TASK

# Simple Linear Regression and Machine Learning

Visit our website

## Introduction

## WELCOME TO THE SIMPLE LINEAR REGRESSION AND MACHINE LEARNING TASK!

In this task, we will show a simple machine learning algorithm in action; and 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, as well as data scientists and statisticians. This task will also introduce you to the most important concepts in machine learning, giving you a general overview of the landscape and preparing you to learn about the intention and theory behind specific supervised and unsupervised learning algorithms.

## 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.

**Extra resource**

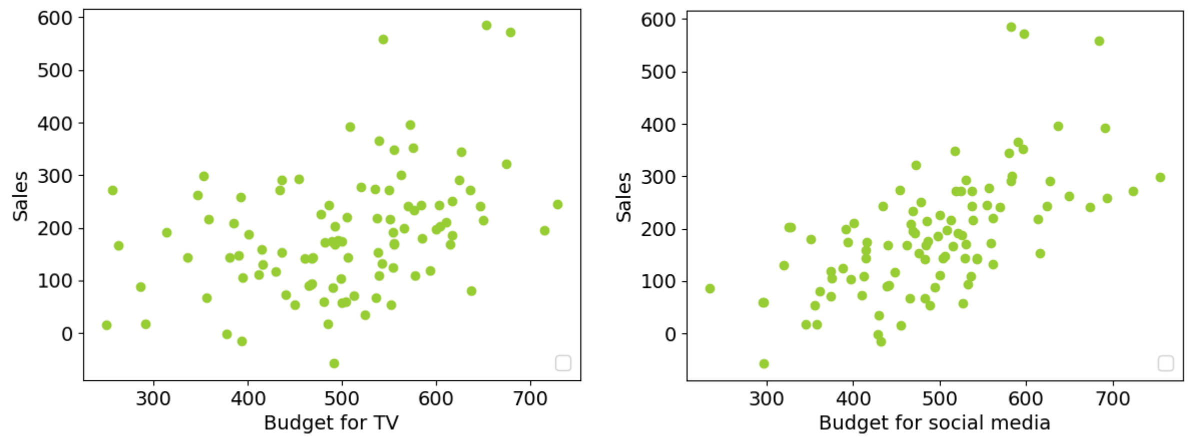
Please consult the highly recommended additional readings on **Discrete maths** and **Calculus** to develop a strong understanding of the mathematical concepts underlying linear regression.

These subjects provide important foundational knowledge, including linear algebra and derivatives, which will enhance your overall understanding of machine learning.

## SIMPLE LINEAR REGRESSION

Suppose that we have been asked by a client to give advice on how to advertise their product most effectively. 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 ads and ads on social media.

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 show that the higher the budget, the higher the sales.



However, it is hard to tell what the difference in impact between TV and social media ads is. 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 ($\mathbf{X_1}$, $\mathbf{X_2}$). 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:

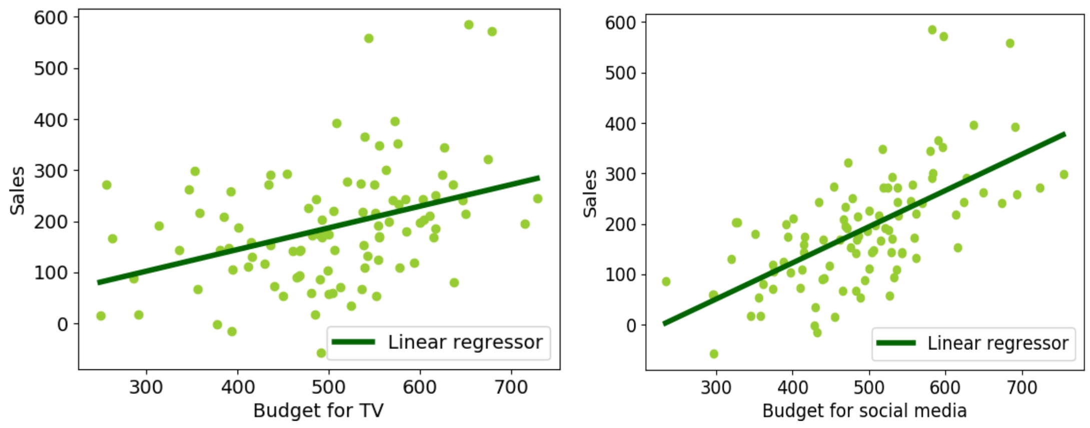
$$\mathbf{Y \approx \beta_0 + \beta_1 X_i}$$

Regardless of which value you specify for $\beta_0$ (the intercept) and $\beta_1$ (the slope), values produced by this equation will fall along a straight line. The intercept shows how many units of the product are sold when there is no money assigned to advertising. The coefficient shows how much the sales increase for each single-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 $\boldsymbol{\beta}_i$ that leave a minimal difference between the straight line produced by **f** and the observed data. There exist formulae 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 "**≈**".

Lines fitted to our toy data look as follows:



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 very simple example

The diabetes data set from scikit-learn (`sklearn`). This data set includes two different features:

- `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.

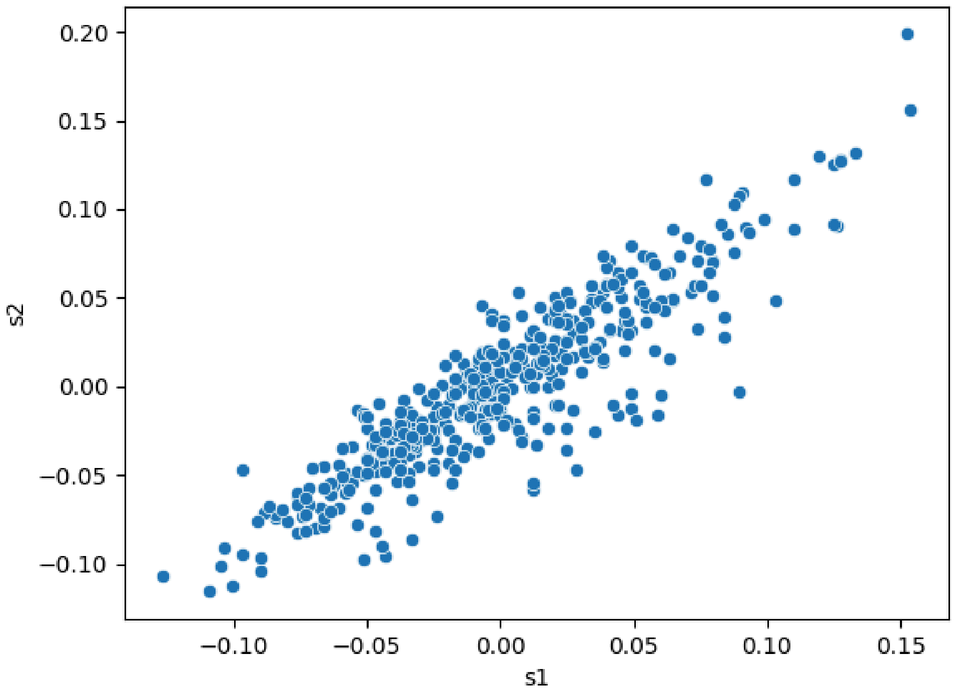
This can simply be loaded as a Pandas DataFrame.

```python
from sklearn.datasets import load_diabetes
import seaborn as sns
import matplotlib.pyplot as plt

# Load data set
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()
```

This code above gives the following scatterplot of the data:



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

```python
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)`:

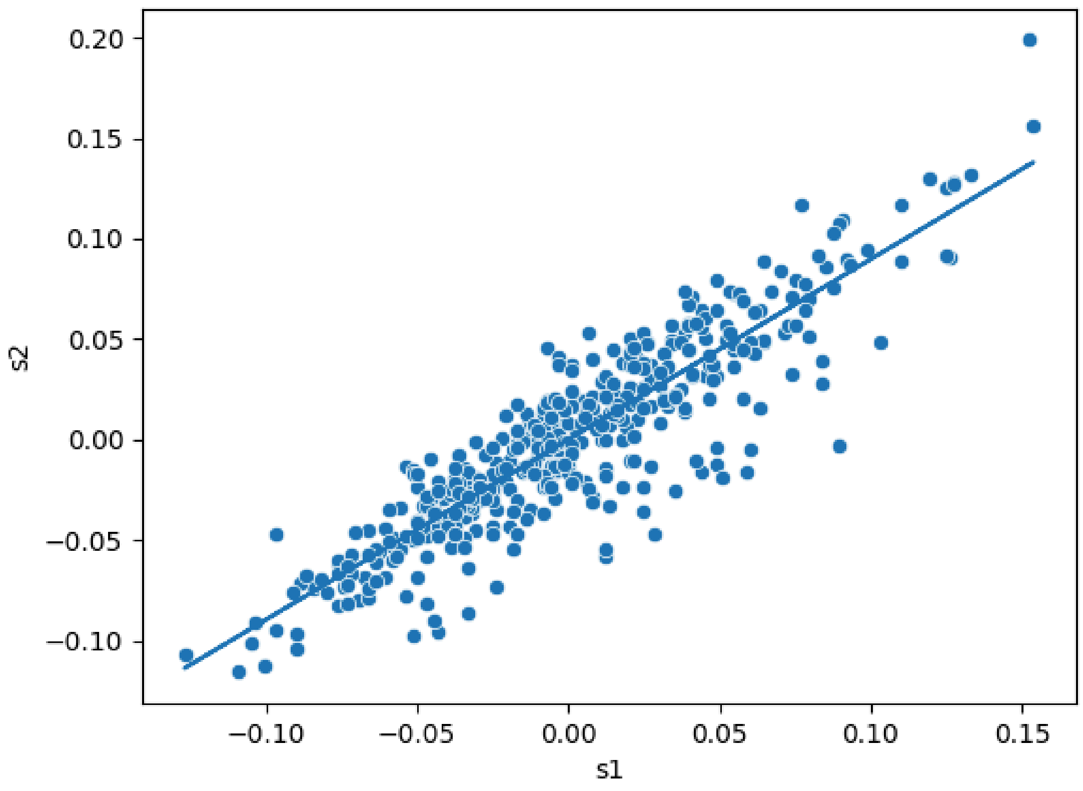
```
X = df['s1'].values.reshape(-1, 1) # create a 2D array
y = df['s2'].values
```

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

```
# Fit and predict
simple_model = LinearRegression()
simple_model.fit(X, y)
y_pred = simple_model.predict(X)
```

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

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



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

A note from the

**HyperionDev Team**

The difference between machine learning and other statistical and mathematical approaches, such as data mining, is another popular subject of debate. Put simply, while machine learning uses many of the same algorithms and techniques as data mining, one difference lies in what the two disciplines predict.

Data mining discovered previously unknown patterns of knowledge, whereas machine learning is used to reproduce known patterns and knowledge, automatically apply that to other data, and then automatically apply those results to decision-making and actions.

Data mining efforts have become extremely prevalent among gamers. Data miners continuously inspect logic in new version releases of games and/or apps to reveal potential future features to other players. This has allowed players to strategise and structure their game so that they can potentially benefit from new features when they arrive.

## 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 overestimate 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 budget and billboards budget were increased simultaneously and sales increased 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:

$$\mathbf{Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2}$$

Note that the coefficients ($\boldsymbol{\beta_i}$) 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.

**Take note:**

Over time, several myths have emerged surrounding machine learning, such as "machine learning is just summarising data" or "learning algorithms just discover correlations between events". Probably one of the most prominent is that "machine learning can't predict unseen events". This line of thought goes thus: if something has never happened before, its predicted probability must be zero. Right?

On the contrary, machine learning is the art of predicting rare events with high accuracy. If A is one of the causes of B, and B is one of the causes of C, A can lead to C, even if we've never seen it happen before. A practical example of the predictive powers of machine learning is the spam filter in your email inbox. Every day, spam filters correctly flag freshly concocted spam emails, based on emails you marked as spam earlier and other predictor data.

Both simple and multiple linear regression can be performed very simply using **sklearn**. Provided that your data set is properly pre-processed, **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 data set, **Advertising.csv**, is also in your lesson folder if you want to try it for yourself.

Multiple linear regression applied to this data set 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**:

$$\textbf{sales} = 2.94 + 0.045(\text{TV}) + 0.189(\text{radio}) - 0.001(\text{newspaper})$$

One thing this shows is that TV and radio advertising has a positive impact on sales, but newspaper advertising has an impact that is close to zero, and even a bit below it. 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 or diminished.

## 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 recovered not just the facts but the pattern behind the facts, 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: 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 data set.

A **training set** is the actual data set 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, 8,000 training examples and 2,000 test cases. **Sklearn** allows us to do this quite easily:

```python
# Import the train_test_split functionality from sklearn
from sklearn.model_selection import train_test_split

# Pass your x and y variables in the function and get the train and test
# sets for the X and y variables (4 variables).
# The test_size parameter determines how the data is split, 0.25 means 25%
# of the data will be in the test set.

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

One thing to watch out for when dividing the data is that the test and training set should not be systematically different. If the total data set is ordered in some way, for example by alphabet or by the time of collection, the test set might contain different kinds of instances from the training set. If that occurs, the model's performance will be poor, not because it didn't learn from the data effectively, but because it is being tested on a task that differs from the one it was trained for. For this purpose, it is customary to investigate the distribution of labels in your data and make sure they are similar across your training and test data. In the example below, notice that there are no `0` 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 train_test_split. This means that the labels are distributed randomly between the training and test sets, so it is less likely that the training and testing data would be systematically different.

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

Notice there is now a `0` label in the test set. 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:**

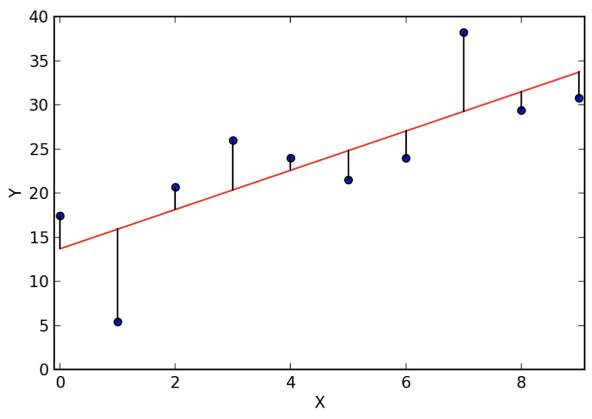
You need to be using `sklearn` version 0.17 or higher to make use of the stratify parameter.

## 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 data set error and the test data set 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 is 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 in 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 have observed data (**Y**) in dark blue and 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**

Read more about the **R-squared metric** to learn how to interpret this metric and the limitations you should be aware of when using it to evaluate regression models.

A note from the

**HyperionDev Team**

**The history of machine learning**

For decades, visions of machines that can learn the same way humans can have captured the imagination of science-fiction authors and researchers alike. But only in recent years have machine learning programs been developed that can be applied on a wide scale, influencing our daily lives.

Machine learning programs are working behind the scenes to produce and curate our playlists, news feeds, weather reports, and email inboxes. They help us find restaurants, translate documents, and even meet potential dates. From a business perspective, machine learning-based software is becoming central to many industries, generating demand for experts.

In 1959, Arthur Samuel, a pioneer in artificial intelligence and gaming, defined machine learning as the ‘field of study that gives computers the ability to learn without being explicitly programmed’. He is known for developing a program capable of playing checkers. Samuel never programmed exactly which strategies the systems could use. Instead, he devised a way in which the program could learn such strategies through the experience of playing thousands of games.

In the ’50s, machines were hard to acquire and not very powerful, so machine learning algorithms were mostly an object of theoretical research. Now that computers are vastly more powerful and more affordable, machine learning has become a very active field of study with a variety of real-world applications.

## DEFINING MACHINE LEARNING

At its essence, machine learning can be defined as a computational methodology focused on deriving insights from data. It enables computers to acquire knowledge from past observations and independently make predictions or decisions without relying on explicit programming instructions. By leveraging data-driven patterns and algorithms, machine learning enables automated systems to adapt and improve their performance over time.

### Unveiling the line between machine learning and artificial intelligence

The term “machine learning” is often used interchangeably with the term “artificial intelligence”. While the two are very much related, they are not the same thing. There is much debate about the difference between the two, but a simple way to look at it for our purposes is to see machine learning as a *type* of artificial intelligence. Any program that completes a task in a way that can be considered human-like can be considered an example of artificial intelligence, but only programs that solve the task by learning without pre-programming are machine learning programs.

## INPUT AND OUTPUT

Whatever it is that we want a machine learning algorithm to learn, we first need to express it numerically. The machine-readable version of a task consists of an **input** and an **output**. The input is whatever we want the algorithm to learn from, and the output is the outcome we want the algorithm to be able to produce. An example of an input would be the budget or number of awards a movie receives. An example of output would be the box office sales of that movie.

Since machine learning is a young field that overlaps with several other disciplines, including statistics, the input and output may be referred to by several other names.

For input, these include:

- features (named after the fact that inputs typically “describe” something),
- independent variables, and
- explanatory variables (because the output is usually assumed to depend on or be explained by the input).

For output, alternate terms are:

- labels,
- predictions,
- dependent variables, and
- response variables.

Once we clearly understand the input-output specifics of machine learning models, we can explore different learning algorithms. While there are various types of learning methodologies, our main focus, in this context, will be on supervised and unsupervised learning. These two approaches are fundamental in the field of machine learning, as they involve training models using labelled or unlabelled data, respectively.

## SUPERVISED LEARNING

In supervised learning problems, a program predicts an output given an input by learning from pairs of inputs and outputs (labels); that is, the program learns from examples that have had the right answers assigned to them beforehand. These assignments are often called **annotations**. Because they are considered the correct answers, they are also called **gold labels**, **gold data**, or the **gold standard**.

The collection of data examples used in supervised learning is called a **training set**. A collection of examples used to assess a program's performance is called a **test set**. Like a student learning in a language course that teaches only through exposure, supervised learning problems see a collection of correct answers to various questions. Then they must learn to provide the correct answers to new but similar questions.

Continuing our exploration, we will delve into two common types of supervised learning: regression and classification, which offer valuable tools for predicting continuous values and categorising data into distinct classes.

### Regression

Regression is a prediction task where a program learns to estimate and predict a continuous output value. It does this by analysing pairs of input features and their corresponding outputs in a training set. By analysing the training examples, the program tries to identify patterns and associations that allow it to make precise estimations.

The main objective of regression is to understand the relationship between the input variables and a continuous target variable, enabling the program to make accurate predictions for new inputs that are similar to the training data.

Commonly used metrics to assess the accuracy of a regression model, which allow for comparing different models or evaluating the performance of a single model, are:

- R-squared ($R^2$),
- mean squared error (MSE),
- root mean squared error (RMSE),
- mean absolute error (MAE),
- and mean absolute percentage error (MAPE).

**$R^2$** is known as the coefficient of determination, quantifying the proportion of variance in the target variable that can be explained by the features in the model. It ranges from zero to one, with higher values indicating a better fit.

**MSE** measures the average squared difference between predicted and actual values, providing an overall measure of prediction accuracy.

**RMSE** is the square root of MSE and represents the average magnitude of prediction errors.

Another metric that provides a measure of the average magnitude of errors is **MAE**, which calculates the average absolute difference between predicted and actual values.

Finally, **MAPE** measures the average percentage difference between predicted and actual values, which is particularly useful when the magnitude of errors needs to be assessed relative to the actual values. By default, lower values of MSE, RMSE, MAE, and MAPE indicate better model performance.

### Classification

Unlike regression tasks, a classification process assumes a program is trained to categorise input data into predefined classes or categories.

By analysing labelled examples, where each example is already assigned to a specific class, the program learns patterns and relationships between input features and classes. This knowledge allows the program to accurately classify new, unseen data, ensuring they are correctly assigned to their respective classes. The ultimate goal of classification is to develop a model that can make reliable predictions for unknown instances, effectively categorising them into the appropriate classes.

To evaluate the effectiveness and performance of classification models, several commonly used evaluation metrics are available. One such metric is the **Gini index**, which measures the impurity or disorder in a set of categorical data. The **confusion matrix** is another valuable tool that summarises the model's performance by displaying counts of true positives, true negatives, false positives, and false negatives. Additionally, **precision** and the **F1 score** can be important metrics, too. Precision calculates the proportion of true positive predictions among all positive predictions, indicating the model's ability to minimise false positives. The **F1 score**, as the harmonic mean of precision and recall, assesses the model's performance by considering both metrics. Together, the discussed metrics provide a comprehensive evaluation of classification models.

### Supervised learning algorithms

Finally, we offer a list of common supervised learning algorithms and their typical usage:

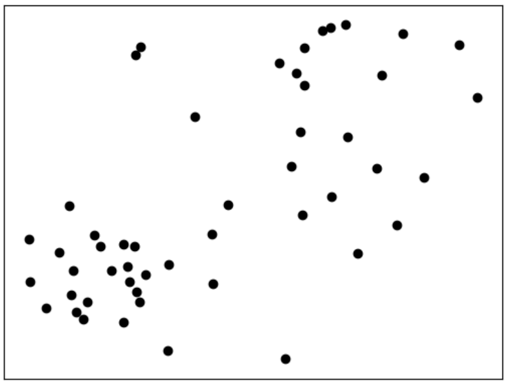
| Supervised learning algorithms | Typical usage | |
|---|---|---|
| | Regression | Classification |

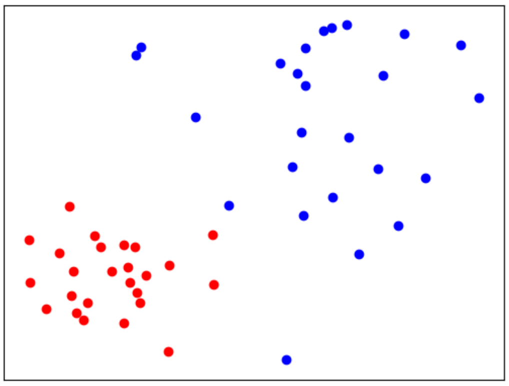
| | | |
|---|---|---|
| Linear regression | ✔ | |
| Logistic regression | | ✔ |
| Decision tree | ✔ | ✔ |
| Random forest | ✔ | ✔ |
| Support vector machines (SVM) | ✔ | ✔ |
| Naïve Bayes | ✔ | |
| K-nearest-neighbour (KNN) | ✔ | ✔ |

## UNSUPERVISED LEARNING

In unsupervised learning, a program does not learn from labelled data. Instead, it attempts to discover patterns in the data on its own.

For instance, suppose you have two classes scattered in a two-dimensional space (as in the first of the images below) and you want to separate the two data sets (as in the second image on the right-hand side). Unsupervised learning finds underlying patterns in the data, allowing the classes to be separated.





To highlight the difference between supervised and unsupervised learning, consider the following example. Assume that you have collected data describing the heights and weights of people. An unsupervised clustering algorithm might produce groups that correspond to men and women, or children and adults. An example of a supervised learning problem is if we label some of the data with the person's sex and then try to induce a rule to predict whether a person is male or female based on their height and weight.

**Unsupervised learning algorithms**

The following algorithms have proven to be highly valuable in practical applications, making them some of the most commonly used methods in unsupervised learning:

- K-means clustering
- Hierarchical clustering
- t-Distributed Stochastic Neighbour Embedding (t-SNE)
- Gaussian Mixture Models (GMM)
- Autoencoders

## SEMI-SUPERVISED LEARNING

Semi-supervised learning is an approach that combines labelled and unlabelled data to harness the benefits of both supervised and unsupervised learning. While supervised learning relies on labelled data with known outcomes and unsupervised learning explores unlabelled data to identify patterns, semi-supervised learning uses a smaller set of labelled data to guide the learning process. Simultaneously, it utilises a larger set of unlabelled data containing valuable information.

By leveraging the combined data set, the algorithm learns from the labelled examples and applies that knowledge to predict outcomes for the unlabelled data, revealing additional patterns and enhancing the model's understanding of the problem's underlying structure. This approach is particularly advantageous when acquiring labelled data is expensive or time-consuming, allowing for optimal resource utilisation and the potential for improved results compared to using just one data type.

## Instructions

First, read **simple_linear_regression.ipynb** and **multiple_linear_regression.ipynb** to better understand regression analysis, and see an example of linear regression in Python.

## Practical Task 1

Follow these steps:

- Create a Jupyter Notebook called **insurance_regression.ipynb**.
- Import **insurance.csv** into your Notebook (**Source**).
- 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:

- Create a Jupyter Notebook called **diabetes_regression.ipynb**.
- Read **diabetes_updated.csv** into your Jupyter Notebook.
- The **diabetes_updated.csv** aims to predict a person's progression in the condition with respect to various attributes about them.
- Differentiate between the independent variables and the dependent variable, and assign them to variables X and Y.
- Generate training and test sets comprising 80% and 20% of the data, respectively.
- Carefully analyse the types of features in this data set. Identify which features require us to use StandardScaler from sklearn.preprocessing. Use StandardScaler on the appropriate features of the train set and test sets.
- Generate a multiple linear regression model using the **training set**. Use all of the independent variables.

- Print out the intercept and coefficients of the trained model.
- Generate predictions for the test set. Compare the values used to make these predictions to the ones in the original diabetes data set. What needs to be done to make the interpretation of our predictions more meaningful?
- Compute R-squared for your model on the **test set**. You can use `r2_score` from `sklearn.metrics` to obtain this score.
- Ensure your Notebook includes comments about what your code is accomplishing, and notes about model outputs such as R-squared.

## Practical Task 3

Answer the following in the provided document titled **machine_learning.ipynb**.

1. For each of the following examples, describe at least one possible **input and output.** Justify your answers:
   1.1. A self-driving car
   1.2. Netflix recommendation system
   1.3. Signature recognition
   1.4. Medical diagnosis

2. For each of the following case studies, determine whether it is appropriate to utilise **regression** or **classification** machine learning algorithms. Justify your answers:
   2.1. Classifying emails as promotional or social based on their content and metadata.
   2.2. Forecasting the stock price of a company based on historical data and market trends.
   2.3. Sorting images of animals into different species based on their visual features.
   2.4. Predicting the likelihood of a patient having a particular disease based on medical history and diagnostic test results.

3. For each of the following real-world problems, determine whether it is appropriate to utilise a **supervised** or **unsupervised** machine learning algorithm. Justify your answers:

3.1. Detecting anomalies in a manufacturing process using sensor data without prior knowledge of specific anomaly patterns.
3.2. Predicting customer lifetime value based on historical transaction data and customer demographics.
3.3. Segmenting customer demographics based on their purchase history, browsing behaviour, and preferences.
3.4. Analysing social media posts to categorise them into different themes.

4. For each of the following real-world problems, determine whether it is appropriate or inappropriate to utilise **semi-supervised** machine learning algorithms. Justify your answers:
   4.1. Predicting fraudulent financial transactions using a data set where most transactions are labelled as fraudulent or legitimate.
   4.2. Analysing customer satisfaction surveys where only a small portion of the data is labelled with satisfaction ratings.
   4.3. Identifying spam emails in a data set where the majority of emails are labelled.
   4.4. Predicting the probability of default for credit card applicants based on their complete financial and credit-related information.

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.

## REFERENCE

IEEE. (1993). *IEEE Standards Collection: Software Engineering*. IEEE Standard 610.12-1990.