K 近鄰、BP 神經網路、支援向量機、決策樹、隨機森林和梯度提升演算法在資料分類方面的統計應用

呂浩

廣東第二師範學院　數學學院

摘　要：網路已經滲透了我們的日常生活當中，同時，IT、金融等各種網路產業也在不斷地崛起，而網路金融行業的發展是最突出的。人們的生活都離不開金融，因此，網路金融業已經成為了一種潮流。相對于傳統銀行業來說，網路金融最大的優點就是它的便利，因此，網路金融的出現讓傳統銀行陷入了顧客流失的窘境。

在本文中，通過使用六種機器學習的方法，同時基於歐洲某一銀行客戶資訊資料，對銀行客戶進行預測和分類客戶流失情況。我們對該銀行客戶資訊資料進行初步描述性統計和視覺化分析，隨後根據資料分析的情況進行資料清洗、處理和類型的轉換。利用處理後的資料建立 K 近鄰模型、BP 神經網路模型、支援向量模型、決策樹模型、隨機森林模型和極端梯度提升模型，分析和評價這些模型，比較它們的評價指標，從而選取預測分類客戶流失情況的最優模型。利用接受者操作特徵曲線下的面積指標、Kappa 係數、精確率、召回率、F1 值、準確率這些評價指標對模型進行比較，根據比較的結果得出最優的預測分類模型。

在對顧客資料進行分析的基礎上，利用優化的模型對客戶進行預警，並對客戶進行歸類，以提高銀行的競爭力和影響力為目的，為銀行準確地對目標客戶進行定位，並向銀行提出有針對性的客戶策略。

關鍵字：K 近鄰模型；BP 神經網路模型；支援向量機模型；決策樹模型；隨機森林模型；極端梯度提升模型

The statistical applications of K-Nearest Neighbors, Back Propagation Neural Networks, Support Vector Machines, Decision Trees, Random Forests, and Gradient Boosting Algorithms in data classification

Lv Hao

Abstract: The Internet is closely related to people's daily life now, and various types of Internet industries have emerged, such as IT and finance. People can't live without finance, so Internet finance has become a popular trend. Traditional banks that lack the convenience of Internet finance are in the predicament of losing customers.

This article uses six machine learning methods to predict and classify customer churn in a European bank. According to the results of preliminary descriptive statistics and visual analysis, we clean, process and transform the data. We use the processed data to establish K-Nearest Neighbor, Back Propagation Neural Network, Support Vector Machine, Decision Tree, Random Forest and eXtreme Gradient Boosting models. We select the optimal model by analyzing, evaluating and comparing the evaluation indicators.

By analyzing customer information and the optimal model, we improve bank's competitiveness and influences by proposing strategies to the bank.

Key Words: K-Nearest Neighbor; BP Neural Network; Support Vector Machine; Decision Tree; Random Forest; Extreme Gradient Boosting

1 前言

1.1 研究背景

在網路的繁榮與快速發展下，互聯網與金融的融合所產生的互聯網金融業。其發展勢頭十分迅猛，而傳統銀行業面臨著關於客戶流失的重大挑戰。

從 20 世紀 90 年代開始，在互聯網資訊化熱潮的帶動下，使得互聯網金融的模式在國外得到了孕育和發展，電子銀行也隨之得到高速的發展和完善。到目前為止互聯網金融的規模在國外已經發展得相對成熟與完善，傳統金融服務也逐漸向互聯網靠近，許多國外的銀行傳統業務都開始實現在互聯網上辦理銀行業務。同時互聯網金融下的支付體系也有較大的創新發展，比如像協力廠商的新興支付方式。信用業務也與互聯網進行融合，形成網路貸款和眾籌等為代表的互聯網信用業務，通過互聯網進行操作，不需要借助線下銀行而完成，這完全脫離于傳統的銀行體系。網路上的虛擬貨幣與真實貨幣的隨意相互轉換在如今時代已經成為了可能，目前已經有相關的國家承認虛擬貨幣的法律和稅收地位，在國外使用虛擬貨幣也逐漸變成普遍現象。

近年我國的互聯網發展速度飛快，互聯網在不同行業中遍地開花，在眾多發展成果中，與生活緊緊相扣的火熱互聯網軟體是微信和支付寶。這兩個巨頭支付方式是以其支付速度快和支付操作便捷而得到眾多人的喜愛和關注，不久之後餘額寶這類金融理財軟體的出現開啟了互聯網金融市場的大門，互聯網金融行業逐漸擴大發展，金融軟體的數量和種類也呈現出了暴增的現象。在金融行業的市場中，競爭的加劇，便捷靈活和高速高效應用的高量誕生，使得銀行業處於流失客戶的危機中。

對於銀行業來講，客戶是銀行的經營根基，銀行客戶的流失將會對銀行的發展和經濟產生嚴重的衝擊。激烈的競爭將會導致客戶對銀行的忠誠度和活躍度下降，從而使得客戶的流失速度加快。因此銀行業現階段的核心任務有兩個，一個是挽留客戶和提高客戶的滿意度，另一個是增加銀行的新使用者數量和開發新目標客戶群體。當前，銀行業的核心工作就是對客戶的去留進行分析，進而對客戶的行為進行更好的瞭解和判斷，進而更好地對銀行的制度內容進行設計，並進行主動的客戶活動。

1.2 國內外研究現狀

1.2.1 國外研究現狀

1968 年，Cover T 和 Hart P 提出了 K 近鄰分類演算法，該演算法的主要功能是解決分類和回歸問題的基本難題。支援向量機演算法最初的提出者是 Corinna Cortes 和 Vapnik，他們在 1995 年首次提出這種演算法，從此在機器學習領域掀起了一股熱潮。Sundarkumar

G G 和 Ravi V 運用支援向量機模型和 K 反向鄰近分類模型相結合的方法，成功探究了客戶信用卡流失情況。他們在探究過程中進行了交叉驗證，並對比了多種學習分類演算法的效果，最終發現融合模型在此方面的表現最為出色。在 1948 和 1949 年間，Claude Elwood Shannon 發佈了資訊理論的文章，資訊理論文章的發佈給決策樹的誕生打下了基礎。最初始的決策樹演算法是 Hunt 演算法，這個演算法是在 1966 年被 Hunt 等人提出的，能用於分類和回歸的 CART 決策樹在 1984 年被 Leo Breiman 開發出來的，隨後 Ross Quinlan 在 1986 年提出 ID3 演算法。BP 反向傳播神經網路是在 1986 年由 Rumelhart 和 Meclelland 等人所提出的，它的核心是將計算出的誤差用於逆方向傳播，然後再進行訓練的。在 2001 年一個新組合分類器演算法誕生，即 Leo Breiman 提出的隨機森林演算法。Lalwani P 等人利用了 K 折交叉驗證方法，同時使用了多種常用的機器學習演算法去構建客戶流失預測模型，這幾種機器學習包括了支持向量機、決策樹等模型，通過比較和分析，最後得到了效果最好的模型是 AdaBoost 以及 XGBoost 模型，這兩個模型的分類能力最好 [1]。

1.2.2 國內研究現狀

目前對於將機器學習演算法和模型運用到銀行業客戶分析上，在我國銀行業是有較為廣泛的應用。在 2014 年的 2 月份，我國的陳天奇博士提出了 XGBoost 演算法，它是基於 CART 決策樹的一種 Boosting 演算法。在 2007 年吳運奇利用支援向量機模型中的結構風險最小化去降低預測某個商業銀行信用卡業務資料時的風險，其中他還構建了嶺回歸模型去分析和預測客戶的流失情況，最後該模型的整體正確率結果高達 86%[2]。單其帥通過創建某一個電信企業的客戶流失風險預警指標，利用粗糙集理論簡約屬性指標，使用專家打分法賦權給由指標影響客戶流失的風險，隨後創建 BP 神經網路模型對某公司進行實證分析，在宏觀層面，可以知道該公司所在的客戶流失風險等級，並說明企業在自身公司層面上把握客戶流失風險 [3]。石楊等人將決策樹貪心演算法進行了改進，並且利用這種演算法去挖掘和預測流失客戶，同時分析判斷他們的共性，再使用訓練集資料訓練出較高的整體準確率模型，最終利用這個模型對潛在的流失客戶進行預測 [1]。

2 處理資料

2.1 資料來源

本文資料是來自於歐洲某一銀行機構的 10000 條銀行客戶資料資訊。

2.2 資料和變數介紹

針對現今的銀行業市場環境，銀行的流失客戶群體不斷擴大，客戶高流失率的現狀變

成了銀行業的棘手難題。各家銀行為了挽留客戶和吸引新客戶，紛紛使出千方百計，並且制定專門策略去維護和增加客戶的數量。因此銀行會基於自身所持有的客戶資訊，對本行的資訊和狀況進行分析，從而挖掘出更多有效資訊去預測流失狀況和發掘挽留客戶的途徑，根據對客戶分析的結果制定出專門的目標客戶策略，從而更高效地降低銀行的客戶的流失率。本文採用了歐洲某一銀行的客戶資訊作為研究資料，對銀行客戶的流失進行預測和分類。該資料中存在著 14 個變數，其中將 13 個變數劃分為特徵變數，將 1 個變數劃分為目標分類變數。假設這 13 個特徵變數對銀行客戶的流失存在著影響，對 14 個變數進行基本的分類：影響流失較小的因素，客戶的基本資訊，與本行相關的資訊，銀行流失情況。

表 2.2.1 14 個變數的分類表

分類	特徵變數
影響流失較小的因素	行號，客戶 ID，客戶姓氏
客戶的基本資訊	客戶所在區域，客戶性別，客戶年齡，客戶的薪資，客戶是否持有信用卡
與本行相關的資訊	帳戶餘額，活躍度，購買銀行產品的數量，客戶在本行的年限，信用評分
銀行流失情況	客戶是否流失

2.3 處理和分析資料

2.3.1 資料描述性統計分析

對歐洲某一銀行的客戶資訊資料進行初步的探索分析，可以得知該歐洲某一銀行客戶的資訊資料有 10,000 條，資料的條數比較多，因此在後續對資料集進行劃分訓練集和測試集的時候採用 70% 訓練集和 30% 測試集的模式。使用 7:3 的劃分方式，可以使以後所構建的模型的分析效果更好，70% 的資料被用來構建模型，30% 的資料被用來驗證和修正模型，這樣既可以減少過擬合的幾率，預測精度更高。

在變數分類中，由於影響流失較小的因素變數對後續的研究並沒有太大的意義且可能會產生干擾，因此先將影響流失較小的因素變數從特徵變數中刪去，再對刪去的資料進行後續的研究分析。對資料進行缺失值的統計後，發現並沒有存在缺失值的情況，但在統計查找文字變數的時候，得知“Geography”和“Gender”這兩個變數是文字型變數，需要在處理資料時將資料類型轉換。在描述性分析中可以得知年齡、信用分、帳戶餘額以及客戶工資這四個變數是連續型變數，因此要在後續的資料處理中進行離散化。

對銀行客戶的流失情況開展單變數分析可以瞭解本行中整體客戶流失的集中趨勢，結果表明約有 79.63% 的客戶是沒有流失，銀行客戶流失佔據的比例為 20.37%。本文以 K 近鄰模型、反向傳播神經網路模型、支援向量模型、決策樹模型、隨機森林模型和極端梯度提升模型這六種機器學習模型為基礎，對銀行客戶進行預測和判斷是否流失。

2.3.2 資料分佈視覺化分析

初步對銀行客戶進行探索，由於圖形往往能更直觀地表現出資料的趨勢，從而更好體現各個特徵變數之間的聯繫，所以將資訊資料進行基本視覺化分析和描述性統計分析，同時對資料的各個特徵進行統計圖表化。利用所得到的統計圖表進行觀察研究，從而分析和推斷每個特徵變數之間存在的顯在和潛在關係。在對銀行客戶是否流失這一分類下，對特徵變數分別進行單變數和多變數的基本分析，判斷是否對銀行客戶流失產生影響。

（1）箱線圖分析

箱線圖表對 4 個連續的變數進行了視覺化，因此可以看到離群的資料。

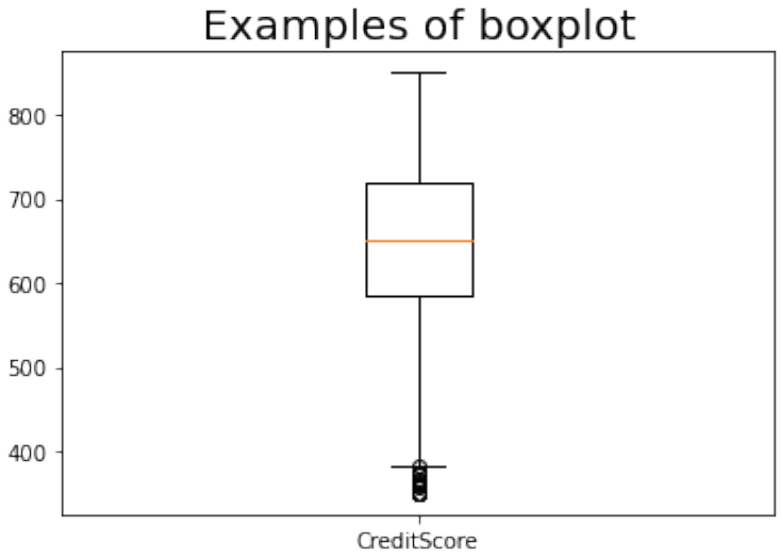


圖 2.3.2.1 信用得分箱線圖

由圖 2.3.2.1 銀行客戶信用得分箱線圖得知，銀行客戶“CreditScore”資料中存在異常值，當客戶的信用得分低於 400 分時，屬於異常值。

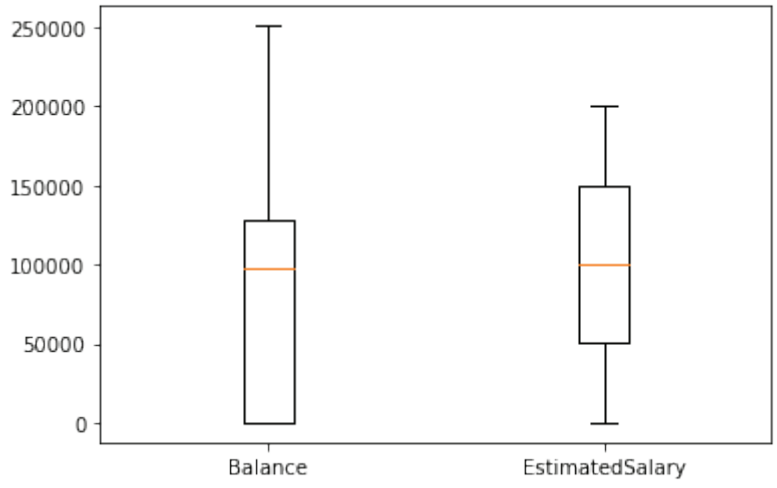


圖 2.3.2.2 帳戶餘額和客戶薪資箱線圖

由圖 2.3.2.2 銀行客戶帳戶餘額和薪資箱線圖得知 銀行客戶“Balance”和“EstimatedSalary”這兩個資料中並沒有存在有異常值。

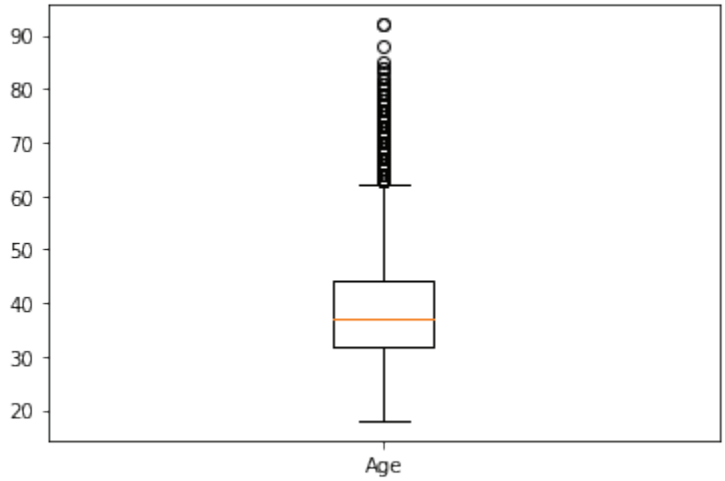


圖 2.3.2.3 年齡箱線圖

由圖 2.3.2.3 銀行客戶年齡箱線圖得知 銀行客戶“Age”資料中高於 60 歲的為異常值。

因為原始資料規模龐大，四個連續變數對客戶流失率有較大的影響，所以我們在對客戶流失率進行分析時，不考慮這些異常值。

（2）單變數分析

接著對銀行客戶中的性別、地區分佈、是否手持有信用卡、年齡段分佈以及估計薪資水準分佈進行單變數的基本分析。

下圖 2.3.2.4 為銀行客戶中性別的統計占比圖：

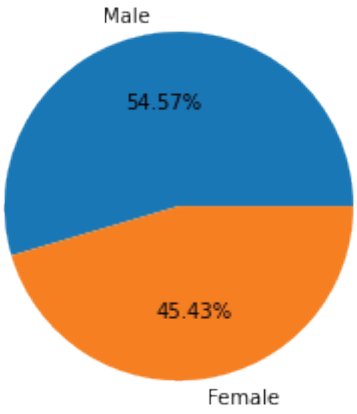


圖 2.3.2.4 性別的統計占比扇形圖

此銀行客戶的女性客戶占比為 45.43%，男性客戶的占比為 54.57%，因此銀行的客戶主要是男性客戶為主 該銀行可以重點關注和分析男性客戶 ,以降低男性客戶的流失。此外，由於男女客戶的比例相差不大，通常情況下，女性顧客的購買力要比男性高，因此，可以

考慮對於女性有關的激勵制度，這樣才能更好地減少客戶的流失，增加客戶人數。

如圖 2.3.2.5 為銀行客戶分佈各個地區的統計占比：

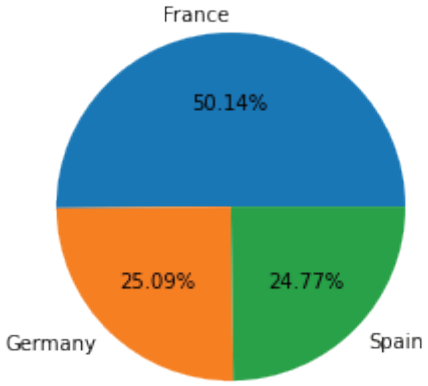


圖 2.3.2.5 分佈地區的統計占比扇形圖

在此銀行客戶中主要分佈的地區為法國、德國和西班牙，其中法國客戶佔據 50.14%，德國客戶佔據 25.09%，西班牙客戶佔據 24.77%。由此可見法國客戶在此銀行中是重要的客戶，佔據了一半的銀行客戶，該銀行在未來可以考慮法國地區多設立多間分行。而且德國和西班牙的客戶加起來，同樣可以成為這家銀行今後重點發展的對象，因為這兩個國家客戶所占的比例相當。

下圖 2.3.2.6 為銀行客戶是否手持有信用卡的統計占比：（1：有信用卡，0：沒有信用卡）

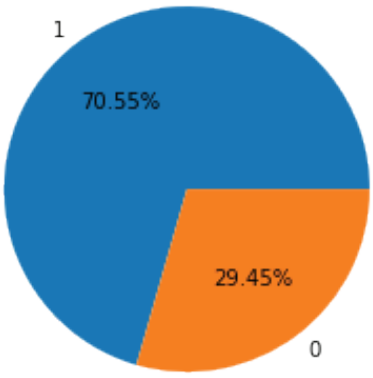


圖 2.3.2.6 手持信用卡統計占比扇形圖

在該銀行，70.55% 的顧客手裡有信用卡的，而無卡的顧客大約有 29.45%。由於信用卡的持有可以增強與銀行的聯繫，能夠降低客戶流失，鞏固銀行與客戶之間關係，所以可以未來考慮制定有關的信用卡激勵活動，比如辦卡禮品和辦卡優惠。

下圖 2.3.2.7 為銀行客戶年齡段分佈的統計：

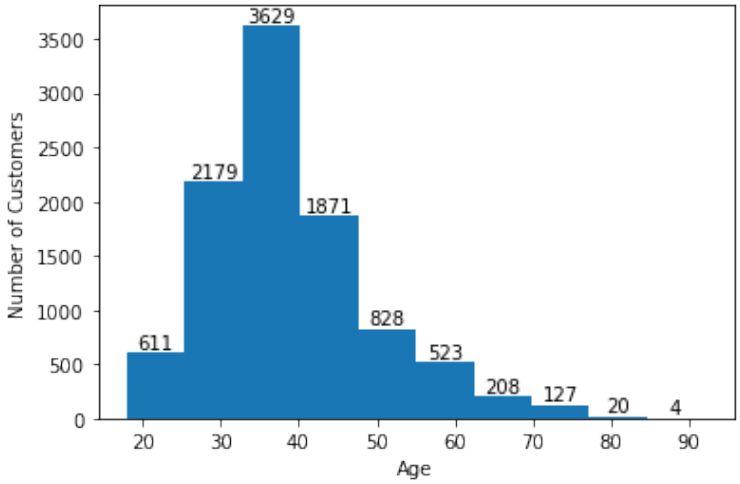


圖 2.3.2.7 各年齡段統計柱狀圖

從圖表中我們可以看到，25-45 的客戶占 76.79%，這是銀行客戶的主要年齡分佈，其中年齡分佈在 35-40 歲的客戶為銀行的重要客戶，同時年齡分佈在 25-35 歲和 40-45 歲的客戶平均各占了約 20%。由此看來 35-40 歲的客戶可以作為重點關注挽留的客戶，且可以拓展年齡分佈在 25-35 歲和 40-45 歲的客戶範圍，成為目標發展客戶。下圖 2.3.2.8 為銀行客戶的估計薪資的各水準的統計：

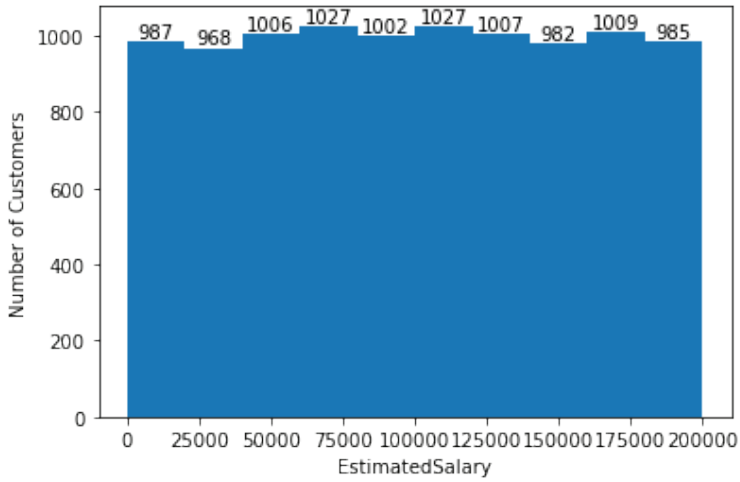


圖 2.3.2.8 各薪資水準的柱狀圖

在此銀行客戶中每個薪資階段的人數的分佈都比較均勻，但是可以考慮針對薪資較高的客戶，制定相應的活動去激勵薪資較高的客戶參與銀行的金融活動等。

（3）雙變數分析

在目標分類下，對多個特徵變數進行統計圖表分析，下圖 2.3.2.9 為在銀行客戶是否流失下客戶信用卡得分的統計分析圖表。（Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

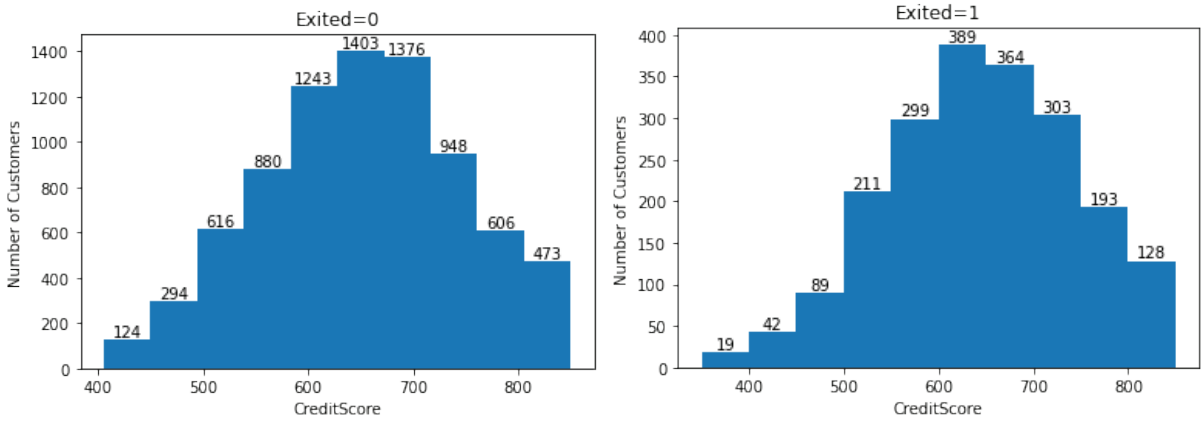


圖 2.3.2.9 信用卡得分柱狀圖

由此方形圖可以得知信用卡得分在 600-700 之間的銀行客戶最多，但是在這個範圍內的流失客戶數量也是最多的。

下圖 2.3.2.10 為在銀行客戶是否流失下銀行客戶所分佈的區域的統計分析圖表：（0：西班牙，1：德國，2：法國；Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

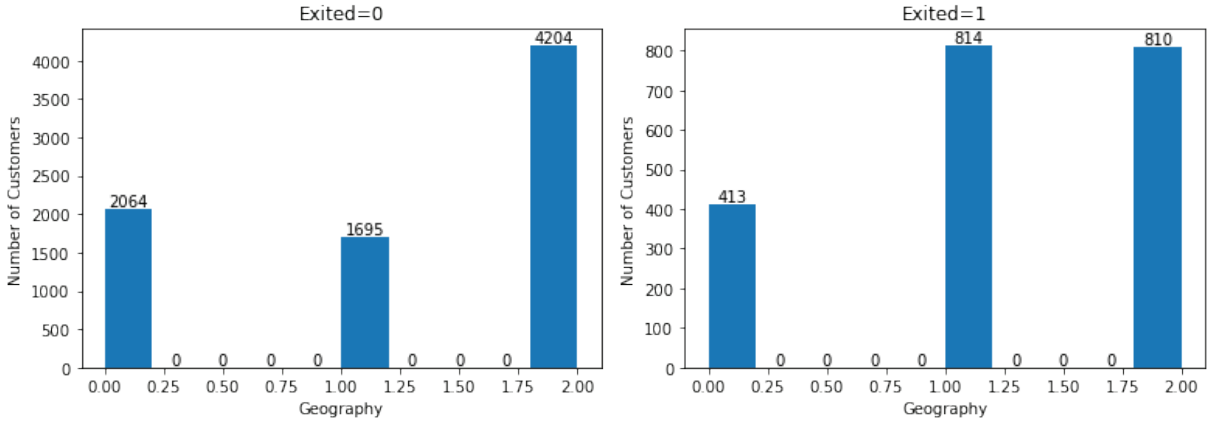


圖 2.3.2.10 區域分佈人數柱狀圖

此圖體現出法國客戶的留存率是比較高的，流失率最高為德國客戶，通過計算可以得到，32% 左右的德國銀行客戶都已經流失，而法國和西班牙各國流失率大約為 16%。由此可以推斷出銀行客戶分佈在西班牙和法國的留存率較高，屬於穩定客戶，而德國的客戶屬於不穩定客戶，銀行應該制定相關的挽留客戶的策略。

下圖 2.3.2.11 為在銀行客戶是否流失下的銀行客戶的性別統計分析圖表:（0:男，1:女; Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

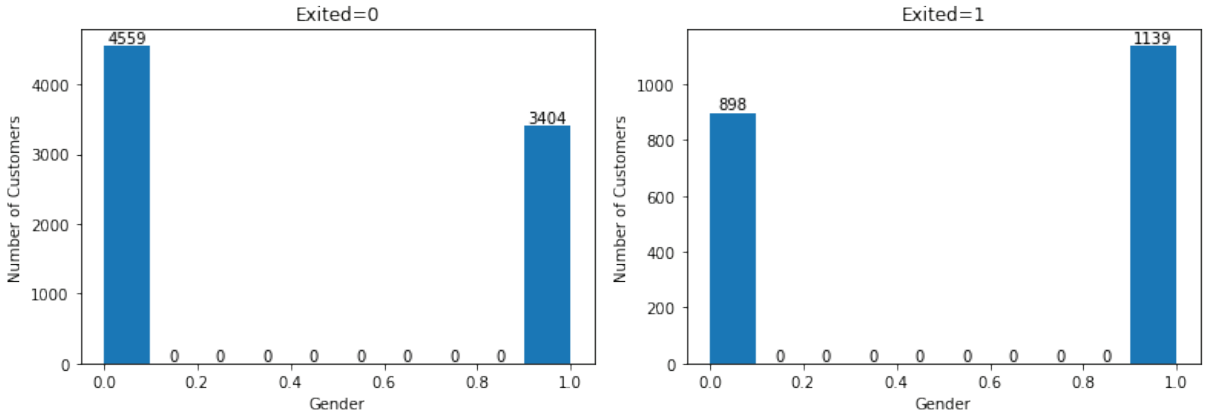


圖 2.3.2.11 男女性別人數柱狀圖

由上表可以看出來該銀行的女性客戶的流失率比男性客戶的流失率高的，由計算可以得到女性客戶的流失率在 25% 左右，而男性客戶的流失率在 16% 左右。女性客戶的流失率遠高於男性客戶的流失率，同時考慮到女性客戶有一定較強的消費能力，所以該銀行應該多關注女性客戶，降低女性客戶的流失率。

下圖 2.3.2.12 為在銀行客戶是否流失下銀行客戶的年齡段分佈的統計分析圖表。（Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

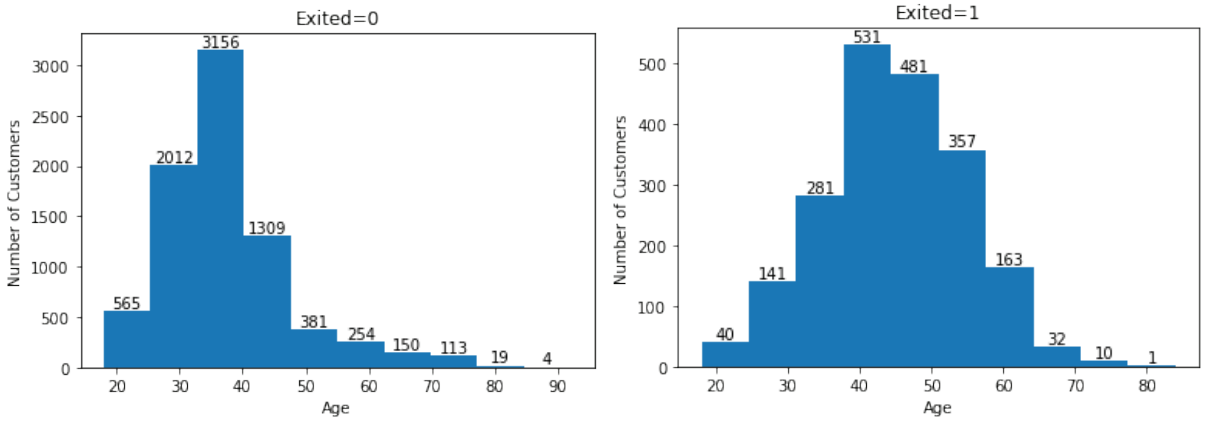


圖 2.3.2.12 各年齡段分佈人數柱狀圖

銀行客戶分佈在 25-40 歲年齡段的客戶留存率是最高的，而在客戶年齡分佈在 40-55 歲之間的流失率是最高的，在 25-40 歲年齡段的客戶對於金錢的管理和存儲是有一定講究的，可以對該年齡段的客戶進行調研，分析和總結他們的金錢管理意向和方式，以便今後在該年齡段群體開展金融活動。

下圖 2.3.2.13 為在銀行客戶是否流失下的銀行客戶在該銀行的年限統計圖表分析。（Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

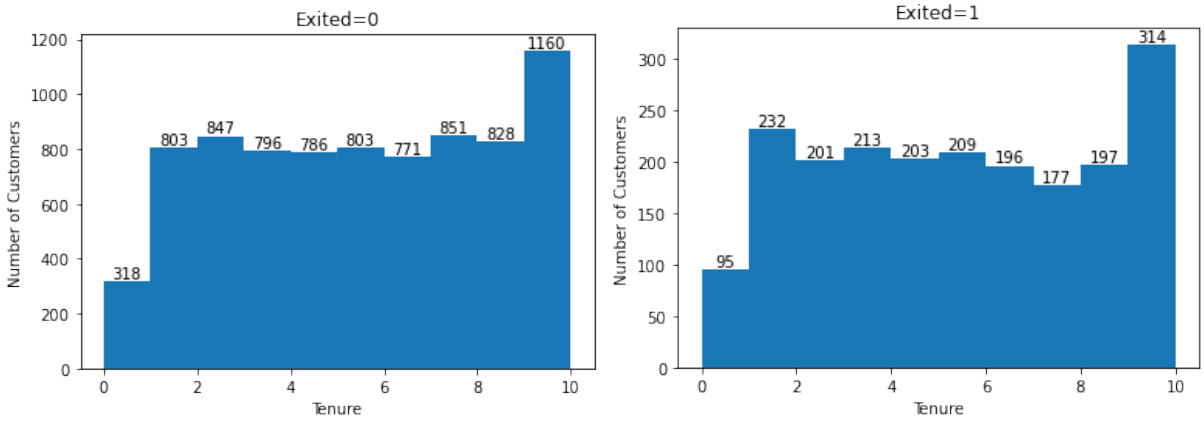
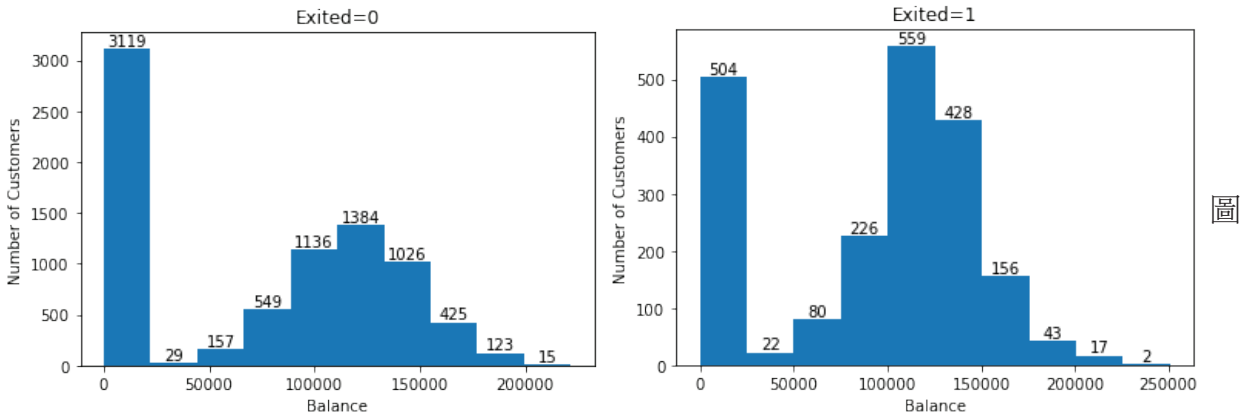


圖 2.3.2.13 在銀行的年限人數柱狀圖

在此銀行中的年限為 1-10 年的客戶占主要，其中銀行年限為 1-9 年間的客戶數量均勻。銀行年限在 1 年以下的客戶比較少，銀行年限為 9-10 年的客戶更多且流失率更低。由此可以看出在該銀行的年限較長時，客戶流失的意向就會相對較弱，所以該銀行的老客戶為主要客戶。

下圖 2.3.2.14 為在銀行客戶是否流失下的該銀行客戶的帳戶餘額的統計圖表分析。（Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）



圖

2.3.2.14 客戶的帳戶餘額柱狀圖

在一家銀行裡，大部分客戶的帳戶餘額在 0-25000 間，而客戶流失比率最高的帳戶餘額範圍是在 100000-150000 間。

下圖 2.3.2.15 為在銀行客戶是否流失下銀行客戶購買該銀行產品數量的統計圖表分析。（Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

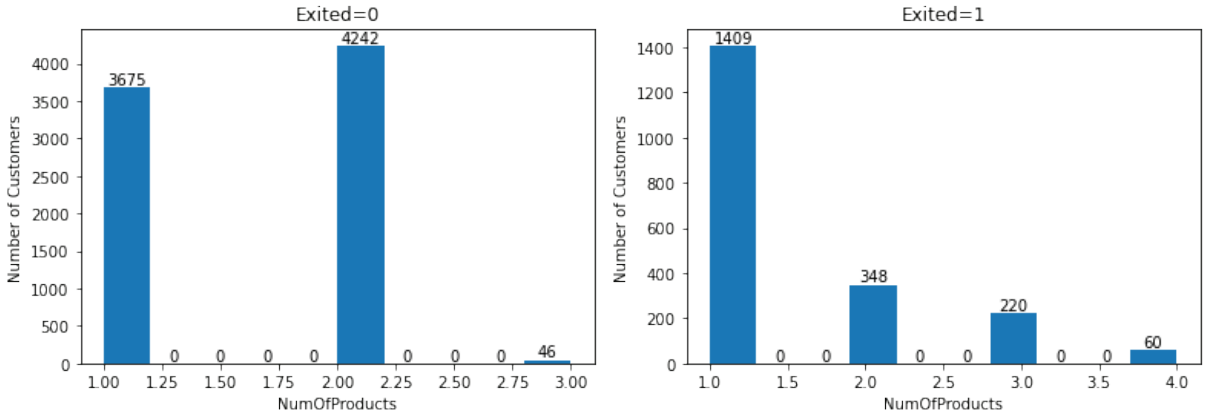


圖 2.3.2.15 客戶購買的產品數量柱狀圖

大多數銀行客戶會購買 1 到 2 個銀行產品，但是購買 1 個銀行產品的客戶比較容易流失，而購買 2 個銀行產品的客戶留存率會更高，擁有 3 個及以上銀行產品的顧客數量相對比較少，並且有很高的流失率。從中可以得出，顧客只對少數幾款產品產生了濃厚的興趣，大部分的銀行產品對他們的顧客沒有太大的吸引力。

下圖 2.3.2.16 為在銀行客戶是否流失下銀行客戶手持信用卡的統計圖表分析：（0：沒有信用卡，1：有信用卡；Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

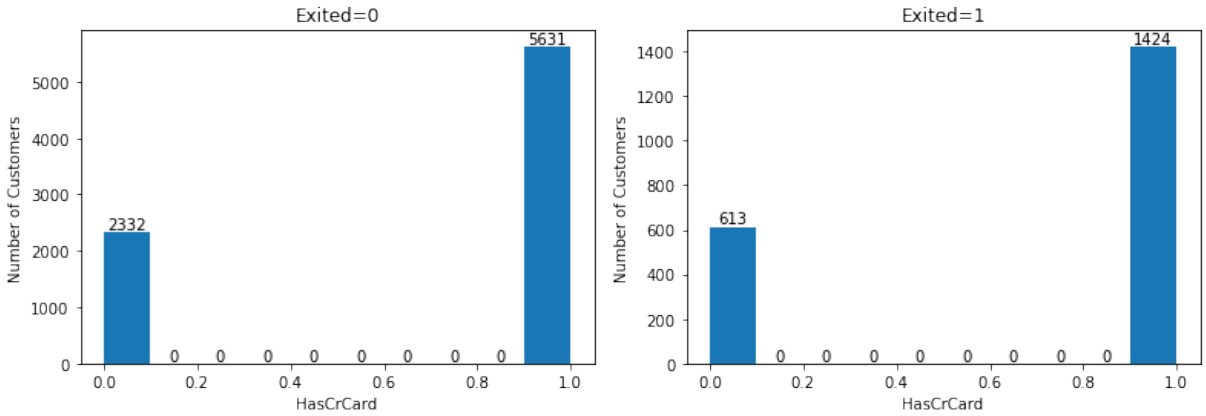


圖 2.3.2.16 客戶手持信用卡數量的統計柱狀圖

無論是在有銀行卡還是沒有銀行卡的情況下，客戶的流失率都大約在 20%，但大多數的客戶都是持有銀行卡。

下圖 2.3.2.17 為在銀行客戶是否流失下客戶活躍度的統計分析圖表：（0：不活躍，1：活躍；Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

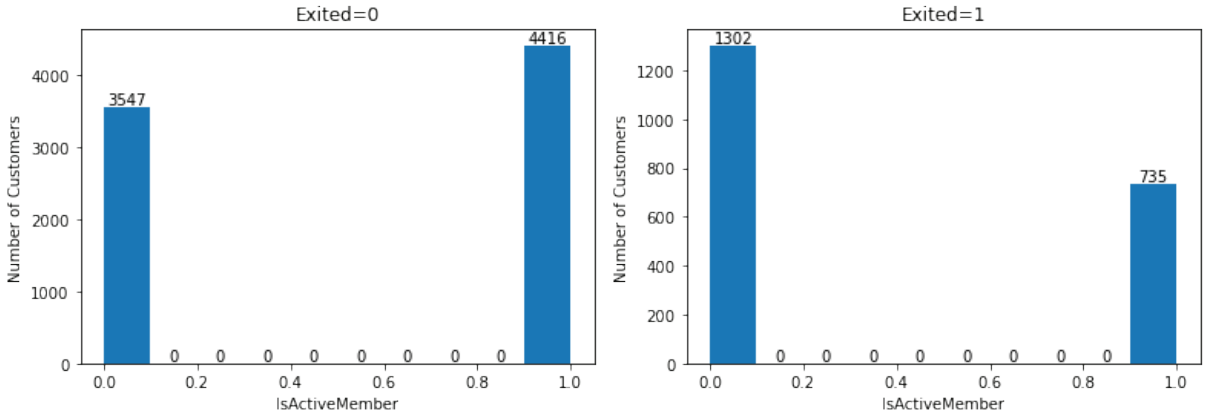


圖 2.3.2.17 客戶活躍度柱狀圖

銀行客戶中的活躍客戶數量比不活躍度的客戶數量多，銀行的活躍客戶的流失率在 14% 左右，而不活躍客戶的流失率在 27% 左右。因此，在判斷客戶是否會離開的時候，客戶的活躍度是一個很重要的指標。因此，在今後，銀行可以將注意力集中在客戶的活躍程度上，並以此為依據，對相應的策略進行調整。

下圖 2.3.2.18 為在銀行客戶是否流失下的客戶各工資段分佈人數統計圖表分析。（0：不活躍，1：活躍；Exited=0 表示為客戶未流失，Exited=1 表示為客戶流失）

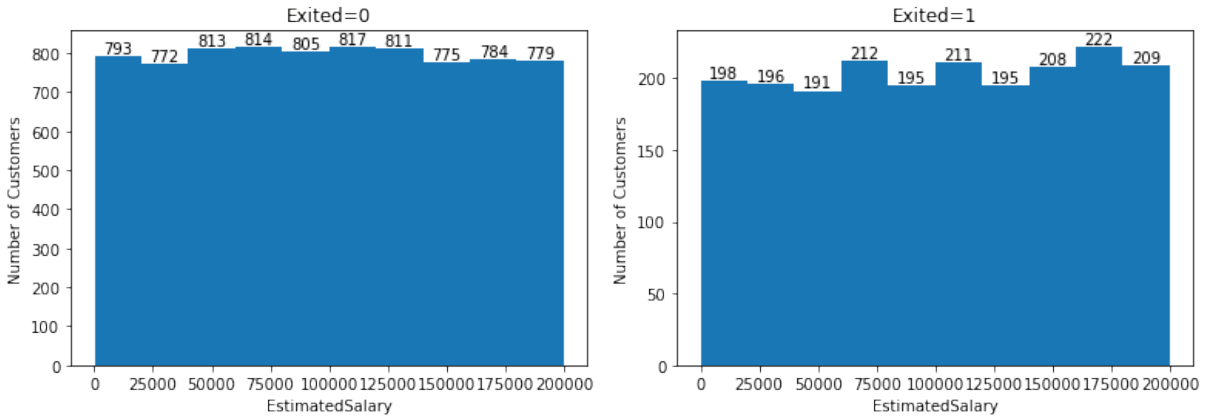


圖 2.3.2.18 客戶工資段分佈柱狀圖

從圖 2.3.2.15 可以得知，銀行客戶的薪資分佈比較均勻，在每個統計區間中流失率基本在 20% 左右。該銀行面向各工資段的客戶時，沒有針對性的調查客戶需求和制定專門的策略，並且沒有對某一範圍工資段的客戶進行開拓。

2.3.3 資料清洗、轉化和離散化處理

基於上面的初步分析，首先將影響流失程度較輕的特徵變數刪除，方便更好的對資料

進行建模分析。再對兩個文字變數“Geography”和“Gender”中的文字屬性值進行數值替換，從而轉換為數值型特徵變數。“Geography”是一個包含“France”，“Germany”和“Spain”三種屬性的變數，分別轉變為 2，1，0 這三個數值屬性；類似地，將包含在“Gender”變數中的文字屬性“Female”與“Male”轉換成資料 1 和數位 0。

因為“Age”、“Balance”、“CreditScore”和“EstimatedSalary”這四個資料在銀行客戶資料中都是連續型變數，因此我們將它們分別離散化來分析。

將“Age”離散化為 5 個區間：0 代表 30 歲以下，1 代表 30-40 歲，2 代表 40-50 歲，3 代表 50-60 歲，4 代表 60-70 歲，5 代表 70 歲以上。

同樣的對“CreditScore”進行離散化：0 代表 350-450 區間，1 代表 450-550 區間，2 代表 550-650 區間，3 代表 650-750 區間，4 代表 750-850 區間。

對“Balance”進行平均劃分 ,劃分為 5 個區間並從小到大的區間由 0 到 4 個數值代表，0 為客戶帳戶餘額落在 0-50179.618、1 為客戶帳戶餘額落在 50179.618-100359.236、2 為客戶帳戶餘額在 100359.236-150538.854、3 為客戶帳戶餘額在 150538.854-200718.472、4 表示客戶帳戶餘額落在 200718.472-250898.09。

對“EstimatedSalary”連續變數也進行平均劃分，數值 0 為客戶薪資在 11.58 - 40007.76、數值 1 為客戶薪資在 40007.76 - 80003.94、數值 2 為 80003.94 - 120000.12、數值 3 為客戶薪資在 120000.12 - 159996.3、數值 4 為客戶薪資在 159996.3 - 199992.48。

2.4 評價指標

2.4.1 混淆矩陣 Confusion Matrix

混淆矩陣（Confusion Matrix）是一個擁有實際和預測兩個維度列的特別列聯表，它往往是用於分析分類問題，它可以作為一個分類評價指標，同時它會衍生出多個評價指標。準確率和錯誤率是很難在分類問題中區分不一樣類別樣本錯分的程度，而混淆矩陣能夠將錯誤率更好的體現出來。混淆矩陣一共具有 n 行 n 列，列主要是代表了預測分類的結果，行則是代表了樣本的實際類別 [2]。

對於二分類，根據主要研究的類別，分為正類（Positive）和負類（Negative），根據模型預測的對錯分為 True 和 False。

（1）TP（True Positive）

TP 表示為真正類，也就是說樣本實際歸屬於正類別，且通過模型對該樣本進行預測所得到的結果也是歸屬正類別，即說明模型對於這個正類別樣本的預測結果是正確的。

（2）FP（False Positive）

FP 表示為假正類，表明樣本實際屬於負類別，而通過模型對該樣本進行預測，所得到

的結果是屬於正類別，也就是表明，模型對於這個負類別樣本的預測結果是錯誤的，FP 可以表現為存偽現象。

（3）TN（True Negative）

TN 表示真負類，也就是說明樣本實際歸屬於負類別，且通過模型對該樣本進行預測所得到的結果也是歸屬於負類別，即說明模型對於這個負類別樣本的預測結果是正確的。

（4）FN（False Negative）

FN 表示為假負類，說明樣本實際歸屬於正類別，而通過模型對該樣本進行預測所得到的結果卻是歸屬於負類別，即說明模型對於這個正類別樣本的預測結果是錯誤的，FN 可以表現為去真現象。

一個二分類的混淆矩陣就由 TP、NP、TN 和 FN 這四個元素組成，如表 2.4.1.1。

表 2.4.1.1 二分類下混淆矩陣

1		**預測結果類別**	
		0	
樣本實際類別	1	TP	FN
	0	FP	TN

2.4.2 準確率 Accuracy

準確率（Accuracy）是經常會在分類問題中使用到的評價性能指標，它主要是用於表示模型的準確度，通常模型的準確率越高，該模型的預測分類效果就會越好。準確率的由被模型正確分類的樣本數和總樣本數兩個部分構成，即可以表示為：

2.4.3 精確率 Precision

精確率（Precision）是指模型能夠正確分類正類樣本的概率，在預測分類為正類結果中，該模型可以正確預測識別的概率。它可以表示為一個模型正確預測正類的個數與模型預測為正類的總數的比值，即可以表示為：

2.4.4 召回率 Recall

召回率（Recall）反映了該模型模型識別出實際正類樣本的概率，是指該模型能準確預測正類的樣本個數與實際的全部正類樣本的總數的比例，即可以表示為：

2.4.5 F1 值 F1-Score

F1 值（F1-Score）是指召回率和精確率的加權調和平均值，主要是為了綜合考慮召回率和精確率，盡可能使得召回率和精確率之間的差距能夠變小，可以表示為：

2.4.6 AUC 值

ROC（Receiver Operating Characteristic）是一個縱軸以召回率，而橫軸則是實際歸屬

於負類卻被預測為正類的樣本數與所有實際為負類的樣本總量的比率為基礎的曲線，利用 ROC 曲線作為評估指標，可以忽略樣本資料不平衡的問題。AUC（Area Under roc Curver）值是 ROC 曲線下的面積，其取值範圍在 0 到 1 之間，當 AUC 的值越接近于 1 時，模型的預測分類能力更強。

2.4.7 Kappa 係數

（1）總體分類精度

指的是預測分類正確所有類別的樣本總數與所有樣本總數的比值，即可以表示為：

（2）偶然一致性誤差

指的是對每個類別中的實際結果個數和預測分類結果個數的乘積進行加法，再與所有樣本總量的比值，即可以表示為：

Kappa 係數能夠用於一致性核對總和衡量預測分類的效果，其取值範圍屬於 [-1,1]。Kappa 係數其實是指預測分類樣本的結果與實際樣本的類別是否相符合，其值越大說明一致性越好。其表示為：

3 K 近鄰分類演算法

3.1 KNN 演算法介紹

KNN（K-Nearest Neighbor）是一種較為簡單、易於理解的機器學習方法。所謂的 KNN，就是指在一個樣本點上，將它與其它幾個點之間的距離進行度量，其中最常見的度量方法有：歐式距離、曼哈頓距離和切比雪夫距離。找出與該樣本點最近的 K 個樣本，並對這些鄰近的樣本點的歸屬類別進行統計，從而得到計數最高的類別，那麼最終這個目標樣本點就歸屬於計數最高類別。KNN 的有點是容易理解和容易實現，然而，在大量的資料下，KNN 的計算工作量會很大，並且在樣本資料不均衡的情況下，會對預測結果造成很大的影響。

該演算法的主要步驟為：輸入了一個給定的實例之後，要確定與這個新樣本距離最近的 K 個實例。如果 K 個實例中有大部分都歸屬於某一個類別時，則依據多數原則，將這一個等待分類的樣本也劃分為這一類別，KNN 是一種懶惰學習的演算法 [4]。

3.2 KNN 模型建立和求解

首先分別計算出在不同的鄰近點個數下的錯誤率以及訓練集和測試集的得分，分別選擇出最低的錯誤率以及兩個訓練集和測試集得分最優情況對應的鄰近點個數，從而確定鄰近點的個數進行建模。一般將鄰近點個數範圍確定為 1 到 20 個。

計算每個鄰近點對應的錯誤率，將其視覺化如下圖 3.2.1：

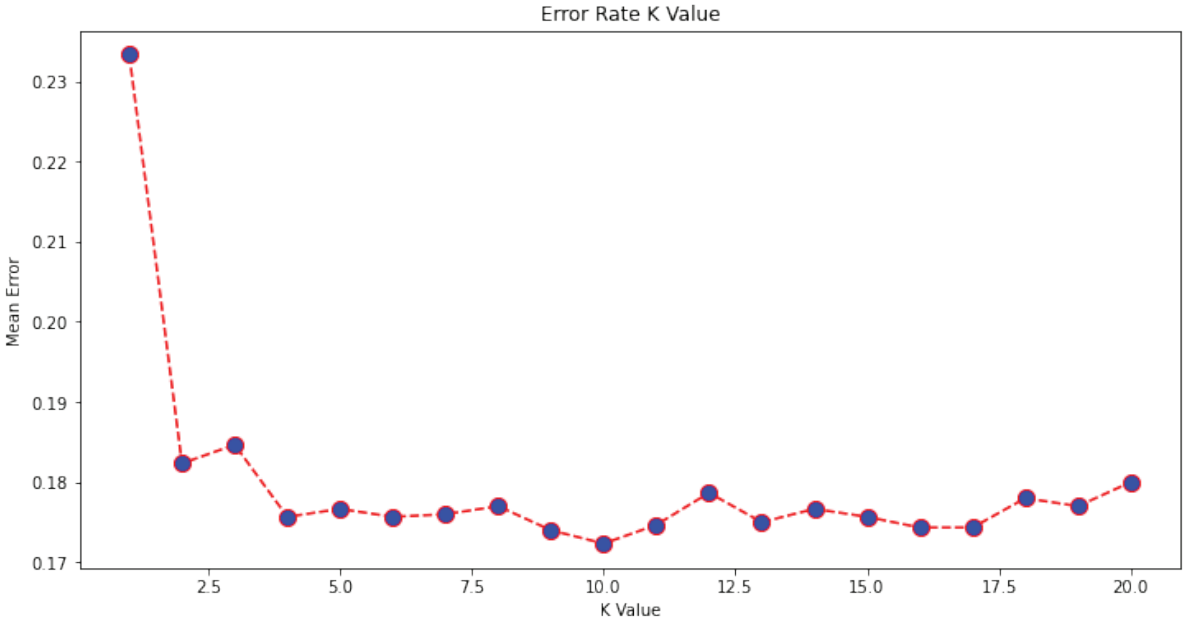


圖 3.2.1 每個鄰近點的錯誤率圖

由此圖可以得知當鄰近點個數 K 取值為 10 的時候，錯誤率是最低的，也就是說此時模型品質更加優質。

計算每個鄰近點對應的訓練集和測試集的得分，將其視覺化如下圖 3.2.2：

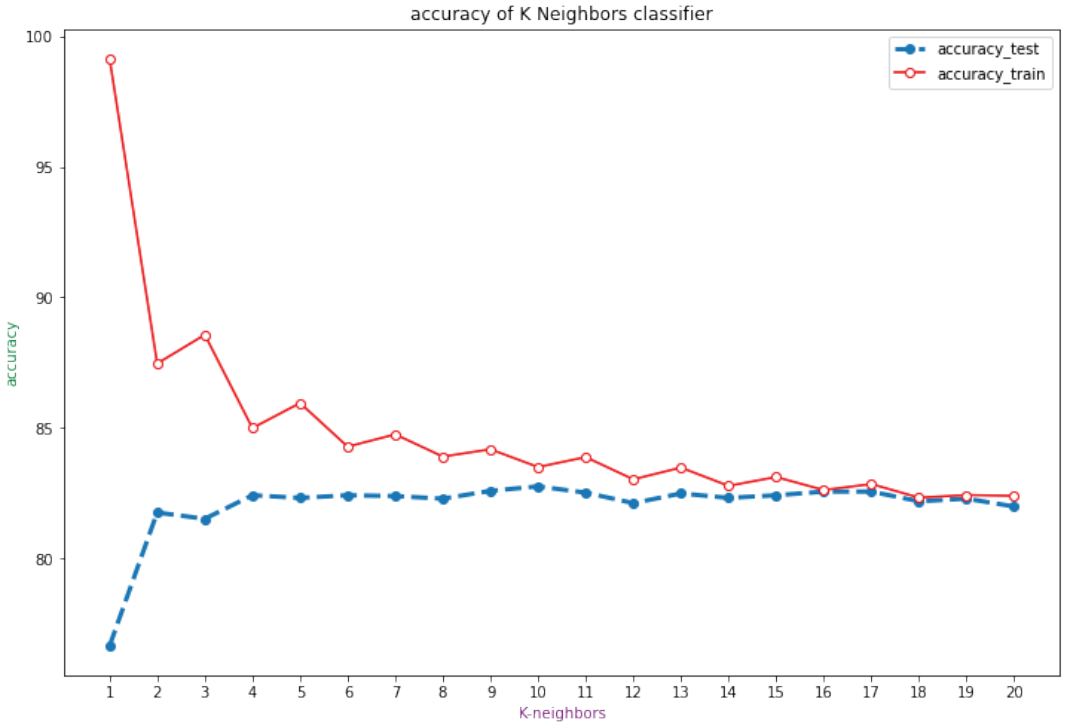


圖 3.2.2 不同鄰近點的測試集和訓練集得分圖

根據上圖可以得知當鄰近點個數 K 取值為 10 的時候，測試集的得分和訓練集的得分適配較高，並且測試集的得分和訓練集的得分差距比較小，所以在鄰近點個數 K 取值為 10 時建立的模型品質更高。

3.3 KNN 模型預測和評價

將鄰近點個數 K 取值為 10 建立 KNN 模式並獲得該模型測試集和訓練集的得分如圖 3.3.1，測試集分類得分為 83.51%，預測測試集分類得分為 82.77%，即該模型的準確率為 82.77%。

```
print('KNN训练集得分：',knn_best.score(x_train,y_train))
print('KNN预测分类得分：',knn_best.score(x_test,y_test))
```

```
KNN训练集得分：  0.8351428571428572
KNN预测分类得分：  0.8276666666666667
```

圖 3.3.1 訓練集和測試集得分圖

建立該模型中測試集實際結果和預測結果的混淆矩陣表 3.3.1，其中統計了預測分類結果為未流失而測試集實際結果為未流失的 TP 個數為 2365，預測分類結果為未流失而測試集實際結果為流失的 FN 個數為 466，預測分類結果為流失而測試集實際結果為未流失的 FP 個數為 51，預測分類結果為流失而測試集實際結果為流失的 TN 個數為 118。

表 3.3.1 KNN 模型的預測混淆矩陣表

	預測結果為 0（未流失）	**預測結果為 1（流失）**
實際結果為 0（未流失）	2365	51
實際結果為 1（流失）	466	118

查看建立的 KNN 模型的準確率報表如表 3.3.2：

表 3.3.2 KNN 模型的準確率表

	precision	**recall**	**f1-score**	**support**
0	0.84	0.98	0.90	2416
1	0.70	0.20	0.31	582
accuracy			0.83	3000
macro avg	0.77	0.59	0.61	3000
weighted avg	0.81	0.83	0.79	3000

繪畫 K=10 時模型的 ROC 曲線，如圖 3.3.2。

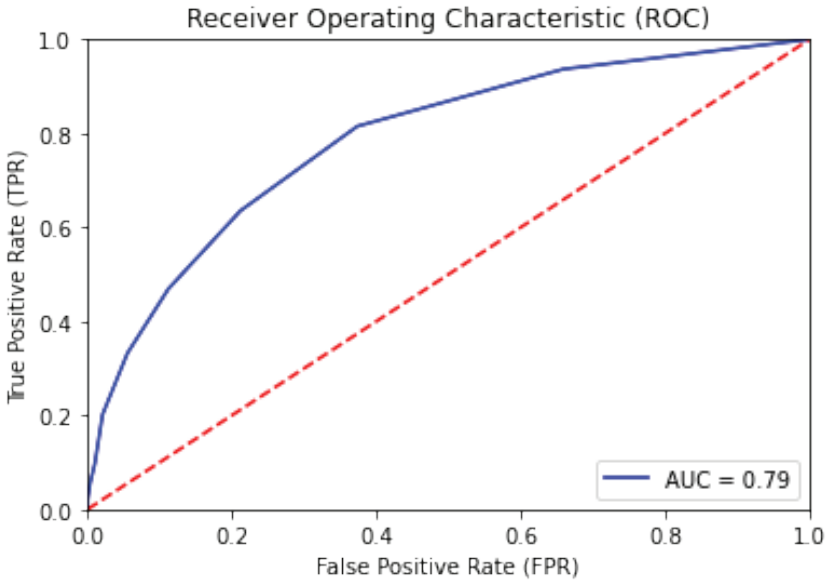


圖 3.3.2 KNN 模型的 ROC 曲線圖

由此可以得到該模型的 AUC 為 0.79，K=10 時的 KNN 分類預測模型是一個預測一般的預測模型。

4 反向傳播神經網路演算法

4.1 BP 神經網路演算法介紹

BP（Back Propagation）神經網路演算法是一個利用反復類比的過程，先計算出誤差，然後將誤差進行反向傳播，然後進行學習訓練的多層前饋神經網路，它是目前應用較為廣泛的一種神經網路。BP 神經網路的主要核心思想是輸入訓練集資料到人工神經網路的輸入層，該資料通過在隱藏層中的計算，再將計算後的結果輸出，然後再把計算後的結果與實際結果進行誤差計算，將誤差結果反向傳播，最終達到輸入層 [5]。

BP 神經網路能夠存入大量的輸入輸出映射關係，並且不需要在構建模型之前對這個映射關係的數學方程進行描述。BP 神經網路主要是由輸入層、隱藏層和輸出層這三個基本結構構成，其中輸入層指的是輸入資訊端，隱藏層主要是指資訊的處理端，輸出層指的是將資訊進行輸出的埠。

BP 神經網路的學習規則是利用最速下降法，BP 神經網路主要是有工作信號正向傳播和誤差信號反向傳播這兩個過程。該方法首先採用正向傳遞的方法，使輸入的資料首先由一個輸入層傳遞給一個神經網路，再由各個隱藏層對其進行運算，最終在輸出層將結果輸出；其次進行的過程是反向傳播過程，根據反向轉播對神經網路的權值和閾值進行不斷的調整，讓神經網路中的誤差平方和最小。

4.2 BP 神經網路模型建立和求解

首先，通過分析各種類型的隱藏層資料，得到誤差率、訓練和測試資料的分數，選取誤差最小以及訓練和測試資料分數的最優情況所對應的隱藏層數，進而得到最佳隱藏層數，再進行建模，隱藏層的層數範圍一般確定為 1 到 20 個。

計算每個隱藏層數對應的訓練集和測試集的得分，將其視覺化如下圖 4.2.1，當隱藏層數為 20 層的時候，訓練集和測試集的訓練得分最高。

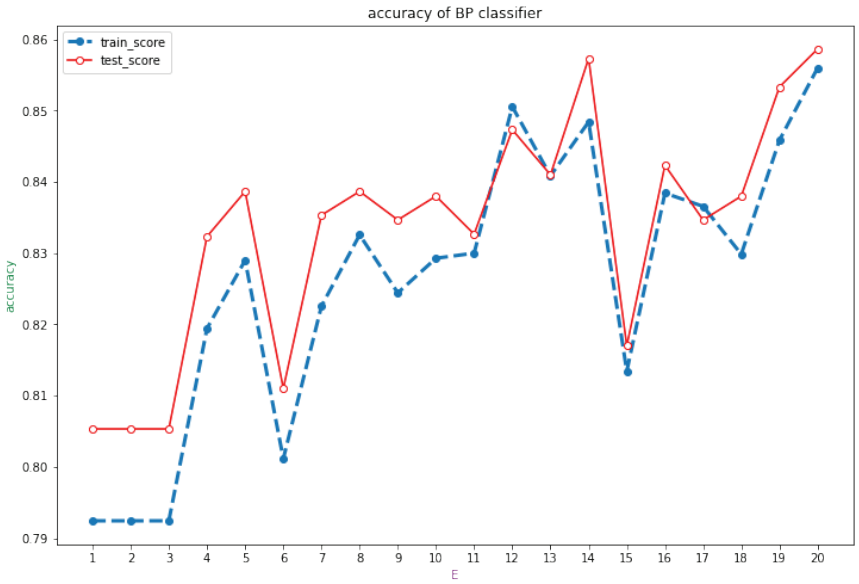


圖 4.2.1 各隱藏層層數的模型得分圖

查看每層隱藏層的錯誤率，由圖 4.2.2 可以得知當隱藏層數取值為 20 的時候，錯誤率是最低的。

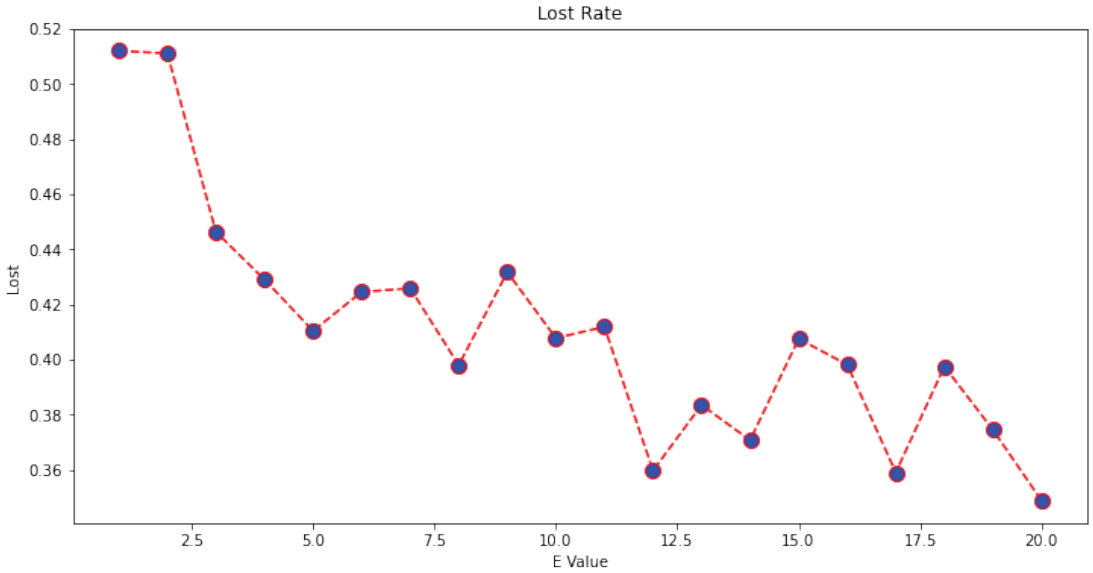


圖 4.2.2 不同隱藏層層數的模型錯誤率圖

根據圖 4.2.1 和圖 4.2.2 可以得知在隱藏層層數為 20 時，測試集的得分高於訓練集的得分，說明可能存在過擬合的問題。在隱藏層層數為 12 時，訓練集得分和測試集的得分相近，且此時的錯誤率也較低，也就是說此時模型的品質會比較優質，因此選取隱藏層層數為 12 的 BP 神經網路進行建模分析。

4.3 BP 神經網路模型預測和評價

在將隱藏層設定為 12 層的時候，對這個 BP 神經網路模型的損失和評分進行觀察，具體見圖 4.3.1，這個模型的損失為 0.3597，訓練集得分為 85.06%，測試集得分為 84.73%。

```
print('BP神经网络损失：',BP_best.loss_)
print('BP神经网络训练集得分：',BP_best.score(x_train,y_train))
print('BP神经网络测试集得分：',BP_best.score(x_test,y_test))
```

```
BP神经网络损失： 0.35966413388115676
BP神经网络训练集得分： 0.8505714285714285
BP神经网络测试集得分： 0.8473333333333334
```

圖 4.3.1 BP 神經網路模型的損失和得分圖

觀察到，在隱藏層層數為 12 的 BP 神經網路模型中，如圖 4.3.2，該模型的訓練集均方誤差為 0.1494，測試集的均方誤差為 0.1527。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.14942857142857144
Test MSE: 0.15266666666666667
```

圖 4.3.2 BP 神經網路模型的均方誤差情況圖

統計構建出 BP 神經網路模型預測結果的混淆矩陣表 4.3.1，由該表可以得知真正例 TP 為 2272，假正例 FP 為 314，假反例 FN 為 144，真反例 TN 為 270。

表 4.3.1 BP 神經網路模型的混淆矩陣表

	預測結果為 0（未流失）	預測結果為 1（流失）
實際結果為 0（未流失）	2272	144
實際結果為 1（流失）	314	270

利用所得真假正反例的個數計算出模型的精確率、召回率和整體準確率：

精確率 Precision：

$$\text{Precision(Exited} = 0) = \frac{TP}{TP + FP} = \frac{2272}{2272 + 314} = 87.86\%$$

$$\text{Precision(Exited} = 1) = \frac{TN}{TN + FN} = \frac{270}{270 + 144} = 65.22\%$$

召回率 Recall：

$$\text{Recall(Exited} = 0) = \frac{TP}{TP + FN} = \frac{2272}{2272 + 144} = 94.04\%$$

$$\text{Recall(Exited} = 1) = \frac{TN}{TN + FP} = \frac{270}{270 + 314} = 46.23\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2272 + 270}{2272 + 270 + 314 + 144} = 84.73\%$$

查看建立的 BP 神經網路模型的準確率報表如表 4.3.2：

表 4.3.2 BP 神經網路模型的準確率表

	Precision	Recall	F1-Score	Support
0	0.88	0.94	0.91	2416
1	0.65	0.46	0.54	584
Accuracy			0.85	3000
Aacro Avg	0.77	0.70	0.72	3000
Weighted Avg	0.83	0.85	0.84	3000

繪畫出 BP 神經網路模型的 ROC 曲線如圖 4.3.3，得到 ROC 曲線下的面積 AUC 值為 0.83，可以看出 BP 神經網路模型是一個比較好的模型。

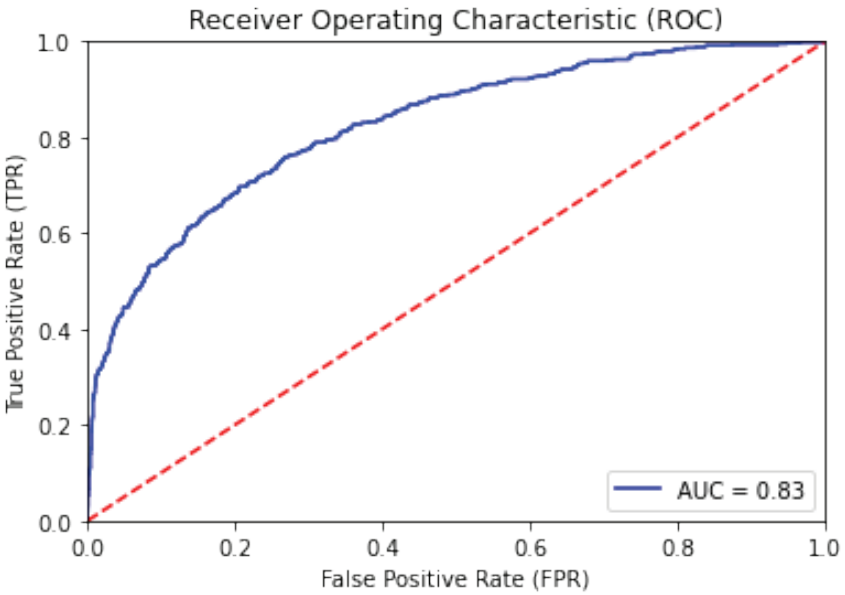


圖 4.3.3 BP 神經網路模型 ROC 曲線圖

5 支援向量機分類演算法

5.1 支援向量機演算法介紹

支持向量機 SVM（Support Vector Machine）是一種用於來進行分類或者是回歸的機器學習，也是一種用來處理二元分類問題的經典方法。SVM 的主要思想是根據間隔最大化原則找到一個超平面來對樣本進行劃分，使其能轉化為一個凸二次規劃問題求解 [6]。

支持向量機是 Vapnik 等人在 90 年代提出來的，目的是找到一個超平面對二分類進行劃分，讓分類的錯誤最小化的一個模型 [7]。簡而言之，就是通過構建一條直線或者一個超平面，將不同的類別進行分割，從實質上講，這是一個最大邊緣的分類器，所以，這種學習方法的核心是對支援向量與直線或者超平面的間距進行最大化的處理。具體操作是通過根據所研究的 n 特徵數建立一個 n 維空間的坐標系，每一樣本資料映射在該 n 維空間中的一個點，找到一條直線或者一個超平面能夠將樣本資料點進行分類。

5.2 支援向量機模型建立和求解

通過在 sklearn 中使用 SVC 對訓練資料進行支援向量機模型化，其預設採用了高斯徑向基 RBF（Radial Basis Function）核函數，將懲罰係數 C 設定為 1，對樹的最大深度沒有任何約束，直到所有葉子都屬於相同類或者為最小樣本劃分資料。

查看該支援向量機初始模型的得分情況，如下圖 5.2.1，訓練集得分和測試集得分分別為 82.07% 和 82.57%。

```
Supportvactormachine训练集得分：',Supportvactormachine.score(x_train, y_train))
Supportvactormachine测试集得分：',Supportvactormachine.score(x_test, y_test))
```

```
Supportvactormachine训练集得分： 0.8207142857142857
Supportvactormachine测试集得分： 0.8256666666666667
```

圖 5.2.1 初始 SVM 模型得分圖

查看該支援向量機初始模型的均方誤差情況，如圖 5.2.2，訓練集均方誤差為 0.1793，測試集均方誤差為 0.1743。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.1792857142857143
Test MSE: 0.17433333333333334
```

圖 5.2.2 初始 SVM 模型均方誤差圖

構建模型測試預測結果的混淆矩陣表 5.2.1，可以得知真正例 TP 為 2409，假正例 FP 為 516，假反例 FN 為 7，真反例 TN 為 68。

表 5.2.1 初始 SVM 模型混淆矩陣表

	預測結果為 0（未流失）	預測結果為 1（流失）
實際結果為 0（未流失）	2409	7
實際結果為 1（流失）	516	68

利用所得真假正反例的個數計算出模型的精確率、召回率和整體準確率。

精確率 Precision：

$$\text{Precision}(\text{Exited} = 0) = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{2409}{2409 + 516} = 82.36\%$$

$$\text{Precision}(\text{Exited} = 1) = \frac{\text{TN}}{\text{TN} + \text{FN}} = \frac{68}{68 + 7} = 90.67\%$$

召回率 Recall：

$$\text{Recall}(\text{Exited} = 0) = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{2409}{2409 + 7} = 99.71\%$$

$$\text{Recall}(\text{Exited} = 1) = \frac{\text{TN}}{\text{TN} + \text{FP}} = \frac{68}{68 + 516} = 11.64\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2409 + 68}{2409 + 68 + 516 + 7} = 82.57\%$$

表 5.2.2 初始 SVM 模型準確率表

	Precision	Recall	F1-Score	Support
0	0.82	1.00	0.90	2416
1	0.91	0.12	0.21	584
Accuracy			0.83	3000
Macro Avg	0.87	0.56	0.55	3000
Weighted Avg	0.84	0.83	0.77	3000

繪畫出 ROC 曲線如圖 5.2.3 可以看出，ROC 曲線偏向于左上角，得到 ROC 曲線下的面積 AUC 值為 0.8108，是一個比較優質的模型，但是仍然對模型進行調參訓練，從而提高 AUC 值。

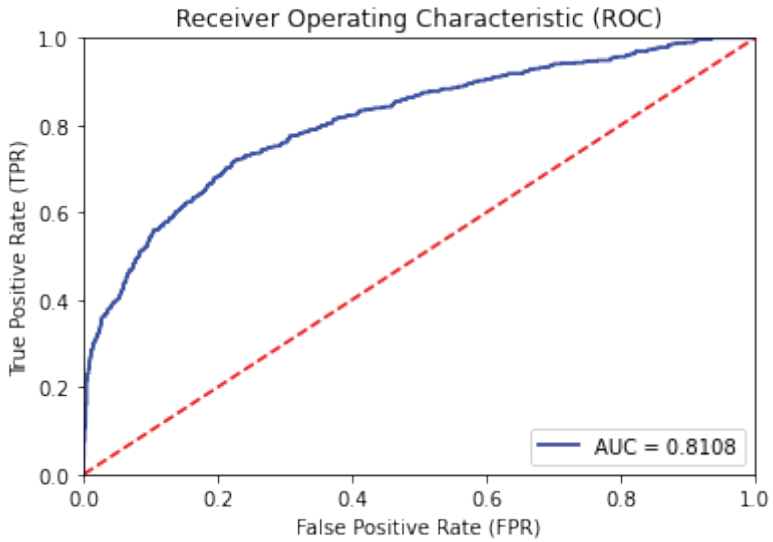


圖 5.2.3 初始 SVM 模型 ROC 曲線圖

對支援向量機模型進行第一次調參，利用網格搜索 GridSearchCV 對初始模型進行最大懲罰係數 C 的最優選擇，得到最優的懲罰係數 C 為 50。對初始模型進行懲罰係數參數的重設，重新建立和擬合模型。

一次調參後的支援向量機模型得分情況如圖 5.2.4，一次調參後的模型的訓練集得分為 86.13%，測試集得分為 85.70%。相比於調參前的 SVM 模型來說，模型的訓練集和測試集得分都有較明顯的提高。

```
print('Supportvactormachine训练集得分：',Supportvactormachine_param1.score(x_t
print('Supportvactormachine测试集得分：',Supportvactormachine_param1.score(x_t
```

```
Supportvactormachine训练集得分： 0.8612857142857143
Supportvactormachine测试集得分： 0.857
```

圖 5.2.4 一次調參後 SVM 模型得分情況圖

查看該支援向量機一次調參後模型的均方誤差情況，如圖 5.2.5，一次調參後的支援向量機模型的訓練集均方誤差為 0.1387，測試集均方誤差為 0.1430。一次調參後的支援向量機模型的誤差有了明顯的下降。因此選用一次調參後的支援向量機模型進行建模預測和分析。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.1387142857142857
Test MSE: 0.143
```

圖 5.2.5 一次調參後 SVM 模型均方誤差圖

5.3 支援向量機模型預測和評價

構建一次調參後模型測試預測結果的混淆矩陣表 5.3.1，可以得知真正例 TP 為 2360，假正例 FP 為 373，假反例 FN 為 56，真反例 TN 為 211。

表 5.3.1 SVM 模型混淆矩陣表

	預測結果為 0（未流失）	預測結果為 1（流失）
實際結果為 0（未流失）	2360	56
實際結果為 1（流失）	373	211

利用所得真假正反例的個數計算出模型的精確率、召回率和整體準確率：

精確率 Precision：

$$\text{Precision(Exited} = 0) = \frac{TP}{TP + FP} = \frac{2360}{2360 + 373} = 86.35\%$$

$$\text{Precision(Exited} = 1) = \frac{TN}{TN + FN} = \frac{211}{211 + 56} = 79.03\%$$

召回率 Recall：

$$\text{Recall(Exited} = 0) = \frac{TP}{TP + FN} = \frac{2360}{2360 + 56} = 97.68\%$$

$$\text{Recall(Exited} = 1) = \frac{TN}{TN + FP} = \frac{211}{211 + 373} = 36.13\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2360 + 211}{2360 + 211 + 373 + 56} = 85.70\%$$

查看最終建立的支援向量機模型的準確率報表如表 5.3.2：

表 5.3.2 SVM 模型準確率表

	Precision	Recall	F1-Score	Support
0	0.86	0.98	0.92	2416
1	0.79	0.36	0.50	584
Accuracy			0.86	3000
Macro Avg	0.83	0.67	0.71	3000
Weighted Avg	0.85	0.86	0.83	3000

繪畫出 ROC 曲線如圖 5.3.1，可以得到 ROC 曲線下的面積 AUC 值為 0.8261，AUC 值有了明顯的提升，而且整體的準確率提升到了 86%，所以經過一次調參後的支援向量機模型是一個品質較高的模型。

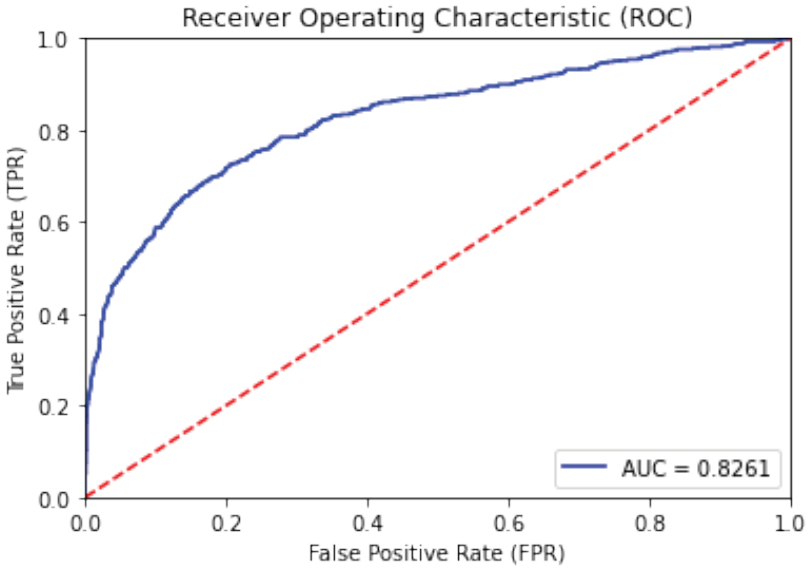


圖 5.3.1 SVM 模型的 ROC 曲線圖

6 決策樹分類演算法

6.1 決策樹演算法介紹

決策樹（Decision Classification Tree）是一種直觀簡單且容易理解的分類模型，其思想與 If-Else 一致，根據相應的規則選擇節點，從而形成對應的條件，根據資料的特徵，構建出多層的判斷分類層，對資料進行多次的判斷分類從而將資料進行最終的分類。決策樹的整體形狀是呈現出一個倒數形的結構，在決策樹每個內部中的節點都表示了一個屬性上面的判斷，其中每一個分支都代表了一個判斷結果的輸出，在這個決策樹尾端的每個葉子結點都表示了一種分類的結果 [8]。

在決策樹基於資訊理論的情況下，根據判斷劃分資料集的原則將決策樹主要分為了三種類型，分別是 ID3 決策樹（Iterative Dichotomiser 3）、C4.5 決策樹和 CART 決策樹（Classification And Regression Tree）。其中 C4.5 決策樹和 ID3 決策樹是屬於多分類決策樹，主要的區別在於 ID3 決策樹的劃分資料集原則是根據計算所得的資訊增益，選擇要劃分的特徵變數作為節點，且 ID3 決策樹只能夠解決離散型特徵變數問題，而 C4.5 決策的劃分資料集原則是根據計算所得的資訊增益率，選擇要劃分的特徵變數作為節點，且 C4.5 決策樹能夠解決連續型特徵變數和離散型特徵變數問題。CART 決策樹屬於二分類決策樹，其分割於是原理是以基尼係數為基礎，選取待分割的特徵量為結點，對其進行分割。

6.2 決策樹模型建立和求解

採用 DecisionTreeClassifier（）功能函數，對已有的訓練樣本進行擬合，構建了該決策樹模型。構建決策樹模型後，計算訓練資料集和測試資料集的模型得分，由圖 6.2.1 可以得知，分別為 99.17% 和 78.57%，由此可以看出訓練集的擬合效果過於好，而測試集的預測結果不是很理想，所以推斷該模型可能存在著過擬合的情況。

```
print('DecisionTree训练集得分：',DecisionTree.score(x_train, y_train))
print('DecisionTree测试集得分：',DecisionTree.score(x_test, y_test))
```

```
DecisionTree训练集得分： 0.9917142857142857
DecisionTree测试集得分： 0.7856666666666666
```

圖 6.2.1 初始決策樹模型得分圖

根據構建的決策樹模型進行計算，如圖 6.2.2 所示，結果表明，訓練集預測的均方誤差為 0.0083，測試集預測的均方誤差 0.2143，從中可以看到，模型中有過擬合的現象，所有有必要對模型進行調參改進。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.008285714285714285
Test MSE: 0.21433333333333332
```

圖 6.2.2 初始決策樹模型均方誤差圖

統計構建出決策樹模型預測結果的混淆矩陣表 6.2.1，可以得知真正例 TP 為 2079，假正例 FP 為 306，假反例 FN 為 337，真反例 TN 為 278。

表 6.2.1 決策樹初始模型混淆矩陣表

	預測結果為 0（未流失）	預測結果為 1（流失）
實際結果為 0（未流失）	2079	337
實際結果為 1（流失）	306	278

利用所得真假正反例的個數計算出模型的精確率、召回率和整體準確率。

精確率 Precision：

$$\text{Precision}(\text{Exited}=0)=\frac{TP}{TP+FP}=\frac{2079}{2079+306}=87.17\%$$

$$(\text{Exited}=1)=\frac{TN}{TN+FN}=\frac{278}{278+337}=45.20\%$$

召回率 Recall：

$$\text{Recall}(\text{Exited}=0)=\frac{TP}{TP+FN}=\frac{2079}{2079+337}=86.05\%$$

$$\text{Recall}(\text{Exited}=1)=\frac{TN}{TN+FP}=\frac{278}{278+306}=47.60\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2079 + 278}{2079 + 278 + 306 + 337} = 78.57\%$$

表 6.2.2 初始決策樹模型準確率表

	Precision	Recall	F1-Score	Support
0	0.87	0.86	0.87	2416
1	0.45	0.48	0.46	584
Accuracy			0.79	3000
Macro Avg	0.66	0.67	0.66	3000
Weighted Avg	0.79	0.79	0.79	3000

繪畫出決策樹模型的 ROC 曲線，由圖 6.2.3 得到 ROC 曲線下的面積 AUC 值為 0.6718，是一個比較差的模型，需要對模型調參後再進行訓練，提高 AUC 值。

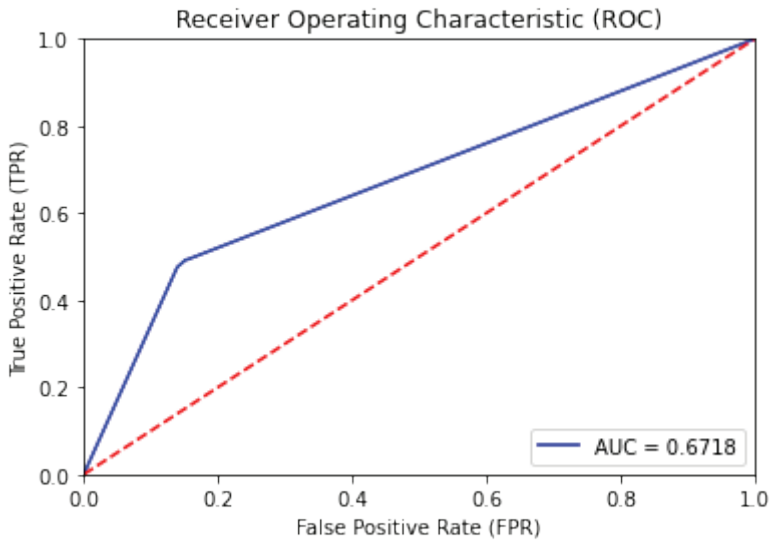


圖 6.2.3 初始決策樹模型 ROC 曲線圖

利用網格搜索選取最優的最大深度參數，得到最優的最大深度為 9，調整模型的最大深度參數為 9，重新構建一次調參後的決策樹模型，由圖 6.2.4 得訓練集和測試集的分為 87.53% 和 85.53%。

```
print('DecisionTree训练集得分：',DecisionTree_param1.score(x_train, y_train))
print('DecisionTree测试集得分：',DecisionTree_param1.score(x_test, y_test))
```

```
DecisionTree训练集得分： 0.8752857142857143
DecisionTree测试集得分： 0.8553333333333333
```

圖 6.2.4 一次調參後決策樹模型得分圖

在一次調參後的決策樹模型中，從圖 6.2.5 中得出的訓練集均方誤差為 0.1247，測試集均方誤差為 0.1447，與一次調參之前相比，儘管訓練集的均方誤差變大，但測試集的均方誤差卻在降低；同時，由於訓練和測試樣本間的均方誤差差異較小，所以降低了過擬合的可能。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.12471428571428571
Test MSE: 0.14466666666666667
```

圖 6.2.5 一次調參後決策樹模型均方誤差圖

模型經過一次調參後，由圖 6.2.6 可得 AUC 值從 0.6916 提升到 0.8047，有了明顯的提升。

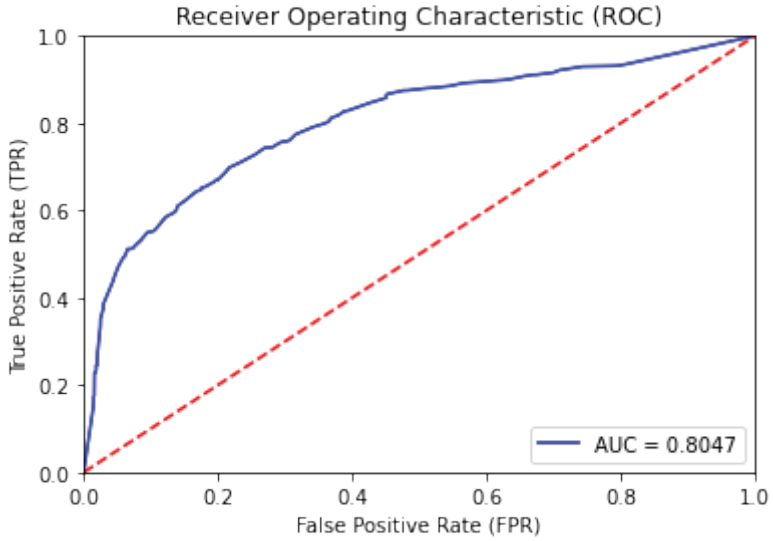


圖 6.2.6 一次調參後決策樹模型的 ROC 曲線圖

對模型展開二次調參，這一次調參的目的是為了限制葉子結點的最少樣本數，以網格搜索為基礎，選出最佳的葉子結點，最少樣本數為 5。觀察到了訓練集和測試集的得分，具體結果如圖 6.2.7，分別 , 為 86.84% 和 86.13%，與一次調參後的訓練集得分有了小幅度的下降。

```
print('DecisionTree训练集得分：',DecisionTree_param2.score(x_train, y_train))
print('DecisionTree测试集得分：',DecisionTree_param2.score(x_test, y_test))
```

```
DecisionTree训练集得分： 0.8684285714285714
DecisionTree测试集得分： 0.8613333333333333
```

圖 6.2.7 二次調參決策樹模型的得分圖

二次調參後模型的訓練集和測試集的均方誤差如圖 6.2.8，分別為 0.1316 和 0.1387，較一次調參後也有小幅度的升高。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.13157142857142856
Test MSE: 0.13866666666666666
```

圖 6.2.8 二次調參決策樹模型的均方誤差圖

由圖 6.2.9 可知，二次調參後的模型 AUC 值為 0.8251，模型的品質相比於一次調參後的模型品質要更好。

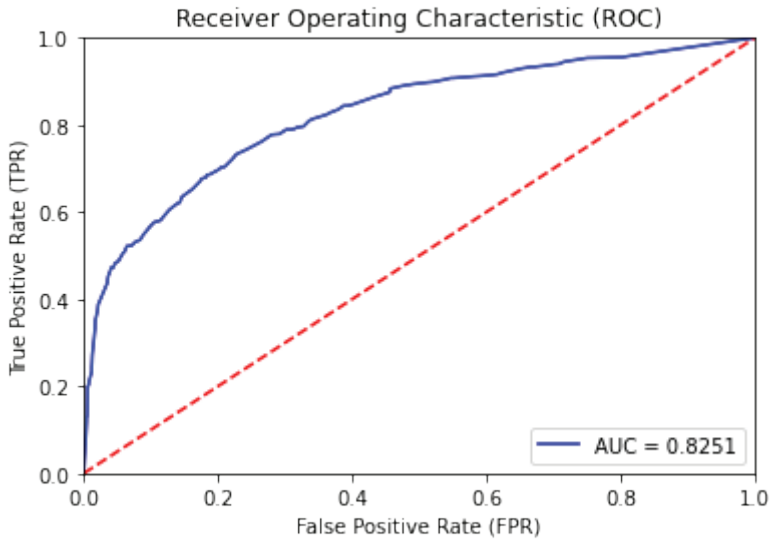


圖 6.2.8 二次調參決策樹模型的 ROC 曲線圖

對二次調參後的模型再進行一次調參，本次調參是針對內部結點再劃分所需要的最小樣本數，經過網格搜索選取出了最優的內部結點再劃分所需要的最小樣本數為 20 個。由圖 6.2.9 查看得到訓練集和測試集的得分分別為 86.74% 和 85.67%，測試集的得分有小幅度的降低。

```
print('DecisionTree训练集得分：',DecisionTree_param3.score(x_train, y_train))
print('DecisionTree测试集得分：',DecisionTree_param3.score(x_test, y_test))
```

```
DecisionTree训练集得分： 0.8674285714285714
DecisionTree测试集得分： 0.8566666666666667
```

圖 6.2.9 三次調參決策樹模型的得分圖

由圖 6.2.10 查看得到訓練集和測試集均方誤差分別為 0.1336 和 0.1433，測試集的得分有小幅度的升高，選用三次調參後的模型進行訓練和預測。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.13157142857142856
Test MSE: 0.13866666666666666
```

圖 6.2.10 三次調參決策樹模型均方誤差圖

6.3 決策樹模型預測和評價

構建三次調參後決策樹模型用測試資料進行預測結果的混淆矩陣表 6.3.1，我們可以知道，真正例 TP 是 2258，假正例 FP 是 299，假反例 FN 是 131，真反例 TN 是 285。

表 6.3.1 決策樹最終模型的預測混淆矩陣表

	預測結果為 0（未流失）	**預測結果為 1（流失）**
實際結果為 0（未流失）	2258	131
實際結果為 1（流失）	299	285

根據得到的正確與錯誤實例數目，對精確率、召回率和整體準確率進行評價：

精確率 Precision：

$$\text{Precision}(\text{Exited} = 0) = \frac{TP}{TP + FP} = \frac{2258}{2258 + 299} = 88.31\%$$

$$\text{Precision}(\text{Exited} = 1) = \frac{TN}{TN + FN} = \frac{285}{285 + 131} = 68.51\%$$

召回率 Recall：

$$\text{Recall}(\text{Exited} = 0) = \frac{TP}{TP + FN} = \frac{2258}{2258 + 131} = 94.52\%$$

$$\text{Recall}(\text{Exited} = 1) = \frac{TN}{TN + FP} = \frac{285}{285 + 299} = 48.01\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2258 + 285}{2258 + 285 + 299 + 131} = 85.54\%$$

查看最終建立的決策樹模型的準確率報表如表 6.3.2：

表 6.3.2 決策樹模型準確率表

	Precision	**Recall**	**F1-score**	**Support**
0	0.88	0.95	0.91	2416
1	0.69	0.49	0.57	584
Accuracy			0.86	3000
Macro Avg	0.78	0.72	0.74	3000
Weighted Avg	0.85	0.86	0.85	3000

將三次調參後的決策樹模型的 ROC 曲線繪製成了圖 6.3.1，三次調參後的 ROC 曲線比原始的 ROC 曲線更加傾向于左上方，計算出了三次調參後的 ROC 曲線下麵的區域面積 AUC 值是 0.8290，與原始模型的 AUC 值 0.6718 相比，三次調參後的決策樹模型表現出了更高的品質，是一種高品質的模型。

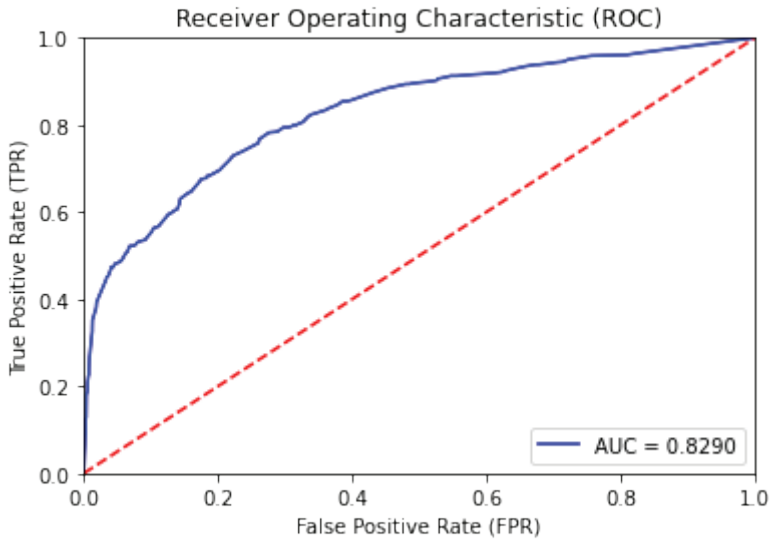


圖 6.3.1 決策樹模型的 ROC 曲線圖

7 隨機森林分類演算法

7.1 隨機森林演算法介紹

隨機森林（Random Forest）是一種集成分類器，它的基礎分類器是決策樹，它將隨機的生成一些互相沒有聯繫的決策樹，將這些隨機生成的決策樹組合一起，組成一個隨機森林。當隨機森林接受到新的一個樣本時，每棵構成隨機森林的決策樹都會對該樣本進行分類，每棵決策樹的分類結果都會被綜合統計，其中最多的類別就是隨機森林預測該樣本的分類結果。

總的來說，隨機森林將決策樹作為基分類器，同時又添加了裝袋演算法（Bagging）的思想，利用自助法（Boot-Strap）重採樣的技術，從初始訓練集中隨機性有放回的抽取 N 個樣本，再從每個樣本中隨機性挑選一部分特徵去構建一棵決策樹，最終組合這 N 棵相互獨立的決策樹，根據多數投票的原則去預測未知樣本 [4]。

隨機森林的步驟主要為：第一，對資料集進行交叉的原理，從中抽取 N 個測試集去建立 N 棵決策樹，剩下的測試集樣本用於判斷模型的能力；第二，從每一棵決策樹的每一個結點隨機性的選取 M 個屬性，再從這個 M 個屬性中挑選出分類能力最高的變數，迴圈反

復最大程度的建立決策樹；第三，將生成的多棵決策樹組合在一起，從而形成了隨機森林，然後將測試集資料用於次分類器中進行預測分類，最終的一個分類結果是由每一棵決策樹的分類結果投票決定 [9]。

7.2 隨機森林模型建立和求解

利用 sklearn 包中的 RandomForestClassifier，對訓練資料進行隨機森林模型化，預設預設了 10 個決策樹，選取基尼指標（Gini）作為最佳的特徵選擇，即使用 CART 決策樹來構造森林，將最小的樣本分割數量限定在 2 個，對最大深度沒有任何的約束，直到所有的葉子被歸入相同的類別或者為最小樣本劃分資料。

根據構建的隨機森林模型進行計算，如圖 7.2.1 所示，訓練所得到的隨機森林模型的訓練準確率為 99.17%，模型預測準確率為 85.10%。

```
print('RandomForest训练集得分：',randomforest.score(x_train, y_train))
print('RandomForest测试集得分：',randomforest.score(x_test, y_test))
```

```
RandomForest训练集得分： 0.9917142857142857
RandomForest测试集得分： 0.851
```

圖 7.2.1 初始隨機森林模型得分圖

如圖 7.2.2 所示，得到訓練集和測試集的均方誤差分別為 0.0083 和 0.1490，由此可以看出訓練集的誤差遠小於測試集的誤差，因此需要對模型的參數進行調整，降低模型的過擬合。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.008285714285714285
Test MSE: 0.149
```

圖 7.2.2 初始隨機森林模型均方誤差圖

構建隨機森林初始模型測試預測結果的混淆矩陣表 7.2.1，可以得知真正例 TP 為 2275，假正例 FP 為 306，假反例 FN 為 141，真反例 TN 為 278。

表 7.2.1 初始隨機森林模型混淆矩陣表

	預測結果為 0（未流失）	預測結果為 1（流失）
實際結果為 0（未流失）	2275	141
實際結果為 1（流失）	306	278

利用所得真假正反例的個數計算出模型的精確率、召回率和整體準確率。

精確率 Precision：

$$\text{Precision(Exited} = 0) = \frac{TP}{TP + FP} = \frac{2275}{2275 + 306} = 88.14\%$$

$$\text{Precision(Exited} = 1) = \frac{TN}{TN + FN} = \frac{278}{278 + 141} = 66.35\%$$

召回率 Recall：

$$\text{Recall(Exited} = 0) = \frac{TP}{TP + FN} = \frac{2275}{2275 + 141} = 94.16\%$$

$$\text{Recall(Exited} = 1) = \frac{TN}{TN + FP} = \frac{278}{278 + 306} = 47.60\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2275 + 278}{2275 + 278 + 306 + 141} = 85.10\%$$

表 7.2.2 初始隨機森林模型準確率表

	Precision	Recall	F1-Score	Support
0	0.88	0.94	0.91	2416
1	0.66	0.48	0.55	584
Accuracy			0.85	3000
Macro Avg	0.77	0.71	0.73	3000
Weighted Avg	0.84	0.85	0.84	3000

繪畫出 ROC 曲線圖 7.2.3 可以得到 ROC 曲線下的面積 AUC 值為 0.8337，是一個比較優質的模型，但是仍然對模型進行調參訓練，從而提高 AUC 值，並降低過擬合的情況。

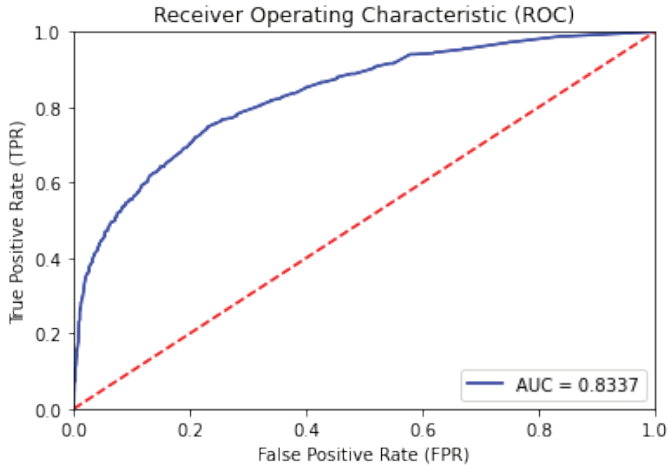


圖 7.2.3 初始隨機森林模型 ROC 曲線圖

針對資料處理中存在的過擬合問題，採取多次網格搜索方法尋找最優化的參數，並通過調整模型參數獲得更好的資料處理效果。

第一次調參為隨機森林學習器個數進行選擇，利用 GridSearchCV 對初始模型進行最優隨機森林學習器個數搜索，得到最優的隨機森林學習器個數為 400 個。

重新構建一次調參後的隨機森林模型，由圖 7.2.4 得到訓練集得分為 99.17%，測試集得分為 85.17%。

```
print('RandomForest训练集得分：',randomforest_param1.score(x_train, y_train))
print('RandomForest测试集得分：',randomforest_param1.score(x_test, y_test))
```

```
RandomForest训练集得分：  0.9917142857142857
RandomForest测试集得分：  0.8516666666666667
```

圖 7.2.4 一次調參隨機森林模型得分圖

根據圖 7.2.5，訓練集和測試集的均方誤差分別為 0.0083 和 0.1483，但是，訓練集的誤差依然比測試集的誤差要小很多，所以，還需要對模型的參數進行調節，從而減少模型的過擬合。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.008285714285714285
Test MSE: 0.14833333333333334
```

圖 7.2.5 一次調參隨機森林模型均方誤差圖

第二次調參是對隨機森林的深度進行了調節，通過 GridSearchCV 對一次參數後的模型進行了檢索，尋找出了最優決策樹最大深度，從而獲得了最優的最大深度為 8。用一次調參後的模型來進行參數的訓練與擬合，從圖 7.2.6 中可以看出，訓練集分數是 86.53%，測試集的分數是 86.67%。調整參數後，訓練樣本分數下降，而對測試樣本的預測分數提高。

```
print('RandomForest训练集得分：',randomforest_param2.score(x_train, y_train))
print('RandomForest测试集得分：',randomforest_param2.score(x_test, y_test))
```

```
RandomForest训练集得分：  0.8652857142857143
RandomForest测试集得分：  0.8666666666666667
```

圖 7.2.6 二次調參隨機森林模型得分圖

與此同時，從圖 7.2.7 中可以看出，在進行了二次調參數之後，訓練集的均方誤差 達到了 0.1347，測試集的均方誤差達到了 0.1333，訓練集的均方誤差與測試集的均方誤差沒有太大的區別，並且測試集的均方誤差還有了下降。為此，我們選擇了經過二次修正的模型來進行資料的預測和評估。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.1347142857142857
Test MSE: 0.13333333333333333
```

圖 7.2.7 二次調參隨機森林模型均方誤差圖

7.3 隨機森林模型預測和評價

構建二次調參後隨機森林模型，並構建二次調參後模型測試預測結果的混淆矩陣表 7.3.1，我們可以知道真正例 TP 是 2366，假正例 FP 是 350，假反例 FN 是 50，真反例 TN 是 234。

表 7.3.1 隨機森林模型混淆矩陣表

	預測結果為 0（流失）	預測結果為 1（未流失）
實際結果為 0（流失）	2366	50
實際結果為 1（為流失）	350	234

利用所得真假正反例的個數計算出模型的精確率、召回率和整體準確率：

精確率 Precision：

$$\text{Precision}(\text{Exited} = 0) = \frac{TP}{TP + FP} = \frac{2366}{2366 + 350} = 87.11\%$$

$$\text{Precision}(\text{Exited} = 1) = \frac{TN}{TN + FN} = \frac{234}{234 + 50} = 82.39\%$$

召回率 Recall：

$$\text{Recall}(\text{Exited} = 0) = \frac{TP}{TP + FN} = \frac{2366}{2366 + 50} = 97.93\%$$

$$\text{Recall}(\text{Exited} = 1) = \frac{TN}{TN + FP} = \frac{234}{234 + 350} = 40.07\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2366 + 234}{2366 + 234 + 350 + 50} = 86.67\%$$

查看最終建立的隨機森林模型的準確率報表如表 7.3.2：

表 7.3.2 隨機森林模型準確率表

	Precision	Recall	F1-Score	Support
0	0.87	0.98	0.92	2416
1	0.82	0.40	0.54	584
Accuracy			0.87	3000
Macro Avg	0.85	0.69	0.73	3000
Weighted Avg	0.86	0.87	0.85	3000

通過對 ROC 曲線的繪製，我們可以看到二次調參後隨機森林模型的 ROC 曲線比最初的 ROC 曲線更加傾向于左上方，從圖 7.3.1 中可以看到，二次調參後隨機森林模型的 ROC 曲線下面的區域面積 AUC 值是 0.8521，與最初的模型的 AUC 值 0.8337 相比，這說明二次調參後的隨機森林分類模型更加優秀，是一個高品質的模型。

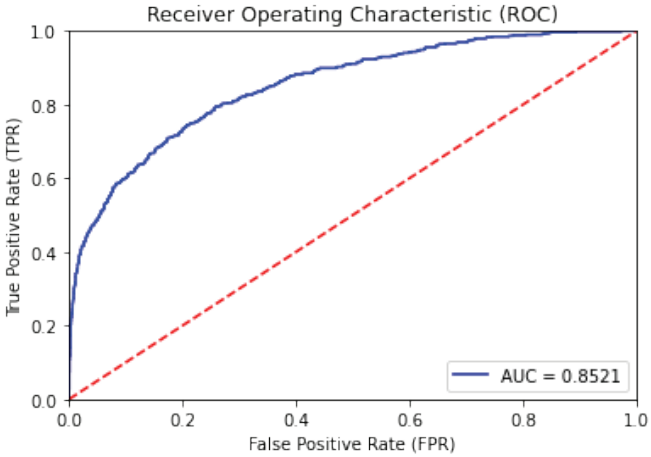


圖 7.3.1 隨機森林模型 ROC 曲線圖

8 極端梯度提升演算法

8.1 極端梯度提升演算法介紹

8.1.1 集成學習

集成學習（Ensemble Learning）指的是在構建多個獨立學習機器後，根據特定的策略，將各個學習器組合一起，形成一個強大的學習器來進行學習和執行任務。集成學習的類別為異質類和同質類，其中異質是指根據不同的學習演算法產生的個體學習器，而後者集成的學習器是基於同種類型的演算法產生的，是目前常用的集成學習器。在同質類的集成學習中，依據個體學習器間是否存在強依賴關係，分為分類提升演算法（Boosting）和裝袋演算法（Bagging）。

8.1.2 提升演算法

提升演算法（Boosting）是一種很好的機器學習手段，同時也是一種很好的有監督的分類器，通過融合多個較弱的分類器，可以得到一個很好的分類器。Boosting 演算法是先基於最初始的權重，通過使用訓練集的資料進行訓練學習，然後產生一個弱學習器，再對弱學習器的訓練學習誤差率進行計算，依據計算的結果，重新對新樣本的權重進行選擇，這對於提高上一個弱學習機器中學習率較高的訓練集樣本的權重是有利的，並且在後面的弱學習器中可以加強對高誤差率樣本的重視，迴圈重複得到規定的學習器個數後，最終依據策略模組進行結合，得到最終的強學習器。

Boosting 演算法的基本思想是每一次計算的目的都是降低上一次的殘差，並且在殘差變小的梯度方向上面構建出新的模型，以便每個新構建的模型都是使得上一個模型的殘差向梯度方向上減少 [10]。

8.1.3 極端梯度提升演算法

XGBoost 極端梯度提升演算法是一種有監督的學習演算法，它也由多棵決策樹組成，並且利用泰勒公式對目標函數去進行近似，進一步簡化目標函數的目的，能夠應用於回歸和分類的問題 [2]。XGBoost 模型在用訓練集進行訓練的時候，運用了貪心演算法，運用了正向分佈演算法來進行貪婪學習，當分裂某個結點的時候，在此基礎上，提出了一種基於局部最優解的演算法。在本次研究客戶流失的預測分類中採用 XGBoost 極端梯度提升演算法。

8.2 XGBoost 模型建立和求解

調用 sklearn 中的 XGBClassifier 包 ,用訓練集進行學習和訓練從而構建 XGBoost 模型，預設採用 100 個較弱的分類器，約束樹的最大深度是 6，而學習速率是 0.3。

在這個 XGBoost 模型下，如圖 8.2.1，可以看到這個 XGBoost 模型的分數，它的訓練集分數是 91.46%，而測試集分數是 85.30%。

```
print('XGBoosting训练集得分：',Xgboosting.score(X_train, y_train))
print('XGBoosting测试集得分：',Xgboosting.score(X_test, y_test))
```

```
XGBoosting训练集得分： 0.9145714285714286
XGBoosting测试集得分： 0.853
```

圖 8.2.1 初始 XGBoost 模型得分圖

查看該模型的均方誤差情況如圖 8.2.2，模型的訓練集均方誤差為 0.0854，測試集均方誤差為 0.1470。由此可以猜測，該模型可能存在過擬合的問題，所以要對該模型進行調參，從而優化模型。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.08542857142857142
Test MSE: 0.147
```

圖 8.2.2 初始 XGBoost 模型均方誤差圖

構建 XGBoost 初始模型測試預測結果的混淆矩陣表 8.2.1，可以得知真正例 TP 為 2275，假正例 FP 為 300，假反例 FN 為 141，真反例 TN 為 284。

表 8.2.1 初始 XGBoost 模型混淆矩陣表

	預測結果為 0（未流失）	預測結果為 1（流失）
實際結果為 0（未流失）	2275	141
實際結果為 1（流失）	300	284

根據所得真假正反例的個數，計算出模型的精確率、召回率和整體準確率。

精確率 Precision：

$$\text{Precision(Exited} = 0) = \frac{TP}{TP + FP} = \frac{2275}{2275 + 300} = 87.36\%$$

$$\text{Precision(Exited} = 1) = \frac{TN}{TN + FN} = \frac{284}{284 + 141} = 66.89\%$$

召回率 Recall：

$$\text{Recall(Exited} = 0) = \frac{TP}{TP + FN} = \frac{2275}{2275 + 141} = 93.68\%$$

$$\text{Recall(Exited} = 1) = \frac{TN}{TN + FP} = \frac{284}{284 + 300} = 48.48\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2275 + 284}{2275 + 284 + 300 + 141} = 84.27\%$$

表 8.2.2 初始 XGBoost 模型準確率表

	Precision	Recall	F1-Score	Support
0	0.88	0.94	0.91	2416
1	0.67	0.49	0.56	584
Accuracy			0.85	3000
Macro Avg	0.78	0.71	0.74	3000
Weighted Avg	0.84	0.85	0.84	3000

繪製出 ROC 曲線，從圖 8.2.3 中可以看出，ROC 曲線下的區域面積 AUC 值為 0.8439，屬於一個較為優秀的模型，但是因為有可能會出現過擬合，因此需要對模型的參數進行調節，這樣就減少了過度匹配的可能。

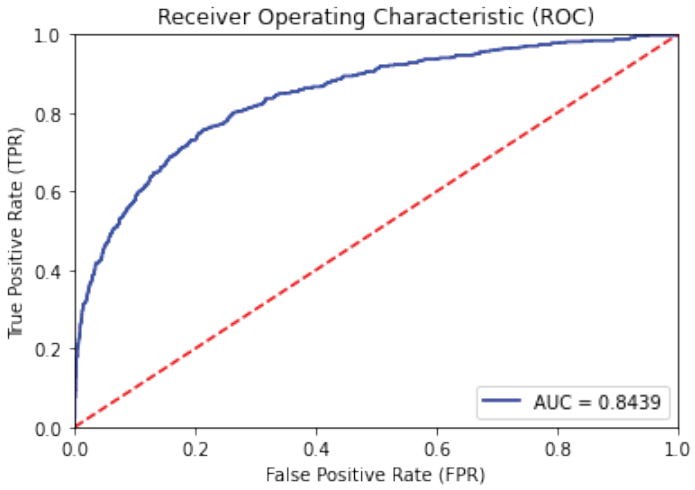


圖 8.2.3 初始 XGBoost 模型 ROC 曲線圖

由於初始的模型訓練集的均方誤差遠小於測試集的均方誤差，因此考慮到了模型過擬合的問題，對 XGBoost 模型進行第一次調參，利用網格搜索 GridSearchCV 對初始模型進行調參得到更優的分類模型。

對初始 XGBoost 模型進行弱學習器數量的調參選擇，利用 GridSearchCV 對初始模型進行最優弱學習器個數搜索，得到最優的弱學習器個數為 9 個。

XGBoost 模型在一次調整後的評分結果顯示在圖 8.2.4 中，該模型在一次調整後的模型中具有 86.16% 訓練得分和 86.73% 測試集得分。與調參前的模型比較，雖然一次調參後的模型的訓練集的得分下降了，但是測試集的得分有所上升，且兩者的得分情況不會有明顯較大的差距。

```
print('XGBoosting训练集得分：',Xgboosting_param1.score(X_train, y_train))
print('XGBoosting测试集得分：',Xgboosting_param1.score(X_test, y_test))
```

```
XGBoosting训练集得分： 0.8615714285714285
XGBoosting测试集得分： 0.8673333333333333
```

圖 8.2.4 一次調參 XGBoost 模型得分圖

一次調參後的 XGBoost 模型的均方誤差情況如圖 8.2.5，訓練集均方誤差為 0.1384，測試集均方誤差為 0.1327。

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
```

```
Train MSE: 0.13842857142857143
Test MSE: 0.13266666666666665
```

圖 8.2.5 一次調參 XGBoost 模型均方誤差圖

一次調參後的模型的測試集均方誤差出現了下降，而雖然訓練集均方誤差提升了，但是降低了過擬合情況的可能性，因此選用一次調參後的模型對資料進行預測和評估。

8.3 XGBoost 模型預測和評價

建立一次調參後 XGBoost 模型 構建一次調參後模型測試預測結果的混淆矩陣表 8.3.1，可以得知真正例 TP 為 2346，假正例 FP 為 328，假反例 FN 為 70，真反例 TN 為 256。

表 8.3.1 XGBoost 模型混淆矩陣表

	預測結果為 0（未流失）	預測結果為 1（流失）
實際結果為 0（未流失）	2346	70
實際結果為 1（流失）	328	256

利用所得真假正反例的個數計算出模型的精確率、召回率和整體準確率：

精確率 Precision：

$$\text{Precision(Exited} = 0) = \frac{TP}{TP + FP} = \frac{2346}{2346 + 328} = 87.73\%$$

$$\text{Precision(Exited} = 1) = \frac{TN}{TN + FN} = \frac{256}{256 + 70} = 78.53\%$$

召回率 Recall：

$$\text{Recall(Exited} = 0) = \frac{TP}{TP + FN} = \frac{2346}{2346 + 70} = 97.10\%$$

$$\text{Recall(Exited} = 1) = \frac{TN}{TN + FP} = \frac{256}{256 + 328} = 43.84\%$$

整體準確率 Accuracy：

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2346 + 256}{2346 + 256 + 328 + 70} = 86.73\%$$

查看最終建立的 XGBoost 模型的準確率報表如表 8.3.2：

表 8.3.2 XGBoost 模型準確率表

	Precision	Recall	F1-Score	Support
0	0.88	0.97	0.92	2416
1	0.79	0.44	0.56	584
Accuracy			0.87	3000
Macro Avg	0.83	0.70	0.74	3000
Weighted Avg	0.86	0.87	0.85	3000

繪畫出 ROC 曲線如圖 8.3.1，可以得到 ROC 曲線下的面積 AUC 值為 0.8520，AUC 值有了明顯的提升，而且整體的準確率提升到了 87%，因此這是一個品質比較好的模型，分類的準確率也更高。

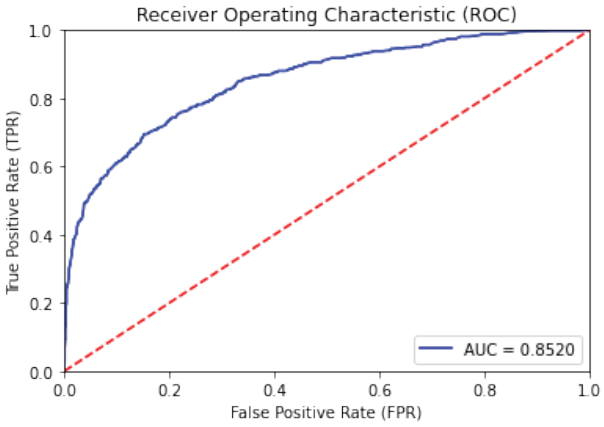


圖 8.3.1 XGBoost 模型的 ROC 曲線圖

9 模型比較和結論

9.1 模型比較

本文對 KNN 模型、BP 神經網路模型、SVM 支援向量機模型、DT 決策樹模型、RF 隨機森林模型以及 XGBoost 模型進行了研究，利用這 6 個模型的 Precision、Recall、F1-Score 和 Accuracy 進行比對比，如圖表 9.1.1。

表 9.1.1 六個模型的評價指標比較表

	KNN	BP	SVM	DT	RF	XGB
Recall	0.2021	0.4623	0.3613	0.4880	0.4007	0.4384
Precision	0.6982	0.6522	0.7903	0.6851	0.8239	0.7853
F1-Score	0.3134	0.5411	0.4959	0.5700	0.5392	0.5626
Accuracy	0.8277	0.8473	0.8570	0.8567	0.8667	0.8673

從該表 9.1.1 中可得知，DT 決策樹模型、RF 隨機森林模型和 XGBoost 模型這四種模型的四個評價指標較高，其他的 KNN 模型、SVM 支援向量機模型和 BP 神經網路模型相比於前面四種較差。

比較六種模型的 ROC 曲線以及 AUC 資料，如圖 9.1.1。

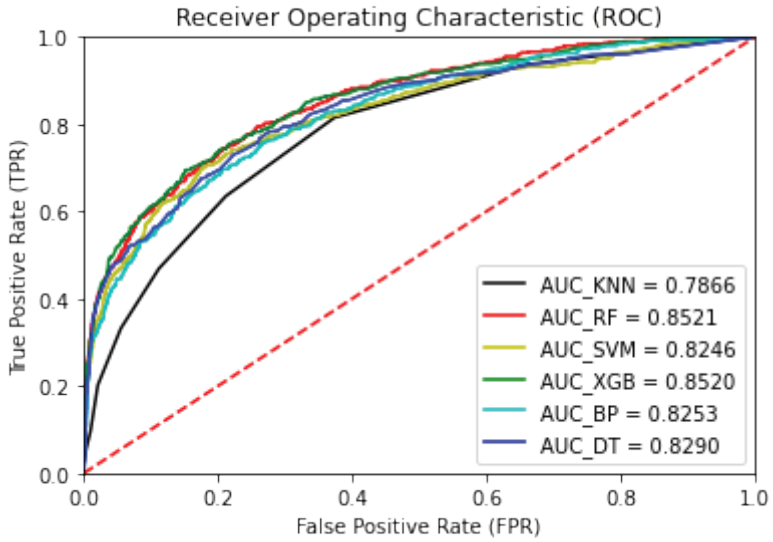


圖 9.1.1 六個機器學習模型的 ROC 曲線圖

從該圖 9.1.1 中可以看出，在 AUC 值中表現較好的是 RF 隨機森林模型和 XGBoost 模型，其中 KNN 模型的 AUC 值最差，其餘 SVM 支援向量機模型、BP 神經網路模型和 DT 決策樹模型的 AUC 值差不多，因此隨機森林模型和 XGBoost 模型的表現較好。RF 隨機森林的 AUC 值和 XGBoost 模型的 AUC 值差別不大 隨機森林模型略微高於 XGBoost 模型的 AUC 值。

Kappa 係數作為一種度量分類準確率的指標，在分類問題中，它可以測試所建立的模型的預測結果與真實的分類結果之間的一致性，所以可以用 Kappa 係數來評估分類模型，對 6 個模型進行了 Kappa 數值對比，如圖 9.1.2。

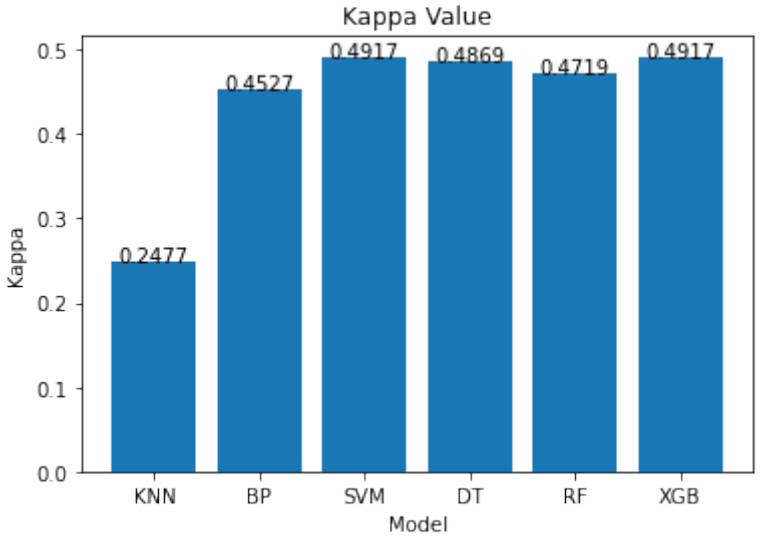


圖 9.1.2 六個機器學習模型的 Kappa 值對比圖

從該圖 9.1.2 中可以看到，KNN 模型的 Kappa 係數為 0.25 左右，表示 KNN 模型的一致性很差，而其它的 5 種模型的 Kappa 係數都為 0.45 到 0.5，表示它們的一致性為中等。在這些方法中，最顯著的是 XGBoost 模型和 SVM 支援向量機模型，它們的 Kappa 係數均為 0.49。

將以上的 Precision、Recall、F1、Accuracy、AUC 以及 Kappa 係數評價指標進行對比，XGBoost 模型的精度 Accuracy 是最好的，而且其它 3 個指標比較好，這說明該模型的預測結果的準確度較高，XGBoost 模型的 AUC 值和 Kappa 係數也較高。結果表明，XGBoost 模型具有較好的綜合性能，可以用來對我國商業銀行的客戶流失率進行預警和分類。這一模型可以為商業銀行今後的客戶保留與離開問題的研究，以及對客戶保留與離開問題的預測起到了預警作用。

9.2 結論

在本文中利用歐洲某一銀行的 10,000 條的資料為基礎資料，使用了六種不同的學習分類器演算法進行資料的研究和分析，從而得到了分類和預測銀行客戶流失情況的最佳分類器——XGBoost 分類模型。這一個分類器具有 87% 的整體準確率 、85% 的 AUC 值以及 0.49 的 Kappa 係數，能夠進一步研究銀行客戶流失的原因，並且能夠識別客戶流失潛在因素的範圍。在分析結果的基礎上，對客戶和銀行產品制定相關的策略和提出建議，從而有效的降低客戶的流失率和增強銀行的經營水準，具體有以下幾點結論：

第一，調查客戶滿意度以避免客戶流失。通過資料分析中可以發現在“Balance”、“HasCard”這兩個關於銀行業務方面的特徵值中，無論是有無銀行卡或者無論是帳戶餘額大小，都存在著一定的流失，因此可能存在業務制度和內容不完善的問題，銀行需要重視這些客戶和業務的制度內容，避免成為銀行長期客戶流失的缺口。可以向客戶進行問卷調查活動，根據客戶的滿意度對業務的制度內容進行相關修改，從而提高客戶對業務的滿意度。

第二，提高客戶忠誠度和銀行產品品質。銀行產品也對客戶的流失也具有主要的影響，當客戶手中持有該銀行的產品時，客戶出現離開該銀行的想法可能性比較低、客戶在銀行的留存時間會更久並且能夠會增加在銀行中參與其他業務的可能性。因此銀行在客戶方面，可以採取一些客戶回饋制度或者忠誠度計畫去提高客戶的忠誠度；在銀行產品方面，還可以通過與品牌合作、創新產品等手段，從而吸引客戶和提高客戶對產品的期待值。

第三，明確目標客戶群體。客戶是銀行的主要經濟命脈，銀行應該挖掘本行中的主要客戶群體、潛在客戶群體和流失客戶群體，不同的客戶群體需要設立不一樣的服務機制和策略活動。通過對年齡分佈的觀察，可以得知高年齡段的客戶群體具有較高的流失率，該銀行對於高年齡段的客戶群體沒有足夠重視。因此銀行可以將高年齡段的客戶設立為潛在客戶群體，同時需要制定以該群體為中心的專門服務和針對性策略，從而擴大高年齡段的客戶群體。

綜上所述，在資料分析和研究中，從客戶的銀行資訊中挖掘出客戶的價值，由此判斷銀行目前所面臨的客戶流失情況和銀行當前存在客戶方案和制度情況。利用機器學習演算法，銀行可以獲取客戶資訊的價值，預測和揭示潛在的客戶流失情況，並提供資訊圖表以說明銀行全方面分析和瞭解客戶的狀況，客觀地指出銀行存在的問題，為提高客戶滿意度和忠誠度、明確客戶群體、提高產品品質和吸引力提供有限的建議和方法，給予銀行有效的建議和途徑。最終實現提高銀行整體競爭力和銀行魅力的目標，讓銀行的經營和管理水準邁向新的臺階。

參考文獻

[1] 史翼璿 . 基於資料採擷的銀行客戶流失預測的研究 [D]. 內蒙古大學 ,2022.

[2] 張孟迪 . 基於 Logistic 回歸和 XGBoost 的銀行信用卡客戶流失預測 [D], 山東大學 ,2021.

[3] 杜紅 . 城市商業銀行客戶流失風險預警與應對策略研究 [D]. 華北電力大學 ,2019.

[4] 張莉莉 . 基於集成模型的銀行個人客戶流失預警 [D]. 蘭州大學 ,2021.

[5] 時丹蕾 , 杜寶軍 . 基於 BP 神經網路的銀行客戶流失預測 [J]. 科學技術創新 ,2021(27):104-106.

[6] 石寒 . 基於重採樣和集成學習的信用卡客戶流失預測研究 [D]. 華中農業大學 ,2022.
[7] 盧美琴 , 吳傳威 . 大資料背景下商業銀行貴賓客戶流失的組合預測研究 [J]. 電子商務 ,2019(06):55-57.
[8] 沈哲 . 基於資料採擷的 G 銀行信用卡客戶流失預測研究 [J]. 中國市場 ,2022(08):49-52.
[9] 姚博 . 客戶流失預測模型研究及其應用 [D]. 西北大學 ,2017.
[10] 程勇 , 梁吉祥 . 基於資料採擷的掌銀客戶流失預測建模方法研究 [J]. 中國金融電腦 ,2019(08):51-60.

附錄

```
# 資料導入代碼：
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
data=pd.read_csv( ‘C:/Users/Ken/Desktop/ 畢業論文 /churn.csv’ )
# 處理和分析資料：
data        # 查看數據
data.describe()  # 資料描述性分析
data.columns     # 查看數據的列名
data.shape       # 查看數據維度
data.info()      # 查看資料情況
data.isnull().sum()       # 資料缺失值統計
data.describe(include=[‘O’]).T          # 查找文字變數
# 根據文字變數進行分類
data[[‘Exited’,’Geography’]].groupby([‘Geography’],as_index=False).count()
data[[‘Exited’,’Gender’]].groupby([‘Gender’],as_index=False).count()
data=data.drop([‘RowNumber’,’CustomerId’,’Surname’],axis=1)  # 刪除無關變數
# 轉換文字變數為資料類型
data[‘Gender’]=data[‘Gender’].map({‘Female’:1,’Male’:0}).astype(int)
data[‘Geography’]=data[‘Geography’].map({‘France’:2,’Germany’:1,’Spain’:0}).
astype(int)
# 單變數視覺化
Exited=data[‘Exited’].value_counts()
plt.pie(Exited,labels=list(Exited.index),autopct=’%.2f%%’)
Gender=data[‘Gender’].value_counts()
plt.pie(Gender,labels=list(Gender.index),autopct=’%.2f%%’)
```

```
Geography=data['Geography'].value_counts()
plt.pie(Geography,labels=list(Geography.index),autopct='%.2f%%')
HasCrCard=data['HasCrCard'].value_counts()
plt.pie(HasCrCard,labels=list(HasCrCard.index),autopct='%.2f%%')
value,bins,bars=plt.hist(data['Age'])
plt.xlabel('Age')
plt.ylabel('Number of Customers')
plt.bar_label(bars)
plt.show
value,bins,bars=plt.hist(data['EstimatedSalary'])
plt.xlabel('EstimatedSalary')
plt.ylabel('Number of Customers')
plt.bar_label(bars)
plt.show
# 雙變數視覺化分析
for i in data.columns:
    p=sns.FacetGrid(data,col='Exited')
    p.map(plt.hist,i)
# 資料清洗和轉換
Age=[]
for i in data['Age']:
    if int(i)>=70:
        Age.append(5)
    elif int(i)>=60:
        Age.append(4)
    elif int(i)>=50:
        Age.append(3)
    elif int(i)>=40:
        Age.append(2)
    elif int(i)>=30:
        Age.append(1)
```

```
    elif int(i)<30:
        Age.append(0)
CreditScore=pd.cut(x=data['CreditScore'],bins=5,labels=range(0,5))
EstimatedSalary=pd.cut(x=data['EstimatedSalary'],bins=5,labels=range(0,5))
Balance=pd.cut(x=data['Balance'],bins=5,labels=range(0,5))
data['Age']=Age
data['CreditScore']=CreditScore
data['EstimatedSalary']=EstimatedSalary
data['Balance']=Balance
data
# 箱線圖
plt.title('Examples of boxplot',fontsize=20)# 標題，並設定字型大小大小
plt.boxplot([data['CreditScore'],data['Geography'] , data['Gender'], data['Age'],data['Tenure'], data['Balance'],
        ], labels = ['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',])
plt.show()
plt.figure(figsize=(9,4))
plt.boxplot([data['NumOfProducts'], data['HasCrCard'], data['IsActiveMember'], data['EstimatedSalary'],data['Exited']], labels = ['NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary','Exited'])
plt.show()
# 劃分資料集
Y = data['Exited']
X = data.drop(['Exited'],axis=1)
feature=list(X.columns)
t='Exited'
x_train,x_test,y_train,y_test= train_test_split(X,Y,test_size = 0.3,random_state=42)

# KNN 模型
from sklearn.neighbors import KNeighborsClassifier
' 在 K 取值為 1-20 的模型下，每個模型的錯誤率計算 '
```

```
error = [] # 存放每個 K 的預測錯誤率
for i in range(1,21): # 對迴圈 20 個 K
    knn = KNeighborsClassifier(n_neighbors=i)# 建模
    knn.fit(x_train, y_train) # 模型訓練
    pred_i = knn.predict(x_test) # 模型預測
    error.append(np.mean(pred_i != y_test)) # 計算錯誤率
kmin=error.index(min(error))+1 # 選取錯誤率最小時的 k 值
print(error) # 輸出 20 個 K 的預測錯誤率
‘20 個模型錯誤率的折線圖 ’
plt.figure(figsize=(12, 6))
plt.plot(range(1, 21), error, color=’red’, linestyle=’dashed’, marker=’o’,
        markerfacecolor=’blue’, markersize=10)
plt.title(‘Error Rate K Value’)
plt.xlabel(‘K Value’)
plt.ylabel(‘Mean Error’)
k_plot=[]# 存放測試集準確率
t_plot=[]# 存放訓練集準確率
‘ 在 K 取值為 1-20 的模型下，每個模型的訓練集準確率和測試集準確率計算 ’
for k in range(1,21,1):
    dt= KNeighborsClassifier(n_neighbors=k) # 建立模型
    dt.fit(x_train,y_train) # 擬合
    accuracy_test=round(dt.score(x_test,y_test)*100,2) # 計算測試集的得分
    accuracy_train=round(dt.score(x_train,y_train)*100,2) # 計算訓練集的得分
    k_plot.append(accuracy_test) # 獲取每次鄰近點的測試得分
    t_plot.append(accuracy_train) # 獲取每次鄰近點的訓練得分
‘20 個模型訓練集和測試準確率的折線圖 ’
fig,axes=plt.subplots(1,1,figsize=(12,8))
axes.set_xticks(range(1,21,1))
plt.title( “accuracy of K Neighbors classifier” )
plt.xlabel( “K-neighbors” , color = “purple” )
plt.ylabel( “accuracy” , color = “green” )
```

```
k=range(1,21,1)
plt.plot(k,k_plot,linewidth = 3.0, linestyle = '--',marker = "o" )
plt.plot(k,t_plot,'r',marker = "o" ,markerfacecolor = 'white')
plt.legend(['accuracy_test','accuracy_train'])
' 選取錯誤率較小，訓練集和測試集擬合準確率適合的 K 值進行建模 '
knn_best = KNeighborsClassifier(n_neighbors=kmin) # 建模
knn_best.fit(x_train, y_train) # 擬合
pred=knn_best.predict(x_test) # 預測結果
print('KNN 預測分類得分：',knn_best.score(x_test,y_test)) # 預測準確率
print()
from sklearn.metrics import confusion_matrix
m=confusion_matrix(y_test,pred) # 預測結果和 y_test 的混淆矩陣
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
display(a)
print()
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, pred))
TP_KNN=confusion_matrix(y_test,pred)[0][0]
FN_KNN=confusion_matrix(y_test,pred)[0][1]
FP_KNN=confusion_matrix(y_test,pred)[1][0]
TN_KNN=confusion_matrix(y_test,pred)[1][1]
from sklearn.metrics import recall_score, f1_score, precision_score, accuracy_score
recall_KNN = recall_score(y_test, pred,pos_label=1)
precision_KNN = precision_score(y_test, pred,pos_label=1)
f1_KNN = f1_score(y_test, pred,pos_label=1)
accuracy_KNN =accuracy_score(y_test, pred)
print('KNN 訓練集得分：',knn_best.score(x_train,y_train))
print('KNN 預測分類得分：',knn_best.score(x_test,y_test))
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs = knn_best.predict_proba(x_test)
```

```
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()

# BP 神經網路
from sklearn.neural_network import MLPClassifier
train_score=[]
test_score=[]
lost=[]
for i in range(1,21):
    BP = MLPClassifier(solver='adam', activation='logistic',hidden_layer_
sizes=(i),learning_rate_init=0.1,random_state=1)
    BP.fit(x_train, y_train)
    y_pred=BP.predict(x_test)
    train_score.append(BP.score(x_train,y_train))
    test_score.append(BP.score(x_test,y_test))
    lost.append(BP.loss_)
fig,axes=plt.subplots(1,1,figsize=(12,8))
axes.set_xticks(range(1,21,1))
plt.title( "accuracy of BP classifier" )
```

```
plt.xlabel(“E”, color =“purple”)
plt.ylabel(“accuracy”, color =“green”)
k=range(1,21,1)
plt.plot(k,train_score,linewidth = 3.0, linestyle = ‘--’,marker =“o”)
plt.plot(k,test_score,’r’,marker =“o”,markerfacecolor = ‘white’)
plt.legend([‘train_score’,’test_score’])
print(‘MaxTrainScore:’,int(train_score.index(max(train_score))+1),’MaxTestScore:’,int(te
st_score.index(max(test_score))+1))
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(range(1, 21), lost, color=’red’, linestyle=’dashed’, marker=’o’,
        markerfacecolor=’blue’, markersize=10)
plt.title(‘Lost Rate’)
plt.xlabel(‘E Value’)
plt.ylabel(‘Lost’)
hidden_layer_sizes=int(lost.index(min(lost))+1)
print(‘E:’,int(lost.index(min(lost))+1),’minLost:’,min(lost))
BP_best = MLPClassifier(solver=’adam’, activation=’logistic’,hidden_layer_
sizes=(12),learning_rate_init=0.1,random_state=1)
BP_best.fit(x_train, y_train)
y_pred=BP.predict(x_test)
print(‘BP 神經網路損失：’,BP_best.loss_)
print(‘BP 神經網路訓練集得分：’,BP_best.score(x_train,y_train))
print(‘BP 神經網路測試集得分：’,BP_best.score(x_test,y_test))
pred_train=BP_best.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = BP_best.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print(‘Train MSE:’, mean_squared_error(y_train,pred_train))
print(‘Test MSE:’, mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
```

```
pred = BP_best.predict(x_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
display(a)
TP_BP=confusion_matrix(y_test,pred)[0][0]
FN_BP=confusion_matrix(y_test,pred)[0][1]
FP_BP=confusion_matrix(y_test,pred)[1][0]
TN_BP=confusion_matrix(y_test,pred)[1][1]
from sklearn.metrics import recall_score
recall_BP = recall_score(y_test,pred ,pos_label=1)
precision_BP = precision_score(y_test, pred,pos_label=1)
f1_BP = f1_score(y_test, pred,pos_label=1)
accuracy_BP =accuracy_score(y_test, pred)
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs = BP_best.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---fi3nd the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
```

```
# SVM 模型
from sklearn.svm import SVC
Supportvactormachine = SVC(kernel='rbf',probability=True)
Supportvactormachine.fit(x_train,y_train)
print('Supportvactormachine 訓 練 集 得 分：',Supportvactormachine.score(x_train, y_
train))  # 精度
print('Supportvactormachine 測試集得分：',Supportvactormachine.score(x_test, y_test))
pred_train=Supportvactormachine.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = Supportvactormachine.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = Supportvactormachine.predict(x_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs = Supportvactormachine.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
```

```
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
param_test1 = {'C':[10,20,30,40,50]}
param_best1 = GridSearchCV(Supportvactormachine, param_test1,cv=3,verbose=1,n_jobs=-1)
param_best1.fit(x_train,y_train)
C=param_best1.best_params_['C']
print(param_best1.best_params_)
print(param_best1.best_score_)
Supportvactormachine_param1= SVC(kernel='rbf',C=C,probability=True)
Supportvactormachine_param1.fit(x_train, y_train)
print('Supportvactormachine 訓 練 集 得 分：',Supportvactormachine_param1.score(x_train, y_train))
print('Supportvactormachine 測試集得分：',Supportvactormachine_param1.score(x_test, y_test))
pred_train=Supportvactormachine_param1.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = Supportvactormachine_param1.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = Supportvactormachine_param1.predict(x_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
TP_SVM=confusion_matrix(y_test,pred)[0][0]
FN_SVM=confusion_matrix(y_test,pred)[0][1]
FP_SVM=confusion_matrix(y_test,pred)[1][0]
```

```
TN_SVM=confusion_matrix(y_test,pred)[1][1]
from sklearn.metrics import recall_score
recall_SVM = recall_score(y_test, pred,pos_label=1)
precision_SVM = precision_score(y_test, pred,pos_label=1)
f1_SVM = f1_score(y_test, pred,pos_label=1)
accuracy_SVM =accuracy_score(y_test, pred)
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs = Supportvactormachine_param1.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()

# 決策樹模型
from sklearn.tree import DecisionTreeClassifier
DecisionTree=DecisionTreeClassifier()
DecisionTree.fit(x_train,y_train)
print('DecisionTree 訓練集得分：',DecisionTree.score(x_train, y_train))
print('DecisionTree 測試集得分：',DecisionTree.score(x_test, y_test))
pred_train=DecisionTree.predict(x_train)
```

```
mse_train=mean_squared_error(y_train,pred_train)
pred_test = DecisionTree.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = DecisionTree.predict(x_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
display(a)
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs = DecisionTree.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
from sklearn.model_selection import GridSearchCV
param_test1={'max_depth':np.arange(1,20)}
param_best1=GridSearchCV(estimator=DecisionTree,param_grid=param_test1)
```

```
param_best1.fit(x_train,y_train)
max_depth=param_best1.best_params_['max_depth']
print(param_best1.best_params_)
print(param_best1.best_score_)
DecisionTree_param1= DecisionTreeClassifier(max_depth=max_depth)
DecisionTree_param1.fit(x_train, y_train)
print('DecisionTree 訓練集得分：',DecisionTree_param1.score(x_train, y_train))
print('DecisionTree 測試集得分：',DecisionTree_param1.score(x_test, y_test))
pred_train=DecisionTree_param1.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = DecisionTree_param1.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = DecisionTree_param1.predict(x_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
display(a)
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs = DecisionTree_param1.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
```

```
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
param_test2={'min_samples_leaf':np.arange(1,6)}
param_best2=GridSearchCV(estimator=DecisionTree_param1,param_grid=param_test2)
param_best2.fit(x_train,y_train)
min_samples_leaf=param_best2.best_params_['min_samples_leaf']
print(param_best2.best_params_)
print(param_best2.best_score_)
DecisionTree_param2= DecisionTreeClassifier(max_depth=max_depth,min_samples_
leaf=min_samples_leaf)
DecisionTree_param2.fit(x_train, y_train)
print('DecisionTree 訓練集得分：',DecisionTree_param2.score(x_train, y_train))
print('DecisionTree 測試集得分：',DecisionTree_param2.score(x_test, y_test))
pred_train=DecisionTree_param2.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = DecisionTree_param2.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = DecisionTree_param2.predict(x_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
display(a)
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

```
probs = DecisionTree_param2.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
param_test3={'min_samples_split':np.arange(10,50,10)}
param_best3=GridSearchCV(estimator=DecisionTree_param2,param_grid=param_test3)
param_best3.fit(x_train,y_train)
min_samples_split=param_best3.best_params_['min_samples_split']
print(param_best3.best_params_)
print(param_best3.best_score_)
DecisionTree_param3= DecisionTreeClassifier(max_depth=max_depth,min_samples_
leaf=min_samples_leaf,min_samples_split=min_samples_split)
DecisionTree_param3.fit(x_train, y_train)
print('DecisionTree 訓練集得分：',DecisionTree_param3.score(x_train, y_train))
print('DecisionTree 測試集得分：',DecisionTree_param3.score(x_test, y_test))
pred_train=DecisionTree_param3.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = DecisionTree_param3.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
```

```
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = DecisionTree_param3.predict(x_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
display(a)
TP_DT=confusion_matrix(y_test,pred)[0][0]
FN_DT=confusion_matrix(y_test,pred)[0][1]
FP_DT=confusion_matrix(y_test,pred)[1][0]
TN_DT=confusion_matrix(y_test,pred)[1][1]
from sklearn.metrics import recall_score
recall_DT = recall_score(y_test, pred,pos_label=1)
precision_DT = precision_score(y_test, pred,pos_label=1)
f1_DT = f1_score(y_test, pred,pos_label=1)
accuracy_DT =accuracy_score(y_test, pred)
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs = DecisionTree_param3.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
```

```
plt.legend(loc = 'lower right')
plt.show()

# Random Forest 模型
from sklearn.ensemble import RandomForestClassifier
randomforest=RandomForestClassifier()
randomforest.fit(x_train,y_train)
print('RandomForest 訓練集得分：',randomforest.score(x_train, y_train))
print('RandomForest 測試集得分：',randomforest.score(x_test, y_test))
from sklearn.metrics import classification_report, confusion_matrix, mean_squared_error
pred_train=randomforest.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = randomforest.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
print(classification_report(y_test, pred_test)) # 模型評估報告
m=confusion_matrix(y_test,pred_test) # 混淆矩陣
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
display(a)
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs = randomforest.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
```

```
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
from sklearn.model_selection import GridSearchCV
param_test1={'n_estimators':np.arange(50,501,50)}
param_best1=GridSearchCV(estimator=RandomForestClassifier(),param_grid=param_
test1,scoring='roc_auc')
param_best1.fit(x_train,y_train)
estimators=param_best1.best_params_['n_estimators']
print(param_best1.best_params_)
print(param_best1.best_score_)
randomforest_param1= RandomForestClassifier(n_estimators=estimators)
randomforest_param1.fit(x_train, y_train)
print('RandomForest 訓練集得分：',randomforest_param1.score(x_train, y_train))
print('RandomForest 測試集得分：',randomforest_param1.score(x_test, y_test))
pred_train=randomforest_param1.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = randomforest_param1.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = randomforest_param1.predict(x_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

```
probs = randomforest_param1.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
param_test2={'max_depth':np.arange(3,15)}
param_best2=GridSearchCV(estimator=RandomForestClassifier(n_
estimators=estimators),param_grid=param_test2,scoring='roc_auc')
param_best2.fit(x_train,y_train)
m_depth=param_best2.best_params_['max_depth']
print(param_best2.best_params_)
print(param_best2.best_score_)
randomforest_param2= RandomForestClassifier(n_estimators=estimators,max_depth=m_
depth)
randomforest_param2.fit(x_train, y_train)
print('RandomForest 訓練集得分：',randomforest_param2.score(x_train, y_train))
print('RandomForest 測試集得分：',randomforest_param2.score(x_test, y_test))
pred_train=randomforest_param2.predict(x_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = randomforest_param2.predict(x_test)
mse_test=mean_squared_error(y_test,pred_test)
```

```
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = randomforest_param2.predict(x_test)
print(classification_report(y_test, pred_test))
m=confusion_matrix(y_test,pred_test)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
TP_RF=confusion_matrix(y_test,pred)[0][0]
FN_RF=confusion_matrix(y_test,pred)[0][1]
FP_RF=confusion_matrix(y_test,pred)[1][0]
TN_RF=confusion_matrix(y_test,pred)[1][1]
from sklearn.metrics import recall_score
recall_RF = recall_score(y_test, pred,pos_label=1)
precision_RF = precision_score(y_test, pred,pos_label=1)
f1_RF = f1_score(y_test, pred,pos_label=1)
accuracy_RF =accuracy_score(y_test, pred)
import matplotlib.pyplot as plt
probs = randomforest_param2.predict_proba(x_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
```

```
plt.show()

# XGBoost 模型
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.fit_transform(x_test)
from xgboost import XGBClassifier
Xgboosting=XGBClassifier()
Xgboosting.fit(X_train,y_train)
print('XGBoosting 訓練集得分：',Xgboosting.score(X_train, y_train))
print('XGBoosting 測試集得分：',Xgboosting.score(X_test, y_test))
pred_train=Xgboosting.predict(X_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = Xgboosting.predict(X_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = Xgboosting.predict(X_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
probs = Xgboosting.predict_proba(X_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
```

```
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
param_test1 = {'n_estimators':np.arange(0,10)}
param_best1 = GridSearchCV(Xgboosting, param_test1,cv=10,verbose=1,n_jobs=-1)
param_best1.fit(X_train,y_train)
n_estimators=param_best1.best_params_['n_estimators']
print(param_best1.best_params_)
print(param_best1.best_score_)
Xgboosting_param1=XGBClassifier(n_estimators=n_estimators)
Xgboosting_param1.fit(X_train,y_train)
print('XGBoosting 訓練集得分：',Xgboosting_param1.score(X_train, y_train))
print('XGBoosting 測試集得分：',Xgboosting_param1.score(X_test, y_test))
pred_train=Xgboosting_param1.predict(X_train)
mse_train=mean_squared_error(y_train,pred_train)
pred_test = Xgboosting_param1.predict(X_test)
mse_test=mean_squared_error(y_test,pred_test)
print('Train MSE:', mean_squared_error(y_train,pred_train))
print('Test MSE:', mean_squared_error(y_test,pred_test))
from sklearn.metrics import classification_report, confusion_matrix
pred = Xgboosting_param1.predict(X_test)
print(classification_report(y_test, pred))
m=confusion_matrix(y_test,pred)
a=pd.DataFrame(m,index=['0 實際 ','1 實際 '],columns=['0 預測 ','1 預測 '])
TP_XGB=confusion_matrix(y_test,pred)[0][0]
FN_XGB=confusion_matrix(y_test,pred)[0][1]
FP_XGB=confusion_matrix(y_test,pred)[1][0]
```

```
TN_XGB=confusion_matrix(y_test,pred)[1][1]
from sklearn.metrics import recall_score
recall_XGB = recall_score(y_test,pred ,pos_label=1)
precision_XGB = precision_score(y_test, pred,pos_label=1)
f1_XGB = f1_score(y_test, pred,pos_label=1)
accuracy_XGB =accuracy_score(y_test, pred)
probs = Xgboosting_param1.predict_proba(X_test)
preds = probs[:,1]
#---find the FPR, TPR, and threshold---
fpr, tpr, threshold = roc_curve(y_test, preds)
#---find the area under the curve---
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()

# 總體模型分析
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
probs_KNN = knn_best.predict_proba(x_test)
probs_RF = randomforest_param2.predict_proba(x_test)
probs_SVM = Supportvactormachine_param2.predict_proba(x_test)
probs_XGB = Xgboosting_param2.predict_proba(X_test)
probs_BP = BP_best.predict_proba(x_test)
probs_DT = DecisionTree_param3.predict_proba(x_test)
```

```
preds_KNN = probs_KNN[:,1]
preds_RF = probs_RF[:,1]
preds_SVM = probs_SVM[:,1]
preds_XGB = probs_XGB[:,1]
preds_BP = probs_BP[:,1]
preds_DT = probs_DT[:,1]

#---find the FPR, TPR, and threshold---
fpr_KNN, tpr_KNN, threshold_KNN = roc_curve(y_test, preds_KNN)
fpr_RF, tpr_RF, threshold_RF = roc_curve(y_test, preds_RF)
fpr_SVM, tpr_SVM, threshold_SVM = roc_curve(y_test, preds_SVM)
fpr_XGB, tpr_XGB, threshold_XGB = roc_curve(y_test, preds_XGB)
fpr_BP, tpr_BP, threshold_BP = roc_curve(y_test, preds_BP)
fpr_DT, tpr_DT, threshold_DT = roc_curve(y_test, preds_DT)
#---find the area under the curve---
roc_auc_KNN = auc(fpr_KNN, tpr_KNN)
roc_auc_RF = auc(fpr_RF, tpr_RF)
roc_auc_SVM = auc(fpr_SVM, tpr_SVM)
roc_auc_XGB = auc(fpr_XGB, tpr_XGB)
roc_auc_BP = auc(fpr_BP, tpr_BP)
roc_auc_DT = auc(fpr_DT, tpr_DT)

plt.plot(fpr_KNN, tpr_KNN, 'b', label = 'AUC_KNN = %0.4f' % roc_auc_
KNN,color='k')
plt.plot(fpr_RF, tpr_RF, 'b', label = 'AUC_RF = %0.4f' % roc_auc_RF,color='r')
plt.plot(fpr_SVM, tpr_SVM, 'b', label = 'AUC_SVM = %0.4f' % roc_auc_
SVM,color='y')
plt.plot(fpr_XGB, tpr_XGB, 'b', label = 'AUC_XGB = %0.4f' % roc_auc_XGB,color='g')
plt.plot(fpr_BP, tpr_BP, 'b', label = 'AUC_BP = %0.4f' % roc_auc_BP,color='c')
plt.plot(fpr_DT, tpr_DT, 'b', label = 'AUC_DT = %0.4f' % roc_auc_DT,color='b')
```

```
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc = 'lower right')
plt.show()
pd.DataFrame({'Recall' :[recall_KNN,recall_BP,recall_SVM,recall_DT,recall_RF,recall_
XGB],
                'Precision':[precision_KNN,precision_BP,precision_SVM,precision_
DT,precision_RF,precision_XGB],
            'F1':[f1_KNN,f1_BP,f1_SVM,f1_DT,f1_RF,f1_XGB],
                'Accuracy':[accuracy_KNN,accuracy_BP,accuracy_SVM,accuracy_
DT,accuracy_RF,accuracy_XGB]},
                index = ['KNN','BP','SVM','DecisitonTree','RandomForest','XGBoost'])
def ka(TP,FN,FP,TN):
    po=(TP+TN)/(TP+FP+TN+FN)
    pe=((TP+FN)*(TP+FP)+(TN+FN)*(FP+TN))/((TP+FP+TN+FN)*(TP+FP+TN+FN))
    kappa=(po-pe)/(1-pe)
    return kappa
kappa_KNN=ka(TP_KNN,FN_KNN,FP_KNN,TN_KNN)
kappa_BP=ka(TP_BP,FN_BP,FP_BP,TN_BP)
kappa_SVM=ka(TP_SVM,FN_SVM,FP_SVM,TN_SVM)
kappa_DT=ka(TP_DT,FN_DT,FP_DT,TN_DT)
kappa_RF=ka(TP_RF,FN_RF,FP_RF,TN_RF)
kappa_XGB=ka(TP_XGB,FN_XGB,FP_XGB,TN_XGB)
kappa=[kappa_KNN,kappa_BP,kappa_SVM,kappa_DT,kappa_RF,kappa_XGB]
model=['KNN','BP','SVM','DT','RF','XGB']
plt.bar(model,kappa)
```

```
plt.title('Kappa Value')
plt.xlabel('Model')
plt.ylabel('Kappa')
for a,b in zip(model,kappa):
    plt.text(a,b,round(b,4),ha='center')
plt.show()
```