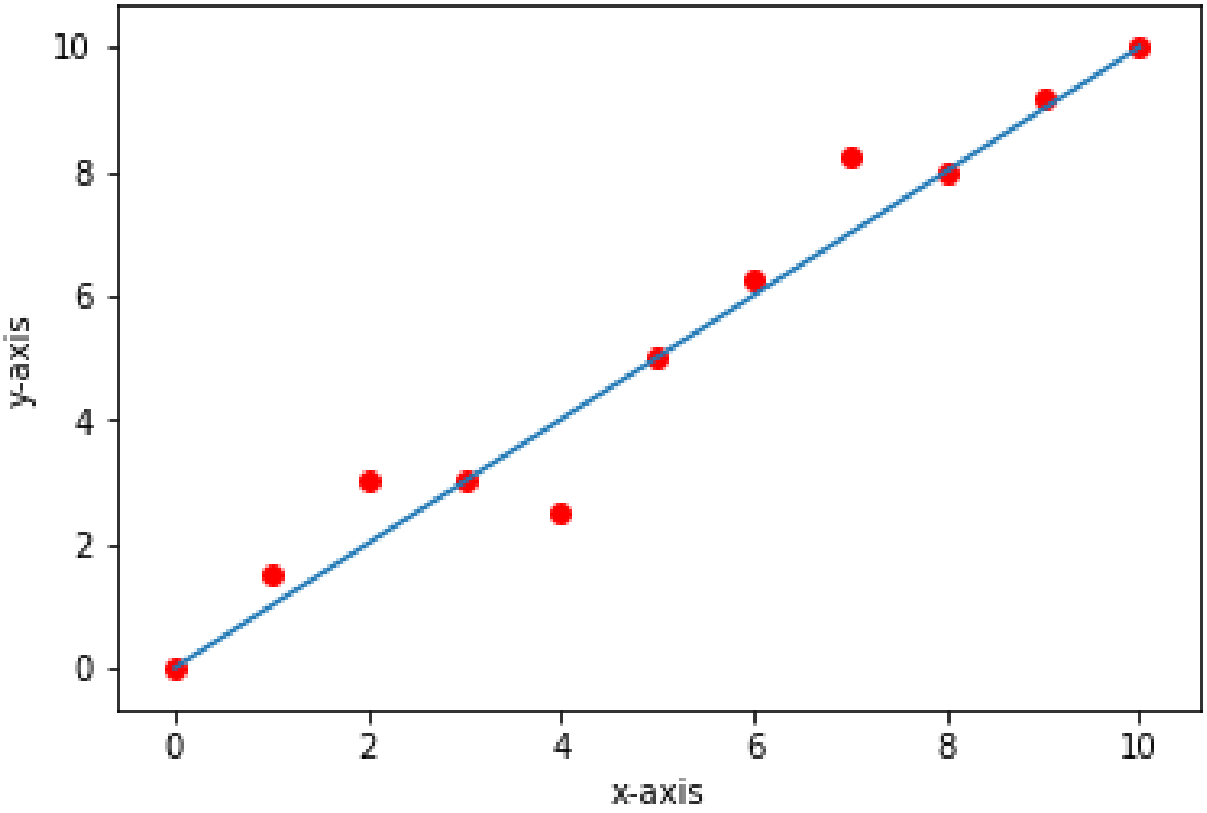
Linear Regression using Tensorflow Estimator



The Theory

Linear Regression is the process of fitting a line to the dataset.

Single Variable Linear Regression

The Mathematics

The equation of Line is

$$y = m * x + c$$

Where,

y = dependent variable

X = independent variable

C = intercept

The algorithm is trying to fit a line to the data by adjusting the values of m and c. Its Objective is to attain to a value of m such that for any given value of x it would be properly predicting the value of y.

There are various ways in which we can attain the values of m and c

1. Statistical approach
2. Iterative approach

Here we are using a scikit learn framework which internally uses iterative approach to attain the linear regression

The Dataset

Dataset consists of two columns namely X and y

Where

For List Price Vs. Best Price for a New GMC Pickup dataset

X = List price (in $1000) for a GMC pickup truck

Y = Best price (in $1000) for a GMC pickup truck

The data is taken from *Consumer's Digest.*

For Fire and Theft in Chicago

X = fires per 100 housing units

Y = thefts per 1000 population within the same Zip code in the Chicago metro area

The data is taken from U.S Commission of Civil Rights.

For Auto Insurance in Sweden dataset

X = number of claims

Y = total payment for all the claims in thousands of Swedish Kronor

The data is taken from Swedish Committee on Analysis of Risk Premium in Motor Insurance.

For Gray Kangaroos dataset

X = nasal length (mm ¥10)

Y = nasal width (mm ¥ 10)
for a male gray kangaroo from a random sample of such animals

The data is taken from Australian *Journal of Zoology*, Vol. 28, p607-613.

Link to All Datasets

The Code

The Code was written in three phases

1. Data preprocessing phase
2. Training
3. Prediction and plotting

Data Preprocessing Phase

Imports

Numpy import for array processing, python doesn't have built in array support. The feature of working with native arrays can be used in python with the help of numpy library.

Pandas is a library of python used for working with tables, on importing the data, mostly data will be of table format, for ease manipulation of tables pandas library is imported

Matplotlib is a library of python used to plot graphs, for the purpose of visualizing the results we would be plotting the results with the help of matplotlib library.

Tensorflow import since we are going to use tensorflow framework for building model.

```
# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
```

Reading the dataset from data

In this line of code using the read_excel method of pandas library, the dataset has been imported from data folder and stored in dataset variable.

On visualizing the dataset, it contains of two columns X and Y where X is dependent variable and Y is Independent Variable.

Note : On using Grey Kangaroos dataset,the data is normalised, standardised,having a lot of inbuilt variance and outliers the code would result in a gradient exploding problem.

```
# Reading the dataset from data
dataset = pd.read_csv(r'..\\data\\prices.csv')
```

On viewing the dataset, it contains of two columns X and Y where X is dependent variable and Y is Independent Variable.

```
In [6]: dataset.head()
Out[6]:
     X      Y
0  108  392.5
1   19   46.2
2   13   15.7
3  124  422.2
4   40  119.4
```

Creating Dependent and Independent variables

The X Column from the dataset is extracted into an X variable of type numpy, similarly the y variable

X is an independent variable

Y is dependent variable Inference

```
# Creating Dependent and Independent variables
X = dataset['X'].values
y = dataset['Y'].values
```

```
In [10]: type(dataset['X'])
Out[10]: pandas.core.series.Series

In [11]: type(dataset['X'].values)
Out[11]: numpy.ndarray
```

On input 10 it would result in a pandas Series object

So, values attribute is used to attain an numpy array

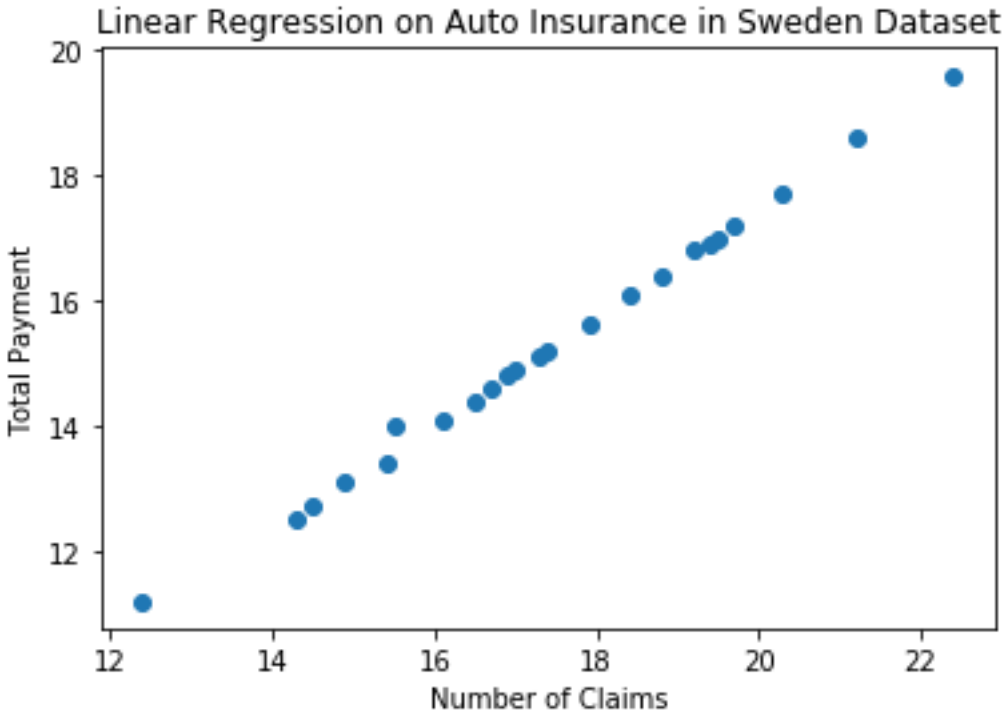
Visualizing the data

The step is to just see how the dataset is

On visualization the data would appear something like this

The X and Y attributes would vary based on dataset.

```
# Visualizing the data
title='Linear Regression on <Dataset>'
x_axis_label = 'X-value < The corresponding attribute of X in dataset >'
y_axis_label = 'y-value < The corresponding attribute of X in dataset >'
plt.scatter(X,y)
plt.title(title)
plt.xlabel(x_axis_label)
plt.ylabel(y_axis_label)
plt.show()
```



Splitting the data into training set and test set

We are splitting the whole dataset into training and test set where training set is used for fitting the line to data and test set is used to check how good the line if for the data.

```
# Splitting the data into training set and test set
```

```
X_train,X_test = np.split(X,indices_or_sections = [int(len(X)*0.8)])
y_train,y_test = np.split(y,indices_or_sections = [int(len(X)*0.8)])
```

Training Phase

Variables for training

- Epochs: stands for how many time the whole data is put through on forward propagation and one backward propagation.
- Learning Rate: is a hyperparameter in backpropagation algorithm to adjust the variables in graph based on loss obtained in forward propagation
-

```
# Variables
epochs = 100
learning_rate = 0.001
```

Feature Columns

These are the features or Independent variables used for training. We are transforming the numpy arrays into tensorflow understandable feature columns specifying the column name as key. This feature column would be fed into tensorflow estimators.

```
# Feature Columns
feature_columns = [tf.feature_column.numeric_column(key="X")]
```

Creating feature dictionaries

These dictionaries are used in creating in the input function to model.train and model.predict

- features_train: used in input function of model.train
- features_test: used in input function of model.predict

```
# Creating feature dictionaries
features_train = {'X':X_train}
features_test  = {'X':X_test}
```

Creating an Input function which would return a batch dataset on every call

The input functions are written for the tensorflow estimator function. The estimator would be expecting a batch dataset of which would return a tuple of features and labels.

The type of processing expected is

```
def train_input_fn(features, labels, batch_size):
    """An input function for training"""
    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))
    # Shuffle, repeat, and batch the examples.
    return dataset.shuffle(1000).repeat().batch(batch_size)
```

It would return a batch tf.data.Dataset Object

Another acceptable format of input function is

```
def input_evaluation_set():
    features = {'SepalLength': np.array([6.4, 5.0]),
                'SepalWidth':  np.array([2.8, 2.3]),
                'PetalLength': np.array([5.6, 3.3]),
                'PetalWidth':  np.array([2.2, 1.0])}
    labels = np.array([2, 1])
    return features, labels
```

It would return a tuple of two elements, first element features dict and second element labels

Other functions which would support input format are numpy_input_fn and pandas_input_fn

For more docs and reference

- Tensorflow Premade Estimator Input Functions
- Estimator Inputs Module

```
# Creating an Input function which would return a batch dataset on every
call
def input_function(features, labels, batch_size):
    data = tf.data.Dataset.from_tensor_slices((dict(features), labels))
    return
```

```
(data.shuffle(10).batch(5).repeat().make_one_shot_iterator().get_next())
```

Making the lambda function of train dataset

Estimator would be expecting lambda function without any arguments

```
# Making the lambda function of train dataset
input_train = lambda: input_function(features_train, y_train,5)
```

Build the Estimator

Tensorflow premade estimator are high level api. These estimators provide a very high level implementation of machine learning models. Here in the code we are using the LinearRegressor class

```
# Build the Estimator.
model = tf.estimator.LinearRegressor(feature_columns=feature_columns)
```

Train the model

Training is the process of tuning the models parameters with the provided input data. model.train would take care of calling the input_train which would feed the model with input data of shuffled batches. The model would be trained for the given number of epochs.

```
# Train the model.
model.train(input_fn = input_train, steps = epochs)
```

Creating an input function for prediction

Similar to train input function predict input function is also create using pre-built `tf.estimator.input` module.

```
# Creating a input function for prediction
predict_input_fn = tf.estimator.inputs.numpy_input_fn(features_test,
shuffle=False)
```

Extracting the y-predicted values into a numpy array

Converting the values in the generator to numpy array for the ease of plotting.

creating a list -> iterating over the generator and appending values -> converting the list to numpy array

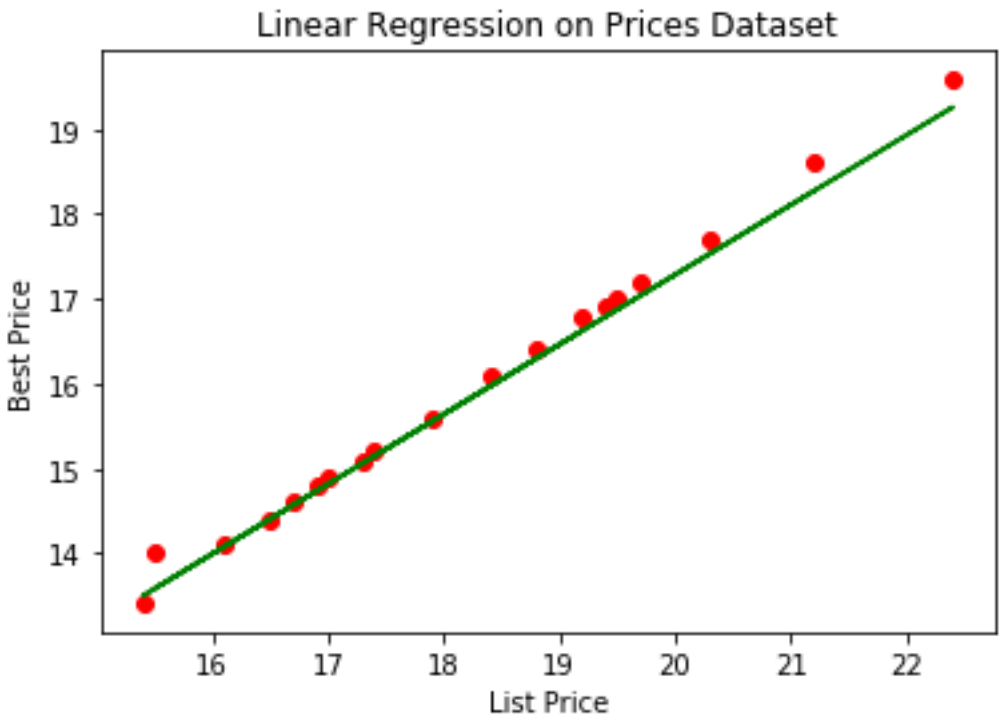
```
# Extracting the y-predicted values into a numpy array
y_predicted = []
for prediction in predict_results:
    y_predicted.append(prediction['predictions'])
y_predicted = np.array(y_predicted)
```

Visualizing the Results

As we have predicted the y-values for a set of x-values we are visualizing the results to check how good did our line fit for our predictions.

The plot shows the red points are the data points are actual values where the blue line is the predictions

```
# Visualizing the Results
plt.scatter(X_test,y_test,c='red')
plt.plot(X_test,y_predicted,c='green')
plt.title(title)
plt.xlabel(x_axis_label)
plt.ylabel(y_axis_label)
plt.show()
```


Linear Regression on Prices Dataset
Best Price
List Price
14
15
16
17
18
19
16
17
18
19
20
21
22