Week 1 - 2 : Data processing and introduction to data mining

Introduction

This week introduces you to data mining and machine learning. You will learn about data wrangling and pre-processing techniques that enable you to pre-process data before the data analysis takes place. You will develop and evaluate machine learning models using linear regression and logistic regression models with gradient descent learning algorithms. You will then evaluate the performance of the models using evaluation metrics. The exercises will provide you with opportunities to reflect on your acquired knowledge and skills.

Find out about the topics covered in this week by watching the below video.

Recommended readings

We recommend you to **first go through the lessons and then see the related topics in the textbook**. Note that **the textbook readings are not mandatory.** They are meant to provide additional information for this week's lessons.

The following chapters of the course textbook will help you with the topics covered this week:

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Ed.). O'Reilly Media, Inc.

- Chapter 1: The Machine Learning Landscape
- Chapter 2: End-to-End Machine Learning Project
- Chapter 3: Classification
- Chapter 4: Training Models

Mitchell, T. (1997). Machine Learning. McGraw-Hill.

- Chapter 4: Neural Networks

This week's lessons were prepared by Dr. R. Chandra. If you have any questions or comments, please email directly: rohitash.chandra@unsw.edu.au.

Data mining vs. Machine learning

Data mining is a well-established field of computer science and statistics that focuses on the extraction of information, typically from large data sets, into some useful structure for data analysis. Data mining employs machine learning, statistics, and database systems.

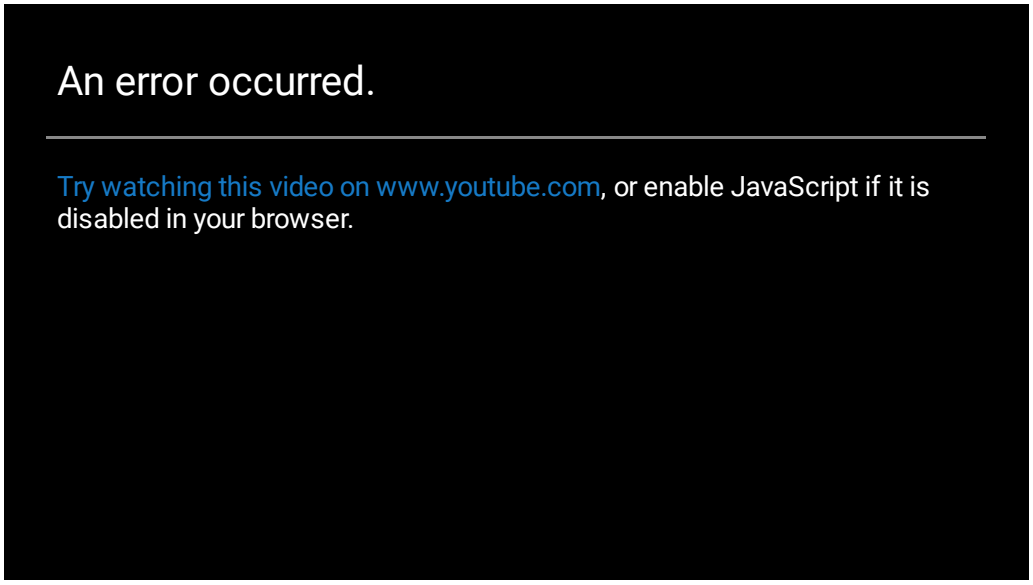
Machine learning is a field under artificial intelligence that is also a part of computer science and computational statistics. Machine learning focuses on the design and study of algorithms to build mathematical models based on data sets. Major problems of machine learning include pattern classification, control, regression and prediction, and clustering. Major machine learning methods are neural networks, support vector machines, decision trees, ensemble learning, Gaussian process and mixture models, and Markov models.

Click through the slides below to see a side by side comparison of data mining and machine learning. **Note that some of the definitions are old and wrong, and you need to identify and correct them in next slide, here: https://edstem.org/au/courses/9969/lessons/26561/slides/198663**

Source: Educba. (n.d). Difference Between Data Mining and Machine Learning. Retrieved from https://www.educba.com/data-mining-vs-machine-learning/.

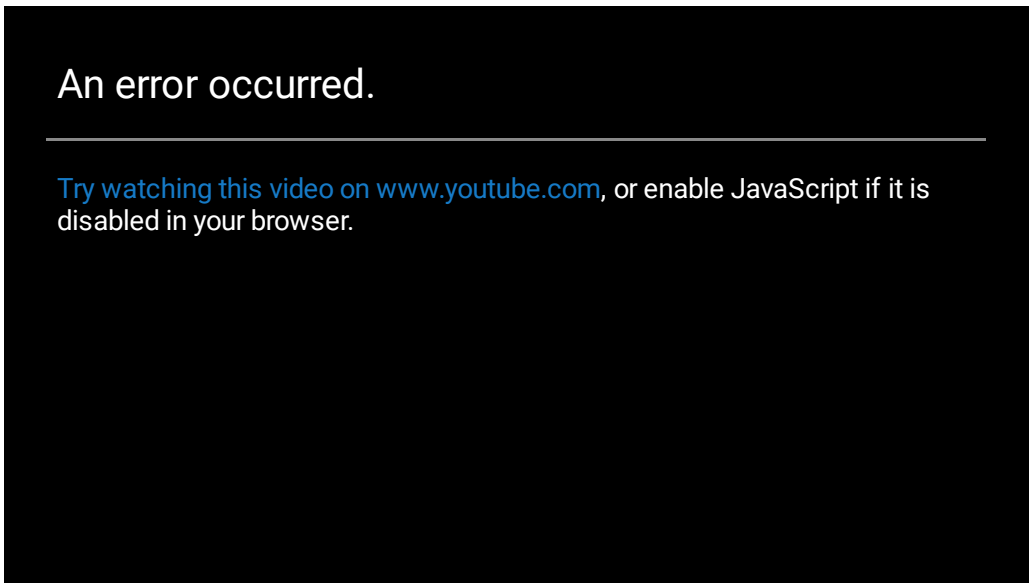
Watch the following videos on how data mining and machine learning work.

How data mining works



Thales Sehn. (2014, September 06). *How Data Mining works* [Video file]. Retrieved from https://youtu.be/W44q6qszdqY.

Introduction to machine learning



Simplilearn. (2018, September 19). *Introduction To Machine Learning* [online video]. Retrieved from https://www.youtube.com/watch?v=ukzFI9rgwfU.

Data Mining vs Machine learning

Use this workspace to compare and correct the definitions of data mining and machine learning in the previous slide.

Gradient descent

Gradient descent is a simple optimisation procedure that you can use with many machine learning algorithms. Gradient descent can only work for differentiable functions. The goal is to find the local minimum of a function using gradient descent where we take steps proportional to the **negative** of the gradient.

Consider the Rosenbrock function of two variables (dimensions):

$$f\left(x_1,\ x_2\right) = \left(1 - x_1\right)^2\ + 100\left(x_2 - \ x_1^2\right)^2$$

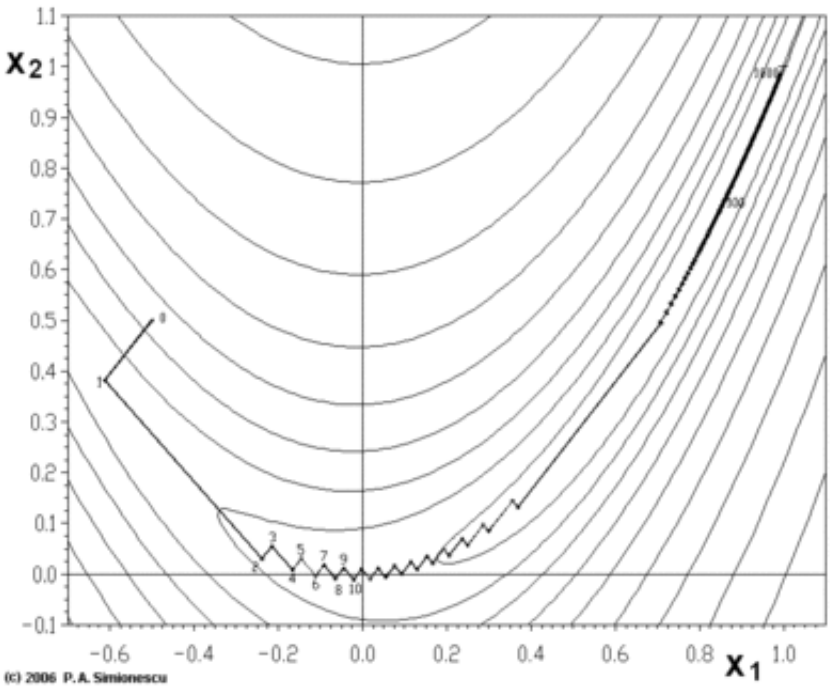


Figure: Plan view of Rosenbrock function. Adapted from "Rosenbrock function" by Wikipedia, 2020. Retrieved from: https://en.wikipedia.org/wiki/Rosenbrock_function.

The above figure shows the plan view of Rosenbrock function $f(x_1, x_2)$ vs $\ (x_1 and x_2)$ and how gradient descent traverses with the help of gradients to iteratively find the lowest point.

Now consider another function:

$$F\left(x.y\right) = \frac{1}{2}x^2\ - \frac{1}{4}y^2 + 3\ \cos\left(2x + 1 - e^y\right)$$

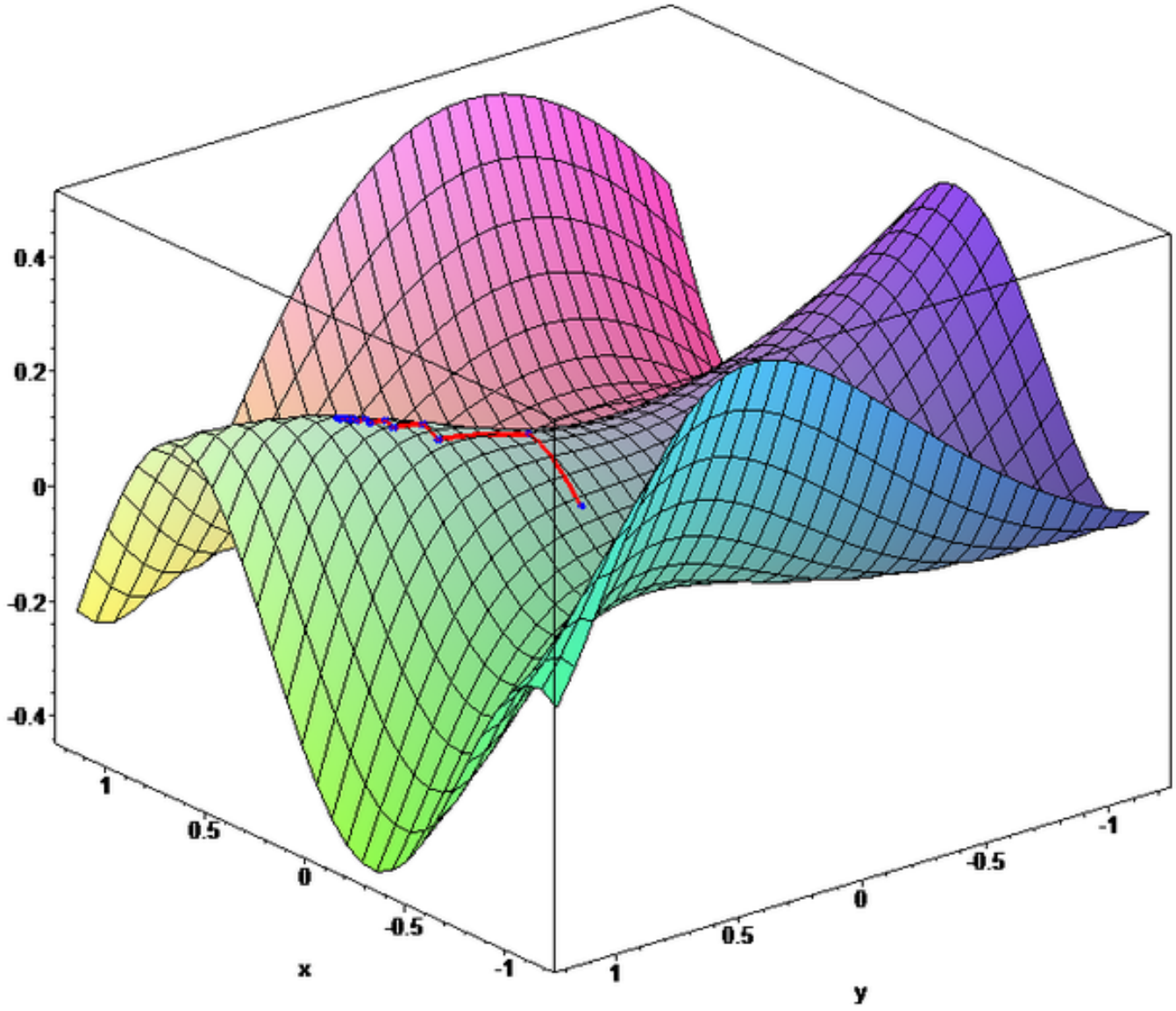


Figure: Three-dimensional view of $F(x, y)$ on the vertical axis vs parameters x and y axes. Adapted from "Rosenbrock function" by Wikipedia, 2020. Retrieved from https://en.wikipedia.org/wiki/Rosenbrock_function.

The above figure shows the three-dimensional view of $F(x, y)$ on the vertical axis vs parameters (x and y axes). Notice how gradient descent traverses with the help of gradients to iteratively find the lowest point.

The following code examples apply the gradient descent algorithm to find the minimum of the function

$$f(x) = x^4 - 3x^3 + 2$$

with derivative

$$f'(x) = 4x^3 - 9x^2$$

Solving for

$$4x^3 - 9x^2 = 0$$

and evaluation of the second derivative at the solutions shows the function has a plateau point at 0 and a global minimum at

$$x = \frac{9}{4}.$$

We note that in the above example, you can use the second derivative and solve it to find the minimum, but in more complex functions and models, that is not possible (such as Rosenbrock function and logistic regression model), hence gradient descent is used.

Practise gradient descent optimisation using Python. If necessary you may wish to go to Orientation week to refresh your Python skills.

Now that you have learned how gradient descent optimisation works, run the following codes by clicking on the 'Run' button.

```
#~source: https: //en.wikipedia.org/wiki/Gradient_descent
# code source: https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=966271567

next_x = 6# We start the search at x = 6
gamma = 0.01# Step size multiplier
precision = 0.00001# Desired precision of result
max_iters = 10000# Maximum number of iterations

# Derivative
#function
def df(x):
  return 4 * x ** 3 - 9 * x ** 2

for i in range(max_iters):
    current_x = next_x
    next_x = current_x - gamma * df(current_x)
    print(i, next_x, df(current_x))

    step = next_x - current_x
    if abs(step) <= precision:
        break

print("Minimum at ", next_x)

# The output for the above will be something like
# "Minimum at 2.2499646074278457"
```

The code above shows how gradient descent can be applied to a simple function using Python. Note the gradient of the function is computed after differentiating the function with respect to x (line 10) in the above code.

Practise gradient descent optimisation using R. If necessary you may wish to go to Orientation week to refresh your R skills.

Below, you can see the same activity completed using R. Notice the difference in the syntax of the two languages.

```
#~ source: https://en.wikipedia.org/wiki/Gradient_descent
#https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=966271567
# set up a stepsize multiplier
gamma = 0.003

# set up a number of iterations
iter = 500

# define the objective function f(x) = x^4 - 3*x^3 + 2
objFun = function(x) return(x^4 - 3*x^3 + 2)

# define the gradient of f(x) = x^4 - 3*x^3 + 2
gradient = function(x) return((4*x^3) - (9*x^2))

# randomly initialize a value to x
set.seed(100)
x = floor(runif(1, 0, 10))

# create a vector to contain all xs for all steps
x.All = numeric(iter)

# gradient descent method to find the minimum
for(i in seq_len(iter)){
        x = x - gamma*gradient(x)
        x.All[i] = x
        print(x)
}

# print result and plot all xs for every iteration
print(paste("The minimum of f(x) is ", objFun(x), " at position x = ", x, sep = ""))
plot(x.All, type = "l")
```

Although gradient descent does not work very well for Rosenbrock function, we used this example to show that gradient descent is an optimisation method, which is extended for learning data-based models such as linear regression, logistic regression, and neural networks models.

For additional information, you may wish to watch the following video on how gradient descent works. The example in the video uses a different function example.

How gradient descent works

Bielinskas.V. (2019, August 09). *How Gradient Descent works?* [online video]. Retrieved from https://www.youtube.com/watch?v=Gbz8RljxIHo.

Linear regression

This lesson will help you understand the mathematics behind gradient descent and linear regression. In a linear regression model, our goal is to find the value of the coefficients or parameters of the linear model for given data such that the output of the linear model best fits the data.

The basics of the linear regression model can be summed up by the following diagram. Note that our linear model is $y = mx + b$ which in the diagram is given as $y = w_0 + w_1 x$.

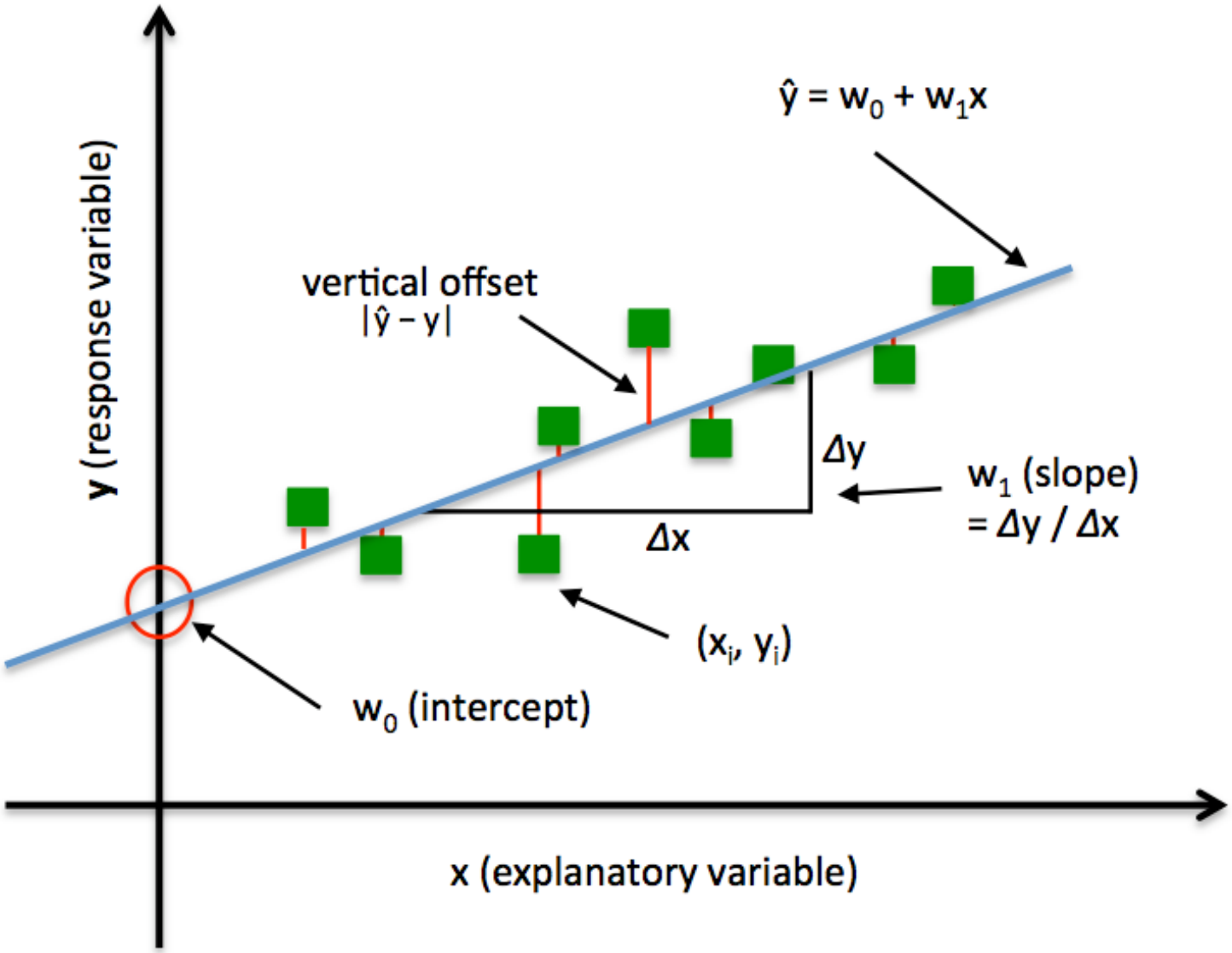


Figure: Linear regression. Adapted from "Linear regression" by Github, n.d. (http://rasbt.github.io/mlxtend/user_guide/regressor/LinearRegression/)

Now that you have learned the mathematics behind linear regression and how to apply it, let's apply linear regression using Python.

Practise linear regression using Python.

Below are the code snippets showing how gradient descent is used to find parameters of a linear model given some data below with visualisation. First, we consider some data that gives x with relationship y as shown below.

```
m = 1.2, b =3.4

y* = (x * 1.2) + 3.4

x, y, y*, e
32.502345269453031,31.70700584656992   32.1    0.6
53.426804033275019,68.77759598163891  64.1     4.6
61.530358025636438,62.562382297945803
47.475639634786098,71.546632233567777
59.813207869512318,87.230925133687393
55.142188413943821,78.211518270799232
52.211796692214001,79.64197304980874
39.299566694317065,59.171489321869508
48.10504169176825,75.331242297063056
52.550014442733818,71.300879886850353
45.419730144973755,55.165677145959123
54.351634881228918,82.478846757497919
44.164049496773352,62.008923245725825
58.16847071685779,75.392870425994957
56.727208057096611,81.43619215887864
48.955888566093719,60.723602440673965
44.687196231480904,82.892503731453715
60.297326851333466,97.379896862166078
45.618643772955828,48.847153317355072
38.816817537445637,56.877213186268506
66.189816606752601,83.878564664602763
65.41605174513407,118.59121730252249
47.48120860786787,57.251819462268969
41.57564261748702,51.391744079832307
51.84518690563943,75.380651665312357
59.370822011089523,74.765564032151374
57.31000343834809,95.455052922574737
63.615561251453308,95.229366017555307
46.737619407976972,79.052406169565586
50.556760148547767,83.432071421323712
52.223996085553047,63.358790317497878
35.567830047746632,41.412885303700563
42.436476944055642,76.617341280074044
58.16454011019286,96.769566426108199
57.504447615341789,74.084130116602523
45.440530725319981,66.588144414228594
61.89622268029126,77.768482417793024
33.093831736163963,50.719588912312084
```

36.436009511386871,62.124570818071781
37.675654860850742,60.810246649902211
44.555608383275356,52.682983366387781
43.318282631865721,58.569824717692867
50.073145632289034,82.905981485070512
43.870612645218372,61.424709804339123
62.997480747553091,115.24415280079529
32.669043763467187,45.570588823376085
40.166899008703702,54.084054796223612
53.575077531673656,87.994452758110413
33.864214971778239,52.725494375900425
64.707138666121296,93.576118692658241
38.119824026822805,80.166275447370964
44.502538064645101,65.101711570560326
40.599538384552318,65.562301260400375
41.720676356341293,65.280886920822823
51.088634678336796,73.434641546324301
55.078095904923202,71.13972785861894
41.377726534895203,79.102829683549857
62.494697427269791,86.520538440347153
49.203887540826003,84.742697807826218
41.102685187349664,59.358850248624933
41.182016105169822,61.684037524833627
50.186389494880601,69.847604158249183
52.378446219236217,86.098291205774103
50.135485486286122,59.108839267699643
33.644706006191782,69.89968164362763
39.557901222906828,44.862490711164398
56.130388816875467,85.498067778840223
57.362052133238237,95.536686846467219
60.269214393997906,70.251934419771587
35.678093889410732,52.721734964774988
31.588116998132829,50.392670135079896
53.66093226167304,63.642398775657753
46.682228649471917,72.247251068662365
43.107820219102464,57.812512976181402
70.34607561504933,104.25710158543822
44.492855880854073,86.642020318822006
57.50453330326841,91.486778000110135
36.930076609191808,55.231660886212836
55.805733357942742,79.550436678507609
38.954769073377065,44.847124242467601
56.901214702247074,80.207523139682763
56.868900661384046,83.14274979204346
34.33312470421609,55.723489260543914
59.04974121466681,77.634182511677864
57.788223993230673,99.051414841748269
54.282328705967409,79.120646274680027
51.088719898979143,69.588897851118475
50.282836348230731,69.510503311494389
44.211741752090113,73.687564318317285
38.005488008060688,61.366904537240131
32.940479942618296,67.170655768995118
53.691639571070056,85.668203145001542

```
68.76573426962166,114.85387123391394
46.230966498310252,90.123572069967423
68.319360818255362,97.919821035242848
50.030174340312143,81.536990783015028
49.239765342753763,72.111832469615663
50.039575939875988,85.232007342325673
48.149858891028863,66.224957888054632
25.128484647772304,53.454394214850524
```

Suppose we want to model a given set of points in data with a line (the linear regression model). We use the linear model $(y = mx + b)$ equation, where m is the line's slope and b is the line's y-intercept. We need to find the best set of slope m and y-intercept b values that cover the data points. Note that this example considers only one feature of the input data (x), but linear models can take a vector or multiple input features.

$$E\left[\vec{w}\right] \equiv \frac{1}{2}\sum_{d\in D}\left(t_d - o_d\right)^2$$

The error function above considers the target labels t_d over data D given output of the lineal model O_d over set of parameters w that in our case is denoted by (m, b), hence also given as:

$$\text{Error}_{(\text{m,b})} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

It takes in a (m, b) pair and returns an error value based on how well the line fits the data. Hence, we iterate through each (x, y) point in our data set and sum the square distances between each point's y value and the candidate line's y value (computed at $mx + b$) as shown in the code below.

```
# Source: https://github.com/mattnedrich/GradientDescentExample
# y = mx + b
# m is slope, b is y-intercept
def compute_error_for_line_given_points(b, m, points):
    totalError = 0
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        totalError += (y - (m * x + b)) ** 2
    return totalError / float(len(points))
```

Next, we need to find a way to adjust the parameters (m, b) so that the line better fits the data over time which is done by gradient descent. Before we proceed with adjusting parameters, we need to calculate the gradient for (m, b) given as (new m, new b) for the entire set of points, with given x and y data features as shown in the equation below.

$$\frac{\partial}{\partial \mathrm{m}} = \frac{2}{N}\sum_{i=1}^{N} - x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial \mathrm{b}} = \frac{2}{N}\sum_{i=1}^{N} - (y_i - (mx_i + b))$$

The learning rate is a user-defined parameter for gradient descent which is used to determine the extent of the steps that need to be taken when adjusting the parameters during training (shown in Lines 11 and 12 of code below).

```
# Source: https://github.com/mattnedrich/GradientDescentExample
def step_gradient(b_current, m_current, points, learningRate):
    b_gradient = 0
    m_gradient = 0
    N = float(len(points))
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        b_gradient += -(2/N) * (y - ((m_current * x) + b_current))
        m_gradient += -(2/N) * x * (y - ((m_current * x) + b_current))
    new_b = b_current - (learningRate * b_gradient)
    new_m = m_current - (learningRate * m_gradient)
    return [new_b, new_m]
```

Now that we have the gradients, we use it to update the parameters given the defined number of iterations set by the user as shown below.

```
def gradient_descent_runner(points, starting_b, starting_m, learning_rate, num_iterations):
    b = starting_b
    m = starting_m
    for i in range(num_iterations):
        b, m = step_gradient(b, m, array(points), learning_rate)
    return [b, m]
```

Next, we place all the above code snippets together by calling them when in need. Below is the key function that sets up the problem by reading data and calling the previous functions that implement the linear regression model by training using gradient descent function. We need to set a value for initial b and m, and in this case its 0, but it can also be randomly assigned.

```
def run():
    points = genfromtxt("data_linearreg.csv", delimiter=",")
    learning_rate = 0.0001
    initial_b = 0 # initial y-intercept guess
    initial_m = 0 # initial slope guess
    num_iterations = 1000
    print ("Starting gradient descent at b = {0}, m = {1}, error = {2}".format(initial_b, initial_m
    print ("Running...")
    [b, m] = gradient_descent_runner(points, initial_b, initial_m, learning_rate, num_iterations)
    print ("After {0} iterations b = {1}, m = {2}, error = {3}".format(num_iterations, b, m, comput
```

A demo of the code is shown below that highlights how the line (in red) iteratively fits into the data (in blue) by updating the slope m and the intercept b. Notice how the slope and the intercept change over time. Note that arbitrary values for the initial values of m and b need to be picked. The figure on the left shows how the values of the slope change over time (shown in green).

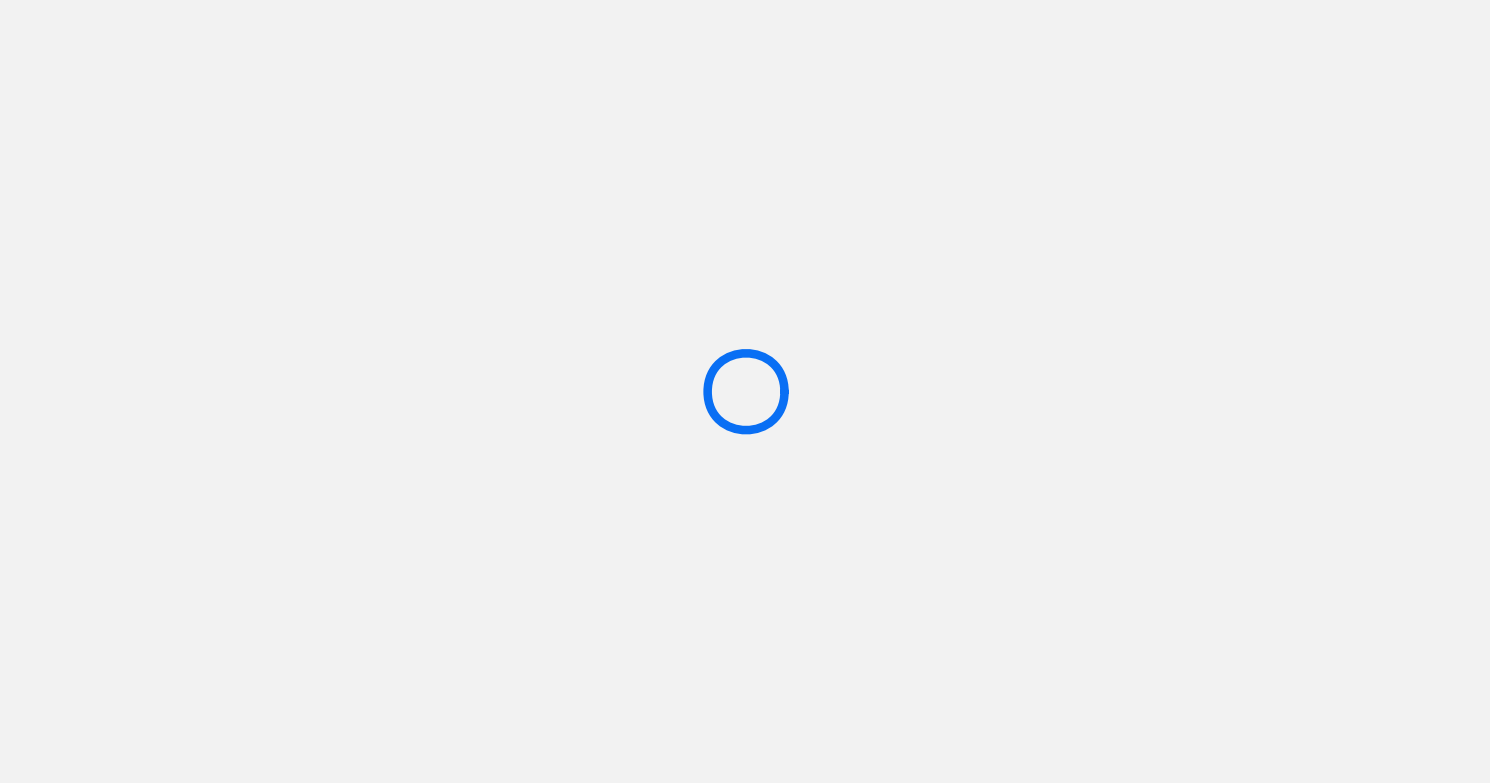


Figure: Gradient descent search on data points. Adapted from "Gradient Descent Example for Linear Regression" by Github, n.d. (https://github.com/mattnedrich/GradientDescentExample).

Note that all the above code is a preview. It is executable with data in the next lesson, Linear regression using Python.

The case of linear regression has a closed form solution that can be solved easily, hence gradient descent is not the best way to solve such a simple linear regression problem. However, gradient descent forms the basics of learning for machine learning and neural networks. Hence, this example was discussed so that it helps you to understand more complex models such as neural networks (week 3). In practice, gradient descent becomes more useful when you have hundreds of parameters instead of two parameters (m and b) considered in the above example. In deep learning models, you can expect millions of parameters for some large or deep neural network models.

Let's consider a case where there is not just one feature in x, but x represents a vector of features, e.g., your data set is about presence/absence of heart disease given a set of features by $x1, x2, x3$; such as age, weight, and body-mass-index.

Below you can see an example of a model where the sum of the incoming links $(w_1, w_2, w_3..w_N)$ over the inputs $(x_1, x_2, ...x_N)$ is computed and then the output (Z) goes through an activation function. A linear activation function is simply a linear regression model which is also known as the perceptron in the neural network literature.

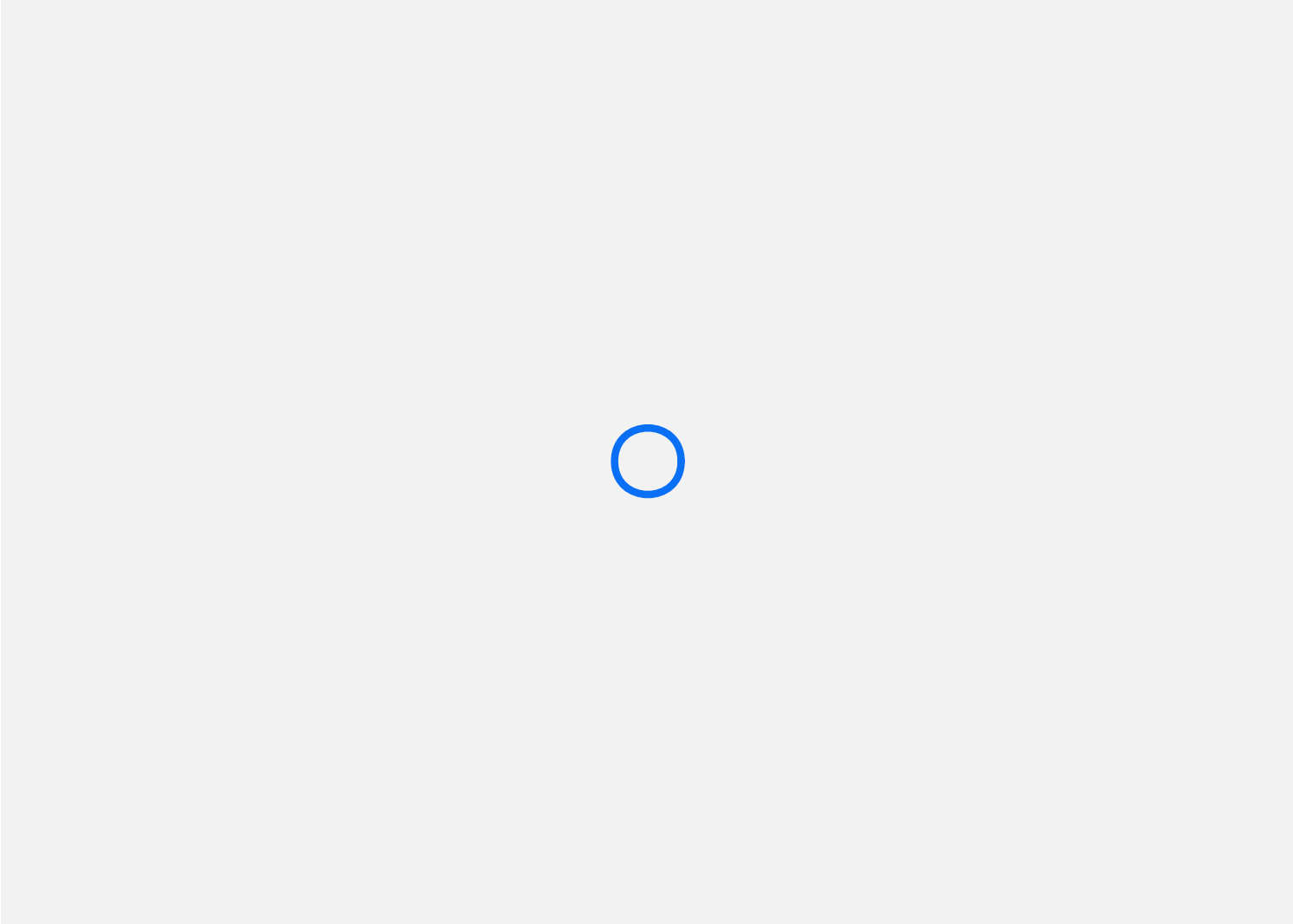


Figure: Activation function. Adapted from *Machine Learning* by T. Mitchell, 1997, Maidenhead; U.K: McGraw Hill.

Now let's examine how gradient descent can be used as a tool for finding the best parameters for a given linear model.

We need a way to capture how well the data fits the linear model, hence we use an *error or cost function* known as the sum-squared-error (E). Our goal is to adjust the parameters of the linear model $(w_0, w_1, w_2, ...w_N)$ for the given input data $(x_1, x_2, ...x_N)$ from the set of training examples denoted by D.

To understand, consider a simpler linear unit where

$$o = w_0 + w_1 x_1 + ... + w_n x_n$$

and w_i is a parameter or weight that minimises the squared error.

The error or cost function can be wrritten as

$$E\left[\vec{w}\right] \equiv \frac{1}{2}\sum_{d\in D}\left(t_d - o_d\right)^2$$

where D is a set of training examples.

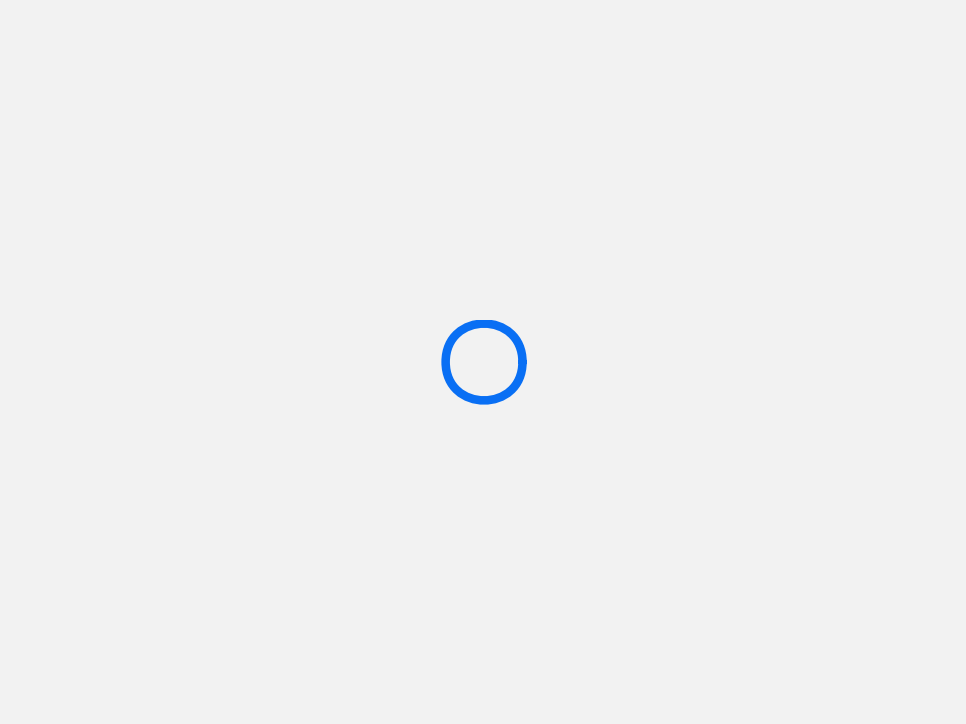


Figure: Error or cost function. Adapted from *Machine Learning* by T. Mitchell, 1997, Maidenhead; U.K: McGraw Hill.

Let's understand the derivation of the gradient descent rule. What we need to calculate is the derivative of E.

If necessary, revise partial derivatives and chain rule by clicking on the links below:

1. Tutorial 1 Partial derivatives (MathisFun, n.d.)
2. Tutorial 2 Partial derivatives (Khan Academy, n.d.)
3. Tutorial 3 Chain rule (Khan Academy, n.d.)

Gradient of E which is a vector derivative can be written as

$$[\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]]$$

The training rule for gradient decent is

$$[\Delta\vec{w} = -\eta\nabla E[\vec{w}]]$$

i.e.,

$$[\Delta w_i = -\eta \frac{\partial E}{\partial w_i}]$$

As you can see, E is differentiated with respect to the weights or parameters $w(w_1, w_1, ...w_N)$ to find a gradient to compute the parameter update, that is delta Δw. Note that the term weights are used as this linear model is one of the building blocks of simple neural networks which will be covered later.

Now let's look at the derivation for the weight update by differentiating the error function E with respect to weights w, where the model is the linear model. Below are the equations:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d \left(t_d - o_d\right)^2$$

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} \left(t_d - o_d\right)^2$$

$$= \frac{1}{2} \sum_d 2 \left(t_d - o_d\right) \frac{\partial}{\partial w_i} \left(t_d - o_d\right)$$

$$= \sum_d \left(t_d - o_d\right) \frac{\partial}{\partial w_i} \left(t_d - \vec{w}.\vec{x_d}\right)$$

$$\frac{\partial E}{\partial w_i} = \sum_d \left(t_d - o_d\right) \left(-x_{i,d}\right)$$

Now let's learn about the training examples. Each training example is a pair of the form $< \vec{x}, t >$ where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate(e.g., 0.5).

Initialise each w_i to some small random value. Until the termination condition is met, do the following:

- Initialise each $\Delta\ w_i$ to zero.
- For each pair in training examples, input the vector of input values to the unit and compute the output o. For each linear unit weight w_i ,

$$\Delta w_i \longleftarrow\ \Delta w_i\ + \eta \left(t - o\right) x_i$$

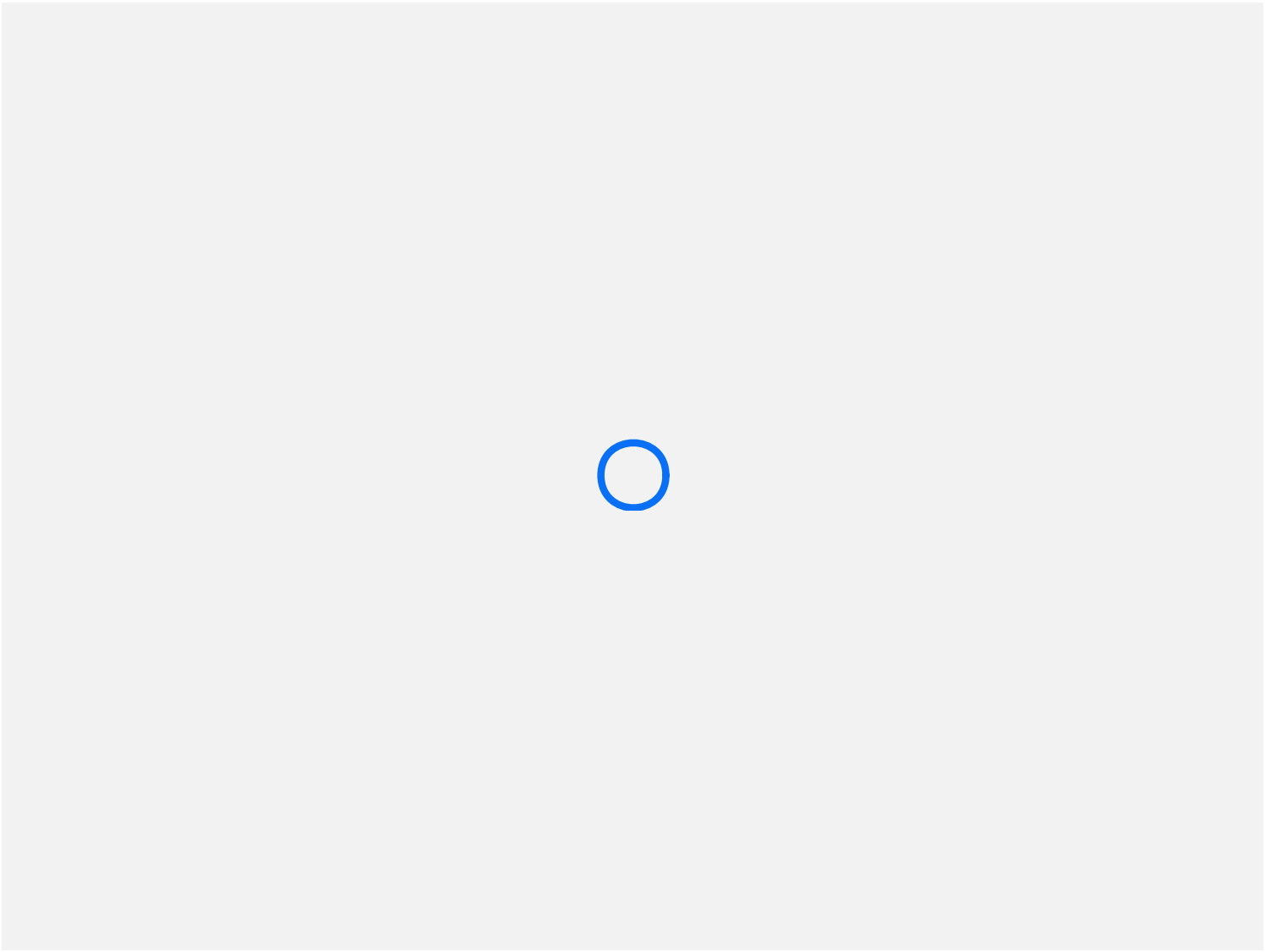
For each linear unit weight w_i,

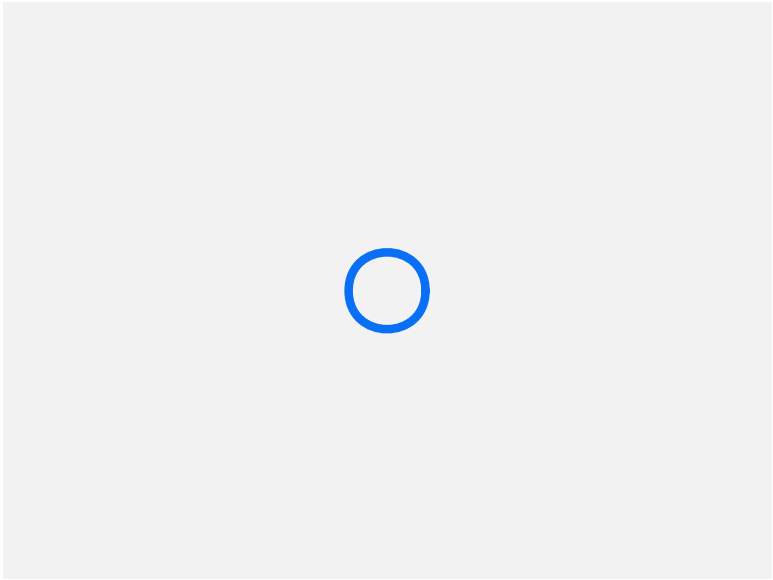
$$w_i \longleftarrow w_i + \Delta w_i$$

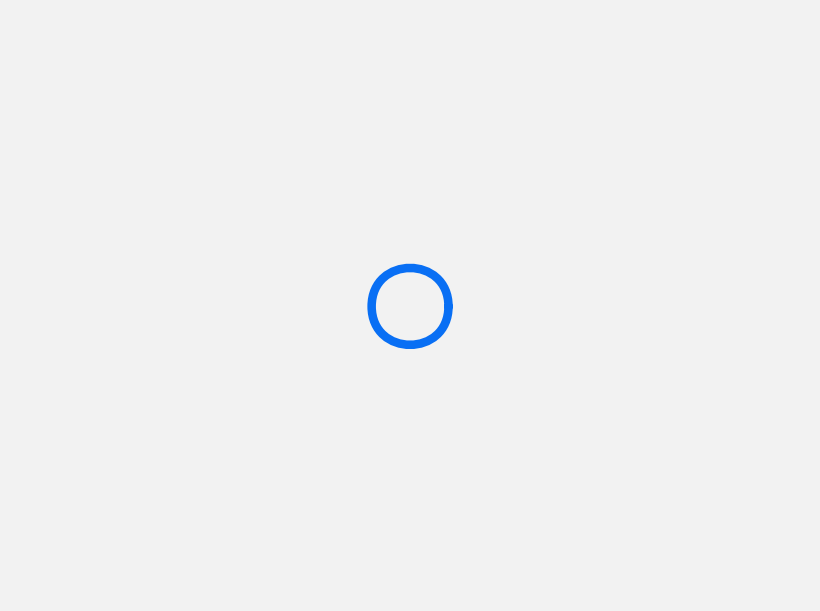
Source: Mitchell, T. (1997). *Machine Learning.* Maidenhead; U.K. McGraw Hill.

Further information: https://folk.idi.ntnu.no/keithd/classes/advai/lectures/backprop.pdf

More visualization:







Source: https://www.cs.toronto.edu/~frossard/post/linear_regression/

Linear regression using Python

The adjoining code shows how gradient descent is used to find parameters of a linear model given some data. You can examine the code by running it. You can print the variables when you run the code which will help you understand how the program updates the gradients for training the linear regression model.

The code example has been taken from the following sources:

Mattnedrich. (n.d). Github. Retrieved from https://github.com/mattnedrich/GradientDescentExample.

Nedrich, M. (2014). Atomic Object. Retrieved from https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/.

Correlation coefficient - R Score

Read the following content on the correlation coefficient and complete the associated activities.

This slide introduces you to the correlation coefficient and the difference between correlation and regression. You will also learn about R square and R score.

The correlation coefficient provides a statistical measure (given by R score) of the strength of the relationship between two variables where the R score values range between -1.0 and 1.0. The R score with a number greater than 1.0 or less than -1.0 indicates an error in the measurement. The R score of -1.0 indicates a perfect negative correlation and 1 a perfect position creation, while the R score of 0 would show no linear relationship between the two variables.

Further information about interpreting the R score is given below:

1. **Exactly –1.** A perfect downhill (negative) linear relationship
2. **–0.70.** A strong downhill (negative) linear relationship
3. **–0.50.** A moderate downhill (negative) relationship
4. **–0.30.** A weak downhill (negative) linear relationship

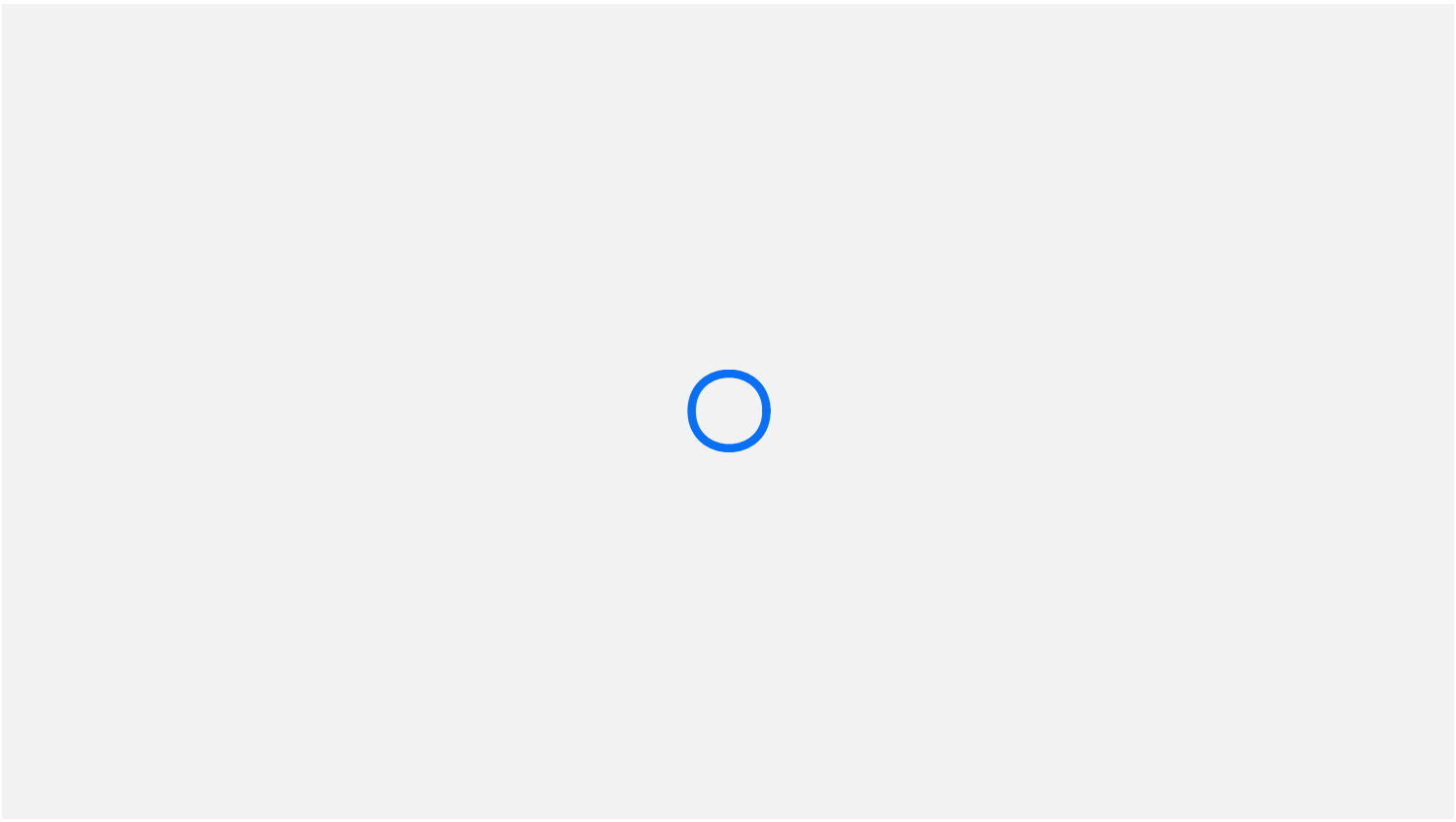
0. No linear relationship

1. **+0.30.** A weak uphill (positive) linear relationship
2. **+0.50.** A moderate uphill (positive) relationship
3. **+0.70.** A strong uphill (positive) linear relationship
4. **Exactly +1.** A perfect uphill (positive) linear relationship

Source: Rumsey, D. (n.d). How to Interpret a Correlation Coefficient r. Retrieved from https://www.dummies.com/education/math/statistics/how-to-interpret-a-correlation-coefficient-r/.

Watch the following video on correlation and regression.

Correlation & regression: Concepts with illustrative examples



Learn and apply. (2017, August 26). *Correlation & Regression: Concepts with Illustrative examples* [online video]. Retrieved from https://www.youtube.com/watch?v=xTpHD5WLuoA.

We compute the r score with equation below

$$r = \frac{\sum(x - m_x)(y - m_y)}{\sqrt{\sum(x - m_x)^2 \sum(y - m_y)^2}}$$

where x and y are two vectors and m_x and m_y represent their means, respectively. More info: http://www.sthda.com/english/wiki/correlation-test-between-two-variables-in-r

```
#source: https://stackoverflow.com/questions/3949226/calculating-pearson-correlation-and-significan

import numpy as np
def pcc(X, Y):
   ''' Compute Pearson Correlation Coefficient. '''
   # Normalise X and Y
   X -= X.mean(0)
   Y -= Y.mean(0)
   # Standardise X and Y
   X /= X.std(0)
   Y /= Y.std(0)
   # Compute mean product
   return np.mean(X*Y)
```

```
# Using it on a random example
from random import random
X = np.array([random() for x in range(100)])
Y = np.array([random() for x in range(100)])
pcof = pcc(X, Y)
print(pcof, ' is pcof')
```

The above code shows a python implementation of R score. Another code example in R is given below.

R-Squared Score

Read the following content and watch the following video on the R-squared score.

The R-squared score is another way to measure the strength of the correlation between two variables, which has more advantages as for larger or more complex linear models. R score is appropriate for simple linear regression models, where one x and one y variable are concerned, but for the case when the model becomes more complex with more than one x variable, then R squared becomes more appropriate.

R-squared is the percentage of the response variable variation that is explained by a linear model.

R-squared = Explained variation / Total variation

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

where, y and $\hat{y}$ are the actual and prediction vectors and $\bar{y}$ is the mean of y.

More information

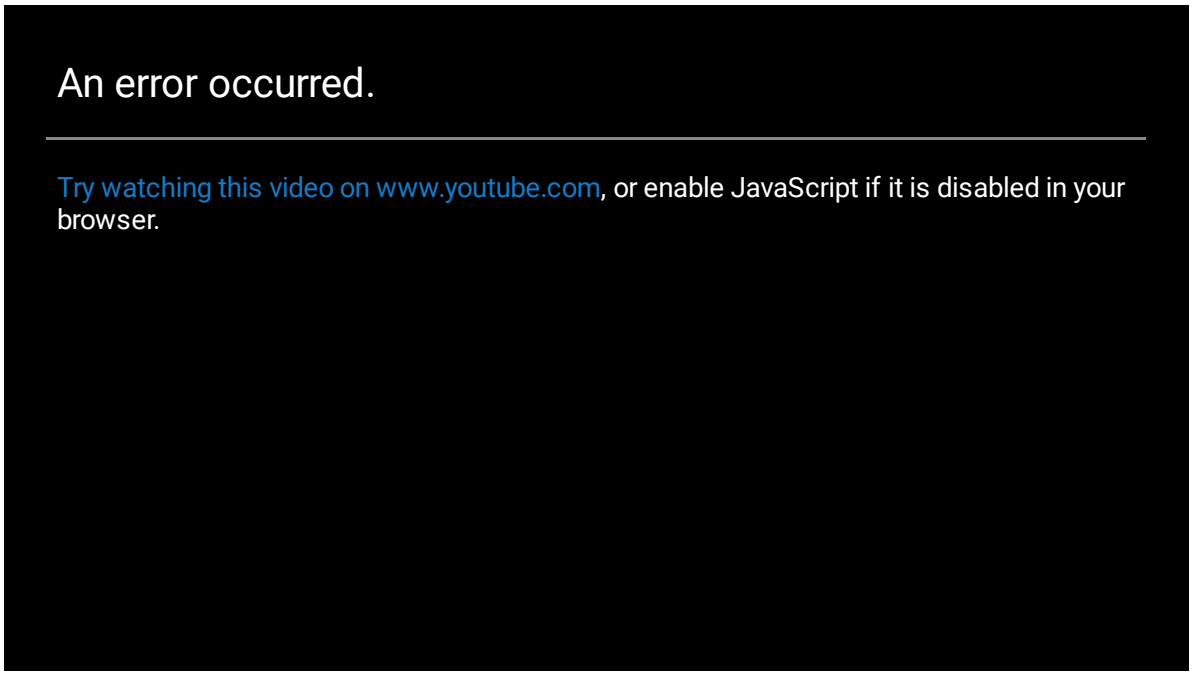
R-squared is always between 0 and 100% and:

1. 0% indicates that the model explains none of the variability of the response data around its mean.
2. 100% indicates that the model explains all the variability of the response data around its mean.

In general, the higher the R-squared, the better the model fits your data.

Minitab. (2013). Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit. Retrieved from https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit.

More information in the video below:



StatQuest. (2015, February 03). *StatQuest: R-squared explained* [online video]. Retrieved from https://www.youtube.com/watch?v=2AQKmw14mHM.

Example code is given below to demonstrate the concept using sklearn library.

```
#source for this code: https://towardsdatascience.com/r-squared-recipe-5814995fa39a
#importing matplotlib inline

import numpy as np
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
from scipy import stats

#creating data
x = np.array([0,1,2,3,4,5,6,7,8,9])
y = np.array([0,2,3,5,8,13,21,34,55,89])

#creating OLS regression
slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
def linefitline(b):
    return intercept + slope * b
```

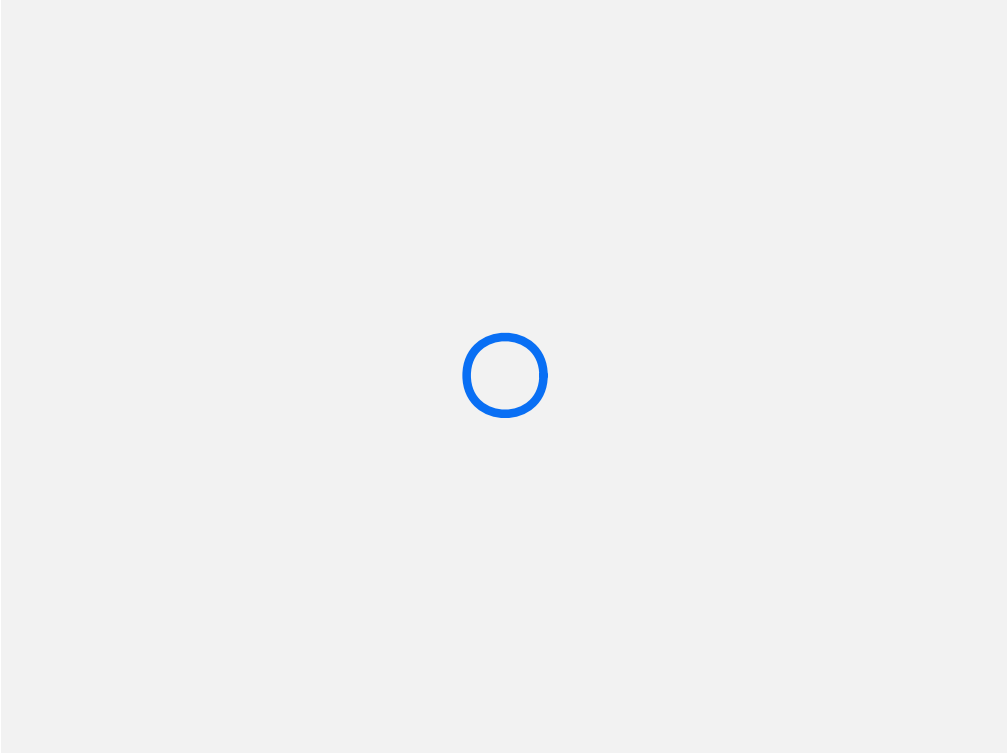
```
line1 = linefitline(x)

#plot line
plt.scatter(x,y)
plt.plot(x,line1, c = 'g')
plt.savefig('plot.png')

differences_line1 = linefitline(x)-y
print(differences_line1, ' differences ')

r2 = r2_score(y, linefitline(x))
print('The rsquared value is: ' + str(r2))
```

The output of code above is given below.



```
#source https://stackoverflow.com/questions/893657/how-do-i-calculate-r-squared-using-python-and-nu

from __future__ import print_function, division
import sklearn.metrics
import numpy as np

def compute_r2_weighted(y_true, y_pred, weight):
    sse = (weight * (y_true - y_pred) ** 2).sum(axis=0, dtype=np.float64)
    tse = (weight * (y_true - np.average(
        y_true, axis=0, weights=weight)) ** 2).sum(axis=0, dtype=np.float64)
    r2_score = 1 - (sse / tse)
    return r2_score, sse, tse

def compute_r2(y_true, y_predicted):
    sse = sum((y_true - y_predicted)**2)
    tse = (len(y_true) - 1) * np.var(y_true, ddof=1)
    r2_score = 1 - (sse / tse)
    return r2_score, sse, tse
```

```
def main():
    '''
    Demonstrate the use of compute_r2_weighted() and checks the results against sklearn
    '''
    y_true = [3, -0.5, 2, 7]
    y_pred = [2.5, 0.0, 2, 8]
    weight = [1, 5, 1, 2]
    r2_score = sklearn.metrics.r2_score(y_true, y_pred)
    print('r2_score: {0}'.format(r2_score))
    r2_score,_,_ = compute_r2(np.array(y_true), np.array(y_pred))
    print('r2_score: {0}'.format(r2_score))
    r2_score = sklearn.metrics.r2_score(y_true, y_pred,weight)
    print('r2_score weighted: {0}'.format(r2_score))
    r2_score,_,_ = compute_r2_weighted(np.array(y_true), np.array(y_pred), np.array(weight))
    print('r2_score weighted: {0}'.format(r2_score))

if __name__ == "__main__":
    main()
    #cProfile.run('main()') # if you want to do some profiling
```

The code above gives further information about implementation from scratch using Python and code below gives implementation in R from scratch.

```
#Source for code: https://stackoverflow.com/questions/40901445/function-to-calculate-r2-r-squared-i

preds <- c(1, 2, 3)
actual <- c(2, 2, 4)

rss <- sum((preds - actual) ^ 2)  ## residual sum of squares
print(rss)
tss <- sum((actual - mean(actual)) ^ 2)  ## total sum of squares
print(tss)
rsq <- 1 - rss/tss

print(rsq)



#[1] 0.25
```

Linear regression in Scikit Learn

```
#Source: "Linear Regression on Boston Housing Dataset"
# https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.datasets import load_boston
boston_dataset = load_boston()

print(boston_dataset.keys())
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston.head()
boston['MEDV'] = boston_dataset.target

correlation_matrix = boston.corr().round(2)
print(correlation_matrix)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
plt.savefig('corr.png')
#plt.clr()

plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = boston['MEDV']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = boston[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
    plt.savefig('feature.png')

  # next we prepare data
X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])
Y = boston['MEDV']
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.4, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lin_model = LinearRegression()
lin_model.fit(X_train, Y_train)

# model evaluation for training set
y_train_predict = lin_model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
r2 = r2_score(Y_train, y_train_predict)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
r2 = r2_score(Y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

CRIM: Per capita crime rate by town
ZN: Proportion of residential land zoned for lots over 25,000 sq. ft
INDUS: Proportion of non-retail business acres per town
CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX: Nitric oxide concentration (parts per 10 million)
RM: Average number of rooms per dwelling
AGE: Proportion of owner-occupied units built prior to 1940
DIS: Weighted distances to five Boston employment centers
RAD: Index of accessibility to radial highways
TAX: Full-value property tax rate per $10,000
PTRATIO: Pupil-teacher ratio by town
B: 1000(Bk — 0.63)², where Bk is the proportion of [people of African American descent] by town
LSTAT: Percentage of lower status of the population
MEDV: Median value of owner-occupied homes in $1000s

Source: "Linear Regression on Boston Housing Dataset" https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155

Linear regression using scikit-learn and Python

Compare gradient descent to identify parameters of a linear model (from scratch) with scikit-learn library in Python.

The adjoining code shows how gradient descent is used to find parameters of a linear model for the given data. Note that the machine learning library scikit-learn with a processed data set in the library is provided. The same data sets can also be downloaded separately and processed.

You can print the variables to understand what happens when you run the code.

The code example has been taken from the following sources:

University of California. (n.d). Machine-learning-databases. Retrieved from http://archive.ics.uci.edu/ml/machine-learning-databases/housing/

Scikit Learn. (n.d). Linear Regression Example. Retrieved from https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html

Frossard, D. (2016). Linear Regression with NumPy. Retrieved from https://www.cs.toronto.edu/~frossard/post/linear_regression/

Dot product. (2020). Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Dot_product

Linear regression using R

Practice using R to fit a linear model.

Another example using R with a built-in library (similar to scikit-learn) to model linear regression is given below.

The input is given by vector x and output is given by vector y and the goal of the linear model is to best fit the linear model (given by the line when you run the code).

In line 9, the built-in R linear regression model is called which takes the data given by x and y. Line 11 helps in printing the information given by the model, such as m and b coefficients.

Coefficients: (Intercept) x
-38.4551 0.6746

Later the error metrics are given in detail that expresses how well the model captures the data. Note that some of these error metrics will be covered in another lesson slide titled Correlation coefficient - R Score.

1. *Residual standard error: 3.253*
2. *R-squared: 0.9548,*
3. *Adjusted R-squared: 0.9491 F-statistic: 168.9 on 1 and 8 DF, p-value: 1.164e-06*

Further details about the output can be found here.

Finally, the graph of the predicted versus the actual is shown where x-intercept is visible.

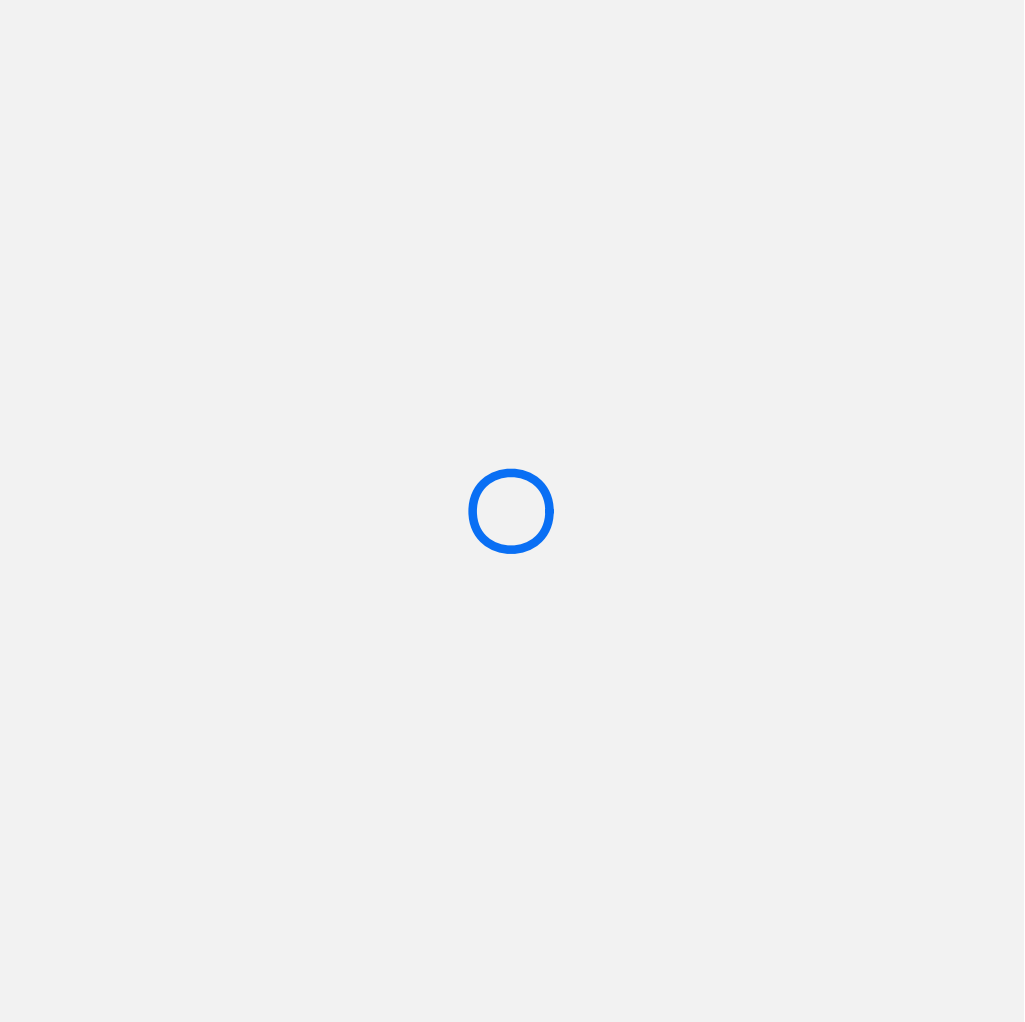


Figure: Graph of the predicted vs the actual model. "Linear Regression" by Tutorialspoint ,n.d. (https://www.tutorialspoint.com/r/r_linear_regression.htm).

```
#source https://www.tutorialspoint.com/r/r_linear_regression.htm


x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.
relation <- lm(y~x)

print(relation)

print(summary(relation))

# Find weight of a person with height 170.
a <- data.frame(x = 170)
result <-  predict(relation,a)
print(result)
```

```
# Give the chart file a name.
png(file = "linearregression.png")

# Plot the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression",
     abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

# Save the file.
dev.off()
```

Tutorials point. (n.d). R - Linear Regression. Retrieved from https://www.tutorialspoint.com/r/r_linear_regression.htm.

Exercise 1.3

Use either functions or methods in a class to do the following:

1. Use simple find and replace to convert the class labels to 1, 2, and 3 in the dataset.
2. Read the data and report mean and standard deviation for each column in the features (4 features)
3. Report the class distribution (i. e number of instances for each class)
4. Show histogram for each feature. Note you need to use a single function/method that outputs the histogram with a given filename. eg. feature1.png which is given as a parameter to the function. A for loop should be used to call the function/method
5. Split data into a train and test. Use 60 percent data in the training and test set which is assigned i. randomly ii. assigned by the first 60 percent as train and the rest as test.
6. Use previous functions to report the mean and standard deviation of the train and test set and class distribution and also the histograms for each feature.
7. Create another subset of the train and test set where only 1 feature selected by the user makes the dataset with the class. This means that you create a dataset with any 1 of the 4 features.
8. Create a subset of the dataset where you consider only instances that feature class 1 or 2, so that you treat this problem as a binary classification problem later, i.e save it as binary_iristrain.txt and binary_iristest.txt. Carry out the stats and visuals in Step 6 for this dataset.
9. Can you normalise the input features between [0 and 1]? Write code that can do so and save normalised versions.

More information about the dataset is here: https://en.wikipedia.org/wiki/Iris_flower_data_set and the original source is here: https://archive.ics.uci.edu/ml/datasets/iris

Exercise 1.4 Part I

Let's look at a multivariate linear regression problem with a dataset:
https://archive.ics.uci.edu/ml/datasets/Energy+efficiency

The dataset contains eight attributes (or features, denoted by X1...X8) and two responses (or outcomes, denoted by y1 and y2). The aim is to use the eight features to predict each of the two responses.

Specifically:
X1 Relative Compactness
X2 Surface Area
X3 Wall Area
X4 Roof Area
X5 Overall Height
X6 Orientation
X7 Glazing Area
X8 Glazing Area Distribution
y1 Heating Load
y2 Cooling Load

1. Create a correlation matrix and pick the best two features for modelling using linear regression. What do you observe about the dataset in general?
2. Develop a linear regression model for **estimating y1 (heating load)** using 60 percent of data picked randomly for training and remaining for testing. Visualise your model prediction using appropriate plots. Report the RMSE and R-squared score.
3. Try the approach with all input features, i) without normalising input data, ii) with normalising input data.
4. Run 30 experiments each and report the mean and std of the RMSE and R-squared score of the train and test datasets. Write a paragraph to compare your results of the different approaches taken.

Exercise 1.4 - Part II

Iris classification data set is a well-known data set in machine learning. More details of the data set can be found here.

> "The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other."

Iris flower data set. (2020). Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Iris_flower_data_set.

1. Try developing a linear regression model for binary classification using the step transfer function using the data produced from Iris binary classification case. Use 60 percent of data picked randomly for training and remaining for testing.
2. Try the approach with all 4 input features, i) without normalising input data, ii) with normalising input data. Report percentage correctly classified in training and test set after fixed number of training time (iterations) using gradient descent.
3. Carry out 30 experiments for both cases with different initial random seeds. Report mean and standard deviation using the appropriate metrics (accuracy, RMSE or AUC) for your results.

Step function is given below:

f(x) = { if x>=0 then x is 0, else 1}

https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

Exercise 1.5: Installation

Depending on your Python or R preference

Install Anaconda and Python libraries for machine learning on your computer

- https://docs.anaconda.com/anaconda/install/windows/
- https://docs.anaconda.com/anaconda/install/linux/
- https://anaconda.org/conda-forge/keras
- https://scikit-learn.org/stable/install.html

Install R libraries on your computer

- https://www.rstudio.com/
- https://cran.r-project.org/web/packages/caret/vignettes/caret.html
- https://keras.rstudio.com/reference/install_keras.html

After installation, try running code from the lesson examples this week.

*Note that your project will require you to use your own computer with the above-installed packages in either the Python or R platform.

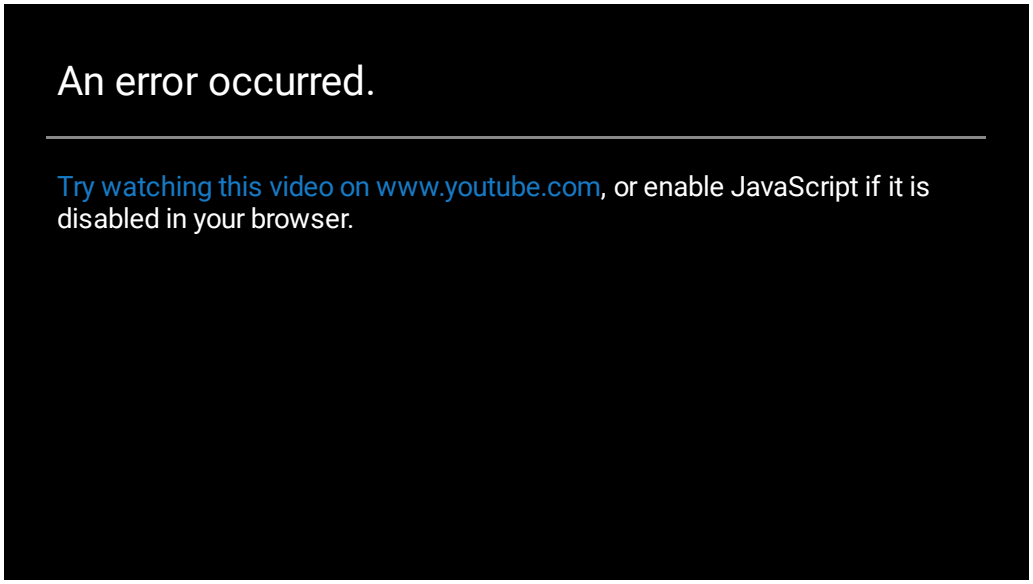
References

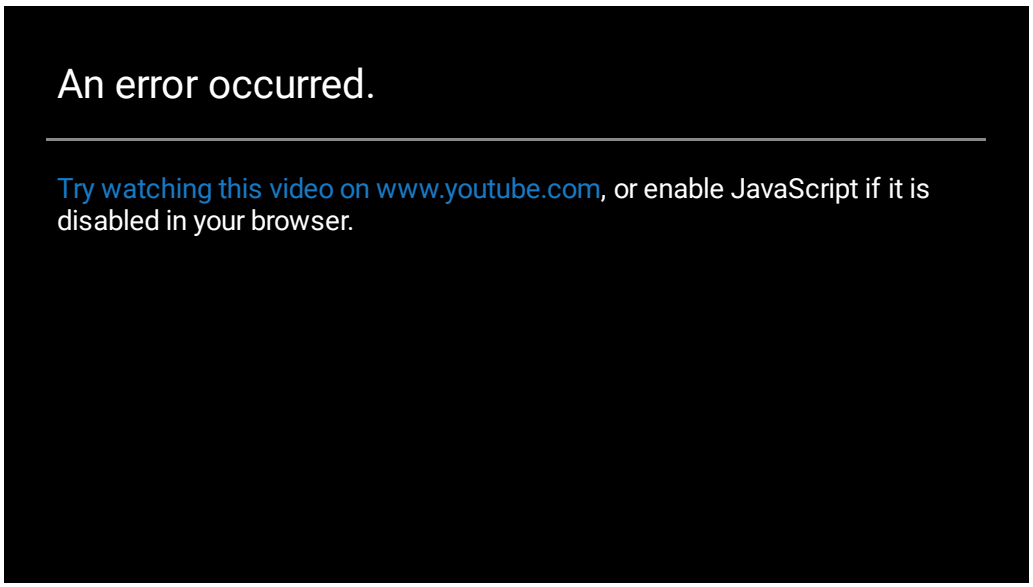
- How to Install sklearn, numpy, & scipy with Anaconda on Windows 10 64-bit: https://www.youtube.com/results?search_query=python+anaconda+scikit-learn+install
- How to download R and install Rstudio on Windows 10 2021 https://www.youtube.com/watch?v=NZxSA80lF1I
- Easily Install Anaconda Python Distribution On Mac OS X https://www.youtube.com/watch?v=V6ZAv7hBH6Y
- Install Anaconda Python, Jupyter Notebook, Spyder on Ubuntu 18.04 Linux / Ubuntu 20.04 LTS https://www.youtube.com/watch?v=DY0DB_NwEu0

Extra lesson: Github

We highly recommend the use of Github for this course

You can try the Week 1 exercise using GitHub and share your solution with others 9note this is not a requirement of the course). Please note the tutorial: https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners





I personally use Gitkraken software - its not complicated like command-line git usage: https://www.gitkraken.com/download

You can also use this: https://desktop.github.com/