LAPORAN TUGAS 1 : [INDIVIDU] LINEAR (POLYNOMIAL) REGRESSION

Linear Regression adalah salah satu algoritma yang digunakan data science dan tergolong pada algoritma Supervised Learning. Algoritma ini menggunakan prinsip regresi. Regresi membuat model prediksi untuk target variabel berdasarkan dari variabel bebasnya. Jenis algoritma ini sering digunakan untuk mencari hubungan antara variabel-variabel yang ada dan prediksinya. Sehingga pada linear regression, ditujukan untuk melakukan prediksi pada variabel terkait (y) berdasarkan variabel bebas yang diberikan (x). hasil akhirnya berupa hubungan linear antara variabel input (x) dengan variabel output (y).

Linear Regression dibedakan menjadi dua jenis berdasarkan jumlah variabel terikatnya. Pertama **Univariate Linear Regression** yaitu jumlah variabel terikatnya hanya ada satu (satu atribut/fitur). Kedua **Multivariate Linear Regression** yaitu jumlah variabel terikatnya lebih dari satu (beberapa atribut/fitur) menjadi satu nilai output (target) berupa nilai real.
Berikut Model yang dihasilkan dari kedua jenis Linear Regression:

1. **Univariate Linear Regression** (satu atribut/fitur)

$$f: \Re^1 \to \Re \quad f(x; w) = w_0 + w_1 x$$

2. **Multivariate Linear Regression** (beberapa atribut/fitur)

$$f: \Re^d \to \Re \quad f(x; w) = w_0 + w_1 x_1 + \cdots + w_d x_d$$

Keterangan:

- $f(x; w)$ atau y adalah variabel output
- x adalah variabel input atau nilai atribut/fitur
- w adalah bobot atau *coefficient*
- w_0 adalah bias atau intercept

Proses Training model Linear Regression dibedakan menjadi dua jenis yaitu **Direct/Matrix Equation** dan **Gradient Descent (Stochastic/Online Learning atau Batch/Offline Learning)**

1. **Direct/Matrix Equation**
    a) Model Univariate Linear Regression

    Linear Regression : $y = w_0 + w_1 x$

    Bobot dan Bias dapat dihitung melalui persamaan **variants(x)** dan **covariants(x, y)**

$$\text{var}\,(x) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

$$\text{cov}\,(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Coefficient dan intercept dari model regresi linear adalah:

$$w_1 = \frac{\text{cov}\,(x,y)}{\text{var}\,(x)}$$

$$w_0 = \bar{y} - w_1\bar{x}$$

- Contoh Perhitungan Manual

No	Feature (x)	Target (y)		x - $\bar{x}$ (mean x)	y - $\bar{y}$ (mean y)	multiply
1	1	1		-2	-1,8	3,6
2	2	3		-1	0,2	-0,2
3	3	2		0	-0,8	0
4	4	3		1	0,2	0,2
5	5	5		2	2,2	4,4
Mean	**3**	**2,8**	**Var**	**2,5**	**Cov**	2
w1 = cov(x,y) / var(x)		w0 = $\bar{y}$ - w1$\bar{x}$	y = w0 + w1x			
0,8		0,4	1.2x			

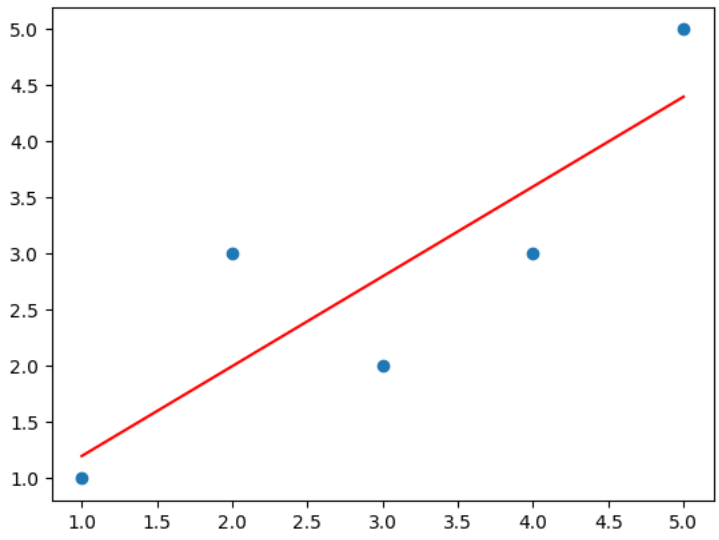
- Hasil uji coba dan Analisa
  - Mean x = 3
  - Mean y = 2.8
  - Variants(x) = 2.5
  - Covariant(x, y) = 2
  - W0 = 0.4
  - W1 = 0.8

- Contoh Implementasi Python dan Hasil

```
import numpy as np
import matplotlib.pyplot as plt

class LinearRegressionDirectUnivariate:
    def __init__(self):
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        meanX = np.mean(X)
        meanY = np.mean(y)
        varX = np.var(X, ddof=1)
```

```
        covXY = np.cov(X, y)
        dataCov = covXY[0,1]
        self.weights = dataCov / varX
        self.bias = meanY - self.weights * meanX

    def predict(self, X):
        return self.bias + self.weights * X

    def display_model(self, X, y):
        plt.plot(X, y, 'o')
        dataX = np.linspace(np.min(X), np.max(X), 20)
        dataY = self.bias + self.weights * dataX
        plt.plot(dataX, dataY, 'r-')
        plt.show()

X = np.array([1, 2, 3, 4, 5])
Y = np.array([1, 3, 2, 3, 5])

linearRegressionModel =
LinearRegressionDirectUnivariate()
linearRegressionModel.fit(X, Y)
print("Intercept:", linearRegressionModel.bias)
print("Koefisien:", linearRegressionModel.weights)
print("Mean X:", np.mean(X))
print("Covariance:", np.cov(X, Y)[0,1])
print("Variance:", np.var(X, ddof=1))
linearRegressionModel.display_model(X, Y)
```

```
Intercept: 0.39999999999999947
Koefisien: 0.8
Mean X: 3.0
Covariance: 2.0
Variance: 2.5
```



2. Gradient Descent

a) Stochastic Gradient Descent (online learning)

$$w_j = w_j - \eta\left(f\left(x^i\right) - y^i\right)x_j^i$$

- Contoh Perhitungan Manual

Stochastic Gradient Descent - Data Multivariate								
Data:								
No	**Feature (x1)**	**Feature (x2)**	**Target (y)**					
1	1	2	1					
2	2	3	3					
3	4	5	3					
4	3	4	2					
5	5	1	5					
Epoch 1						w0 = w0 - learnRate . 1/n . SUM(error)		
epoch = 1, += 1, w0 = 0, w1 = 0, w2 = 0, learnRate = 0.01, f(x) = w0 + w1(x) + w2(x) = 0, error = 0-y = -y								
	Ket :				w0 = w0 - learnRate . SUM(error)		0,01	Error
	- epoch = 1	- learnRate = 0.01			w1 = w1 - learnRate . error . x =		0,01	-1
	- w0 = 0	- f(x) = w0 + w1(x)			w2 = w2 - learnRate . error . x =		0,02	-1
	- w1 = 0	- error = f(x) - y						
	- w2 = 0							
epoch = 1, += 2, w0 = 0.01, w1 = 0.01, w2 = 0.02, learnRate = 0.01, f(x) = w0 + w1(x) + w2(x) = 0, error = 0-y = -y								
	Ket :				w0 = w0 - learnRate . SUM(error)		0,0391	Error
	- epoch = 1	- learnRate = 0.01			w1 = w1 - learnRate . error . x =		0,0682	-2,91
	- w0 = 0.01	- f(x) = w0 + w1(x)			w2 = w2 - learnRate . error . x =		0,1073	-2,91
	- w1 = 0.01	- error = f(x) - y						
	- w2 = 0.02							

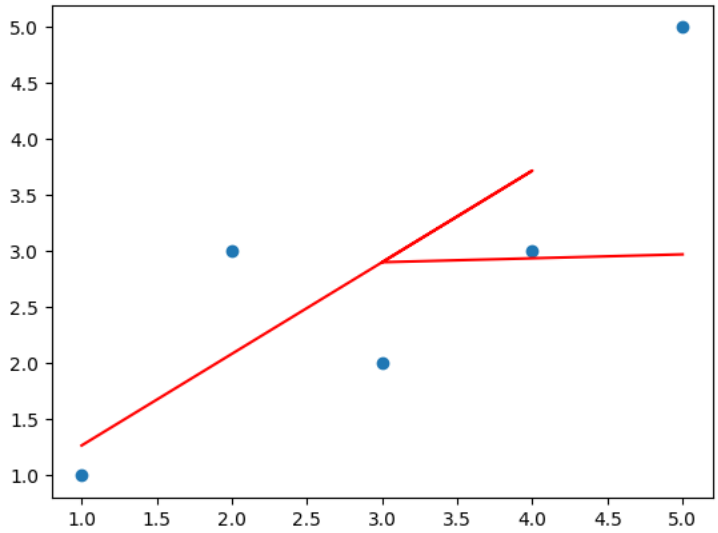
- Hasil uji coba dan Analisa
  - Pada Epoch 1 data ke-1 diperoleh w0 = 0.01, w1 = 0.01 dan w2 = 0.02
  - Pada Epoch 1 data ke-2 diperoleh w0 = 0.0391, w1 = 0.0682 dan w2 = 0.1073
- Contoh Implementasi Python dan Hasil

```
import numpy as np
import matplotlib.pyplot as plt

class LinearRegression:
    def __init__(self, learning_rate=0.01,
epochs=1000):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def fit(self, X, y):
```

```
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for epoch in range(self.epochs):
            for i in range(n_samples):
                y_predicted = np.dot(X[i],
self.weights) + self.bias

                error = y_predicted - y[i]
                dw = X[i] * error
                db = error

                self.weights -= self.learning_rate *
dw
                self.bias -= self.learning_rate * db

                w =
np.concatenate((np.array([self.bias]), self.weights))

                print('Epoch-', epoch, ' Data:', X[i],
' Error:', error, ' w:', w)

    def predict(self, X):
        y_predicted = np.dot(X, self.weights) +
self.bias
        return y_predicted

    def r_squared(self, X, y):
        y_mean = np.mean(y)
        y_predicted = self.predict(X)
        ss_tot = np.sum((y - y_mean)**2)
        ss_res = np.sum((y - y_predicted)**2)
        r2 = 1 - (ss_res / ss_tot)
        return r2

    def display_model(self, X, y):
        if X.shape[1] > 2:
            print("Tidak dapat menampilkan model
dengan lebih dari 2 fitur")
            return

        x_values = X[:, 0]

        plt.scatter(x_values, y)
        plt.plot(x_values, self.predict(X),
color='red')
        plt.show()
```

```
X = np.array([[1, 2], [2, 3], [4, 5], [3, 4], [5, 1]])
Y = np.array([1, 3, 3, 2, 5])
linearRegressionModel = LinearRegression(epochs=2,
learning_rate=0.01)
linearRegressionModel.fit(X, Y)
print("Weights:", linearRegressionModel.weights)
print("Bias:", linearRegressionModel.bias)
print("R2 score:", linearRegressionModel.r_squared(X,
Y))
linearRegressionModel.display_model(X, Y)
```

```
Epoch- 0  Data: [1 2]  Error: -1.0  w: [0.01 0.01 0.02]
Epoch- 0  Data: [2 3]  Error: -2.91  w: [0.0391 0.0682
0.1073]
Epoch- 0  Data: [4 5]  Error: -2.1516  w: [0.060616
0.154264 0.21488 ]
Epoch- 0  Data: [3 4]  Error: -0.6170719999999998  w:
[0.06678672 0.17277616 0.23956288]
Epoch- 0  Data: [5 1]  Error: -3.8297695999999997  w:
[0.10508442 0.36426464 0.27786058]
Epoch- 1  Data: [1 2]  Error: 0.025070208000000038  w:
[0.10483371 0.36401394 0.27735917]
Epoch- 1  Data: [2 3]  Error: -1.33506089472  w:
[0.11818432 0.39071516 0.317411  ]
Epoch- 1  Data: [4 5]  Error: 0.2680999395328003  w:
[0.11550332 0.37999116 0.304006  ]
Epoch- 1  Data: [3 4]  Error: 0.4715008049909759  w:
[0.11078832 0.36584613 0.28514597]
Epoch- 1  Data: [5 1]  Error: -2.7748350446559233  w:
[0.13853667 0.50458789 0.31289432]
Weights: [0.50458789 0.31289432]
Bias: 0.13853666586852148
R2 score: 0.2786911841095945
```



b) Batch Gradient Descent (batch learning

$$w_j = w_j - \eta \frac{1}{n} \sum_{i=1}^{n} \left(f(x^i) - y^i\right) x_j^i$$

- Contoh Perhitungan Manual

Batch Gradient Descent								
Data:								
No	**Feature (x1)**	**Feature (x2)**	**Target (y)**					
1	2	4	10					
2	2	5	12					
3	3	1	12					
4	4	5	16					
Epoch 1								
epoch = 1, += 1, w0 = 0, w1 = 0, w2 = 0, learnRate = 0.01, f(x) = w0 + w1(x) + w2(x) = 0, error = 0-y = -y								Error
	Ket :				Data 1 = w1 = w1 - learnRate . 1/n . error . x =		0,05	-10
	- epoch = 1	- learnRate = 0.01			Data 1 = w2 = w2 - learnRate . 1/n . error . x =		0,1	-10
	- w0 = 0	- f(x) = w0 + w1(x)			Data 2 = w1 = w1 - learnRate . 1/n . error . x =		0,06	-12
	- w1 = 0	- error = f(x) - y			Data 2 = w2 = w2 - learnRate . 1/n . error . x =		0,15	-12
	- w2 = 0				Data 3 = w1 = w1 - learnRate . 1/n . error . x =		0,09	-12
					Data 3 = w2 = w2 - learnRate . 1/n . error . x =		0,03	-12
					Data 4 = w1 = w1 - learnRate . 1/n . error . x =		0,16	-16
					Data 4 = w2 = w2 - learnRate . 1/n . error . x =		0,2	-16
						w0 = w0 - learnRate . 1/n . SUM(e	0,125	
						w1	0,36	
						w2	0,48	
Epoch 2								
epoch = 2, += 2, w0 = 0.125, w1 = 36, w2 = 48, learnRate = 0.01, f(x) = 0.125 + 0.36(x) + 0.48 (x) = ?, error = ?-y = -y								
	Ket :				Data 1 = w1 = w1 - learnRate . 1/n . error . x =		0,036175	error -7,235
	- epoch = 1	- learnRate = 0.01			Data 1 = w2 = w2 - learnRate . 1/n . error . x =		0,07235	-7,235
	- w0 = 0.125	- f(x) = w0 + w1(x)			Data 2 = w1 = w1 - learnRate . 1/n . error . x =		0,043775	-8,755
	- w1 = 0.36	- error = f(x) - y			Data 2 = w2 = w2 - learnRate . 1/n . error . x =		0,1094375	-8,755
	- w2 = 0.48				Data 3 = w1 = w1 - learnRate . 1/n . error . x =		0,0773625	-10,315
					Data 3 = w2 = w2 - learnRate . 1/n . error . x =		0,0257875	-10,315
					Data 4 = w1 = w1 - learnRate . 1/n . error . x =		0,12035	-12,035
					Data 4 = w2 = w2 - learnRate . 1/n . error . x =		0,1504375	-12,035
						w0 = w0 - learnRate . 1/n . SUM(e	0,09585	0,22085
						w1	0,2776625	0,6376625
						w2	0,3580125	0,8380125

- Hasil uji coba dan Analisa
  - Pada Epoch 1 diperoleh w0 = 0.125, w1 = 0.36 dan w2 = 0.48
  - Pada Epoch 2 diperoleh w0 = 0.22085, w1 = 0.6376625 dan w2 = 0.8380125

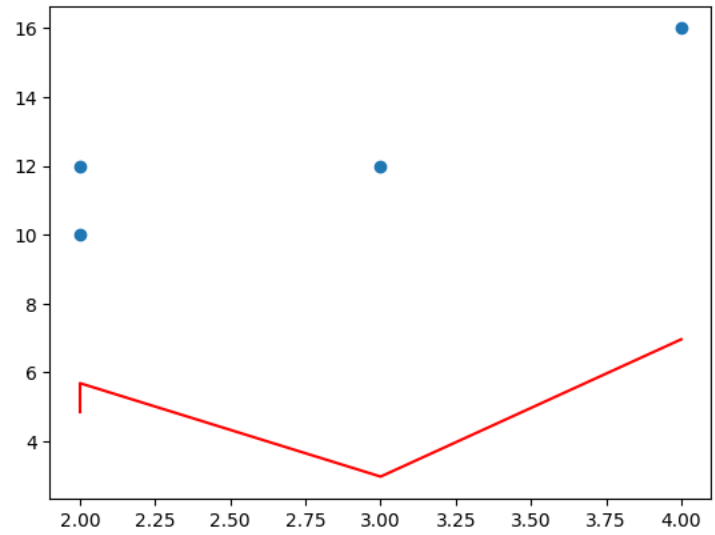
- Contoh Implementasi Python dan Hasil

```
import numpy as np
import matplotlib.pyplot as plt

class LinearRegression:
    def __init__(self, learning_rate=0.01,
epochs=1000):
        self.learning_rate = learning_rate
```

```
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for epoch in range(self.epochs):
            y_predicted = np.dot(X, self.weights) +
self.bias

            error = y_predicted - y
            dw = (1/n_samples) * np.dot(X.T, error)
            db = (1/n_samples) * np.sum(error)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

            w = np.concatenate((np.array([self.bias]),
self.weights))

            print('Epoch-', epoch, ' Data:',
np.concatenate(X), ' Error:', error, ' w:', w)

    def predict(self, X):
        y_predicted = np.dot(X, self.weights) +
self.bias
        return y_predicted

    def score(self, X, y):
        y_mean = np.mean(y)
        y_predicted = self.predict(X)
        ss_tot = np.sum((y - y_mean)**2)
        ss_res = np.sum((y - y_predicted)**2)
        r2 = 1 - (ss_res / ss_tot)
        return r2

    def display_model(self, X, y):
        if X.shape[1] > 2:
            print("Tidak dapat menampilkan model
dengan lebih dari 2 fitur")
            return

        x_values = X[:, 0]

        plt.scatter(x_values, y)
```

```
        plt.plot(x_values, self.predict(X),
color='red')
        plt.show()

X = np.array([[2, 4], [2, 5], [3, 1], [4, 5]])
Y = np.array([10, 12, 12, 16])
linearRegressionModel = LinearRegression(epochs=2,
learning_rate=0.01)
linearRegressionModel.fit(X, Y)
print("Weights:", linearRegressionModel.weights)
print("Bias:", linearRegressionModel.bias)
print("R2 score:", linearRegressionModel.score(X, Y))
linearRegressionModel.display_model(X, Y)
```

```
Epoch- 0  Data: [2 4 2 5 3 1 4 5]  Error: [-10. -12. -12.
-16.]  w: [0.125 0.36  0.48 ]
Epoch- 1  Data: [2 4 2 5 3 1 4 5]  Error: [ -7.235  -
8.755 -10.315 -12.035]  w: [0.22085   0.6376625
0.8380125]
Weights: [0.6376625 0.8380125]
Bias: 0.22085
R2 score: -11.084485653733553
```



Keterangan:

- n adalah jumlah data dalam data training
- $f(x^i)$ adalah prediksi data ke-i
- y^i adalah target output untuk data ke-i
- d adalah jumlah fitur
- x_j adalah fitur ke-j
- x_0 adalah 1 bias/intercept
- w_j adalah bobot untuk fitur ke-j

Akurasi

Perhitungan kinerja untuk mengukur seberapa akurat model yang dihasilkan untuk proses generalisasi. *R-Square* untuk mengukur model linear regresi.

Nilai *r-squared* antara 0 s.d 1, nilai 1 berarti ouput dapat diprediksi secara benar tanpa ada error.

$$SS_{\text{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

$$SS_{res} = \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$$

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Keterangan:

- y_i adalah target data ke-i
- $\bar{y}$ adalah mean target
- n adalah jumlah data
- x_i adalah data ke-i
- $f(x_i)$ adalah prediksi untuk data ke-i