

randomforest

2018 年 7 月 5 日



1 隨機森林 + Kaggle 初參賽

1.1 目標

我們使用 Kaggle 的鐵達尼號資料集來教你一個重量級的演算法，隨機森林

隨機森林是一個非常方便的演算法，可以應用在很多現實生活的問題並且得到還不錯的結果

我們順便在資料科學的大本營 (Kaggle)，參加我們第一個練習賽

當然，雖然只是學習一個簡單的演算法並且參加一個練習賽，我們還是希望能在這個比賽上面拿到一個不錯的名次

1.2 隨機森林

徹底地貫徹『三個臭皮匠，勝過一個諸葛亮』的概念，

我們蒐集很多不用到最佳的樹，讓他們用多數決來決定答案是哪個分類

但這裡有一點要特別強調

上面那句話是對的，但要加上一點特別的聲明，要是三個擁有不同經歷的臭皮匠

如果三個臭皮匠根本長得一模一樣，無論如何都不會勝過一個諸葛亮的

所以如果我們每個決策樹都長得一模一樣，我們的森林永遠建造不起來

那如何讓他不一樣呢？

很簡單，就每個決策樹在創建的時候不使用全部的訓練資料，只使用其中的一部份就好

這招我們叫 **bagging**，在不使用全部資料的情況下，每個建造出來的樹自然就會不一樣

還有另外一種建造組合的方式叫作 **boosting**, 我們剛剛的 **bagging** 是並行的建造很多樹, 但 **boosting** 是把每一次的結果錯的地方修正 **boosting** 和 **bagging** 可以參考 <https://www.jianshu.com/p/708dff71df3a>

隨機森林雖然簡單, 但在許多的分類問題都可以得到非常非常好的效果

1.3 資料集位置

<https://www.kaggle.com/c/titanic/data>

1. 需要登入才能下載
2. train.csv: 訓練的資料集
3. test.csv: 測試的資料集

1.4 資料處理

1.4.1 Step 0. 讀入鐵達尼訓練集

```
In [1]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
%matplotlib inline

%matplotlib inline

# 為了顯示的漂亮, 我刻意的把印出來的 row 只顯示 15 個和 column 只顯示十個
# 大家練習的時候可以去掉下面兩行
pd.set_option('display.max_rows', 15)

In [2]: df = pd.read_csv("train.csv")
df
```

```
Out[2]:
```

	PassengerId	Survived	Pclass	\
0	1	0	3	
1	2	1	1	
2	3	1	3	
3	4	1	1	
4	5	0	3	

5	6	0	3
6	7	0	1
..
884	885	0	3
885	886	0	3
886	887	0	2
887	888	1	1
888	889	0	3
889	890	1	1
890	891	0	3

	Name	Sex	Age	SibSp	\
0	Braund, Mr. Owen Harris	male	22.0	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	
2	Heikkinen, Miss. Laina	female	26.0	0	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	
4	Allen, Mr. William Henry	male	35.0	0	
5	Moran, Mr. James	male	NaN	0	
6	McCarthy, Mr. Timothy J	male	54.0	0	
..	
884	Sutehall, Mr. Henry Jr	male	25.0	0	
885	Rice, Mrs. William (Margaret Norton)	female	39.0	0	
886	Montvila, Rev. Juozas	male	27.0	0	
887	Graham, Miss. Margaret Edith	female	19.0	0	
888	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	
889	Behr, Mr. Karl Howell	male	26.0	0	
890	Dooley, Mr. Patrick	male	32.0	0	

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S
5	0	330877	8.4583	NaN	Q
6	0	17463	51.8625	E46	S
..

884	0	SOTON/OQ	392076	7.0500	NaN	S
885	5		382652	29.1250	NaN	Q
886	0		211536	13.0000	NaN	S
887	0		112053	30.0000	B42	S
888	2	W./C.	6607	23.4500	NaN	S
889	0		111369	30.0000	C148	C
890	0		370376	7.7500	NaN	Q

[891 rows x 12 columns]

1.4.2 Step 1. 遺漏值的處理

在還沒建立模型，處理現實資料的時候，我們通常會端上的第一道處理是遺失值的處理
因為遺失值不可避免會對我們建立模型造成一定的影響

我們先檢查一下到底有哪些欄位是有缺失值，而且缺失幾個

如果缺失太多，我們會選擇直接放棄這個欄位

如果缺失只有少數，我們會在不影響資料整體的前提下，補上遺漏值

In [3]: df.isnull()

```
Out[3]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	\
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	
5	False	False	False	False	False	True	False	False	False	
6	False	False	False	False	False	False	False	False	False	
..	
884	False	False	False	False	False	False	False	False	False	
885	False	False	False	False	False	False	False	False	False	
886	False	False	False	False	False	False	False	False	False	
887	False	False	False	False	False	False	False	False	False	
888	False	False	False	False	False	True	False	False	False	
889	False	False	False	False	False	False	False	False	False	
890	False	False	False	False	False	False	False	False	False	

Fare Cabin Embarked

```

0    False    True    False
1    False    False   False
2    False    True    False
3    False    False   False
4    False    True    False
5    False    True    False
6    False    False   False
..      ...      ...      ...
884   False    True    False
885   False    True    False
886   False    True    False
887   False    False   False
888   False    True    False
889   False    False   False
890   False    True    False

```

```
[891 rows x 12 columns]
```

In [4]: # *sum* 會針對上面得到的結果加出一個答案

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

```

Out[4]: PassengerId      0
        Survived        0
        Pclass          0
        Name            0
        Sex             0
        Age            177
        SibSp           0
        Parch           0
        Ticket          0
        Fare            0
        Cabin          687
        Embarked        2
        dtype: int64

```

1.1 連續數值的缺失處理 如果選擇補上的話

這裡在補上缺失值的時候，通常大家會有兩種不同的做法

做法 1: 補上 *mean*，平均值

做法 2: 補上 median, 中位數值

我個人比較不喜歡補平均值, 因為平均值會被極高或者極小值影響, 所以我會補上中位數值

你可以直接對整個 df 使用 df.fillna(數字), 就會幫你把所有數字型態的欄位直接補上那個數字

```
In [5]: df = df.fillna(df.median())
```

```
df
```

```
Out [5]:
```

	PassengerId	Survived	Pclass	\
0	1	0	3	
1	2	1	1	
2	3	1	3	
3	4	1	1	
4	5	0	3	
5	6	0	3	
6	7	0	1	
..	
884	885	0	3	
885	886	0	3	
886	887	0	2	
887	888	1	1	
888	889	0	3	
889	890	1	1	
890	891	0	3	

	Name	Sex	Age	SibSp	\
0	Braund, Mr. Owen Harris	male	22.0	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	
2	Heikkinen, Miss. Laina	female	26.0	0	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	
4	Allen, Mr. William Henry	male	35.0	0	
5	Moran, Mr. James	male	28.0	0	
6	McCarthy, Mr. Timothy J	male	54.0	0	
..	
884	Sutehall, Mr. Henry Jr	male	25.0	0	
885	Rice, Mrs. William (Margaret Norton)	female	39.0	0	
886	Montvila, Rev. Juozas	male	27.0	0	
887	Graham, Miss. Margaret Edith	female	19.0	0	
888	Johnston, Miss. Catherine Helen "Carrie"	female	28.0	1	

889		Behr, Mr. Karl Howell	male	26.0	0
890		Dooley, Mr. Patrick	male	32.0	0

	Parch		Ticket	Fare	Cabin	Embarked
0	0		A/5 21171	7.2500	NaN	S
1	0		PC 17599	71.2833	C85	C
2	0	STON/O2.	3101282	7.9250	NaN	S
3	0		113803	53.1000	C123	S
4	0		373450	8.0500	NaN	S
5	0		330877	8.4583	NaN	Q
6	0		17463	51.8625	E46	S
..
884	0	SOTON/OQ	392076	7.0500	NaN	S
885	5		382652	29.1250	NaN	Q
886	0		211536	13.0000	NaN	S
887	0		112053	30.0000	B42	S
888	2	W./C.	6607	23.4500	NaN	S
889	0		111369	30.0000	C148	C
890	0		370376	7.7500	NaN	Q

[891 rows x 12 columns]

In [6]: # 補完數字以後只剩 *Cabin*(字串) 和 *Embarked*(字串) 有遺漏值了
`df.isnull().sum()`

Out[6]:

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	0
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

dtype: int64

1.2 類別 (字串) 數值的缺失處理 如果選擇補上的話，這裡的字串就會直接補上最常出現的字串
但是 Cabin 實在遺漏太多了 687/891，所以我直接選擇放棄這個欄位

```
In [7]: df = df.drop(["Cabin"], axis = 1)
```

Embarked 我選擇補上最常出現的值

```
In [8]: df['Embarked'].value_counts()
```

```
Out[8]: S    644  
       C    168  
       Q     77  
       Name: Embarked, dtype: int64
```

```
In [9]: print("Embarked 最常出現:", df['Embarked'].value_counts().idxmax())  
       df['Embarked'] = df['Embarked'].fillna(df['Embarked'].value_counts().idxmax())  
       print("補完後的 nan:")  
       print(df.isnull().sum())
```

Embarked 最常出現: S

補完後的 nan:

```
PassengerId    0  
Survived       0  
Pclass         0  
Name           0  
Sex            0  
Age            0  
SibSp          0  
Parch         0  
Ticket         0  
Fare           0  
Embarked       0  
dtype: int64
```

1.4.3 Step 2. 特徵的處理

在隨機森林的時候，數值特徵並不需要特別的處理，因為我們是選擇一個數，分成左右兩份，
所以就算你把數做標準化 (濃縮成 0 到 1) 也不會影響任何事物

但是在處理類別 (字串) 特徵的時候，我們通常會選擇一種處理方式叫做 One-Hot Encoding(獨熱編碼)

為什麼要特別處理類別 (字串)，因為事實上我們的 `scikit-learn` 和大多數的機器學習函式庫在處理特徵的時候，都只會把特徵值當成數值來處理

所以我們會把我們的字串轉成整數表示

這時候就有一個問題了，假設你的特徵值是 0(紅色) 1(藍色) 2(綠色)

但其實紅藍綠根本沒有一個誰大誰小的順序這就會造成問題了

這時候我們會選擇把這個欄位分成三個欄位

One-Hot Encoding 例子

id	顏色
0	紅
1	藍
2	綠

id	紅
0	1
1	0
2	0

我們這裡使用 `get_dummies` 就會幫我們創造出 one-hot encoding 之後再把它連接回去原本的 `df` 就好

In [10]: # 創造出 one-hot 欄位

```
dummy = pd.get_dummies(df['Embarked'])
# concat 是連結的意思, axis = 1 指的是水平的連接
df = pd.concat([df, dummy], axis=1)
df = df.drop(["Embarked"], axis = 1)

dummy = pd.get_dummies(df['Sex'])
df = pd.concat([df, dummy], axis=1)
df = df.drop(["Sex"], axis = 1)

df
```

```
Out[10]:
```

	PassengerId	Survived	Pclass	\
0	1	0	3	

1	2	1	1
2	3	1	3
3	4	1	1
4	5	0	3
5	6	0	3
6	7	0	1
..
884	885	0	3
885	886	0	3
886	887	0	2
887	888	1	1
888	889	0	3
889	890	1	1
890	891	0	3

	Name	Age	SibSp	Parch	\
0	Braund, Mr. Owen Harris	22.0	1	0	
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0	
2	Heikkinen, Miss. Laina	26.0	0	0	
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	1	0	
4	Allen, Mr. William Henry	35.0	0	0	
5	Moran, Mr. James	28.0	0	0	
6	McCarthy, Mr. Timothy J	54.0	0	0	
..	
884	Sutehall, Mr. Henry Jr	25.0	0	0	
885	Rice, Mrs. William (Margaret Norton)	39.0	0	5	
886	Montvila, Rev. Juozas	27.0	0	0	
887	Graham, Miss. Margaret Edith	19.0	0	0	
888	Johnston, Miss. Catherine Helen "Carrie"	28.0	1	2	
889	Behr, Mr. Karl Howell	26.0	0	0	
890	Dooley, Mr. Patrick	32.0	0	0	

	Ticket	Fare	C	Q	S	female	male
0	A/5 21171	7.2500	0	0	1	0	1
1	PC 17599	71.2833	1	0	0	1	0
2	STON/O2. 3101282	7.9250	0	0	1	1	0
3	113803	53.1000	0	0	1	1	0

```

4          373450    8.0500  0  0  1          0          1
5          330877    8.4583  0  1  0          0          1
6          17463    51.8625  0  0  1          0          1
..          ...          ... .. .. ..          ...          ...
884  SOTON/OQ 392076    7.0500  0  0  1          0          1
885          382652   29.1250  0  1  0          1          0
886          211536   13.0000  0  0  1          0          1
887          112053   30.0000  0  0  1          1          0
888      W./C. 6607   23.4500  0  0  1          1          0
889          111369   30.0000  1  0  0          0          1
890          370376    7.7500  0  1  0          0          1

```

```
[891 rows x 14 columns]
```

接下來處理名字，名字對我而言有用的東西是稱謂。

這裡就一定要畫個圖看一下到底每個稱謂出現了幾次

我畫完了以後發現 Mr Mrs Miss 是三個出現最多而且看起來真的有影響的稱謂，其餘都是一些稀少稱謂

所以我就只留 Mr Mrs Miss，其餘直接丟掉

```

In [11]: s = df['Name'].str.split(",", expand = True)[1]
s = s.str.split(" ", expand = True)[1]
# 這裡我要產生 pdf 的時候 style 會印製不出來，所以我用最素的
# 讀者可以把下面的註解拿掉替換
# pd.crosstab(s, df['Survived']).T.style.background_gradient(cmap = "autumn")
pd.crosstab(s, df['Survived']).T

```

```

Out[11]: 1      Capt.  Col.  Don.  Dr.  Jonkheer.  Lady.  Major.  Master.  Miss.  \
Survived
0          1      1      1      4          1      0          1      17      55
1          0      1      0      3          0      1          1      23     127

1      Mlle.  Mme.  Mr.  Mrs.  Ms.  Rev.  Sir.  the
Survived
0          0      0  436     26      0      6      0      0
1          2      1   81     99      1      0      1      1

```

```

In [12]: def name_filter(data):
          if data == 'Mr.':

```

```

        return 'Mr'
    elif data == 'Mrs.':
        return 'Mrs'
    elif data == 'Miss.':
        return 'Miss'
    else:
        return 'Unknown'
df['Name'] = s.apply(name_filter)

dummy = pd.get_dummies(df['Name'])
df = pd.concat([df, dummy], axis=1)
df = df.drop(["Name"], axis = 1)
df = df.drop(["Unknown"], axis = 1)

```

```
df
```

```
Out[12]:
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Ticket \
0	1	0	3	22.0	1	0	A/5 21171
1	2	1	1	38.0	1	0	PC 17599
2	3	1	3	26.0	0	0	STON/O2. 3101282
3	4	1	1	35.0	1	0	113803
4	5	0	3	35.0	0	0	373450
5	6	0	3	28.0	0	0	330877
6	7	0	1	54.0	0	0	17463
..
884	885	0	3	25.0	0	0	SOTON/OQ 392076
885	886	0	3	39.0	0	5	382652
886	887	0	2	27.0	0	0	211536
887	888	1	1	19.0	0	0	112053
888	889	0	3	28.0	1	2	W./C. 6607
889	890	1	1	26.0	0	0	111369
890	891	0	3	32.0	0	0	370376

	Fare	C	Q	S	female	male	Miss	Mr	Mrs
0	7.2500	0	0	1	0	1	0	1	0
1	71.2833	1	0	0	1	0	0	0	1
2	7.9250	0	0	1	1	0	1	0	0

3	53.1000	0	0	1	1	0	0	0	1
4	8.0500	0	0	1	0	1	0	1	0
5	8.4583	0	1	0	0	1	0	1	0
6	51.8625	0	0	1	0	1	0	1	0
...
884	7.0500	0	0	1	0	1	0	1	0
885	29.1250	0	1	0	1	0	0	0	1
886	13.0000	0	0	1	0	1	0	0	0
887	30.0000	0	0	1	1	0	1	0	0
888	23.4500	0	0	1	1	0	1	0	0
889	30.0000	1	0	0	0	1	0	1	0
890	7.7500	0	1	0	0	1	0	1	0

[891 rows x 16 columns]

Ticket 的數字太多，而且沒有一個很直覺的規律，所以我先將其丟棄

PassengerId 只是一個一直增加的數，也並不影響我們的結果，所以也先將其丟棄

```
In [13]: df = df.drop(["Ticket"], axis = 1)
         #df = df.drop(["Name"], axis = 1)
         df = df.drop(["PassengerId"], axis = 1)
```

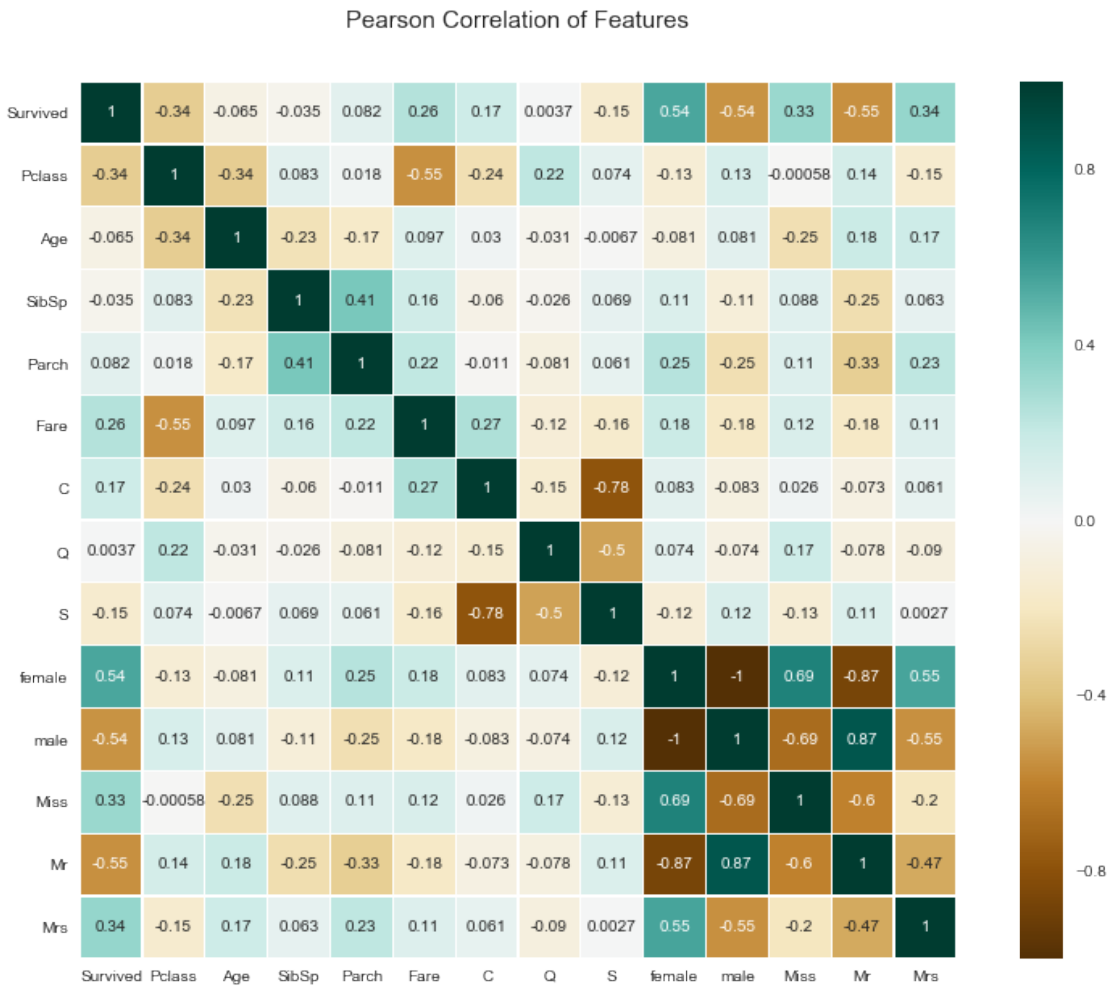
```
In [14]: # 我們把我們擁有的資料集分成兩份，一份測試，一份訓練
         from sklearn.model_selection import train_test_split
         # 把資料分成兩部分 (1. 訓練資料 2. 測試資料)
         data_train, data_test, target_train, target_test = train_test_split(
                                                         df.drop(["Survived"], axis = 1),
                                                         df['Survived'],
                                                         test_size=0.1)
```

1.4.4 Step 3. 初步感覺

畫個 heatmap 感覺一下什麼是比較重要的，基本你發現了，性別大概就是最重要的差別在那個事件上，大部分男士都很紳士的讓女士先上救生艇了！

```
In [15]: plt.figure(figsize=(14,10))
         plt.title('Pearson Correlation of Features', y=1.05, size=15)
         sns.heatmap(df.astype(float).corr(), cmap = "BrBG",
                     linewidths=0.1, square=True, linecolor='white',
                     annot=True)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x11c094cc0>



1.5 開始預測

1.5.1 Step 1. 初步感覺

我們先使用之前說過的 `DecisionTreeClassifier` 看看一個分類樹會怎麼處理我們的問題

```
In [16]: from sklearn.tree import DecisionTreeClassifier
         clf = DecisionTreeClassifier(max_depth = 15, min_samples_leaf = 20)
         clf = clf.fit(data_train, target_train)
```

```
In [17]: from sklearn.tree import export_graphviz
         import graphviz
```

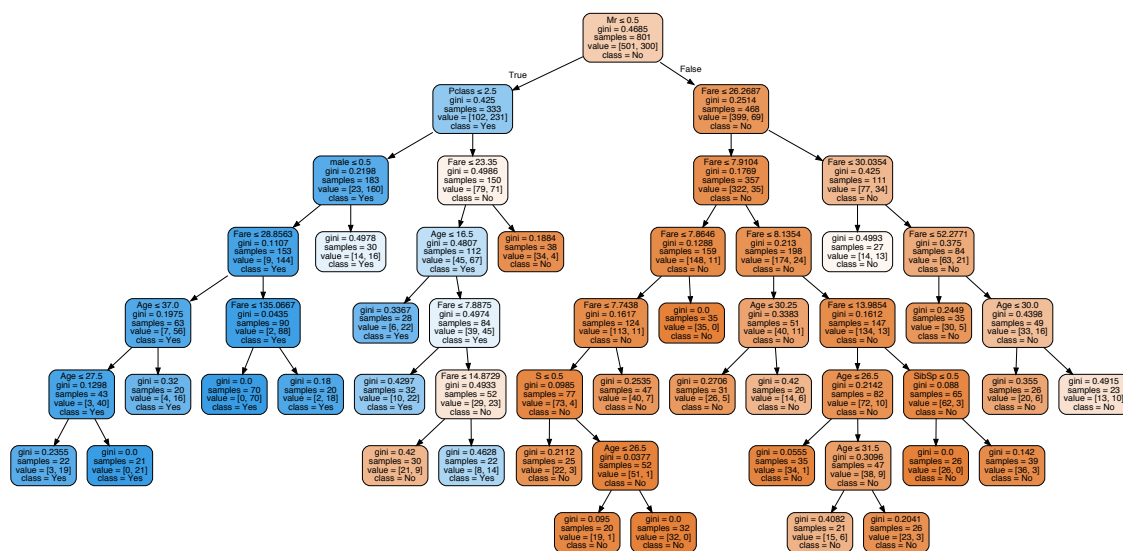
```

dot_data = export_graphviz(clf, out_file=None,
                             feature_names=df.drop(["Survived"], axis = 1).columns,
                             class_names=["No", "Yes"],
                             filled=True, rounded=True,
                             special_characters=True)

graph = graphviz.Source(dot_data)
# 這行可以輸出一個 pdf, 讀者可以自行把註解拿掉試試看
# graph.render("iris2")
graph

```

Out[17]:



In [18]: `from sklearn.metrics import accuracy_score`

```

predict = clf.predict(data_test)
print("預測:", predict)
print("正確標籤:", list(target_test))
print("正確率: ", accuracy_score(target_test, predict) * 100, "%")

```

預測: [0 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1

1 0 0 1 1 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 0

1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0]

正確標籤: [0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1,

正確率: 82.2222222222221 %

1.5.2 Step 2. 建立模型

開始使用隨機森林來建立模型，不過這次我們不使用隨機切割了

我們做一件事叫做交叉驗證

交叉驗證指的是每次把一部分的 `samples` 拿出來當訓練，另外一部分拿來當測試

最後算出每次驗證的平均

交叉驗證可以較好的表示我們模型的好壞

`RandomForest` 的參數調整我們通常我們會調整

`n_estimators`: 要有幾顆樹，理論上越多越好，但是如果你的資料集沒有這麼大，產生太多樹反而沒那麼有意義，因為你會有很多同類型的臭皮匠

`max_depth`: 理論上不需要被調整，但我還是會建議調整一下，不要讓你每個臭皮匠過擬合
這裡經過我的調整我選擇 26 顆樹，每棵樹六層的深度

```
In [19]: from sklearn.ensemble import RandomForestClassifier
         clf = RandomForestClassifier(n_estimators = 26, max_depth = 6)

In [20]: # 這裡就不用 fit 了, fit 和 predict 會由交叉驗證幫你做, cv 參數代表要幾次的交叉驗證
         from sklearn.model_selection import cross_val_score
         scores = cross_val_score(clf, df.drop(["Survived"], axis = 1),
                                df['Survived'], cv = 10)

         print("十次分數:", scores)
         # 由於 score 是 ndarray, 可以直接使用 average 來計算平均
         import numpy as np
         print("平均:", np.average(scores))
```

十次分數: [0.81111111 0.85555556 0.7752809 0.88764045 0.85393258 0.83146067
0.83146067 0.7752809 0.86516854 0.85227273]

平均: 0.8339164113040518

1.5.3 Step 3. 正式預測

正式的預測我們的 `test.csv`，並且把結果輸出並上傳

```
In [21]: from sklearn.ensemble import RandomForestClassifier
         # 6 or 7 is good
         clf = RandomForestClassifier(n_estimators = 26, max_depth = 6)
```



```

clf = clf.fit(df.drop(["Survived"], axis = 1), df['Survived'])
test_df = pd.read_csv("test.csv")

result_df = pd.DataFrame(columns = ["PassengerId", "Survived"])
result_df["PassengerId"] = test_df["PassengerId"]

test_df = test_df.fillna(df.median())
test_df = test_df.drop(["Cabin"], axis = 1)
test_df['Embarked'] = test_df['Embarked'].fillna("S")

dummy = pd.get_dummies(test_df['Embarked'])
test_df = pd.concat([test_df, dummy], axis=1)
test_df = test_df.drop(["Embarked"], axis = 1)
test_df = test_df.drop(["Ticket"], axis = 1)
test_df = test_df.drop(["PassengerId"], axis = 1)

dummy = pd.get_dummies(test_df['Sex'])
test_df = pd.concat([test_df, dummy], axis=1)
test_df = test_df.drop(["Sex"], axis = 1)

s = test_df['Name'].str.split(",", expand = True)[1]
s = s.str.split(" ", expand = True)[1]

test_df['Name'] = s.apply(name_filter)

dummy = pd.get_dummies(test_df['Name'])
test_df = pd.concat([test_df, dummy], axis=1)
test_df = test_df.drop(["Name"], axis = 1)
test_df = test_df.drop(["Unknown"], axis = 1)

test_df

```

Out[21]: Pclass Age SibSp Parch Fare C Q S female male Miss Mr \

0	3	34.5	0	0	7.8292	0	1	0	0	1	0	1
1	3	47.0	1	0	7.0000	0	0	1	1	0	0	0
2	2	62.0	0	0	9.6875	0	1	0	0	1	0	1
3	3	27.0	0	0	8.6625	0	0	1	0	1	0	1
4	3	22.0	1	1	12.2875	0	0	1	1	0	0	0
5	3	14.0	0	0	9.2250	0	0	1	0	1	0	1
6	3	30.0	0	0	7.6292	0	1	0	1	0	1	0
..
411	1	37.0	1	0	90.0000	0	1	0	1	0	0	0
412	3	28.0	0	0	7.7750	0	0	1	1	0	1	0
413	3	28.0	0	0	8.0500	0	0	1	0	1	0	1
414	1	39.0	0	0	108.9000	1	0	0	1	0	0	0
415	3	38.5	0	0	7.2500	0	0	1	0	1	0	1
416	3	28.0	0	0	8.0500	0	0	1	0	1	0	1
417	3	28.0	1	1	22.3583	1	0	0	0	1	0	0

	Mrs
0	0
1	1
2	0
3	0
4	1
5	0
6	0
..	...
411	1
412	0
413	0
414	0
415	0
416	0
417	0

[418 rows x 13 columns]

```
In [22]: pre = clf.predict(test_df)
```


```
result_df["Survived"] = pre  
result_df.to_csv("result_test.csv", index = False)
```

上傳你的第一個競賽的結果吧!!

你會發現我們的模型在沒看過的資料上面大概是 78% - 81% 的正確率

排名在 11000 多個排名排名約 700

已經表現得非常不錯了!!

714	▼ 56	Elwing		0.80861
-----	------	--------	---	---------