2020

1. Linear Regression Analysis and Predictions using Artificial Neural Network

Name : Raja Muppalla

Professor : Dr. Kim, Minkyu

2. Introduction :

The project goal is to predict the accuracy of models with the help of Linear Regression and Artificial Neural Network (ANN) by training the models with known data.

3. Linear Regression :

3.1 Theory :

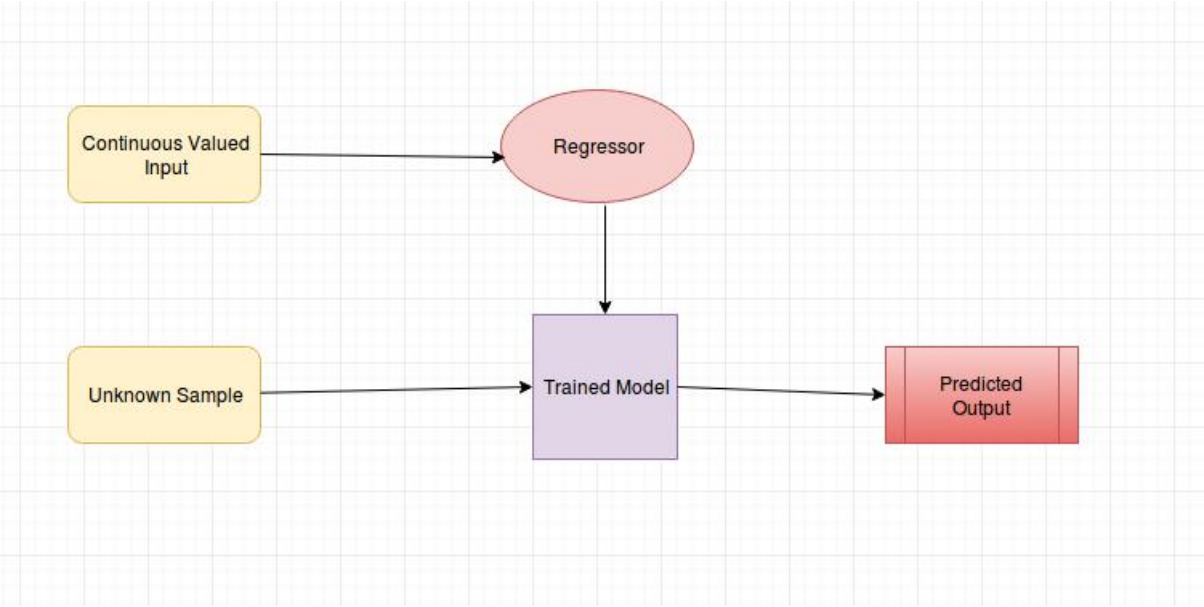
A simple and widely used form of predictive analysis is linear regression. To describe the relationship between one dependent variable and one or more independent variables, these regression estimates are used. Regression implementations are various and exist in almost every field, including engineering, physical and chemical sciences, economics, management, life and biological sciences, and social sciences. These regression estimates are used to clarify the relationship. In fact, regression analysis may be the most widely used statistical technique.

An essential aspect of regression analysis is data collection. Any regression analysis is only as good as the data on which it is based. Three basic methods for collecting data are as follows:

(i) A retrospective study based on historical data
(ii) An observational study
(iii) A designed experiment

Regression Models are used for several purposes, including the following:

i. Data Description
ii. Parameter Estimation
iii. Prediction and estimation
iv. Control



3.2 Implementation and Analysis Approach :

A mathematical technique for finding the association between two or more continuous quantitative variables is linear regression. For example, a real estate agent knows that the house's square footage is linked to the property's price. Machine learning adopted this concept and used it to predict an unknown quantity from known quantities of another variable called a dependent variable). That means that we can estimate the cost of it if we know the square footage of a building.

Calculating the best fit is by selecting two random points, the estimation will not be accurate at all. We would like a line that passes through(almost) all the points. In machine learning to achieve this, we present a function called a cost function. Our goal is to minimize the result of the cost function until it meets a minimization threshold. The line with the minimum cost is called the linear best fit.

Simple linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable. It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other. The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line. The general steps for predicting the linear regression as follows:

i. Importing Libraries
ii. Loading the Dataset
iii. Split to independent and dependent variables
iv. Splitting data into training and testing data
v. Choosing the model
vi. Fit our model
vii. Predict the output
viii. Plot the graph

3.3 Check Data :

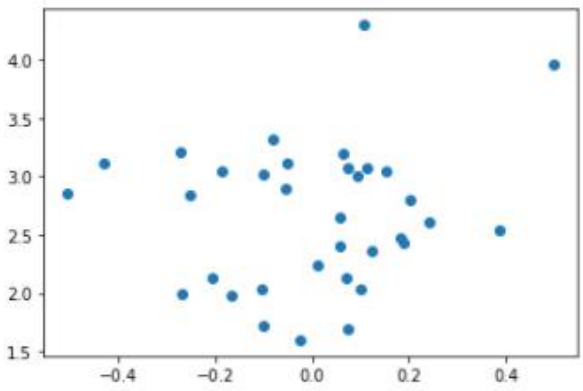
LINEAR REGRESSION ANALYSIS

Importing The Data

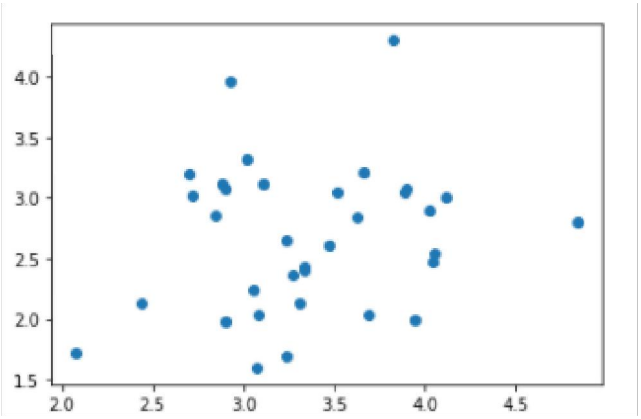
```
data = pd.read_csv('linear_regression.csv')
data.head()
```

	x1	x2	x3	x4	x5	y1	y2	y3
0	-0.099362	-0.085577	-0.143910	-0.104020	0.29407	2.72	3.02	31.52
1	-0.166140	0.277080	-0.190330	-0.056387	-0.13422	2.90	1.98	21.73
2	-0.252780	-0.085391	-0.096496	0.422420	-0.27044	3.63	2.84	29.20
3	0.072448	0.554370	-0.046507	0.325990	-0.33308	3.24	1.69	20.32
4	-0.266970	-0.108820	0.038903	0.215480	-0.44817	3.94	1.99	22.73

Plotting the Input and Output



Input (X)



Output (Y)

To find the bias of the model (or method), perform several assumptions and add errors to each estimate relative to the actual value. Dividing by the number of estimates gives the method bias. In statistics, there can be many statistics to find the same value. The difference between the mean and the actual value of these estimates is biased.

Therefore, bias is a measure of how close the mapping function model is between inputs and .puts. Having a high bias for the model means that it is much simpler and it can be improved by adding more features. The Bias does not rely on data.

The test-MSE equation is calculated using training data appropriate to the model, so it should more accurately be called training MSE. But in general, we do not really care how well this method of training works on training data. Instead, we are interested in the accuracy of the predictions we make when applying our method to previously unseen test data.

3.4 Analysis Results :

Mean and Variance calculation

```
# Calculate the mean value of a list of numbers
def mean(values):
    return sum(values) / float(len(values))

# Calculate the variance of a list of numbers
def variance(values, mean):
    return sum([(x-mean)**2 for x in values])

# calculate mean and variance
dataset = data.values
x = [row[0] for row in dataset]
y = [row[1] for row in dataset]
mean_x, mean_y = mean(x), mean(y)
var_x, var_y = variance(x, mean_x), variance(y, mean_y)
print('x stats: mean=%.3f variance=%.3f' % (mean_x, var_x))
print('y stats: mean=%.3f variance=%.3f' % (mean_y, var_y))
```

```
x stats: mean=-0.000 variance=1.467
y stats: mean=-0.000 variance=2.030
```

Covariance calculation

```
# calculate covariance
dataset = data.values
x = [row[0] for row in dataset]
y = [row[1] for row in dataset]
mean_x, mean_y = mean(x), mean(y)
covar = covariance(x, mean_x, y, mean_y)
print('Covariance: %.3f' % (covar))
```

```
Covariance: 0.227
```

The goal of any supervised machine learning model is to optimize the mapping function (f) for the output input / dependent variable (y) given by the input / independent variable (X). The mapping function is called the target function because it is the object of the monitored machine learning algorithm.

The expected error of any machine learning algorithm can be divided into three parts:

i. Bias error
ii. Variance error
iii. Irreducible error

Cost function computation

```
#Setting the hyper paramenters
# notice small alpha value
alpha = 0.1
iters = 1000

# theta is a row vector
theta = np.array([[1.0, 1.0]])

#Creating the cost function
def computeCost(X, y, theta):
    inner = np.power(((X @ theta.T) - y), 2) #
    return np.sum(inner) / (2 * len(X))

computeCost(X, y, theta)
```

```
1.5888764729276617
```

Prediction using Gradient Descent

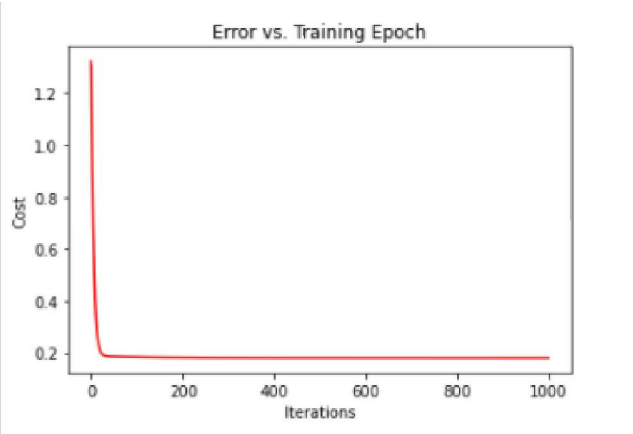
```
#Create the Gradient Descent function
def gradientDescent(X, y, theta, alpha, iters):
    for i in range(iters):
        theta = theta - (alpha/len(X)) * np.sum((X @ theta.T - y) * X, axis=0)
        cost = computeCost(X, y, theta)
        # if i % 10 == 0: # just look at cost every ten loops for debugging
        #     print(cost)
    return (theta, cost)

g, cost = gradientDescent(X, y, theta, alpha, iters)
print(g, cost)
```

```
[[2.67382388 0.4091104 ]] 0.18029560863335958
```

Cost= 0.18029

g= [2.67382388 0.4091104]



Variance and Bias trade-off

The ideal model is one with less deviation and less bias.

The model with less bias and more variation is the more fitted model. In general, over-fitting is worse than learning a general concept behind the data, remembering the training set, or training.

The model with the highest bias and the least difference is the most suitable model.

A model with high bias and high variance is the worst-case scenario because it is the model that produces the largest predictive error.

RMSE calculation

```
# Simple linear regression algorithm
def simple_linear_regression(train, test):
    predictions = list()
    b0, b1 = coefficients(train)
    for row in test:
        yhat = b0 + b1 * row[0]
        predictions.append(yhat)
    return predictions
```

```
[-0.015351663067568867, -0.025668762046749
361265, -0.04164853903043692, 0.00901877144
1585426217075, 0.008942449199664094, 0.014
015559155270246854, -0.015961623035968726,
1, -0.07770848245103645, 0.0164181667758966
83, -0.008029053477325784, 0.01121172275991
RMSE: 27.685
```

Total estimation error = bias + Variance.

4. Artificial Neural Network :

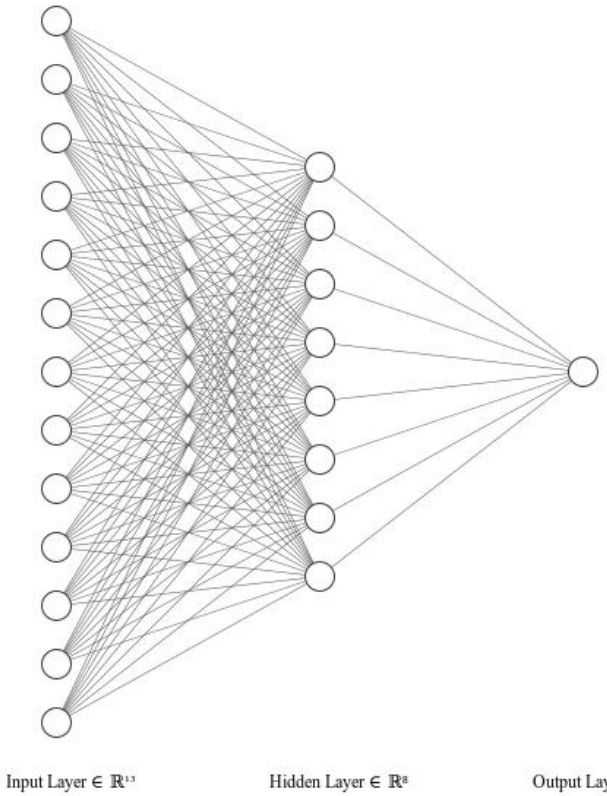
4.1 Theory :

An artificial neural network (ANN) is part of a computing system designed to mimic the way the human brain analyses and processes information. It is the basis of Artificial Intelligence (AI) and solves problems that prove impossible or difficult based on human or statistics. ANNs have self-learning capabilities that can achieve better results when more data is available.

Key Takeaways

An artificial neural network (ANN) is a component of artificial intelligence intended to mimic the functioning of the human brain.

- Processing units produce ANNs that contain inputs and pits. Teaching ANN to build the output that requires inputs.
- Back propaganda is a set of rules of practice used to guide artificial neural networks.
- Remote practical applications for ANNs including finance, personal communication, industry and education.
- The neural network combines multiple neurons to form large and complex mathematical operations.



2 layer neural network

The neural net above consists of a hidden layer and a final output layer. The input layer has 2 nodes except the target. The hidden layer can accept any number of nodes and the last layer to make predictions is 1 node.

4.2 Implementation :

Artificial neural networks are one of the most important tools used in machine learning. As the "neural" part of their name suggests, they are brain-stimulated systems that are meant to reflect the way we humans learn. Neural networks have input and output layers (in most cases) a hidden layer that contains units that make the input layer useful for the output layer. They are excellent tools for capturing and teaching a machine to identify and identify the most complex or multiple patterns for the human programmer.

1. Building Feed Forward ANN
2. Building Feed Back ANN
3. Defining Activation Function
4. Calculation logistic loss
5. Calculation of Cost function
6. Calculation of Confusion Matrices
7. Calculation of Accuracy score

4.2 Analysis approach :

Implementation :

In this tutorial, we are going to build a simple neural network that supports multiple layers and validation. The main function is NeuralNetwork, which will train the network for the specified number of epochs. At first, the weights of the network will get randomly initialized by InitializeWeights. Then, in each epoch, the weights will be updated by Train and finally, every 20 epochs accuracy both for the training and validation sets will be printed by the Accuracy function. As input the function receives the following:

- X_train, Y_train: The training data and target values.
- X_val, Y_val: The validation data and target values. These are optional parameters.
- epochs: Number of epochs. Defaults at 10.
- nodes: A list of integers. Each integer denotes the number of nodes in each layer. The length of this list denotes the number of layers. That is, each integer in this list corresponds to the number of nodes in each layer.
- lr: The learning rate of the back-propagation training algorithm. Defaults at 0.1.

4.3 Check Data :

Importing The Data

```
df = pd.read_csv('ANN.csv')
df
```

	0	1	y
0	0.694565	0.426664	0
1	1.683530	-0.800166	0
2	-0.250468	0.243922	1
3	-1.133380	-0.611279	1
4	1.769056	-0.310254	0
...	...	...	...
196	1.419756	0.067688	0
197	0.344427	0.025001	1
198	0.349166	1.063136	1
199	1.454850	-0.060964	0
200	NaN	NaN	1

201 rows × 3 columns

Data Splitting into input and output features

```
#Splitting the dataset into input features(X) and predict features (Y)
dataset = df.values
dataset
```

```
array([[ 0.694565  ,  0.42666408,  0.        ],
       [ 1.68353008, -0.80016643,  0.        ],
       [-0.25046823,  0.24392224,  1.        ],
       [-1.13337973, -0.6112787 ,  1.        ],
       [ 1.76905577, -0.31025439,  0.        ],
       [ 2.00225511, -0.18592   ,  1.        ],
       [ 0.91169861,  0.46995543,  1.        ],
       [ 0.88211794, -0.46701178,  0.        ],
       [ 0.75006972,  0.33995342,  1.        ],
       [ 1.30208867, -0.72334923,  0.        ],
       [-0.39837104,  0.29667178,  1.        ],
       [-0.01048054, -0.36792973,  0.        ],
       [ 1.78113658, -0.45000393,  1.        ],
       [ 1.58676882,  0.00796155,  1.        ],
       [-0.8236347 ,  1.01881137,  1.        ],
       [ 1.00678606,  0.14949759,  0.        ],
       [ 0.74583051,  0.98688233,  0.        ],
       [-0.01152947, -0.2332013 ,  0.        ],
       [ 0.17916419,  0.7178944 ,  1.        ],
       [-1.15514408,  1.37176148,  0.        ],
```

Weight and bias

Weight and bias are learnable parameters that help the neural network to properly study a function. Think of weight as a measure of how accurately you think a trait contributes to an estimate and a bias to the base value starting from your estimates.

The machine learning model uses many examples to understand the correct weight and bias, helping to accurately estimate each aspect of the machine product in the dataset.

```python
#Neural Network
class NeuralNet():
    '''
    A two layer neural network
    '''

    def __init__(self, layers=[13,8,1], learning_rate=0.001, iterations=100):
        self.params = {}
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.loss = []
        self.sample_size = None
        self.layers = layers
        self.X = None
        self.y = None

    def init_weights(self):
        '''
        Initialize the weights from a random normal distribution
        '''
        np.random.seed(1) # Seed the random number generator
        self.params["W1"] = np.random.randn(self.layers[0], self.layers[1])
        self.params['b1']  =np.random.randn(self.layers[1],)
        self.params['W2'] = np.random.randn(self.layers[1],self.layers[2])
        self.params['b2'] = np.random.randn(self.layers[2],)
```

Activation Function

Activation is the activation of a neural network that can learn complex non-linear functions. Non-linear functions are difficult to learn in traditional machine learning algorithms such as logistics and linear regression. The activation function allows the neural network to understand these functions.

A variety of activation functions are used for in-depth study - sigmoid, reel, tan and leaky reel are popular.

```python
#Activation Function
def relu(self,Z):
        '''
        The ReLu activation function is to performs a threshold
        operation to each input element where values less
        than zero are set to zero.
        '''
        return np.maximum(0,Z)
```

ReLU (Rectified Linear Unit) is a simple operation that compares the value to zero. That is, if it is greater than zero, it returns the transferred value; Otherwise, it returns zero.

In short, the hidden layer derives values from the input layer, calculates the weighted amount, adds the bias word, and then sends each result through an activation function called a reel in our case.

```
#Activation Function
def relu(self,Z):
        '''
        The ReLu activation function is to performs a threshold
        operation to each input element where values less
        than zero are set to zero.
        '''
        return np.maximum(0,Z)
```

```
relu(X,Y)
```

```
array([0., 0., 1., 1., 0., 1., 1., 0., 1., 0., 1., 0., 1., 1., 1., 0., 0.,
       0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 1.,
       1., 0., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 0., 0., 0., 1., 1.,
       0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0., 1., 0., 1., 0., 0.,
       1., 0., 0., 1., 0., 1., 1., 1., 0., 1., 0., 0., 1., 1., 0., 1., 1.,
       1., 0., 0., 0., 1., 1., 0., 0., 1., 0., 1., 1., 1., 1., 0., 1., 1.,
       1., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1.,
       1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 1., 1., 0., 0., 0., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 1., 1., 0., 1., 1., 1.,
       0., 0., 1., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 1., 1.,
       1., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 1., 1., 0.,
       0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 1.])
```

Lose Function

The lose function must be designed correctly so that the wrong model can be properly punished and the right model will be rewarded. If a prophecy is too close or too close to the original prophecy, you want to make a loss. The choice of damage function depends on the work of the class and classification problems, you can use cross entropy loss.

```
#Loss Function
def entropy_loss(self,y, yhat):
        nsample = len(y)
        loss = -1/nsample * (np.sum(np.multiply(np.log(yhat), y) + np.multiply((1 - y), np.log(1 - yhat))))
        return loss
```

Forward Propagation

Forward propaganda is the name given to a series of calculations performed by a neural network prior to an assessment. In the code, first, all dot products and additions are handled using the weight and bias you previously started, call the entropy_loss function to calculate the loss, save the estimated parameters and return the values and losses beyond the values. These values are used during back propagation.

```
#Forward Propagation

def forward_propagation(self):
        '''
        Performs the forward propagation
        '''

        Z1 = self.X.dot(self.params['W1']) + self.params['b1']
        A1 = self.relu(Z1)
        Z2 = A1.dot(self.params['W2']) + self.params['b2']
        yhat = self.sigmoid(Z2)
        loss = self.entropy_loss(self.y,yhat)

        # save calculated parameters
        self.params['Z1'] = Z1
        self.params['Z2'] = Z2
        self.params['A1'] = A1

        return yhat,loss
```

Back propagation

Back propagation is the name given to the process of updating and training the weight and bias of the neural network.

A neural network wants to constantly test different values for weights and compare risks to estimate the correct values. If the loss function is reduced, the current load will be better than the previous one or vice versa. To get the best weight and bias the neural net has to go through several training (forward campaign) and update (back propagation) cycles. This cycle is what we commonly call the training phase and the process of looking for the right weight is called optimization.

```
#Backward Propagation
def back_propagation(self,yhat):
        '''
        Computes the derivatives and update weights and bias according.
        '''
        def dRelu(x):
            x[x<=0] = 0
            x[x>0] = 1
            return x

        dl_wrt_yhat = -(np.divide(self.y,yhat) - np.divide((1 - self.y),(1-yhat)))
        dl_wrt_sig = yhat * (1-yhat)
        dl_wrt_z2 = dl_wrt_yhat * dl_wrt_sig

        dl_wrt_A1 = dl_wrt_z2.dot(self.params['W2'].T)
        dl_wrt_w2 = self.params['A1'].T.dot(dl_wrt_z2)
        dl_wrt_b2 = np.sum(dl_wrt_z2, axis=0)

        dl_wrt_z1 = dl_wrt_A1 * dRelu(self.params['Z1'])
        dl_wrt_w1 = self.X.T.dot(dl_wrt_z1)
        dl_wrt_b1 = np.sum(dl_wrt_z1, axis=0)
```

4.5 Analysis Results :

```
sknet.fit(Xtrain, ytrain)
preds_train = sknet.predict(Xtrain)
preds_test = sknet.predict(Xtest)

print("Train accuracy of sklearn neural network: {}".format(round(accuracy_score(preds_train, ytrain),2)*100))
print("Test accuracy of sklearn neural network: {}".format(round(accuracy_score(preds_test, ytest),2)*100))
```

```
Train accuracy of sklearn neural network: 68.0
Test accuracy of sklearn neural network: 66.0
```

5. Conclusions :

We see that the Linear regression model performed well on the given dataset based on the results obtained from both models than the ANN model since the ANN model needs larger data sets to train from. If the dataset is larger, ANN could have done well for this type of problem and the training model performance is simple for linear regression compared to ANN model performance.