Linear Regression

Amira Gaber

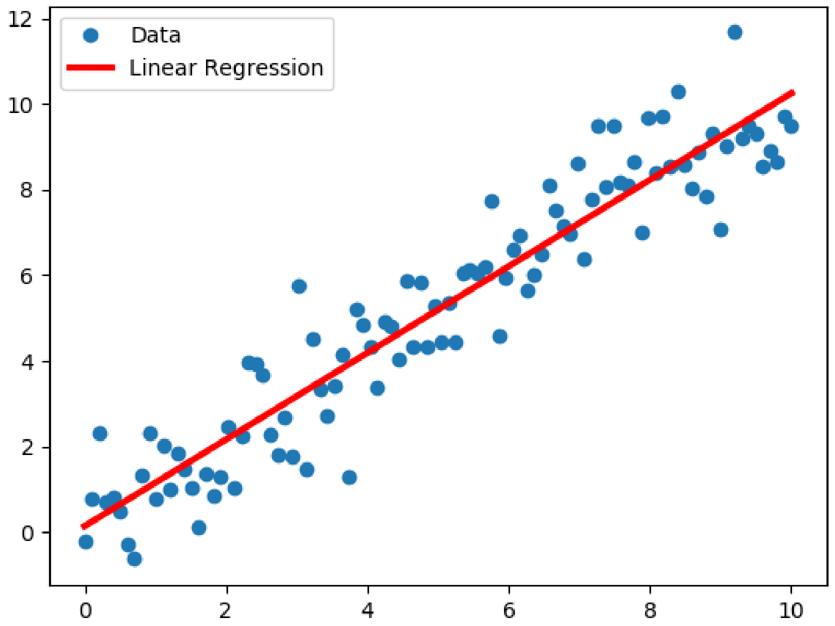
Regression

- Regression searches for relationships among variables.

- Regression problems usually have one continuous and unbounded dependent variable (y). The inputs (x_1,x_2,x_3,.....), however, can be continuous, discrete, or even categorical data such as gender, nationality, brand, and so on

- **simple linear regression** model - **multiple linear regression** -**polynomial regression**

Simple Linear Regression

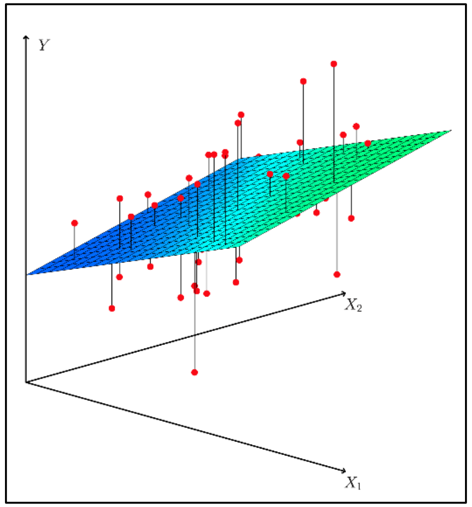


- a linear relationship between y and $\mathbf{x}$

- y = $f(x) = b_0 + b_1 x$. b_0 **intercept** & b_1 **slope**

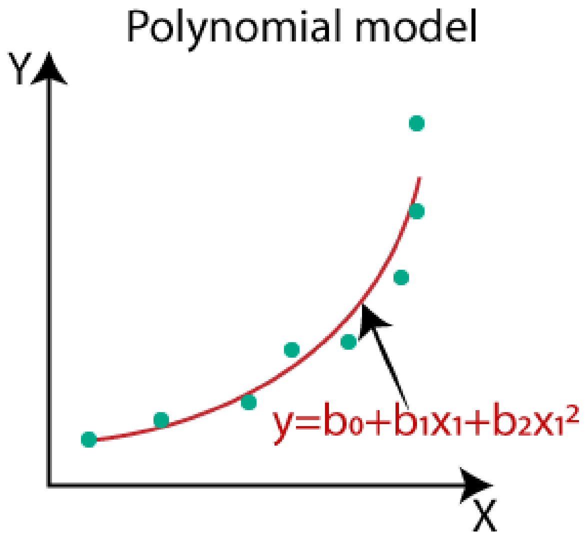
- Objective: calculate the optimal values of the predicted weights b_0 and b_1 that minimize SSR (sum of squared residuals)

Multiple Linear Regression



- a case of linear regression with two or more independent variables
- y = $f(x_1, x_2) = b_0 + b_1x_1 + b_2x_2$ (if two independent variables)
- y = $f(x_1, \ldots, x_r) = b_0 + b_1x_1 + \cdots + b_rx_r$ (r inputs)
- x is a two-dimensional array with at least two columns, while y is usually a one-dimensional array.

Polynomial Regression



- The regression function f can include non-linear terms

$b_2x_1^2$, $b_3x_1^3$, or even $b_4x_1x_2$, $b_5x_1^2x_2$

Performance Evaluation-Regression

- **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}\left|y_j - y_j\right|$$

- **Mean Squared Error** (MSE) is the mean of the squared errors.

$$MSE = \frac{1}{N}\sum_{i}^{n}(Y_i - y_i)^2$$

- **Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Coefficient of determination R^2

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y}_i)^2}$$

Normalized RSS

$$R^2 = 1 - \frac{RSS}{ns_y^2}$$

- The range of is (-∞,1)
- A larger value indicates that the model can better fit the training data.
- RSS indicates the difference between predicted value and sample value. (represents the sum of squares of the residual errors of the data model)
- TSS indicates the difference between samples. (represents the total sum of the errors.)

Parameter optimization

$$\hat{y} = \beta_0 + \beta_1 x \qquad MSE = \frac{1}{N}\sum_{i}^{n}(Y_i - \hat{y}_i)^2$$

To minimize the MSE, we take partial derivative and then set it to zero:

$$\beta_1 = s_{yx}/s_{xx}, \qquad \beta_0 = \bar{y} - \beta_1 \bar{x}.$$

where $\bar{x}$ and $\bar{y}$ are the sample means and s_{yx} and s_{xx} are the cross- and auto-covariances.

syy = $\frac{1}{N}\sum (Yi - \bar{Y})^2$
syx = $\frac{1}{N}\sum (Yi - \bar{Y})(xi - \bar{x})$
sxx = $\frac{1}{N}\sum (xi - \bar{x})^2$

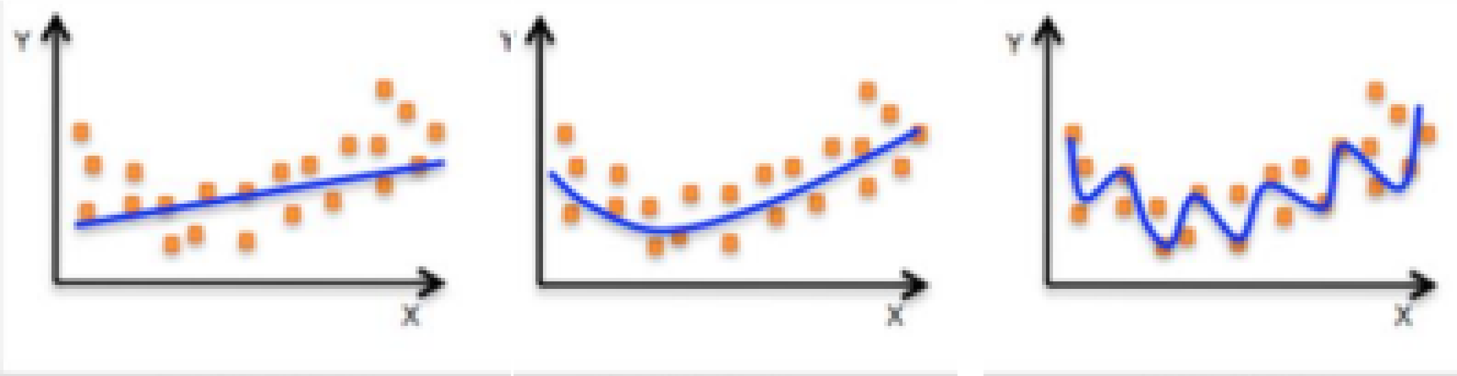
Underfitting and Overfitting

- The choice of the optimal degree of the polynomial regression function depends on the case
- **Underfitting**: a model can't accurately capture the dependencies among data

known data → low R^2 new data → bad generalization

- **Overfitting** a model learns the existing data too well & don't generalize well

known data → high R^2 new data → lower R^2



Underfitting
Not all features are learned.

Good fitting

Overfitting
Noises are learned.

Implementing Linear Regression in Python

Python Packages for Linear Regression

- **NumPy** (mathematics) allows many high-performance operations on multi-dimensional arrays.

- **Scikit-learn** (machine learning) built on top of NumPy and some other packages. It provides the means for preprocessing data, reducing dimensionality, implementing regression, classification, clustering, and more.

- **Pandas** (data manipulation) formats data into dataframes

Implementing Linear Regression in Python

Steps

1. Import packages and classes

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

2. Read a dataset, Load a dataset, OR Create a dataset

```
dataset = pd.read_csv('/content/Weather.csv')
```

Implementing Linear Regression in Python

Steps

3. Explore the dataset (dimensions, describe attributes, plotting data…)

Dataset.shape
dataset.describe()

	STA	WindGustSpd	MaxTemp	MinTemp	MeanTemp	YR	MO	DA	DR	SPD	...
count	119040.000000	532.000000	119040.000000	119040.000000	119040.000000	119040.000000	119040.000000	119040.000000	533.000000	532.000000	...
mean	29659.435795	37.774534	27.045111	17.789511	22.411631	43.805284	6.726016	15.797530	26.998124	20.396617	...
std	20953.209402	10.297808	8.717817	8.334572	8.297982	1.136718	3.425561	8.794541	15.221732	5.560371	...
min	10001.000000	18.520000	-33.333333	-38.333333	-35.555556	40.000000	1.000000	1.000000	2.000000	10.000000	...
25%	11801.000000	29.632000	25.555556	15.000000	20.555556	43.000000	4.000000	8.000000	11.000000	16.000000	...
50%	22508.000000	37.040000	29.444444	21.111111	25.555556	44.000000	7.000000	16.000000	32.000000	20.000000	...
75%	33501.000000	43.059000	31.666667	23.333333	27.222222	45.000000	10.000000	23.000000	34.000000	23.250000	...
max	82506.000000	75.932000	50.000000	34.444444	40.000000	45.000000	12.000000	31.000000	78.000000	41.000000	...

8 rows × 24 columns

4. dividing the data into “attributes” and “labels”.

```
X = dataset['MinTemp'].values.reshape(-1,1)
Y = dataset['MaxTemp'].values.reshape(-1,1)
```

.reshape() on x because this array is required to be **two-dimensional**, or to be more precise, to have **one column and as many rows as necessary**.

Implementing Linear Regression in Python

Steps

5. Split the dataset into training and test sets

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

6. Train the linear regression model with the training data

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

7. Check the results of model fitting

```
print(regressor.intercept_)  #intercept
print(regressor.coef_)       #slope
```

```
r_seq = regressor.score(X_train, y_train)   #(R²)
print('coefficient of determination: ', r_seq)
```

Implementing Linear Regression in Python

Steps

8. Apply the model for predictions using the test data.

```
y_pred = regressor.predict(X_test)
```

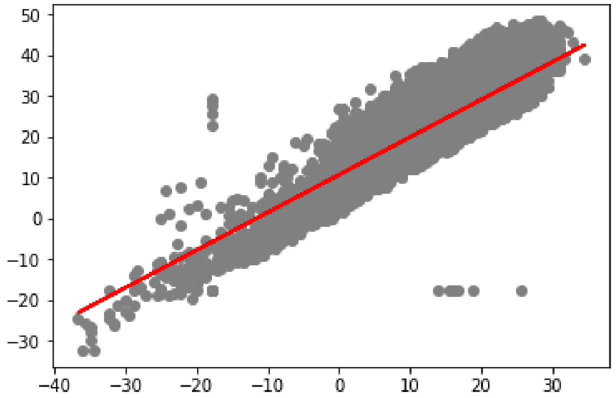
9. Compare the actual output values for the test data with the predicted values

```
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
print(df)
```

.flatten() reduce the number of dimensions to one

OR

```
plt.scatter(X_test, y_test, color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```



Implementing Linear Regression in Python

Steps

10. Evaluate the performance of the model

- Mean Absolute Error
- Mean Squared Error
- Root Mean Squared Error
- Coefficient of determination

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('Coefficient of determination:', metrics.r2_score(y_test, y_pred))
```

OR

```
r_seq = regressor.score(X_test, y_test)   #(R²)
print('coefficient of determination: ', r_seq)
```

References

- https://realpython.com/linear-regression-in-python/
- https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html
- https://towardsdatascience.com/optimization-of-supervised-learning-loss-function-under-the-hood-df1791391c82

Thank You