**Data Mining hw2 F76074402 張圜華**

開發環境: PYTHON

***Dataset***

使用 Kaggle(titanic/train.csv) 進行修改, 訂定新的規則, 使存亡結果不同

存亡規則:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Survived | Pclass | Sex | Age | SibSp | Parch |
| 0 | 3 |  | <=20 |  |  |
| 0 | 3 |  | >40 |  |  |
| 0 | 2 | Male | <=20 |  |  |
| 0 | 2 | Male | >40 |  |  |
| 0 | 1 | Male | <=10 |  |  |
| 0 | 1 | male | >50 |  |  |
| 0 |  |  | 40>=age>20 |  | >1 |
| 0 |  |  | >40 |  | 0 |
| 0 |  |  | <=20 | >1 |  |
| 1 | 不符合上面規則的, 都是可以存活的 | | | | |

Data Dictionary:

|  |  |  |
| --- | --- | --- |
| Variable | Definition | Key |
| survived | Survival | 0 = No, 1 = Yes |
| pclass | Ticket class | 1 = 1st, 2 = 2nd, 3 = 3rd |
| sex | Sex | Male, Female |
| Age | Age in years |  |
| sibsp | # of siblings / spouses aboard the Titanic |  |
| parch | # of parents / children aboard the Titanic |  |
| ticket | Ticket number |  |
| fare | Passenger fare |  |
| cabin | Cabin number |  |
| embarked | Port of Embarkation | C = Cherbourg,  Q = Queenstown,  S = Southampton |

***Decision Tree***

決策樹的本質是一種貪婪演算法, 一開始所有的訓練樣本都在根節點, 並以計算過後各屬性的統計性測量當作基礎, 挑選最好的屬性來當作分割點, 反覆地將樣本分隔開來, 一直到以下條件發生，我們就停止分支：A. 某個分支子集合內的所有樣本都屬於同一個類別時 B.可能所有的屬性都用完了, 用多數投票法以樣本數較多類別來代表此葉節點 C.選取屬性後, 發生某個分支完全沒有測試樣本的情況

主要方法:

from sklearn import tree

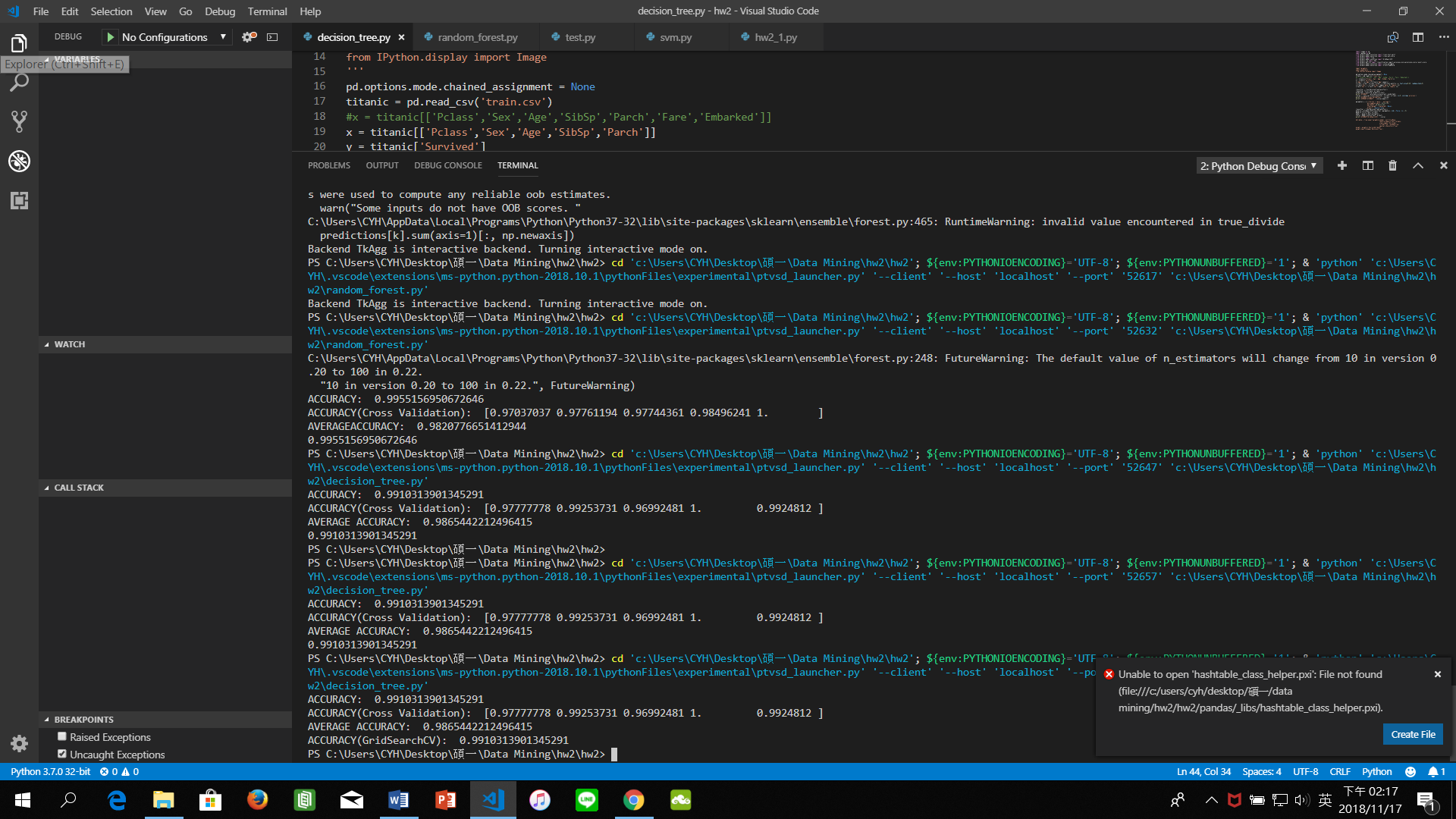
classifier = tree.DecisionTreeClassifier()

clf = classifier.fit(x\_train, y\_train)

predicted = classifier.predict(x\_test)

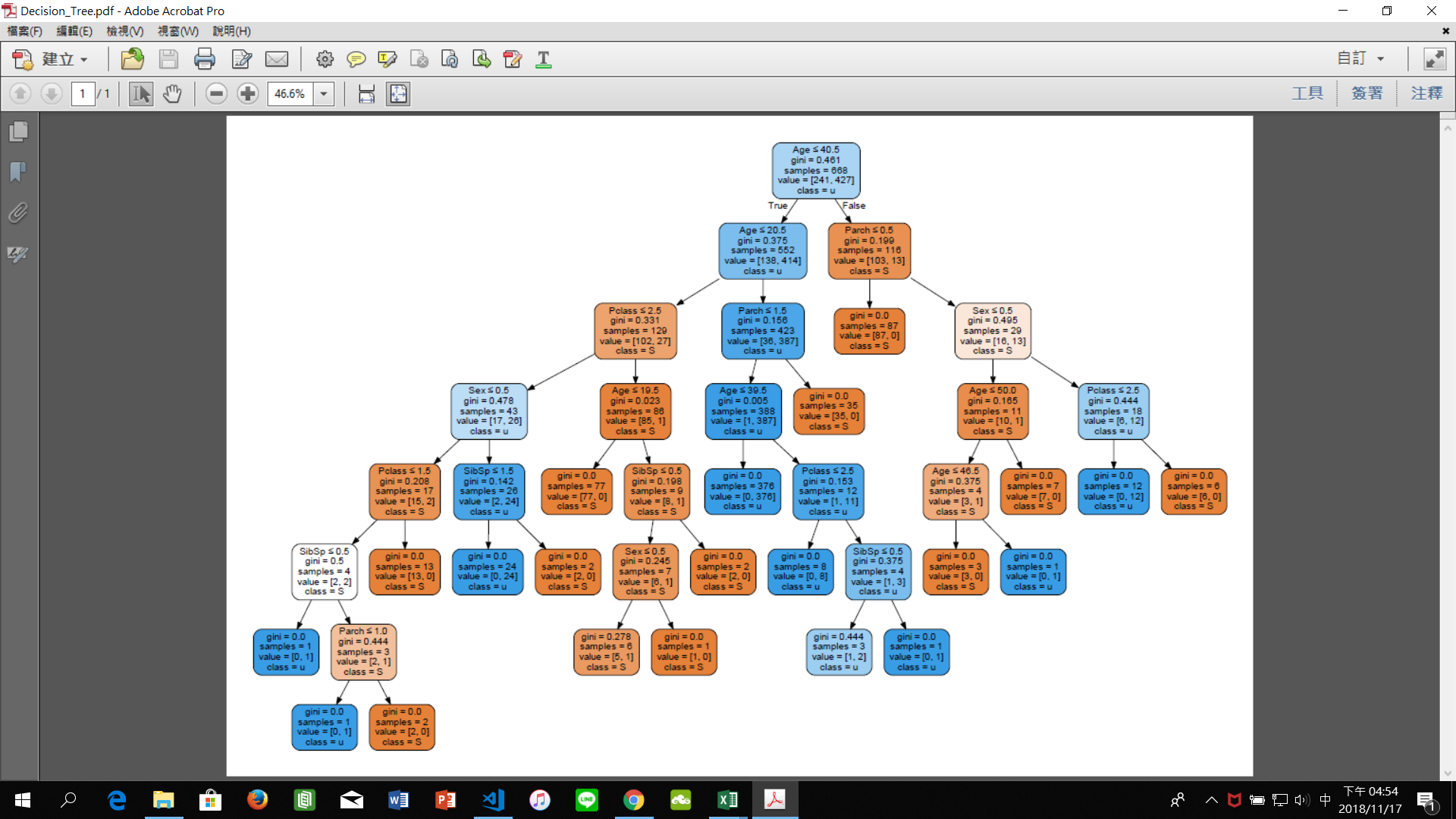
print('ACCURACY: ',accuracy\_score(y\_test, predicted))

結果:



* ACCURACY: 是使用預設參數的 accuracy 結果
* ACCURACY(CROSS Validation): 是使用交叉驗證五次後的 accuracy 結果
* cross\_val\_score(classifier, x\_train, y\_train, cv=5, scoring='accuracy')
* AVERAGE ACCURACY: 是交叉驗證後五次的平均 accuracy
* ACCURACY(GridSearchCV): 是使用GridSearchCV 函式, 對所設置的所有參數做所有的可能排列組合下去測, 然後取得一個最佳參數, 最後獲得的 accuracy結果
* classifier = tree.DecisionTreeClassifier()
* gsearch = GridSearchCV(classifier, parameters, iid = False, cv = 5)
* gsearch.fit(x\_train, y\_train)
* model = gsearch.best\_estimator\_
* score = model.score(x\_test, y\_test)

輸出圖:



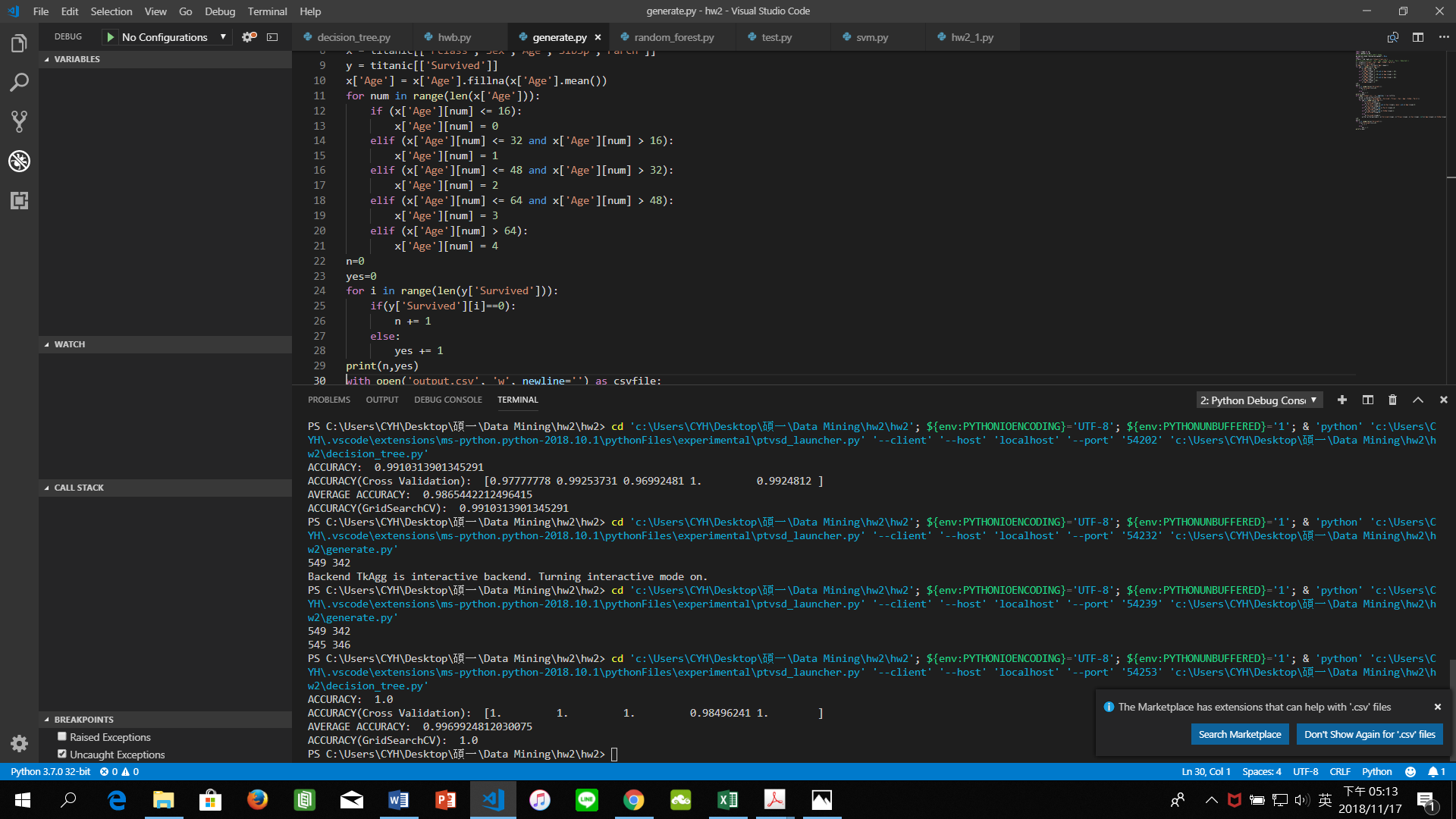
***Decision Tree*** ***VS*** ***Absolutely Right Rules(自訂的規則)***

因為原先訂定的規則太多(總共有九個死亡規則), 因此輸出的圖太過複雜, 不容易比較和觀察, 所以另外重新定義規則(共四個死亡規則), 以此做討論與分析(在這主要都討論會死亡的規則)

新的存亡規則:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Survived | Pclass | Sex | Age | SibSp | Parch |
| 0 | 3 |  |  |  |  |
| 0 | 2 | Male | >64 |  |  |
| 0 |  |  | >48 |  | 0 |
| 0 |  |  | <=16 | >1 |  |
| 1 | 不符合上面規則的, 都是可以存活的 | | | | |

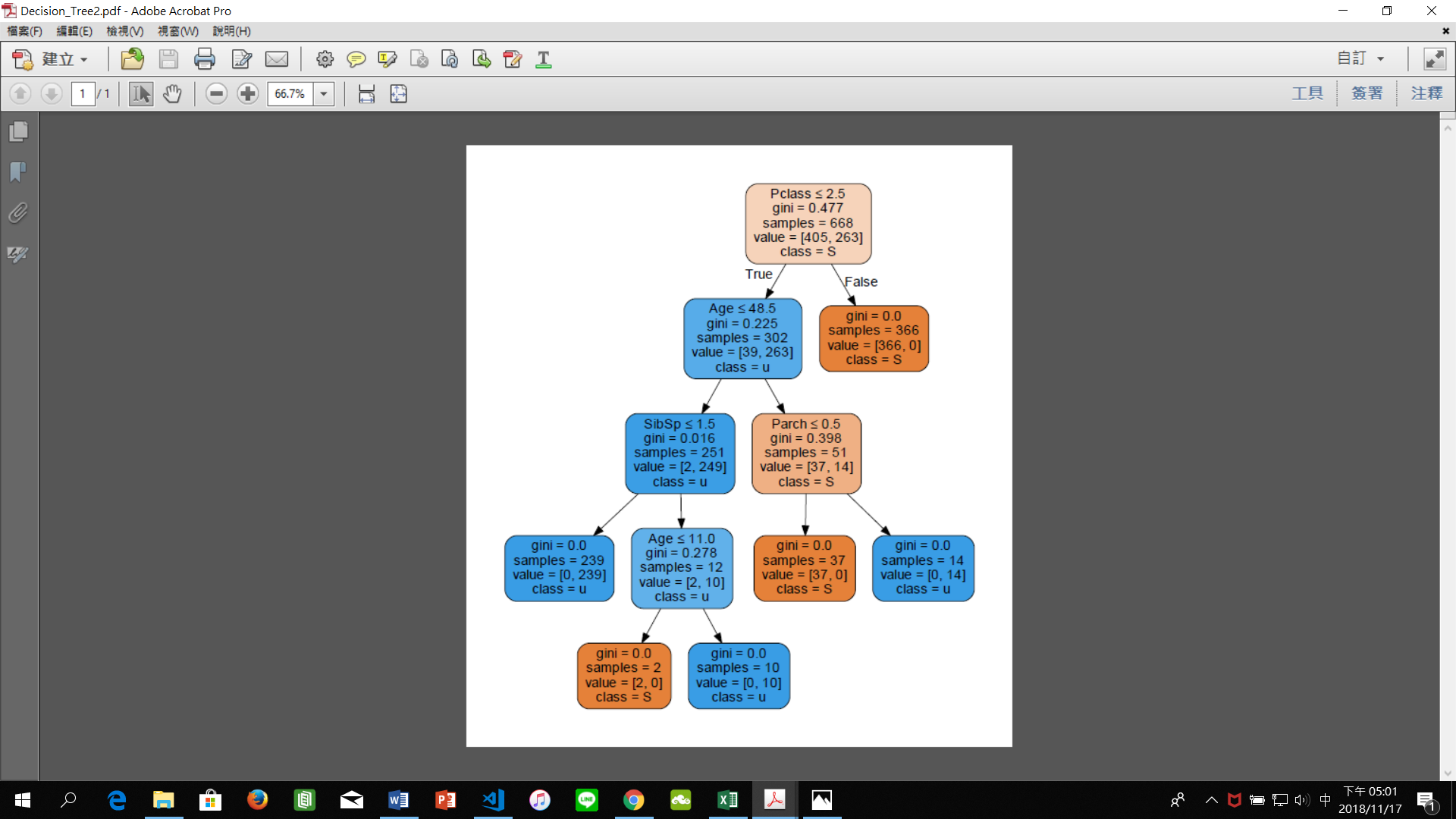
結果:



* ACCURACY: 是使用預設參數的 accuracy 結果
* ACCURACY(CROSS Validation): 是使用交叉驗證五次後的 accuracy 結果
* cross\_val\_score(classifier, x\_train, y\_train, cv=5, scoring='accuracy')
* AVERAGE ACCURACY: 是交叉驗證後五次的平均 accuracy
* ACCURACY(GridSearchCV): 是使用GridSearchCV 函式, 對所設置的所有參數做所有的可能排列組合下去測, 然後取得一個最佳參數, 最後獲得的 accuracy結果
* classifier = tree.DecisionTreeClassifier()
* gsearch = GridSearchCV(classifier, parameters, iid = False, cv = 5)
* gsearch.fit(x\_train, y\_train)
* model = gsearch.best\_estimator\_
* score = model.score(x\_test, y\_test)

從上圖可發現新的存亡規則的準確率較原始的存亡規則高, 這可能跟複雜度有關, 因為新的存亡規則簡單, 所以可以很正確的分好類別

輸出圖:



**DecisionTree:**

會對目標類嘗試進行最佳的分割, 從根到葉子節點都有一條路,這條路徑就是一條規則, 所以可以由上面的圖看到, 死亡的規則有三個, 存活的規則也有三個

**DecisionTree 所產生的規則(主要探討死亡規則)如下:**

1. pclass >2.5 (也就是 pclass = 3) 時, 會死亡

* 第一條規則

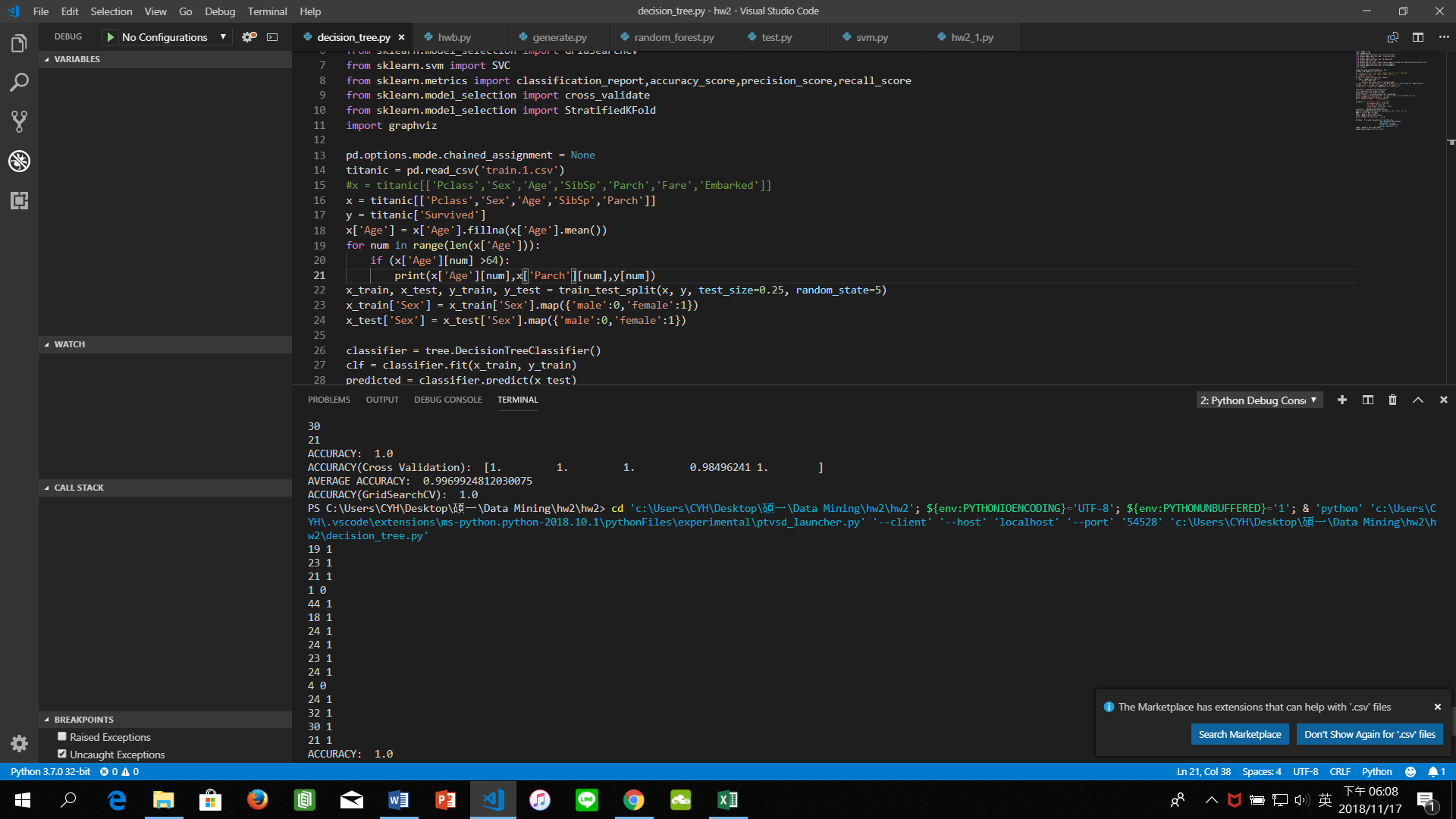
1. age > 48.5(也就是 age > 48) 且 parch <= 0.5(也就是 parch = 0) 時, 會死亡

* 第三條規則

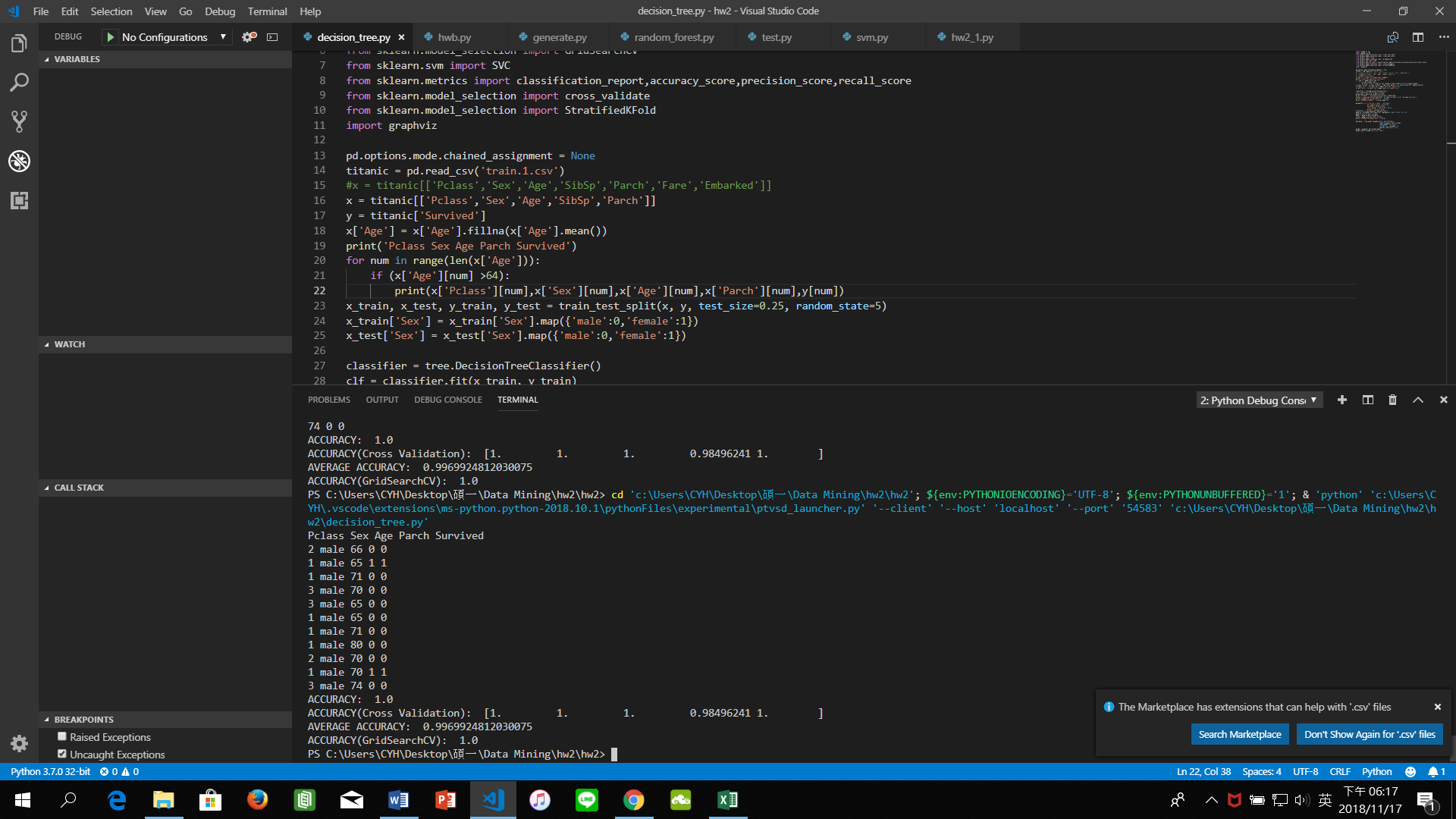
1. sibsp > 1.5(也就是 sibsp >1) 且 age <= 11時, 會死亡

* 第四條規則

1. 在第A點時,可以很明顯觀察到,完全符合absolutely right rules(自訂的規則)的第一條規則
2. 在第B點時,也可以很明顯觀察到,完全符合absolutely right rules(自訂的規則)的第三條規則
3. 在第C點時,可以看到(age < 11時, 會死亡), 這和原先訂定的規則不同(age < 16時, 會死亡), 這是因為經由GINI的計算, 在age = 11 時, 所得到的gini 值會最大, 而再觀察並查看資料中符合pclass < 3 and age <= 48 and sibsp >1 的data有年紀 1 和 4(會死亡), 而會存活的有年紀 18, 19, 21,…., 所以不論是年紀小於16或是小於11都會符合規則,只是選擇不同而已



1. 而發現沒有符合absolutely right rules(自訂的規則)的第二條規則, 這是因為在資料中所有符合第二條規則的都符合第三條規則,因此只要第三條規則就可以了,所以也可以發現我們可以藉此決策樹找到一些不需要的規則



***SVM***

SVM, 最主要的概念就是希望可以在一個由不同類別混合而成的資料集中，依據一些特徵, 找到一個最佳的超平面, 將不同類別的資料分開來, 讓資料在空間中能夠被區分成兩類，所以SVM又被稱為二元分類器(binary classifier), 而所謂最佳的超平面就是其距離兩個類別的邊界可以達到最大, 而最靠近邊界的這些樣本點提供SVM最多的分類資訊, 就叫做支持向量。

主要方法:

from sklearn.svm import SVC

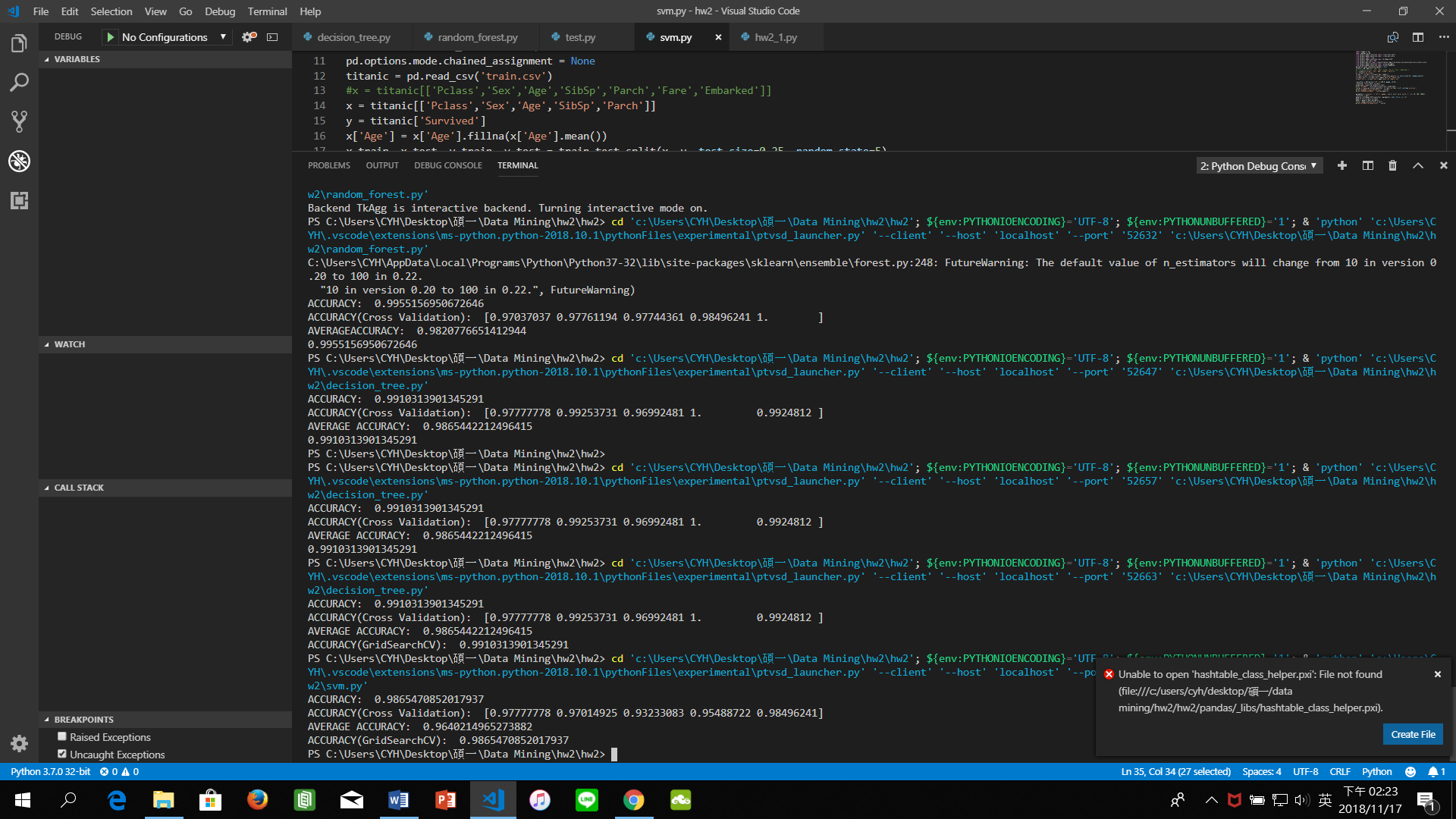
classifier = SVC(kernel="rbf", C=100.0, gamma = 0.01)

classifier.fit(x\_train, y\_train)

predicted = classifier.predict(x\_test)

print('ACCURACY: ',accuracy\_score(y\_test, predicted))

結果:



* ACCURACY: 是使用參數random\_state=10的 accuracy 結果
* ACCURACY(CROSS Validation): 是使用交叉驗證五次後的 accuracy 結果
* cross\_val\_score(classifier, x\_train, y\_train, cv=5, scoring='accuracy')
* AVERAGE ACCURACY: 是交叉驗證後五次的平均 accuracy
* ACCURACY(GridSearchCV): 是使用GridSearchCV 函式, 對所設置的所有參數做所有的可能排列組合下去測, 然後取得一個最佳參數, 最後獲得的 accuracy結果
* classifier = tree.DecisionTreeClassifier()
* gsearch = GridSearchCV(classifier, parameters, iid = False, cv = 5)
* gsearch.fit(x\_train, y\_train)
* model = gsearch.best\_estimator\_
* score = model.score(x\_test, y\_test)

***Random Forest***

隨機森林, 是指用隨機的方式建立一個森林，森林是由很多的決策樹所組成, 隨機森林中的每一棵決策樹之間是沒有關聯的。在得到森林之後，當有一個新的輸入樣本進入的時候，就讓森林中的每一棵決策樹分別進行一下判斷，看看這個樣本應該屬於哪一類（對於分類算法），然後看看哪一類被選擇最多，就預測這個樣本為那個類別

主要方法:

from sklearn.ensemble import RandomForestClassifier

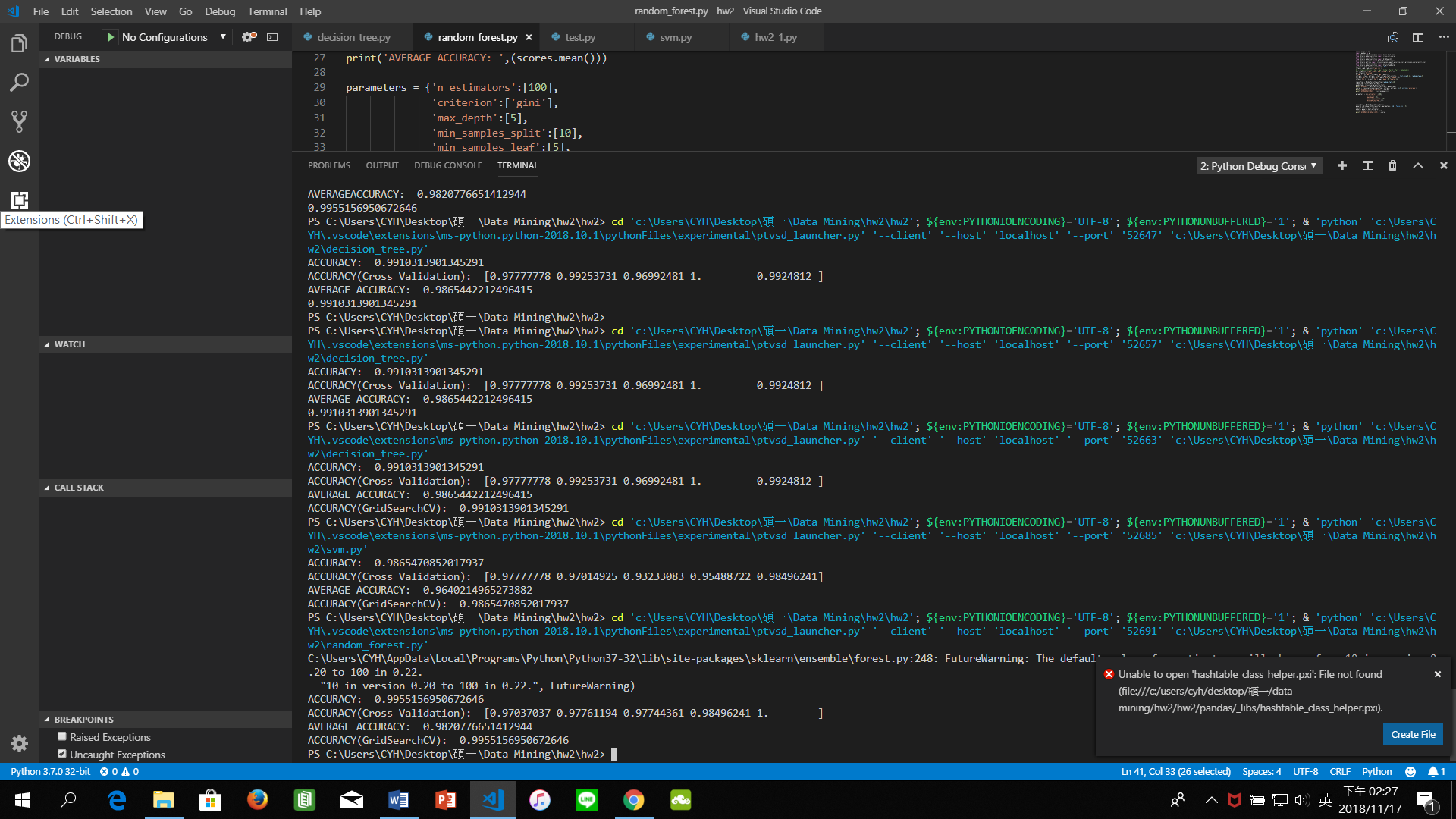
classifier = RandomForestClassifier(random\_state=10)

classifier.fit(x\_train, y\_train)

predicted = classifier.predict(x\_test)

print('ACCURACY: ',accuracy\_score(y\_test, predicted))

結果:



* ACCURACY: 是使用參數kernel="rbf", C=100.0, gamma = 0.01的 accuracy 結果
* ACCURACY(CROSS Validation): 是使用交叉驗證五次後的 accuracy 結果
* cross\_val\_score(classifier, x\_train, y\_train, cv=5, scoring='accuracy')
* AVERAGE ACCURACY: 是交叉驗證後五次的平均 accuracy
* ACCURACY(GridSearchCV): 是使用GridSearchCV 函式, 對所設置的所有參數做所有的可能排列組合下去測, 然後取得一個最佳參數, 最後獲得的 accuracy結果
* classifier = tree.DecisionTreeClassifier()
* gsearch = GridSearchCV(classifier, parameters, iid = False, cv = 5)
* gsearch.fit(x\_train, y\_train)
* model = gsearch.best\_estimator\_
* score = model.score(x\_test, y\_test)

***Decision Tree VS SVM VS Random Forest***

ACCURACY 比較表:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Decision Tree | SVM | Random Forest |
| ACCURACY | 0.99103 | 0.98655 | 0.99552 |
| ACCURACY(Cross Validation) | 0.97778  0.99254  0.96992  1  0.99249 | 0.97778  0.97015  0.93233  0.95489  0.98496 | 0.97037  0.97761  0.97744  0.98496  1 |
| AVERAGE ACCURACY | 0.98654 | 0.96402 | 0.98208 |
| ACCURACY(GridSearchCV) | 0.99103 | 0.98655 | 0.99552 |

ACCURACY:

Random Forest > Decision Tree > SVM

Decision Tree:

優點：

1. 可以生成可以理解的規則
2. 計算量相對來說不是很大
3. 可以處理連續和種類欄位
4. 決策樹可以清晰的顯示哪些欄位比較重要

缺點：

1. 對連續性的欄位比較難預測
2. 對有時間順序的數據, 需要很多預處理的工作
3. 當類別太多時, 錯誤可能就會增加的比較快
4. 一般的演算法分類的時候, 只是根據一個欄位來分類

SVM:

優點:

1. 可以解決高維問題, 即大型特徵空間
2. 能夠處理非線性特徵的相互作用
3. 無需依賴整個數據
4. 可以提高泛化能力

缺點:

1. 當觀測樣本很多時, 效率並不是很高
2. 對非線性問題沒有通用解決方案，有時候很難找到一個合適的核函數
3. 對缺失數據敏感

Random Forest:

優點:

1. 可以處理類別資料與連續資料
2. 對大多數資料而言, 隨機森林演算法的擬和結果準確率高
3. 接受高維度特徵資料
4. 使用 Bagging 採樣方式，以 OOB 方式進行誤差分析，能提升運算效率
5. 對雜訊容忍度高
6. 處理非平衡誤差資料時，能平衡誤差
7. 分類資料時亦能算出相似度

缺點:

1. 運算需要大量記憶體，儲存每顆樹的資訊
2. 因隨機森林是決策樹的組合學習，無法針對單一顆樹作解釋。