Programming Assignment 1

Nishanth Ravula
Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
nravula@buffalo.edu

1 Simple Linear Regression

It is a type of regression algorithm in which it models the dependent variable with one independent variable. As from the name Linear Regression we will be getting a Linear or a sloped straight line when we visualize the model. The dependent variable must be a continuous / real values and independent variable can have a continuous values or a categorical value.

The Simple Linear Regression model can be represented using an equation, the equation is as follows:

$$y=c+ m_1x_1 + m_2x_2 +...+ m_nx_n$$

y is the response

c is the intercept

m_1 is the coefficient for the first feature

m_n is the coefficient for the nth feature

In our case: $y=c+m_1 \times TV$

The m values are called the model coefficients or model parameters.

1.1 Dataset

The dataset used in this is tv marketing data. In which we consider sales based on TV marketing budget. We'll be using linear regression model and TV as the predictor variable to predict sales of the TV's. The dataset is in the references.

1.2 Process

Basically, Simple Linear Regression can be implemented in four steps. The steps are as follows:
Step 1: Data Pre-processing
Step 2: Fitting the Simple Linear Regression to the Training Set
Step 3: Predicting the test result
Step 4: Visualizing the training set

Step1: Data Pre-processing:
1.Import the libraries that which are needed to load, process and map the data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

2.Load the data-set into the code. Here the crucial part comes where we need to extract dependent and independent variables from the loaded data-set.

```
dataset = pd.read_csv('tvmarketing.csv')
```

4. In the above code, for X variable, as we need to remove the last column, we use -1. And for Y variable we need the only last column so we use 1 as the indexing starts from 0.

```
X = dataset.iloc[ : , : 1 ].values
Y = dataset.iloc[ : , 1 ].values
```

5. Here the X is dependent variable and the Y is independent variable.

6. Now the needs to be split into 2 data-sets i.e., training data-set and test data-set. Where we train our model with training data-set and test the model with test data-set.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, train_size = 0.7, random_state = 0)
```

Step 2: Fitting the Simple Linear Regression to the Training Set
Now we need to fit the model to the training dataset. For that we will import the LinearRegression class of the linear_model library from the scikit learn. After importing the class, we are going to create an object of the class named as a regressor.

```
from sklearn.linear_model import LinearRegression
train_test_split

regressor = LinearRegression()
regressor = regressor.fit(X_train, Y_train)
```

We fitted our Simple Linear Regression object to the training set in the above code by using the fit() function. We have passed the training datasets for the dependent and independent variables, x train and y train, to the fit() method. To make it simple for the model to learn the correlations between the predictor and target variables, we fitted our regressor object to the training set.

Step 3: Predicting the test result
Sales is a dependent variable and an independent one (Marketing). As a result, our model is now prepared to forecast the results for the fresh observations. In this stage, we'll give the model a test dataset (new observations) to see if it can predict the intended result.
A prediction vector called y pred and x pred, containing predictions for the test dataset and the training set, respectively, will be created.

```
Y_pred = regressor.predict(X_test)
X_pred = regressor.predict(X_train)
```

On executing the above lines of code, two variables named y_pred and x_pred will be generated.

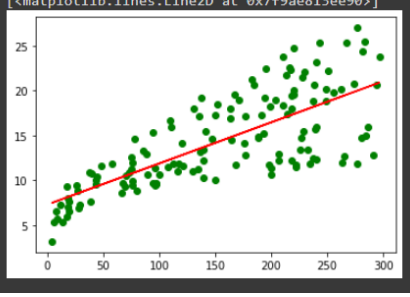
Step: 4. visualizing the Training set results:

We will now visualize the training set outcome in this phase. The scatter() function of the pyplot package, which we had loaded during the pre-processing stage, will be used for this. A scatter plot of the observations will be produced via the scatter () function.

We will plot the marketing is on the x-axis and their tv on the y-axis. We will pass the training set's actual values x train, y train, and the color of the observations—to the function. For the sake

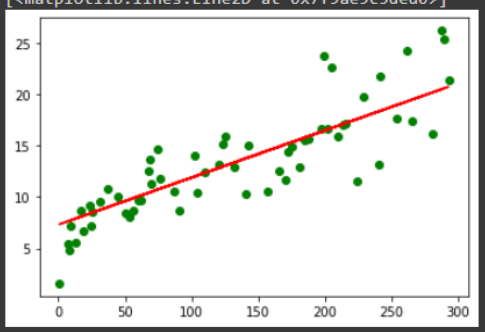
of this observation, we will choose the color green, but you can use any other color of your choosing.

```
[ ]  plt.scatter(X_train , Y_train, color = 'green')
     plt.plot(X_train , regressor.predict(X_train), color ='red')

     [<matplotlib.lines.Line2D at 0x7f9ae813ee90>]
```



We visualized our performance of the model on the training set in the preceding phase. We will now repeat the process for the Test set. The entire program will be identical to the one above, with the exception that x test and y test will be used in place of x train and y train.

```
[ ]  plt.scatter(X_test , Y_test, color = 'green')
     plt.plot(X_test , regressor.predict(X_test), color ='red')

     [<matplotlib.lines.Line2D at 0x7f9ae9c5ded0>]
```



To get the accuracy of the model and how much percent the model is not matched we need to find mean square error and RMSE. RMSE is the standard deviation of the errors which occur when a prediction is made on a dataset. This is the same as MSE (Mean Squared Error) but the root of the value is considered while determining the accuracy of the modelTo find mean square and r_square first we need to import mean_squared_error from sklearn metrics. After calculating the mean_squared_error and r_squared for Y_test, Y_pred for the above data we got and accuracy of 72 percent and a 7.9 percent error rate.

```
[ ]  from sklearn.metrics import mean_squared_error, r2_score
     meanSqrd = mean_squared_error(Y_test, Y_pred)
     meanSqrd

     7.497479593464674

[ ]  r_squared = r2_score(Y_test, Y_pred)
     r_squared

     0.725606346597073

[ ]  # this mse =7.9 means that this model is not able to match the 7.9 percent of the values
     # r2 means that your model is 72% is accurate on test data .
```

References

[1] zohaib123. (2021, February 7). *TV marketing prediction*. Kaggle. Retrieved September 30, 2022, from https://www.kaggle.com/code/zohaib123/tv-marketing-prediction/data?select=tvmarketing.csv

2 Multiple Linear Regression

It is a type of regression algorithm in which it models the dependent variable with two or more independent variable. It can also be a non-linear, where the dependent and independent do not follow a straight line.

Multiple linear regression formula
The formula for a multiple linear regression is:

$$y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n + \epsilon$$

y = the predicted value of the dependent variable

B0 = the y-intercept (value of y when all other parameters are set to 0)

B1X1 = The regression coefficient (B1) of the first independent variable (X1) (a.k.a. the effect that increasing the value of the independent variable has on the predicted y value)

… = do the same for however many independent variables you are testing

BnXn = the regression coefficient of the last independent variable

ϵ = model error (a.k.a. how much variation there is in our estimate of y)

Both linear and nonlinear regression use graphs to track a specific response using two or more variables. Non-linear regression, on the other hand, is typically difficult to implement because it is based on trial-and-error assumptions.

Assumptions:

1.A linear relationship between the dependent and independent variables.

2. The independent variables are not highly correlated with each other.

3. The variance of the residuals is constant.

4. Independence of observation

5. Multivariate normality

2.1 Dataset

The dataset used in this is Test scores for general Psychology data. In which final score is dependent variable and exam1, exam2, exam3 are the independent variables. We'll be using multiple linear regression model to predict final exam scores using exam1, exam2, exam3. The data set is in the reference.

2.2 Process

Step1: Data Pre-processing:
1.Import the libraries that which are needed to load, process and map the data.

```
import pandas as pd
import numpy as np
```

2.Load the data-set into the code. Here the crucial part comes where we need to extract dependent and independent variables from the loaded data-set.

```
dataset = pd.read_csv('Psychology data.csv')
"""
The data (X1, X2, X3, X4) are for each student.
X1 = score on exam #1
X2 = score on exam #2
X3 = score on exam #3
X4 = score on final exam
"""
X = dataset.iloc[ : , :-1].values
Y = dataset.iloc[ : ,  3 ].values
```

4. In the above code, for X variable, as we need to remove the last column, we use -1. And for Y variable we need the only last column so we use 3 as the indexing starts from 0.

5. Here the X is dependent variable and the Y is independent variable.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.7, random_state = 0)
```

6. Now the needs to be split into 2 data-sets i.e., training data-set and test data-set. Where we train our model with training data-set and test the model with test data-set.

Step 2: Fitting the Multiple Linear Regression to the Training Set
Now we need to fit the model to the training dataset. For that we will import the LinearRegression class of the linear_model library from the scikit learn. After importing the class, we are going to create an object of the class named as a regressor.

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, Y_train)
```

Step 3: Predicting the test result
Our model is now prepared to forecast the results for the fresh observations. In this stage, we'll give the model a test dataset (new observations) to see if it can predict the intended result.
A prediction vector called y pred and x pred, containing predictions for the test dataset and the training set, respectively, will be created.

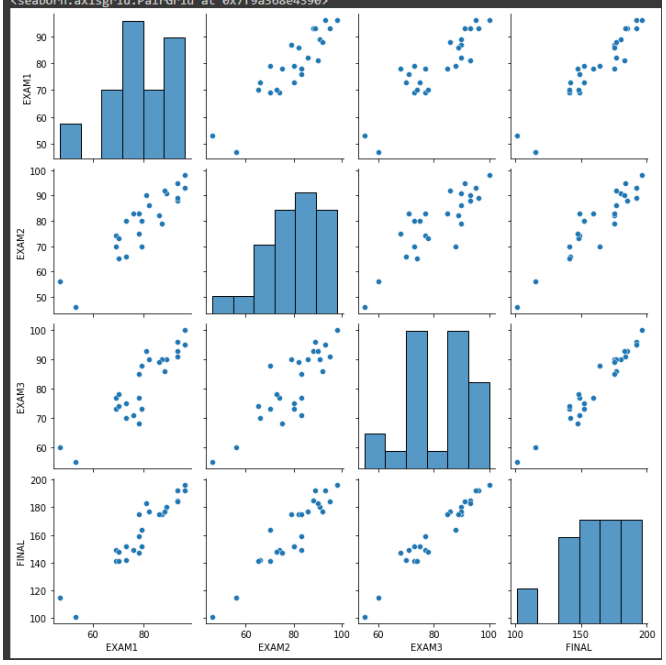
```
y_pred = regressor.predict(X_test)
x_pred = regressor.predict(X_train)
regressor.score(X_test,Y_test)
```

After that by running regressor.score(X_test, Y_test) we will get the accuracy of the model that is predicting the data. For the given data the accuracy is 95 percent and by finding the error term from mean squared error we are getting an error term of 18.

```
from sklearn.metrics import mean_squared_error, r2_score
meanSqrd = mean_squared_error(Y_test, y_pred)

r_squared = r2_score(Y_test, y_pred)
meanSqrd, r_squared
```

```
import seaborn as sc
sc.pairplot(dataset)
```

```
<seaborn.axisgrid.PairGrid at 0x7f9a368e4390>
```



Reference:

[1] Data for multiple linear regression. (n.d.). Retrieved September 30, 2022, from https://college.cengage.com/mathematics/brase/understandable_statistics/7e/students/datasets/mlr/frames/frame.html

3 Logistic Linear Regression

Despite the word 'regression' in its name, Logistic Regression is a type of parametric classification model in the realm of Machine Learning. This means that logistic regression models contain a fixed number of parameters that depend on the amount of input features and output categorical prediction, such as whether a plant belongs to a specific species or not.

Unlike linear regression, we do not fit a straight line to our data in logistic regression. Instead, we fitted our observations to an S-shaped curve known as a Sigmoid.
The formula for the sigmoid function is the following:

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

Some of the usecases of the logistic regression are as follows:

1. In drug research to tease apart the effectiveness of medicines on health outcomes across age, gender and ethnicity;

3.1 Dataset

The dataset we used to execute the logistic regression is BMI data, which is the body mass index of a group of people that includes both male and female members. The data includes height, weight, gender, and body mass index calculated using the specified height and weight. The dataset can be referred from the reference section.

3.2 Process

Step1: Data Pre-processing:

1.Import the libraries pandas, matplotlib and numpy which are needed to load, process and map the data.

2. Load the dataset using pd.read_csv and need to separate the dependent and independent variables from the uploaded dataset.

```
dataset = pd.read_csv('bmi.csv')
X = dataset.iloc[:,1 : 3].values
y = dataset.iloc[:, 3].values
```

3. Now the data needs to be split into 2 data-sets i.e., training data-set and test data-set. Where we train our model with training data-set and test the model with test data-set. The test size is 0.25

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Step 2: Fitting the Multiple Linear Regression to the Training Set

we will import the StandardScaler class of the sklearn.preprocessing library from the scikit learn. Here we will be using sc.fit_transform and sc.transform.
fit() : used for generating learning model parameters from training data.
transform() : parameters generated from fit() method,applied upon model to generate transformed data set.
fit_transform() : combination of fit() and transform() api on same data set.

Basically Fit learns the structure of your data in order to identify categories and other preprocessing information. Once you have fitted your preprocessor, you can then use that fitted preprocessor to transform your data using that fitting information.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
Y_train
```

```
array([115, 177, 196, 152, 175, 192, 184])
```

Now we need to fit the model to the training dataset. For that we will import the LogisticRegression class of the linear_model library from the scikit learn. After importing the class, we are going to create an object of the class named as a classifier.

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()

classifier.fit(X_train, y_train)
```

Step 3: Predicting the test result

Our model is now ready to anticipate the outcomes of the new observations. We'll give the model a test dataset (new observations) at this point to determine if it can predict the desired result.

A prediction vector named y pred and an x pred will be constructed, each containing predictions for the test dataset and the training set.

```
y_pred = classifier.predict(X_test)
classifier.score(X_test, y_test)
```

```
0.864
```

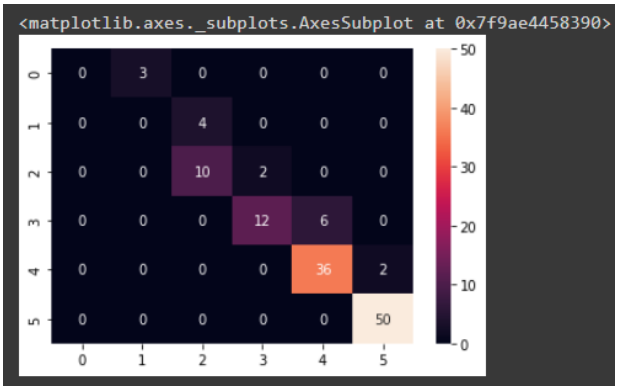
We will get the accuracy of the model using score. After executing the we got an accuracy of 86 percent. According to the accuracy our model is predicting data accurately.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

A confusion matrix is a table that is commonly used to describe the performance of a classification model (or "classifier") on a set of test data with known true values.

```
import seaborn as sns #using heatmaps
sns.heatmap(cm, annot=True)
```

The following heatmap is used to represent the confusion matrix.



Reference:
[1] H, M. Y. (2021, December 24). BMI Dataset. Kaggle. Retrieved October 1, 2022, from https://www.kaggle.com/datasets/yasserh/bmidataset

4 Decision Tress Regression

Decision Tree is one of the most commonly used, practical approaches for supervised learning. It can be used to solve both Regression and Classification tasks with the latter being put more into practical application.

It is a tree-structured classifier with three types of nodes. The Root Node is the initial node which represents the entire sample and may get split further into further nodes. The Interior Nodes represent the features of a data set and the branches represent the decision rules. Finally, the Leaf Nodes represent the outcome. This algorithm is very useful for solving decision-related problems.

Information Gain, or IG for short, measures the reduction in entropy or surprise by splitting a dataset according to a given value of a random variable. A larger information gain suggests a lower entropy group or groups of samples, and hence less surprise. You might recall that information quantifies how surprising an event is in bits. Lower probability events have more information, higher probability events have less information. Entropy quantifies how much information there is in a random variable, or more specifically its probability distribution. A skewed distribution has a low entropy, whereas a distribution where events have equal probability has a larger entropy.

In information theory, we like to describe the "surprise" of an event. Low probability events are more surprising therefore have a larger amount of information. Whereas probability distributions where the events are equally likely are more surprising and have larger entropy.

- Skewed Probability Distribution (unsurprising): Low entropy.
- Balanced Probability Distribution (surprising): High entropy.

4.1 Dataset

The dataset we used to execute the logistic regression is BMI data, which is the body mass index of a group of people that includes both male and female members. The data includes height, weight, gender, and body mass index calculated using the specified height and weight. The dataset can be referred from the reference section.

4.2 Process

Step1: Data Pre-processing:

1.Import the libraries pandas, matplotlib and numpy which are needed to load, process and map the data.

2. Load the dataset using pd.read_csv and need to separate the dependent and independent variables from the uploaded dataset.

```
dataset = pd.read_csv('bmi.csv')
X = dataset.iloc[:, [1, 3]].values
y = dataset.iloc[:, 3].values
```

3. Now the data needs to be split into 2 data-sets i.e., training data-set and test data-set. Where we train our model with training data-set and test the model with test data-set. The test size is 0.25

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Step 2: Fitting the Multiple Linear Regression to the Training Set

we will import the StandardScaler class of the sklearn.preprocessing library from the scikit learn. Here we will be using sc.fit_transform and sc.transform.
fit() : used for generating learning model parameters from training data.
transform() : parameters generated from fit() method,applied upon model to generate transformed data set.
fit_transform() : combination of fit() and transform() api on same data set.

Basically Fit learns the structure of your data in order to identify categories and other preprocessing information. Once you have fitted your preprocessor, you can then use that fitted preprocessor to transform your data using that fitting information.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Now we need to fit the model to the training dataset. For that we will import the DecisionTreeClassifier class of the tree library from the scikit learn. After importing the class, we are going to create an object of the class named as a classifier.

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Step 3: Predicting the test result

Our model is now ready to anticipate the outcomes of the new observations. We'll give the model a test dataset (new observations) at this point to determine if it can predict the desired result.

A prediction vector named y pred and an x pred will be constructed, each containing predictions for the test dataset and the training set.

```
y_pred = classifier.predict(X_test)
classifier.score(X_test, y_test)
```

```
1.0
```

We will get the accuracy of the model using score. After executing the we got an accuracy of 100 percent. According to the accuracy our model is predicting data accurately.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

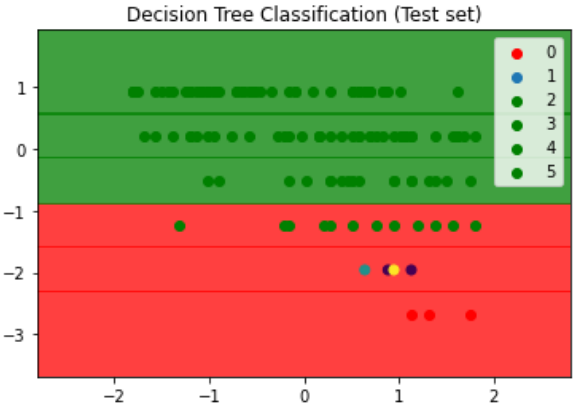
A confusion matrix is a table that is commonly used to describe the performance of a classification model (or "classifier") on a set of test data with known true values.

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.legend()
plt.show()
```

Using the above code we will plot the decision tree graph for that we need to import listedcolormap from matplotlib.colors. The below graph is for training dataset.

And for the test dataset we will be using the similar method of implementation to represent the decision tree graph.

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
#plt.xlabel('Age')
#plt.ylabel('Index')
plt.legend()
plt.show()
```



Reference:

[1] H, M. Y. (2021, December 24). BMI Dataset. Kaggle. Retrieved October 1, 2022, from https://www.kaggle.com/datasets/yasserh/bmidataset