TASK

Machine Learning III

Visit our website

Introduction

WELCOME TO THE THIRD MACHINE LEARNING TASK!

By this point, you should have a clear understanding of machine learning. In this task, we will delve one step further by making use of the machine learning algorithms included in scikit-learn. More specifically, we will be doing some regression analysis, which is a statistical process used to estimate the relationship between variables.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

REMINDER ON LOADING THE DATASET

There are two ways in which you can load a dataset from sklearn. If you are using a slow PC or laptop this might take a few seconds.

1. The first is to use the **`sklearn`** module to import all datasets and then use the dot (**`.`**) notation to specify which dataset to load.

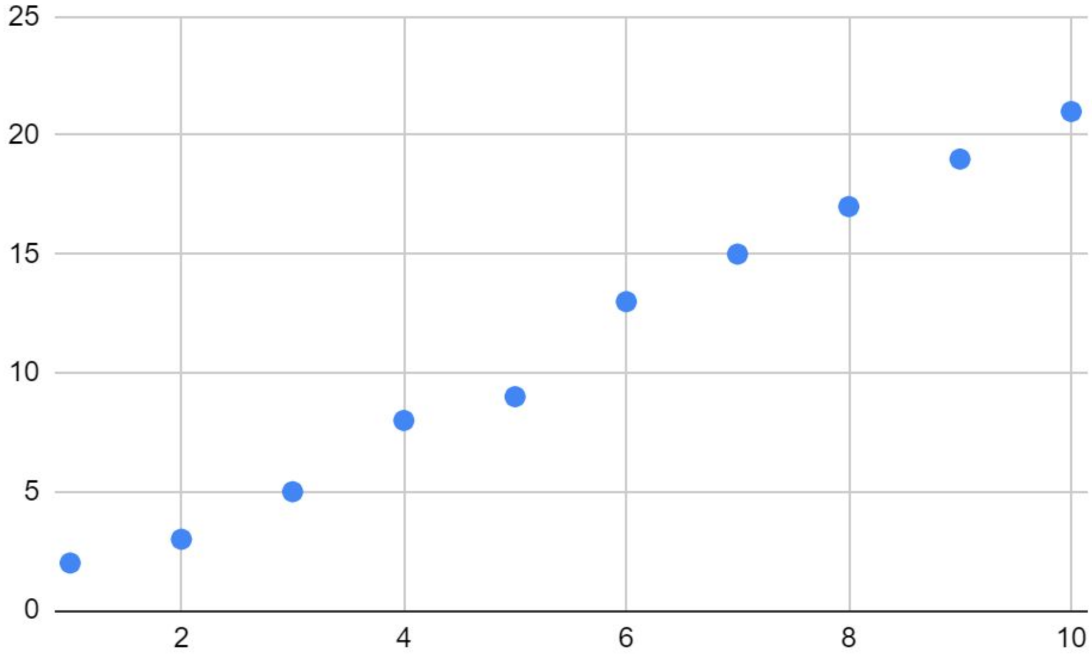
```
from sklearn import datasets
iris = datasets.load_iris()
```

2. Another way is to use **`sklearn.datasets`** module to import **`load_iris`**. Using this approach will omit the use of the dot (**`.`**) notation and just invoke the **`load_iris()`** function directly.

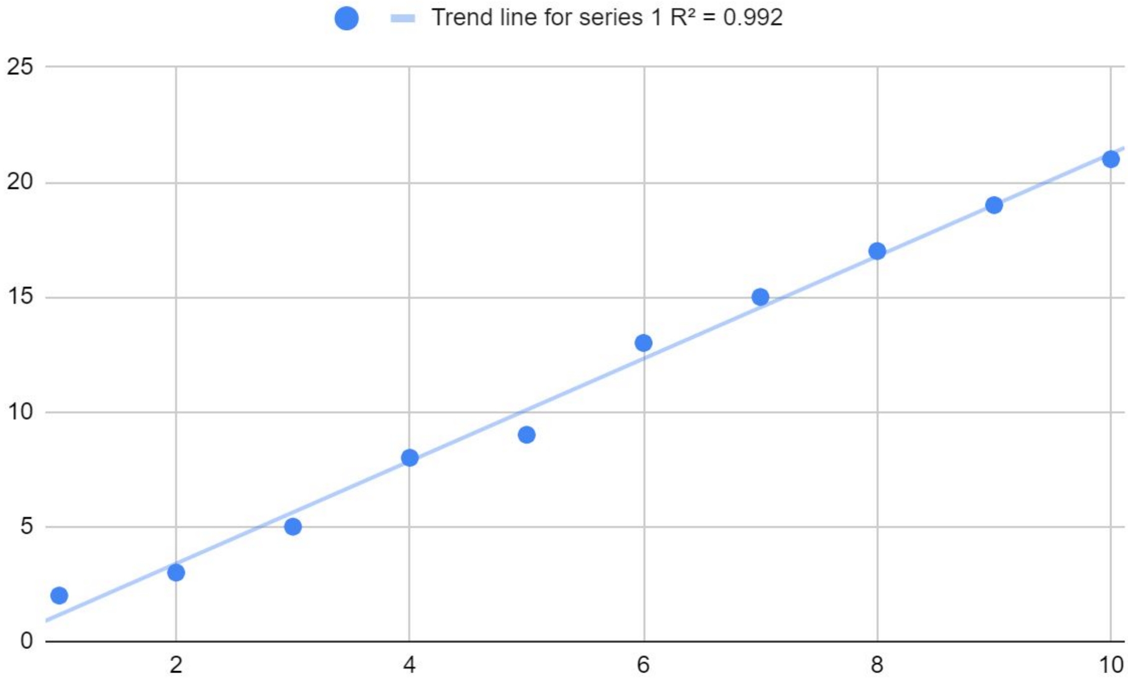
```
from sklearn.datasets import load_iris
iris = load_iris()
```

LINEAR REGRESSION

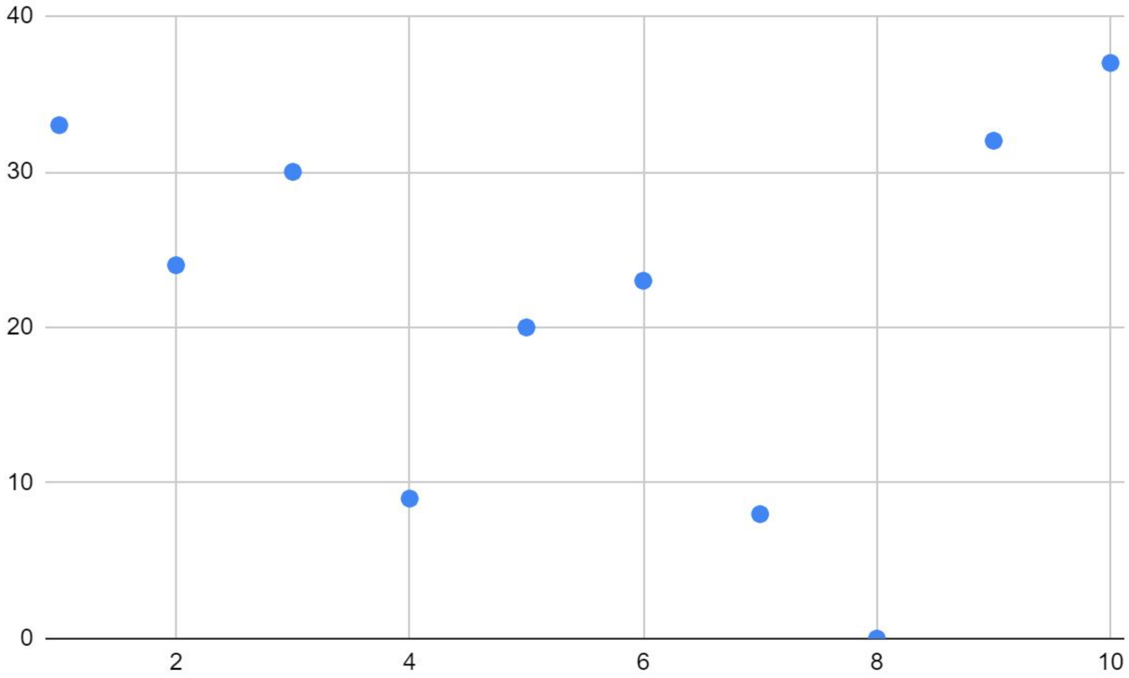
The point of linear regression is to find 'the line of best fit'. That is to say, if we have scattered points plotted on a graph, we want to draw a line through them so that as many of the dots as possible are as close to the line as possible. Let's look at an example:



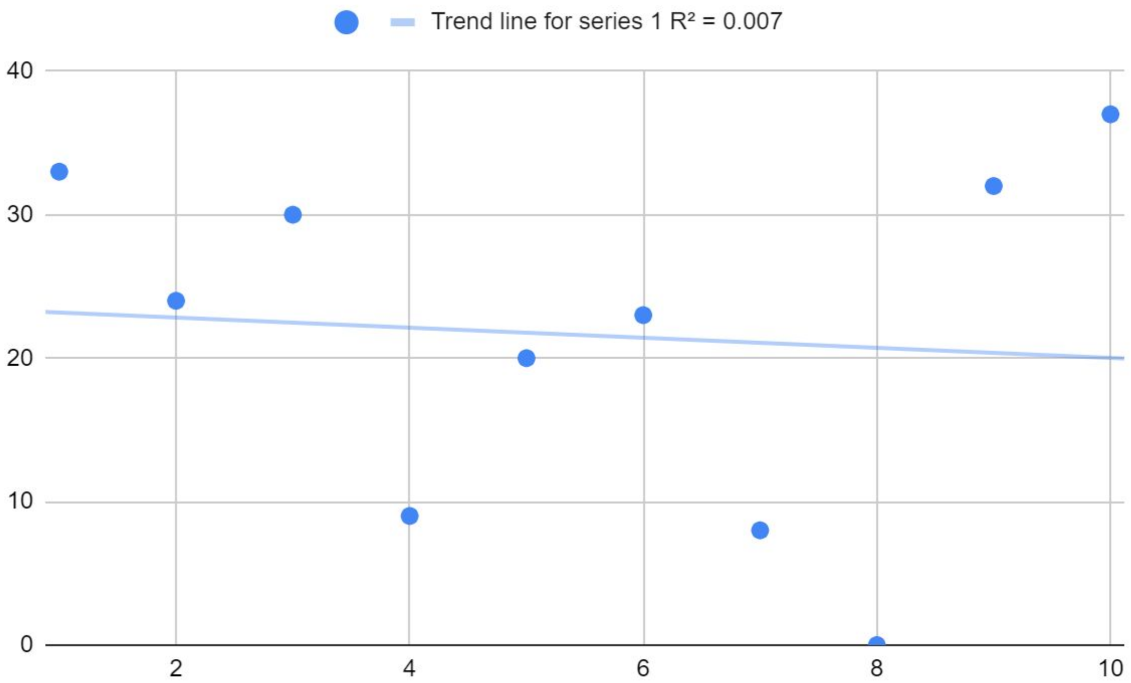
In the graph above, we can see the points scattered on the graph. The points are fairly close together, so it's easy to estimate where the line of best fit will go. Have a look at the linear regression line below.



As you can see, most of the points are close to/touching the line. That shows that it is the line of best fit and that there is a clear pattern in the data that follows this line, i.e. there is a strong correlation between the X and y variables. With this data, we can predict fairly easily where future data points will lie based on the line of best fit. Now let's have a look at another example:



In this example you can see that it's a little bit harder to determine where the line of best fit would be based on the points. This means that there is a weak correlation between the X and y variables, if there is a correlation at all. This means it will be difficult to predict where future data points will lie.



We fit the line using **least-squares** and the correlation is measured using the R-squared (R^2) value you can see on the top of the graph. The closer to 1 the number, the more correlated the X and y values are.

A linear model is defined by $y = X\beta + \varepsilon$, where y is the target variable, X is the data, β represents the coefficients and ε is the observation noise.

Let's try and predict something using linear regression. The diabetes dataset that comes with `sklearn` consists of 10 physiological variables (age, sex, weight and blood pressure). These were measured on 442 patients, and are an indication of the disease progression after one year. The goal is to predict disease progression from physiological variables.

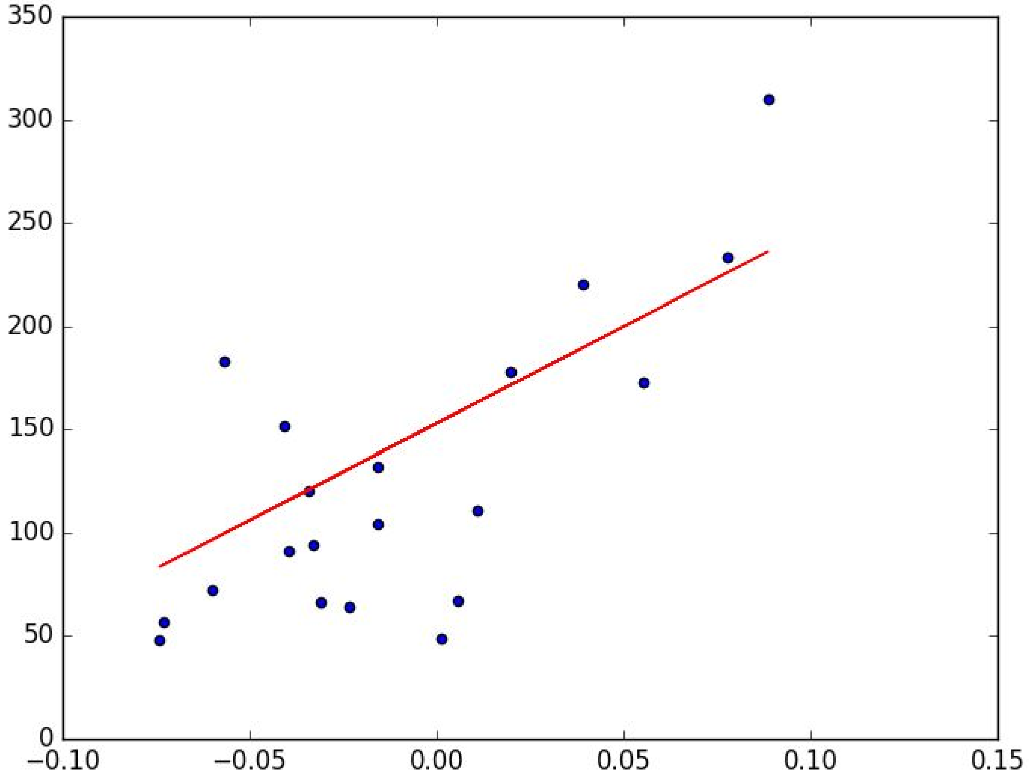
```
import matplotlib.pyplot as plt

import numpy as np
from sklearn.datasets import load_diabetes
from sklearn import linear_model

d = load_diabetes()
d_X = d.data[:, np.newaxis, 2]
dx_train = d_X[:-20]
dy_train = d.target[:-20]
dx_test = d_X[-20:]
dy_test = d.target[-20:]
```

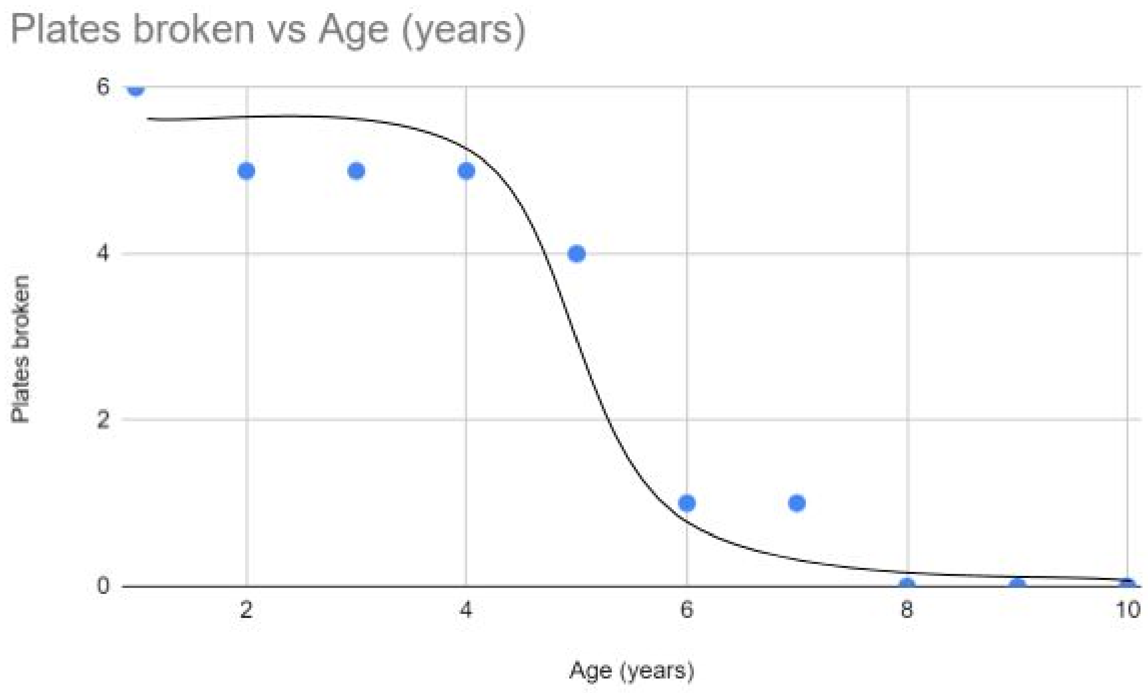
```
13
14  lr = linear_model.LinearRegression()
15  lr.fit(dx_train, dy_train)
16
17  mse = np.mean((lr.predict(dx_test) - dy_test) **2)
18  lr_score = lr.score(dx_test, dy_test)
19
20  print(lr.coef_)
21  print(mse)
22  print(lr_score)
23
24  plt.scatter(dx_test, dy_test)
25  plt.plot(dx_test, lr.predict(dx_test), c='r')
26  plt.show()
```

We start with importing our relevant modules and datasets. The `load_diabetes` dataset comes from `sklearn` so in **line 3** we import `load_diabetes`. Then, in **line 4,** `linear_model` is imported so that we can calculate linear regression. Once the dataset is loaded, select which features to use (which of the physiological variables). In this example only one feature is selected on **line 7**. The data is then split up into training and testing data. The last 20 samples are being used as testing data in this example. You can play around with how much data to use for testing if you want. The next step is to create a linear regression object and then train it with the training data by using the `fit()` function. The mean square error, which is related to the strength of correlation, can be easily calculated by finding the error for each sample in the test set and taking the mean of these values using the `numpy_mean()` function. To obtain the variance score of the model, the `score()` function is used; passing the test data as parameters. Lastly, the scatter plot of the test data and the line plot are then graphed and presented to the screen. The above example will produce the following figure:



LOGISTIC REGRESSION

Logistic regression is similar to linear regression, except we are *classifying* whether something is **True** or **False**. That way, the y-axis simply measures from 0 to 1 and the line of best fit is an ‘s’ shaped curve. This curve is determined using **maximum likelihood**. Have a look at the graph below:



In this graph, you can see if a child is less likely to break a plate as they get older. The line shows the probability of this: a younger child has a much higher probability of breaking a plate than an older child. We can also predict that a 5-year-old has about a 50% chance of breaking a plate. Note that we talk about the response variable (breaking plates) as binary: breaking a plate, or not.

Logistic regression is applicable if:

- The aim is to model the probabilities of a response variable as a function of some explanatory variables.
- The aim is to perform descriptive discriminative analysis such as describing the difference between individuals in separate groups as a function of explanatory variables.
- The aim is to predict probabilities that individuals fall into two categories of the binary response as a function of some explanatory variables.
- The aim is to classify individuals into two categories based on explanatory variables.

Compulsory Task 1

Follow these steps:

- Create a Python file called **theory.py**.
- In this task you will be required to identify whether linear or logistic regression is more suited to each of the example applications described below. In addition, identify the explanatory variable(s) in each case and state whether they are continuous or discrete.

Case 1: A real estate agent is in charge of assigning prices to empty plots in new development. The estate agent has some examples from a very similar development nearby and identifies that the cost of a property should be dependent on the size of the plot of land the property is on. They wish to create a model to represent this to predict the price of the new plots.

Case 2: A programmer is developing a spam detector. They notice that spam emails often have the words "prize" or "win" in them, as well as the presence of spelling errors. They wish to create a model to predict whether or not an email is indeed spam or not.

Case 3: A researcher is investigating how long after its expiry date a particular brand of milk tends to actually go off. For their research they assume that this transition is sudden. They test samples of some of the milk at regular time intervals to determine the status of the milk as sour/not sour and use this data to develop a model.

Compulsory Task 2

Follow these steps:

- Create a Python file called **linearRegression.py**.

- In this task you will use the diabetes dataset mentioned above to perform linear regression to find the best fit line through the data.

- Reserve the last 20 observations for testing and use the rest for training your model.

- Instead of using `linear_model.LinearRegression()` from `sklearn`, write a function and make use of numpy to calculate the gradient and the y-intercept of the best fit line, which has equation $y = mx + b$. The equations below describe how both the gradient and the y-intercept can be calculated from the training data and labels. Note: when you calculate the gradient, you will need to reshape the x array to remove an extra dimension of 1 from its shape (it has this as the dataset was formatted for use with the `sklearn` functions, which require this extra dimension). You can easily do this by applying `.squeeze()` to the x array when you pass it as an argument to the method. Hint: if the line doesn't look like it fits the data well, there is a bug in your code.
  - $m = (\mu(x) * \mu(y) - \mu(x * y))/((\mu(x))^2 - \mu(x^2))$
  - $b = \mu(y) - m * \mu(x)$

    $Where\ \mu\ is\ a\ mean\ function$

- Use these values to produce a figure with the following:
  - Scatter plot of training data colored red.
  - Scatter plot of testing data colored green.
  - Line graph for the best-fit line colored blue.
  - Legend.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we’ve done a good job?

Click here to share your thoughts anonymously.