Department of Computer Science and Engineering

Course Code:CSE422 **Credits: 1.5**

Course Name: Artificial Intelligence **Prerequisite: CSE111, CSE221**

Lab 06

Logistic Regression

I. Lab Overview:

The students will learn the regression as well as Logistic regression analysis for the machine learning approach.

II. Learning Objective:

I. What is Logistic regression analysis?

II. How to solve problem using Logistic regression.

III. Lesson Fit:

There is pre-requisite to this lab: CSE111, CSE221. You should have intensive Programming Knowledge and capability to understand algorithms.

IV. Acceptance and Evaluation

Students will show the output using different datasets and python code. They will be marked according to their lab performance. The main evaluation criteria will be based on project report and demonstration.

V. Learning Outcome:

After this lab, the students will be able to:

I. **Understand the Logistic regression analysis.**

II. **How to apply Logistic regression for prediction.**

VI. Activity Detail

Hour: 1.0 - 2.0

Logistic Regression

Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable(or output), y, can take only discrete values for given set of features(or inputs), X.
We can also say that the target variable is **categorical**. Based on the number of categories, Logistic regression can be classified as:

1. **binomial:** target variable can have only 2 possible types: “0” or “1” which may represent “win” vs “loss”, “pass” vs “fail”, “dead” vs “alive”, etc.
2. **multinomial:** target variable can have 3 or more possible types which are not ordered(i.e. types have no quantitative significance) like “disease A” vs “disease B” vs “disease C”.
3. **ordinal:** it deals with target variables with ordered categories. For example, a test score can be categorized as:“very poor”, “poor”, “good”, “very good”. Here, each category can be given a score like 0, 1, 2, 3.

First of all, we explore the simplest form of Logistic Regression, i.e **Binomial Logistic Regression**.

Linear Regression Equation:

$$y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots\ldots + \beta nXn$$

Where, y is dependent variable and x1, x2 ... and Xn are explanatory variables.

Sigmoid Function:

$$p = 1/1 + e^{-y}$$

Apply Sigmoid function on linear regression:

$$p = 1/1 + e^{-(\beta 0+\beta 1X1+\beta 2X2.....\beta nXn)}$$

Properties of Logistic Regression:

- The dependent variable in logistic regression follows Bernoulli Distribution.
- Estimation is done through maximum likelihood.

Let regression coefficient matrix/vector:

$$x_i = \begin{bmatrix} 1 \\ x_{i1} \\ x_{i2} \\ . \\ . \\ x_{ip} \end{bmatrix} \qquad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ . \\ . \\ \beta_p \end{bmatrix}$$

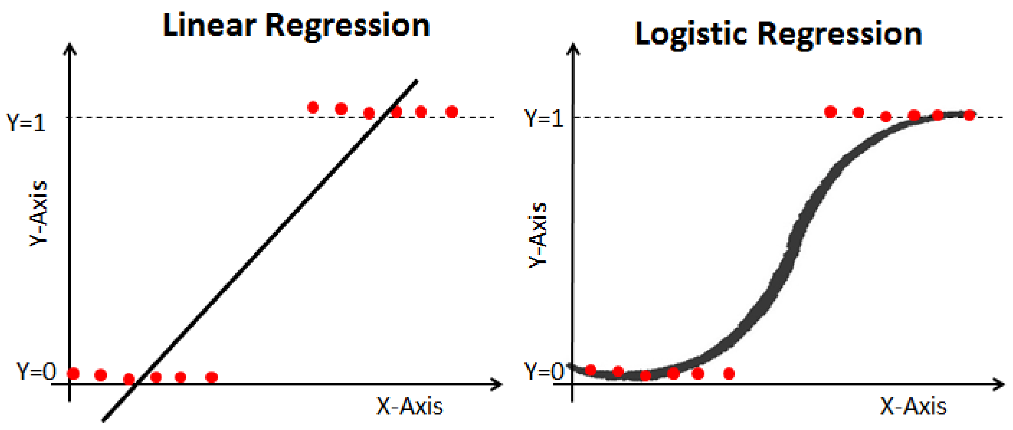
Then, in a more compact form,

$$h(x_i) = g(\beta^T x_i) = \frac{1}{1 + e^{-\beta^T x_i}}$$

Details Application: Find how logistic regression has been used to classify sentiment and embedded in a web service
https://www.youtube.com/watch?v=H6ii7NFdDeg
https://github.com/llSourcell/logistic_regression

Linear Regression Vs. Logistic Regression

Linear regression gives you a continuous output, but logistic regression provides a constant output. An example of the continuous output is house price and stock price. Example's of the discrete output is predicting whether a patient has cancer or not. Linear regression is estimated using Ordinary Least Squares (OLS) while logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach.



Sigmoid Function

The sigmoid function, also called logistic function gives an ‘S’ shaped curve that can take any real-valued number and map it into a value between 0 and 1. If the curve goes to positive infinity, y predicted will become 1, and if the curve goes to negative infinity, y predicted will become 0. If the output of the sigmoid function is more than 0.5, we can classify the outcome as 1 or YES, and if it is less than 0.5, we can classify it as 0 or NO. For example: If the output is 0.75, There is a 75 percent chance that patient will suffer from cancer.

$$f(x) = \frac{1}{1+e^{-(x)}}$$

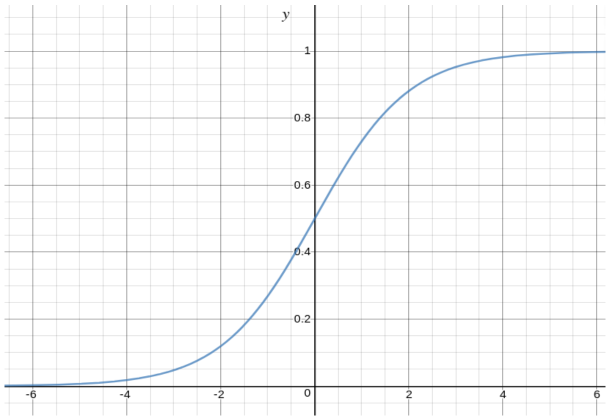
The activation function that is used is known as the **sigmoid** function.

```
# logistic_func from sigmoid equation given in this tutorial
```

def logistic_func(beta, X):
    return 1.0/(1 + call numpy exponential function (- call numpy Dot product of two arrays function (X, Transpose beta)))

# numpy exponential https://docs.scipy.org/doc/numpy/reference/generated/numpy.exp.html
# numpy Dot https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html
# Transpose https://docs.scipy.org/doc/numpy-1.14.2/reference/generated/numpy.ndarray.T.html

The plot of the sigmoid function looks like



We can see that the value of the sigmoid function always lies between 0 and 1. The value is exactly 0.5 at X=0. We can use 0.5 as the probability threshold to determine the classes. If the probability is greater than 0.5, we classify it as **Class-1(Y=1)** or else as **Class-0(Y=0)**.

```
#For prediction

def pred_values(beta, X):
    '''
    function to predict labels
    '''
    pred_prob = logistic_func(beta, X)
    pred_value = np.where(pred_prob >= .5, 1, 0)
    return np.squeeze(pred_value)
```

Maximum Likelihood Estimation Vs. Least Square Method

The MLE is a "likelihood" maximization method, while OLS is a distance-minimizing approximation method. Maximizing the likelihood function determines the parameters that are most likely to produce the observed data.

Ordinary Least squares estimates are computed by fitting a regression line on given data points that has the minimum sum of the squared deviations (least square error). MLE assumes a joint probability mass function.

We can define conditional probabilities for 2 labels(0 and 1) for observation as:

$$P(y_i = 1|x_i; \beta) = h(x_i)$$
$$P(y_i = 0|x_i; \beta) = 1 - h(x_i)$$

We can write it more compactly as:

$$P(y_i|x_i; \beta) = (h(x_i))^{y_i}(1 - h(x_i))^{1-y_i}$$

Now, we define another term, **likelihood of parameters** as:

Likelihood is nothing but the probability of data (training examples). It measures the support provided by the data for each possible value. We obtain it by multiplying all and for easier calculations, we take **log likelihood**:

$$l(\beta) = \sum_{i=1}^{n} y_i log(h(x_i)) + (1 - y_i) log(1 - h(x_i))$$

Cost function intuition

We can take inverse of Likelihood function equations like below:

$$cost(h(x), y) = -ylog(h(x)) - (1 - y)log(1 - h(x))$$

The cost for all the training examples computed by taking the average of all the training samples

$$J(\beta) = \sum_{i=1}^{n} -y_i log(h(x_i)) - (1 - y_i) log(1 - h(x_i))$$

```
def cost_func(beta, X, y):
    '''
    cost function, J
    '''
    log_func_v = logistic_func(beta, X)
    y = np.squeeze(y)
    step1 = y * np.log(log_func_v)
    step2 = (1 - y) * np.log(1 - log_func_v)
    final = -step1 - step2
    return np.mean(final)
```

We will use gradient descent to minimize the cost function. The gradient can be given by

$$\frac{\partial J(\beta)}{\partial \beta_j} = (h(x) - y)x_j$$

```
def log_gradient(beta, X, y):
    '''
    logistic gradient function
    '''
    first_calc = logistic_func(beta, X) - y.reshape(X.shape[0], -1)
    final_calc = np.dot(first_calc.T, X)
    return final_calc
```

Then we update the weights by subtracting to them the derivative times the learning rate. We should repeat this steps several times until we reach the optimal solution.

```
beta -= learning_rate * log_gradient
```

```
def grad_desc(X, y, beta, lr=.01, converge_change=.001):
    '''
    gradient descent function
    '''
    cost = cost_func(beta, X, y)
    change_cost = 1
    num_iter = 1

    while(change_cost > converge_change):
        old_cost = cost
        beta = beta - (lr * log_gradient(beta, X, y))
        cost = cost_func(beta, X, y)
        change_cost = old_cost - cost
        num_iter += 1

    return beta, num_iter
```

```
def plot_reg(X, y, beta):
    '''
    function to plot decision boundary
    '''
    # labelled observations
    x_0 = X[np.where(y == 0.0)]
    x_1 = X[np.where(y == 1.0)]

    # plotting points with diff color for diff label
    plt.scatter([x_0[:, 1]], [x_0[:, 2]], c='b', label='y = 0')
    plt.scatter([x_1[:, 1]], [x_1[:, 2]], c='r', label='y = 1')

    # plotting decision boundary
    x1 = np.arange(0, 1, 0.1)
    x2 = -(beta[0,0] + beta[0,1]*x1)/beta[0,2]
    plt.plot(x1, x2, c='k', label='reg line')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.show()
```

Task: 01

1. Read "Logistic_regression_dataset.csv" dataset using pandas
   Convert pandas dataframe as array
   https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html

2. Define first and second column together as 'X' and second column as 'Y' so that we can build a relationship something like X{column1, column2} → Y
   Hints: X = dataframe[dataset length, start_col : end_col]
   Y = dataframe[dataset length, end_col]

3. Normalize the data of X within 0 → 1 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#

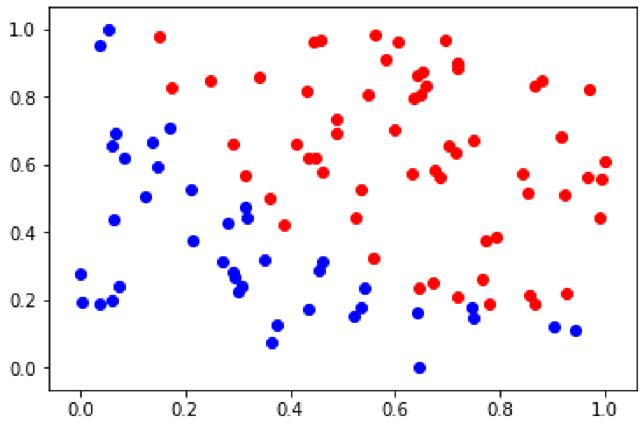
4. Add a new column within X filled with ones
   Hints: ones = numpy.ones(dataset length)
   ones = reshape it to something like below of size (dataset length,1)

```
ones=array([[1.],
            [1.],
            [1.],
            [1.],
            [1.]])
```

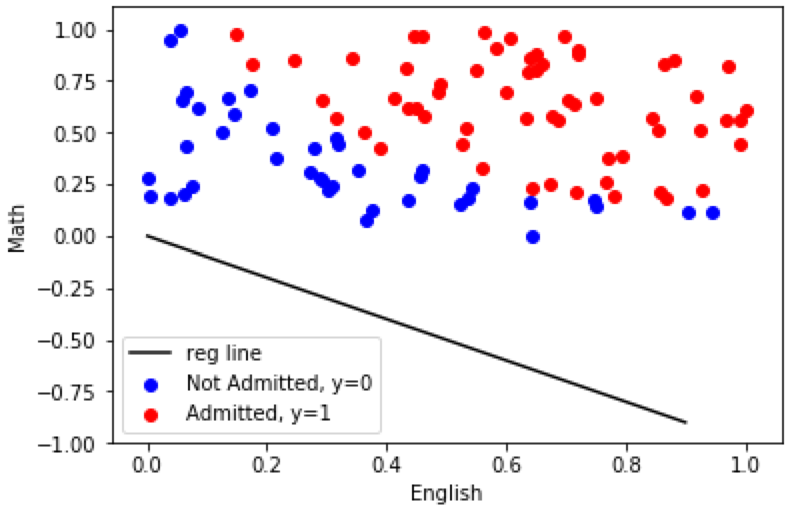
Hints: to check dimention of X use X.shape
And for reshape use command
variable.reshape(dimention size)

Then, X = merge two array ones and X columnwise
https://docs.scipy.org/doc/numpy/reference/generated/numpy.c_.html
https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.concatenate.html

5. Create scatter plot for only X and Y
See `plot_reg()` function given above to get an idea how to do something like below,
Hints: remove the code from plot_reg() function where beta is used



6. Call `plot_reg(X, y, beta)` function again for any given beta . Create your beta matrix with three elements (beta0, beta1, beta2) since we have 2 data columns and one intercept column. You can assume beta0, beta1, beta2 are [1, 10, 10]. Define the beta as matrix https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.matrix.html



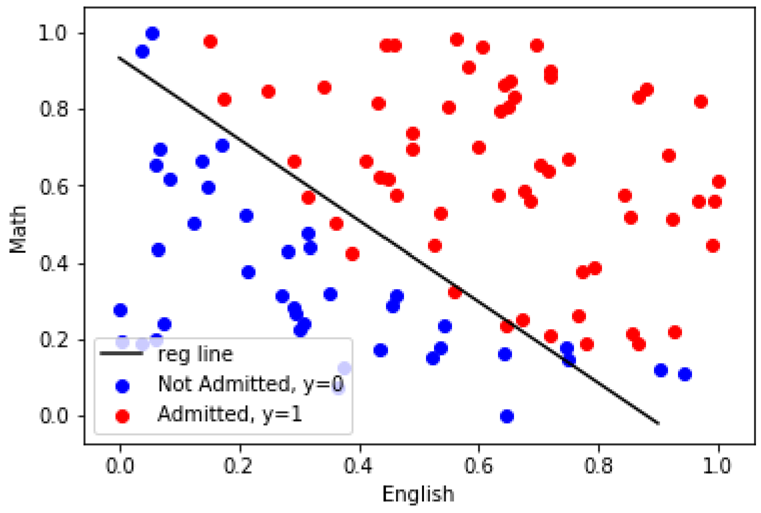
7. We have to minimize the error using gradient descent function call given above

```
beta, num_iter = grad_desc(X, y, beta, learning_rate=.01, converge_change=.001)
```

it will return most accurate `beta, num_iter,` beta can be used to get perfect separator and we will also know how many iterations are needed to converge to actual desired result

print your new `beta and num_iter`

8. call plot_reg(X, y, newbeta) to see something like below



9. Get some prediction using the function `pred_values(beta, Given_X)` It will return you whether the student should get admission or not, 1 or 0. `Given_X` Will contain three parameter 1 or 0 followed by English mark and Math marks. If you print your X column after 4th Step, you will get an idea how to provide value for `Given_X parameter.`

Task: 02

Apply logistic regression to classify customers, who could buy a product based on his Age, Gender and Salary. Use "Social_Network_Ads" data set and python programming to model the classification problem. Finally, draw the ROC (receiver operating characteristic) curve from the actual *y* and predicted *y* to evaluate the performance of your logistic regression model.

Evaluation Process (VIVA): You have to explain your program to the Lab Instructor

---End---