Regression

Tools and Techniques for Data Science

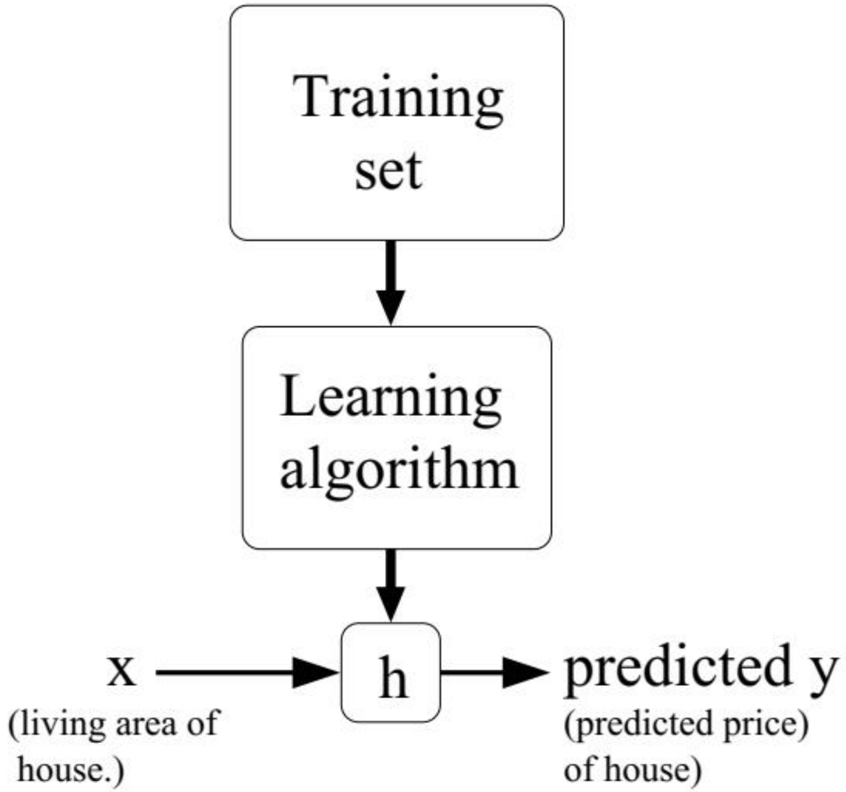
Dr. Murk Marvi

Outline

- Regression
  - Introduction
  - Performance metrics
  - Implementation in Python
  - A few types of regression

Supervised Learning

- In **supervised learning** we teach or train the machine using **data** which is well **labeled.**

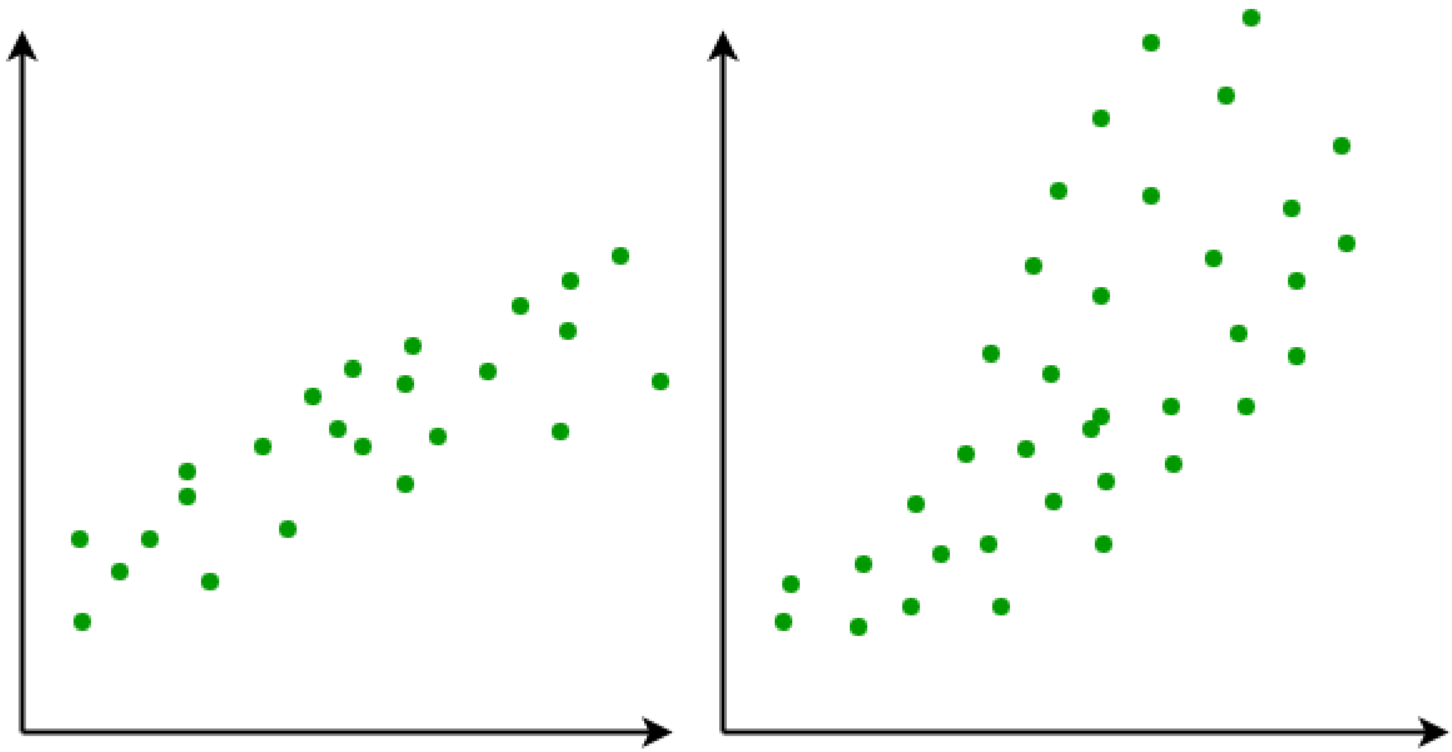


Regression

- When the **target variable** that we're trying to predict **is continuous,** we call the learning problem a **regression problem.**

- When **target variable** can take on only a small number of **discrete values**, we call it a **classification problem.**

Regression



Linear Regression

- It is a technique in which the **dependent variable is continuous** in nature.

- The relationship between the dependent variable and independent variables is assumed to be linear in nature.

- Simple linear regression vs. multiple linear regression?

Linear Regression

- **Assumptions of linear regression:**
  1. There must be a linear relation between independent and dependent variables.
  2. There should not be any outliers present.
  3. No heteroscedasticity
  4. Sample observations should be independent.
  5. Error terms should be normally distributed with mean 0 and constant variance.
  6. Absence of multicollinearity and auto-correlation.

Terminologies

- **Multicollinearity:** when the independent variables are highly correlated to each other then the variables are said to be multicollinear. Many types of regression techniques assumes multicollinearity should not be present in the dataset. It is because it causes problems in ranking variables based on its importance.

- **Little or no auto-correlation**: Another assumption is that there is little or no autocorrelation in the data. Autocorrelation occurs when the residual errors are not independent from each other.

Terminologies

- **Homoscedasticity**: It describes a situation in which the error term (that is, the “noise” or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variables.

- **Heteroscedasticity:** When dependent variable's variability is not equal across values of an independent variable, it is called heteroscedasticity.

- **EXAMPLE**

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

A pair $(x^{(i)}, y^{(i)})$ is called a **training example**

$$h : \mathcal{X} \mapsto \mathcal{Y}$$

Linear Regression

- **Hypothesis**

$$\boxed{h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2}$$

Performance Metrics

- **Mean Absolute Error (MAE):** The Mean Absolute Error measures the average of the absolute difference between each ground truth and the predictions. Whether the predictions is 10 or 6 while the ground truth was 8, the absolute difference is 2.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|$$

Performance Metrics

- **What is the unit?**

```
|1000 – 980| = 20
|1500 – 1943| = 443
|2000 – 1239| = 761
|2500 – 2020| = 480

# The average of the error summation
(20 + 443 + 761 + 480) / 4 = 426
```

- The 426 might seem our model is performing great if people's salary ranged from $100 to $100,000,000. But if the range is from $1,000 to $2,500, 426 indicates our model is underperforming.

Performance Metrics

- **Root Mean Squared Error (RMSE):** The Root Mean Squared Error measures the square root of the average of the squared difference between the predictions and the ground truth.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Performance Metrics

```
(1000 – 980)^2 = 400
(1500 – 1943)^2 = 196249
(2000 – 1239)^2 = 579121
(2500 – 2020)^2 = 230400


# The average of the square error summation
(400 + 196249 + 579121 + 230400) / 4 = 251543


# Square root of the square error summation
Square root of $251,543 is ~$502
```

- The **RMSE is larger than the MAE**. Since the RMSE is squaring the difference between the predictions and the ground truth, any significant difference is made more substantial when it is being squared. **RMSE is more sensitive to outliers**.

Performance Metrics

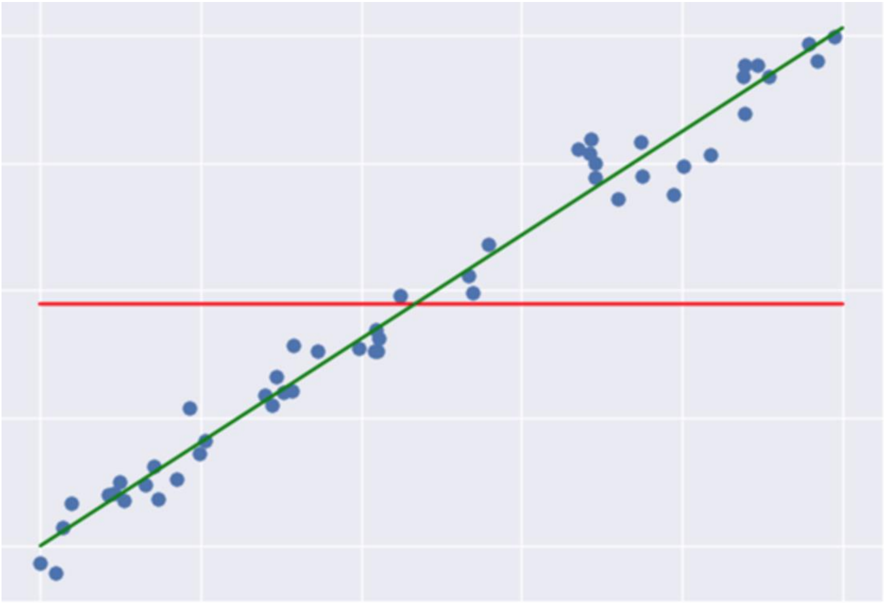
- **R-Squared:** If you like to understand how well the independent variables "explain" the variance in your model, the R-Squared formula can be powerful.

$$\hat{R}^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i-\hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i-\bar{Y})^2} = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(Y_i-\hat{Y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(Y_i-\bar{Y})^2}$$

- **R-Squared = SSR / SSTO,**
- **SSTO = SSE + SSR**

Performance Metrics

- **SSR: Regression Sum of Squares** quantifies how far the estimated sloped regression line (green line) is from the horizontal (red line). The red line is the average of the ground truth.
- **SSE: Error sum of squares** quantifies how much the data points (blue dots) is from the prediction (green line).
- **SSTO: Total Sum of Squares** quantifies how much the data points (blue dot) is from the horizontal (red line).



Performance Metrics

- **Adjusted R square:** One of the pitfalls of the R-squared is that it will always improve as we increase the number of variables.
- The Adjusted R-Squared fixes this problem. It adds a penalty to the model. Notice that there are two additional variables in the model.
- The n represents the total number of observations. The k represents the total number of variables in your model.

$$R^2_{adj} = 1 - \left[\frac{(1 - R^2)(n - 1)}{n - k - 1} \right]$$

NED University of Eng. & Tech.

Performance Metrics

- For the **R-Squared** and the **Adjusted R-Squared**, the closer the value to 1, the better performer our model!

- **RMSE/MAE** is used to evaluate the variance in the errors. Additionally, the value within itself doesn't tell you much. You must compare different models to reap from the RMSE/MAE.

- However, **R-Square/Adjusted R-Squared** doesn't need to be compared between different models.

- If the R-Squared/Adjusted R-Squared **is .10**, we can acknowledge that the model is not doing a great job.

Linear Regression

- **Implementation in Python:**
- **In one of the previous classes, we preprocessed three csv files and listed the states of US by their population density in descending order.**

- **By using the same processed data, we want to predict the population?**

- **Before we implement the regression model, please visualize the process data.**

Linear Regression

- **Implementation in Python:**

```
import matplotlib.pyplot as plt
%matplotlib inline
data=final[final.ages=='total']
data.head()
```

```
list_states=data.state.unique()
map_dict={list_states[c]:c for c in range(len(list_states))}
data['state_encode']=data.state.map(map_dict)
data['density']=data.population/data['area (sq. mi)']
```

Linear Regression

○ Implementation in Python:

```
from sklearn import linear_model, metrics
from sklearn.model_selection import train_test_split
X=data.drop(['state','state/region','ages','population'],axis=1)
Y=data.population

# splitting X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
print('train samples: ', len(X_train))
print('test samples: ', len(X_test))

# create linear regression object
reg = linear_model.LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# regression coefficients
print('Coefficients: \n', reg.coef_)

# variance score: 1 means perfect prediction, coeficient of determination (R^2)
print('Variance score: {}'.format(reg.score(X_test, y_test)))
```

Linear Regression

- **Implementation in Python:**

```
train samples:  990
test samples:  248
Coefficients:
 [  5.59345301e+04   5.29117839e+00  -2.85239428e+04  -3.53991373e+02]
Variance score: 0.040114553534893779
```

```
reg.intercept_
```

```
-105909924.59864859
```

```
reg.predict(X_test)
```

```
array([  5249467.68100861,   5306063.8486217 ,   6206416.51046818,
         5396343.18908967,   6747706.854691  ,   5579367.8340134 ,
         6485167.08916394,   4659717.57743095,   6220594.08952849,
```

Linear Regression

- **Implementation in Python:**

```
X_test.iloc[0]
```

```
year                2000.000000
area (sq. mi)      53821.000000
state_encode          33.000000
density              150.157262
Name: 1602, dtype: float64
```

```
np.sum(X_test.iloc[1]*np.asarray(reg.coef_))+reg.intercept_
```

```
5306063.8486216962
```

- **Please write the hypothesis function for the above problem?**

- **Collect the data for Puerto Rico and Alabama state only and apply linear regression.**

Linear Regression

- **Implementation in Python:**

```
data2=final[((final.state=='Puerto Rico') | (final.state=='Alabama')) & (final.ages=='total')]
data2['density']=data2.population/data2['area (sq. mi)']
data2.head()
```

LW Linear Regression

- **Locally weighted linear regression:**

In the original linear regression algorithm, to make a prediction at a query point x (i.e., to evaluate $h(x)$), we would:

1. Fit θ to minimize $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$.
2. Output $\theta^T x$.

In contrast, the locally weighted linear regression algorithm does the following:

1. Fit θ to minimize $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$.
2. Output $\theta^T x$.

A fairly standard choice for the weights is[4]

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

LW Linear Regression

- **LWLR** is an example of a **non-parametric** algorithm.

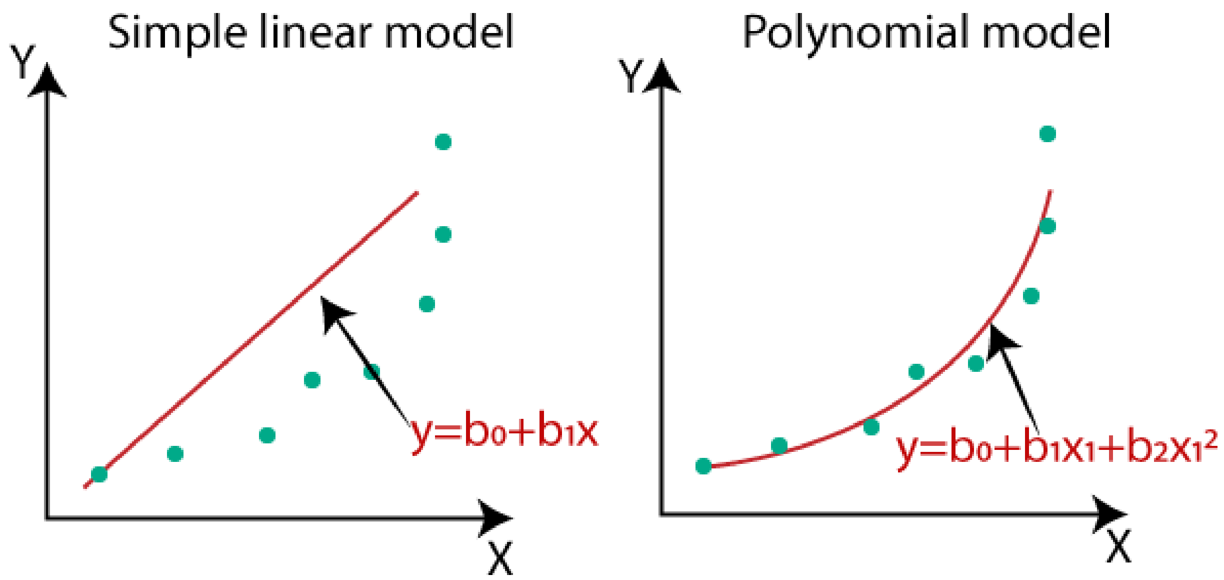
- The **(unweighted) LR** is known as a parametric learning algorithm, because it has a fixed, finite number of parameters (the θi’s), which are fit to the data. Once we’ve fit the θi’s and stored them away, we no longer need to keep the training data around to make future predictions.

- In contrast, to make predictions using **LWLR**, we need to keep the entire training set around.

- The term “non-parametric” (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the **hypothesis “h”** grows linearly with the size of the training set.

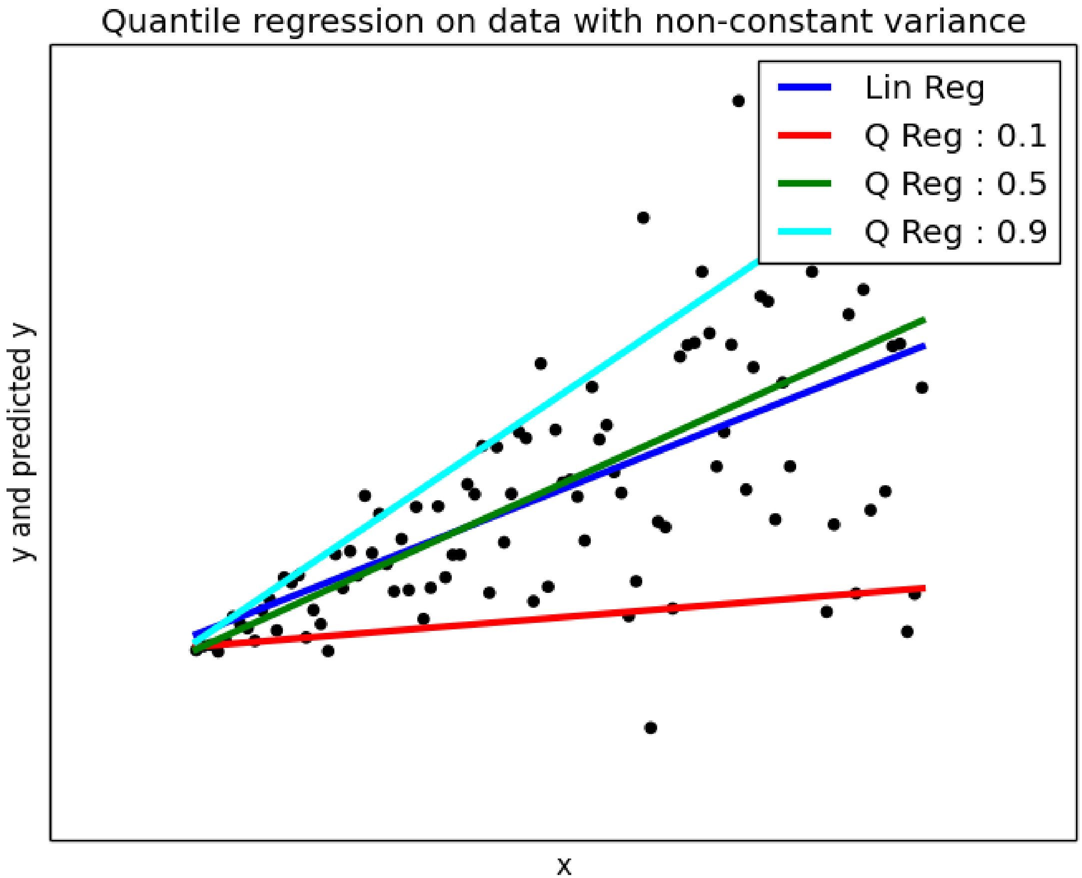
Polynomial Regression

- It is a technique to fit a nonlinear equation by taking polynomial functions of independent variable.

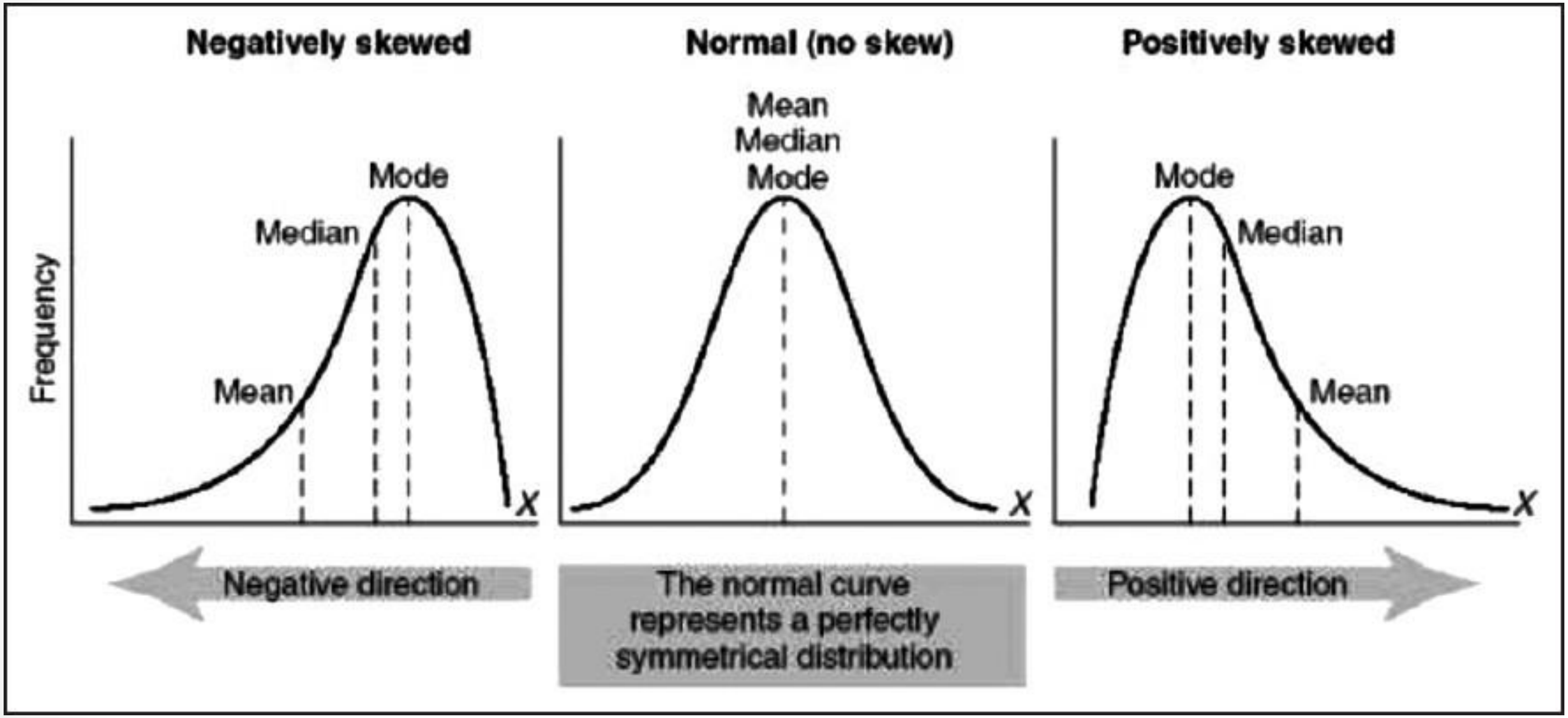


Quantile Regression

- **Quantile regression** is the extension of linear regression and we generally use it when **outliers, high skeweness and heteroscedasticity** exist in the data.

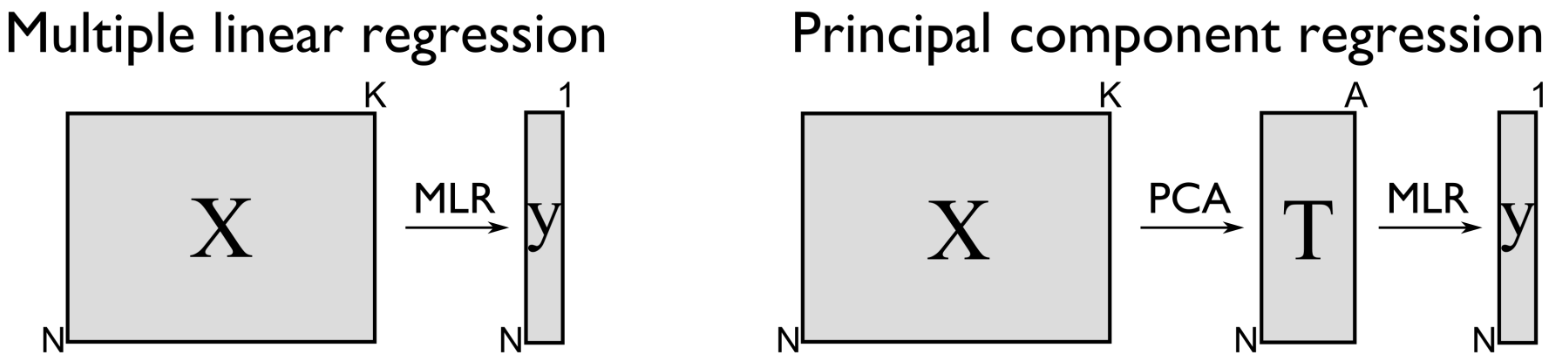


Quantile Regression



PC Regression

- **Principal Component Regression:** PCR is a regression technique which is widely used when you have **many independent variables or multicollinearity exist in your data.**



- The most common features of PCR are:
  - Dimensionality Reduction
  - Removal of multicollinearity

PC Regression

- **Drawbacks:**
- It is to be mentioned that PCR is **not a feature selection** technique instead it is a **feature extraction technique**.
- **Each principle component** we obtain is a **function of all the features.**
- Hence on using principal components one would **be unable to explain** which **factor is affecting the dependent variable** to what extent.

SV Regression

- **Support vector regression** can solve both linear and non-linear models.
- SVM uses **non-linear kernel functions** (such as polynomial) to find the optimal solution for non-linear models.