

# SVM 支援向量機

NCU MIS 106403551 呂晟維

1. 使用 weka 對 heart.arff 進行SVM分析，過程中對所有重要步驟進行截圖並加以說明，越詳盡好：
  - (a) 請嘗試修改 heart.arff，使其可以使用 SMO function 進行 SVM 分析，並說明原本為何無法使用 SMO (5%)
  - (b) 請嘗試去除有空值的資料 (5%)
  - (c) 用SMO function 對前處理過的 heart.arff 進行 SVM 分析，kernel 設為'linear'，Percentage spilt 設為 66%，截圖並附上過程及準確率 (30%)
2. 使用 python 依照步驟對 heart.csv 進行 SVM 分析，過程中對所有重要程式步驟進行截圖並加以說明，越詳盡越好：
  - (d) 請問資料集是否有空值？有幾筆資料含有空值？如有空值即去掉該筆資料 (5%)
  - (e) 將最後一個屬性值 "target" 切分成 Label，其餘屬型切分為 Feature (5%)
  - (f) 將 Feature 用 sklearn.preprocessing 的 StandardScaler 進行標準化 (10%)
  - (g) 切分資料集與測試集，設 test\_size=0.33，random\_state=1 (10%)
  - (h) 最後，使用 sklearn.svm 裡的 SVC 進行分析，kernel 設為'linear'，並印出模型最終的準確度 (30%)

## Weka部分

**(a) 請嘗試修改 heart.arff，使其可以使用 SMO function 進行 SVM 分析，並說明原本為何無法使用 SMO (5%)**

可以觀察到原本的.arff有空值，也都是Numeric型態的數值資料，Weka無法執行的主要原因是SVM是Binary Classifier，需要將Label也就是 target 欄位轉成Nominal型態。所以將.arff的 target 欄位稍作修改成下方樣式就可以執行了。

```
@attribute target {0, 1}
```

這裡我另外save為 heart\_Nom.arff 。

Weka GUI Screenshot: NumericToNominal filter configuration. The filter is applied to the 'target' attribute (index 14). The 'About' dialog is open, showing the filter's purpose: 'A filter for turning numeric attributes into nominal ones.' The 'attributeIndices' field is set to 14, with a red box around it and the text '只轉換第14個attr' (only convert the 14th attribute). The 'Current relation' shows 14 attributes and 303 instances. The 'Selected attribute' shows 'target' with a numeric type and 2 distinct values.

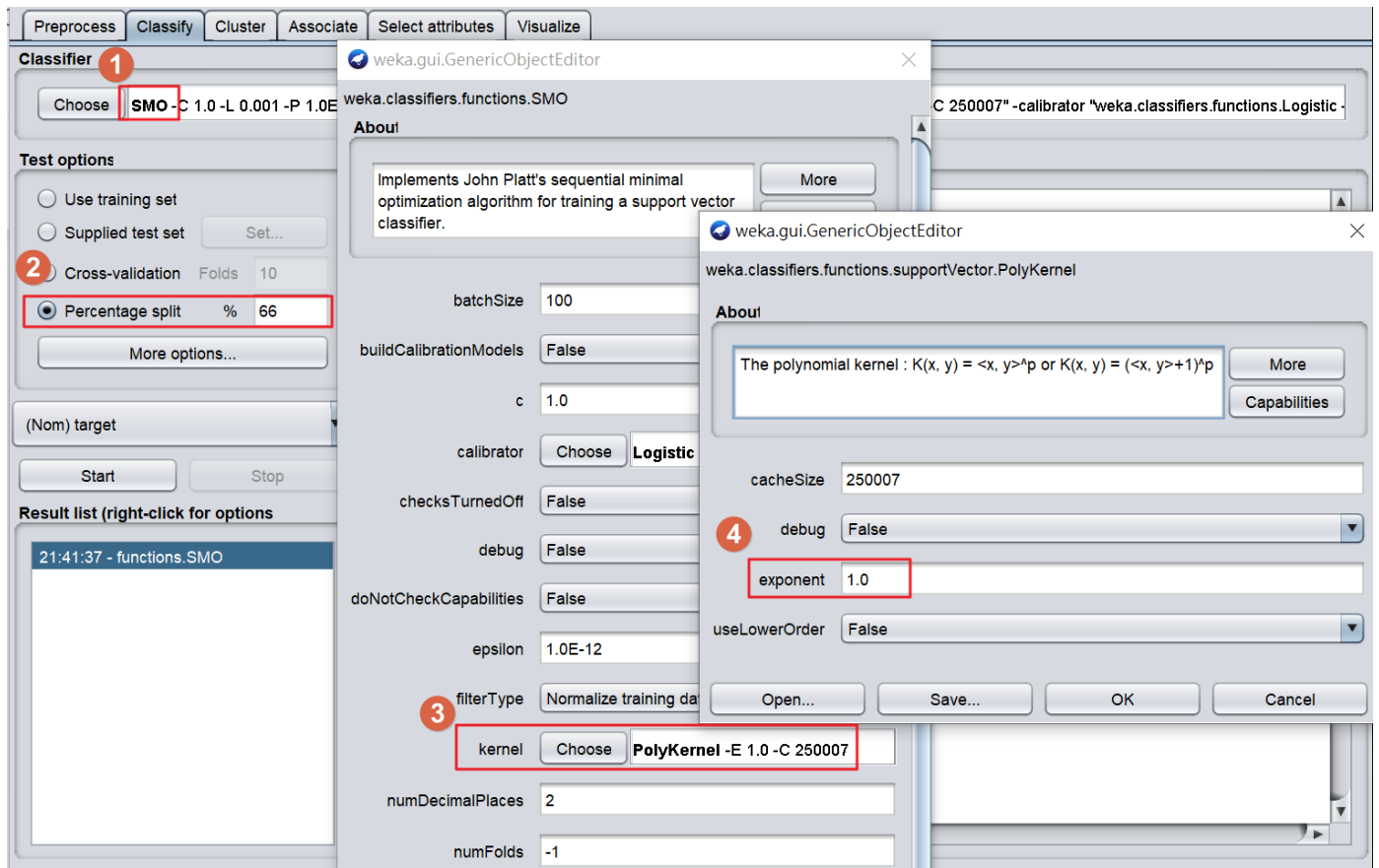
## (b) 請嘗試去除有空值的資料 (5%)

由於原本的.arff有空值，我們使用weka內建的editor來手動刪除有空值的資料列，選取時同時按住 **ctrl** 鍵可以同時選取多筆資料再一次刪除，這裡我另外save為 heart\_rmNull.arff。

Weka GUI Screenshot: RemoveMisclassified filter configuration. The filter is applied to the 'target' attribute (index 14). The 'About' dialog is open, showing the filter's purpose: 'A filter for removing misclassified instances.' The 'attributeIndices' field is set to 14, with a red box around it and the text '只轉換第14個attr' (only convert the 14th attribute). The 'Current relation' shows 14 attributes and 303 instances. The 'Selected attribute' shows 'target' with a nominal type and 2 distinct values.

## (c) 用 SMO function 對前處理過的 heart.arff 進行 SVM 分析，kernel 設為'linear'，Percentage spilt 設為 66%，截圖並附上過程及準確率 (30%)

SVM方法在Weka中稱作SMO，SVM最精華的地方在於Kernel function，我們這次要求做linear的Kernel function，因此選擇多項式的PolyKernal，把次方數設為1表示線性SVM。調整完後按下確定執行就可以囉！



執行結果的 Classifier model 如下，可以看到我們的Kernel是 Linear Kernel:  $K(x,y) = \langle x,y \rangle$ ，代表線性回歸，沒有把資料投射到更高維度的空間中。

```
=== Classifier model (full training set) ===
```

SMO

Kernel used:

Linear Kernel:  $K(x,y) = \langle x,y \rangle$

Classifier for classes: 0, 1

BinarySMO

Machine linear: showing attribute weights, not support vectors.

```

-0.3014 * (normalized) i>age
+ -1.0026 * (normalized) sex
+ 1.734 * (normalized) cp
+ -0.9088 * (normalized) trestbps
+ -0.4322 * (normalized) chol
+ 0.0432 * (normalized) fbs
+ 0.2627 * (normalized) restecg
+ 1.2303 * (normalized) thalach
+ -0.769 * (normalized) exang
+ -1.7558 * (normalized) oldpeak
+ 0.7044 * (normalized) slope
+ -1.6552 * (normalized) ca
+ -1.6355 * (normalized) thal
+ 1.7718

```

Number of kernel evaluations: 13465 (77.441% cached)

然後我們的準確率是86%，如果Kernel選擇更高次方可提升準確率，但怕有overfit的現象。

```
=== Summary ===
```

Correctly Classified Instances	86	86	%
Incorrectly Classified Instances	14	14	%
Kappa statistic	0.72		
Mean absolute error	0.14		
Root mean squared error	0.3742		
Relative absolute error	28	%	
Root relative squared error	74.2256	%	
Total Number of Instances	100		

## Python部分

詳細的程式碼和參考資料等解說還是寫放在 SVM - Heart Attack.ipynb 裡。

(d) 請問資料集是否有空值？有幾筆資料含有空值？如有空值即去掉該筆資料 (5%)

將csv資料讀入成 `df` 後，我們檢查是否有空值並確認有10筆空資料。

```
>>> df.isnull().values.any()
```

```
True
```

```
>>> df.isnull().sum()
```

```
age          0
sex          0
cp           1
trestbps     0
chol         2
fbs          0
restecg      3
thalach      0
exang        1
oldpeak      2
slope        1
ca           0
thal         0
target       0
dtype: int64
```

刪除有空值的資料，資料筆數從303改為293。

```
df.dropna(axis=0, inplace=True)
```

**(e) 將最後一個屬性值 "target" 切分成 Label，其餘屬型切分為 Feature (5%)**

- `label` 變數是target欄位的資料，是資料集的output
- `features` 變數資料集的14欄input

```
features = df.iloc[:, 0:13]
label = df['target']
```

**(f) 將 Feature 用 sklearn.preprocessing 的 StandardScaler 進行標準化 (10%)**

由於 SVM 的資料需要標準化。用 sklearn.preprocessing 的 StandardScaler 進行特徵標準化。

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(features)

StandardScaler(copy=True, with_mean=True, with_std=True)
```

After fitting, Mean and standard deviation are then stored to be used on later data using transform.

```
>>> scaler.transform(features)
```

```
array([[ 0.96008384,  0.68190908,  1.97865831, ..., -2.26047188,
        -0.71658705, -2.12994828],
       [-1.90587175,  0.68190908,  1.00592864, ..., -2.26047188,
        -0.71658705, -0.50051004],
       [-1.46495551, -1.46647115,  0.03319897, ...,  0.9703489 ,
        -0.71658705, -0.50051004],
       ...,
       [ 1.51122914,  0.68190908, -0.93953071, ..., -0.64506149,
        1.23652928,  1.1289282 ],
       [ 0.29870947,  0.68190908, -0.93953071, ..., -0.64506149,
        0.25997112,  1.1289282 ],
       [ 0.29870947, -1.46647115,  0.03319897, ..., -0.64506149,
        0.25997112, -0.50051004]])
```

### (g) 切分資料集與測試集，設 **test\_size=0.33**，**random\_state=1 (10%)**

切分資料集與測試集，設 **test\_size=0.33**, **random\_state=1**

**train\_size**：三種類型。float，int，None。

- float：0.0-1.0之間，代表訓練數據集占總數據集的比例。
- int：代表訓練數據集具體的樣本數量。
- None：設置為**test\_size**的補。
- default：默認為None。

**random\_state**：三種類型。int，randomstate instance，None。

- int：是隨機數生成器的種子。每次分配的數據相同。
- randomstate：**random\_state**是隨機數生成器的種子。（這裡沒太理解）
- None：隨機數生成器是使用了np.random的**randomstate**。
- 種子相同，產生的隨機數就相同。種子不同，即使是不同的實例，產生的種子也不相同。

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split

>>> X_train, X_test, y_train, y_test = train_test_split(features, label, test_size=0.33,
random_state=1)
```

### (h) 最後，使用 **sklearn.svm** 裡的 **SVC** 進行分析，**kernel** 設為'**linear**'，並印出模型最終的準確度 (**30%**)

```
>>> from sklearn import svm

>>> linear_svc = svm.SVC(kernel='linear').fit(X_train, y_train)
>>> linear_svc

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

### Accurecy in Training set

```
>>> linear_svc.score(X_train, y_train)

0.8367346938775511
```

### Accurecy in Testing set

```
>>> linear_svc.score(X_test, y_test)

0.865979381443299
```

在繳交程式碼檔案裡我還有視覺化confusion matrix 。  $d(d' \quad \forall')$  ).