計算機演算法 期末論文報告

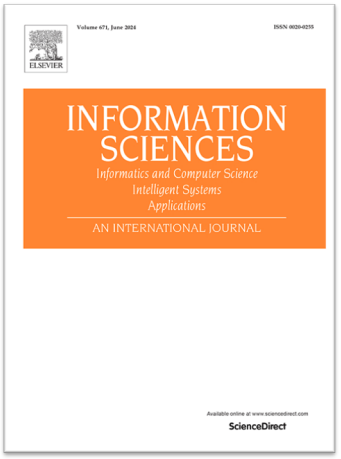
M11202117 呂長恩

論文網址：[Ensemble k-nearest neighbors based on centroid displacement - ScienceDirect]

Ensemble k-nearest neighbors based on centroid displacement

Journal：
Information Science

Author：
Yuanyuan Zhu, Lu Qin, Jeffrey Xu Yu, Hong Cheng



Introduction

分類器的種類與應用

- 分類器是一種監督式機器學習的演算法，需要有訓練資料才能運算。流程為先使用訓練資料訓練出模型後，再將測試資料輸入模型內得到該測試資料對應的類別(標籤)。
- 常見的分類演算法有：
  - **決策樹（Decision Tree）**

    決策樹是一種樹形結構的模型，每個節點表示一個特徵，每個分支表示特徵的取值，每個葉子節點表示一個類別。

- **隨機森林（Random Forest）**

  隨機森林是由多棵決策樹組成的集成學習算法，每棵樹由一個隨機子集的數據和特徵訓練而成。隨機森林通過集成多棵樹的結果來提高分類的準確性和穩定性，並能有效防止過擬合。

- **支持向量機（Support Vector Machine, SVM）**

  SVM 通過找到一個超平面來最大化類別間的間隔，以實現最佳分類。SVM 對於高維數據具有良好的性能，並且可以使用核函數 (kernel function) 來處理非線性問題。但會需要花費更久的運算時間。

- **邏輯回歸（Logistic Regression）**

  邏輯回歸是一種線性分類模型，通過學習數據的特徵權重，使用 sigmoid 函數將輸出映射到概率值範圍內。該算法適合於二元分類問題。

- **朴素貝葉斯（Naive Bayes）**

  朴素貝葉斯基於貝葉斯定理，假設特徵之間相互獨立。該算法計算每個類別的後驗概率，並選擇概率最大的類別作為預測結果。朴素貝葉斯在文本分類等應用中效果特別好。

- **高斯混合模型（Gaussian Mixture Model, GMM）**

  GMM 是一種概率模型，假設數據由多個高斯分布組成。通過估計每個高斯成分的參數，可以對數據進行分群或分類。

- **神經網絡（Neural Networks）**

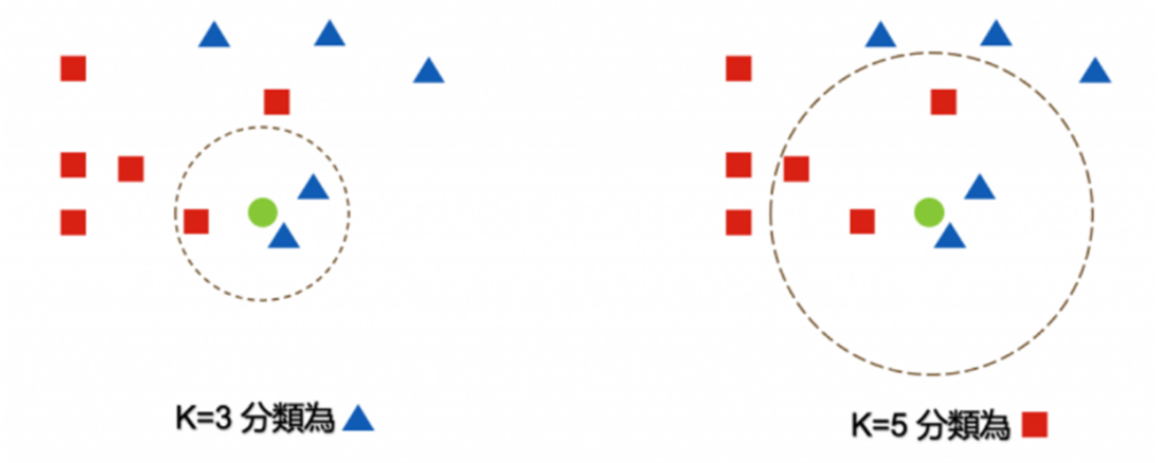
  神經網絡是一種模仿生物神經網絡的非線性模型，由多層神經元組成。每個神經元接收輸入並應用激活函數，然後將輸出傳遞給下一層。

  這篇所使用的是 **K 近鄰演算法 (k-nearest neighbors, K-NN)** 來進行分類，這將會在下個章節介紹。

- 分類器的常見應用在：臉部辨識、文字辨識、圖論分析、疾病判斷、異常偵測等議題上。

K 近鄰演算法 (K-Nearest Neighbors Classification)

K 近鄰演算法是一種基於**多數投票決**的分類機制。在找到離測試資料點最鄰近的 k 個訓練資料點後進行投票，將該測試點分類到多數點的類別上。



舉例而言，左圖中的 k 為 3 ，所以找到離綠點最近的 3 個點，其中有兩個點是藍色的類別、有一個是紅色的類別。因此在多數決的投票下，綠點最終會被分類為藍色類別。

而右圖中的 k 為 5 ，所以找到離綠點最近的 5 個點，其中有兩個點是藍色的類別、有三個是紅色的類別。因此在多數決的投票下，綠點會被分類為紅色類別。

KNN 的優點如下：

- 簡單容易理解
- 良好的泛用性
- 不需要對資料進行前提假設
    - 針對不同分佈、密度的資料集都適用，不會因為密度的高低差異導致嚴重的分類錯誤。
- 良好的彈性與整合性
    - KNN 具有良好的彈性，可以使用不同的距離量度單位來找到最近的鄰居點，可以根據使用狀況進行調整。容易被整合到其他演算法當中或者延伸出新的演算法。

KNN 的缺點如下：

- 過於簡單的判斷機制
    - 多數決投票機制過於簡陋，沒有扎實的理論基礎使用。
- 計算成本高

    - 計算歐氏距離需要使用到大量的計算成本，當點數量或者維度提高後會使用大量的運算資源與記憶體空間儲存資料。
- 對特徵縮放敏感
    - 使用歐式距離下，不同比例尺下的特徵會放在一起被加總。導致某些較小比例尺的特徵容易被大比例尺的特徵吸收，造成在結果上忽略了某些特徵考量。
- 選擇 K 值困難
    - K 值的選擇沒有一定的理論基礎。不同資料集有不同適當的 K 值，導致常使用暴力解找到對應最適合的 K 值。

本論文將針對**「過於簡單的判斷機制」**進行改良，使用不同的判斷機制來取代簡單多數決。

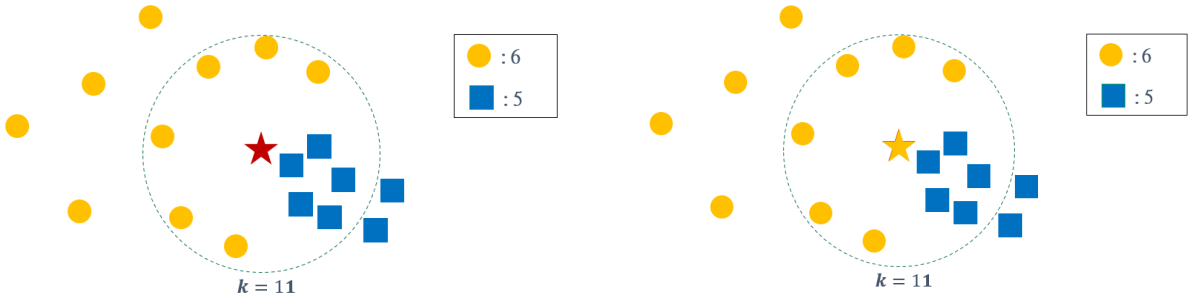
本論文的貢獻

本篇論文提出基於 KNN 機制上的改良分類演算法，取代使用簡單的多數決，並使用**群心變動量**來判斷分類結果。論文貢獻如下。

1. 提出的 ECDNN 算法直觀且易於執行和解釋預測。
2. 對各種數據集的廣泛實驗結果顯示了 ECDNN 的計算效率和預測準確性。
3. 這項研究為開發類似演算法提供了方向，這些演算法可以利用最近鄰的同質性作為集成學習的基礎，並且根據資料集特性應用不同的演算法。

Definition

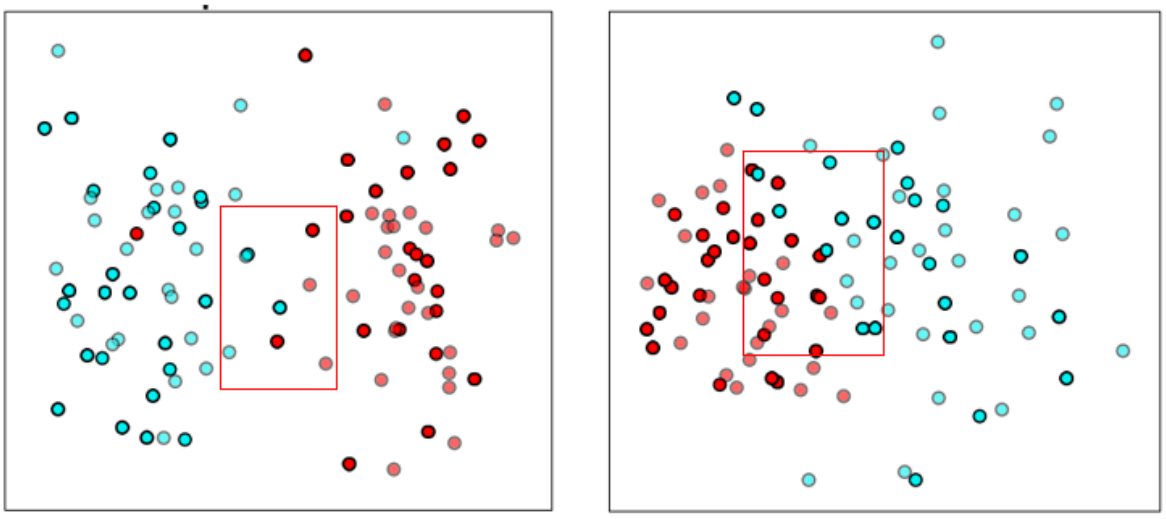
問題描述



可以看到左圖中的紅色星形點在 KNN 的機制下，找到最近的 11 個點。依圖而言，他應該會被分作藍色的類別比較適當。然而因為資料點的分佈和密度問題，導致紅色點被分為黃色的類別，如同右圖所示。

這顯露出簡單多數決的機制並不一定適用於所有的資料分佈下，因此可以對它提出改善機制。

這類問題常發生於如下：



1. 不同密度的區域 (Heterogeneous)
2. 過於複雜的資料集
3. 類別交界處

提出的演算法

- **Ensemble Centroid Displacement-based k-NN (ECDNN)**
  - 基於 K-means 演算法結合 KNN 判斷機制，解決上述問題。
  - 新增的自信度的判斷機制降低運算時間。

演算法相關定義

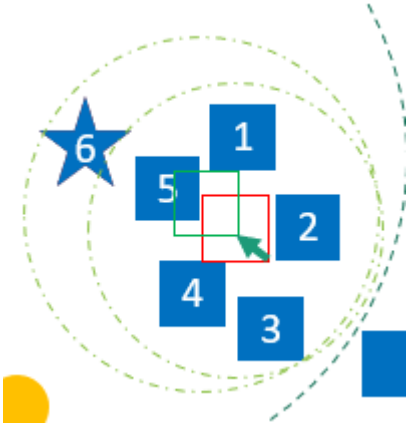
Centroid Displacement

- 每一個群都可以利用下述公式得出質心點 (centroid)，亦即該群的平均點 (mean point)。

$$\mu_i = \frac{1}{n_i} \sum_{x_j \in C_i} x_j$$

- 當質心變動的時候，變動前和變動後的質心位置差變是質心變動量 (centroid displacement)。

$$\delta_i = |\mu_{i(t)} - \mu_{i(t-1)}|$$



舉例而言，可以看到 $p_1, ..., p_5$ 可以得到紅色的質心點。加入 p_6 後，重新計算可以得到心的質心點，綠點。而由紅點移動到綠點的變動量，即是質心變動量。

Confidence

- 自信度 (Confidence) 代表一個點它周圍鄰居的類別數量。當一個點它的 KNN 都是由不同的類別的點構成，則為低自信度 (low confidence)。反之，當一個點它的 KNN 都是由同一個類別的點構成，則為高自信度 (high confidence)。

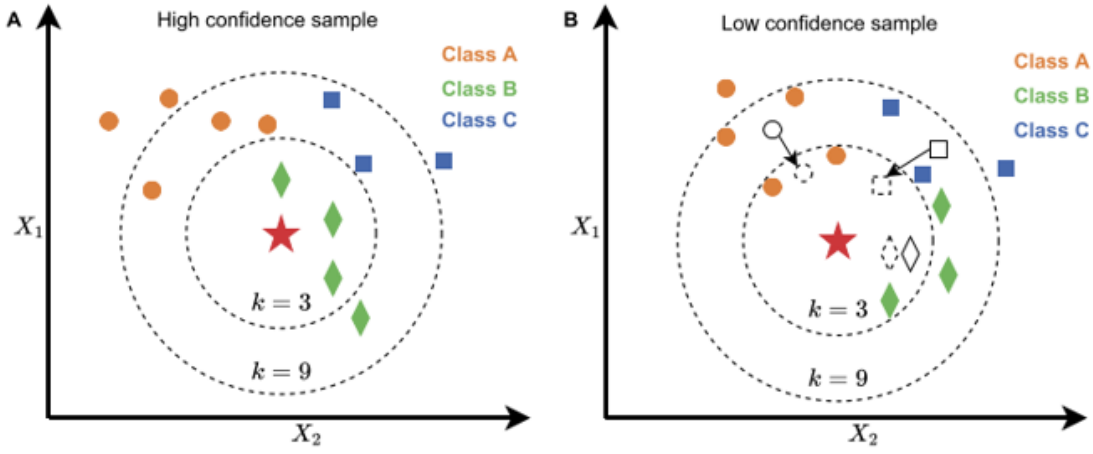


Fig. 1. Visual illustration of the proposed algorithm.

舉例而言，左圖中的紅點，在 $k = 3$ 時，周遭的鄰居都來自綠色的群，如此該紅點便屬於高自信度。而右圖當中，紅點的 KNN 是由兩個橘點和一個綠點所構成，如此該紅點便是屬於低自信度。

Algorithm

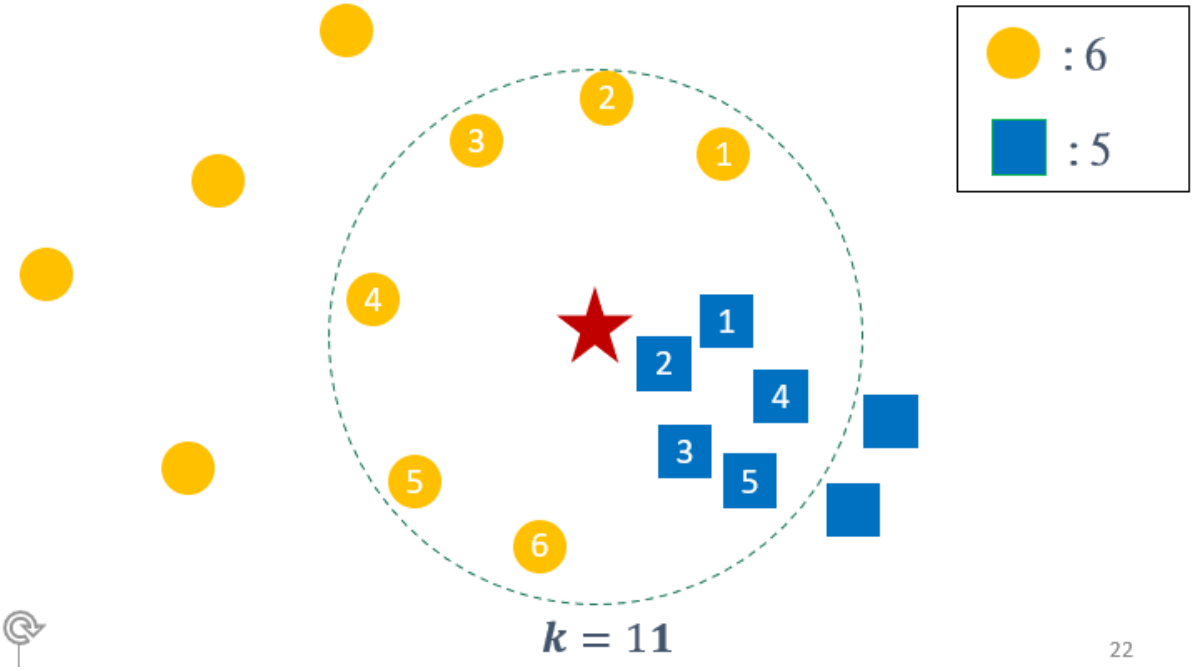
輸入資料

1. Trained Dataset: $D \in \{X_1, ..., X_i\}$
2. Testing Instance: x
3. Number of nearest neighbors: k

演算法流程

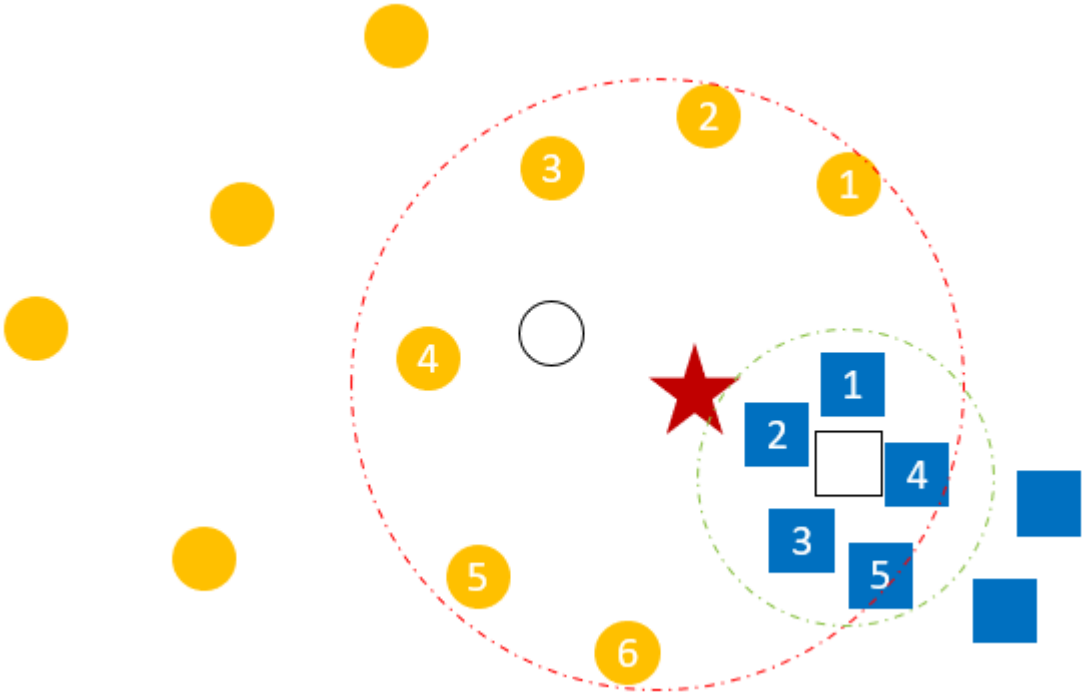
Centroid Displacement-based k-NN (CDNN)

1. 計算待測點 x 和所有點的歐氏距離，並且利用 Max-heap 去儲存離 x 最近的 k 個點。
2. Max-heap 內會存放所有 x 的 KNN 們的 ID。
3. 計算在 KNN 內各個類別的點數量。



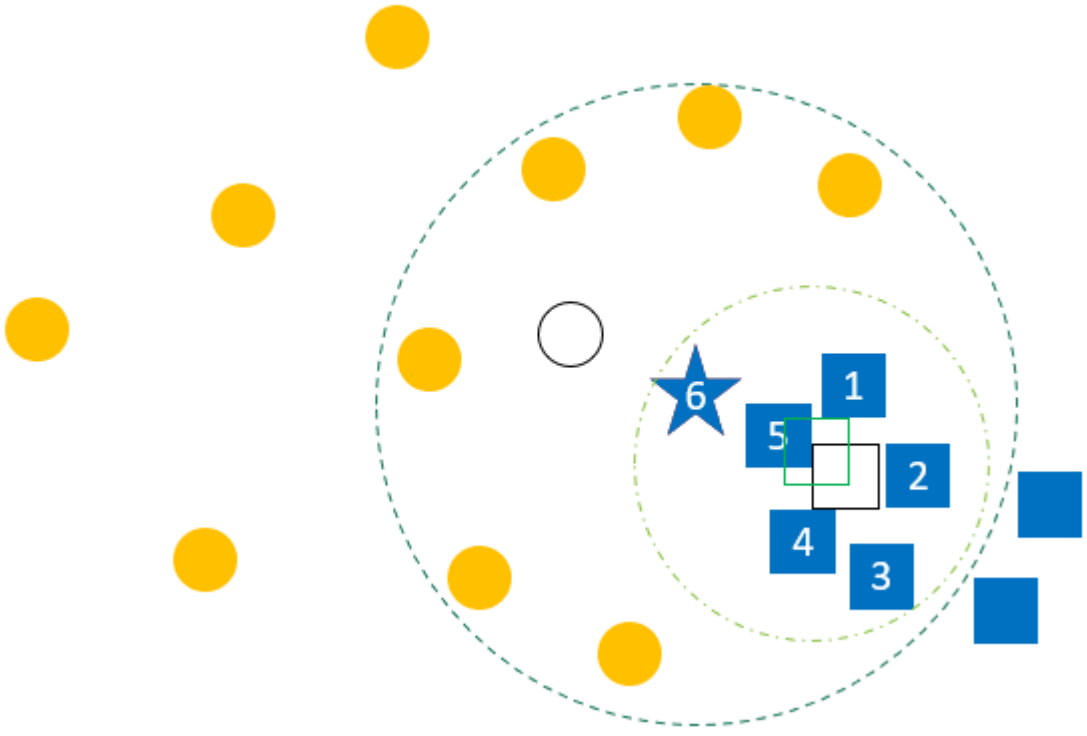
由圖可以得知，紅色的 KNN 範圍即是圓圈的範圍。其中有6顆屬於黃色的群，5顆屬於藍的群。

4. 找尋 KNN 中每一群的質心點位置。

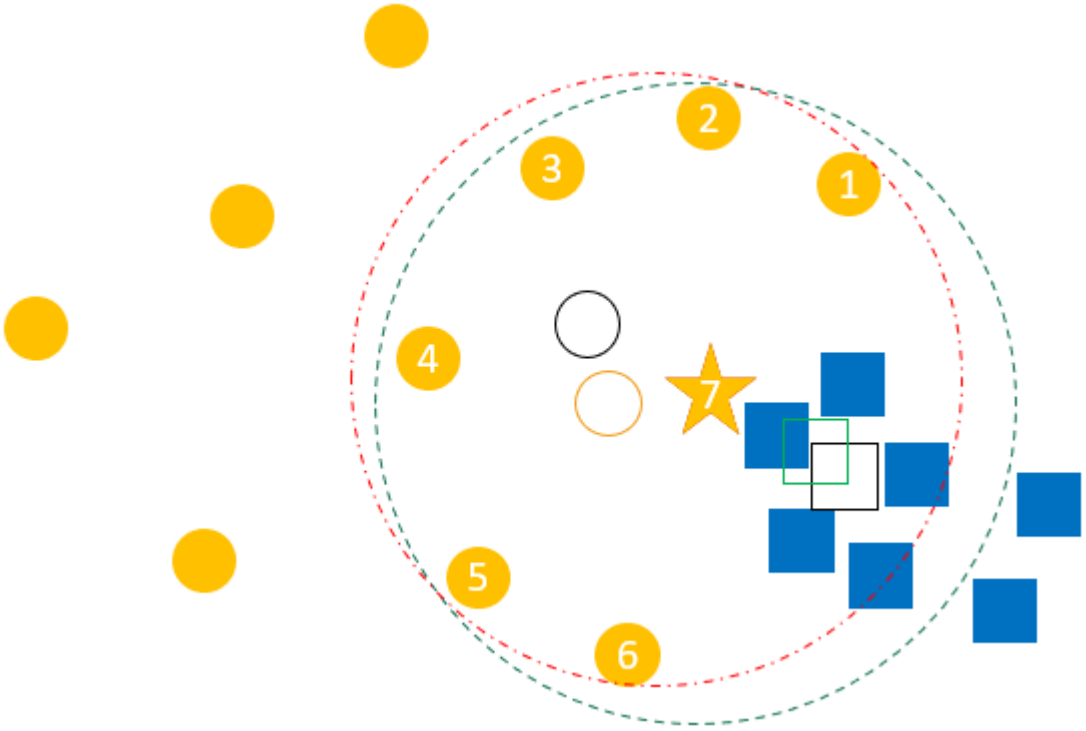


由圖可以得知，藍色類別的質心為方塊，黃色點的質心為圓形。

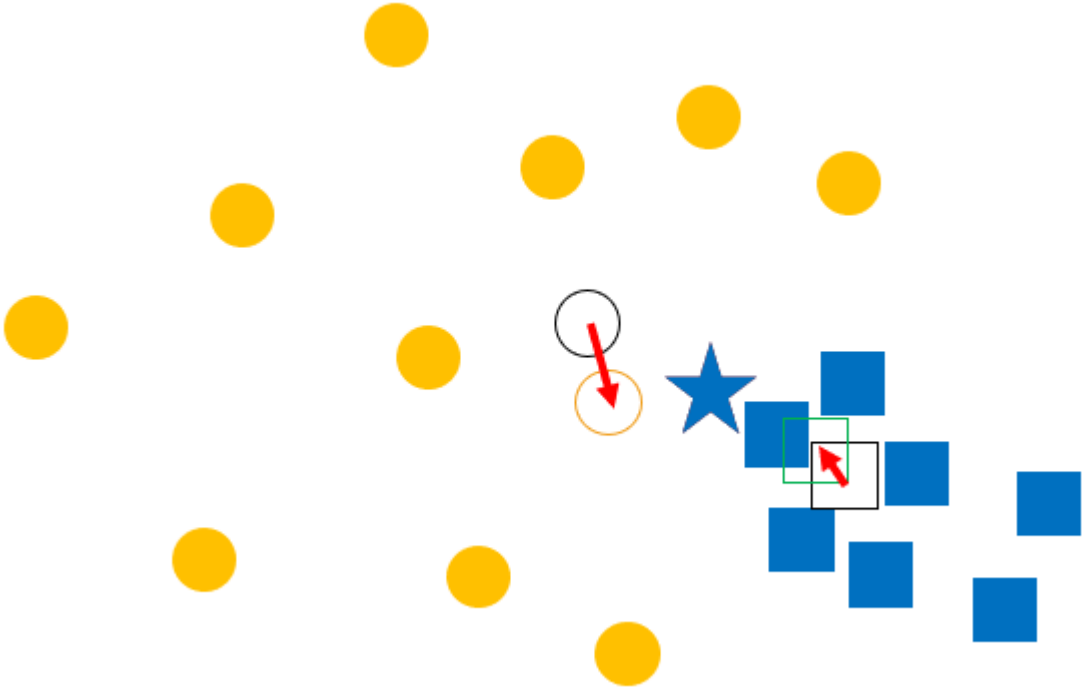
5. 加入 x 後，重新計算藍色類別的質心。



6. 加入 x 後，重新計算黃色類別的質心。



7. 計算各個類別的質心移動量。

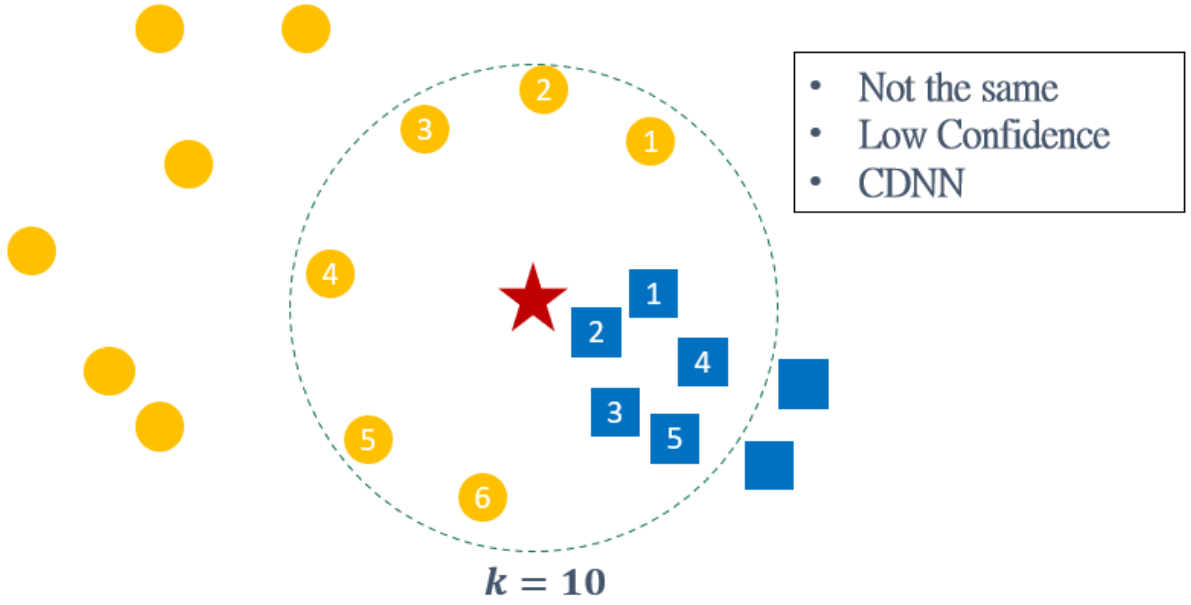


8. 找到最小的質心移動量的類別，並將 x 劃分到該類別。

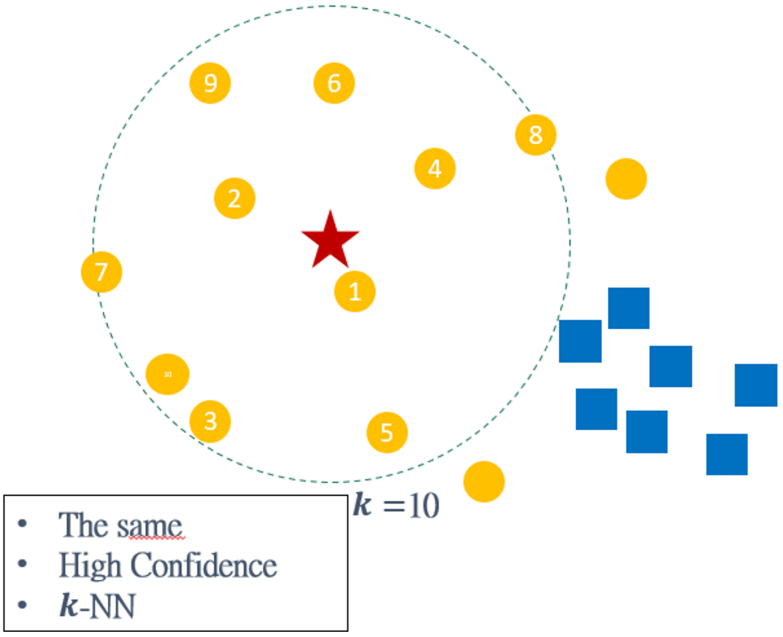
Ensemble Centroid Displacement-based k-NN (ECDNN)

- 考慮 KNN 的組成

1. KNN 由不同的類別所組成，低自信度，使用 CDNN



2. KNN 皆由相同的類別所組成，高自信度，直接使用 KNN



時間複雜度

功能	時間複雜度
找尋 KNN	O(n^2)

功能	時間複雜度
計算 KNN 內群的數量	O(k)
計算各群的原始質心位置	O(k)
計算更心質心位置	O(k)
找到各群的最小移動量	O(i), i 是類別的數量

- 總和的時間複雜度：$O(n^2)$

Experiment

比較演算法

1. KNN
2. WKNN (Weighted KNN)
    a. 基於 KNN 的延伸演算法。舉例來說，比如今天我們要根據體重和身高進行分類。我們通常用 kg表示體重，而身高用 cm 表示。一個通常是2 位數，另一個是 3 位數。假如我們直接根據這個數據做分類，可能會因為 cm 是 3 位數而佔較重的權重，因此我們加上一個權重去衡量，有點像是對其正規化。
3. Radius NN (RNN)
    a. 類似 DBSCAN，考慮在一個半徑內所有的鄰居來決定分類，因此 DBSCAN 會發生的問題他也會發生。會無法處理 varying-density dataset。
4. NC (Nearest Centroid Neighbors)
    a. 將所有的 trained data 的 class 去計算一個質心點，找一個離 instance x 最近的質心點劃分進去。
5. CDNN
6. ECDNN

實驗參數設計

- $K = [5, 25]$, 每次加 2
- Radius 使用該演算法推薦的參數
- 實驗重複執行 10 次，並且使用 5-fold cross validation 去分訓練集和測試集。
- 總共跑了 50 次，再取平均。

評量指標

Macro-F1 Score

$$F1 - scroe_i = \frac{Precision * Recall}{Precision + Recall}$$

$$Macro - F1 = \frac{F1 - score_1 + F1 - score_2 + ... + F1 - score_c}{c}$$

- Macro-F1 的區間在於 [0, 1] 之間，數值越大越好。
- 這是 Recall 和 Precision 的調和平均數。
- 由於每一個 class 都是獨立計算 F1-scroe 再取平均，所以不會受每一個 class 的數量多寡影響整體的 Macro-F1，對於不平衡資料集有較好的評量性。

測試資料集

Name	Source	$\#C$	IR	$\#S$	$\#F$	$\#F/\#S$	$\#F_{PCA}/\#S$	$\#F/\#F_{PCA}$	Sparsity
iris	UCI	3	-	150	4	0.0267	0.0133	2.00	0.0000
wine	UCI	3	-	178	13	0.0356	0.0223	1.60	0.4971
breastcancer	UCI	2	1.7:1	569	30	0.0527	0.0176	3.00	0.0047
digits	UCI	10	-	1797	64	0.0730	0.0562	1.30	0.0000
olivetti	UCI	40	-	4096	400	10.2400	0.3075	33.30	0.0000
paris	UCI	2	1:1	5828	2200	2.6491	0.1314	20.17	0.0001
S1	Synthetic	2	1:1	100	2	0.0200	0.0200	1.00	0.0000
S2	Synthetic	2	1:1	1000	2	0.0020	0.002	1.00	0.0000
S3	Synthetic	3	-	1000	10	0.0100	0.0090	1.11	0.0000
S4	Synthetic	3	-	1000	50	0.0500	0.0420	1.19	0.0000
S5	Synthetic	10	-	5000	50	0.0100	0.0092	1.09	0.0000
S6	Synthetic	10	-	5000	100	0.0200	0.0182	1.10	0.0000
ecoli	UCI	2	8.6:1	336	7	0.0208	0.0179	1.17	0.0020
optical_digits	UCI	2	9.1:1	5620	64	0.0114	0.0075	1.52	0.4961
satimage	UCI	2	9.3:1	6435	36	0.0056	0.0009	6.00	0.0000
pen_digits	UCI	2	9.4:1	10992	16	0.0015	0.0009	1.60	0.1369
abalone	UCI	2	9.7:1	4177	10	0.0024	0.0010	2.50	0.2223
sick_euthyroid	UCI	2	9.8:1	3163	42	0.0133	0.0057	2.33	0.4467
spectrometer	UCI	2	11:1	531	93	0.1751	0.0075	23.25	0.0000
car_eval_34	UCI	2	12:1	1728	21	0.0122	0.0087	1.40	0.7500
isolet	UCI	2	12:1	7797	617	0.0791	0.0260	3.04	0.0036
us_crime	UCI	2	12:1	1994	100	0.0502	0.0176	2.86	0.0560

實驗結果

dataset	*k*-NN		WKNN		Radius NN		NC		CDNN		ECDNN	
	F1	rank	F1	rank	F1	rank	F1	rank	F1	rank	F1	rank
iris	0.9513	4	0.9586	2	0.9406	5	0.8574	6	**0.9599**	1	0.9518	3
wine	0.9681	3	0.9665	4	0.4705	6	0.9744	1	0.9721	2	0.9664	5
breastcancer	0.9571	4	0.9609	3	0.6193	6	0.9246	5	0.9626	2	**0.9659**	1
digits	0.9637	4	0.9669	3	0.1815	6	0.8889	5	0.9704	2	**0.9763**	1
olivetti	0.6914	5	0.7089	4	0.0527	6	0.8775	1	0.8222	3	0.8771	2
paris	0.5946	4	0.5881	5	0.5977	3	0.5181	6	0.5993	2	**0.6049**	1
S1	0.8844	5	0.9019	3	**0.9095**	1	0.8243	6	0.9009	4	0.9074	2
S2	**0.8992**	1	0.8943	2	0.8781	5	0.8408	6	0.8936	3	0.8899	4
S3	0.979	4	0.9798	3	0.5319	6	0.8736	5	0.9814	2	**0.9837**	1
S4	0.9642	4	0.9659	2	0.4655	6	0.7729	5	**0.9686**	1	0.9655	3
S5	0.9026	4	0.9095	3	0.1802	6	0.5837	5	**0.9181**	1	0.9140	2
S6	0.9075	4	0.9154	3	0.1802	6	0.5644	5	**0.9252**	1	0.9177	2
Average	0.8886	4	0.8931	3	0.5006	6	0.7917	5	0.9062	2	**0.9101**	1
ecoli	0.7892	2	**0.7974**	1	0.7754	5	0.6926	6	0.7881	3	0.7816	4
optical_digits	0.9769	4	0.9773	3	0.4741	6	0.7652	5	0.9791	2	**0.9823**	1
satimage	0.8127	4	0.8149	3	0.6901	5	0.4727	6	0.8222	2	**0.8340**	1
pen_digits	0.9933	4	0.9939	3	0.9726	5	0.5860	6	0.9949	2	**0.9964**	1
abalone	0.5304	4	0.5337	3	0.4754	6	**0.5842**	1	0.5271	5	0.5497	2
sick_euthyroid	0.7134	4	0.7451	3	0.6123	5	0.5332	6	0.7583	2	**0.7718**	1
spectrometer	0.8524	4	0.859	3	0.4804	6	0.6304	5	0.8692	2	**0.8727**	1
car_eval_34	0.7099	5	0.7274	3	0.4798	6	0.7251	4	0.7547	2	**0.7640**	1
isolet	0.9140	3	0.9140	3	0.4800	6	0.6018	5	0.9156	2	**0.9173**	1
us_crime	0.6241	5	0.6302	4	0.4804	6	**0.6862**	1	0.6428	3	0.6548	2
Average	0.7916	4	0.7993	3	0.5920	6	0.6277	5	0.8052	2	**0.8125**	1

Wine

Complexity

Dataset	$\#C$	IR	$\#S$	$\#F$	$\frac{\#F}{\#S}$	$\frac{\#F_{PCA}}{\#S}$	$\frac{\#F}{\#F_{PCA}}$	Sparsity
Wine	3	-	178	13	0.0356	0.0223	1.60	0.4971

Algorithm	F1	Rank
K-NN	0.9681	3
WKNN	0.9665	4
Radius NN	0.4705	6
NC	0.9744	1
CDNN	0.9721	2
ECDNN	0.9664	5

- 當 dataset 越複雜， #FPCA 會越高，所以從中挑出兩者一個較大一個較小的可以發現，在複雜的 dataset 分布下， Radius 的效果會較差，DBSCAN的問題會出現。

Olivetti

A lot of clusters

Dataset	$\#C$	IR	$\#S$	$\#F$	$\frac{\#F}{\#S}$	$\frac{\#F_{PCA}}{\#S}$	$\frac{\#F}{\#F_{PCA}}$	Sparsity
Olivetti	40	-	4096	400	10.24	0.3075	33.30	0.0000

Algorithm	F1	Rank
K-NN	0.6914	5
WKNN	0.7089	4
Radius NN	0.0527	6
NC	0.8775	1
CDNN	0.8222	3
ECDNN	0.8771	2

- Class 數量眾多時，代表我們在 kNN 裡面會有高機率出現多個不同的 class，因此多數投票決的問題會很有可能出現，使準確度降低。而 Radius 也會降低，也是因為上面的原因加上 class 數量越多越容易出現 varying densities 使其準確度更低。

car_eval_34

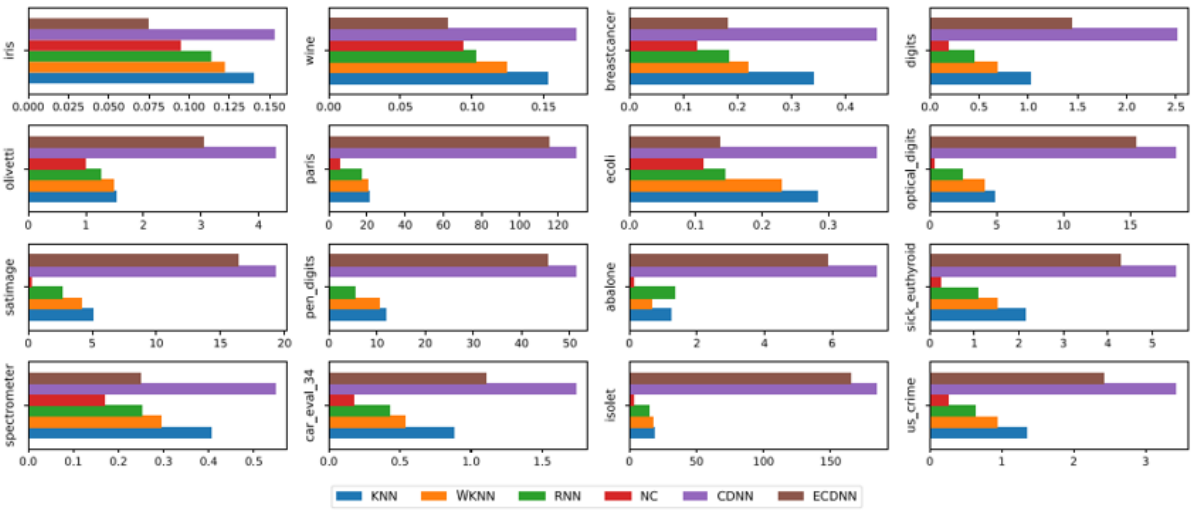
Imbalance

Dataset	$\#C$	IR	$\#S$	$\#F$	$\frac{\#F}{\#S}$	$\frac{\#F_{PCA}}{\#S}$	$\frac{\#F}{\#F_{PCA}}$	Sparsity
Car_eval_34	2	12:1	1728	21	0.0122	0.0087	1.40	0.7500

Algorithm	F1	Rank
K-NN	0.7099	4
WKNN	0.7274	3
Radius NN	0.4798	5
NC	0.7251	6
CDNN	0.7547	2
ECDNN	0.7640	1

- 此 dataset 很不平衡，並且在高維上， sparsity 又高，這使多數決機制出現問題。再來就是 sparsity 在高維上，由於很多的 elements 都是 0，使其在一定範圍內的分類更難執行，所以 Radius NN 更難分類。

執行時間



- 由圖可知，由於 ECDNN 多了判斷自信度的機制，使不是所有測試點都需要做 CDNN，所以 ECDNN 的速度皆快於 CDNN。

實作實驗

測試實驗資料集

- Iris
- Breastcancer
- Satimage
- Abalone

測試演算法

- ECDNN
- CDNN
- KNN
- Radius-NN (RNN)

測試環境

程式語言

C++ (ver. 9.4.0)

硬體規格

- Intel Core I7-8700 @ 3.20GHz
- DDR4 16GB
- Ubuntu 20.04

實驗參數設計

- $K = [4, 26]$, 每次加 1
- $Radius = [0.5, 30]$, 每次加 0.5
- 實驗重複執行 20 次，並且使用 5-fold cross validation 去分訓練集和測試集。
- 總共跑了 100 次，再取平均。

Macro-F1 實驗結果

Iris

Algorithm	Paper	My Result(k)
ECDNN	0.9518	**0.9710(14)**
CDNN	**0.9599**	**0.9710(14)**
KNN	0.9513	0.9690(11)
RNN	0.9406	0.9501(1.0)

breastcancer

Algorithm	Paper	My Result(k)
ECDNN	**0.9659**	**0.9331(17)**
CDNN	0.9626	**0.9331(17)**
KNN	0.9571	0.9304(10)
RNN	0.6139	0.9081(30.0)

satimage

Algorithm	Paper	My Result(k)
ECDNN	**0.8340**	**0.7698(4)**
CDNN	0.8222	**0.7698(4)**
KNN	0.8127	0.7626(5)
RNN	0.6901	0.7167(2.0)

abalone

Algorithm	Paper	My Result(k)
ECDNN	**0.5497**	**0.5331(24)**
CDNN	0.5271	**0.5331(24)**
KNN	0.5304	0.5306(5)
RNN	0.4754	0.4988(0.5)

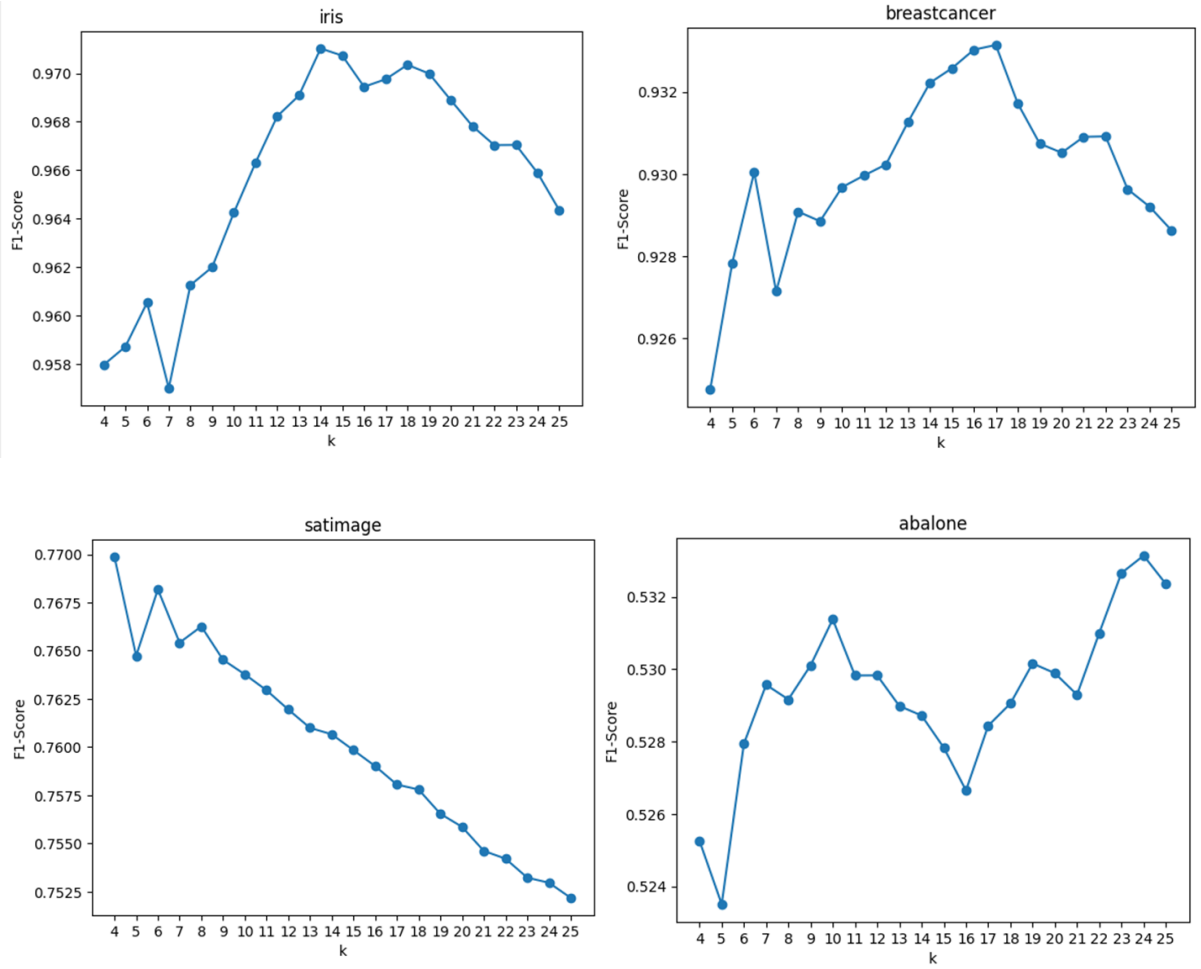
- 很符合預期 CDNN、ECDNN 都會是裡面測試後，準確度最高的演算法。

運算時間結果

Algorithm	Iris (ms)	breastcancer (ms)	satimage (ms)	abalone (ms)
ECDNN	257	6453	759819	134499
CDNN	272	6697	768572	137200
KNN	237	**6242**	757313	128028
RNN	**207**	6261	**748977**	**126160**

- 可以看到在速度上，KNN 快於 ECDNN 快於 CDNN，符合預期。

不同 k 值下 ECDNN 準確度的變化



- 由圖可以知道，k 值的選去會影響到準確度的變化，這也是所有 KNN-based 的演算法最需要注意的地方。

Conclusion

1. ECDNN 適用於那些有不同密度大小的資料集上。
2. ECDNN 的速度雖然慢於 KNN，但自信度的判斷機制使其快於 CDNN。
3. ECDNN 的精準度等同於 CDNN，且優於 KNN。
4. KNN-based 的演算法必須要考慮到 k 值的選取。

Feedback

這是的重現實驗使我明白了論文上的數據不一定都是真的。在實驗的過程中，看到 dataset table 的某一個資料集寫有 2 個 class，然而我從 UCI 下載後發現其實有 3 個 class 存在。導致我在還原實驗結果上存在嚴重的差異性，無法還原到如它論文表示那麼好的準確度，這點我認為作者在撰寫論文上存在瑕疵。此外，在實現比較演算

法中，我發現我寫的 RNN 比它跑出來的準確度更高，其實沒有輸到那麼慘。我認為可能是作者在撰寫該演算法時，沒有選到最合適的參數大小導致的，像是 RNN 這樣的演算法會對 radius 的大小敏感，而作者可能沒有找到適當的大小來實驗。這提醒了我在還原比較演算法時必須保持公平才有價值，不應該為了投稿等目的、將別的比較演算法寫爛。