Hyperiondev

Task

Supervised Learning – Linear Regression

Visit our website

Introduction

WELCOME TO THE LINEAR REGRESSION TASK!

In this task we will see a simple machine learning algorithm in action. Through this, we will learn about regression analysis, a statistical process used to estimate the relationship between some variables. This classic method is used by machine learning experts, data scientists, and statisticians.

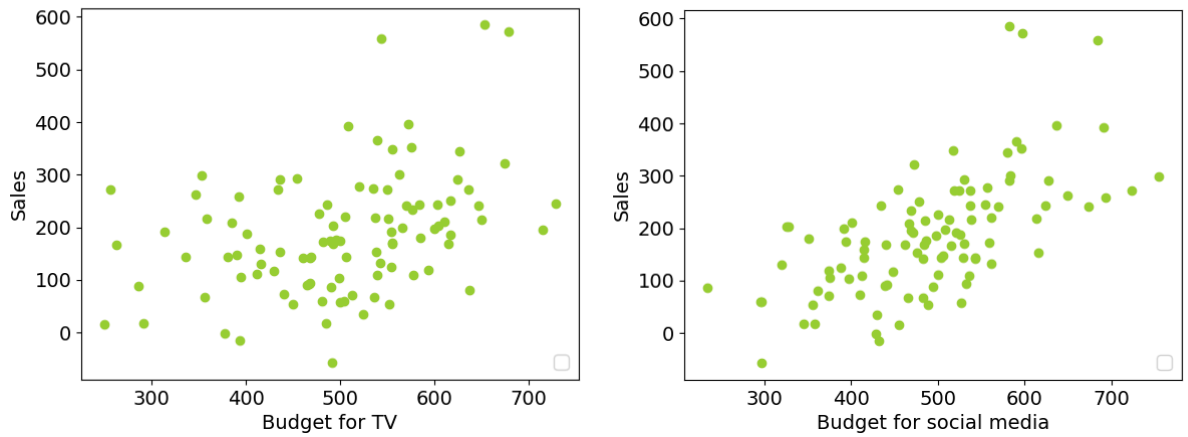
REGRESSION ANALYSIS

Regression analysis is a statistical process used to estimate relationships between variables. First, we will use examples involving a limited number of variables to illustrate the concepts of linear regression. Statisticians typically work with a small number of variables, such as demographic information on a population of citizens. In machine learning settings, the number of variables may be much larger, but the basic principles still apply.

SIMPLE LINEAR REGRESSION

Suppose that we have been asked by a client to give advice on how to most effectively advertise their product. The client offers us some data on which to base our recommendation. They have the sales and advertising budgets of the product in 200 markets, where the advertising budget is split into television and social media adverts.

Common sense suggests that spending more money on advertising will increase sales and that different kinds of advertising do so at various rates. Indeed, as seen in the graphs below, the data highlights that the higher the budget, the higher the sales.



Relationship between sales and a TV or social media advertising budget

However, it's difficult to tell the difference in impact between TV and social media ads. Simple linear regression lets us quantify this difference.

We start by expressing our assumption of a relationship between sales and advertising in the following general mathematical form:

$$\mathbf{Y = f(X)}$$

Here, Y represents sales, and X is the marketing budget, divided into TV and social media budgets (X_1; X_2). The unknown function *f* takes these variables and performs mathematical operations that convert the values for X into the values for Y.

Simple linear regression proposes the following specification of *f*, which approximates the equation of a straight line:

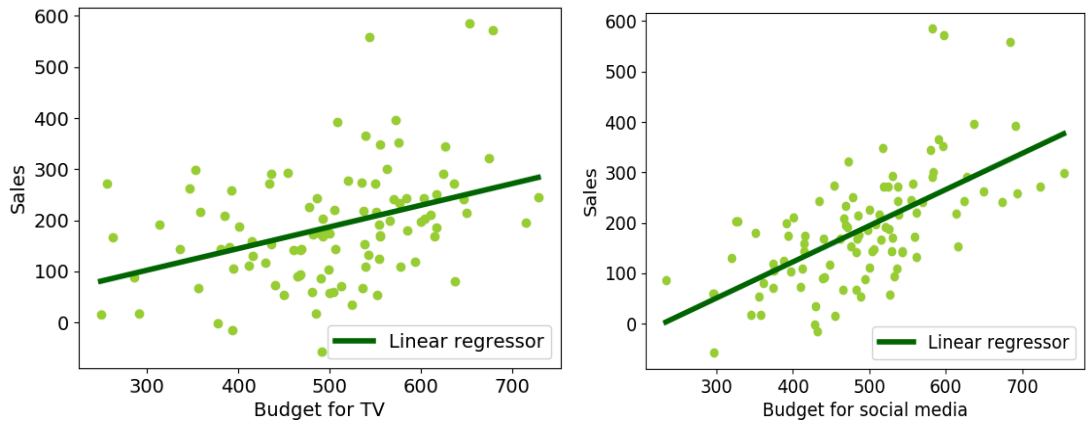
$$\mathbf{Y \approx \beta_0 + \beta_1 X_i}$$

Regardless of which value you specify for β_0 (the intercept) and β_1 (the slope), values produced by this equation will fall along a straight line. The intercept models how many units of the product are sold when there is no money assigned to advertising. The coefficient models the increase in sales per unit increase in the advertising budget.

The purpose of simple linear regression is to find the straight line that "best fits" the data. The best fit here refers to values for β_i that leave a minimal difference between the straight line produced by *f* and the observed data. Formulae exist that determine at what intercept and slope this difference has been minimised.

A minimised difference is *still* a difference: after linear regression, there is still some difference between observed values for Y, and values for Y predicted by *f*. If your data, when plotted, does not seem to fall along a straight line, linear regression is not the right model for the problem. But even if it does, the straight line is only a model of the data, hence the use of the symbol "**≈**".

The regression lines that best fit our data points are shown below:



These lines tell us that, according to the data at our disposal, sales are increased by social media advertising more than by TV advertising.

Note that there are fewer instances in our data at the higher end of the x-axis. The assumption that sales will increase linearly may not hold beyond this point if we were to continue to increase the x-value. It is, however, a reasonable approximation for the range of values we have. Moreover, this tried-and-true algorithm is easy to understand, easy to perform, and can provide valuable information.

Let's take a look at a simple practical example

The **Diabetes dataset** from scikit-learn (sklearn) includes these two features amongst others:

- `s1`: total serum cholesterol
- `s2`: low-density lipoproteins (a type of cholesterol)

By the nature of these features, there is naturally a correlation between them. This makes for a nice bit of practice in using linear regression.

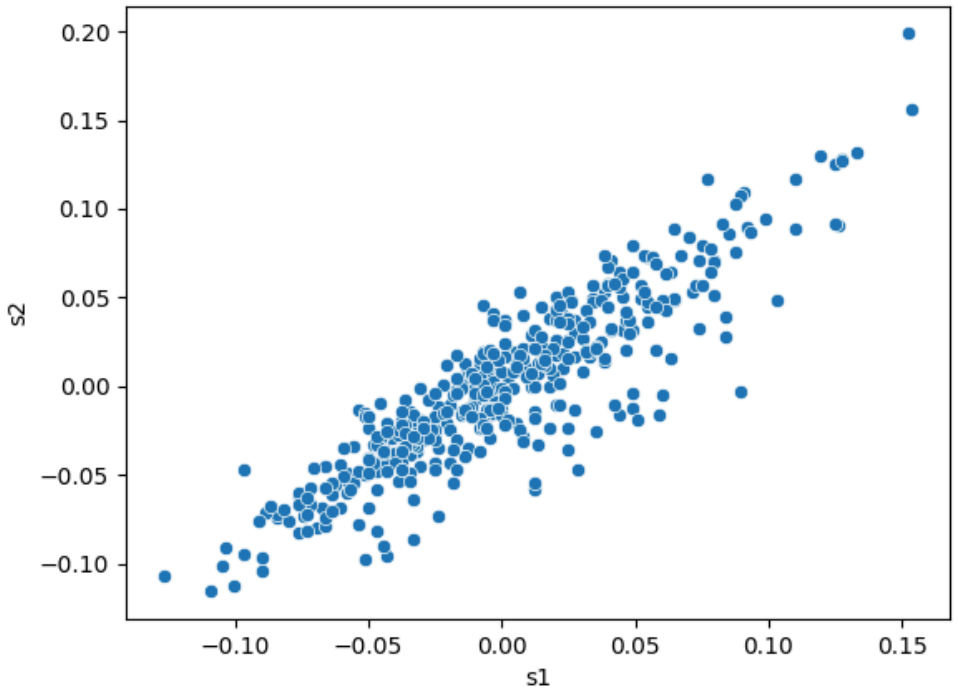
This data can simply be loaded as a pandas DataFrame in the following way:

```
# Import libraries
from sklearn.datasets import load_diabetes
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
df = load_diabetes(as_frame=True).data[['s1', 's2']]

# Plot data
plt.figure()
sns.scatterplot(data=df, x='s1', y='s2')
plt.show()
plt.close()
```

The code above gives the following scatterplot of the data:



To create our model, we import `LinearRegression` from the `linear_model` module in sklearn:

```
from sklearn.linear_model import LinearRegression
```

To allow compatibility, we must ensure that our model inputs are in the shape `(n_samples, n_features)`.

Because there is only one feature, it will be `(n_samples, 1)`:

```
# Reshape the “s1” column into a 2D array for model input
X = df['s1'].values.reshape(-1, 1)

# Extract the “s2” column as the target variable
y = df['s2'].values
```

Now that we have our data in the desired shape, we can fit the data and make a prediction:

```
# Fit the linear regression model to the data
simple_model = LinearRegression()
simple_model.fit(X, y)

# Predict target values based on input data
y_pred = simple_model.predict(X)
```

Let’s take a look at our model performance using a plot:

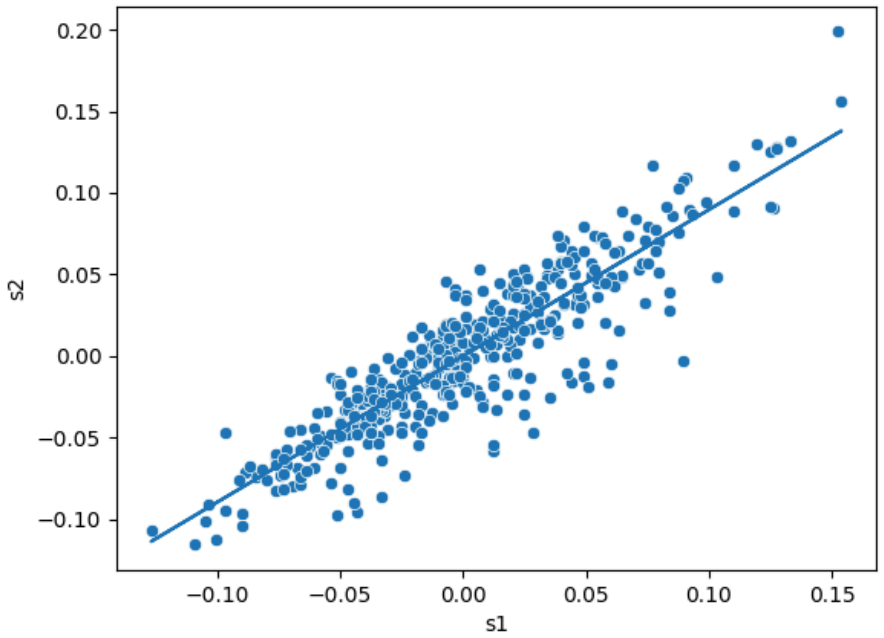
```
# Create a new figure for plotting
plt.figure()

# Plot the scatter plot of the original data
sns.scatterplot(data=df, x='s1', y='s2')

# Plot the linear regression line based on the predictions
plt.plot(X, y_pred)

# Display and close the plot
plt.show()
plt.close()
```

The generated plot looks like this:



You can see a clear line of best fit going through the data. This was done with just a few lines of code!

MULTIPLE LINEAR REGRESSION

We have explored the relationship between one explanatory variable and the response or dependent variable. In many cases, you will be interested in more than a single input variable. In the example case, there were two independent variables: TV and social media. Multiple linear regression allows us to model the impact of both variables on the outcome variable at the same time.

Modelling multiple variables is quite useful. A simple linear regression approach may over- or underestimate the impact of a variable on the outcome because it does not have any information about the impact of other variables. Imagine, for example, that our social media and billboard budgets increased simultaneously and sales went up shortly thereafter. Our simple regression models would attribute the entire increase in sales to the single variable they were modelling, which would be incorrect. A multiple linear regression model could make a less biased prediction of how much each type of advertisement contributes to sales.

The extension of simple linear regression is fairly straightforward. Instead of a function for Y with only one coefficient, the function has coefficients for each variable. In the case of two variables, the formula takes the form:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Note that the coefficients (β_i) will not just be the same values as independent simple linear regression models would return. This extended model will adjust each value according to the relative contribution of each variable.

Both simple and multiple linear regression can be performed very simply using sklearn. Provided that your dataset is properly preprocessed, sklearn infers on its own whether your input has one or multiple features. Fitting looks like this:

```
# Fit a multiple regression model

multiple_model = LinearRegression()
multiple_model.fit(X, y)
```

Let's apply this to some data on the impact of TV, radio, and newspaper advertising on sales. This dataset, **Advertising.csv**, is also in your lesson folder if you want to try it for yourself.

Multiple linear regression applied to this dataset returns the coefficients `0.045`, `0.189`, and -`0.001`.

To see what these coefficients say about the variables, it helps to see them in the context of the formula for Y:

sales = 2.94 + 0.045(TV) + 0.189(radio) - 0.001(newspaper)

This formula shows that TV and radio advertising positively impacts sales, but newspaper advertising has a negligible or even slightly negative effect. Negative coefficients mean that the variables have an opposite relationship: as one increases, the other one decreases. When comparing the coefficients for TV and radio, we see that the coefficient for radio is larger than that for TV. This means that radio advertising contributes more to total sales than TV advertising does.

Based on these findings, we may recommend a focus on radio advertising. We might also recommend that the newspaper advertising budget be scrapped.

Extra resource

Take note of the **assumptions** that need to be met for linear regression to ensure the accuracy and dependability of its results.

TRAINING AND TEST DATA IN MACHINE LEARNING

When you are teaching a student arithmetic summation, you want to start by teaching them the pattern of summation, and then test whether they can apply that pattern. You will show them that 1 + 1 = 2, 4 + 5 = 9, 2 + 6 = 8, and so forth. If the student memorises all the examples, they could answer the question "what is 2 + 6?" without understanding summation. To test whether they have grasped the underlying principles, you need to test them on numbers that you have not exposed them to directly. For example, you might ask them "what is 4 + 9?"

This intuition forms the motivation behind training and testing data in machine learning. We create a model (e.g., regression) based on some data that we have, but we do not use all of our data. Instead, we hide some data samples from the model and then test our model on the hidden data samples to confirm that the model is making valid predictions, as opposed to reiterating what was given to it in the training dataset.

A **training set** is the actual dataset that we use to train the model. The model sees and learns from this data. A **test set** is a set of withheld examples used only to assess the performance of a model. Although the best ratio depends on how much data is available, a common split is a ratio of 80:20.

For example, 8000 training examples and 2000 test cases. Sklearn allows us to do this quite easily:

```
# Import the train_test_split function from scikit-learn to split data
into training and test sets
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
# X_train and y_train are the features and targets for training
# X_test and y_test are the features and targets for testing
# test_size = 0.25 indicates that 25% of the data will be used for
testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.25)
```

When splitting your data into training and test sets, it's crucial to avoid systemic differences. These can occur when there's a pattern or bias in the way data is divided. For example, if data is geographically distributed, splitting without considering location can lead to a model that performs well in one region but poorly in another. In other words, the model's performance suffers because it's being evaluated on a task that diverges from its training data. To prevent this, examine the label distribution in your data and ensure it's balanced across your training and test data.

Examine the example output below. What is the problem with the samples in the test set?

```
# Data sample
X = [1,2,3,4,2,6,7,8,6,7]
y = [0,0,0,0,0,1,1,1,1,1]

# Split data
X_train, X_test, y_train,y_test = train_test_split(X, y, test_size = 0.2,
shuffle=False)

# Compare train and test sets for y
print("y_train {}".format(y_train))
print("y_test {}".format(y_test))
```

```
Output:
>> y_train [0, 0, 0, 0, 0, 1, 1, 1]
```

```
>> y_test [1, 1]
```

Notice there are no **0** labels in the test set (**y_test**). This can largely be solved by using **shuffle=True** as a parameter in the **train_test_split** function. This means that the labels are distributed randomly between the training and test sets, so it's less likely that the training and testing data would be systematically different. Notice there is now a **0** label in the test set.

```
Output:
>>y_train [1, 0, 0, 1, 0, 1, 0, 1]
>>y_test [0, 1]
```

However, there is still the possibility that most samples of one type end up in the training set, without a representative proportion in the test set. This could also result in lower performance than expected. Consider the case where we set **test_set=0.4**, and we get the following:

```
Output:
>>y_train [0, 1, 1, 1, 0, 1]
>>y_test [0, 0, 0, 1]
```

Notice the labels are not distributed proportionally between **y_test** and **y_train**. Another way to ensure that data is represented equally in both the training and testing data is to make use of the stratify parameter. This ensures that labels are represented in as close to the same proportions as possible in both the training and testing data.

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.5,
shuffle=True, stratify=y)
```

```
Output:
>>y_train [0, 0, 1, 1, 1, 0]
>>y_test [1, 0, 0, 1]
```

Note that the labels are now distributed in the same proportion across the training and test sets.

Take note:

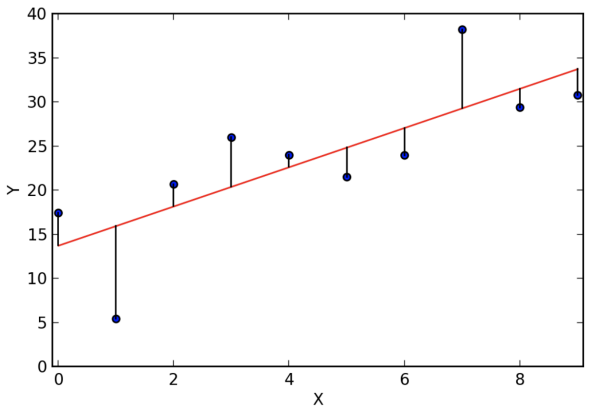
You need to be using sklearn version 0.17 or higher to make use of the `stratify` parameter.

TRAINING AND TEST ERROR

We've discussed before how a regression model is only an approximation of the data. The model predicts values that are close to the observed training values in hopes of making good predictions on unseen data. The difference between the actual values and the predictions is called the "error".

The training dataset error and the test dataset error can be obtained by comparing the predictions to the known, true labels (the gold standard). The test error value is more important as it's the more reliable assessment of the value of the model. After all, we want to use our model on unknown data points, as opposed to applying it to cases for which we already have the actual outcome. In most cases, the training error value will also be lower than the test error value.

There are many different ways to measure the error. As mentioned, the error is the difference between observed and predicted values, as in this plot:



Here we see the observed data (Y) in dark blue and the prediction of a regression model (**y_pred**) along the red line. The error is depicted by vertical black lines. Recall there are a number of different ways one can aggregate the error to get a final score for the model. Two common ones are the root mean squared error (RMSE) and R squared (R^2).

Extra resource

Learn more about the **R-squared metric** to learn how to interpret this metric when using it to evaluate regression models.

Instructions

Before attempting the following tasks, first read the **additional resources** in your task folder to better understand regression analysis and see an example of linear regression in Python.

Practical task 1

Follow these steps:

1. Create a Jupyter notebook called **insurance_regression.ipynb**.
2. Import **insurance.csv** into your notebook (**source**).
3. Use the data in the relevant columns to determine how age affects insurance costs:
    - Plot a scatter plot with age on the x-axis and charges on the y-axis.
    - Using `linear_model.LinearRegression()` from sklearn, fit a model to your data, and make predictions on the data.
    - Plot another scatter plot with the best-fit line.

Practical task 2

Follow these steps:

1. Create a Jupyter notebook called **diabetes_regression.ipynb**.
2. Read **diabetes_dirty.csv** into your Jupyter notebook (**source**).
    - The **diabetes_dirty.csv** dataset contains data that can be used to model a person's diabetes progression based on various attributes.
    - You may explore the **dataset metadata** to gain insight into the variables. In the sex variable, 1 indicates female and 2 indicates male.
3. Differentiate between the independent variables and the dependent variable, and assign them to variables X and Y.
4. Generate training and test sets comprising 80% and 20% of the data respectively.
5. Investigate the necessity for scaling or normalisation of the data. Employ `MinMaxScaler` and `StandardScaler` if necessary. Fit these scalers on the training set and apply the fitted scalers to transform both the training and test sets accordingly.
6. Generate a multiple linear regression model using the **training set**. Use all of the independent variables.
7. Print out the intercept and coefficients of the trained model.
8. Generate predictions for the **test set**.
9. Compute R squared for your model on the **test set**. You can use `r2_score` from `sklearn.metrics` to obtain this score.
10. Ensure your Notebook includes comments about what your code is accomplishing and notes about model outputs such as R squared.

Be sure to place files for submission inside your **task folder** and click "**Request review**" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.