Lab 2. Gradient descent and Linear regression

Intro to Machine Learning
Fall 2018, Innopolis University

Lecture recap

- What is linear regression?
- Why study linear regression?
- What can we use it for?
- How to perform linear regression?
- How to estimate its performance?
- What are its extensions?
- What are dummy variables?

Questions about the lecture

Was the material already familiar to you?

What new things have you learned?

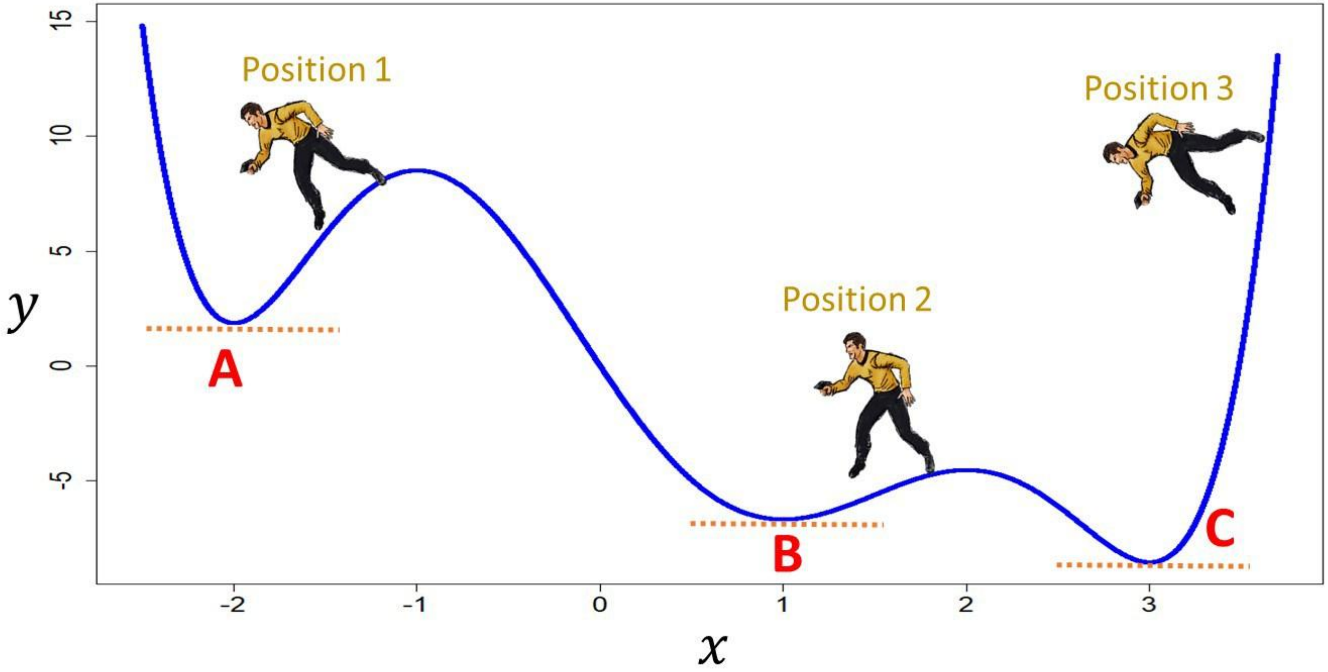
What was hard to understand?

What is p-value?
What is R^2 metric?
What is F-metric?

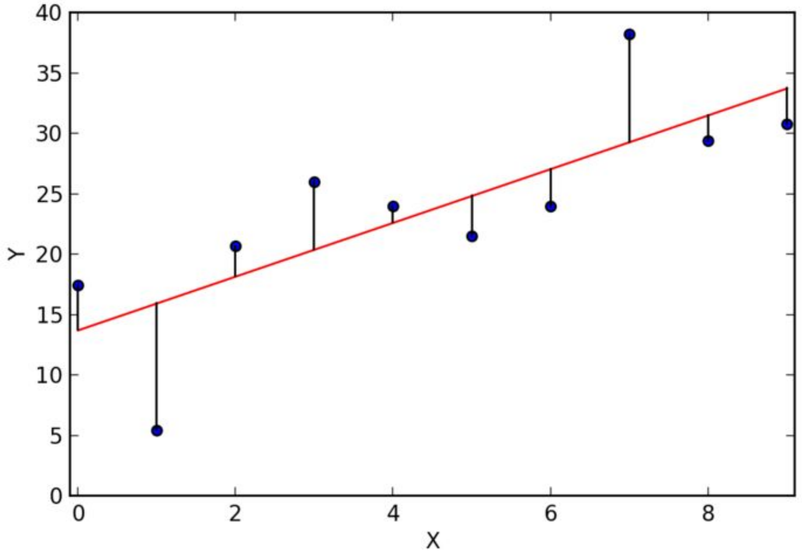
Today's plan

1. Gradient descent
    a. Introduction, purpose and derivations
    b. Types of GD
2. Linear regression
    a. How it works
    b. Implementation
3. Introduction to Scikit-learn
    a. Linear model Implementation
4. Homework explanation

Gradient descent



Linear case



$$y = \beta_1 x + \beta_0$$

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

$$J = \frac{1}{n} \sum_{i=1}^{n} (\hat{y_i} - (\beta_1 x + \beta_0))^2$$

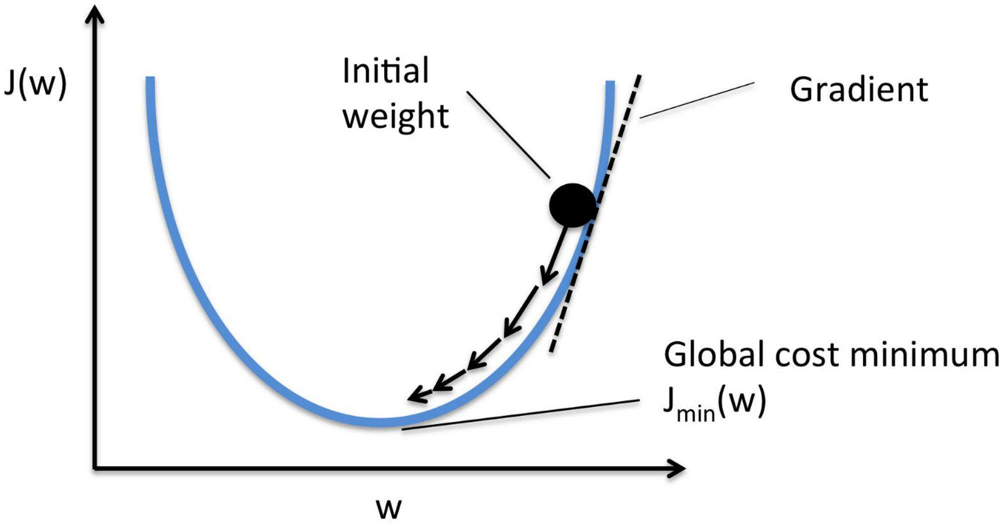
Gradient descent step

$$\frac{\partial J}{\partial \beta_1} = -\frac{1}{n}\sum_{i=1}^{n} 2x_i(\hat{y}_i - (\beta_1 x_i + \beta_0))$$

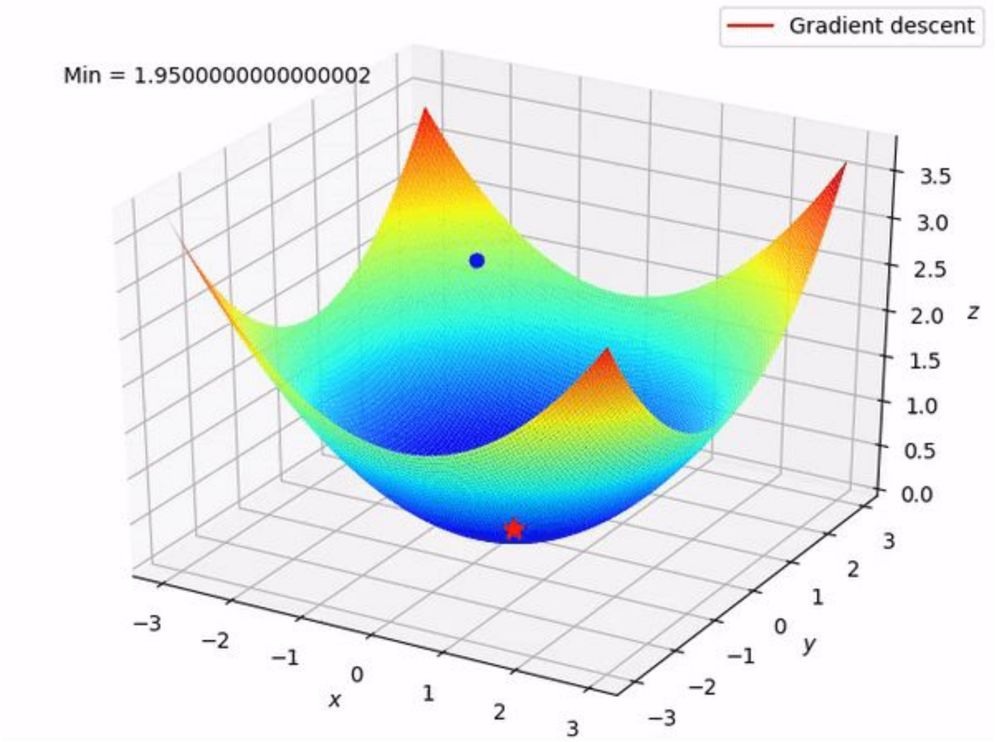
$$\frac{\partial J}{\partial \beta_0} = -\frac{1}{n}\sum_{i=1}^{n} 2(\hat{y}_i - (\beta_1 x_i + \beta_0))$$

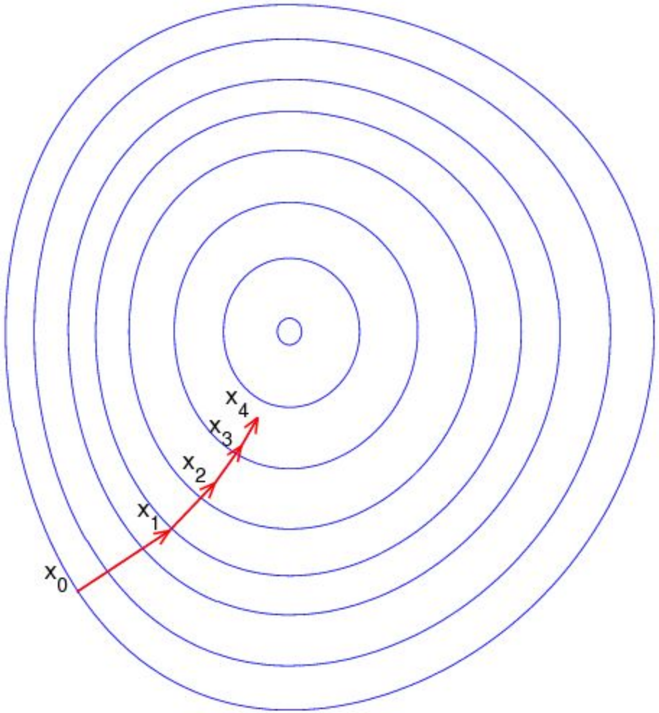
$$\beta_1 = \beta_1 - \alpha \frac{\partial J}{\partial \beta_1}$$

$$\beta_0 = \beta_0 - \alpha \frac{\partial J}{\partial \beta_0}$$



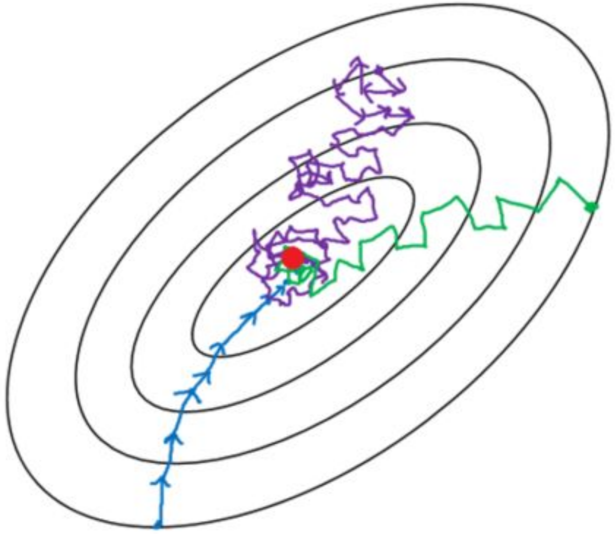
GD visualization





Different types of GD

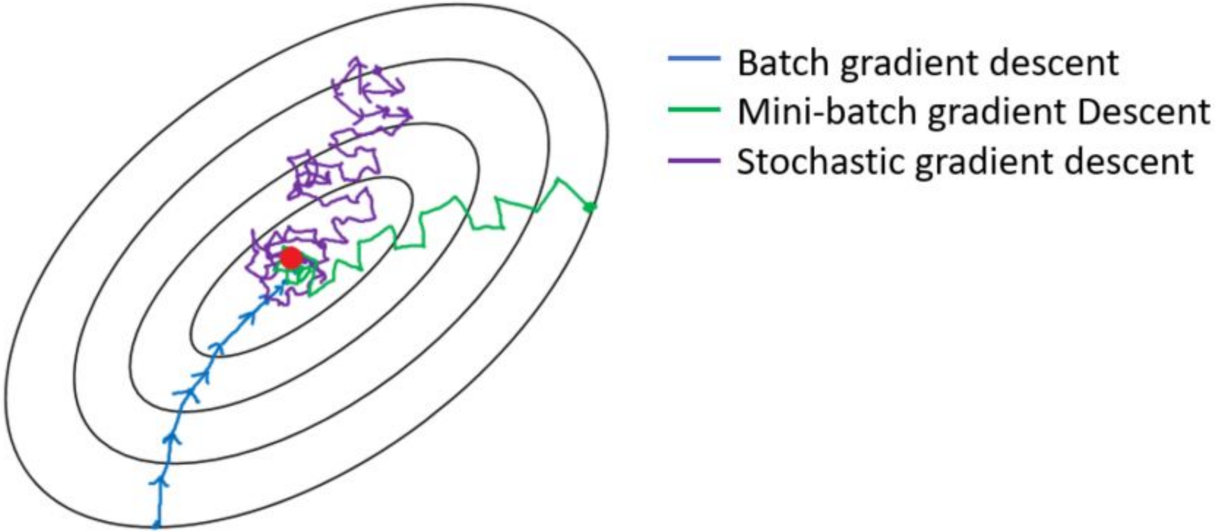
1. Batch GD
2. Mini-batch GD
3. Stochastic GD



Try to guess which is which?

Different types of GD

1. Batch GD
2. Mini-batch GD
3. Stochastic GD



Linear regression

$$RSS = e_1^2 + e_2^2 + \ldots + e_n^2$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{y}_i = \hat{\beta}_1 x_i + \hat{\beta}_0$$

$$e_i = y_i - \hat{y}_i$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Let’s find β0

$$RSS = (y_1 - \hat{\beta}_1 x_1 - \hat{\beta}_0)^2 + (y_2 - \hat{\beta}_1 x_2 - \hat{\beta}_0)^2 + \ldots + (y_n - \hat{\beta}_1 x_n - \hat{\beta}_0)^2$$

$$\frac{\partial}{\partial \hat{\beta}_0} RSS =$$

$$\frac{\partial}{\partial \hat{\beta}_0} [(y_1 - \hat{\beta}_1 x_1 - \hat{\beta}_0)^2 + (y_2 - \hat{\beta}_1 x_2 - \hat{\beta}_0)^2 + \ldots + (y_n - \hat{\beta}_1 x_n - \hat{\beta}_0)^2] =$$

$$-2(y_1 - \hat{\beta}_1 x_1 - \hat{\beta}_0) - 2(y_2 - \hat{\beta}_1 x_2 - \hat{\beta}_0) - \ldots - 2(y_n - \hat{\beta}_1 x_n - \hat{\beta}_0) =$$

$$-2 \sum_{i=1}^{n} (y_i - \hat{\beta}_1 x_i - \hat{\beta}_0) = 0$$

$$\sum y_i - \hat{\beta}_1 \sum x_i - n\hat{\beta}_0 = 0$$

$$\hat{\beta}_0 = \frac{1}{n} \sum y_i - \frac{1}{n} \hat{\beta}_1 \sum x_i$$

Try to find β1 by yourself

$$RSS = \sum_{i=1}^{n}(y_i - \bar{y} - \hat{\beta}_1(x_i - \bar{x}))^2$$

$$\frac{\partial}{\partial \hat{\beta}_1} RSS = \ldots$$

Let’s implement it

First of all, we need to generate data

```
def generate_data(b1, b0, size, x_range = (-10, 10), noise_mean = 0,
                  noise_std = 1):
    """
    input:
    b1, b0 - true parameters of data
    size - size of data, numbers of samples
    x_range - tuple of (min, max) x-values
    noise_mean - noise mean value
    noise_std - noise standard deviation

    output:
    data_x, data_y - data features
    """
    noise = np.random.normal(noise_mean, noise_std, size)
    rnd_vals = np.random.rand(size)
    data_x =  x_range[1] * rnd_vals + x_range[0]*(1-rnd_vals)
    data_y = b1 * data_x + b0 + noise

    return data_x, data_y
```

Linear regression using LS

Secondly, we need function for data visualization

```
def animate(data_x, data_y, true_b1, true_b0, b1, b0, x_range = (-10,10),
            label="Least squares"):
  plt.scatter(data_x, data_y)
  plt.plot([x_range[0], x_range[1]],
           [x_range[0]*true_b1 + true_b0, x_range[1]*true_b1 + true_b0],
           c="r", linewidth=2, label="True")
  plt.plot([x_range[0], x_range[1]],
           [x_range[0]*b1 + b0, x_range[1]*b1 + b0],
           c="g", linewidth=2, label=label)
  plt.legend()
  plt.show()
```

Linear regression using LS

Now, let's implement prediction function

```
def least_squares(x, y):
    """
    input:
    x, y - data features

    output:
    b1, b0 - predicted parameters of data
    """
    mean_x = x.mean()
    mean_y = y.mean()

    b1 = np.dot(y - mean_y, x - mean_x) / np.dot(x - mean_x, x - mean_x)
    b0 = mean_y - b1*mean_x

    return b1, b0
```

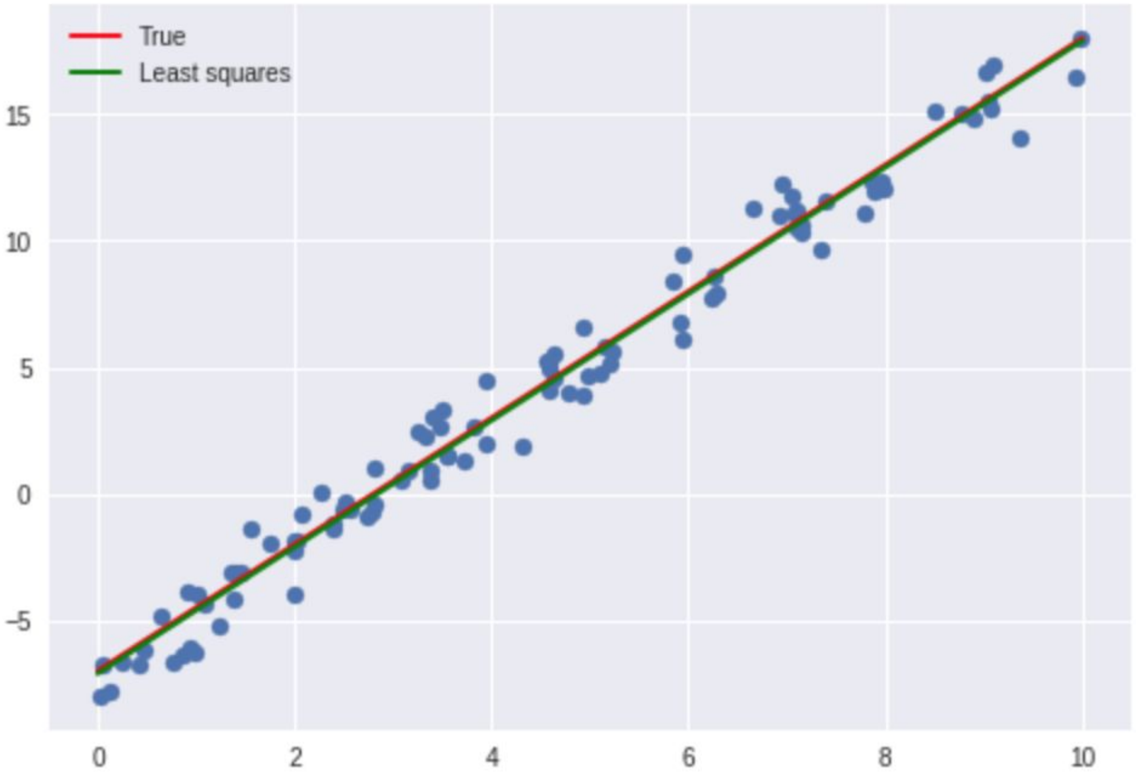
Linear regression using LS

Let’s check how it works

```
### Parameters for data generation ###
true_b1 = 2.5
true_b0 = -7
size = 100
x_range = (0,10)
noise_mean = 0
noise_std = 1

# Generate the data
data_x, data_y = generate_data(true_b1, true_b0, size,
                               x_range = x_range,
                               noise_mean = noise_mean,
                               noise_std = noise_std)

# Predict data's parameters
b1, b0 = predict(data_x, data_y)

# Visualize the data
print("true b1 : {}\ntrue b0 : {}".format(true_b1, true_b0))
print("calculated b1 : {}\ncalculated b0 : {}".format(b1, b0))
animate(data_x, data_y, true_b1, true_b0, b1, b0,x_range=x_range)
```

Linear regression using LS

What we should get



Scikit-learn implementation

```python
from sklearn.linear_model import LinearRegression

regression_model = LinearRegression()
# feed the linear regression with the train data to obtain a model.
regression_model.fit(X_train, y_train)
```

How to evaluate

R^2 metric

```
regression_model.score(X_test, y_test)
```

MSE

```
from sklearn.metrics import mean_squared_error
import math

y_predict = regression_model.predict(X_test)
regression_model_mse = mean_squared_error(y_predict, y_test)
```

How to predict

```
regression_model.predict([[4, 121, 110, 2800, 15.4, 81, 0, 1, 0]])
```

Homework explanation

On generated toy dataset

- Implement LR using GD and compare with LR based on LS
- Implement LR using Scikit-learn and compare with previous methods

On real dataset

- Implement multiple LR for the same using Scikit-learn

Let’s take a look on a dataset

mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
17.0	8	302.0	140	3449	10.5	70	1	ford torino

We want to predict how many miles will car pass on one gallon of gasoline - MPG (miles per gallone)
What do you think about the dataset? What can we do with it ?
Does it have categorical predictors ? If yes, what we could do with them ?
Does any sample has all the fields (predictor values)?

Questions

1) Is there a relationship between mpg (response) and weight (predictor)? How we could check it?
2) What is the extent to which the model fits the data?
3) Is at least one of the predictors useful in predicting the response?
4) What could you say about dataset? What should we drop?

That’s it for today! Questions?