基本資訊

學號:E94074029

姓名:江羿賢

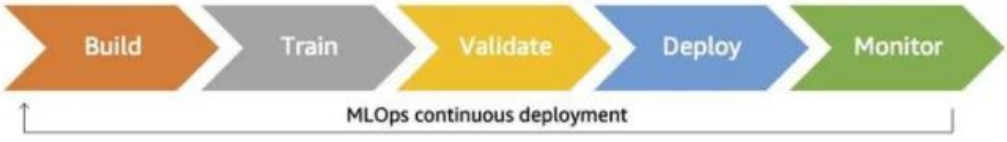
系級:資訊系 111 級

MLOps 的 OS 設計

要比較能夠設計出可以較為適合 MLOps 的 OS 的話，首先要先來看一下 ML 會有甚麼樣子的流程，可以歸納出這些步驟，1.框架的設定以及目標的確定 2.取得資料並探索資料 3.資料的前處理 4.嘗試各種機器學習的模型，並且整合出最符合目標的方案 5.撰寫文件並且發佈、監控並維護你的系統(參考資料[9])。

而接下來來看一下 MLOps 的一個大方向的流程，另外並且可以從中切入看有沒有甚麼功能是可以透過 OS 來提供較佳的服務以及資源的分配，讓使用可以更快速又輕鬆的。

以下是一個簡單的生命週期小流程圖(參考資料[1])



但這個比較算是一個大方向，而好的 OS 設計可以使得 MLOps 的整個流程更加的順利(這邊的順利包含了順暢、自動化)

接下來來看 MLOps 比較細節的一些流程(參考資料[1]):

DataSet 的來源、版本控管、Training Model History、Training Model Costs、Model 執行環境的一致性、Model 的回溯性、平行訓練 Model 的工作流程……等等

可以透過這邊的流程中可以來看一下在 OS 要提供甚麼服務給我們會比較好，可以來更有效率的完成這些事情，以下挑出幾個我認為在 OS 上可以進行設計來改善的一些流程

DataSet sources 和 Model 的回溯性(這邊一起講): 這個部分比較能夠討論的是資料的一個安全性以及存取的方式，如果這邊是常常需要更新的資料的話，那比較要注意的是 DataSet 如果是一個全新的一個來源的話，OS 可以提供一個隨機檢測的方式，或者是看檔案的大小，來看一下裡面有沒有異常大小的資料，甚至是惡意程式等等，並且提出警告，而如果是一個現有的來源要更新的話，或許可以提供看是哪裡變動了並且檢測是否有異狀，如果覺得這個 DataSet 需要保留的話，OS 可以提供透過非對稱加密的 RSA(參考資料[8])，並且將金鑰給信任的協作者，來確保資料的安全性，以及可以減少 OS 要去檢測 Data 的時間，並且如果資料偵測到有變動的同時，OS 也要可以自己再重新 Build，而這邊如果發現有一些 Data 是會離峰值的 Data，也可以在檢測的時候去註記，或者是由

平均值之類的去補上這個 Data，也可以在日後訓練的時候增加準確度，而離峰值得檢測可以利用 Standard Deviation、。Boxplots、DBScan Clustering、Isolation Forest、Robust Random Cut Forest (參考資料[3])，來使 OS 自己判斷要用甚麼方式來去對離峰值的處理以及去做檢測。而如果日後想要去局部做訓練的話可以使 OS 自己裡面可以利用特徵提取的方式如(RNN、CNN、Transformer)來對要訓練的東西去提取而詳細的方放我這邊就不再多贅述，可以參考(參考資料[4])，而這邊 OS 要有能力可以自動記錄下來想要提取的東西才算是比較好的一個設計，並且特徵的輸出可以去離散化成 0 和 1，看特徵分布關於某些值是對稱的，並將其設定成 threshold，然後使用 tanh 或 sigmoid(參考資料[5])，之後一樣可以提供可以較好 search 的方式來使 user 往後要再對這邊特別進行 training 的時候可以更方便去找出來，更甚至比較好的話，OS 可以自動判斷需要重新再 train 的那個特徵的部分，在資源閒置時可以自己進行 training 並下次使用者再上線時再進行提醒，看結果有沒有變得比較好。

Training Model History:這個部分就是可以使 OS 提供一個可以記錄工作歷程的方式，可以提供一些磁碟的資源來去記錄如是使用哪

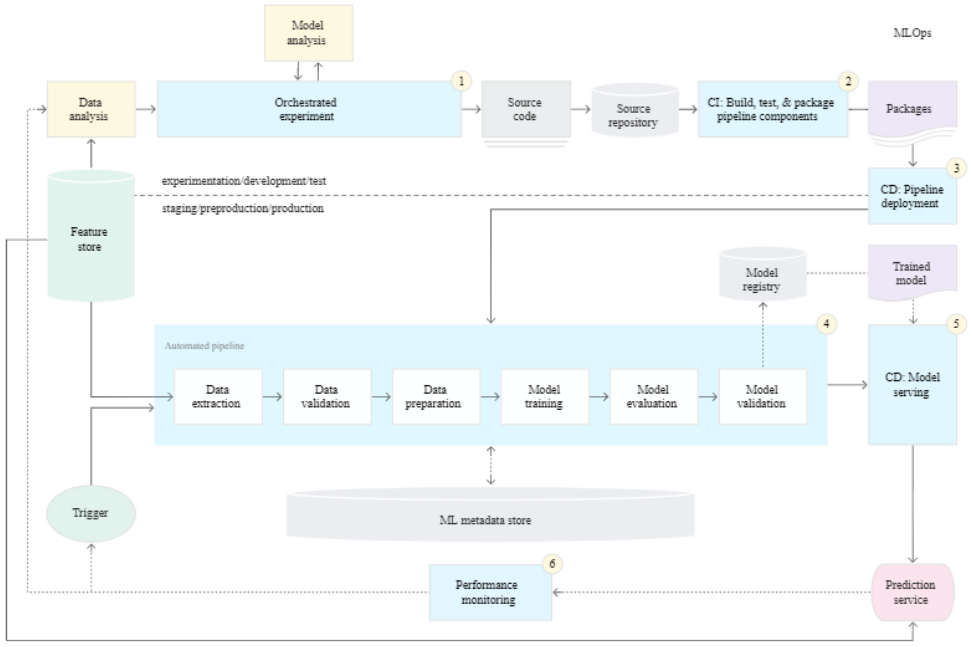
些 Data、哪些 Model，以及可以利用一個歷程檔的方式去自動抓取正在運行的程式的一些 function 的參數，又或者是在運行的時候自動直接將程式複製一份起來，並且在檔名紀錄一下 train 跟 validation 的一個準確度以及時間在磁碟當中，透過這些方式使得 OS 可以自動記錄使用過的痕跡，並且在當日後需要時，也可以提供一個搜尋的方式(如準確率達 90%以上)來做一個歸類，讓使用者可以方便查找，而不是一直手動 commit，我覺得 OS 提供這項服務是滿重要的，因為自己手動 commit 實為有點繁瑣。

Training Model Costs 和 平行訓練 Model 的工作流程(這邊一起講): 這邊就是看要來如何使 OS 去做適當的資源分配了，雖然現在常常會使用如 colab 等方式來去做，但是有時候還是需要 OS 設計方式來達到更有效率的做法，而這邊的話，因為 ML 可能常常會需要讀取非常大量的資料去做，而這個部分我認為在 I/O 所消耗的速度將會非常驚人，所以我認為 OS 在前面的流程去檢驗 DataSet 的時候，就已經有這個 DataSet 的一個獲取管道了，而這邊可以使 OS 去查看 MEM 的一個使用量，如果 RAM 很空的話，讓 OS 可以使得 DataSet 利用 key-value 的一個方式來讀進 MEM 中，這樣子可以去減少一些要開始使用時的一些 I/O 的時間，可以使得能夠快

速地 access 到 Data 並且 training，而如果在 CPU 及 GPU 在閒置的話，應該使 OS 將其分配工作讓他們去做資料的一個分析以及整理，如果剛剛有 training 過東西可以整理一下剛剛的資料(如建立 Hash table, 紅黑樹)等方式，讓資料能夠更快速的在下次要使用時能夠被找到，而在 validation 的時候也能夠透過此方式來更快速的獲取到資料。而在這邊在排程的部分，如果是有很多 user 以及有個裝置要使用的話，考慮到每個 user 要使用的時間的不同，可以利用類似於 ROA(Random Optimization Algorithm)的概念以及 IA(Intelligent Agents)的一個概念(參考資料[2])，雖然這是比較偏向工業用的，但我認為如果 OS 能夠偏向這些概念去做設計的話，可以在很多人跟很多機器要一起共同開發的時候可以讓 OS 給設備去做出良好的排程的方式，並且在如果同時沒有很多 user 在使用時，可以讓一些能夠做平行處理的部份去做平行處理。而至於在 OS 中 process 的排程的部分，可以設計類似程像是使用 genetic algorithm(基因演算法)來實現，可以去評價 process 在這邊像是如適應度的一個方法來去決定說哪些應該要先去實現的，並且可以去淘汰掉一些不適合的 Data，可以利用這個概念來設計 OS，來當作其 process 的排程演算法(參考資料[6])

總結來說我認為在流程上，可以使 OS 來去提供這一些服務來給

MLOps 來去做使用，能夠更有效率的去運作，並且在 MLOps level 2: CI/CD pipeline automation(如下圖)(參考資料[7])



的某些部份，這些 OS 的設計也可以為一些功能來達到更適合的加速，也是比較適合 MLOps 的設計。

參考資料:

[1] https://shazi.info/%E4%BB%80%E9%BA%BC%E6%98%AF-mlops-%E4%BB%A5-machine-learning-%E8%A7%92%E5%BA%A6%E7%9C%8B-devops-%E6%BD%AE%E6%B5%81/

[2] http://lawdata.com.tw/tw/detail.aspx?no=357079

[3] https://towardsdatascience.com/5-ways-to-detect-outliers-

that-every-data-scientist-should-know-python-code-70a54335a623

[4] https://www.itread01.com/content/1569859263.html

[5] https://kknews.cc/zh-tw/code/6oeqe4m.html

[6] https://zh.wikipedia.org/wiki/%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95

[7] https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[8] https://zh.wikipedia.org/wiki/RSA%E5%8A%A0%E5%AF%86%E6%BC%94%E7%AE%97%E6%B3%95

[9] https://medium.com/%E5%BC%B1%E5%BC%B1%E9%96%8B%E7%99%BC%E5%A5%B3%E5%AD%90-%E5%9C%A8%E6%9D%B1%E4%BA%AC%E7%9A%84%E9%96%8B%E7%99%BC%E8%80%85%E4%BA%BA%E7%94%9F/%E5%B0%88%E6%A1%88%E7%B6%93%E7%90%86-

%E9%AB%98%E9%9A%8E%E7%AE%A1%E7%90%86%E4%BA%BA%E9%83%BD%E6%87%89%E8%A9%B2%E8%A6%81%E7%9F%A5%E9%81%93%E7%9A%84-%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92%E5%B0%88%E6%A1%88%E7%9A%84%E5%9F%BA%E6%9C%AC%E6%B5%81%E7%A8%8B-e9d6564c0462