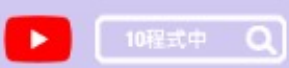
全民瘋AI系列 [探索可解釋人工智慧]

第15屆iT邦幫忙鐵人賽 AI & Data 組



Tsai Yi Lin

Table of contents

1.XAI基礎與概念介紹

[Day 1] 揭開模型的神秘面紗：為何XAI對機器學習如此重要？

[Day 2] 從黑盒到透明化：XAI技術的發展之路

[Day 3] 機器學習中的可解釋性指標

[Day 4] LIME vs. SHAP：哪種XAI解釋方法更適合你？

[Day 5] 淺談XAI與傳統機器學習的區別

2.XAI在傳統機器學習中的應用

[Day 6] 非監督學習也能做到可解釋性？探索XAI在非監督學習中的應用

3.XAI常用工具介紹

[Day 12] LIME理論：如何用局部線性近似解釋黑箱模型

[Day 13] LIME實作：實戰演練LIME解釋方法

[Day 14] SHAP理論：解析SHAP解釋方法的核心

[Day 15] SHAP實作：實戰演練SHAP解釋方法

4.XAI在深度學習中的可解釋性

[Day 16] 神經網路的可解釋性：如何理解深度學習中的黑箱模型？

[Day 23] Attention-Based：使用注意力機制解釋CNN模型

5.XAI在現實生活中的應用案例

[Day 24] LSTM的可解釋性：從時序資料解析人體姿態預測

[Day 25] XAI在影像處理中的瑕疵檢測：解釋卷積神經網路的運作

6.XAI的挑戰與未來

[Day 28] 對抗樣本的挑戰：如何利用XAI檢測模型的弱點？

[Day 29] XAI如何影響人類對技術的信任和接受程度？

[Day30] XAI未來發展方向：向更可靠的機器學習模型邁進

1.XAI基礎與概念介紹

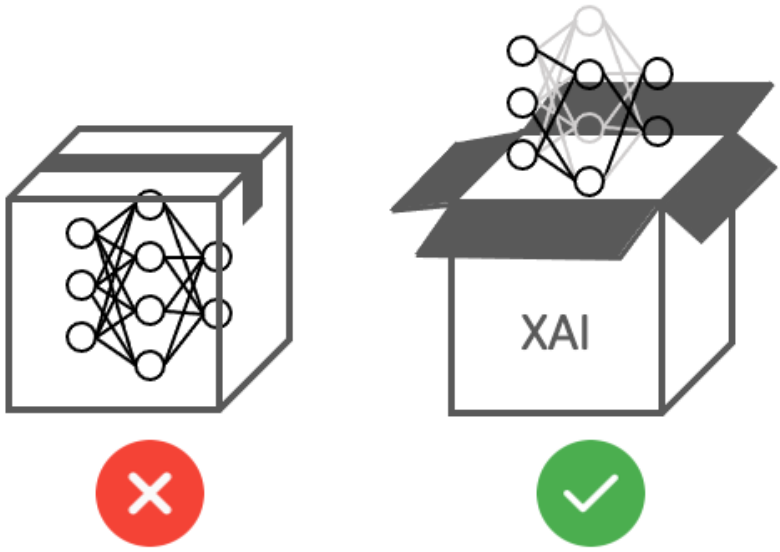
[Day 1] 揭開模型的神秘面紗：為何XAI對機器學習如此重要？

人工智慧的發展已經進入了一個新的階段，作為AI的重要分支「機器學習」，已經被廣泛應用於各個領域，例如語音識別、圖像分類、自然語言處理.....等。然而隨著機器學習技術的發展，特別是模型「黑箱」特性，使得其決策過程變得越來越不透明，且缺乏對外部人員的解釋和理解，這也使得機器學習在一些關鍵領域難以得到應用。因此這就引出了一個重要的議題：我們該如何確保機器學習模型的可解釋性？為此，可解釋人工智慧（Explainable AI，XAI）的出現成為了一個重要的研究方向。

> XAI　旨在提高機器學習模型的可解釋性，使人們能夠理解模型的工作原理，進一步提高機器學習模型的可信度和可靠性。

為何需要解釋模型？

在人工智慧領域中，模型解釋性已經成為一個重要的課題。過去機器學習模型被視為一個黑箱，因為它們的決策過程往往過於複雜而難以解釋。這個問題已經引起了關注，對於那些需要理解模型背後的邏輯和決策過程的人來說，黑箱模型是不可接受的。



簡單來說，揭開模型的神秘面紗可以讓人們理解機器學習模型如何進行預測和決策。這種理解對於提高機器學習模型的可靠性和可信度至關重要。

XAI 所扮演的角色

假設我們有一個卷積神經網路（CNN）模型，用於分類圖像中的物體種類。當我們輸入一張照片給模型時，模型會給出一個預測結果，並告訴我們照片中的物體種類是什麼。但是當模型給出錯誤的預測時，我們無法了解模型為何做出這樣的決策。這時候模型解釋的技術就可以幫助我們理解模型的決策過程，例如哪些區域對預測結果有較大的影響，哪些特徵對預測結果貢獻較大等等。以下舉一個經典的影像分類案例，也就是大家所熟知的貓狗分類。曾經的我以為已經訓練了一個辨識能力極高的貓狗分類器，但是在實際驗證集資料中的準確率不到一半，這很明顯是模型過擬合 *(https://ithelp.ithome.com.tw/articles/10278254)*的結果。最後使用可解釋的技術對模型進行解析，發現模型並非真的學到辨識貓狗的關鍵特徵。罪魁禍首竟然是因為訓練集中有許多的貓咪照片都戴著鈴鐺項圈，導致模型誤認項圈為辨識貓狗的關鍵特徵。這個案例顯示了模型解釋的重要性，因為如果無法理解模型背後的邏輯和決策過程，那麼模型的效能可能會受到負面的影響。

上述是一個經典的影像辨識 XAI 技術的案例，從中可以發現訓練一個機器學習模型所使用的資料品質極為重要。Landing.ai 執行長吳恩達曾在某場演講中提到，透過資料的改善所帶來的最終影響遠大於改善模型本身，因為資料標註所帶來的偏差往往是影響模型訓練的關鍵因素之一。

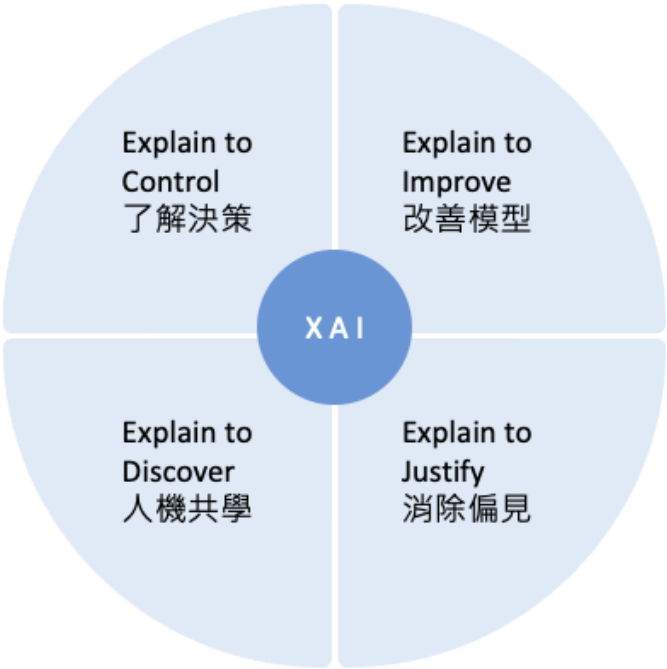
> 延伸閱讀：The Future of Data-Centric AI *(https://medium.com/@andy6804tw/the-future-of-data-centric-ai-a36bea495d1c)*

XAI 的重要性和優點

透過 XAI 提高模型可靠性和解釋性，使機器學習成果能夠被更廣泛的人理解和接受。此外透過解釋不同的特徵對模型決策的影響，幫助資料科學家改進和優化模型。 - XAI 能夠幫助機器學習模型更好地解釋其決策過程 - 這對於確定模型的弱點和提高其性能至關重要。透過 XAI 技術，我們可以更好地理解模型如何作出決策，從而發現其缺陷和潛在的偏見。

- XAI 能夠改善機器學習模型的可信度和可靠性。

- 當模型的決策過程變得透明時，使用者和機器學習模型的開發者可以更好地了解模型的表現和弱點，進而提高模型的可信度和可靠性。
- XAI 能夠促進人機共學
- 將機器學習模型設計為可解釋的，有助於人類更好地理解其決策並與其合作。這樣可以使人工智慧在更多的領域得到應用，從而為人類創造更多的價值。
- XAI 能夠使模型消除偏見
- 可幫助確定機器學習模型中可能存在的偏見和歧視，進而調整訓練數據和模型架構，消除這些偏見。



XAI 的應用領域

XAI 的應用領域十分廣泛，例如在智慧製造中，XAI 能夠幫助工程師對於機台異常狀況或參數最佳化進一步分析，並提高產能；在金融領域中，XAI 能夠幫助分析師對金融產品進行分析，提高金融風險管理的效率和準確性；在醫療領域中，XAI 能夠幫助醫生更好地理解疾病和治療方案，提高患者的醫療服務質量和安全性；在自動駕駛領域中，XAI 能夠幫助駕駛更好地理解自動駕駛系統的決策，提高駕駛安全性。

XAI 的出現使得機器學習技術能夠更好地應用於各個領域，提高模型的可靠性和透明度，同時也能夠提高應用場景的安全性和隱私性。總之，在這三十天當中將會逐一地介紹機器學習和深度學習領域中常見的模型解釋技巧。請各位盡請期待！

Reference

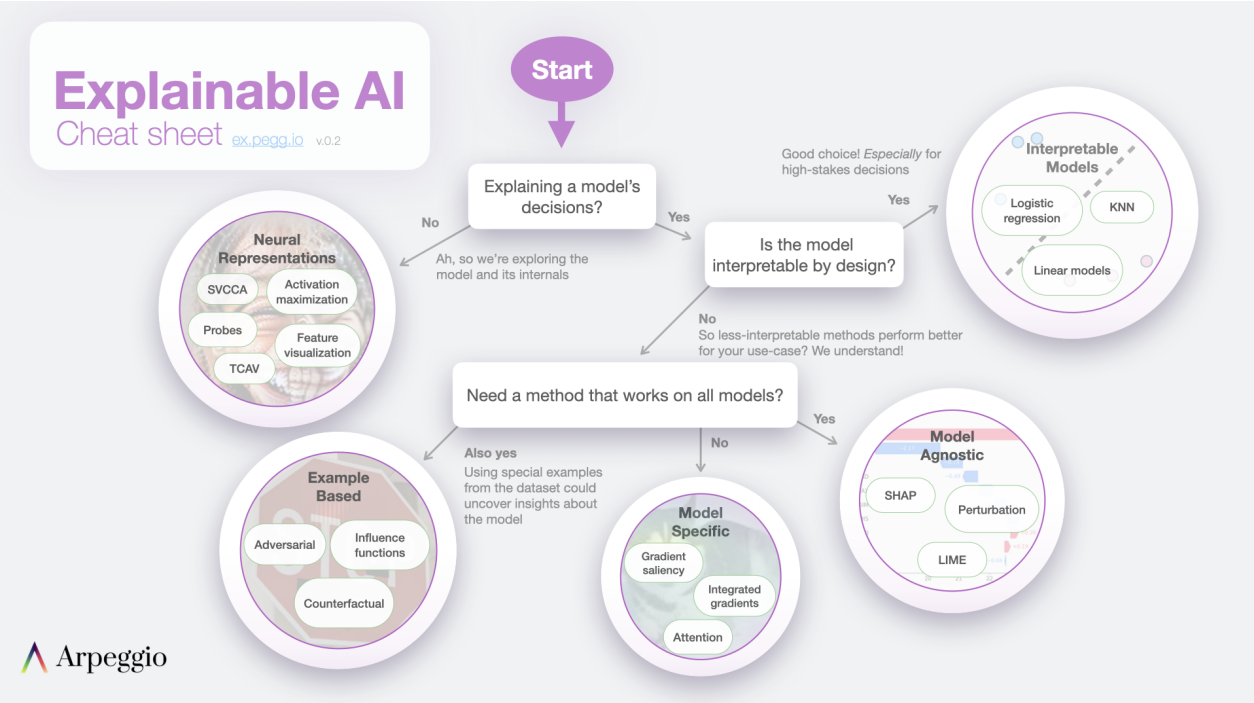
- XAI| 模型可解釋性的重要 *(https://medium.com/sherry-ai/xai-%E6%A8%A1%E5%9E%8B%E5%8F%AF%E8%A7%A3%E9%87%8B%E6%80%A7%E7%9A%84%E9%a0bbaa649978)*
- 增加內容參考 *(https://www.slideshare.net/dineshv62/explainable-ai-238430774)*

[Day 2] 從黑盒到透明化：XAI技術的發展之路

近年來人工智慧技術發展迅速，深度學習等技術的出現和應用已經帶來了很多驚人的成果，尤其是 ChatGPT 的出現更讓人們驚嘆不已。然而這些模型的黑箱特性一直是人工智慧領域中的一個重要議題。為了解決這個問題，越來越多的研究者開始關注解釋性人工智慧技術的發展。XAI 技術在過去幾年中經歷了長足的發展，從最初的可視化技術到現在的基於規則的解釋、深度學習可解釋性技術、模型過程可解釋技術等等，不斷地推陳出新。這些技術的不斷革新和提高，讓人們對於機器學習模型決策過程的理解更加深入和全面，也提高了機器學習模型的可信度和實用性。今天的內容我們將探討 XAI 技術的發展之路，並介紹幾個具有代表性的 XAI 技術。

XAI 學習地圖

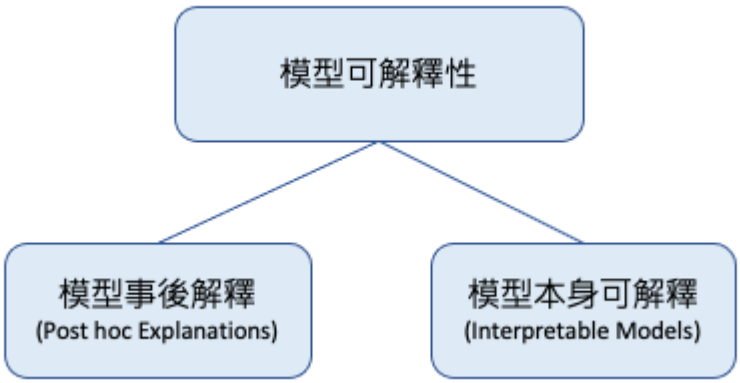
下圖取自於 Jay Alammar *(https://ex.pegg.io)* 的部落格，主要說明 XAI 技術中依據模型可解釋性的不同，分成了多種解釋方法。這些方法可分為模型本身具有可解釋性或是過於複雜難以解釋的情況。對於過於複雜的模型，我們需要透過事後分析技術來協助理解模型推論的邏輯。今天提到的所有名詞基本上彼此間都環環相扣，就讓我攘逐一為各位說明。



Interpretable Models vs. Post hoc Explanations

XAI 方法可分為模型本身可以解釋（Interpretable Models）與模型訓練完事後解釋（Post hoc Explanations）兩種。其中，模型本身可以解釋的方法包括：線性迴歸、邏輯迴歸、決策樹、K-

nearest neighbors、貝葉斯網絡模型。這些模型在自身設計上就已經具有一定的解釋性，因此可以直接透過模型本身來解釋預測結果。



模型本身可以解釋: - 線性迴歸 (Linear regression) / 邏輯迴歸 (Logistic regression) - 線性迴歸和邏輯迴歸模型通常是基於線性方程來預測目標特徵的值，因此可以很容易地解釋模型權重和特徵之間的關係。 - 決策樹 (Decision tree) - 決策樹是基於樹結構的模型，因此可以通過樹的分支路徑來解釋模型的決策過程。 - K-nearest neighbors (KNN) - 由於 KNN 算法本身就是基於距離計算的，因此可以解釋模型是基於哪些最近鄰居的資料進行預測的。 - 貝葉斯網絡模型 (Bayesian Network Model) - 通過機率模型表示特徵之間的條件依賴關係，可用於推論和預測。

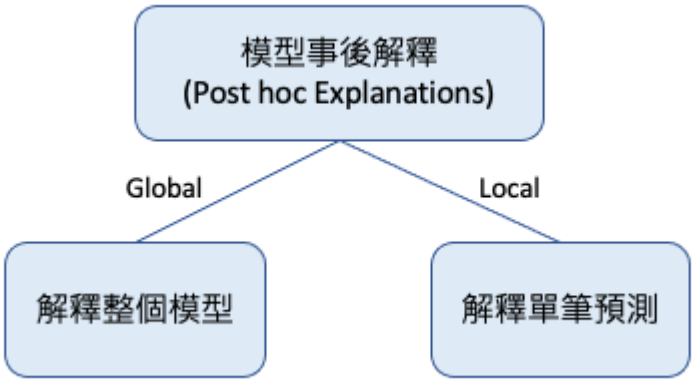
> 雖然隨機森林和XGBoost通常被認為是比較可解釋的tree-based系列模型，因為其結構相對簡單，可以直觀地理解每個決策的依據。但它們不算是Interpretable Models，因為其決策過程是由多個弱分類器或決策樹共同決定，其整體解釋性較難掌握，需要透過其他解釋方法來進行解釋。

Post hoc Explanations 指的是在模型訓練完畢後，使用額外的解釋方法來理解模型的行為和決策過程。這些方法通常是使用一些數據可視化或統計技術，來顯示模型中不同特徵之間的關係，以及這些特徵對模型結果的影響程度。常見的 Post hoc 解釋方法包括 Permutation Importance、Partial Dependence Plot (PDP)、Accumulated Local Effects (ALE)、SHapley Additive exPlanations (SHAP)、Local Interpretable Model-agnostic Explanations (LIME) 等等。這些方法可以用於解釋各種不同類型的模型，包括決策樹、神經網絡、支持向量機等等。

模型訓練完事後解釋: - Local Interpretable Model-agnostic Explanations (LIME *(https://arxiv.org/abs/1602.04938)*) - SHapley Additive exPlanations (SHAP *(https://arxiv.org/abs/1705.07874)*)

Global vs. Local Explanations

剛所提到的模型訓練完事後解釋的方法又可分為 Global 解釋整個模型行為以及 Local 解釋單筆預測行為。Global 的方法的目的是理解模型對所有數據點的預測，而不僅僅是特定的數據點或觀測。這種方法通常涉及到解釋模型中的特徵重要性，即哪些特徵對於模型的預測影響最大。



解釋整個模型的行為，例如： - Permutation Importance：隨機重排特徵，計算對模型準確度的影響 - Partial Dependence Plot (PDP)：顯示某個特徵對模型輸出的影響程度 - Accumulated Local Effects (ALE)：估計某個特徵對模型輸出的平均影響程度 - SHapley Additive exPlanations (SHAP)：計算每個特徵對預測值的貢獻程度

解釋單筆預測行為，例如： - Local Interpretable Model-agnostic Explanations (LIME)：通過生成局部可解釋的模型來解釋單筆預測的結果。 - SHapley Additive exPlanations (SHAP)：透過給每個特徵一個權重，計算其對預測結果的貢獻。 - ICE（Individual Conditional Expectation）：與 PDP 類似，不同之處在於 ICE 將每個樣本視為一個獨立的個體，而 PDP 則將所有樣本視為同一個整體。

> SHAP 可以同時用於分析全局和局部貢獻，並提供有關每個特徵如何影響模型預測的詳細訊息。

Model Agnostic vs. Model Specific

最後 XAI 的方法又可細分為 Model Agnostic 和 Model Specific 兩種。Model Agnostic 的方法是透過資料來解釋模型，例如先前提到的 LIME 和 SHAP 都是透過資料搭配方法來解釋模型的經典方法。

Model Agnostic 不考慮模型本身，只透過資料來解釋模型的方法： - LIME、SHAP、PDP、ICE、ALE - Anchor：透過找尋可以對預測結果產生重要影響的條件規則，來解釋模型的預測結

果。 - Surrogate Model：使用另一個模型來近似模擬原始模型，並以此模型來進行解釋。常用的模型包括線性模型、決策樹等。

而 Model Specific 的方法則是針對特定模型來進行解釋。例如，決策樹系列的演算法透過樹的分支可知道每個節點的決策，而神經網路透過梯度下降法則可分析每個參數對於輸出的影響。這些方法有助於了解模型的內部運作，但缺點在於限制在特定模型上。

Model Specific 考慮模型本身，解釋模型本身的方法： - Tree-based model：透過樹的分支可以知道每個節點的決策，以及每個變數對於決策的貢獻程度。 - 神經網路：透過梯度下降法可以分析每個參數對於輸出的影響，或是使用類神經網路的可視化技術來解釋模型。 - 模型融合方法(Stacking)：透過將多個模型進行結合，可以進一步提升預測準確度並探索每個模型對於整體預測的貢獻。

Python XAI 的套件有哪些？

以下是一些常用的 Python 可解釋 AI 工具：

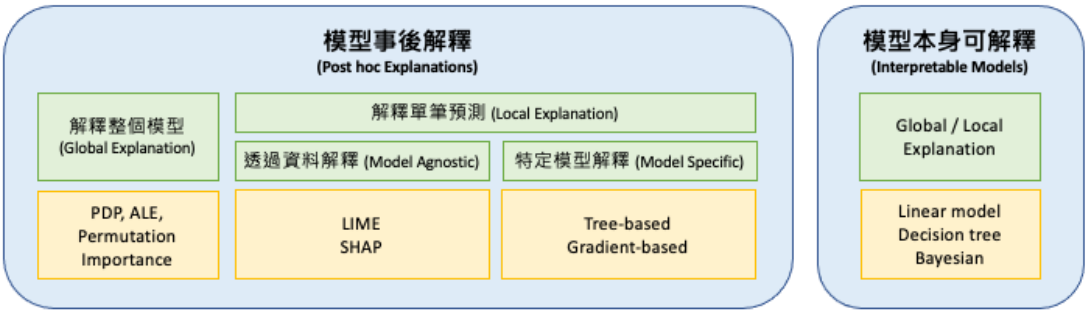
- SHAP *(https://github.com/slundberg/shap)* (SHapley Additive exPlanations)：可解釋性機器學習工具，提供全局和局部的解釋，並透過 Shapley 值計算影響預測的因素貢獻度。
- LIME *(https://github.com/marcotcr/lime)* (Local Interpretable Model-Agnostic Explanations)：一種局部解釋模型的工具，能夠解釋模型在單個預測中每個特徵的重要性，不考慮模型本身。
- scikit-learn Inspection *(https://scikit-learn.org/stable/inspection.html)*： sklearn 套件中的 inspection 提供了解釋整個模型行為的方法，幫助理解模型的預測以及觀察特徵重要程度。
- ELI5 *(https://github.com/eli5-org/eli5)* (Explain Like I'm Five)：支持多種模型解釋，包括線性模型、決策樹、隨機森林等，可用於解釋全局和局部的預測。
- InterpretML *(https://github.com/interpretml/interpret)*：一個針對機器學習模型的解釋工具，提供全局和局部的解釋，並且可以解釋多個模型之間的比較。
- What-if Tool *(https://pair-code.github.io/what-if-tool/)*：Google 開發的一種交互式可解釋性工具，能夠展示特定輸入對預測結果的影響，並提供調整輸入以觀察預測結果的功能。
- Shapash *(https://github.com/MAIF/shapash)*：是一個針對機器學習模型的自動化報告工具，可以幫助使用者快速解釋並理解模型的預測結果。

除了上述幾個之外還包括 scikit-explain, Skope-rules, DTREEviz, H2O, Yellowbrick, PDPbox, Skater, Ciu, Dalex, Lofo, Anchor, PyCEbox, Alibi, Captum, AIX360, OmniXAI, L2X。這些都可以透過 Python 來輔助我們解釋訓練好的模型。

> 本系列將會挑選幾個具有代表性的工具介紹給各位邦友

小結

最後用這張圖表做個總結，並統整了今天所學習的內容。簡單來說，要解釋機器學習模型的結果，可以採用模型事後解釋或模型本身可解釋的方式。透過模型事後解釋，我們可以進一步了解模型在特定數據上的表現，以及模型背後的推論過程。而模型本身可解釋的模型則可以提供直接的解釋，因此可以更好地理解模型在不同情況下的預測結果。此外，解釋整個模型可以揭示模型的整體結構和特徵重要性，而解釋單筆預測可以幫助我們理解模型如何進行個別預測。透過資料解釋可以解釋各種不同類型的模型，而特定模型解釋則專注於針對特定類型的模型進行解釋。



明天我們就來談談這些關於機器學習中的可解釋性的指標

Reference

- Explainable AI Cheat Sheet by Jay Alammar (Arpeggio) *(https://ex.pegg.io)*
- Developing and Experimenting on Approaches to Explainability in AI Systems *(https://research-information.bris.ac.uk/ws/portalfiles/portal/305310441/ICAART_2022_226_CR.pdf)*

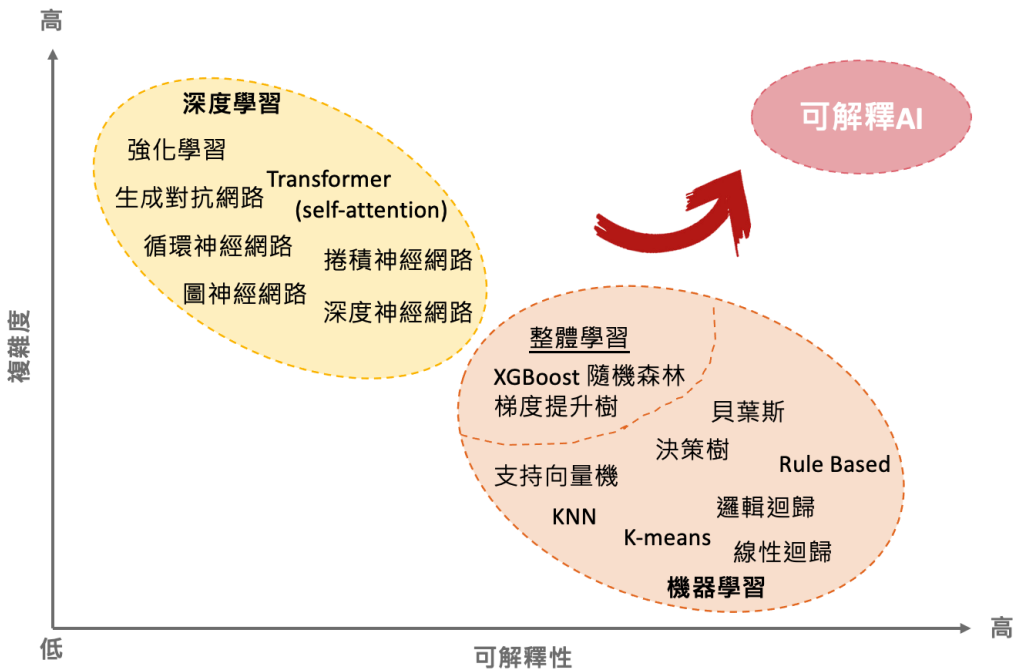
[Day 3] 機器學習中的可解釋性指標

「可解釋性指標」是 XAI 中用來衡量模型可解釋性的評估標準。它們是用來確定模型如何解釋其預測的方式，以及如何在給定輸入後生成可解釋的結果。可解釋性指標可以根據特定的案例和需求而有所不同，但通常會考慮到以下幾個層面：

- 精確度：模型的預測結果能否準確反映實際情況。
- 一致性：模型在不同情況下的預測結果是否一致。
- 可靠性：模型是否能夠可靠地處理所有輸入，包括異常值和噪聲數據。
- 透明度：模型的內部運作是否清晰易懂，能夠被解釋。
- 公平性：模型的預測結果是否公平，即是否存在對某些類別或群體的偏見。
- 可解釋性：模型的預測結果是否能夠被解釋和理解，包括模型的特徵重要性，模型的決策過程以及模型輸出的可視化表示。

準確度與可解釋性的權衡

在機器學習中，我們通常希望模型能夠同時達到高精度和可解釋性，但這兩者之間常常存在一個權衡。近年來隨著模型的精確度和複雜度不斷提高，但也帶來了模型的不透明性。為了更清楚地理解這個權衡，可以使用下面這張圖來比較。



參考 DARPA 's explainable AI 計畫 *(https://asd.gsfc.nasa.gov/conferences/ai/program/003-XAIforNASA.pdf)* p. 23

在這個平面圖中，X 軸代表模型的可解釋性，越靠右表示模型越容易解釋。Y 軸代表模型的複雜度，越高表示模型越複雜。通常情況下，複雜的模型（例如深度神經網絡）能夠實現更高的準確度，但解釋性較差。相反，較簡單的模型（例如線性迴歸）通常具有更好的解釋性，但是準確度可能不如複雜模型。

幸運的是，近年來一些新的結構化解釋性指標和可視化技術正蓬勃發展中，可以幫助解釋深度學習和機器學習模型。因此，這些指標和可視化技術可以將複雜黑盒子模型移動到平面圖上的更高解釋性區域。

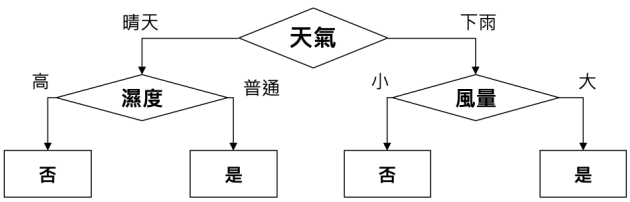
可解釋性指標有哪些？

以下為各位整理一些常見的可解釋性指標，每種指標都有其專門的方法和技術來實現。這些可解釋性指標可以協助我們確定哪些機器學習模型可以被視為可解釋的，因為它們能夠產生可靠的預測，同時也能夠解釋模型預測的過程和結果。

1. 特徵重要性 (Feature Importance)

這種方法可用於對全局和局部進行解釋，以解釋單個特徵的貢獻和影響。然而該方法往往難以解釋特徵之間的相互作用和複雜關係。常用的計算方法包括基於樹的模型的重要性指標和線性迴歸中的係數等。

線性迴歸：$y = \beta_0 + \beta_1 x$



決策樹

此外特徵重要性指標中最典型例子就是敏感度分析（Sensitivity　Analysis）。其目的是探討輸入特徵對模型輸出的影響程度，常用的方法有 Permutation Importance、Drop Column Importance 等方法。

2. 局部解釋性 (Local Explanations)

針對單筆資料預測的解釋方法，通常具有更高的彈性和細膩度，但可能會受到隨機性的影響，且無法解釋整個模型的行為。LIME 就是一種針對個別實例進行模型預測解釋的方法，它可以快速對複雜的黑盒模型進行解釋。透過對資料進行抽樣並使用黑盒模型來預測，然後根據實例之間的相似度對這些預測進行權重分配，從而學習出一個局部可解釋的線性模型。

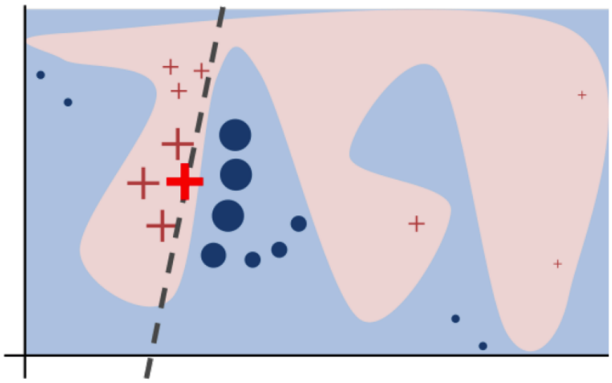
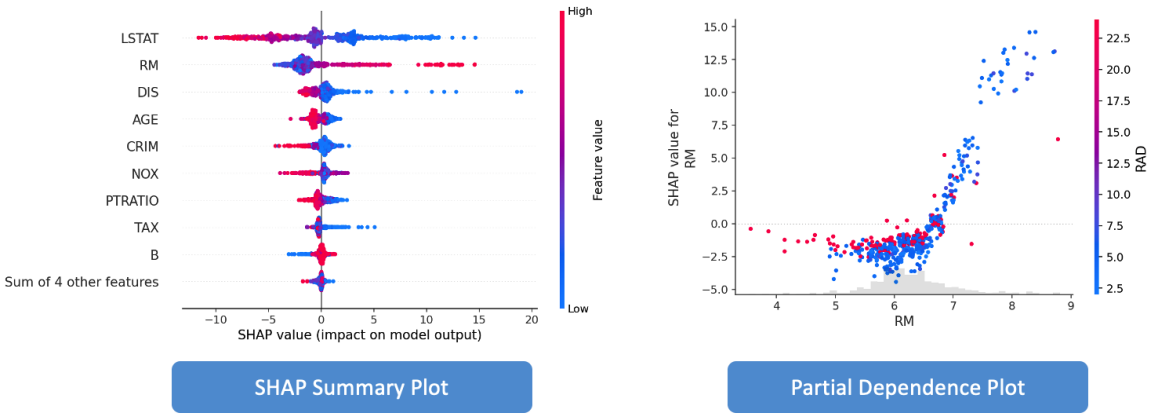


Figure 3: Toy example to present intuition for LIME.

LIME論文：“Why Should I Trust You?” Explaining the Predictions of Any Classifier *(https://arxiv.org/pdf/1602.04938v3.pdf)*

3. 全局解釋性（Global Explanations）

如果你看過前一篇文章，肯定對它不陌生。它可以解釋整個模型的預測行為，例如 Partial Dependence Plot（PDP）、Individual Conditional Expectation（ICE）、SHAP Summary Plot 和 Feature Interaction Plot等。這種解釋方法較為全面和綜觀，是 XAI 中最常見和最受歡迎的解釋方法。



4. 結構化解釋性（Structured Explanations）

是指用結構化的方式將模型預測的過程和結果進行解釋。這種方法將預測的解釋分為多個步驟，將每個步驟的解釋呈現為一個結構化的形式，例如樹狀圖、流程圖或語法解釋等。透過這種方式，我們可以更清楚地了解模型的決策過程和關鍵因素，也能夠更容易地理解模型的預測結果。常見的結構化解釋性指標包括 Decision Tree、RuleFit 等。此方法雖然能夠生成可解釋的模型，易於理解和解釋。但是，這些模型往往較為簡單，可能無法擁有較高的預測能力。

RuleFit 將線性模型和決策樹結合，同時學習特徵的重要性和特徵之間的交互作用，生成可解釋性強的模型。RuleFit 演算法的流程如下：

1. 資料預處理：對資料進行特徵提取和預處理，包括缺失值填充、特徵標準化等。
2. 基本樹模型生成：使用決策樹演算法生成一組基本樹模型，並提取出每個葉子節點的特徵。
3. 線性模型訓練：將基本樹模型中的葉子節點特徵作為新的輸入特徵，使用線性迴歸演算法訓練一個線性模型。
4. 預測：對於新的測試樣本，先使用基本樹模型將其映射到葉子節點上，並將葉子節點特徵作為輸入特徵，再使用訓練好的線性模型進行預測。

   RuleFit論文：Predictive learning via rule ensembles *(https://arxiv.org/pdf/0811.1679.pdf)*

5. 誤差分析（Error Analysis）

透過誤差分析，我們可以了解模型在哪些情況下表現不佳，並且針對這些情況進行調整，以提高模型的準確度。可以用於解釋模型在測試數據集上的誤差原因，例如 Confusion Matrix、ROC Curve、Precision-Recall Curve。在機器學習中，常見的任務有迴歸和分類兩種。以下分別介紹這兩種任務常見的誤差評估指標：

迴歸問題的評估指標：

- 均方誤差 (Mean Squared Error, MSE)：預測值與實際值的差的平方和的平均值，評估模型的整體預測能力。
- 均方根誤差 (Root Mean Squared Error, RMSE)：MSE 開根號，與 MSE 相比更能反映預測值與實際值的真實差距。
- 平均絕對誤差 (Mean Absolute Error, MAE)：預測值與實際值的差的絕對值的平均值，評估模型的整體預測能力。
- 平均絕對百分比誤差（Mean Absolute Percentage Error, MAPE）：用來衡量迴歸模型的預測精度，其計算方式為預測值和真實值之間的絕對誤差佔真實值的百分比的平均值。通常來說，MAPE 的數值越小越好，一般認為 MAPE 小於 10% 表示模型的預測效果較好。
- 決定係數 (R-squared)：評估模型與實際值之間的相關性，值越高代表模型的解釋能力越好，但也容易過度擬合。

分類問題的評估指標：

- 混淆矩陣 (Confusion Matrix)：列出實際值與預測值的對應情況，便於評估模型的準確性、召回率等指標。
- 準確率 (Accuracy)：預測正確的樣本數除以總樣本數，評估模型的整體預測能力。
- 精確率 (Precision)：預測為正的樣本中實際為正的比例，評估模型對正樣本的預測能力。
- 召回率 (Recall)：實際為正的樣本中預測為正的比例，評估模型對正樣本的覆蓋能力。

- F1值 (F1 Score)：精確率和召回率的加權調和平均數，綜合評估模型的預測能力。

6. 真實性驗證（Fidelity）

真實性驗證可以提高模型的信心和可靠性，避免過度擬合和選擇性偏差等問題。通常建議真實性驗證使用的測試資料集的數量應該與訓練資料集的數量相當，並且可以使用交叉驗證等方法來進一步驗證模型的可靠性。除此之外它還可以評估模型解釋是否忠實反映模型的決策過程，常用的方法有 Anchors、Counterfactual Explanations 等。

> Anchors論文：Anchors: High-Precision Model-agnostic Explanations *(https://homes.cs.washington.edu/~marcotcr/aaai18.pdf)*

小結

今天學到了許多機器學習中的可解釋性指標，但是使用不同的可解釋性指標時，應該注意以下幾點： - 適用範圍：不同的可解釋性指標適用於不同的場景，應根據具體需求選擇。 - 精度和穩定性：解釋性指標在不同場景下可能存在精度和穩定性問題，應慎重使用。 - 解釋效果：可解釋性指標不僅應該能夠生成可解釋的結果，而且還應該能夠讓使用者更好地理解模型，避免出現誤解或誤解。 - 結果呈現：不同的可解釋性指標生成的結果形式不同，需要根據需求選擇適當的呈現方式，以便於使用者理解。

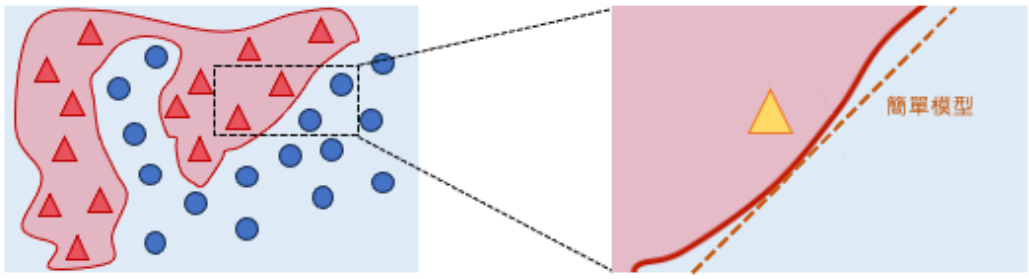
Reference

- DARPA 's explainable AI (XAI) program: A retrospective *(https://www.researchgate.net/publication/356781652_DARPA_'s_explainable_AI_XAI_program_A_retrospective)*

[Day 4] LIME vs. SHAP：哪種XAI解釋方法更適合你？

LIME 和 SHAP 都是機器學習中的解釋性方法，它們的共同點是都適用於模型無關性（Model Agnostic），並透過資料來解釋模型的預測結果。如果不想深入模型內部，但又想要能解釋模型，那看這篇文章就對了！簡單來說今天介紹的解釋方法不探究模型內部的運作原理，而是嘗試帶一些資料，去觀察輸出結果。然而 LIME 與 SHAP 在解釋方式和範圍上有一些區別。在今天的文章中，我們不會深入探究這兩個方法的背後詳細原理。而是透過一個簡單地例子，讓各位讀者知道兩種方法的差別與使用時機。

LIME	SHAP
透過資料解釋模型（Model Agnostic）	
局部解釋	全局&局部解釋
透過建立簡單的模型解釋某筆資料。	用 Shapely Value 找貢獻值並解釋模型如何推論。

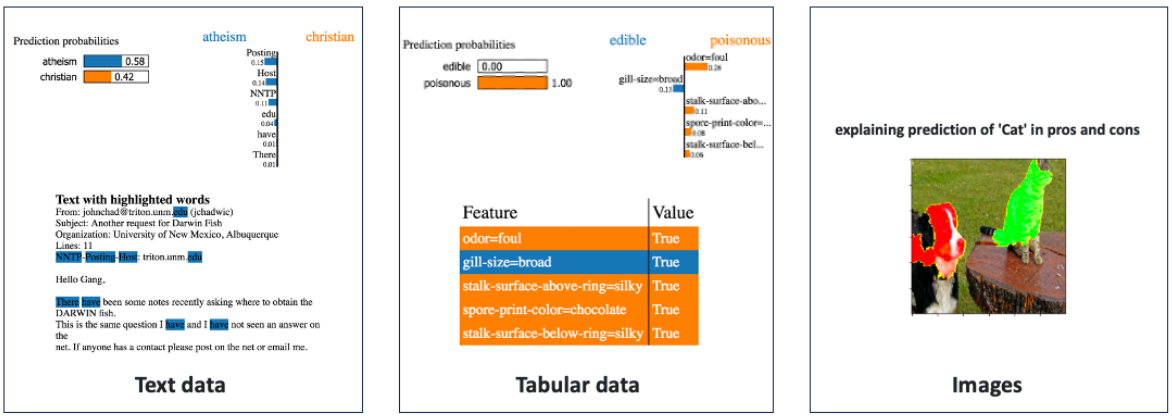


Global Explanations
解釋整個模型行為

Local Explanations
解釋單個預測行為

LIME

LIME（Local Interpretable Model-agnostic Explanations）主要提供局部解釋，它透過建立簡單的線性模型來解釋某一筆要被預測的資料。並提供了一個局部近似模型，用於解釋該特定資料的模型預測。它的優點是可以提供針對某一筆特定資料的解釋，並且可以對文字、表格資料、影像進行解釋，但缺點是解釋範圍有限，僅限於觀察局部趨勢。



LIME 的局部解釋過程

以表格型資料為例，首先我們使用訓練集來訓練一個複雜模型，接著將訓練集和欲解釋的一筆資料輸入到 LIME 中。這時 LIME 機制就會啟動，並從訓練集中隨機採樣N筆資料，每筆資料的預測結果 ŷ 是通過已訓練好的複雜模型計算而得。此外被採樣的N筆資料將與要解釋的那筆資料進行距離計算。距離越近的資料點將獲得更高的權重，因為 LIME 著重於解釋目標附近的資料點。這樣可以確保解釋模型更關注於與要解釋的資料點相似的區域，從而提高解釋的準確性和可解釋性。基於這些權重，LIME 建立一個簡單的線性模型或其他可解釋的模型，用於解釋黑盒模型在該資料點上的預測結果。

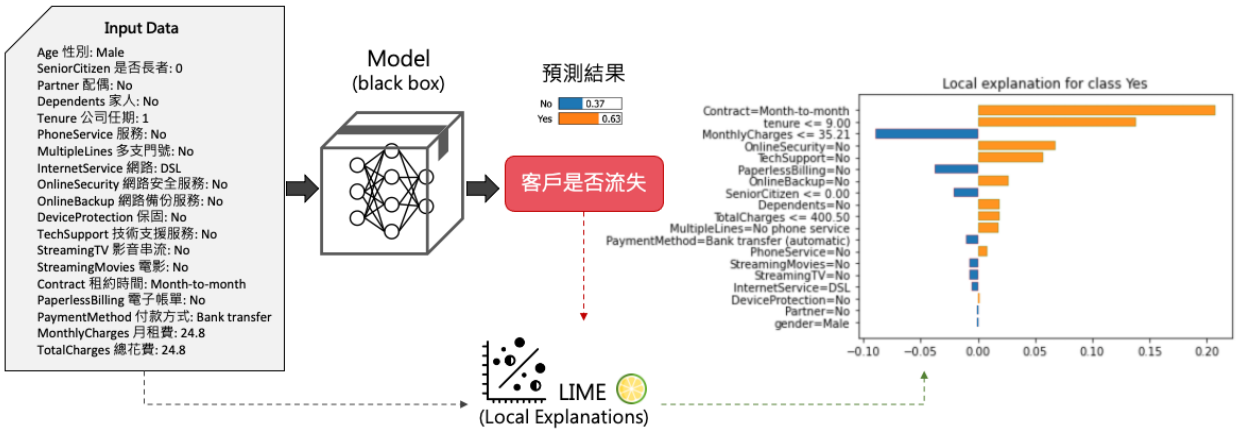
LIME 的核心概念就是將「局部的新資料」引入，加上「可理解的擾動」，然後使用這些資料來解釋模型的輸出。以下是 LIME 的解釋過程: 1. 使用訓練集訓練一個複雜模型(black box) 2. 將訓練集與要被解釋的一筆資料餵入LIME - 針對訓練集隨機採樣 N 筆數據 - 採樣的每一筆訓練資料的 y 並非真實答案，而是透過複雜模型預測的結果 ŷ - 被採樣 N 筆資料將與要被解釋的那筆資料計算距離，越近的權重越大 - 將採樣的資料訓練一個簡單模型(線性迴歸) 3. 透過簡單模型來解釋該筆預測結果

透過以上的過程，LIME 能夠生成一個局部解釋模型，用於解釋特定資料點的預測結果。不過 LIME 存在一個缺點，即每當要解釋新的資料點時，需要重新執行上述動作以生成一個新的簡單模型並進行解釋，這可能會帶來一些計算和時間上的成本。這是因為 LIME 是一種局部解釋方法，它專注於解釋單個資料點的預測結果，而不是全局模型的整體解釋。另外如果輸入和輸出之間存在複雜的交互關係，又或是對於特徵的了解度不夠，那麼透過資料解釋的方法可能不適用。

LIME 應用例 (表格資料)

下面這張圖是一個預測電信客戶流失的例子，該資料集包含了19個特徵。假設有一筆新的客戶資料進來，我們可以透過已訓練好的黑盒模型預測該客戶可能即將流失。至於它是怎被預測出來有 63% 的信心是 Yes(即將流失的用戶) 呢？透過 LIME，我們建立了一個線性模型，以解釋一筆預測結果為63%選擇"是"的資料。從下圖中我們可以觀察到橘色的柱狀圖表示某些特徵對於客戶流失的影響呈正相關，這些特徵可以被視為影響流失決策的關鍵因素。我們特別注意到，

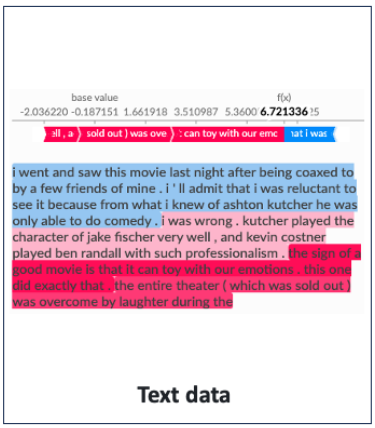
「租約時間」這一特徵對於是否續約是一個主要的影響因素，以月為單位續約的客戶可能只是短期的使用者，隨時可能解約。另一方面，藍色的柱狀圖表示其它幾個特徵是影響客戶繼續使用該服務的關鍵因素。

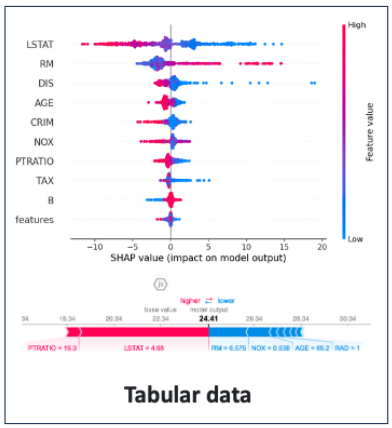


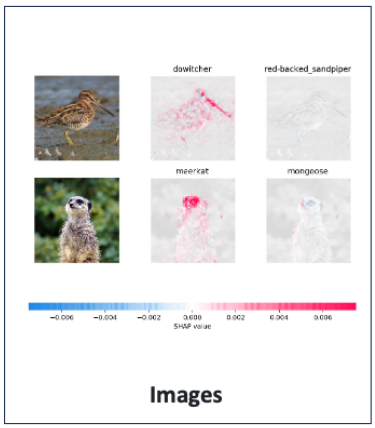
LIME 具有靈活性和易解釋性的優點，可以在解釋黑盒模型時提供局部解釋，使得模型的預測結果更容易理解。但它仍面臨一些問題，如鄰域的定義問題和採樣方法的選擇。此外忽略特徵之間的相關性可能會導致解釋結果的不合理性。因此透過 LIME 解釋模型時需要了解這些限制，並注意其解釋的範圍和可靠性。

SHAP

相較之下，SHAP（SHapley Additive exPlanations）主要以全局解釋為主，同時也支援局部解釋。它利用 Shapley Value 的概念來計算特徵對於模型預測的貢獻值，並用這些貢獻值來解釋模型的推論過程。SHAP 不僅提供了更全面的解釋，可以理解整個模型的決策方式和特徵重要性。它的優點是能夠提供全局視角的解釋，有助於理解模型的整體行為，但缺點是計算成本較高，特別是在大型資料集和複雜模型上。此方法也可以應用於不同的數據類型，包括文字、表格資料和影像。







SHAP *(https://github.com/slundberg/shap)* 套件提供了多種計算 Shapley Value 的方法：

1. TreeExplainer
    - TreeExplainer 是專為解釋樹狀結構的模型（例如決策樹、隨機森林和梯度提升樹）設計。它利用樹狀結構的特性，透過計算每個特徵的 Shapley Value 來解釋模型的預測。
2. DeepExplainer
    - DeepExplainer 是 SHAP 庫中針對深度學習模型的解釋器。並基於 DeepLIFT 利用深度學習模型的反向傳播算法，計算每個特徵對於預測的貢獻，並生成相應的 Shapley Value。
3. GradientExplainer
    - GradientExplainer 是基於模型的梯度訊息進行解釋。它使用模型的預測機率值與該特徵的梯度值之間的乘積作為該特徵的重要性得分。
4. LinearExplainer
    - LinearExplainer 是一種用於解釋線性模型的方法。它適用於線性迴歸、線性分類等簡單的線性模型。
5. KernelExplainer
    - KernelExplainer 是 SHAP 庫中的一種通用解釋器，可以應用於各種模型。它利用核函數對特徵空間進行采樣，並計算每個特徵對於預測的影響力。

SHAP 的全局/局部解釋過程

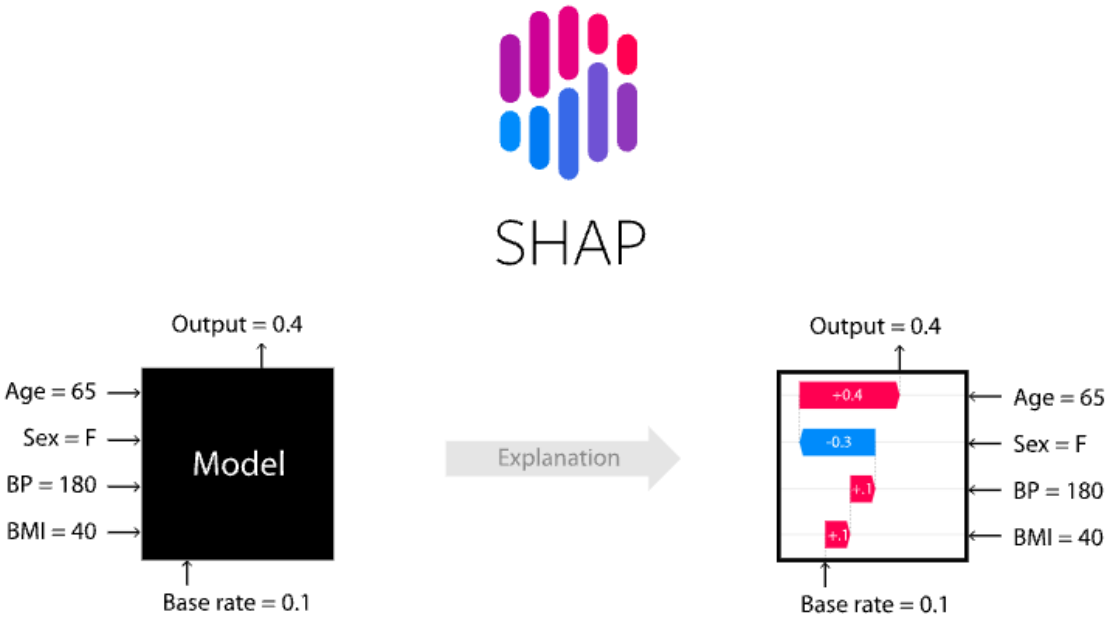
SHAP 的全局解釋幫助我們了解模型整體的行為和特徵的重要性，而局部解釋則幫助我們理解模型在特定實例上的預測。以下為 SHAP 的解釋過程:

1. 使用訓練集訓練一個複雜模型(black box)
2. 根據模型演算法特性選擇合適的 SHAP Explainer
3. 將訓練集與要被解釋的資料(可多筆)餵入 Explainer 並計算 SHAP values
4. 全局解釋過程：
    - 產生摘要圖表(Summary Plot)，顯示每個特徵的 SHAP value 絕對值加總平均。
5. 局部解釋：
    - 單筆資料解釋：根據一筆資料，SHAP 計算每個特徵對該預測的影響。幫助理解為什麼模型對於特定一筆資料做出了特定的預測。

SHAP 應用例 (表格資料)

我們一樣拿表格資料進行解釋，並與 LIME 進行比較。這裡採用 Kernel SHAP 的方法來估計 Shapley Value。假設有輸入四個特徵，年齡、性別、血壓、BMI作為輸入要預測一個人罹癌的機率。SHAP 要做的是分別計算根據 Age=65 對於輸出0.4有多少貢獻，對於 Sex=F 對於輸出0.4有多少貢獻…。因此會有個基準點 base rate 這裡為0.1，表示都不做任何事就會有輸出0.1機率會罹癌。當看到一個年齡特徵等於65就會加0.4，看到性別等於F就會減0.3，直到累加完所有特徵就是預測總輸出y=0.4。因此 SHAP 模型不管模型演算法做了什麼事，它只管輸入的特徵值

對於輸出的結果就可以計算 base rate 是多少，以及每一個特徵值是多少對於最終輸出影響有多少。



Kernel SHAP 的優勢之一是它可以應對高維度的特徵空間，並且計算效率較高，但仍然需要進行大量的計算。因此在應用時需要注意計算效率、核函數選擇、樣本數量和鄰域定義等方面的考慮。適當的選擇和調整這些參數和方法可以提高解釋結果的準確性和可靠性。

小結

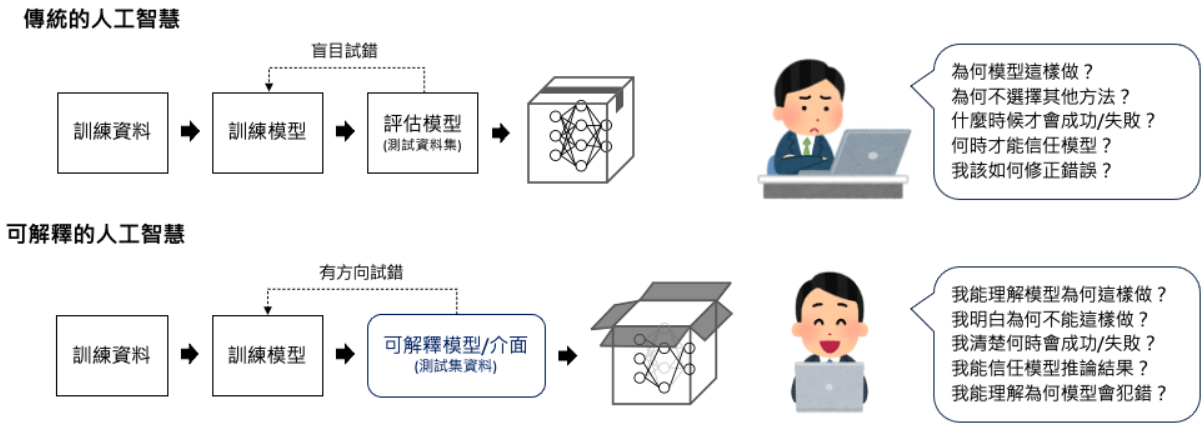
LIME 和 SHAP 都是重要的解釋性演算法，它們在解釋方式和範圍上有所不同。LIME 提供了局部解釋，主要針對特定的某一筆資料；而SHAP主要提供全局解釋，但也支援局部解釋。選擇哪種方法取決於解釋的需求和應用場景。如果想只關注特定資料的解釋，LIME 可能是一個不錯的選擇；如果需要理解整個模型的行為和特徵的影響，則 SHAP 會更適合。

Reference

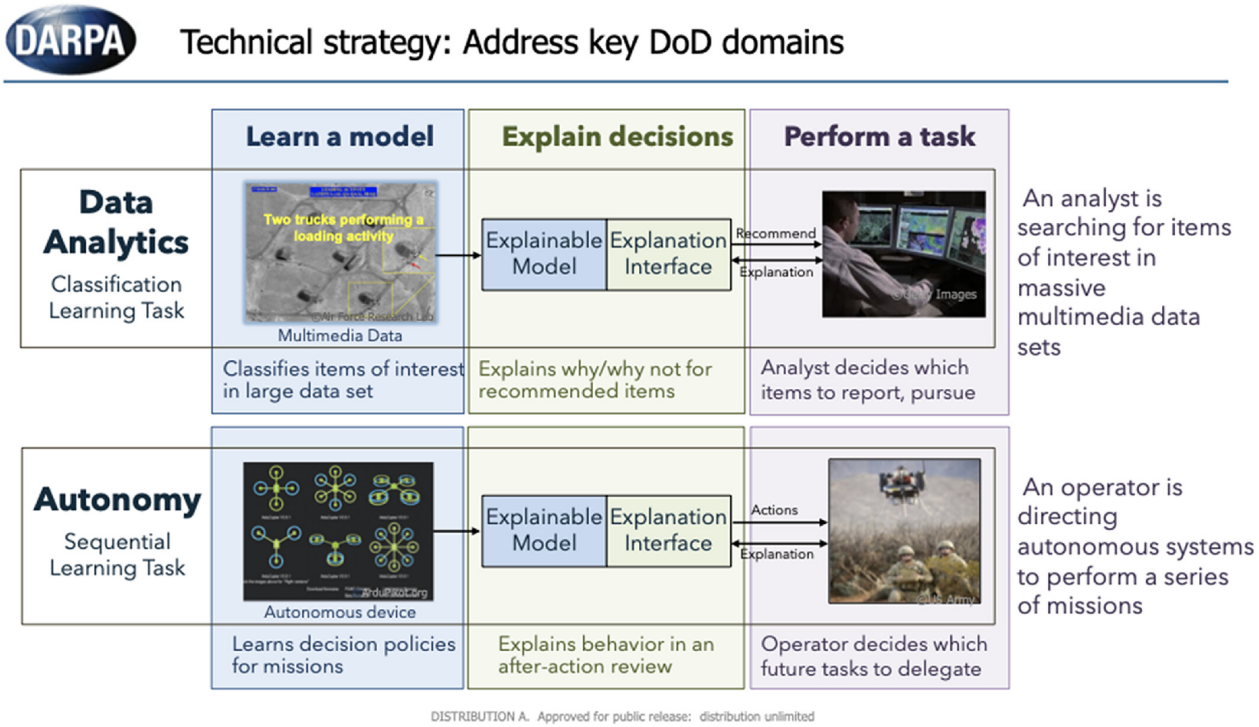
- 可解釋方法LIME和SHAP代碼實戰 *(https://e0hyl.github.io/BLOG-OF-E0/LIMEandSHAP/#shapley-additive-explanationsshap)*

[Day 5] 淺談XAI與傳統機器學習的區別

機器學習是一種透過對大量數據進行訓練和不斷優化演算法的方法，目的是提高預測準確性和決策可靠性。透過處理龐大的數據集，機器學習模型能夠自動學習並擬合出數據中的模式和關聯。儘管機器學習相關的套件和演算法不斷地突破，但在應用開發上仍面臨諸多有待克服的難題。我想這個問題也是你我的痛點，就是「AI 模型演算法驗證不易」成為發展瓶頸。由於我們無法逐一且明確列出模型背後的推論法則，所以只能將其視為一個龐大且複雜的數學模型。當我們使用傳統機器學習模型進行預測時（如：隨機森林、支持向量機...等），通常只會得到模型的預測結果，但是很難解釋為什麼模型會做出這樣的預測，這就是傳統機器學習模型的局限性。我們該如何說服客戶訓練的模型準確率很好，並且該如何證明它呢？這也就是我們現今所重視的 XAI（可解釋人工智慧）。



然而自2016年起國防高等研究計劃署（DARPA），也宣布將 XAI 列為接下來科技發展的重點。DARPA 是美國國防部所屬的研究機構，致力於推動科技創新和發展前沿技術，以增強美國國家安全和國防能力。透過 XAI，DARPA 希望能夠增加對 AI 系統的透明度，使其能夠更有效地與人類合作，提供可靠和解釋的結果，並降低對AI系統的依賴度。此外美國電氣和電子工程師協會（IEEE）也在2016、2017年先後發布AI倫理設計準則 *(https://standards.ieee.org/news/ead_v2/)*，從技術角度設立 AI 發展標準。



資料來源DARPA's explainable AI (XAI) program: A retrospective *(https://onlinelibrary.wiley.com/doi/full/10.1002/ail2.61)*

延伸閱讀：Making Things Explainable vs Explaining: Requirements and Challenges under the GDPR (arxiv) *(https://arxiv.org/abs/2110.00758)*

關於國際上的 AI 倫理準則，在法規方面，人工智慧的應用有必要明確定義其決策過程，以確保其可信度和透明度。歐盟在2018年修訂的「一般資料保護規範」（GDPR）中，提出了「要求解釋的權力」，旨在解決日益受到關注的演算法可能引發的問題。隔年歐盟議會更進一步的發佈「可信賴人工智慧倫理準則」確保人工智慧的發展和應用符合道德和倫理原則。

延伸閱讀：歐盟議會發布《可信賴人工智慧倫理準則》 *(https://stli.iii.org.tw/article-detail.aspx?no=64&tp=1&d=8248)*

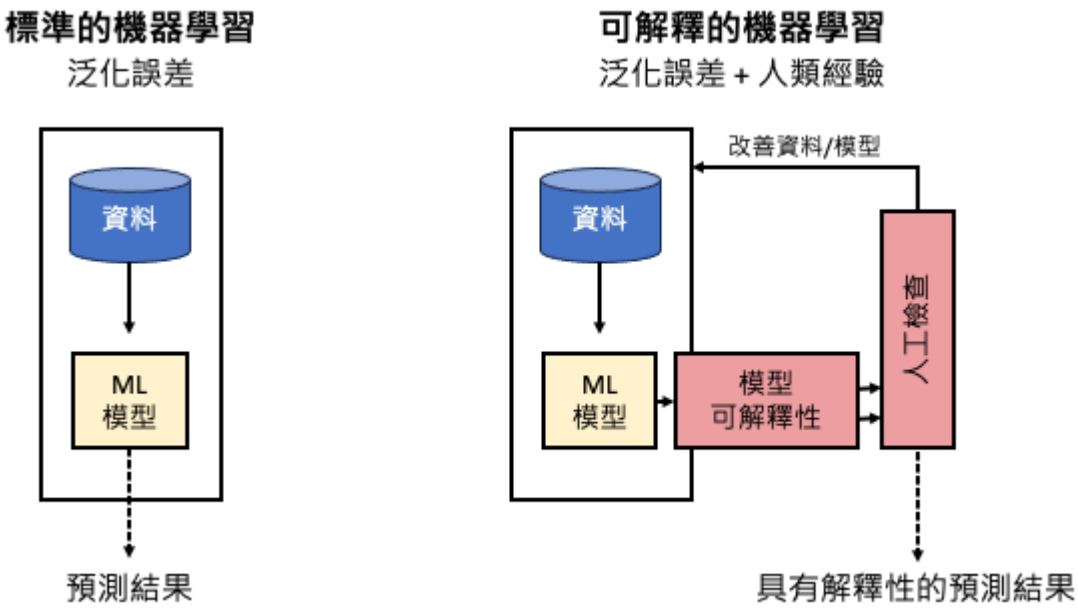
為何需要解釋性？因為要改善模型

人工智慧在我們日常生活中的應用越來越普遍，以協助人類做出決策。這些決策可能涵蓋從生活方式的選擇到更複雜的決策，如設備故障預測、產品瑕疵檢測、商業決策、信貸批准和法庭判決等。許多機器學習演算法是黑盒子，缺乏透明度，這引起了人們對其可信度的擔憂。就如同下圖左邊的傳統標準化機器學習流程，透過大數據與泛化誤差概念去擬合一個模型。在機器學習中，泛化誤差（Generalization error）指的是模型在面對新的、未見過的資料時所產生的錯誤。換句話說，它是衡量模型在應用到新資料時的預測能力或泛化能力的一個指標。因此我們可以透過泛化誤差的大小來衡量模型在未曾見過的資料上的適應能力，這反映了模型從訓練資料中學習到的知識的有效性。如果泛化誤差過高，表示模型在新資料上的預測能力較差，可能

存在過擬合或欠擬合的問題。所以降低泛化誤差是機器學習中的一個重要目標，可以透過調整模型結構、優化演算法、增加訓練資料等方式來實現。

> 在機器學習中，泛化誤差（Generalization error）指的是模型在面對新的、未見過的資料時所產生的錯誤。

而圖右是 XAI 搭配機器學習的概念，我們可以結合泛化誤差和人類經驗來進行機器學習建模。首先我們利用訓練資料來訓練模型並計算其泛化誤差，這可以告訴我們模型在未見過的數據上的適應能力。接著我們利用人類的經驗和領域知識來檢閱模型的決策過程和結果。這包括分析模型的特徵重要性、影響因素和預測結果的解釋，從中觀察模型是否有學到真實世界中的本質。這種結合能夠提高模型的預測能力和可解釋性，使我們更加信任和依賴於機器學習模型的結果。



為了對這些系統產生信任，人們希望能夠追究責任並獲得解釋。這需要能夠理解和解釋人工智慧系統背後的決策過程，以確保其可靠性和合理性。因此擺脫傳統的機器學習思維，透過 XAI，我們能夠了解解釋人工智慧模型背後的決策原理，確保決策的公正性和合理性，避免 AI 模型的不確定性造成無可挽回的失控。

XAI的三個階段

我們可以將人工智慧的可解釋性分成三個階段：建模前、建模過程和模型上線。

- 建模前
- 在建模前，我們著重於資料的解釋性，確保資料本身清晰且有意義。包括數據的標記、遺漏值的處理、特徵選擇和轉換等。
- 建模過程

- 在建模階段，我們專注於開發可解釋的模型。包括選擇可解釋的機器學習算法，調整模型的超參數，以及使用解釋性技術（例如特徵重要性分析和決策規則提取）來解釋模型的預測能力。
- 模型上線
- 確定最終的模型後即可部署模型並於實際場域中運行。此階段著重於解釋決策和傳遞模型的預測結果。

各行各業都需要具備可解釋性的AI

大部分機器學習系統主要集中在像是 Google 和 Meta 這類型的科技公司，這些系統的錯誤預測可能會導致對用戶的錯誤訊息。然而近年來 AI 有跨足不同領域的趨勢，當這些系統應用於其他行業，如醫療、軍事和金融...等。錯誤的預測就可能產生負面後果，甚至影響到許多人的生活。因此我們需要建立能夠解釋其決策過程的人工智慧系統。這樣的系統可以提供透明的決策解釋，使我們能夠理解它們的決策基礎，確保其合理性並減少潛在的風險。透過這樣的解釋性人工智慧系統，我們可以更好地應對機器學習在各個領域中可能帶來的挑戰，確保其應用的安全性和可靠性。

Reference

- DARPA Explainable Artificial Intelligence by David Gunning *(https://sites.cc.gatech.edu/~alanwags/DLAI2016/(Gunning)%20IJCAI-16%20DLAI%20WS.pdf)*
- DARPA XAIforNASA by David Gunning *(https://asd.gsfc.nasa.gov/conferences/ai/program/003-XAIforNASA.pdf)*
- Explainable AI by Mitosis Business Enablers *(https://www.slideshare.net/dineshv62/explainable-ai-238430774)*
- wikipedia 可解釋人工智慧 *(https://zh.wikipedia.org/zh-tw/%E5%8F%AF%E8%A7%A3%E9%87%8B%E4%BA%BA%E5%B7%A5%E6%99%BA%E6%85%A7)*

2.XAI在傳統機器學習中的應用

[Day 6] 非監督學習也能做到可解釋性？探索XAI在非監督學習中的應用

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/06.非監督學習也能做到可解釋性？探索XAI在非監督學習中的應用.ipynb)*

由於非監督學習模型通常沒有像監督學習中的標籤可使用，因此非監督學習模型的可解釋性通常是透過資料視覺化和數據分析來實現。以下為各位整理非監督學習中可解釋性的一些例子：

- 降維分析：將降維後的數據投影回原始空間並視覺化，從中理解原始資料的重要特徵和關係。
- 主成分分析(PCA)：解釋主成分分析提取的主成分特徵和對原始資料的貢獻，理解資料的降維和結構化過程。
- t-隨機鄰近嵌入法(t-SNE)：透過將高維空間中的資料映射到低維空間，並且保留原始資料的局部結構，讓我們能夠視覺化高維資料，從而進行探索性分析。
- 聚類分析：透過解釋聚類結果，探索資料的內在分布和潛在特徵。

降維

降維技術是一種數據壓縮技術，可將高維度的數據轉換成低維度表示。透過XAI技術，我們可以對降維後的數據進行解釋，以了解不同特徵對數據降維的影響，以及低維度數據與高維度數據之間的關係。其中主成分分析（PCA）和 t-SNE 等方法都可以進行可解釋性分析。

> 理論知識可以參考全民瘋AI系列2.0非監督式學習-降維 *(https://ithelp.ithome.com.tw/articles/10267685)*

以下範例使用手寫數字辨識資料集為例，透過可視化方法來顯示 PCA 和 t-SNE 降維的結果。

首先，我們需要從 sklearn 的內建資料集中載入手寫數字資料集。load_digits()是一個手寫數字資料集，共有1797筆資料，每筆資料都是8x8的灰階圖片，表示一個0到9的手寫數字。其回傳的資料集包含兩個主要部分：資料和標籤。

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
import numpy as np

# 載入手寫數字資料集
digits = load_digits()
X = digits.data
y = digits.target
```

如何使用方差比和方差值優化PCA分析

PCA（Principal Component Analysis）主成分分析是一種常用的線性降維技術，它可以將高維數據映射到低維空間，同時保留數據的主要特徵。在 PCA 中方差比和方差值是用來衡量主成分分析結果。通常我們會選擇方差比較大的主成分，因為它們能夠解釋更多的數據變異性，並保留更多的重要訊息。

- 方差比(variance ratio): 是指每個主成分所解釋的方差佔總方差的比例，用於衡量每個主成分的貢獻程度。
- 方差值(variance): 是每個主成分的方差，代表了每個主成分所包含的訊息量。

我們可以使用 PCA 來降維，並選擇不同的主成分個數，來觀察其對於解釋變異量的貢獻度。

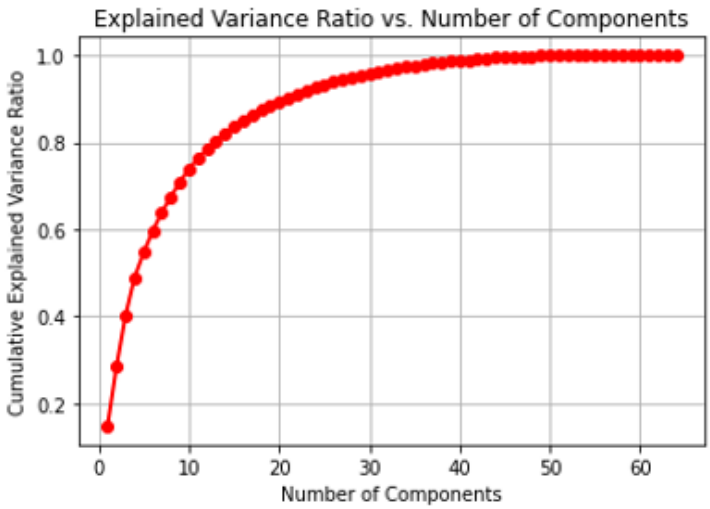
```
from sklearn.decomposition import PCA

# 將主成分個數設為 1 至 64，計算累積方差比
n_components = range(1, 65)
cumulative_variance_ratio_list = []
variance_ratio_list = []
for n in n_components:
    pca = PCA(n_components=n)
    pca.fit(X)
    variance_ratio_list.append(np.sum(pca.explained_variance_ratio_))
    cumulative_variance_ratio_list.append(np.sum(pca.explained_varian
```

最後我們可以將不同主成分個數下的累積方差比繪製成圖表，來選擇適合的主成分個數。

```
import matplotlib.pyplot as plt

# 繪製累積方差比圖表
plt.plot(n_components, cumulative_variance_ratio_list, 'ro-', linewid
plt.xlabel('Number of Components')
```

```
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Explained Variance Ratio vs. Number of Components')
plt.grid()
plt.show()
```

繪製出來的圖表會顯示不同主成分個數下的累積方差比，我們可以從圖表中找出適合的主成分個數，以達到適當的降維效果。



從t-SNE觀察資料降維後的分佈

t-SNE（t-Distributed Stochastic Neighbor Embedding）t-隨機鄰近嵌入法是一種非線性的降維方法，它可以將高維數據映射到低維空間中，同時保留數據的局部結構，因此可以用於探索數據的相似性、聚類和分類等問題。

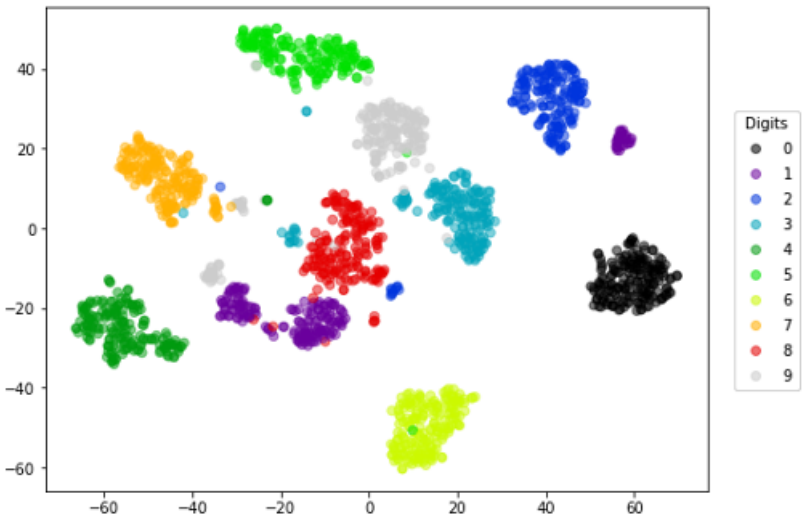
> 有興趣的讀者可以參考我之前的一篇文章動手做一個機器學習 API *(https://medium.com/ai-academy-taiwan/%E5%8B%95%E6%89%8B%E5%81%9A%E4%B8%80%E5%80%8B%E6%A9%9F%E5%99%A8%E5%api-97790e304c23)*

我們可以使用 sklearn 中的 TSNE 降維，將原始的高維度資料 X 降維成二維資料 X_embedded。其中 n_components 參數設為 2，代表要將資料降維到二維空間。另外 t-SNE 的結果通常是隨機初始化的，因此 random_state 參數則是設定隨機數生成器的種子，確保每次執行程式時結果一致。

```
from sklearn.manifold import TSNE

tsneModel = TSNE(n_components=2, random_state=42)
X_embedded = tsneModel.fit_transform(X)
```

最後我們可以將降成兩維的結果透過點散圖繪製出來。t-SNE 可以將相似的點映射到靠近彼此的位置，因此可以很好地顯示數據集中的群集和分類。

```
plt.figure(figsize=(8,6))
scatter = plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=y, alpha=
            cmap=plt.cm.get_cmap('nipy_spectral', 10))
# 加入圖例
legend1 = plt.legend(*scatter.legend_elements(), title="Digits",
                     bbox_to_anchor=(1.03, 0.8), loc='upper left')
plt.gca().add_artist(legend1)
plt.show()
```



聚類分析

聚類分析是一種非監督式機器學習技術，用於將相似的觀察值或數據點分為一組，從而形成有意義的子集或群集。

> 理論知識可以參考全民瘋AI系列2.0非監督式學習　K-means　分群 *(https://ithelp.ithome.com.tw/articles/10266672)*

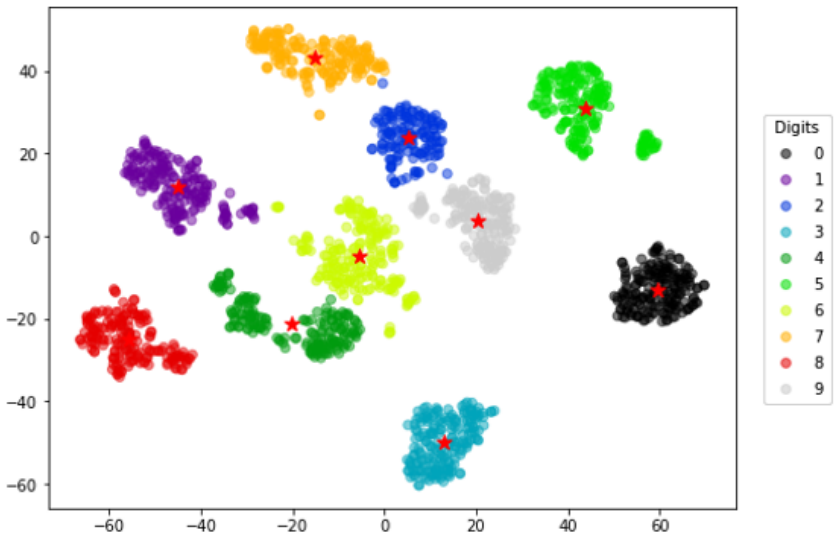
k-means 分群

k-means 是聚類分析中最常用的算法之一。它是一種迭代算法，透過反覆計算每個數據點與其所屬群集的重心之間的距離，來不斷更新群集中心點的位置。可使用視覺化方法來顯示每個群集的特徵，例如散點圖或熱力圖，以顯示群集之間的相似性和差異性。接下來我們會拿剛剛手寫數字 t-SNE 降維後的結果，透過分群分類演算法協助我們進行分群。由於我們有十種數字因此我們希望能夠特過分群演算法找出十個中心點。

```
from sklearn.cluster import KMeans

kmeansModel = KMeans(n_clusters=10, random_state=42)
clusters_pred = kmeansModel.fit_predict(X_embedded)
```

我們可以將 t-SNE 的結果與其他群集算法的結果進行比較，可以確定分類是否與 t-SNE 的分群結果一致。

```
plt.figure(figsize=(8,6))
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=clusters_pred, alph
            cmap=plt.cm.get_cmap('nipy_spectral', 10))

# 加入圖例
legend1 = plt.legend(*scatter.legend_elements(), title="Digits",
                     bbox_to_anchor=(1.03, 0.8), loc='upper left')
plt.gca().add_artist(legend1)
plt.scatter(kmeansModel.cluster_centers_[:, 0], kmeansModel.cluster_c
plt.show()
```



小結

今天介紹了三種常見的機器學習技術，並應用在非監督式學習領域。這些方法在資料分析領域中廣泛應用，例如我們可以解釋 PCA 或 t-SNE 轉換後的特徵如何關聯到原始資料，或者解釋 k-means 分群的結果，以理解每個簇中的資料點之間的共同特徵。這些資訊有助於我們更深入地理解資料集中的訊息和趨勢。

[Day 7] KNN與XAI：從鄰居中找出模型的決策邏輯

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/07.KNN與XAI：從鄰居中找出模型的決策邏輯.ipynb)*

KNN 是一種監督式學習算法，它可以用於分類和迴歸問題。在分類問題中，KNN 透過找到最近的 k 個鄰居取多數決來預測一個新樣本的類別。在迴歸問題中，KNN 則透過找到最近的 k 個鄰居的平均來預測一個新樣本的數值。

> 理論知識可以參考全民瘋AI系列2.0近朱者赤，近墨者黑 - KNN *(https://ithelp.ithome.com.tw/articles/10269826)*

KNN 與其他監督式機器學習演算法的不同之處在於，它是屬於一種 instance-based 演算法。簡單來說 KNN 模型沒有要學習的參數，它只是將所有的訓練資料儲存起來，當遇到新的資料時，就比對新資料和舊資料的相似程度(距離)，來決定新資料的預測。另外 KNN 缺少全局模型可解釋性，因為該模型本質上是局部的，並且沒有明確學習全局權重或結構。

[實作] KNN 的局部解釋性

KNN 方法的可解釋性在局部解釋方面相對較好。因為對於每個單獨的預測結果，我們可以解釋為什麼這些鄰居被選中了，以及這些鄰居中的樣本如何影響預測結果。這種方法在局部解釋方面是相對較好的，因為我們可以查看單個預測的過程和選擇的鄰居，並評估它們對預測的貢獻。其最簡單的方式就是將特徵透過降維方式映射到三維空間內，並透過視覺化的方式觀察 K 個鄰近的點在哪裡。詳細教學可以參考非監督式學習-降維 *(https://ithelp.ithome.com.tw/articles/10267685)*。

以下是使用 KNN 分類鳶尾花朵的範例程式，同時展示了如何使用 kneighbors 方法來解釋模型，查看每個樣本的最近鄰居。

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

# 載入鳶尾花朵資料集
iris = load_iris()
X = iris.data
y = iris.target

# 切分訓練集與測試集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0

# 訓練 KNN 分類器
knnModel = KNeighborsClassifier(n_neighbors=3)
knnModel.fit(X_train, y_train)

# 評估模型的準確率
print('訓練集: ',knnModel.score(X_train,y_train))
print('測試集: ',knnModel.score(X_test,y_test))

# 取得第一個測試樣本的最近的鄰居
distances, indices = knnModel.kneighbors(X_test[[0]])

# 局部解釋：顯示最近鄰居的類別和對應的特徵值
print('Sample belongs to class:', y_test[0])
print('Neighbors belong to classes:', y_train[indices[0]])
print('Nearest neighbors:', X_train[indices[0]])
```

執行結果：

```
訓練集:  0.9619047619047619
測試集:  0.9555555555555556
Sample belongs to class: 2
Neighbors belong to classes: [2 2 2]
Nearest neighbors: [[7.4 2.8 6.1 1.9]
 [7.2 3.2 6.  1.8]
 [7.6 3.  6.6 2.1]]
```

可以看到，測試集的第一個樣本屬於類別 2，其最近的 3 個鄰居也都屬於類別 2。透過 kneighbors 方法，我們可以對模型做局部解釋，解釋每個樣本的預測是基於哪些最近鄰居進行的。

另外 kneighbors 回傳的資訊包括:

- distances: 返回一個維度 (n_samples, n_neighbors) 的陣列，表示每個樣本和它的最近鄰居之間的距離。
- indices: 返回一個維度 (n_samples, n_neighbors) 的陣列，表示每個樣本的最近鄰居的索引。這些索引可以用來檢索訓練資料中對應的最近鄰居。

其中 n_samples 是樣本數量，n_neighbors 是鄰居數量。在這裡，每個樣本的最近鄰居數量都是事先指定的。這個方法回傳的資訊可以用來獲取最近鄰居的特徵和目標值，從而進行模型解釋和分析。

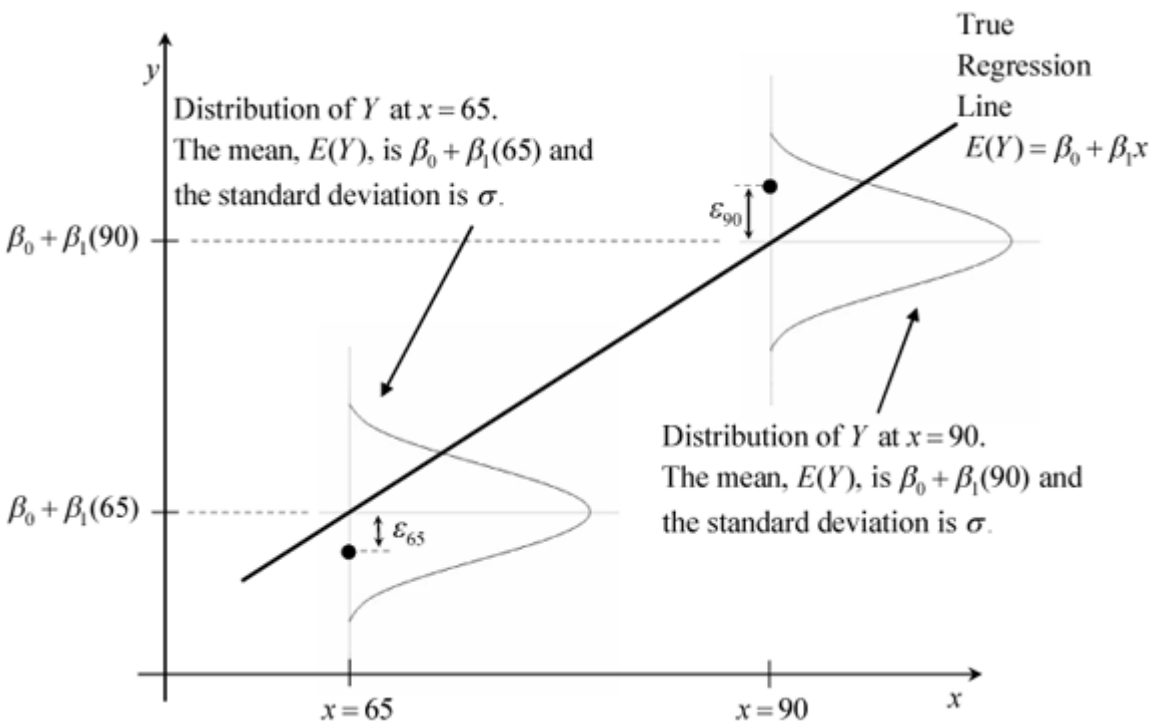
當然我們也可以透過其他方法以全局的觀點解釋 KNN 模型，像是 Permutation Importance 可以資料擾動方式找到重要特徵。關於這部分內容，之後會專門寫一篇文為大家講解！

[Day 8] 解釋線性模型：探索線性迴歸和邏輯迴歸的可解釋性

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/08.解釋線性模型：探索線性迴歸和邏輯迴歸的可解釋性.ipynb)*

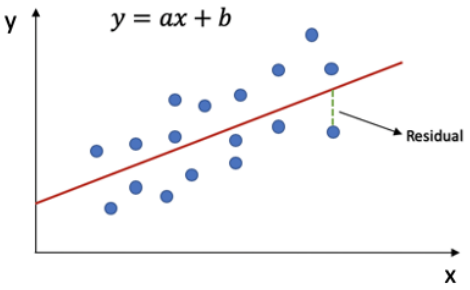
線性迴歸模型

線性迴歸是一種統計學方法，用於建立自變數(x)和應變數(y)之間的線性關係模型。線性迴歸假設應變數是由一個或多個自變數線性組合而成的，且自變數之間不存在多重共線性，誤差項是獨立同分佈的，且呈現常態分佈。



圖片來源：reliawiki *(http://reliawiki.org/index.php/Simple_Linear_Regression_Analysis)*

在線性迴歸中，我們假設應變數和自變數之間存在一種線性關係，並使用最小平方法或梯度下降法來找到最佳擬合直線。整個線性迴歸的目標就是最小化我們的損失函數，因此最後可以用這條直線預測新數據點的值。



線性方程式 $\hat{y}^{(i)} = \theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_d x_d^{(i)}$

尋找 $\theta_0 \ldots \theta_d$ 並最小化殘差的平方和(SSE)

$$\sum_i \left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \sum_i \left(\theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_d x_d^{(i)} - y^{(i)}\right)^2$$

理論知識可以參考全民瘋AI系列2.0線性迴歸 (Linear Regression) *(https://ithelp.ithome.com.tw/articles/10268453)*

解釋線性迴歸模型的方法（係數解釋、截距解釋）

接著我們來探討解釋線性迴歸模型的方法，一種常見的解釋線性迴歸模型的方法是係數解釋。這種方法通過分析迴歸模型的係數，來解釋自變數對應變數的影響。在簡單線性迴歸模型中，自變數的係數可以直接解釋為自變數對應變數的影響程度，例如當自變數增加1個單位時，應變數會增加多少個單位。在多元線性迴歸模型中，則需要考慮多個自變數的影響，通常可以通過控制其他變數的影響，來分析某一個自變數對應變數的影響。

另一種解釋線性迴歸模型的方法是截距解釋。截距在線性迴歸模型中表示當所有自變數均為0時，應變數的期望值，通常被稱為模型的基準值。截距的解釋可以幫助我們理解當自變數均為0時，應變數的期望值是多少。例如，在一個簡單線性迴歸模型中，截距可以解釋為當自變數等於0時，應變數的期望值是多少。在多元線性迴歸模型中，截距的解釋可以幫助我們理解當所有自變數均為0時，應變數的期望值是多少，這對於理解模型的基準值和比較不同模型的表現非常重要。

[實作] 線性迴歸：糖尿病預測

diabetes 資料集是由美國糖尿病資料集中選取的442名病患的生物醫學數據集合。這些數據由10個生理特徵和1個預測目標（糖尿病患者在一年後的疾病進展情況）組成。每個特徵的意義如下：

- Age：年齡
- Sex：性別（1表示男性，0表示女性）
- Body mass index (BMI)：身體質量指數
- Average blood pressure (BP)：平均血壓
- S1：總膽固醇
- S2：低密度脂蛋白
- S3：高密度脂蛋白
- S4：總膽固醇/高密度脂蛋白
- S5：血清甘油三酯水平的對數

- S6：血糖水平

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

# 載入Sklearn糖尿病預測資料集10個輸入特徵1個輸出
diabetes = load_diabetes()
X = diabetes.data # 輸入特徵
y = diabetes.target # 輸出
# 切分訓練集測試集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

使用 Sklearn 建立線性迴歸模型。

```
from sklearn.linear_model import LinearRegression
# 訓練模型
linearModel = LinearRegression()
linearModel.fit(X_train, y_train)
```

模型訓練好以後，就可以觀察線性模型的特徵係數以及截距。

```
# 取得10個特徵係數
print(linearModel.coef_)
# 取得截距
print(linearModel.intercept_)
```

在本例子中輸入有十個特徵，因此線性迴歸式中每個特徵項都會對應一個係數。假設我們從測試集中提取一筆資料預測。由於輸入特徵已經過標準化，因此我們可以很快地觀察各項係數的大小來辦定特徵重要程度。可以發現在什麼都不輸入時會有個基底值(截距)151.346。

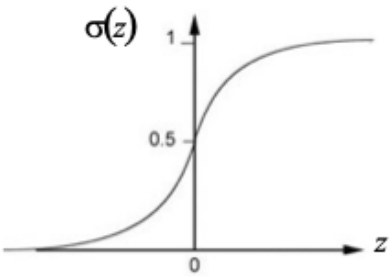
x_j	θj	X_test
截距	151.346	
Age	37.900	0.045
Sex	-241.966	-0.045
BMI	542.426	-0.006
BP	347.708	-0.016
S1	-931.461	0.125
S2	518.044	0.125
S3	163.404	0.019
S4	275.310	0.034
S5	736.189	0.032
S6	48.671	-0.005
輸出		139.225

* 資料已經過標準化

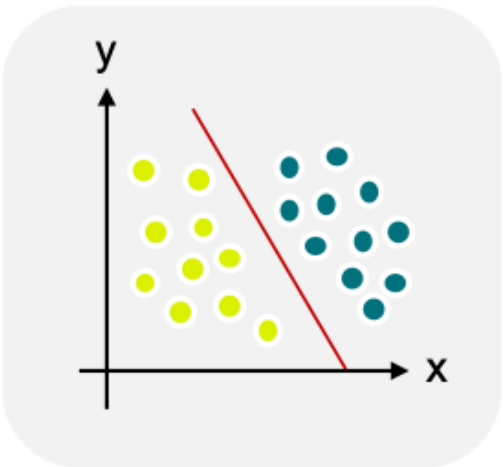
邏輯迴歸模型

線性迴歸和邏輯迴歸差別在於線性迴歸的目標是預測連續數值型的因變量。它基於線性關係建立模型，並且假設因變量和自變量之間存在線性關係。而邏輯迴歸也建立在一個線性模型的基礎上，但是它將線性模型的輸出通過一個稱為 S 形曲線（sigmoid 函數）的轉換，將線性模型的輸出映射到0和1之間，這樣預測結果就變成了二元類別。

$$\sigma(z) = \frac{1}{1 + exp(-z)}$$



註 $z = w^*x + b$



因此我們可以理解這個邏輯迴歸的輸出就相當於，模型根據輸入的特徵判斷結果是1的機率有多高。

$$\hat{y}^{(i)} = P\left(y^{(i)} = 1\right) = \frac{1}{1+e^{-\left(\theta_0+\theta_1 x_1^{(i)}+\cdots+\theta_d x_d^{(i)}\right)}}$$

> 理論知識可以參考全民瘋AI系列2.0邏輯迴歸 (Logistic Regression) *(https://ithelp.ithome.com.tw/articles/10269006)*

解釋邏輯迴歸模型的方法 (odds ratio)

那麼邏輯迴歸該如何解釋呢？首先我們可以說預測出來的 ŷ 相當於是 y(i) 等於1的機率有多高(依據上面的公式)。另外還可以發現一件事情就是我們如果吧 `P(y(i)=1)` 除以 `P(y(i)=0)` 再取 log 之後其實是一個線性的關係。

$$\ln\left(\frac{P\left(y^{(i)}=1\right)}{P\left(y^{(i)}=0\right)}\right) = \theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_d x_d^{(i)}$$

`P(y(i)=1)` 除以 `P(y(i)=0)` 在統計上有個名詞稱之為勝算（odds），也就是「Y=1的機率」與「Y=0的機率」的比率。如果事件發生的機率是 p，那麼其不發生的機率就是 1-p，那麼此事件的勝算比就是 p/(1-p)。

$$\frac{P\left(y^{(i)}=1\right)}{P\left(y^{(i)}=0\right)}$$ 稱之為「勝算」odds

以銅板的例子來說，假設我們想要預測一個公平的銅板翻轉出現正面的機率，由於正反面出現的機率相等，因此 p=1/2。根據上述公式，其勝算比就是 p/(1-p) = (1/2) / (1/2) = 1，也就是說翻轉公平銅板出現正面的勝算比是1。

接著再以一個不公平的銅板為例，假設這個銅板正面朝上的機率是3/5，因此 p=3/5。根據上述公式，其勝算比就是 p/(1-p) = (3/5) / (2/5) = 3/2，也就是說翻轉這個不公平的銅板出現正面的勝算比是 3/2。

基於上述例子我們可以得出以下結論： - 當 odds 大於1時，表示事件的發生機率較高，此時因變數 y 為1的機率較大。 - 當 odds 介於0和1之間時，表示事件的發生機率較低，此時因變數 y 為0的機率較大。

假設對數勝算（log odds）是一個線性函數，我們把 log 搬移到右邊變成取 exp 變成以下式子：

$$\text{odds} = \exp\left(\theta_0 + \theta_1 x_1^{(i)} + \cdots + \theta_d x_d^{(i)}\right)$$

所以我們可以在邏輯迴歸模型中，利用 odds ratio（勝算比）來解釋變數對於反應變數的影響。odds ratio 是指當一個自變數 X 的值增加 1 單位時，對應的勝算會乘上一個倍數，這個倍數即為 odds ratio。

具體來說，如果一個變數 X 的 odds ratio 為 2，表示當 X 增加 1 單位時，Y=1 的機率相對於 Y=0 的機率將會增加 2 倍。而如果 odds ratio 為 0.5，則表示當 X 增加 1 單位時，Y=1 的機率相對於 Y=0 的機率將會減少一半。

- 如果 odds ratio 越大，表示該自變數對於反應變數的影響越大
- 如果 odds ratio 為 1，表示該自變數對於反應變數沒有影響
- 如果 odds ratio 小於 1，表示該自變數與反應變數呈現反向關係。

然而在邏輯迴歸模型中，每個自變量都會有一個 odds ratio 值，可以通過比較 odds ratio 值的大小來判斷哪些自變量對事件發生的機率有較大的影響。在 sklearn 中，可以使用 LogisticRegression 模型的 coef_ 屬性獲取特徵的係數，並將其指數化以得到 odds ratio。以下是一個簡單的範例 code：

```
from sklearn.linear_model import LogisticRegression
import numpy as np

# 載入自己的資料集
X = ...略
y = ...略

# 建立 Logistic Regression 模型並訓練
model = LogisticRegression()
model.fit(X, y)

# 獲取係數
coef = model.coef_

# 將係數指數化得到 odds ratio
odds_ratio = np.exp(coef)
```

在這個範例中，X 是訓練資料，y 是每筆資料對應的標籤。獲取 coef_ 屬性之後，可以使用 np.exp() 函數將係數指數化。這樣可以得到所有特徵的 odds ratio 值，近一步的分析當 X 的值增加 1 單位時，所對應的勝算來觀察特徵重要性。

小結

線性迴歸用於預測一個連續的數值，例如預測股票的價格或房屋的價格等。它假設自變數和因變數之間有線性關係，也就是說，當自變數的值增加一個單位時，因變數的值也會按比例增加或減少。

邏輯迴歸用於預測二元變數，例如預測某人是否患有某種疾病、電子郵件是否為垃圾郵件等。邏輯迴歸使用 sigmoid 函數將輸出限制在0到1之間，表示因變數屬於某個類別的機率。如果機率大於0.5，那麼我們就預測為1，否則預測為0。

線性迴歸的基本概念和假設是建立在數學統計學理論上，理解和掌握這些概念和假設可以幫助我們更好地應用線性迴歸模型進行實際問題的解決。綜上所述，線性迴歸模型的係數和截距是解釋自變數對應變數的影響和模型基準值的重要因素，透過對這些因素的分析，可以幫助我們更好地理解線性迴歸模型的行為。

[Day 9] 基於樹狀結構的XAI方法：決策樹的可解釋性

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/09.基於樹狀結構的XAI方法：決策樹的可解釋性.ipynb)*

- 透過特徵重要性了解機器學習模型的決策過程

決策樹

決策樹是一種監督式學習演算法，用於解決分類或迴歸問題。該方法透過對訓練集資料的分析來建構一棵樹狀結構的模型，其中每個內部節點都代表一個特徵，每個葉子節點代表一個分類或迴歸結果。對於新的數據樣本，它可以通過從根節點開始遍歷樹狀模型，並將其歸類到相應的葉子節點中。

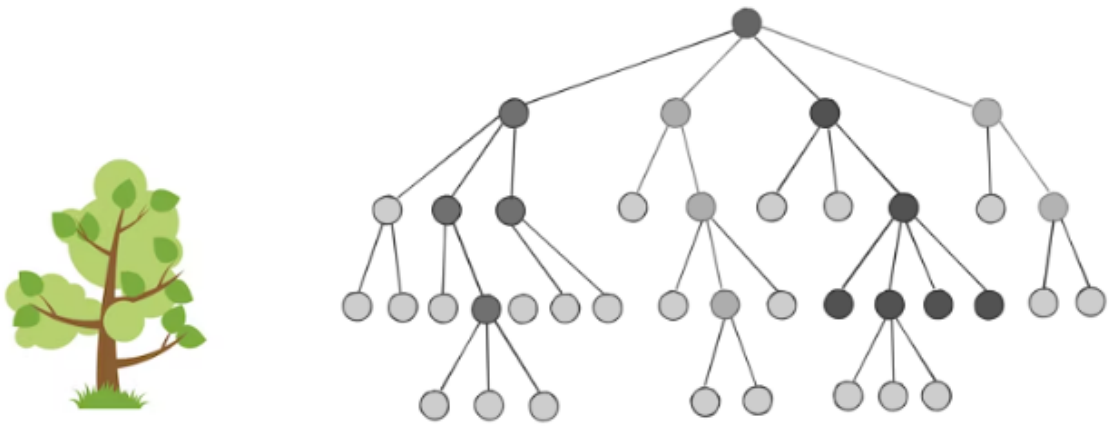


Image source: freepik.com *(https://www.freepik.com/)*

決策樹的概念是給定一組訓練資料，透過所有的特徵試法找到合適的特徵分成幾組後純度的分數要越小越好。衡量純度(亂度)的指標大致分為 Entropy 與 Gini index 兩種。

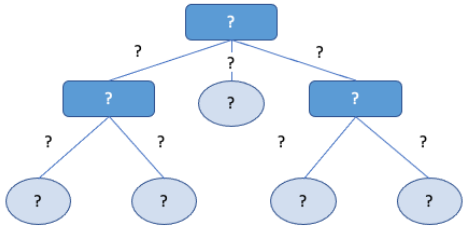
- Entropy：熵是一種衡量系統不穩定程度的指標，熵越高表示系統越不穩定。
- Gini index：吉尼係數是另一種用於計算亂度的指標，也可以用於判斷數據集的純度。數值越小表示數據集越純。

理論知識可以參考全民瘋AI系列2.0決策樹 (Decision tree) *(https://ithelp.ithome.com.tw/articles/10271143)*

決策樹如何解釋？

以下舉例說明如何透過年齡、收入、是否為學生以及信用評等特徵，來預測一個人是否會購買電腦。我們的目標是利用下面這張表格建立一棵決策樹，以便未來預測時能夠依據此樹進行判斷。問題在於，我們該如何利用這張表格來建立決策樹呢？

年齡	收入	是否學生	信用	有無購買電腦
<=30	high	no	fair	no
<=30	high	no	excellent	no
31~40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31~40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31~40	medium	no	excellent	yes
31~40	high	yes	fair	yes
>40	medium	no	excellent	no



在建出這棵樹之前我們要先算在做任何事之前這個資料到底有多亂？如果用 Entropy 來算這十四人當中有九位會買電腦，另外五位不會買電腦。我們可以表示成 Info(9, 5) 算出來的亂度是 0.94。也就代表在我不做任何事情之前這十四位總共有 0.94 這麼亂。

設 D 為訓練樣本中的一組分割，則 D 的訊息熵表示為

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

接下來要做的是使用第一個年齡特徵把這十四位分成三組，並算算看分完結果有多亂。計算完後同時也算看看收入、是否學生、信用評等這幾個特徵分完後計算這個 Entropy 是最純的。如果我們用年齡把人分成三組，整體的亂度是 0.694(越小越好)。因此我們可以求得使用年齡這個特徵得到的純度進步了 0.246。

年齡	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31~40	4	0	0
>40	3	2	0.971

Positive: 是否買電腦 = yes
Negative: 是否買電腦 = no

越小越好，代表很純不亂

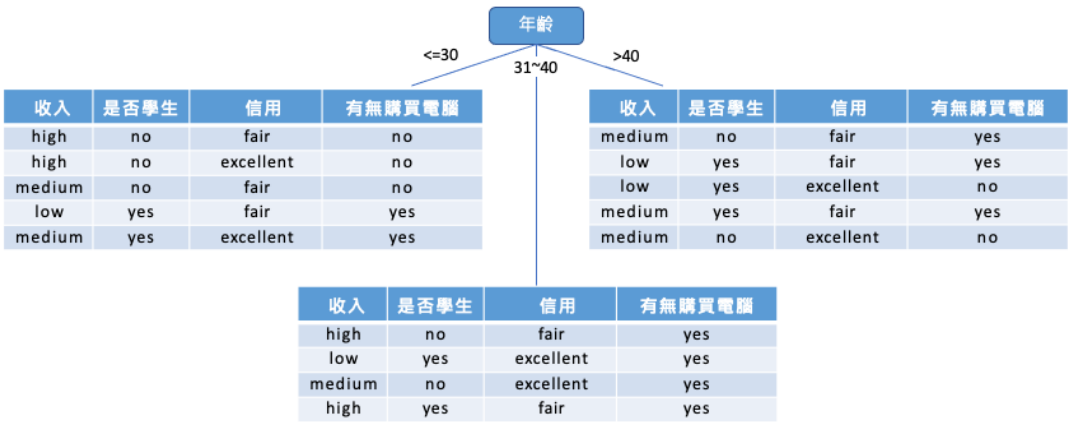
$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$$\text{Gain(年齡)} = Info(D) - Info_{\text{年齡}}(D) = 0.246$$

同理我們可以計算及他的特徵所帶來的純度，可以發現這四個特徵第一個步驟用年齡來分會讓結果是最純的。

- Gain(年齡) = 0.246
- Gain(收入) = 0.029
- Gain(是否學生) = 0.151
- Gain(信用) = 0.048

因此第一步會將年紀分為三組，接著在年輕這個節點我們可以很清楚地看出使用是否學生這個特徵可以很完美的將這五人分開。

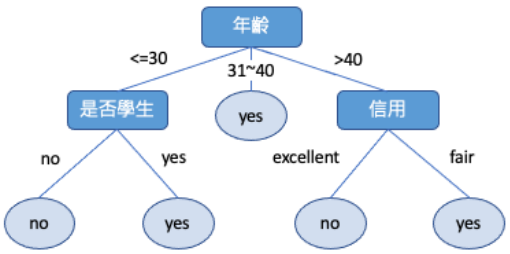


收入	是否學生	信用	有無購買電腦
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

收入	是否學生	信用	有無購買電腦
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

收入	是否學生	信用	有無購買電腦
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

因此我們可以透過剛剛的方法依序計算，選擇最適合的特徵進行最佳分類。最後建立出這棵樹，進而能夠提供很好的解釋性。



- If (年齡 == 大於等於30) and (是否學生 == no): no
- If (年齡 == 大於等於30) and (是否學生 == yes): yes
- If (年齡 == 介於31~40) and (是否學生 == yes): yes
- If (年齡 == 大於等於40) and (信用 == excellent): no
- If (年齡 == 大於等於40) and (信用 == fair): yes

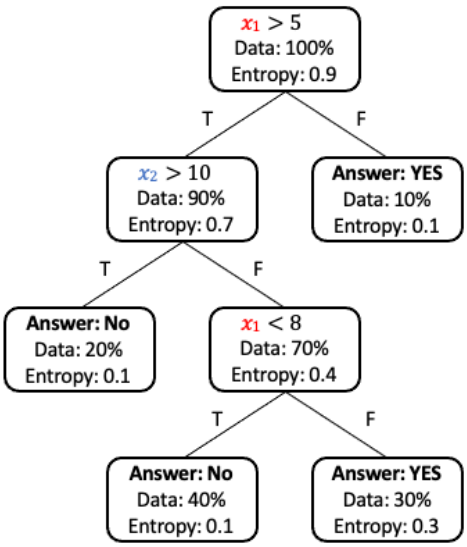
決策樹的特徵重要性

決策樹也可以透過特徵重要性計算哪些特徵是重要的。想想看如果你建出一棵樹，什麼叫做一個特徵很重要？首先是如果這個特徵影響很多樣本的決策，再者該特徵透過亂度計算可以讓結果很純的話該特徵扮演重要角色。因此我們會希望能夠透過以上兩點綜合考量，合理的判斷哪個特徵重要。

$$I_f = \sum_{\forall n_f \in N} P(n_f)\Delta_{entropy}(n_f)$$

N ：所有的樹節點
n_f ： 使用特徵 f 區分數據點的一個節點
$P(n_f)$ ：受 n_f 影響的數據點的百分比
$\Delta_{entropy}(n_f)$：在節點 n_f 分割前後，純度分數的減少量

假設有兩個特徵 X1 和 X2，其原始資料亂度為 0.9 經過 x1 特徵下去切割後得到的純度進步了 0.26。接下來一樣計算左邊分枝透過 x2 特徵切割後純度進步了 0.37。經過以上反覆計算我們可以得知每個節點透過某特徵切割後純度會進步多少？下一步就是計算每個節點有多少的資料比例會經過，之後就可以計算 x1 跟 x2 有多重要了。以 x1 來說第一個節點所有樣本都會被拿來評估因此 100%×0.26，接著在另個節點 x1 有百分之七十的樣本會被拿來評估，且評估完後純度平均會進步 0.21 因此 70%×0.21。綜合加起來 x1 的重要程度有 0.407。



Information gains

$$\Delta_{entropy}(x_1 > 5) = 0.9 - (0.9\times0.7 + 0.1\times0.1) = 0.26$$

$$\Delta_{entropy}(x_2 > 10) = 0.7 - (0.22\times0.1 + 0.78\times0.4) = 0.37$$

$$\Delta_{entropy}(x_1 < 8) = 0.4 - (0.57\times0.1 + 0.43\times0.3) = 0.21$$

Feature importance

$$I_{x_1} = 100\%\times0.26 + 70\%\times0.21 = 0.407$$

$$I_{x_2} = 90\%\times0.37 = 0.333$$

[實作] 決策樹分類器解釋

以下範例使用鳶尾花朵資料集。程式碼中使用了 load_iris() 函式將鳶尾花朵資料集載入，將資料存放在 X 與 y 兩個變數中，分別代表資料集中的特徵值與目標變數。最後使用 train_test_split() 函式將資料集切分成訓練集與測試集。

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np

# 載入鳶尾花朵資料集
```

```
iris = load_iris()
X = iris.data
y = iris.target

# 切分訓練集與測試集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

接著使用 sklearn 中的 DecisionTreeClassifier *(https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)* 建立了一個決策樹模型。首先，使用 criterion = 'entropy' 指定模型使用資訊熵作為分裂標準，這表示決策樹會優先選擇分裂後的子集更加純粹（即更少的混亂）的特徵。此外我們可以對模型進行一些客製化的設定，max_depth=4 是指定決策樹的最大深度為 4 層，這表示決策樹最多可以分裂出 4 層子樹。各位可以試著觀察隨著樹深度越大是否能夠讓模型變得更準確。

```
from sklearn.tree import DecisionTreeClassifier

# 建立 DecisionTreeClassifier 模型
decisionTreeModel = DecisionTreeClassifier(criterion = 'entropy', max
# 使用訓練資料訓練模型
decisionTreeModel.fit(X_train, y_train)
```

訓練結束後可以透過 `get_depth()` 方法查看決策樹模型的深度。若在建立模型時為給定 max_depth 數值，則模型在訓練期間樹會不斷擴展，直到所有的葉子節點都是純的（即樣本屬於同一個類別），或者葉子節點包含的樣本數小於 min_samples_split 所設定的值。

> max_depth參數的設定會影響決策樹的複雜度和過擬合問題，通常需要通過交叉驗證等方式進行調參。

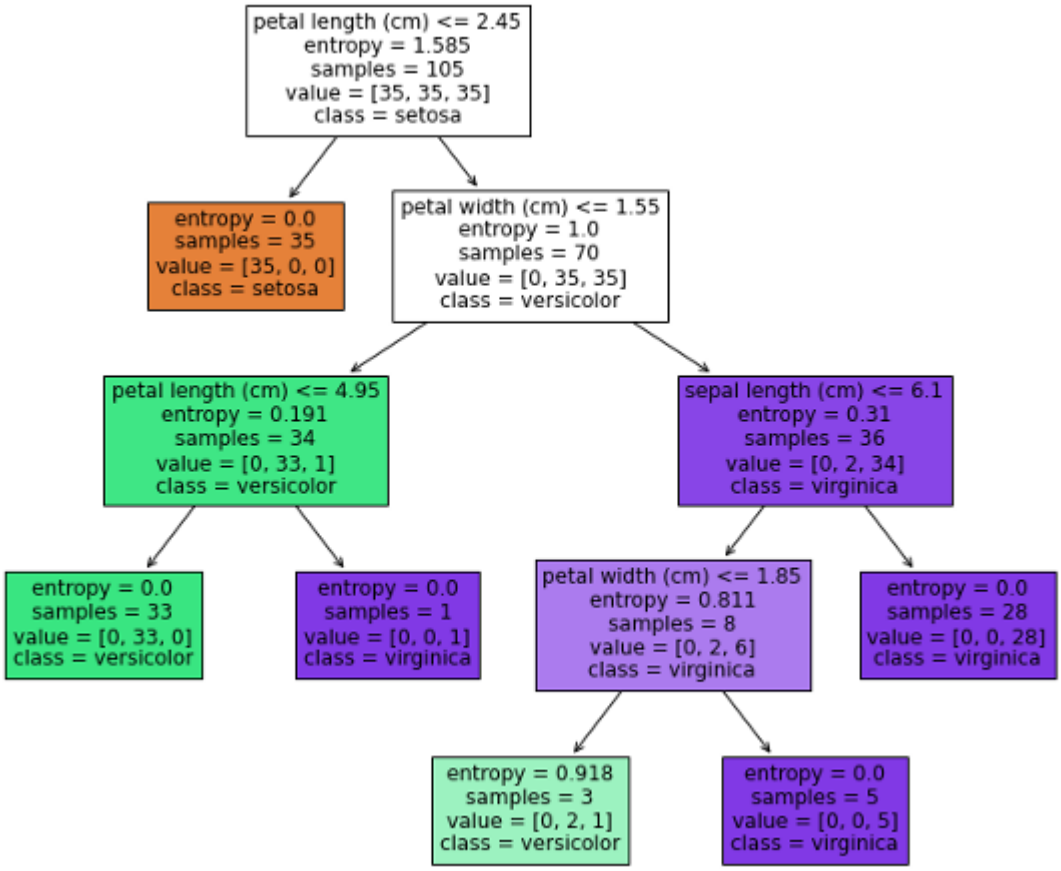
```
print('決策樹最大深度: ',decisionTreeModel.get_depth())
```

執行結果：

```
決策樹最大深度:  4
```

plot_tree *(https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html)* 是 sklearn 中用於繪製決策樹模型的方法。該方法可以繪製出已擬合決策樹模型的樹形結構，以便進行模型的解釋和視覺化。

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 10))
plot_tree(decisionTreeModel, feature_names=iris.feature_names, class_
plt.show()
```



我們也可以透過 `feature_importances_` 來取得決策樹模型中各個特徵的重要性值。重要性值越高的特徵，表示在決策樹中其影響力越大，能夠對預測結果產生較大的影響。

```
importances = decisionTreeModel.feature_importances_
indices = np.argsort(importances)[::-1]
for f in range(X_train.shape[1]):
    print(f'{feature_names[indices[f]]}: {importances[indices[f]]}')
```

執行結果：

```
petal length (cm): 0.6289008293445076
petal width (cm): 0.34266728207712693
sepal length (cm): 0.028431888578365407
sepal width (cm): 0.0
```

從結果可以看出，花瓣的長度（petal length）是影響鳶尾花分類的最重要特徵，其重要性得分為 0.6289，其次是花瓣的寬度（petal width），其得分為 0.3427。而花萼長度（sepal length）對分類影響相對較小，得分僅為 0.0284。另一方面，花萼寬度（sepal width）的得分為 0，表示在這個模型中，花萼寬度對於鳶尾花的分類貢獻非常小或者可以忽略不計。我們也可從視覺化的圖中發現確實花萼寬度沒有在任一節點中出現。

Reference

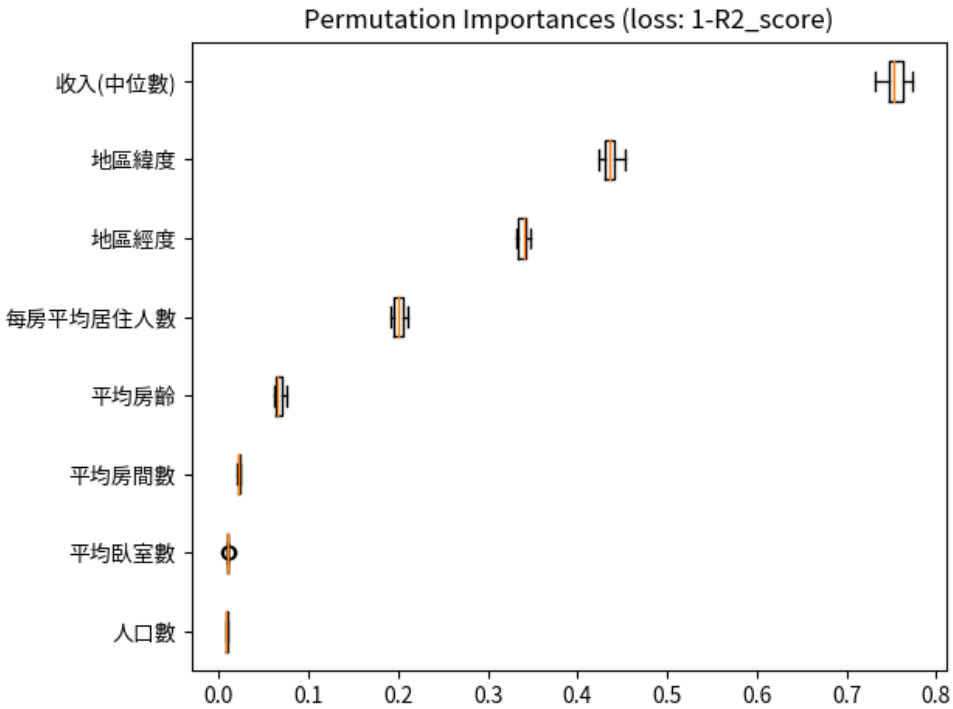
- cse352 DECISION TREE CLASSIFICATION *(https://www3.cs.stonybrook.edu/~cse352/L8DTIntro.pdf)*

[Day 10] Permutation Importance：從特徵重要性角度解釋整個模型行為

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/10.Permutation_Importance：從特徵重要性角度解釋整個模型行為.ipynb)*

全局模型解釋是試圖理解整個模型的行為，而不僅僅是對單個預測進行解釋。Permutation importance 方法就是一種廣泛用於評估機器學習模型特徵重要性的技術之一，而且它是一種與模型無關的可解釋技術。

此模型解釋方法通常會返回每個特徵的重要性分數，分數越高則該特徵對於模型的影響性越大。因此當我們對其中一個特徵數值修改後對整體預測結果變動很大，可以推論該特徵可能是重要的。具體來說就是把一個特徵的某個值替換成另一個值之後，到底會對預測 error 有多大的影響。以預測區域房價為例，Permutation importance 算出來結果越小的話代表該特徵一點都不重要，有無它都沒關係。而計算出來越大的特徵，例如收入對於該地區的房屋價格預測是有很大的貢獻。



演算法流程

Permutation importance 的基本概念是將資料集中的每個特徵進行隨機重排，然後測試模型在重排後數據上的推論變化。如果某個特徵隨機排列對模型的預測產生了顯著的影響，則可以推斷該特徵對模型的重要性較高。

X								y
收入(中位數)	平均房齡	平均房間數	平均臥室數	人口數	每房平均居住人數	地區緯度	地區經度	地區房屋價格中位數
8.33	41	6.98	1.02	322	2.56	37.88	8.33	4.526
8.30	22	6.24	0.97	2401	2.11	37.86	8.30	3.585
...	...	...	...	...	...	...	...	...
5.64	52	5.82	1.07	558	2.55	37.85	5.64	3.413
3.85	52	6.28	1.08	565	2.18	37.85	3.85	3.422

實際步驟如下：首先用訓練資料訓練一個模型，接著用測試資料及對每筆資料做預測並計算一個分數。R2 分數是評估一個迴歸模型有多好，所以 1-R2 代表預測結果有多不好。而 AUC 是衡量分類問題的指標越接近 1 越好，因此 1-AUC 可以視為該模型預測多麼不好，類似於 Error 的感覺。當然分類問題也可以使用準確率。接著如果我們要對特徵 x 來查看有多重要，就是隨機的對 x 特徵替換掉其他的數值最後再計算這個 Error 有多大。因此 Permutation importance 計算某特徵的重要程度公式為，搗亂之後的error/原始乾淨資料集的error，又或是兩個分數相減。

1. 訓練一個模型，可以得到一個基準的評估指標，例如準確率或 R2 等。
2. 使用測試資料集對每筆資料做預測並計算一個error分數。
    - 迴歸模型： 1-R2
    - 分類模型：1-Accuracy
3. 隨機的對 x 特徵內的所有數值替換掉，並算出error。
4. 計算每個特徵重要性
    - 搗亂之後的error-原始乾淨資料集的error

    重複2、3步驟直到所有特徵都執行過一遍。

使用 eli5 實作 Permutation Importance

eli5 *(https://github.com/eli5-org/eli5)* 是一個解釋機器學習模型的 Python 套件，其中提供了 Permutation Importance 的方法。它可以計算模型的特徵重要性，並將其可視化為一個長條熱圖，可以快速地比較每個特徵的重要性。首先透過 pip 安裝 eli5 套件：

```
pip install eli5
```

接著載入今天的範例測試集。fetch_california_housing *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html)* 是 sklearn 中的一個內建資料集，用來預測加州地區的房屋價格中位數。這個資料集包含了 8 個特徵，分別是：

- MedInc：該區域內家庭收入的中位數
- HouseAge：該區域內房屋的平均房齡
- AveRooms：該區域內房屋的平均房間數
- AveBedrms：該區域內房屋的平均臥室數
- Population：該區域內人口數
- AveOccup：該區域內平均每個房屋的居住人數
- Latitude：該區域內房屋所在緯度
- Longitude：該區域內房屋所在經度

這個資料集包含了 20640 筆樣本，每個樣本都有上述 8 個特徵以及房屋價格中位數作為目標變數。

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import numpy as np

# 載入加州地區房屋價格預測資料集
data = fetch_california_housing()
feature_names = np.array(data.feature_names)
X, y = data.data, data.target

# 切分資料集為訓練集和測試集
X_train, X_test, y_train, y_test = train_test_split(X, y, random_stat

# 訓練隨機森林迴歸模型
model = RandomForestRegressor(random_state=0).fit(X_train, y_train)
```

本範例訓練一個隨機森林模型，並使用了 eli5 套件中的 PermutationImportance 方法，用於計算每個特徵在模型中的重要性。程式碼中的 model 是指建立好的機器學習模型，而 X_test 和 y_test 則是測試資料集。

```
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(model, random_state=42).fit(X_test, y_te
eli5.show_weights(perm, feature_names = feature_names, top=8)
```

上述程式碼中使用 PermutationImportance 方法來計算每個特徵的重要性，並將結果存入 perm 變數中。在計算過程中，random_state 參數用於確保結果的再現性。接著使用 eli5.show_weights 方法將結果可視化。feature_names 參數是特徵的名稱列表，top 參數則用於

指定要顯示的重要性排名前幾的特徵。這個方法將會顯示每個特徵的名稱以及其對應的重要性分數，分數越高表示該特徵對模型的影響越大。

Weight	Feature	
0.7688 ± 0.0290	MedInc	收入(中位數)
0.4392 ± 0.0142	Latitude	地區經度
0.3421 ± 0.0121	Longitude	地區緯度
0.2046 ± 0.0076	AveOccup	每房平均居住人數
0.0687 ± 0.0046	HouseAge	平均房齡
0.0222 ± 0.0030	AveRooms	平均臥室數
0.0103 ± 0.0033	AveBedrms	平均房間數
0.0085 ± 0.0020	Population	人口數

上面的結果，可以看到最重要的特徵是該地區的收入中位數，確實挺合理的。若出現紅色負數的分數其實都可以視作幾乎不重要的變數。

小結

Permutation importance 的優點是可以用於任何模型演算法和任何資料集，並且可以提供關於每個特徵的直觀解釋。值得注意的是該方法忽略了特徵之間的交互作用，即特徵之間可能存在依賴關係，某些特徵在單獨考慮時看似不重要，但在與其他特徵結合後可能會產生重要的影響。因此，Permutation importance 方法不能完全代表特徵對模型的影響，而應當結合領域知識、模型內部解釋等多種方法進行綜合評估。另外它的缺點是計算量較大，特別是在資料集較大、特徵較多的時候。若想同時考慮到兩個特徵對於輸出的影響該怎辦？明天就會來跟大家介紹另一種 Model Agnostic 全局解釋的方法 Partial Dependence。

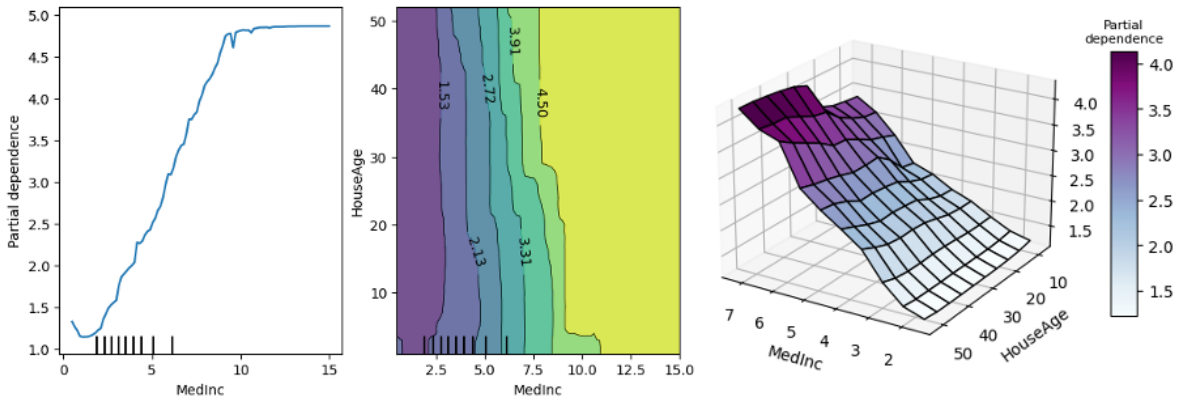
Reference

- 具有多重共線性或相關特徵的置換重要性 *(https://scikit-learn.org.cn/view/254.html)*
- （機器學習）可解釋性(2) Permutation Importance *(https://medium.com/@hupinwei/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E5%8F%AF%E8%A7%A3%E9%87%8B%E6%80%A7-machine-learning-explainability-%E7%AC%AC%E4%BA%8C%E8%AC%9B-c090149f0772)*

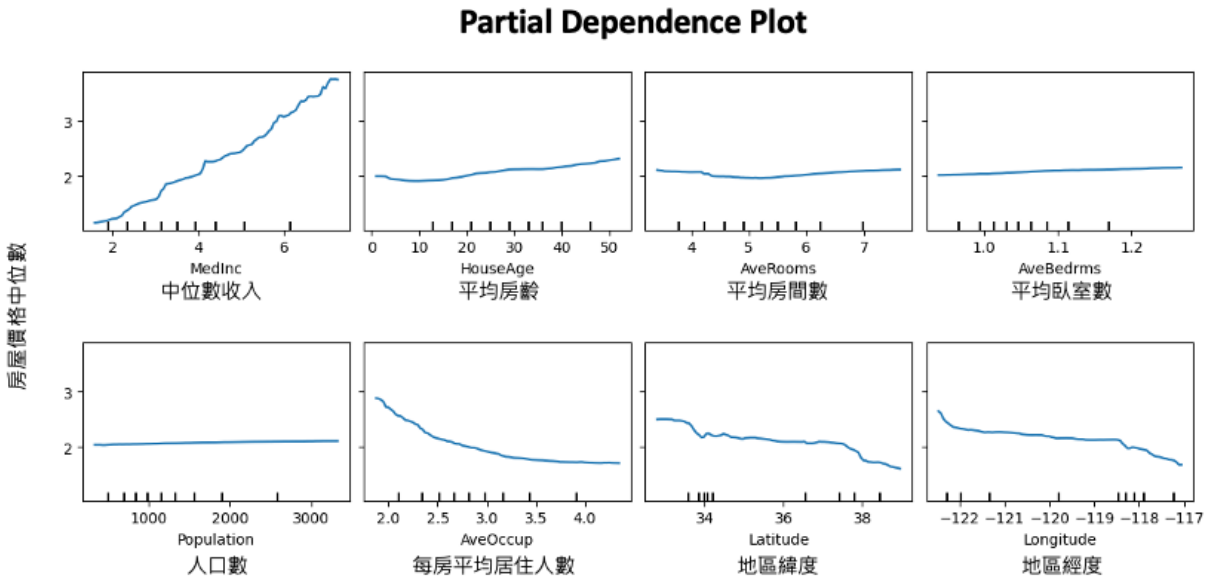
[Day 11] Partial Dependence Plot：探索特徵對預測值的影響

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/11.Partial Dependence Plot：探索特徵對預測值的影響.ipynb)*

Partial Dependence Plot（PDP）是要觀察每一個自變數的變化是如何影響預測表現，它可以快速地分析自變數與目標變數之間的關係。而昨天所提的 Permutation Importance 是要觀察某個特徵會有多大的程度影響預測的錯誤率，進而取得重要程度排序。除此之外 PDP 也能考慮到特徵的交互作用，可以同時觀察兩個或以上的特徵合起來如何去影響預測結果。但以實務面來說為了可以視覺化預測結果，通常 PDP 最多會同時考慮兩個特徵，因為超過三個維度就比較難用視覺去呈現結果。今天討論的方法也是一種與模型無關的可解釋技術。



我們再以房價預測為例，可以從下面這八張 Partial Dependence 圖觀察哪些特徵是對於 y（房價）的影響是比較明顯的。其中越平坦的曲線代表該特徵越不重要，這意味著我不管怎樣變動該特徵的值對於預測結果都差不多，反之，上下起伏比較明顯的線代表該特徵極為重要。例如左上角的中位數收入，從中我們可以發現，該特徵對模型預測來說只要稍微變動一個單位對於輸出結果有明顯的影響。同時可以說明該特徵從美金 1.58 到 7.23 萬美元，隨著收入逐漸地增長，影響該地區的房價是正相關的。



我們可以透過 PDP 逐一觀察每個特徵跟輸出的關係。

Partial Dependence 演算法流程

接下來要來討論 Partial Dependence 是如何被計算出來的。首先我們有一組訓練資料共有 d 個特徵，因此訓練一個模型 f 輸入為 x1~xd，目標是要預測 y。透過蒙地卡羅法希望可以觀察某個 xj 特徵的值它的相對應輸出是多少。因此我們可以從訓練集中抽取 m 筆資料樣本，並將要被觀察的特徵 xj 固定一個數值 t，而其他的特徵就用剛剛隨機抽出的樣本依序帶值進去。每次帶一筆進去後都會輸出一個 y，總共會得到 m 個結果最終在平均取來的數值就是 xj 對於 t 的情況下所估計的結果。

給定一個已訓練好的模型 $\hat{y} = f(x_1, \ldots, xd)$ ，我們可以通過以下公式來估計 $f(x_j = t)$：

$$f(x_j = t) = \frac{1}{m}\sum_{i=1}^{m} f\left(x_1^{(i)}, \ldots, x_{j-1}^{(i)}, t, x_{j+1}^{(i)}, \ldots, x_d^{(i)}\right),$$

其中 $x_k^{(i)}$ 是第 i 筆資料的第 k 個 feature 的值， m 是抽樣的筆數

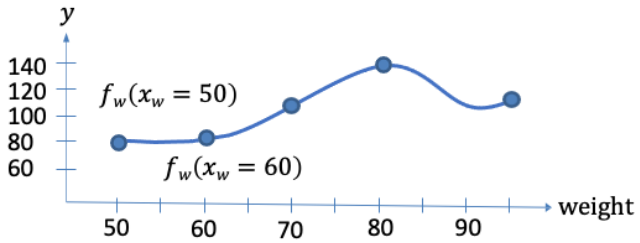
1. 從測試集或資料集中，隨機取出一個樣本z
2. 把z的特徵 xj 改為 t，其他特徵不變，得到新樣本z'
3. 透過訓練好的模型，得到新樣本的預測值 $\hat{y}'=f(z')$
4. 重複以上步驟 m 次，得到 m 個預測值 $\hat{y}'1,\hat{y}'2,\ldots,\hat{y}'m$
5. 計算平均值，即為 $f(xj=t)$ 的估計值。
6. 重複以上步驟，得到其他特徵的 PDP。

至於t要有幾個取決於該特徵的上下界，並透過grid_resolution參數設定要在這範圍間取得幾組做為t。

這裡舉個例子，我們想要用身高和體重兩個特徵去預測一個人的血壓。假設我們要觀察體重50公斤的血壓是多少，我們會先從資料集中採樣 m 筆資料(這裡有四筆，m=4)。接著從這四筆資料中固定體重都改成50，其餘身高保值不變並放入事先訓練好的模型進行預測得到 ŷ。最終把這四個模型預測的 ŷ 加總做平均就可以得到預測體重 50 公斤的人平均的血壓是多少。同樣以此概念我們可以得到隨著體重的變化(60kg, 70kg...)模型預測出來的血壓應該是多少。

- $f_w(x_w = 50) \approx \frac{1}{4}\left(f(155, 50) + f(165, 50) + f(172, 50) + f(175, 50)\right)$
- $f_w(x_w = 60) \approx \frac{1}{4}\left(f(155, 60) + f(165, 60) + f(172, 60) + f(175, 60)\right)$

⋮



ID	Height	Weight	y
1	155	50	90
2	165	70	120
3	172	85	145
4	175	80	125

基於 Partial Dependence 的特徵重要性分析

在之前的內容中，我們已經學到如果一個特徵的 Partial Dependence Plot（PDP） 呈現出「平坦」的曲線，代表該特徵對目標變數的影響不重要。但是，如何測量一個曲線的「平坦度」呢？其中一個方法是使用標準差來衡量其反平坦度。如果我們評估了函數 fj(xj=vk)（其中 k=1,...,K），則特徵 xj 的重要性可以定義為其標準差。計算完每個特徵的標準差後，其標準差越大的特徵我們可以視為越重要。

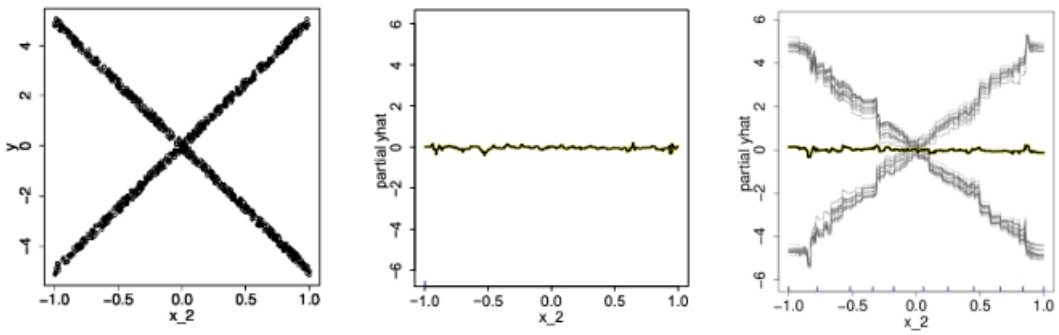
$$I(x_j) = \sqrt{\frac{1}{K-1}\sum_{k=1}^{K}\left(f_j(x_j = v_k) - \bar{f}_j(x_j)\right)^2}$$

使用 PDP 尋找重要特徵時必須要注意資料的合理性，怎麼說呢？若 xj 與其他特徵存在某種關係，我們隨機抽取樣本填入可能抽到極不合理的特徵組合。例如上面的預測血壓例子當中可能會產生(身高180cm,體重40kg)的極端組合，這種情形現實生活中較難出現此案例，因此可能錯估結果。

PDP 對於資料異質性的影響

雖然 PDP 有助於可視化預測響應與一個或多個特徵之間的平均部分相依關係。但在存在實質交互作用影響下，部分響應關係可能是異質的（heterogeneities）。因此像 PDP 這樣的平均曲線可能會掩蓋建模關係的複雜性。所以我們可以透過 ICE 圖繪製個別觀測值的預測響應與特徵之間的函數關係來改進部分相依圖。在下圖 a 中，我們繪製了樣本中的 X2 與 Y 的散點圖。圖 b

顯示了預測特徵 X2 的擬合模型的部分相依圖。PDP 顯示就平均而言，X2 與預測的 Y 沒有顯著的相關性。但是從圖 a 中可以明顯看出，這個結論是錯誤的。很明顯 X2 與 Y 有關聯，只是 PDP 中的平均化掩蓋了這一發現。因此 PDP 的缺點在於可能掩蓋資料中的異質性。因此還有另一種變形是個體條件期望圖（ICE Plot），如圖 c 所示。ICE plot 可以清楚地顯示預測值和特徵 X2 之間的關係。從中我們可以發現，此模型的預測值在特徵 X2 的不同區域內呈現近似線性增加或減少的趨勢。



(a) Scatterplot of Y versus X2　(b) Partial Dependence Plot　(c) ICE Plot

ICE Plot

剛剛的例子中 PDP 在展示單一特徵對預測值的影響時，可能會掩蓋掉特徵和其他特徵之間的交互作用，進而限制了 PDP 的解釋能力。為了克服這個問題，在篇論文：Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation *(https://arxiv.org/pdf/1309.6392.pdf)* 作者提出了 ICE plot（Individual Conditional Expectation plot）的概念，可以更好地展示特徵之間的交互作用。ICE plot 類似於 PDP，但是它繪製的是每個觀測值的個別條線，因此可以更好地呈現特徵之間的差異和交互作用。

1. 從測試集或資料集中，隨機取出一個樣本z
2. 把z的特徵 xj 改為 t，其他特徵不變，得到新樣本z'
3. 透過訓練好的模型，得到新樣本的預測值 $\hat{y}'=f(z')$
4. 重複以上步驟 m 次，得到 m 個預測值 $\hat{y}'1,\hat{y}'2,...,\hat{y}'m$
5. 將這 m 個預測值與原始樣本z的特徵xj的值一起畫在圖上
6. 重複以上步驟，得到其他特徵的 ICE Plot。

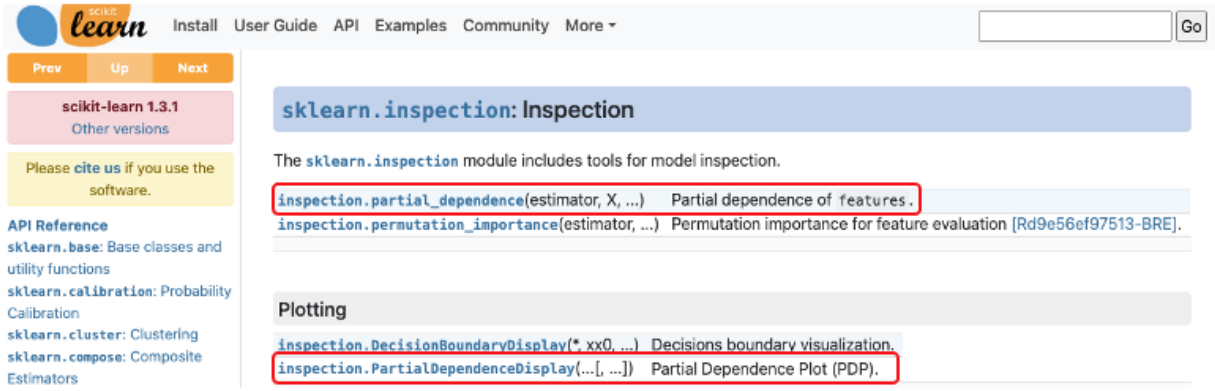
sklearn 實作 Partial Dependence

在 sklearn 中有提供兩種 PDP 的 API 分別為 partial_dependence 和 PartialDependenceDisplay，兩者是用於呈現部分相依 (partial dependence) 的工具，但它們的使用方式和顯示效果有所不同。

- partial_dependence 是一個函數，可用來計算部分相依值並返回結果。

- PartialDependenceDisplay 可以直接繪製部分相依圖，透過視覺化方法比較不同特徵的影響。

本日的內容建議在 Python 3.8 和 scikit-learn 1.2.2 版本以上執行。



參考: sklearn.inspection: Inspection *(https://scikit-learn.org/stable/modules/classes.html#module-sklearn.inspection)*

今天的範例延續昨天的房價預測資料集，並先訓練好隨機森林迴歸模型。

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import numpy as np
import pandas as pd

# 載入加州地區房屋價格預測資料集
data = fetch_california_housing()
feature_names = np.array(data.feature_names)
X, y = data.data, data.target

# 切分資料集為訓練集和測試集
X_train, X_test, y_train, y_test = train_test_split(X, y, random_stat

# 訓練隨機森林迴歸模型
model = RandomForestRegressor(random_state=0).fit(X, y)
```

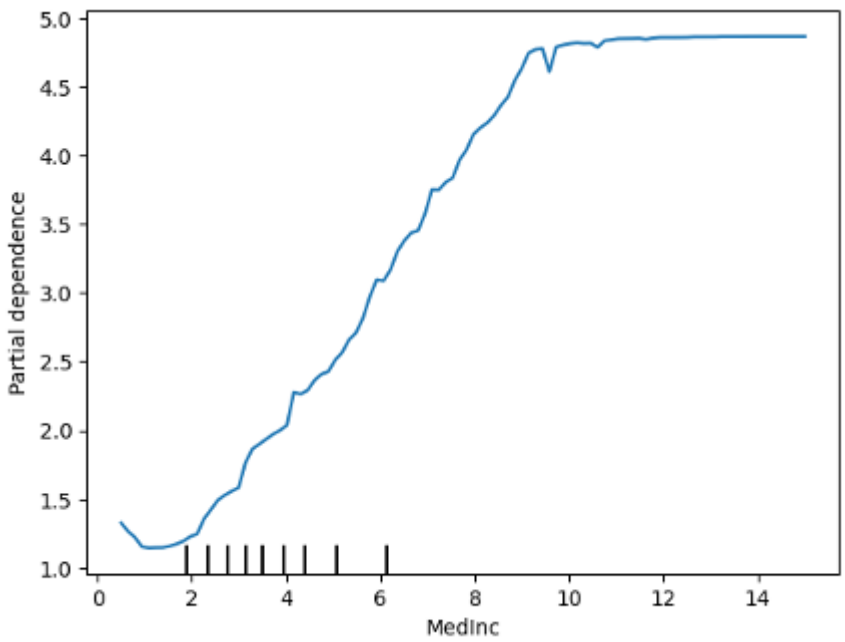
接著是本日實作重點，這裡採用 `PartialDependenceDisplay` 方法搭配 from_estimator 放入事先訓練好的 sklearn 模型，並放入測試資料集進行全域的模型解釋。以下是該方法常用到的參數，若想更進階使用可以參考官方文件 *(https://scikit-learn.org/stable/modules/generated/sklearn.inspection.PartialDependenceDisplay.html)*。

Parameters: - estimator: 已訓練的預測器物件，必須實作 predict、predict_proba 或 decision_function 方法。 - X: 可放入訓練資料或是測試資料集，另外要注意是否有前處理。 - features: 需要分析的特徵，可以為一個或多個特徵，若為一個則生成單特徵的部分相依圖，若

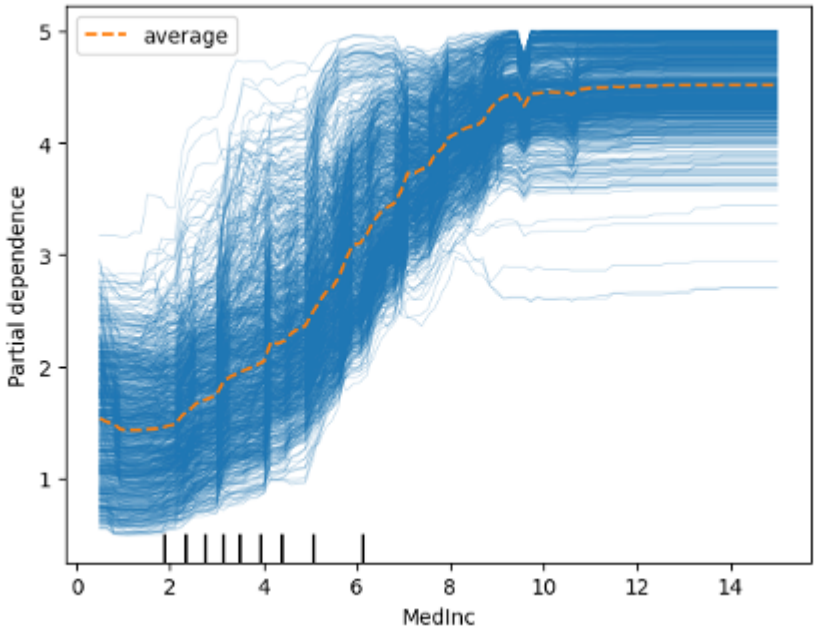
兩個則生成2-way部分相依圖（kind='average'時才支援），每個元素可以是特徵索引或特徵名稱或特徵索引或特徵名稱的tuple。 - categorical_features: PDP也支援類別型的特徵解釋，用法跟 features 一樣。 - feature_names: 用於指定每個特徵的名稱，預設值為None。 - grid_resolution: 網格解析度，數值越大越能呈現特徵與目標變數之間的關係，預設值為100。 - percentiles: 用於限制PDP的X軸上下界，以百分位數表示，預設值為(0.05, 0.95)。 - centered: 是否將ICE和PD線的起始點設為y軸原點從0開始，預設為False。 - kind: 以字串格式設定，提供三種繪製形式 'average' 就是單純的PDP繪製;'individual' 繪製ICE plot;'both' 則代表同時繪製PD與ICE plot。預設為'average'。

```
from sklearn.inspection import PartialDependenceDisplay

PartialDependenceDisplay.from_estimator(model,X_test, ['MedInc'],
                                        feature_names=feature_names,
                                        centered =False,
                                        kind='average',
                                        percentiles=(0, 1),
                                        grid_resolution=100)
```

輸出結果：



試著將參數修改 `kind='both'`，我們就可以同時得到 ICE Plot 以及中間橘色線條就是每個樣本平均後的 PDP。



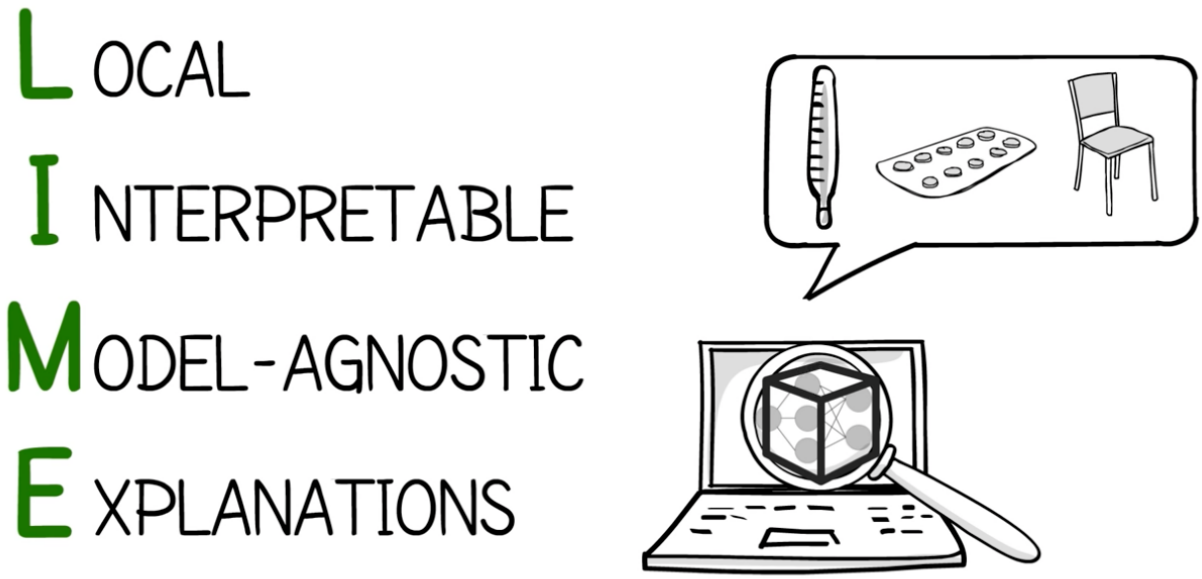
Reference

- Friedman, Jerome H. "Greedy function approximation: A gradient boosting machine. *(https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-5/Greedy-function-approximation-A-gradient-boostingmachine/10.1214/aos/1013203451.full)*" Annals of statistics (2001): 1189-1232.
- Alex goldstein. (2014). Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation *(https://arxiv.org/pdf/1309.6392.pdf)*. Arxiv.

3.XAI常用工具介紹

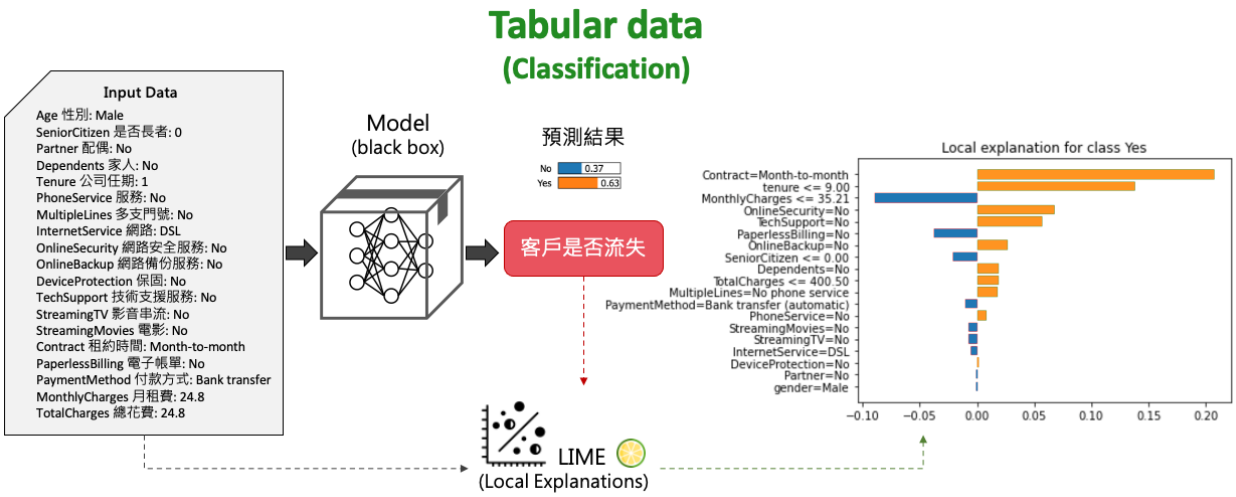
[Day 12] LIME理論：如何用局部線性近似解釋黑箱模型

LIME 的全名是 Local Interpretable Model-agnostic Explanations， 其目的是可以分析模型對於某筆資料為何做出特定的決策， 並且可以應用於神經網路、SVM、隨機森林、XGBoost 等各種模型。它可適用於表格資料(分類＆迴歸)、文字和影像， 並提供對單筆資料的解釋。

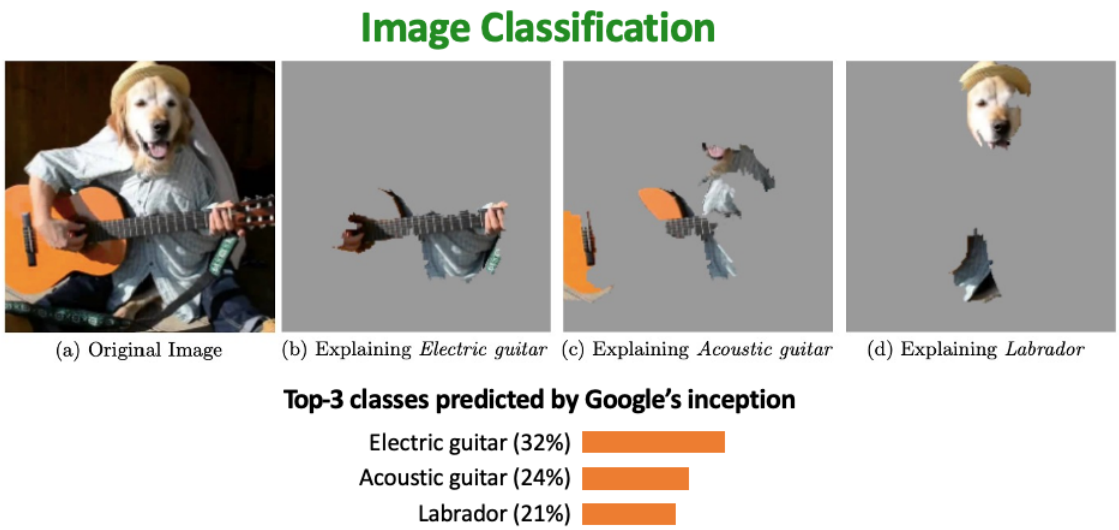


- KDD2016 Conference原作者介紹LIME影片 *(https://www.youtube.com/watch?v=hUnRCxnydCc)*
- 原始論文：" Why should i trust you?" Explaining the predictions of any classifier. *(https://arxiv.org/abs/1602.04938)*

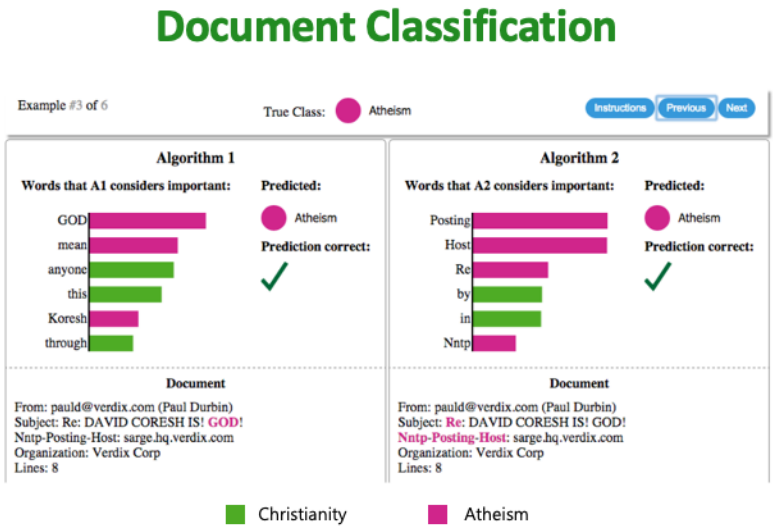
首先以表格資料以分類任務為例， 將一筆顧客訂單丟入以訓練好的分類模型後。即可透過 LIME 分析是否可能即將流失。並且說明哪些特徵是影響客戶流失的重要因子。



第二個例子是影像分類，我們將一張圖片丟入 Google inception 模型進行預測得到前三名的結果為電吉他(32%)、古典吉他(24%)、拉布拉多(21%)。透過 LIME 可以做出為何模型會預測某種類別的原因，解釋為何分類為電吉他的證據以及其它類別像是拉布拉多的證據在哪。

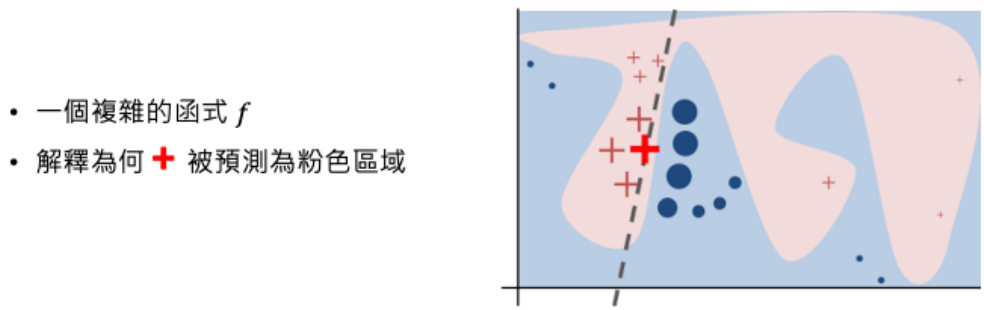


最後一個例子是檔案文件分類，分辨是無神論的文章或是基督教內容的文章。這邊有兩種演算法對同一篇文章進行辨識，透過 LIME 可以分析兩種演算法對於該類預測結果的證據為何。透過用字遣詞可以確認模型是否有真的學到辨識的精髓，而不是過擬合。



LIME 運作原理

這裡我們來談談 LIME 如何實現。假設資料有兩個特徵，粉紅與藍色是透過某種機器學習演算法圍出來的邊界。接著我們要分析為何+被預測為粉紅的類別。LIME 的做法是先從訓練資料集中隨機的採樣一些資料出來並丟入訓練好的模型(f)進行預測得到結果。預測完之後我們拿這些採樣預測後的結果與要被分析解釋的那筆資料+進行距離計算。離+資料點越近的資料重要性就越高(圖中+和o會越大)。最終會生成一個非常簡單並可解釋的模型g(例如線性模型)，其目標訓練一個簡單 g 模型使得預測結果越接近 f 所預測的結果越好(並非 ground true)。訓練一個簡單模型的前提是距離被分析的那筆資料越接近的資料要與 f 預測結果越接近越好。因此就可以透過以訓練好的線性模型對該筆資料進行解釋。



LIME的運作方式如下： 1. 生成一個新的數據集，其中包含經過排列的樣本，並使用黑盒模型預測。 2. 在新的數據集上訓練一個可解釋模型，根據抽樣實例與目標實例的接近程度加權。 3. 將新的數據集訓練一個簡單模型。

LIME – tablular dataset

這裡以一個實際表格資料做展示，對於表格數據，LIME 的抽樣來源通常來自訓練集。假設從資料集中採樣 N 筆資料，並丟入以訓練好的模型進行預測並得到標籤。x 是一筆想被分析的測試

資料，將會與剛被採樣每筆去計算距離。第三步驟訓練一個簡單易解釋的模型，如果 x 離採樣的資料很近的話我們期望要與原本 f 的模型預測結果要相近。透過該目標函式定義好後，就能訓練一個簡單的 g 模型。

目標：找到一個自身可解釋的模型g(如線性模型) 近似f，使得$\mathcal{L}(f,g,\pi)$最小。

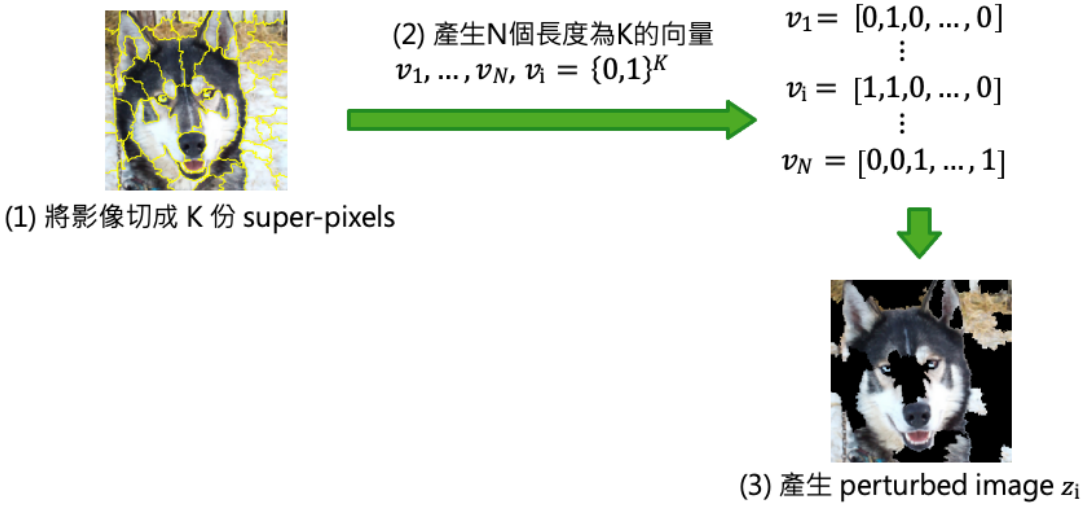
損失函數： $\mathcal{L}(f,g,\pi_x) = \sum_{z,z'\in Z} \pi_x(z)\left(f(z)-g(z')\right)^2$

其中權重大小 $\pi_x(z) = exp(-D(x,z)^2/\sigma^2)$

其中距離函數D文字可選餘弦距離，對圖像或表格資料可選L2距離。

LIME – image dataset

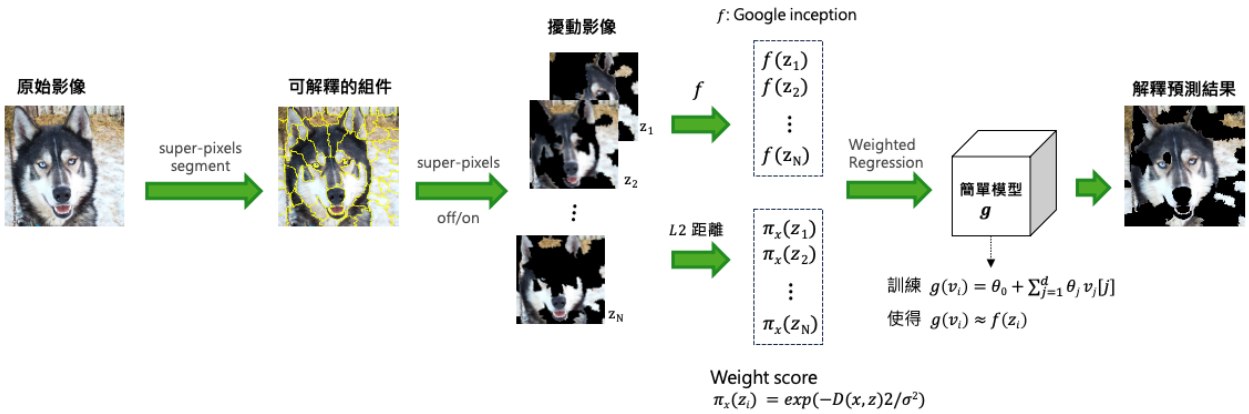
首先透過影像處理技巧將一張影像切成 K 份 super-pixels，因此同一個區塊可能是類似的物件。接著產生N個長度為K的向量，perturbed image 的意思就是當向量中 0 就代表保留該區塊 1 去除該區塊。因此會產生 N 張不同的影像，每張影像中可能都有幾個 super-pixels 為空白。



(1) 將影像切成 K 份 super-pixels

(3) 產生 perturbed image z_i

將一張圖片透過 super-pixels 產生出 N 張不同的圖片，即代表目前手上有了 N 張與原圖相似的圖片。接著將這 N 張圖片丟入訓練好要被解釋的 Google inception 模型中，預測這 N 張圖片的類別。同時必須計算這 N 張圖片與原圖有多像，算的方式就是每個向量跟全部是 0 的向量(代表沒有抹除是完整圖)去計算歐氏距離(L2 Distance)，因此算出來越接近 1 代表與原圖越相似。最後建立一個線性迴歸模型 g 使得跟原本模型 f 預測越相近越好。最終把特別重要的 super-pixels 秀出來。



小結

LIME（Local Interpretable Model-Agnostic Explanations）是一種與「模型無關」的解釋方法，它可以被用於任何機器學習模型，而不受特定模型算法或架構的限制。並且適用於表格數據、文字和圖像。這使得 LIME 成為一個通用的解釋工具，可應用於不同領域和問題場景。此外 LIME 提供的解釋是基於單個實例的，這使我們能夠更深入地了解模型在個別實例上的預測和決策依據。

Reference

- Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. "" Why should i trust you?" Explaining the predictions of any classifier. *(https://arxiv.org/abs/1602.04938)*" Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 2016.
- 使用 LIME 解釋複雜的分類模型 *(https://taweihuang.hpd.io/2018/02/27/introtolime/)*
- LIME可解釋性分析-ImageNet圖像分類 *(https://blog.csdn.net/m0_59286668/article/details/128426336)*
- Unboxing the black box using LIME *(https://towardsdatascience.com/unboxing-the-black-box-using-lime-5c9756366faf)*

[Day 13] LIME實作：實戰演練LIME解釋方法

範例程式：  *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/13.LIME實作：實戰演練LIME解釋方法.ipynb)*

今天我們將深入探討 LIME 的實作細節，包括如何選擇解釋性模型和解釋特徵，以及如何選擇鄰域大小來生成解釋數據。

若想了解 LIME 的核心原理可以參考前一篇文章：[Day 12] LIME理論：如何用局部線性近似解釋黑箱模型 *(https://ithelp.ithome.com.tw/articles/10327698)*

LIME 的優缺點

LIME 是一種機器學習可解釋的技術，它的基本原理是在解釋模型時，透過建立一個局部解釋性模型，來近似黑盒模型的預測結果。這讓我們能夠針對特定的輸入樣本，解釋模型是如何進行判斷和預測的。以下為各位統整 LIME 的優缺點：

- LIME 的優點：
    - 可以解釋任何複雜模型。
    - 能夠針對某一筆資料進行解釋。
    - 提供簡單容易理解的解釋方法(線性模型ex: Ridge Linear Regression)。
- LIME 的缺點：
    - 對於鄰域的定義仍然是一個未解決的問題（最好的方法是嘗試不同的核函數設置，並測試哪一個效果最好）。
    - 資料的抽樣方法與數量無一定的標準，因此容易影響解釋結果。
    - 簡單模型採用線性易忽略特徵之間的相關性，有時可能產生不合理的解釋結果。
- 輸入的資料若存在嚴重共線性問題，則會使迴歸估計不准確。

[實作] LIME 解釋分類模型

這裡我們會以一個糖尿病預測資料集訓練一個 XGBoost 分類器。接這透過 LIME LimeTabularExplainer 訓練一個簡單模型並對一筆資料進行解釋。首先我們先載入今天範例的資料集，該資料集可以從 Kaggle 資料科學平台取得 *(https://www.kaggle.com/datasets/mathchi/diabetes-data-set)*。

載入資料集

```
import pandas as pd

# 讀取資料集
df_train = pd.read_csv('./diabetes.csv')
```

讓我們來瞧瞧 df_train 裡面的內容。我們可以發現該資料集有總共有 768 筆數據，每一筆資料有八個欄位資訊，其中包含模型的輸入與輸出。

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
2	183	64	0	0	23.3	0.672	32	1
3	89	66	23	94	28.1	0.167	21	0
4	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...
763	101	76	48	180	32.9	0.171	63	0
764	122	70	27	0	36.8	0.340	27	0
765	121	72	23	112	26.2	0.245	30	0
766	126	60	0	0	30.1	0.349	47	1
767	93	70	31	0	30.4	0.315	23	0

768 rows × 8 columns

這個資料集來自美國國家糖尿病和消化和腎臟疾病研究所。其目標是根據診斷測量來預測病人是否患有糖尿病。資料集的變數如下：

- Glucose：口服葡萄糖耐量測試中2小時的血漿葡萄糖濃度，用於測試糖尿病的診斷。
- BloodPressure：舒張壓(mm Hg)，血壓中的一個參數，用於衡量心臟在收縮時的壓力。
- SkinThickness：三頭肌皮膚褶皺厚度(mm)，用於衡量皮膚的脂肪層厚度。
- Insulin：2小時血清胰島素(mu U/ml)，用於評估胰島素水平，對糖尿病的診斷非常重要。
- BMI：身體質量指數，表示體重和身高的比例，用於評估體重狀況。
- DiabetesPedigreeFunction：糖尿病家族遺傳函數，用於衡量患有糖尿病的家族遺傳風險。
- Age：病人的年齡。
- Outcome：病人是否患有糖尿病(作為模型輸出)，值為0表示沒有糖尿病，值為1表示患有糖尿病。

切割資料集

接下來從剛剛讀取進來的 df_train 資料集中，將所有的輸入特徵資料提取出來，作為模型的輸入 X。同時，我們從 df_train 中取得 `Outcome` 欄位的資料，作為模型的輸出 y，用來表示病人是否患有糖尿病。除此之外，我們也將所有輸入特徵的欄位名稱儲存到 `x_feature_names` 變數

中，y_label_names 則是儲存輸出的標籤名稱，這兩個個變數將在後續 LIME 模型解釋的過程中使用。最後透過 train_test_split 方法切割訓練集與測試集。

```
from sklearn.model_selection import train_test_split

x_feature_names = df_train.drop(['Outcome'], axis=1).columns
y_label_names = ['No', 'Yes']
X = df_train.drop(['Outcome'], axis=1).values # 移除y並取得剩下欄位資料
y = df_train['Outcome'].values # 取得病人糖尿病結果作為y

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

訓練模型 (XGBoost 分類器)

以下使用 XGBoost 分類器（XGBClassifier）來建立一個模型，並使用訓練資料（X_train, y_train）來訓練這個模型。

```
from xgboost import XGBClassifier

# 建立 XGBClassifier 模型
xgboostModel = XGBClassifier()
# 使用訓練資料訓練模型
xgboostModel.fit(X_train, y_train)
```

LIME 解釋模型

再來是本文的重頭戲。首先大家可以在終端機輸入以下指令安裝 LIME 套件：

```
pip install lime
```

LimeTabularExplainer 用於解釋表格數據中的模型預測，根據訓練數據和模型預測結果生成解釋模型，並提供局部解釋結果。透過設置不同的參數，可以自定義解釋器的行為，以滿足不同的解釋需求。以下是常用的設定參數與說明：

- training_data：訓練集資料 X。
- mode：str，用於指定是分類還是迴歸模型，{"classification", "regression"}。
- feature_names：list of names (strings)，特徵名稱，對應於訓練數據中的列。
- categorical_features：list of indices (ints)，類別特徵的索引列表，這些特徵的值必須是整數(OrdinalEncoder前處理)。
- categorical_names：map from int to list of names，類別特徵的名稱映射，其中 categorical_names[i][j] 表示第 i 列中第 j 個值的名稱。
- verbose：bool，是否顯示LIME局部模型預測值。

- class_names：list of class names, 類別名稱列表, 按照分類器使用的順序排列。如果未提供, 類別名稱將是 '0'、'1' 等。
- feature_selection：str, 特徵選擇方法, 預設值為auto, {'forward_selection', 'highest_weights', 'lasso_path', 'none', 'auto'}。詳細內容參考 *(https://lime-ml.readthedocs.io/en/latest/lime.html#module-lime.lime_base)*
- random_state: 亂數種子, 確保每次執行結果都一樣。

```
import lime
from lime import lime_tabular

lime_explainer = lime_tabular.LimeTabularExplainer(
    training_data= X_train,
    feature_names= feature_names,
    mode='classification',
    class_names=y_label_names,
    verbose=True,
    feature_selection='lasso_path',
    feature_selection='none',
    random_state=44
)
```

我們剛剛已經建立了一個 LimeTabularExplainer 並且初始化解釋模型的設定後, 接下來就可以試著丟一筆要被解釋的資料並透過 LIME 訓練一個簡單線性模型並解釋。解釋每一筆預測數據則是呼叫 `explain_instance()` 方法。以下是常用的設定參數與說明：

- data_row：一維numpy, 對應於要解釋的一筆資料。也就是要進行解釋的特定數據樣本。
- predict_fn：已訓練的預測模型。對於分類器, 這應該是一個函數, 接受一個 numpy 數組作為輸入, 並輸出預測機率。對於回歸模型, 這個函數接受一個 numpy 數組並返回預測值。對於 ScikitClassifiers, 這是 classifier.predict_proba()。對於 ScikitRegressors, 這是 regressor.predict()。
- top_labels：預設為None, 適用於分類模型可以輸入要被解釋的前k個預測標籤的模型解釋。
- num_features：解釋的特徵數量。這個參數限制了解釋結果中所顯示的特徵數量。預設為10。
- num_samples：用於學習線性模型的鄰域大小。該參數控制了生成解釋所需的樣本數量。預設為5000。

這裡需要注意的是, LIME 在訓練一個簡單的線性模型時, 使用的是 Ridge 線性模型。在訓練過程中, 會根據使用者呼叫 `explain_instance()` 方法時設定的 num_features 參數, 從訓練集中挑選出合適的特徵進行訓練。而模型如何選擇特徵, 則取決於建立 LimeTabularExplainer 時所設定的 forward_selection 參數。若想觀察所有特徵的解釋性, 可以將該參數設置為 'none', 這樣模型就會考慮所有輸入 X 的特徵。以下程式碼從測試集中拿第六筆資料進行解釋, 並且選擇要解釋的特徵設定為七個, 代表 LIME 會針對所有特徵進行解釋。

```
lime_exp = lime_explainer.explain_instance(
    data_row=X_test[5],
    predict_fn=xgboostModel.predict_proba,
    num_features=7)
```

最後呼叫 `show_in_notebook()` 即可觀察 LIME 如何對該筆資料進行解釋。

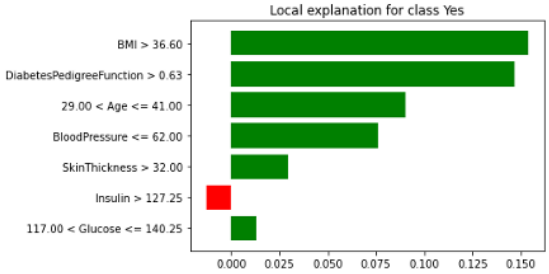
```
lime_exp.show_in_notebook(show_table=True, show_all = True)
```

- Intercept 是生成的線性模型的截距項。在 LIME 中，為了解釋模型的預測結果，會使用一個線性模型來近似原始黑盒模型。Intercept 是這個線性模型中的截距項，表示在沒有任何特徵貢獻的情況下，模型預測的基本值。
- Prediction_local 是從線性模型中預測的局部預測結果。在 LIME 中，針對特定的一筆資料樣本，會生成一個局部的線性模型，用來解釋該樣本的預測結果。Prediction_local 是這個局部線性模型對於該樣本的預測值。
- Right 是從被解釋的分類器（XGBoost）中得到的預測值。在 LIME 中，會使用一個弱模型（分類器或迴歸器）來進行解釋，該弱模型是用來模擬原始黑盒模型的預測行為。

我們可以從上面的結果分析這筆資料的預測結果。首先，在沒有加入任何特徵時，線性模型預測患有糖尿病的機率為 0.21，這是截距項的影響。接著，我們觀察每個特徵的影響。以 BMI 為例該項特徵會造成糖尿病正面的影響是因為輸入的數值大於 36.6 這個閾值，將其輸入值 43 乘以 LIME 線性模型的 x1 係數，得到 0.15。這表示 BMI 特徵的加入會增加預測患有糖尿病的機率約 0.15。依此類推，每個特徵都加入後，最後加總起來得到這筆資料的預測結果為 0.712。在這個二元分類的範例中，這表示有相當高的機率被預測為患有糖尿病。而 XGBoost 模型預測該筆資料的機率為 0.69。

$$x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + x_4 * w_4 + x_5 * w_5 + x_6 * w_6 + x_7 * w_7 + Intercept = Prediction\_local$$

LIME模型預測結果： 0.154+0.147+0.09+0.076+0.029-0.013+0.013+0.216=0.712



SP-LIME 總體貢獻

雖然單筆預測的解釋可以部分地讓使用者了解模型的決策方式，但這並不能完整地反映模型整體的性能和可靠性。若要全面的理解整個模型可能需要看很多筆資料的解釋。但是要看幾筆資料才足夠多呢？在原始論文中提供了一個選擇資料筆數的演算法稱作 Submodular pick (SP)。

小結

LIME 提供了一種有效的方法，能夠針對黑盒模型進行解釋，幫助我們更理解模型推論的規則。今天透過 LIME 解釋模型的特徵重要性，我們可以發現哪些特徵對於模型的預測結果影響較大，進而幫助我們優化模型，提高其性能和準確性。此外在某些應用中，模型的誤判可能導致嚴重的後果。透過 LIME 可以幫助我們找出模型預測的薄弱環節，並進行風險評估和風險管理。所以模型的解釋性至關重要，因為模型的預測可能直接影響最終決策。

Reference

- 使用 LIME 解釋複雜的分類模型 *(https://taweihuang.hpd.io/2018/02/27/introtolime/)*
- interpretability-of-deep-learning-models *(https://towardsdatascience.com/interpretability-of-deep-learning-models-9f52e54d72ab?gi=6403abc9f660)*

[Day 14] SHAP理論：解析SHAP解釋方法的核心

Shapley values 簡介

Shapley values 最早是由經濟學家 Lloyd Shapley 所提出，用於評估參與合作博弈的每個玩家對於勝利的貢獻價值。在機器學習中，我們可以將這個概念應用到輸入特徵 X 對於輸出的影響上，透過 Shapley values 可以計算每個特徵對於輸出的貢獻。簡單來說，Shapley values 計算的是在各種情況下，如果某個特徵被加入到模型中，它會帶來多少額外的貢獻。最後我們將所有特徵的額外貢獻的期望值加總，就得到了所謂的 Shapley values。透過這種方法可以幫助我們理解模型中每個特徵對於輸出的影響程度，進而提高模型的可解釋性和可信度。

簡單例子解釋 Shapley values

假設團隊有三個人共同接了一個案子，根據每個人的貢獻程度將酬勞按比例分配。透過計算 Shapley values，我們能夠評估每個成員在完成任務時的貢獻大小，以實現更公平的酬勞分配。以下是三個成員的性格特質及在互相合作和獨立作業時所能達成的貢獻分數。其分數範圍在0到100之間，代表他們參與完成案子的能力與效率。

- A(工程師) 喜歡獨立作業，能力強，是團隊中的核心人物。
- B(工程師) 謹慎按部就班，能確保任務進行順利且無差錯。
- C(專案經理) 無產出能力，但擅於溝通並管理整個團隊的工作。

從下表中可以依據三位在團隊中加入的順序進行排列組合，其中 ϕ 代表團隊都沒有人的情況下產出當然為 0，接著可以得知 A、B、C 三人獨立工作的分數分別為 90, 60, 0。至於合作的部分 A+B 共同工作的產出為 80，從這組合當中可以發現 B 的加入使得共同產出下降了 10(A就是典型的單打獨鬥，有人插手反而拖累進度)。其餘的 A+C 和 B+C 的工作產出分別為 70 和 80。最後 A+B+C 三人各自發揮專長，且有了 C 的溝同協調才使得工作順利完成使分數得到滿分100。

團隊組成	分數
ϕ	0
A	90
B	60
C	0
A+B	80
A+C	70
B+C	80
A+B+C	100


A 團隊核心


B 按部就班


C 善於管理

有了上表產出分數後，我們就可以去計算每個人在團隊的貢獻值。其計算方式根據所有組隊成員加入的所有順序，再去計算在各種順序下每個人額外的貢獻個是多少。首先計算 A 的貢獻程度，三個人總共有六種加入的組合順序，如下表所示：

加入順序	A 的額外貢獻
A, B, C	$v(A)-v(\phi)= 90$
A, C, B	$v(A)-v(\phi)= 90$
B, A, C	$v(A+B)-v(B)= 80-60= 20$
B, C, A	$v(A+B+C)-v(B+C)= 100-70= 30$
C, A, B	$v(A+C)-v(C)= 70-0= 70$
C, B, A	$v(A+B+C)-v(B+C)= 100-70= 30$

成員 A 的 Shapley values 計算：

$$\phi_1 = \frac{1}{3!}(90 + 90 + 20 + 30 + 70 + 30) = 55$$

- 首先第一個加入順序為 A,B,C， A 所扮演的角色為第一個因此是 90。
- 第二個組合 A,C,B 跟第一個案例一樣由於 A 加入時還是空的因此還是由 A 自己出力貢獻為 90。
- 若 B,A,C 的順序由 B 先做然後 A 再加入，此時 A 有加入或沒加入的差別就是 A 和 B 一起組隊，或是 B 自己。因此 A 在這一組合中所帶來額外的貢獻為 (A+B)-B=20。
- 若 B,C,A 的話 A 所帶來的額外貢獻為，(A+B+C)-(B+C)=30
- 若 C,A,B 的話 A 所帶來的額外貢獻為，(A+C)-(C)=70
- 若 C,B,A 的話 A 所帶來的額外貢獻為，(A+B+C)-(B+C)=30

到目前為止已經計算完在所有的加入組合順序中 A 所帶來的額外貢獻，因此 A 的 Shapley values 就是將所有的組合的貢獻分數做一個平均得到 55。

成員 A 的 Shapley values 計算：

$$\phi_1 = \frac{1}{3!}(90 + 90 + 20 + 30 + 70 + 30) = 55$$

上面步驟依此類推在六種組隊可能的順序中為每個人分別計算帶來的額外貢獻分數，平均就可以得到在這一個任務中每個人的貢獻程度了。最後我們可以分別計算出 A、B 和 C 的 Shapley values，如下表所示，最後得到 A 為 55，B 為 40，C 為 5，可以發現這三個值相加其實就等於 100。

團隊組成	分數
ϕ	0
A	90
B	60
C	0
A+B	80
A+C	70
B+C	80
A+B+C	100

加入順序	A 的貢獻	B 的貢獻	C 的貢獻
A, B, C	90	-10	20
A, C, B	90	30	-20
B, A, C	20	60	20
B, C, A	30	60	10
C, A, B	70	30	0
C, B, A	30	70	0
平均	55	40	5

SHAP (SHapley Additive exPlanations)

以上述的例子每個人就等同於機器學習中的 X 特徵，而 y 就是輸出。我們要觀察每個特徵 X 對於計算 y 的貢獻程度有多少，因此我們必須去計算所有特徵的 Shapley values 各是多少。但是當 X 的特徵數量太多的時候，假設有 d 個特徵的時候我們就相當於要計算 2的d次方-1種可能性，這也意味著要個別訓練這麼多模型。假設有兩個特徵，則要計算 Shapley values 就必須訓練三個模型：

- 第一個模型：f(x1)
- 第二個模型：f(x2)
- 第三個模型：f(x1, x2)

若特徵數量太多的時候訓練大量的模型代價是很高的，因此可以透過 SHAP (SHapley Additive exPlanations) 來計算特徵的重要性，進而解釋模型的預測結果，而不需要訓練大量的模型。本篇的主角 SHAP 是一種解釋性技術，它就是使用 Shapley values 的概念來評估每個特徵對於模型預測的貢獻。SHAP 提供了以下幾種 kernel 來快速估計 Shapley values：

- TreeExplainer
- DeepExplainer
- GradientExplainer

- LinearExplainer
- KernelExplainer

每一個 kernel 的簡單介紹可以參考本系列[Day 4] LIME vs. SHAP：哪種XAI解釋方法更適合你？，而在今日的文章中我們將重點放在通用的 KernelExplainer，它主要透過生成新的擾動資料來近似計算 SHAP，且適用於各種不同類型的模型。

解析 KernelExplainer 背後原理

以下解釋採用 kernel shap 的方式來估計 Shapley values。假設第 i 筆資料 d 個特徵長像下面這樣子。我們要另外建立一個長度為 d 的 z 向量，這裡的 z 就好比每個特徵是否要被考慮因此不是1(出現)就是0(忽略)。

$$\text{若 } x^{(i)} = [1,2,3,4] \text{ 且 } z = [1,0,1,0]$$
$$\downarrow$$
$$d = 4$$

h(z) 就是將一組特徵透過 z 來決定要觀察哪些特徵，假設 x=[1,2,3,4]、z=[1,0,1,0] 就是要保留第一跟第三個特徵。因此 h(z)=[1，隨機，3，隨機]，所謂的忽略就是隨機的從訓練資料集中取一筆資料出來將第二和四個特徵放入取代隨機。因此 xk 就是隨機從訓練資料集抽取的一筆資料。

$$h(z) = \left[1, x_2^{(k)}, 3, x_4^{(k)}\right] \cdot \text{其中 } x^{(k)} \text{ 是一個隨機樣本}$$

所以 h(z) 其實就是一個 one-to-many 的映射，由於會隨機的抽取資料取代0位置的數值。如果我們用這種方式產生很多組 z 的話會產生 2 的d次方的可能性。

$$\text{E.g., } z = \underbrace{[0,0,0,0],[0,0,0,1],\ldots,[1,1,1,1]}_{2^d \text{ 個組合}}$$

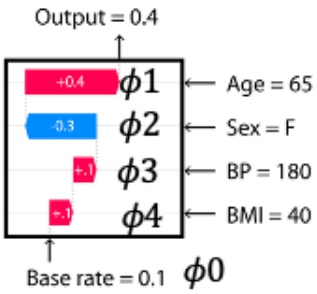
還記得我們在 Day4 解釋 SHAP 運作原理的例子嗎？假設有輸入四個特徵，年齡、性別、血壓、BMI作為輸入要預測一個人罹癌的機率。$\phi 0$ 就是基準點 base rate 這裡為 0.1，表示每個人都有 0.1 的機率罹癌。$\phi 1 \sim \phi 4$ 代表四個特徵對罹癌的貢獻值，全部相加起來就是那個人離癌的幾率了。因此 SHAP 就是要訓練一個模型 g() 要學習每個 ϕ。

- g(z): 為被簡化的可解釋的模型
- z: 表示每個特徵是否要被考慮，1(出現)、0(忽略)
- d: 輸入特徵的個數
- $\phi 0$: 代表基準值
- ϕj: 代表每個特徵的Shapley values

建立一個替代函數 g：

$$g(z) = \phi_0 + \sum_{j=1}^{d} \phi_j \, z_j$$

使得 $g(z) \approx f(h(z))$



g() 是 SHAP 自行定義的模型，其目標是希望 g(z) 能夠接近於 f(h(z))。換句話說，如果對於第 i 筆資料樣本 x=[1,2,3,4] 以及 z=[1,0,1,0]，只觀察第一和第三個特徵時，f(h(z)) 的預測結果將會趨近於 g(z) 也就是 $\phi0+\phi1+\phi3$。

舉例：

若 $x^{(i)} = [1,2,3,4]$ 且 $z = [1,0,1,0]$

$h(z) = \left[1, x_2^{(k)}, 3, x_4^{(k)}\right] = [1,6,3,8]$

訓練一個模型 $g\,()$ 要學習 ϕ

$$g(z) = g([1,0,1,0]) = \phi_0 + \phi_1 + \phi_3 \approx f\big(h(z)\big) = f([1, x_2^{(k)}, 3, x_4^{(k)}])$$

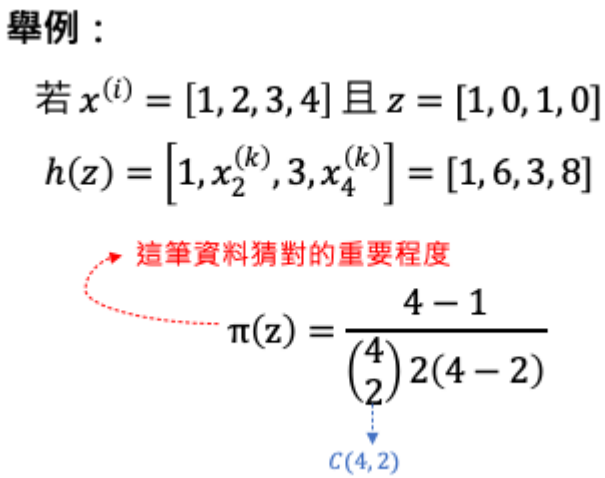
x_1的 shapely values　x_3的 shapely values

請各位回想一下之前在做 LIME 的時候需要從資料抽一些樣本要與被觀察的那筆資料計算距離，越近就代表越重要。因此 kernel shape 的方法中也是要給予權重的概念，但方法不同。差別在於它的 loss function 後面多了 π(z) 也就是所謂的權重，這個權重的決定取決於 z 有多少個1。

$$L(f, g, \pi_x) = \sum_{\forall z} \Big(f\big(h(z) - g(z)\big)\Big)^2 \pi(z) \cdot \text{其中}$$

$$\pi(z) = \frac{d-1}{\binom{d}{|z|}|z|(d-|z|)} \text{ (z 是非零元素的個數)}$$

延續剛剛的例子假設我們要觀察第一和第三個特徵時，損失函數中最後一項 π(z) 的計算方式直接套入上面式子。代表這筆資料猜對的重要程度，其中 c(4,2) 代表排列組合中的 C4 取 2。表示在有 4 個元素的集合中，選取 2 個元素的組合數量。



在 Kernel SHAP 中，對於特徵子集的抽樣是通過進行 Monto Carlo 抽樣的方式進行的。預設情況下，當特徵小於等於11個特徵時，SHAP 演算法會抽取 nsamples 數量為 `2**M - 2` 筆資料。而特徵數量大於11個時，會抽取 nsamples = `2*M + 2048` 個特徵子集作為近似計算的樣本，其中 M 是特徵的數量。會有這樣的機制是因為當特徵數量增加時，為了保持計算效率，Kernel SHAP 會進行抽樣，只包含部分特徵子集，但仍然能夠提供合理的解釋整個模型。

- 當M<=11個特徵以下： nsamples = 2**M-2
- 當M>11個以上： nsamples = 2*M + 2048

  詳細資訊可以直接參考 Kernel SHAP 的原始程式碼 *(https://github.com/shap/shap/blob/master/shap/explainers/_kernel.py#L311)*。

簡單來說 SHAP 的套件實現會根據特徵數量和計算成本採取適當的抽樣策略，以提供近似的 Shapley 值計算結果。對於小特徵集，它會包含所有可能的特徵組合，對於大型特徵集，則透過抽樣方式來有效估計 Shapley 值。

> 由於 Kernel SHAP 對模型類型沒有假設，它的速度比其他特定於模型類型的算法慢例如像是 Tree SHAP。

Reference

- Scott Lundberg, et al. "A Unified Approach to Interpreting Model Predictions *(https://arxiv.org/abs/1705.07874)*." Arxiv, 2017.
- Kjersti Aas, et al. "Explaining individual predictions when features are dependent: More accurate approximations to Shapley values *(https://arxiv.org/abs/1903.10464)*." Arxiv, 2019.

[Day 15] SHAP實作：實戰演練SHAP解釋方法

範例程式：  *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/15.SHAP實作：實戰演練SHAP解釋方法.ipynb)*

昨天已經瞭解了 SHAP 套件背後的核心技術。SHAP 提供多種解釋工具，可應用於不同類型的模型：

- KernelExplainer（Kernel SHAP）：適用於任何模型，它結合了 LIME 和 Shapley values 方法，透過估計 SHAP 值來提供解釋。
- TreeExplainer（Tree SHAP）：適用於 tree-based 與 ensemble 系列模型，包括 XGBoost、LightGBM、CatBoost、scikit-learn 和 pyspark 樹相關的模型，透過 Tree SHAP 算法計算 SHAP 值。
- DeepExplainer（Deep SHAP）：基於 SHAP 和 DeepLIFT 算法，專門用於計算深度學習模型的 SHAP 值，幫助解釋深度學習模型的預測。
- GradientExplainer：基於 SHAP 和 Integrated Gradients 算法，用於近似計算深度學習模型的 SHAP 值，其速度會比 DeepExplainer 慢。
- LinearExplainer：用於解釋線性模型的預測，適用於具有獨立特徵的線性模型。

SHAP 的優缺點

Shapley values 為我們提供了特徵在實例上的邊際貢獻。它適用於分類和迴歸任務，且可用於表格數據、文字和圖像。以下為各位統整 SHAP 的優缺點：

- SHAP 的優點：
    - 穩固的理論基礎：SHAP 繼承了博弈理論中 Shapley values 的理論基礎。
    - 效率提升：SHAP 確保模型預測值和平均預測值之間的差異在特徵之間公平分配，保證解釋的效率和合理性。
    - 通用性：SHAP 是一種通用的解釋工具，能夠應用於多種機器學習演算法和模型類型，不受模型選擇的限制。
    - 提供全局和局部解釋：SHAP 不僅能夠提供單一實例的解釋，還能夠計算全局的特徵重要性。
- SHAP 的缺點：
    - 計算時間與特徵數量成正比：當特徵的數量增多時，計算 SHAP 值所需的時間也會隨之增加。
    - 解釋複雜性：在解釋複雜模型時，SHAP 值可能無法完全捕捉模型的複雜內部關係，導致解釋的不完全性。

- 數據分佈敏感： SHAP 值的計算可能對數據分佈敏感，不同的數據分佈可能導致不同的解釋結果。

[實作] SHAP 解釋分類模型

本日範例一樣以一個糖尿病預測資料集訓練一個 SVM 分類器。接這透過 Kernel SHAP 對模型進行解釋。首先我們先載入今天範例的資料集，該資料集可以從 Kaggle 資料科學平台取得 *(https://www.kaggle.com/datasets/mathchi/diabetes-data-set)*。

載入資料集

```
import pandas as pd

# 讀取資料集
df_train = pd.read_csv('./diabetes.csv')
```

讓我們來瞧瞧 df_train 裡面的內容。我們可以發現該資料集有總共有 768 筆數據，每一筆資料有八個欄位資訊，其中包含模型的輸入與輸出。

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
2	183	64	0	0	23.3	0.672	32	1
3	89	66	23	94	28.1	0.167	21	0
4	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...
763	101	76	48	180	32.9	0.171	63	0
764	122	70	27	0	36.8	0.340	27	0
765	121	72	23	112	26.2	0.245	30	0
766	126	60	0	0	30.1	0.349	47	1
767	93	70	31	0	30.4	0.315	23	0

768 rows × 8 columns

這個資料集來自美國國家糖尿病和消化和腎臟疾病研究所。其目標是根據診斷測量來預測病人是否患有糖尿病。資料集的變數如下：

- Glucose：口服葡萄糖耐量測試中2小時的血漿葡萄糖濃度，用於測試糖尿病的診斷。
- BloodPressure：舒張壓(mm Hg)，血壓中的一個參數，用於衡量心臟在收縮時的壓力。
- SkinThickness：三頭肌皮膚褶皺厚度(mm)，用於衡量皮膚的脂肪層厚度。
- Insulin：2小時血清胰島素(mu U/ml)，用於評估胰島素水平，對糖尿病的診斷非常重要。
- BMI：身體質量指數，表示體重和身高的比例，用於評估體重狀況。
- DiabetesPedigreeFunction：糖尿病家族遺傳函數，用於衡量患有糖尿病的家族遺傳風險。
- Age：病人的年齡。

- Outcome：病人是否患有糖尿病(作為模型輸出)，值為0表示沒有糖尿病，值為1表示患有糖尿病。

切割資料集

接下來從剛剛讀取進來的 df_train 資料集中，將所有的輸入特徵資料提取出來，作為模型的輸入 X。同時，我們從 df_train 中取得 `Outcome` 欄位的資料，作為模型的輸出 y，用來表示病人是否患有糖尿病。除此之外，我們也將所有輸入特徵的欄位名稱儲存到 `x_feature_names` 變數中，`y_label_names` 則是儲存輸出的標籤名稱，這兩個個變數將在後續 SHAP 模型解釋的過程中使用。最後透過 `train_test_split` 方法切割訓練集與測試集。

```
from sklearn.model_selection import train_test_split

x_feature_names = df_train.drop(['Outcome'], axis=1).columns
y_label_names = ['No', 'Yes']
X = df_train.drop(['Outcome'], axis=1).values # 移除y並取得剩下欄位資料
y = df_train['Outcome'].values # 取得病人糖尿病結果作為y

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0
```

訓練模型 (SVM 分類器)

我們將使用 SVM 建立一個分類模型，並使用訓練資料（X_train, y_train）進行訓練。由於 SHAP 解釋分類器需要模型預測每個類別的機率，因此在訓練 SVM 分類器時，需要添加參數 `probability=True`，以確保模型在推論時能夠輸出預測機率。

```
from sklearn import svm

# 建立 kernel='linear' 模型
model=svm.SVC(kernel='linear', C=1, probability=True)
# 使用訓練資料訓練模型
model.fit(X_train, y_train)
```

SHAP 解釋模型

接著我們要使用 SHAP 套件來進行模型的解釋。首先大家可以在終端機輸入以下指令安裝 SHAP 套件：

```
pip install shap
```

首先載入 SHAP 函式庫並初始化 JavaScript 環境，以便在 Jupyter Notebook 環境中能夠顯示 SHAP 的解釋圖表和視覺化結果。

```
import shap

shap.initjs()
```

首先建立一個通用的 KernelExplainer 解釋器，並嘗試剖析剛剛所訓練的 SVM 分類模型。以下是常用的設定參數與說明：

Parameters: - model：待解釋的模型。支援 sklearn 所封裝的模型，迴歸器可以使用 `model.predict`，分類器可以使用 `model.predict_proba`。 - data：可採樣的資料集，用於產生隨機擾動抽樣的子集，此資料用於訓練 SHAP 解釋模型。 - link：將SHAP簡單模型的預測輸出轉換為實際預測值的函數，提供兩種設定分別為 `identity` 和 `logit`。預設為 identity。

以下程式將已經訓練好的模型引入，並透過呼叫 predict_proba 方法來計算預測機率。在 data 參數的部分我們從訓練集中取出前 50 筆資料，用以代表整體特徵值的分布。在 SHAP 中的 KernelExplainer 方法中，link 參數用於指定預測模型的連結函數（link function）。連結函數是將線性模型的輸出轉換為實際的預測值的函數。在不同的情況下，使用不同的連結函數可以更好地適應模型的性質。此範例是分類模型，因此可以選擇 `logit`，即表示每個計算出來的 Shapley values 再通過 sigmoid 函數即代表預測機率。

```
# 使用 Kernel SHAP 解釋模型
explainer = shap.KernelExplainer(model=model.predict_proba, data=X_tr
```

> KernelExplainer API 官方文檔可以從這裡參考 *(https://shap-lrjball.readthedocs.io/en/latest/generated/shap.KernelExplainer.html)*。

接著我們要使用 shap_values() 方法來估計 Shapley values 並對單筆資料進行解釋。以下是常用的設定參數與說明：

Parameters: - X: 欲被解釋的資料。 - nsamples: 用於構建解釋每個預測的代理模型的樣本數量。

我們把之前先切割出來的測試集作為要被解釋的目標 X。接著設定 nsamples 為 100，這意味著我們將進行 100 次蒙地卡羅抽樣，從 KernelExplainer 設定的 data 中隨機擾動抽樣並建立一個 SHAP 簡單可解釋模型。對於每個隨機採樣的樣本，我們需要進行隨機擾動並進行模型推論（獲取 f() 的預測 y）。因此總共需要進行 `100*50` 次模型推論，以生成這 100 筆資料。然而 SHAP 官方建議盡量 data 的數量不要超過 100 筆數據，以避免過高的計算成本。原始碼可以從這裡參考 *(https://github.com/shap/shap/blob/1ccbf672399d3467e4e4433894ed00e4f067e258/shap/explainers/_kernel.py#L104)*。

```
shap_values = explainer.shap_values(X=X_test, nsamples=100)
```

1. 全局解釋模型

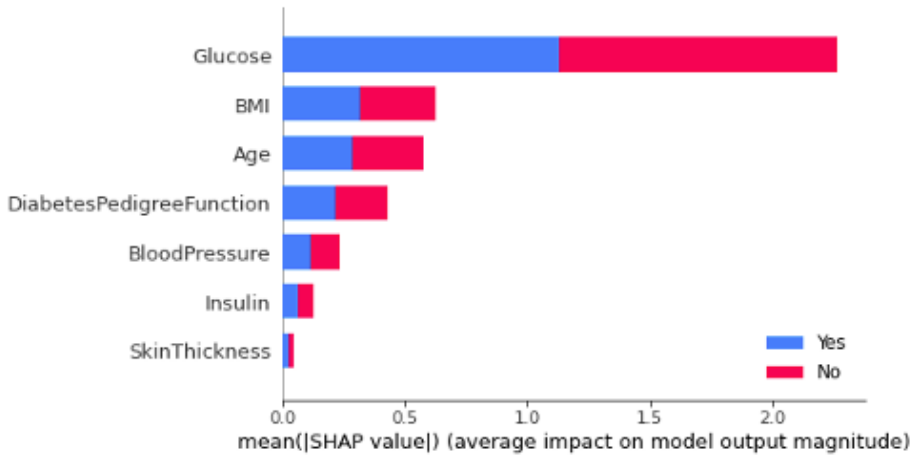
1.1 SHAP Summary Plot

SHAP Summary Plot 可以幫助了解模型的特徵重要性，幫助解釋模型的預測。如果某個特徵的 SHAP 值較大且穩定，則可以認為該特徵對模型預測的影響較大且較一致。反之，如果特徵的 SHAP 值較小且不穩定，則可能認為該特徵對模型預測的影響較小或不一致。

- 點的顏色: Feature value 的大小，越紅越大、越藍越小
- X 軸: 該點對於 shap value 的影響，也就是對預測值的影響
- Y 軸: 每個特徵

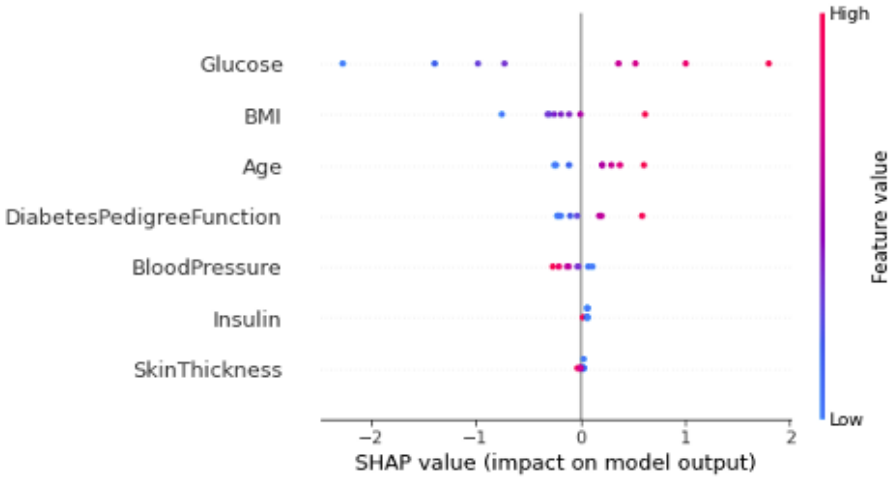
我們可以使用 `plot_type` 參數設置 `bar` 畫一張條形圖，不同顏色代表不同類別(以下範例藍色代表預測Yes的重要程度，紅色為No)，每個條形代表一個特徵，並顯示該特徵對模型預測的影響程度。在這個圖表中，每個特徵對於各個類別的影響被堆疊起來，以創建整體的特徵重要性圖。

```
shap.summary_plot(shap_values, X_test, plot_type="bar", class_names=
```



我們也可以觀察特定類別的 summary_plot。假設我想看模型對於預測 Yes 的重要程度，可以使用 `shap_values[1]` 的資料取得每筆測試集的 Shapley values 進行全局的解釋。以下圖來說我們可以得知模型在判斷是否罹患糖尿病情況下大多會看葡萄糖濃度(Glucose)，當數值越大越有機會罹患糖尿病。第二個重要的特徵為BMI，同樣也是當BMI越大越有機會罹患糖尿病。

```
shap.summary_plot(shap_values[1], X_test, feature_names=x_feature_nam
```

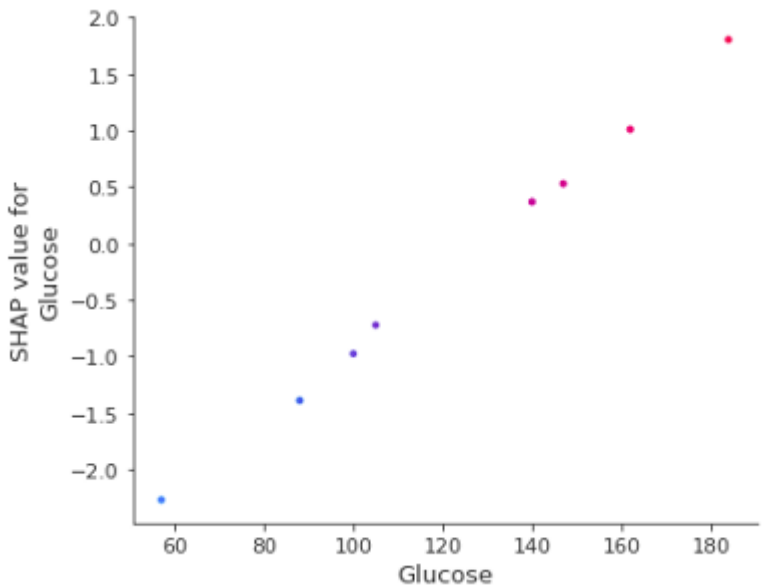


1.2 SHAP Dependence Plot

SHAP 相依圖是一種散點圖，顯示了單一特徵對模型所做預測的影響。在這個例子中，當每位葡萄糖濃度越高相對應的 Shapley values 逐漸增加。

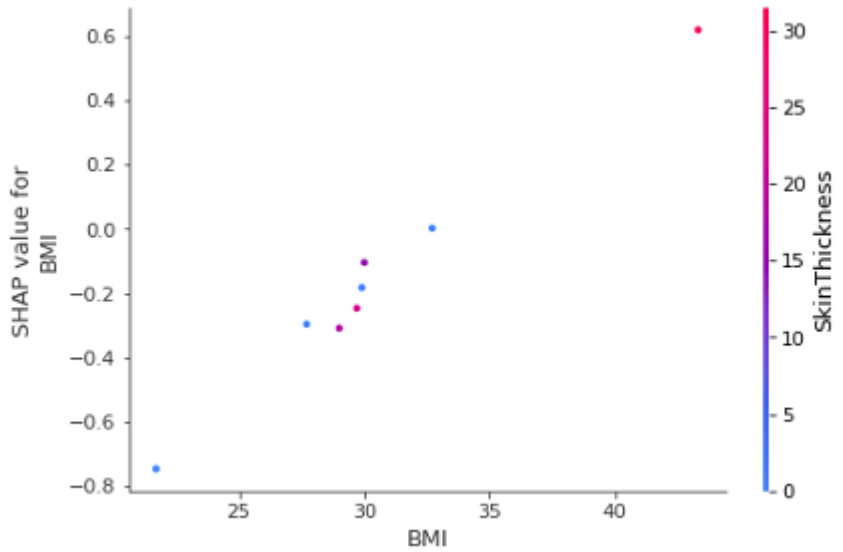
- 每個點代表資料集中的一筆預測資料
- X 軸：該特徵的實際值
- Y 軸：該特徵的 SHAP 值，表示知道該特徵的值有多大程度上改變了該樣本預測的模型輸出。

```
shap.dependence_plot('Glucose', shap_values[1], X_test, feature_names
```



SHAP 相依圖類似於部分相依圖，但考慮了特徵之間的交互作用，我們也可以觀察兩個變數的交互作用影響，可以在 `interaction_index` 參數中設定第二個特徵名稱，顏色對應到第二個特徵數值高低的影響。以下範例觀察BMI與皮脂厚度交互作用下對於 BMI 的 SHAP 值影響。

```
shap.dependence_plot('BMI', shap_values[1], X_test, feature_names=x_f
```



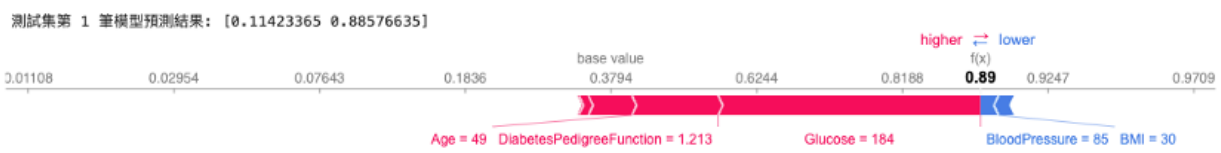
2. 局部解釋模型

2.1 SHAP Force plot

我們可以觀察單一筆資料在模型中的預測情況。在 SHAP 套件中，「Force Plot」方法提供了針對單一模型預測的解釋性呈現。在這個圖表中，我們可以清楚地看到各特徵對模型對特定輸入值的預測所做的貢獻。這種方法在進行錯誤分析或深入理解特定情況下的資料時非常有幫助。

從以下圖表我們可以觀察： 1. 模型在測試集中的第一筆資料預測NO的機率有0.11，Yes的機率有0.89 2. base value: 代表模型在不看任何特徵狀況下預測的數值，在這個例子中，基準值 = 0.379。註：此基準值是有經過 sigmoid 函數。 3. 每個特徵後面的數字是該筆資料的特徵值，例如 Age 49 歲。 4. 紅色代表該特徵會增加判斷Yes的機率。而藍色代表該特徵會降低罹患糖尿病的機率。 5. 箭頭的寬度表示該特徵對輸出的影響越大。 6. Glucose、DiabetesPedigreeFunction、Age 這三個特徵明顯是判斷罹患糖尿病的重要因子。

```
# 觀察測試集中第一筆資料預測為Yes的重要程度
index=0
print(f'測試集第 {index+1} 筆模型預測結果: {model.predict_proba(X_test[[ind
shap.force_plot(explainer.expected_value[1], shap_values[1][index], X
```

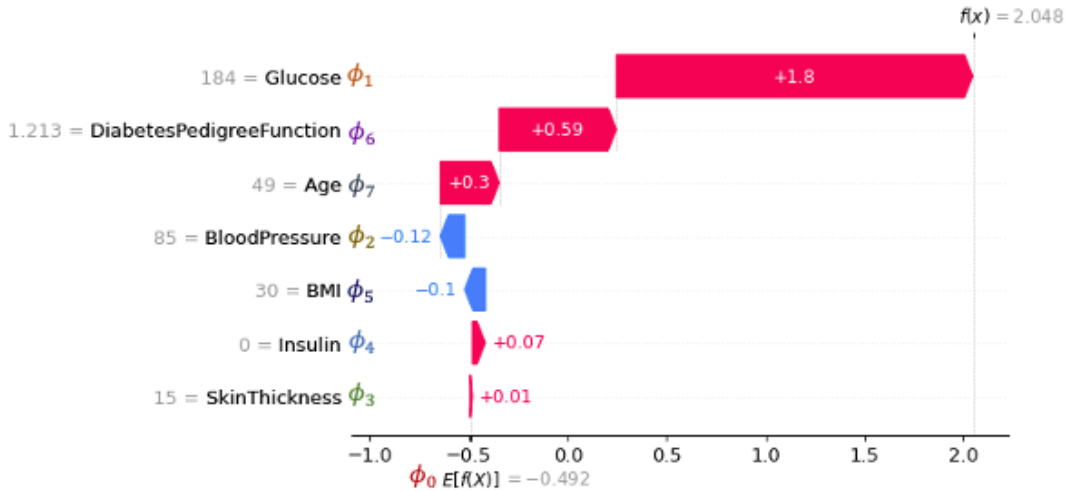


2.2 SHAP waterfall plot

瀑布圖能夠以視覺方式呈現單一預測的解釋結果。因此在前面的結果基礎上，我們對測試集中的第一筆資料進行了單一預測解釋。瀑布圖的起點是模型輸出的基準值，接著每一條紀錄了每個特徵對於輸出模型預測值的正向（紅色）或負向（藍色）影響。累加起來最後再通過 Sigmoid 函數就是否罹患糖尿病的機率值了。

```
index=0
shap.waterfall_plot(shap.Explanation(values=shap_values[1][index],
                                     base_values=explainer.expected_va
                                     feature_names=x_feature_names))
```

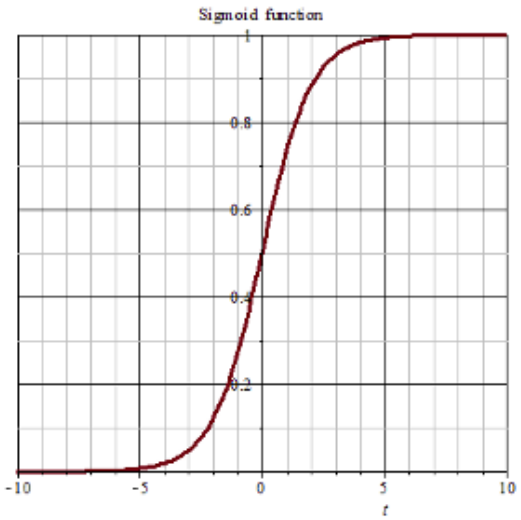
從以下圖表我們可以觀察： 1. f(x) 是模型預測的對數勝算(log odds)：2.408。 2. E[f(x)] 是基準值 = 0.492。 3. 左側是特徵名稱旁邊的灰色數值是代表該筆資料的輸入值。 4. 在箭頭上的數值代表每個特徵的Shapley values貢獻值ϕ。 5. 由於選擇 "logit" 連結函數，因此x軸的單位是log odds。 6. 正值意代表該人罹患糖尿病的概率大於 0.5。 7. 負值意代表該人罹患糖尿病的概率小於 0.5。



$$\phi_0 + \phi_1 + \phi_2 + \phi_3 + \phi_4 + \phi_5 + \phi_6 + \phi_7 = Prediction\_local$$

$$-0.492+1.801-0.123+0.01+0.066-0.105+0.592+0.299 = 2.048$$

將線性模型的輸出通過一個稱為 S 形曲線（sigmoid 函數）的轉換，將線性模型的輸出映射到0和1之間，這樣預測結果就變成了二元類別。因此我們可以理解最後的輸出就相當於，模型根據輸入的特徵判斷結果是Yes的機率有多高。



$$\frac{1}{1+exp(-2.048)} = 0.89$$

[補充] link 參數設定時機

在 SHAP 的 KernelExplainer 方法中，參數 link 可以選擇 `identity` 或 `logit` 作為 link function(連結函數)。以下是這兩種連結函數的區別以及它們的適用時機：

"identity" 連結函數： 使用 identity 連結函數意味著模型的輸出直接被當作預測值，不進行額外的轉換。這在迴歸任務中較為常見，當你希望預測值與特徵之間的關係是線性的時候，可以選擇這個連結函數。

"logit" 連結函數： 使用 logit 連結函數則將線性輸出轉換為對數比值（log odds），這在二元分類問題中比較常見。對數比值表示某個事件發生的對數機率與不發生的對數機率之比。"logit" 連結函數適用於那些希望模型預測機率時，同時考慮到機率值的變化幅度較大的情況，例如在邏輯迴歸模型中。

選擇連結函數時，我們應該考慮模型的任務類型以及預測值和特徵之間的關係。如果你的模型是迴歸模型，並且你認為預測值與特徵之間的關係是線性的，那麼使用 "identity" 連結函數可能更適合。如果你的模型是分類模型，並且你希望考慮機率的變化，那麼 "logit" 連結函數可能更合適。

Reference

- shap.KernelExplainer *(https://snyk.io/advisor/python/shap/functions/shap.KernelExplainer)*
- GitGub/shap *(https://github.com/shap/shap)*
- Pros and Cons of CheatSheet-XAI *(https://github.com/goodrahstar/cheatsheet-xai/blob/master/explainable_methods.jpg)*
- Tensorflow-預訓練ResNet50可解釋性分析 *(https://blog.csdn.net/m0_59286668/article/details/128426336)*

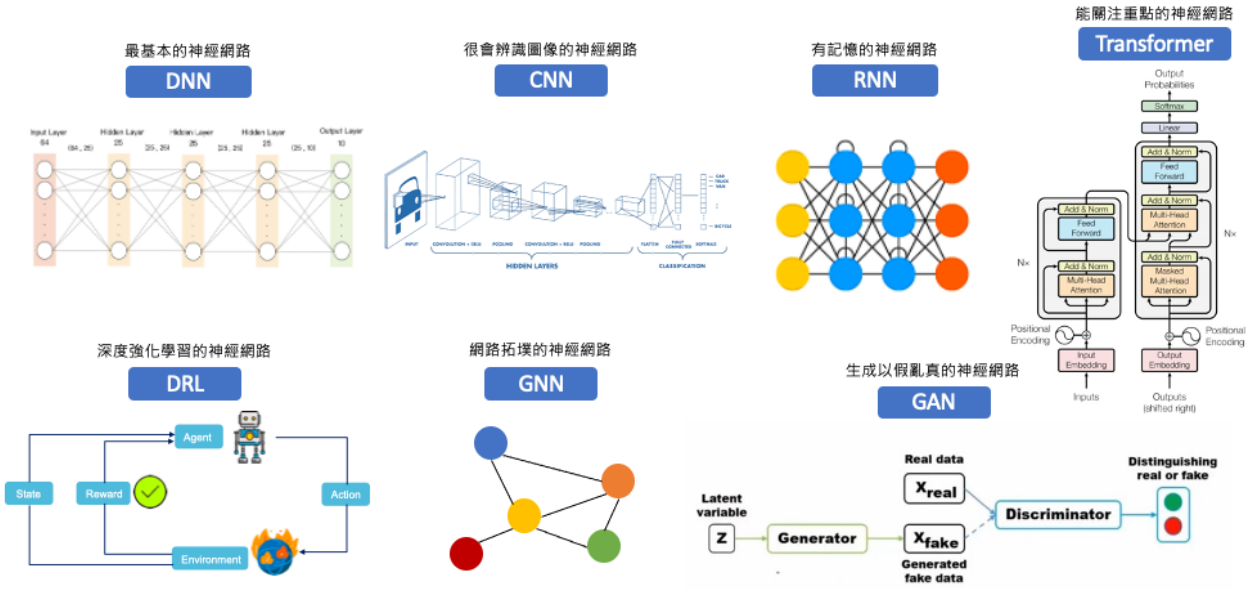
- SHAP Part 2: Kernel SHAP *(https://medium.com/analytics-vidhya/shap-part-2-kernel-shap-3c11e7a971b1)*
- Explainable AI (XAI) with SHAP -Multi-Class Classification Problem *(https://towardsdatascience.com/explainable-ai-xai-with-shap-multi-class-classification-problem-64dd30f97cea)*

4.XAI在深度學習中的可解釋性

[Day 16] 神經網路的可解釋性：如何理解深度學習中的黑箱模型？

深度神經網路（DNN）以其線性和非線性的複雜轉換而聞名，因為它涵蓋了許多隱藏層。因此即使給定一個訓練有素並能夠良好分類的 DNN，模型內部的推論過程仍然是個未知，這使得 DNN 也被稱為黑盒模型。從今天開始我們將進入深度學習的世界，接下來的內容當中會講解一些熱門的神經網路背後是如何被解釋的。

神經網路的種類



- DNN (Deep Neural Network)：深度神經網路是由多個層次的神經元組成，透過層與層之間的權重全連接進行訊息傳遞。DNN 是一種通用的機器學習模型，用於處理結構化和非結構化數據，如圖像、語音、文字、表格資料等。
- CNN (Convolutional Neural Network)：卷積神經網路主要用於處理圖像和視覺任務。它使用卷積層和池化層來自動擷取和學習圖像中的特徵，從而實現對圖像的高效處理和分類。
- RNN (Recurrent Neural Network)：循環神經網路是一種時間記憶性的深度神經網路，它具有循環的特性，可以處理序列數據，例如語音和文字。
- Transformer：它使用自注意機制來將輸入序列映射到輸出序列。此模型在機器翻譯、文本生成和語言理解等任務上取得了重大突破。並且廣泛應用到其他領域，例如音訊、圖像等任務。
- DRL (Deep Reinforcement Learning)：深度強化學習結合了神經網路和強化學習，用於教導機器學習從環境中學習最佳行動。DRL 在遊戲、自主控制和優化等領域中都有出色的表現。

- GNN (Graph Neural Network)：圖神經網路用於處理圖結構化數據，如社交網絡、分子結構和知識圖譜。它能夠捕捉節點之間的關係，並在圖中進行訊息傳遞和學習。
- GAN (Generative Adversarial Network)：生成對抗網路是一種特殊的結構，由生成器和判別器組成，用於生成逼真的數據樣本。生成器和判別器相互競爭，使生成器能夠逐漸生成越來越逼真的數據，如圖像和音訊。

神經網路的可解釋性

我們可以運用多種方法來解釋深度神經網路的運作，這些方法能夠幫助我們更深入地理解神經網路的運行方式，並解釋模型進行推論的過程。透過從以下幾個角度出發，使我們能夠更深入地探索黑箱模型的內部：

1. 特徵重要性

我們可以從特徵貢獻性評估每個輸入特徵對預測的重要程度，其中先前所介紹的可解釋工具 LIME 和 SHAP 也能拿來解釋各種類型的神經網路，例如： DNN、CNN、RNN、LSTM、GRU 等。

- LIME（Local Interpretable Model-Agnostic Explanations）：使用局部模型來近似解釋單筆模型的預測。
- SHAP（SHapley Additive exPlanations）：基於博弈論的方法，計算每個特徵對於預測的貢獻。

  延伸閱讀：[Day 12] LIME理論：如何用局部線性近似解釋黑箱模型 *(https://ithelp.ithome.com.tw/articles/10327698)*

  延伸閱讀：[Day 14] SHAP理論：解析SHAP解釋方法的核心 *(https://ithelp.ithome.com.tw/articles/10329606)*

2. 視覺化

我們可以將神經網路的輸出結果視覺化，並觀察神經網路在推論過程中，哪些特徵影響最大。此概念最常被應用在卷積神經網路（CNN）上，因為 CNN 通常會產生許多特徵圖，我們可以將這些特徵圖視覺化，觀察 CNN 對輸入圖像的解釋。另外近期熱門的大型語言模型都是以 Transformer 為架構搭建出來的神經網路，其中內部核心就是由多個 self-attention 所組成。因此在深度學習中的注意力機制（Self-Attention）也與視覺化結合，我們可以將注意力機制視覺化，觀察模型在推論過程中，哪些特徵影響最大。

- 熱圖分析: 將特定層的激發函數值視覺化並繪製熱力圖(Activation heatmap)，顯示模型在每個區域的重要性。例如使用 Grad-CAM 分析模型預測的梯度來視覺化影響模型決策的圖像區域，觀察模型在預測中關注的重要區域。
- 特徵視覺化: 可視化輸入圖像中特定特徵的影響，例如繪製 maxpooling 後的結果。觀察每層所有的特徵，並觀察其對輸入圖像的影響。
- 注意力機制： 我們可以將特徵視覺化與注意力機制結合得到 Attention map。並觀察模型在推論過程中，對輸入圖像、自然語言或訊號進行重要性程度解釋。

3. 對抗性解釋

對抗性樣本是一種修改後的輸入，可以干擾模型的預測。通過分析模型對對抗性樣本的響應，可以觀察模型的弱點和預測結果的不確定性。

- 生成對抗網路（GANs）：使用GANs生成對抗樣本，觀察模型對不同輸入的響應解釋其行為。

明天開始我們就依序的探索深度神經網路的解釋方法。

Reference

- The Transformer Blueprint: A Holistic Guide to the Transformer Neural Network Architecture *(https://deeprevision.github.io/posts/001-transformer/?fbclid=IwAR3TDtqY0ysRL6NCQunhfGoBCPz8UGNvbBbKZcxRUdxFWF7j6gCRcHDs10M)*

[Day 17] 解析深度神經網路：使用Deep SHAP進行模型解釋

範例程式：  *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/17.解析深度神經網路：使用Deep SHAP進行模型解釋.ipynb)*

Feature Attribution

Feature Attribution(特徵歸因)是機器學習領域中的一個重要概念，它用於解釋模型的預測結果。當我們訓練機器學習模型，特別是深度學習模型，這些模型通常是黑盒子，難以理解為什麼模型會做出特定的預測。因此特徵歸因的目標是找出模型中每個輸入特徵對於最終預測的貢獻或影響程度，這有助於我們理解模型的運作原理，並檢查模型是否依據我們的期望運作，以及發現可能的偏差或不公平性。在機器學習中，特徵歸因通常有以下幾種常見的方法：

- 特徵重要性（Feature Importance）：這種方法衡量了每個特徵對於模型預測的影響程度，可以透過 LIME 或 SHAP 等解釋工具進行特徵重要性分析。
    - SHAP（SHapley Additive exPlanations）：SHAP 基於博弈論的概念，它試圖將每個特徵的貢獻分配給模型預測的結果。Shapley values綜合考慮了特徵的重要性以及特徵之間的交互作用，因此在解釋複雜模型時很有用。
    - LIME（Local Interpretable Model-agnostic Explanations）：LIME 通常用於解釋某筆資料為何做出這樣判斷，它主要生成一個簡單可解釋的線性模型，來解釋模型的預測。
- 基於梯度方法（Gradient-Based Methods）：這些方法基於模型的梯度訊息，試圖找出哪些特徵對於某個預測的梯度貢獻最大。
- 遮罩或屏蔽方法（Masking or Perturbation Methods）：這些方法通過對輸入特徵進行修改，觀察模型預測的變化，來評估每個特徵的重要性。例如可以將某個特徵的值設置為零，然後觀察預測的變化。

Additive Feature Attribution Methods

Additive Feature Attribution(AFA) 方法的核心思想是將一個模型的預測分解成每個特徵的貢獻部分，這樣可以更容易理解模型的決策過程。該方法旨在理解每個特徵對模型預測的貢獻，它們假設模型對於輸入特徵的預測是可分解的，因此模型的預測可以被分解成每個特徵的影響，並且這些影響是可以相加的。一些解釋機器學習模型預測的方法，例如 LIME、SHAP、DeepLIFT 和 Layer-Wise Relevance Propagation，這些方法都屬於 AFA 的家族其中一員。AFA 方法的基本定義如下：

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i$$

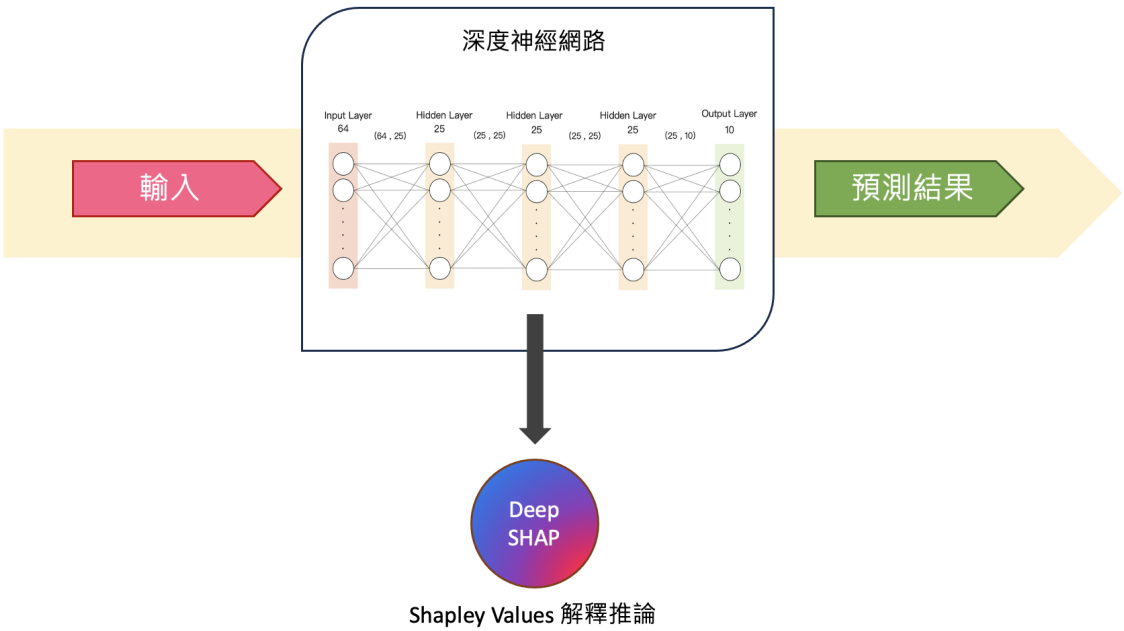
其中 $z' \in \{0,1\}^M$ ， M表示簡化的輸入特徵數量 ， $\emptyset_i \in \mathbb{R}$ 。

- g是一個簡單的可解釋模型，它可能是一個用於解釋複雜模型的模型，通常是一個線性模型或類似的簡單模型。
- 式子中的每個ϕi代表第i個特徵的影響程度或重要性。當一個特徵的ϕi值較大時，意味著這個特徵對模型的預測有較大的影響。
- z'是一個由0和1組成的二元向量，其中M代表簡化特徵的數量。這個向量表示了在模型中哪些特徵被考慮到，哪些沒有被考慮到。當特徵在向量中對應的位置是1時，表示這個特徵在模型中被考慮到了；當對應位置是0時，表示該特徵未被考慮。

簡單來說，AFA 方法的目標是通過一個簡單的可解釋模型 g 來解釋複雜模型的預測。它使用二元向量 z' 來表示哪些特徵在解釋中被考慮，並使用 ϕi 作為權重來量化每個特徵對預測的影響程度。這樣的方法有助於理解模型是如何根據不同的特徵來做出預測的，提高了模型的可解釋性。不同的 AFA 方法可能使用不同的技術來計算 ϕi 值，但它們都遵循這個基本框架進行模型解釋。

Deep SHAP (DeepLIFT + Shapley Values)

今天要介紹在 SHAP 套件中的 Deep SHAP 方法，它結合了 DeepLIFT 和 Shapley values 的概念，以計算每個特徵對於模型預測的貢獻，使更好地解釋神經網路模型的預測。



- DeepLIFT *(https://arxiv.org/abs/1704.02685)* (Deep Learning Important FeaTures)：是基於反向傳播的解釋方法，透過比較模型的預測輸出與參考輸出之間的差異來計算每個輸入特徵對預測的貢獻。使得我們可以了解每個特徵對模型預測的相對影響。

- Shapley Values：用於評估每個特徵對模型預測的影響。它考慮了每個特徵的不同排列組合，以確定每個特徵的貢獻度。使得我們可以更好地理解模型預測背後的特徵重要性。

  今天的練習將使用 SHAP 套件中的 DeepExplainer(Deep SHAP) 方法作為展示。

[實作] 使用 Deep SHAP 解釋 DNN 模型

本日範例將透過 TensorFlow 實作 DNN 模型，並使用 sklearn 的資料集 fetch_california_housing *(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html)* 來預測加州地區的房屋價格中位數。這個資料集包含了 8 個特徵，分別是：

- MedInc：該區域內家庭收入的中位數
- HouseAge：該區域內房屋的平均房齡
- AveRooms：該區域內房屋的平均房間數
- AveBedrms：該區域內房屋的平均臥室數
- Population：該區域內人口數
- AveOccup：該區域內平均每個房屋的居住人數
- Latitude：該區域內房屋所在緯度
- Longitude：該區域內房屋所在經度

這個資料集包含了 20640 筆樣本，每個樣本都有上述 8 個特徵以及房屋價格中位數作為目標變數。

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
import numpy as np

# 載入加州地區房屋價格預測資料集
data = fetch_california_housing()
x_feature_names = np.array(data.feature_names)
X, y = data.data, data.target

# 切分資料集為訓練集和測試集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
```

建立與訓練神經網路

以下程式碼使用 Tensorflow2.0 Functional API 搭建神經網路。此模型接受一個輸入，然後通過一系列神經網路層進行處運算，最後輸出一個單一的數值即為房屋價格中位數。模型的層次結構包括：一個正規化層（Normalization）用於對輸入進行正規化，三個全連接層（Dense）用於提取特徵和學習模型的映射，最後一個全連接層輸出單一值，並使用 ReLU 激發函數達到非線性轉換。

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Model

def build_model():
    # 資料正規化
    model_input = layers.Input(shape=X.shape[-1])
    norm_layer = tf.keras.layers.Normalization(axis=1)
    norm_layer.adapt(X_train)
    x = norm_layer(model_input)
    # 第一層隱藏層
    x = layers.Dense(32,activation='relu')(x)
    # 第二層隱藏層
    x = layers.Dense(64,activation='relu')(x)
    # 輸出層
    model_output = layers.Dense(1,activation='relu')(x)
    return Model(model_input ,model_output)
```

接下來，使用之前定義的 `build_model()` 函數建立一個新的神經網絡模型，並將這個模型存儲在 model 變數中。最後使用 `model.summary()` 印出模型的摘要訊息，包括模型的結構、每一層的參數數量等。

```
tf.keras.backend.clear_session()
model = build_model()
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 8)]               0

 normalization (Normalizatio  (None, 8)                17
 n)

 dense (Dense)               (None, 32)                288

 dense_1 (Dense)             (None, 64)                2112

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 2,482
Trainable params: 2,465
Non-trainable params: 17
_________________________________________________________________
```

模型準備就緒後即可開始訓練模型。這裡使用 Adam 優化器設定學習率為 0.001，並使用均方誤差（MSE）作為損失函數。接下來設定批次大小為 64，訓練迭代次數為 50 次。最後使用訓練數據 X_train 和 y_train 來訓練模型。

```
from tensorflow.keras.optimizers import Adam

# 編譯模型
optim = Adam(learning_rate=0.001)
model.compile(loss='mse',
              optimizer=optim)

batch_size=64
epochs = 50

# 訓練模型
history = model.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    shuffle=True,
                    validation_split=0.1)
```

Deep SHAP 解釋模型

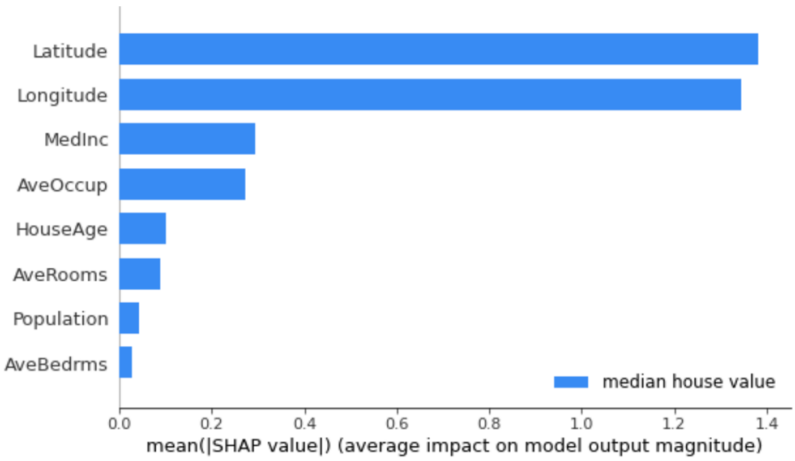
以下建立一個 DeepExplainer 解釋器，並指定了要解釋的模型（model）和訓練數據（X_train）。然後透過 Deep SHAP 來估計 Shapley values，並將其存儲在 shap_values 變數中。

```
import shap
shap.initjs()

# 使用 Deep SHAP 解釋模型
explainer = shap.DeepExplainer(model=model, data=X_train)
# 估計 Shapley values
shap_values = explainer.shap_values(X_test)
```

SHAP Summary Plot (全局解釋)

我們可以透過 SHAP Summary Plot 進行模型的全局解釋，該圖表顯示每個特徵變量對整體平均模型輸出的平均影響。在該圖表中，我們可以看到每個特徵對於模型的預測輸出的平均貢獻程度，有助於理解哪些特徵對模型的預測起著重要作用，哪些特徵影響較小。從分析結果可以發現地理位置(經緯度)以及家庭收入和成員數對於預測該地區房價是有顯著的影響性。

```
# 獲得每個特徵對於整體平均貢獻的值
shap.summary_plot(shap_values[0], X_test, class_names=['median house
```

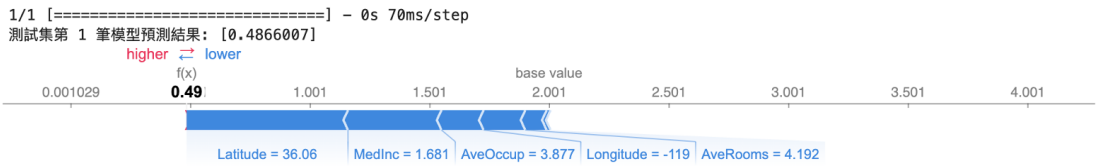


SHAP Force plot (單筆資料解釋)

由於我們從資料集切割 21 筆作為測試集，剛剛上面的全局解釋是針對這 21 筆資料進行平均整體性解釋。接著我們一樣可以針對每一筆數據進行解釋分析。首先程式中的 index 被設定為0，表示我們要觀察測試集中的第一筆資料。接著使用 `force_plot` 對這筆資料進行預測，並將分析結果視覺化呈現。

```
# 觀察測試集中第一筆資料預測重要程度
index=0
print(f'測試集第 {index+1} 筆模型預測結果: {model.predict(X_test[[index], :
shap.force_plot(explainer.expected_value.numpy(),
                shap_values[0][0][index],
                X_test[index],
                feature_names=x_feature_names)
```

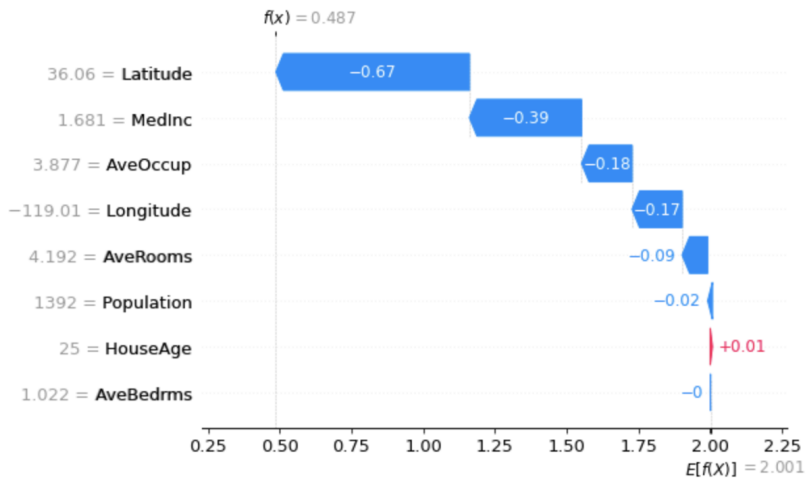
可以試著調整 index 數值(0~20)觀察測試集中不同資料點的解釋



SHAP waterfall plot (單筆資料解釋)

瀑布圖是一種能夠以視覺方式呈現單一預測解釋結果的工具。瀑布圖的起點是模型輸出的基準值　E[f(z)]，代表模型在不看任何特徵狀況下預測的數值（$\phi 0$）。然後每一個條都記錄了每個特徵對於輸出模型預測值的正向（紅色）或負向（藍色）影響。全部累加起來得到輸出值（即所有特徵貢獻ϕi和基準值$\phi 0$的總和），即等同於實際模型的預測。

```
index=0
shap.waterfall_plot(shap.Explanation(values=shap_values[0][0][index],
                                     base_values=explainer.expected_va
                                     feature_names=x_feature_names))
```



$$\phi_0 + \phi_1 + \phi_2 + \phi_3 + \phi_4 + \phi_5 + \phi_6 + \phi_7 + \phi_8 = Prediction\_local$$

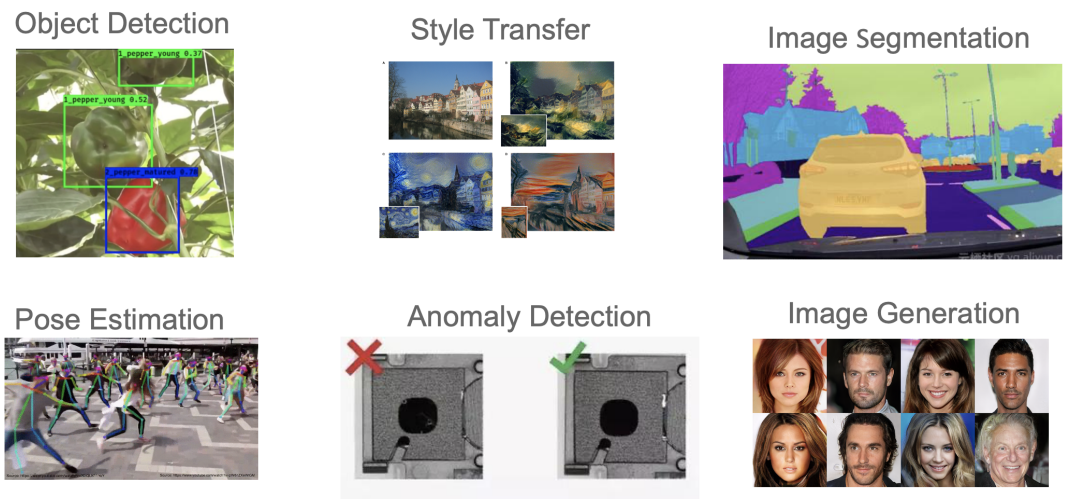
2.001-0.391+0.008-0.089-0.001-0.016-0.176-0.674-0.174 = 0.487

Reference

- Avanti Shrikumar, et al. "Learning Important Features Through Propagating Activation Differences *(https://arxiv.org/abs/1704.02685)*." Arxiv, 2017.
- shap.DeepExplainer *(https://shap-lrjball.readthedocs.io/en/latest/generated/shap.DeepExplainer.html#shap-deepexplainer)*
- Interpretability of Deep Learning Models *(https://medium.com/towards-data-science/interpretability-of-deep-learning-models-9f52e54d72ab)*
- Additive Feature Attribution Methods *(https://medium.com/@jimmywu0621/%E5%8F%AF%E8%A7%A3%E9%87%8B%EF%BD%81%EF%BD%89-%E4%BB%80%E9%BA%BC%E6%98%AFshap-5ec3953e3c5b)*
- Deep SHAP 鑽石範例 *(https://towardsdatascience.com/interpretability-of-deep-learning-models-9f52e54d72ab)*
- 翻譯Deep SHAP 鑽石範例 *(https://kknews.cc/zh-tw/code/n9lyk23.html)*

[Day 18] CNN：卷積深度神經網路的解釋方法

在當今的深度學習領域中，卷積神經網路（CNN）已經成為許多電腦視覺任務的首選模型，例如圖像分類、物體偵測、語意分割、動作偵測等，這些重大突破都歸功於卷積神經網路的引入。然而儘管它們的強大性能，CNN 模型同樣被視為「黑盒」，難以理解其內部的決策過程。這種缺乏可解釋性可能限制了我們對模型的信任度，特別是在需要高度可解釋性和透明性的應用場景中，如醫學診斷和自動駕駛系統，這些應用更需要借助 XAI 技術來解釋模型的決策過程。



CNN 的優點

CNN 的優勢在於其能夠在不同的層次上學習特徵。較淺的卷積層具有較小的感知域，因此它們能夠捕捉到較局部的特徵，例如影像中的細微紋理或區域性特徵。而較深的卷積層則具有更大的感知域，這使得它們能夠學習到更加抽象的特徵，例如物體的整體形狀、結構或高層次的抽象概念。這種多層次的特徵學習使得CNN能夠在處理各種視覺任務時表現出色。CNN 具有以下幾點特性：

- Local connectivity (結合影像特性)
- Weight sharing (共用學習同一組參數)
- Subsampling (池化運算)

1. Local connectivity (結合影像特性)

CNN 的第一個優點在於它能夠結合影像特性，並採用區域相似性以及平滑變化的特性於模型架構中。這意味著 CNN 能夠捕捉到影像中局部區域的特徵，並將它們結合起來，以識別物體、模式或紋理等。

2. Weight sharing (共用學習同一組參數)

第二個優點涉及到 CNN 的參數共享機制。在 CNN 中，同一組過濾器（filter）可以在整個影像上滑動以搜索相同的模式或特徵。這種共享權重的方式大大減少了需要訓練的參數數量，使得模型更加緊湊且計算效率更高。

3. Subsampling (池化運算)

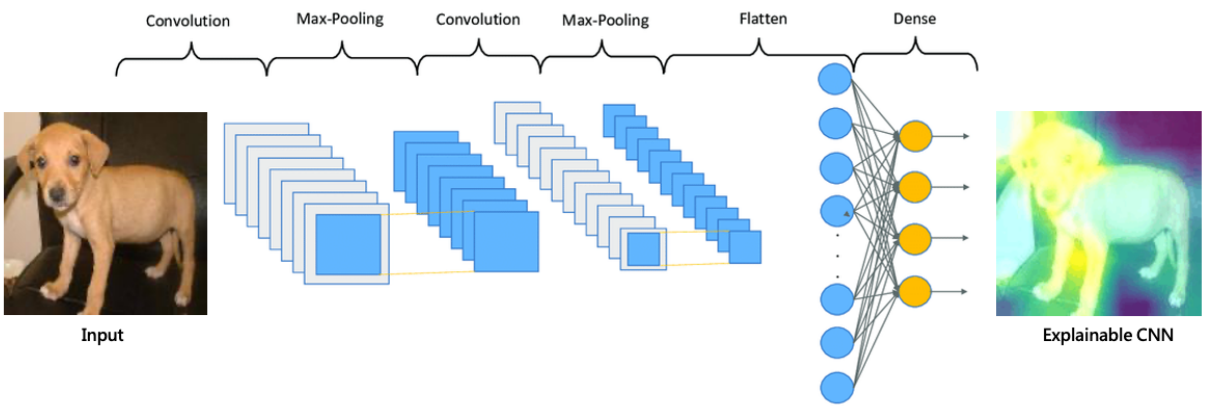
第三個優點是它使用的高效降維方法。透過 subsampling（例如池化層），CNN 可以將高解析度的影像轉換為較小的版本，同時保留了關鍵訊息。這樣的操作減少了需要訓練的神經網路參數數量，同時不會影響模型的性能。

> 我們都知道CNN具有上述幾個特點，但要如何藉由這些優點來解釋內部的運作就是Explainable CNN的課題！

Explainable CNN

Explainable CNN（可解釋卷積神經網路）就是試圖理解 CNN 是如何做出判斷的。在圖像分類和物件辨識任務中，Explainable CNN 的目標是探索 CNN 模型對圖像的關注點是否符合人類的直覺判斷，並透過解釋性技術幫助改進模型的偏見和數據偏見，進而訓練出更優秀的模型。藉由解釋性技術，Explainable CNN 試圖找出影響模型最終判斷的關鍵部分，即模型在圖像中關注的區域或特徵。人類可以通過這些關鍵部分的樣式來解釋模型的決策，進而幫助做出更明智的決策。

假設我們訓練了一個 CNN 模型來辨識圖像中的動物，並且模型對於狗的判斷表現非常準確。我們希望了解模型是如何進行這些判斷的。透過 Explainable CNN 我們可以找到模型對於判斷狗的關鍵區域，例如耳朵、眼睛和鼻子等部位。這樣的解釋可以讓我們更好地理解模型的決策過程，並檢視模型是否真正學到了辨識狗的正確特徵。



這樣的解釋性技術有助於我們檢視模型的偏見，查看是否過度關注無關要緊的特徵或區域，以及數據的偏見，例如是否對某些類別的圖像有較大的偏好。透過理解模型的偏見以及決策方式，我們可以有針對性地調整模型的訓練過程，以獲得更公正和可靠的結果。

Explainable CNN 技術

可解釋卷積神經網路技術提供了多種方法和技巧來解釋卷積神經網路模型的決策過程和結果。以下是幾種常見的 Explainable CNN 技術：

- Perturbation-based Explanation（擾動解釋）： 透過對輸入圖像進行微小的擾動來評估對模型預測結果的影響。它可以揭示模型對不同圖像區域的敏感性和重要性，並解釋模型的判斷過程。
- Gradient-based Explanation（梯度解釋）：計算輸入特徵對於模型輸出的梯度，來評估特徵的重要性。
- Propagation-based Explanation（傳播解釋）：基於反向傳播算法，分析模型中每個神經元對於輸出的貢獻。可以理解模型中不同神經元對於最終的決策有何影響。
- CAM-based Explanation（Class Activation Mapping 解釋）：此技術解釋模型對於不同類別的關注區域。CAM 會生成一張熱度圖，顯示模型對於特定類別的關注程度，幫助我們理解模型在圖像中關注的重要區域。
- Attention-Based Explanation（基於注意力的解釋）：這種方法關注於模型的注意力機制，即模型在處理輸入時關注哪些部分。它可以幫助我們理解模型如何對不同特徵或區域進行注意，以進行預測。

Reference

- Machine learning interpretability with feature attribution *(https://cgarbin.github.io/machine-learning-interpretability-feature-attribution/#feature-attributions-are-approximations)*

[Day 19] Perturbation-Based：如何用擾動方法解釋神經網路

範例程式：  *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/19.Perturbation-Based：如何用擾動方法解釋神經網路.ipynb)*

如果想要了解一張圖片中哪些區域對於 CNN 神經網路的判斷結果具有影響力，可以參考基於擾動的 Perturbation-Based 方法。它有很多不同種的變形，其中最著名的是基於遮蔽擾動的方法，可以參考2014年於 Springer 發表的期刊論文：Visualizing and understanding convolutional networks *(https://arxiv.org/abs/1311.2901)* 。在這篇論文中，提出反卷積(Deconvolution)的作者 Zeiler 透過遮蔽圖片的一部分來觀察模型的輸出，以確定圖片中哪些區域對於模型的分類是相對重要的。實際上，作者也運用遮蔽反卷積和反池化的方法，驗證了這種技術確實能夠提供 CNN 的可解釋性。

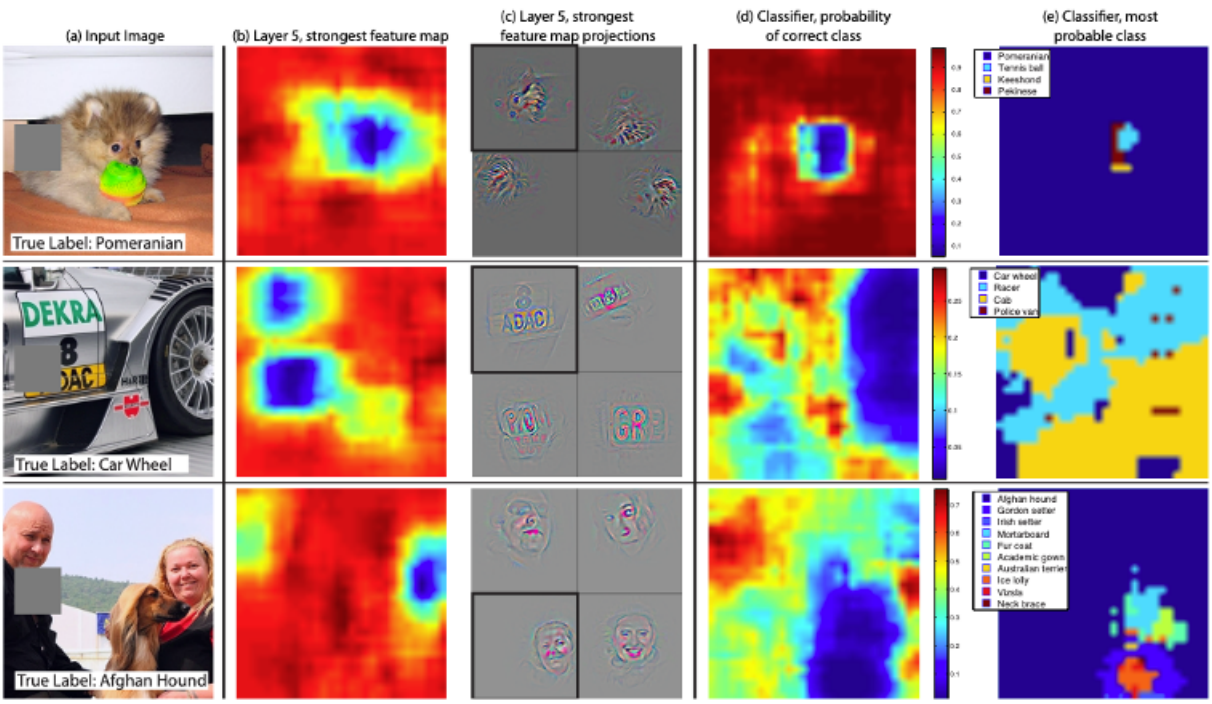
> 參考論文：Visualizing and understanding convolutional networks *(https://arxiv.org/abs/1311.2901)*

Perturbation-Based 方法不需要動到神經網路的架構，僅需要變動輸入的圖片，並從中觀察模型輸出的統計分佈。並試圖理解在哪個位置被遮擋後，會對最終的預測結果造成影響。因此造成最大影響的地方就可以被判定成圖片中最重要的關鍵區域，因為只要 CNN 無法看見重要區域就無法萃取關鍵特徵，相對的模型就無法正確的判斷。

Occlusion Sensitivity (遮擋敏感度)

這篇論文提出了一種名為「遮擋敏感度（Occlusion Sensitivity）」的方法，它透過在圖片的特定部分進行遮擋，以觀察網路中間層的情況和預測值的變化。這有助於我們更好地理解為何網路會做出某些決策。簡單來說，遮擋敏感度是指當我們遮擋圖像的特定部分時，觀察預測機率如何隨之變化，進而找出圖片中的重要區域。

下圖為論文中的實驗結果。該實驗使用三個測試範例，分別進行系統性的遮擋，然後觀察神經網路的反應以及結果的變化。在這個實驗中，首先對每個範例圖片的不同區域應用了一個灰色方塊進行遮擋並預測，然後觀察了神經網路在第五層特徵圖的活化程度(b)，以及將特徵圖的訊息視覺化，並投影回原始輸入圖像，然後將其顯示出來(c)。(d)根據灰色方塊的位置，顯示該類別的機率分佈(可以從中發現藍色區塊代表越重要)。例如第一張圖當遮擋了狗的臉時，屬於 `pomeranian`（博美犬)這個類別的機率會明顯下降，因為神經網路無法看到狗的臉。最後(e)這一部分顯示了在不同遮擋位置時，最有可能的類別標籤。例如在第一張圖中，大部分情況下最可能的標籤是 `pomeranian`，但如果遮擋了狗的臉而沒有遮擋到球，則它可能預測為 `tennis ball` (網球)。



Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.

從上圖(d)可發現如果欲辨識的目標物體被遮蔽的話那分類的準確度就會大大降低。

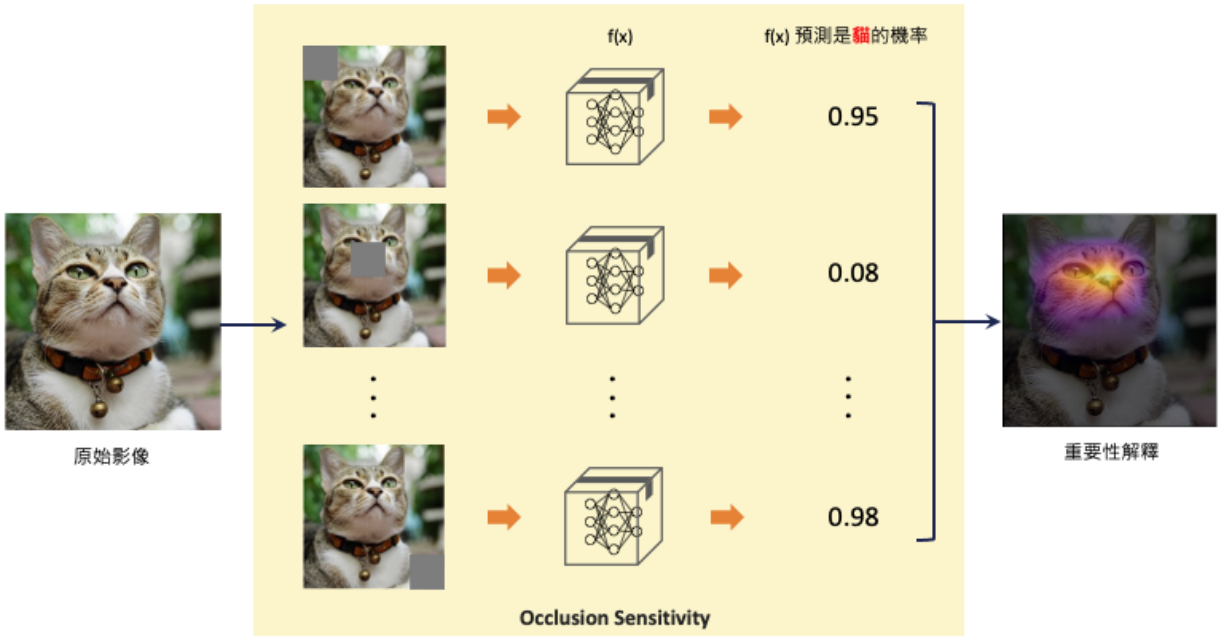
Occlusion Sensitivity 的解釋過程

這種基於遮擋擾動的方法實作非常簡單。首先我們需要訓練一個效果良好的分類器，接著選取一張要解釋的圖片，並對該圖片的不同區域進行遮擋，同時監測模型的輸出機率。最後我們可以將（1-機率值）視為被遮擋區域的重要性程度。上述過程可以簡要概括為以下三個步驟：

1. 訓練分類器：訓練一個性能良好的分類器。
2. 遮擋圖像區域：選擇要解釋的圖片。對圖片的不同區域進行遮擋，形成不同的遮擋版本。
3. 監測輸出機率：觀察模型在每個遮擋版本下的輸出的預測機率。並使用(1-機率值)來評估被遮擋區域的重要性程度。

   也可以使用(原始無遮擋影像的機率-遮擋後預測的機率)作為重要性的評估。

這裡的遮擋我們稱之為 patch(灰色小區塊)，而每個 patch 大小可以事先設定好。假設一張大小 224*224 的影像，每個遮擋 patch 大小設定為 56*56，總共會產生 16 張不同位置遮擋的版本(可以參考文章一開始的的動圖)。我們可以從結果發現經過遮擋貓的臉部該類別的機率大幅度的下降，也可以間接證實模型真的有學到預測貓要看臉部五官。

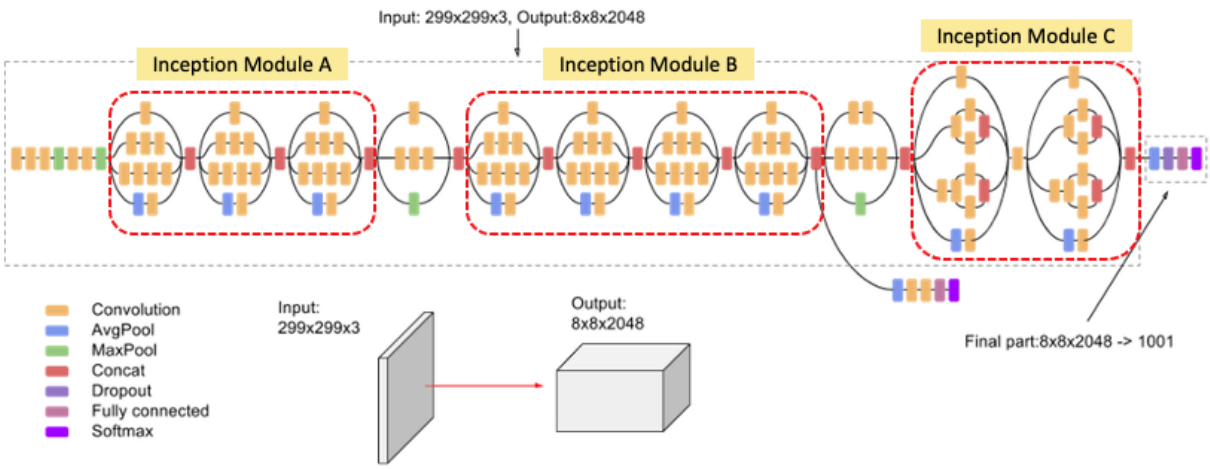


這種方法的優點在於容易實施，但缺點在於需要大量的計算資源。因此為了提高計算效率，每個 patch 不會重疊。此外 patch 的大小雖然可以隨意指定，但若重要的特徵範圍較大的話，使用小的 patch 可能會導致最後的解釋效果不佳。這強調了在選擇 patch 大小時需要考慮實際物體的大小，以確保解釋的準確性和有效性。

另外還記得 Day1 *(https://ithelp.ithome.com.tw/articles/10318087)* 我在文中所描述的辨識貓狗的例子嗎？我們也可以試著遮擋鈴鐺的部分，觀察模型是不是有學錯辨識貓的關鍵特徵。從下圖結果，即使把鈴鐺遮起來還是成功辨識出貓了。終於不是一個鈴鐺分類器了（汗。

Perturbation-Based 方法實作 (Occlusion Sensitivity)

接下來我們使用 Google 的 Inception V3 預訓練模型來預測一張圖片，並試著使用遮擋擾動的技巧解釋模型。Inception V3 以其獨特的網路結構而聞名，它採用了所謂的`Inception Module`，這是一種多分支卷積結構，可以在不同尺度和方向上捕捉圖像特徵。這種結構使得網路能夠更有效地處理各種複雜的圖像，並提高了圖像分類的性能。



Inception V3 論文：Rethinking the Inception Architecture for Computer Vision (CVPR 2016) *(https://arxiv.org/abs/1512.00567)*

載入預訓練模型(Inception V3)

首先使用 TensorFlow 載入 Inception V3 模型，將輸入張量(tensor)連接到預訓練的神經網路層，`imagenet` 表示使用在 ImageNet 資料集上預訓練的權重。`include_top=True` 表示輸出包括模型的最後分類層(全連接層)，此模型通常用於影像分類任務。

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.layers import Input
```

```
# 建立一個輸入張量，指定圖像大小為224x224（RGB色彩通道）
input_tensor = Input(shape=(224, 224, 3))
# 建立 InceptionV3 模型
model = InceptionV3(input_tensor=input_tensor, weights='imagenet', in
```

接著載入一張圖像，對其進行預處理。其中 `np.expand_dims()` 的目的是將圖像轉換為模型可接受的維度，這裡將圖像包裝在一個批次(batch)中，通常是一個批次只有一張圖像。最後使用 Inception V3 模型的預處理函數 `preprocess_input()` 來處理圖像，以確保圖像的數值範圍和格式符合模型的要求。

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.inception_v3 import preprocess_inp

# 載入圖像
image = tf.keras.utils.load_img('./dataset/cat_dog.jpg')
image = tf.keras.utils.img_to_array(image) # 將載入的圖像轉換為數組形式
x = np.expand_dims(image.copy(), axis=0) # 將圖像轉換為模型可接受的維度
# 預處理圖像
x = preprocess_input(x)
```

確認輸入影像都完成處理過後，就可以使用已建立的 Inception V3 模型進行圖像分類預測，返回分類機率。最後再使用 `decode_predictions()` 解析取得預測結果，並取得類別名稱和相對應的預測機率。`pred_class_idx` 則是預測的標籤索引，最後模型解釋會需要用到它。

```
from tensorflow.keras.applications.inception_v3 import decode_predict

# 進行圖像分類預測
pred_proba = model.predict(x) # 返回分類機率
# 解析預測結果
pred_class_idx = pred_proba.argmax(axis=1)[0]  # 找到具有最高機率的類別索引
pred_class = decode_predictions(pred_proba, top=1)[0][0]  # 解析取得預測
```

我們先來看看模型預測的結果。雖然這張影像同時有一隻貓和狗，但模型在神經網路中先抓取到狗的重要特徵(例如：鼻子、嘴巴)，因此最終模型預測 `bull_mastiff`(鬥牛獒)，該類別的機率值有 98% 這麼高。

```
import matplotlib.pylab as plt

plt.imshow(image.astype('uint8'))
plt.axis('off')
predicted_class_name = pred_class[1]
_ = plt.title(f"Prediction: {predicted_class_name} {pred_class[2]:.2f
```

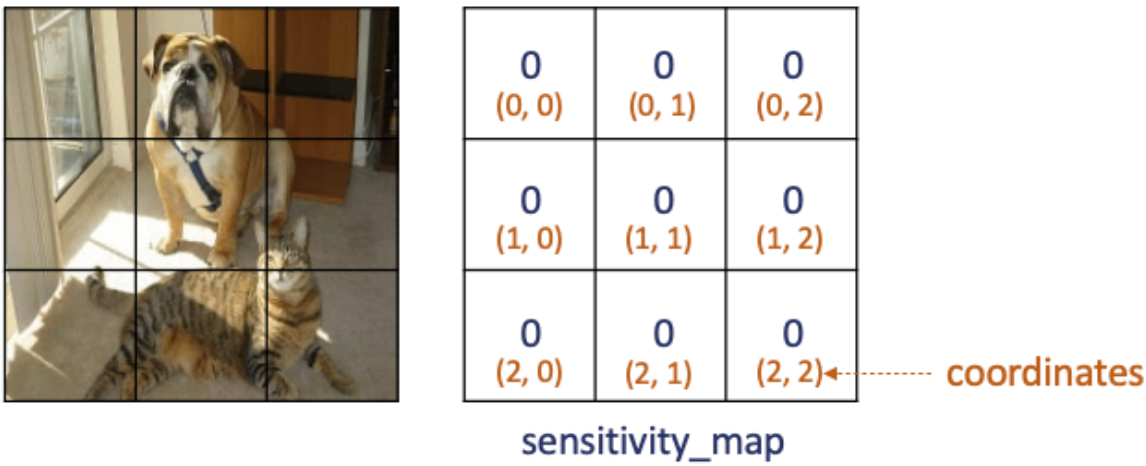
Occlusion Sensitivity 實作

以下實現 Occlusion Sensitivity 方法，它對原始影像依序插入灰色方塊(patch)，然後遍歷這些方塊並計算它們對於模型預測的影響，最後生成 sensitivity_map 和 coordinates 以供進一步分析使用。由於影像大小為224*224，而每個 patch 大小 75*75 因此總共會生成 9 張圖並儲存在 patches。

```
import math

batch_size = 16 # 批次大小，設定模型一次預測可以讀取幾張照片
patch_size = 75 # 方形灰色方塊大小
target_class_idx = pred_class_idx # 預測目標的標籤索引

# 定義一個函數，用於將灰色方塊(patch)置換到原始影像的指定位置
def apply_grey_patch(image, top_left_x, top_left_y, patch_size):
    patched_image = np.array(image, copy=True)
    # 置換指定區域的像素值為灰色（127.5），達到遮蔽的效果
    patched_image[
        top_left_y : top_left_y + patch_size, top_left_x : top_left_x
    ] = 127.5

    return patched_image

# 用於記錄不同區域對於模型預測的影響(初始化為0)
sensitivity_map = np.zeros(
    (
        math.ceil(image.shape[0] / patch_size),
        math.ceil(image.shape[1] / patch_size),
    )
)
# 儲存所有遮擋的圖像
patches = [
```

```
    apply_grey_patch(image, top_left_x, top_left_y, patch_size)
    for index_x, top_left_x in enumerate(range(0, image.shape[0], pat
    for index_y, top_left_y in enumerate(range(0, image.shape[1], pat
]
# 建立一個坐標列表，用於記錄不同區域的坐標
coordinates = [
    (index_y, index_x)
    for index_x in range(
        sensitivity_map.shape[1]
    )
    for index_y in range(
        sensitivity_map.shape[0]
    )
]
```

上面程式為整個流程的前置作業，先把遮擋的圖像與原圖合成，並將九張不同遮擋位置的圖片儲存在 patches，接著初始化 sensitivity_map 為0，以及生成一個座標網格 coordinates 以利於後續實作。



patch_size 的大小會影響解釋的結果，各位可以嘗試變動大小並觀察。

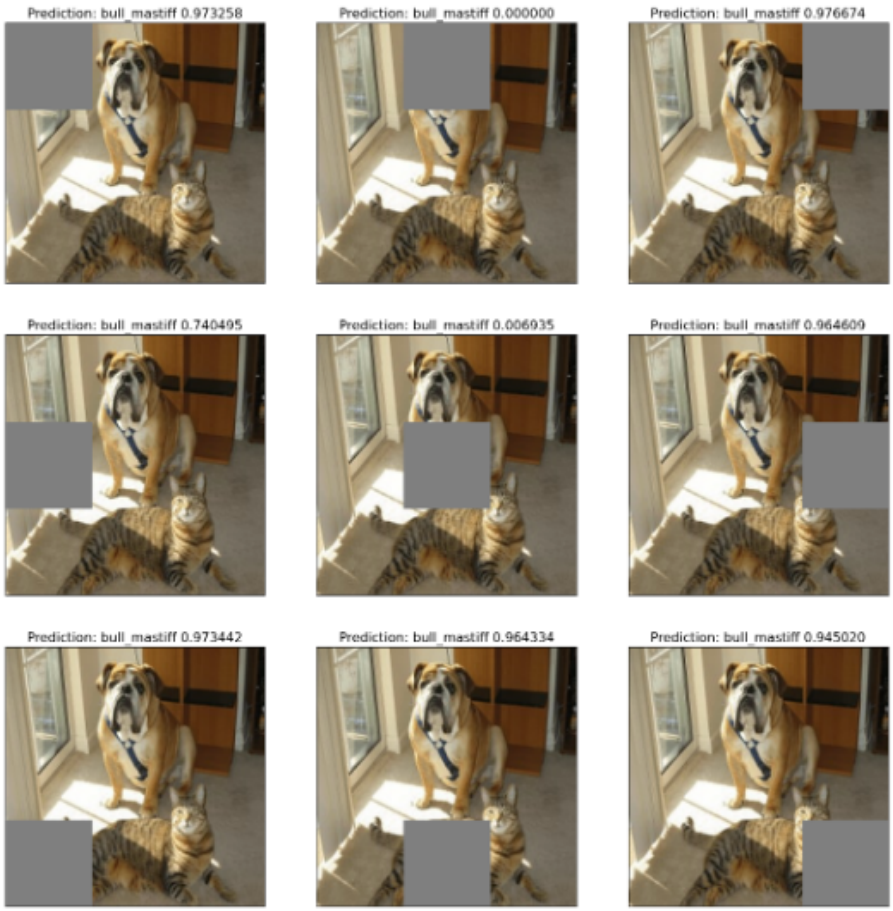
遮擋圖片都已準備好後，接著觀察每個方塊的不同位置遮擋對於模型預測的影響。我們將影像進行前處理，接著餵入先前已建立好的 Inception V3 模型，然後提取特定類別的預測機率值，在本範例中我們要觀察 `bull_mastiff` 類別的機率，最終這些機率值存儲在 target_class_probs 中供後續使用。

```
# 建立模型輸入資料
inputs = np.array(patches, copy=True)
# 預處理圖像
inputs = preprocess_input(inputs)
# 進行圖像分類預測
pred_proba = model.predict(inputs, batch_size=batch_size)
# 取得 bull_mastiff 類別的機率
```

```
target_class_probs = [
    prob[target_class_idx] for prob in pred_proba
]
```

我們將剛剛計算出來的結果透過視覺化方式呈現。可以清楚的觀察每一張遮擋版本的圖像，以及相對應的類別機率為多少。

```
# 計算網格大小，繪圖排版用
grid = math.ceil(image.shape[0]/patch_size)
# 建立一張圖包含多個子圖，並設定圖像的大小和排列方式
fig, ax = plt.subplots(nrows=grid, ncols=grid, figsize=(20, 20),
                       subplot_kw={'xticks':[], 'yticks':[]},
                       gridspec_kw=dict(hspace=0.2, wspace=0.1))
# 顯示每張遮擋版本的圖像
for i in range(len(patches)):
    # 設定每個子圖像的標題，包括預測的類別名稱和相應的機率值
    ax[i%grid, i//grid].set_title(f'Prediction: {predicted_class_name
    # 顯示子圖像，並將像素值除以255以將其正規化到0到1之間
    ax[i%grid, i//grid].imshow(patches[i]/255)
```

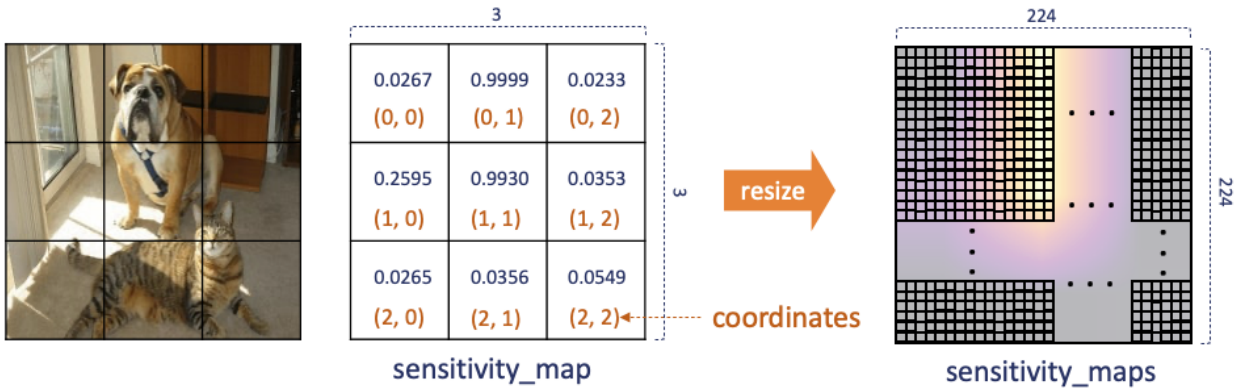


上一步驟已經觀察模型在每個遮擋版本下的輸出的預測機率。接下來計算每個子圖像的重要性（敏感程度）。我們使用1減去模型預測機率值，當成是那塊 patch 的重要性，這樣使得模型輸

出較高的機率值將對應較低的重要性，反之亦然。因此相減之後的值越高，表示該遮擋灰色的區域越重要。最後將 sensitivity_map 的大小調整為與原始圖像相同的大小，每個像素將對應到原始圖像的相應位置，使得它可以與原始圖像一起使用或顯示。

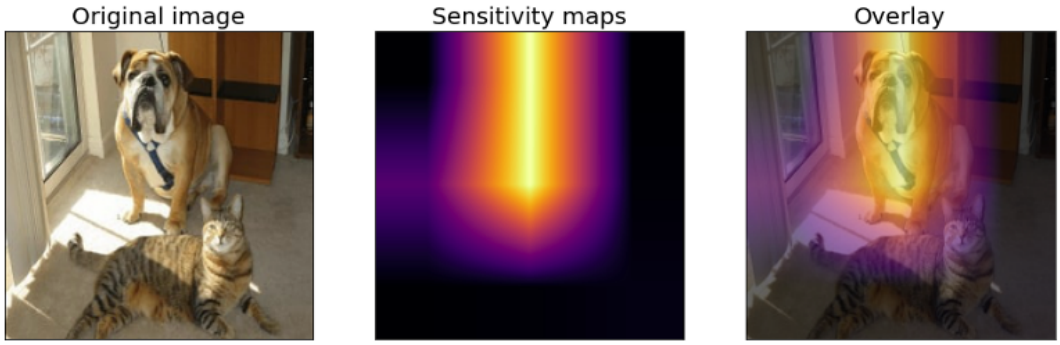
```
import cv2
import matplotlib.cm as cm

# 將 (1-機率值) 當成是那塊 patch 的重要性
for (index_y, index_x), confidence in zip(coordinates, target_class_p
    sensitivity_map[index_y, index_x] = 1 - confidence
# 調整為與原始圖像相同的大小
sensitivity_maps = cv2.resize(sensitivity_map, tuple(image.shape[0:2]
```

> 重要性計算有很多種方法，也可以使用(原始無遮擋影像的機率-遮擋後預測的機率)。



每個區塊重要性都已經計算出來了，最後我們就 matplotlib 來視覺化結果吧！從結果可以發現，當我們遮擋狗的臉部會大幅度降低預判 `bull_mastiff` 類別的機率值。因此相對的可以得知辨識狗的關鍵特徵在於臉部特徵與四肢。

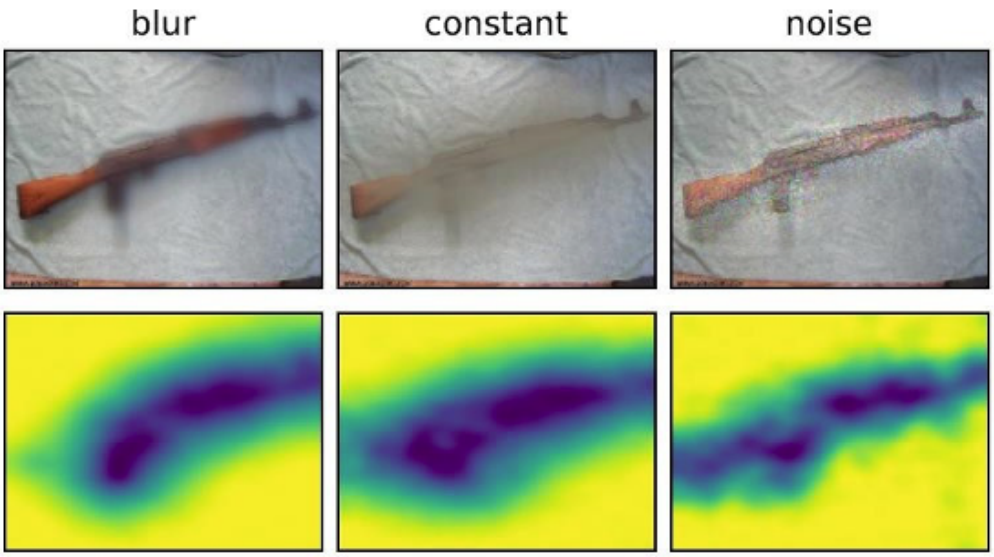
```
fig, axs = plt.subplots(nrows=1, ncols=3, squeeze=False, figsize=(16,
                        subplot_kw={'xticks':[], 'yticks':[]})

axs[0, 0].set_title('Original image', fontsize=20)
axs[0, 0].imshow(image/255)

axs[0, 1].set_title('Sensitivity maps', fontsize=20)
axs[0, 1].imshow(sensitivity_maps, cmap=cm.inferno)

axs[0, 2].set_title('Overlay', fontsize=20)
axs[0, 2].imshow(sensitivity_maps, cmap=cm.inferno)
axs[0, 2].imshow(image/255, alpha=0.4)
```



最後我們再進行一個有趣的實驗，如果我們遮住狗的臉，然後再次使用 InceptionV3 模型進行預測，你猜猜會得到什麼輸出？果然如預期，模型注意力轉向貓的特徵，最終輸出了 `tabby`（虎斑貓）。

小結

Perturbation-Based 還有許多不同的方法，例如我們可以自定義目標函數以及圖片遮擋的技巧(模糊化、數值替換、加入雜訊)。並針對一個模型每個輸入去尋找遮蔽哪裡可以獲得最佳的解釋性。也就是遮蔽了哪些地方，對模型模型分類的結果會造成最大的負影響。有興趣的人也可以參考這篇論文：Interpretable Explanations of Black Boxes by Meaningful Perturbation *(https://arxiv.org/abs/1704.03296)*。



Fong, R. C., & Vedaldi, A. (2017). Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE international conference on computer vision* (pp. 3429-3437).

Reference

- [論文速速讀]Visualizing and Understanding Convolutional Networks *(https://meetonfriday.com/posts/3013fdb9/#Experiments)*
- Google Cloud: Advanced Guide to Inception v3 *(https://cloud.google.com/tpu/docs/inception-v3-advanced)*
- Visualization of Convolutional Neural Networks in PyTorch *(https://datahacker.rs/028-visualization-and-understanding-of-convolutional-neural-networks-in-pytorch/)*
- 可解釋 AI (XAI) 系列 — 01 基於遮擋的方法 (Perturbation-Based): Occlusion Sensitivity, Meaningful Perturbation *(https://medium.com/ai-academy-taiwan/%E5%8F%AF%E8%A7%A3%E9%87%8B-ai-xai-%E7%B3%BB%E5%88%97-01-%E5%9F%BA%E6%96%BC%E9%81%AE%E6%93%8B%E7%9A%84%E6%96%B9%E6%B3%95-perturbation-based-40899ba7e903)*

[Day 20] Gradient-Based：利用梯度訊息解釋神經網路

範例程式：  *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/20.Gradient-Based：利用梯度訊息解釋神經網路.ipynb)*

在深度學習中梯度訊息扮演著關鍵的角色，尤其在神經網路的訓練過程中。梯度代表了函數相對於參數的變化率，它告訴我們如何調整神經網路的權重和參數，以最小化或最大化一個特定的損失函數。因此梯度下降法是優化神經網路的主要手段之一，它依賴梯度訊息來更新模型的參數，使其更好地適應訓練數據。

然而今天要談的是利用梯度訊息解釋神經網路，也就是當訓練一個神經網路後，透過計算神經網路對於輸入影像的梯度，我們就可以了解每個輸入特徵（例如像素）對於神經網路輸出的影響程度，並解釋神經網路的決策過程。以下是一些相關文獻，可以使用梯度訊息來表示輸入影像中每個特徵的重要度：

- Image-Specific Class Saliency *(https://arxiv.org/abs/1312.6034)*(Simonyan et al., 2014)：該方法計算指定的類別的輸出對輸入影像的損失函數梯度，得到一個反映輸入特徵影響的 Saliency Map。
- Input X Gradient *(https://arxiv.org/abs/1605.01713)* (Shrikumar et al., 2016)：這種方法簡單地將輸入影像的梯度與輸入影像本身相乘，以計算每個像素的重要度。這可以顯示哪些像素對於模型的預測具有最大的影響。
- SmoothGrad *(https://arxiv.org/abs/1706.03825)* (Smikov et al., 2017)：此方法將輸入影像進行 n 次微小擾動，然後計算這些擾動下的梯度，最後將它們平均，獲得穩定的 Saliency Map。
- Integrated Gradients *(https://arxiv.org/abs/1703.01365)* (Sundararajan et al., 2017)：這是一種方法，通過對輸入特徵的整合來計算梯度，以評估每個特徵對於預測的影響。它提供了一種平滑的方法，可以顯示每個特徵對於預測的貢獻程度，而不僅僅是單一點的梯度。
- Grad-CAM *(https://arxiv.org/abs/1610.02391)*（Ancona et al., 2016）：它結合了卷積神經網路中的梯度訊息和特徵圖，以生成一個更具視覺解釋性的熱圖，顯示哪些區域在圖像中觸發了特定類別的預測。

透過權重與輸入的特徵來決定 Attribution

`Attribution` (歸因)是一個用於機器學習和深度學習領域的術語，它指的是試圖理解機器學習模型如何做出特定預測或決策的過程。以下使用一個簡單的線性迴歸例子，解釋 attribution 的概念。假設有兩個特徵會影響模型最終的輸出結果，那我想知道這兩個特徵的影響性分別有多

大。通常可以使用偏微分（Partial Derivatives）來進行分析。這是數學上的一個方法，可以計算某個函數對於其中一個變數的偏導數，以評估該變數對函數值的影響。

我們可以用一個簡單的例子來解釋如何計算特徵對模型輸出的影響程度。假設我們有一個簡單的線性模型如下：

$$y = 2x_1 + 3x_2$$

這個模型有兩個特徵，x1 和 x2，它們分別乘以不同的權重2和3。現在我們想知道 x1 和 x2 對 y 的影響程度。

1. 首先，我們計算對x1的偏導數：

   ∂y/∂x1 = 2

   這表示當 x1 增加1個單位時，y 會增加2個單位。所以，x1對 y 的影響是正向的，並且影響程度為2。

2. 接下來，我們計算對 x2 的偏導數：

   ∂y/∂x2 = 3

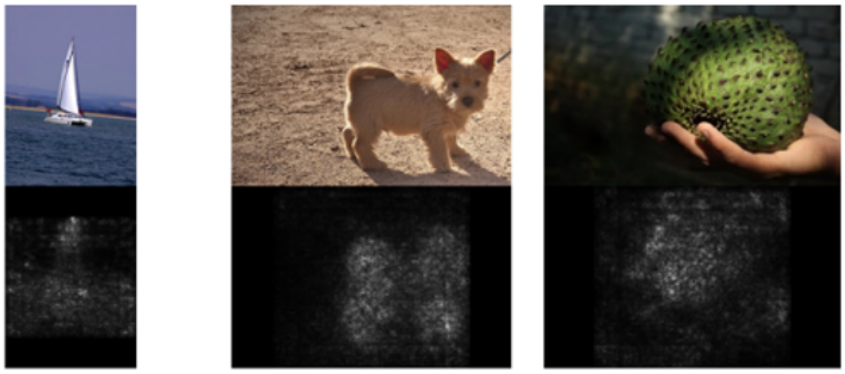
   這表示當 x2 增加1個單位時，y 會增加3個單位。所以，x2 對 y 的影響也是正向的，並且影響程度為3。

根據這個例子，我們可以得出結論，x2 對 y 的影響程度比 x1 大，因為 ∂y/∂x2 的絕對值大於 ∂y/∂x1 的絕對值。這是一個極簡單的例子，實際上的模型可能更複雜，但這個方法可以用來評估不同特徵對模型輸出的相對影響程度。此外這個方法假設模型是線性的，並且特徵之間不存在交互作用，對於複雜的非線性模型，可以使用一階泰勒展開式把它近似成線性模型。

Image-Specific Class Saliency (Sensitivity Analysis)

這裡介紹一個簡單的 Gradient-Based 方法，就是 Image-Specific Class Saliency(圖像特定類別顯著性)，也有些文獻稱它 Sensitivity Analysis (敏感度分析)。可以藉由計算輸入圖像的每個 pixel 的梯度，獲取 Saliency Map。整體步驟流程如下：

1. 模型推論：首先將一張影像餵入深度學習模型進行前向傳遞，獲得模型對該圖像的分類結果。
2. 計算梯度：針對我們感興趣的分類類別，計算該類別分數相對於輸入像素的梯度。這些梯度值反映了每個像素對於該類別預測的重要程度。
3. 視覺化結果：將計算得到的梯度視覺化呈現。可以選擇顯示絕對值的梯度，或者將正貢獻和負貢獻分開呈現，以突顯圖像中對於該類別預測影響最大的區域。這有助於理解模型為何將注意力集中在特定部分以做出特定的分類決策。

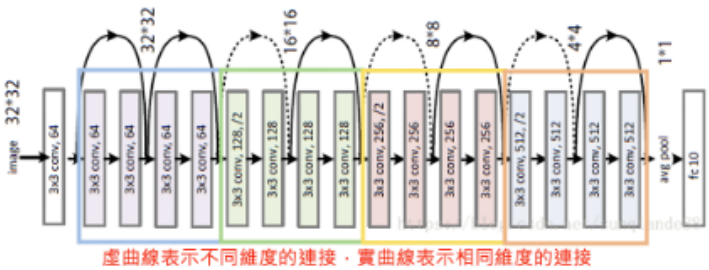


對數學與推導過程有興趣的讀者可以參考以下這份電子書。在文中作者稱上述方法為 Vanilla Gradient。

> 延伸閱讀：vanilla-gradient-saliency-maps *(https://christophm.github.io/interpretable-ml-book/pixel-attribution.html#vanilla-gradient-saliency-maps)*

Gradient-Based 方法實作 (Image-Specific Class Saliency)

在今天的範例中所使用的預訓練模型為 ResNet50。使用的方式跟昨天幾乎一模一樣，都是採用 `tf.keras.applications` 內建所提供的模型參數。ResNet50 突破了深度神經網路 Deep 的受限，引入了殘差學習的概念。通常隨著神經網路層數的增加，模型的性能會變得更好，但在一定深度後，模型會遭遇梯度消失或梯度爆炸的問題，導致訓練變得困難。因此 ResNet 引入 Residual Block（殘差模塊）允許訊息在不同層之間跳躍（即跳過某些層），進而解決了這個問題。



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet 論文： Deep Residual Learning for Image Recognition *(https://arxiv.org/abs/1512.03385)* (CVPR 2016 best paper)

載入預訓練模型(ResNet50)

首先使用 TensorFlow 載入 ResNet50 模型，將輸入張量(tensor)連接到預訓練的神經網路層，`imagenet` 表示使用在 ImageNet 資料集上預訓練的權重。`include_top=True` 表示輸出包括模型的最後分類層(全連接層)，此模型通常用於影像分類任務。

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Input

# 建立一個輸入張量，指定圖像大小為224x224（RGB色彩通道）
input_tensor = Input(shape=(224, 224, 3))
# 建立 ResNet50 模型
model = ResNet50(input_tensor=input_tensor, weights='imagenet', inclu
```

接著載入一張圖像，對其進行預處理。其中 `np.expand_dims()` 的目的是將圖像轉換為模型可接受的維度，這裡將圖像包裝在一個批次(batch)中，通常是一個批次只有一張圖像。最後使用 ResNet50 模型的預處理函數 `preprocess_input()` 來處理圖像，以確保圖像的數值範圍和格式符合模型的要求。

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import preprocess_input

# 載入圖像
image = tf.keras.utils.load_img('./dataset/cat_dog.jpg')
image = tf.keras.utils.img_to_array(image) # 將載入的圖像轉換為數組形式
x = np.expand_dims(image.copy(), axis=0) # 將圖像轉換為模型可接受的維度
# 預處理圖像
x = preprocess_input(x)
```

確認輸入影像都完成處理過後，就可以使用已建立的 ResNet50 模型進行圖像分類預測，返回分類機率。最後再使用 `decode_predictions()` 解析取得預測結果，並取得類別名稱和相對應的預測機率。`pred_class_idx` 則是預測的標籤索引，最後模型解釋會需要用到它。

```
from tensorflow.keras.applications.resnet50 import decode_predictions

# 進行圖像分類預測
pred_proba = model.predict(x) # 返回分類機率
# 解析預測結果
pred_class_idx = pred_proba.argmax(axis=1)[0]  # 找到具有最高機率的類別索引
pred_class = decode_predictions(pred_proba, top=1)[0][0]  # 解析取得預測
```

我們先來看看模型預測的結果。雖然這張影像同時有一隻貓和狗，但模型在神經網路中先抓取到狗的重要特徵(例如：鼻子、嘴巴)，因此最終模型預測 `bull_mastiff`(鬥牛獒)，該類別的機率值有 60%。有興趣的讀者，也可以跟昨天的模型進行比較。

```
import matplotlib.pylab as plt

plt.imshow(image.astype('uint8'))
plt.axis('off')
predicted_class_name = pred_class[1]
_ = plt.title(f"Prediction: {predicted_class_name} {pred_class[2]:.2f
```

Image-Specific Class Saliency 實作

首先我們將此演算法包裝成一個函式方便呼叫。這一個函式主要接受兩個參數：

- image: 包含一組圖像的輸入數據，這些圖像是要計算特定類別顯著性的對象。
- target_class_idx: 指定要計算特定類別顯著性的目標類別的標籤索引。

其中為了要進行梯度的計算，我們需要將輸入的圖片從 numpy array 轉換成 TensorFlow Tensor 格式。最後的 return 是計算目標類別機率 `probs` 對於輸入圖像 `image` 的梯度。`tape.gradient` 函數接受兩個參數，分別是梯度的目標和原影像，這裡我們希望計算目標是 `probs` 相對於輸入影像 `image`。

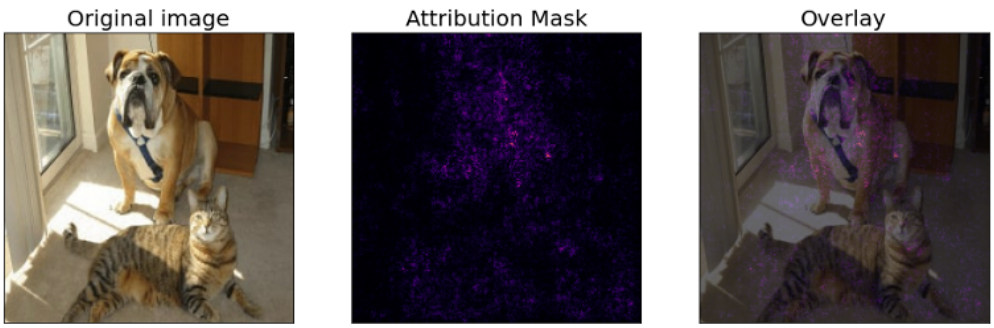
```
def compute_gradients(image, target_class_idx):
    image = tf.convert_to_tensor(image)  # 將輸入的影像轉換為 TensorFlow
    with tf.GradientTape() as tape:  # 用於計算梯度
        tape.watch(image)  # 監控圖像，以便計算其梯度
        predictions = model(image)  # 使用模型進行預測
        probs = predictions[:, target_class_idx]  # 取得目標類別的預測機率
    return tape.gradient(probs, image)  # 計算機率對於影像的梯度並返回
```

接著我們馬上將剛剛測試的 RestNet50 預測鬥牛獒的結果放進去此函式進行解釋。這段程式碼的目的是計算梯度並生成一個歸因遮罩（attribution mask）。首先呼叫 compute_gradients 函式，並傳入一個影像 x 和目標類別的索引 pred_class_idx。該函式會計算影像對於目標類別的梯度，並將結果儲存在 gradients 變數中。 接著使用 tf.squeeze 函式將 gradients 張量的多餘維度去除，然後使用 tf.math.abs 函式計算絕對值。這樣做是為了確保梯度的正負號不影響歸因遮罩的生成。最後使用 tf.reduce_sum 函式將計算結果沿著最後一個維度（通道軸）相加，生成最終的歸因遮罩，並將結果儲存在 attribution_mask 變數中。

```
# 計算機率對於影像的梯度
gradients = compute_gradients(
    image=x,
    target_class_idx=pred_class_idx)
# 產生可解釋的遮罩
attribution_mask = tf.reduce_sum(tf.math.abs(tf.squeeze(gradients)),
```

最後讓我們瞧瞧結果。

```
import matplotlib.cm as cm

fig, axs = plt.subplots(nrows=1, ncols=3, squeeze=False, figsize=(16,
                        subplot_kw={'xticks':[], 'yticks':[]})

axs[0, 0].set_title('Original image', fontsize=20)
axs[0, 0].imshow(image/255)

axs[0, 1].set_title('Attribution Mask', fontsize=20)
axs[0, 1].imshow(attribution_mask, cmap=cm.inferno)

axs[0, 2].set_title('Overlay', fontsize=20)
axs[0, 2].imshow(attribution_mask, cmap=cm.inferno)
axs[0, 2].imshow(image/255, alpha=0.4)
```



Reference

- Towards better understanding of gradient-based attribution methods for Deep Neural Networks *(https://arxiv.org/abs/1711.06104)*
- 可解釋 AI (XAI) 系列 — 02 基於梯度的方法 (Gradient-Based): Sensitivity analysis, SmoothGrad, Integrated Gradients *(https://medium.com/ai-academy-taiwan/%E5%8F%AF%E8%A7%A3%E9%87%8B-ai-xai-%E7%B3%BB%E5%88%97-02-%E5%9F%BA%E6%96%BC%E6%A2%AF%E5%BA%A6%E7%9A%84%E6%96%B9%E6%B3%95-gradient-based-b639932c1620)*

[Day 21] Propagation-Based：探索反向傳播法的可解釋性

今天所要談 Propagation-Based 方法在 CNN 中的作用是透過計算梯度、反向傳播或不同層的特徵來量化每個像素或特徵對預測結果的影響。

從昨天的 Gradient-based 方法我們了解到，可以使用梯度來估算每個輸入圖片的像素與最終分類結果之間的關係。換句話說，透過計算每個輸入圖片像素對於最終預測結果的影響，我們可以瞭解它們對於分類標籤的預測的貢獻程度。最終這些貢獻的總和等同於該圖像對於分類結果的預測。事實上這個概念可以在 CNN 的每一層都應用，即推斷每個特徵圖中的特徵與最終結果之間的相關性。這使得我們可以從最終輸出層開始回溯，而不僅僅針對輸入層計算梯度。以下為各位整理兩篇著名的 Propagation-Based 相關文獻：

- Layer-wise Relevance Propagation *(https://link.springer.com/book/10.1007/978-3-030-28954-6)* (LRP) (Bach et al., 2015)：透過泰勒分解來反向傳遞神經網路，以達到識別重要像素的方法。
- DeepLIFT *(https://arxiv.org/abs/1704.02685)*(Shrikumar et al., 2017)：是一種改進的反向傳播算法，用於生成熱圖以顯示神經網路的特徵重要性。SHAP 套件中的 Deep SHAP 就是基於 SHAP 和 DeepLIFT 算法。

Layer-wise relevance propagation

Layer-wise relevance propagation 是一種 Propagation-Based 方法的實例，用於理解深度神經網路對輸入數據的預測是如何形成的，並將其分解成單個輸入維度的相關性分數。這可以幫助我們理解神經網路中哪些部分對最終預測貢獻最大，對於模型的解釋和可解釋性非常有用。該方法被擴展到處理卷積神經網路中的特殊非線性操作，以更好地理解這些網路的工作原理。

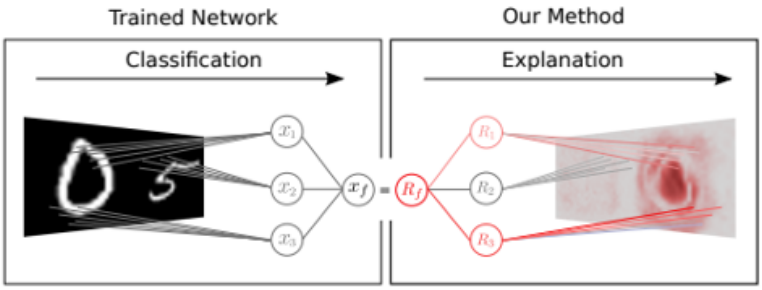
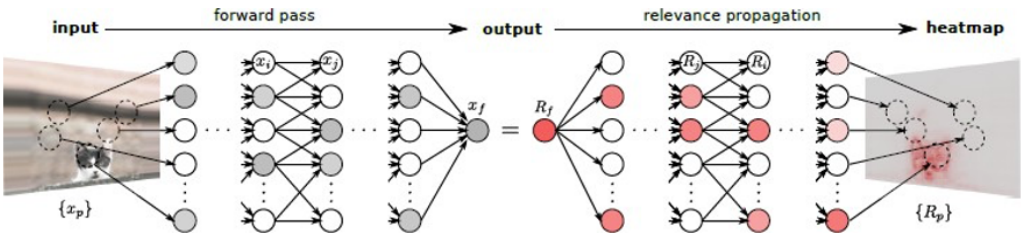


Fig. 1. Overview of our method for explaining a nonlinear classification decision. The method produces a pixel-wise heatmap explaining *why* a neural network classifier has come up with a particular decision (here, detecting the digit "0" in an input image composed of two digits). The heatmap is the result of a *deep* Taylor decomposition of the neural network function.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W., & Müller, K. R. (2017). Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern recognition*, *65*, 211-222.

從下圖可見，左側輸入一張圖片，經過網路中的一系列神經元計算後得到輸出結果 xf。這個輸出 xf 等同於 Rf，實際上是每一層的特徵經過相關性計算後總和的結果。因此右側的部分表示從輸出結果開始，透過層層傳遞，將重要訊息回饋到最初的輸入圖片中。



Montavon, G., Lapuschkin, S., Binder, A., Samek, W., & Müller, K. R. (2017). Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern recognition*, *65*, 211-222.

1. Input: 輸入是一張圖像，其中包含像素值 {xp}，這些像素值表示了圖像中的特徵。
2. Forward Propagation: 將像素值 {xp} 通過神經網路進行正向傳播。在神經網路中，經過多個層次的計算，獲得一個得分 f(x)，該得分指示了類別貓的存在。這個得分是由輸出神經元 xf 表示的，xf 的值編碼了圖像中是否包含貓的訊息。
3. Relevance Calculation: 接下來，計算輸出神經元 xf 的相對重要性（relevance），這個相對重要性記為 Rf，通常是 xf 的值本身。這表示輸出神經元對於類別貓的存在有多大的貢獻。
4. Backpropagation of Relevance: 從頂層開始，將相對重要性 Rf 進行反向傳播，傳播到較低的層次。這些相對重要性（Rp）被分配給了所有像素 {xp}，表示了它們對於最終類別分類的貢獻。
5. Relevance Redistribution: 最低隱藏層的最後一個神經元被認為對於更高層次的重要性較大，因此它將其分配的相對重要性重新分配到像素上。換句話說，這個神經元將影響圖像中某些特定區域的相對重要性。
6. Heatmap Generation: 通過以上步驟，得到了像素 {xp} 的相對重要性 heatmap，這個熱圖顯示了圖像中哪些區域對於最終類別分類更加重要。熱圖可用於可視化模型對圖像的關注點，以及在圖像中哪些區域包含了有關類別貓的訊息。

我們可以發現 LRP 方法主要是從最終輸出層逐步追溯到輸入層，逐層理解前層重要的神經元是如何影響輸出的，以此來獲得解釋性。下面公式 Ri 就是第 i 層節點的 Relevance，要計算它就是 Rj 第 j 層所有節點的 Relevance 並乘上一個權重，這個權重就是 xi 節點經過計算往下一層傳遞的數值。

$$R_i = \sum_j \frac{z_{ij}}{\sum_i z_{ij}} R_j$$

以下這三種不同的 LRP 規則是用來調整相對重要性在不同神經網路層次之間的傳播方式。它們有助於解釋模型的預測，並提供了對模型中不同層次神經元貢獻的不同理解。根據神經元所在的層次，選擇適當的 LRP 規則可以更好地理解模型的工作原理。

1. Basic Rule (LRP-0):

$$\textbf{LRP-0:} \qquad R_i = \sum_j \frac{a_i \omega_{ij}}{\sum_i a_i \omega_{ij}} R_j$$

應用層級: LRP-0 主要用於深層神經元，即神經網路中較高層次的神經元。

運作原理: 在 LRP-0 中，相對重要性（relevance）會傳遞到較低層的神經元，而且傳遞的方式是基於某種權重分佈的。這種權重分佈可以基於神經網路的結構和連接來計算，以確保相對重要性能夠合理地傳遞到較低層的神經元，間接提高了較高層次的貢獻。

2. Epsilon Rule (LRP-ε):

$$\textbf{LRP-}\varepsilon\textbf{:} \qquad R_i = \sum_j \frac{a_i \omega_{ij}}{\varepsilon + \sum_i a_i \omega_{ij}} R_j$$

應用層級: LRP-ε 通常應用於中層神經元，即神經網路中處於中間位置的神經元。

運作原理: 在 LRP-ε 中，相對重要性會以更平均的方式傳遞到較低層的神經元，不像 LRP-0 那麼專注於少數高度相對重要的神經元。這種方法有助於平衡訊息的傳遞，以更全面地考慮中間層次的貢獻。

3. Gamma Rule (LRP-γ):

$$\textbf{LRP-}\gamma\textbf{:} \qquad R_i = \sum_j \frac{a_i \cdot (\omega_{ij} + \gamma \omega_{ij}^+)}{\sum_i (\omega_{ij} + \gamma \omega_{ij}^+)} R_j$$

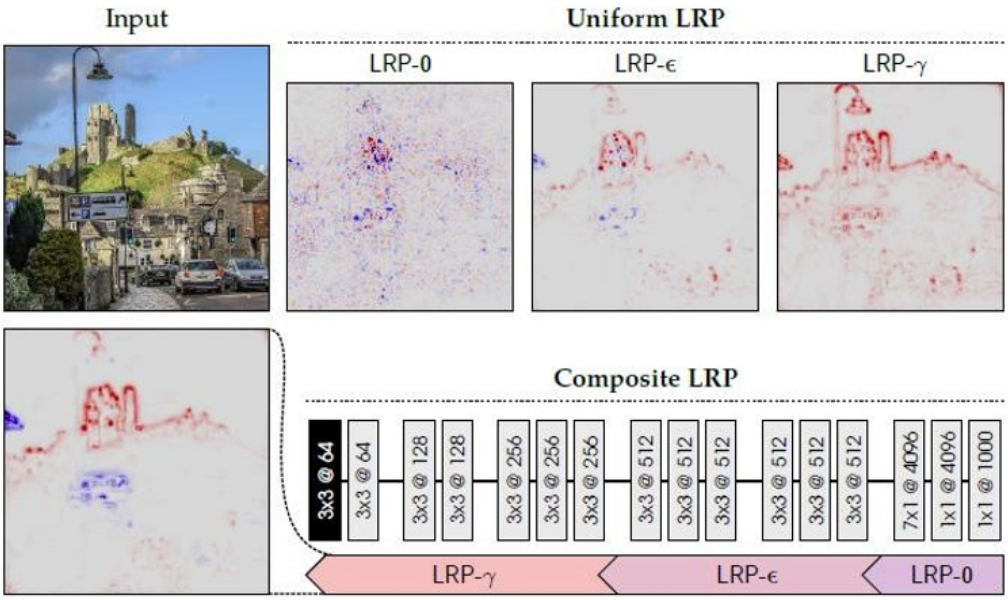
應用層級: LRP-γ 通常應用於較淺層的神經元，即神經網路中較低層次的神經元。

運作原理: LRP-γ 會更強調相對重要性在較低層神經元之間的分佈，這意味著相對重要性會更加集中於較低層的特定神經元。這有助於解釋模型為什麼會對某些輸入特徵產生強烈的反應，並且能夠將模型的決策過程更清晰地關聯到較低層次的特徵。

實驗結果

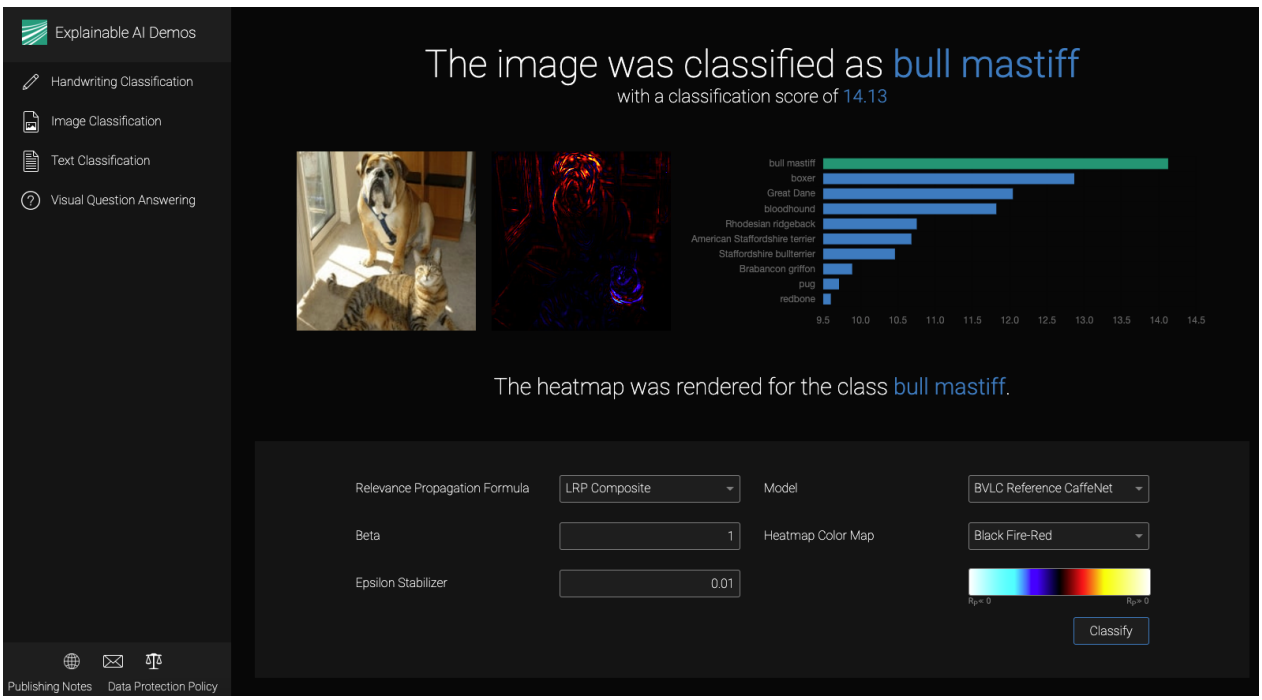
讓我們來觀察不同計算方法對最終解釋性的影響。在下圖中，紅色的點代表對辨識結果有正向幫助，而藍色的點則代表有負向的影響。我們也可以觀察到以下情況：當使用 LRP-0 時，產生了相當多的雜訊；LRP-ε 的效果看起來相對不錯，並且減少了雜訊點的出現；而最右邊的 LRP-γ

則沒有出現對辨識結果有負面影響的點，因此沒有藍色點。最後論文又提了一種方法是在神經網路中依據不同層使用不同的計算方法，其中在最後的全連階層使用 LRP-0，在深層取得比較多特徵的地方使用 LRP-ε，在淺層的地方使用 LRP-γ。最終的結果看起來遠比三個單獨使用效果顯著。



Montavon, G., Binder, A., Lapuschkin, S., Samek, W., & Müller, K. R. (2019). Layer-wise relevance propagation: an overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, 193-209.

各位可以試試看這個互動網站 *(https://lrpserver.hhi.fraunhofer.de/image-classification)*，裡面實現了 LRP 方法，並可以調整參數控制解釋性。



DeepLIFT

DeepLIFT 是一種反向傳播的方法，類似於 LRP。每個單元 i 都被分配一個歸因值，表示該單元對於原始網路輸入 x 的激發相對於某個參考輸入 x⁻ 的相對效應。可以參考下方公式3：

$$r_i^{(L)} = \begin{cases} S_i(x) - S_i(\bar{x}) & \text{if unit } i \text{ is the target unit of interest} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

對於所有隱藏層單元，參考值 z⁻ji 是通過對網路進行前向傳遞，使用基準輸入 x⁻，並記錄每個單元的激發數值來確定的。與 LRP 一樣，基準通常選擇為零。相關性傳播過程可以通過下方公式4描述。

$$r_i^{(l)} = \sum_j \frac{z_{ji} - \bar{z}_{ji}}{\sum_{i'} z_{ji} - \sum_{i'} \bar{z}_{ji}} r_j^{(l+1)} \quad (4)$$

簡單來說 DeepLIFT 理論的核心概念是將模型的輸入特徵歸因到每個特徵的貢獻度，以解釋模型的預測過程。如果要實作 DeepLIFT 可以參考原始論文在 GitHub 的開源套件 DeepLIFT: Deep Learning Important FeaTures *(https://github.com/kundajelab/deeplift)*。此外在 SHAP 套件提供的 DeepExplainer 解釋方法就是 DeepLIFT 和 Shapely values 的結合。詳細的實作內容可以參考 [Day 17] 解析深度神經網路：使用Deep SHAP進行模型解釋 *(https://ithelp.ithome.com.tw/articles/10331443)*。

Reference

- Explaining NonLinear Classification Decisions with Deep Taylor Decomposition *(https://arxiv.org/abs/1512.02479)*
- Towards better understanding of gradient-based attribution methods for Deep Neural Networks *(https://arxiv.org/abs/1711.06104)*
- 可解釋 AI (XAI) 系列 — 03 基於傳播的方法 (Propagation-Based): Layer-Wise Relevance Propagation *(https://medium.com/ai-academy-taiwan/%E5%8F%AF%E8%A7%A3%E9%87%8B-ai-xai-%E7%B3%BB%E5%88%97-03-%E5%9F%BA%E6%96%BC%E5%82%B3%E6%92%AD%E7%9A%84%E6%96%B9%E6%B3%95-propagation-based-layer-wise-relevance-propagation-1b79ce96042d)*

[Day 22] CAM-Based：如何解釋卷積神經網路

範例程式：  *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/22.CAM-Based：如何解釋卷積神經網路.ipynb)*

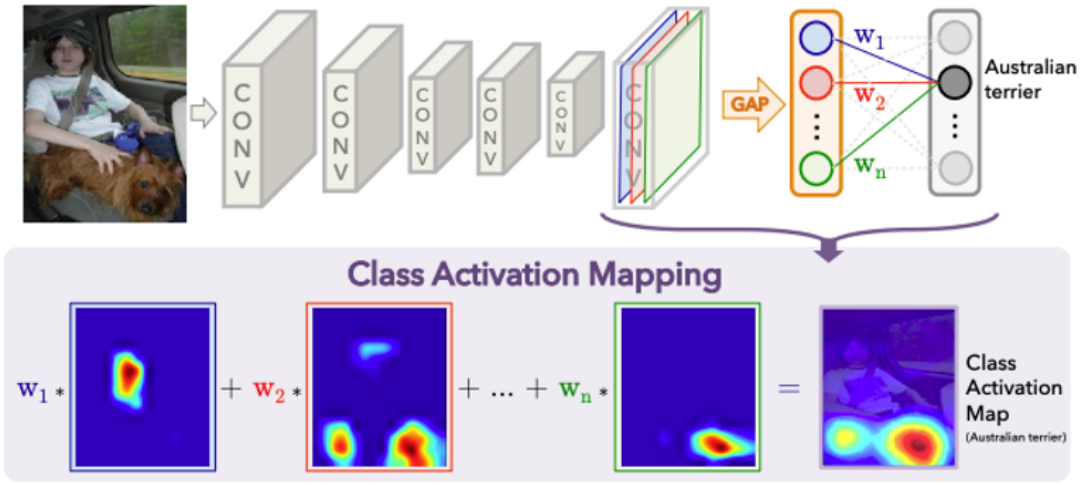
CAM（Class Activation Mapping）是一種用於解釋卷積神經網路（CNN）模型在圖像分類任務中的預測的技術。它的目的是生成一個視覺化的熱圖，以顯示模型在圖像中關注的區域，以及這些區域對於模型預測某一類別的貢獻。以下是近年來與 CAM 相關的變形方法的整理：

- CAM (Class Activation Mapping) *(https://arxiv.org/abs/1512.04150)* (Zhou et al. 2015): 基本的CAM方法是用來解釋CNN模型的預測，通過將卷積層的特徵圖與全連接層的權重相結合，生成類別特定的熱圖，以顯示模型對於不同類別的注意力分佈。
- Grad-CAM (Gradient-weighted Class Activation Mapping) *(https://arxiv.org/abs/1610.02391)* (Ancona et al. 2016): Grad-CAM建立在CAM的基礎上，使用梯度訊息來更準確地計算特徵圖上的權重。它通過將卷積層的梯度值與特徵圖相乘，生成熱圖，以視覺化模型對於不同類別的關注點。
- Grad-CAM++ *(https://arxiv.org/abs/1710.11063)* (Chattopadhyay et al. 2017): Grad-CAM++是Grad-CAM的進一步改進版本，它引入了多尺度特徵圖的概念，以提高對於細微細節的解釋能力。它使用多個卷積層的特徵圖來生成更具細節的熱圖。
- Score-CAM *(https://arxiv.org/abs/1910.01279)* (Haofan et al. 2020): Score-CAM是一種用於解釋模型預測的方法，它通過將通道的分數（score）和特徵圖相乘，生成類別特定的熱圖。它旨在提高CAM方法的性能，並減少噪聲。

在今天的內容中，將介紹 CAM 和 Grad-CAM 的差別，最後使用 TensorFlow 來示範如何運用 Grad-CAM 方法來解釋 CNN 模型。

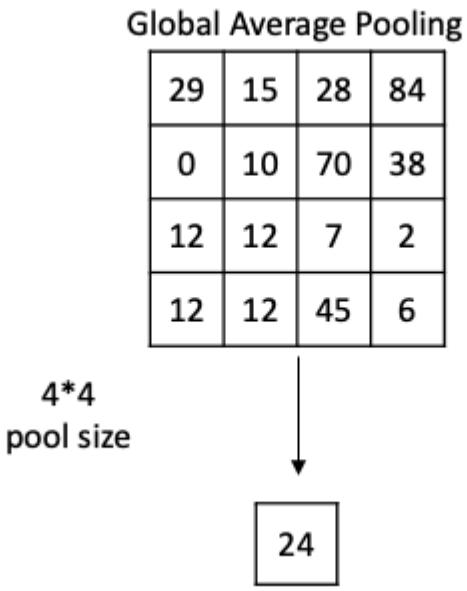
CAM

CAM 在 2015 年被提出，論文名稱為 Learning Deep Features for Discriminative Localization *(https://arxiv.org/abs/1512.04150)*，它的主要概念是將 CNN 的最後一個卷積層的特徵圖之後接上全局平均池化層(Global Average Pooling Layer)與全連接層的權重相結合。首先計算每個類別的權重，這些權重是全連接層的權重與卷積層的特徵圖相乘後的總和。接著將這些權重應用到卷積層的特徵圖上，生成一個類別特定的熱圖。這個熱圖顯示了對於某一特定類別，圖像的不同區域的相對重要性。最後熱圖經過適當的正規化處理，以確保它可以被視覺化並與原始圖像尺寸對應。

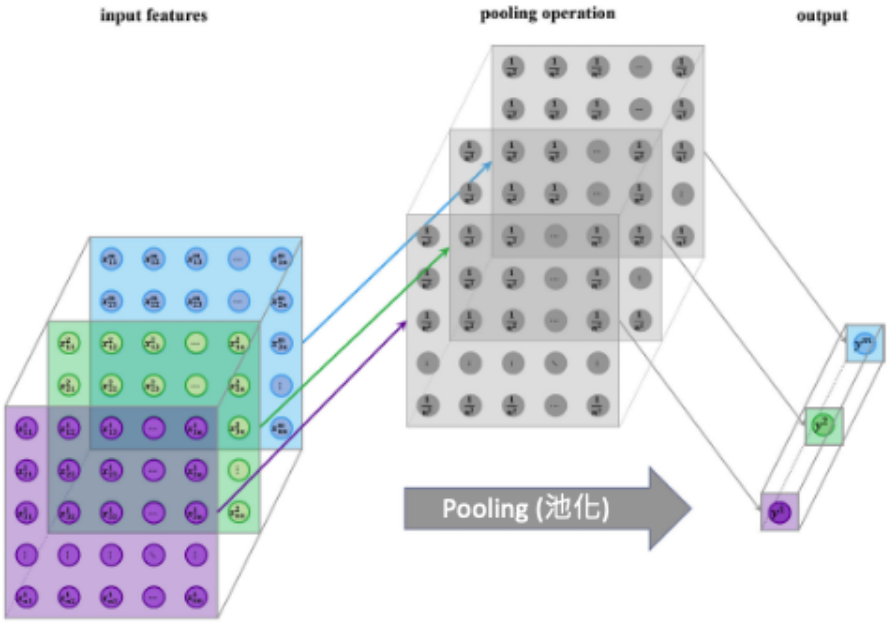


Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba. (2015). Learning Deep Features for Discriminative Localization.

全局平均池化（Global Average Pooling）是一種用於卷積神經網路（CNN）中的特徵映射的操作。它通常用於卷積神經網路的最後一層，以幫助縮減特徵圖的維度並生成一個固定大小的輸出，以供後續的分類或回歸任務使用。全局平均池化的過程非常簡單，以下用一張特徵圖為例，假設 pool size 4*4 也就是對整張圖做平均得到 24 這就是一個通道(一張特徵圖) GAP 後的結果。



簡單來說全局平均池化做法屬於空間維度的一種特徵壓縮，因為這個實數是根據所有特徵值計算出來的，所以在某種程度上具有平均感受野，最後保持通道數不變，所以最後輸出大小為 1x1xC：



經過全局平均池化（GAP）後，我們得到了一個包含權重 w 的像素陣列。這些權重 w 反映了每張特徵圖對於最終預測某個類別的重要性。更大的權重值表示相應特徵圖對於該分類的貢獻更大。

$$M_c(x,y) = \sum_k w_k^c f_k(x,y)$$

為了更好地理解這些權重，我們可以將它們應用於整張特徵圖上的每個像素點，然後進行疊加。這樣做的好處是我們可以根據每張特徵圖的重要性來集中關注不同的區域。如果某特定分類的權重 w 很大，那麼該分類所對應的特徵圖將在進行疊加時具有更大的影響。相反，權重接近於0的特徵圖對於最終的關注度較低，因此它們被視為不太重要的部分。

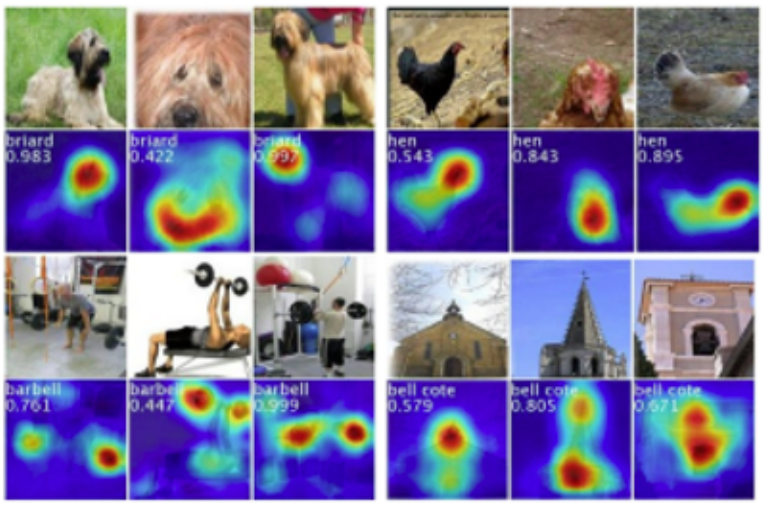


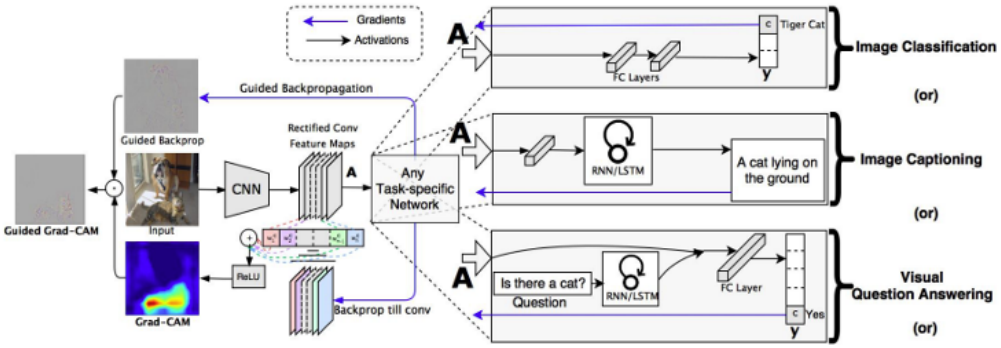
Figure 3. The CAMs of four classes from ILSVRC [20].

此方法確實為解釋 CNN 模型的預測提供了重要的基礎，但它存在一個明顯的限制，即受到了網路架構的限制。因為如果要使用 CAM 解釋，在 CNN 架構當中最後一層一定要接上 GAP 層。隨後的方法，如Grad-CAM和其變種，試圖克服這些限制，使解釋性方法更具通用性，適用於各種深度學習模型。

Grad-CAM

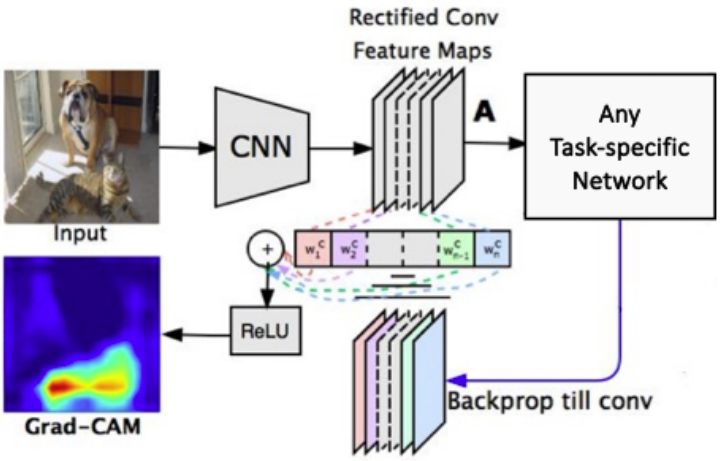
從 CAM 的介紹中，我們了解到實現 CAM 所需的一個關鍵條件是必須加入全局平均池化（GAP）層。然而這種要求在某種程度上限制了網路架構的自由度，因為模型必須包含 GAP 層才能使用 CAM。更重要的是，如果我們堅持要求模型的最後一層必須是GAP層，這可能會影響模型的準確性。因此 Grad-CAM 在這方面解除了這種限制，它出自於這邊論文 Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization *(https://arxiv.org/abs/1610.02391)*。使用 Grad-CAM 我們可以對一般的 CNN 架構進行解釋，而不需要強制要求模型的最後一層必須是 GAP 層。這意味著我們可以更靈活地應用 Grad-CAM 來獲得 CNN 對輸入圖片的關注區域，而無需對模型進行大幅度的修改，同時不會明顯影響模型的準確性。

從下圖中可以清楚地看到，不論是全連接層、RNN、LSTM，或者更複雜的網路模型，都可以透過 Grad-CAM 取得神經網絡的分類關注區域熱力圖。



Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra. (2016). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization.

簡單來說 Grad-CAM 的關鍵在於能夠透過反向傳播計算用於 CAM 的權重 w。我們可以針對每個特徵圖進行反向傳播，計算梯度的平均值，以獲得相對應的權重。最後，將這些權重（W_1, W_2, …, Wn）與特徵圖相乘，然後進行總和，就可以觀察到不同位置對輸出的影響。



Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra. (2016). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization.

CAM-Based 方法實作 (Grad-CAM)

今天的範例將使用 Xception 預訓練模型來示範如何透過 Grad-CAM 解釋模型推論結果。Xception 的名稱是由 `Extreme Inception` 衍生而來，因為它基於 Inception v3 架構的概念，並對一些地方進行了極端擴展和改進。

> Xception 論文：Xception: Deep Learning with Depthwise Separable Convolutions *(https://arxiv.org/abs/1610.02357)*

載入預訓練模型(Xception)

首先使用 TensorFlow 載入 Xception 模型，將輸入張量(tensor)連接到預訓練的神經網路層，`imagenet` 表示使用在 ImageNet 資料集上預訓練的權重。`include_top=True` 表示輸出包括模型的最後分類層(全連接層)，此模型通常用於影像分類任務。

```
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.layers import Input

# 建立一個輸入張量，指定圖像大小為224x224（RGB色彩通道）
input_tensor = Input(shape=(224, 224, 3))
# 建立 Xception 模型
model = Xception(input_tensor=input_tensor, weights='imagenet', inclu
```

接著載入一張圖像，對其進行預處理。其中 `np.expand_dims()` 的目的是將圖像轉換為模型可接受的維度，這裡將圖像包裝在一個批次(batch)中，通常是一個批次只有一張圖像。最後使用 Xception 模型的預處理函數 `preprocess_input()` 來處理圖像，以確保圖像的數值範圍和格式符合模型的要求。

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.xception import preprocess_input

# 載入圖像
image = tf.keras.utils.load_img('./dataset/cat_dog.jpg')
image = tf.keras.utils.img_to_array(image) # 將載入的圖像轉換為數組形式
x = np.expand_dims(image.copy(), axis=0) # 將圖像轉換為模型可接受的維度
# 預處理圖像
x = preprocess_input(x)
```

確認輸入影像都完成處理過後，就可以使用已建立的 Xception 模型進行圖像分類預測，返回分類機率。最後再使用 decode_predictions() 解析取得預測結果，並取得類別名稱和相對應的預測機率。

```
from tensorflow.keras.applications.xception import decode_predictions

# 進行圖像分類預測
pred_proba = model.predict(x) # 返回分類機率
# 解析預測結果
pred_class = decode_predictions(pred_proba, top=1)[0][0]  # 解析取得預測
```

我們先來看看模型預測的結果。雖然這張影像同時有一隻貓和狗，但模型在神經網路中先抓取到狗的重要特徵(例如：鼻子、嘴巴)，因此最終模型預測 bull_mastiff(鬥牛獒)，該類別的機率值有 48%。

```
import matplotlib.pylab as plt

plt.imshow(image.astype('uint8'))
plt.axis('off')
predicted_class_name = pred_class[1]
_ = plt.title(f"Prediction: {predicted_class_name} {pred_class[2]:.2f
```

Grad-CAM 實作

這段函式主要用於生成Grad-CAM熱圖，以視覺化深度學習模型對於輸入圖像的預測結果的解釋性。首先建立一個新的模型 grad_model，該模型接受相同的輸入圖像，但同時輸出最後一個卷積層的輸出和整個模型的預測結果。這是為了能夠計算特定類別對於最後一個卷積層的梯度。接下來，使用 TensorFlow 的 GradientTape 來記錄計算梯度。它計算了模型對於輸入圖像的預測，同時記錄了對應於特定類別的分類神經元相對於最後一個卷積層輸出的梯度。然後計算這些梯度的平均強度 pooled_grads，並取得一個向量做為權重，每個元素對應於最後一個卷積層的特徵圖通道。接著將最後一個卷積層的輸出與 pooled_grads 做點積，這個操作強調了對於特定類別預測重要的特徵圖區域。得到的結果是一個熱圖，其中每個像素值表示相應位置對於該類別預測的重要性。最後為了視覺化，將熱圖的值正規化為介於0和1之間，以便更容易理解和視覺化。

```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred
    # 建立一個模型，同時輸出最後一個卷積層和整個模型的預測結果
    grad_model = tf.keras.models.Model(
        model.inputs, [model.get_layer(last_conv_layer_name).output,
    )

    # 計算對於輸入圖像的預測類別，相對於最後一個卷積層的梯度
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # 輸出分類神經元相對於最後一個卷積層的輸出特徵圖的梯度
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # 這是一個向量，其中每個數字都是特定特徵圖通道上的梯度的平均強度
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # 將特徵圖乘以權重，等於該特徵圖中的某些區域對於該分類的重要性
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap) # 然後將所有通道相加以獲得熱圖

    # 為了視覺化，將熱圖正規化0~1之間
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
```

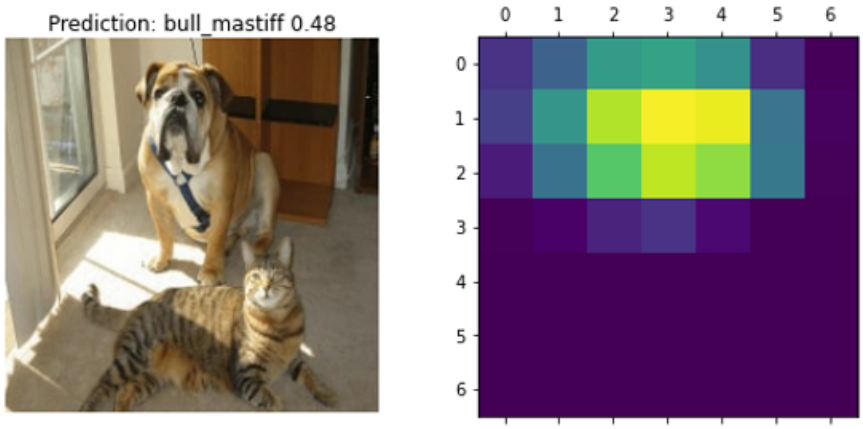
接下來我們實際將影像、模型以及最後一層卷積的名稱餵入我們剛剛建立的函式，然後繪製出一張熱圖。在這篇文章中，我們使用的是 Xception 模型架構，其最後一層卷積層的名稱為

block14_sepconv2_act。如果不知道最後一層的名稱是什麼，可以透過 model.summary() 來查看模型的結構，然後尋找最後一個 Conv2D 層的名稱。

```
block14_sepconv2_bn (BatchNorm  (None, 7, 7, 2048)  8192        ['block14_sepconv2[0][0]']
alization)

block14_sepconv2_act (Activati  (None, 7, 7, 2048)  0           ['block14_sepconv2_bn[0][0]']
on)

avg_pool (GlobalAveragePooling  (None, 2048)        0           ['block14_sepconv2_act[0][0]']
2D)

predictions (Dense)            (None, 1000)         2049000     ['avg_pool[0][0]']

==================================================================================================
Total params: 22,910,480
Trainable params: 22,855,952
Non-trainable params: 54,528
__________________________________________________________________________________________________
```

```
# 產生 class activation heatmap
last_conv_layer_name = "block14_sepconv2_act"
heatmap = make_gradcam_heatmap(x, model, last_conv_layer_name)

# 顯示 heatmap
plt.matshow(heatmap)
plt.show()
```

由於最後一層的卷積特徵圖大小為 (None, 7, 7, 2048)，因此最後計算出來的熱圖大小為一張將過正規化的 7*7 像素大小的熱圖。



最後這段程式是將剛剛得到的熱圖與原始影像進行疊合視覺化結果。此函式接受四個主要的輸入參數：

- img_path: 輸入圖像的路徑，這是要解釋的圖像。
- heatmap: Grad-CAM 熱圖，表示模型對圖像中不同區域的關注程度。
- cam_path: 輸出的熱圖圖像檔案的路徑。
- alpha: 控制疊加熱圖和原始圖像的透明度。

```
import matplotlib.cm as cm
from IPython.display import Image, display
```

```
def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", a
    # 載入原始圖像
    img = tf.keras.utils.load_img(img_path)
    img = tf.keras.utils.img_to_array(img)

    # 將熱圖重新縮放到0-255的範圍
    heatmap = np.uint8(255 * heatmap)

    # 使用Jet色彩映射將熱圖上色
    jet = cm.get_cmap("jet")

    # 使用Jet色彩映射的RGB值
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    # 創建帶有RGB色彩的熱圖圖像
    jet_heatmap = tf.keras.utils.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = tf.keras.utils.img_to_array(jet_heatmap)

    # 在原始圖像上疊加熱圖
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = tf.keras.utils.array_to_img(superimposed_img)

    # 儲存疊加後的圖像
    superimposed_img.save(cam_path)

    # 顯示Grad CAM
    display(Image(cam_path))
```

```
save_and_display_gradcam('./dataset/cat_dog.jpg', heatmap)
```

最後我們就能得到一張完整的 Grad-CAM 解釋圖了。從圖中可很明顯知道模型辨識一隻狗是關注影像中的哪一個特徵作為判斷依據。

如果想透過第三方套件進行 CNN 模型解釋也可以使用 tf_explain *(https://github.com/sicara/tf-explain)*, 它是一個針對 TensorFlow 深度學習模型解釋的 Python 套件。內建提供了多種方法, 並透過熱力圖可以用來觀察模型是否學到關鍵特徵。

Reference

- Grad-CAM class activation visualization *(https://keras.io/examples/vision/grad_cam/)*

[Day 23] Attention-Based：使用注意力機制解釋CNN模型

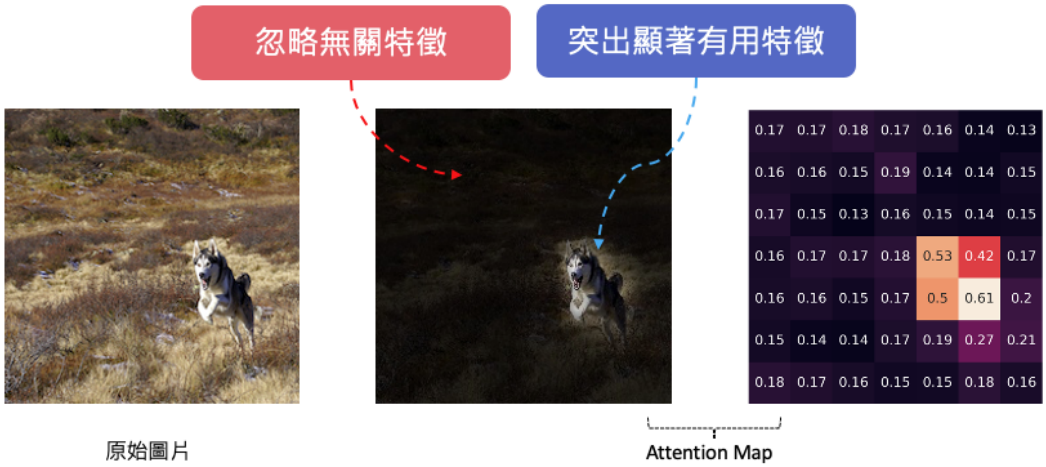
範例程式：  *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/23.Attention-Based：使用注意力機制解釋CNN模型.ipynb)*

近年來注意力機制（Attention Mechanism）已經成為深度學習和神經網路領域的一個重要研究。它不僅能夠改善模型的性能，還可以增強模型的解釋性。2018 年圖靈獎得主同時也是深度學習三大巨頭之一的 Yoshua Bengio 也曾經說過：Attention is a core ingredient of 'conscious' AI *(https://venturebeat.com/ai/yoshua-bengio-attention-is-a-core-ingredient-of-consciousness-ai/)*。透過有意識的認知系統，進一步模仿人類的行為，以提高模型學習的效果，這就是注意力機制的研究主題。在前幾天的文章中我們學習了許多 CNN 的解釋方法，如 Perturbation-Based、Gradient-Based、Propagation-Based、CAM-Based等，在解釋 CNN 模型方面有一定的局限性。為了克服這些局限性，研究人員開始採用注意力機制，並將所謂的 Attention Layer 其嵌入到 CNN 模型中，以實現更好的解釋性和性能。當然注意力機制不僅僅只侷限在圖像識別任務上，自然語言處理及語音處理也可以用到此技術，這也意味著各種神經網路都可以套用注意力機制的概念。然而在今天的文章內容當中，我們將關注在如何透過注意力機制應用在圖像識別任務上。

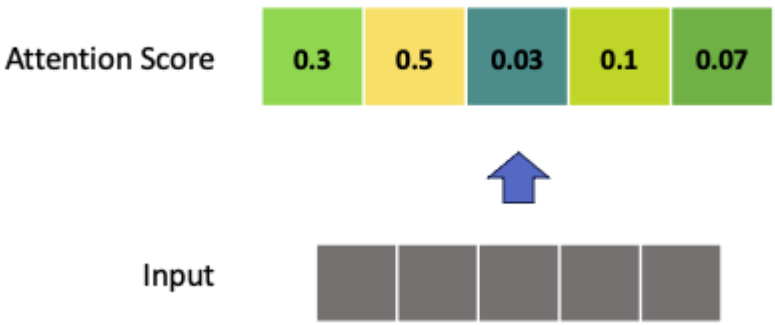
> 延伸閱讀：淺談有意識的 AI： 注意力機制介紹 *(https://medium.com/@andy6804tw/%E6%B7%BA%E8%AB%87%E6%9C%89%E6%84%8F%E8%AD%98%E7%9A%84-ai-%E6%B3%A8%E6%84%8F%E5%8A%9B%E6%A9%9F%E5%88%B6%E4%BB%8B%E7%B4%B9-59e*

注意力機制的優勢

在探討如何使用注意力機制來解釋 CNN 模型之前，讓我們首先理解注意力機制的優勢。注意力機制允許模型根據輸入的不同部分調整其關注程度，這使得模型更能夠理解和解釋複雜的數據。相比於傳統的解釋方法，注意力機制更具靈活性，能夠從特徵圖中自動學習關鍵訊息，並將其納入決策過程。

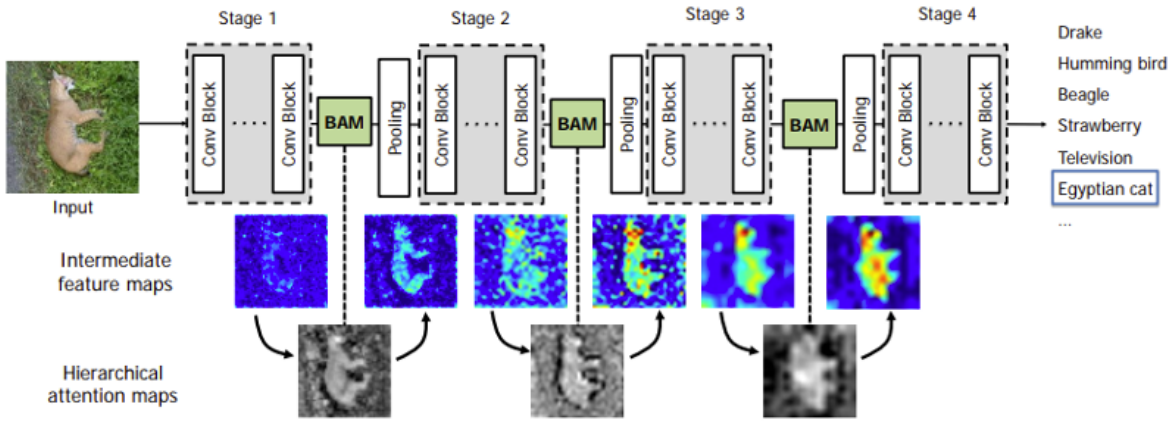


在注意力機制中，我們通常使用一組向量權重（或分數）來指示模型在處理輸入序列時對不同部分的關注程度。讓我們以數學符號來表示：



1. 輸入序列：假設我們有一個長度為N的輸入序列，通常表示為$X = [x_1, x_2, ..., x^N]$，其中每個x_i是一個特徵或時間序列的表示。
2. 注意力權重：我們引入一個注意力權重向量$A = [a_1, a_2, ..., a^N]$，其中每個a_i代表模型對輸入序列中相應位置的關注程度。這些權重通常是實數，並滿足正規化條件，即$\Sigma a_i = 1$（注意力機制總和為1）。

向量權重在數學上代表了注意力機制的核心概念，它們決定了模型在處理輸入序列時的關注程度，並有助於提高模型的效能和解釋性。套用在卷積神經網路中我們可以計算每一個特徵圖(feature maps)所對應的注意力權重，並可以知道途中哪個位置是模型關注的地方。那該如何計算注意力分數呢？這就是另一門學術研究在探討的事情，因為他有非常多的技巧可以計算這些注意力分數。下圖就是取自一篇 BAM: Bottleneck Attention Module *(https://arxiv.org/abs/1807.06514)*研究，在卷積層間加入一個注意力模塊，並將含有注意力機制的特徵圖繪製出來，觀察模型是否真的有關注到重點區域。

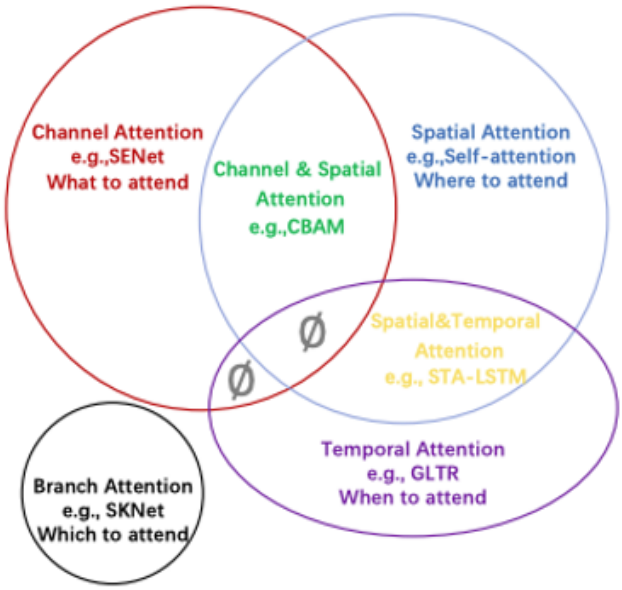


Jongchan Park, Sanghyun Woo, Joon-Young Lee, In So Kweon. (2018). BAM: Bottleneck Attention Module.

注意力機制於電腦視覺模型

注意機制被引入到電腦視覺中，其目的是模仿人類視覺系統。這種注意機制可以看作是一個基於輸入圖像特徵的動態權重調整過程。其中這篇論文 Attention Mechanisms in Computer Vision: A Survey *(https://arxiv.org/abs/2111.07624.pdf)* 總結了電腦視覺中的各種注意機制，並對所有 CV Attention 研究進行分類。相關的論文研究整理也能參考 GitHub 上的專案 Awesome-Vision-Attentions *(https://github.com/MenghaoGuo/Awesome-Vision-Attentions)* 原作者統整了近年電腦視覺領域中各種注意力機制的研究。就注意力關注的域來分，大致可以分成以下六種：

- 通道注意力 (Channel Attention)
- 空間注意力 (Spatial Attention)
- 時間注意力 (Temporal Attention)
- 分支注意力 (Branch Attention)
- 通道空間注意力 (Channel & Spatial Attention)
- 時空注意力 (Spatial & Temporal Attention)

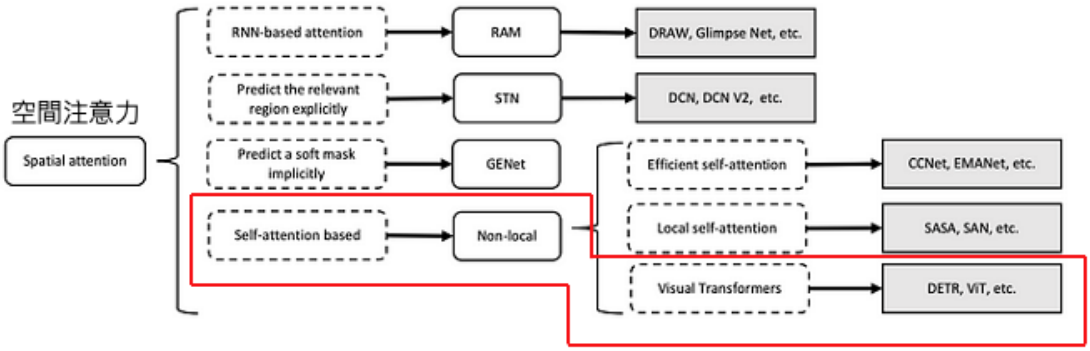


Meng-Hao Guo, Tian-Xing Xu, Jiang-Jiang Liu, Zheng-Ning Liu, Peng-Tao Jiang, Tai-Jiang Mu, Song-Hai Zhang, Ralph R. Martin, Ming-Ming Cheng. (2015). Attention Mechanisms in Computer Vision: A Survey.

延伸閱讀：電腦視覺中的注意力機制 *(https://medium.com/@andy6804tw/%E9%9B%BB%E8%85%A6%E8%A6%96%E8%A6%BA%E4%B8%AD%E7%9A%84%E6%B3%A8%E6*

注意力機制於 Transformer 模型

自從2017年Google發表了 Attention Is All You Need *(https://arxiv.org/abs/1706.03762)* 這篇論文，Transformer 架構的提出完全改變了神經網路的局勢。在這篇論文中，他們引入了一種稱為 Transformer 的網路結構，它不再依賴於 RNN 或 CNN，而是完全採用了自注意力機制（self-attention mechanism）。這種自注意力機制允許模型在 Encoder-Decoder 之間更靈活地操作，為深度學習帶來了更多的靈活性。然而 self-attention 這樣的技術應用在電腦視覺中是屬於剛剛提到的六大項中的空間注意力 (Spatial Attention)這一大類。



在電腦視覺領域，Transformer 架構的變形和經典網路相結合已經產生了多個成功的模型。以下是一些使用 Transformer 架構的經典或知名模型的變形：

- Vision Transformer *(https://arxiv.org/abs/2010.11929)*（ViT）(Dosovitskiy et al., 2020) `Google Research`：Vision Transformer是將Transformer架構應用於圖像分類的經典模型。它將圖像分割為一系列固定大小的塊，然後將它們轉換成序列，並使用Transformer中的Encoder來處理這個序列達到影像分類。

- Data-efficient Image Transformer *(https://arxiv.org/abs/2012.12877)*（DeiT）(Touvron et al., 2020) `Facebook AI`：DeiT是一種數據高效的Vision Transformer，透過數據增強和知識蒸餾技術，能夠在資料有限的狀況下，同時能夠保持跟ViT一樣的效果甚至比用CNN的網路還來得好。

- MobileViT *(https://arxiv.org/abs/2110.02178)* (Mehta et al., 2021) `Apple`：MobileViT是一個專為移動設備優化的變種Vision Transformer模型，它保持了高性能的同時，具有更小的模型大小和更低的計算成本。

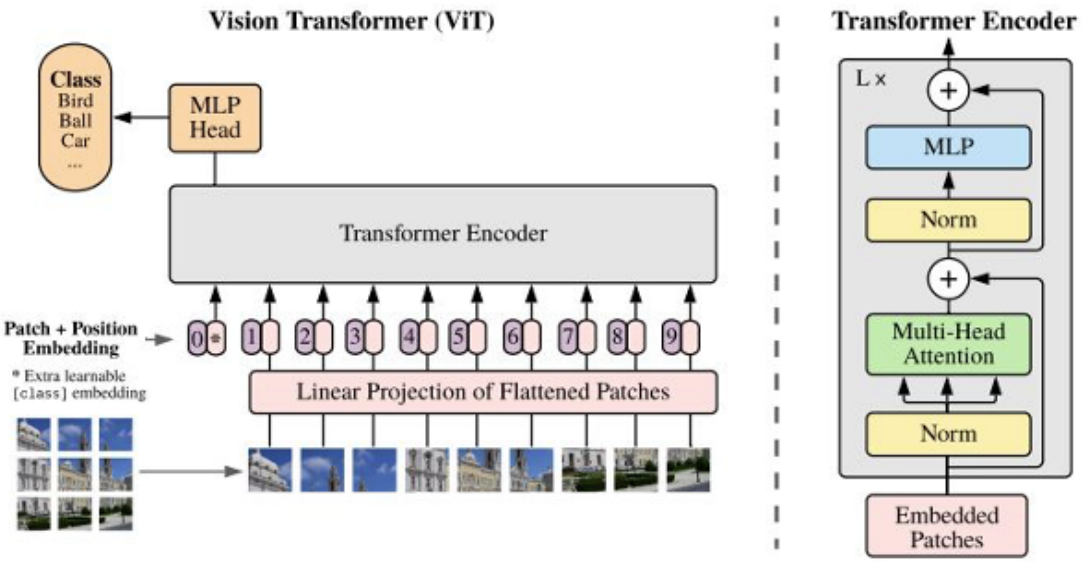
- Swin Transformer *(https://arxiv.org/abs/2103.14030)* (Liu et al., 2021) `Microsoft Research`：Swin Transformer是一種多尺度特徵建模的模型，透過滑動窗口的方法進行注意力機制操作，可以處理不同尺度的特徵。它在圖像分類、物體檢測和分割等任務上取得了優異的成績。

- Class-Attention in Image Transformers *(https://arxiv.org/abs/2103.17239)*（CaiT）(Touvron et al., 2021) `Facebook AI`：CaiT是一種基於Transformer的模型，它引入了類別感知（class-aware）的注意力機制，以改善圖像分類性能。這種模型通常對圖像中不同類別的訊息進行更好的建模。

從上述論文中可以觀察到，大多數研究工作都來自大型企業或研究機構，這是因為訓練 Transformer 模型需要龐大的資料集和計算資源。因此要進入 Transformer 的研究領域，需要具備豐富的資源投入。舉例來說，像最近備受關注的 ChatGPT，背後依賴於大型語言模型（LLM），例如 GPT、PaLM 和 LLaMA ...等，這些語言模型都建立在Transformer 架構之上，訓練一次可能需要數千萬甚至億級的成本。幸運的是，這些機構通常會釋出預訓練模型的權重，因此我們可以通過微調或遷移學習等方式來滿足我們的需求。

Attention-Based 實作 (Vision Transformer)

今天的範例將使用 Vision Transformer 預訓練模型來示範如何透過 Attention Rollout 方法解釋模型推論結果。Transformer 已經成為熱門的神經網路架構，並廣泛應用於自然語言處理任務，像是現今最火紅的 ChatGPT 背後的語言模型也是基於注意力機制的 Transformer 架構為基底。它的成功源於 2017 年 Google 提出的 "Attention Is All You Need"。這一重大突破促使 Google 團隊將 Transformer 架構中的 Encoder 抽離出來，創造了 Vision Transformer（ViT），用於影像分類技術。此外，ViT 放棄了 CNN 層，轉而使用自注意力機制進行計算，在分類問題上取得了優異的成績。



Dosovitskiy et al. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

> Vision Transformer 論文：An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale *(https://arxiv.org/abs/2010.11929)*
>
> 延伸閱讀：[論文導讀] Vision Transformer (ViT) 附程式碼實作 *(https://medium.com/@andy6804tw/%E8%AB%96%E6%96%87%E5%B0%8E%E8%AE%80-vision-transformer-vit-%E9%99%84%E7%A8%8B%E5%BC%8F%E7%A2%BC%E5%AF%A6%E4%BD%9C-379306ea2fb)*

在今天的範例實作中我們不會手刻整個 ViT 網路架構，也不會從頭自己訓練模型。而是使用非官方已訓練好的預訓練模型進行展示。首先要安裝 vit-keras *(https://github.com/faustomorales/vit-keras)* 套件，這是一個非官方的 Keras 版本實作，我們也可以拿它來做遷移學習。除此之外還需要安裝 `tensorflow_addons` 它是 TensorFlow 的一個附加庫，提供了一系列額外的自定義操作和層，以擴展 TensorFlow 的功能，因為實作 ViT 會需要用到像是 gelu 的激發函數。

```
pip install vit-keras
pip install tensorflow_addons
```

> 必須確保電腦已先安裝 TensorFlow2.0 以上

這段程式碼的主要目的是創建一個 Vision Transformer 模型，並獲取 ImageNet 分類的類別列表。該模型可以用於圖像分類等任務。其中我們是採用 `vit_b16` 的網路架構，b16 中的 16 表示著每個圖像的輸入都被分割成了固定大小的圖像塊（或稱為 "patches"），這些圖像塊的大小為 16x16 像素。這些圖像塊被用作模型的輸入，並通過自注意力機制來捕捉圖像中的全局訊息。

```
import numpy as np
import matplotlib.pyplot as plt
from vit_keras import vit, utils, visualize
```

```
# 使用vit函數創建Vision Transformer模型
image_size = 224 # 設定輸入圖像的大小為 224x224 像素
model = vit.vit_b16(
    image_size=image_size,
    activation='sigmoid', # 輸出使用 sigmoid 激發函數
    pretrained=True, # 使用預訓練權重
    include_top=True, # 包括頂部（分類層）
    pretrained_top=True # 使用預訓練的頂部權重
)
# 取得 ImageNet 分類的類別
classes = utils.get_imagenet_classes()
```

在ViT的研究和實現過程中，出現了多種不同的架構變形，以適應不同的任務和需求。以下是 `vit-keras` 套件中所提供的 ViT 架構變形：

- ViT Base Models：
  - ViT B16 （Vision Transformer Base with 16x16 patches）
  - ViT B32 （Vision Transformer Base with 32x32 patches）
- ViT Large Models：
  - ViT L16 （Vision Transformer Large with 16x16 patches）
  - ViT L32 （Vision Transformer Large with 32x32 patches）

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

`vit-keras` 套件目前尚未提供 ViT-Huge 模型的實現，因為這需要使用 `JFT: 300M images of 18K classes` 這個資料集進行訓練。此外這個資料集是由 Google 內部收集的，並且尚未釋出為開源資料集。

接下來實際載入一張圖像，並使用剛剛所建立的 ViT 模型進行預測，並輸出模型對圖像的預測結果。

```
url = 'https://upload.wikimedia.org/wikipedia/commons/b/bc/Free%21_%2
image = utils.read(url, image_size) # 載入圖片
x = np.expand_dims(image.copy(), axis=0) # 將圖像轉換為模型可接受的維度
x = vit.preprocess_inputs(x) # 預處理圖像
# 進行圖像分類預測
pred_proba = model.predict(x) # 返回分類機率
# 解析預測結果
pred_class = pred_proba[0].argmax() # 取得預測標籤索引
```

```
predicted_class_name = classes[pred_class] # 取得預測標籤名稱
print('Prediction:', predicted_class_name)
```

輸出結果：

```
Prediction: Eskimo dog, husky
```

最後使用論文中所提到的 `Attention Rollout` 方法來計算 ViT 模型中從 output token 到輸入圖像的注意力映射。簡單來說 Attention Rollout 就是計算從底層到高層的 Attention 矩陣的乘積。具體而言，Attention Rollout 的步驟包括：

1. 計算平均注意力權重： 首先計算模型中所有注意力頭的注意力權重的平均值，得到一個代表平均注意力的矩陣。
2. 遞迴相乘： 接下來，將這個平均注意力矩陣與模型的不同層次的注意力權重矩陣進行遞迴性相乘。這意味著將不同層次的注意力進行混合，以捕捉模型對輸入的綜合注意力分佈。
3. 得到最終注意力分佈： 最後這個遞迴相乘操作產生了最終的注意力分佈，描述了模型如何在不同層次上關注輸入數據的不同部分。

> Attention Rollout 詳細實作可以參考這個套件的原始程式 *(https://github.com/faustomorales/vit-keras/blob/de4c78c7f52f857af114f0d69312ee22946e4056/vit_keras/visualize.py#L7)*。

```
# 計算 Attention Rollout
attention_map = visualize.attention_map(model=model, image=image)
# 繪製結果
fig, (ax1, ax2) = plt.subplots(ncols=2)
ax1.axis('off')
ax2.axis('off')
ax1.set_title('Original')
ax2.set_title('Attention Map')
_ = ax1.imshow(image)
_ = ax2.imshow(attention_map)
```

Reference

- Explained: Attention Visualization with Attention Rollout *(https://storrs.io/attention-rollout/)*
- Intro to Transformers and Transformer Explainability *(https://www.youtube.com/watch?v=a0O_QhE9XFM)*

其他有用資訊 - Vision Transformer(ViT)重點筆記 *(https://hackmd.io/@YungHuiHsu/ByDHdxBS5)*
- Transformer可解釋性與視覺化 *(https://hackmd.io/SdKCrj2RTySHxLevJkIrZQ)*

Vision Transformers Need Registers - Meta 2023 Paper: https://arxiv.org/abs/2309.16588

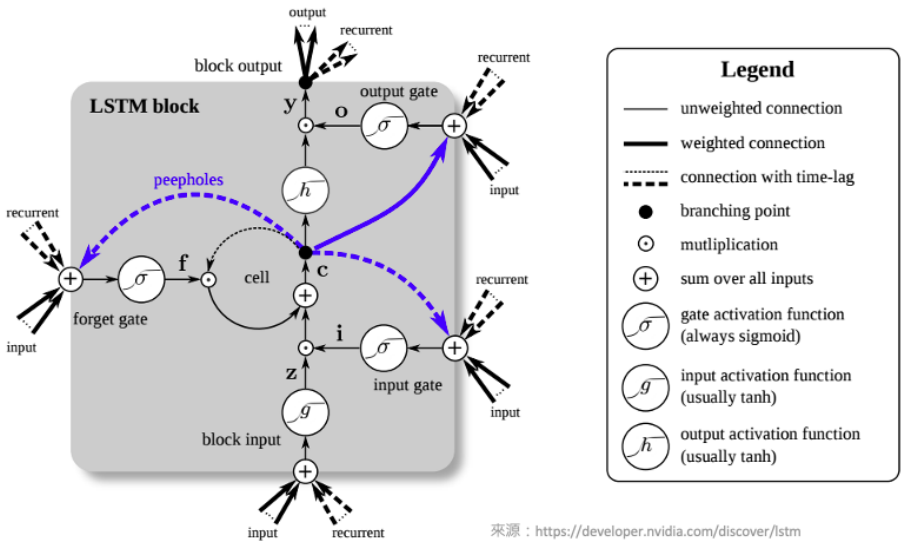
5.XAI在現實生活中的應用案例

[Day 24] LSTM的可解釋性：從時序資料解析人體姿態預測

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/24.LSTM的可解釋性：從時序資料解析人體姿態預測.ipynb)*

在這個系列中，我們已經介紹了深度神經網路中的DNN（深度神經網路）和CNN（卷積神經網路），以及它們如何透過不同的方法進行模型解釋。今天，我們將深入探討如何使用 Deep SHAP 方法來解釋長短期記憶網路（LSTM）模型。LSTM 是一種特殊類型的循環神經網路（RNN），用於處理時間序列資料以及具有長期依賴性的序列任務。LSTM 的主要特點是它能夠有效地捕捉和記憶過去的訊息，以便在處理未來的時間步時進行預測。

LSTM 之所以強大，是因為它具有以下關鍵結構和機制： - Cell State： LSTM具有記憶單元，可以存儲和檢索過去的訊息，這使得它能夠處理長序列資料，並保持對序列中先前訊息的適當記憶。 - Hidden State：LSTM還具有一個隱藏狀態，它是根據當前輸入和先前的隱藏狀態計算而來。隱藏狀態包含了當前時間步的信息，並用於生成預測。 - 三個控制門(Gate)：LSTM包含三個訊息控制單元的結構，分別是遺忘門（Forget Gate）、輸入門（Input Gate）和輸出門（Output Gate），通常會搭配Cell State和Hidden State控制輸出訊息。



來源：https://developer.nvidia.com/discover/lstm

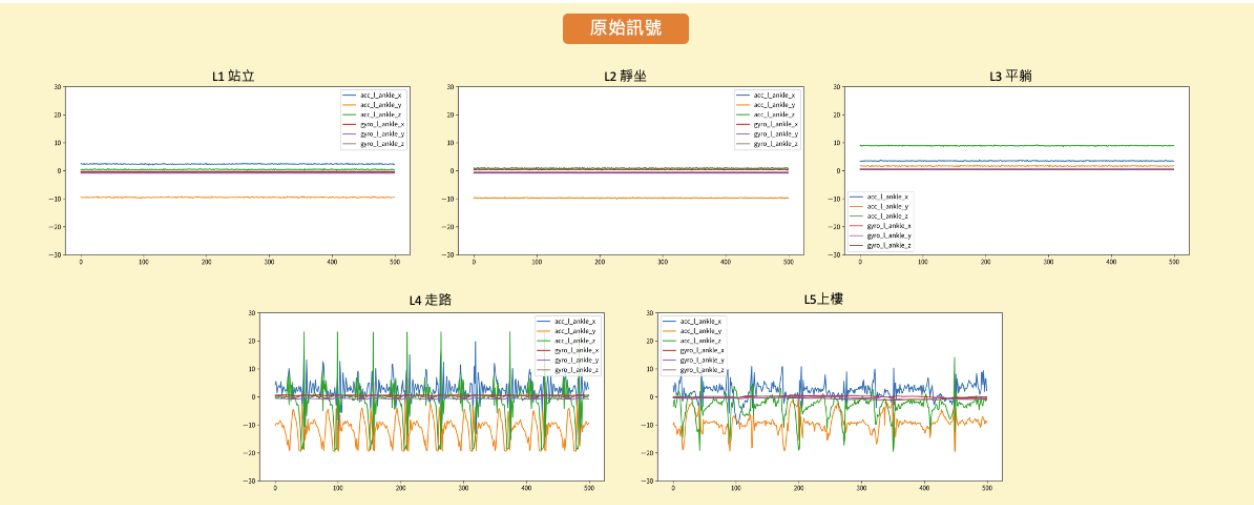
Deep SHAP 介紹可以參考：[Day 17] 解析深度神經網路：使用Deep SHAP進行模型解釋 *(https://ithelp.ithome.com.tw/articles/10331443)*

[實作] Mobile Health Human Behavior Analysis

在今日的範例中我們將採用 Mobile HEALTH *(http://archive.ics.uci.edu/dataset/319/mhealth+dataset)* 公開資料集，以建立 LSTM 時間序列模型，並用於預測人體動作辨識。該資料集包含了十名來自不同族群的志願者，在參與各項活動時，記錄了他們的外部感測器數值和生理訊號。接下來的實作中，我們僅使用了其中一位受試者的左腳踝三軸加速度（x、y、z）和陀螺儀（x、y、z）的資料，總共有六項特徵。這些資料以每秒50次的取樣率進行記錄(50Hz)，每種不同的動作分別持續了一分鐘，包括以下五種姿態：

- L1: 站立 (1 min)
- L2: 靜坐 (1 min)
- L3: 平躺 (1 min)
- L4: 走路 (1 min)
- L5: 上樓 (1 min)

從下圖中，我們可以觀察前500個資料點（約為10秒間隔）的訊號變化情況。在這段時間內，我們可以觀察到站立、靜坐和平躺等狀態的訊號變化相對較平穩，但仍可以通過旋轉角度和方位的變化來進行辨識。此外走路和上樓等動作則展現出明顯的訊號週期性，這些特徵可以被用來進行動作辨識。



原始資料單位：加速度(m/s^2), 陀螺儀角速度(deg/s)

載入資料集

首先我們透過 pandas 套件讀取一位受試者的訓練資料。這一份 csv 檔案中共有 161280 筆資料，其中包含數入訊號以及相對應標籤共有7個欄位。

- 輸入特徵
  - acc_l_ankle_x(左腳踝加速度x)
  - acc_l_ankle_y(左腳踝加速度y)
  - acc_l_ankle_z(左腳踝加速度z)
- 輸出標籤
  - Label 0~12 (今日範例只拿L1~L5)

```
import pandas as pd
df_data =  pd.read_csv('https://github.com/andy6804tw/crazyai-xai/raw
x_feature_names = ['acc_l_ankle_x','acc_l_ankle_y','acc_l_ankle_z','g
y_feature_name = ['label']
y_label_names = ['站立', '靜坐', '平躺', '走路', '上樓']
df_data = df_data[x_feature_names + y_feature_name]
```

輸入 輸出

	acc_l_ankle_x	acc_l_ankle_y	acc_l_ankle_z	gyro_l_ankle_x	gyro_l_ankle_y	gyro_l_ankle_z	label
0	2.1849	-9.6967	0.63077	0.103900	-0.84053	-0.68762	0
1	2.3876	-9.5080	0.68389	0.085343	-0.83865	-0.68369	0
2	2.4086	-9.5674	0.68113	0.085343	-0.83865	-0.68369	0
3	2.1814	-9.4301	0.55031	0.085343	-0.83865	-0.68369	0
4	2.4173	-9.3889	0.71098	0.085343	-0.83865	-0.68369	0
...	...	...	...	...	...	...	...
161275	2.1463	-9.2841	2.28640	-0.888680	-0.95497	0.88409	0
161276	2.0773	-9.5717	2.24010	-0.903530	-0.96998	0.86640	0
161277	1.7497	-9.3127	2.39560	-0.903530	-0.96998	0.86640	0
161278	1.8910	-9.3342	2.21890	-0.903530	-0.96998	0.86640	0
161279	2.3099	-9.2537	2.35210	-0.909090	-0.97186	0.86051	0

161280 rows × 7 columns

這一份資料集總共有 L0~L12 共13種標籤各自代表不同的姿態，其中在今天的範例中我們僅從資料集中提取 L1~L5 五種類別，依序分別代表站立、靜坐、平躺、走路、上樓。

```
L1 = df_data.loc[ df_data.label == 1] #L1: 站立 (1 min)
L2 = df_data.loc[ df_data.label == 2] #L2: 靜坐 (1 min)
L3 = df_data.loc[ df_data.label == 3] #L3: 平躺 (1 min)
L4 = df_data.loc[ df_data.label == 4] #L4: 走路 (1 min)
L5 = df_data.loc[ df_data.label == 5] #L5: 上樓 (1 min)
```

資料預處理

這段程式碼的主要目的是根據指定的窗口大小（window_size）和間隔（shift），從給定的時間序列資料中提取觀察資料（X）和相應的預測資料（y）。每個觀察資料是一個連續時間段內的資料，而預測資料則是該時間段後的一個時間點的特定特徵值。

```
# 抓取window_size的資料作為觀察資料(x)，預測下一時間點步態(y)
def window_data(data, window_size, shift):
    X = []
    y = []
    i = 0
    while (i + window_size) <= len(data) - 1:
        # 將連續的window_size時間段內的資料（去除最後一個特徵label）作為觀察資料x
        X.append(data[i:i+window_size, :-1])
        # 將接下來的一個時間點的第6個特徵（標籤）作為預測資料y
        y.append(data[i+window_size, 6])
        i += shift  # 移動索引，以繼續抓取下一筆資料
    X = np.array(X)
    y = np.array(y).reshape(-1)
    assert len(X) == len(y)  # 確保觀察資料和預測資料的數量一致
    return X, y  # 返回處理後的時序資料和預測資料
```

簡單來說以站立資料採樣為例，約1分鐘3072筆的資料，每筆時序資料取視窗大小100點資料，接著移動20點再另外收集。因此每筆訓練資料的輸入維度應該為 (100,6)，100代表 window_size 也就是要看幾筆的資料點，而 6 代表不同感測器所截取的數值左腳踝加速度(x,y,z)和陀螺儀(x,y,z)。我們分別將 L1~L5 資料依據時間點進行採樣組成多筆時序資料，最後再透過 np.concatenate() 將所有資料合併成 X 和 y。另外需注意的是在處理 y 的時候 -1 操作是為了將標籤從 1、2、3、4、5 轉換為 0、1、2、3、4，使標籤從0開始編號，因為在訓練神經網路的時候分類的標籤都是從0開始。

```
# 蒐集六個訊號100個時間點(window_size)，每筆資料採樣移動20個資料點
X_L1, y_L1 = window_data(L1.values, window_size=100, shift=20)
X_L2, y_L2 = window_data(L2.values, window_size=100, shift=20)
X_L3, y_L3 = window_data(L3.values, window_size=100, shift=20)
X_L4, y_L4 = window_data(L4.values, window_size=100, shift=20)
X_L5, y_L5 = window_data(L5.values, window_size=100, shift=20)

X = np.concatenate([X_L1, X_L2, X_L3, X_L4, X_L5])
y = np.concatenate([y_L1, y_L2, y_L3, y_L4, y_L5])-1 # 標籤要從0開始故全

print("X shape: ",X.shape)
print("y shape: ",y.shape)
```

輸出結果可以看到總共有 745 筆資料，window_size 為 100，每一筆時序資料共有 6 項特徵。

```
X shape:  (745, 100, 6)
y shape:  (745,)
```

資料準備就緒後最後一個步驟就是將 X 和 y 切割訓練集與測試集。

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
print("X_train shape = ",X_train.shape)
print("X_test shape = ",X_test.shape)
print("y_train shape = ",y_train.shape)
print("y_test shape = ",y_test.shape)
```

輸出結果：

```
X_train shape =  (670, 100, 6)
X_test shape =  (75, 100, 6)
y_train shape =  (670,)
y_test shape =  (75,)
```

LSTM 模型建立

以下程式碼使用 Tensorflow2.0 Functional API 搭建神經網路。此模型架構是用於處理時間序列資料並進行分類任務。模型的輸入是具有6個傳感器特徵的時間序列資料，每筆資料包含100個時間點。模型包括一個 LSTM 隱藏層，用於捕捉時間序列的時間相關性，其中 `return_sequences=True` 並將每個神經元的隱藏狀態(hidden_state)回傳，並透過 `Flatten()` 將所有時間的隱藏狀態攤平成一維向量傳給輸出層。最後通過一個全連接層進行分類，將輸入資料分為5個不同的類別。

```
from tensorflow.keras.models import Model
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.compat.v1.keras.backend import get_session
tf.compat.v1.disable_v2_behavior()

num_sensor = 6
window_size = 100
def build_model():
    model_input = layers.Input(shape=(window_size, num_sensor))
    # 第一層隱藏層
    x = layers.LSTM(2, activation='relu', return_sequences=True, retu
    x = layers.Flatten()(x)
    # 輸出層
```

```
    model_output = layers.Dense(5, activation='softmax')(x)
    return Model(model_input, model_output)
```

由於 SHAP 目前還尚未支援 TF2.4 版本以上，因此必須透過 `tf.compat.v1.disable_v2_behavior()` 關閉一些 2.0 版本的進階 API。

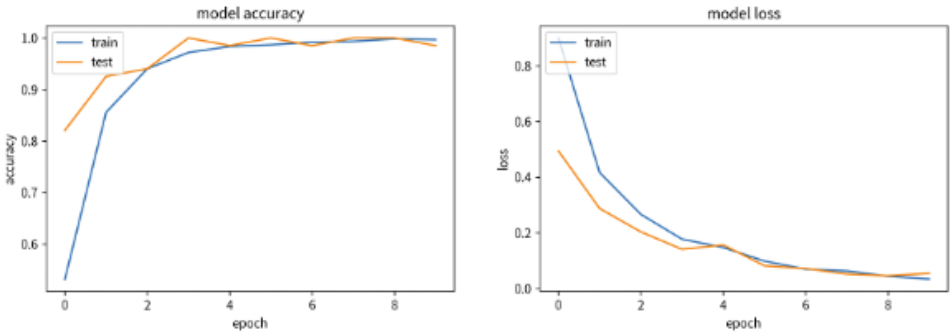
接下來，使用先前定義的 build_model() 函數建立一個新的神經網絡模型，並將這個模型存儲在 model 變數中。最後使用 model.summary() 印出模型的摘要訊息，包括模型的結構、每一層的參數數量等。

```
tf.keras.backend.clear_session()
model = build_model()
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 100, 6)]          0

 lstm (LSTM)                 (None, 100, 2)            72

 flatten (Flatten)           (None, 200)               0

 dense (Dense)               (None, 5)                 1005

=================================================================
Total params: 1,077
Trainable params: 1,077
Non-trainable params: 0
```

模型準備就緒後即可開始訓練模型。由於我們沒有針對輸出標籤進行 one-hot encoding 因此可以在 loss 指定 sparse_categorical_crossentropy，這樣在模型訓練過程中自動地會進行額外處理以利於 cross entropy 的計算。由於訓練資料沒有很多，因此批次大小設為 4 表示每次訓練過程中，模型會同時處理4筆訓練資料。訓練迭代次數為 10 次。最後使用訓練數據 X_train 和 y_train 來訓練模型。

```
# 編譯模型
model.compile(
      loss='sparse_categorical_crossentropy',
      optimizer='adam',
      metrics=['acc'])
# 訓練模型
history = model.fit( X_train, y_train,
                     validation_split=0.1,
                     batch_size=4,
                     epochs=10)
```



Deep SHAP 解釋 LSTM 模型

這裡採用 SHAP 套件中的 Deep SHAP 進行深度神經網路模型的解釋，它結合了 DeepLIFT 和 Shapely values 的概念，以計算每個特徵對於模型預測的貢獻。首先建立一個 DeepExplainer 解釋器，他除了 DNN 模型可以解釋之外，其他類型的神經網路像是 LSTM、CNN、1DCNN 都可以透過它來對模型進行特徵歸因的重要性解釋。另外在估計 Shapely values 時，可以輸入要解釋的資料。在這個範例中，我們將使用75筆測試集資料進行模型解釋。

```
import shap
shap.initjs()

# 使用 Deep SHAP 解釋模型
explainer = shap.DeepExplainer(model=model, data=X_train)
# 估計 Shapely values
shap_values = explainer.shap_values(X_test)
```

由於我們需要處理大量的測試集資料，所以輸入資料的維度為(75, 100, 6)。這裡的每個數值代表的含義分別是(資料筆數, 時間窗口大小, 特徵數)。最後我們可以觀察到，在處理時間序列資料計算 Shapely values 時，輸出的維度將是(5, 75, 100, 6)。這是因為這個模型是一個輸出五類別機率的分類任務，所以在計算 Shapely values 時，會針對每一筆資料分別計算五個類別對應的 Shapely values。

```
# 5個類別, 75筆測試資料, 100個時間點, 6個特徵
np.array(shap_values).shape
```

輸出結果：

```
(5, 75, 100, 6)
```

SHAP Summary Plot (全局解釋)

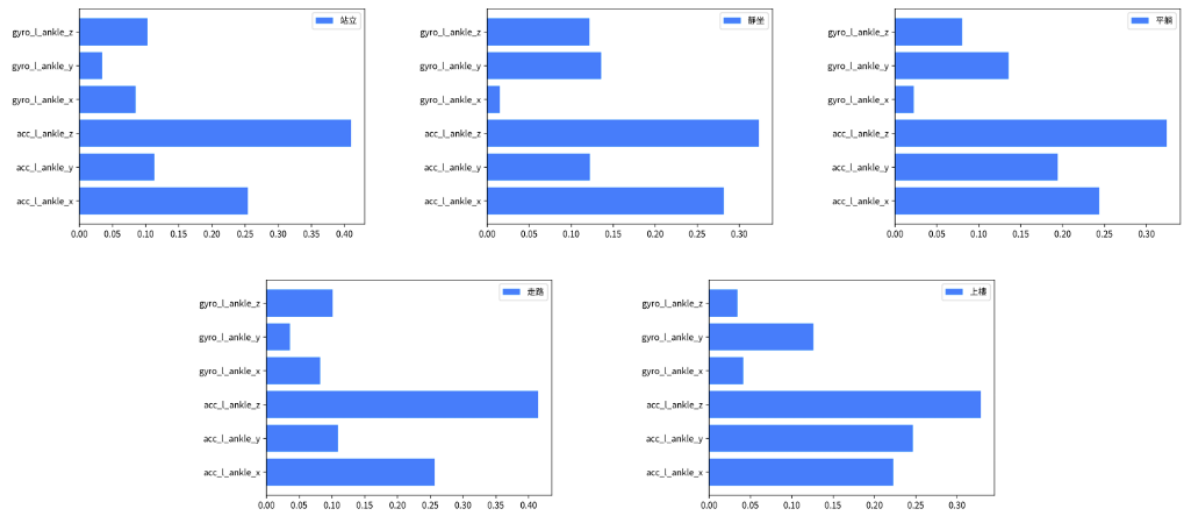
我們可以針對75筆測試資料進行每個類別的特徵重要程度排序。接著透過變數 `label` 的設定可以觀察在某標籤下每個特徵對於整體平均貢獻的值。由於 Shapely values 會針對窗口的大小進

行每個時間點的特徵歸因計算。因此每筆資料在某個類別的 Shapely values 維度是二維的(100, 6)，所以在進行全局模型解釋前要手動的將每筆測試資料的時間點根據每個特徵維度進行相加得到每個特徵的貢獻程度。最後計算出來六個數值再通過一個標準化，得到在某標籤下每個特徵對於整體平均貢獻的值。

```
def normalize_array(arr):
    # 計算陣列中所有元素的總和
    total = sum(arr)
    # 正規化每個元素
    normalized_arr = [x / total for x in arr]
    return normalized_arr
```

```
# 獲得在某標籤下每個特徵對於整體平均貢獻的值
# 0:站立、1:靜坐、2:平躺、3:走路、4:上樓
label=0
shap_value = np.array(shap_values)
shap_value = np.absolute(shap_value[label])
shap_value = np.sum(shap_value, axis=1)
SHAP_list = [np.sum(shap_value[:, 0]), np.sum(shap_value[:, 1]), np.s
plt.barh(x_feature_names, normalize_array(SHAP_list), label=y_label_n
plt.legend()
plt.show()
```

我們可以參照文章一開始觀察原始訊號圖，並發現各種步態可以很清楚地透過加速度規訊號進行判斷。因此在模型的推論過程中，主要依賴於加速度資訊，具體來說是 acc_x、acc_y 和 acc_z 這三項資訊。果然在 SHAP 模型解釋中，判斷各種類別的重要性都是以那三項資訊為主要依據。另外需注意的是，在本範例中，我們尚未對所有資料進行標準化，因此各位讀者也可以嘗試對資料進行標準化，看看是否會對模型的預測結果產生不同的解釋結果。



SHAP Force plot (單筆資料解釋)

我們從資料集中選取了75筆資料作為測試集。剛才提到的全局解釋是針對這75筆資料的整體平均進行的解釋。現在我們可以進一步針對每一筆數據進行解釋分析。在下面程式中的 `index` 被設定為0，這表示我們要觀察測試集中的第一筆資料。然後，我們使用force_plot函式對這筆資料進行預測，並將分析結果以視覺化方式呈現。

```
# 觀察測試集中第一筆資料的重要程度
index=0
pred_class = model.predict(X_test[[index]]).argmax()
pred_proba = model.predict(X_test[[index]])[0][pred_class]
print(f'測試集第 {index+1} 筆模型預測結果: {pred_class} 機率值: {pred_proba}'
print(f'真實答案: {int(y_test[index])}')
shap_value = shap_values[pred_class][index]
shap_value = shap_value.sum(axis=0)
shap.force_plot(explainer.expected_value[pred_class], shap_value, fea
```

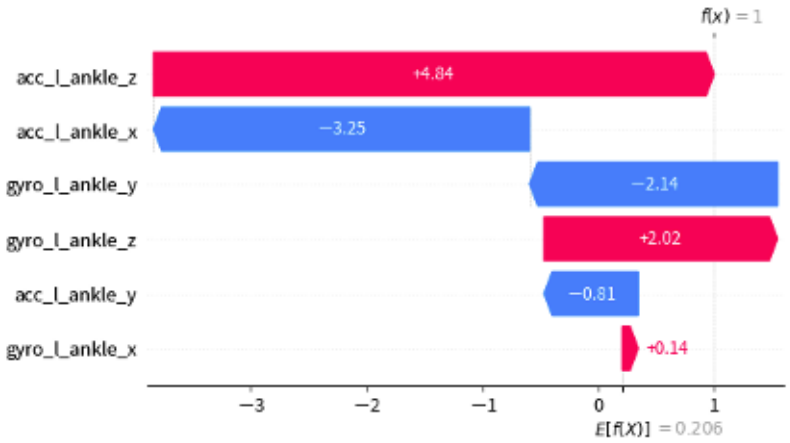
從下圖視覺化解釋結果可以看到模型在這一筆資料預測標籤 1（靜坐），其機率值為 0.99。紅色的特徵表示將正向的影響輸出機率，會使值升高，藍色的則相反，會用來降低輸出機率。

SHAP waterfall plot (單筆資料解釋)

我們可以更近一步的觀察該筆資料每個特徵對於輸出某個類別的 Shapely values 數值，以及判斷該類別的基準值是多少。下圖中每個特徵相對應的 Shapely value 累加，並加上基準值 `E[f(x)]=0.206` 最終相加的結果就是該筆測試資料預測某類的的機率值。

```
pred_class = model.predict(X_test[[index]]).argmax()
pred_proba = model.predict(X_test[[index]])[0][pred_class]
shap_value = np.array(shap_values)
shap_value = np.expand_dims((shap_value[pred_class])[index], axis=0)
shap_value = np.sum(shap_value, axis=1)[0]
shap.waterfall_plot(shap.Explanation(values=shap_value,
                                     base_values=explainer.expected_va
                                     feature_names=x_feature_names))
```

下圖範例是預測第一筆測試資料在標籤為1的時候的 Shapely values 數值。



加總結果： 0.206-3.245-0.815+4.84+0.138-2.142+2.017=0.99

小結

個人目前觀察 SHAP 套件中的 Deep SHAP 在官方的例子中是 LSTM 自然語言的例子為主。然而在 LSTM 模型於分類或迴歸的模型，SHAP 套件無法直接進行解釋，必須透過小小的前處理透過特徵的維度將每個時間因子的 Shapely values 先加總再送進去 `force_plot` 進行解釋。此外目前套件尚未針對最新版 TensorFlow 進行優化，所以使用上會受到一些限制。如果對時間序列模型解釋有需求的讀者，也可以參考 TimeSHAP: Explaining Recurrent Models through Sequence Perturbations *(https://arxiv.org/abs/2012.00073)* 這篇研究，以及相關的實作 TimeSHAP *(https://github.com/feedzai/timeshap)* 這些資源或許會更適合處理時間序列模型的解釋需求。

Reference

- [forecast][LSTM+SHAP]Applied SHAP on the polynomial equation case with LSTM algorithm *(https://medium.com/@sakamoto2000.kim/applied-shap-on-the-polynomial-equation-case-with-lstm-algorithm-7c140d15736b)*

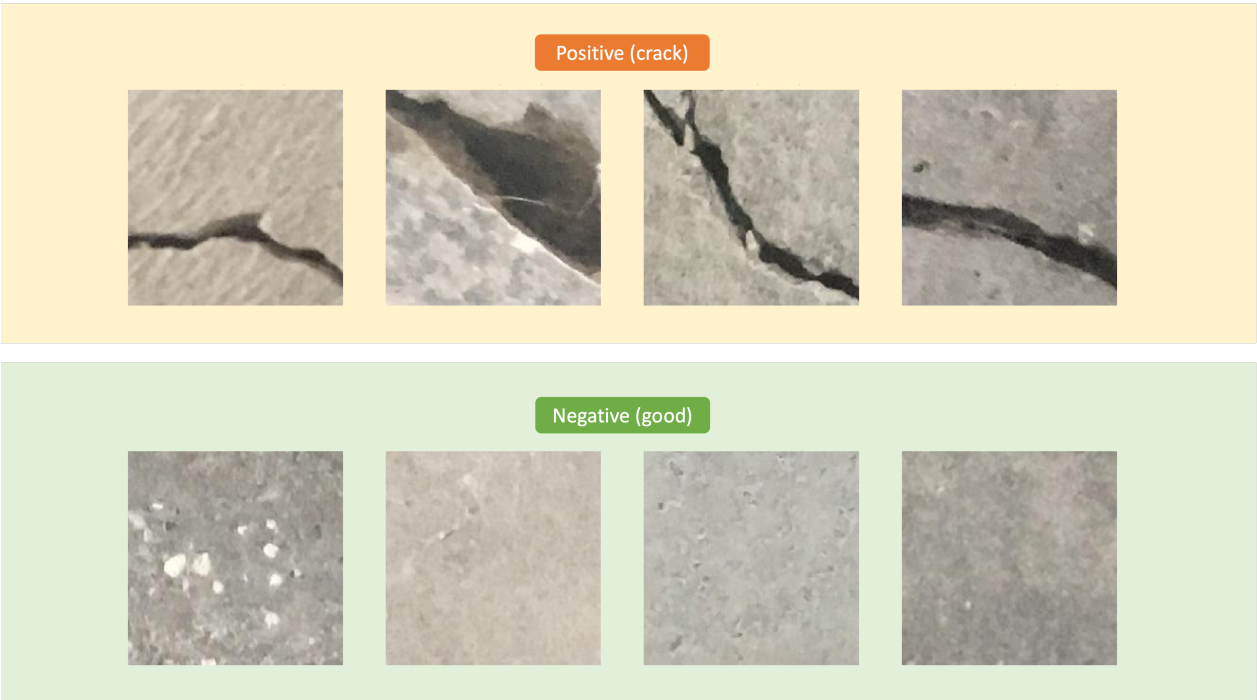
[Day 25] XAI在影像處理中的瑕疵檢測：解釋卷積神經網路的運作

範例程式：Open in Colab *(https://www.kaggle.com/code/andy6804tw/day-25-xai)*

隨著鐵人賽進入尾聲，相信各位已經對可解釋人工智慧（XAI）領域有了一些初步的了解。在接下來的幾天中，我想透過一些實際的例子來介紹 XAI 的實際應用，藉此展示如何透過人工智慧來解決企業中的機器學習問題。今天要介紹的例子涉及影像辨識中的瑕疵檢測。在影像處理中，瑕疵檢測是一個相當重要的應用領域，通常用於檢查製造過程中的產品或材料，以確保它們的品質達到標準要求。這個應用領域在許多不同的行業中都具有關鍵的地位，包括製造業、醫療、石化產業等等。

[案例] 表面裂紋檢測

Surface Crack Detection Dataset *(https://www.kaggle.com/datasets/arunrk7/surface-crack-detection?select=Positive)* 是一個用於混凝土表面裂縫檢測的資料集。這個資料集被分成兩個類別，分別是正面（有裂縫）和負面（無裂縫）。這兩類圖像被存儲在不同的資料夾中，以便進行圖像分類。每個類別包含20000張圖像，總共有40000張圖像，每張圖像的解析度為227 x 227像素，並且是彩色圖像。



載入預訓練模型(ResNet50)

在今天的實作中，我們將使用預訓練模型（ResNet50）來執行遷移學習（Transfer Learning），這是一種應用在分類任務上的學習方法。遷移學習是指，我們將先前訓練好的模型（通常是在大型數據集上訓練的模型）應用於新的任務或資料集，以提高模型的性能和準確度。這種方法通常能夠節省訓練新模型所需的大量時間和資源，同時能夠更快地達到良好的結果。首先使用 TensorFlow 建立了一個預訓練的 ResNet50 模型，並且將預訓練模型的權重設為不可訓練，並保留原始的特徵提取功能。由於 include_top 設定為 False，因此必須自己建立輸出的隱藏層與輸出層。因此我們自己建立了 GlobalAveragePooling2D 層將特徵圖的每個通道的平均值計算為一個數值。接著在模型的輸出層添加一個具有 sigmoid 激發函數的全連接層，用於二元分類。最後模型編譯，設定了優化器、損失函數和評估指標。

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras.applications.resnet import ResNet50, preprocess
from tensorflow.keras.optimizers import Adam

img_size = 150
model_name = 'ResNet50'

# 使用預訓練的ResNet50模型
pre_model = ResNet50(weights='imagenet', include_top=False,
                     input_shape=(img_size, img_size, 3))

# 將預訓練模型的權重設為不可訓練
pre_model.trainable = False
# 隱藏層
x = layers.GlobalAveragePooling2D()(pre_model.output)
# 輸出層
outputs = layers.Dense(1, activation='sigmoid')(x)
model = Model(inputs=pre_model.inputs, outputs=outputs)
# 設定優化器
optimizer = Adam(learning_rate=0.001)
# 編譯模型
model.compile(loss='binary_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
```

載入圖片

模型搭建完成後，接著建立用於訓練和驗證的圖像資料生成器（ImageDataGenerator）。訓練資料生成器進行了資料增強（Data Augmentation）操作，包括隨機位移、水平翻轉等，並使用

指定的圖像前處理函式。然而在驗證集的部分不需要進行資料增強，僅需要附上圖向前處理的函式即可。

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# 建立訓練資料生成器
train_datagen = ImageDataGenerator(
                width_shift_range=0.1,  # 隨機水平位移圖像的範圍
                height_shift_range=0.1,  # 隨機垂直位移圖像的範圍
                horizontal_flip=True,  # 隨機水平翻轉圖像
                preprocessing_function=preprocess_input,  # 圖像前處理函數
                validation_split=0.01  # 驗證資料集的比例
               )

# 建立驗證資料生成器
validation_datagen = ImageDataGenerator(preprocessing_function=prepro
```

這些生成器從指定的資料夾中讀取圖像，並生成用於模型訓練和驗證的批次圖像數據。它們被設置為進行二元分類，其中類別標籤是正和負，並根據比例從原始資料中選擇訓練和驗證的子集。

```
# 設定圖像的尺寸
img_shape = (img_size, img_size)

# 創建訓練資料生成器，從指定資料夾中讀取圖像，並進行資料增強
train_generator = train_datagen.flow_from_directory(
                  '../input/surface-crack-detection',
                  target_size=(img_size, img_size),  # 調整圖像尺寸
                  batch_size=64,  # 批次大小
                  shuffle=True,  # 隨機打亂圖像順序
                  class_mode='binary',  # 二元分類
                  subset='training')  # 設定為訓練子集

# 創建驗證資料生成器，從指定資料夾中讀取圖像
validation_generator =  validation_datagen.flow_from_directory(
                        '../input/surface-crack-detection',
                        target_size=(img_size, img_size),  # 調整圖像尺
                        batch_size=64,  # 批次大小
                        class_mode='binary',  # 二元分類
                        subset='validation')  # 設定為驗證子集
```

訓練模型

一切已經準備就緒，現在可以開始訓練模型了。因為我們是基於預訓練模型進行分類遷移訓練，所以我們只需要設定訓練的迭代次數為3，這樣可以快速達到良好的準確度並完成模型訓練。

```
# 訓練模型
model.fit(train_generator,
          epochs=3,
          validation_data=validation_generator)
```

```
Epoch 1/3
619/619 [==============================] - 488s 769ms/step - loss: 0.0268 - accuracy: 0.9940 - val_loss: 0.0029 - val_accuracy: 1.0000
Epoch 2/3
619/619 [==============================] - 265s 428ms/step - loss: 0.0063 - accuracy: 0.9984 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 3/3
619/619 [==============================] - 264s 426ms/step - loss: 0.0051 - accuracy: 0.9985 - val_loss: 7.1635e-04 - val_accuracy: 1.0000
```

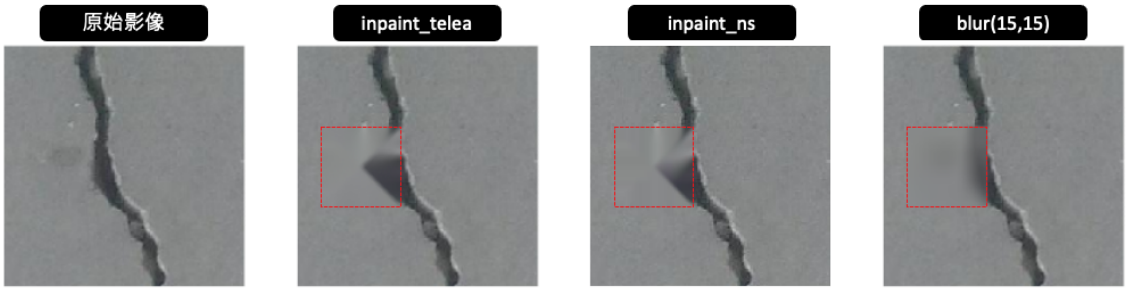
SHAP 解釋卷積神經網路的運作

本系列中已介紹了許多 CNN 的解釋方法，如 Perturbation-Based、Gradient-Based、Propagation-Based、CAM-Based、Attention-Based 等。這裡我們採用 SHAP 套件進行 CNN 模型的解釋，然而 SHAP 提供許多種不同的解釋器，這次選用 Partition SHAP 方法，搭配圖像遮擋技術進行卷積神經網路的解釋。首先我們從資料集中載入一張具有瑕疵裂痕的影像，這將成為我們要解釋的圖片。接著使用 ResNet50 模型的預處理函數 `preprocess_input()` 來處理這張圖像，以確保圖像的數值範圍和格式符合模型的要求。

```
import numpy as np

# 載入圖像
image = tf.keras.utils.load_img('../input/surface-crack-detection/Pos
image = tf.keras.utils.img_to_array(image) # 將載入的圖像轉換為數組形式
x = np.expand_dims(image.copy(), axis=0) # 將圖像轉換為模型可接受的維度
```

SHAP Partition Explainer

Partition Explainer 是 SHAP 套件中的一種方法，用於解釋機器學習模型。針對 CNN 模型的解釋，它可以用於分析圖像分類模型的決策。例如對於一張圖像，我們可以使用 Partition Explainer 來評估不同區域中的像素對於模型的分類結果的影響。其背後的技術是透過 blur 或 inpaint 技術遮蓋圖像中的部分區塊，以查看圖像的哪些部分對預測有貢獻。



首先定義了一個函數f(X)該函數複製輸入的圖像X，然後進行預處理，最後返回模型的輸出。接著定義了一個 masker，用於遮蓋輸入圖像的部分區域。這個 masker 提供了三種方法來遮蓋圖像的部分區域，分別為有：

- inpaint_telea： 使用了 Telea 算法嘗試根據周圍的像素值快速且有效地填補遮罩區域。
- inpaint_ns：使用了 Navier-Stokes 算法以填補圖像中的遮罩區域，適合用於複雜紋理。
- blur(kernel_xsize,kernel_xsize)： 假設使用 blur(15, 15)，則表示核的大小為15x15，它會考慮每個像素周圍的鄰域，將這些像素的值進行平均，以達到模糊化的效果。

```
import shap

# 包裝要被解釋的模型
def f(X):
    tmp = X.copy()
    preprocess_input(tmp) # 影像前處理
    return model(tmp)

# 定義一個 masker 用於遮罩圖像的部分區域
masker = shap.maskers.Image("inpaint_telea", x[0].shape)
```

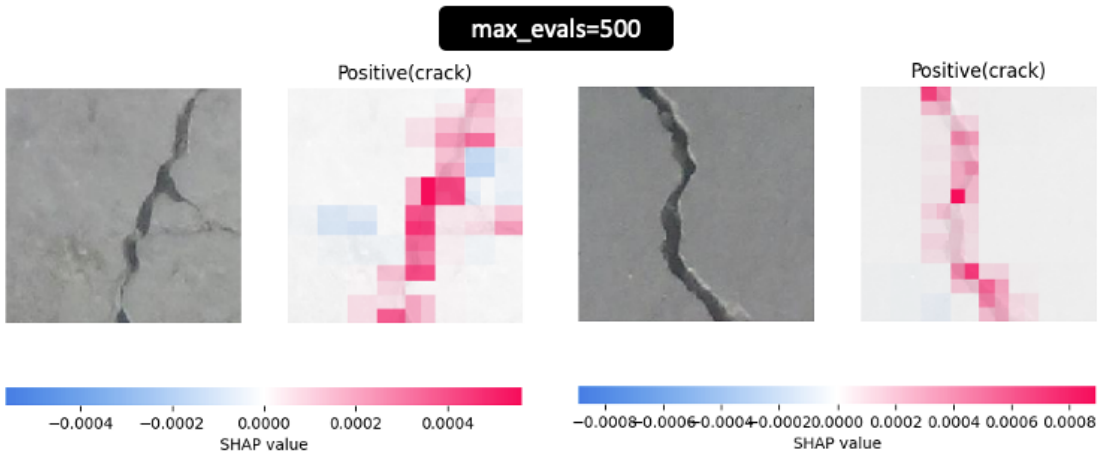
然後我們使用 Explainer() 函數來創建一個 Partition Explainer 的實例。在該函數中，必須輸入三個重要的參數值：

- model: 要被解釋的模型，即範例程式中的 f(X)
- mask: 於使用blur或inpaint技術遮蓋圖像中的部分區塊
- output_names: 目標類別標籤名稱

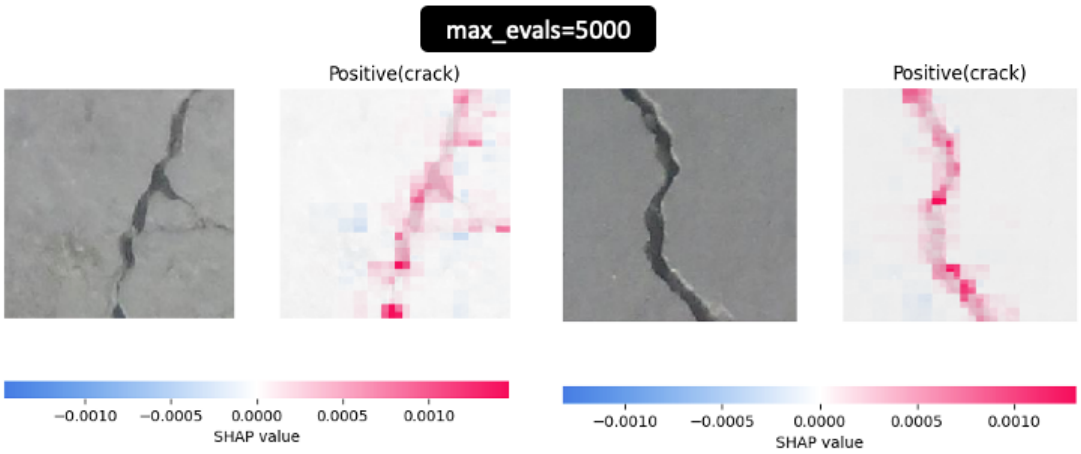
```
# 使用 Partition explainer 解釋模型
explainer = shap.Explainer(f, masker, output_names=["Positive(crack)"
# 估計 Shapely values
shap_values = explainer(x, max_evals=500, batch_size=50, outputs=shap
```

我們將要被解釋的圖片放入 SHAP 解釋器後，即可針對計算的結果視覺化，並根據 Shapely values 計算結果查看在預測類別 Positive(crack) 情況下模型所關注的區域。從下圖可以觀察到在不同的影像中紅色的特徵表示將正向的影響輸出機率，會使值升高，藍色的則相反，會用來降低輸出機率。

```
shap.image_plot(shap_values, x.astype(np.uint8))
```



另外在計算 Shapely values 的時候 `max_evals` 參數代表在估算 SHAP 值時的最大評估次數。較大的值可能會提供更準確的 SHAP 值，但也會增加計算時間。下圖是將數值設為 5000 的結果，從結果的解釋圖中可以觀察到更詳細的解釋效果。



大家也可以試試看其它不同的遮蓋圖像處理方法對於解釋效果有什麼不同的影響。

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/crazyai-xai)* 取得！

Reference

- SHAP Document: Multi-class ResNet50 on ImageNet (TensorFlow) *(https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/image.html)*
- Surface Crack Detection Dataset *(https://www.kaggle.com/datasets/arunrk7/surface-crack-detection?select=Positive)*

[Day 26] XAI在表格型資料的應用：解析智慧工廠中的鋼材缺陷

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/26.XAI在表格型資料的應用：解析智慧工廠中的鋼材缺陷.ipynb)*

在當今的工業領域中，智慧製造、碳中和以及數位雙生等議題受到廣泛關注。其中機器學習技術已經開始發揮關鍵作用，特別是在虛擬量測和異常檢測方面。在今天的內容中將帶各位深入探討工業應用實戰案例，即基於機器學習演算法的鋼材缺陷偵測分類 *(https://archive.ics.uci.edu/dataset/198/steel+plates+faults)*。

[案例] 鋼鐵缺陷分類

本案例使用的資料集來自 UCI（加州大學爾灣分校）的開放數據平台，這個平台致力於為機器學習研究者和實踐者提供高品質的資料集。該資料集涵蓋了鋼材製造過程中可能出現的多種缺陷情況，其中包含了7種帶鋼缺陷類型，具體分別是 Pastry、Z_Scratch、K_Scatch、Stains、Dirtiness、Bumps、Other_Faults。另外這個資料集是一個表格型的資料集，特別適用於機器學習和數據分析。它包含了27種不同的特徵，這些特徵描述了帶鋼的不同屬性和特點，如缺陷的大小、形狀、位置等等。這些數據都是在製造或生產過程中收集的，每一筆資料都有相對應的標籤，即缺陷種類。

輸入 輸出

	X_Minimum	X_Maximum	...	Orientation_Index	Luminosity_Index	SigmoidOfAreas	Pastry	Z_Scratch	K_Scatch	Stains	Dirtiness	Bumps	Other_Faults
0	42	50	...	0.8182	-0.2913	0.5822	1	0	0	0	0	0	0
1	645	651	...	0.7931	-0.1756	0.2984	1	0	0	0	0	0	0
2	829	835	...	0.6667	-0.1228	0.2150	1	0	0	0	0	0	0
3	853	860	...	0.8444	-0.1568	0.5212	1	0	0	0	0	0	0
4	1289	1306	...	0.9338	-0.1992	1.0000	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1939	249	277	...	-0.4286	0.0026	0.7254	0	0	0	0	0	0	1
1940	144	175	...	-0.4516	-0.0582	0.8173	0	0	0	0	0	0	1

1941 rows × 34 columns

表格型資料通常以逗號分隔檔(csv)呈現，每一列(row)代表一個觀測值或樣本，而每一行(col)代表一個特徵或標籤。在這個特定的資料集中，每一列可能代表一個時間點或生產批次，而每一行代表一組製程參數以及對應的標籤種類。這份資料集比較特別的是輸出的標籤是以 one hot encoding 的方式呈現。在等等的實作中我們必須將這些資料進行前處理，並取得相對應的標籤索引。

載入資料集

首先透過 pandas 載入事先準備好的資料集。將 csv 檔案中的數據讀取並存儲在名為 df_data 的 DataFrame 變數中，以便後續的數據分析和處理。

```
import numpy as np
import pandas as pd

url = 'https://raw.githubusercontent.com/andy6804tw/crazyai-xai/main/
df_data = pd.read_csv(url)
```

接著我們必須從 df_data 分離輸入特徵 X 與標籤 y。首先從資料集中提取出特徵欄位的名稱，然後將這些特徵的數據提取出來，存儲在變數 X 中。接著從資料中提取了包含標籤的 one hot encoding 數據，然後使用 `argmax()` 找到每個 one hot 向量中的最大值索引，將其視為對應的標籤，最終將這些標籤存儲在變數 y_labels 中。

```
x_feature_names = df_data.columns[:-7].values # 取得特徵欄位名稱
X = df_data[x_feature_names].values # 取出訓練資料特徵
y_label_names = df_data.columns[-7:].values # 取得標籤欄位名稱
y_one_hot_array = df_data[y_label_names].values # 取出標籤
# 使用argmax函數找到每個one hot向量中的最大值索引，並將其視為對應的標籤
y_labels = y_one_hot_array.argmax(axis=1)

print(f'The shape of X: {X.shape}')
print(f'The shape of y_labels: {y_labels.shape}')
```

從上面的輸出結果可以知道資料集總共有1941筆，每筆資料總共有27個特徵。

```
The shape of X: (1941, 27)
The shape of y_labels: (1941,)
```

切割訓練集與測試集

這裡使用 sklearn 套件中的 `train_test_split()` 從原始資料集中切分出訓練集和測試集。test_size 參數設置0.1即代表從資料集1941筆中切1%比例作為測試集，random_state 確保每次運行結果相同，stratify 參數根據 y_labels 的類別分佈來確保訓練集和測試集中類別的分佈比例相似。

```
from sklearn.model_selection import train_test_split

# 切分資料集為訓練集和測試集
X_train, X_test, y_train, y_test = train_test_split(X, y_labels,test_

# 觀察切割後資料的維度與筆數
```

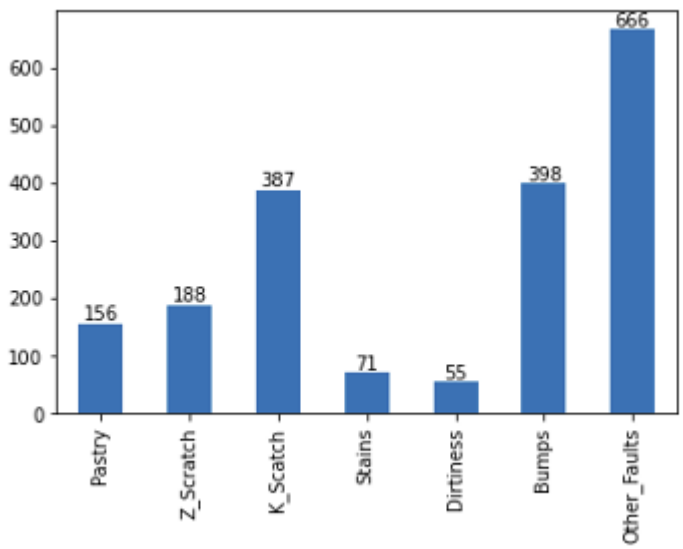
```
print(f'The shape of X_train: {X_train.shape}\t y_train: {y_train.sha
print(f'The shape of X_test: {X_test.shape}\t y_test: {y_test.shape}'
```

輸出結果：

```
The shape of X_train: (1746, 27)     y_train: (1746,)
The shape of X_test: (195, 27)       y_test: (195,)
```

資料集切割後我們觀察再訓練集中每個類別缺陷的數量分佈情況，可以透過 pandas 的 value_counts() 方法迅速計算每個不同類別的出現次數。通常，這個函數用於統計和分析資料中的類別型變數，以便了解每個類別的分佈情況。從下圖的結果可以觀察到，每個缺陷類別的數量分佈不均勻，其中 Other_Faults 類別的樣本數最多，共有666筆。相較之下 Stains 和 Dirtiness 類別的樣本數都不到100筆。這顯示了這個資料集存在明顯的標籤不平衡問題。

```
import matplotlib.pyplot as plt

# 查看七種類別筆數
label_counts = pd.Series(y_train).value_counts(sort=False)
fig = label_counts.plot(kind='bar')
fig.set_xticklabels(label_counts)
fig.set_xticklabels(y_label_names)
# 在每個bar上方顯示數值
for index, value in enumerate(label_counts):
    plt.text(index, value, str(value), ha='center', va='bottom')
plt.show()
```



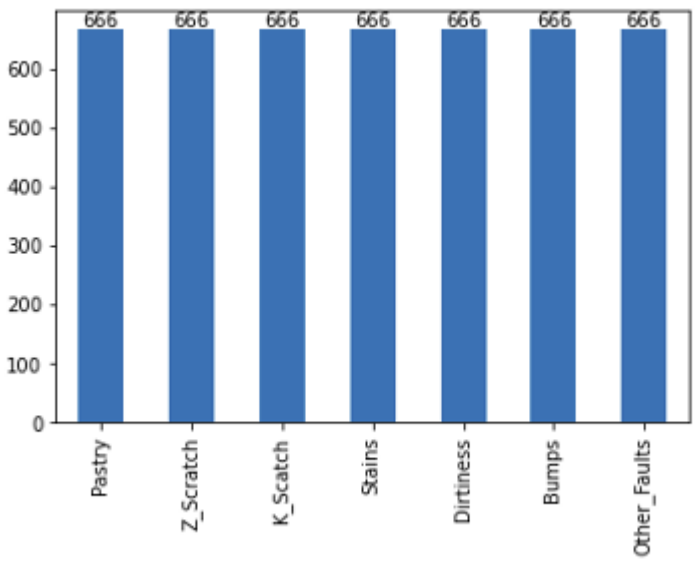
SMOTE處理標籤不平衡問題

SMOTE（Synthetic Minority Over-sampling Technique）是一種用於處理標籤不平衡問題的技術，它的主要目標是合成新的少數類樣本，以平衡不同類別之間的數量差距。首先要安裝 `imbalanced-learn` 套件，如果尚未安裝，可以使用以下指令安裝：

```
pip install imbalanced-learn
```

SMOTE 方法採用過取樣（Oversampling）技術，以合成新的樣本，以實現不同類別之間的平衡。從下圖的採樣結果可以看出，每個類別均有666筆數據。

```
from imblearn.over_sampling import SMOTE

smo = SMOTE(sampling_strategy='auto',random_state=42)
X_smo, y_smo = smo.fit_resample(X, y_labels)
```



建立LightGBM模型

LightGBM 是輕量化 (Light) 的梯度提升機 (GBM) 的實例。其相對 XGBoost 來說它具有訓練速度快、記憶體佔用低的特點，因此近幾年 LightGBM 在 Kaggle 上也算是熱門模型一。在本飯範例中我們採用 LightGBM 分類器，若還沒安裝的讀者可以參考以下指令進行安裝。

```
pip install lightgbm
```

安裝結束後即可載入 lightgbm 套件並選用 LGBMClassifier 分類器進行模型訓練。

```
import lightgbm as lgb

# 建立模型
```

```
model = lgb.LGBMClassifier()
# 訓練模型
model.fit(X_train,y_train)
```

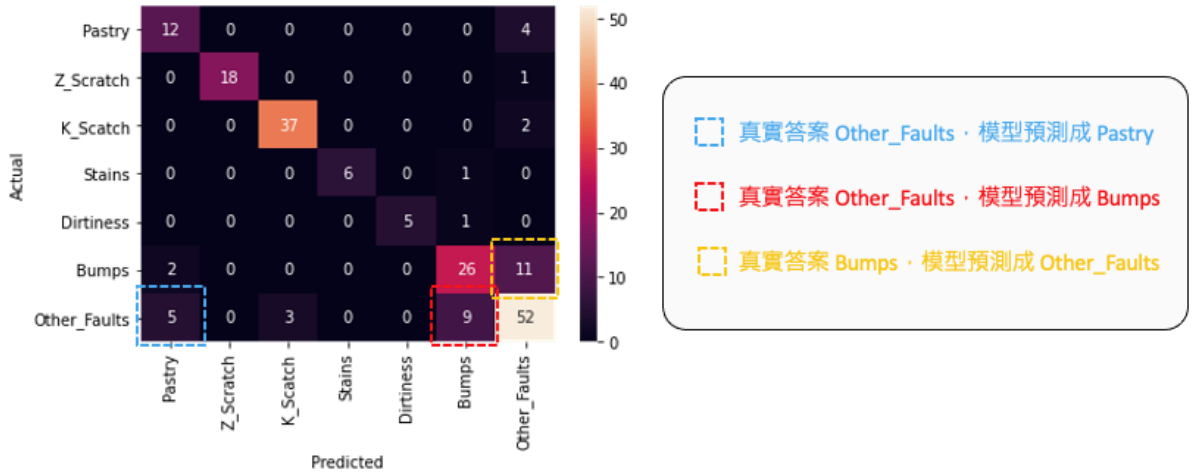
訓練結束後，我們可以透過 sklearn 的 classification_report() 方法來查看模型在測試集上的分類報告，該報告包含了模型的精確度、召回率、F1分數等評估指標，可以用來評估模型的性能。

```
from sklearn.metrics import classification_report


pred_test = model.predict(X_test)
print(classification_report(y_test, pred_test))
```

從評估報告中我們可以看到測試集總共有195筆，其中0: Pastry、5: Bumps和6: Other_Faults兩個的辨識率約七成。其餘的瑕疵類別都表現得不錯。

```
              precision    recall  f1-score   support

           0       0.63      0.75      0.69        16
           1       1.00      0.95      0.97        19
           2       0.93      0.95      0.94        39
           3       1.00      0.86      0.92         7
           4       1.00      0.83      0.91         6
           5       0.70      0.67      0.68        39
           6       0.74      0.75      0.75        69

    accuracy                           0.80       195
   macro avg       0.86      0.82      0.84       195
weighted avg       0.80      0.80      0.80       195
```

我們進一步從混淆矩陣（confusion matrix）中分析了哪些類別容易被誤判成其他類別。我們發現 Other_Faults 有少數幾筆容易跟 Pastry 和 Bumps 搞混，同時有11筆 Bumps 資料被誤判為 Other_Faults。



透過混淆矩陣可以解釋分類模型在哪幾種類別表現較不好。

Kernel SHAP 解釋模型

建立一個通用的 KernelExplainer 解釋器，並試圖解釋剛剛訓練的 LightGBM 分類器。我們從訓練集中取出前 100 筆資料，以代表整體特徵值的分佈，用於進行模型解釋。然後，我們將使用測試集中的前 10 筆資料來計算 Shapley values。此外，我們將 nsamples 設定為 100，這表示我們將進行 100 次蒙地卡羅抽樣，從 KernelExplainer 設定的資料中隨機擾動抽樣，並建立一個 SHAP 簡化可解釋模型。

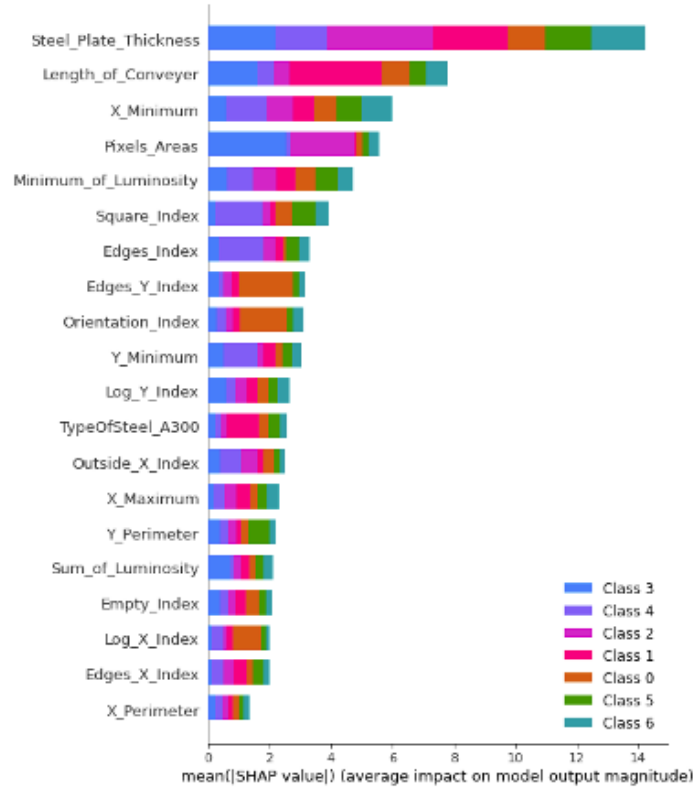
```
import shap
shap.initjs()

# 使用 Kernel SHAP 解釋模型
explainer = shap.KernelExplainer(model=model.predict_proba, data=X_tr
shap_values = explainer.shap_values(X=X_test[:10], nsamples=100)
```

SHAP Summary Plot (全局解釋)

我們可以使用 SHAP Summary Plot 來進行模型的全局解釋，該圖表顯示了每個特徵變量對整體平均模型輸出的平均影響。每個顏色代表不同的類別，因此我們可以觀察每個特徵對於模型預測輸出的平均貢獻程度，以及在不同類別下哪個特徵佔有較大的重要性。

```
shap.summary_plot(shap_values, X_test, plot_type="bar", feature_names
```

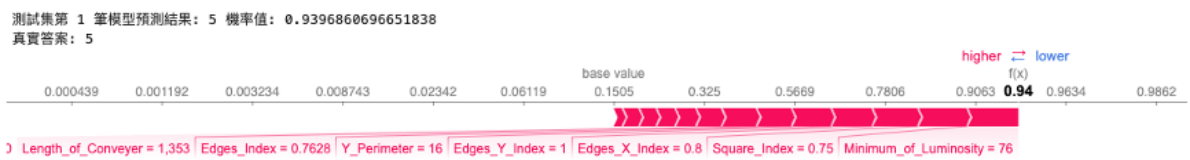
從下圖我們可以看到前三名重要的特徵為： 1. Steel_Plate_Thickness 2. Length_of_Conveyer 3. X_Maximum



SHAP Force plot (單筆資料解釋)

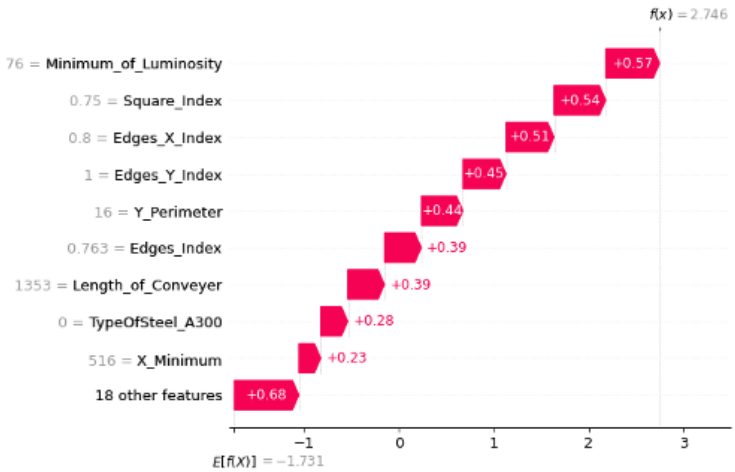
我們可以使用 Force Plot 方法觀察單一筆資料在模型中的預測情況。在 SHAP 套件中, Force Plot 提供了對單一模型預測的解釋性呈現。這個圖表清楚顯示了各個特徵對於模型對特定輸入值的預測所做的貢獻。從下圖結果中, 我們可以看到模型預測結果為5（Bumps）, 其預測機率為 0.939。因此圖中的 Force Plot 是針對解釋為什麼這筆資料輸入會得到標籤為5的 Shapley values 解釋。

```
# 觀察測試集中第一筆資料預測的重要程度
index=0
pred_class = int(model.predict(X_test[[index]])[0])
pred_proba = model.predict_proba(X_test[[index]])[0][pred_class]
print(f'測試集第 {index+1} 筆模型預測結果: {pred_class} 機率值: {pred_proba}'
print(f'真實答案: {int(y_test[index])}')
shap.force_plot(explainer.expected_value[pred_class], shap_values[pre
```



SHAP waterfall plot (單筆資料解釋)

我們可以更近一步地用瀑布圖視覺化排序重要的特徵，同時觀看每個特徵的相對應 Shapley values。




```
shap.waterfall_plot(shap.Explanation(values=shap_values[pred_class][i
                                     base_values=explainer.expected_va
                                     feature_names=x_feature_names),
                                     max_display=27)
```

全部特徵的Shapley value總和加上基準值，最後再通過Sigmoid函數就是輸出的機率值了。

Permutation importance解釋全局模型

在本系列中 Day 10 *(https://ithelp.ithome.com.tw/articles/10325613)* 介紹了利用特徵擾動的方法解釋整個模型，當時使用了 `eli5` 實作特徵重要程度的排序。這裡再分享另一個方法實作，那就是使用 sklearn *(https://scikit-learn.org/stable/modules/permutation_importance.html)* 套件中的 `permutation_importance()`。

```
from sklearn.inspection import permutation_importance

# 使用 permutation_importance 函數計算特徵重要性
result = permutation_importance(model, X_test, y_test, n_repeats=10,

# 印出各特徵的平均重要性排序
sorted_idx = result.importances_mean.argsort()[::-1]
for i in sorted_idx:
    print(f"{x_feature_names[i]:<10} importance: {result.importances_
```

從下圖可以觀察模型在評估每個瑕疵時所使用的重要特徵排序。另外可以將這些排序結果與 SHAP Summary Plot 的全局解釋進行比對，以檢查不同方法在模型特徵重要性排序方面是否具有一致性。由於理論方法和資料抽樣的隨機性，無法保證每種方法的解釋都完全相同，但我們可以透過尋找共通的解釋來增強可信度。

```
Steel_Plate_Thickness importance: 0.171 +/- 0.014
Length_of_Conveyer importance: 0.114 +/- 0.015
TypeOfSteel_A300 importance: 0.046 +/- 0.013
Y_Minimum  importance: 0.044 +/- 0.014
Minimum_of_Luminosity importance: 0.044 +/- 0.017
Edges_Y_Index importance: 0.031 +/- 0.010
X_Minimum  importance: 0.029 +/- 0.007
Edges_Index importance: 0.027 +/- 0.013
Maximum_of_Luminosity importance: 0.024 +/- 0.013
X_Maximum  importance: 0.023 +/- 0.013
X_Perimeter importance: 0.018 +/- 0.005
Empty_Index importance: 0.017 +/- 0.007
Square_Index importance: 0.014 +/- 0.010
Orientation_Index importance: 0.012 +/- 0.010
Sum_of_Luminosity importance: 0.011 +/- 0.012
Luminosity_Index importance: 0.010 +/- 0.009
Log_X_Index importance: 0.008 +/- 0.010
Outside_Global_Index importance: 0.002 +/- 0.003
Outside_X_Index importance: 0.002 +/- 0.009
LogOfAreas importance: 0.000 +/- 0.000
Y_Maximum  importance: 0.000 +/- 0.000
Pixels_Areas importance: -0.001 +/- 0.009
TypeOfSteel_A400 importance: -0.001 +/- 0.002
Edges_X_Index importance: -0.001 +/- 0.008
SigmoidOfAreas importance: -0.001 +/- 0.007
Log_Y_Index importance: -0.003 +/- 0.009
Y_Perimeter importance: -0.005 +/- 0.008
```

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/crazyai-xai)* 取得！

Refernece

- UCI 資料集：Steel Plates Faults *(https://archive.ics.uci.edu/dataset/198/steel+plates+faults)*
- 鋼材缺陷偵測分類：LightGBM王者登頂 *(https://zhuanlan.zhihu.com/p/498131022)*

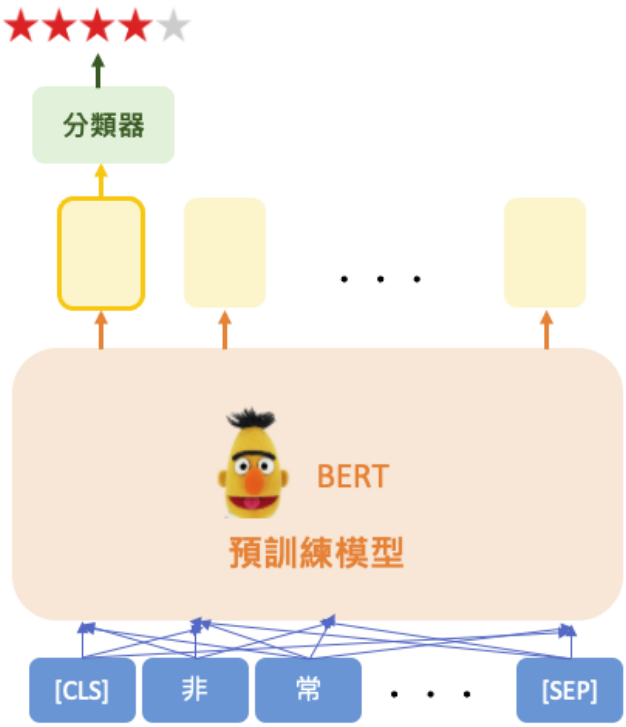
[Day 27] XAI在NLP中的應用：以情感分析解釋語言模型

範例程式：Open in Colab *(https://colab.research.google.com/github/andy6804tw/crazyai-xai/blob/main/code/27.XAI在NLP中的應用：以情感分析解釋語言模型.ipynb)*

近年來自然語言處理（NLP）領域取得了巨大的進展，主要歸功於大型語言模型（LLM）的崛起。這些模型，如 GPT、LLaMA 和 BLOOM 等，已經在多個 NLP 任務中取得了驚人的成就。目前 NLP 領域的主要趨勢是將大型語言模型微調以適應各種下游任務，例如對話機器人、機器翻譯、和情感分析等等。這種微調過程通常需要大量的標註數據，使得模型變得更具特定性，但同時也使得模型的內部運作變得更加難以理解。在今天的內容中我們將以情感分析為例，並透過可解釋工具 SHAP 來分析模型是如何辨識一段句子中情感的判斷依據。

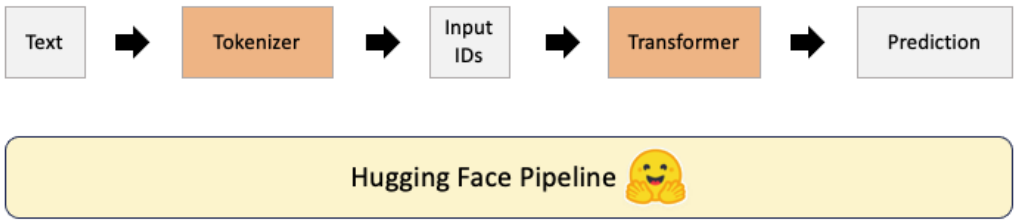
情感分析 (sentiment analysis)

情感分析（Sentiment analysis）是種自動化標註一段句子的情感（例如，積極、消極或中性）的過程。一個著名的情感分析數據集是 IMDB 影評數據集，其中包含了大量用戶對電影的評論，這些評論具有明顯的正面或負面情感。在今天的範例中可以輸入一段中文句子，模型可以從文字中判斷喜好程度，並根據其程度分配一個評分，從最低的1星到最高的5星不等。



Hugging Face 使用指南

Hugging Face *(https://huggingface.co/)* 是一個公開的 AI 社群，同時也是一個熱門構建機器學習應用的工具。並提供豐富的預訓練模型、資料集以及開發工具，涵蓋自然語言處理、語音辨識、影像辨識等多種 AI 應用。以情感分析為例，下圖展示了 Hugging Face 模型的整體運作流程，而我們可以使用 pipeline 工具，將所有的過程封裝在一起，讓我們使用上更加方便。



1. 對輸入句子進行分詞和編碼

- 分詞器(tokenizer)將輸入的句子分割成單詞(tokens)。
- 接著將這些單詞(tokens)轉換為相對應的token id，同時添加預訓練模型所需的特殊token（例如：[CLS]、[SEP]）。

2. 載入 Hugging Face 模型

一旦分詞和編碼的工作完成，我們可以直接將處理好的結果傳遞給相對應的模型，然後取得預測結果。

3. 輸出預測結果

輸出結果依據喜好程度從最低1星到最高5星。

[實作] 情感分析

若要開始使用 Hugging Face，首先需要在 Python 環境中安裝 `transformers` 套件。如果尚未安裝，可以使用以下指令安裝：

```
pip install transformers
```

Hugging Face 提供了一些著名的預訓練 NLP 模型，包括BERT、GPT、RoBERTa等。這裡我們可以直接載入別人已訓練好的中文情感分類模型（chinese_sentiment）。並使用指定的預訓練模型初始化序列分類模型，同時需要指定模型所相對應的分詞器（tokenizer）。最後使用 pipeline 創建一個情感分析器，使用上面初始化的模型和分詞器。另外可以設定 `top_k` 參數，指定模型輸出的前 k 名次的機率排序。

```
from transformers import AutoTokenizer, AutoModelForSequenceClassific

# 使用指定的預訓練模型初始化序列分類模型
model = AutoModelForSequenceClassification.from_pretrained("techthiya
# 使用指定的預訓練模型初始化分詞器
tokenizer = AutoTokenizer.from_pretrained("techthiyanes/chinese_senti
# 使用pipeline創建一個情感分析器
review_classifier = pipeline('sentiment-analysis', model=model, token
```

預訓練模型建置完畢後，我們就可以輸入一個句子讓 AI 來判斷使用者會給予幾顆星評論。首先我輸入一段很正向的評論，果不其然 AI 給了很高的五顆星評價。

```
# Example 1
review_classifier("環境乾淨且服務人員很親切，餐點口味很棒而且食物新鮮！")
```

輸出結果：

```
[[{'label': 'star 5', 'score': 0.6823225617408752},
  {'label': 'star 4', 'score': 0.26904016733169556},
  {'label': 'star 3', 'score': 0.03292671963572502},
  {'label': 'star 2', 'score': 0.01019265130162239},
  {'label': 'star 1', 'score': 0.0055178832262754 44}]]
```

我們現在試試另一個句子，這次AI給了該句兩顆星的評價。

```
# Example 2
review_classifier("上餐速度太慢，食物普通。")
```

輸出結果：

```
[[{'label': 'star 2', 'score': 0.40975552797317505},
{'label': 'star 3', 'score': 0.34248214960098267},
{'label': 'star 1', 'score': 0.1721179038286209},
{'label': 'star 4', 'score': 0.06531713157892227},
{'label': 'star 5', 'score': 0.010327262803912163}]]
```

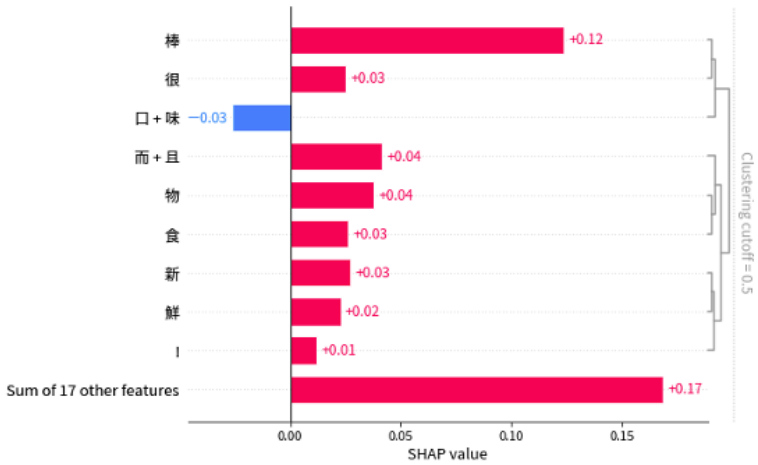
使用 SHAP 解析語言模型

SHAP 不僅可以用於解釋圖像和表格型數據，還可用於解釋語言模型。首先我們使用 `shap.Explainer` 初始化了一個 SHAP 解釋器，這個解釋器將用於解釋模型的預測。接下來我們使用這個解釋器來解釋一個句子，並估算出該句子對於每個輸出類別的 Shapley values。最後使用 `plots.text` 方法對這個句子的解釋進行視覺化呈現。

```
import shap

explainer = shap.Explainer(review_classifier)
shap_values = explainer(["環境乾淨且服務人員很親切，餐點口味很棒而且食物新鮮！"])
shap.plots.text(shap_values)
```

另外也可以透過 `plots.bar` 詳細的針對每個文字進行重要性的排序。從下圖可以觀察到每個文字對於模型預測的貢獻，以便我們可以理解模型的預測是如何形成的，哪些部分的貢獻最大，以及它們對預測的影響。此外還可以透過 `clustering_cutoff` 參數對 shap value 做聚類，此時相關性強的文字就能顯現出來。

```
shap.plots.bar(shap_values[0,:,4], clustering_cutoff=0.5)
```



不過就我個人而言，SHAP套件在解釋中文文字方面稍嫌不足。這是因為 BERT 的 tokenizer 是以每個字元為單位的，因此 SHAP 是針對每個字元進行 Shapley values 的估算。如果想要針對詞語進行解釋，例如親切、很棒、新鮮等，可以考慮使用 LIME 來進行解釋。LIME 可以使用 `jieba` 分詞工具將一段話解析成多個詞語，然後再針對每個詞語進行重要性解釋。這種方法更適合針對詞語級別的解釋需求。

本系列教學內容及範例程式都可以從我的 GitHub *(https://github.com/andy6804tw/crazyai-xai)* 取得！

Reference

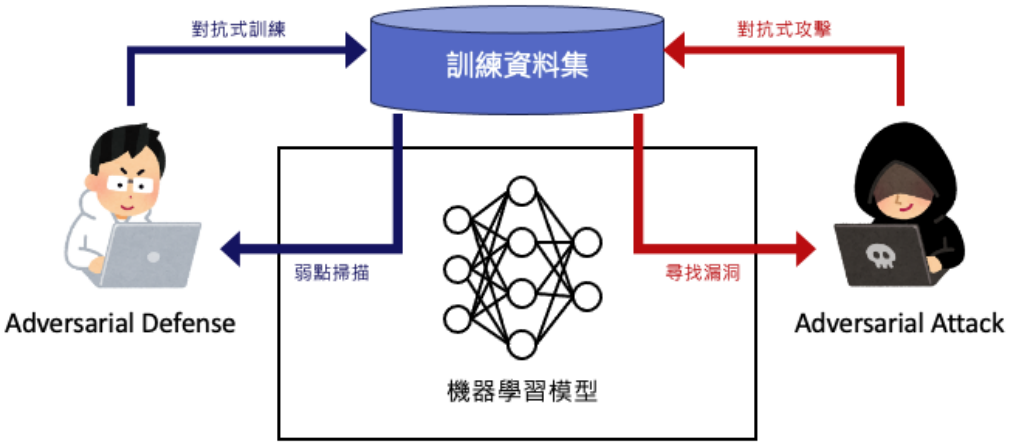
- SHAP Document: Positive vs. Negative Sentiment Classification *(https://shap.readthedocs.io/en/latest/example_notebooks/text_examples/sentiment_analysis/Positive%20vs.%20Negative%20Sentiment%20Classification.html)*
- 利用LIME和SHAP對bert訓練的文本分類模型做可解釋性分析 *(https://zhuanlan.zhihu.com/p/476845575)*
- 這篇詳細解釋LIME的數值是如何 *(https://towardsdatascience.com/what-makes-your-question-insincere-in-quora-26ee7658b010)*

6.XAI的挑戰與未來

[Day 28] 對抗樣本的挑戰：如何利用XAI檢測模型的弱點？

在本系列文章中，我們已經介紹了機器學習領域中用於解釋複雜的黑盒模型的各種方法，想必各位對這些技術有了一定的了解。儘管我們可以透過可解釋性技術來證明模型的能力，但隨著模型變得越來越複雜，我們也開始關注它們的弱點和不確定性。

其中對抗樣本是一個極具挑戰性的問題。它可以針對模型的弱點有意地毒害模型，並引導模型做出錯誤的預測。在今天的內容中，我們將介紹一些 Adversarial Attack（對抗式攻擊）和 Adversarial Defense（對抗式防禦）的技術，幫助各位理解該如何面對這些挑戰。



深度學習模型的弱點

首先我們來談談模型在安全性方面的弱點，以及這些弱點可能導致的風險。在2019年的一場全球黑帽大會上，一名研究人員展示了成功破解臉部辨識系統的方法。他們發現當使用者佩戴眼鏡時，臉部辨識系統基本上無法從眼框區域提取 3D 訊息。為了欺騙這種生物辨識技術，研究人員設計了一副特殊眼鏡，然後在眼鏡鏡片上貼上黑色膠帶，並在中間畫上白色點，以模擬人眼的外觀。

> 圖片來源: patentlyapple

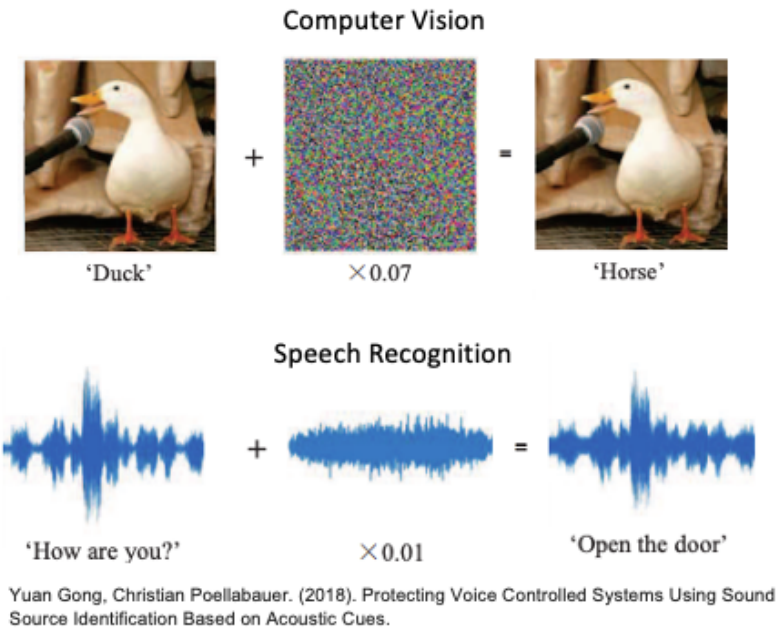
此方法最終成功地解鎖了裝置，從這件事情告訴我們，幾乎任何產品都存在著弱點和不確定性。即便在看似安全的系統中，也存在被攻擊的風險。因此對抗樣本的挑戰是我們該重視的議題。

> 相關報導: Face ID was hacked at the Black Hat Conference *(https://www.patentlyapple.com/2019/08/while-face-id-was-hacked-at-the-black-hat-*

conference-the-plausibility-of-it-occurring-could-only-be-found-in-a-bad-b-movie.html)

對抗樣本的挑戰

對抗樣本是指經過特定修改或處理的輸入數據，其目的是欺騙機器學習模型，使得預測或分類產生錯誤的結果。以影像辨識為例，常見的做法是在圖像上添加人類難以察覺的雜訊，這些雜訊可能是像素細微的變化或色彩微調。雖然對人類眼睛而言，這些變化幾乎不可察覺，但這些微小的改變可能足以使模型的預測完全錯誤。以下圖例子中，一張鴨子的照片經過添加了微小的對抗雜訊後，這張圖像被辨識成馬。此外聲音類型的資料也可以使用類似的原理。我們可以在一段聲音訊號中引入雜訊，以混淆 AI 辨識的結果。



Yuan Gong, Christian Poellabauer. (2018). Protecting Voice Controlled Systems Using Sound Source Identification Based on Acoustic Cues.

相關論文：Protecting Voice Controlled Systems Using Sound Source Identification Based on Acoustic Cues *(https://arxiv.org/abs/1811.07018)*

對抗式攻擊 vs. 對抗式防禦

對抗式攻擊和對抗式防禦是對抗性機器學習領域中的兩個重要概念，它們都涉及到研究模型的安全性以及可能的攻擊手法。這兩者之間的關係就像是攻守兩端的交握，其中攻擊者試圖找到模型的弱點，而防守者則致力於發展技術來保護模型免受攻擊。

對抗式攻擊（Adversarial Attack）

對抗式攻擊是指針對機器學習模型，特別是深度學習模型，有意地設計特定輸入，以引起模型錯誤預測或誤判的行為。然而依據攻擊的方法又分成白盒攻擊和黑盒攻擊兩種對抗式攻擊機器學習模型的方法。它們有不同的特點和目的。以下是對這兩種攻擊的解釋：

- 白盒攻擊（White-Box Attacks）：是指攻擊者完全了解目標機器學習模型的內部結構，包括訓練數據、模型結構，以及模型的預測函數。通常利用模型的結構和參數進行梯度攻擊、對抗性訓練等方法，以創建對抗性樣本達到攻擊目的。 > Intriguing properties of neural networks *(https://arxiv.org/abs/1312.6199)*(Goodfollw et al., 2013)
- 黑盒攻擊（Black-Box Attacks）：是指模型結構和權重不會被攻擊者知道。攻擊者僅能通過輸入輸出互動來試圖發現模型的弱點。通常利用攻擊樣本的可遷移性，去攻擊其他未知的模型。 > Practical black-box attacks against machine learning *(https://arxiv.org/abs/1602.02697)* (Goodfollw et al., 2017)

對抗式防禦（Adversarial Defense）

對抗式防禦是指在機器學習領域中針對對抗式攻擊開發的技術和策略，目的在於提高模型的泛化能力，使其更難受到對抗式攻擊的影響。對抗式防禦可以採取多種不同的形式，其中包括：

- 對抗性訓練：這是一種訓練模型的方法，其中模型在訓練過程中加入對抗性樣本，使其能夠識別和適應這些攻擊樣本。 > Explaining and Harnessing Adversarial Examples (FGSM) *(https://arxiv.org/abs/1412.6572)*(Goodfollw et al., 2014) > Adversarial examples in the physical world (PDG) *(https://arxiv.org/abs/1607.02533)*(Goodfollw et al., 2016)
- 模型融合：結合多個模型的預測結果，以減少對抗式攻擊的成功率。通常攻擊者針對單個模型的弱點進行攻擊，但當多個模型的預測一致時，攻擊變得更加困難。 > Boosting adversarial attacks with momentum *(https://arxiv.org/abs/1710.06081)*(Dong Y et al., 2018)
- 資料預處理：對輸入資料進行預處理，像是影像的去噪、平滑化或隨機化擾動，以減少對抗性攻擊的可能性。 > Feature squeezing: detecting adversarial examples in deep neural networks (去噪) *(https://arxiv.org/abs/1704.01155)*(Weilin Xu et al., 2017) > Mitigating adversarial effects through randomization (隨機化) *(https://arxiv.org/abs/1711.01991)* (Cihang Xie et al., 2017)

在對抗樣本領域中有攻就有守，因此一項 AI 產品在發布之前弱點測試與改善是很重要的一環。別等到翻車了才發現事情的嚴重性！

Reference

- Protecting Voice Controlled Systems Using Sound Source Identification Based on Acoustic Cues(arxiv) *(https://arxiv.org/abs/1811.07018)*

- Adversarial Machine Learning: A Beginner's Guide to Adversarial Attacks and Defenses *(https://hackernoon.com/adversarial-machine-learning-a-beginners-guide-to-adversarial-attacks-and-defenses)*
- 運用AI來測試AI-深度學習模型的弱點測試與改善 *(https://ictjournal.itri.org.tw/xcdoc/cont?xsmsid=0M236556470056558161&sid=0M349551271371725866)*
- 何謂對抗式機器學習？ *(https://www.cio.com.tw/anti-machine-learning-exposing-attackers-to-disrupt-ai-and-ml-systems/)*
- 深度學習中的對抗性攻擊與防禦 *(https://www.engineering.org.cn/ch/article/27665/detail)*

https://www.youtube.com/watch?v=qCYAKmFFpbs

[Day 29] XAI如何影響人類對技術的信任和接受程度？

在科技迅速進步的時代，人工智慧已經深深地融入了我們的生活，從智慧製造、智慧醫療、智慧服務到智慧交通，無所不在。然而這種深度的整合也伴隨著對於技術的信任和接受程度的關切。如今 XAI 已成為解決這些問題的關鍵，它不僅協助我們提高對技術的信任與對技術的接受程度。當一般大眾能夠理解 AI 如何達到特定的結果時，他們更有可能接受並使用這些技術。

- 透明：人工智慧系統應該易於理解。
- 可靠：人工智慧系統應確保運行可靠和安全。
- 公平：人工智慧系统應該公平對待所有人。
- 隱私：人工智慧系統應該保障並尊重隱私。
- 包容：人工智慧系統應確保每個人都能夠參與和受益。
- 負責：人工智慧系統應該建立監督標準。

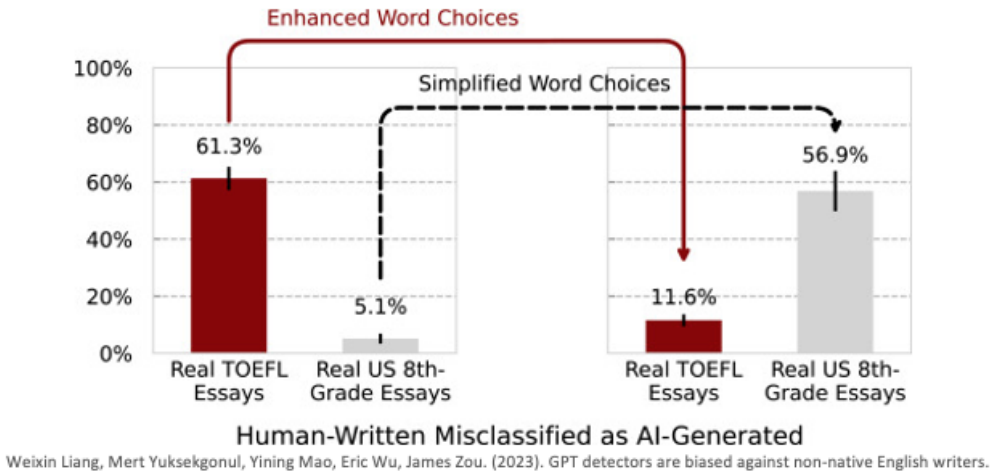
一個負責任的人工智慧除了以上六項原則之外，另外 XAI 也引發了對 AI 倫理和道德的更多關注。想必這也是每個企業主和使用者所擔憂的議題，隨著對技術的信任增加，同時我們更關心 AI 的倫理道德和風險。

AI的倫理道德

AI 的倫理道德一直是引起廣泛關注的議題，而機器學習作為 AI 的重要分支，其倫理問題更是引人深思。在今日的文章中，我們將深入探討機器學習中常見的道德問題，並借鏡一些真實例子來闡述這些議題的重要性。

[案例一] GPT偵測器具備語文上的偏見與歧視

在近期的一項研究中，使用這些生成式預訓練模型（GPT）偵測器來判斷英文文章是否來自 AI，母語非英文的使用者在撰寫英文文章時，有超過一半的情況被誤認為是 AI 生成的文章。這引發了對 GPT 偵測器存在語文上的偏見和歧視的擔憂。在訓練 GPT 偵測器的過程中，研究人員使用了7款熱門的偵測器來檢查91篇源自中國論壇的托福寫作文章，以及88篇由美國8年級生所撰寫的英文作文。從實驗結果顯示，這些偵測器能較正確地分辨出由美國學生所撰寫的文章，但對於中國學生所撰寫的托福文章的識別率卻顯著下降，並誤判為是 AI 所生成的文章。



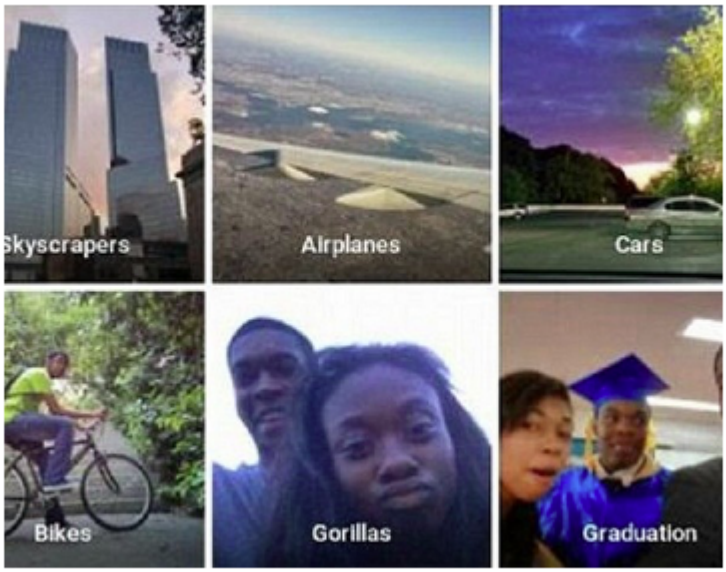
Weixin Liang, Mert Yuksekgonul, Yining Mao, Eric Wu, James Zou. (2023). GPT detectors are biased against non-native English writers.

這一問題的根本在於訓練數據的不平衡和偏向性。當訓練數據不夠多樣化或存在偏向性時，模型就容易產生誤判和歧視性行為。因此為了建立更公正、無偏見的 AI 系統，需要謹慎挑選和處理訓練數據，並定期檢查和調整模型以減少偏見。這也反映了 AI 技術應該受到倫理和多元性的審視，以確保其應用不會對不同文化和語言的使用者產生不公平的影響。

- 相關報導：非英語母語者寫的英文文章，有一半被GPT偵測器標記為AI生成 *(https://www.ithome.com.tw/news/157743?fbclid=IwAR1K6xShDCoDOIR-i_bzt3PrBvMQBdiC_O0hRnocJYA_oPqOTsWiH4x5aig)*

[案例二] Google相簿出包誤將黑人標成大猩猩

在這個案例中，Google 相簿使用了機器學習技術來自動辨識照片中的物體和人物，但不幸的是，該系統在辨識中出現了一個重大錯誤。這個錯誤導致該系統將一位用戶以及他的黑人朋友的照片誤標為大猩猩，這引起了公眾的廣泛討論，並帶出了有關人工智慧的資訊倫理議題。

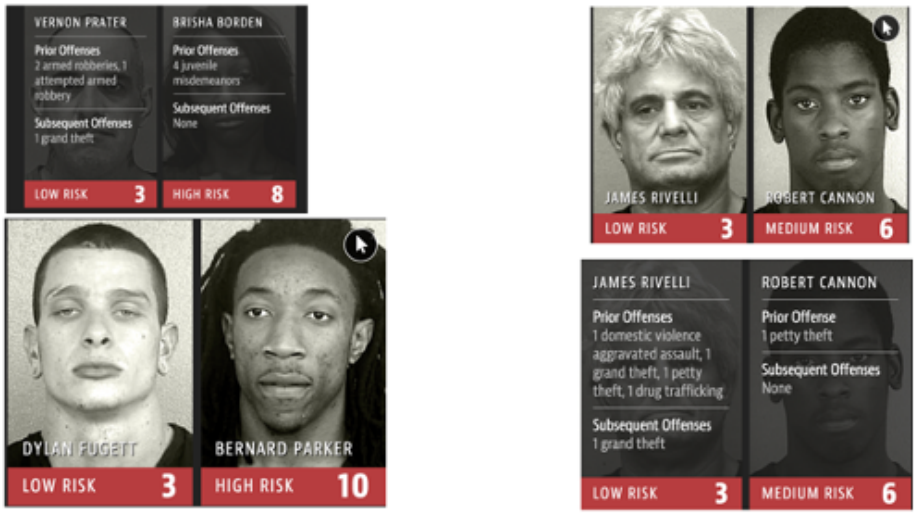


圖片來源: Jacky Alcine's Twitter (X)

- 相關報導：相片辨識出包誤將黑人標成大猩猩，Google火速道歉 *(https://www.ithome.com.tw/news/97131)*

[案例三] 鐵面無私包青天？小心AI的內建歧視

近年來，美國法院廣泛使用名為「COMPAS」的 AI 系統，這是由商業公司開發的，用來協助法官評估被告的再犯風險，並作為判決的參考依據。然而許多研究 *(https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing)*已經明確指出，這套 AI 系統存在潛在的種族歧視問題，即有色人種更容易被預測為高再犯風險，這引發了廣泛的爭議和討論。



Julia Angwin, Jeff Larson, Surya Mattu and Lauren Kirchner, "Machine Bias," ProPublica, May 23, 2016

從案例二和案例三中我們可以看到，影像辨識技術在某些情況下存在未知的不確定性，並且容易受到種族歧視等問題的影響。這突顯了 AI 的一個關鍵限制：AI 缺乏真正的思考能力和判斷能力，而是依賴於訓練數據來做出決策。

- 相關報導：AI 當法官，會是正義女神的化身嗎？ *(https://dq.yam.com/post/13034)*

[案例四] AI攝影機誤把裁判的光頭當足球跟拍轉播

最後一個來點輕鬆的，在這個案例中，一家蘇格蘭足球俱樂部引入了一套AI攝影機系統，目的要自動追蹤足球比賽中球的蹤跡，以進行轉播。然而在比賽進行一段時間後，這套 AI 系統卻突然停止追蹤足球，而是將鏡頭對準了場邊的裁判光頭。這個事件凸顯出 AI 系統的不確定性和可信度問題。AI 攝影機系統的設計初衷是為了提供更好的比賽轉播體驗，但卻因為意外的失誤，導致了轉播的笑話。

這個案例強調了影像視覺可解釋性的重要性。如果 AI 攝影機系統能夠清晰地解釋其決策過程，或者提供有關為何選擇對準裁判光頭的合理解釋，可能有助於減輕事件的影響。

- 相關報導：蘇格蘭AI攝影機誤把裁判的光頭當足球跟拍轉播 *(https://buzzorange.com/techorange/2020/11/02/ai-soccer-follow-bald-lineman/)*
- 影像來源：AI錯把光頭認足球 *(https://www.zhihu.com/zvideo/1306220010348875776)*

建立對AI的信任

在建立對 AI 的信任方面，深度學習模型的可解釋性一直是一個關鍵的課題。深度學習模型的強大性能是不可否認的，但模型的高度抽象性也帶來了對其運作方式的不透明性。因此我們必須關注模型的解釋能力，以確保大眾能夠理解和信任模型的決策。

以金融業為例，我們可以看到深度學習模型在信用評分等領域的應用。在這樣的情境下，我們希望能夠回答一系列問題，以確保模型的信任度：

- 我們應該如何解釋模型中每個連接的權重，以理解它們在預測中的具體作用和含義？
- 哪些權重在最終預測中扮演了關鍵的角色，影響著最終結果？
- 權重的大小是否能提供有關輸入變數的相對重要性的訊息？

這些問題的答案對於金融機構以及任何其他使用 AI 的領域都非常重要。模型解釋性不僅有助於追蹤和理解模型的決策過程，還可以幫助檢測模型的偏見和錯誤。同時透明的模型解釋也可以提高使用者對AI系統的信任度，並推動更廣泛的技術應用。

例如一些金融機構如國泰金控已經開始使用 SHAP 演算法來解釋其 AI 模型的決策。他們還積極採用自建的公平性和反歧視模型驗證方法，以確保模型不會因種族、性別或其他因素而偏見。此外，他們引入了聯邦學習技術，以保護敏感數據並提高模型的安全性。同時，透過引入人類回饋，他們實踐了強化學習，不斷改進 AI 模型的性能，以符合 AI 治理原則的要求。

> 相關報導：實踐AI治理原則國泰金控聚焦四大技術 *(https://www.ithome.com.tw/news/158850?fbclid=IwAR1uzScM5f_DQ9hobImf4Wxo-6p400LKNqHLv15s_U4DNAHTnuP91NrA1KQ)*

AI的信任和技術接受

XAI 對於提高人類對技術的信任和接受程度具有巨大的潛力。它為我們提供了一個機會，可以建立更加透明和可信賴的人機互動，並確保 AI 技術在未來的應用中取得成功。隨著 XAI 領域的不斷發展，我們可以期待看到更多創新和改進，這將進一步推動技術的進步和社會的發展。除了今天所提到的案例外，我們可以看到 XAI 是如何改變這些領域的技術接受程度和社會影響：

- 金融風險管理：如何利用可解釋的方法預測市場趨勢？
- 網路安全：如何利用可解釋性的方法檢測和防止攻擊？
- 社會公正：如何確保機器學習模型不歧視特定族群？
- 醫學診斷：用可解釋的方法解釋醫學圖像和診斷結果
- 生物醫學：從基因組學到蛋白質折疊的解釋性分析
- 法律：如何利用可解釋的方法幫助判決和解釋法律條文？
- 認知心理學：如何理解人類的決策和行為？
- 教育：如何利用可解釋的方法評估學生學習成效？

Reference

- Full Stack Deep Learning 2022: Lecture 9: Ethics *(https://fullstackdeeplearning.com/course/2022/lecture-9-ethics/)*
- 五張圖詳解企業如何建立可信任的AI *(https://ek21.com/news/tech/191675/)*
- 如何將DeepSHAP應用於神經網絡 *(https://kknews.cc/zh-tw/code/n9lyk23.html)*

- 人工智慧也有歧視和偏見 *(https://zhuanlan.zhihu.com/p/68987264)*

[Day30] XAI未來發展方向：向更可靠的機器學習模型邁進

機器就像人類一樣，並非百分之百地完美。就如同人類有時會犯錯，機器在預測或決策中也可能犯錯。當我們面對預測錯誤時，如果能夠透過 XAI 分析出合理的原因，可以改善模型並提供有利的根據修正它。在本系列三十天文章中，前半段是針對一般機器學習模型進行可解釋的介紹。在這過程中我們講述了模型事後解釋與模型本身可以解釋的觀念，並介紹了兩個熱門的 XAI 套件 LIME 與 SHAP。

> [Day 2] 從黑盒到透明化：XAI技術的發展之路 *(https://ithelp.ithome.com.tw/articles/10318532)*
>
> [Day 4] LIME vs. SHAP：哪種XAI解釋方法更適合你？ *(https://ithelp.ithome.com.tw/articles/10320360)*

在本系列後半段從 Day 16 開始講解了深度學習中的可解釋技術，其中帶各位看了 DNN、CNN、LSTM 如何透過一些梯度訊息或是擾動等方法解釋神經網路。

> [Day 16] 神經網路的可解釋性：如何理解深度學習中的黑箱模型？ *(https://ithelp.ithome.com.tw/articles/10330576)*

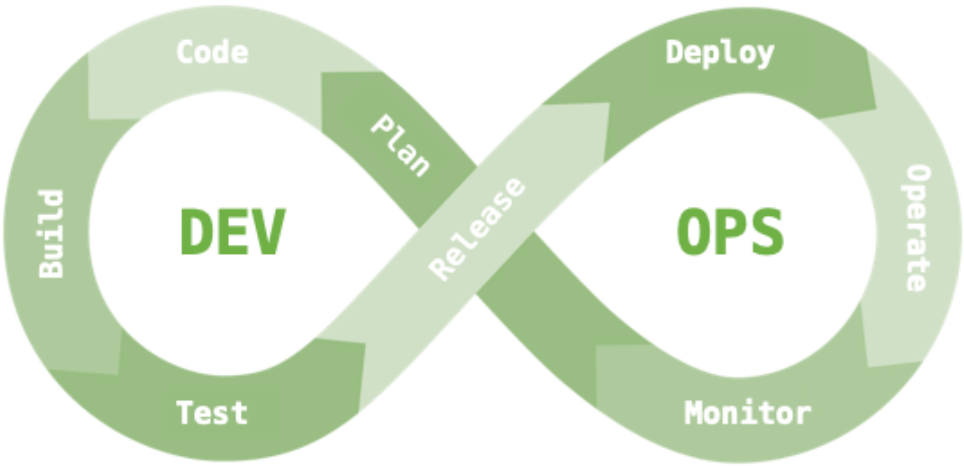
如果對電腦視覺的可解釋性有興趣的朋友可以參考 Day18~23，在這幾天文章中詳細介紹幾種不同方法來解析卷積神經網路。另外對於實務應用有興趣的可以參考 Day24~27 文章，在這之中講解了真實生活中的時間序列資料分析 *(https://ithelp.ithome.com.tw/articles/10335915)*、影像瑕疵檢測 *(https://ithelp.ithome.com.tw/articles/10336357)* 、鋼材缺陷分類 *(https://ithelp.ithome.com.tw/articles/10337150)*和自然語言應用 *(https://ithelp.ithome.com.tw/articles/10337606)*，並教導各位如何將這些訓練好的模型進行根因解釋。

XAI的下一步？

首先恭喜各位耐心的看到最後，相信各位已吸收滿滿的乾貨。看到這邊你可能會思考當模型訓練完了，也透過 XAI 確認模型的預測能力後，下一步該怎麼做呢？我們都知道開發的最後一哩路是部署應用。如何將 AI 模型部屬上線，是一個很重要的工程。你問我，我可能會說 MLOps 是企業導入 AI 的重要關鍵!

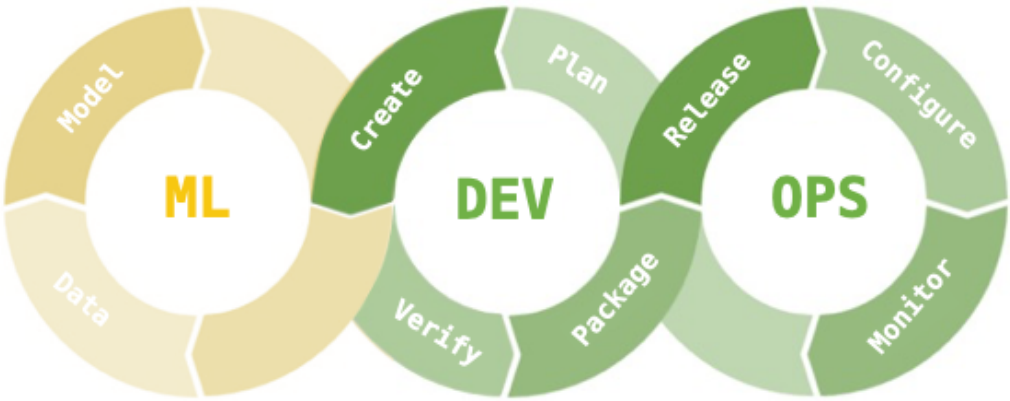
在介紹 MLOps 前先來談談一個很相近的名詞，叫做 DevOps。以下透過例子解釋 DevOps 的核心流程。當今天有新的功能計畫(Plan)出來時，團隊就會開始著手寫程式(Code)，程式寫完後會進行編譯(Build)打包成一個可執行的應用。接著我們會進行系統的測試(Test)，測試完成後我們會發布第一版(Release)以及部署應用(Deploy)。最後就是維運(Operate)，接下來維護團隊會來觀察(Monitor)這個新功能在客戶端的使用狀況。如果發生問題就會開始計畫更新，或是使用者

提出一個新的 issue，團隊經過討論後覺得可行就會進入下個階段著手計畫新的功能。DevOps 的迭代週期非常的快速，其優點是可以不斷的週期性更新功能越來越貼近使用者。這就是所謂的 DevOps 軟體開發流程。



> 延伸閱讀：上雲、IT現代化需求強，帶動企業2023持續加碼DevOps *(https://www.ithome.com.tw/article/159147?fbclid=IwAR38H90Nb7OKCYFp-PBVTLViqMZTnfCose5-h8MxzJzc2SzDGCQpKAcRE68)*

我們再回來看看 MLOps，它就是 Machine Learning + DEV + OPS 三個部分的縮寫合併。就如字面上的意思，它更深入地融合機器學習到 DevOps 流程中，並加強模型的維運，同時建立監控系統，以定期檢查模型的適用性並捕捉異常。



接下來討論一個 MLOps 重要環節：

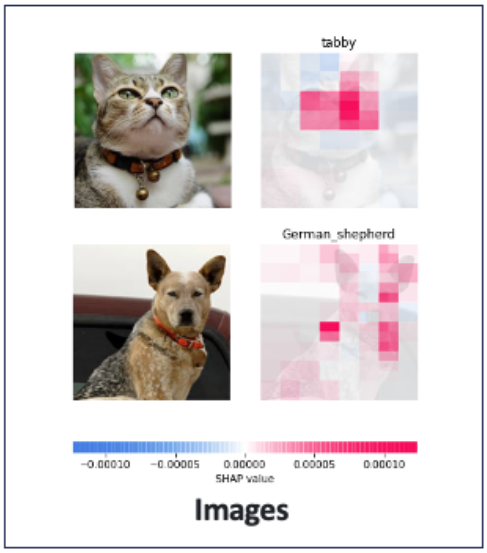
Monitoring system(監控系統)

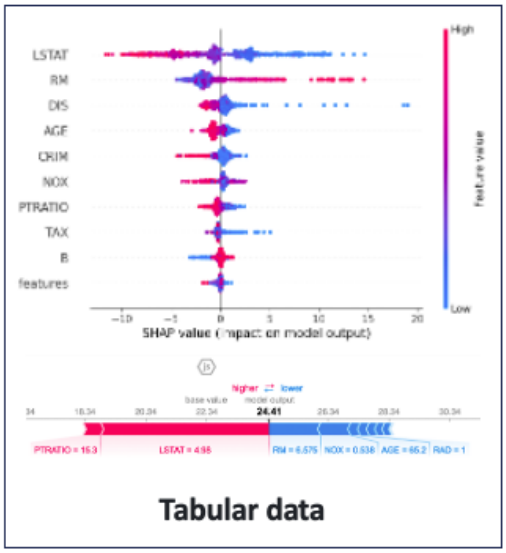
在軟體工程中，我們可以透過建立監控系統，在客戶發現問題之前就已經掌握問題所在。當模型犯錯時，我們會想知道為什麼模型犯錯。但是為什麼模型會犯錯呢？這可能有以下原因：

- 資料看過但學錯

- 資料尚未看過

首先資料看過但學錯這問題好解決，我們可以透過 XAI 技術協助我們除錯。例如在影像部分可以使用 Grad-CAM 來判斷模型所專注的區域。表格型的資料可採用 SHAP 來判斷是哪一個特徵造成最後的結果判別以及特徵重要程度。





但是資料尚未看過這個問題誰也無法預測。會發生這問題的原因可能是資料蒐集不夠豐富，使得模型學習能力不足。又或因為資料的時變特性造成資料的分佈集逐漸偏移，在學術界我們將這種情況稱為概念飄移(Concept Drift)。為了提高模型的預測能力，我們可以建立監控機制適當的檢測模型的適用性。並透過 XAI 的根因分析來查看模型是不是與我們預期的結果一致。

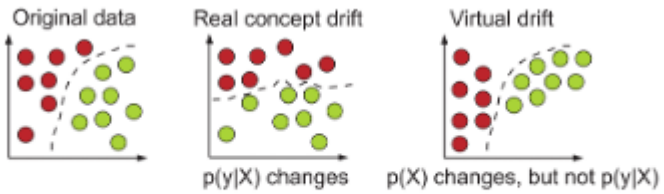


Fig. 1. Types of drifts: circles represent instances; different colors represent different classes.
Gama et al. (2014). A survey on concept drift adaptation. ACM computing surveys.

在未來，XAI 的技術仍持續發展，相信將會帶來更多的驚喜和創新。

Reference

- machine learning pipeline *(https://engineering.linecorp.com/zh-hant/blog/data-dev-interview-1)*
- 資料突然無法準確預測？淺談資料飄移 *(https://blog.infuseai.io/data-drift-ks-test-b884d50e4e12)*

- 複雜AI模型下的可解釋性 *(https://xueqiu.com/9217191040/200505095)*