

# Random Forests using Python

05/08/2020



# Activate Tensorflow in the base

- Type this line on the prompt in the base

```
-pip install --ignore-installed tensorflow==1.14.0  
matplotlib==3.1.1 h5py==2.9.0 scipy==1.3.1 tqdm==4.32.1  
nibabel==3.0.0 numpy==1.16.5 typed-ast==1.3.0
```

Note:

tutorial 不會有 pandas

base 必須要自己 activate Tensorflow,

nibabel 版本要換成 3.0.0, for DIPY

typed-ast==1.3.0 for cpython in new version of python3.7

Python 的基本變數類型分為以下這幾類：

- 數值
  - float
  - int
  - complex
- 布林值 ( bool )
- 文字 ( str )

Python 回傳變數類型的函數是 `type()`，如果不清楚這個函數有哪些參數可以使用，你可以在 cell 中輸入 `help(type)` 來看說明文件。

跟 R 語言不同的地方是 Python 會自動區別 int 與 float，複數的宣告使用 j 而不是 i，布林值使用 True/False 而不是 TRUE/FALSE。當然，兩者使用的變數類型名稱也有所區別，但是我們可以看到大抵是都能夠很直觀地相互對應，例如：character 對應 str，然後 logical 對應 bool，以及 numeric 對應 float。

- Python 儲存布林值的方式與 R 語言相同，因此也可以彈性地運算數值和布林值，除此以外，在文字上運算的彈性較 R 語言更大一些，可以利用 + 進行合併，以及利用 \* 進行複製。
- 在 Python 中將變數指派給物件的運算子是慣用的 =

```
days = 30
```

```
days = days + 3 days # 33
```

其中的 `days = days + 3` 可以寫作 `days += 3`

# Example:

```
days = 30  
days += 3  
print(days) # 33
```

```
days -= 3  
print(days) # 30
```

```
days *= 5  
print(days) # 150
```

```
days /= 5  
print(days) # 30.0
```

```
days %= 7  
print(days) # 2.0
```

```
days = 30 print("In order to become an ironman, you have to publish an article a day for " +  
days + " days in a row.")
```

```
days = 30  
print("In order to become an ironman, you have to publish an article a day for " + str(days)  
+ " days in a row.")
```

用print()來顯示結果

# Python 變數類型的轉換

Python 轉換變數類型的函數：

- `float()`：轉換變數類型為 `float`
- `int()`：轉換變數類型為 `int`
- `complex()`：轉換變數類型為 `complex`
- `bool()`：轉換變數類型為 `bool`
- `str()`：轉換變數類型為 `str`

```
my_bool = True
print(type(my_bool)) # 'bool'
print(float(my_bool)) # 1.0
print(int(my_bool)) # 1
print(complex(my_bool)) # 1+0j
print(type(str(my_bool))) # 'str'
```

# Python 基本的資料結構：

- 大致有三類

-- **List**，**Tuple** 與 **Dictionary**

- 用一個表格來整理記錄所有的資料

-- **Data Frame**

# List

Python 的 list 跟 R 語言的 list 相似，可以容納不同的變數類型與資料結構，雖然它的外觀長得跟 R 語言的 vector 比較像，但是他們之間有著截然不同的特性，那就是 R 語言的 vector 會將元素轉換成為同一變數類型，但是 list 不會，我們參考一下底下這個簡單的範例。

```
participated_group <- "Big Data"
current_ttl_articles <- 4
is_participating <- TRUE

# 建立一個 vector
my_vector <- c(participated_group, current_ttl_articles, is_participating)
class(my_vector[1])
class(my_vector[2])

# 建立一個 list
my_list <- list(participated_group, current_ttl_articles, is_participating)
class(my_list[[1]])
class(my_list[[2]])
```



# List

在建立 Python 的 list 時候我們只需要使用中括號 `[]` 將元素包起來，而在選擇元素也是使用中括號 `[]` 搭配索引值，Python 的索引值由 0 開始，這跟 R 語言索引值由 1 開始有很大的區別。

```
participated_group = "Big Data"
current_ttl_articles = 4
is_participating = True

my_status = [participated_group, current_ttl_articles, is_participating]
print(type(my_status[0]))
print(type(my_status[1]))
print(type(my_status[2]))
```

# tuple

tuple 跟 list 很像，但是我們不能新增，刪除或者更新 tuple 的元素，這樣的資料結構沒有辦法對應到 R 語言。我們可以使用 `tuple()` 函數將既有的 list 轉換成為 tuple，或者在建立物件的時候使用小括號 `()` 有別於建立 list 的時候使用的中括號 `[]`。

```
# 分別建立 list 與 tuple
ironman_groups_list = ["Modern Web", "DevOps", "Cloud", "Big Data", "Security"]
ironman_groups_tuple = tuple(ironman_groups_list)

# 新增一個元素
ironman_groups_list.insert(5, "自我挑戰組")
ironman_groups_tuple.insert(5, "自我挑戰組")
```

# dictionary

dictionary 是帶有鍵值 ( key ) 的 list，這樣的特性讓我們在使用中括號 `[]` 選擇元素時可以使用鍵值，在建立 Python 的 dictionary 時候我們只需要使用大括號 `{}` 或者使用 `dict()` 函數轉換既有的 list。

```
participated_group = "Big Data"
current_ttl_articles = 4
is_participating = True

# 建立 dictionary
my_status = {
    "group": participated_group,
    "ttl_articles": current_ttl_articles,
    "is_participating": is_participating
}

# 利用鍵值選擇元素
print(my_status["group"])
print(my_status["ttl_articles"])
print(my_status["is_participating"])
```

# element-wise 的運算

- 如果我們希望在 Python 輕鬆地使用 element-wise 的運算，我們得仰賴 numpy 套件中提供的一種資料結構 numpy array，或者採用更精準一點的說法是 **ndarray** 這個資料結構。

```
import numpy # 引用套件

ironmen = numpy.array([46, 8, 11, 11, 4, 56]) # 將 list 透過 numpy 的 array 方法進行轉換
print(ironmen) # 看看 ironmen 的外觀
print(type(ironmen)) # 看看 ironmen 的資料結構
articles = ironmen * 30
print(articles)
```

# element-wise 的運算

## Element-wise 運算

NumPy 的 ndarray 完全支持 element-wise 運算。

```
import numpy as np

my_np_array = np.array([1, 2, 3, 4])
print(my_np_array ** 2)

my_2d_array = np.array([[1, 3],
                        [2, 4]])
print(my_2d_array ** 2)
```

# Numpy的ndarray

## 單一資料類型

NumPy 的 ndarray 只能容許一種資料類型，如果同時儲存有數值，布林值，會被自動轉換為數值，如果同時儲存有數值，布林值與文字，會被自動轉換為文字。

```
import numpy as np

my_np_array = np.array([1, True])
print(my_np_array.dtype) # int64
my_np_array = np.array([1, True, "one"])
print(my_np_array.dtype) # unicode_21
```

# 選擇元素

NumPy 的 ndarray 透過中括號 `[]` 或者布林值選擇元素。

```
import numpy as np

ironmen = np.array([46, 8, 11, 11, 4, 56])
print(ironmen[0]) # 選出 Modern Web 組的鐵人數
print(ironmen > 10) # 哪幾組的鐵人數超過 10 人
print(ironmen[ironmen > 10]) # 超過 10 人的鐵人數
```

```
import numpy as np

ironmen_2d_array = np.array([[46, 11, 4],
                             [8, 11, 56]])
print(ironmen_2d_array[0, 0]) # 選出 Modern Web 組的鐵人數
print(ironmen_2d_array > 10) # 哪幾組的鐵人數超過 10 人
print(ironmen_2d_array[ironmen_2d_array > 10]) # 超過 10 人的鐵人數
```

# 了解 2d array 外觀的屬性

NumPy 可以透過 `.size` 與 `.shape` 來了解 2d array 的規模。

```
import numpy as np

ironmen_2d_array = np.array([[46, 11, 4],
                             [8, 11, 56]])

print(ironmen_2d_array.size) # 6
print(ironmen_2d_array.shape) # (2, 3)
```



# Python 資料結構: Data Frame

- 用一個**表格**來整理記錄所有的資料
- 如果我們希望在 Python 中也能夠使用 data frame，我們得仰賴 **pandas** 套件
- 最基本建立 data frame 的方式是利用 pandas 套件的 DataFrame() 方法將一個 dictionary 的資料結構轉換為 data frame
- **讀入excel檔, 其自動轉為data frame**

# Python 資料結構: Data Frame

```
import pandas as pd # 引用套件並縮寫為 pd

groups = ["Modern Web", "DevOps", "Cloud", "Big Data", "Security", "自我挑戰組"]
ironmen = [46, 8, 12, 12, 6, 58]

ironmen_dict = {"groups": groups,
                "ironmen": ironmen
                }

ironmen_df = pd.DataFrame(ironmen_dict)

print(ironmen_df) # 看看資料框的外觀
print(type(ironmen_df)) # pandas.core.frame.DataFrame
```

# Python 資料結構: Data Frame

## 包含多種資料類型

跟 list 的特性相仿，不會像 ndarray 僅限制容納單一資料類型。

```
import pandas as pd

groups = ["Modern Web", "DevOps", "Cloud", "Big Data", "Security", "自我挑戰組"]
ironmen = [46, 8, 12, 12, 6, 58]

ironmen_dict = {"groups": groups,
                "ironmen": ironmen
                }

ironmen_df = pd.DataFrame(ironmen_dict)
print(ironmen_df.dtypes) # 欄位的變數類型
```

## 選擇元素

Pandas 透過使用中括號 `[]` 與 `.iloc` 可以很靈活地從 data frame 中選擇想要的元素。要注意的是 Python 在指定 `0:1` 時不包含 `1`，在指定 `0:2` 時不包含 `2`，這一點是跟 R 語言有很大的不同之處。

```
import pandas as pd

groups = ["Modern Web", "DevOps", "Cloud", "Big Data", "Security", "自我挑戰組"]
ironmen = [46, 8, 12, 12, 6, 58]

ironmen_dict = {"groups": groups,
                "ironmen": ironmen
                }

ironmen_df = pd.DataFrame(ironmen_dict)

print(ironmen_df.iloc[0:1, 1]) # 第一列第二欄: Modern Web 組的鐵人數
print("----")
print(ironmen_df.iloc[0:1,:]) # 第一列: Modern Web 組的組名與鐵人數
print("----")
print(ironmen_df.iloc[:,1]) # 第二欄: 各組的鐵人數
print("----")
print(ironmen_df["ironmen"]) # 各組的鐵人數
print("----")
print(ironmen_df.ironmen) # 各組的鐵人數
```

## 可以使用布林值篩選

Pandas 可以透過布林值來針對 data frame 進行觀測值的篩選。

```
import pandas as pd

groups = ["Modern Web", "DevOps", "Cloud", "Big Data", "Security", "自我挑戰組"]
ironmen = [46, 8, 12, 12, 6, 58]

ironmen_dict = {"groups": groups,
                "ironmen": ironmen
                }

ironmen_df = pd.DataFrame(ironmen_dict)

print(ironmen_df[ironmen_df.loc[:, "ironmen"] > 10]) # 選出鐵人數超過 10 的 data frame
```

## 了解 data frame 概觀

Pandas 的 data frame 資料結構有一些方法或屬性可以幫助我們了解概觀。

```
import pandas as pd

groups = ["Modern Web", "DevOps", "Cloud", "Big Data", "Security", "自我挑戰組"]
ironmen = [46, 8, 12, 12, 6, 58]

ironmen_dict = {"groups": groups,
                "ironmen": ironmen
                }

ironmen_df = pd.DataFrame(ironmen_dict)

print(ironmen_df.shape) # 回傳列數與欄數
print("---")
print(ironmen_df.describe()) # 回傳描述性統計
print("---")
print(ironmen_df.head(3)) # 回傳前三筆觀測值
print("---")
print(ironmen_df.tail(3)) # 回傳後三筆觀測值
print("---")
print(ironmen_df.columns) # 回傳欄位名稱
print("---")
print(ironmen_df.index) # 回傳 index
```

# NumPy 的 ndarray

## 單一資料類型

NumPy 的 ndarray 只能容許一種資料類型，如果同時儲存有數值，布林值，會被自動轉換為數值，如果同時儲存有數值，布林值與文字，會被自動轉換為文字。

```
import numpy as np

my_np_array = np.array([1, True])
print(my_np_array.dtype) # int64
my_np_array = np.array([1, True, "one"])
print(my_np_array.dtype) # unicode_21
```

```
import numpy as np

my_2d_array = np.array([[1, True],
                        [0, False]])
print(my_2d_array)

my_2d_array = np.array([[1, True],
                        ["zero", False]])
print(my_2d_array)
```

# Python資料分析工具

Python資料分析絕對繞不過的四個包是numpy、scipy、pandas還有matplotlib。

numPy是Python數值計算最重要的基礎包，大多數提供科學計算的包都是用numPy的陣列作為構建基礎。專門用來處理矩陣，它的運算效率比列表更高效。

scipy是基於numpy的科學計算包，包括統計、線性代數等工具。

pandas是基於numpy的資料分析工具，能夠快速的處理結構化資料的大量資料結構和函數。

matplotlib 是最流行的用於繪製資料圖表的 Python 庫。



# numpy

- **numpy**是Python數值計算最重要的基礎包，大多數提供科學計算的包都是用**numPy**的陣列作為構建基礎。專門用來處理矩陣，它的運算效率比列表更高效。
- **NumPy 的 ndarray**：多維陣列物件
- **numpy**的資料結構是n維的陣列物件，叫做**ndarray**。可以用這種陣列對整塊資料執行一些數學運算

<https://medium.com/@allaboutdataanalysis/python%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E4%B8%89-numpy-3a938f435286>

# numpy

```
In [20]: import numpy as np
```

```
In [21]: data = np.array([[0.9526, -0.246, 0.08856],[0.5639, 0.2397, 0.9104]])
```

```
In [22]: data
```

```
Out[22]: array([[ 0.9526 , -0.246  ,  0.08856],  
                [ 0.5639 ,  0.2397 ,  0.9104 ]])
```

```
In [23]: data * 10
```

```
Out[23]: array([[ 9.526 , -2.46  ,  0.8856],  
                [ 5.639 ,  2.397 ,  9.104 ]])
```

```
In [24]: data + data
```

```
Out[24]: array([[ 1.9052 , -0.492  ,  0.17712],  
                [ 1.1278 ,  0.4794 ,  1.8208 ]])
```

<https://medium.com/@allaboutdataanalysis/python%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E4%B8%89-numpy-3a938f435286>

# numpy

ndarray物件中所有元素必須是相同類型的，每個陣列都有一個shape和dtype。

- shape：表示各維度大小的元組
- dtype：說明陣列資料類型的物件

```
In [25]: data.shape
```

```
Out[25]: (2, 3)
```

```
In [26]: data.dtype
```

```
Out[26]: dtype('float64')
```

<https://medium.com/@allaboutdataanalysis/python%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E4%B8%89-numpy-3a938f435286>

# Pandas

Pandas 是 python 的一個數據分析 lib，2009 年底開源出來，提供高效能、簡易使用的資料格式(Data Frame)讓使用者可以快速操作及分析資料，主要特色描述如下：

1. 在異質數據的讀取、轉換和處理上，都讓分析人員更容易處理，例如：從列欄試算表中找到想要的值。
2. Pandas 提供兩種主要的資料結構，Series 與 DataFrame。Series 顧名思義就是用來處理時間序列相關的資料(如感測器資料等)，主要為建立索引的一維陣列。DataFrame 則是用來處理結構化(Table like)的資料，有列索引與欄標籤的二維資料集，例如關聯式資料庫、CSV 等等。
3. 透過載入至 Pandas 的資料結構物件後，可以透過結構化物件所提供的方法，來快速地進行資料的前處理，如資料補值，空值去除或取代等。
4. 更多的輸入來源及輸出整合性，例如：可以從資料庫讀取資料進入 Dataframe，也可將處理完的資料存回資料庫。

# Pandas 讀取資料

可以從異質資料來源讀取檔案內容，並將資料放入 DataFrame 中，進行資料查看、資料篩選、資料切片等運算。

## ❖ 讀取 CSV 檔案

```
# 讀取 CSV File
import pandas as pd # 引用套件並縮寫為 pd
df = pd.read_csv('shop_list.csv')
print(df)
```

## ❖ 讀取 Html 檔案

```
# 讀取 HTML
import pandas as pd # 引用套件並縮寫為 pd
dfs = pd.read_html('http://rate.bot.com.tw/xrt?Lang=zh-TW')
dfs[0]
```

# Pandas 提供的資料結構

- **1.Series**：用來處理時間序列相關的資料(如感測器資料等)，主要為建立索引的一維陣列。
- **2.DataFrame**：用來處理結構化(Table like)的資料，有列索引與欄標籤的二維資料集，例如關聯式資料庫、CSV 等等。
- **3.Panel**：用來處理有資料及索引、列索引與欄標籤的三維資料集。

# Python 的機器學習套件 scikit-learn

## Sklearn

- scikit-learn 套件是專門用來實作機器學習以及資料採礦的

機器學習是一門設計如何讓演算法能夠學習的電腦科學，讓機器能夠透過觀察已知的資料學習預測未知的資料。典型的應用包含概念學習（Concept learning）、函數學習（Function learning）、預測模型（Predictive modeling）、分群（Clustering）與找尋預測特徵（Finding predictive patterns）。終極目標是讓電腦能夠自行提升學習能力，預測未知資料的準確性能夠隨著已知資料的增加而提高，節省使用者人工調整校正的精力。

# scikit-learn 套件的應用領域

- 監督式學習（Supervised learning）
  - 分類（Classification）：**RandomForestClassifier**
  - 迴歸（Regression）：**RandomForestRegressor**
- 非監督式學習（Unsupervised learning）
  - 分群（Clustering）
- 降維（Dimensionality reduction）
- 模型選擇（Model selection）
- 預處理（Preprocessing）



# 實例練習

# Random Forests Classifiers in Python



Avinash Navlani  
May 16th, 2018

PYTHON

## Understanding Random Forests Classifiers in Python

Learn about Random Forests and build your own model in Python, for both classification and regression.



**EXPLORE DATA CAMP'S PYTHON  
COURSE LIBRARY**

Explore Now

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

<https://www.datacamp.com/community/tutorials/random-forests-classifier-python>