報告

黃允誠
fabin1790@gmail.com

***Abstract*—我盡力的想 match 到這個 template，如果真的差太多請多包涵…這次作業當中我使用的一些提升 performance 的方式，只是我自己憑之前對程式語言的理解加上自身的邏輯思考能力東拼西湊、並在 Kaggle 上測試確實能提升效能的改動。我合理猜測這些應該都是十分非主流、甚至可以說是不入流的作法，而且跟 AI 應該也沒有太大的關係…總之這些之後再慢慢學吧。**

Keywords—吃老本, 怎樣好怎樣做, 菜鳥胡鬧

I. Introduction

首先我從學長助教課的示範程式碼開始，先馬上備份免得被我搞壞，畢竟這是第一次碰 Python。當然了，光直接把助教的程式碼改一下或許真的也能交作業，不過當然不可能真的這樣做。

第一件事是讓自己徹底理解學長的 code 每一行在做甚麼，基本上就是讀 code、思考、理解、查文件、理解、煩助教(抱歉…)。把整個 code 搞懂大概花了整個作業一成多一點的時間。

第二件事，就是改 code。完全理解之後改起來就輕鬆很多，用我自己的邏輯，把 code 慢慢改成可以交差這次作業的 code。就是基本的底標，把 data 讀進去，然後可以輸出結果，符合 Kaggle 要求的格式，上傳成功。

至少可以交出東西，就是我已經大概有理解這個架構的佐證。最後就是提升 performance，其實就是在衝榜，主要是我想知道我這個菜鳥到底有幾把刷子，可以讓我至少在榜上不要看起來那麼可悲，分數拿得漂不漂亮再說…

II. 原始架構

A. *讀取資料*

主要就是 multi-hot encoding。本來的資料，標記全部擠在一起，最重要的重點是：「中學、高中、大學」的標記和「高中、大學」的標記，在機器眼中看起來是完全無關的！這是非常致命的，所以才需要透過 multi-hot encoding 把標記分開獨立，這樣機器才有辦法學習其中的關連性。

B. *資料前處理*

這邊主要就是結巴斷句，然後用 vectorizer 轉成向量。其他的就是一些迴圈串接資料的動作而已。

C. *訓練/測試效能*

因為我一開始想說 Kaggle 是直接上傳結果的，也不知道有次數限制，在改 code 的時候就直接把 evaluation 的地方刪掉了。事後回想起來，要做 5 倍交叉驗證那些東西的話，應該在這邊做。

III. 改 code

這邊就稍微提一下，把學長的示範程式碼改成這次作業用的程式架構時，幾個主要的改動。

我並沒有留下這個剛改過來的版本，只有留下最終的程式碼。沒有想到這一點，抱歉…

A. *讀取資料*

這邊最大的改動，是我把 multi-hot encoding 移走了，移到了下面的資料前處理那邊。因為我理解之後，覺得資料本來讀進來不是長這樣，所以 multi-hot endcoding 應該歸類為一種前處理的方式，還有把漏標記的資料刪除也是一樣的道理。這是我自己的想法，也不知道有沒有搞錯甚麼就是了…

B. *資料前處理*

這邊關於 multimulti-hot encoding 跟示範程式碼還有一個最主要的差別，那就是：示範是對測試資料做 multi-hot encoding，這次作業反而是要對訓練資料做 multi-hot encoding。這就是源自資料格式跟要處理的 task 的差異而已。

剩下的部分就跟示範的大同小異，一樣結巴然後 vectorizer，我也沒能力改成別的東西…

C. *訓練/測試效能*

前面有提到，這邊在改的時候我就想著輸出了。具體方式就是先把輸出結果的矩陣用迴圈一塊一塊拼成 Pandas 的 DataFrame，就可以直接用 to_csv 輸出成 csv 檔了。然後參數傳入 header 跟 index 為 false，可以直接把兩條 index 軸去掉，符合 Kaggle 要求的格式。

然後這邊主要還是直接引用 sklearn 的 model，我有試著去看他們的文件，可是老實說真的看不太懂…

IV. 重大改良

這邊就是紀錄幾個比較重要，也就是提升的幅度特別大，且牽涉的規模比較廣的改良。

A. *Model*

這個是最直覺的，畢竟都有這麼好用的工具人家都幫你寫好了，而且我記得很清楚，在助教課上學長有特別說到，組員可以分別嘗試各種不一樣的 model。

但是直接一個個試根本就沒什麼用還浪費時間。雖然完全不清楚這些 model 是甚麼，我還是試著去看 sklearn 官方的文件，找到了所謂的「multi class labeling」，我認為這應該就是我們這次作業的意思了。所以就決定從文件上提到支援這個 task 的那些 model 裡面挑幾個教授上課好像有印象的來看：

- 第一個就是挑了 MLP。其實我不知道這是甚麼，只是看到另一個關鍵字，就是 NN。在這之前我曾經修過一門比較接近這主題的選修課，那門課是非常非常入門的，基本上就是稍微的講解了 NN，於是當然就先試試看了。結果不出所料，超級悲慘…不只沒有很準，training 還要等超久！當然這也很有可能是我不會調參數，不過看了文件，參數實在太多了，當中也包含層數、大小、激活函數等等。在沒什麼概念的情況下，根本連該往哪

邊調的大方向都不明白，簡直無從下手，最後還是只能放棄。

- 第二個就看到了 KNN。我不知道我有沒有記錯，但我好像有印象教授在講 KNN 的時候有提到說這次作業蠻適合之類的話。我是這樣理解：因為我們把活動的描述轉成向量，而 vectorizer 會傾向把類似的字串轉到對應的距離較接近的向量，所以 KNN 取接近的幾個向量的方式，就非常契合這次作業的情況。最一開始的時候，其實沒有很明顯的提升，大概是比 MLP 好一點點而已。不過速度倒是狠甩 MLP，幾乎不需要等，跑 KNN 訓練的時間都快跟前處理差不多了。問題就是一樣，因為沒有調參數。但是 KNN 的參數沒有那麼多那麼複雜，這次去文件看就稍微有看懂一點了。首先最重要的是權重：如果沒有設定，權重預設會是平均，也就是接近的那幾個鄰居直接平均，非常的莫名其妙。但是改成距離越近權重越大之後，就可想而知的有了一定的提升。第二就是 KNN 的這個 K 了，經過測試大概在 4 是比較好的。不過最終這個仍舊不是最好的 model。
- 最後試到的 model 是所謂的 extreamly randomized trees classifier，我引用的這個版本有實作「ensemble」，在直接引用都不調任何參數的情況下，它已經直接輾壓了我前面辛辛苦苦調半天的 KNN 了…因為是隨便點進去看的，所以也是有點運氣成分。至於「ensemble」是甚麼最後再說。

B. *Kappa*

這邊開始就要進入比較…就像我說的，「胡鬧」的階段了。雖然說是胡鬧，這些胡鬧跟模型本身的核心概念是沒有太大的關聯，但是之所以會有提升是有它的道理在的，我這好歹是「有策略的胡鬧」。

說到 Kappa 應該就知道是學長在標記作業叫我們算的 Kappa 值。學長們還特地的提供了每一組別的 Kappa 值，甚至叮囑大家，要自己斟酌把 Kappa 值過低的組別標的資料給過濾掉，避免影響模型的學習。

我們關注 Kappa 值最原始的目的，就是要衡量標記的品質：如果兩個人有高度的共識，兩個人的標記同時有問題的機率就很低。但是仔細想就會發現，這個做法其實很不直接、不細膩。打個比方，今天有一組的 Kappa 值很高，大家可能會覺得他們標得很好，但是萬一他們其中有一筆其實標得很爛呢？反之同理，因為某組的 Kappa 值太低而放棄他們的標記，但要是其中有幾筆其實標得很好，那也很可惜。

這樣說答案應該已經呼之欲出了：我們關注的是資料到底標記得好不好，而不是負責標記資料的人標記得好不好。因此最好的做法，理所當然，就是對每一筆資料都考慮一次 Kappa 值。雖然聽起來可能很奇怪，但是將學長們教的 Kappa 值定義轉個角度看，對每一筆資料，把八種類別當成是八筆資料，就可以算出兩個標記在這單獨一筆資料上的「Kappa 值」。雖然我不知道我這樣搞定義上還能不能算是 Kappa 值…暫且就稱它為「共識度」吧！

對所有的資料算完之後，就很直觀了：選個 threshold 把所有共識度太低的資料都扔掉。這部分的 code，我有放在程式碼最一開始第一個 block「備份」裡面，如果學長有興趣又有耐心的話…因為我沒空給我的程式碼加註解…

那麼為甚麼備份起來而不用了呢？其實就是剛才說到的 threshold，當然是可以調整的。然而調整之後，我發現 performance 最好的情況，就正好是 threshold 設定 1 的情況，也就是只留下共識度為 1 的資料，其他通通都扔掉。

這樣一來就有點尷尬了，因為根據定義，這個值是 1 的情況，就只有在兩個標記正好從頭到尾一模一樣的情況下。也就是說其實只要對每筆資料查看，兩個標記一樣就留下，有任何一類不同就扔掉。那 code 一定是簡潔一點比較好了。

其實在改完這一段之後，我已經衝到第一了。就這個結果看來，雖然這不是甚麼多進階的技術，但說不定只有我特別考慮到，或是至少可以說，有想到的人之中，應該只有我真的去實踐，因為提升幅度不小。

但這時我跟第二名的差距非常的小，而且這當中有很多環節有運氣成分，加上我又是個甚麼都不懂的菜鳥，說是矇中的都不為過。同時作業的時間還有，所以我不會停下。

接下來的「改良」，是我覺得最浮誇、最會讓人覺得「你怎麼可以這樣」、最讓我心虛的一項了。因為這根本就不是模型，而是…

C. *N 個模型*

其實我覺得在看報告的人應該已經知道我要說甚麼了…顧名思義，最後選擇的的這個模型是「極度隨機」的模型。一開始跑的時候，每次跑的輸出都不一樣，一度讓我很困擾，我其實不太喜歡運氣成分太高的感覺。

幸虧這樣的模型，sklearn 有提供將亂數種子作為參數指定的方式，大家都知道電腦中大部分的隨機其實都不是真隨機，指定亂樹種子之後輸出就固定下來了。

然而這又衍生出另一個問題：究竟亂數種子要設定多少？其實嚴格來說，問題就是這個問題的本身。因為是亂數種子，在概念上，它根本就是不存在的數值，重點是你不管怎麼調整，都跟 performance 毫無關係。但是偏偏你一動它，performance 又會被影響，我前面說我很不喜歡有運氣成分的感覺就是這個意思。

而就是在我困擾的時候，忽然就腦洞大開了。它極度隨機，也就是說它極度不穩定。那要怎麼讓它穩定一點呢？這個問題其實就是：要怎麼更加確定一粒骰子真的是公正的呢？答案不言而喻。

我跑一次，得到一個不穩定的答案。那我跑三次，每次給不一樣的亂數種子，就得到三個答案。這邊注意：亂數種子是在建立 model 的時候就要指定，所以我其實是建立了三個 model，各訓練一次、各預測一次。

這三個答案是同樣大小的二維矩陣，上面的每個元素代表對某比特定活動，是否適合某個特定族群的標記。標記只有 0 跟 1，於是對上面每個位置，將三個答案採用「多數決」，便得到了我最終的預測結果，把這個結果上傳，不出所料又有了一段提升。

程式是沒有寫死的，最後的差別就在於這個嘗試的次數，只要是奇數都可以。但這其實不用經過甚麼測試，可想而知隨著次數提升，performance 只會跟著穩定的提升。真正的問題在於，隨著次數提升，我要等待的時間也跟著穩定的提升。而且更糟的是，執行時間是線性的提升，但 performance 的提升會越來越趨緩。

那麼究竟要跑多少呢？我最後選擇：3001 次，然後去做其他的事情。因為我們的作業並沒有特別關注訓練的速度，這個方法其實是有點在佔便宜的。

那這個方法的道理在哪裡呢？這留到最後來解釋。

D. *忘了參數*

最後這個就是比較正常的了。我忙著在搞這些 code，結果忘了參數的事情。在寫報告的時候，因為想再了解一下最後用的樹狀決策模型，所以去找了資料，又想起來了。有些資料有提到，如果 feature 很多，訓練出來的樹可能會超級大，就會有 over-fitting。那 sklearn 這個模組有提供一個參數可以限制樹的高度，調整了一下，到 200，最後又提升了一點。

V. 最終程式架構

這邊簡單說明一下最終版本的程式碼。

A. *引用函式庫*

只是給這個 block 也加上一個名字而已，沒什麼特別的。比較不一樣的地方是 LABELS 定義的部分，根據學長指教的比較 robust 的資料偵測方法，而改成只記錄每個標籤的關鍵字。

B. *讀取資料*

這邊變得非常簡短，主要就是前面說的，我只留下直接讀取資料的片段，剩下的都移到前處理的 block。然後在 converters 宣告的地方，為了實作共識度過濾的方法，要使用到 Label1 跟 Label2 的資料，所以要加上去。

C. *資料前處理*

這邊結巴斷句接 vectorizer 的地方都沒怎麼改洞。最大的不同就是那個「multi_hot_then_sort」的函式。顧名思義就是把 multi-hot encoding 跟我自己的那個共識度過濾的方法合在一起，在 encode 的時候就順便過濾，同時記錄兩個方法要刪除掉的資料，也就是漏標記的跟標記不同的，最後一起拔掉。

D. *訓練/預測/輸出*

這邊就是預測方式改動的部分，大致上就是先跑回圈看要重複幾次，不斷的建立不同亂數種子的 ExtraTrees 模型，然後訓練並預測，所有的結果合在一起儲存成 3 維陣列。接著第二部分就是再用迴圈把所有的結果作多數決形成最終結果，這個最終結果就是一個二維矩陣。最後把最終結果依據 Kaggle 要求的格式拼成對應的 Pandas DataFrame，然後輸出成 csv 檔案。

VI. 結果討論

有兩個重點：

A. *Ensemble*

Sklearn 的官方文件有類似這樣的提示：「請注意，隨機性高的 model，一般必須在 ensemble 的情況下使用，才能有可行的效果。」

這是在我找資料的時候，注意到有另一個跟我引用的模型很像，幾乎是一模一樣的模型，點進去看到的。其實那就是沒有 ensemble 的版本。而 ensemble 也不是甚麼很複雜的概念，大意就是先把訓練資料隨機分成一些不同部分，然後用這些訓練資料的子集合來訓練一大堆的樹，所以又會被稱作「forest」。這方法的目的主要就是分散隨機性帶來的傷害，進而提升精確度並減少 over-fitting。而對於最一開始的模型概念來說，這每顆樹其實都是一個模型了，最後分類時是把所有樹的判斷結合起來，當成最終的結果。

說到這邊，是不是有似曾相識的感覺？應該不難發現我的「N 個模型法」誤打誤撞的做了幾乎一樣的事情。唯一的差異在我是用有點 tricky 的方式，每次都使用了整個訓練資料集。如果不要那麼嚴格，其實這情況可以簡單的理解成：我引用 sklearn 的這個模型時，它已經先做了一次 ensemble，而我又拿這個 ensemble 過的結果，再做了一次 ensemble。

這就是為甚麼我要等那麼久。當然了，提升也是很顯著。

B. *Feature 到底多少？*

剛才說到，Feature 太多的話，樹狀決策模型可能會 over-fitting。或許有人會覺得很奇怪：這次作業最多也就 1300 比，哪來 Feature 太多？

我基於 Kappa 值概念的改良，也同樣留下了類似的問題。只要標記稍有不同的通通都踢掉，最後留下的資料又更少了，然而我卻沒有墜入 over-fitting 的深淵。

這點我認為是這樣：我們把活動的相關資訊全部餵給結巴斷句，得到的 tokens 再餵給 vectorizer 轉成向量，最後的這個向量才是餵給模型的輸入。如果有觀察一下訓練資料，就能發現在活動敘述這一欄文字量其實非常的多。也就是說，雖然表面上看起來是 1300 比，但實際上處理完之後餵給模型的向量，當中包含的資訊量可能比想像的多很多。

這也只是我粗淺的理解，我也不敢肯定對不對，但就結果看來確實是有可能的。