**第六組 罹患糖尿病演算法**

**目錄**

* **簡介**
* **研究內容**
* **模型表現的含義**

**簡介**

1. **何謂資料分類演算法?**

資料分類演算法是一種監督是學習演算法，用來分類、整理資料，讓我們瞭解資料的特徵與分布。我們可以透過監督他學習訓練資料及樣本，建立模型後對未知的資料集進行分析。

1. **有哪些分類演算法?**

KNN, Random Forest Tree, Feature Selection, SVM, 樸素貝葉斯……等等。

1. **為什麼有那麼多分類法?**

每種演算法都有自己的獨特性格，人們通常透過以下指標來評估演算法的表現:

1. **準確率**: 正確的樣本數佔總樣本數的比例
2. **精確率:** 正確的正樣本數佔分類為正樣本的樣本數的比例
3. **召回率**: 分類正確的正樣本數佔正樣本總數的比例
4. **F1值**: 綜合了精確率和召回率的指標

通過以上指標，你可以選擇自己偏好的演算法。

1. **研究簡介:** 利用各種演算法，先通過訓練資料集訓練模型，再放入測試資料集，預測是否有糖尿病。

**研究內容**

* + 1. **K-近群分群法(K-Nearst Neighbor, KNN)**

**簡單概念:** 根據資料點彼此之間的距離來進行分類，距離哪一種類別最近則該資料點就會被分到哪類。

一張含有 文字, 圖表, 圓形, 字型 的圖片

AI 產生的內容可能不正確。

**實作方法:**

1. def compute\_distance(self, point\_1, point\_2):
2. if(self.distance\_method == 'euclidean'):
3. return np.sqrt(np.sum((point\_1 - point\_2) \*\* 2))
4. elif(self.distance\_method == 'manhattan'):
5. return np.sum(np.abs(point\_1 - point\_2))

用「歐幾里得距離」或「曼哈頓距離計算」距離。

1. def predict(self, x\_test):
2. predictions = []
3. for point in x\_test:
4. distances = [self.compute\_distance(point, train\_point) for train\_point in self.x\_train]
5. k\_idx = np.argsort(distances)[:self.k] # 找出前 k 個最近的鄰居
6. k\_nearest\_labels = [self.y\_train[i] for i in k\_idx] # 鄰居的標籤
7. most = np.argmax(np.bincount(k\_nearest\_labels)) # 投票結果
8. predictions.append(most)
9. return np.array(predictions)

對測試數據集進行預測，它計算每個測試點與所有訓練點的距離，找出 與**k** 個最近的鄰居，然後進行多數投票以確定預測的類別。

1. def cross\_validation(self, x, y, folds=5):
2. fold\_size = len(x) // folds
3. scores = []
4. for i in range(folds):
5. test\_start = i \* fold\_size
6. test\_end = (i + 1) \* fold\_size if i != folds - 1 else len(x)
7. x\_train = np.concatenate([x[:test\_start], x[test\_end:]], axis=0)
8. y\_train = np.concatenate([y[:test\_start], y[test\_end:]], axis=0)
9. x\_test = x[test\_start:test\_end]
10. y\_test = y[test\_start:test\_end]
11. self.fit(x\_train, y\_train)
12. scores.append(self.evaluate(x\_test, y\_test))
13. return np.mean(scores)

**k 折交叉驗證:** 將測試集分成K個(折)，抓出一折為測試集，其他為訓練集，重複K次，回傳指標性能。

**Euclidean跟Manhattan比較:**

一張含有 文字, 螢幕擷取畫面, 字型, 設計 的圖片

AI 產生的內容可能不正確。 # Euclidean一張含有 文字, 螢幕擷取畫面, 字型, 設計 的圖片

AI 產生的內容可能不正確。 # Manhattan

兩者相似，都用Manhattan就好，比較穩定。

* + 1. **隨機森林 Embedded Random Forest**

**簡單概念:** 舉例以下的決策為一棵樹，隨機森林則是很多不同結構的樹所形成的集合。因為每棵樹都有自己的決策方式，我們稱之為過度配置(Over-fitting、過度擬合、過適)，就是單一模型太符合某一面向，導致完全不適用於其他地方(不夠多樣性)。為了解決偏見，我們決定綜合一堆樹，取他們(特徵的重要性)的平均。

我們只是用Random Forest 找出較無關聯的特徵，讓KNN可以決定要不參考哪些(兩個)特徵。

**Embedded feature selection:** 以Random Forest 訓練模型取得個特徵重要程度，用於決定需要剃除哪兩個特徵，篩選完後的準確率提升到接近100 %。

一張含有 文字, 螢幕擷取畫面, 圖表, 字型 的圖片

AI 產生的內容可能不正確。

**實作方法:**

1. def random\_forest\_feature\_importance(df):
2. x = df.drop(columns=['Outcome'])
3. y = df['Outcome']
4. rf = RandomForestClassifier(n\_estimators=100, random\_state=42)
5. rf.fit(x, y)
6. importances = rf.feature\_importances\_
7. return importances, list(x.columns)

feature\_importances\_: 回傳計算特徵的重要性。

**實作結果:**

一張含有 文字, 螢幕擷取畫面, 圖表, 數字 的圖片

AI 產生的內容可能不正確。

* + 1. **支援向量機Support vector machines, SVM**

**簡單概念:** 找出一個對各點有最大間隔的超平面。

一張含有 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

**實作方法:**

1. def fit(self, x\_train, y\_train):
2. self.model = SVC(kernel=self.kernel, C=self.C, random\_state=40)
3. self.model.fit(x\_train, y\_train)`

**fit擬合模型:** 找出超平面後，依據訓練模型調整內部特徵重要性。

1. def cross\_validation\_for\_C(self, x, y, C\_values, cv=5):
2. best\_C = None
3. best\_score = 0
4. for c in C\_values:
5. self.C = c
6. score = self.cross\_validation(x, y, folds=cv)
7. if score > best\_score:
8. best\_score = score
9. best\_C = c
10. return best\_C

遍歷每個 **C** 值，使用交叉驗證計算準確率，尋找最適合的正則化參數’C’ ，適應各種強度模型。

(選擇K & C 值方法皆是Cross Validation -> 避免overfitting 或是 underfitting 提升model reliability)

* + 1. **KNN V.S SVM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | KNN\_accuracy | **KNN\_Recall** | **SVM\_Accuracy** | **SVM\_Recall** |
| Test-a | 0.7910447761 | 0.605633802 | 0.766169154 | 0.5352112676 |
| Test-b | 0.81 | 0.621621621 | 0.81 | 0.5945945945 |

**結論:** 兩者相似。若對正樣本的漏檢非常敏感，KNN較適用。若資料具有高維特性，SVM較適用。

* + 1. **Missing data performance**

**舉例:** KNN performance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | **With Missing Handle Accuracy** | **Without Missing Handle Accuracy** | **With Missing Handle Recall** | **Without Missing Handle Recall** |
| Test-a | 0.79104477611 | 0.7761194029850 | 0.60563380281690 | 0.5633802816 |
| Test-b | 0.81 | 0.75 | 0.62162162162162 | 0.4324324324 |

印出missing handle or not的比較結果，如果要印出其他比較實驗結果(ex. SVM vs. Random Forest ...)可再新增其他.py

**模型表現的含義**

1. K值、C值:

2. 糖尿病預測:

3. 各演算法特性: