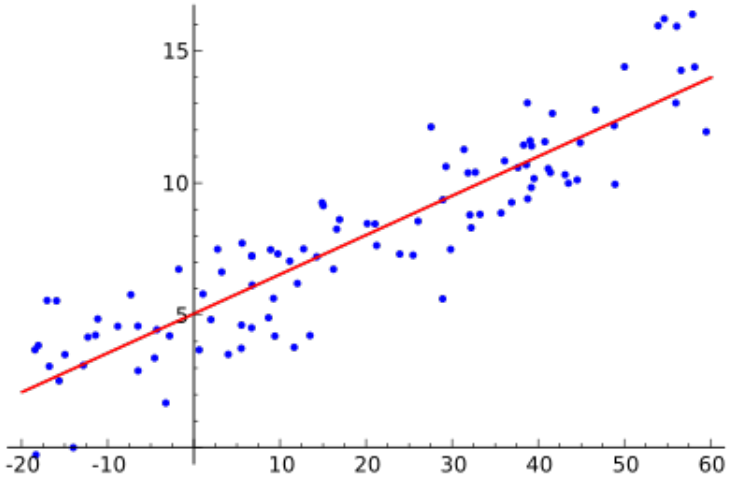
Linear Regression Worksheet[1] **NextGen AI: Lecture 1**
Sunday, April 22nd 2018

A Machine Learning Approach To Linear Regression

Today, we explored the concept of linear regression. Using your conceptual understanding of the idea, this worksheet will aim to guide you through coding linear regression from scratch in python. To remind you, the ultimate goal is to take a dataset that has a linear relationship and try to form a linear regression that can effectively make a prediction. For example, if we have a dataset that tries to correlate crime rate of town to house price, our linear regression will be able to make a prediction of how expensive a house might be given its crime rate. We can form the linear regression model by finding the optimal slope (m) and y-intercept (b) values that minimize the residuals, or error (the difference between the true and predicted values obtained from our regression).

The reason we first explore linear regression is because it does a good job of introducing the core concepts of machine learning. Although we are tackling a somewhat low-dimensional problem, what is important to understand is that the a majority of machine learning - be it image classification, or playing chess - is about giving a large amount of factors (usually independent variables) and having the machine make a prediction/decision. The process in which it produces its output is ultimately a function: just in a high dimensional space with other manufactured complexities that allow it to perform better.

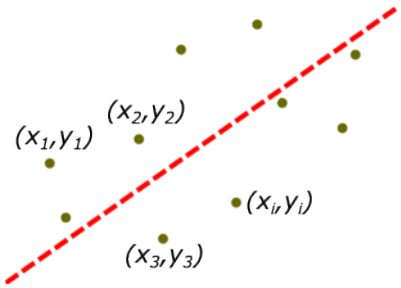
Goal:



[1] *This worksheet can be found on the syllabus of lecture 1 at nextgen-ai.org.*

Linear Regression

Linear regression aims to draw the best line it can between a collection of points. We want to find a linear (proportional) relationship between the inputs and outputs.



There are an infinite number of lines that can be drawn through a collection of points. Some are obviously better than others. What we want is a way to judge (measure) the quality of each line, then attempt to maximize the quality of the line.

There are ways to find the optimal line using one equation if you know calculus, but because that's not how machine learning tackles most problems - machine learning is more guess and check rather than precisely algorithmic - so we are going to try to discover an iterative approach that can find the optimal line.

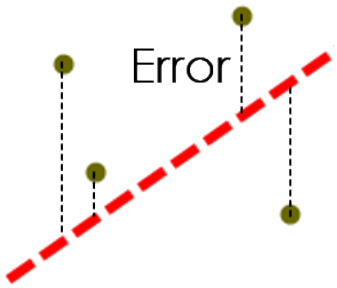
Representing Our Problem In Code

Data: First things first, we're going to need a dataset to work with. Running this code will give you a dataset about the correlation between average rooms per dwelling versus house price.

```
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd

boston = datasets.load_boston()

# If you want to read more about the dataset we are using uncomment this function
    .
# print(boston.DESCR)

bs = pd.DataFrame(boston.data)
bs.columns = boston.feature_names
bs['PRICE'] = boston.target
X = bs['RM'] #getting our X-values (what we are using to predict)
Y = bs['PRICE'] #getting our Y-values (what we are predicting)
X = X[:50]
Y = Y[:50]
plt.scatter(X,Y)
```

Listing 1: Setting our Data Up

Finding Error: A line can be defined through two values: its slope (m) and b (y-intercept). Therefore, the best-fit line we will be searching for can be described using just these two parameters. What we need to do is find the best m and b for the job, but we need to do it in regards of understanding a quality we would like to optimize.



Unless all our test points are already in a perfect line, which has a trivial solution, there will be an error between the value predicted by the line, and the observed dependent variable. These differences are called residuals.

What we want is to find a line for which the total or average of the residuals between all the points and the proposed line is at a minimum. Using just the sum of all the residuals has two problems: The first is that some of the residuals are positive (above the line), and some are negative (below the line). If we simply added the residuals together, these may cancel out and give a lower-than-true error/loss. The second issue is that there is no penalty for a large residual. We want to make sure that points further away from the line suffer more severe consequences. Similarly, points that differ by a minor amount should be treated with low regard. Notice how squaring a number less than 1 makes it even smaller.

The solution to both these problems is to square the residuals.

Writing Our Error Function:

```
def error(m, b, X, Y):
    """ Calculate the average of the residuals of a line defined
    by a slope and intercept from the X, Y values.

    Make sure to remember that each residual is the distance
    to the line vertically squared(use your math skills!)

    We will use the value this function returns as the error of
    slope and y-intercept.

    Important CS Concepts to Use:
        1. loops
        2. array handling
    If you are confused about how to do these things in python,
    refer to the python cheatsheet on our website or ask for help!"""
```

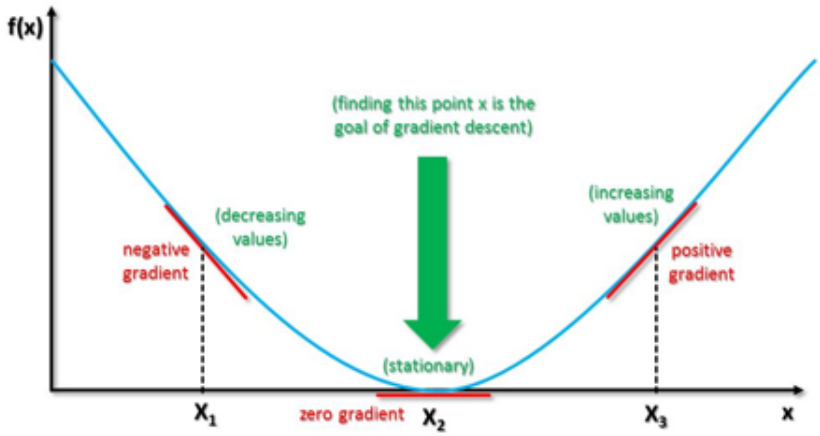
Listing 2: Error Function

Minimizing Error: The error function we coded on the last page can be written mathematically as follows:

$$\frac{1}{n} * \sum_{i=1}^{n} (truth - prediction)^2$$

Now the question becomes how do we uncover optimal values of m and b? Brute force? Brute force isn't helpful and won't scale well in to bigger problems. Therefore, we will use the concept of derivatives, or finding the slope at a certain point of a function to efficiently reach our minimum error. (Don't worry if you don't fully understand the math behind what we are explaining, we'll go in more in depth about reducing loss next week.)

Imagining the Relationship Between Slope (Gradient) and Finding the Mininum:



If we can calculate the slope at a certain value at m or b of our error function, we can determine whether we need to increase or decrease m or b to lower our error. From observing the images above we can observe two things:

- If our slope is negative, we want to increase our m and b values respectively to lower the error.
- If our slope is positive, we want to decrease our m and b values respectively to lower the error.

From this, we can code a function that shifts our m and b values respectively based on the slope respectively at different points (m and b have their own slopes). Hopefully, after many iterations our program will be able to get close to the mininum of the error function.

Finding the slopes for m and b in the error function:

To get these equations, we took the derivative of the error function. This gives us an equation that computes the tangent line at any given point on our error graph in regards of m or b. As you might recall from the slides, the slope of this tangent line tells us which direction minimizes the error function. Moving in the direction of the opposite sign of the slope, reduces error. These equations will tell you the slope of the error function for m and b respectively.

$$m_{slope} = \frac{2}{n}\sum_{1}^{n} -x_i(y_i - (mx_i + b))$$

$$b_{slope} = \frac{2}{n}\sum_{1}^{n} -(y_i - (mx_i + b))$$

```
def mslope(m_current, b_current, X, Y):
    """ use the formula to write this function """

def bslope(m_current, b_current, X, Y):
    """ use the formula to write this function """


def calculateNewM(m_slope, m_current, learning_rate):

    """ using the important patterns regarding slope
    and how we want it to affect our m value (pg 4),
    write a function that returns a new m_current value.

    The learning rate determines how much you want to shift
    our m_value in proportion to m_slope.
    (For example, if our m_slope was one and our learning_rate
    was 0.5, we would shift m_current by .5) """


def calculateNewB(b_slope, b_current, learning_rate):

    """ using the important patterns regarding slope
    and how we want it to affect our b- value (pg 4),
    write a function that returns a new m_current value.

    The learning rate determines how much you want to shift
    our b_value in proportion to b_slope.
    (For example, if our b_slope was one and our learning_rate
    was 0.5, we would shift b_current by .5) """

```

Listing 3: Slope Functions

Final Steps: Linear Regression!

Write a linear regression function that combines all the helper functions we made and accomplishes the following:

- Inputs: X, Y, m-current=0, b-current = 0, epochs = 100000 (amount of updates we want to do), learning-rate = 0.01.
- Write a loop (for the amount of epochs) that computes and prints the error every 1000 epochs, and computes the slopes of m and b in regards to error to update them for the next iteration of the loop (at the same time).
- After the loop is finished, return the final m, b, and error values (only the final ones).

```
def linear_regression(X, Y, m_current, b_current, epochs, learning_rate):
    """ Follow the directions above. """


    return m_current, b_current

slope, intercept = linear_regression(X, Y, 0, 0, 100000, 0.01)

print(slope,intercept)


#Graphing Our Regression
import numpy as np

f = lambda x: slope*x + intercept
x = np.array([0,10])

plt.plot(x,f(x),lw=2.5, c="k",label="fit line between 0 and 100")
plt.scatter(X,Y)
plt.ylim(4,9)
plt.ylim(0,60)

```

Listing 4: Slope Functions

Feel free to ask us any questions if you are confused. This is definitely a challenging worksheet. Answers will be available online soon.