1.1 機器學習、MLOps、系統

要將機器學習送到使用者面前，需要能執行推論的系統。系統工程具有開發與維護這兩個方面，而這兩個組成的作業就稱為 DevOps。本書將為大家說明包含機器學習的開發與改善流程的 MLOps。

1.1.1 前言

深度學習的問世掀起了第三次 AI 熱潮，也有許多人著手開發機器學習與深度學習的模式。機器學習這個字眼不只在科學或軟體工程這些業界出現，也已於商界普及，有許多的企業試著利用機器學習建立全新的商業模式或是改善業務流程。機器學習雖不是一針見血的萬靈藥，但只要資料足夠，就能完成人類無法想像的精準預測。透過資料與機器學習，就能踏入目前的技術未能觸及的領域。比方說，向來仰賴人類直覺與經驗的人工製造業若能利用機器學習的異常偵測與變化偵測找出不良品，整個流程就會變得更有效率與標準化，而這類可透過機器學習改善的流程可說是不勝枚舉。透過機器學習解決這類依賴個人經驗或直覺的工作流程，或是尋找改善方式，是近來機器學習界的最新動向。

此外，機器學習是利用電腦進行計算，以及利用軟體制定動作。由於機器學習本身也是利用軟體建置，所以機器學習可說是軟體工程的一部分。既然是軟體工程的一部分，要讓機器學習付諸實用，就必須製作軟體，將軟體移植到執行軟體的架構，再讓整個架構形成一套系統。機器學習除了需要撰寫程式碼，還因為觸及資料處理與機率計算這類領域，所以需要以有別於傳統軟體的開發方式開發。不過，要使用機器學習，就必須先準備電腦（CPU、記憶體、外部儲存裝置，有必要的話，還得準備網路或感測器這類周邊機器，其中又包含伺服器、電腦、平板電腦、智慧型手機、微電腦），也需要撰寫程式，再於電腦安裝，而這些部分與傳統的軟體沒有不同。若將機器學習視為軟體，就必須具備將機器學習移植到系統的技術與維護相關系統的技術。

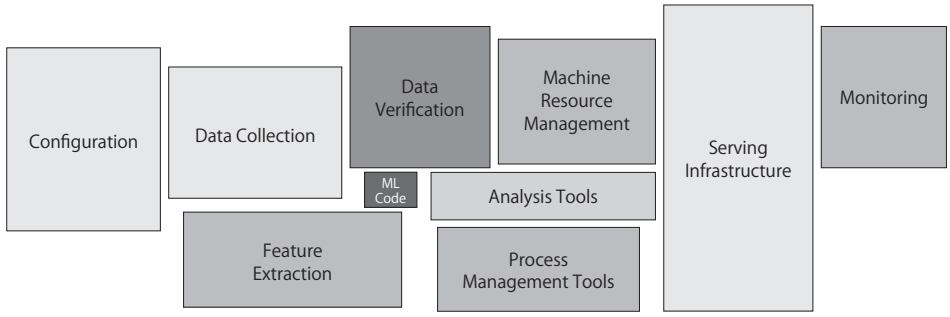

圖 1.1 機器學習所需的系統元件

出處 節錄自『Hidden Technical Debt in Machine Learning Systems』（D. Sculley、Gary Holt、Daniel Golovin、Eugene Davydov、Todd Phillips、Dietmar Ebner、Vinay Chaudhary、Michael Young、Jean Francois Crespo、Dan Dennison、2015）、Figure 1

URL http://papers.neurips.cc/paper/5656 hidden technical debt in machine learning systems.pdf

要開發與維護機器學習系統，就要定義包含機器學習的系統工作流程（圖 1.2）。若將焦點放在機器學習，機器學習的流程可分成學習與推論這兩大部分。學習階段的目標在於收集與清洗資料，再根據資料傾向學習模型，建立能精確推論的模型。此時需要實驗或驗證的元素較多，也需要依照目標選用適當的演算法，以及微調各種參數，再於最後的評估階段以測試資料確認機器學習模型的實用度。

進入機器學習的推論階段之後，就會在正式的系統使用於學習階段建置的模型，將這個模型當成推論器使用。也就是將實際的資料輸入模型，再輸出推論結果。推論器需要包含機器學習的技術元素（資料的事前處理以及利用模型進行推論）與軟體工程的元素（讓推論器得以運作的基礎建設、與外部系統連接的網路設計、安全性策略、輸出入介面的定義、撰寫程式碼、各種測試、收集日誌資料、設定監控通知、建置維護機制），如此一來，機器學習的模型才能真的付儲實用與發揮效果。將學習所得的模型當成推論器移植到正式的系統，以及執行推論，才能看出機器學習模型的產品價值，而推論的結果是否具有價值，則可由事件日誌資料與使用者動向評估。根據收集到的日誌資料評估推論的精確度以及衍生的附加價值（業績或異常偵測率）。根據推論器的評估結果以及新收集的資料改善機器學習的模型，藉此開發出更具商業價值的模型。

2.4 管線學習模式

機器學習的學習可分割成不同的階段與多個流程，一般來說，可分割成取得資料、前置處理、學習、評估、建置這些流程，而將這些流程視為各階段任務執行，再於過程記錄學習結果，就能循環利用這些學習結果，也能迅速修正部分的設定。

2.4.1 使用時機

- 希望分割管線學習的資源，於每項任務選擇函式庫與沿用函式庫的時候。
- 希望記錄資料於每項任務的狀態與變化，快速重啟作業的時候。
- 希望分別控制各任務的執行過程的時候。

2.4.2 想解決的課題

機器學習的學習階段包含取得資料、處理資料、進行學習與評估學習結果。這個工作流程通常被稱為「前置處理」、「學習」、「評估」、「建置」、「系統評估」這些流程，每個流程都會輸出加工過的資料、模型或是評估值，而這些輸出值則會成為下個流程的輸入值。因此，學習的工作流程很像是讓資料往下游流動的資料管線。

管線學習模式會記錄每個步驟的輸出值，讓使用者隨時能從中途重新學習。學習結束後，也可以利用透過學習產生的模型評估結果，以及執行建置的流程。即使是正在學習的階段，也可以輸出學習到一半的模型，之後也能繼續從這個中斷學習的地方重啟學習流程。管線學習模式的優點之一是可分段執行整個流程，也能隨時測試流程。由於流程之間是透過檔案互傳資料，所以每個流程都可獨立開發與進行測試。

除了上述的函式庫之外，還支援 Amazon SageMaker（URL https://docs.aws.amazon.com/zh_tw/sagemaker/）、Kubeflow（URL https://www.kubeflow.org/）、Metaflow（URL https://metaflow.org/）這類機器學習管線框架或函式庫，也內建了各種相關的工具。

這次之所以選用 MLflow，是因為這套開源碼軟體相對容易使用。若要在雲端或 Kubernetes 進行管線學習，可改用 Amazon SageMaker 或是 Kubeflow。不過，本書的重點不是說明機器學習管線函式庫的使用方式，所以才選擇採用成本較低的 MLflow。

MEMO

Cifar-10

Cifar-10（圖 2.8）是專為圖片辨識設計的圖片資料集，提供了 10 個類別（飛機、汽車、鳥、貓、鹿、狗、青蛙、馬、船、卡車）的資料。每個類別包含了 5,000 張用於學習的圖片，1,000 張用於測試的圖片，總計共有 60,000 張圖片。每張圖片的大小都是 32×32 像素，每個像素都是 RGB 色彩，是非常方便好用的資料庫。

如果使用 PyTorch 學習的話，其實可以直接撰寫利用 Cifar-10 學習的程式碼，不需要刻意將每個流程打造成機器學習管線的一部分，但這次為了介紹相關的內容，而特意寫成機器學習管線的格式。

圖 2.8 Cifar-10

這次要試著將透過 PyTorch 學習 Cifar-10（URL https://www.cs.toronto.edu/~kriz/cifar.html）這個影像分類模型的流程打造成管線學習模式。之所以選擇 Cifar-10 與 PyTorch，也是因為採用成本較低，又很方便重現相同的結果。

機器學習管線是以下列四個步驟撰寫。

步驟 1　取得資料：取得圖片資料，再於本地終端設備儲存。

步驟 2　學習與評估：利用 VGG11 這個深度學習模型學習於步驟 1 取得的圖片。評估完成學習的模型，記錄 Accuracy 與 Loss。

步驟 3　建置：將步驟 2 產生的模型建置為推論所需的 Docker 映像檔。

步驟 4　系統評估：啟動步驟 3 產生的 Docker 映像檔，發出推論要求，測試模型與推論器的連線狀況。

上述的每個步驟會分別在不同的 Docker 容器執行，資料則以 MLflow 的 artifact 傳遞。artifact 是於 MLflow 的每個步驟產生的資料或檔案。後半段的步驟會將前半段的步驟產生的 artifact 當成輸入資料，利用該資料進行學習與評估，圖 2.9 則是整個流程的示意圖。

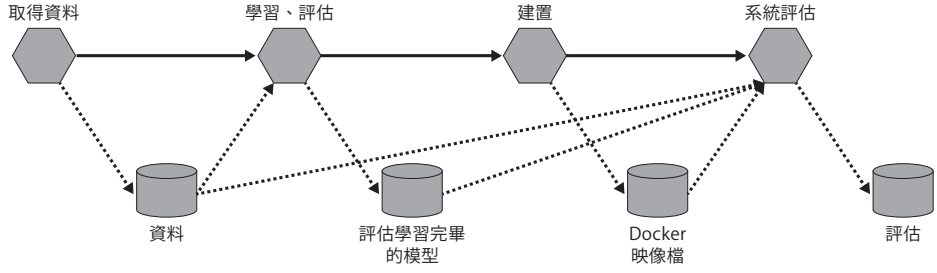

圖 2.9　打造機器學習管線的流程

接下來讓我們一起了解相關的程式碼。由於程式碼非常長，在此僅節錄重要的部分。

3.2 反面模式 版本不一致模式

在正式系統發佈機器學習的模型時，最常發生的問題就是學習環境與推論環境使用的 OS、函式庫或程式不一致，尤其是要求函式庫的版本一致的模型更是會因為版本有出入而無法善用資源。

在此先從發佈機器學習模型的反面模式說明。一如 **Chapter 3** 的 **3.1.2** 節「學習環境與推論環境」所述，將機器學習的模型發佈為推論器的時候，最大的問題在於學習環境與推論環境的規格有明顯的落差。身為 MLOps 工程師的筆者到目前為止，看過不少專案因為不了解上述的落差而在發佈模型的時候受挫，所以本節將為大家說明學習環境與推論環境的函式庫必須一致的理由與重要性。

3.2.1 狀況

- 明明學習環境與推論環境使用了相同的函式庫，但是函式庫的版本卻不一致。
- 無法將模型載入推論器。
- 推論器的推論結果與學習環境產生的推論結果不一致。

3.2.2 具體問題

要將學習環境建置的模型移植到推論器時，最重要的就是讓學習環境與推論環境使用相同版本的程式設計語言與函式庫。比方說，是以 Python 2.7 學習模型，卻在推論器這邊使用 Python 3.9 的話，會發生問題應該是很正常的事情。除了程式設計語言的版本之外，函式庫的版本也必須一致。

最顯著的例子就是 scikit-learn（URL https://scikit-learn.org/stable/）的版本不一致。scikit-learn 是非常簡潔好用的機器學習函式庫，只要是機器學

習工程師，就一定使用過這個函式庫，就算不是從頭到尾都使用 scikit-learn 開發模型，也有可能只在前置處理的時候使用 scikit-learn。

要將 scikit-learn 開發的模型移植到推論器的時候（圖 3.3），通常會利用 Python 的 pickle 將模型輸出成 .pkl 檔案，再利用這個 .pkl 檔案將模式載入推論器。pickle 會儲存 scikit-learn 建立的模型（其實是具有機器學習參數的類別物件）的實體變數。載入 .pkl 檔案之後，以 pickle 儲存的物件的類別就會實體化，所以當 pickle 與推論器的函式庫是不同的版本，就會發生類別相同，但變數與函數的內容不一致的問題，也就無法成功載入 .pkl 檔案。如果在 pickle 與推論器使用不同的類別（即使是相同的函式庫或類別名稱），當然無法載入 .pkl 檔案，但真正的問題在於無法單從 .pkl 檔案判斷 .pkl 檔案是以哪種函式庫的哪個版本產生。

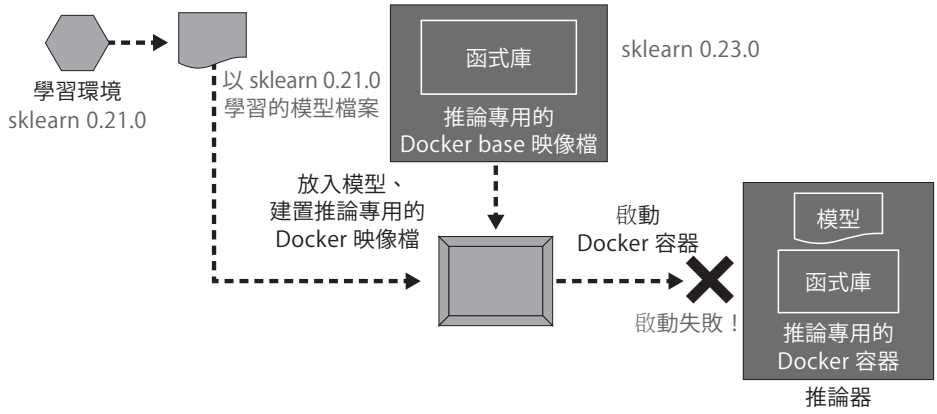

圖 3.3 學習環境與推論環境使用了不同版本的 scikit-learn 的情況

除了 scikit-learn 之外，所有的軟體都會發生這個問題。比方說，TensorFlow 從 1.x 更新至 2.x 的時候，做了大幅度的修訂，所以就算是同一個類別，其中的規格也大不相同（例如 `tensorflow.keras`），而且 TensorFlow 1.x 的某些功能也無法在 TensorFlow 2.x 使用（`tensorflow.lite.TFLiteConverter` 不支援部分的操作）。新版的 PyTorch 也不一定與舊版完全相容，有時版本一更新，PyTorch Tensor 的預設格式以及預設常數就會不同。這類相容性問題會於各種函式庫發生，使用者也必須想辦法解決。

3.5 Model Loader 模式

在介紹 Model in Image 模式的時候，說明了將模型放入伺服器映像檔的方法。這節將介紹 Model Loader 模式不將模型放入伺服器映像檔建置，只在推論器啟動時下載與移植到模型的方法。

3.5.1 用例

- 希望推論模型的版本比伺服器映像檔的版本更常更新的時候。
- 希望以同一個伺服器映像檔驅動多種推論模型的時候。

3.5.2 想解決的課題

前一節介紹的 Model in Image 模式的優點在於伺服器映像檔與模型的版本一致，但缺點則是會需要不斷地建置伺服器映像檔，伺服器映像檔的檔案容量也會變大。如果是一個模型搭配一個伺服器映像檔這種專為某個模型建置推論器的用途，Model in Image 模式會是比較合理的選擇。

不過，若是同一個基礎映像檔裡的模型會不斷更新版本的話，就不太適合選用 Model in Image 模式，否則會花很多時間維護。就算是以相同的前置處理以及決策樹進行學習，只要用於學習資料集不同，每次學習都得建置伺服器映像檔，而這當然不是合理的方式。如果選擇的是通用的學習參數，有時只會頻繁地更新資料集，藉此建置新模型。若要以這種工作流程開發模型，那麼 Model Loader 模式將是發行模型的最佳方式。

3.5.3 架構

將伺服器映像檔與模型分開來管理，可讓建置伺服器映像檔的步驟（或是管理函式庫版本的步驟）與學習模型的步驟分開來。假設採取的是 Model loader

模式，將會分別執行建置推論伺服器映像檔的步驟與儲存模型的步驟，如此一來，伺服器映像檔的檔案容量就會變小許多（圖 3.6）。而且還能提升伺服器映像檔的通用性，能以同一個伺服器映像檔驅動多個推論模型。假設想讓單一的伺服器映像檔擁有多項用途，就可以採用 Model loader 模式。

若是採用 Model loader 模式，會在部署推論器的時候，先以 Pull 下載伺服器映像檔，接著再啟動推論器，然後取得模型檔案，再讓推論器正式運作。利用環境變數調整模型載入的位置，就能隨時變更在推論伺服器運作的模型。

這種模式的問題在於模型是以特定的函式庫的版本建置時，就必須分頭管理伺服器映像檔的版本與模型檔案的版本（管理支援的函式庫的版本），此時會需要製作伺服器映像檔與模型的支援表，而且伺服器映像檔與模型一旦增加，版本就會越來越難管理，維護的成本也會大增。

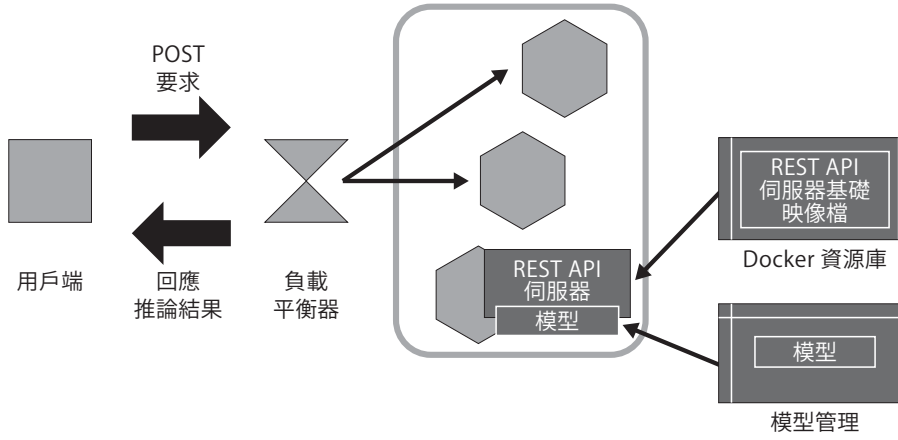

圖 3.6 Model loader 模式

3.5.4 實際建置

接著要建立 Model loader 模式。與 Model in Image 模式不同的是，Model loader 模式不會將模型檔案放入 Docker 映像檔，而是在 Docker 容器啟動時下載模型。模型檔案可先放在 AWS S3（AWS 的物件儲存服務）或 GCP

4.1 為什麼要建立系統

要於產品或服務應用機器學習，就必須將機器學習移植到系統。不過，機器學習的模型若只是在本地端電腦進行推論，是無法做出任何貢獻的，必須建立一個模型能與其他軟體搭配，或是從其他軟體呼叫模型的系統。

4.1.1 前言

到目前為止，說明了學習與管理機器學習模型的方法。要建置一個機器學習的模型，需要使用各種資料、演算法與參數，而這些資料、演算法與參數會影響模型的效能，也會讓人無法從模型了解這個模型是利用哪些資料或參數學習。要想活用機器學習，就必須管理用於模型學習的設定以及模型的版本。

更重要的是，要將機器學習的模型移植到正式系統，讓機器學習得以付諸實用。只有當機器學習的模型開始於公司業務應用才算真的有價值。不管模型的效能有多好，無法於正式系統應用就毫無意義。本章除了說明將機器學習模型移植到正式系統的各種模式，還要介紹建置這些模式的方法以及優缺點。

4.1.2 讓機器學習付諸實用

要讓機器學習的模型付諸實用，就必須讓模型扮演推論器的角色，進而與外部系統建立互動（圖 4.1）。將學習所得的模型檔案載入推論器之後，驅動推論器，再根據正式的資料輸出推論結果。重點在於，將推論結果提供給公司內外的終端使用者，讓系統變得更方便好用。

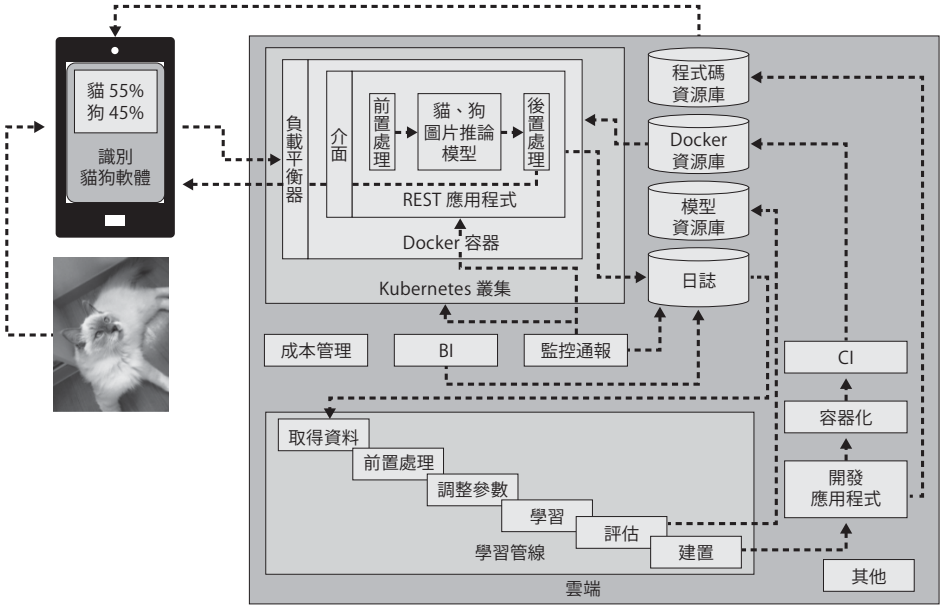

圖 4.1 讓機器學習模型付諸實用的流程

要將學習完畢的模型當成推論器移植到正式系統的方法非常多，但要選擇哪種方法得根據系統的用途與模型的效能決定。比方說，在接收要求時，要立刻回應推論結果的話，最好同步建置推論器。此外，如果推論器是由多個模型組成，就必須將這些模型部署成獨立的微服務模型。如果想在晚上根據累積的大筆資料進行推論，則可以開發成批次系統。

以下是本章要說明的各種推論器模式，以及各種模式的簡易說明。

- Web Single 模式：以一個推論器同步推論一個小模型。
- 同步推論模式：對要求進行同步推論。
- 非同步推論模式：對要求進行非同步推論。
- 批次推論模式：利用批次處理進行推論。
- 前置處理推論模式：以前置處理與推論分割伺服器。

4.2 Web Single 模式

Web Single 模式是將機器學習的推論模型移植到一台 WebAPI 服務。將資料與要求一併傳送至 API，就能取得推論結果，而這也是最簡單、最基礎的推論系統架構。

4.2.1 用例

- 希望以最簡單的架構早一步發佈推論器，驗證模型的效能。

4.2.2 想解決的課題

機器學習的模型在完成學習之後，盡快以實際的資料測試模型的效能是至關重要的部分。用於學習的資料，尤其是於網路服務使用的資料每天都在改變，這也意謂著訓練模型的資料越來越不符合時代的需求，所以模型在剛學習完畢的時候，最能反映正式資料的真實情況（≒商業現況），機器學習模型的正確解答率也是在學習完畢的當下達到顛峰。以具有淡季、旺季的服務為例，以兩個月前的資料進行學習的模型，很可能無法滿足當季的需求，所以要盡可能使用一個月之內的資料學習，以及趁著資料還夠新鮮的時候，讓模型移植到正式系統，才能產生預期的價值。換言之，在商業應用的世界裡，儘早發佈模型是非常重要的一環。

4.2.3 架構

那麼該怎麼做才能儘速發佈模型呢？

為此，要快速開發與建置推論器。要想加快開發的腳步，最簡單的方法就是不要多做無謂的東西。比方說，伺服器只開發最低需求的功能，外部介面也使用能與大部分系統相容的 REST API（當然也可以使用 gRPC）。推論器則利用 Python 的網路框架（**Flask** URL https://flask.palletsprojects.com/）