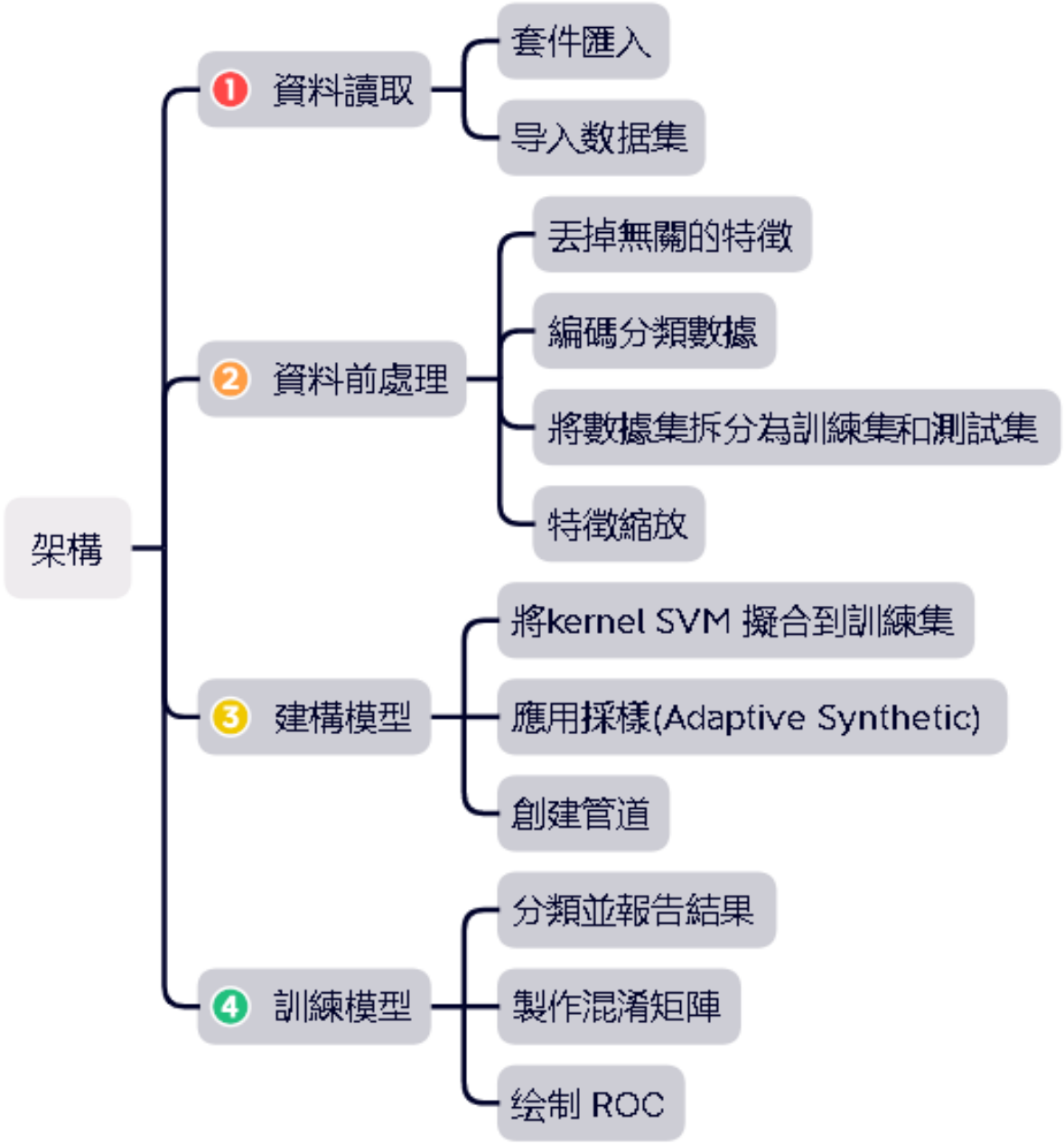
AML detection/反洗錢檢測

動機:
瞭解號稱萬能分類器的SVM
試著了解如何處理數據不平衡的問題

程式工具介紹
程式意義與邏輯
模型理論

資管所 / M1144014 紀勻翔

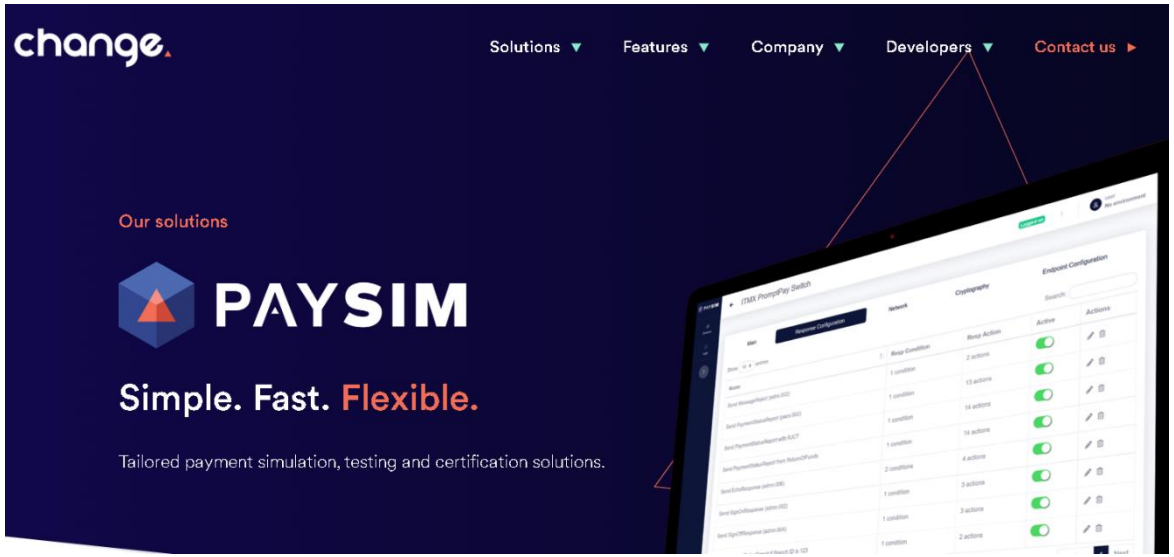
專案架構


架構
1 資料讀取
套件匯入
导入数据集
2 資料前處理
丟掉無關的特徵
編碼分類數據
將數據集拆分為訓練集和測試集
特徵縮放
3 建構模型
將kernel SVM 擬合到訓練集
應用採樣(Adaptive Synthetic)
創建管道
4 訓練模型
分類並報告結果
製作混淆矩陣
绘制 ROC

資料讀取

導入數據集

資料源於 paysim 電子錢包平臺的交易數據，該平臺透過移動應用程式和visa 卡向沒有銀行帳戶與銀行餘額不足的人提供零售銀行服務，因此該資料集屬於該平臺內的網路電子支付相關數據。



資料集來源:
https://www.kaggle.com/code/x09072993/aml-detection/data

網址連結:
https://changefinancial.com/paysim/

資料讀取

數據型態解析

```
1  dataset.info()
2
```

✓ 0.3s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

資料讀取

數據集特徵

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig
3390344	255	CASH_OUT	182402.89	C1020081092	203481.0	21078.11

nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
C2094964347	109879.04	292281.93	0	0

Step	映射現實世界中的時間單位。在這種情況下，1 step是 1 小時的時間。
Type	現金進賬、現金出賬、借記、付款和轉帳
Amount	以當地貨幣計算的交易金額
nameOrig	開始交易的客戶
oldbalanceOrg	交易前初始餘額
newbalanceOrig	交易後客戶的餘額。
nameDest	交易的收件人 ID。
oldbalanceDest	交易前的初始收款人餘額。
newbalanceDest	交易後收款人的餘額。
isFraud	識別=>{欺詐交易 (1) 和非欺詐交易 (0)}

資料前處理

1.丟掉無關的特徵

2.抽樣

3.編碼分類數據

4.將數據集拆分為訓練集和測試集

5.特徵縮放

資料前處理

1.丟掉無關的特徵

```
# 丟掉無關的特徵
dataset.drop('nameOrig', axis=1, inplace=True)
dataset.drop('nameDest', axis=1, inplace=True)
dataset.drop('isFlaggedFraud', axis=1, inplace=True)
dataset.head(1)
# dataset.shape()
```

資料集變數名稱 .drop()

labels	一個字元或者數值，加上axis ，表示帶label標識的行或者列；如 (labels='A', axis=1) 表示A列
axis(軸)	默認axis=0 axis=1 表示行，axis=1 表示列
inplace	默認False True: 表示直接在數據上操作刪除動作 False: 不改變原始數據，而是返回一個執行刪除動作後的數據集

資料來源
https://blog.csdn.net/W_weiying/article/details/84626260
https://blog.csdn.net/mahoon411/article/details/114777623

資料前處理

2.抽樣與轉換資料型態(object => array)

```
#
sample_dataframe = dataset.sample(n=100000)
X = sample_dataframe.iloc[:, :-1].values
y = sample_dataframe.iloc[:, 7].values

print(sample_dataframe.isFraud.value_counts())
```

✓ 1.6s

```
0    99857
1      143
Name: isFraud, dtype: int64
```

Sample(n=num)	這是一個可選參數, 由整數值組成, 並定義生成的隨機行數(隨機抽樣的數量)
.iloc[]	結構 => .iloc[行,列]
	只能使用整數索引，不能使用標籤索引，通過整數索引切片 選擇資料時，前閉後開(不包含邊界結束值)。索引都是從 0 開始。
.values	
.value_counts()	類別_i 的數量

```
dataset.info()

```

✓ 0.3s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
```

資料來源
https://blog.csdn.net/W_weiying/article/details/84626260
https://blog.csdn.net/mahoon411/article/details/114777623

資料前處理

2.抽樣與轉換資料型態(object => array)

```
#
sample_dataframe = dataset.sample(n=100000)
X = sample_dataframe.iloc[:, :-1].values
y = sample_dataframe.iloc[:, 7].values

print(sample_dataframe.isFraud.value_counts())
```

✓ 1.6s

```
0    99857
1      143
Name: isFraud, dtype: int64
```

```
dataset.info()
```

✓ 0.2s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 8 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   oldbalanceOrg   float64
 4   newbalanceOrig  float64
 5   oldbalanceDest  float64
 6   newbalanceDest  float64
 7   isFraud         int64
dtypes: float64(5), int64(2), object(1)
memory usage: 388.3+ MB
```

資料來源
https://blog.csdn.net/W_weiying/article/details/84626260
https://blog.csdn.net/mahoon411/article/details/114777623

資料前處理

3.編碼分類數據

```python
# 編碼分類數據
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
import scipy.sparse
labelencoder = LabelEncoder()
X[:, 1] = labelencoder.fit_transform(X[:, 1])
onehotencoder = ColumnTransformer(
    [('encoder', OneHotEncoder(), [1])])
X = onehotencoder.fit_transform(X).toarray()

# 避免虛擬變量陷阱(Dummy Variable Trap)
# https: // www.cnblogs.com/HuZihu/p/11330853.html  # :~:text=%E
X = X[:, 1:]
```

LabelEncoder()
即將離散型的資料轉換成 0 到 n – 1 之間的數，這裡 n是一個列表的不同取值的個數，可以認為是某個特徵的所有不同取值的個數。也就是用來對分類型特徵值進行編碼，即對不連續的數值或文本進行編碼。
OneHotEncoder()
有一些特徵並不是以連續值的形式給出。例如：人的性別 [“male”, “female”]，來自的國家 [“from Europe”, “from US”, “from Asia”]，使用的流覽器 [“uses Firefox”, “uses Chrome”, “uses Safari”, “uses Internet Explorer”]。這種特徵可以採用整數的形式進行編碼

資料來源
https://blog.csdn.net/quintind/article/details/79850455
https://blog.csdn.net/lw_power/article/details/82981122

資料前處理

3.編碼分類數據

```
# 編碼分類數據
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
import scipy.sparse
labelencoder = LabelEncoder()
X[:, 1] = labelencoder.fit_transform(X[:, 1])
onehotencoder = ColumnTransformer(
    [('encoder', OneHotEncoder(), [1])])
X = onehotencoder.fit_transform(X).toarray()

# 避免虛擬變量陷阱(Dummy Variable Trap)
# https: // www.cnblogs.com/HuZihu/p/11330853.html  # :~:text=%E
X = X[:, 1:]
```

ColumnTransformer() :可以選擇地進行資料轉換。例如，它允許將特定的轉換或轉換序列僅應用於數字列，而將單獨的轉換序列僅應用於類別列

要使用ColumnTransformer()，必須指定一個轉換器列表。每個轉換器是一個三元素結構，用於定義轉換器的名稱，要應用的轉換以及要應用於其的列索引，例如：（名稱，對象，列）。

資料來源
https://blog.csdn.net/qq_43201403/article/details/109552348

資料前處理

3.編碼分類數據

```python
# 編碼分類數據
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
import scipy.sparse
labelencoder = LabelEncoder()
X[:, 1] = labelencoder.fit_transform(X[:, 1])
onehotencoder = ColumnTransformer(
    [('encoder', OneHotEncoder(), [1])])
X = onehotencoder.fit_transform(X).toarray()

# 避免虛擬變量陷阱(Dummy Variable Trap)
# https: // www.cnblogs.com/HuZihu/p/11330853.html  # :~:text=%E
X = X[:, 1:]
```

fit_transform(y)
相當於先進行fit()再進行transform()，即把y塞到字典(dict)中去以後再進行transform得到索引值。
fit(y)
fit可看做一本空字典，y可看作要塞到字典中的詞。
transform(y)
將y轉變成索引值。
toarray()
將List轉為陣列的一個非常方便的方法

資料來源
https://blog.csdn.net/qq_43201403/article/details/109552348
https://www.yisu.com/zixun/581802.html
https://blog.csdn.net/weixin_48135624/article/details/114491856

資料前處理

4.將數據集拆分為訓練集和測試集

```python
# 將數據集拆分為訓練集和測試集

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5, random_state=1)

counts = np.unique(y_train, return_counts=True)
```

✓ 0.1s

.train_test_split()	隨機劃分訓練集和測試集
參數解釋	
train_data	所要劃分的樣本特徵集
train_target	所要劃分的樣本結果
test_size	樣本占比，如果是整數的話就是樣本的數量
random_state	是亂數的種子。

資料來源
https://blog.csdn.net/mrxjh/article/details/78481578

資料前處理

4.將數據集拆分為訓練集和測試集

```
# 將數據集拆分為訓練集和測試集

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5, random_state=1)

counts = np.unique(y_train, return_counts=True)
counts
```

✓ 0.8s

```
(array([0    1]    dtype=int64)    array([69900    100]    dtype=int64))
```

No results.

.unique(輸入陣列 , return_counts)	
y_train	可轉換為陣列的陣列或物件
return_counts	原始數據集內的元素_i 出現的總次數

資料來源
https://www.delftstack.com/zh-tw/api/numpy/python-numpy-unique/

資料前處理

5.特徵縮放

```
# 特徵縮放
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_val = sc.transform(X_val)
X_test = sc.transform(X_test)
print(counts)
```

✓ 0.1s

```
(array([0, 1], dtype=int64), array([69904,    96], dtype=int64))
```

StandardScaler()
標準縮放，去均值和方差歸一化。且是針對每一個特徵維度來做的，而不是針對樣本。 $$z = \frac{x - \mu}{\sigma}$$ 其中 $\sigma \neq 0$。
Transform()
將X_test轉變成索引值。

資料來源
https://blog.csdn.net/wzyaiwl/article/details/90549391/
https://zh.wikipedia.org/zh-tw/%E6%A8%99%E6%BA%96%E5%88%86%E6%95%B8

建構模型

資料讀取 — 資料前處理 — 建構模型 — 訓練模型

套件匯入

```
# 將kernel SVM 擬合到訓練集
from sklearn.svm import LinearSVC
from imblearn.under_sampling import NearMiss
from imblearn import over_sampling as os
from imblearn.pipeline import make_pipeline
from imblearn.combine import SMOTETomek
from imblearn.over_sampling import ADASYN
from imblearn.under_sampling import ClusterCentroids
from imblearn.over_sampling import RandomOverSampler
```

LinearSVC	線性SVM 模組
Imblearn	處理不均衡資料 NearMiss ClusterCentroids RandomOverSampler SMOTETomek() ADASYN()
os	os模組提供的就是各種 Python 程式與作業系統進行交互的介面。
Make_pipeline	設定工作流，組合成較複雜的工作

資料來源
https://blog.csdn.net/qq_41185868/article/details/107204245

建構模型

1. 什麼是linearSVC？

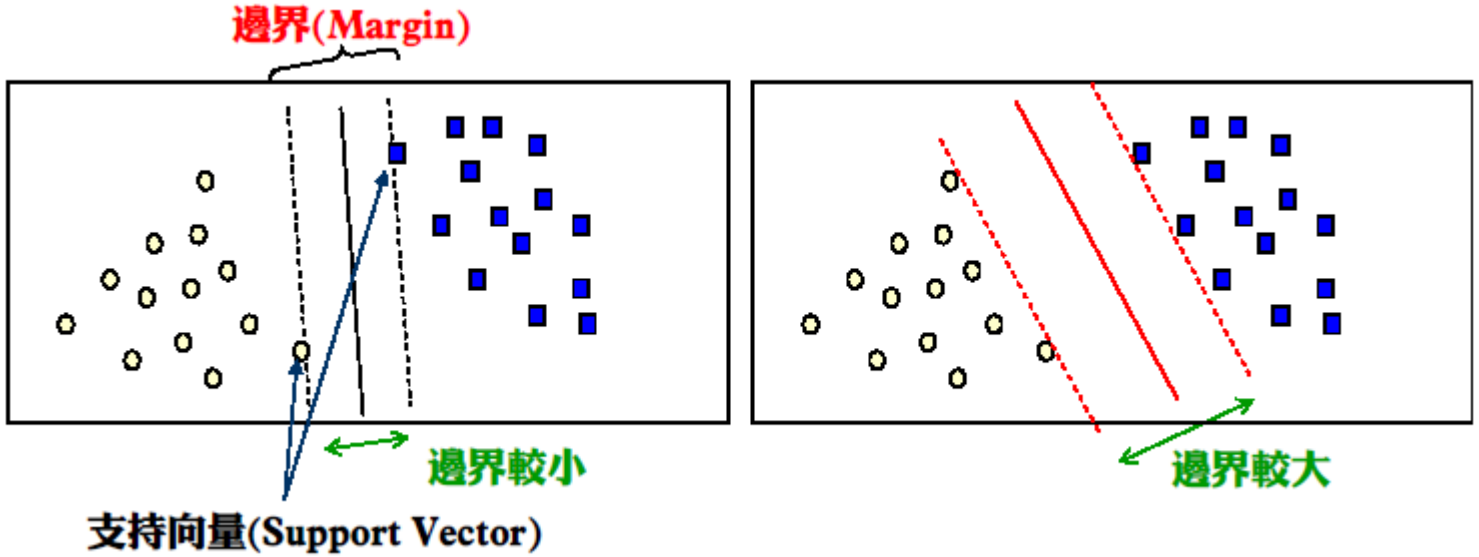
2. 什麼是ADASYN？

建構模型

SVM 模型

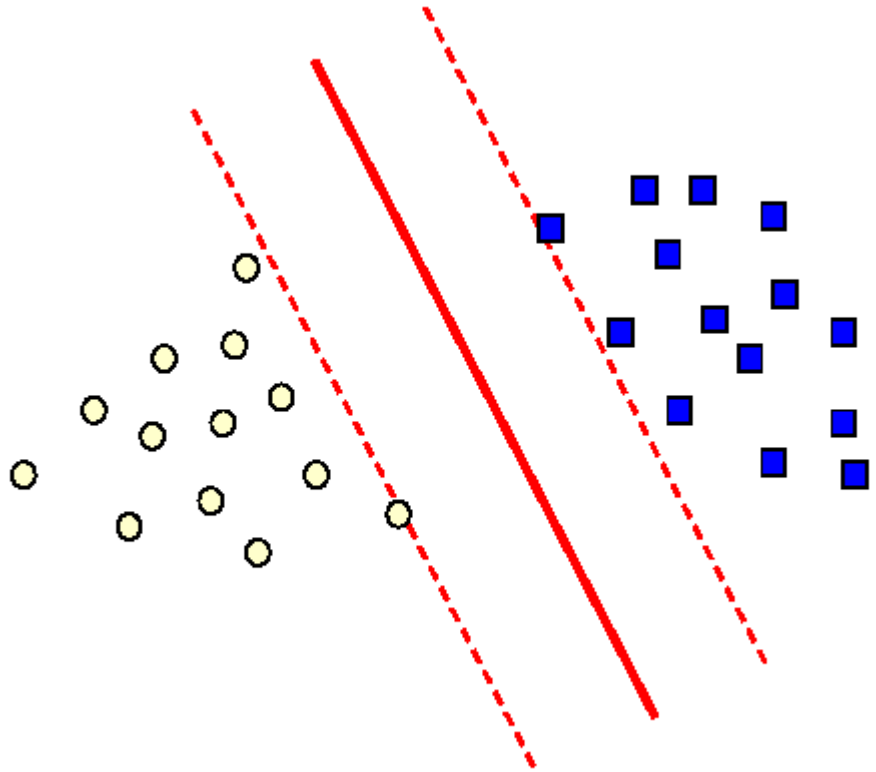
支持向量機(SVM: Support Vector Machine)是一種可用來做分類或迴歸的方法。給予一群已經分類好的資料， SVM可以經由訓練(Training)獲得一組模型。爾後若有尚未分類的資料，支持向量機可以利用先前訓練好的模 型去預測(Predict)這筆資料所屬的類別。

因為支持向量機在建立模型時，必須要先有已經分類好的資料作為訓練用，所以支持向量機是監督式學習(Supervised Learning)的方法之一。

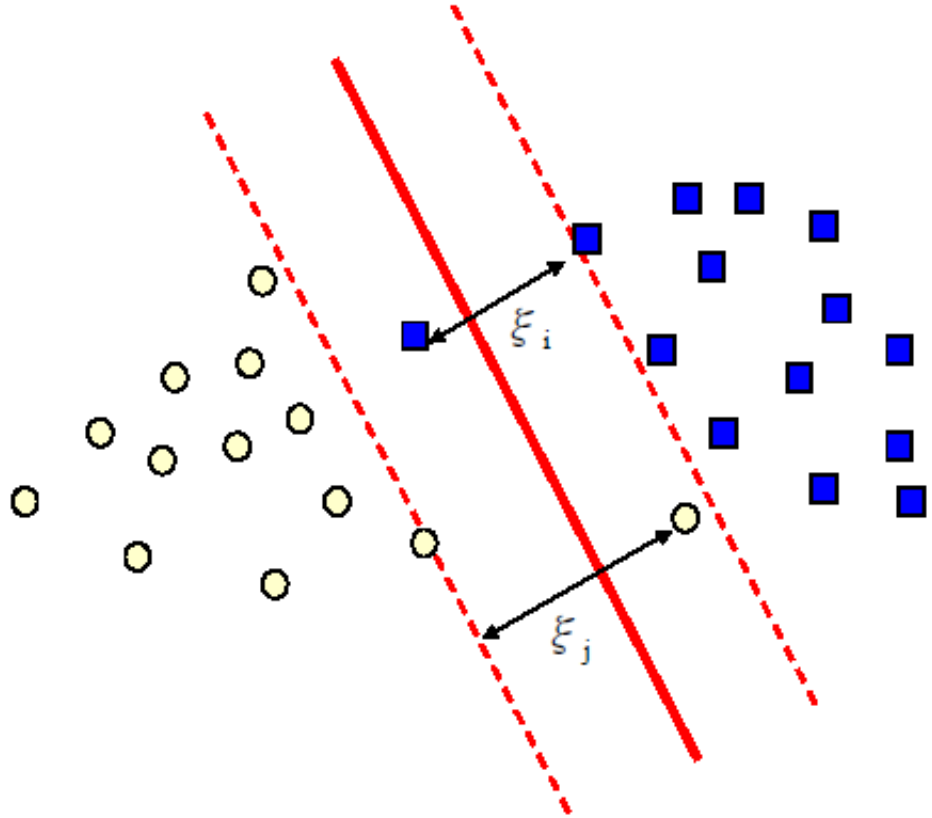


建構模型

線性可分SVM 模型



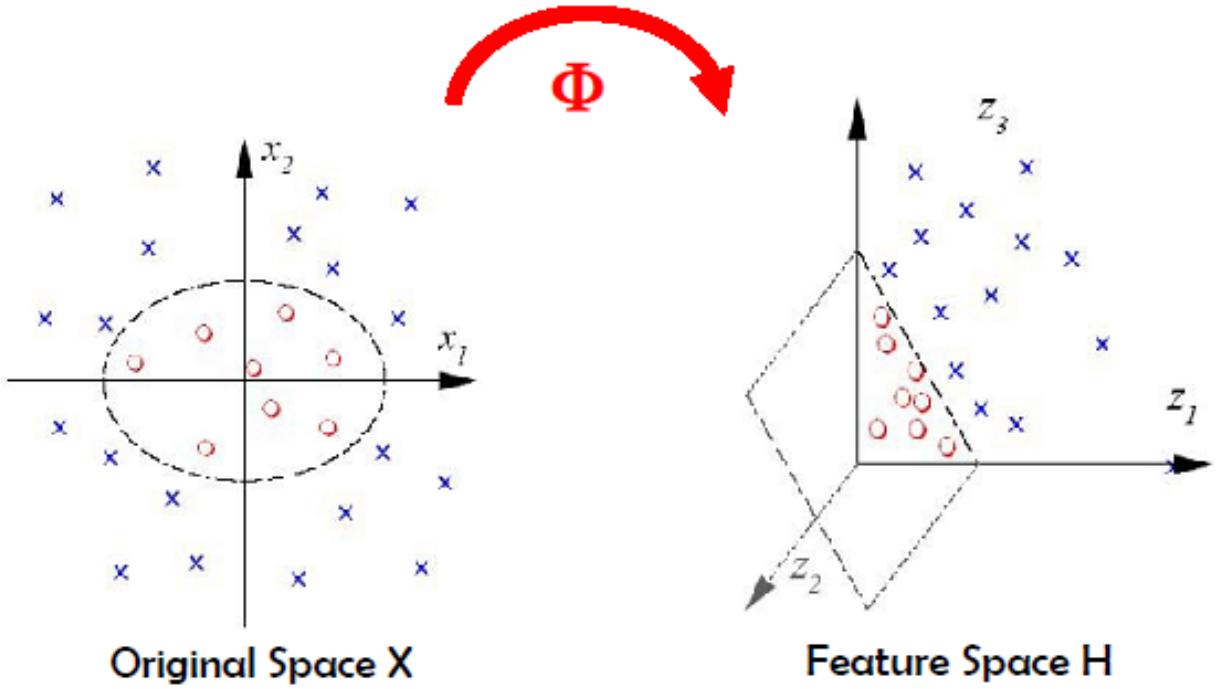
線性不可分SVM 模型



建構模型

非線性SVM 模型

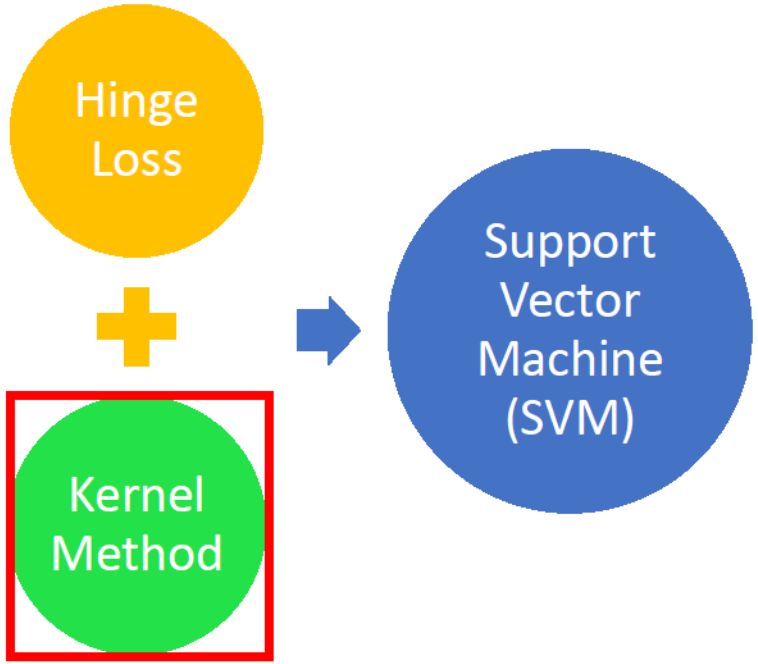
透過映射函數 => 透過kernel function



建構模型

LinearSVC(線性支援向量分類)模型理論

線性支持向量分類，類似於SVC，但是其使用的核函數是”linear“上邊介紹的兩種是按照brf（徑向基函數計算的，其實現也不是基於LIBSVM，所以它具有更大的靈活性在選擇處罰和損失函數時，而且可以適應更大的資料集，他支持密集和稀疏的輸入是通過一對一的方式解決的



Minimizing loss function L:

$$L(f) = \sum_n \varepsilon^n + \lambda \|w\|_2$$

$$\varepsilon^n = max\left(0, 1 - \hat{y}^n f(x)\right)$$

ε^n: slack variable

Quadradic programming problem

=

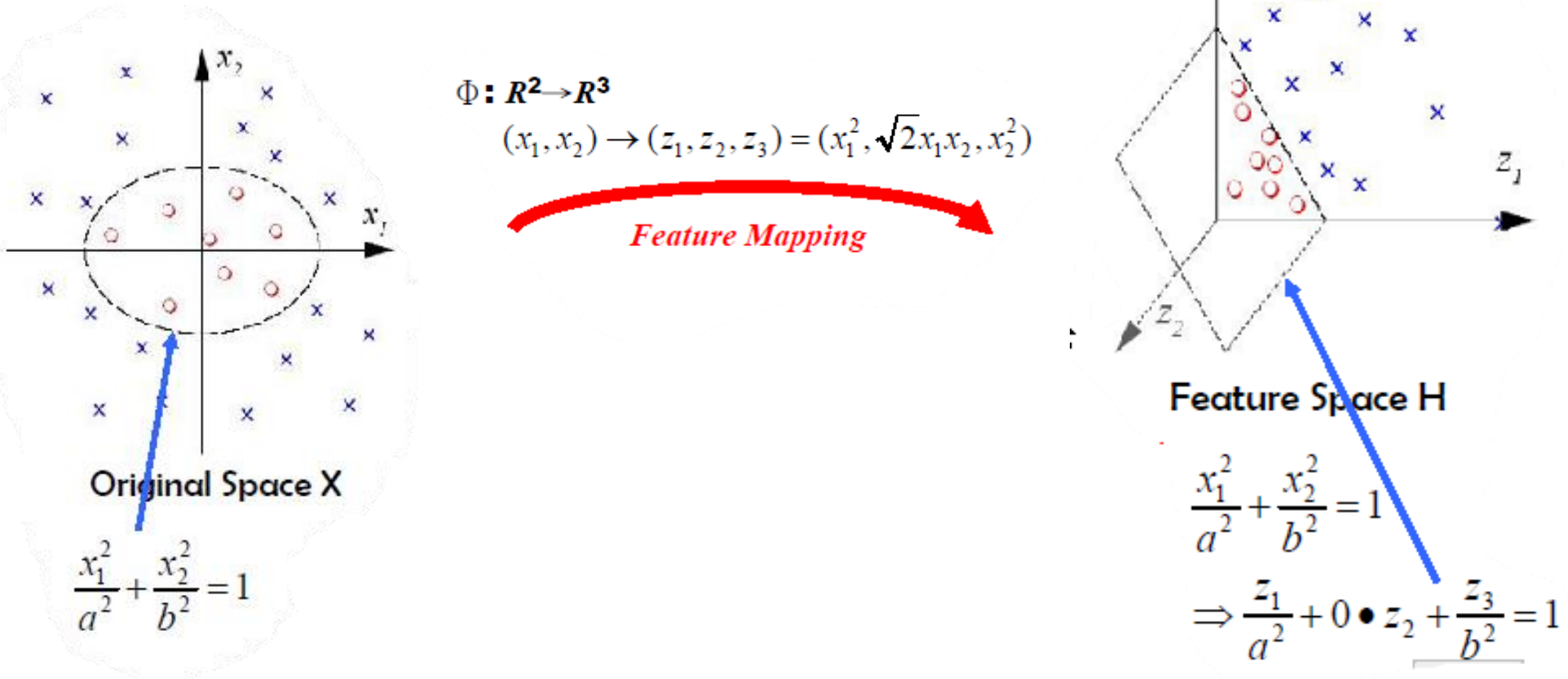
$$\varepsilon^n \geq 0$$

$$\varepsilon^n \geq 1 - \hat{y}^n f(x) \Rightarrow \hat{y}^n f(x) \geq 1 - \varepsilon^n$$

建構模型

LinearSVC(線性支援向量分類)模型理論

核方法(kernel method)概念



建構模型

LinearSVC(線性支援向量分類)模型理論

核方法(kernel method)概念

在原始空間 X 不好區分的資料可以用非線性的映射函數將這些資料轉換到另一個空間 H，可能可以找到一條線做分割超平面 (Hyperplane)

建構模型

LinearSVC(線性支援向量分類)模型理論

核方法(kernel method)概念

缺點: 但是映像函數需要對該問題有清楚的了解，才有機會找出一條符合邏輯的映象函數

- 因此我們可以透過核函數(kernel function) 去找到分割超平面。

建構模型

LinearSVC(線性支援向量分類)模型理論

核函數(kernel function)

Kernel function 就可以知道特徵空間終點跟點的內積、距離、角度

$$\langle \Phi(x_1,x_2),\Phi(x'_1,x'_2)\rangle = \langle (z_1,z_2,z_3),(z'_1,z'_2,z'_3)\rangle = \langle (x_1^2,\sqrt{2}x_1x_2,x_2^2),(x'^2_1,\sqrt{2}x'_1x'_2,x'^2_2)\rangle$$

$$= x_1^2x'^2_1+2x_1x_2x'_1x'_2+x_2^2x'^2_2 = (x_1x'_1+x_2x'_2)^2 = (\langle X,X'\rangle)^2 = k(X,X') \text{ (內積)}$$

$$\|\Phi(X)-\Phi(X')\|^2 = \|\Phi(X)\|^2+\|\Phi(X')\|^2-2<\Phi(X),\Phi(X')> = \Phi(X)^T\Phi(X)+\Phi(X')^T\Phi(X')-2\Phi(X)^T\Phi(X')$$

$$=<\Phi(X),\Phi(X)>+<\Phi(X'),\Phi(X')>-2<\Phi(X),\Phi(X')>$$

$$=k(X,X)+k(X',X')-2k(X,X') \quad \text{(距離)}$$

$$\langle \Phi(X),\Phi(X')\rangle = \|\Phi(X)\|\times\|\Phi(X')\|\cos\theta$$

$$\Rightarrow \cos\theta = \frac{\langle \Phi(X),\Phi(X')\rangle}{\|\Phi(X)\|\times\|\Phi(X')\|} = \frac{\langle \Phi(X),\Phi(X')\rangle}{\sqrt{<\Phi(X),\Phi(X)>}\times\sqrt{<\Phi(X'),\Phi(X')>}}$$

$$= \frac{k(X,X')}{\sqrt{k(X,X)}\times\sqrt{k(X',X')}} \quad \text{(角度)}$$

建構模型

LinearSVC(線性支援向量分類)模型理論

核函數(kernel function)

常用的核函數

$$k(X_i, X_j) = e^{[-\|X_i - X_j\|^2/(2\sigma^2)]}$$

$$k(X_i, X_j) = (<X_i, X_j>+1)^p \ , P \in Z^+$$

$$k(X_i, X_j) = <X_i, X_j>$$

建構模型

2. 什麼是ADASYN？

```
(array([0, 1], dtype=int64), array([69904,    96], dtype=int64))
```

當前問題:

目前的數據不平衡，在現實世界中，採集的數據往往是比例失衡的。

欺詐性交易的數量要遠低於正常和健康的交易，也就是說，它只占到了總觀測量的大約 0.1%。這裡的問題是提高識別罕見的少數類別的準確率，而不是實現更高的總體準確率。因此我們需要針對該情況作抽樣上的調整

```python
# 應用採樣
# Adaptive Synthetic
ada = ADASYN()
X_resampled, y_resampled = ada.fit_resample(X_train, y_train)
count = np.unique(y_resampled, return_counts=True)
```

建構模型


```
# 應用採樣
# Adaptive Synthetic
ada = ADASYN()
X_resampled, y_resampled = ada.fit_resample(X_train, y_train)
count = np.unique(y_resampled, return_counts=True)
```

處理不平衡學習的其中一種方式 =>過採樣 (Onver-Sampling)

建構模型


```python
# 應用採樣
# Adaptive Synthetic
ada = ADASYN()
X_resampled, y_resampled = ada.fit_resample(X_train, y_train)
count = np.unique(y_resampled, return_counts=True)
```

處理不平衡學習的其中一種方式 =>過採樣 (Onver-Sampling)

*SMOTE(*Synthetic Minority Oversampling Technique)

Adaptive Synthetic(ADASYN)

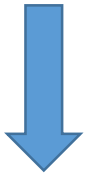
建構模型

插值演算法ADASYN

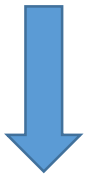
基於k臨近和插值演算法。不同的是ADASYN演算法在考慮K個臨近樣本點的時候是包括所有類別的不僅僅是 Xi 同類。並且根據每個少數類樣本周圍異類樣本點的個數賦予權重 ri ,權重越高，一會根據同類臨近點生成的樣本數也就越多。

建構模型


```
(array([0, 1], dtype=int64), array([69904,    96], dtype=int64))
```



```
# 應用採樣
# Adaptive Synthetic
ada = ADASYN()
X_resampled, y_resampled = ada.fit_resample(X_train, y_train)
count = np.unique(y_resampled, return_counts=True)
```



```
(array([0, 1], dtype=int64), array([69904, 69888], dtype=int64))
```

建構模型


```
# 創建管道

pipeline4 = make_pipeline(ADASYN(),LinearSVC(random_state=1))
pipeline4.fit(X_train, y_train)
print(count)
```

✓ 35.1s

```
(array([0, 1], dtype=int64), array([69904, 69888], dtype=int64))
```

make_pipeline()	建立管道
LinearSVC(*random_state*=1)	*random_state*=1 在随机数据混洗时使用的伪随机数生成器的种子。 如果是int，则random_state是随机数生成器使用的种子; 如果是RandomState实例，则random_state是随机数生成器; 如果为None，则随机数生成器是np.random使用的RandomState实例。
Fit()	用訓練數據擬合分類器模型，**擬合**就是把平面上一系列的點，用一條光滑的曲線連接起來。

訓練模型結果


```python
# 分類並報告結果
from imblearn.metrics import classification_report_imbalanced
print(classification_report_imbalanced(y_test, pipeline4.predict(X_test)))
```

	pre	rec	spe	f1	geo	iba	sup
0	1.00	0.57	1.00	0.73	0.76	0.55	14983
1	0.00	1.00	0.57	0.01	0.76	0.60	17
avg / total	1.00	0.57	1.00	0.73	0.76	0.55	15000

訓練模型結果

最後，我們該如何去判斷一個分類模型好不好?

```
# 製作混淆矩陣
from sklearn.metrics import confusion_matrix, precision_score,auc,roc_auc_score,roc_curve,recall_score
cm = confusion_matrix(y_val, pipeline4.predict(X_val))
roc = roc_auc_score(y_val, pipeline4.predict(X_val))
fpr, tpr, thresholds = roc_curve(y_val, pipeline4.predict(X_val))
roc_auc = auc(fpr,tpr)
```

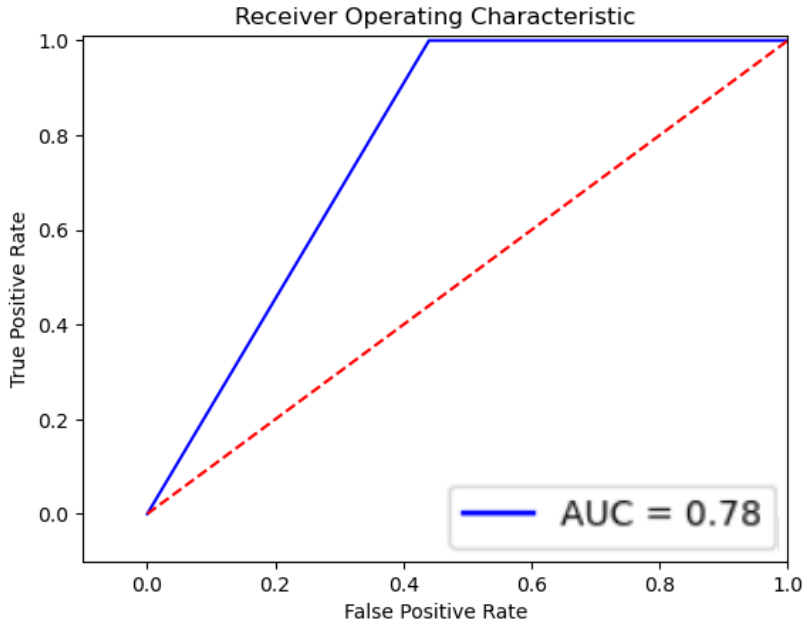
製作混淆矩陣(confusion matrix)

組成混淆矩陣的四個元素(TP,TN,FP,FN)

TP(True Positive)	正確預測成功的正樣本
TN(True Negative)	正確預測成功的負樣本
FP(False Positive)	錯誤預測成正樣本，實際上為負樣本(第一型錯誤(Type 1 Error)
FN(False Negative)	錯誤預測成負樣本(或者說沒能預測出來的正樣本) (Type 2 Error)

訓練模型結果


```python
# 绘制 ROC
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b',label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```



ROC(Receiver operator characteristic）X軸為假陽率，Y軸為真陽率

AUC 即ROC曲線之下所覆蓋的面積除以總面積的比率
二分類的分配差異越顯著，AUC的分數就越高

延伸閱讀

SVC（C-Support Vector Classification）：支持向量分類，基於libsvm實現的（libsvm詳情參考 或者百科），資料擬合的時間複雜度是資料樣本的二次方，這使得他很難擴展到10000個資料集，當輸入是多類別時（SVM最初是處理二分類問題的），通過一對一的方案解決，當然也有別的解決辦法，比如說（以下為引用）：

資料來源
https://blog.csdn.net/weixin_43746433/article/details/97808078

sklearn svm.LinearSVC的参数说明

https://blog.csdn.net/ustbclearwang/article/details/81236732

延伸閱讀

SMOTE原理

SMOTE的全称是Synthetic Minority Over-Sampling Technique 即“人工少数类过采样法”，非直接对少数类进行重采样，而是设计算法来人工合成一些新的少数样本。

資料來源

https://blog.csdn.net/weixin_43746433/article/details/97808078

GitHub QRCode(https://github.com/cgit6/AML_SVM.git)