機器學習原理及工業應用 FINAL PROJECT

糖尿病風險預測

報告人:**109611099** 機械 大三 顏彦臣
109611090 機械大三 劉定珩

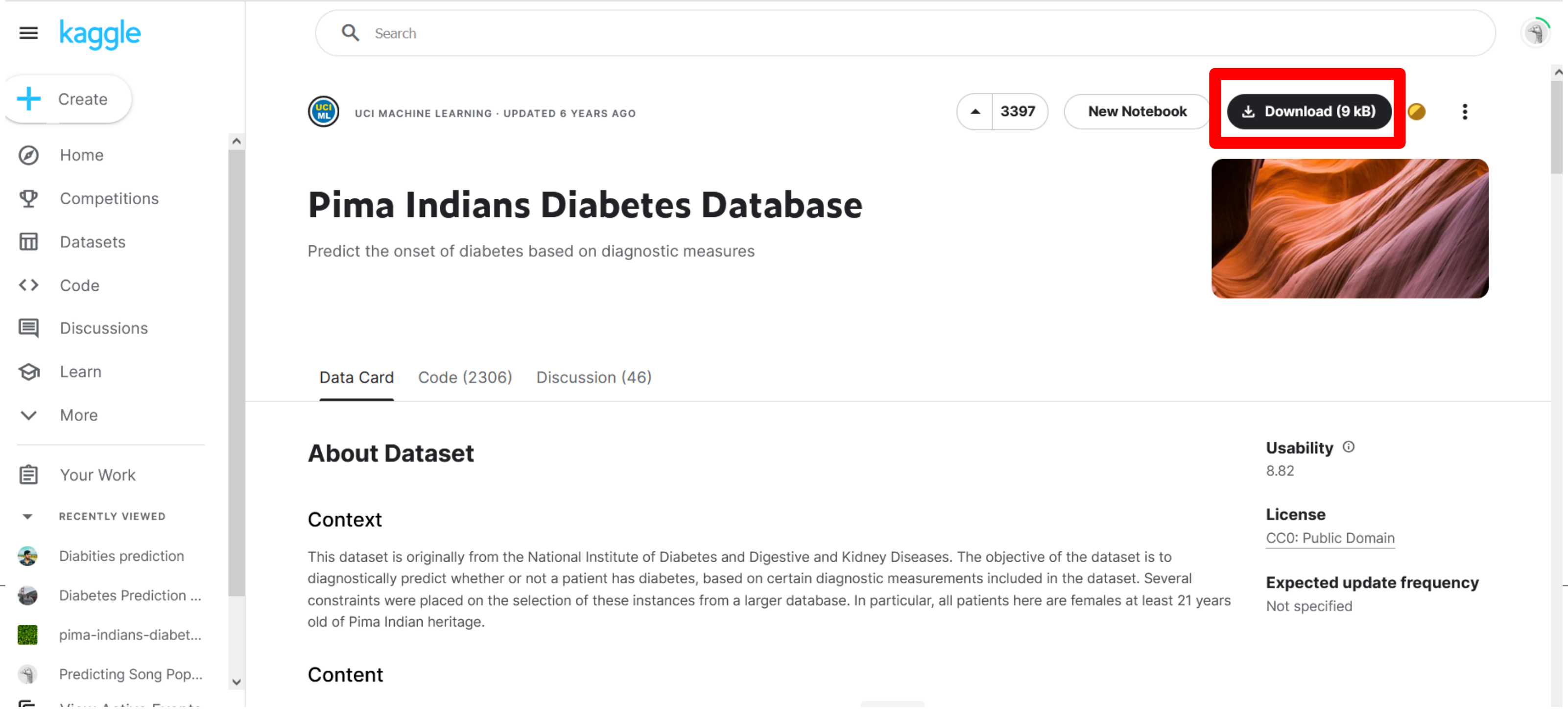
為甚麼選這個題目????

室友的健檢報告

血中白血球(WBC)	8.54	4~10	
血中紅血球(RBC)	5.56	男:4.5~6、女:4~5.5	
血紅素(Hgb)	16.4	男:13.5~17.5女:12~16	
血容積(B_HCT)	47.5	男:41~53、女:36~46	
平均血球體積(MCV)	85.4	80~100	
平均血球血紅素(MCH)	29.5	26~34	
平均血球濃度(MCHC)	34.5	31~37	
血小板(Platelet)	402	150~450	
轉胺脢SGOT	65	13~39	
轉胺脢SGPT	187	7~52	
B肝表面抗原HBsAg	-	<1 Negative COI	
B型肝炎表面抗體(Anti-HBs)	-	>10 Positive IU/L	
尿素氮(BUN)	13	7~25	
肌酸酐(Creatinine)	0.95	男:0.7~1.3、女:0.6~1.2	
尿酸(Uric Acid)	9.9	男:4.4~7.6、女:2.3~6.6	
飯前血糖	83	70~99	
總膽固醇(CHOL)	214	<200	mg/d
三酸甘油脂(TG)	198	<150	mg/dl
高密度脂蛋白膽固醇(HDL)	37	男:≥40;女≥50	mg/dl
低密度脂蛋白膽固醇(LDL)	172	<130	mg/dl

目標:預測自己及室友目前是否有得到糖尿病之風險

收集資料


kaggle
Search
Create
Home
Competitions
Datasets
Code
Discussions
Learn
More
Your Work
RECENTLY VIEWED
Diabities prediction
Diabetes Prediction ...
pima-indians-diabet...
Predicting Song Pop...
UCI MACHINE LEARNING · UPDATED 6 YEARS AGO
3397
New Notebook
Download (9 kB)
Pima Indians Diabetes Database
Predict the onset of diabetes based on diagnostic measures
Data Card
Code (2306)
Discussion (46)
About Dataset
Context
This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.
Content
Usability
8.82
License
CC0: Public Domain
Expected update frequency
Not specified

資料選用(刪去不合適的資料集)

Pregnancie	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1

Pregnancies(懷孕次數)	Glucose(葡萄糖濃度)	BloodPressure(血壓)	SkinThickness(皮下肌肉厚度)	Insulin(胰島素濃度)	BMI	DiabetesPedigreeFunction	Age	Outcome
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
3	78	50	32	88	31	0.248	26	1
2	197	70	45	543	30.5	0.158	53	1
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
0	118	84	47	230	45.8	0.551	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0

資料比數:392
資料來源: UCI
特徵(變數量):8種

- Pregnancies(懷孕次數)
- Glucose(葡萄糖濃度)
- BloodPressure(血壓)
- SkinThickness(皮下肌肉厚度)
- Insulin(胰島素濃度)
- BMI
- DiabetesPedigreeFunction(**糖尿病函數**)
  這個函數使用了家族糖尿病史來導出個人得糖尿病的風險值
- Age(年紀)

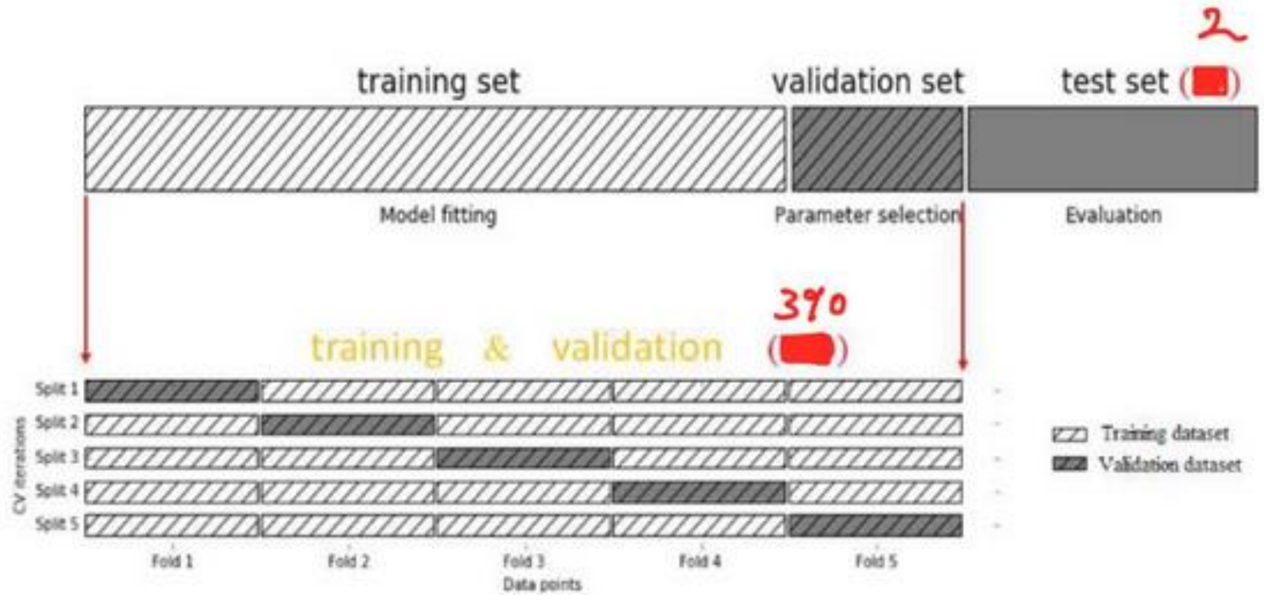
對照結果:
1:得到糖尿病
0:未得糖尿病(健康)

讀取檔案

```python
############################################################
def readfinal(inFileName):
    # init
    recArr = []
    clsArr = []

    # open input text data file, format is given
    inFile = open(inFileName, 'r')
    s = inFile.readline() # skip

    row = 0
    while True:
        s = inFile.readline()
        data1 = s.strip() # remove leading and ending blanks
        if (len(data1) <= 0):
            break

        # since we use append, value must be created in the loop
        value = []

        strs9 = data1.split(',') # array of 31 str

        # convert to real
        for ix in range(8):
            value.append( eval(strs9[ix]) )
        # end for

        target = eval(strs9[8])

        recArr.append(value) ;  # add 1 record at end of array
        clsArr.append(target) ; # add 1 record at end of array

        row = row+1 # total read counter
    # end while
    # close input file
    inFile.close()
    # convert list to Numpy array
    npXY = np.array(recArr)
    npC  = np.array(clsArr)
    # pass out as Numpy array
    return npXY, npC
# end function
```

```python
##############main################################################
X,y=readfinal("C:\prog\ML\ML_FINAL_PROJECT\diabetes.csv")
```

5-fold cross validation



取兩筆資料做TEST(目的僅是為了看各種METHOD 的預測差異)
剩餘390比資料用作5折交叉驗證

```
#392比資料 2比做test 390比做5折驗證
X_5fold, X_check, y_5fold, y_check = train_test_split(X, y, test_size = 2, random_state = 0)
```

取兩筆資料做預測(test case)

146	2	146	70	38	360	28	0.337	29	1
282	8	186	90	35	225	34.5	0.423	37	1

```
Tpredict1=np.array([[2,146,70,38,360,28,0.337,29]])
Ttest1=[1]
Tpredict2=np.array([[8,186,90,35,225,34.5,0.423,37]])
Ttest2=[1]
```

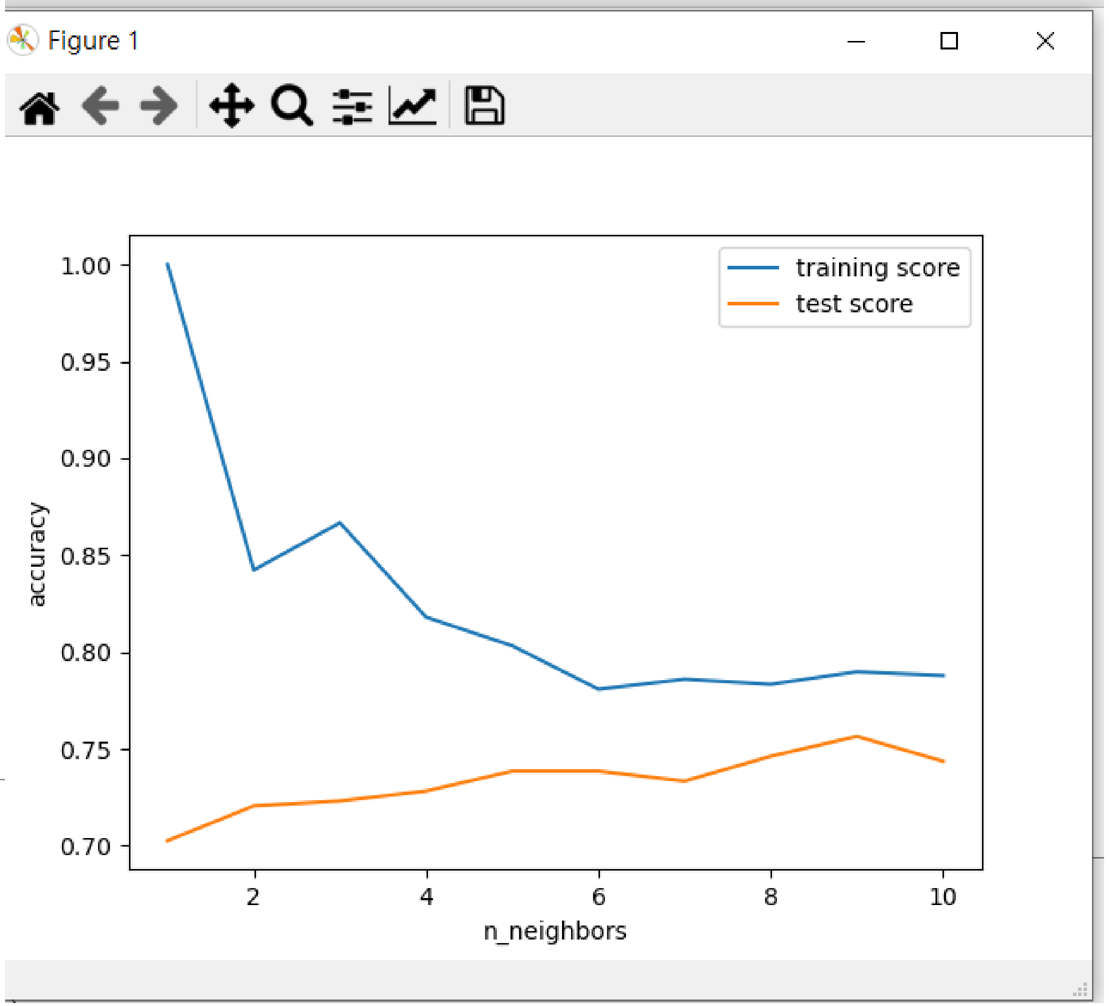
使用library

取用sklearn的四種方法

```
import numpy as np
import numpy as np
import mglearn
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
################model##################
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

KNN

```python
total_test = 0
############################knn############
from sklearn.neighbors import KNeighborsClassifier
neighbors_setting=range(1,11)
training_accuracy=[]
test_accuracy=[]
for num in neighbors_setting:
    total_train = 0
    total_test = 0
    for fold in range(num_folds):
        X_train, X_test, y_train, y_test = TrainTestSplit_Fold(X_5fold, y_5fold, fold, test_size)
        knn=KNeighborsClassifier(n_neighbors=num)
        knn.fit(X_train,y_train)
        y_predict=knn.predict(X_test)
        train_s=knn.score(X_train, y_train)
        test_s=knn.score(X_test,y_test)
        total_train += train_s
        total_test  += test_s
    average_train = total_train/num_folds
    average_test  = total_test/num_folds
    training_accuracy.append(average_train)
    test_accuracy.append(average_test)
plt.plot(neighbors_setting,training_accuracy,label="training score")
plt.plot(neighbors_setting,test_accuracy,label="test score")
plt.ylabel("accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.show()
plt.savefig('knn model比較')
```



N=9時會是最佳模型!

5-fold Training score=0.78
5-fold Testscore=0.75

對兩筆測試數據做預測
兩者皆為預測皆為1
和結果相同。

```
Tpredict1=np.array([[2,146,70,38,360,28,0.337,29]])
Ttest1=[1]
Tpredict2=np.array([[8,186,90,35,225,34.5,0.423,37]])
Ttest2=[1]
```

```
>>> p1
array([1])
>>> p2
array([1])
>>>
```

LogisticRegression

```
##############################LogisticRegression##################
from sklearn.linear_model import LogisticRegression
# loop through folds
total_train = 0
total_test = 0
for fold in range(num_folds):
    X_train, X_test, y_train, y_test = TrainTestSplit_Fold(X_5fold, y_5fold, fold, test_size)
#c=1
    logreg=LogisticRegression(C=1,max_iter=1000).fit(X_train, y_train)
    train_s=logreg.score(X_train, y_train)
    test_s=logreg.score(X_test,y_test)
    total_train += train_s
    total_test  += test_s
average_train = total_train/num_folds
average_test  = total_test/num_folds
print("logic regression for c=1 train/test :{:.3f}/{:.3f}".format(average_train,average_test))
total_train = 0
total_test = 0
```

```
logic regression for c=1 train/test score :0.794/0.782
```

嘗試調整C

當C=100

```
logic regression for c=100 train/test score :0.795/0.785
```

model在training準確度提高一些但test score略降

當C=0.01

```
logic_regression for c=0.01 train/test score :0.787/0.774
```

model在training 和test 準確度都下降

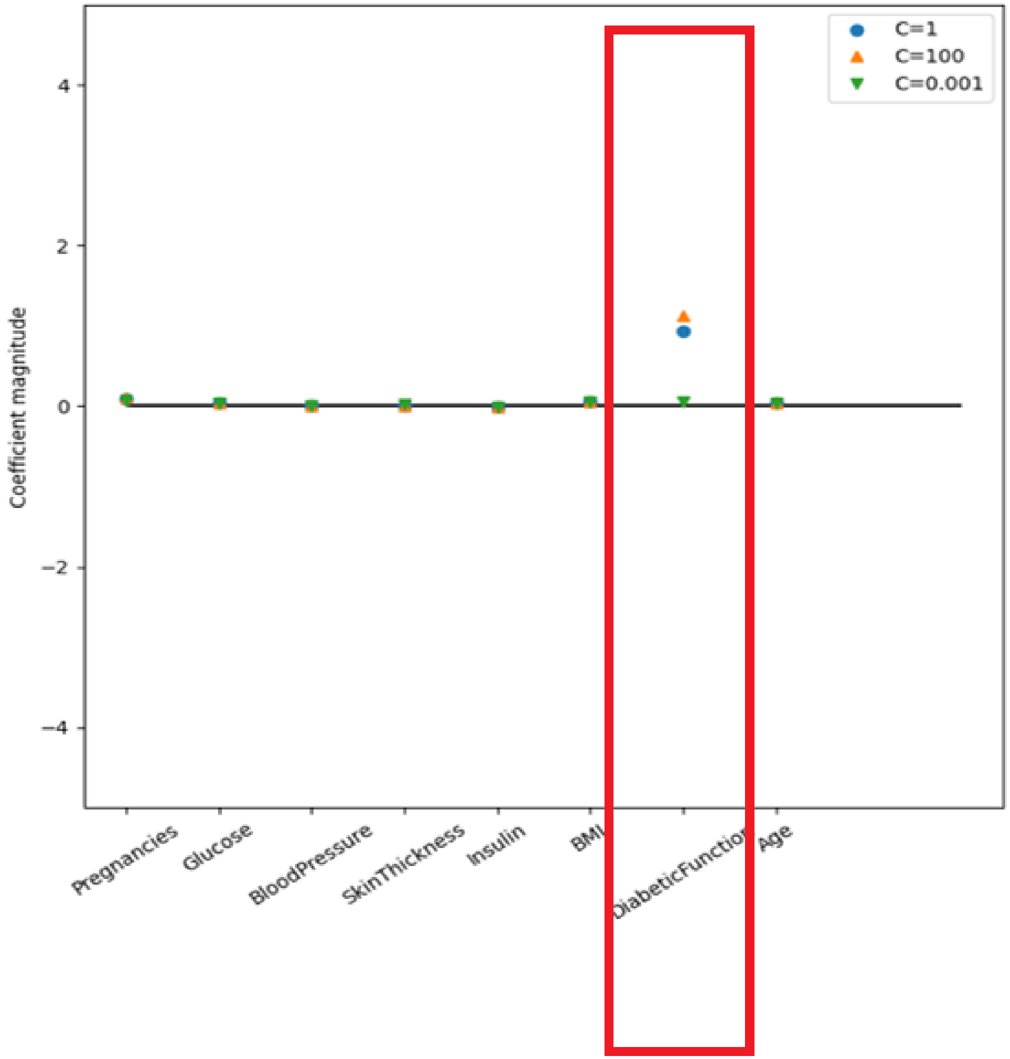
結論:更多或更少的正則化或更複雜的模型並不一定會使模型的預測效果更好。

對testcase 做預測

```
Tpredict1=np.array([[2,146,70,38,360,28,0.337,29]])
Ttest1=[1]
Tpredict2=np.array([[8,186,90,35,225,34.5,0.423,37]])
Ttest2=[1]
```

```
>>> p1
array([0])
>>> p2
array([1])
>>>
```

```python
#######pic###########
feature=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabeticFunction","Age"]
plt.figure(figsize=(8,8))
plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(8), feature, rotation=35)
plt.hlines(0, 0, 9)
plt.ylim(-5, 5)
plt.xlabel("Feature")
plt.ylabel("Coefficient magnitude")
plt.legend()
plt.show()
plt.savefig('log11')
```



使用LogisticRegression 做預測時

DiabetesPedigreeFunction(家族糖尿病函數)影響很大!!!!!

Decision Tree

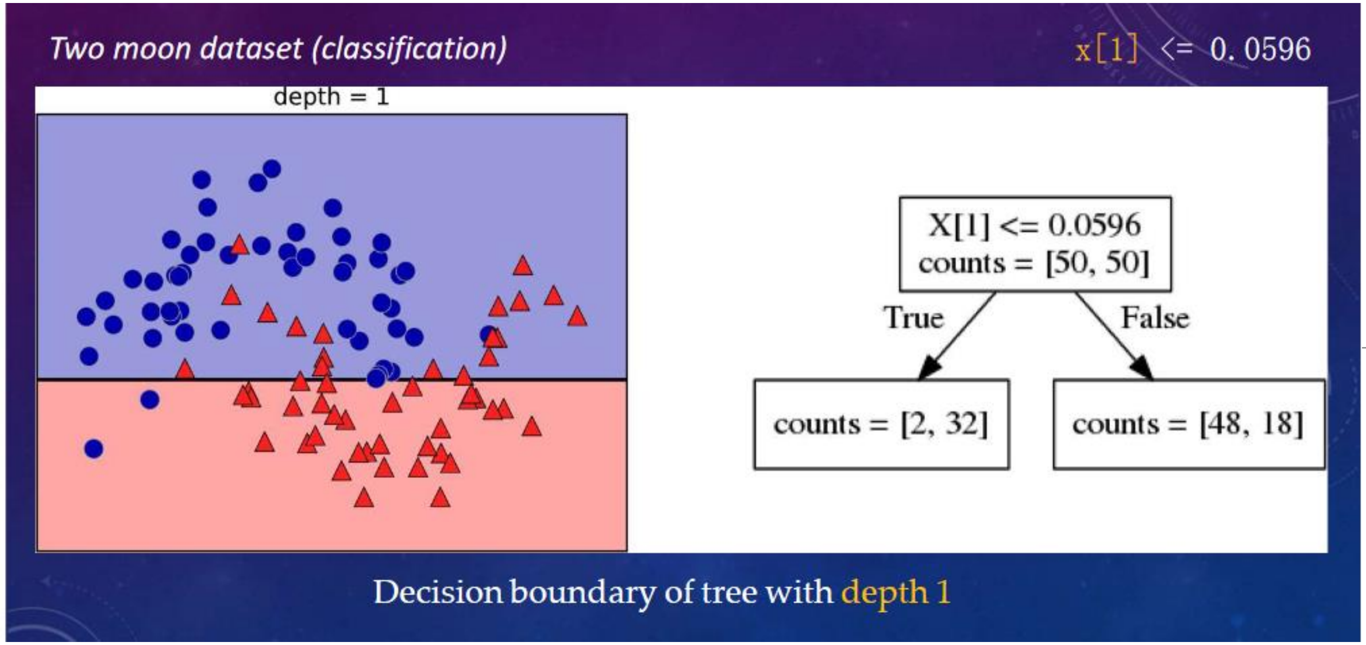
```python
################decision tree######################################
from sklearn.tree import DecisionTreeClassifier
# loop through folds
total_train = 0
total_test = 0
tree=DecisionTreeClassifier(random_state=0)
for fold in range(num_folds):
    X_train, X_test, y_train, y_test = TrainTestSplit_Fold(X_5fold, y_5fold, fold, test_size)
    tree=DecisionTreeClassifier(random_state=0)
    tree.fit(X_train,y_train)
    train_s=tree.score(X_train, y_train)
    test_s=tree.score(X_test,y_test)
    total_train += train_s
    total_test  += test_s
average_train = total_train/num_folds
average_test  = total_test/num_folds
print("decision tree train/test score :{:.3f}/{:.3f}".format(average_train,average_test))
```

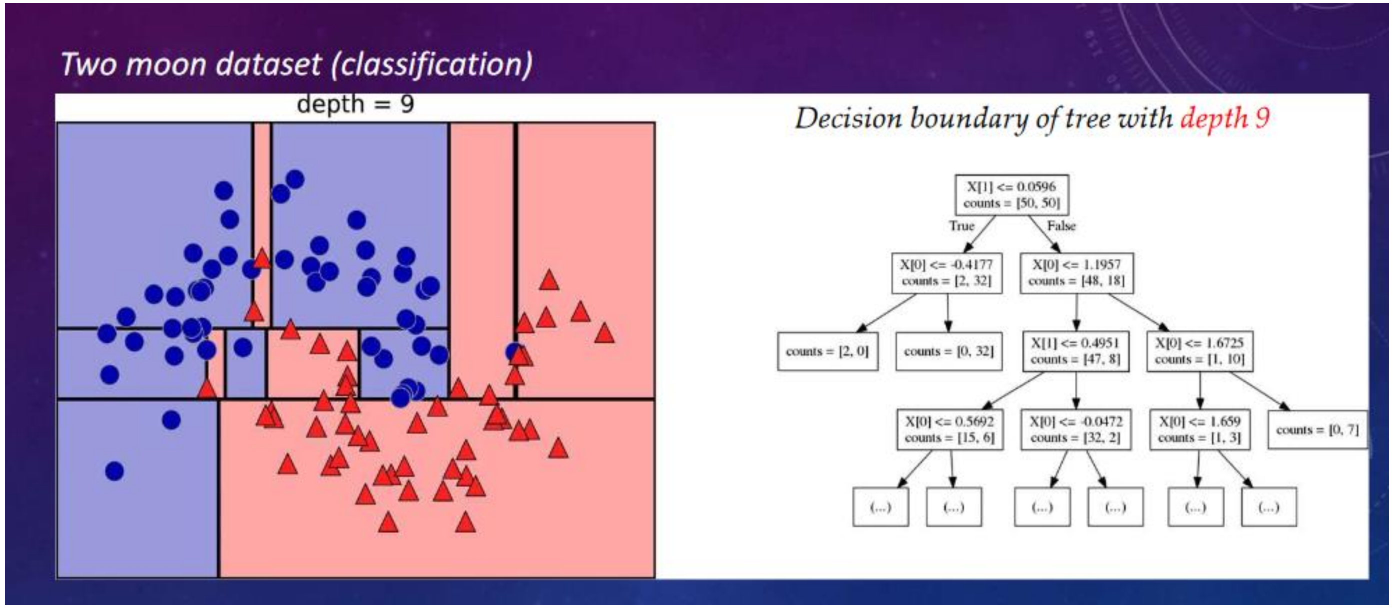
Train score:1.000

Test score:0.715

Overfit!!

Why???





引用:上課講義
[ML_Ch02f.pdf](ML_Ch02f.pdf)

調整樹的最大深度

```
###########################################################################
#max_depth=3
# loop through folds
total_train = 0
total_test = 0

for fold in range(num_folds):
    X_train, X_test, y_train, y_test = TrainTestSplit_Fold(X_5fold, y_5fold, fold, test_size)
    tree=DecisionTreeClassifier(max_depth=3,random_state=0)
    tree.fit(X_train,y_train)
    train_s=tree.score(X_train, y_train)
    test_s=tree.score(X_test,y_test)
    total_train += train_s
    total_test  += test_s
average_train = total_train/num_folds
average_test  = total_test/num_folds
print("decision tree max_depth=3 train/test score :{:.3f}/{:.3f}".format(average_train,average_test))
###########################################################################
```

```
decision tree max_depth=3 train/test score :0.829/0.754
decision tree max_depth=4 train/test score :0.862/0.744
decision tree max_depth=5 train/test score :0.901/0.728
```

Best model

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeFunction Age

```
Tpredict1=np.array([[2,146,70,38,360,28,0.337,29]])
Ttest1=[1]
Tpredict2=np.array([[8,186,90,35,225,34.5,0.423,37]])
Ttest2=[1]
```

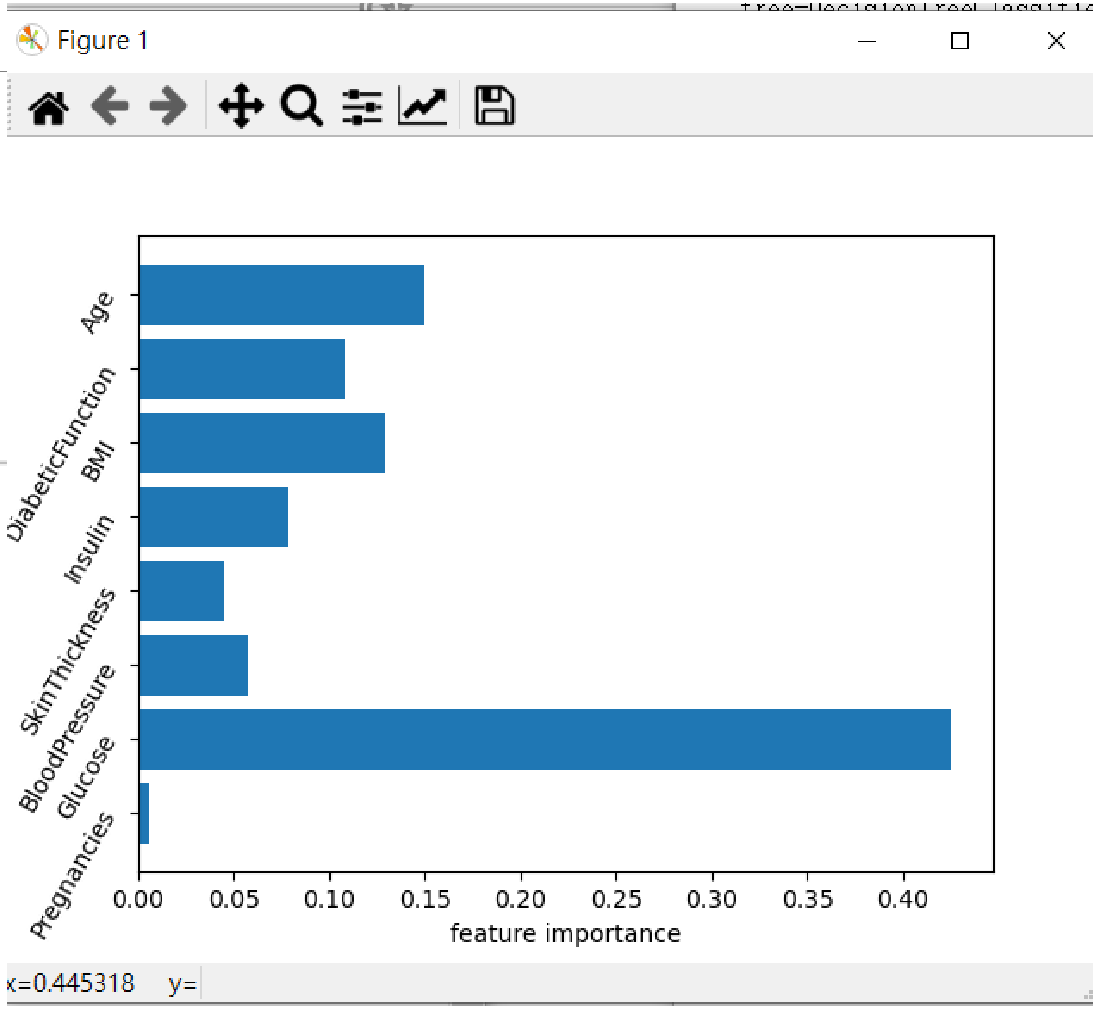
```
>>> p1
array([1])
>>> p2
array([1])
>>>
```

符合預測!

Decision Tree 各特徵重要性

```
###########plot importance pic##########################
feature=["Pregnancies","Glucose","BloodPressure","SkinThickness",\
         "Insulin","BMI","DiabeticFunction","Age"]
n_feature=8
plt.barh(range(8),tree.feature_importances_)
plt.yticks(np.arange(8),feature,rotation=60)
plt.xlabel("feature importance")
plt.ylabel("feature")
plt.show()
```

葡萄糖濃度影響最大!



Random forest

```
##############random forest###################
from sklearn.ensemble import RandomForestClassifier
# loop through folds
total_train = 0
total_test = 0

for fold in range(num_folds):
    X_train, X_test, y_train, y_test = TrainTestSplit_Fold(X_5fold, y_5fold, fold, test_size)
    rf=RandomForestClassifier(n_estimators=100,random_state=0)
    rf.fit(X_train,y_train)
    train_s=rf.score(X_train, y_train)
    test_s=rf.score(X_test,y_test)
    total_train += train_s
    total_test  += test_s
average_train = total_train/num_folds
average_test  = total_test/num_folds
print("ramdom forest tree train/test score :{:.3f}/{:.3f}".format(average_train,average_test))
```

```
ramdom forest tree train/test score :1.000/0.774
```

Overfitting!

調整樹的最大深度

```
ramdom forest tree train/test score :1.000/0.774
ramdom forest tree(max_depth=3) train/test score :0.956/0.785
ramdom forest tree(max_depth=4) tree train/test score :0.876/0.777
ramdom forest tree(max_depth=5) train/test score :0.906/0.777
ramdom forest tree(max_depth=6) train/test score :0.949/0.779
ramdom forest tree(max_depth=7) train/test score :0.975/0.772
```

Best model

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeFunction Age

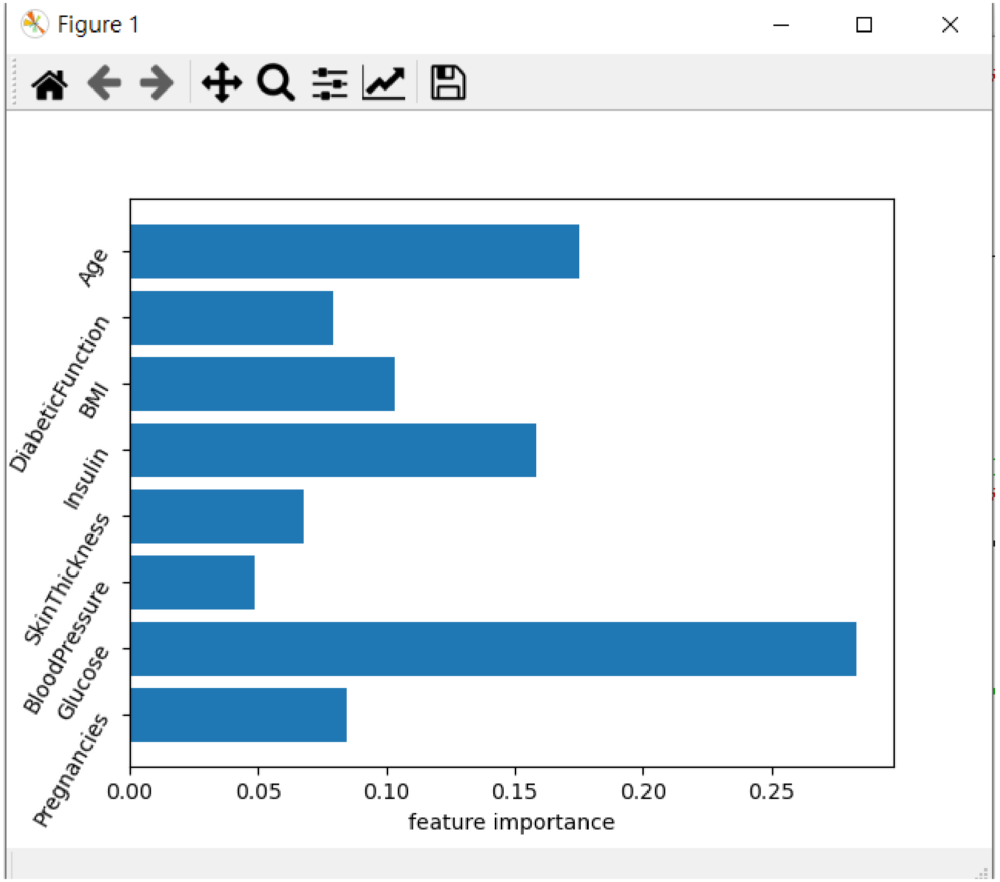
```
Tpredict1=np.array([[2,146,70,38,360,28,0.337,29]])
Ttest1=[1]
Tpredict2=np.array([[8,186,90,35,225,34.5,0.423,37]])
Ttest2=[1]
```

```
>>> p1
array([0])
>>> p2
array([1])
...
```

1 預測錯誤

各特徵重要性

```
###########plot importance pic########################
feature=["Pregnancies","Glucose","BloodPressure","SkinThickness",\
         "Insulin","BMI","DiabeticFunction","Age"]
n_feature=8
plt.barh(range(8),rf6.feature_importances_)
plt.yticks(range(8),feature,rotation=60)
plt.xlabel("feature importance")
plt.ylabel("feature")
plt.show()
```



相較decision tree
除了葡萄糖濃度依舊影響最大外
Age(年紀) insulin(胰島素)影響變大

random forest tree 考慮更多可能性，複雜度更高，但model訓練分數反而較decision tree 差

預測自己!!

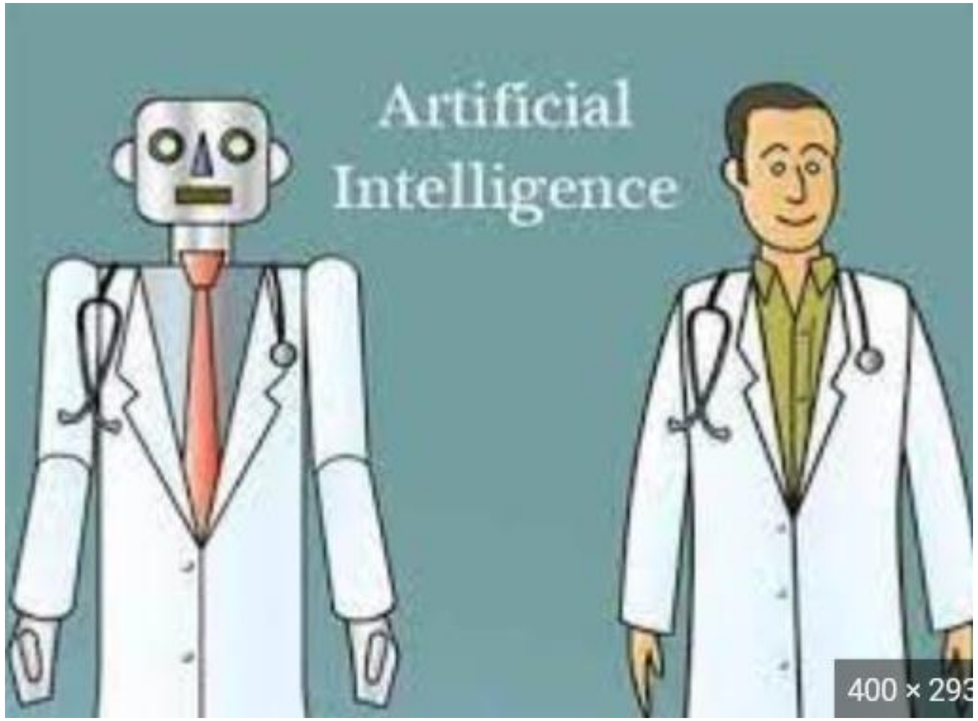
1	Pregnancies(懷	Glucose(葡萄糖	BloodPressure(	SkinThickness(,	Insulin(胰島素濃	BMI	DiabetesPedigr	Age
2	0	83	82	30	100	25.7	0	19
3	0	90	69	30	100	23.8	0	19

```
###########預測自己#######################
Tpredict3=np.array([[0,83,82,30.1,100,25.7,0,19]])
Tpredict4=np.array([[0,90,69,30,100,23.8,0,19]])
```

```
用knn model 預測結果[0],[0]
用logregression 預測結果[0],[0]
用logregression 預測結果[0],[0]
用random forest tree 預測結果[0],[0]
```

不用擔心!!

應用



Thank YOU!!!!!!!!!!!