Name: Adit Devendra Patil Id:1001757372

Linear Regression on IRIS Dataset

Introduction:

In statistics, linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

A machine learning model using multivariate linear regression on IRIS dataset (http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data)
We have worked on Multivariate Linear Regression as we have 4 independent values and 1 dependent values in the IRIS Dataset.

Section of Problem:

We need to train the model via Multivariate Linear Regression along with cross validation and also achieve classification. In this data there are 5 columns in total. In this, 4 are feature columns which will decide the value of 5th column which is the label column.
There are 3 kind of values in label column, Iris-setosa, Iris-versicolor and Iris-virginica.
Our model should be able to predict and classify which of them should be based on the feature columns.

Data:

The data is following. We have added the column headers for better explanation.

```
In [3]: data.head()
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [ ]:
```

```
In [4]: rows, col = data.shape
        print("Rows : %s, column : %s" % (rows, col))
```

```
Rows : 150, column : 5
```

As we can see with the following screenshot of our notebook, the IRIS dataset has 150 rows and 5 columns. Out of those 5 columns, 4 of them are independent columns (sepal_length, sepal_width, petal_length, petal_width) and 1 of them is dependent (species).

Method:

1. Data Preprocessing: Data Preprocessing is that step in which the data gets transformed, or Encoded, to bring it to such a state that now the machine can easily parse it. Firstly, we will convert the dataset in features and label. Therefore, we are storing features in X and label in Y. Secondly, in this process we will first remove the null values if any.
   As we know the linear regression is a mathematical model, so there should not be any kind of categorical data in the label. Therefore, Thirdly, we will manually encode Iris-setosa, Iris-versicolor and Iris-virginica into numerical values 1, 2 , 3 in our Y.
   We will add a column bias units(1s) in the input matrix.

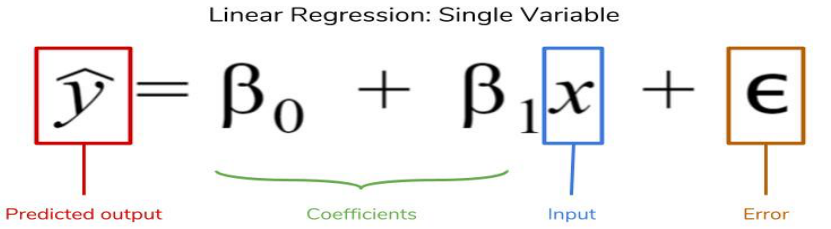
```
In [11]: data = data.dropna()
         y = data['species'].values.tolist()
         Y = [] #Lable preprocessing
         for x in y:
             if x == 'Iris-setosa':
                 Y.append(1)
             elif x == 'Iris-versicolor':
                 Y.append(2)
             else:
                 Y.append(3)

         Y = np.asarray(Y)
         Y = Y.reshape(rows,1)
         X = data.drop(['species'], axis=1).values # Input Feature Values

         X = np.hstack(((np.ones((rows,1))), X))# Adding one more column for bias
```

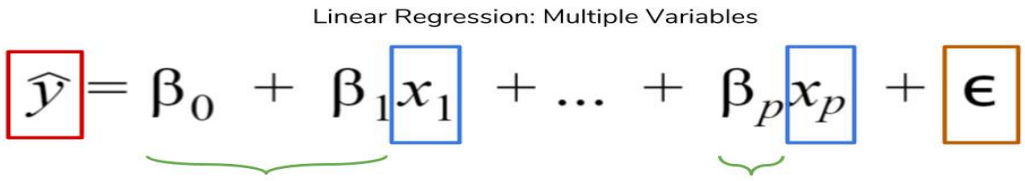
```
In [33]: print(Y)
```

```
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[2]
[2]
[2]
[2]
[2]
[2]
[2]
[2]
```

2. Training Model: Training a model simply means learning (determining) good values for all the weights and the bias from labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called empirical risk minimization.
   As we have more than one feature columns, we will be using multivariate linear regression.

The equation for Linear regression is.





Further it is represented as:

$$Y = \beta^T X$$

In mathematical optimization and decision theory, a loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event.
We will be using Root Mean square

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

In [40]:
```
iteration = 10000 #More iterations more learning
alpha = 0.003 # Learning rate
J = np.zeros(iteration)
```

In [41]:
```
def predict(X,b):
    return np.round(np.dot(X, b.T))
```

In [42]:
```
def error_func(b, X, Y):
    rows = X.shape[0]
    b -= ((alpha/rows) * np.dot((np.dot(X, b.T) - Y).reshape(1,rows), X))
    return b
```

In [51]:
```
def trainn(X, Y):
    np.random.seed(0)
    b = np.random.randn(1,5) #Random values of beta
    print("b : %s" % (b))
    for i in range(iteration):

        J[i] = np.sqrt(sum((predict(X, b) - Y) ** 2) / rows)

        error_func(b, X, Y)

    diagram = plt.subplot(111)
    diagram.plot(np.arange(iteration), J)
    #ax.set_ylim([0,0.15])
    plt.ylabel("Cost Values", color="Blue")
    plt.xlabel("Iterations Count", color="Blue")
    plt.title("Mean Squared Error vs Iterations")
    plt.show()
    return b
```

3. Cross Validation: Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

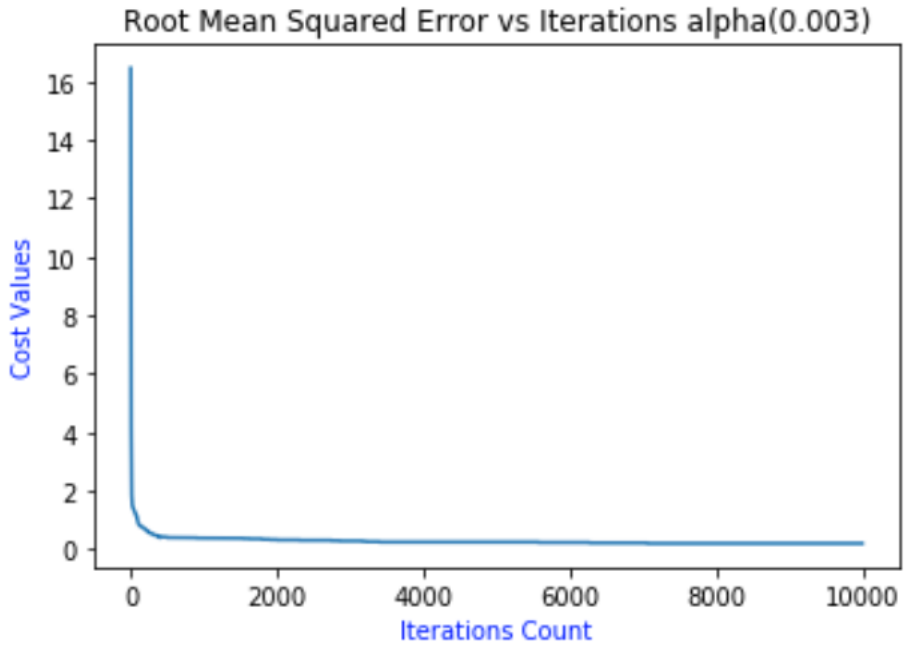
   Our data is divided into training and testing data. We will be using 10 folds as k-fold in our case.

In [48]:
```
def cross_val(data, k=10):
    data = data.sample(frac=1)
    splits = np.array_split(data, k)
    accuracies = []
    for i in range(len(splits)):
        test = splits[i] # Choosing the 10% data splits as testing
        X_test = test.iloc[:,:-1].values
        class_lables = test.iloc[:,-1].values # Y_test
        class_lables = class_lables.reshape(len(class_lables), 1)
        temp = splits[:]
        temp.pop(i) # Choosing the other 90% data splits as training
        train = pd.concat(temp) #merge all the splits forming training DS
        X_train = train.iloc[:,:-1].values
        Y_train = train.iloc[:,-1].values
        Y_train = Y_train.reshape(len(X_train),1)
        b = trainn(X_train, Y_train)
        # train function returns beta value
        prediction = predict(X_test, b)
        accuracies.append((sum(prediction == class_lables)/float(len(class_lables)) * 100)[0])
    return accuracies
```

Results:

Our model is able to predict the class of the test data. The model is trained well with the training data and able to give the results in test data.
Through the predicted values we are able to classify the label.
The cost exponentially decreases as our model trains.

As we see from below screenshots.
Our accuracy of model increases with the increase in alpha value.




```
In [34]: accuracy = (sum(prediction == Y)/float(len(Y)) * 100)[0]
         print("Overall accuracy of the model is %s" % (accuracy))

         Overall accuracy of the model is 96.0
```

References:

https://machinelearningmastery.com/k-fold-cross-validation/#:~:text=Cross%2Dvalidation%20is%20a%20resampling,k%2Dfold%20cross%2Dvalidation.
https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85
https://www.kaggle.com/amarpandey/implementing-linear-regression-on-iris-dataset
https://www.geeksforgeeks.org/python-basics-of-pandas-using-iris-dataset/
https://www.kaggle.com/jchen2186/machine-learning-with-iris-dataset
https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/
https://statweb.stanford.edu/~susan/courses/s60/split/node60.html