 142 \\ 34 & 162 & 631 \\ 142 & 631 & 2616 \end{bmatrix}$$

$$(\mathbf{X}^t\mathbf{X})^{-1} = \begin{bmatrix} 3.5015 & 0.0899 & -0.2117 \\ 0.0899 & 0.1044 & -0.0301 \\ -0.2117 & -0.0301 & 0.0191 \end{bmatrix}$$

$$\mathbf{X}^* = \begin{bmatrix} 0.7168 & 0.2933 & 0.8966 & 0.2613 & 0.1395 & -0.4958 & 0.4411 & -1.2529 \\ -0.1221 & -0.1822 & 0.0866 & -0.0036 & 0.0708 & -0.0194 & 0.2052 & -0.0352 \\ -0.0041 & 0.0342 & -0.0642 & -0.0068 & -0.0178 & 0.0396 & -0.0669 & 0.0861 \end{bmatrix}$$

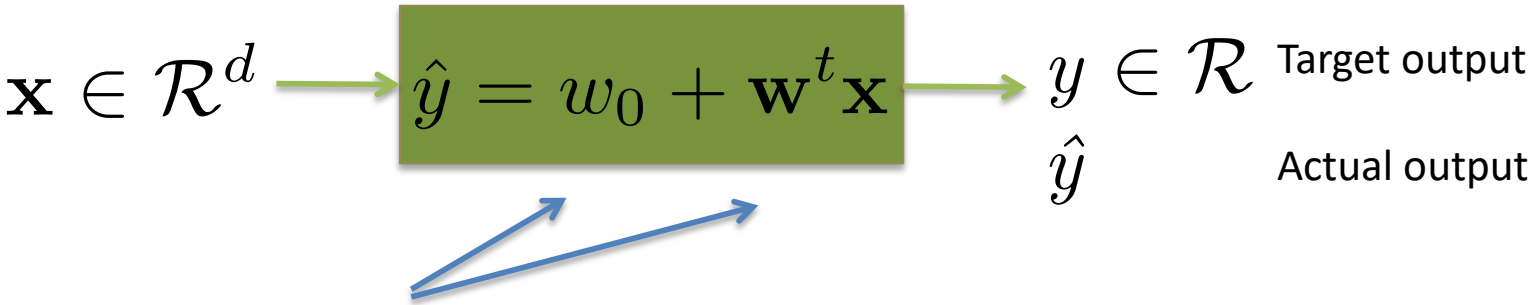
$Y = [4\ 6\ 6\ 7\ 8\ 7\ 8\ 10]t$

$$\mathbf{a} = \mathbf{X}^*\mathbf{Y} = \begin{matrix} 0.4817 \\ 0.6322 \\ 0.2158 \end{matrix}$$

$\hat{Y} = [4.76\ 5.19\ 6.03\ 6.68\ 7.52\ 8.17\ 7.94\ 9.67]^t$

Predicted number of CCs.

Linear Regression Model Recapped



Find the model parameters using *n* pairs of examples:

$$\mathbf{x}_i, y_i, i = 1, n$$

Model:

$$\hat{y} = w_0 + \mathbf{w}^t \mathbf{x}$$

Model Parameters:

$$w_0, \mathbf{w}$$

Cost Function:

How do we choose model parameters?

By defining a cost function and selecting those parameter values that minimize the cost function

We will perform minimization using gradient descent

$$J(w_0, \mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} (y_i - (w_0 + \mathbf{w}^t \mathbf{x}_i))^2$$

$$J(w_0, \mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - (w_0 + \mathbf{w}^t\mathbf{x}_i))^2$$

$$\frac{\partial J}{\partial w_0} = -\sum_{i=1}^{n}(y_i - (w_0 + \mathbf{w}^t\mathbf{x}_i))$$

$$\frac{\partial J}{\partial \mathbf{w}} = -\sum_{i=1}^{n}(y_i - (w_0 + \mathbf{w}^t\mathbf{x}_i))\mathbf{x}_i$$

We can now write the updating rule. To simplify writing equations, the bias term is absorbed in weight/parameter vector by appending 1 to X vector. We will assume this too for convenience. With this understanding, we have

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^t\mathbf{x}_i)^2$$

$$\frac{\partial J}{\partial \mathbf{w}} = -\sum_{i=1}^{n}(y_i - \mathbf{w}^t\mathbf{x}_i)\mathbf{x}_i$$

Updating using all examples is known as *batch gradient descent*

LMS Rule

- Lets do solution updating after every training example. In that case:

$$\frac{\partial J_i}{\partial \mathbf{w}} = -(y_i - \mathbf{w}^t \mathbf{x}_i)\mathbf{x}_i$$

- Thus, updating rule becomes

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \alpha(y_i - \mathbf{w}_{old}^t \mathbf{x}_i)\mathbf{x}_i$$

This updating rule is also known delta or *Widrow-Hoff rule. We saw this rule earlier also.*

```
# This example does linear regression for a problem with 2 independent and one dependent variables.
N = 32
alpha = 1.3
beta = np.array([0.5, 1.9])
```

True model parameter Values

```
x0 = np.ones((N,1))
x_data = np.random.randn(N,2)
X = np.hstack((x0,x_data))# Absorbs contant term of 1
Y = x_data.dot(beta) + alpha + 0.1*np.random.randn(1)
```

Data is being generated from a Gaussian distribution

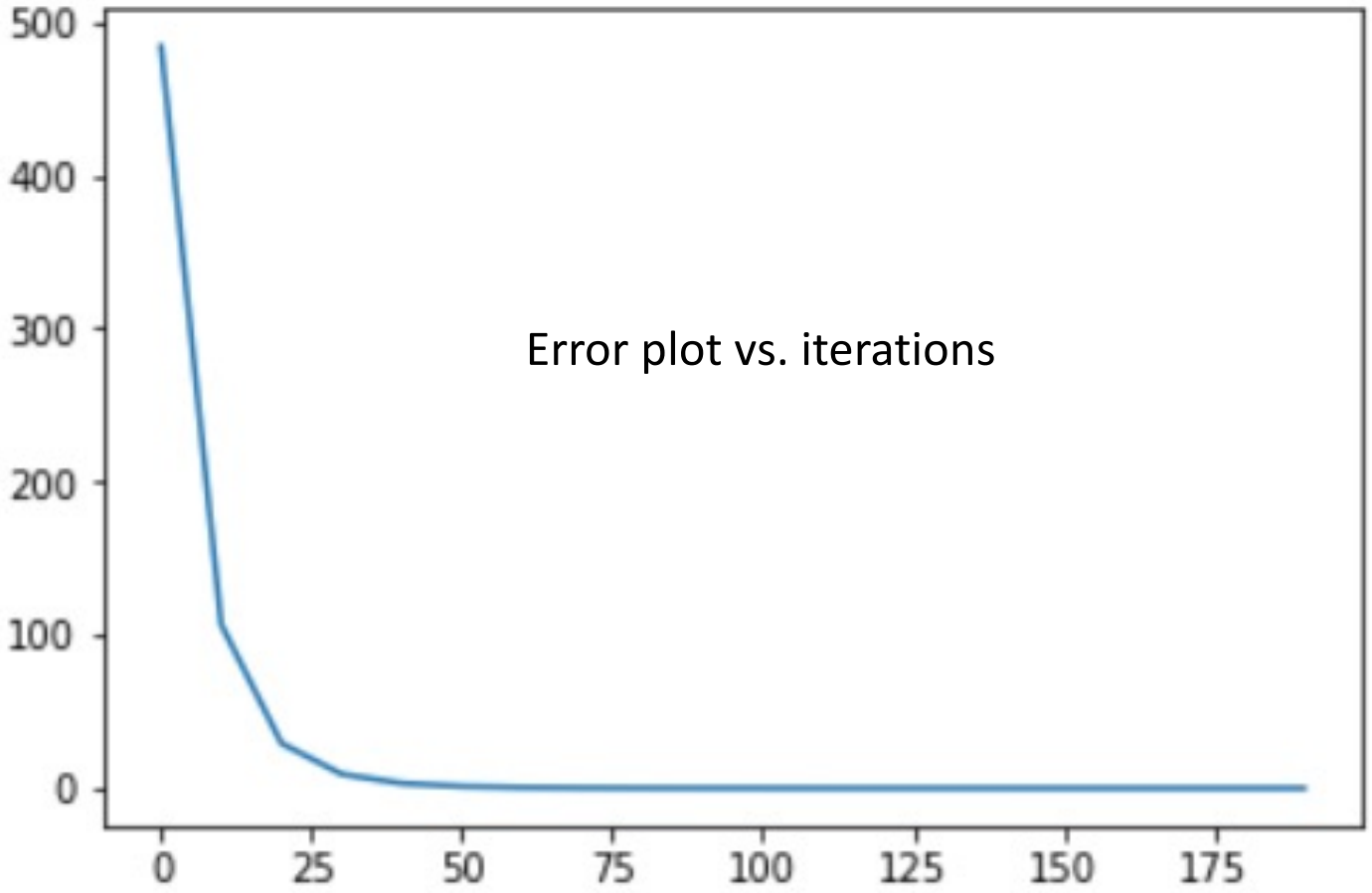
```
w = np.random.randn(3)# Initialize weight vector
learning_rate = 1e-3
result =[]
loss = 0
for t in range(200):
    # Forward pass: compute predicted y
    y_pred = X.dot(w)
    # Compute and print loss
    loss = np.square(y_pred - Y).sum()
    if t % 10 == 0:
        print(t, loss)
        result.append(loss)
  # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - Y)
    grad_w = X.T.dot(grad_y_pred)

    # Update weights
    w -= learning_rate * grad_w
```

```
print(w)
```

```
[1.25914417 0.50003894 1.89945083]
```

Learned model parameters

```
plt.plot(np.arange(0,200,10),result)
plt.show()
```



Error plot vs. iterations

 # Predict the power output of a plant based on historical readings

Data is in Excel file. After reading, print first two rows

```
#Read an Excel worksheet
file = '/Users/isethi/Downloads/CCPP/ccpp.xlsx'# Select the file
xl = pd.ExcelFile(file)#This loads the file
df = xl.parse('Sheet1')# Loads sheet1 as dataframeprint(df.shape)
print(df[0:2])
```

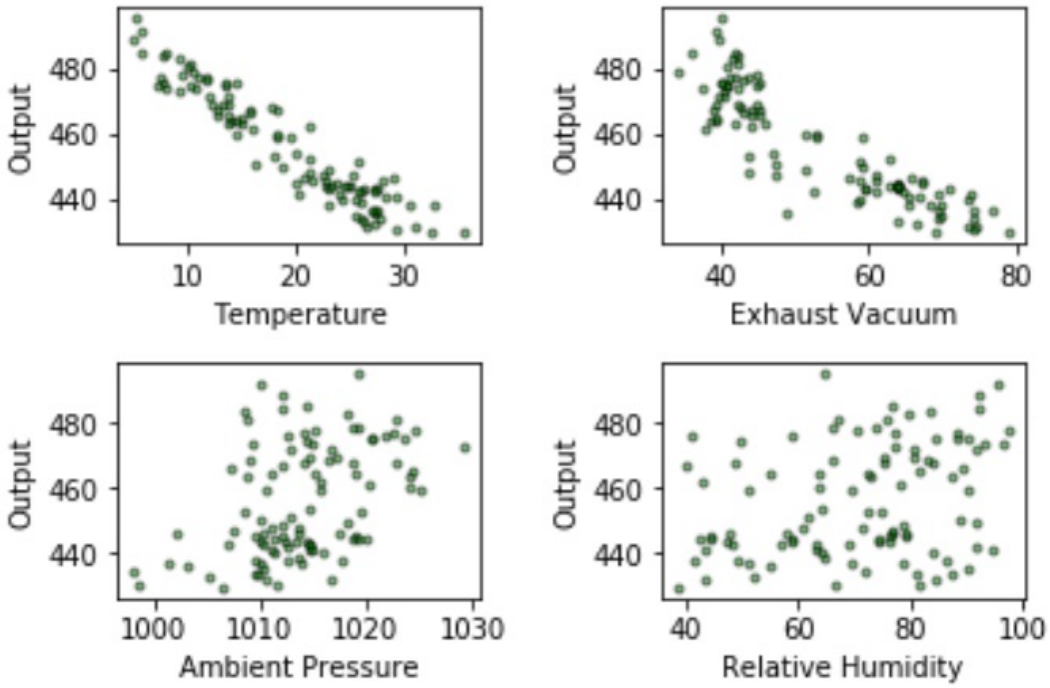
```
      AT      V       AP     RH      PE
0  14.96  41.76  1024.07  73.17  463.26
1  25.18  62.96  1020.04  59.08  444.37
```

```
#Correlation matrix
corr = df[0:100].corr()
print(corr)
```

```
       AT      V      AP      RH      PE
AT  1.000  0.863 -0.448 -0.486 -0.950
V   0.863  1.000 -0.368 -0.236 -0.881
AP -0.448 -0.368  1.000 -0.012  0.432
RH -0.486 -0.236 -0.012  1.000  0.306
PE -0.950 -0.881  0.432  0.306  1.000
```

Visualize each independent variable wrt output variable

```
data = np.array(df[0:100])
X = data[:,:-1]
Y = data[:,-1]
heading = ['Temperature', 'Exhaust Vacuum', 'Ambient Pressure', 'Relative Humidity']
for i in range(4):
    plt.subplot(2,2,i+1)
    plt.scatter(X[:, i], Y,c = 'g',marker = ".",edgecolor='k', alpha = 0.50)
    plt.xlabel(heading[i])
    plt.ylabel('Output')
plt.subplots_adjust(wspace=0.50, hspace=0.50)
plt.show()
```



Lets divide the data into training and test sets

```
split = int(0.7*X.shape[0])
print (split)
```

```
70
```

```
X_train = X[:split,:]
Y_train = Y[:split]
X_test = X[split:,:]
Y_test = Y[split:]
```

```
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
regr.fit(X_train, Y_train)
Y_pred = regr.predict(X_test)
```

```
print('Coefficients: \n', regr.coef_)
print('Intercept: \n', regr.intercept_)
mse = np.mean((Y_test-Y_pred)**2)
print('mse: \n',mse)
```

```
Coefficients:
 [-2.32024685 -0.0303063  -0.13458604 -0.2022868 ]
Intercept:
 653.7252298935637
mse:
 18.431693401310625
```

```
print('regr.Score: \n',regr.score(X_test,Y_test))# Score closed to 1 is a perfect fit;
#close to 0 means no linear relationship
```

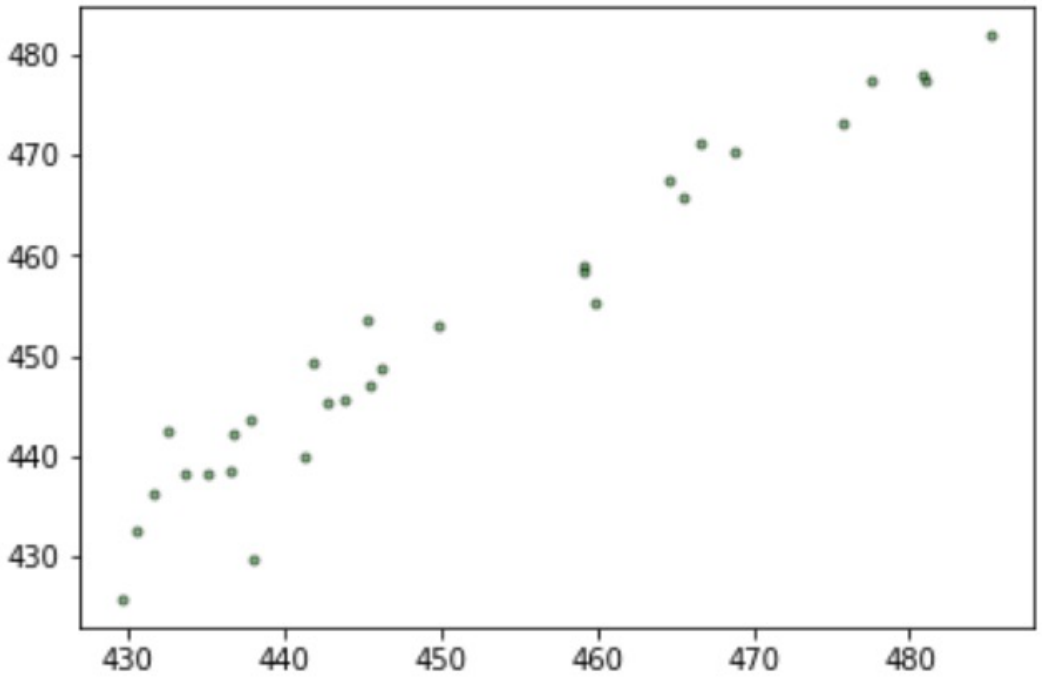
```
regr.Score:
 0.9365276088114
```

This score close to 1 means a good fit

```
from sklearn.model_selection import cross_val_score
np.mean(cross_val_score(LinearRegression(), X_train, Y_train, cv=10))
```

```
0.8938592147745295
```

```
plt.scatter(Y_test, Y_pred,c = 'g',marker = ".",edgecolor='k', alpha = 0.50)
plt.show()
```



Subset Selection in Regression

- Should we use all features or a subset in our multiple regression model?
  - Same problem as that of feature selection
    - Backward and forward stepwise selection methods
    - PCA
    - Shrinkage methods

Shrinkage Methods

- Basic idea?
  - Use additional constraints to reduce (shrink) coefficients
  - Since shrinking is a continuous operator compared to stepwise selection, better regression results with less variability are obtained
  - Ridge Regression
    - Minimize least squares subject to:

$$\sum_{j=1}^{k} a_j^2 \leq s$$

a_i : model parameters

  - The LASSO (Least Absolute Shrinkage & Selection Operator)
    - Minimize least squares subject to:

$$\sum_{j=1}^{k} | a_j | \leq s$$

http://statweb.stanford.edu/~tibs/lasso.html

Shrinkage Methods

- Also help in avoiding overfitting
- The term *regularization* is also used in many instances

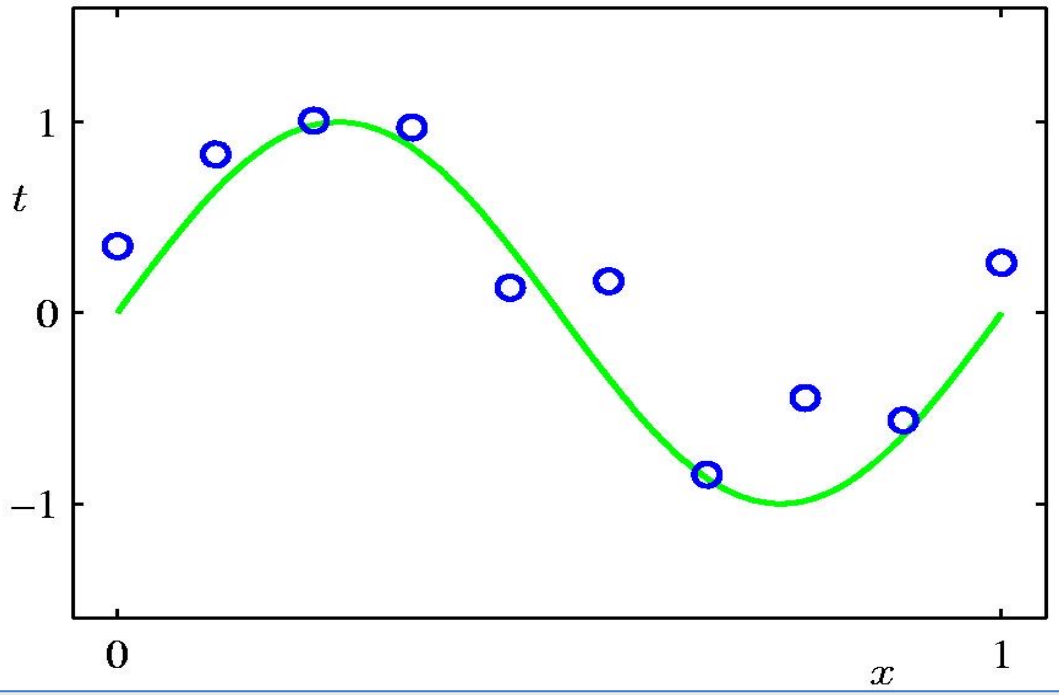
A closed form solution exists for ridge regression

In Lasso, the solution is obtained via quadratic programming

The lasso does better when the response is a function of only a relatively small number of predictors

An Example Illustrating Use of a Shrinkage Method

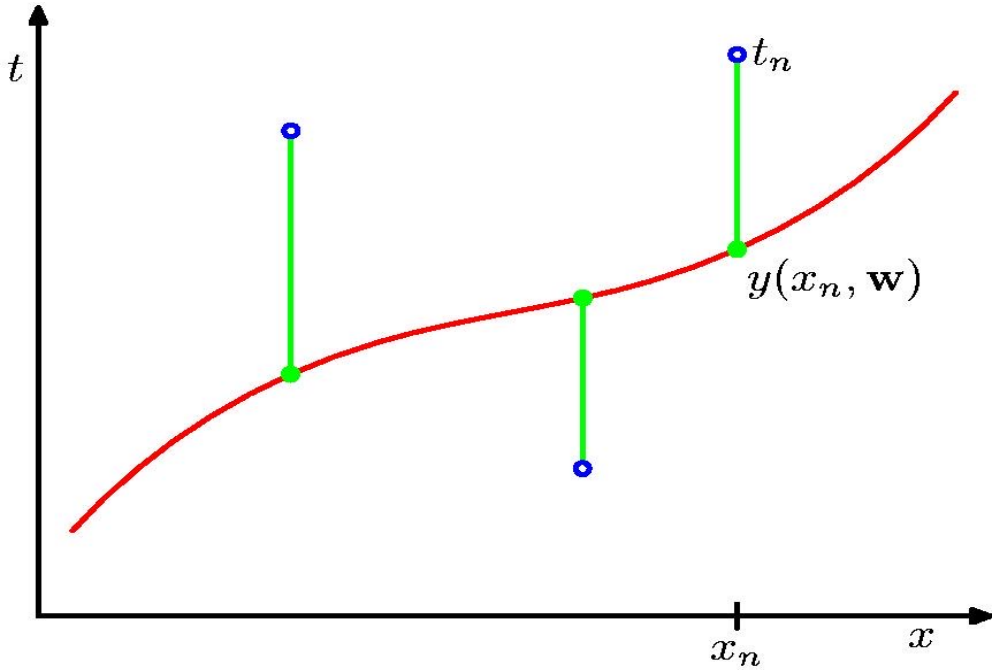
We are given 10 (x,y) pair of points to fit a regression model



Model

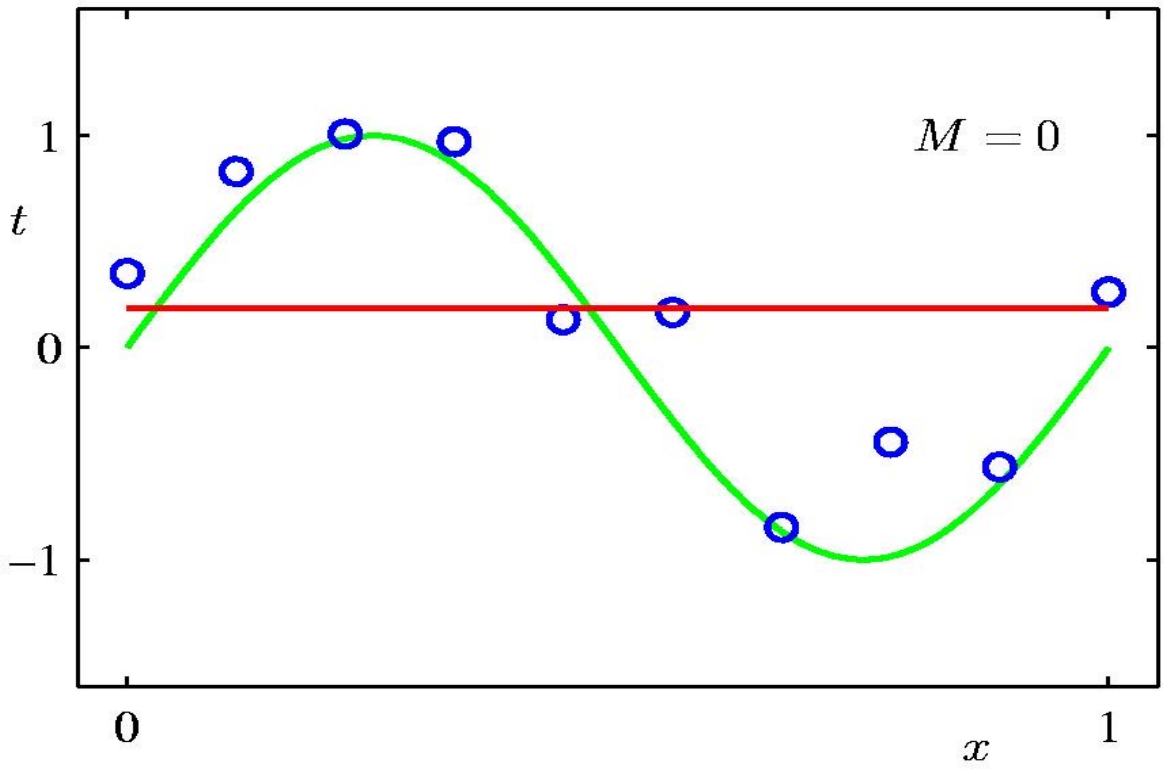
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Sum-of-Squares Error Function

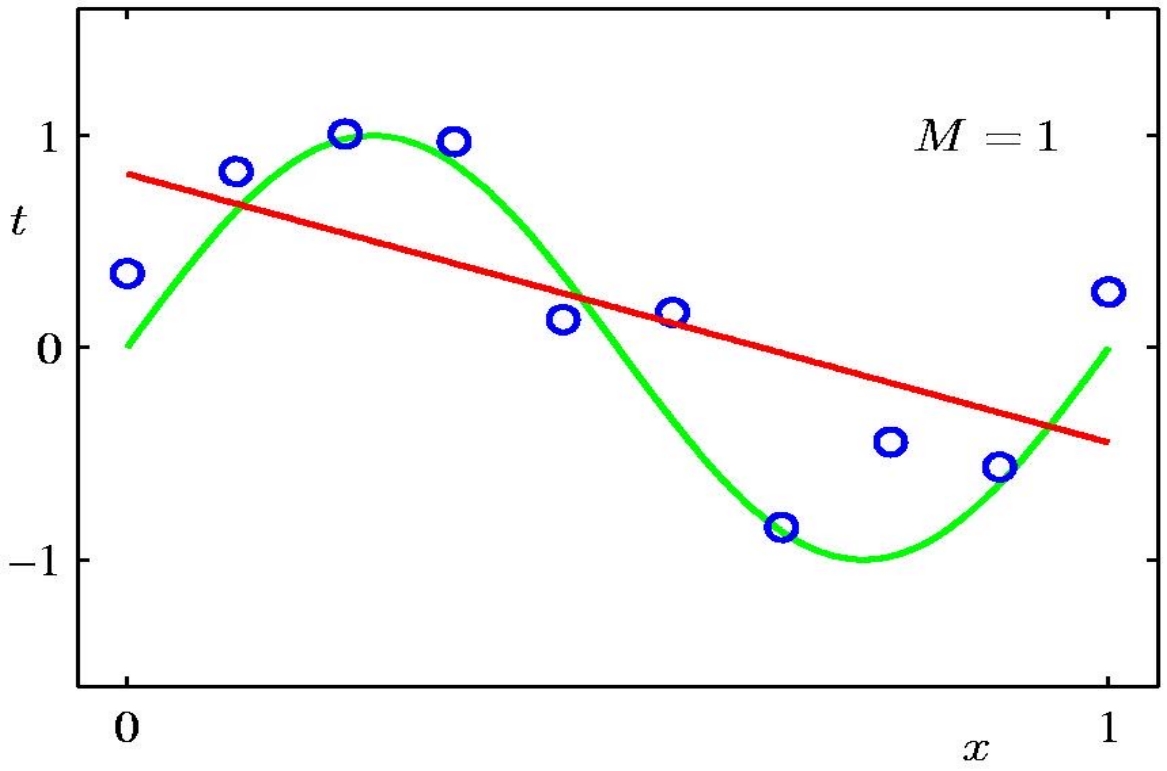


$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

0^{th} Order Polynomial Fit

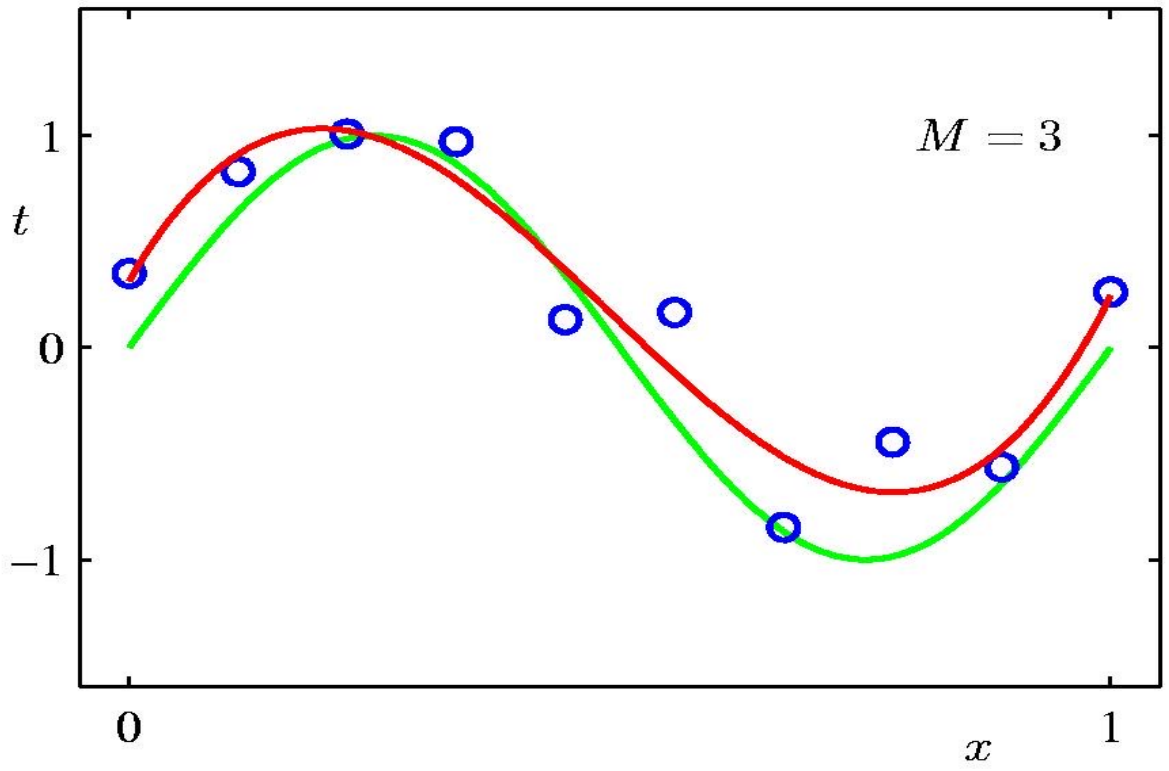


1st Order Polynomial Fit

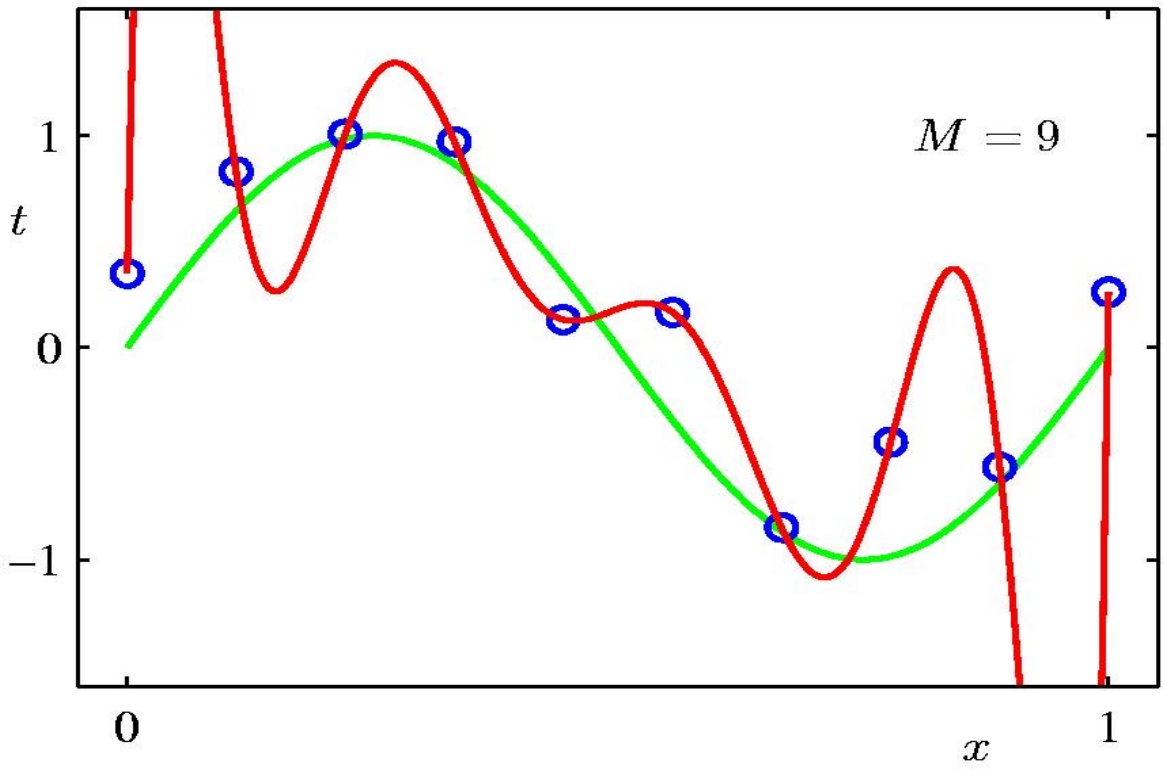


3rd Order Polynomial Fit

Looks like a good fit



9th Order Polynomial Fit



Zero error fit. Is this the best model for this problem?

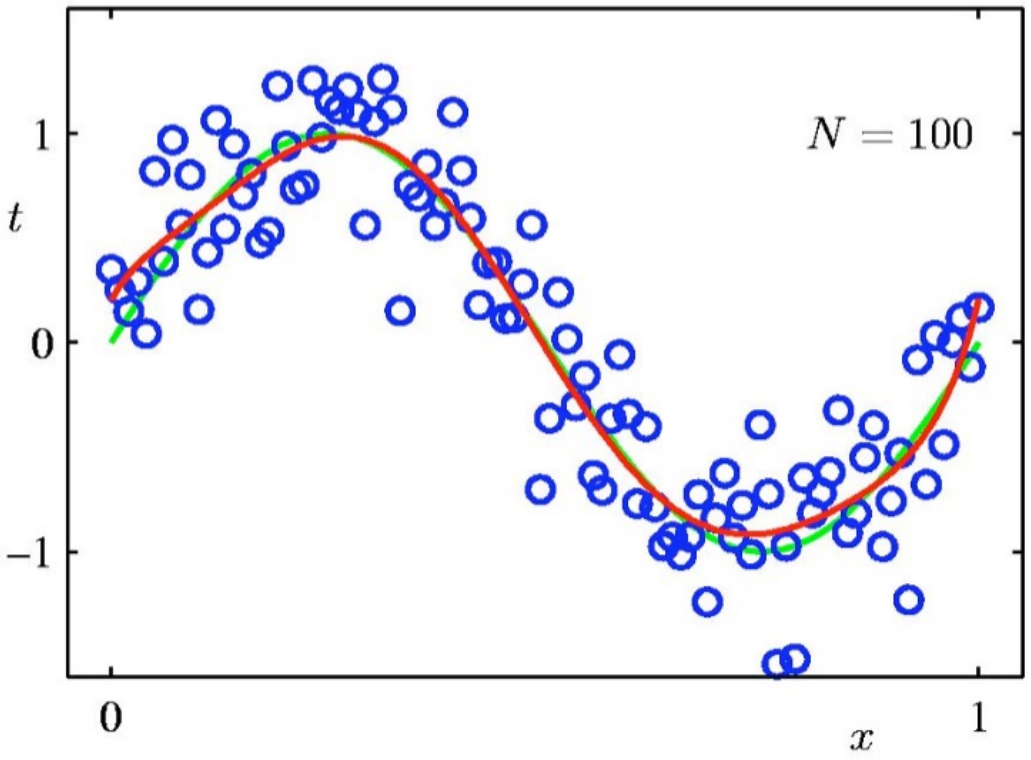
Polynomial Coefficients

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

These are the values of the fitted coefficients. Note extremely high values for the 9-th order model

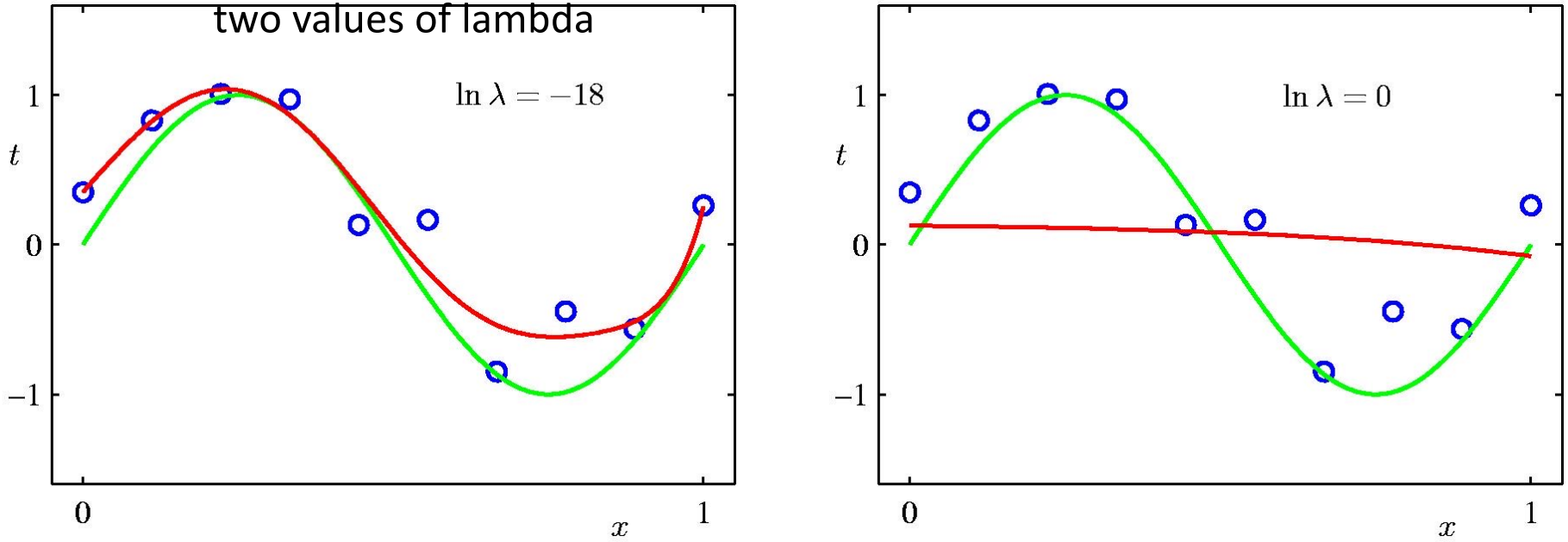
Data Set Size: $N = 100$

9th Order Polynomial

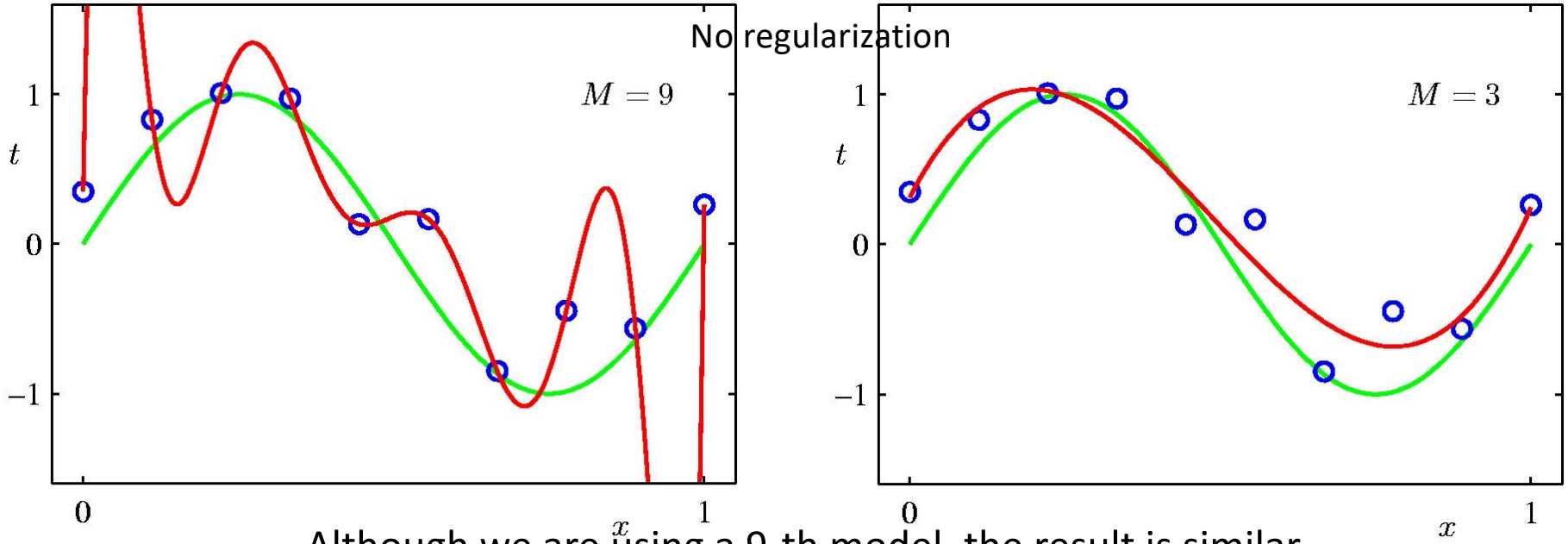


Overfitting reduces with 100 pairs of (x,y) values

9-th order model fitting with regularization/ridge regression for two values of lambda



No regularization



Although we are using a 9-th model, the result is similar to the 3-rd order model without shrinkage

Polynomial Coefficients

Shrinkage effect on coefficients

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Lasso (Least Absolute Shrinkage & Selection Operator Demo

```
from sklearn import datasets
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
# Load the Boston Data Set
bh = datasets.load_boston()
X = bh.data
y = bh.target
# Create training and test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Create an instance of Lasso Regression implementation
lasso = Lasso(alpha=1.0)
# Fit the Lasso model
lasso.fit(X_train, y_train)
# Create the model score
#
lasso.score(X_test, y_test), lasso.score(X_train, y_train)

(0.655906082915434, 0.6899591642958296)

# Let's look at the model coeff.
lasso.coef_

array([-0.09191567,  0.03466685, -0.        ,  0.        , -0.        ,
        1.28131738,  0.01143974, -0.61602099,  0.19150077, -0.00954028,
       -0.70335005,  0.01083224, -0.77992069])

# Let's repeat with different regularization parameter alpha. Lower alpha means less regularization
lasso = Lasso(alpha=0.1)
# Fit the Lasso model
lasso.fit(X_train, y_train)
# Create the model score
lasso.score(X_test, y_test), lasso.score(X_train, y_train)
(0.6918147952283056, 0.7335268365408683)
lasso.coef_
array([-1.22854422e-01,  3.86385849e-02, -4.97188385e-04,  1.48353759e+00,
       -0.00000000e+00,  3.89367421e+00, -1.96048179e-02, -1.10561397e+00,
        2.16673479e-01, -1.05844004e-02, -7.44014516e-01,  1.23959059e-02,
       -6.02281637e-01])
```

Results using Linear Regression

```
# Let's apply linear regression
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
regr.fit(X_train, y_train)
regr.score(X_test, y_test), regr.score(X_train, y_train)
```

```
(0.7112260057484943, 0.7434997532004697)
```

```
regr.coef_
```

```
array([-1.33470103e-01,  3.58089136e-02,  4.95226452e-02,  3.11983512e+00,
       -1.54170609e+01,  4.05719923e+00, -1.08208352e-02, -1.38599824e+00,
        2.42727340e-01, -8.70223437e-03, -9.10685208e-01,  1.17941159e-02,
       -5.47113313e-01])
```

Ridge Regression Results

```
# Let's apply Ridge Regression
ridgeR = Ridge(alpha=1.0)
ridgeR.fit(X_train,y_train)
ridgeR.score(X_test, y_test),ridgeR.score(X_train,y_train)
```

```
(0.7041586727559436, 0.7415671063241829)
```

```
ridgeR.coef_
```

```
array([-0.1284272 ,  0.03695233,  0.01791436,  2.93269454, -7.84806046,
        4.06357438, -0.01724174, -1.27176091,  0.22549398, -0.00938149,
       -0.82710453,  0.01198771, -0.56347377])
```

Logistic Regression

- Let's consider the following model used in linear regression. But in this case the dependent/outcome variable (y) is a discrete variable (*good/bad customer*)

$$y = a_0 + a_1 x_1 + a_2 x_2 + \ldots\ldots\ldots\ldots\ldots\ldots + a_k x_k$$

- Let's look at the following expression

$$p = \frac{e^{(a_0 + a_1 x_1 \ldots + a_k x_k)}}{1 + e^{(a_0 + a_1 x_1 \ldots + a_k x_k)}}$$

- The quantity *p* will always lie in the range 0-1 and thus can be interpreted as the probability of outcome *y* being good or bad.

Logistic regression is a classification model

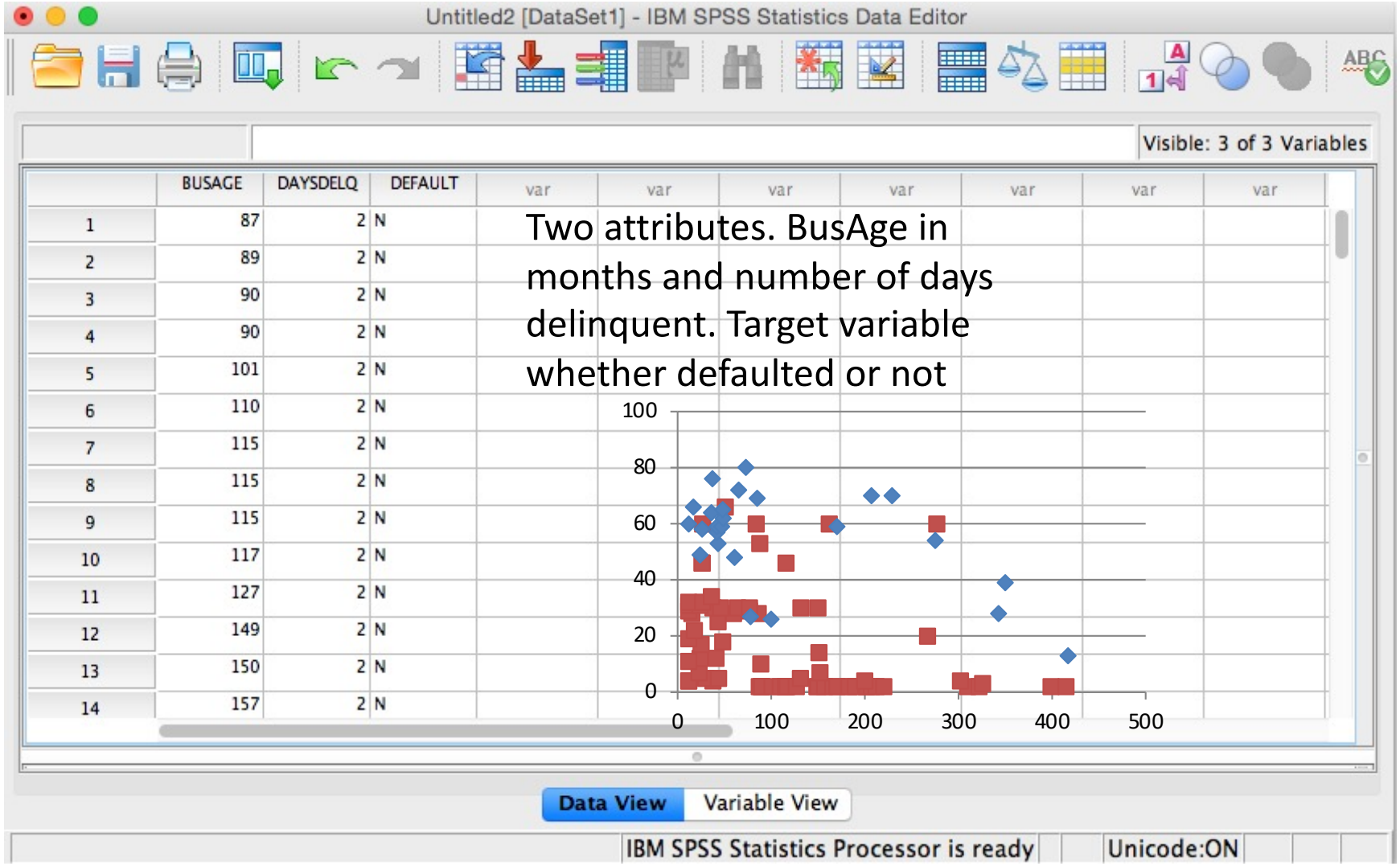
Logistic Regression

- By simple rewriting, we get:

$$log(p/(1-p)) = a_0 + a_1x_1 + a_2x_2 + \cdots + a_kx_k$$

- This ratio is called *log odds*
- The parameters of the logistic model are determined using the likelihood maximization-based cost function optimized via gradient search

Credit Default Example



Credit Scoring Example

Classification Table[a]

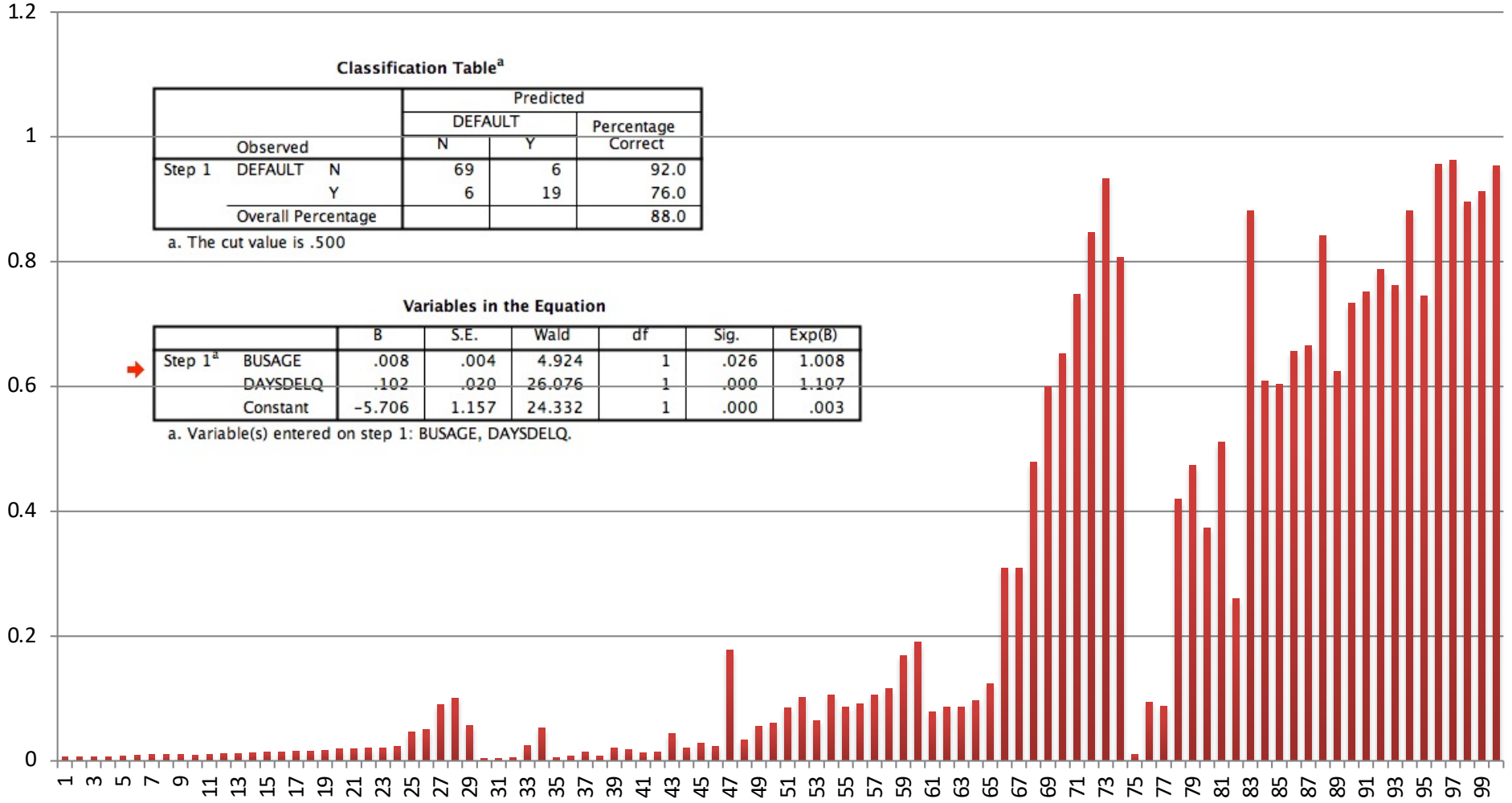
Observed			Predicted DEFAULT N	Predicted DEFAULT Y	Percentage Correct
Step 1	DEFAULT	N	69	6	92.0
		Y	6	19	76.0
	Overall Percentage				88.0

a. The cut value is .500

Variables in the Equation

		B	S.E.	Wald	df	Sig.	Exp(B)
Step 1[a]	BUSAGE	.008	.004	4.924	1	.026	1.008
	DAYSDELQ	.102	.020	26.076	1	.000	1.107
	Constant	-5.706	1.157	24.332	1	.000	.003

a. Variable(s) entered on step 1: BUSAGE, DAYSDELQ.



Using the Model

- What is the probability of a business defaulting given that business has been with the bank for 26 months and is delinquent for 58 days?

BUSAGE: 0.008; DAYSDELQ: 0.102; Intercept: -5.076

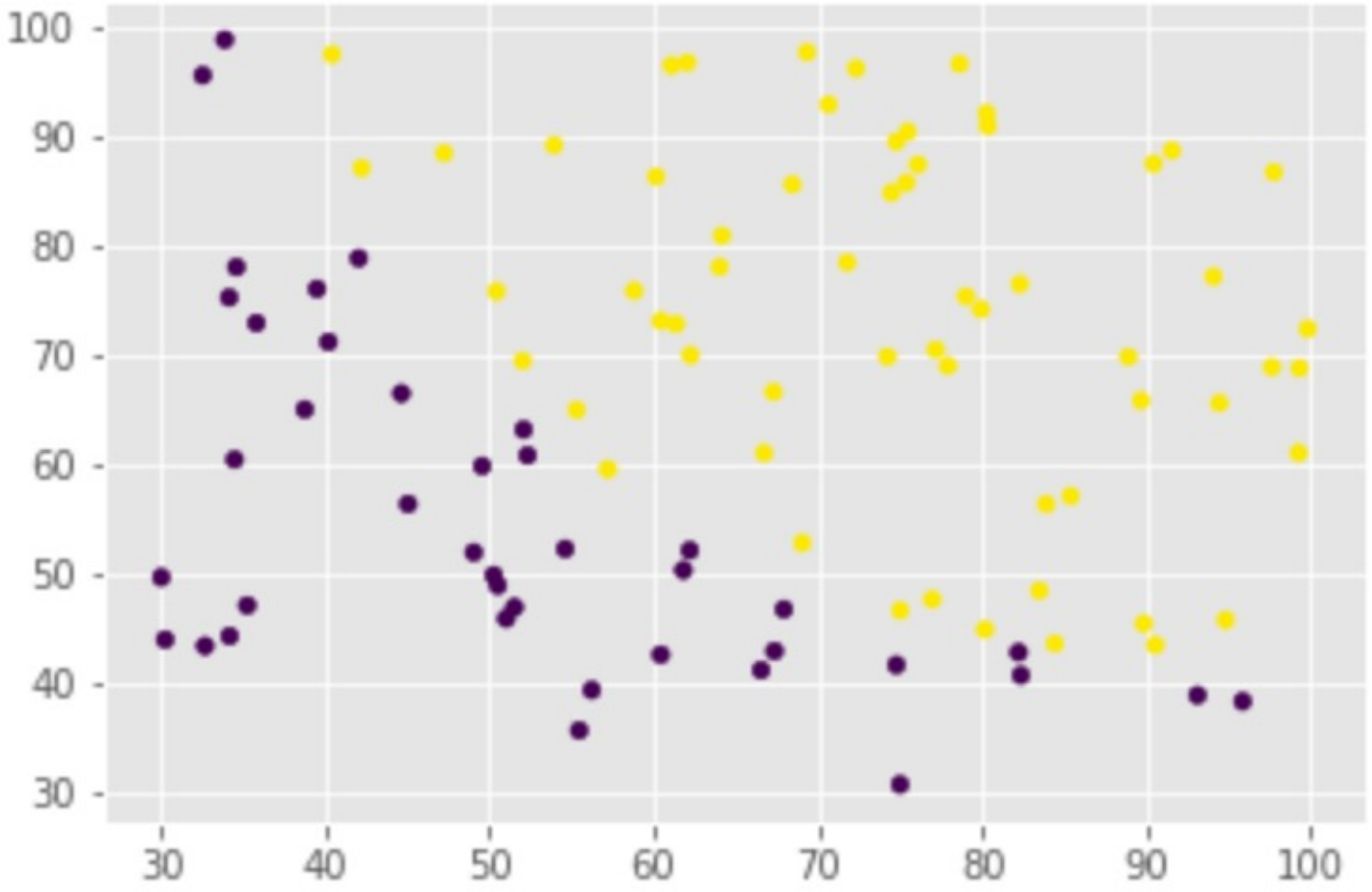
Plug the model parameters to calculate *p*

$$e^{0.008*26+0.102*58-5.706}/(1+e^{0.008*26+0.102*58-5.706})$$

0.603

Python Example using 2-D Data

```
plt.scatter(data[:,0],data[:,1],c=data[:,2],s=20)
plt.show()
```



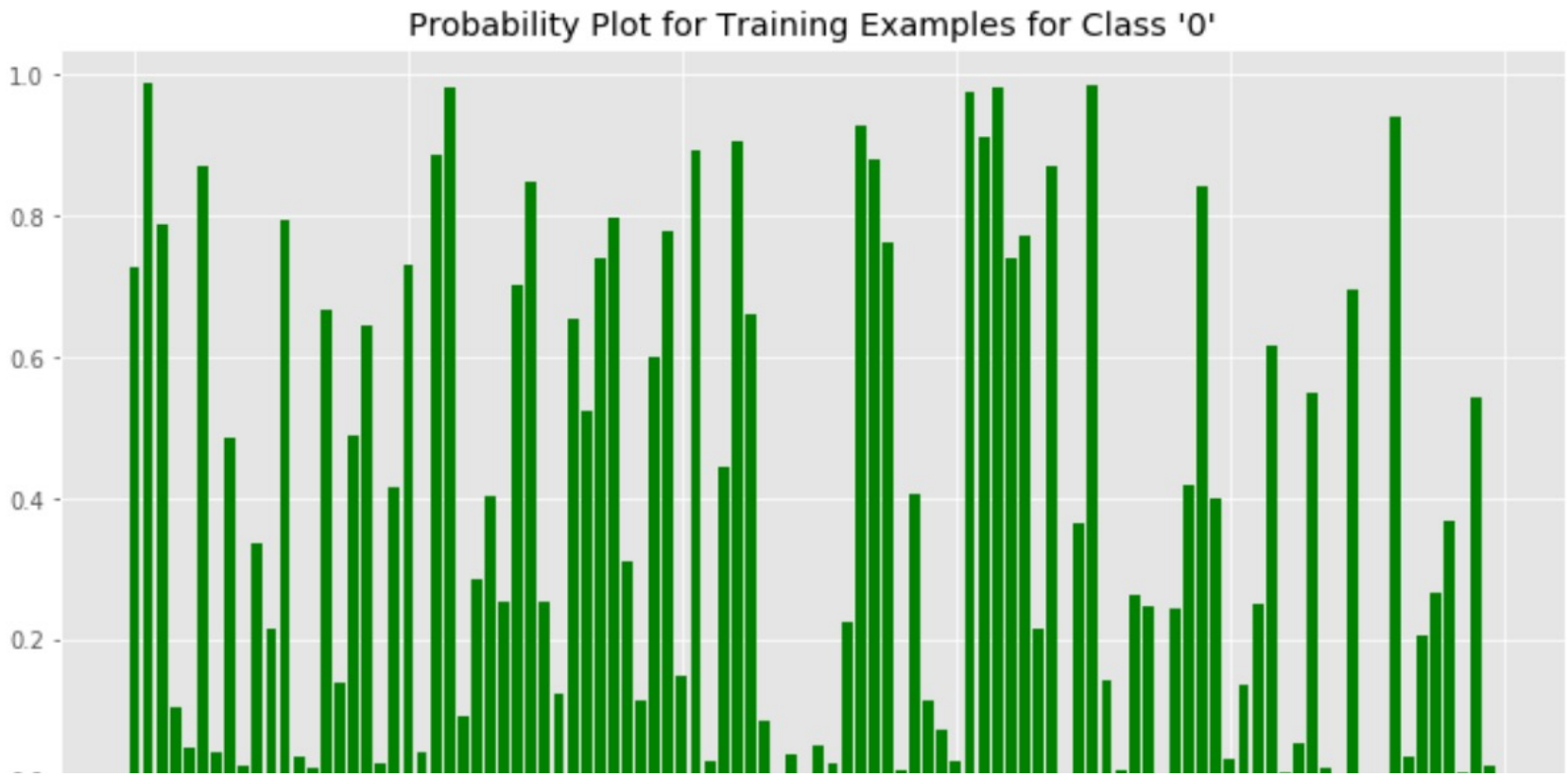
```
X = data[:,0:2]
y = data[:,2]
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=10).fit(X, y)# C controls regularization
```

```
print(clf.score(X,y))#This gives the mean accuracy.
```

```
0.91
```

```
prob = clf.predict_proba(X)
```

```
plt.figure(figsize=(12,6))
h = np.arange(100)
plt.bar(h,prob[:,0],color='g')
plt.title("Probability Plot for Training Examples for Class '0'")
plt.show()
```




```
#Print weights
w = clf.coef_[0]
print(w)
```

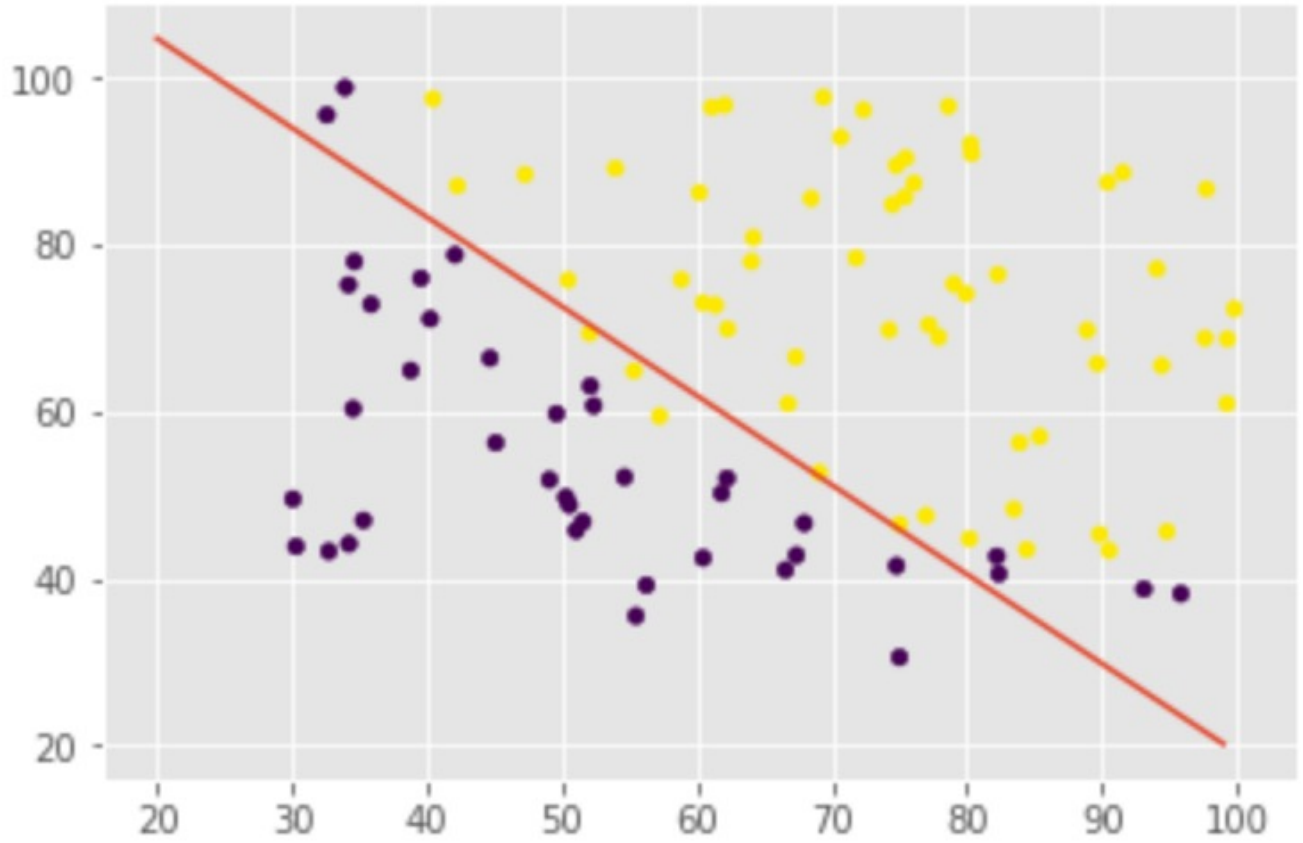
```
[0.09432673 0.08817996]
```

```
#Print intercept value
w0 = clf.intercept_
print(w0)
```

```
[-11.12335114]
```

```
xp = np.arange(100)
yp = -w[0]*xp/w[1]-w0/w[1]
```

```
plt.plot(xp[20:100],yp[20:100])
plt.scatter(data[:,0],data[:,1],c=data[:,2],s=20)
plt.show()
```



Logistic regression is identical to a single neuron classifier with sigmoidal nonlinearity

Example

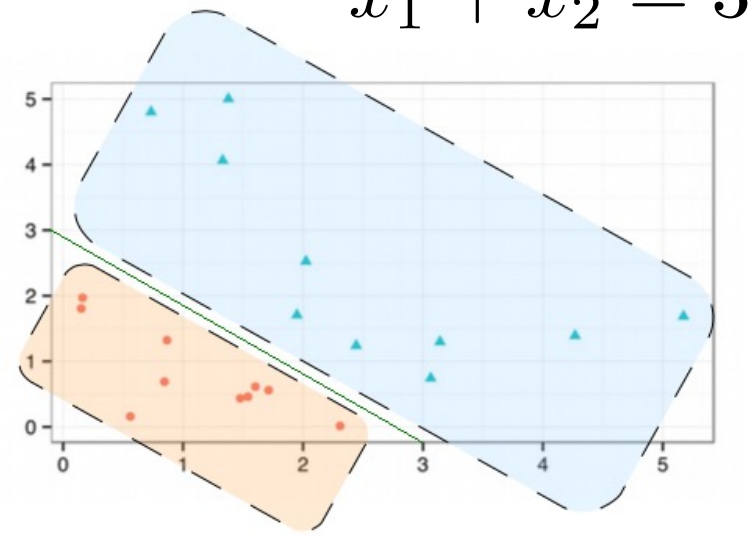
$$\mathbf{w} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Coefficients in a logistic regression model

What will be the prediction, 0 or 1, of the logistic regression model for an input $[2\ 5]^T$?

How does the decision boundary look like in x_1-x_2 space?

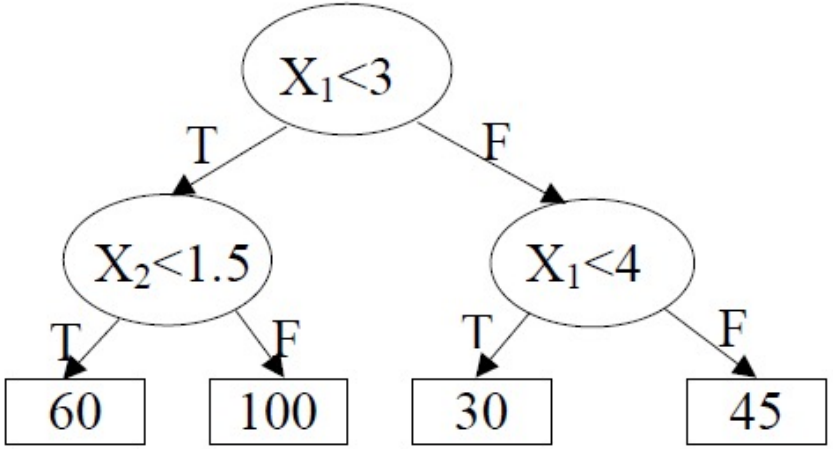
$$x_1 + x_2 = 3$$



Non-linear Regression

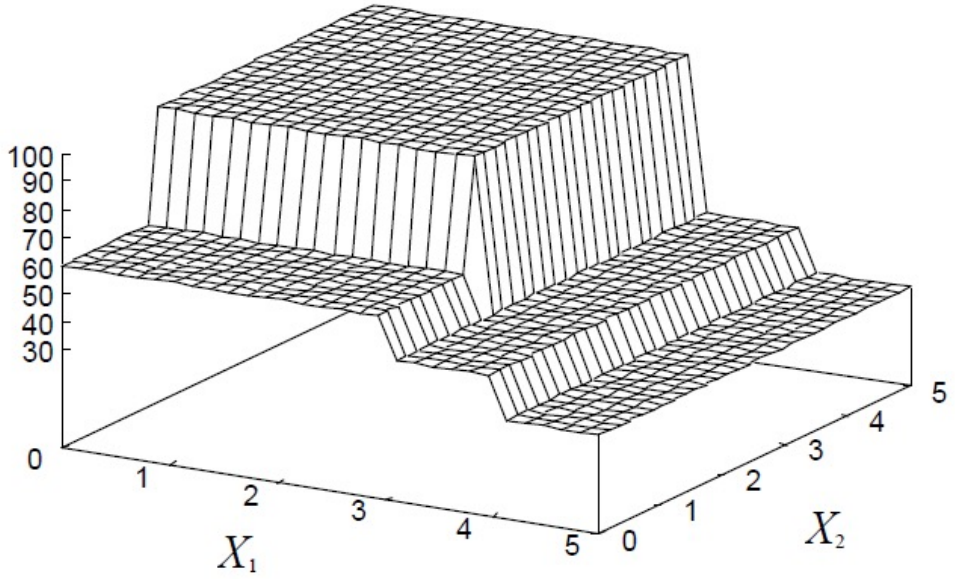
- How can we perform nonlinear regression?
- We can use some of the non-parametric methods that were used for classification
  - KNN Regression
    - Find k neighbors and combine their dependent variable values
  - Regression Tree
    - Discussed next
  - Feedforward Neural Network
    - Example shown later

Regression Tree



Regression tree modelling uses MSE criterion to determine splits

This tree roughly corresponds to the following regression surface (assuming that there were only the predictor variables X_1 and X_2) :



Regression Tree Example

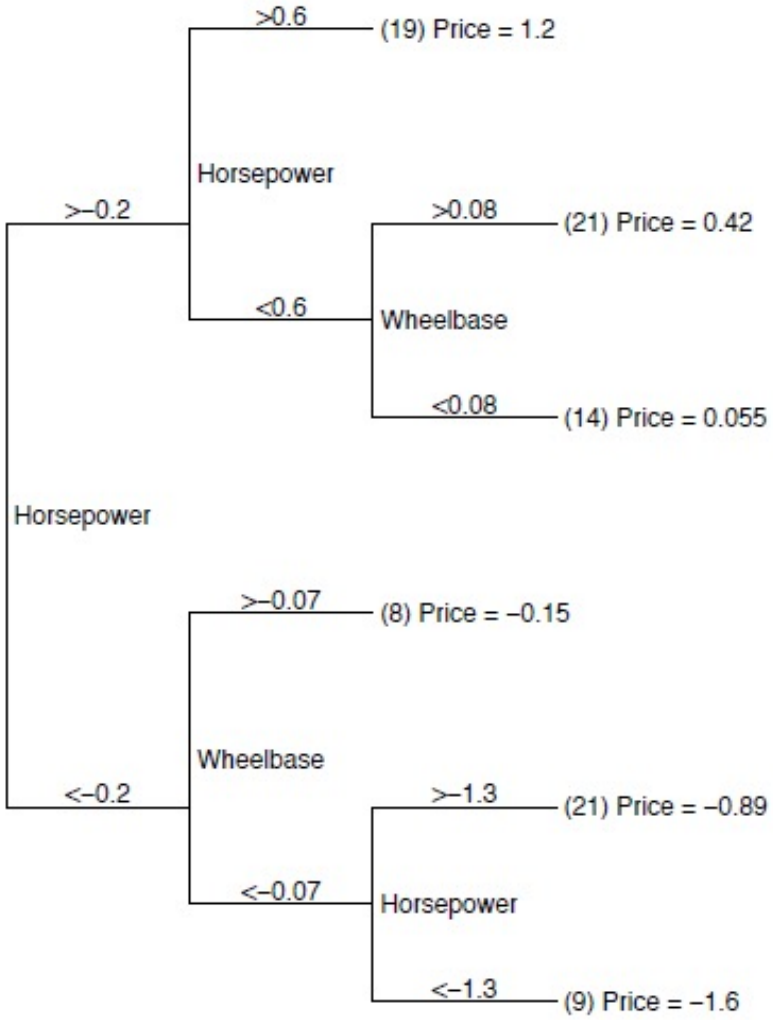


Figure 1: Regression tree for predicting price of 1993-model cars. All features have been standardized to have zero mean and unit variance. Note that the order in which variables are examined depends on the answers to previous questions. The numbers in parentheses at the leaves indicate how many cases (data points) belong to each leaf.

Regression Tree Example

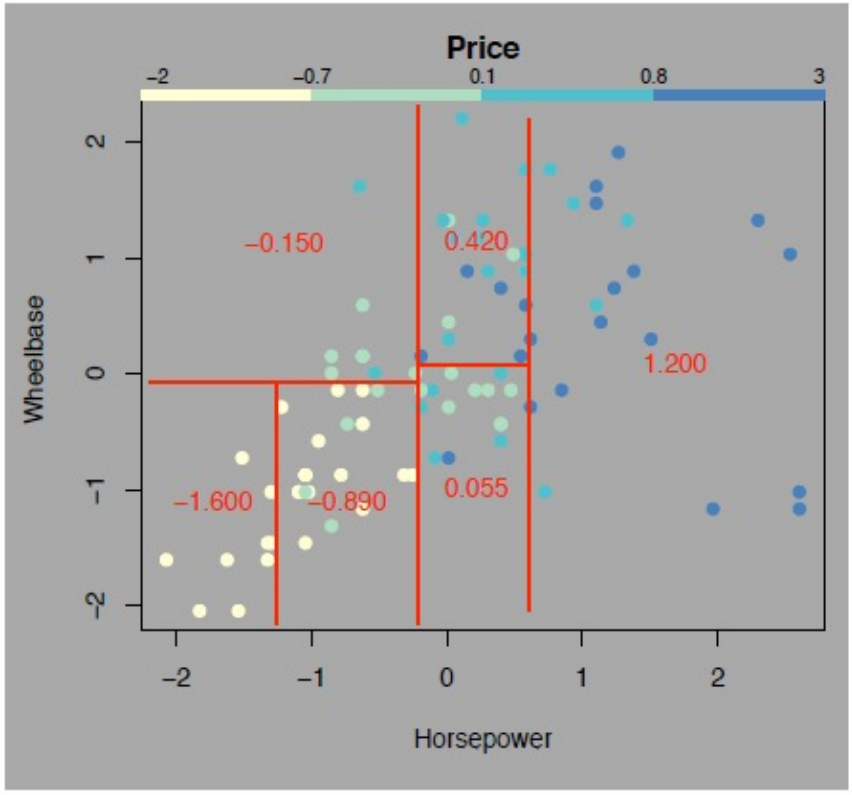


Figure 2: The partition of the data implied by the regression tree from Figure 1. Notice that all the dividing lines are parallel to the axes, because each internal node checks whether a single variable is above or below a given value.

Regression Tree Example using Sklearn

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
```

```
# We will use a noisy sin function for data generation
rng = np.random.RandomState(1)
X = np.linspace(0,5,100).reshape(-1,1)
#X = np.sort(5 * rng.rand(100, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 0.5*(0.5-rng.rand(20))
```

```
# Let's model data with a regression tree
# We will produce two trees with different depths
regr_1 = DecisionTreeRegressor(max_depth=2)
regr_2 = DecisionTreeRegressor(max_depth=4)
regr_1.fit(X, y)
regr_2.fit(X, y)
```

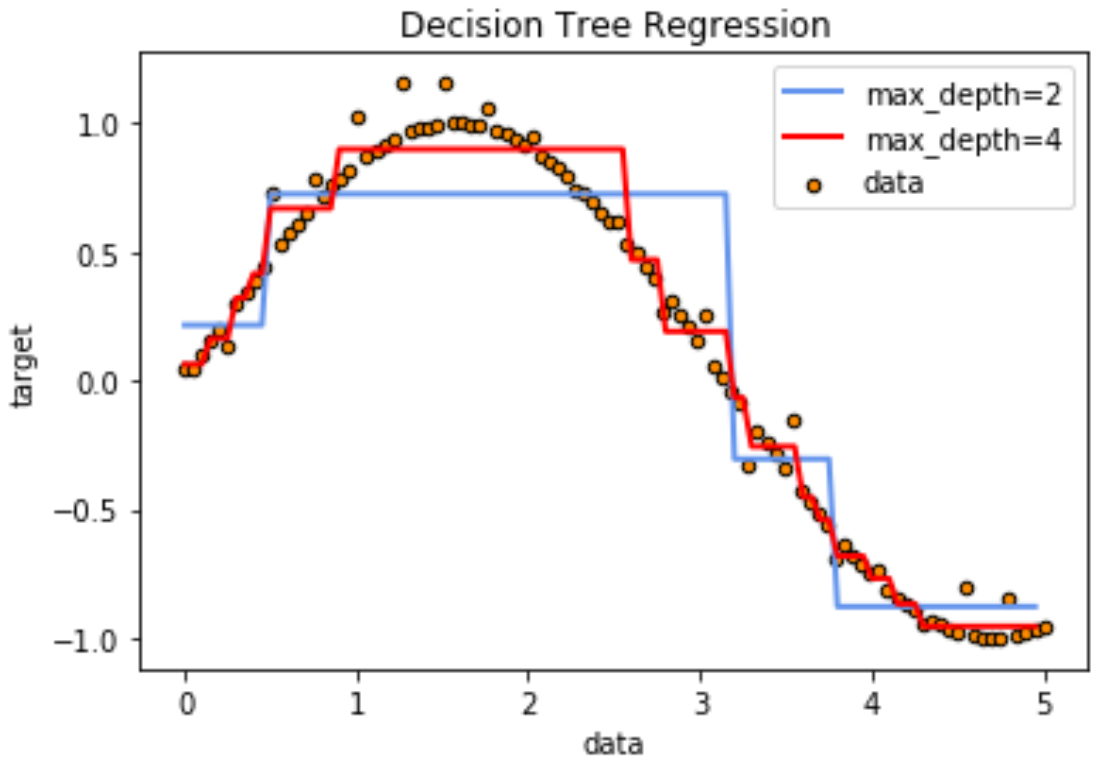
Regression Tree Example Contd.

```
# Predict
X_test = np.arange(0.0, 5.0, 0.05)[:, np.newaxis]
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)
```

```
# Let's plot results
plt.figure()
plt.scatter(X, y, s=20, edgecolor="black",
c="darkorange", label="data")
plt.plot(X_test, y_1, color="cornflowerblue",
label="max_depth=2", linewidth=2)
plt.plot(X_test, y_2, color="red", label="max_depth=4",
linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```

Regression Tree Plot

Shows how to sin function is approximated by a combination of step functions

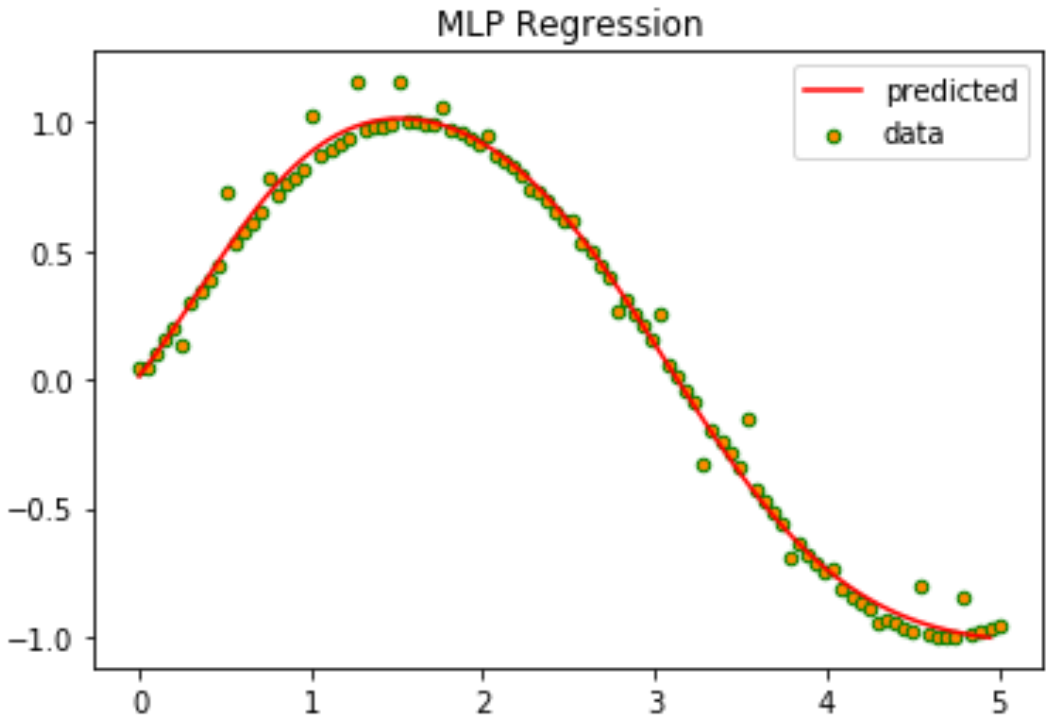


Regression using MLP Network

- In place of output being class labels, the output in this case is a continuous number.
- The following code shows MLP-based regression using the same data from the regression tree example

```
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(hidden_layer_sizes=(4), max_iter = 200, solver='lbfgs', \
                   alpha=0.01, activation = 'tanh', random_state = 8)
yfit = mlp.fit(X, y)
X_test = np.arange(0.0, 5.0, 0.05)[:, np.newaxis]
y_1 = yfit.predict(X_test)
plt.figure()
plt.scatter(X, y, s=20, edgecolor="black", c="darkorange", label="data")
plt.plot(X_test, y_1, '-r', label = 'predicted')
plt.title("MLP Regression")
plt.legend()
plt.show()
```

MLP Regression Result



The MLP has accurately learned to predict the output.

Regression Summary

- A type of supervised learning for quantitative dependent variables
- Numerous applications
- Multicollinearity (situation when predictors are highly correlated) requires additional attention. Check the following post for details.

https://iksinc.online/2019/01/07/how-to-perform-regression-with-more-predictors-than-observations/