BUAN 6341 APPLIED MACHINE LEARNING ASSIGNMENT 1

Project Report Linear and logistic regression & Experimentations

By Arpit Chaukiyal (axc176630)

Tasks:

1. Divide the dataset into train and test sets sampling randomly. Use only predictive attributes and the target variable (do not use non-predictive attributes). Also, do not use G1 and G2.

I have divided the dataset into test and train by using "train_test_split" method of sklearn.model_selection.train_test_split keep the split ration as 80% for train and 20% for test.

```
In [59]: train, test = train_test_split(data, test_size=0.2,random_state=123)

         print(train.shape)
         print(test.shape)

         (316, 31)
         (79, 31)
```

The two variables has been removed as follows:

```
data = pd.read_csv("student-mat.csv", delimiter =";")

#-------------------------------------------------------------
# have to delete the 'G1' and 'G2' variables also
#-------------------------------------------------------------
data = data.drop(['G1', 'G2'], axis = 1)
```

2. Use linear regression to predict the final grade (G3). Report and compare your train and test error/accuracy metrics. You can pick any metrics you like (e.g. mean squared error, mean absolute error, etc.).

A function in python has been created, to implement the linear regression which uses the gradient decent as an optimization algorithm. The cost function for this gradient decent is mean squired error.

```
def cost_LinearRegression(X_train,y_train,theta):
    return np.sum(np.power(((X_train*theta.T) - y_train),2))/(2*len(y_train))
```

Mean squired error and Mean absolute error has been used as two accuracy matrices.

Functions “mean_squared_error” and “mean_absolute_error” has been used form “sklearn.metrics” to calculate these matrices.

```
print("Linear Regression -> Running the Gradient Descent Algorithm for "+data_set)
print("Mean Squared error = {}".format(round(mean_squared_error(y,y_predictions),6)))
print("Mean Absolute error = {}".format(round(mean_absolute_error(y,y_predictions),6)))
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
```

The accuracy for train and test dataset is as follows:

```
--------------------------------------------------------------------------------
Linear Regression -> Running the Gradient Descent Algorithm for Train-Data Set
Mean Squared error = 15.415601
Mean Absolute error = 2.99322
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Linear Regression -> Running the Gradient Descent Algorithm for Test-Data Set
Mean Squared error = 16.832029
Mean Absolute error = 3.331172
--------------------------------------------------------------------------------
```

3. Convert this problem into a binary classification problem. The target variable should have two grade categories.

Median is been choose for column “G3” as a threshold to make it a binary classification problem.

1: for all marks greater than or equal to median: Good_Marks

0: for all marks less than median: Not_Good_Marks

```
#Logistic regression

data = pd.read_csv("student-mat.csv", delimiter =";")

median = data['G3'].median()
data['G3'] = np.where(data['G3']>=median, 1, 0)


# 1: for all marks greater than or equal to median: Good_Marks
# 0: for all marks less than median: Not_Good_Marks

X_train,X_test,y_train,y_test,theta = prepare_data(data)
```

4. Implement logistic regression to carry out classification on this data set. Report accuracy/error metrics for train and test sets

Sigmoid function and the cost function has been created for implementing the logistic regression. To optimize the values of “theta”, “scipy.optimize.fmin_tnc” method is been used.

```
X_train,X_test,y_train,y_test,theta = prepare_data(data)

result = opt.fmin_tnc(func=cost_LogisticRegression, x0=theta, fprime=gradientDescent_LogisticR, args=(X_train, y_train)
```

Cost function:

```
# making the cost function for the logistic function.

def cost_LogisticRegression(theta, X,y):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)

    first = np.multiply(-y.T,np.log(sigmoid_func(X*theta.T)))
    second = np.multiply((1-y).T,np.log(1-sigmoid_func(X*theta.T)))
    return np.sum((first - second)/len(X))
```

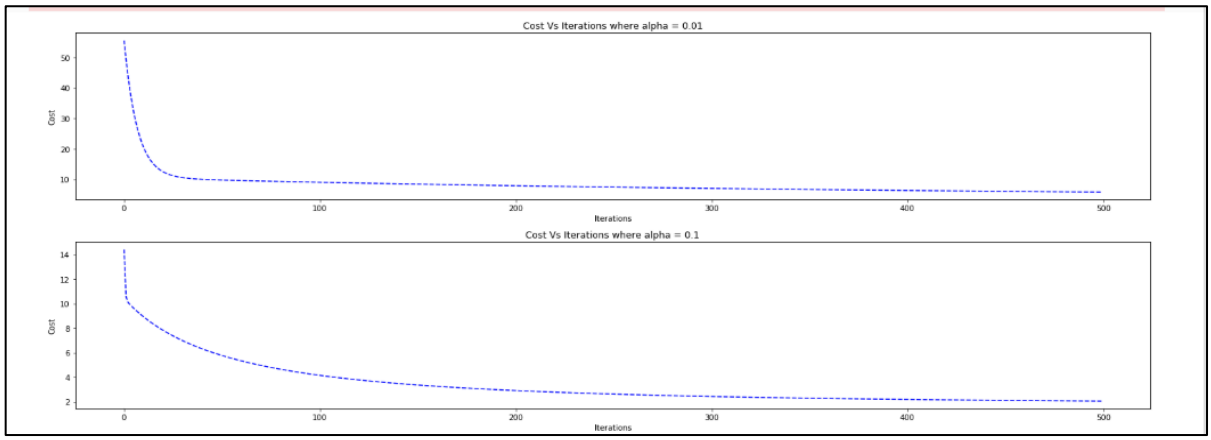
The accuracy for the logistic regressions are as follows:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Logistic Regression -> Running the algo on Test set
Accuracy is 60.76 %
Confusion Matrix =
[[ 0 31]
 [ 0 48]]
Mean Absolute error = 0.3924
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Logistic Regression -> Running the algo on Train set
Accuracy is 51.9 %
Confusion Matrix =
[[  3 152]
 [  0 161]]
Mean Absolute error = 0.481
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

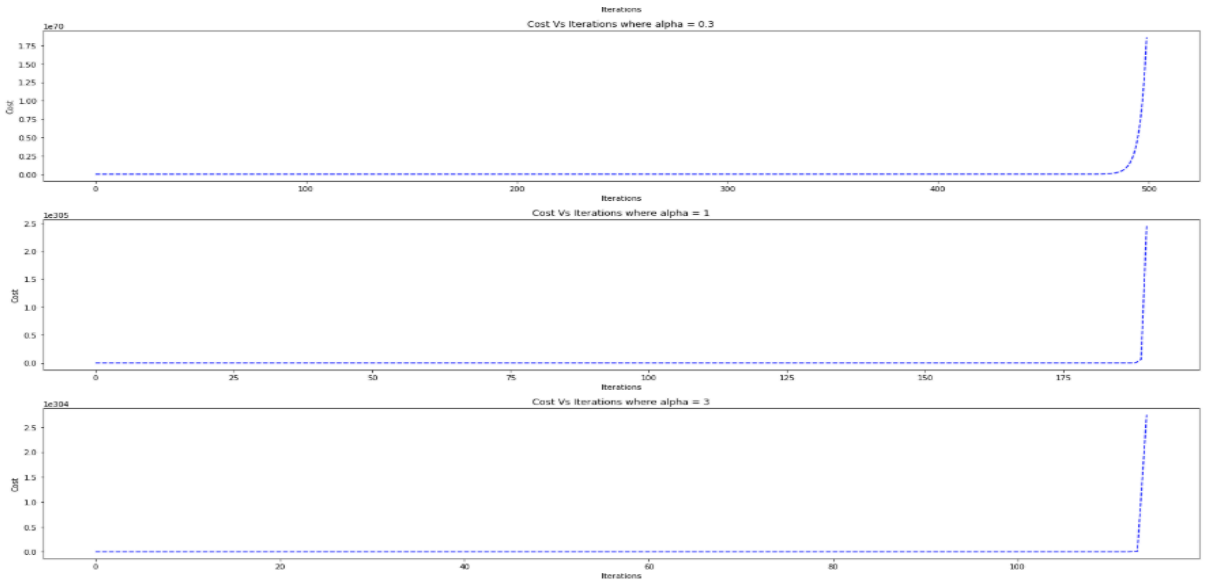
Experiments:

1. Experiment with various model parameters for both linear and logistic regression and report on your findings as how the error varies for train and test sets with varying these parameters. Plot the results. Report your best parameters. Examples of these parameters can be learning rate for gradient descent, convergence threshold, etc.

   Five different learning rates has been taken for the experimentation purpose, 0.01,0.1,0.3,1,3.



   Above is the graph for the error as the function of learning- rate – 0.01 and 0.1. The shape curve for alpha 0.01 depicts that is the better option as compared to alpha 0.1.



For the alpha values 0.3, 1 and 3, the graph above shows the missovers.

2. Pick ten features randomly and retrain your model only on these ten features. Compare train and test error results for the case of using all features to using ten random features. Report which ten features did you select randomly

Below is the method used for randomly selecting the 10 columns.

```
data_rand_features = pd.DataFrame(data.sample(n=10,axis = 1, random_state = 123))

if 'G3' not in data_rand_features.columns.names:
    data_rand_features = data_rand_features.drop(columns = [data_rand_features.columns[-1]])
    data_rand_features = pd.concat([data_rand_features, data['G3']], axis = 1)

X_train, X_test, y_train, y_test, theta = prepare_data(data_rand_features)
```

Accuracy:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Linear Regression -> Running the Gradient Descent Algorithm for Random_Train-Data Set
Mean Squared error = 18.946271
Mean Absolute error = 3.267415
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Linear Regression -> Running the Gradient Descent Algorithm for Random_Test-Data Set
Mean Squared error = 22.802704
Mean Absolute error = 3.802999
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

3. Now pick ten features that you think are best suited to predict the output, and retrain your model using these ten features. Compare to the case of using all features and to random features case. Did your choice of features provide better results than picking random features?

Below are the feature selected:

```
data_picked_features = data[['traveltime', 'failures', 'absences', 'Pstatus', 'Fjob','schoolsup','Medu','studytime','Mj

X_train, X_test, y_train, y_test, theta = prepare_data(data_rand_features)
alpha = 0.01
```

Accuracy:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Linear Regression -> Running the Gradient Descent Algorithm for Handpicked_Train-Data Set
Mean Squared error = 18.946271
Mean Absolute error = 3.267415
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Linear Regression -> Running the Gradient Descent Algorithm for Handpicked_Test-Data Set
Mean Squared error = 22.802704
Mean Absolute error = 3.802999
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

This is because when we are selecting the random features, then the model tries to fit the data to less important features and the less significant features often have more noises in them which decreases the accuracy of model