113 學年度　國立彰化師範大學

資訊管理學系　人工智慧課程

期末報告

細懸浮顆粒濃度預測：多種深度學習模型的比較研究

指導教授：

翁政雄　博士/教授

學生：

S1161010 林品儀

S1161113 陳毓欣

S1161129 施佩茹

中華民國 113 年 11 月

摘要

本研究旨在利用多種深度學習模型（如 CNN、RNN、LSTM、GRU 及 Transformer）對北京 PM2.5 濃度進行預測分析。通過選取 2010 年至 2014 年的歷史數據，基於多種氣象變量（如風向、氣溫、氣壓）構建模型，並比較各模型的性能指標（MAE、RMSE、MAPE）。實驗結果表明，CNN 結合 LSTM 或 RNN 的混合架構在短期預測中具有最佳表現，相較於其他基礎模型如 Prophet 及 ARIMA，顯著降低了誤差。

研究同時指出數據來源單一、模型泛化能力不足以及計算資源限制等挑戰，並提出未來改進方向，包括引入多地區多維度數據、優化模型結構及參數調整策略，以及提升模型的解釋性與資源效率。本研究證實深度學習模型在空氣質量預測中的潛力，為環境管理與政策制定提供了重要參考依據。

致謝

首先，我們衷心感謝指導教授翁政雄在本研究中的悉心指導與支持，尤其是在研究方法設計和分析過程中提供的寶貴建議。

感謝本研究使用的數據來源——UC Irvine Machine Learning Repository，其公開的數據為本研究提供了重要支持。若無此數據，研究的順利進行將變得更加困難。

此外，我們特別感謝我們的研究團隊成員陳冠詠、黃紹嘉，他們為研究的數據分析提供了重要幫助。他們的合作對研究結果的可靠性與完整性起到了關鍵作用。

目錄

圖目錄

表目錄

第一章　引言

第一節　研究背景

北京的空氣污染問題由於城市的快速工業化和城鎮化，在 20 世紀末達到前所未有的嚴重程度，而除了移動污染源、火力發電廠、高污染產業工廠排放外，臺灣受境外污染（主要受中國霧霾影響）的佔比（約有 34%）也不容小覷[1]。$PM_{2.5}$（粒徑小於 2.5 微米的懸浮顆粒物）是空氣污染中的主要成分之一，來源包括燃煤、汽車尾氣排放、工業生產以及生物質燃燒。

由於其微小顆粒可深入呼吸系統，$PM_{2.5}$ 被證實對人體健康有顯著的長期危害，如引發哮喘、慢性支氣管炎和心血管疾病。北京特殊的地理位置——四周被燕山山脈和太行山包圍，導致空氣流動不易擴散，使污染物在不利的氣象條件下長時間滯留，特別是冬季的逆溫現象加劇了霧霾天氣的頻發與持續。此外，來自周邊城市的區域性污染輸送也是影響北京空氣品質的重要因素。儘管北京市政府已經採取了如「藍天保衛戰」等政策，這些措施包括加強排放控制、推廣新能源車輛以及提高環保標準，空氣品質雖有顯著改善，但其進展依然受到氣象條件與周邊地區污染源輸送的限制。

在新冠疫情期間，由於封鎖和限制措施導致經濟活動減少，空氣污染水平曾一度下降。然而，隨著疫情緩解和防控政策放寬，經濟活動恢復帶來的排放增加導致霧霾問題重新浮現，呈現出疫情後的倒退現象。這突顯出在面對區域性污染傳輸和經濟復蘇雙重挑戰下，持續推進跨區域協作和加強長期污染防治的重要性[2]。

這些因素凸顯了建立準確且穩健的 $PM_{2.5}$ 預測模型的重要性，唯有透過高效的預測模型，才能在空氣品質變化前提供及時的預警，協助相關部門制定有效的應對策略，減少污染對公共健康的影響，並加強空氣品質管理和預防措施。這樣的模型不僅需要捕捉本地污染源的變化，還必須考慮區域性污染輸送和氣象因素的複雜動態，從而提升預測的精準度和可靠性，為城市空氣污染防治提供科學依據。

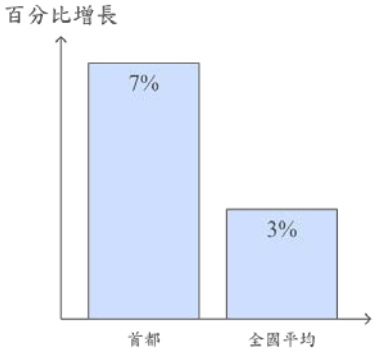

圖 1 首都增長率及全國平均增長率比較

第二節　研究動機及目的

本研究的動機源於現有 $PM_{2.5}$ 預測方法在應對複雜非線性變化和捕捉多重因素影響時具有一定的局限性。傳統的統計時間序列分析方法，雖然在處理簡單模型時能夠提供一定的準確度，但在揭示空氣污染數據中複雜的模式和多變的動態方面存在局限性。

基於此，本研究旨在應用並比較多種先進的人工智慧模型，包括卷積神經網絡（Convolutional Neural Network, CNN）、循環神經網路（Recurrent neural network, RNN）、長短期記憶網絡（Long Short-Term Memory, LSTM）、門控循環單元（Gated Recurrent Unit, GRU），以及 Transformer 模型，以實現 $PM_{2.5}$ 濃度的精確預測並比較其誤差表現。這些深度學習技術擁有處理高維度與非線性數據的能力，並具備出色的序列建模性能，對於解析和預測空氣污染濃度的時間序列變化尤為有效。

本實驗期望這些模型在準確度方面優於傳統方法，能夠提供更可靠的未來 $PM_{2.5}$ 濃度預測，為政府及環保機構提供更高效的預警依據，進而推動制定更具針對性的污染治理策略，減少污染對市民健康的影響，最終提升城市的生活質量和環境的可持續發展。

第二章　文獻探討

過去對於 $PM_{2.5}$ 空氣污染的研究大致可以被分為來源解析及健康影響三個領域，並且透過不同的方法如統計分析、機器學習或深度學習等技術進行研究。本章節先探討 $PM_{2.5}$ 研究領域及相關研究方法過往的文獻回顧，後討論每個領域的研究重點及應用，參考各方法的功能目的，進而引出本次實驗目標。

第一節　空氣污染研究

一、 來源解析

大氣細懸浮微粒的金屬成分特徵與來源分析是空氣污染研究中不可或缺的一環，對於深入理解污染物的主要來源及其在多變環境下的動態變化具有重要意義。$PM_{2.5}$ 中的金屬成分雖然含量相對稀少，但其對於人類健康和生態系統的潛在威脅不容忽視。這些微量金屬多源於燃燒活動、工業排放、交通污染、以及土壤和建築揚塵等多重來源。因而，準確識別和量化這些來源的相對貢獻，成為制定科學的空氣污染防治策略的基礎。

透過源解析和金屬成分特徵分析，大眾能夠辨識在不同氣象條件下，污染物來源如何變化，這不僅幫助人們理解污染物的複合成因，還能預測其在特定情境下的濃度變化趨勢，從而為公共健康風險評估和政策制定提供有力支持。這樣的深入分析有助於建立針對性的治理措施，優化資源分配，實現環境治理的最大效益。

要想找到 $PM_{2.5}$ 的來源，可以先以加強因子法分析微粒組成與汙染的相關性，而後使用主成分分析對氣膠汙染源做定性的分析，以此了解懸浮微粒可能的汙染源，最後再以化學質量平衡法對氣膠汙染源做定量的分析，推估各地區行業別對受體點汙染源的貢獻量[3]。

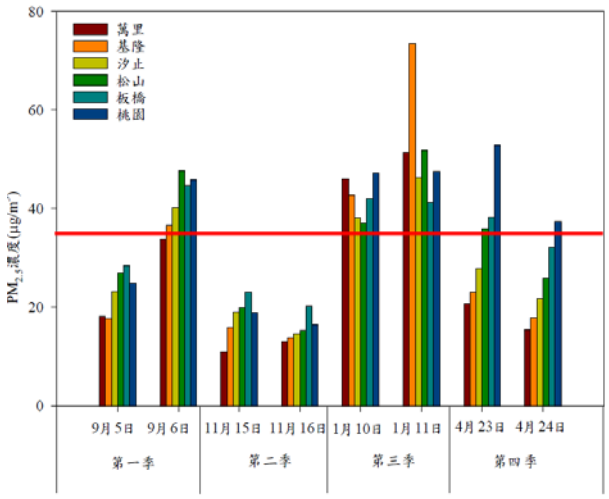

圖 2 四縣市同步 $PM_{2.5}$ 檢測結果，圖取自[3]中圖 4-1

二、 健康影響

研究顯示，北京市的年平均 $PM_{2.5}$ 濃度與同期肺癌發病人數之間存在顯著的正相關，相關係數達到 0.87，表明 $PM_{2.5}$ 的季度平均濃度與肺癌的發病率密切相關。北京市的 $PM_{2.5}$ 年均濃度約為 100 $\mu g/m^3$，且呈現非正態分佈，長期暴露在這種環境中會增加居民罹患肺癌的風險。$PM_{2.5}$ 及釷 210 等空氣污染物被認為是導致肺癌的重要因素。雖然傳統上吸煙被視為肺癌的主要原因，但近年的臨床數據強調，空氣污染，特別是來自汽車排放、工業廢氣和揮發性化學物質的霧霾，才是更為顯著的影響因素。調查顯示，重工業城市的肺癌死亡率與空氣中釷的濃度呈正相關，學術界對大氣污染在肺癌發病中的作用越來越重視[4]。

霧霾問題對社會和家庭造成了嚴重的經濟損失，且其健康成本無法完全用金錢來衡量。霧霾不僅影響人們的生活質量，還威脅到生命的安全。要根治霧霾，關鍵在於平衡就業與治理標準，並協調空氣保護與現有生活方式，這需要周密的規劃和有效的執行。政府應該優化產業結構，深化工業污染治理，推進 20 噸以下燃煤鍋爐的清潔能源改造，加強對工業排放的監管。此外，還應採取多方面的持續措施，例如調整城市功能、推廣綠色建築、有序提升工業標準、根本改變能源結構、規範餐飲業管理，以及提供財政補貼和金融激勵。這樣才能促進居民生活方式和經濟發展模式的轉型，徹底解決長期以來的霧霾問題。

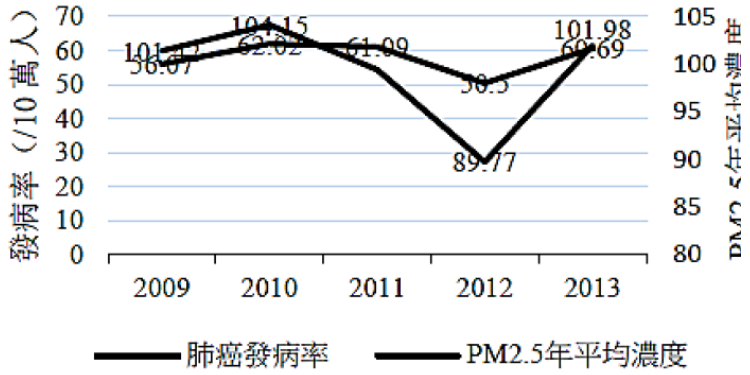

圖 3 2009~2013 年北京年微懸浮粒子 $PM_{2.5}$ 平均濃度與肺癌發病率，圖取自[4]中圖 3

第二節　研究方法

一、 卷積神經網絡 (Convolutional Neural Network, CNN)

卷積神經網路(Convolutional Neural Network, CNN)是由 Yann LeCun 於 1989 年提出的一種深度學習模型，利用局部特徵檢測、權重共享和池化壓縮的方式，使其既能處理大規模數據，又具有較強的泛化能力[5]。

基本的 CNN 包含卷積層（Convolution Layer）、池化層（Pooling Layer）與全連接層（Fully Connected Layer）三層。卷積層會使用過濾器（Filter）在數據上滑動並進行卷積計算，提取輸入資料的局部特徵並得到特徵圖（Feature Map），設定多個過濾器則會得到多個特徵圖。CNN 通常由多個卷積層組成，而卷積層生成的特徵圖會使資料量變大，因此，為了減少計算量和降低模型複雜度，會在卷積層後加入池化層。池化層會使用滑動視窗，並用最大池化（Max Pooling）或平均池化（Average Pooling）等方式提取數值，形成一個較小但依然保有重要特徵的新特徵圖。最後，全連接層會將二維特徵圖集合轉換成一維。

CNN 的特性：

- 應用於時間序列預測：CNN 能自動提取並有效的處理局部特徵，適用於具有規律性和局部依賴性的資料。在處理時間序列資料時，CNN 不需要依賴前一個時間步驟的輸出，因此可以更有效率的進行並行計算[6]。
- CNN 的擴展應用：在實際應用中，CNN 常與其他模型進行結合，以提高其在不同領域的預測能力和泛化性能。例如循環神經網絡（RNN）、

長短期記憶網絡（LSTM）以及門控循環單元（GRU）等模型，用於處理時間序列預測中的長期依賴性問題[7][8]。

CNN 能有效處理大規模數據，還具備良好的泛化能力。除此之外，在時間序列預測中，CNN 的混和模型也可以在發揮局部特徵上提取的優勢上，捕捉到序列中的長期依賴關係。

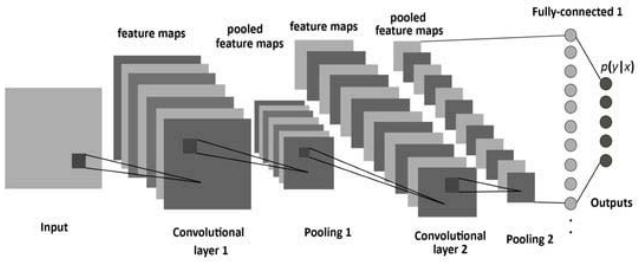

圖 4 卷積神經網路的示意圖，圖取自[9]中圖 2

表 1 有關 CNN 在回歸預測相關文獻

作者	名稱	內容
Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel (1989).	Backpropagation Applied to Handwritten Zip Code Recognition.	學習網絡的泛化能力可以通過提供來自任務領域的約束來大大增強。本文展示了如何通過網絡架構將這些約束整合到反向傳播網絡中。這種方法已成功應用於美國郵政局提供的手寫郵政編碼數字識別。一個單一的網絡可以學習整個識別操作，從字符的標準化圖像到最終的分類結果。
Y. LeCun, Y. Bengio (1995)	Convolutional Networks for Images, Speech, and Time-Series.	多層反向傳播網絡具有學習來自大量範例的複雜、高維和非線性映射的能力，使其成為圖像識別或語音識別任務的明顯候選方法（參見模式識別與神經網絡）。在傳統的模式識別模型中，通常使用手工設計的特徵提取器從輸入中收集相關信息並消除不相關的變異性。
黃綱正（2019）	結合 CNN 與 LSTM 模型建構分類惡意程式	本論文主要是利用深度學習模型進行惡意程式分類。目前已有研究利用機器學習方式實作(如：SVM、

	方法	Decision Tree 等)，近年研究已利用卷積神經網路及循環神經網路等類神經網路模型進行分類，相較於機器學習方式，具備較高準確率。本論文提出之模型，利用 Malimg 資料集作為惡意程式分類實驗，並與原提出論文作比較，準確率由 84.92%提高至 87.79%。所運用模型結合卷積神經網路與循環神經網路之優點，解決惡意程式樣本之特徵提取後資料序列分類問題。
陳雨玟（2024）	結合 CNN 圖像識別與時間序列分析股票重點指標	結果分析指出：在不同技術指標中，相對大的數值都會是漲跌的特徵，也無論漲跌圖，部分指標以固定數值大小作為特徵所在，而少數指標只能以逐漸上漲或是逐漸下跌作為特徵範圍，總之，透過本次研究，能夠進一步的了解指標的數值範圍以及特定規律的漲跌圖像，對於那些平時不熟悉指標，以及對指標數字變化不敏感的人，能為他們帶來參考資訊，更重要的是能為數據分析有了新的分析方式，透過可視化可以更方便、簡單的觀察圖像，也能夠替卷積神經網路應用上有更多發掘。
Nagi H. Al-Ashwal, Khaled A. M. Al Soufy, Mohga E. Hamza, Mohamed A. Swillam	Deep Learning for Optical Sensor Applications: A Review	過去十年中，深度學習（DL）已在大量光學傳感器應用中得到了廣泛應用。DL 算法能夠提高光學傳感器的準確性並降低噪聲水平。光學傳感器被認為是現代智能感知平台的一項具有前景的技術，廣泛應用於過程監測、質量預測、污染防治、國防、安全以及其他眾多領域。然而，光學傳感器面臨著一些主要挑戰，例如大規模數據集的生成、數據處理速度較低以及高昂的設備成本

		等。這些挑戰可以通過將 DL 系統與光學傳感器技術相結合來緩解。本文展示了最近將 DL 算法整合到光學傳感器應用中的研究進展，並突出介紹了一些 DL 算法在光學傳感器應用中具有顯著影響的方向。此外，本研究還為相關研究的未來發展提供了新的方向。

二、 循環神經網路（Recurrent neural network, RNN）

循環神經網路 (RNN) 是一種強大的人工神經網絡，設計用於處理順序資料 (Kaur & Mohta, 2019)。與傳統神經網路不同，RNN 包含內部循環，這些循環引入了跨處理元素的遞歸動態和延遲激活依賴性 (Hidasi et al., 2013)。這種內部記憶使 RNN 能夠隨著時間的推移維護訊息，使其對於涉及語音、文字、財務資料、音訊和視訊的任務特別有效 (Kanagachidambaresan et al., 2021)。 RNN 的運作方式是一次處理一個範例，保留反映長期依賴性的元素 (Kaur & Mohta, 2019)。雖然多層感知器 (MLP) 具有多個層，但 RNN 通常具有定向循環，透過該循環將輸入轉換為輸出 (Beysolow, 2017)。與其他演算法相比，這種獨特的結構使 RNN 能夠更深入地理解序列和意義，使其成為複雜序列資料分析和預測任務的首選(Kanagachidambaresan et al., 2021)。

特性：

- RNN 是一種具有內部循環的人工神經網絡，這些循環引入了遞歸動態，並使得網絡中的處理單元（PEs）之間存在延遲的激活依賴性。數學建模在研究大腦信息處理原理方面非常有用，但由於不同模型可能產生不同結果，通常使用標準模型來避免此問題。透過將一個神經模型簡化為標準模型，可以獲得大範圍模型的通用結果，並揭示這些模型之間共享的計算特性。[10]
- RNN 由於梯度消失和爆炸問題，訓練困難，且難以學習長期模式。RNN 層中的神經元間彼此糾纏，行為難以解釋。為了解決這些挑戰，文中提出了一種新的 RNN 類型——獨立循環神經網絡（IndRNN），其中同層神經元彼此獨立，並且跨層連接。IndRNN 可以有效防止梯度爆炸與消失問題，並能學習長期依賴性。該架構允許使用非飽和激活函數（如

ReLU)，並仍可穩定訓練。多個 IndRNN 可堆疊成更深的網路，實驗結果顯示，IndRNN 在處理長序列（超過 5000 時間步）和建立深層網路（最多 21 層）上表現優異，並且在各項任務中比傳統 RNN 和 LSTM 擁有更佳的效能。[11]

表 2 有關 RNN 在回歸預測相關文獻

作者	名稱	內容
Balázs Hidasi, Alexandros Karatzoglou, L. Baltrunas, D. Tikk (2013)	Recurrent neural networks	本文章為已熟悉人工神經網絡，特別是使用梯度下降算法（反向傳播）訓練的多層感知機的讀者介紹了循環神經網絡(RNN)。循環神經網絡是一種具有內部迴路的人工神經網絡。這些內部迴路在網絡中引入了遞歸動態，從而在網絡的處理單元(PEs)之間產生延遲激活的依賴性。
Shuai Li, W. Li, Chris Cook, Ce Zhu, Yanbo Gao (2018)	Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN	循環神經網絡(RNNs)已廣泛應用於處理序列數據。然而，由於眾所周知的梯度消失和梯度爆炸問題，RNN 的訓練通常十分困難，並且難以學習長期模式。為了解決這些問題，長短期記憶(LSTM)和門控循環單元（GRU）相繼被開發出來，但由於使用了雙曲正切（tanh）和 sigmoid 激活函數，導致梯度隨層數的增加而逐漸衰減。因此，構建一個高效可訓練的深層網絡變得極具挑戰性。此外，RNN 層中的所有神經元彼此緊密耦合，其行為難以解釋。 為了解決這些問題，本文提出了一種新型的 RNN——獨立循環神經網絡（IndRNN）。在 IndRNN 中，同一層的神經元彼此獨立，僅在層間相互連接。我們證明了 IndRNN

		可以輕鬆調控以防止梯度爆炸和消失問題，並且能有效地學習長期依賴。此外，IndRNN 可以與非飽和激活函數（如 ReLU，修正線性單元）配合使用，仍然能夠穩健地進行訓練。多個 IndRNN 可以堆疊起來構建比現有 RNN 更深的網絡。

三、 長短期記憶 (Long Short-Term Memory, LSTM)

根據文獻，長短期記憶網絡（LSTM）是一種非常適合用於時間序列預測的模型。LSTM 的設計旨在克服傳統遞歸神經網絡（RNN）所面臨的「梯度消失和梯度爆炸」問題，從而能夠更有效地處理長期依賴性。

LSTM 研究方法總結：

- LSTM 的基礎構造：LSTM 是由 Hochreiter 和 Schmidhuber 在 1997 年提出，並且被設計成能夠捕捉和保留長期資訊[12]。其獨特之處在於記憶單元（memory cell），它包括輸入閘（input gate）、遺忘閘（forget gate）和輸出閘（output gate），這些閘門控制信息的寫入、保留和讀出，從而在長期和短期信息之間取得平衡。
- 在時間序列預測方面，LSTM 常被用於處理具有時間順序的數據集，如股票價格和環境數據等。由於其出色的記憶和建模能力，LSTM 能夠更準確地預測長期依賴的數據趨勢，尤其在非線性時間序列數據中表現優異[12][13]。
- 模型訓練與性能評估：在研究中，LSTM 經常與其他模型（如 CNN 和基本的 RNN）進行比較。LSTM 通過優化均方誤差（MSE）、平均絕對誤差（MAE）和平均絕對百分比誤差（MAPE）等指標，顯示出其在處理複雜時間序列任務方面的優勢。例如，在二氧化碳排放預測的研究中，LSTM 顯示出較高的預測準確性[12]。
- LSTM 的應用範圍擴展：LSTM 經常作為混合模型的一部分，例如 CNN-LSTM。這類模型結合了 CNN 在特徵提取方面的優勢和 LSTM 在時間序列建模方面的能力，適合用於更為複雜的預測任務，如金融市場預測和環境變化[12][13]。

總的來說，LSTM 因其在處理長期依賴性和穩定性方面的優越表現，已成為時間序列預測的重要工具，並在多種應用中展現出卓越的預測能力。

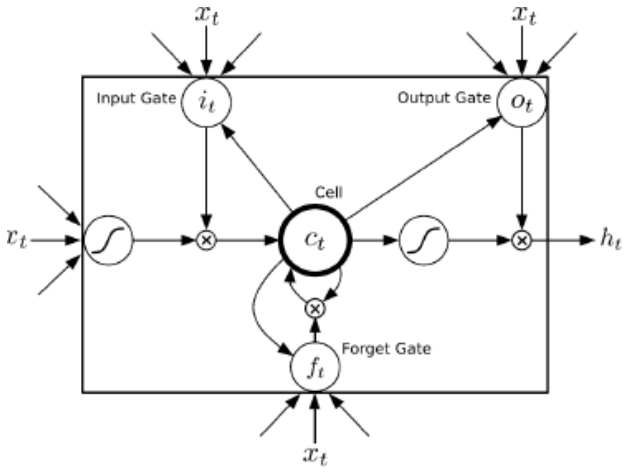

圖 5 長短期記憶神經網路示意圖，圖取自[14]中的 Figure 1

表 3 有關 LSTM 在回歸預測相關文獻

作者	名稱	內容
張曉妮（2024）	應用 CNN-LSTM 深度學習模型於預測台灣二氧化碳排放量	本研究結合卷積神經網路 (convolutional neural network, CNN) 和長短期記憶神經網路 (long short-term memory neural network, LSTM) 兩種深度學習演算法建立台灣 1990 年至 2022 年之二氧化碳排放量之預測模型，並加入影響碳排放量的相關經濟發展指標進行模型訓練，使用平均絕對百分誤差 (mean absolute percentage error, MAPE) 作為主要績效指標衡量模型表現，再與 CNN 與 LSTM 模型比較分析。另外本研究在使用資料擴增中提出修正方法，利用線性插值法 (linear interpolation approach) 建立數據間關係後使用比例縮放得到符合實際狀況之資料。而實驗結果顯示 CNN-LSTM 的績

		效表現最佳，其次為 LSTM 模型。
葉國毅（2024）	使用多頭注意力機制的多重輸入 LSTM 股價預測模型	本研究以台灣 50 指數的成分股作為訓練資料，透過皮爾森相關係數找出當日與標的股價相關的證券以及基準指數作為多重輸入特徵，結合注意力機制與 LSTM 建構股價預測模型，並應用模型預測值作為交易訊號進行交易績效回測。研究結果顯示多重的輸入資料以及注意力機制相對於其他基礎模型有更好的預測效果，在交易績效回測上相對於持有到期的方法，更能夠規避風險，穩定獲得正報酬。

四、 門控循環單元 (Gate Recurrent Unit, GRU)

門控循環單元（GRU）由韓國科學家 Kyunghyun Cho 於 2014 年在其博士後研究期間發表，是一種改進版的循環神經網絡（RNN）。GRU 的目的在於解決標準 RNN 中常見的梯度消失與梯度爆炸問題，從而更有效地學習長期依賴。相比於 LSTM，GRU 具有更高的計算效率，克服了 LSTM 計算速度較慢的缺點，是一種改良過的深度學習模型。

GRU 的特性與結構：

- GRU 的基礎結構：GRU 的基本結構包含更新門（Update Gate）和重置門（Reset Gate）。這兩個門控機制負責調節新輸入與先前記憶的結合方式，決定應該遺忘多少過去的信息，以及應保留多少舊信息，同時決定需要加入多少新信息以生成當前時間步驟的輸出。
- 模型訓練與性能比較：GRU 的結構與 LSTM 類似，但更加簡化，這種設計帶來了顯著的計算優勢，使其在訓練過程中能夠更高效、更快速地運行。在保持性能的同時，GRU 去除了 LSTM 中的部分複雜元件，從而減少了計算成本。在許多應用中，GRU 不僅能實現與 LSTM 相似的效果，還能在一定程度上提供訓練速度和資源使用的優勢，特別是在需要快速迭代和處理較大數據集的任務中更為突出。

- 應用於處理時間序列的資料集：GRU 被廣泛應用於需要處理序列數據的各種領域，例如股價波動預測、電力消耗和再生能源發電量等。這些應用場景中，GRU 的設計使其特別適合於捕捉和處理數據中的時序關聯，尤其是預測未來的變量。其高效的運算和簡化的結構不僅縮短了訓練時間，還能在實際應用中提供穩定的性能表現，這使 GRU 成為許多任務中的理想選擇[15][16]。

GRU 在解決時序依賴和序列預測問題中展現出極高的靈活性，特別是在涉及長期記憶的任務中尤為高效。由於其結構比 LSTM 更加簡潔，GRU 提供了更高的計算效率和更快的訓練速度，這使其成為處理各類序列數據時的理想選擇。

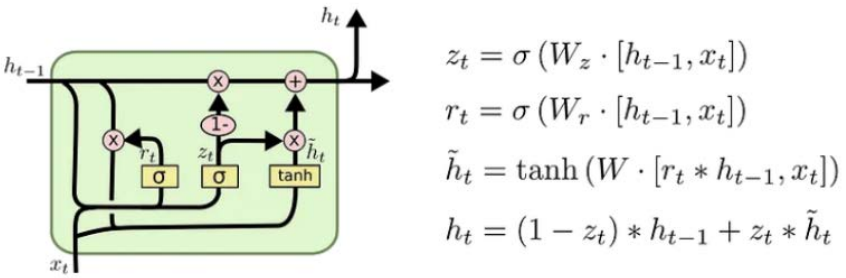

圖 6 門控循環單元神經網路的示意圖，圖取自[17]中的圖三

表 4 有關 GRU 在回歸預測相關文獻

作者	名稱	內容
李思賢（2024）	利用 LSTM、RNN 與 GRU 模型預測比特幣價格：一個深度學習方法的效能分析	實驗結果顯示，使用 LSTM 模型在比特幣價格預測中具有較高的準確度，尤其是當結合交易量和 MACD 等交易相關特徵時，預測精度會進一步提升。特徵間的相關性分析進一步增強了模型選擇合適特徵的能力，從而獲得更精確的價格預測。
游涴喻（2024）	邁向淨零碳排放利用深度學習 LSTM 與 GRU 預測台灣電力消耗與再生	深度學習模型能有效捕捉電力消耗和再生能源發電量的時序變化，透過

	能源未來用量之研究	多種評估指標如 R2 Score、MAE、MSE 和 RMSE 的比較，我們發現這些模型能夠有效捕捉電力的趨勢和季節性波動。這些結果不僅加深了對深度學習模型在能源預測領域應用的理解，為能源管理和政策制定提供了科學的數據支持，有助於推動台灣實現淨零碳排放的長遠目標。

五、 Transformer 模型

在深度學習領域，Transformer 模型因其出色的性能和靈活性而受到廣泛關注。隨著計算需求的增加，研究人員開始尋找提高 Transformer 計算效率的方法，特別是在乘累加運算（MAC）方面。傳統的優化技術如知識蒸餾、剪枝和量化已經被深入探討，但隨著對能耗和計算效率要求的提高，查找-累加（LAC）運算等新方法逐漸成為研究的焦點。

Transformer 模型的關鍵在於其多頭自注意力機制，該機制依賴於大量的乘法和加法運算。這些運算不僅計算量龐大，還消耗大量能量，因此許多研究者正在尋找替代方案。例如，AdderNet 和 ShiftCNN 等新方法旨在用加法運算來取代乘法運算，以降低能耗。最近，MADDNESS 提出了一種創新的方法，將乘法和加法運算完全替換為查找和加法運算，這一方法顯示出顯著的性能提升。

參考論文採用以下幾種主要方法來優化 Transformer 模型：

- 查找-累加運算（LAC）：
  - LAC 運算通過查找預先計算的值來減少乘法操作，從而降低計算負擔。這一方法不僅提高了計算速度，還減少了能耗。
- 可微分查找矩陣乘法：
  - 這種方法允許在訓練過程中對查找表進行優化，使查找可以通過反向傳播進行調整[18]。這樣可以有效地提高模型的準確性，同時保持計算效率。
- 量化技術：

- 通過將模型參數從高精度浮點數轉換為低精度整數，可以顯著減少存儲需求和計算時間。量化不僅能提高推理速度，還有助於降低能耗。

- 知識蒸餾：
  - 知識蒸餾技術透過將大型模型的知識轉移至較小模型，來減少模型的大小和計算需求，同時保持其性能。
- 端到端訓練：
  - 本參考論文提出了一種端到端的訓練流程，將上述所有技術集成在一起，以實現更高效的 Transformer 模型訓練。

在 ImageNet 數據集上的實驗結果顯示，本參考論文提出的方法在準確率上相較於基線 LUT-NN 模型提高了最多 11%[18]。這顯示透過結合查找-累加運算與可微分查找矩陣乘法，可以顯著提升 Transformer 模型的性能與效率。

另外，此論文也顯示，透過查找累加運算及其他優化技術，可以顯著提升 Transformer 模型的性能潛力。未來的研究可進一步探討不同量化策略對各類任務的影響，並探索這些技術如何在各種硬體平台上實現以達到最佳效能。

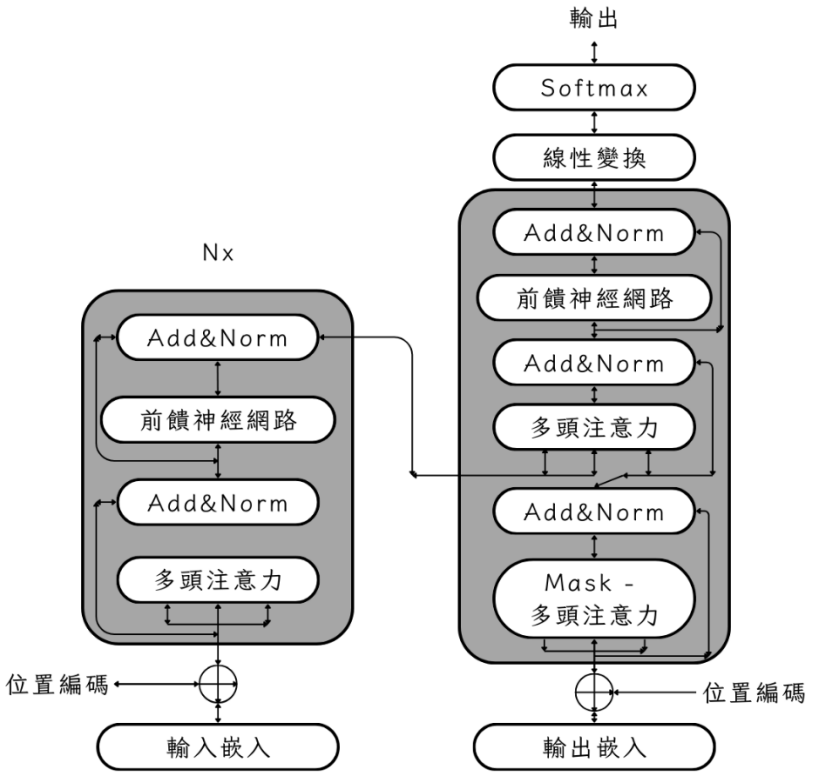

圖 7 Transformer 模型

表 5 有關 Transformer 在回歸預測相關文獻

作者	名稱	內容
周家興（2024）	可微分查找矩陣乘法用於壓縮 Transformer 網路	近年，研究者努力追求更高效的深度神經網絡，尤其是降低乘累加運算的計算量。傳統如知識蒸餾、剪枝、和量化的策略已被深入挖掘。由於乘法運算耗能問題，新策略如 AdderNet 和 ShiftCNN 應運而生，它們目標替換原有運算，從而節能。 不久前，MADDNESS 提出一全新策略，直接用查找-累加方法取代了乘累加運算。繼而有如 PECAN 和 LUT-NN 等研究也秉持此方向。我們的研究進一步完善了 LUT-NN，並提出了端到端的訓練方式。在 ImageNet 數據上的成果表明，我們的方法使 LUT-NN 的基礎準確率上升最多至 11%。
沈家丞（2024）	適用於人類誘導多能幹細胞之微模式明場顯微影像的 Transformer 細胞分割深度學習技術	本研究提出以 Transformer 技術為基礎之深度神經網路架構，BHM-UCT，用來直接對微模式明場顯微影像（Micropattern Brightfield Microscopy Images）進行細胞分割。在不需要使用 Hoechst 螢光染劑染色下，BHM-UCT 直接分割細胞可降低染劑對細胞成長的影響，並可達到高細胞分割精確度。

六、 預言家（Prophet）

預言家（Prophet）是由 Meta 團隊於 2017 年發表的時間序列預測模型，利用時間數據資料中的趨勢、季節、假日及誤差影響項進行預測。

預言家模型創建至今七年，研究多用於預測商品、股市等會隨時間變動的資料，表 6 為近期使預言家模型進行時間序列預測的相關文獻。

表 6 Prophet 應用於時序預測相關文獻

作者	名稱	內容
劉晴、董平軍	時間序列長度	使用 Prophet 模型對自動販賣機進行不同時

（2020）	對基於 Prophet 的自動販賣機銷量預測影響研究	間序列長度的銷售預測，同時考慮外部因素（天氣），使用平均絕對百分比誤差（MAPE）進行模型精確度評估。 結論：隨著時序長度增加，預測精確度隨之提高。
揚昇翰（2023）	台灣股市股票價格預測模型之比較：以 LSTM、GRU、Prophet 模型為例	透過使用長短期記憶（LSTM）、門控循環單元（GRU）和預言家（Prophet）來預測台灣不同產業的股票，並使用均方根誤差（RMSE）和平均絕對百分比誤差（MAPE）的損失函數來評估構建模型的準確性。 結論：在少量變數下，預言家模型表現高效能且穩定；在多量變數下，GRU 更具優勢。
Eber Romero（2023）	應用深度學習及時間序列分析技術於餐飲外送平台需求預測之比較研究	使用季節性自回歸整合移動平均法（SARIMA）、混合模型捲積神經網絡—長短期記憶法（CNN-LSTM）、長短期記憶法（LSTM）模型和預言家（Prophet）預測食品外送平台的配送訂單。運用均方根誤差（RMSE）和平均絕對百分比誤差（MAPE）的來評估模型預測的準確性。 結論：SARIMA 是可靠的比較基線，表現優於 CNN-LSTM、LSTM 和 Prophet；Prophet 是精度最高的方法，但 LSTM 是更好的實用選擇。
邱智清（2024）	時間序列套件及基於預處理的深度模型學習再加密貨幣價格預測之績效比較	比較了 ARIMA、Weka、Prophet 及深度學習機器(RNN、LSTM、BiLSTM 與 CNN+LSTM)在預測加密貨幣的表現。結果顯示，ARIMA 透過套件選擇最佳模型參數，出現過度擬合現象；Weka 學習器為隨機森林時，也出現過度擬合。Prophet 的初始誤差較大，但經過加入前七期收盤價作為回歸變數和自定義季節性後，表現有所提升。深度學習方面，對資料進行對數轉換和標準處理後，再訓練 CNN、LSTM 等模型，準確率明顯提升。因加密貨幣市場波動迅速，資料頻率的選擇至關重要。 結論：經過預處理的深度學習模型準確度高，而套件在使用適當參數進行預測時，也能取

		得不錯的結果甚至超越深度學習。

七、 ARIMA 模型（Autoregressive Integrated Moving Average model）

自我回歸整合移動平均模型（ARIMA）是由 George Box & Gwilym Jenkins 於 1976 年所提出。通過整合自回歸模型（Autoregressive, AR）和移動平均模型（Moving Average Model, MR），並加上差分（integrated, I）的概念使模型能觀察過往的資料與誤差，處理非穩態的資料。作為時間序列預測方法之一，應用於多領域中。表 7 為近年 ARIMA 應用在時序預測的相關文獻。

表 7 ARIMA 應用於時序預測相關文獻

作者	名稱	內容
李培煜（2020）	時間序列模型 ARIMA 與深度學習 LSTM 預測模型之比較：以台灣股票市場為例	比較 ARIMA 模型與 LSTM 模型在預測四檔不同特性的上市上櫃股票的差異，結果顯示，ARIMA 在預測隔一天的股價上表現較好。循環預測架構下的 LSTM 模型的 MSE 值則較低，適合中短期股價預測。增加 LSTM 神經網絡複雜度不能提高預測能力,推測是因資訊量過少產生雜訊影響預測結果。
余光倫（2022）	應用 ARIMA 模型對影響台灣工業生產指數之產業預測與分析	運用 ARIMA 模型建構 2000 年到 2021 年影響台灣工業生產指數之產業趨勢模型。包含 Dickey-Fuller Test、整合自迴歸移動平均 ARIMA，探討預測不同期數之誤差，透過預測誤差衡量方法之絕對百分 RMSE 求得最適參數。 結論：季節性 ARIMA 模型在短期的趨勢預測較有效率，能有效判定此案例的未來趨勢。
余奇潔（2013）	美元兌新台幣匯率預測－ARIMA 與 SVM 之應用	運用 ARIMA 預測模型、人工智慧 SVM 分類預測和 ARIMA-SVM 預測，研究 2010 年至 2012 年中央銀行新臺幣對美元銀行間成交之收盤匯率,最後計算各模型實際值與預測值之均方誤差、均方根誤差與平均誤差百分比。 結論：ARIMA 預測模型較 SVM 與 ARIMA-SVM 預測模型佳。
王百祿	ARIMA 與適應性	傳統 SVM 模型並未考量時間的影響，故本研

（2007）	SVM之混合模型於股市指數預測之研究	究將固定的ε係數調整為隨時間遞減的形式。將 ARIMA 模型預測道瓊工業指數的殘差項以下降ε支援向量機模型訓練，得到殘差項預測值與 ARIMA 模型的預測值加總即為混合模型之預測值。 結論：混合模型的預測效能及精確度，均較 ARIMA、SVM 與 ARIMA+SVM 為佳，可有效改善預測效能並減少預測誤差。

八、 支持向量迴歸（Support Vector Regression, SVR）

原始的支持向量機（SVM）演算法在 1963 年由蘇聯數學家弗拉基米爾·瓦普尼克（Vladimir Vapnik）和亞歷克塞·澤范蘭傑斯（Alexey Chervonenkis）共同發表。機器學習中 SVM 是一種強大的監督式學習模型，被廣泛應用於分類與迴歸分析等領域。它的獨特之處在於能夠有效地處理線性和非線性分類問題，並在許多實際應用中表現出色。

SVM 是一種非機率的二元線性分類器，透過將實例映射到空間中的點，並最大化類別間的邊界距離，以找到最優的分類超平面，使不同類別的實例能被盡可能廣泛地間隔開來，從而提高泛化能力。SVM 用於分類問題，基於相同原理，SVR 用於回歸問題，兩者皆著重於找到一個超平面以解決問題，並利用支援向量來定義超平面，使用者可根據任務的不同選擇適合的算法。透過隱式映射輸入數據到高維特徵空間，SVM 能更好地捕捉數據的非線性關係，從而提高分類的準確性和效率。表 8 為近年來有關 SVM 及 SVR 在銷售預測的相關文獻，對其應用與效能進行了詳細整理。

表 8 有關 SVM 在回歸預測相關文獻

作者	名稱	內容
陳俊宏、呂奇傑（2015）	應用與比較多種預測技術於光學膜產業銷售額預測	應用統計與機器學習技術，如天真預測法（Naïve forecast）、逐步迴歸法（SR）、反向傳播神經網路（BPN）、支援向量迴歸（SVR）和極限學習機（ELM）等，對光學膜產業的銷售額進行了預測。結論：在三家光學膜相關公司的數據中，支援向量迴歸無論是針對單期預測還是多期預測，都呈現出最佳的預測效能。
林芳毅（2015）	Hadoop 結	利用 Hadoop 的平行分散式處理技術與 R 語言

	合R之資料探勘－以TH公司產品銷售預測為例	的統計軟體演算，來探討大數據的儲存、處理、及檢索之功能。利用 RHadoop 的倒傳遞類神經網路、C-SVC 支援向量機與指數平滑法三種模型進行預測。
A.L.D Loureiro, V.L. Miguéis, L.F.M. da Silva（2018）	Exploring the use of deep neural net-works for sales fore-casting in fashion retail	比較深度學習 DNN 與機器學習包含決策樹（DT）、隨機森林（RF）、支持向量回歸（SVR）、人工神經網絡（ANN）和線性回歸（LR）。 結論：研究發現 DNN 在時尚零售市場的銷售預測方面表現良好，然而，在考慮的部分評估指標方面，其性能並不顯著優於某些較簡單的模型，特別是隨機森林（RF）。
Austin Schmidt, Md Wasi Ul Kabir, & Md Tamjidul Hoque（2022）	Machine Learning Based Restaurant Sales Forecasting	使用了超過 20 種，涵蓋 Linear Models、Decision Tree Regressor Models、Support Vector Models、Neural Networks、Recurrent Neural Networks 等模型，以展示創建靜態數據集對特徵工程和模型訓練過程的影響。 結論：循環神經網絡（RNN）類別的模型表現優於其他模型。

九、 決策樹回歸（Decision Tree Regressor）

回歸樹是一種廣受歡迎的機器學習方法，由 J. Ross Quinlan 在 1980 年代初期發展，對現代回歸樹算法的發展產生了深遠影響。Quinlan 的著名算法包括 ID3 和 C4.5，這些算法至今仍是學術研究和實際應用中的重要基石(Quinlan, 1986; Salzberg, 1994)。回歸樹不僅易於應用，還能在有限的算力下產生良好的結果，使其成為一種廣泛使用的機器學習技術 (Layfield & Neri, 2023)。

回歸樹透過樹狀結構來組織分類規則，其中每個分枝代表決策過程中的一個邏輯判斷，而每個葉節點代表一個分類決策。這種結構使得回歸樹適合處理具有明顯邏輯分層的數據 (Layfield & Neri, 2023)。

在銷售預測領域，回歸樹能有效的處理分類特徵和捕捉非線性關係，這使其能夠有效預測特定市場趨勢(Johannes & Alamsyah, 2021)。然而，回歸樹存在過適的風險，尤其是在樹的深度很大或分枝過多時。為了解決此問題，分枝修剪技術被用來避免模型過度學習特定的訓練數據 ("Avoiding Overfitting of

Decision Tree ," 2007)。

表 9 有關 Decision Tree Regressor 在回歸預測相關文獻

作者	名稱	內容
萬絢、陳婷瑜、謝雯如、陳熹蓉、黃舒雅 (2019)	線性判別分析和決策樹台灣半導體股票投資組合報酬率之實證	此研究探討線性判別分析(LDA)和決策樹在台灣半導體股票投資的應用，透過這兩種模型建立一套投資決策支援系統。研究分析了 2015 年至 2016 年間 50 家半導體公司的數據，並選擇 50 家同性質產業的公司驗證模型的有效性。 結論：使用決策樹的投資模型能達到 68%的預測正確率，而 LDA 模型則有 58%的正確率。在投資報酬率方面，決策樹模型達到 20.43%，LDA 模型則達到 25.53%。雖然決策樹在預測正確率上優於 LDA，但在投報率上略低。
蘇甲辛 (2018)	應用決策樹預測傳統通路銷售潛力之研究-以雲嘉地區傳統菸酒零售通路為例	面對激烈的市場競爭，台灣菸酒公司利用決策樹模型來優化傳統通路的管理。模型利用 2017 年的銷售數據，包括烈酒、啤酒和菸品的銷售額，以及通路配合度和店家發展潛力，以預測不同通路的銷售潛力。 結論：銷售額是影響通路分級的主要因素，並證明此模型可以讓台灣菸酒公司有效的投資具潛力的通路，以提升市佔率。
Pradip Kumar Bala (2010)	Decision Tree Regressor based demand forecasts for improving inventory	本文探討利用決策樹預測零售銷售需求，並將客戶分類知識作為輸入資訊。此研究提出的模型已在零售業中被實際應用，用於改善整體供應鏈管理中的庫存表現。此外，該預測模型與庫存

		補貨系統結合，成功降低了庫存量並提升客戶服務品質。文章進一步指出，這種系統比超商現行的模型更精確的預測未來需求，提供全面且有效的營運對策，以增加零售商的盈利。
R. Johannes, A. Alamsyah (2021)	Sales Prediction Model Using Classification Decision Tree Regressor Approach For Small Medium Enterprise Based on Indonesian E-Commerce Data	這項研究探討了印尼鞋業的銷售預測。利用鞋業的實際數據，在印尼最大的電子商務提供商上建立了銷售預測模型。透過資料探勘中的分類方法，預測商品在國內各個地區的銷售潛力。當中，重要的決策變量包括「鞋子類型」、「觀看量」和「價格」。這個模型成功的預測了銷售量，為印尼的中小企業提供了實用的銷售預測工具。

十、 彙整比較表格

表 10 此篇論文與部分參考論文之比較

論文名稱	使用模型	研究方向
Backpropagation Applied to Handwritten Zip Code Recognition [5]	反向傳播算法下的卷積神經網絡（CNN）	手寫郵政編碼識別系統
Recurrent neural networks [10]	反向傳播算法下的循環神經網絡（RNN）	指出 RNN 在多種應用中表現出色，包括機器翻譯、情感分析、音樂生成等。這些應用展示了 RNN 在理解和生成序列數據方面的潛力
應用 CNN-LSTM 深度學習模型於預測台灣二氧化碳排放量 [12]	CNN 和 LSTM 結合	碳排放預測
利用 LSTM、RNN 與 GRU 模型預測比特幣價格：一個深度學習方法的效能分析 [15]	LSTM、GRU、RNN	比特幣價格預測

可微分查找矩陣乘法用於壓縮 Transformer 網路 [18]	Transformer	提高對 hiPSCs 的細胞分割準確性，以改善細胞影像分析的效率
本研究	CNN、RNN、LSTM、GRU、Transformer	找出預測北京 $PM_{2.5}$ 誤差最小的模型

第三章　研究方法

第一節　資料集

Beijing $PM_{2.5}$ 資料集是由中國北京市多個空氣品質監測站所提供，該資料集用於研究空氣品質，特別是大氣中細顆粒物（$PM_{2.5}$）的濃度。資料集包含了數年的每小時數據，涵蓋多項空氣品質參數。

資料集的特徵包括：

- pm2.5：目標變數，表示大氣中直徑小於 2.5 微米的顆粒物濃度。
- 溫度：空氣溫度測量值。
- 濕度：空氣的相對濕度。
- 氣壓：大氣壓力數據。
- 風速：監測站位置的風速數據。
- 太陽輻射：光照強度，以豪斯計（lux）為單位。
- 降水量：降水量數據。
- 月份：標示測量的月份，作為時間特徵。

該資料集廣泛應用於機器學習領域，尤其是在時間序列預測和空氣污染預測模型中。它對於研究和預測城市空氣質量、$PM_{2.5}$ 濃度的變化因素具有重要價值。

第二節　資料處理

一、 資料前處理

在這段資料前處理程式中，本實驗針對北京 $PM_{2.5}$ 資料集進行了以下處理步驟，以確保資料的可用性和準確性，並且對空氣品質的預測模型提供了清晰的基礎。

- 讀取資料集：資料集是以 CSV 格式存儲，包含了各種與空氣品質有關的變數，包括 $PM_{2.5}$ 濃度、露點溫度、氣溫、氣壓、風速、風向、降雪量和降水量等。首先使用 read_csv 函數讀取資料並進行日期解析，將 year, month, day, hour 四個欄位合併為單一的 date 索引，並使用自訂的 parse 函數將這些欄位轉換為 datetime 格式，確保資料能夠正確處理為時間序列。

- 資料清理與欄位命名：為了進行分析，丟棄了不必要的「No」欄位，並重新命名資料集中的其他欄位以提高可讀性。這些欄位包括 pm2.5（$PM_{2.5}$ 濃度）、dew（露點溫度）、temp（氣溫）、press（氣壓）、wnd_dir（風向）、wnd_spd（風速）、snow（降雪量）和 rain（降水量）。此外，為了處理缺失值，對 pm2.5 欄位進行了填補，將缺失值設為 0。
- 資料刪除：為了刪除可能影響模型準確度的數據，本實驗移除了資料集的前 24 小時數據，因為這段時間內的 $PM_{2.5}$ 值為 0，這有可能是數據錄入錯誤或其他原因所造成的。
- 標籤編碼：使用 LabelEncoder 將類別型變量轉換為數值格式。
- 正規化：使用 StandardScaler 將所有特徵標準化至[0, 1]範圍，以提高模型收斂及運行的速度，也可以節省程式運作時占用的記憶體容量。
- 刪除 Outlier：透過統計學中 IQR 方法刪除；首先，我們計算了資料的第一四分數（Q1）及第三四分數（Q3）並利用四分位距 (IQR = Q3 - Q1) 定義合理範圍：

$$下界 = Q1 - 1.5 \times IQR \quad (1)$$

$$上界 = Q3 + 1.5 \times IQR \quad (2)$$

定義範圍後，利用篩選條件過濾掉超出上下界的異常值。

二、 特徵工程

在特徵工程部分，本研究進行了多步驟設計，以確保模型能夠有效學習時間序列資料中的複雜依賴關係。資料轉換的核心是自訂函數 series_to_supervised()，該函數旨在將原始時間序列資料轉換為適合監督學習的格式。具體來說，該函數將前幾個時間步驟的特徵轉換為輸入變量，而未來的時間步驟則用於建立目標標籤。這種轉換方法可以幫助模型從歷史資料中學習並預測未來趨勢。

在 series_to_supervised()的實作中，為每個特徵建立了一個滯後變量，以捕捉時間上的延遲關係。例如：設定 n_in=3，則將產生三個時間步驟 t-3、t-2 和 t-1 的資料作為輸入，並將 t 作為目前時間步驟的目標。這不僅使模型能夠分析短期波動，還有助於捕捉更長期的趨勢，而在我們的實驗中，我們將 n_in 初始設為 1。

此外，為了避免資料遺失導致模型效能下降，產生的資料框中會移除所有空

值。這個過程確保在模型訓練和測試階段使用完整、一致的資料。經過此階段的資料變換，資料集結構使得模型能夠充分利用時間序列的複雜性進行預測。

三、 重複執行模型

了確保模型評估的結果具有更高的穩定性和代表性，我們採用了重複執行的策略，即透過迴圈多次訓練和測試模型。在實際應用中，單次模型訓練與測試可能因為數據切割方式的不同而導致結果有所偏差，特別是在數據集規模較小或數據分佈具有異質性的情況下。為了避免因單次執行可能出現的隨機性和數據切割對模型性能的影響，我們在程式中使用了 for 迴圈重複執行模型的訓練與預測共十次，並將每次迴圈計算出的誤差值（包括 MAE、RMSE 及 MAPE）儲存在對應的陣列中。此外，這種方法還可以幫助我們發現模型的泛化能力，從而更全面地評估其在處理未見數據時的預測能力。

四、 標準化處理

為了盡可能縮小模型預測中的誤差值，我們對數據的標準化過程進行了多次測試與優化。數據的標準化處理是機器學習模型訓練中的重要步驟之一，其目的是將特徵值轉換到一個統一的尺度上，從而提升模型的收斂速度並減少由特徵值量級差異引起的偏差。考慮到不同的標準化方法對模型性能可能產生顯著影響，我們嘗試了多種常見的標準化模式，包括 StandardScaler、MinMaxScaler、RobustScaler、MaxAbsScaler 以及 QuantileTransformer（分別測試其正態分佈與均勻分佈設定）等等。

- StandardScaler：該方法將數據的均值移至 0，標準差縮放至 1，是處理特徵數據較為常見的標準化方法。它假設數據分佈為高斯分佈（正態分佈），在我們的實驗中表現穩定。
- MinMaxScaler：此方法將數據壓縮到一個固定的範圍（通常是[0,1]）。這種方法在特徵值分佈較均勻時效果良好，但對於具有異常值（outliers）的數據可能會受到影響。
- RobustScaler：相較於其他方法，該方法基於數據的中位數和四分位距進行標準化，因此對異常值的影響較小。在處理具有較多異常值的數據集時，我們觀察到該方法有明顯優勢。
- MaxAbsScaler：此方法對數據進行絕對值縮放，將數據壓縮到[-1,1]範圍，

適用於稀疏數據的處理。

- QuantileTransformer：該方法可以將數據轉換為具有特定分佈的樣本，無論是正態分佈還是均勻分佈，均可適應不同應用場景。通過測試，我們發現其能有效處理偏態分佈數據。

在每種標準化方法下，我們分別對訓練過程進行了多次測試，並比較模型在驗證集和測試集上的性能指標（如 MAE、RMSE 和 MAPE）。這種實驗設計不僅幫助我們確定了最適合當前數據集的標準化方式，也為模型性能的進一步提升提供了重要的參考依據。經過多次比較，最終為單純以模型進行橫向評估，因此，我們一致採用 StandardScaler 作為標準化處理。

五、 拆分訓練集與測試集

本研究採用時間序列拆分策略，根據資料的時間先後順序進行劃分，避免未來數據「洩漏」到訓練集，確保模型能在真實場景下預測未來數據。

在資料的拆分比例與分配上，本研究將資料集劃分為訓練集與測試集，比例設定為「『全部資料筆數減去最後 168 筆』比最後 168 筆」，意即預測最後七天每天每小時的數據。其中，訓練集包含資料集的全部資料筆數減去最後 168 筆，即歷史較為靠前的數據，這些數據用於模型的學習與參數調整。模型通過學習訓練集數據，建立輸入特徵與目標變數之間的映射關係，並不斷優化其內部參數。測試集則包含後 168 筆的數據，即時間序列中較新的部分，用於檢驗模型在未見數據上的表現。測試集的評估能夠揭示模型是否具備良好的泛化能力，並判斷其在實際應用場景中的準確性與穩定性。

在劃分過程的細節方面，本研究主要考慮以下兩點：

- 按時間順序劃分：由於時間序列資料具有強烈的順序性，資料必須按照時間的先後順序進行劃分，確保測試集數據僅包含「未來」的觀測值。這樣可以更真實地模擬實際預測場景，避免因隨機拆分打亂時間依賴性而導致模型性能評估失準。
- 避免數據洩露：在訓練集與測試集之間設置適當的時間間隔，避免過於接近的數據相互影響，進一步降低數據洩露的風險，確保模型對未知數據的預測能力能被更準確地評估。

訓練集與測試集的作用也不容忽視。訓練集是模型構建的核心，其數據質量與

規模直接影響模型的學習能力，包括參數學習、超參數調整及模型結構的優化。測試集則不參與模型訓練，而是作為模型性能最終評估的重要依據，能真實反映模型在處理未見數據時的表現，並為後續的模型選擇與改進提供參考。

透過上述方法，本研究有效避免時間序列數據拆分中的常見問題，確保模型訓練與測試的科學性與可靠性。

第三節　模型架構

為了找出對於我們資料集解釋度最高的模型，我們採用了五個不同的模型架構作為基礎，並在每個架構中進行了進一步的模型改良與優化，這些優化包括單獨使用或混合使用以下策略：堆疊模型（Stacked Model）、雙向模型（Bidirectional Model）、堆疊雙向模型（Stacked Bidirectional Model）、以及多策略混合模型（Hybrid Model）。此設計旨在探索不同結構對於模型性能的影響，以確保在不同設置下可以捕捉到資料集中更多的特徵模式，並進一步提升預測的準確性和穩定性。

首先，堆疊模型是一種通過在模型中引入多層結構以加強對數據深度特徵捕捉的策略。藉由增加層數，我們希望模型能夠學習到更高層次的抽象特徵，從而提高解釋能力。同時，我們控制層數增長以防止過度擬合。

其次，雙向模型的引入是為了更全面地學習數據序列中的前後文信息。此結構特別適用於時間序列或語義相關的數據，因為它能夠同時捕捉過去（前向傳播）和未來（反向傳播）的資訊，進一步提升模型對數據內部相關性的理解能力。

此外，堆疊雙向模型將上述兩種策略結合，通過多層雙向結構的設計，不僅增加了模型的深度，也提升了對時序數據的上下文捕捉能力，這對於長序列數據的處理尤其有效。我們在設計過程中特別關注了每層的權重初始化與正則化策略，以保證計算效率與模型的收斂性。

在實驗中，我們對五種不同的深度學習架構（CNN、RNN、LSTM、GRU、Transformer）分別應用上述優化策略，並詳細記錄了每種設計在性能指標（如 MAE、RMSE 和 MAPE）上的表現。透過這些分析，我們希望能更全面地比較不同模型架構在處理特定資料集時的表現優勢。

一、 卷積神經網絡 (Convolutional Neural Network, CNN)

CNN 在提取空間特徵方面具有優勢，尤其適合處理高維數據或具有強烈局部相關性的資料。通過堆疊與混用策略，CNN 的深層結構可以捕捉更抽象的特徵，提升模型的解釋能力。我們主要使用了以下結構處理時間序列的回歸任務：

- 模型透過 Conv1D 處理時間序列數據，結構如下：
  - 卷積層 (Conv1D)：
    - 使用一維卷積，從特徵中提取局部模式。
    - 激活函數默認為 ReLU (relu)。
    - input_shape 指定輸入數據的形狀，為 (X.shape[1], 1)。
  - 扁平化層 (Flatten)：
    - 將卷積層的輸出展平成一維張量，以便進行全連接層的處理。
  - 全連接層 (Dense)：
    - 單個神經元，用於輸出回歸任務的預測值（PM2.5）。
- 模型編譯：
  - 優化器：adam
  - 損失函數：mean_squared_error（均方誤差）
  - 評估指標：mae（平均絕對誤差）

在處理主要模型外，我們運用了網格搜尋及多層迴圈計算每組超參數的模型表現（MAE 和 RMSE），尋找卷積核數量、大小及最佳超參數的最佳組合。

二、 循環神經網路（Recurrent neural network, RNN）

RNN 在序列數據的處理上表現出良好的基本能力，但其記憶長序列的效果受到梯度消失問題的限制。雙向與堆疊策略對 RNN 的改進效果顯著，有助於捕捉前後依賴性更強的特徵。本程式設計了一個以 SimpleRNN 為基礎的深度學習模型來預測 $PM_{2.5}$ 濃度。具體結構如下：

- SimpleRNN 層
  - 單層 SimpleRNN（64 個單元），用於處理時序數據並提取特徵。
  - 輸入形狀為 (n_hours, n_features)，即每個樣本包含 n_hours 歷史步長的 n_features 個特徵。

- 全連接層（Dense）
  - 第一層 Dense：512 個神經元，使用 ReLU 激活函數。
  - 第二層 Dense：128 個神經元，使用 ReLU 激活函數。
  - 第三層 Dense：1 個神經元，無激活函數，輸出預測的 $PM_{2.5}$ 濃度。
- 模型編譯
  - 損失函數：mae（平均絕對誤差）。
  - 優化器：adam。

三、 長短期記憶 (Long Short-Term Memory, LSTM)

LSTM 改進了 RNN 在處理長序列時的記憶能力，對時間序列特徵具有更強的解釋度。特別是在堆疊雙向設計中，LSTM 能夠同時捕捉資料的遠程依賴性和上下文特徵，對提升模型性能有明顯作用。我們設計了一個以 LSTM 為基礎的深度學習模型來預測 $PM_{2.5}$ 濃度。具體結構如下：

- LSTM 層：採用 64 個單元的 LSTM 層，處理時間序列的特徵。
- Dense 全連接層：
  - 第一層：包含 512 個神經元，激活函數為 ReLU（relu），負責提取高階特徵。
  - 第二層：包含 128 個神經元，激活函數為 ReLU，進一步壓縮特徵。
  - 第三層：包含 1 個神經元，輸出最終的預測結果。
- 損失函數與優化器：
  - 損失函數：平均絕對誤差（MAE）。
  - 優化器：Adam，自適應學習率優化方法。

四、 門控循環單元 (Gate Recurrent Unit, GRU)

GRU 相較於 LSTM，結構更為簡潔，計算效率更高。在處理短序列數據時，GRU 表現出更好的效率，而其性能與 LSTM 在多數實驗場景中表現相當。我們的模型利用 GRU 擅長捕捉序列時間依賴的特性，並通過全連接層進一步擴展學習能力，以提高對 $PM_{2.5}$ 時間序列的預測準確性，具體結構如下：

- 模型架構：
  - 第一層：GRU 隱藏層，包含 64 個單元，輸入維度為 (n_hours,

n_features)。

- 第二層：全連接層，包含 512 個神經元，使用 ReLU（relu）激活函數。
- 第三層：全連接層，包含 128 個神經元，使用 relu 激活函數。
- 最後一層：全連接輸出層，包含 1 個神經元，用於單值回歸。

- 編譯與配置：
  - 損失函數：mae（平均絕對誤差）。
  - 優化器：adam。

五、 Transformer 模型

Transformer 是一種基於注意力機制的架構，擅長處理長距離依賴和多維特徵。透過混用策略結合堆疊與注意力機制，我們能顯著提升其對於特定資料集的解釋能力。尤其在高維數據與大規模資料下，Transformer 展現了極高的準確率與穩定性，具體結構如下：

- Transformer 層：採用 64 個單元的 Transformer 層，處理時間序列的特徵。
- 定義網格參數 (param_grid)
  - 包含超參數的多種組合，例如注意力頭數 (num_heads)、鍵的維度 (key_dim)、前向神經網絡的維度 (ff_dim)、丟棄率 (dropout_rate) 等。
- 構建 Transformer Encoder
  - 自定義了一個 Transformer Encoder 模塊，內含多頭注意力機制、殘差連接、前向神經網絡層及正則化操作。
- 網格搜索

  遍歷所有參數組合：
  - 構建模型並訓練。
  - 預測並將結果反轉縮放。
  - 計算指標（MAE、RMSE、MAPE）以評估模型性能。

六、 預言家（Prophet）

據提出此模型此團隊所述，預言家模型是由四個函數所組成的加法模型：

$$y(t) = g(t) + s(t) + h(t) + \varepsilon t \quad (3)$$

在式 3 中，g(t)代表趨勢項、s(t)表季節項、h(t)表假日項、ε t 表誤差項，下列四點為四項函數的代表意義及其計算方法：

- 趨勢項

  資料呈現為非週期性的變化趨勢，可以依照資料預測是否會飽和來調整使用的趨勢函數。

  - 非飽和預測模型(Saturating Growth Model)

$$g(t) = k \cdot t + m \quad (4)$$

  - 飽和模型預測 (Piecewise Logistic Growth Model)

$$g(t) = \frac{C(t)}{1 + \exp(-k \cdot (t - m))} \quad (5)$$

  C(t)為不同時間點的最大容量，有些資料上限會隨時間而有變動。

- 季節項

  欲預測資料隨著時間而呈現週期性變化的相關函數。

$$s(t) = \sum_{n}^{N} (a_n \cos(\frac{2\pi nt}{P}) + b_n \sin(\frac{2\pi nt}{P})) \quad (6)$$

  P 為週期長度(ex.365、30、7)，控制 a_n、b_n 可獲得需要的週期。

- 假日項

  欲預測資料可能因特定時間(非週期性)而呈現變化的相關函數，

$$\left(h(t) = \sum_{i}^{D} \kappa_i \cdot 1(t \in D_i) \right) \quad (7)$$

  每種假日 D_i 都會對欲預測資料帶來不同影響。

- 誤差項

  其他模型沒有預測到的波動皆屬於誤差項。

預言家模型即是透過擬合上述四項函數得到時間序列預測值。根據模型特性，預言家雖無法透過更多面的資訊預測產品銷售資料（天氣狀況、商品價

格、門市分布情況)，但在時間序列分析上已提供豐富的預測能力。

七、 ARIMA 模型（Autoregressive Integrated Moving Average model）

自我回歸整合移動平均模型（ARIMA）結合了自回歸模型（AR）、差分（I）和移動平均模型（MA）三個組件。其中一個重要的組成部分「自回歸模型（AR)」，是時間分析序列中常見的一種模型，AR 模型只考慮時間序列的自回歸結構，即將當前值與先前的值相關聯，通過自回歸係數（AR coefficient）來捕捉時間序列的自相關結構，其係數的大小和符號反映時間序列的特性和動態變化。

該模型基於時間序列數據，假設當前觀測值與前期觀測值存在一定線性關係，通常表達為當前觀測值是前期觀測值的線性組合，其基本形式可表示為：

$$x_t = C + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \phi_3 X_{t-3} + \cdots + \phi_p X_{t-p} + \epsilon_t \quad (8)$$

式 8 中，X_t 是當前觀測值；C 為常數項，決定過程的平均值；$\phi_1, \phi_2, \ldots\ldots, \phi_p$ 稱為自我回歸參數，代表影響程度，通常介於 1 ~ -1 之間，越接近當前時間點影響程度越大；ϵ_t為誤差值，是無法用已知變數解釋的部分。

與 AR 模型不同，MA 模型專注於過去觀察到的隨機誤差項的加權總合，假設此隨機誤差是獨立並且符合常態分配，時間序列中的每個值都是過去觀察到的隨機誤差的線性組合，捕捉時間序列中的隨機波動。可表示為：

$$r_t = C + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \theta_3 \epsilon_{t-3} + \cdots + \theta_q \epsilon_{t-q} \quad (9)$$

式 9 中，rt 是預測值；θ 是移動平均參數。

AR 模型通過捕捉前期觀測值與當前值之間的相關性來建立模型，MA 模型則通過捕捉隨機波動的影響來建立模型。然而當時間序列是非平穩的時候，可能會使上述兩種模型失效，因此可透過差分將相鄰時間點的觀測值相減，從而消除趨勢和季節性等影響，有助於將非平穩的時間序列轉換為平穩序列，其方法為：

$$\nabla Y_t = Y_t - Y_{t-1} = (1 - L) Y_t \quad (10)$$

式 10 中，L 為滯後算子（Lag operator)。如果$\nabla^d = (1 - L)^d$則稱為 d 階差

分。需要注意的是，若差分超過二次可能導致資料變異數變大且不易解釋。

將 AR 模型、MA 模型和 I 結合可得 ARIMA 模型，記成 ARIMA(p, d, q)，可表示為下列算式：

$$\left(1-\sum_{i=1}^{p}\varphi_i L^i\right)(1-L)^d X_t = (1+\sum_{i=1}^{q}\theta_i L^i)\varepsilon_t \tag{11}$$

在式 11 中，p 為 AR(p)的階次、d 為差分階次、q 為 MA(q)的階次、L 為滯後算子。

透過 ARIMA 模型，我們可以考慮時間序列中自相關結構（AR）、隨機波動（MA）以及非穩態性等因素，從而更好的進行時間序列的建模和預測。

八、 支持向量迴歸（Support Vector Regression, SVR）

在機器學習領域，支援向量機（Support Vector Machine，SVM）和支援向量迴歸（Support Vector Regression，SVR）是兩個重要的方法，分別應用於分類和回歸問題。兩種方法基於相似的原理，但適用於不同的任務類型。

分類和回歸是機器學習中兩種不同的任務。在分類問題中，目標是將數據實例劃分到不同的類別或標籤中，其輸出是離散的類別標籤。常見的分類算法包括 SVM、決策樹、隨機森林和邏輯回歸等。而在回歸問題中，則是預測連續值的輸出，其輸出是連續的數值。常見的回歸算法包括線性回歸、SVR、決策樹回歸和神經網絡等。。在研究方法方面，針對不同的問題類型，可運用不同的算法進行分析與研究。

支援向量機（Support Vector Machine，SVM）是一種基於統計學習理論基礎的機器學習模型，針對小樣本、非線性、高維度與局部最小點等問題具有相對的優勢。這個概念其實早在 1960-1990 年代就由數學家 Vapnic 及 Chervonenkis 等人所提出，並建立了這套統計學習理論。除了在文字分類、圖像分類及醫學中分類蛋白質等領域有不錯的成效外，因具有計算速度快且空間成本低等優勢，在工業界也有廣泛的應用。具象化來說，SVM 就是將在低微度空間線性不可分的樣本映射到高維度空間去，找到一個超平面將這些樣本做有效的切割，且這個超平面兩邊的樣本要盡可能地遠離這個超平面（坡度最大化）。

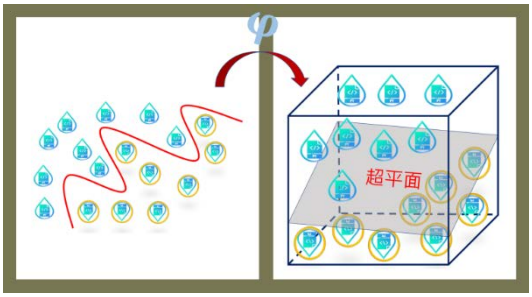

圖 8 分類並轉換為超平面示意圖

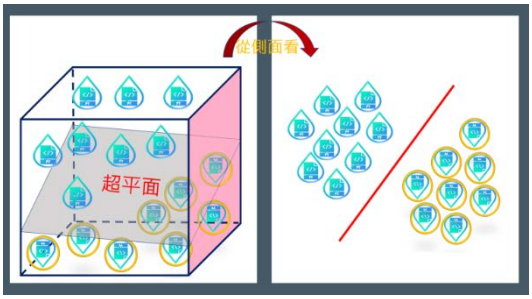

圖 9 決定分類線坡度示意圖

另一方面，支援向量迴歸（SVR）則是用於回歸問題。與 SVM 類似，SVR 也是通過找到一個超平面，但其目標是使得數據點與該超平面的距離盡可能小，同時保證超平面兩側的誤差都在一個可接受的範圍內。支援向量在 SVR 中扮演著重要角色，它們是那些與預測函數距離最接近的數據點，用於限制預測誤差。同樣地，SVR 可以處理線性和非線性回歸問題，並且也可以通過核函數擴展到高維空間中。

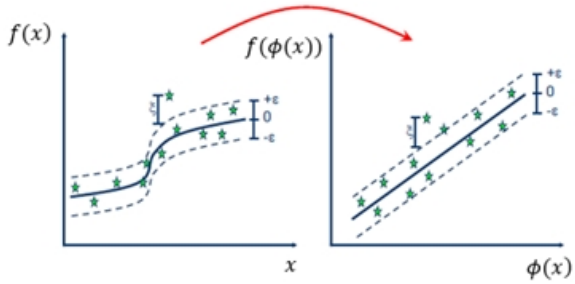

圖 10 SVR 非線性轉換示意圖

給定 SVM 一組訓練實例，每個實例都被標記為屬於兩個類別中的其中一個，SVM 訓練算法會建立一個模型，將新實例分配給兩個類別之一，形成一個非概率的二元線性分類器。SVM 模型將實例表示為空間中的點，通過這種映射，使得單個類別的實例被盡可能寬的間隔分開。然後，將新實例映射到同一

空間，並根據它們落在間隔的哪一側來預測其所屬類別。

除了進行線性分類之外，SVM 還可以使用核技巧有效地進行非線性分類，將輸入隱式映射到高維特徵空間中。

在資料未被標記時，無法進行監督式學習，需要使用非監督式學習。非監督式學習試圖找出資料的自然聚類，並將新資料映射到這些已形成的簇中。改進的支持向量機聚類算法被稱為支持向量聚類，在工業應用中常用作分類步驟的預處理。

更正式地說，支持向量機在高維或無限維空間中構造超平面或超平面集合，可用於分類、回歸或其他任務。直觀上，分類邊界距離最近的訓練資料點越遠越好，因為這樣可以縮小分類器的泛化誤差。

儘管原始問題可能是在有限維空間中陳述的，但用於區分的集合在該空間中往往線性不可分。為此，人們提出將原有限維空間對映到維數高得多的空間中，在該空間中進行分離可能更容易。為了保持計算負荷合理，人們選擇適合該問題的核函式來定義 SVM 方案使用的對映，以確保用原始空間中的變數可以很容易計算點積。高維空間中的超平面定義為與該空間中的某向量的點積是常數的點的集合。通過選擇超平面，被對映到超平面上的特徵空間中的點集由以下關係定義：

$$\sum_i a_i k(x_i, x) = constant \tag{12}$$

在這種情況下，我們希望找到一個超平面，能夠將給定的點集分成兩個類別，並使得這個超平面與最近的點之間的距離最大化。

超平面的一般方程式如下：

$$w \cdot x + b = 0 \tag{13}$$

在式 13 中，w 是法向量（可以不用歸一化）、b 是偏移量。對於每個樣本點 x_i，其對應的類別標籤 y_i 為 1 或-1。

支持向量機的目標是找到這樣一個超平面，使得對於所有的樣本點 x_i，滿足以下條件：

$$y_i(w \cdot x_i + b) \geq 1 \tag{14}$$

同時最大化 $\frac{2}{\|w\|}$ ，這樣做可以確保超平面到兩個類別中最近的樣本點的距離最大化。

如果資料是線性可分的，我們可以選擇兩個平行超平面來分隔兩類資料，使得它們之間的距離最大化。這兩個超平面之間的距離稱為「間隔」，而最大化這個間隔的超平面則稱為最大間隔超平面。這些超平面可以用方程式來描述：

$$\vec{w} \cdot \vec{x} - b = 1 \tag{15}$$

或是

$$\vec{w} \cdot \vec{x} - b = -1 \tag{16}$$

通過幾何推導，我們可以得出兩個超平面之間的距離。因此，為了使兩個平面之間的距離最大化，我們需要最小化超平面的法向量的範數，即超平面的斜率。同時，為了確保所有樣本點都位於超平面的間隔區以外，我們需要滿足以下條件之一：

$$\vec{w} \cdot \vec{x} - b \geq 1, 若\, y_i = 1 \tag{17}$$

或是

$$\vec{w} \cdot \vec{x} - b \leq -1, 若\, y_i = -1 \tag{18}$$

也就是說，每個資料點都必須位於間隔的正確一向。而這兩個式子也可以寫作：

$$y_i(\vec{w} \cdot \vec{x} - b) \geq 1, \qquad for\ all\ 1 \leq i \leq n \tag{19}$$

為了將 SVM 擴充到資料線性不可分的情況，我們引入鉸鏈損失函式，

$$\max(0, 1 - y_i(\vec{w} \cdot \vec{x} - b)) \tag{20}$$

當約束條件式 19 滿足時（也就是如果位於邊界的正確一側），對應的函式值為零。對於位於間隔錯誤一側的資料，該函式的值與到間隔的距離成正比。我們的目標是最小化：

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0,1-\ y_i(\vec{w}\cdot\vec{x}-b)\right)\right]+\lambda\|\vec{w}\|^2 \tag{21}$$

參數λ用來平衡增加間隔大小和確保$\vec{x_i}$位於正確間隔一側之間的關係。因此，對於足夠小的λ，如果資料可以線性分類，軟間隔支援向量機（SVM）與硬間隔支援向量機（SVM）將表現相同。即使資料不可線性分類，也能夠學習出合理的分類規則。

總而言之，支援向量機（SVM）和支援向量迴歸（SVR）是基於相似的原理，但應用於不同的問題類型，其區別在於輸出的類型。在研究方法方面，可以根據具體的問題需求和數據特徵，選擇適合的算法進行分析與研究，以提高模型的準確性和預測能力。為了因應我們的主題，我們對於銷售量的預測將使用支援向量迴歸（SVR）。

九、 決策樹回歸（Decision Tree Regressor）

回歸樹（Decision Tree Regressor）的建構過程包括特徵選擇、樹的建構和樹的剪枝：

- 特徵選擇

  從當前數據的特徵中，找出最佳分割特徵作為節點的劃分標準。此步驟通過評估每個特徵的資訊增益（ID3），資訊增益率（C4.5）或 Gini 係數（CART），以衡量分割數據後純度或資訊量的變化。數學式如下：

  - ID3
    - 熵 H(D)公式：

$$\mathrm{H(D)}=-\sum_{k=1}^{k}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|} \tag{22}$$

    - 條件熵 H(D|A) 公式：

$$\mathrm{H(D|A)}=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i) \tag{23}$$

    - 資訊增益公式：

$$Gain(D,A)=H(D)-H(D|A) \tag{24}$$

- C4.5：為了解決 ID3 傾向於選擇擁有更多類別的特徵，通過使用資訊增益率來減少特徵偏好，進而提高回歸樹的泛化能力。
  資訊增益率公式：

$$SplitInformation(D,A) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|} \tag{25}$$

$$GainRatio(D,A) = \frac{Gain(D,A)}{SplitInformation(D,A)} \tag{26}$$

- CART（Classification and Regression tree）：不同於 ID3 或 C4.5 可能產生多個分支的情況，Gini 係數每次分割僅產生兩個節點，這有助於簡化決策過程、減少了模型複雜度，以提高計算效率。
  - Gini Impurity 公式：

$$Gini(D) = 1 - \sum_{k=1}^{k} p_k^2 \tag{27}$$

  - Gini 係數公式：

$$Gini\ index(D,A) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} Gini(D_i) \tag{28}$$

  - Gini Gain 公式：

$$Gini\ Gain(D,A) = Gini(D) - Gini\ index(D,A) \tag{29}$$

- 樹的建構

  從根節點開始選擇特徵，數據集會被分割成兩個或多個子集，這些子集構成當前節點的子節點。接著，透過遞迴的方式，對每個子節點進行特徵選擇和分割，直到滿足停止條件，例如：所有特徵的資訊增益都非常小、無法再進行有效分割，或是達到預定深度等。

- 樹的剪枝

  模型在學習的過程中，為了盡可能的正確的分類訓練樣本，不停的對結點進行劃分，這可能會導致分支過多，也就是模型過適。為了避免此狀況，可以採用剪枝技術來簡化回歸樹。剪枝技術有兩種主要形式：預剪枝

（停止樹的進一步生長）和後剪枝（移除樹中的部分節點）。

- 預剪枝（Pre-pruning）

  預剪枝是一種在回歸樹的建構過程中提前介入的方法。在每個節點進行分割前，先對其進行評估：如果進一步分割該節點不能顯著提升回歸樹的泛化能力，則停止分割並將該節點標記為葉節點。我們可以設置一些條件來控制，例如回歸樹的最大深度、節點內的最小樣本數，或是分割後的最小信息增益。

- 後剪枝（Post-pruning）

  後剪枝則在回歸樹完全建構完成後執行，從樹底向上對子節點進行評估。如果將某個子節點替換為葉節點能夠提升模型的泛化能力，則將該子樹替換為葉節點。

第四章　實驗結果

第一節　衡量指標

在本次研究中，為了評估及比較模型的預測性能與準確性，我們選用了平均絕對誤差（Mean Absolute Error, MAE）、均方根誤差（Root Mean Squared Error, RMSE）與平均絕對百分比誤差 (Mean Absolute Percentage Error, MAPE)這三項誤差值作為主要的衡量指標。這三項指標在時間序列預測和迴歸問題中常被廣泛應用，被普遍認為能夠提供對於模型準確性及穩定性的全面評價以便使用者觀察及運用。

一、 平均絕對誤差 (Mean Absolute Error, MAE)

MAE 計算了預測值與實際觀測值之間的平均偏差，公式如下：

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}| \tag{30}$$

其中，y_i為實際觀測值，$\hat{y_i}$為模型的預測值，n 為樣本總數。MAE 的單位與目標變數一致，因此能直觀呈現預測誤差的平均程度。此指標的優勢在於其易於解釋，能顯示模型整體預測的精準度，反映模型在常見情況下的穩定性。

二、 均方根誤差 (Root Mean Squard Error, RMSE)

RMSE 提供了預測值與實際值偏差的平方平均後的平方根，計算公式如下：

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2} \tag{31}$$

與 MAE 相比，RMSE 更加敏感於大幅偏離的數據，因為誤差在平方過程中被放大。這使 RMSE 能夠提供額外的信息，揭示模型在處理異常數據或急劇變化時的能力與不足，尤其是預測中出現極值偏差時的影響。

三、 平均絕對百分比誤差 (Mean Absolute Percentage Error, MAPE)

MAPE 適用於評估模型預測準確性的指標，特別常用於回歸問題中。公式如下：

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y_i}}{y_i}\right| \times 100\% \quad (32)$$

與前兩者相比，因其對相對誤差的解釋非常直觀且適用於不同單位的數據集比較的原因，MAPE 通常用於經濟、財務、能源需求等時間序列預測問題中；但相對的，MAPE 對於零值十分敏感且其不對稱性，需要使用其他誤差指標（如 MAE 或 RMSE）輔助評估。

四、 綜合評價與應用

MAE 與 RMSE 這兩項指標的對比分析可用來判斷模型的誤差分布特性。若 RMSE 顯著高於 MAE，則表明模型在某些預測點上的誤差較大，存在少量明顯偏離的情況。反之，若兩者數值相近，則代表模型誤差穩定。在兩者比較外，我們也加入 MAPE 使我們的模型可以脫離資料集與其他模型進行比較。透過這樣的分析，我們能對模型在 $PM_{2.5}$ 濃度預測中的表現有更全面的了解，從而選擇最佳的模型設置，以便更精確地進行空氣污染預測和決策支持。

第二節　實驗結果

本研究使用北京 $PM_{2.5}$ 作為訓練資料，旨在分析其變化趨勢與可能的影響因素。我們選取了近年來的歷史資料，結合多種機器學習模型進行訓練，探討風向、氣溫、氣壓等變量與 $PM_{2.5}$ 濃度之間的關聯性。

本次訓練資料使用 2010/01/01 - 2014/12/31 期間內的細懸浮顆粒資料。以去掉最後七天的資料作為訓練資料集，最後七天作為測試資料集。每筆訓練資料包含標準化後的各項天氣細項指標以及對應的細懸浮顆粒指數，並依照日期、時間序列依序編制為訓練資料。

為了驗證本次提出的模型架構是否能充分說明資料，除了深度學習的神經網路模型，例：CNN、RNN、LSTM、GRU、Transformer，我們也以此為基礎做出 Bidirectional、Stacked、混用等變化。而後將整體的訓練資料依照日期、時間序列輸入個別模型後，計算各模型的 MAE、RMSE、MAPE，驗證此模型架構與其他基礎模型相比較下是否有更好的效果，實驗結果如下表。

表 11 模型變化前後比較

模型架構	MAE	RMSE	MAPE
CNN 1D	1.344	1.856	8.23%
CNN - RNN	0.855	1.136	2.59%
CNN - LSTM	**0.744**	**0.925**	**2.75%**
CNN - GRU	2.842	3.663	10.88%
S - CNN	2.861	3.666	10.37%
RNN	**13.020**	**26.634**	**29.85%**
B - RNN	14.620	24.972	24.31%
S - RNN	14.268	24.482	25.07%
SB - RNN	17.061	28.774	28.48%
LSTM	14.481	24.219	24.82%
B - LSTM	14.646	24.752	24.91%
S - LSTM	**14.426**	**24.531**	**24.07%**
SB - LSTM	14.614	24.770	24.30%
GRU	**13.896**	**23.832**	**23.20%**
B - GRU	14.072	24.501	24.24%
S - GRU	14.068	24.809	25.03%
SB - GRU	14.277	24.409	23.62%
Transformer	20.406	33.442	29.77%
S - Transformer	**19.936**	**35.586**	**22.39%**

註：B - ：Bidirectional、S - ：Stacked、SB - ：Stacked Bidirectional

表 12 依照表 11 中各模型誤差最小之通用設定

模型	Dense（隱藏層）神經元個數	Dense（輸出層）神經元個數	batch_size	epochs
CNN - LSTM(2)	(64)	1	32	30
RNN(64)	X		8	50
S – LSTM(64, 64)	(128)		64	30
GRU(256)	(64, 2)		64	30
S – Transformer(64, 64)	(128)		2	1

表 13 依照表 11 中各模型誤差最小之個別設定

模型	設定
CNN - LSTM(2)	filters: 8

GRU(256)	QuantileTranformer(normal)
S – Transformer(64, 64)	dropout_rate: 0.01, key_dim: 64, ff_dim: 8, num_heads: 2, optimizer: adam

表 14 過往實驗記錄

模型架構	MAE	RMSE
Prophet	58.74	79.73
Autoregressive Integrated Moving Average(ARIMA)	69.42	94.21
Support Vector Regression(SVR)	57.04	88.62
Decision Tree Regressor	46.27	26.05

從上表中可以得知 CNN - LSTM 的表現相對於其他基礎模型比較之下有較好的效果，顯示出以 CNN 作為預測本數據集會得到最小誤差的預測，而當撇除掉數據集的限制下，CNN - RNN 的程式設計與貼合度則是最佳的。

第三節　細懸浮顆粒預測

為了進一步觀察模型預測細懸浮顆粒的效果，我們將北京細懸浮顆粒的測試訓練集輸入至模型中進行 $PM_{2.5}$ 預測，並計算每次 MAE 後以 MAE 作為評判標準找出我們制定的範圍內的最佳參數，MAE 最小的前三個模型分別是 CNN - RNN、CNN - LSTM、CNN1D，以此可知，在使用這些模型做為未來預測中可能取得與現實將相符的結果，整體模型測量之 MAE 如下圖所示。

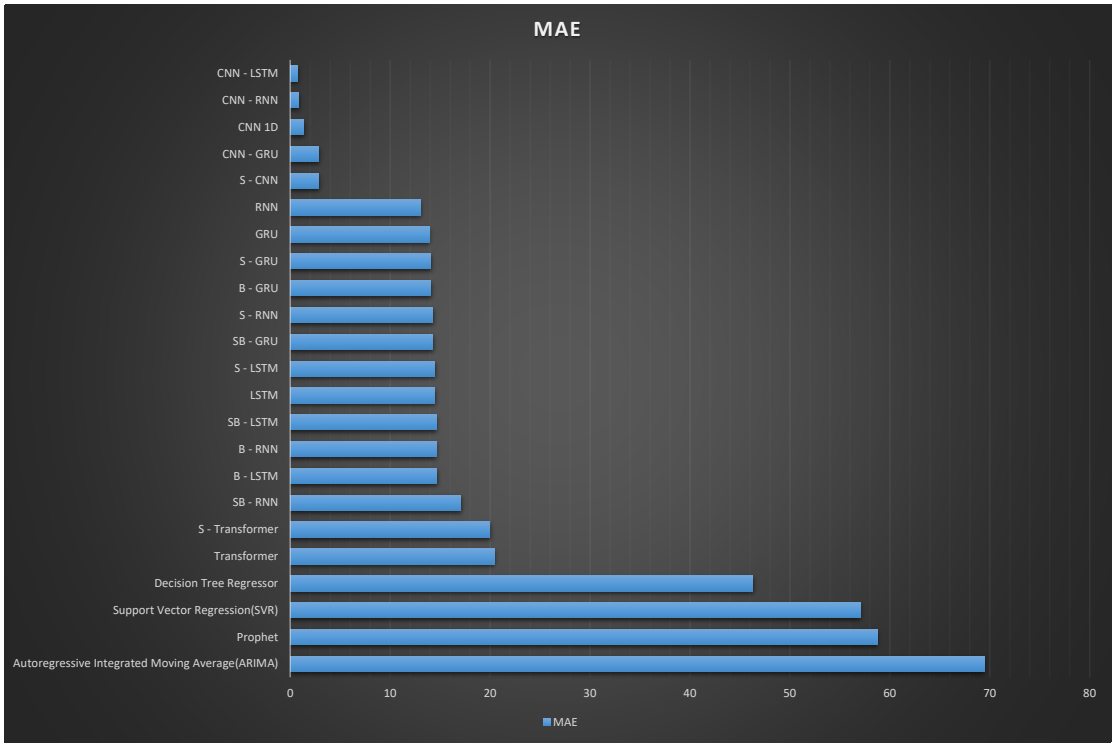

圖 11 本次實驗中所有模型測出之 MAE

第四節　研究限制與未來方向

本研究基於 CNN、RNN、LSTM、GRU 與 Transformer 等深度學習模型對北京 $PM_{2.5}$ 濃度進行預測，然而仍存在若干限制與改進空間。首先，數據來源主要集中於北京地區，可能無法完全反映其他地區的污染情況，限制了模型的泛化能力。此外，數據中可能存在缺失值與異常值，對模型的準確性造成影響，特別是在長期預測中，誤差可能進一步累積。模型超參數調整的範圍受計算資源限制，未能窮舉所有可能的組合，這可能影響模型性能的最優化。同時，特徵選擇僅限於可獲取的氣象與污染數據，未能考慮交通流量、工業活動或政策變化等潛在因素。此外，部分模型（如 Transformer）對計算資源需求較高，導致其性能可能未完全發揮。本研究主要聚焦於短期預測，對長期趨勢的捕捉能力相對有限。

針對上述限制，未來研究可在以下幾方面展開：首先，引入多地區、多樣化的數據，結合數據增強技術（如合成數據生成），提升模型的適用性與泛化能力。其次，探索模型融合策略（如 GRU - LSTM 或 Transformer - LSTM 混合架構）與自適應超參數調整技術（如貝葉斯優化或遺傳算法），進一步優化模型性能。為提升模型的解釋性，可結合 SHAP 或 LIME 等方法分析模型對不同特徵的依賴

性，盡可能降低 MAPE 以提高模型解釋度。此外，未來可融入社會經濟指標、交通流量與政策變化等非傳統數據，增強模型對多維度因素的綜合預測能力。在資源高效化方面，可設計輕量級模型，實現模型在邊緣設備上的實時部署，滿足即時預測需求。針對長期預測，可設計多步預測框架，並引入循環更新策略或情境假設，以改善長期趨勢的預測性能。最後，應將模型應用於實際空氣質量管理場景中，如輔助政府制定減排政策，或作為即時警報系統的一部分，以提升研究對實際問題的影響力。

第五章　結論

本研究基於多種深度學習模型對北京 $PM_{2.5}$ 濃度進行了詳細的預測分析，結果顯示以 CNN 為基礎的混合模型（如 CNN - LSTM、CNN - RNN）在精度與穩定性上表現優異，特別是在 MAE、RMSE 與 MAPE 等評估指標中均顯示出相對較低的誤差，證明其在細懸浮顆粒濃度預測中的潛力。相比之下，傳統模型（如 Prophet、ARIMA）在處理複雜時間序列與高非線性數據時表現不如深度學習模型。

儘管如此，本研究亦暴露出部分限制，例如數據來源的局限性、模型泛化能力的不足及計算資源限制對模型優化的影響。然而，這些限制為未來研究提供了方向，包括引入更多元的數據來源、融合多種模型架構、優化超參數調整策略以及提升模型的解釋性與資源高效性。

綜上所述，本研究不僅驗證了深度學習模型在 $PM_{2.5}$ 預測中的應用價值，亦為未來改進相關模型與方法提供了參考。隨著研究的深入，這些模型有望在空氣質量預測與管理中發揮更大的作用，為環境保護與公共健康政策提供有力支持。

參考文獻

[1] 行政院（2024）。推動氣候變遷調適行動方案進展報告。行政院全球資訊網。https://www.ey.gov.tw/Page/9277F759E41CCD91/65caab52-28ae-4c9c-b0fc-4e0bc9ae5642。

[2] 法國國際廣播電台（2024）。中國的霧霾死灰復燃給我們敲響了警鐘。法國報紙摘要。https://www.rfi.fr/tw/%E5%B0%88%E6%AC%84%E6%AA%A2%E7%B4%A2/%E6%B3%95%E5%9C%8B%E5%A0%B1%E7%B4%99%E6%91%98%E8%A6%81/20240511-%E5%9C%8B%E9%9A%9B%E9%80%9A%E8%A8%8A-%E4%B8%AD%E5%9C%8B%E7%9A%84%E9%9C%A7%E9%9C%BE%E6%AD%BB%E7%81%B0%E5%BE%A9%E7%87%83%E7%B5%A6%E6%88%91%E5%80%91%E6%95%B2%E9%9F%BF%E4%BA%86-%E8%AD%A6%E9%90%98。

[3] 謝岳書（2013）。大氣細懸浮微粒金屬成份特徵及來源貢獻分析研究。〔碩士論文。朝陽科技大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/k8audk

[4] 王鵬 (Peng Wang), 陳武義 (Wu-Xi Chen), 林招膨 (Jao-Perng Li), & 賴律翰 (Lu-Han Lai). (2016). 北京地區霧霾之微懸浮粒子 $PM_{2.5}$ 汙染與肺癌患病率關係分析. 台灣應用輻射與同位素雜誌, 12(4), 1405-1410.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel (1989). Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation, 1(4), 541-551

[6] Y. LeCun, Y. Bengio (1995), Convolutional Networks for Images, Speech, and Time-Series. The Handbook of Brain Theory and Neural Networks, 3361(10), 1995

[7] 黃綱正（2019）。結合 CNN 與 LSTM 模型建構分類惡意程式方法。〔碩士論文。國立臺北科技大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/s4j874

[8] 陳雨玟（2024）。結合 CNN 圖像識別與時間序列分析股票重點指標。〔碩士論文。中原大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/8q75cw

[9] Al-Ashwal, N. H., Al Soufy, Al-Ashwal, N.H., Al Soufy, K.A.M., Hamza, M.E.E., Swillam, M.A (2023). Deep Learning for Optical Sensor Applications: A Review. Sensors, 23, 6486. https://doi.org/10.3390/s23146486

[10] Balázs Hidasi, Alexandros Karatzoglou, L. Baltrunas, D. Tikk (2013). Recurrent neural networks. https://doi.org/10.4249/scholarpedia.1888

[11] Shuai Li, W. Li, Chris Cook, Ce Zhu, Yanbo Gao (2018). Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN. IEEE/CVF Conference on Computer Vision and Pattern Recognition

[12] 張曉妮（2024）。應用 CNN-LSTM 深度學習模型於預測台灣二氧化碳排放量。〔碩士論文。元智大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/b4rjpf

[13] 葉國毅（2024）。使用多頭注意力機制的多重輸入 LSTM 股價預測模型。〔碩士論文。國立政治大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/nd5rqm

[14] A. Graves, A. Mohamed, and G. E. Hinton, “Speech recognition with deep recurrent neural networks,” *CORR*, 2013

[15] 李思賢（2024）。利用 LSTM、RNN 與 GRU 模型預測比特幣價格：一個深度學習方法的效能分析。〔碩士論文。國立高雄科技大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/5542gh

[16] 游淵喻（2024）。邁向淨零碳排放利用深度學習 LSTM 與 GRU 預測台灣電力消耗與再生能源未來用量之研究。〔碩士論文。國立中興大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/8n9nsd

[17] TengYuan Chang（2019）。比較長短期記憶模型 (LSTM) 與改良後的遞歸神經網路模型：GRU。Medium。https://reurl.cc/WNjKx5

[18] 周家興（2024）。可微分查找矩陣乘法用於壓縮 Transformer 網路。〔碩士論文。國立清華大學〕國立清華大學博碩士全文系統。https://etd.lib.nycu.edu.tw/cgi-bin/gs32/hugsweb.cgi?o=dnthucdr&s=id=%22G021100626470%22.&searchmode=basic

[19] 沈家丞（2024）。適用於人類誘導多能幹細胞之微模式明場顯微影像的 Transformer 細胞分割深度學習技術。〔碩士論文。中原大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/gf6385

[20] 劉晴, 董平軍 (2020)。時間序列長度對基於 Prophet 的自動販賣機銷售預測影響研究。管理科學與工程, 9(4), 266 – 276。https://doi.org/10.12677/MSE.2020.94035

[21] 楊昇翰（2023）。台灣股市股票價格預測模型之比較：以 LSTM GRU Prophet 模型為例。〔碩士論文。淡江大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/kv7uyn。

[22] 羅艾帛（2023）。應用深度學習和時間序列分析技術於餐飲外送平台需求預測之比較研究。〔碩士論文。元智大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/xb83mq。

[23] 邱智清（2024）。時間序列套件及基於預處理的深度學習模型在加密貨幣價格預測之績效比較。〔碩士論文。國立屏東大學〕臺灣博碩士論文知識加值系統。 https://hdl.handle.net/11296/p56uhc。

[24] 李培煜（2021）。時間序列 ARIMA 與深度學習 LSTM 預測模型之比較：以台灣股票市場為例。〔碩士論文。東吳大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/cb4j5s。

[25] 余光倫（2022）。應用 ARIMA 模型對影響台灣工業生產指數之產業預測與分析。〔碩士論文。逢甲大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/jc7tvy。

[26] 余奇潔（2013）。美元兌新台幣匯率預測 -ARIMA 與 SVM 之應用-。〔碩士論文。國立高雄應用科技大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/wzmz43。

[27] 王百祿(2007)。ARIMA 與適應性 SVM 之混合模型於股市指數預測之研究。〔碩士論文。國立成功大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/x3yd3b。

[28] 陳俊宏（2015）。應用與比較多種預測技術於光學膜產業銷售預測。〔碩士論文。健行科技大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/ywph2f。

[29] 林芳毅（2015）。Hadoop 結合 R 之資料探勘 – 以 TH 公司產品銷售預測為例。〔碩士論文。國立高雄應用科技大學〕臺灣博碩士論文知識加值系統。https://hdl.handle.net/11296/234g3a。

[30] Loureiro, A. L. D., Miguéis, V. L., & da Silva, L. F. M. (2018). Exploring the use of deep neural networks for sales forecasting in fashion retail. Decision Support

Systems, 114, 81–93. https://doi.org/10.1016/j.dss.2018.08.010

[31] Schmidt, A., Kabir, M. W. U., & Hoque, M. T. (2022). Machine learning based restaurant sales forecasting. Machine Learning and Knowledge Extraction, 4(1), 105–130. https://doi.org/10.3390/make4010006

[32] 萬絢、陳婷瑜、謝雯如、陳熹蓉、黃舒雅（2019）。線性判別分析和決策樹台灣半導體股票投資組合報酬率之實證。嶺東學報，(44)，213-229。https://www.airitilibrary.com/Article/Detail?DocID=18111912-201906-201907020004-201907020004-213-229

[33] 蘇甲辛（2019）。應用決策樹預測傳統通路銷售潛力之研究-以雲嘉地區傳統菸酒零售通路為例。〔碩士論文。國立高雄科技大學〕臺灣博碩士論文知識加值系統。 https://hdl.handle.net/11296/ae9a38。

[34] Kumar, S., & Patel, N. (2010). Decision tree based demand forecasts for improving inventory performance. In 2010 IEEE International Conference on Industrial Engineering and Engineering Management (pp. 1355–1359). IEEE. https://doi.org/10.1109/IEEM.2010.5674628

[35] Johannes, R., & Alamsyah, A. (2021). Sales prediction model using classification decision tree approach for small medium enterprise based on Indonesian e-commerce data. arXiv preprint arXiv:2103.03117. https://arxiv.org/abs/2103.03117

附錄（一個模型以一個為例）

CNN 1D :

```
# pm2.5 預測，使用 CNN
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot as plt
from pandas import read_csv, DataFrame
import numpy as np
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
Dropout, BatchNormalization
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error
import datetime
from sklearn.model_selection import train_test_split

# 紀錄開始時間
starttime = datetime.datetime.now()
print('開始時間: ', starttime)

filename = 'PRSA_final.csv'
filenameOut = 'prediction/pollution_CNN.csv'
filenameModel = 'model/pollution_CNN.h5'
df = read_csv(filename)

# 選取適合的特徵
features = ['pm2.5', 'dew', 'temp', 'press', 'wnd_dir', 'wnd_spd', 'snow', 'rain']
df_X = df[features]
df_y = df["pm2.5"]

# 標準化特徵
scaler = StandardScaler()
X = scaler.fit_transform(df_X.values)
y = scaler.fit_transform(df_y.values.reshape(-1, 1))
```

```
#y = df_y.values

# 調整形狀為 CNN 所需格式（使用 Conv1D）
X = X.reshape(-1, X.shape[1], 1)

# 分割數據集
#train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.3,
random_state=0)
#train_size = int(len(X) * 0.7)
test_size = 7 * 24
train_size = len(X) - test_size
train_X, test_X = X[:train_size], X[train_size:]
train_y, test_y = y[:train_size], y[train_size:]

# 超參數設定
n_filters = 64
n_kernel_size = 3
n_batch_size = 32
n_epochs = 30

# CNN 模型（使用 Conv1D）
def create_model(activation='relu', optimizer='adam', filters=n_filters,
kernel_size=n_kernel_size):
    model = Sequential()

    model.add(Conv1D(filters, kernel_size, activation=activation,
input_shape=(X.shape[1], 1)))
    model.add(MaxPooling1D(pool_size=2, padding="same"))
    '''
    for i in range(2):
        model.add(Conv1D(filters, kernel_size, activation=activation,
padding="same", return_sequences=True))
        model.add(MaxPooling1D(pool_size=2, padding="same"))
        #model.add(Dropout(0.2))
    '''
    # 展平並輸出
    model.add(Flatten())
```

```
    #model.add(Dense(512, activation='relu'))
    #model.add(Dense(128, activation='relu'))
    model.add(Dense(1))

    model.compile(optimizer=optimizer, loss='mean_squared_error',
metrics=['mae', 'accuracy'])
    return model

# 保存每次 mae, rmse, mape
train_mae_list = []
train_rmse_list = []
train_mape_list = []
test_mae_list = []
test_rmse_list = []
test_mape_list = []

for i in range(10):
    print(f"第{i+1}次訓練開始")
    # 建立與訓練模型
    model = create_model()
    history = model.fit(
        train_X, train_y,
        epochs=n_epochs, # [1, 5, 10, 15, 20, 30, 40, 50]
        batch_size=n_batch_size, # [2, 4, 8, 16, 32, 64, 128, 256]
        validation_data=(test_X, test_y),
        verbose=1,
        shuffle=False
    )
    '''
    #print(history.history.keys())
    # 顯示模型訓練結果 loss
    from matplotlib import pyplot
    pyplot.plot(history.history['loss'], label='train')
    pyplot.plot(history.history['val_loss'], label='test')
    pyplot.legend()
    pyplot.show()
```

```
    from matplotlib import pyplot
    pyplot.plot(history.history['accuracy'], label='train')
    pyplot.plot(history.history['val_accuracy'], label='test')
    pyplot.legend()
    pyplot.show()
    '''
    # 訓練集
    # 預測與反標準化
    yhat_train = model.predict(train_X)
    #inv_yhat_train = scaler.inverse_transform(concatenate((yhat_train, train_X[:,
1:, 0]), axis=1))[:, 0]
    #inv_y_train = scaler.inverse_transform(concatenate((train_y.reshape(-1, 1),
train_X[:, 1:, 0]), axis=1))[:, 0]
    inv_yhat_train = scaler.inverse_transform(yhat_train)
    inv_y_train = scaler.inverse_transform(train_y.reshape(-1, 1))
    # 訓練集 MAE 和 RMSE
    train_mae = mean_absolute_error(inv_y_train, inv_yhat_train)
    train_rmse = sqrt(mean_squared_error(inv_y_train, inv_yhat_train))
    train_mape = mean_absolute_percentage_error(inv_y_train, inv_yhat_train) *
100
    # 儲存
    train_mae_list.append(train_mae)
    train_rmse_list.append(train_rmse)
    train_mape_list.append(train_mape)

    # 測試集
    # 預測與反標準化
    yhat = model.predict(test_X)
    #inv_yhat = scaler.inverse_transform(concatenate((yhat, test_X[:, 1:, 0]),
axis=1))[:, 0]
    #inv_y = scaler.inverse_transform(concatenate((test_y.reshape(-1, 1), test_X[:,
1:, 0]), axis=1))[:, 0]
    inv_yhat = scaler.inverse_transform(yhat)
    inv_y = scaler.inverse_transform(test_y.reshape(-1, 1))
    '''
    test_X_original = test_X.reshape((test_X.shape[0], -1))  # 還原形狀為 2D
    # 拼接預測值和原始特徵
```

```
    n_features = test_X_original.shape[1]  # 獲取特徵數
    n_target = 1  # 目標變量數目 (例如 pm2.5 是單一變量)
    inv_yhat = concatenate((yhat, test_X_original[:, -(n_features - n_target):]),
axis=1)
    # 對拼接後的數據進行反標準化
    inv_yhat = scaler.inverse_transform(inv_yhat)
    # 提取反標準化後的目標變量（pm2.5）
    inv_yhat = inv_yhat[:, 0]
    # 對真實值也進行同樣的反標準化
    test_y = test_y.reshape((len(test_y), 1))  # 確保是 2D
    inv_y = concatenate((test_y, test_X_original[:, -(n_features - n_target):]),
axis=1)
    inv_y = scaler.inverse_transform(inv_y)
    inv_y = inv_y[:, 0]
    '''
    # 計算 MAE 和 RMSE
    mae = mean_absolute_error(inv_y, inv_yhat)
    rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
    mape = mean_absolute_percentage_error(inv_y, inv_yhat) * 100
    # 儲存
    test_mae_list.append(mae)
    test_rmse_list.append(rmse)
    test_mape_list.append(mape)

    print(f"第 {i+1} 次訓練結束 - 測試集 MAE: {mae}, RMSE: {rmse},
MAPE: %.2f%%" % mape)

# 計算平均值
avg_train_mae = np.mean(train_mae_list)
avg_train_rmse = np.mean(train_rmse_list)
avg_train_mape = np.mean(train_mape_list)
avg_test_mae = np.mean(test_mae_list)
avg_test_rmse = np.mean(test_rmse_list)
avg_test_mape = np.mean(test_mape_list)

# 顯示訓練結果
print("訓練集")
```

```
#print("MAE:", train_mae, "RMSE:", train_rmse)
print(f"MAE: {avg_train_mae}, RMSE: {avg_train_rmse}, MAPE: %.2f%%" %
avg_train_mape )
# 顯示測試結果
print("測試集")
#print("MAE:", mae, "RMSE:", rmse)
print(f"MAE: {avg_test_mae}, RMSE: {avg_test_rmse}, MAPE: %.2f%%" %
avg_test_mape)

endtime = datetime.datetime.now()
print('開始時間: ', starttime)
print('結束時間: ', endtime)
print('花費時間: ', endtime - starttime)

# 預測 vs 實際
plt.plot(inv_yhat, 'b:', label='Predict')
plt.plot(inv_y, 'r-', label='Real')
plt.legend(['Predict', 'Real'])
plt.xlabel("Time Index")
plt.ylabel("PM2.5")
plt.title("PM2.5 Prediction with 1DCNN")
plt.show()
'''
# 只顯示最後 7 天的實際值和預測值
last_7_days_real = inv_y[-(7*24):]
last_7_days_predict = inv_yhat[-(7*24):]
# 繪圖
plt.plot(last_7_days_predict, 'b:', label='Predict (Last 7 days)')
plt.plot(last_7_days_real, 'r-', label='Real (Last 7 days)')
plt.legend(['Predict', 'Real'])
plt.xlabel("Day Index")
plt.ylabel("PM2.5")
plt.title("PM2.5 Prediction for Last 7 Days")
plt.show()
'''
'''
# 儲存模型及預測結果
```

```
#model.save(filenameModel)
#print("Saved Model:", filenameModel)

df_PredResult = DataFrame({"Real": inv_y, "Pred": inv_yhat})
print('Displaying first 5 rows of prediction results')
print(df_PredResult.head())
df_PredResult.to_csv(filenameOut)
print("Saved predictions to:", filenameOut)
'''
```

RNN :

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  3 14:15:56 2024

@author: ASUS
"""

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot as plt
from pandas import read_csv, DataFrame, concat
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler,
RobustScaler, MaxAbsScaler, QuantileTransformer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU, Bidirectional,
SimpleRNN
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error
from tensorflow.keras.optimizers import Adam
import datetime

import warnings
warnings.filterwarnings("ignore")

#紀錄開始時間
```

```
starttime = datetime.datetime.now()
print('開始時間: ', starttime)

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = [], []
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    agg = concat(cols, axis=1)
    agg.columns = names
    if dropnan:
        agg.dropna(inplace=True)
    return agg

# 儲存 10 次的 MAE 和 RMSE 結果
mae_list = []
rmse_list = []
mape_list = []

# 使用最佳參數進行模型訓練和預測 10 次
for i in range(10):
    # Load and preprocess dataset
    filename = 'pollution_cleaned.csv'
    filenameOut = 'pollution_RNN.csv'
    filenameModel = 'pollution_RNN.h5'

    dataset = read_csv(filename, header=0, index_col=0)
    values = dataset.values
```

```
    encoder = LabelEncoder()
    values[:, 4] = encoder.fit_transform(values[:, 4])
    values = values.astype('float32')
    #normalize features
    #scaler = MinMaxScaler(feature_range=(0,1)) #26.03
    scaler = StandardScaler() #24.26
    #scaler = RobustScaler() #24.81
    #scaler = MaxAbsScaler() #27.27
    #scaler = QuantileTransformer(output_distribution='normal')   #24.55
    #scaler = QuantileTransformer(output_distribution='uniform')   #24.81
    scaled=scaler.fit_transform(values)

    # Specify lag hours and features
    n_hours = 3
    n_features = 8
    reframed = series_to_supervised(scaled, n_hours, 1)

    # Split data into train and test sets
    values = reframed.values
    n_train_hours=len(values) - 7*24   #365*24 #7 天
    train = values[:n_train_hours, :]
    test = values[n_train_hours:, :]

    n_obs = n_hours * n_features
    train_X, train_y = train[:, :n_obs], train[:, -n_features]
    test_X, test_y = test[:, :n_obs], test[:, -n_features]

    train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
    test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))

    # Function to create model for GridSearchCV
    def create_model():
        model = Sequential()
        model.add(SimpleRNN(64, input_shape=(train_X.shape[1],
train_X.shape[2])))
        model.add(Dense(512, activation='relu'))
        model.add(Dense(128, activation='relu'))
```

```
        model.add(Dense(1))
        model.compile(loss='mae', optimizer='adam')
        return model

    # Define hyperparameters for tuning
    param_grid = {
        'batch_size': [2, 4, 8, 16, 32, 64],
        'epochs': [1, 5, 10, 15, 20, 30]
    }

    # Manual Grid Search (without units and learning_rate)
    best_score = float("inf")
    best_params = {}
    Algorithmtime=datetime.datetime.now()
    print("演算時間: ",Algorithmtime)

    for batch_size in param_grid['batch_size']:
        for epochs in param_grid['epochs']:
            model = create_model()
            history = model.fit(
                train_X, train_y,
                epochs=epochs,
                batch_size=batch_size,
                validation_data=(test_X, test_y),
                verbose=0,
                shuffle=False
            )

            # Evaluate the model
            yhat = model.predict(test_X)
            test_X_reshaped = test_X.reshape((test_X.shape[0], n_hours *
n_features))
            inv_yhat = concatenate((yhat, test_X_reshaped[:, -(n_features - 1):]),
axis=1)
            inv_yhat = scaler.inverse_transform(inv_yhat)
            inv_yhat = inv_yhat[:, 0]
```

```
            # Invert scaling for actual test values
            test_y_reshaped = test_y.reshape((len(test_y), 1))
            inv_y = concatenate((test_y_reshaped, test_X_reshaped[:, -(n_features -
1):]), axis=1)
            inv_y = scaler.inverse_transform(inv_y)
            inv_y = inv_y[:, 0]

            # Calculate MAE and RMSE
            mae = mean_absolute_error(inv_y, inv_yhat)
            rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
            mape = mean_absolute_percentage_error(inv_y, inv_yhat) * 100

            if mae < best_score:
                best_score = mae
                best_params = {
                    'batch_size': batch_size,
                    'epochs': epochs
                }
                best_model = model
                best_rmse = rmse

    '''
    print("Best Parameters:", best_params)
    print("Best MAE:", best_score)
    print("Best RMSE:", best_rmse)
    '''

    endtime=datetime.datetime.now()

    '''
    print('開始時間: ',starttime)
    print('演算時間: ',Algorithmtime)
    print('結束時間: ', endtime)
    print('花費時間: ',endtime-starttime)
    '''

    # Plot predictions vs actual values
```

```
yhat = best_model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_hours * n_features))
inv_yhat = concatenate((yhat, test_X[:, -(n_features - 1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]

test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, -(n_features - 1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]

plt.plot(inv_yhat, 'b:', label='Predict')
plt.plot(inv_y, 'r-', label='Real')
plt.legend(['Predict', 'Real'])
plt.xlabel("Time Index")
plt.ylabel("PM2.5")
plt.title("PM2.5 Prediction with RNN")
plt.show()

'''
# Save best model and predictions
best_model.save(filenameModel)
print("Saved Model:", filenameModel)

df_PredResult = DataFrame({"Real": inv_y, "Pred": inv_yhat})
print('Displaying first 5 rows of prediction results')
print(df_PredResult.head())
df_PredResult.to_csv(filenameOut)
print("Saved predictions to:", filenameOut)
'''

mae_list.append(mae)
rmse_list.append(rmse)
mape_list.append(mape)

# Calculate MAE & RMSE
print('第',i+1,'次訓練')
```

```
        print("Best Parameters:", best_params)
        print('Test MAE: %.3f' % mae)
        print('Test RMSE: %.3f' % rmse)
        print('Test MAPE: %.2f%%' % mape)
        print('開始時間: ',starttime)
        print('演算時間: ',Algorithmtime)
        print('結束時間: ', endtime)
        print('花費時間: ',endtime-starttime)

    # 計算 MAE 和 RMSE 的平均值
    mean_mae = sum(mae_list) / len(mae_list)
    mean_rmse = sum(rmse_list) / len(rmse_list)
    mean_mape = sum(mape_list) / len(mape_list)

    print(f'10 次訓練的平均 MAE: {mean_mae:.3f}')
    print(f'10 次訓練的平均 RMSE: {mean_rmse:.3f}')
    print(f'10 次訓練的平均 MAPE: {mean_rmse:.3f}')
```

LSTM :

```
    # -*- coding: utf-8 -*-
    """
    Created on Sun Nov  3 14:15:56 2024

    @author: ASUS
    """

    from math import sqrt
    from numpy import concatenate
    from matplotlib import pyplot
    from pandas import read_csv, DataFrame, concat
    from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler,
    RobustScaler, MaxAbsScaler, QuantileTransformer
    from keras.models import Sequential
    from keras.layers import Dense, SimpleRNN, LSTM
    from sklearn.metrics import mean_absolute_error, mean_squared_error,
    mean_absolute_percentage_error
```

```
import datetime

import warnings
warnings.filterwarnings("ignore")

starttime = datetime.datetime.now()
print('開始時間: ', starttime)

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df=DataFrame(data)
    cols, names = list(), list()
    #input sequense(t-n, ...t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    #forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i ==0:
            names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names+=[('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    #put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    if dropnan:
        agg.dropna(inplace=True)
    return agg
'''
# 儲存 10 次的 MAE 和 RMSE 結果
mae_list = []
rmse_list = []
'''

Algorithmtime=datetime.datetime.now()
```

```
print("演算時間: ",Algorithmtime)

# 使用最佳參數進行模型訓練和預測 10 次
#for i in range(10):
#設定檔名(資料集、預測值結果、模型輸出)
filename='pollution_cleaned.csv'
filenameOut='pollution_LSTM.csv'
filenameModel='pollution_LSTM.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values

#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float

#資料型態轉換
values=values.astype('float32')
#normalize features
#scaler = MinMaxScaler(feature_range=(0,1)) #26.03
scaler = StandardScaler() #24.26
#scaler = RobustScaler() #24.81
#scaler = MaxAbsScaler() #27.27
#scaler = QuantileTransformer(output_distribution='normal')   #24.55
#scaler = QuantileTransformer(output_distribution='uniform')   #24.81
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_hours=3 #用前 n_hours 預測第 n_hours 的結果，即第 n_hours+1 的結果
n_features=8 #特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_hours, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
#print("reframed.shape:",reframed.shape)
```

```
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_hours=len(values) - 7*24   #365*24
train=values[:n_train_hours,:]
test=values[n_train_hours:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其預測輸入
(x1,x2...n_features)
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
#print("train_X.shape:", train_X.shape, len(train_X), train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
#print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
#print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(LSTM(64, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(512, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                    validation_data=(test_X, test_y), verbose=0, shuffle=False)
```

```
# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
#print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)
inv_yhat=inv_yhat[:,0]

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
#print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
#print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0]

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
#print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
```

```
#print('Test MAE: %.3f' % mae)
mape = mean_absolute_percentage_error(inv_y, inv_yhat) * 100
#print(f"MAPE: {mape:.2f}%")

test_y,predict_y=inv_y,inv_yhat

import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("pm2.5")
plt.title("Prediction(LSTM)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
#print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
#print('顯示前 5 筆資料')
#print(df_PredResult.head())
#print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
# Calculate the mean of the actual test values
mean_value = inv_y.mean()
'''
mae_list.append(mae)
rmse_list.append(rmse)
'''

endtime=datetime.datetime.now()
```

```
print('開始時間: ',starttime)
print('演算時間: ',Algorithmtime)
print('結束時間: ', endtime)
print('花費時間: ',endtime-starttime)
# Calculate MAE & RMSE
#print('第',i+1,'次訓練')
print('Test MAE: %.3f' % mae)
print('Test RMSE: %.3f' % rmse)
print('Test MAPE: %.2f%%' % mape)
'''
# 計算 MAE 和 RMSE 的平均值
mean_mae = sum(mae_list) / 10
mean_rmse = sum(rmse_list) / 10

print(f'10 次訓練的平均 MAE: {mean_mae:.3f}')
print(f'10 次訓練的平均 RMSE: {mean_rmse:.3f}')
'''
```

GRU :

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  3 14:15:56 2024

@author: ASUS
"""

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv, DataFrame, concat
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler,
RobustScaler, MaxAbsScaler, QuantileTransformer
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, LSTM, GRU
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error
```

```
import warnings
warnings.filterwarnings("ignore")

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df=DataFrame(data)
    cols, names = list(), list()
    #input sequense(t-n, ...t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    #forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i ==0:
            names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names+=[('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    #put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    if dropnan:
        agg.dropna(inplace=True)
    return agg
'''
# 儲存 10 次的 MAE 和 RMSE 結果
mae_list = []
rmse_list = []
'''
# 使用最佳參數進行模型訓練和預測 10 次
#for i in range(10):
#設定檔名(資料集、預測值結果、模型輸出)
filename='pollution_cleaned.csv'
filenameOut='pollution_GRU.csv'
filenameModel='pollution_GRU.h5'
```

```
#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values

#integer encode direction
encoder = LabelEncoder()
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float

#資料型態轉換
values=values.astype('float32')
#normalize features
#scaler = MinMaxScaler(feature_range=(0,1)) #26.03
scaler = StandardScaler() #24.26
#scaler = RobustScaler() #24.81
#scaler = MaxAbsScaler() #27.27
#scaler = QuantileTransformer(output_distribution='normal')   #24.55
#scaler = QuantileTransformer(output_distribution='uniform')   #24.81
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_hours=3 #用前 n_hours 預測第 n_hours 的結果，即第 n_hours+1 的結果
n_features=8 #特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_hours, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
#print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_hours=len(values) - 7*24   #365*24
```

```
train=values[:n_train_hours,:]
test=values[n_train_hours:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其預測輸入
(x1,x2...n_features)
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
#print("train_X.shape:", train_X.shape, len(train_X), train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
#print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
#print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)

# Design network
model = Sequential()
model.add(GRU(64, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(512, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                    validation_data=(test_X, test_y), verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
```

```
yhat = model.predict(test_X)

# Reshape from 3D to 2D
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))

# Invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
#print("inv_yhat.shpe=",inv_yhat.shape)
inv_yhat=scaler.inverse_transform(inv_yhat)
inv_yhat=inv_yhat[:,0]

#真實資料(test_y)還原成原始數值，先前用 minmaxscaler 將數值正規化(0~1)
#維度轉換:1D(220,)=>2D(220,1)
#print("test_y.shape=", test_y.shape,len(test_y))#1D(220,)
test_y=test_y.reshape((len(test_y),1)) #1D(220,)=>2D(220,1)
#print("test_y.shape=", test_y.shape,len(test_y))#2D(220,1)

#合併資料:inv_y=test_y+test_X:標籤欄(真實)+特徵欄
inv_y=concatenate((test_y, test_X[:,-(n_features-1):]),axis=1)
#真實資料(inv_y)還原成原始數值(test_y)
inv_y=scaler.inverse_transform(inv_y)
#只取第一欄的值，即標籤欄(真實)的值(value)
inv_y=inv_y[:,0]

#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
#print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
#print('Test MAE: %.3f' % mae)
mape = mean_absolute_percentage_error(inv_y, inv_yhat) * 100
#print(f"MAPE: {mape:.2f}%")

test_y,predict_y=inv_y,inv_yhat

import matplotlib.pyplot as plt
```

```
plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("pm2.5")
plt.title("Prediction(GRU)")
plt.show()

#from keras.model import load_model
model.save(filenameModel)
#print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
#print('顯示前 5 筆資料')
#print(df_PredResult.head())
#print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
# Calculate the mean of the actual test values
mean_value = inv_y.mean()
'''
mae_list.append(mae)
rmse_list.append(rmse)
'''
# Calculate MAE & RMSE
#print('第',i+1,'次訓練')
print('Test MAE: %.3f' % mae)
print('Test RMSE: %.3f' % rmse)
print('Test MAPE: %.2f%%' % mape)
'''
# 計算 MAE 和 RMSE 的平均值
mean_mae = sum(mae_list) / 10
mean_rmse = sum(rmse_list) / 10

print(f'10 次訓練的平均 MAE: {mean_mae:.3f}')
```

```
print(f'10 次訓練的平均 RMSE: {mean_rmse:.3f}')
'''
```

Transformer :

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  3 14:15:56 2024

@author: ASUS
"""

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv, DataFrame, concat
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler,
RobustScaler, MaxAbsScaler, QuantileTransformer
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, LSTM
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error
import datetime

import warnings
warnings.filterwarnings("ignore")

starttime = datetime.datetime.now()
print('開始時間: ', starttime)

# Convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df=DataFrame(data)
    cols, names = list(), list()
    #input sequense(t-n, ...t-1)
    for i in range(n_in, 0, -1):
```

```
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    #forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i ==0:
            names+=[('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names+=[('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    #put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    if dropnan:
        agg.dropna(inplace=True)
    return agg
'''
# 儲存 10 次的 MAE 和 RMSE 結果
mae_list = []
rmse_list = []
'''

Algorithmtime=datetime.datetime.now()
print("演算時間: ",Algorithmtime)

# 使用最佳參數進行模型訓練和預測 10 次
#for i in range(10):
#設定檔名(資料集、預測值結果、模型輸出)
filename='pollution_cleaned.csv'
filenameOut='pollution_LSTM.csv'
filenameModel='pollution_LSTM.h5'

#load dataset
dataset=read_csv(filename, header=0, index_col=0)
values=dataset.values

#integer encode direction
encoder = LabelEncoder()
```

```
values[:,4]=encoder.fit_transform(values[:,4])
#ensure all data is float

#資料型態轉換
values=values.astype('float32')
#normalize features
#scaler = MinMaxScaler(feature_range=(0,1)) #26.03
scaler = StandardScaler() #24.26
#scaler = RobustScaler() #24.81
#scaler = MaxAbsScaler() #27.27
#scaler = QuantileTransformer(output_distribution='normal')   #24.55
#scaler = QuantileTransformer(output_distribution='uniform')   #24.81
scaled=scaler.fit_transform(values)

#specify the number of lag hours
n_hours=3 #用前 n_hours 預測第 n_hours 的結果，即第 n_hours+1 的結果
n_features=8 #特徵數
#frame as supervised learing
reframed=series_to_supervised(scaled, n_hours, 1)
#(1220,20),用前 3 天預測第四天的結果，必須 4 天資料(4x5=20 特徵欄位)，合併成一列(row)
#print("reframed.shape:",reframed.shape)
#print(type(reframed))#DataFrame
#print(type(values))#numpy.ndarray

#split into train and test sets
values=reframed.values

##取前 200 周、每周 5 天的資料作為訓練資料(train)，剩餘資料做為測試資料
n_train_hours=len(values) - 7*24   #365*24
train=values[:n_train_hours,:]
test=values[n_train_hours:,:]

#切割欄位(x1,x2,x3...xn,y) 變成 timesteps 組成的 3D 數據(假設 timesteps=5)
# x1,x2,x3...xn 要變成 timesteps 組成的 3D 數據(x1,x2,x3)...y)
#n_ts = 3, 為每個 timesteps, 每個包含 5 個訓練例數據作為其預測輸入(x1,x2...n_features)
```

```
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
#print("train_X.shape:", train_X.shape, len(train_X), train_y.shape)

# Reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
#print("train_X.shape, train_y.shape:", train_X.shape, train_y.shape)
#print("test_X.shape, test_y.shape:", test_X.shape, test_y.shape)
'''
# Design network
model = Sequential()
model.add(LSTM(64, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(512, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                        validation_data=(test_X, test_y), verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
'''
from sklearn.model_selection import ParameterGrid
from keras.layers import Input, Dense, MultiHeadAttention, LayerNormalization,
Dropout
from keras.models import Model
from keras.callbacks import EarlyStopping

# 定義網格參數
param_grid = {
```

```
    "num_heads": [2, 4, 8, 16, 32],
    "key_dim": [64, 96, 128, 160, 256],
    "ff_dim": [8, 16, 32, 64, 128, 256, 512],
    "dropout_rate": [0.01, 0.05, 0.1, 0.2, 0.3],
    "optimizer": ["adam", "rmsprop"],
    "epochs": [1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    "batch_size": [2, 4, 8, 16, 32, 64, 128, 256, 512]
}

# 生成所有參數組合
grid = ParameterGrid(param_grid)

# 儲存結果
results = []

# 定義 Transformer Encoder Block
def transformer_encoder(inputs, num_heads, key_dim, ff_dim, dropout_rate):
    attn_output = MultiHeadAttention(num_heads=num_heads,
key_dim=key_dim)(inputs, inputs)
    attn_output = Dropout(dropout_rate)(attn_output)
    out1 = LayerNormalization(epsilon=1e-6)(inputs + attn_output)
    ffn_output = Dense(ff_dim, activation="relu")(out1)
    ffn_output = Dense(inputs.shape[-1])(ffn_output)
    ffn_output = Dropout(dropout_rate)(ffn_output)
    return LayerNormalization(epsilon=1e-6)(out1 + ffn_output)

# 遍歷參數組合
for params in grid:
    print(f"Testing with params: {params}")

    # 定義模型
    input_layer = Input(shape=(n_hours, n_features))
    x = transformer_encoder(input_layer,
                            num_heads=params["num_heads"],
                            key_dim=params["key_dim"],
                            ff_dim=params["ff_dim"],
                            dropout_rate=params["dropout_rate"])
```

```
    x = transformer_encoder(x,
                            num_heads=params["num_heads"],
                            key_dim=params["key_dim"],
                            ff_dim=params["ff_dim"],
                            dropout_rate=params["dropout_rate"])
    '''
    x = Dense(512, activation="relu")(x)
    x = Dense(128, activation="relu")(x)
    '''
    output_layer = Dense(1)(x)
    model = Model(inputs=input_layer, outputs=output_layer)

    # 編譯模型
    model.compile(loss='mae', optimizer=params["optimizer"])

    # 訓練模型
    history = model.fit(
        train_X, train_y,
        epochs=params["epochs"],
        batch_size=params["batch_size"],
        validation_data=(test_X, test_y),
        verbose=0,
        shuffle=False,
        callbacks=[EarlyStopping(monitor="val_loss", patience=5,
restore_best_weights=True)]
    )
    '''
    # 預測並評估
    yhat = model.predict(test_X)
    yhat = yhat[:, -1, :]
    test_y = test_y.reshape((len(test_y), 1))
    inv_yhat = concatenate((yhat, test_X_selected), axis=1) #inv_yhat =
concatenate((yhat, test_X[:, -(n_features-1):]), axis=1)
    inv_yhat = scaler.inverse_transform(inv_yhat)
    inv_yhat = inv_yhat[:, 0]
    inv_y = concatenate((test_y, test_X[:, -(n_features-1):]), axis=1)
    inv_y = scaler.inverse_transform(inv_y)
```

```
    inv_y = inv_y[:, 0]
    '''
    # Make a prediction
    yhat = model.predict(test_X)

    # 確保 yhat 為 2D (168, 1)
    yhat = yhat[:, -1, :] # 選擇最後一個時間步的預測值

    # 將 test_X 降維為 2D
    # 這裡選擇最後 n_features-1 特徵的時間步數據並展平
    test_X_flat = test_X.reshape((test_X.shape[0], -1))  # (168, n_hours *
n_features)

    # 選擇最後 n_features-1 的特徵作為額外維度
    test_X_selected = test_X_flat[:, -(n_features-1):]

    # 確認維度一致後執行拼接
    #print("yhat.shape:", yhat.shape)                    # 應為 (168, 1)
    #print("test_X_selected.shape:", test_X_selected.shape)  # 應為 (168,
n_features-1)

    # 拼接並進行縮放反轉
    inv_yhat = concatenate((yhat, test_X_selected), axis=1)
    inv_yhat = scaler.inverse_transform(inv_yhat)
    inv_yhat = inv_yhat[:, 0] # 提取預測值

    # 真實值處理
    test_y = test_y.reshape((len(test_y), 1)) # 確保 test_y 為 2D
    inv_y = concatenate((test_y, test_X_selected), axis=1)
    inv_y = scaler.inverse_transform(inv_y)
    inv_y = inv_y[:, 0] # 提取真實值
    # 計算指標
    mae = mean_absolute_error(inv_y, inv_yhat)
    rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
    mape = mean_absolute_percentage_error(inv_y, inv_yhat) * 100

    # 儲存結果
```

```
    results.append({
        "params": params,
        "mae": mae,
        "rmse": rmse,
        "mape": mape
    })
    print(f"MAE: {mae:.3f}, RMSE: {rmse:.3f}, MAPE: {mape:.2f}%")

# 找到最佳參數
best_result = min(results, key=lambda x: x["mae"])
print("Best parameters found:")
print(best_result["params"])
print(f"MAE: {best_result['mae']:.3f}, RMSE: {best_result['rmse']:.3f}, MAPE:
{best_result['mape']:.2f}%")

'''
from keras.layers import Input, Dense, MultiHeadAttention, LayerNormalization,
Dropout
from keras.models import Model

# Transformer Encoder Block
def transformer_encoder(inputs, num_heads, key_dim, ff_dim, dropout_rate):
    # Multi-head self-attention
    attn_output = MultiHeadAttention(num_heads=num_heads,
key_dim=key_dim)(inputs, inputs)
    attn_output = Dropout(dropout_rate)(attn_output)
    # Skip connection + LayerNorm
    out1 = LayerNormalization(epsilon=1e-6)(inputs + attn_output)

    # Feed-forward network
    ffn_output = Dense(ff_dim, activation="relu")(out1)
    ffn_output = Dense(inputs.shape[-1])(ffn_output)
    ffn_output = Dropout(dropout_rate)(ffn_output)
    # Skip connection + LayerNorm
    return LayerNormalization(epsilon=1e-6)(out1 + ffn_output)

# Define Transformer-based model
```

```
input_layer = Input(shape=(n_hours, n_features))
x = transformer_encoder(input_layer, num_heads=4, key_dim=512, ff_dim=128,
dropout_rate=0.05)
x = transformer_encoder(x, num_heads=4, key_dim=512, ff_dim=128,
dropout_rate=0.05)
x = Dense(512, activation="relu")(x)
x = Dense(128, activation="relu")(x)
output_layer = Dense(1)(x)

model = Model(inputs=input_layer, outputs=output_layer)
model.compile(loss='mae', optimizer='rmsprop')

# Fit the model
history = model.fit(train_X, train_y, epochs=20, batch_size=72, \
                    validation_data=(test_X, test_y), verbose=0, shuffle=False)

# Plot training and validation loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# Make a prediction
yhat = model.predict(test_X)

# 確保 yhat 為 2D (168, 1)
yhat = yhat[:, -1, :]  # 選擇最後一個時間步的預測值

# 將 test_X 降維為 2D
# 這裡選擇最後 n_features-1 特徵的時間步數據並展平
test_X_flat = test_X.reshape((test_X.shape[0], -1))  # (168, n_hours * n_features)

# 選擇最後 n_features-1 的特徵作為額外維度
test_X_selected = test_X_flat[:, -(n_features-1):]

# 確認維度一致後執行拼接
print("yhat.shape:", yhat.shape)                    # 應為 (168, 1)
```

```
print("test_X_selected.shape:", test_X_selected.shape)  # 應為 (168, n_features-1)

# 拼接並進行縮放反轉
inv_yhat = concatenate((yhat, test_X_selected), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]  # 提取預測值

# 真實值處理
test_y = test_y.reshape((len(test_y), 1))  # 確保 test_y 為 2D
inv_y = concatenate((test_y, test_X_selected), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]  # 提取真實值
'''
'''
#calculate 均方根誤差 RMSE(root mean squared error)
rmse=sqrt(mean_squared_error(inv_y, inv_yhat))
#print('Test RMSE: %.3f' % rmse)
#calculate 平均絕對誤差 MAE(mean absolute error)
mae=mean_absolute_error(inv_y, inv_yhat)
#print('Test MAE: %.3f' % mae)
mape = mean_absolute_percentage_error(inv_y, inv_yhat) * 100
#print(f"MAPE: {mape:.2f}%")
'''

test_y,predict_y=inv_y,inv_yhat

import matplotlib.pyplot as plt

plt.plot(predict_y, 'b:') #預測
plt.plot(test_y, 'r-') #real
plt.legend(['predict', 'real'])
plt.xlabel("Ti")
plt.ylabel("pm2.5")
plt.title("Prediction(Transformer)")
plt.show()
```

```
#from keras.model import load_model
model.save(filenameModel)
#print("儲存 Model",filenameModel)

#將預測結果與真實資料合併成 DataFrame
#from pandas import DataFrame
df_PredResult=DataFrame({"Real":inv_y,"Pred": inv_yhat})
#print("DataFrame: Pred,Real")
#print('顯示前 5 筆資料')
#print(df_PredResult.head())
#print("儲存檔案",filenameOut)
df_PredResult.to_csv(filenameOut)
# Calculate the mean of the actual test values
mean_value = inv_y.mean()
'''
mae_list.append(mae)
rmse_list.append(rmse)
'''
endtime=datetime.datetime.now()

print('開始時間: ',starttime)
print('演算時間: ',Algorithmtime)
print('結束時間: ',endtime)
print('花費時間: ',endtime-starttime)
# Calculate MAE & RMSE
#print('第',i+1,'次訓練')
print('Test MAE: %.3f' % mae)
print('Test RMSE: %.3f' % rmse)
print('Test MAPE: %.2f%%' % mape)
'''
# 計算 MAE 和 RMSE 的平均值
mean_mae = sum(mae_list) / 10
mean_rmse = sum(rmse_list) / 10
print(f'10 次訓練的平均 MAE: {mean_mae:.3f}')
print(f'10 次訓練的平均 RMSE: {mean_rmse:.3f}')
'''
```