Student Name:_____Roll No:_____Section:_____

CS334 - Machine Learning

Lab 11 - Error Based Learning

Instructor: Ms. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce Error based learning models, such as Gradient Descent algorithm, Regression based classifier for categorical and continues features, and Support Vector Machine (SVM).

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 Simple Linear Regression Model

Regression is a method of modeling a target value based on independent predictors. This method is mostly used for forecasting and finding out cause and effect relationship between variables. Regression techniques mostly differ based on the number of independent variables and the type of relationship between the independent and dependent variables.

Simple linear regression is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable. The red line in the Figure-1 is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best. The line can be modeled based on the linear equation shown below.

$$M_W(d) = W[0] + W[1]d[1]$$

The motive of the linear regression algorithm is to find the best values for $W[0]$ and $W[1]$. Before moving on to the algorithm, let's have a look at two important concepts you must know to better understand linear regression.

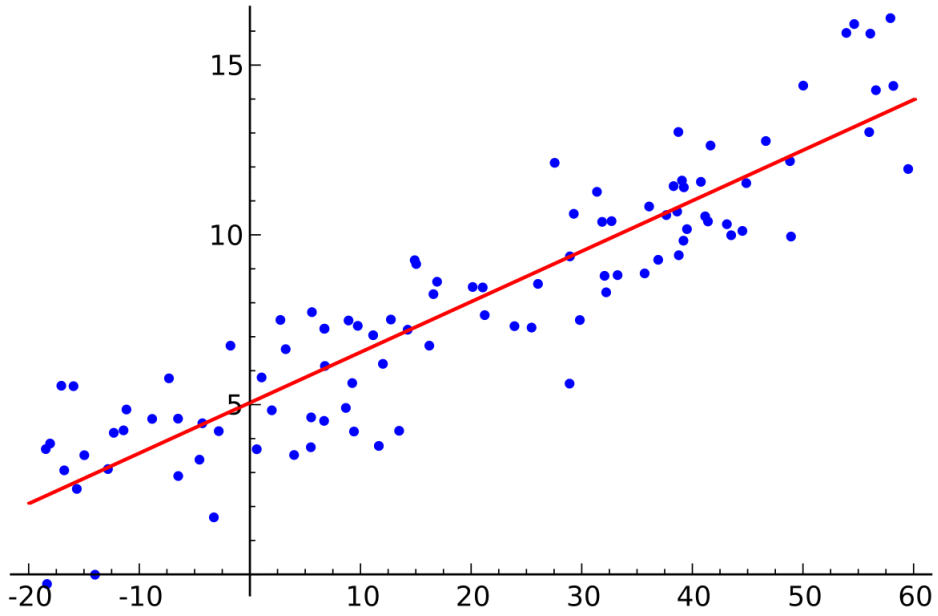

Figure 1: Linear Regression

1.1 Cost Function

The cost function helps us to figure out the best possible values for $W[0]$ and $W[1]$ which would provide the best fit line for the data points. Since we want the best values for $W[0]$ and $W[1]$, we convert this search problem into a minimization problem where we would like to minimize the error between the predicted value and the actual value.

$$L_2((M_W), \overrightarrow{D}) = \sum_{i=1}^{n} (t_i - M_W(d_i))^2$$

$$L_2((M_W), \overrightarrow{D}) = \sum_{i=1}^{n} (t_i - (W[0] + W[1]d[1]))^2$$

1.2 Gradient Descent

The next important concept needed to understand linear regression is gradient descent. Gradient descent is a method of updating $W[0]$ and $W[1]$ to reduce the cost function(MSE). The idea is that we start with some values for $W[0]$ and $W[0]$ and then we change these values iteratively to reduce the cost. Gradient descent helps us on how to change the values.

To draw an analogy, imagine a pit in the shape of U and you are standing at the topmost point in the pit and your objective is to reach the bottom of the pit. There is a catch, you can only take a discrete number of steps to reach the bottom. If you decide to take one step at a time you would eventually reach the bottom of the pit but this would take a longer time. If you choose to take longer steps each time, you would reach sooner but, there is a chance that you could overshoot the bottom of the pit and not exactly at the bottom. In the gradient descent algorithm, the number of steps you take is the learning rate. This decides on how fast the algorithm converges to the minima.

Sometimes the cost function can be a non-convex function where you could settle at a local minima but for linear regression, it is always a convex function.

You may be wondering how to use gradient descent to update $W[0]$ and $W[1]$. To update $W[0]$ and $W[1]$, we take gradients from the cost function. To find these gradients, we take partial derivatives with respect to $W[0]$ and $W[1]$. Now, to understand how the partial derivatives are found below you would require some calculus but if you don't, it is alright. You can take it as it is.

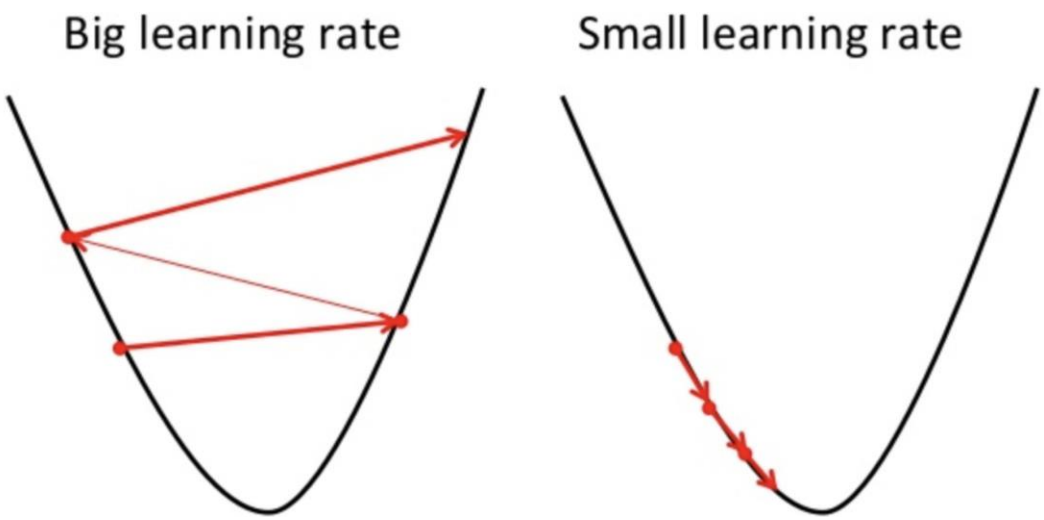

Figure 2: Gradient Descent

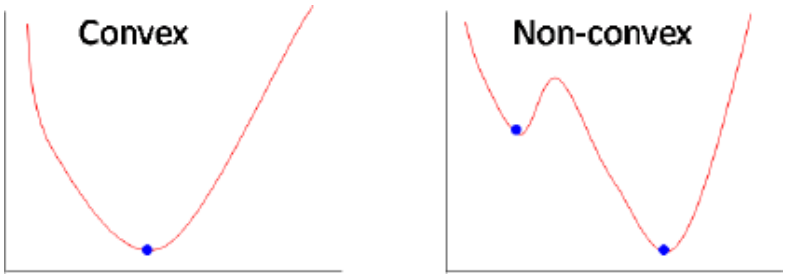

Figure 3: Convex vs Non-Convex

$$\frac{\partial}{\partial W[0]} \frac{1}{2} \sum_{i=1}^{n} (t_i - (W[0] + W[1]d[1]))^2 = 0$$

$$\frac{\partial}{\partial W[1]} \frac{1}{2} \sum_{i=1}^{n} (t_i - (W[0] + W[1]d[1]))^2 = 0$$

The partial derivatives are the gradients and they are used to update the values of $W[0]$ and $W[1]$. Alpha is the learning rate which is a hyper-parameter that you must specify. A smaller learning rate could get you closer to the minima but takes more time to reach the minima, a larger learning rate converges sooner but there is a chance that you could overshoot the minima.

1.3 Code

Let's start with the easiest of the two methods, i.e using *scikit-learn* library to build our linear regression model.

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('./LinearRegression/train.csv')
df_test = pd.read_csv('./LinearRegression/test.csv')

x_train = df_train['x']
```

```
y_train = df_train['y']
x_test = df_test [ ' x ']
y_test = df_test['y']

x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np . array ( x_test )
y_test = np.array(y_test)

x_train = x_train.reshape(-1,1)
x_test = x_test.reshape(-1,1)
```

We use pandas library to read the train and test files. We retrieve the independent(x) and dependent(y) variables and since we have only one feature(x) we reshape them so that we could feed them into our linear regression model.

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

clf = LinearRegression(normalize=True)
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
print(r2_score(y_test,y_pred))
```

We use *scikit-learn* to import the linear regression model. we fit the model on the training data and predict the values for the testing data. We use R2 score to measure the accuracy of our model. Now, let's build our own linear regression model from the equations above. We will be using only *numpy* library for the computations and the R2 score for metrics.

```
## Linear Regression
import numpy as np

n = 700
alpha = 0.0001

a_0 = np.zeros((n,1))
a_1 = np.zeros((n,1))

epochs = 0
while(epochs < 1000):
    y = a_0 + a_1 * x_train
    error = y - y_train
    mean_sq_er = np.sum(error**2)
    mean_sq_er = mean_sq_er/n
    a_0 = a_0 - alpha * 2 * np.sum(error)/n
    a_1 = a_1 - alpha * 2 * np.sum(error * x_train)/n
    epochs += 1
    if ( epochs % 10 == 0) :
        print(mean_sq_er)
```

We initialize the value 0.0 for $w[0]$ and $w[1]$. For 1000 epochs we calculate the cost, and using the cost we calculate the gradients, and using the gradients we update the values of $w[0]$ and $w[1]$. After 1000 epochs, we would've obtained the best values for $w[0]$ and $w[1]$ and hence, we can formulate the best fit straight line.

```
import matplotlib.pyplot as plt

y_prediction = a_0 + a_1 * x_test
print('R2 Score:',r2_score(y_test,y_prediction))

y_plot = []
for i in range ( 100 ) :
    y_plot.append(a_0 + a_1 * i)
plt . figure ( figsize =( 10 , 10 ) )
plt . scatter ( x_test , y_test , color = ' red ', label = ' GT ')
plt.plot(range(len(y_plot)),y_plot,color='black',label = 'pred')
plt.legend()
plt.show()
```

The test set contains 300 samples, therefore we have to reshape $W[0]$ and $W[1]$ from 700x1 to 300x1. Now, we can just use the equation to predict values in the test set and obtain the R2 score 0.99.

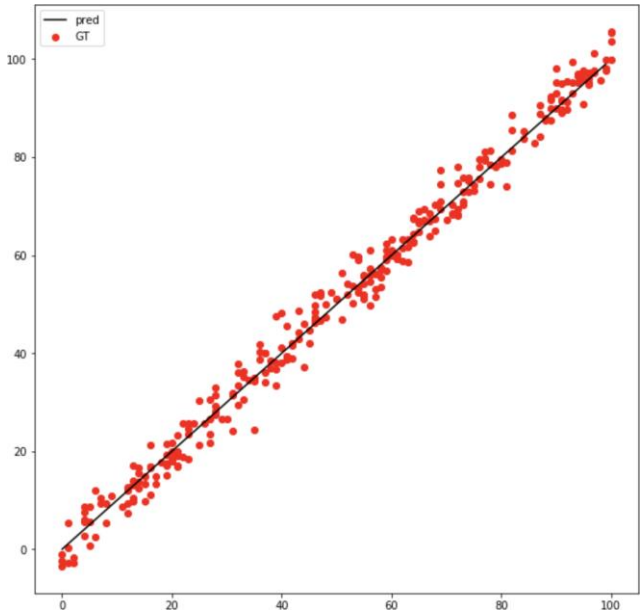

Figure 4: Regression Line

2 Build Regression Model Using Gradient Descent Algorithm

Now follow the step and write a class that build regression model using gradient descent algorithm. The class initialize with data frame, and target variable.

```
--------------------------------------------------------------------------
Iteration -  1
--------------------------------------------------------------------------
Current Weights :  [-0.14593556814, 0.2337559429,
-0.043436189199999996, 0.12081484572]
Current target values :  [320, 380, 400, 390, 385, 410, 480, 600, 570, 620]
Predicted target values :  [116.55829112506, 128.11577970246, 144.39182332706002,
146.90312751285998, 154.95427694676, 163.30947970506, 179.41177857286002,
205.03805991346002, 214.30142525106, 233.21908162906]
Updated Weights:  [-0.14587819220247372, 0.27716778469514236,
-0.042933663679943716, 0.12243092509046345]
--------------------------------------------------------------------------
```

Step 1: Write the Linear Regression class (e.g. LinearRegression),

```
class  LinearRegression:
    def  __init__(df,target):
         self  .  df   =   df
         self.target=target
```

Step 2: Write setter functions for alpha value and weight vector,

```
def setAlpha(self,value):
        self.alpha  =  value

    def  setWeightVector(self,WVector=[],random=False):
             if  rnadom==False  and  len(WVector)!=0:
                 W. self = WVector
             else:
                 n = df . shape [ 1 ] -1
                 self.W  =  genRandom(n)
```

Step 3: Write function that assign random weight vector,

```
def gen Random ( n):
    import numpy as np
    W = np.random.uniform(low=-1,high=1,size=n)
    return W
```

Step 4: Define loss function for constant weight,

```
def L2c(df,target,pred_target):
    y = df.loc[:,[target]].values.flatten().tolist()
    y_pred = pred_target
    nrows = df.shape[0]
    loss=0
    for j in range(nrows):
        loss += (y[j]-y_pred[j])
    return loss
```

Step 4: Define loss function for feature weights,

```
def L2f(df,target,pred_target,feature):
    y = df.loc[:,[target]].values.flatten().tolist()
    y_pred = pred_target
    d =df.iloc[:,feature].values.flatten().tolist()
    nrows = df.shape[0]
    loss=0
    j=0
    for j in range(nrows):
        loss += (y[j]-y_pred[j])*d[j]
    return loss
```

Step 5: Error delta function,

```
def errorDelta(loss, alpha):
    return alpha*loss
```

Step 6: Write prediction function for target values on current weights,

```
def pred(dFrame,target,WVector):
    if dFrame.shape[1]!=len(WVector):
        print('feature length is not equal to weight vector')
        return []
    pred = list()
    predict=0
    df = dFrame.drop(['Rental Price'],axis=1)
    W = WVector
    nrows = df . shape [ 0 ]
    ncolumns = df.shape[1]
    for i in range(nrows):
        j=0
        predict=0
        A = df.iloc[i]
        row = A.values.flatten().tolist()
        for j in range(ncolumns):
            if j==0:
                predict +=W[0]
            else:
                predict+=W[j]*row[j-1]
        pred.append(predict)
    return pred
```

Step 7: Write weights update function that returns new weight vector,

```
def weights_update(df,target,WVector):
    loss_vec = list()
    new_WVector =list()
    W = WVector
    print('Current Weights : ', W)
    print('Current target values : ', df.loc[:,[target]].values.flatten().
    tolist())
    pred_target = pred ( df , target , W)
    print ( ' Predicted target values : ', pred_target )
    nrows = df.shape[0]
    ncolumns = df.shape[1]
    #print('Shape :',nrows,'x',ncolumns)
    j=0
```

```
    loss = L2c(df,target,pred_target)
    loss_vec . append ( loss )
    #print('W[0]: ',new_WVector[0])
    for j in range(ncolumns-1):
        loss=0
        #feature = df.iloc[j]
        loss = L2f(df,target,pred_target,j)
        loss_vec.append(loss)

    for i in range ( len ( loss_vec )):
        new_WVector.append(WVector[i]+(loss_vec[i]*alpha))
    print('Updated Weights: ',new_WVector)

    return new_WVector
```

Step 8: Write setter function that set the values of iteration,

```
def epoch(self,n):
        self.epochs = n
```

Step 9: Write a function that fit your model and print each iteration,

```
def fit(self):
        for i in range(self.epochs):
            print('Iteration - ',i)
            print('-------------------------------------------------------------------------------')
            newW = weights_update(df,target,W)
            print('-------------------------------------------------------------------------------')
            W = newW
```

SAMPLE OUTPUT

```
--------------------------------------------------------------------------------
Iteration -  1
--------------------------------------------------------------------------------
Current Weights :  [-0.14593556814, 0.2337559429,
-0.043436189199999996, 0.12081484572]
Current target values :  [320, 380, 400, 390, 385, 410, 480, 600, 570, 620]
Predicted target values :  [116.55829112506, 128.11577970246, 144.39182332706002,
146.90312751285998, 154.95427694676, 163.30947970506, 179.41177857286002,
205.03805991346002, 214.30142525106, 233.21908162906]
Updated Weights:  [-0.14587819220247372, 0.27716778469514236,
-0.042933663679943716, 0.12243092509046345]
--------------------------------------------------------------------------------
```

Step 10: Write a function that print accuracy R^2 score of after each iteration,

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

3 Lab Tasks

1. Write a complete Linear Regression class as defined in Section 2.

4 Home Work Tasks

1. Submit a complete report on Assignment - 1 & 1.1 in following format,
   - Title of Page with following details,
     - UIT logo
     - Assignment name & and number
     - Student name & roll number
     - program name and section
     - Date of submission
   - Table of contents

- Introduction
- Description of data set
- Detailed discussion on Feature Selection
- Discussion on selected features and their importance
- Detailed discussion on models Selection
- Discussion on selected models and their performance
- System diagram of complete ML project and discuss each component of the diagram
- Detailed discussions on results
- Strong and weak points of the developed model
- conclusion