Fourth Industrial Summer School

Day 3

Supervised Learning: Regression

Session Objectives

- ✓ Part 1:
  - Ordinary Linear Regression
  - Regression Evaluation
  - Multi-Linear Regression
- ✓ Part 2
  - Polynomial Regression
  - Spline Regression
- ✓ Part 3
  - Regularization
    - Ridge Regression
    - Lasso Regression

Introduction

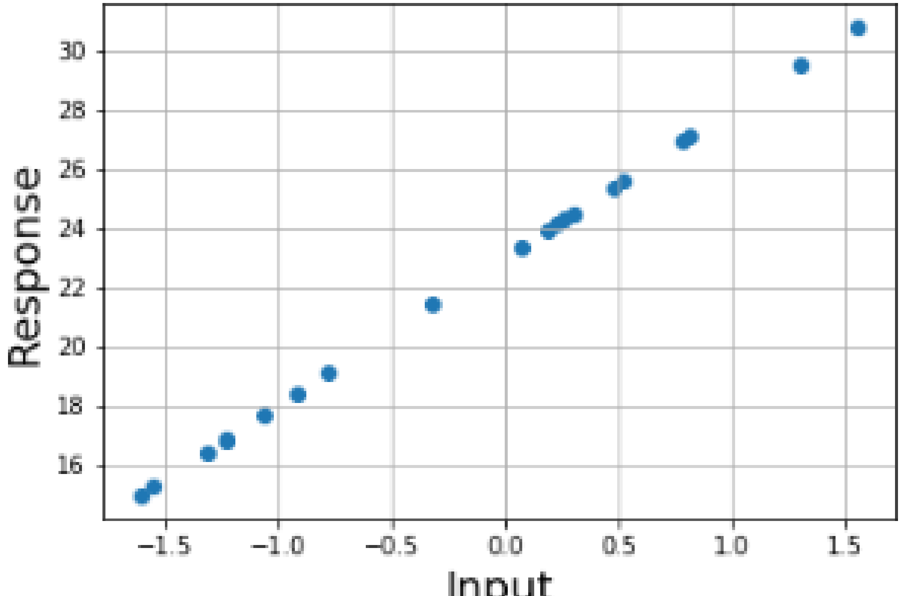
- Supervised Learning is a set of learning techniques where the ‘right answer’ for each datapoint exists at the learning stage.

- There are two types of supervised learning
  - Regression problems: the right answers are in the form of Continues Real values
  - Classification problems: the right answers are in the form of Finite Integer values.

Regression

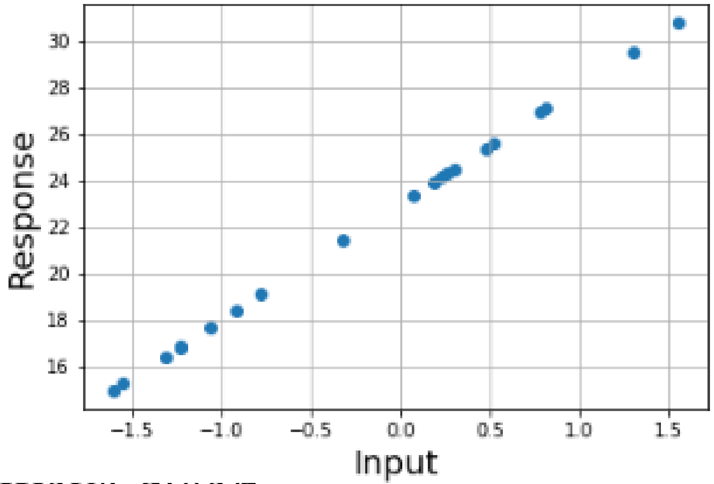
- Regression is a statistical technique that is used widely to predict a continuous future output.
- It models a relationship between two sets of variables

$$x\text{: (Input)}, \quad y\text{: (Response)}.$$



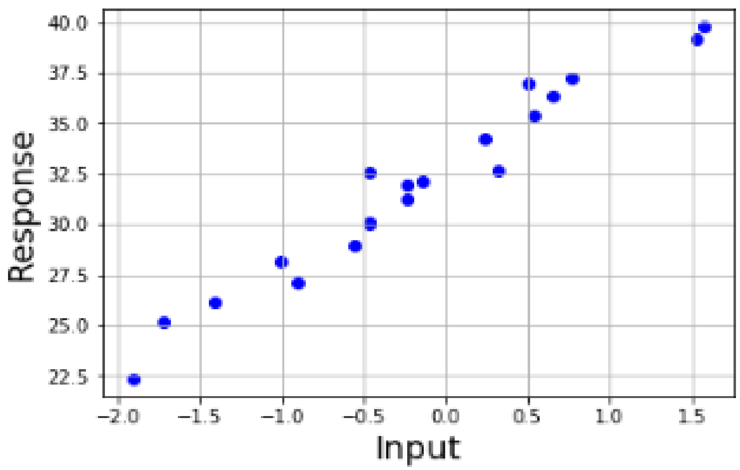
Regression data

- Data: $(X_i, Y_i)\ for\ i = 1, \ldots, n$



$$y = \beta_0 + \beta_1 X$$

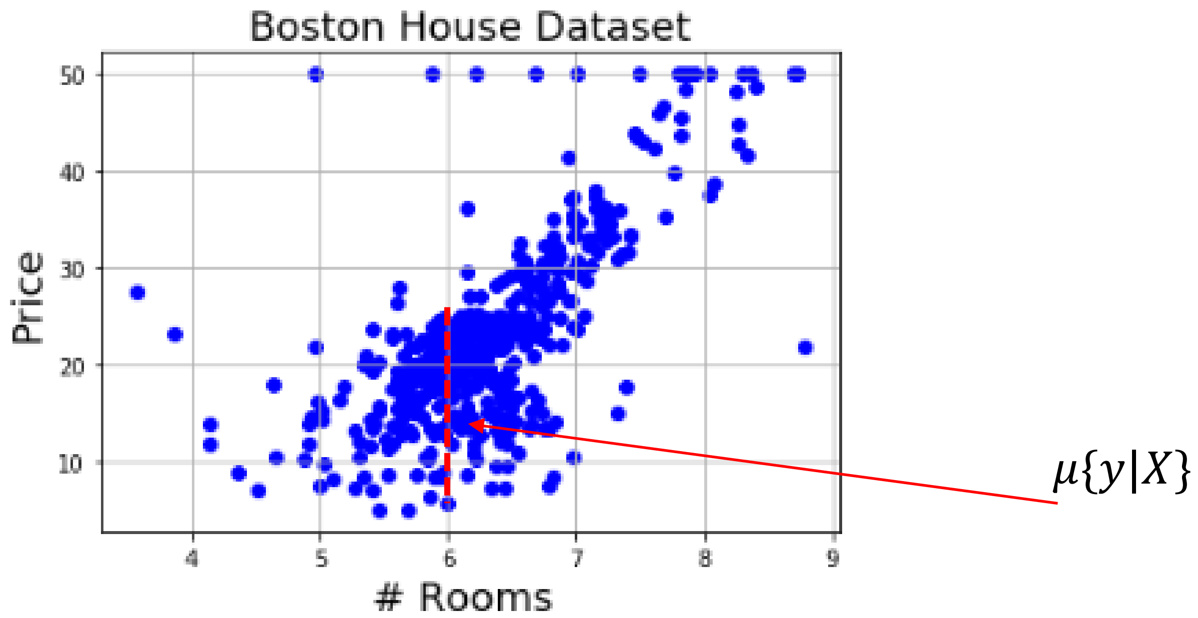
- They are not always clean



$$y = \beta_0 + \beta_1 X + \epsilon$$

Example:

- Suppose we have one Independent variable (number of rooms per hours)
- And a history of a selling price for each house



- Suppose someone asked you to estimate the price of his House (6 rooms)

Linear Regression Modeling

- Given

$$x\text{: (Feature/s)},\ \ y\text{: (Outcome).}$$

- Mean of Y is a straight line function of X, plus an error term (residual)
- Goal is to find the best fit line that minimizes the sum of the error terms

Contd.

- **Regression:** the mean of a response variable as a function of one or more explanatory variables:

$$\mu\{Y \mid X\}$$

- A Simple linear regression model:

$$\mu\{Y \mid X\} = \beta_0 + \beta_1 X$$

  - $\mu\{Y \mid X\}$: “mean of Y given X” or “regression of Y on X”
  - β_0: The intercept
  - β_1 : The slope
  - X: Independent variable

Regression Procedure

- A fitter value for sample x_i is its estimated mean:
$$\hat{y}_i = \mu\{y_i|x_i\} = \beta_0 + \beta_1 x_i$$

- Residual (Error) for observation x_i:
$$E_i = y_i - \hat{y}_i$$

- The objective is to make the Residual E_i as small as possible:
$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\left(y_i - (\beta_0 + \beta_1 x_i)\right)^2$$

Least Square Procedure

- Expand the Residual : $\sum_{i=1}^{n}(y_i - \beta_0 \; - \beta_1 x_i)^2$
- Compute the partial derivative of the objective function LS, and equate the result to zero, we get these two equations

1. $\sum_i \beta_0 + \sum_i \beta_1 x_i = \sum_i y_i$
2. $\sum_i \beta_0 \, x_i + \sum_i \beta_1 x_i^2 = \sum_i x_i y_i$

- It can be written as

$n.\beta_0 + \beta_1 \sum_i x_i \; = \sum_i y_i \,,$

$\beta_0 \sum_i x_i + \beta_1 \sum_i x_i^2 = \sum_i x_i y_i$

It will be clearer if we write them in matrix form:

$$\begin{bmatrix} n & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{bmatrix}$$

Contd.

$$\begin{bmatrix} n & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{bmatrix}$$

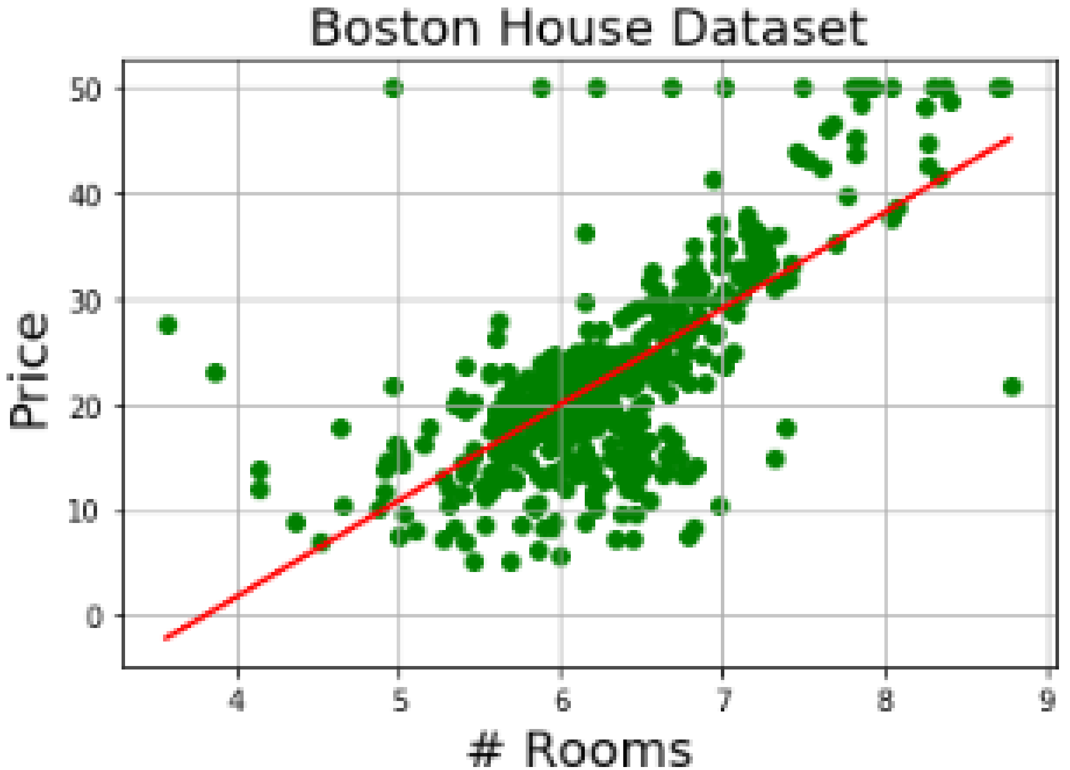
- By Cramer's rule or Gaussian-elimination to solve this system of liner equations:

$$\beta_1 = \frac{(n \sum_i x_i y_i - \sum_i x_i \sum_i y_i)}{n \sum_i x_i^2 - \left(\sum_i x_i\right)^2}$$

$$\beta_0 = \frac{1}{n} \sum_i y_i - \frac{\beta_1}{n} \sum_i x_i$$

Contd.

- The reply to our friend is $\approx \$20K$

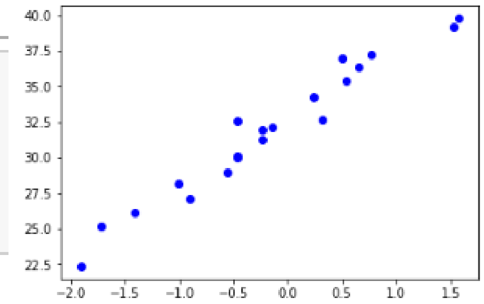



```
print("Your House with 6 Rooms Can be Sold with: $%0.2fK"% (b0 + b1* 6) )
```

```
Your House with 6 Rooms Can be Sold with: $19.94K
```

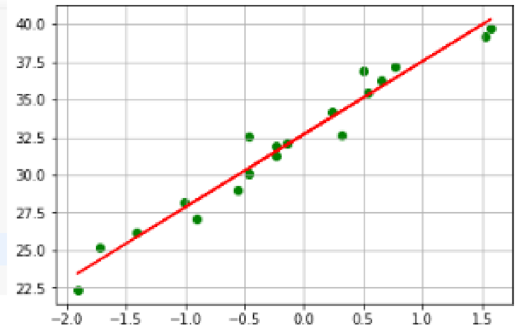
Try it out

```
np.random.seed(42)
#X = 8 * np.random.randn(20)+20  # std and mean
X = 1 * np.random.randn(20) # std and mean
y = 23 + 5 * X   + np.random.randn(20) + 10 # approximately std=1, mean=0
plt.scatter(X,y, c='b')
```

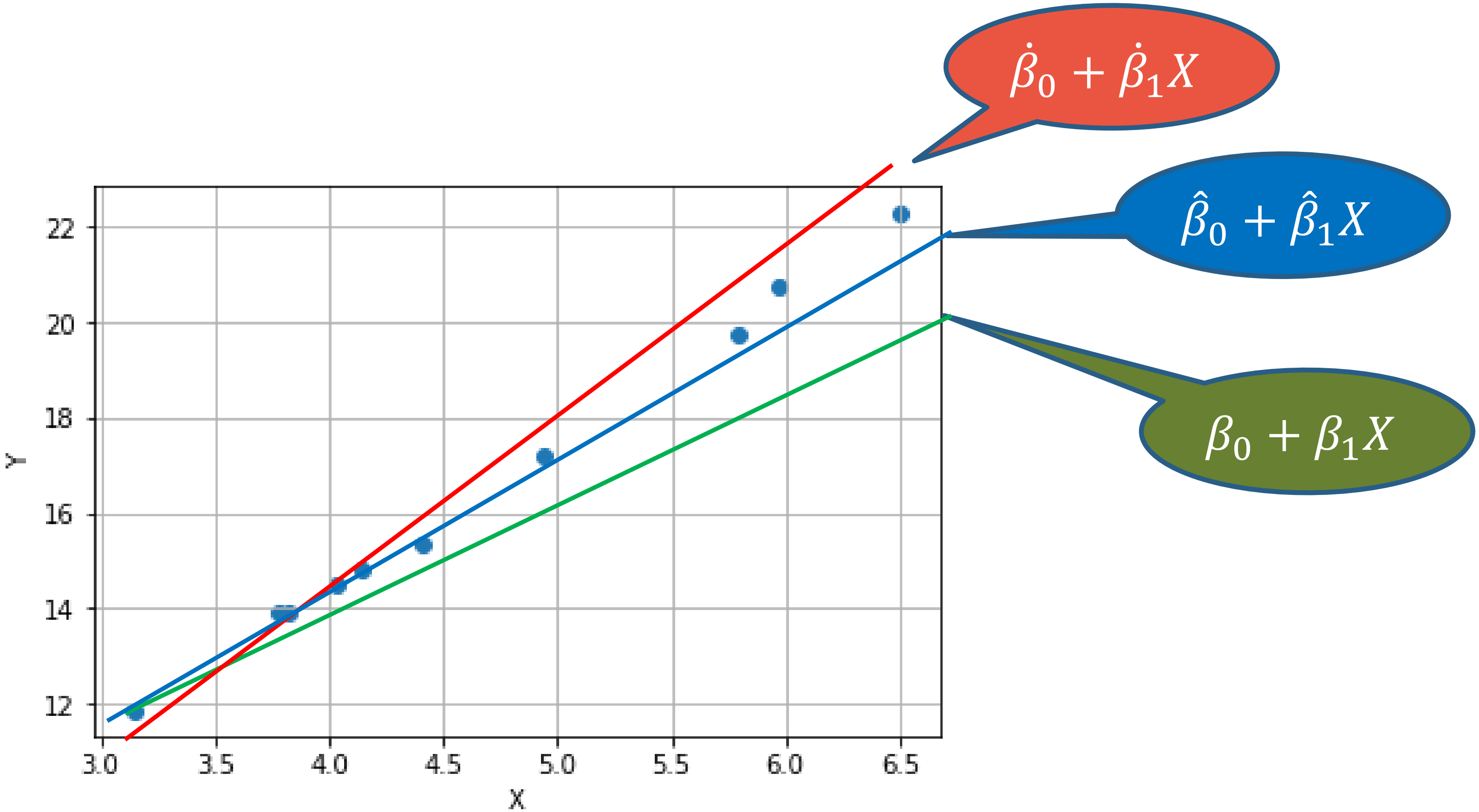


```
# Estimator
def estimate_coef(x,y):

  # get size of samples
  n = np.size(x)

  # compute b1
  b1 = (n * np.sum(x*y) - (np.sum(x) * np.sum(y)) ) / ( n* (np.sum(x**2)) - (np.sum(x)**2)   )
  b0 = np.mean(y) - b1 * (np.mean(x))
  return (b0, b1)
```

```
# Estimate the papmetres
b0, b1 = estimate_coef(X,y)
y_pred = b0 + X *b1
plt.scatter(X,y, c='g')
plt.plot(X, y_pred, c='r')
plt.grid()
plt.show()
```



Multiple solutions



We may have similar situation when we do not have enough data!

Scikit-learn

- **LinearRegression** Class can be found in **linear_model** package in scikit learn.

```
from sklearn.linear_model import LinearRegression
```

- **Use .fit( ) function to train the model on some data**

```
LR = LinearRegression()
LR.fit(X[:,None],y) # most of the Scikit learn algorithms requires 2d data
```

- **To predict response values**

```
y_pred= LR.predict(Xts[:,None])
```

Attributes

- To get the β_0 and β_1

```
print('From Scikit:\n b0:', LR.intercept_, '    b1:', LR.coef_)
```

Evaluation: How well our model is?

- R-squared is the **proportion of variance explained**
- R-squared can be used to understand the power of the predictions

$$R^2 = 1 - \frac{\textbf{Explained variation(SSE)}}{\textbf{Total Variation (SST)}}$$

$$\hat{y} = (\beta_0 + \beta_1 x_1)$$

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

$$SST = \sum (\hat{y}_i - \bar{y})^2$$

- $\bar{y}$: is the mean of y
- What is a good value for R^2?

Evaluation: How well our model is?

- R-Squared is also called **coefficient of determination.**

```
from sklearn.metrics import r2_score
```

- It ranges [**negative , 1]**
  - A R-squared value of 1 means the model explains all the variation of the target variable.
  - A value of 0 measures zero predictive power of the model.
  - negative values means our model is arbitrary worse!
  - We may expect a value of R-squared $\geq \frac{\#\, Features}{\#\, of\ samples}$

- **Higher R-squared value, better the model. But require adjustment**

What is a good value for R^2

- The threshold for a good R-squared value depends on the domain!
- In other words, the nature of the spread in the dependent variable may vary according to the data.
- R-squared is a fraction by which the variance of the errors is less than the variance of the dependent variable.
- But, it is useful as a tool for **comparing different models**
- **Note:** R-squared will always increase as you add more features to the model.

Multi-LR

- Is a generalization case where we have multiple features instead of only 1
- The data is in this form: $X = [X_1, X_2, \ldots, X_n]$ and one response value

- Exercise 3 introduces a dataset that is collected from our labs at the RI(KFUPM). The goal of the experiments were to refine to types of fuels and reduce sulfur content from them using a processing called hydrodesulfurization.
  - Response variable 'sulfur_concentraction'
  - And there are 5 independent variables (temperature, pressure, dosage, initial concentration, and fuel type)
  -

Exr3 Data Source: https://www.sciencedirect.com/science/article/pii/S0167732218358161#s0075

Multi-LR

- Training

click to scroll output; double click to hide

	temperature	Pressure	Dosage	Init_concentration	fuel_type	sulfur_concentraction
0	200.0	25.0	0.4	500.0	1.0	415.0
1	200.0	25.0	0.4	1500.0	1.0	1258.0
2	200.0	25.0	0.8	500.0	1.0	390.0

- Testing

	temperature	Pressure	Dosage	Init_concentration	fuel_type	sulfur_concentraction
0	332.0	66.0	0.61	649.0	2.0	212.0
1	251.0	58.0	0.58	563.0	1.0	245.0
2	289.0	65.0	0.41	1183.0	1.0	215.0

Exercise

- Explore the dataset and make sure its clean
- Use all features to train a model for predicting a sulfur concentration after the process.
- Evaluate your model using the test dataset
- Try a combination of features to improve if possible

<u>Part 2</u>
Polynomial Regression
Spline Regression

Quadratic Regression

- A quadratic regression is the process of finding the equation of the parabola that best fits the data.

- The quadratic equation has the form of:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2, \qquad where\ \beta_2 \neq 0$$

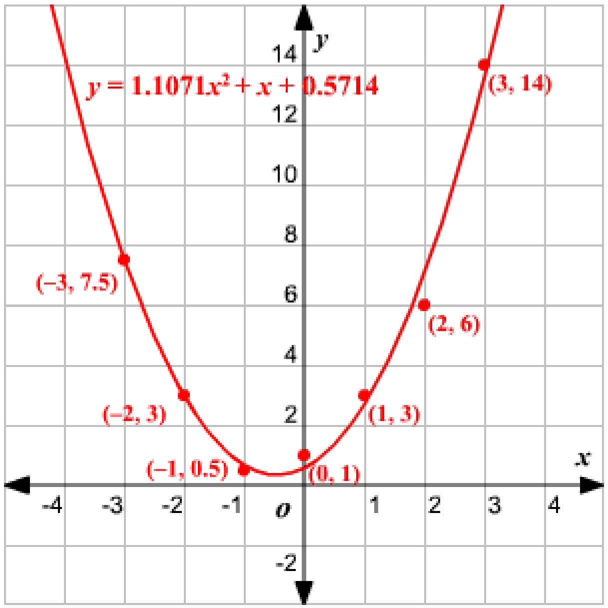
- One way to find this equation is using the lest squares method.
  - Find $\beta_0, \beta_1, \beta_2$ such that the squared vertical distances between each point (x_i, y_i) and the quadratic curve is minimal:

$$\begin{bmatrix} \sum x_i^4 & \sum x_i^3 & \sum x_i^2 \\ \sum x_i^3 & \sum x_i^2 & \sum x_i \\ \sum x_i^2 & \sum x_i & n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \sum x_i^2 y_i \\ \sum x_i y_i \\ \sum y_i \end{bmatrix}$$

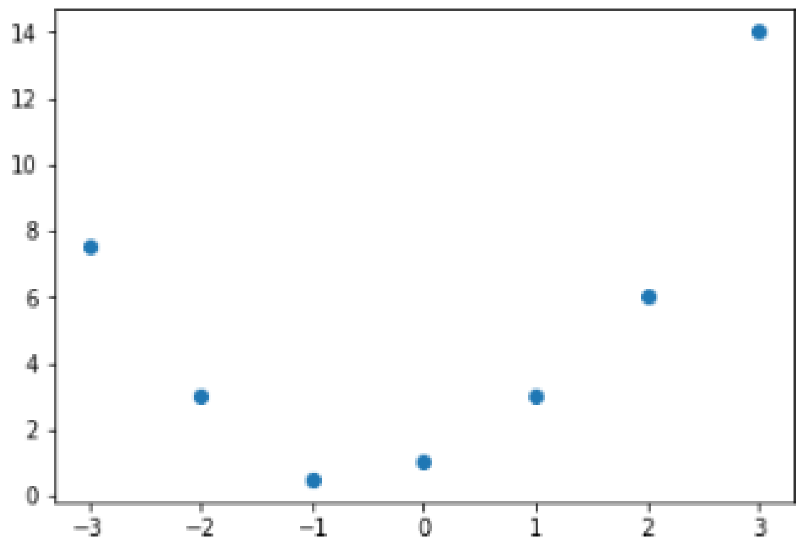
Quadratic Regression

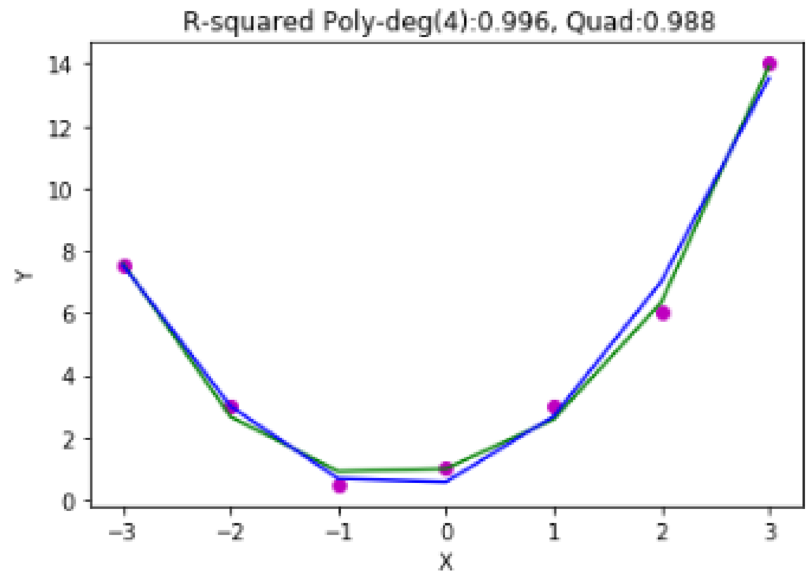
- Consider the set of data. Determine the quadratic regression for the set.

	X	Y
0	-3	7.5
1	-2	3.0
2	-1	0.5
3	0	1.0
4	1	3.0
5	2	6.0
6	3	14.0



PL results (degree 2 and degree 4)





Original data

Y	X
7.5	-3
3	-2
0.5	-1
1	0
3	1
6	2
14	3

Transform

Transformed data

Y	X	X^2	X^3	X^4
7.5	-3	9	-27	81
3	-2	4	-8	16
0.5	-1	1	-1	1
1	0	0	0	0
3	1	1	1	1
6	2	4	8	16
14	3	9	27	81

Transform to polynomial features

- Transform to higher degrees

```
from sklearn.preprocessing import PolynomialFeatures
```

- Quadratic example

```
ploy = PolynomialFeatures(degree = 2)
ploy.fit_transform(X[:,0].reshape(-1,1) )
```

```
array([[ 1., -3.,  9.],
       [ 1., -2.,  4.],
       [ 1., -1.,  1.],
       [ 1.,  0.,  0.],
       [ 1.,  1.,  1.],
       [ 1.,  2.,  4.],
       [ 1.,  3.,  9.]])
```

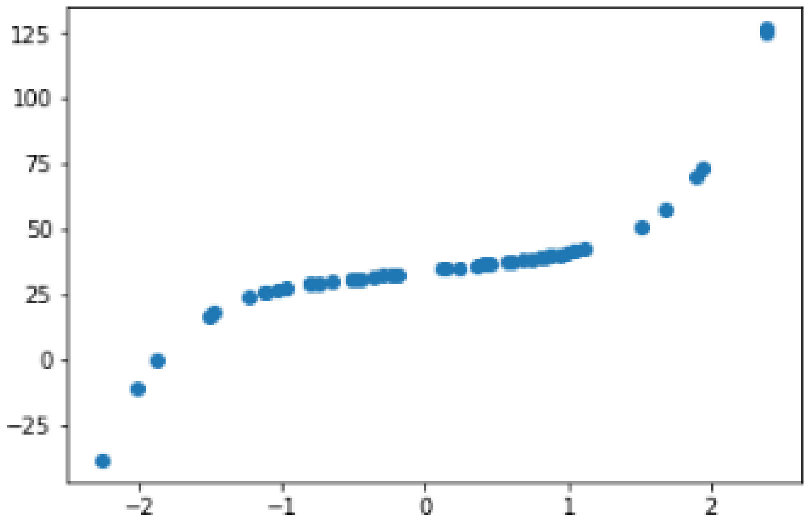
Steps to carryout

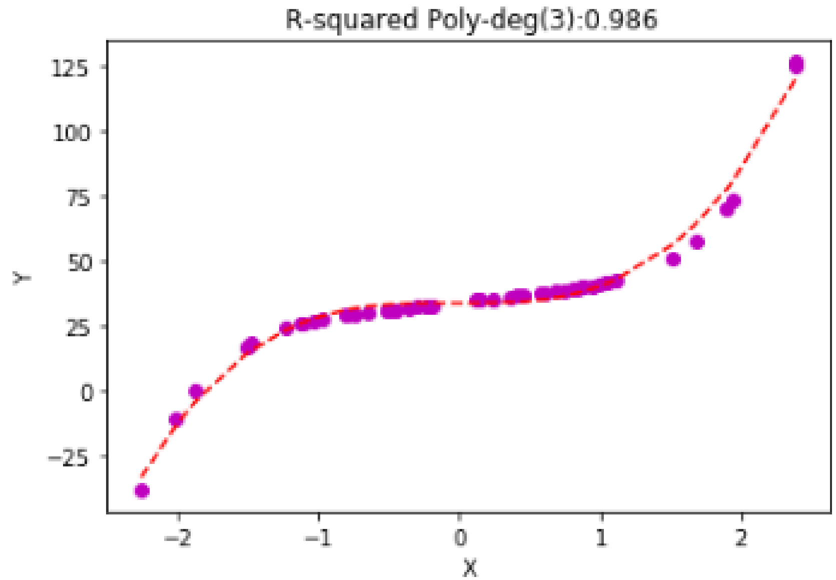
- There are three important steps

```python
# 1- do some feature polynomial transformation
dg = 2
poly_reg = PolynomialFeatures(degree = dg)
X_poly     = poly_reg.fit_transform(X_train[:,None])

# 2- Generate a linear model to fit the new features
lrP = LinearRegression()
lrP.fit(X_poly, yd)

# 3- test values must pass through feature transformation again
Xtest_poly     = poly_reg.fit_transform(X_test[:,None])
```

Find Possible Ploy degree (Brute-Force)

- Loop through a list of numbers (degrees) 2,3, 5, 7, .. 11
- Use R-squared to judge, should I stop
- Stop when the difference between the previous and the current R^2 is less or equal to 0.1



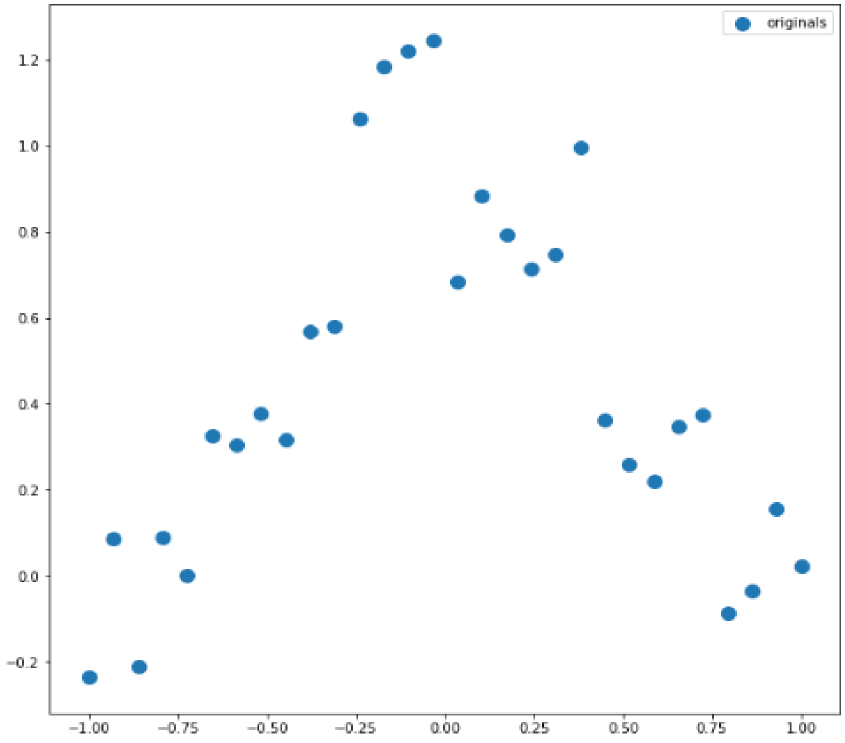


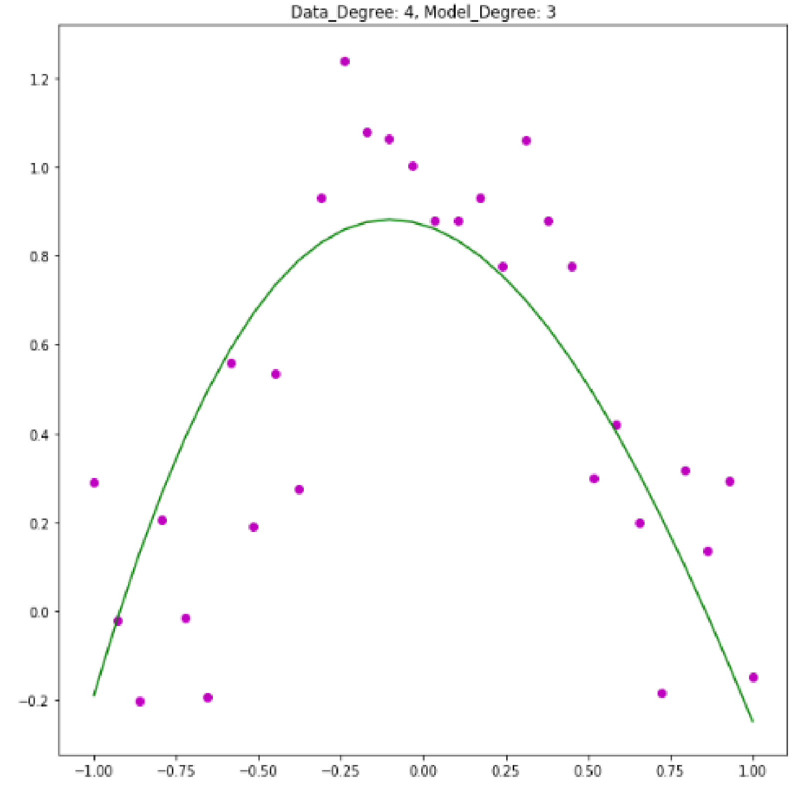
Piece-wise polynomial regression (spline)

- Unfortunately, Polynomial Regression has a number of issues:
  - By increasing the complexity of the formula, the number of features is also increased (might be difficult to handle).
  - Polynomial regression models suffer over-fitting problem(High variance), even on these simple one dimensional data.
  - It is inherently non-local. (The fit is affected greatly by any change in Y values during training)
- PR can be substituted with many different small degree polynomial functions, i.e. piece-wise polynomials (splines)

Piece-wise polynomial regression (spline)

- An example: The blue dots are generated using a polynomial function of degree 4, compared to a deg(4) model

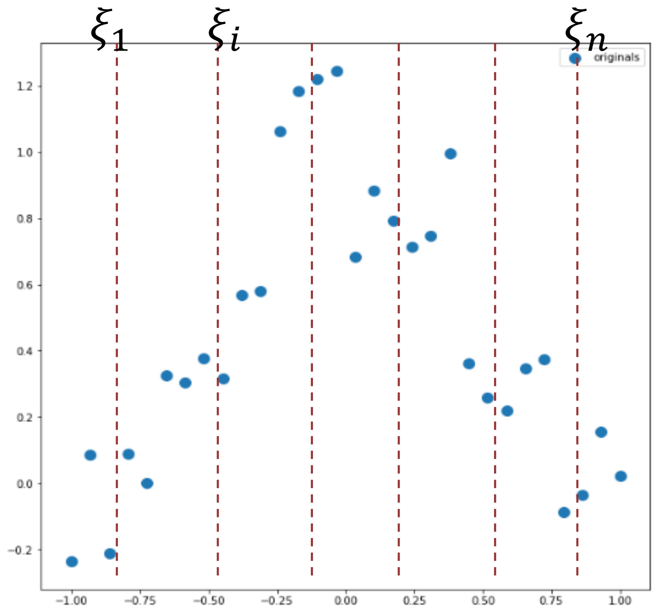


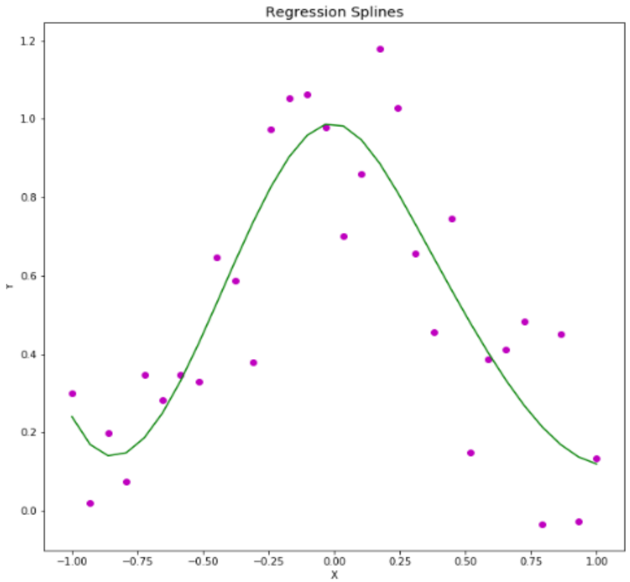


How spline regression works?

- It divides the space into several knots.
- Then, it transforms X into piece-wise polynomials that can captures general shapes. (Basis function)

$$y_i = \beta 0 + \beta_1 b_1 (x_i) + \beta_2 b_2(x_i) + \beta_3 b_3 (x_i) + \ldots + \beta_K b_K (x_i)$$





General points

- Unlike polynomial Regression, which tries to use a high degree polynomials to produce flexible fits, splines introduce flexibility by increasing the number of knots but keep the degree fixed.
- Splines allow more flexibility, we can place more knots around rapidly changing regions of a function
- Do not go beyond cubic-splines (unless one is interested in smooth derivatives).

Upgrade Colab statsmodels library

- We need to update these packages

```
# Please install this package and restrate the notebook from Runtime menu (statsmodels)
! pip install --upgrade Cython
! pip install --upgrade git+https://github.com/statsmodels/statsmodels

```

```
Collecting Cython
  Downloading https://files.pythonhosted.org/packages/c5/c6/c4fabec42bf5f6220e06621b0a239f9758f
     |████████████████████████████████| 2.1MB 5.1MB/s
Installing collected packages: Cython
  Found existing installation: Cython 0.29.10
    Uninstalling Cython-0.29.10:
      Successfully uninstalled Cython-0.29.10
Successfully installed Cython-0.29.11
```

Exercise on splines (statsmodels required)

- Load both formula.api and api from **statsmodels**. Then load design matrix (dmatrix) from **patsy** library

```
# for the stas model, it requires an update above
import statsmodels.formula.api  as smf
import statsmodels.api as sm
from patsy import dmatrix
```

- Generate some hard data

```
def Concave(x):
    return 1/(1+25*x**4)

# make example data
tmpx = np.linspace(-1,1,30)
y = Concave(tmpx) + np.random.normal(0, 0.2, len(tmpx))
```

Contd.

- For binning data, you can use dmatrix

```
# Generating cubic spline with 3 knots

Ttmpx3 = dmatrix("bs(train, knots=(-1,0,0.3), degree=3, include_intercept=False)",
                {"train": tmpx},return_type='dataframe')
Ttmpx3
```

- The data frame shows different datapoints divided by knots. As we have 3 knots, the result columns will be 6.

	Intercept	bs(train, knots=(-1, 0, 0.3), degree=3, include_intercept=False)[0]	bs(train, knots=(-1, 0, 0.3), degree=3, include_intercept=False)[1]	bs(train, knots=(-1, 0, 0.3), degree=3, include_intercept=False)[2]	bs(train, knots=(-1, 0, 0.3), degree=3, include_intercept=False)[3]	bs(train, knots=(-1, 0, 0.3), degree=3, include_intercept=False)[4]	bs(train, knots=(-1, 0, 0.3), degree=3, include_intercept=False)[5]
0	1.0	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	1.0	0.807044	0.182426	0.010403	0.000126	0.000000	0.000000
2	1.0	0.640658	0.319010	0.039323	0.001009	0.000000	0.000000
3	1.0	0.498872	0.414397	0.083324	0.003406	0.000000	0.000000
4	1.0	0.379720	0.473235	0.138970	0.008074	0.000000	0.000000
5	1.0	0.281233	0.500170	0.202827	0.015770	0.000000	0.000000

Contd

- For splines, we use the general model from the **statsmodels.api**

```
Spline3 = sm.GLM(y, Ttmpx3).fit()
```

```
# Fitting Generalised linear models
SplineL = sm.GLM(y, Ttmpx).fit()
SplineL.conf_int ()
```

	0	1
Intercept	-0.071583	0.458375
bs(train, knots=[-0.5, 0, 0.5], degree=2, include_intercept=False)[0]	-0.744392	0.179102
bs(train, knots=[-0.5, 0, 0.5], degree=2, include_intercept=False)[1]	0.294302	0.924674
bs(train, knots=[-0.5, 0, 0.5], degree=2, include_intercept=False)[2]	0.553348	1.284763
bs(train, knots=[-0.5, 0, 0.5], degree=2, include_intercept=False)[3]	-0.687198	0.071796
bs(train, knots=[-0.5, 0, 0.5], degree=2, include_intercept=False)[4]	-0.516231	0.239133

Predict

- Then, we use **Spline3.predict(dmatrix())** to compute **y_pred**

```
# Predicting new examples, remember we need to do the same transformation of data

X_new = dmatrix("bs(train, knots=[-0.5, 0, 0.5], degree=2, include_intercept=False)",
                {"train": tmpx},return_type='dataframe')
pred2 = SplineL.predict(X_new)

```

Part 3: Regularization

- Ridge
- Lasso

Regularization

- Assessment:
  - Sometimes, we have very few features on a dataset and the score is poor for both the training and testing - Underfitting (High bias)
  - But sometimes, we have large number of features and test score is relatively poor than the training score, then we have a problem of generalization- Overfitting

- Lasso and Ridge Regression are two techniques that prevent Overfitting problem

Ridge Regression

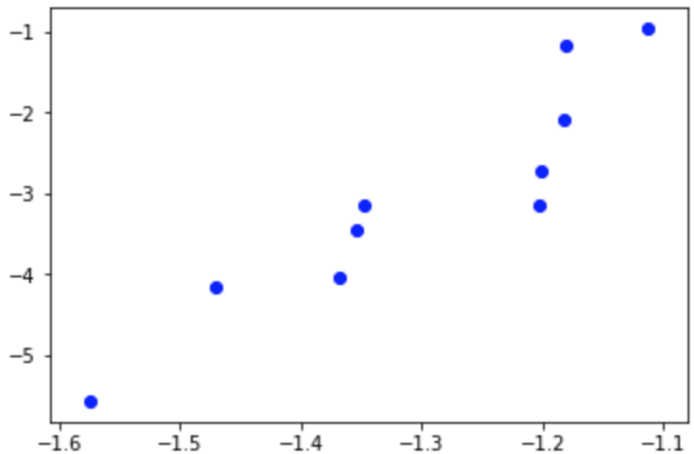
$$\sum_{i=1}^{n}\left(y_i - (\beta_0 + \beta_1 x_i)\right)^2 + \alpha\sum_{i=1}^{n}\beta_i^2$$

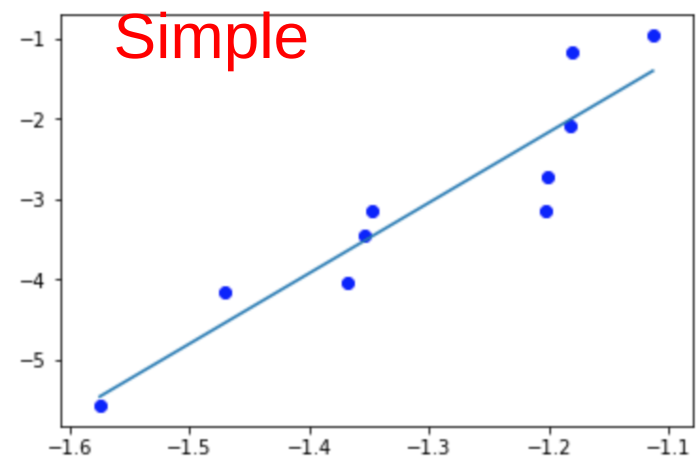
- In Ridge regression the cost function is modified by adding a penalty equivalent to square of the magnitude of the coefficients.
- It is similar to simple LR, but with adding the following condition:

$$for\ some\ \ \mathrm{c} > 0, \sum_{i=1}^{n}\beta^2 < \mathrm{c}$$

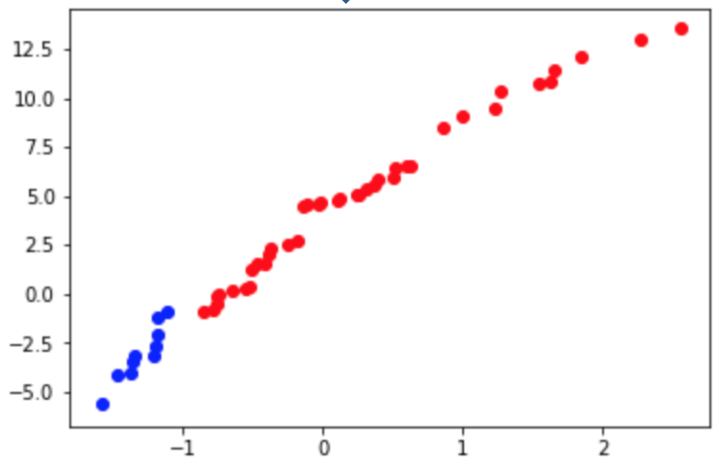
- The penalty term α regularizes the coefficients such that if the coefficients take large values the optimization function penalized.
- Ridge regression shrinks the coefficients and it helps to reduce the model complexity and multi-collinearity.

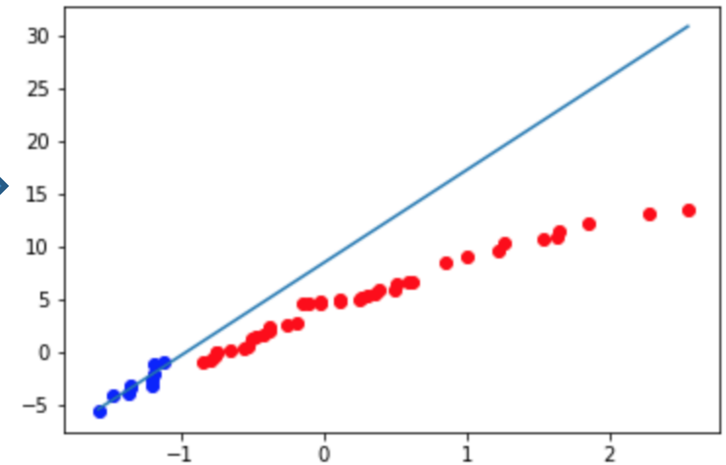
What Ridge is trying to do?



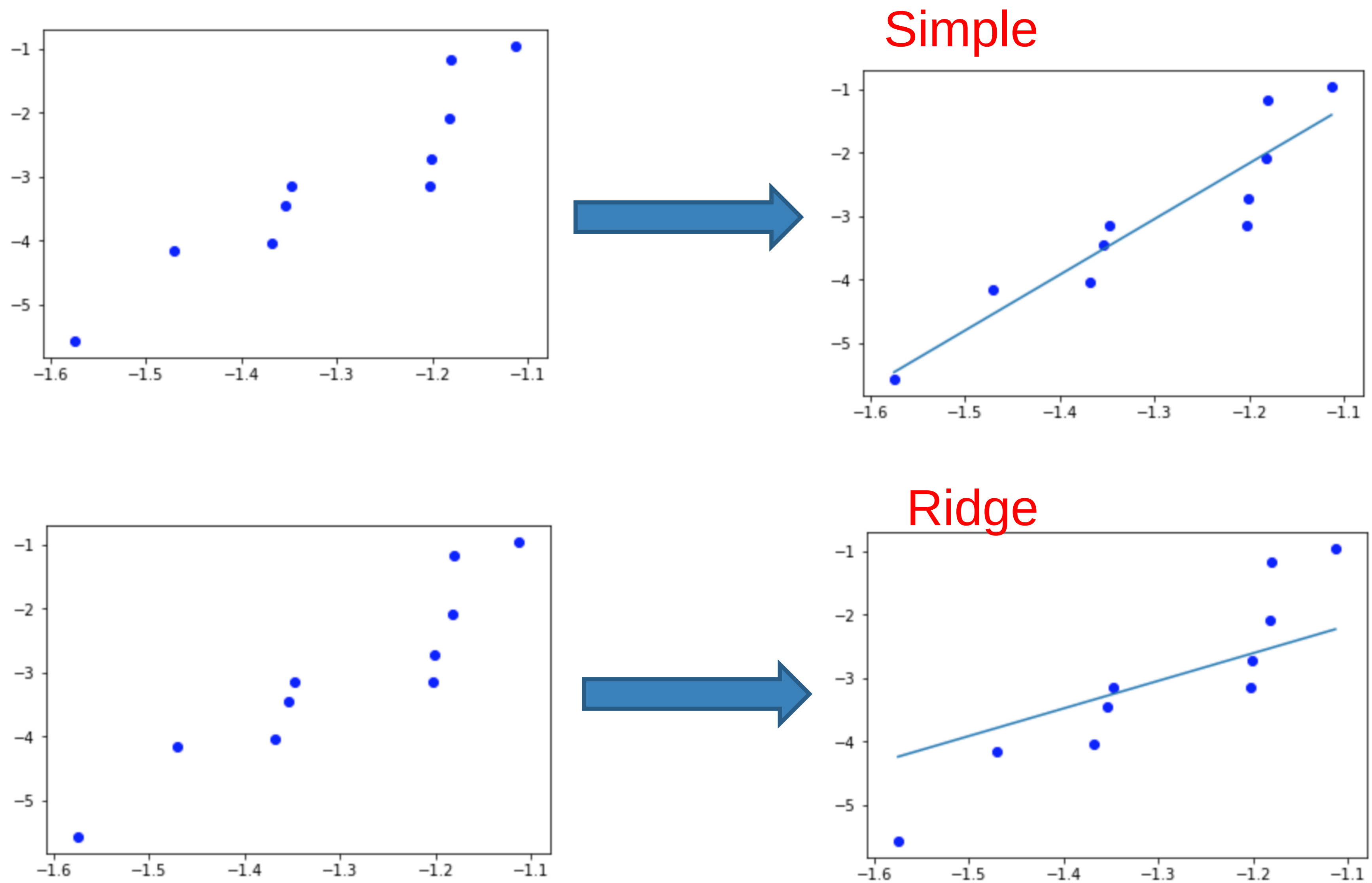


Extension with complete picture

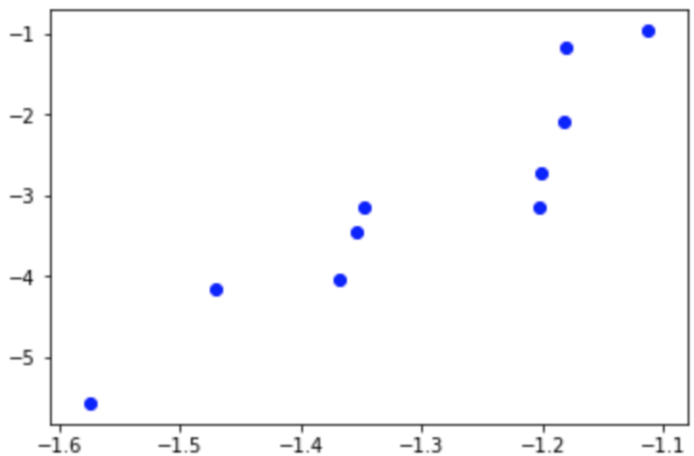




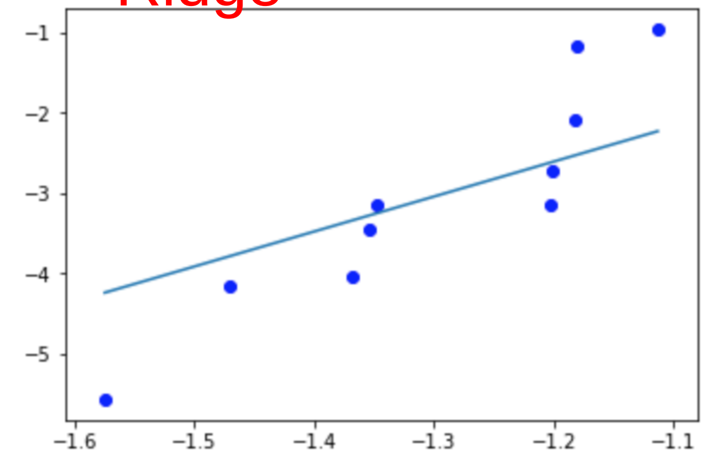
What Ridge is trying to do?



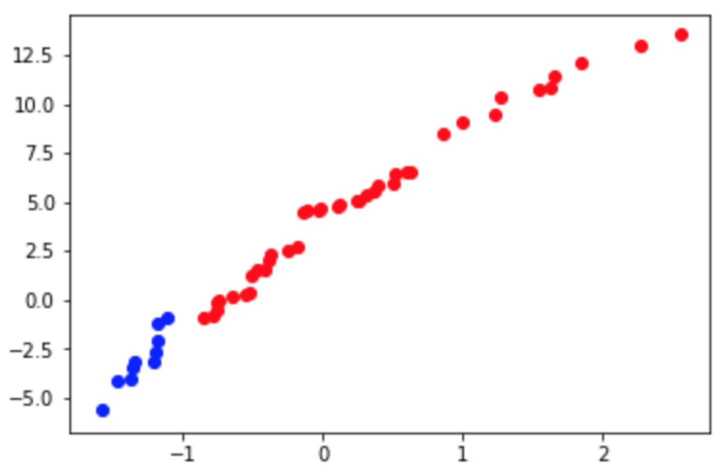
What Ridge is trying to do?

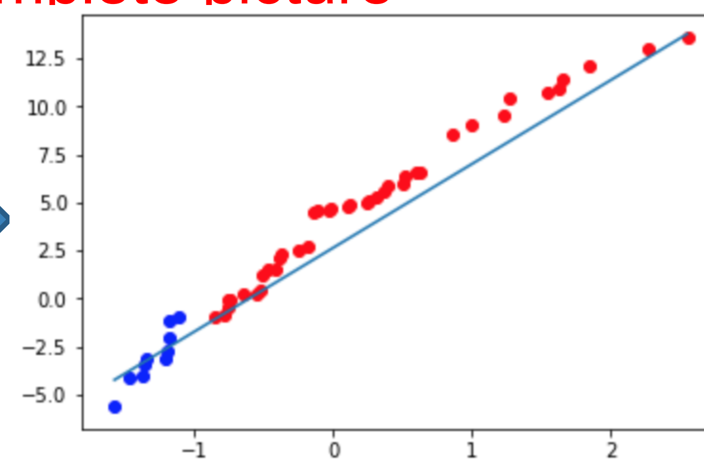


Ridge



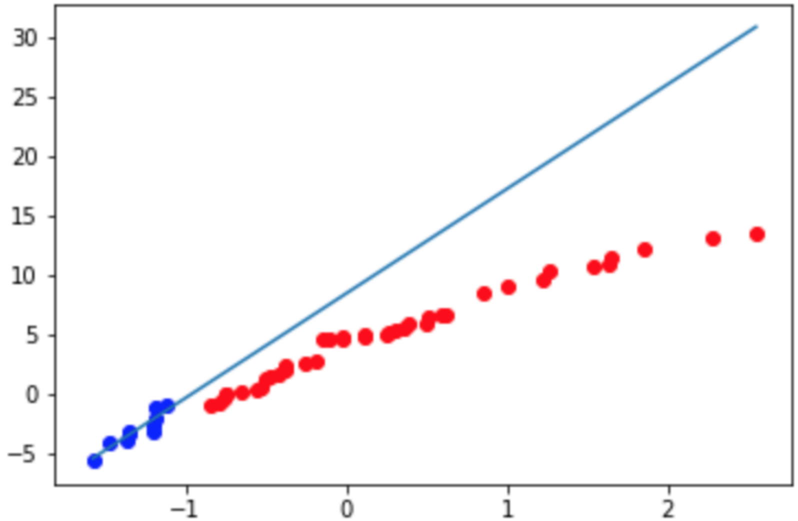
Extension with complete picture



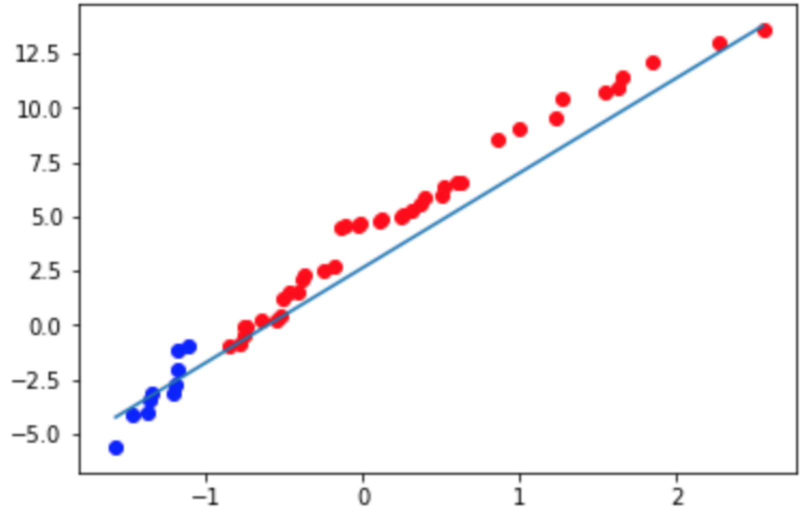


What Ridge is trying to do?

Simple



Ridge



Lasso Regression

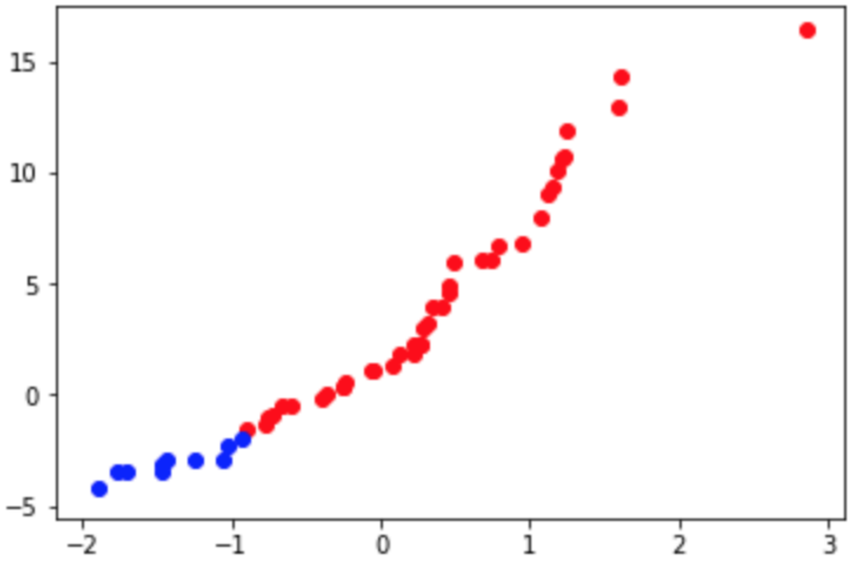
- The cost function for Lasso (least absolute shrinkage and selection operator) regression can be written as

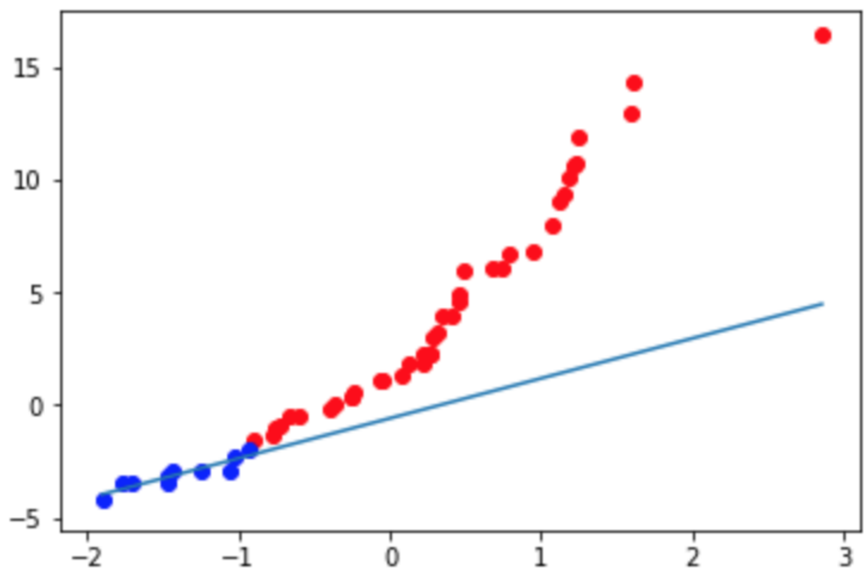
$$\sum_{i=1}^{n}\left(y_i - (\beta_0 + \beta_1 x_i)\right)^2 + \lambda \sum_{i=1}^{n} |\beta|$$

$$for\ some\ c > 0, \sum_{i=1}^{n} \beta^2 < c$$

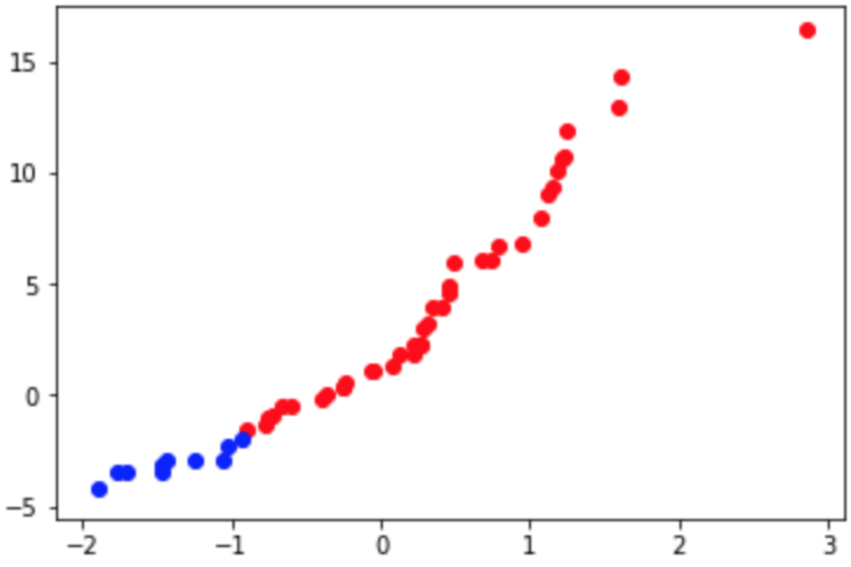
- Lasso regression coefficients subject to similar constrain as Ridge.

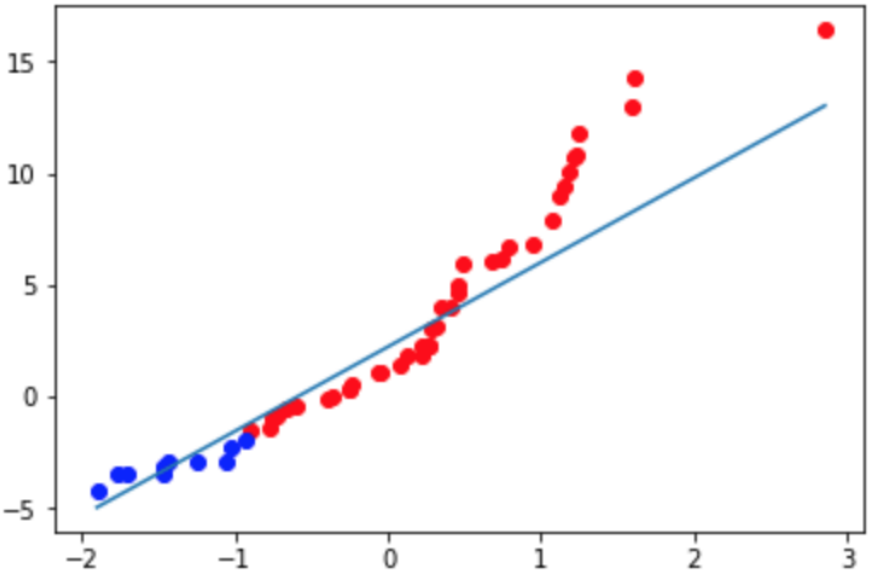
Simple regression





Lasso Regression





When to use:

- Regularization can used in these two situations:
  - Use either method when number of features are greater than samples.
  - If the features are multi-colinear

Scikit-learn

- Ridge and lasso regressions are in scikit-learn **linear_model** package.

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

```
ridge = Ridge(alpha=0.02)
ridge.fit(X[:,None], y)

lasso = Lasso(alpha=0.1)
lasso.fit(X[:,None], y)
```

```
yr_pred = ridge.predict(X[:,None])
yr_pred = lasso.predict(X[:,None])
```

Tips To Improve

- Normalize your data, i.e., shift it to have a mean of zero, and a spread of 1 standard deviation

- Do feature engineering:
  - Are the features collinear?
  - Do any of them have cross terms/higher-order terms?

```
from sklearn.preprocessing import StandardScaler
```

```
zscore = StandardScaler()
zscore.fit(X.reshape(-1,1))

```

```
X = zscore.transform(X[:,None])
```

Thank you