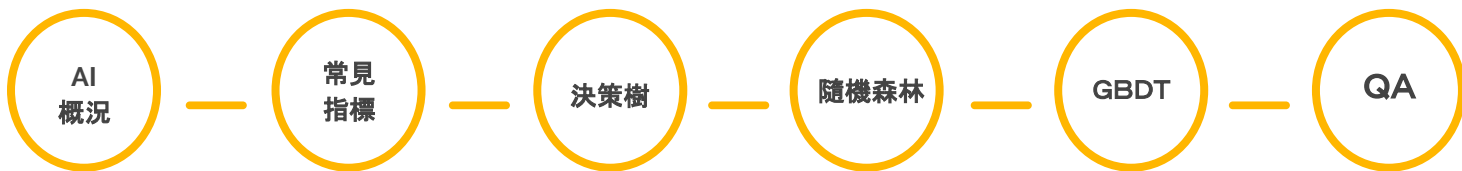


機器學習 - NTUDAC

James Yeh

今天我們會



安裝套件(看決策樹長相)

要在OS先安裝graphviz:

Mac(terminal):

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"  
brew install graphviz
```

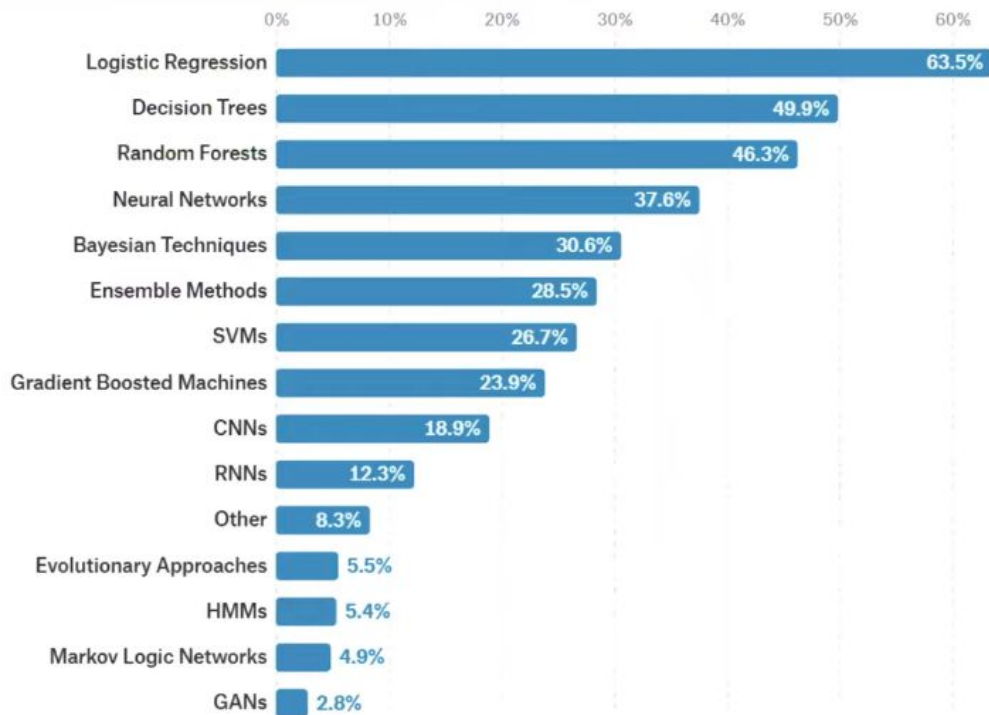
Windows:

https://graphviz.gitlab.io/_pages/Download/Download_windows.html

安裝套件(看決策樹長相)

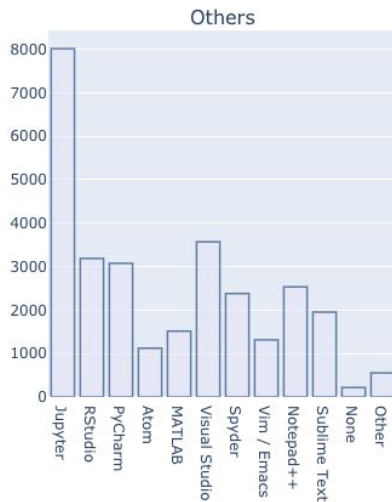
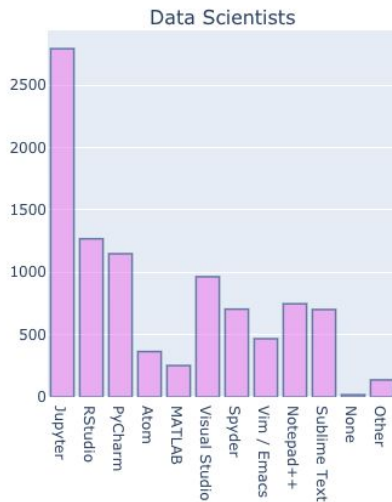
```
pip install graphviz
```

工作中常見的模型(Kaggle統計)



Kaggle統計 人工智慧概況

Development Environments Used Regularly



<https://www.kaggle.com/kaggle/kaggle-survey-2017/kernels>

<https://www.kaggle.com/kaggle/kaggle-survey-2018/kernels>

<https://www.kaggle.com/c/kaggle-survey-2019/notebooks>

常見指標

- Accuracy
- Confusion Matric
- Precision
- Recall
- F1
- PR curve
- ROC curve
- RMSE

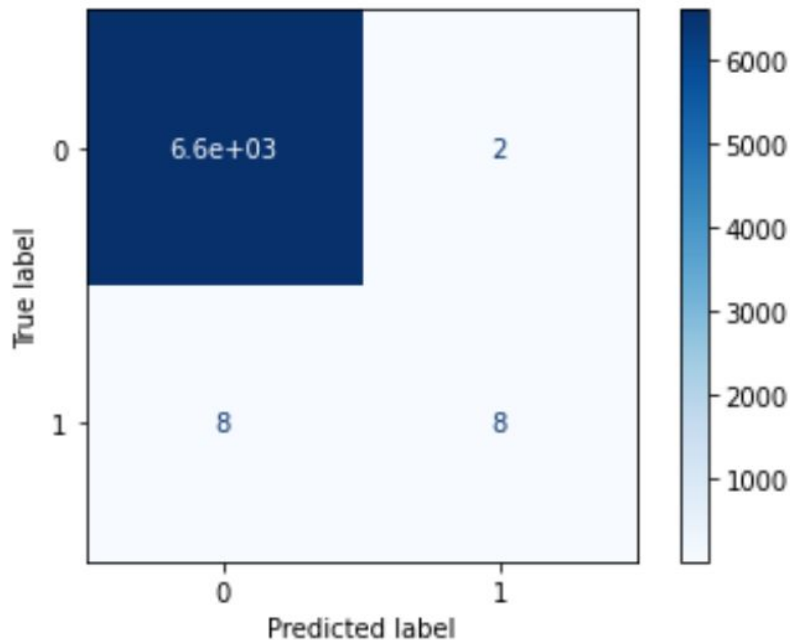
每種指標都只能片面的反應模型的部分性能

Accuracy

$$Accuracy = \frac{N_{correct}}{N_{total}}$$

Confusion matrix(混淆矩陣)

[6599, 2]
[8, 8]



Accuracy的問題

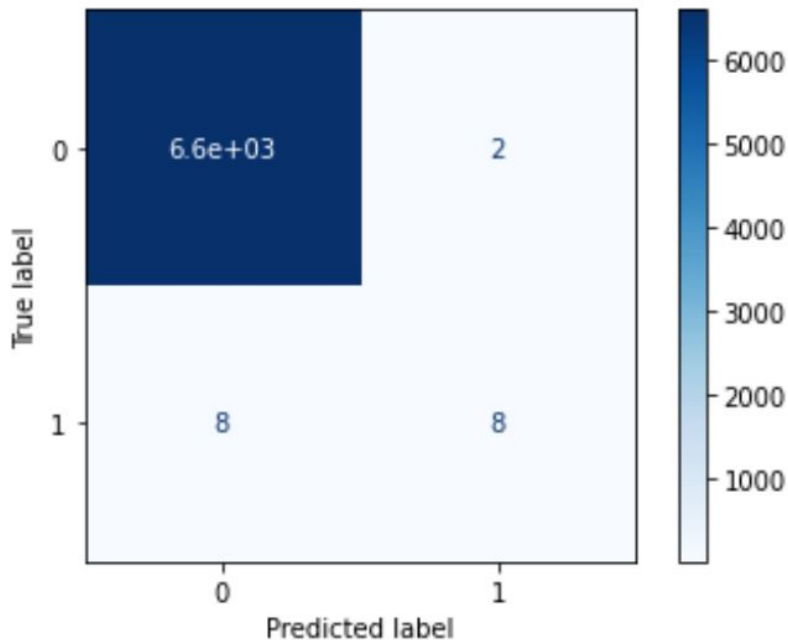
[6599, 2]
[8, 8]

Accuracy
= $6607/6617 = 0.998$
= 99.8%

準確度(Accuracy)

```
lr.score(X_test, y_test)
```

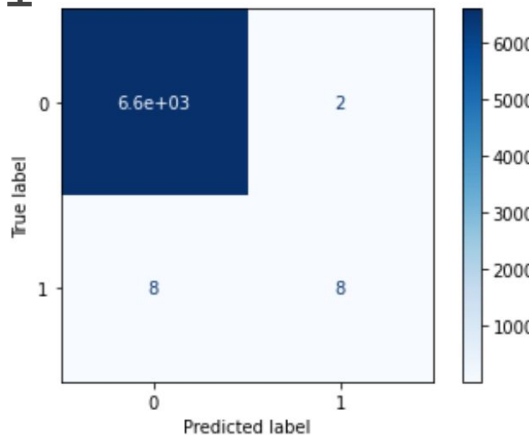
0.9984887411213541



True Negative(TN), False Positive(FP)
預測為0 實際為0 預測為1 實際為0

False Negative(FN), True Positive(TP)
預測為0 實際為1 預測為1 實際為1

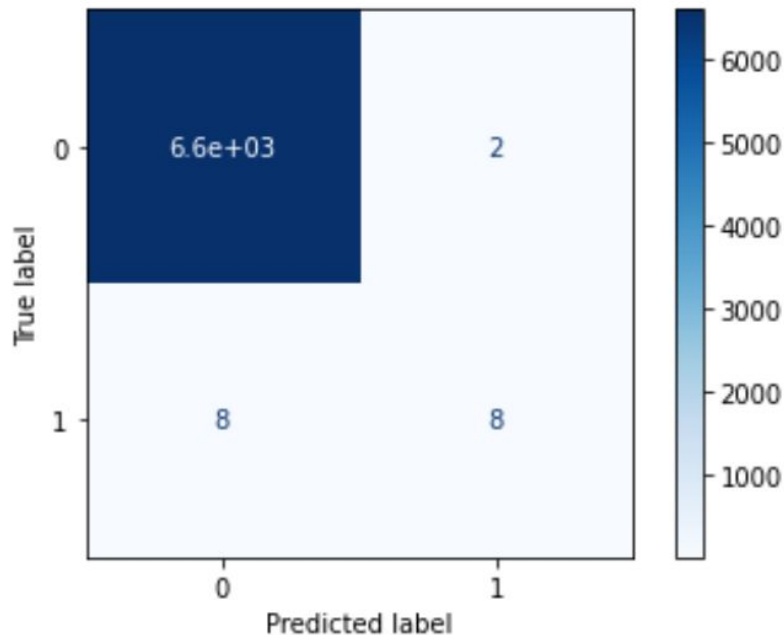
前面是形容詞 後面是名詞



Precision & Recall

$$Precision = \frac{TP}{TP + FP}$$

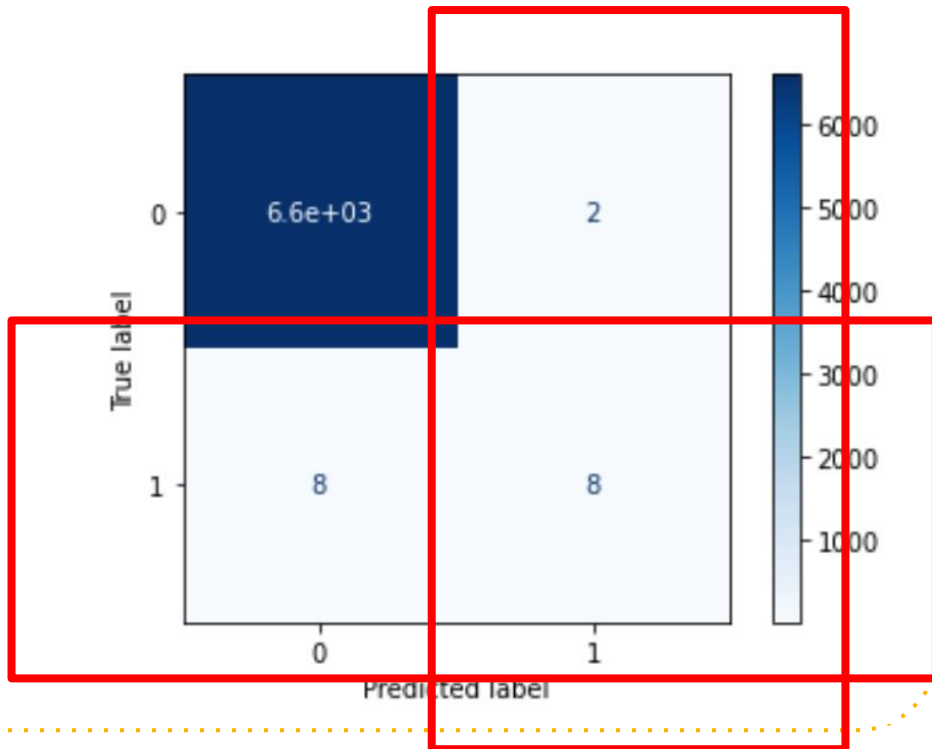
$$Recall = \frac{TP}{TP + FN}$$



Precision & Recall

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$



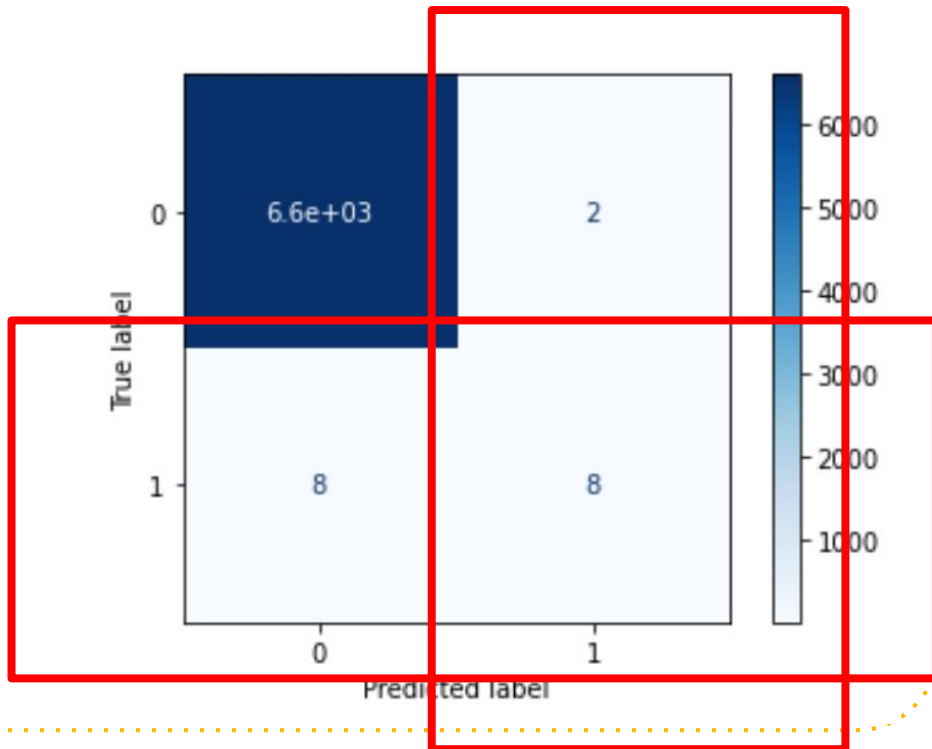
Precision & Recall

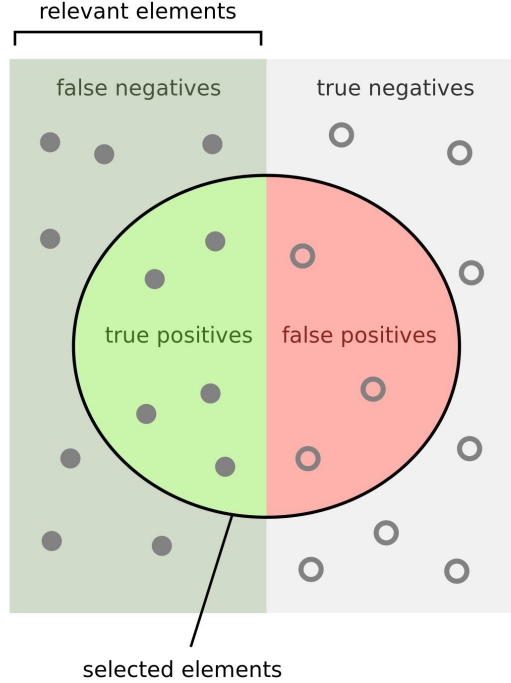
$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

precision = $8/(8+2) = 0.8$

recall = $8/(8+8) = 0.5$





How many selected
items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant
items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Precision & Recall

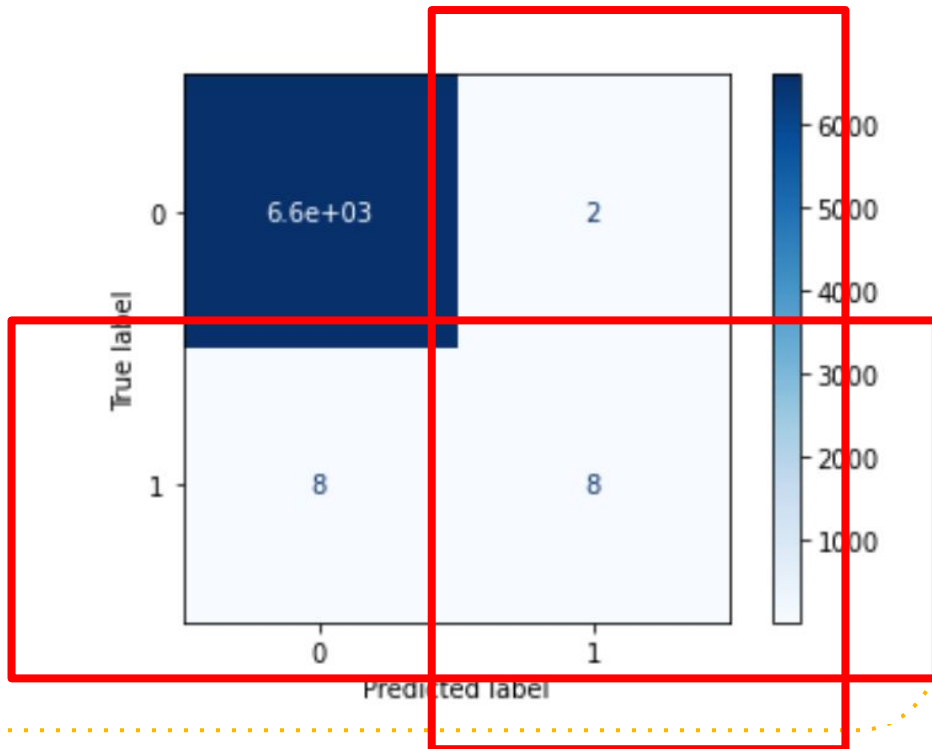
$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

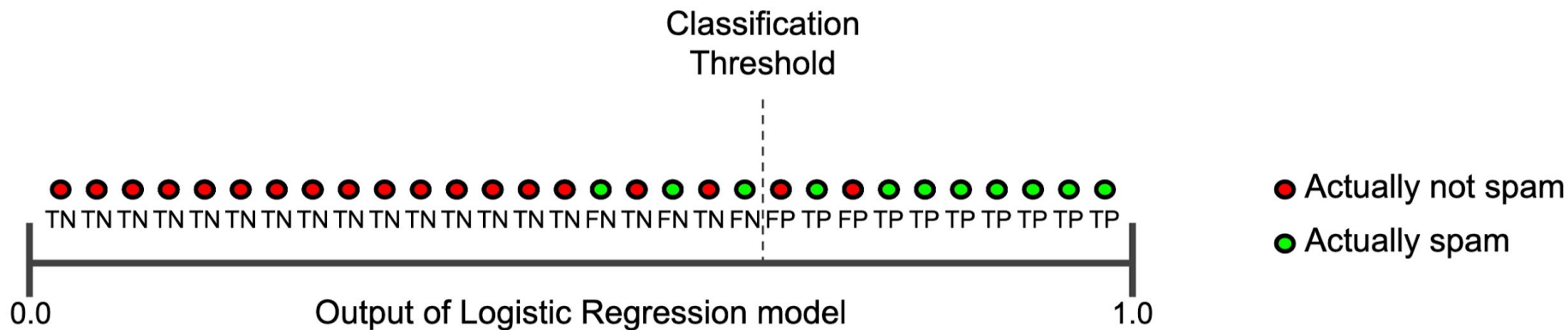
precision = $8/(8+2) = 0.8$

recall = $8/(8+8) = 0.5$

想要提高precision, 越有把握的才預測為正樣本
想要提高recall, 只要有一點可疑就預測為正樣本



Precision & Recall 互為矛盾



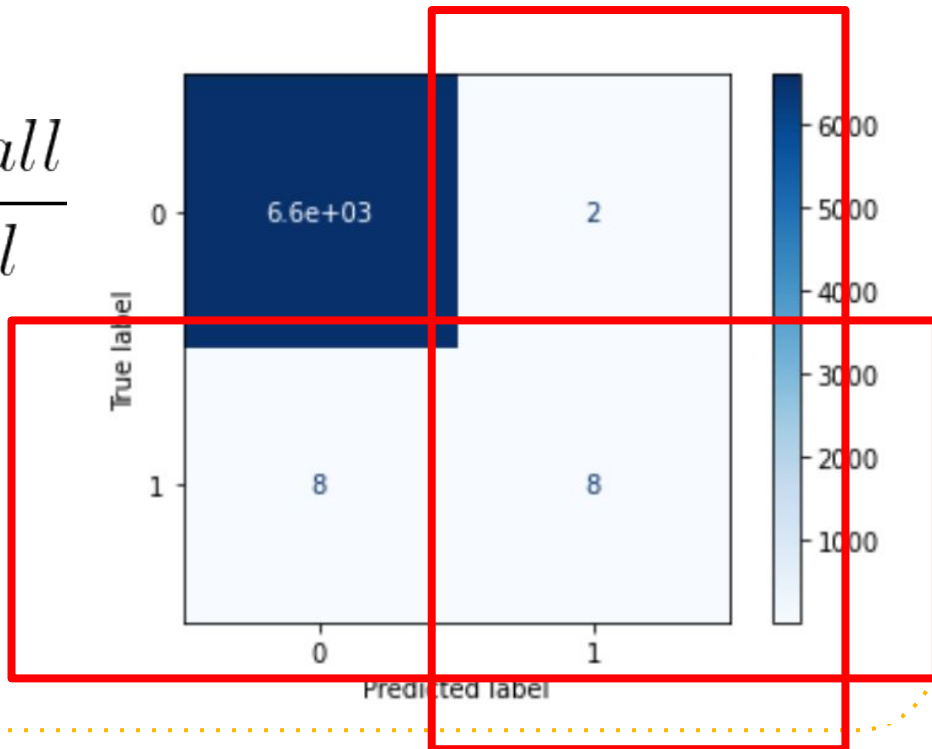
F1 Score(綜合考慮precision+Recall)

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

$$precision = 8 / (8 + 2) = 0.8$$

$$recall = 8 / (8 + 8) = 0.5$$

$$F1 = (2 * 0.8 * 0.5) / 1.3 = 0.615$$



為什麼不直接Precision Recall 平均就好？

F1 結果範圍0~1之間

假設p和r一個是1.0一個是0.1,算術平均會接近0.5 而調和平均接近0.2,這說明調和平均會強調兩者的一致性,明顯不一致時傾向於小的值,這更符合人們的直觀感受。

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

$$precision = 8 / (8 + 2) = 0.8$$

$$recall = 8 / (8 + 8) = 0.5$$

$$F1 = (2 * 0.8 * 0.5) / 1.3 = 0.615$$

F1 score特性 (希望平均都高)

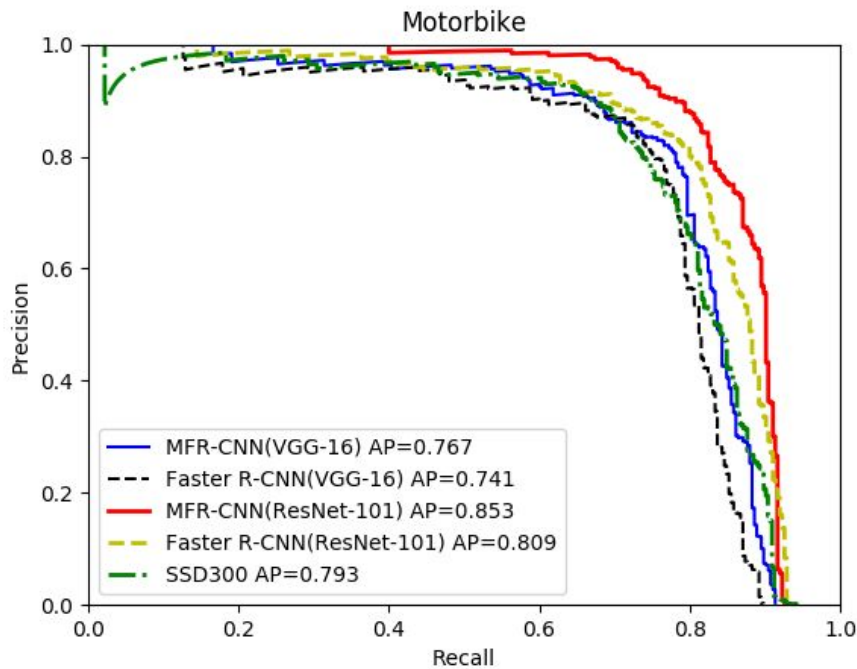
$$F1 = \frac{2 * precision * recall}{precision + recall}$$

兩個case:

1. precision =1 recall = 0.1 -> F1=0.18
2. precision =0.6 recall = 0.5 -> F1=0.55

PR 曲線

Threshold: 0~1之間

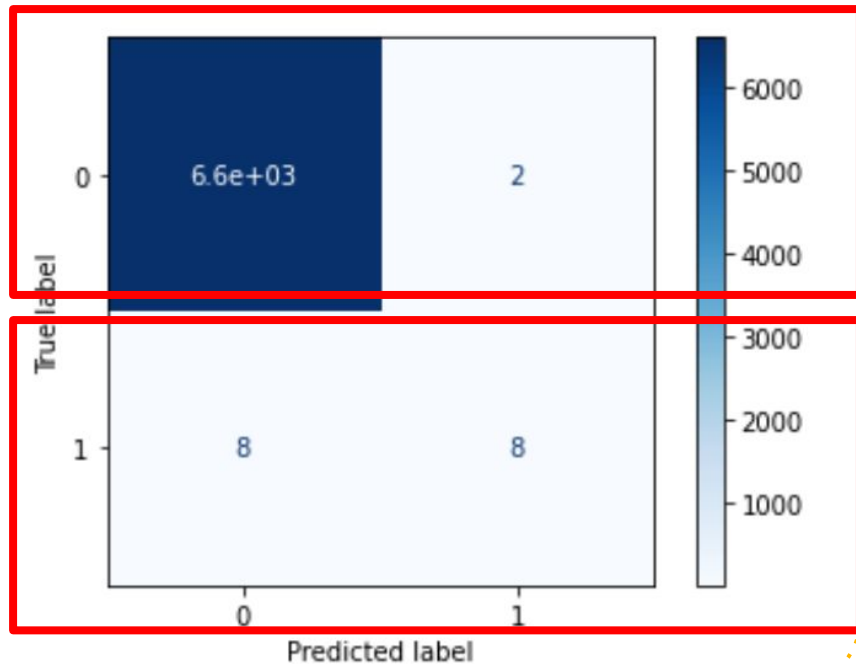


樣本1 predict probability:0.1
樣本2 predict probability:0.5
樣本3 predict probability:0.8

ROC(AUC) 曲線

$$\text{FalsePositiveRate}(FPR) = \frac{FP}{N}$$

$$\text{TruePositiveRate}(TPR) = \frac{TP}{P}$$



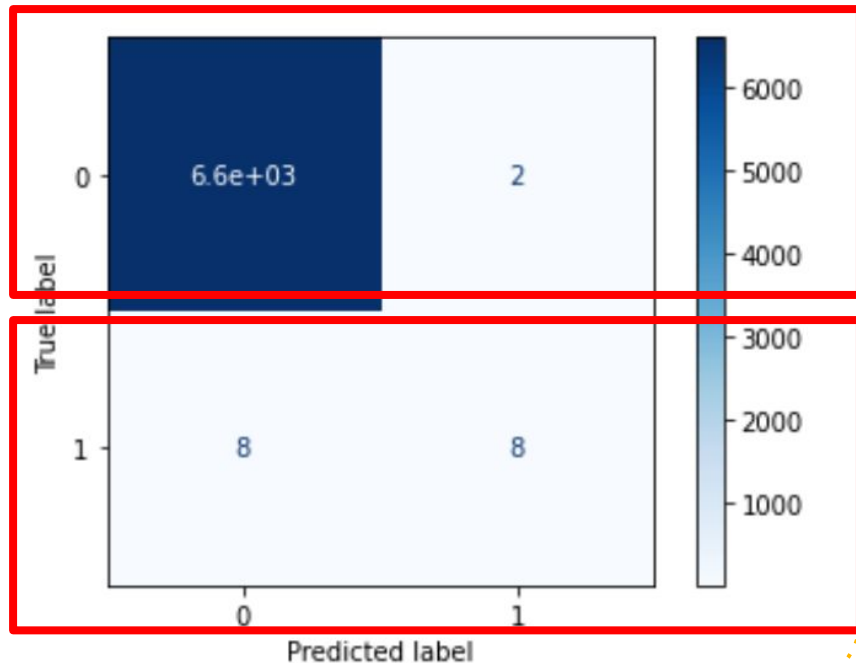
ROC(AUC) 曲線

[6599, 2]
[8, 8]

Threshold = 0.5

FPR = $8/6599 = 0.001$

TPR = $8/16 = 0.5$



ROC(AUC) 曲線

```
fpr, tpr, thresholds = roc_curve(y_test, lr.predict_proba(X_test)[: ,1], pos_label=1)
np.around(fpr, decimals=3)
np.around(tpr, decimals=3)
np.around(thresholds, decimals=3)
```

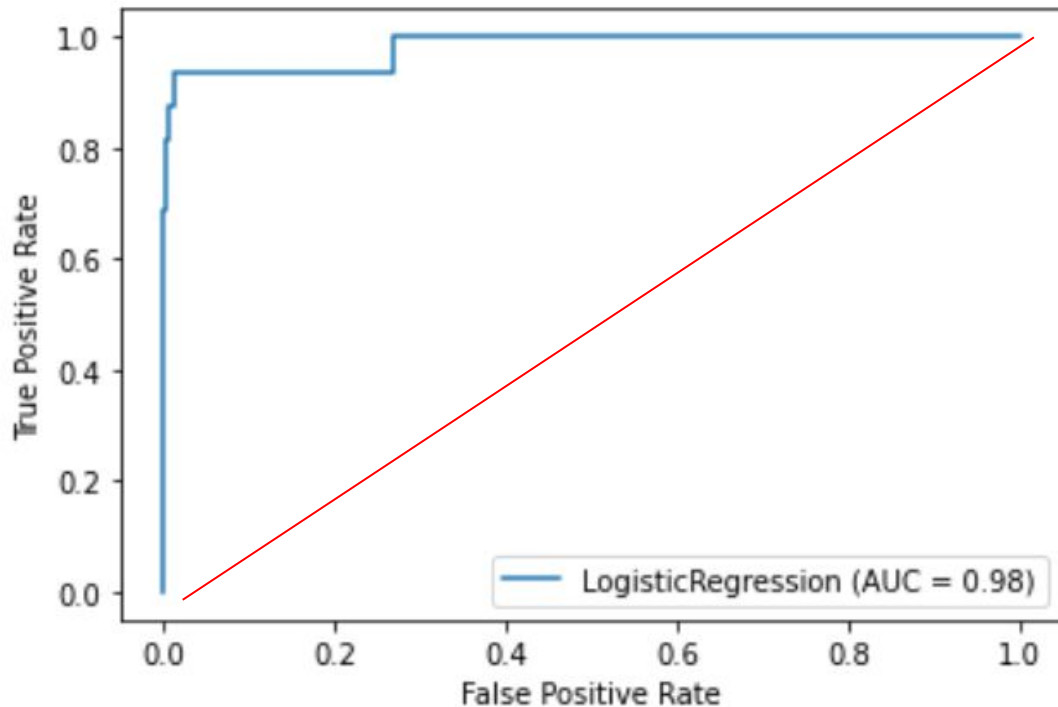
```
array([0.    , 0.    , 0.    , 0.    , 0.    , 0.001, 0.001, 0.002, 0.002,
        0.003, 0.003, 0.006, 0.006, 0.012, 0.012, 0.268, 0.268, 1.    ])

array([0.    , 0.062, 0.5    , 0.5    , 0.625, 0.625, 0.688, 0.688, 0.75 ,
        0.75 , 0.812, 0.812, 0.875, 0.875, 0.938, 0.938, 1.    , 1.    ])

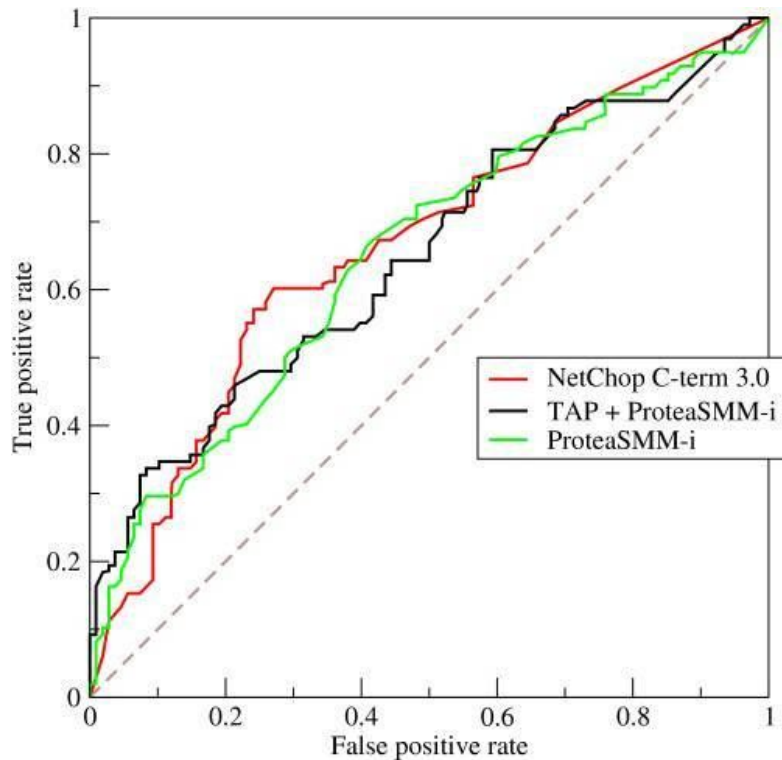
array([1.998, 0.998, 0.611, 0.504, 0.313, 0.212, 0.183, 0.106, 0.098,
        0.065, 0.06 , 0.03 , 0.03 , 0.012, 0.012, 0.    , 0.    , 0.    ])
```


ROC(AUC) 曲線

ROC值在0.5~1之間
(越大越好)



ROC(AUC) 曲線



多元分類指標

概念跟二元分類一樣，
改成當下這個類別vs 其他類別做比較

```
: print(classification_report(lr.predict(X_test), y_test))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6607
1	0.50	0.80	0.62	10
accuracy			1.00	6617
macro avg	0.75	0.90	0.81	6617
weighted avg	1.00	1.00	1.00	6617

回歸類模型指標

MAE (mean absolute error)

$$MAE = \frac{1}{n} \sum_{k=1}^n |(actual_1 - predicted_1)| + \dots + |(actual_n - predicted_n)|$$

RMSE (root mean squared error)

$$RMSE = \sqrt{\frac{\sum_{k=1}^n (actual_1 - predicted_1)^2 + \dots + (actual_n - predicted_n)^2}{n}}$$

回歸類模型指標

MAE (mean absolute error)

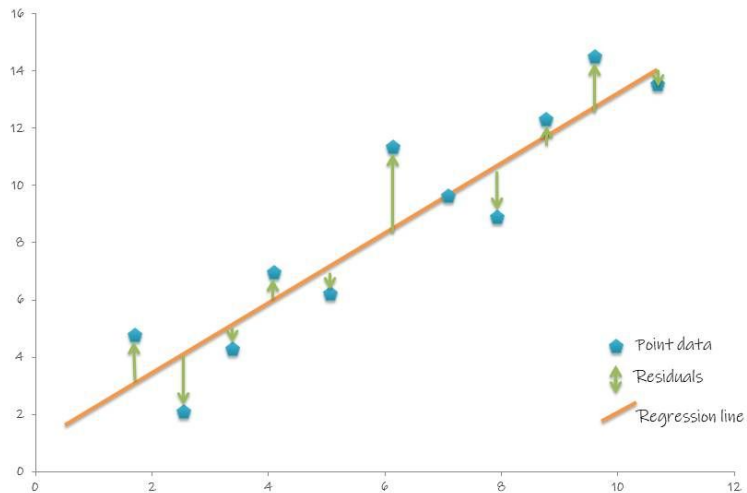
$$MAE = \frac{1}{n} \sum_{k=1}^n |(actual_1 - predicted_1)| + \dots + |(actual_n - predicted_n)|$$

RMSE (root mean squared error)

$$RMSE = \sqrt{\frac{\sum_{k=1}^n (actual_1 - predicted_1)^2 + \dots + (actual_n - predicted_n)^2}{n}}$$

有少數差異很大的 outlier 會造成 RMSE 變大, 誤差會被平方放大

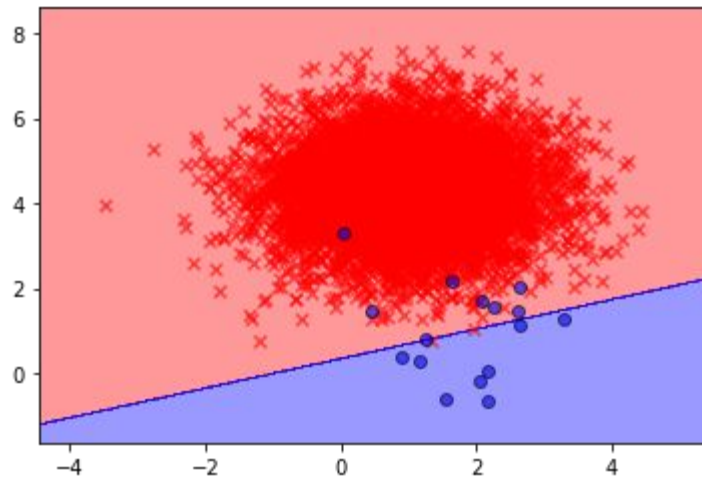
回歸類模型指標



樹模型

可解釋性的最佳模型

解釋性



模型的決策邊界: $-100 + 6 \times \text{溫度} + 3 \times \text{濕度} = 0$

$-100 + 6 \times \text{溫度} + 3 \times \text{濕度} > 0$ 預測是一個美味的披薩

$-100 + 6 \times \text{溫度} + 3 \times \text{濕度} < 0$ 預測是一個難吃的披薩

解釋性

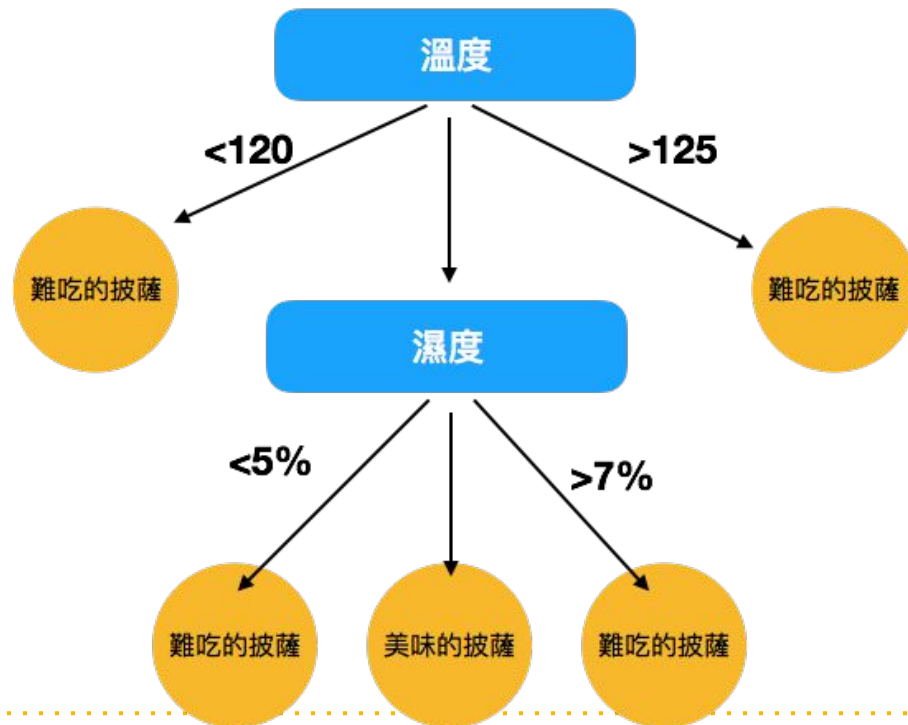
這只是一個最簡單的模型，實際上的模型會複雜超級多，像是：

$$-100 + 6 * \text{溫度}^2 + 3 * \text{濕度}^3 + 20 * \text{溫度} * \text{濕度}^2 + -70 * \text{溫度} * \text{濕度}^2 * \text{氣壓} + \dots$$

解釋性

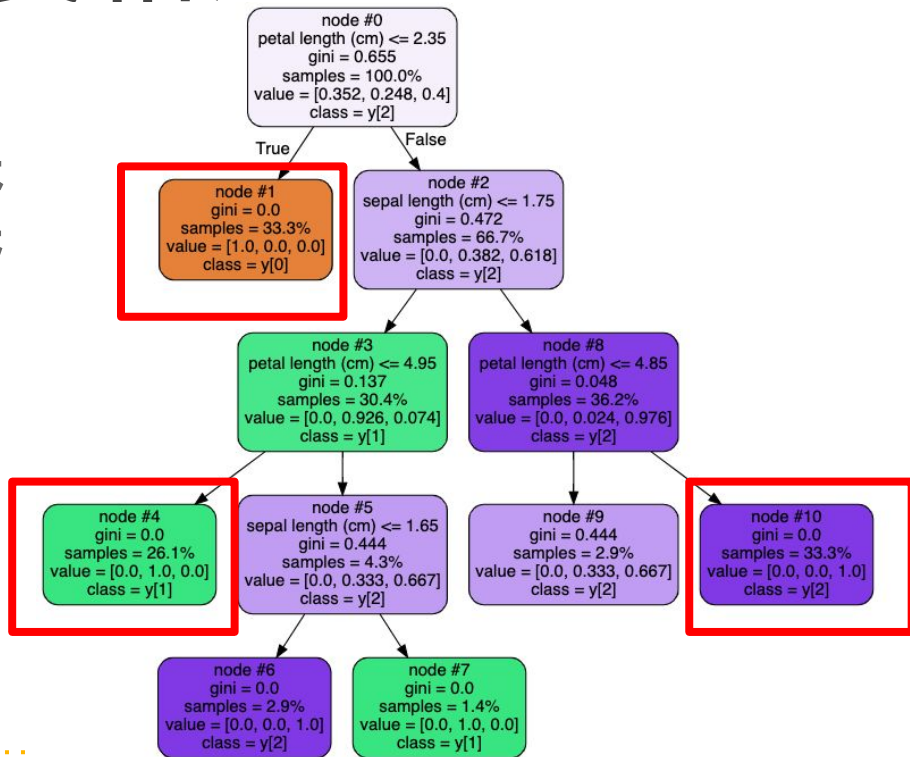


決策樹(抽象概念)



決策樹(實際長相)

- 分支通常是2
- 根據Gini, Entropy(結果基本一樣)來做分類問題的分支決策



決策樹切割原理

資訊增益

不確定性

不確定性

獲得的資訊量 原本的資訊量 經由分割後的資訊量

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^m \frac{N_j}{N_p} I(D_j)$$

二元分類資訊增益

不確定性

不確定性

不確定性

獲得的資訊量 原本的資訊量 分割後左邊資訊量 分割後右邊資訊量

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

細節請參考: [\[資料分析&機器學習\] 第3.5講：決策樹\(Decision Tree\)以及隨機森林\(Random Forest\)介紹](#)

決策樹切割原理(Entropy)

Entropy(熵) 熱力學第二定律 熵增原理: 在孤立系統中, 體系與環境沒有能量交換, 體系總是自發的向混亂度增大的方向變化, 使整個系統的 熵值越來越大。

簡單來說就是: 描述亂度 -> 在資訊領域描述不確定性



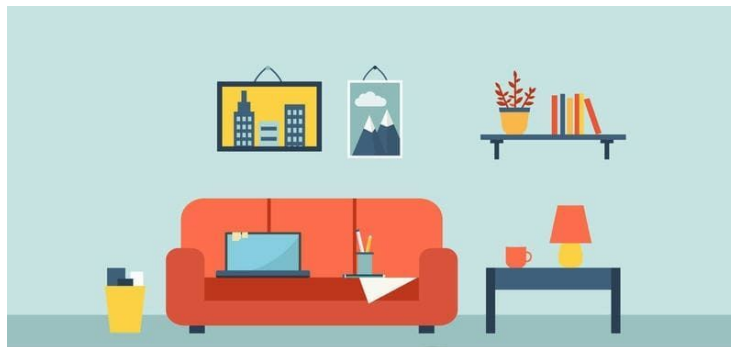
知乎 @马同学

Source:

<https://www.zhihu.com/question/24053383>

決策樹切割原理(Entropy)

房間越來越亂是熱力學第二定律之下的必然結果



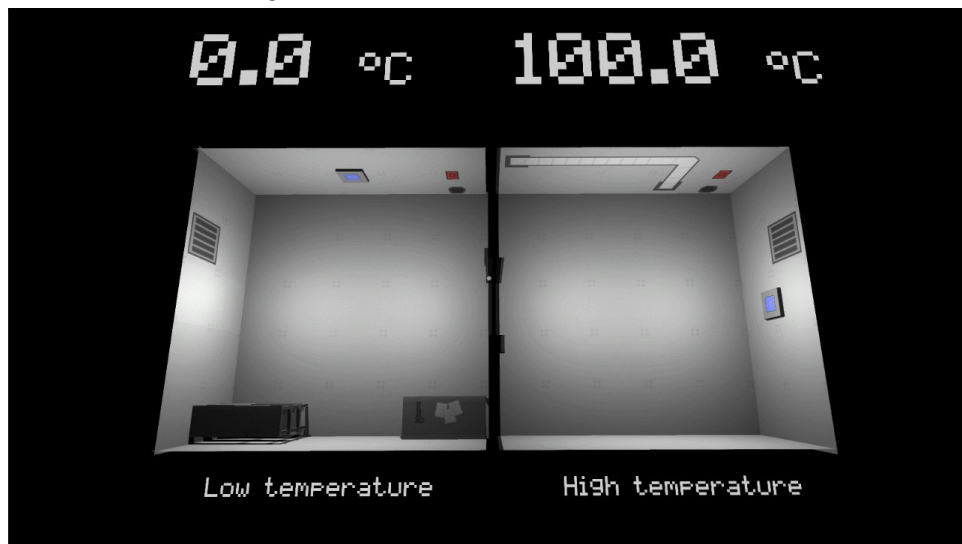
Source:

<https://www.zhihu.com/question/24053383>

決策樹切割原理(Entropy)

兩個房間溫度 0, 100 Entropy最小

兩個房間溫度 1 樣 Entropy最大(越自然的狀態熵越大)



Source:

<https://www.zhihu.com/question/24053383>

決策樹切割原理

資訊增益

不確定性

不確定性

獲得的資訊量 原本的資訊量 經由分割後的資訊量

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^m \frac{N_j}{N_p} I(D_j)$$

二元分類資訊增益

不確定性

不確定性

不確定性

獲得的資訊量 原本的資訊量 分割後左邊資訊量 分割後右邊資訊量

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

細節請參考: [\[資料分析&機器學習\] 第3.5講：決策樹\(Decision Tree\)以及隨機森林\(Random Forest\)介紹](#)

決策樹切割原理(Entropy)



決策樹切割原理(Entropy)

$$I_H(D_p) = -(0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = 1$$

$$A: I_H(D_{left}) = -\left(\frac{3}{4} \log_2\left(\frac{3}{4}\right) + \frac{1}{4} \log_2\left(\frac{1}{4}\right)\right) = 0.81$$

$$A: I_H(D_{right}) = -\left(\frac{1}{4} \log_2\left(\frac{1}{4}\right) + \frac{3}{4} \log_2\left(\frac{3}{4}\right)\right) = 0.81$$

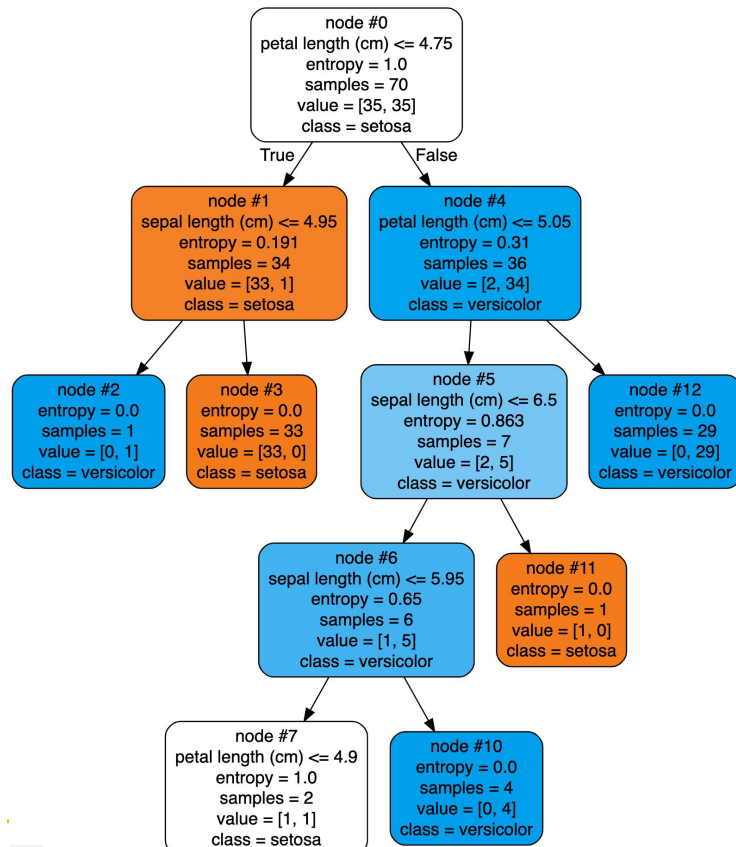
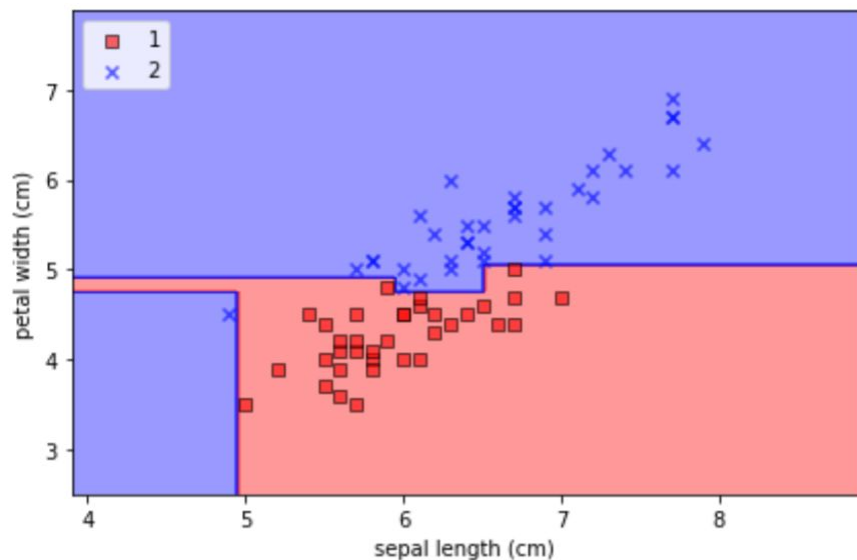
$$A: IG_H = 1 - \frac{4}{8} \cdot 0.81 - \frac{4}{8} \cdot 0.81 = 0.19$$

$$B: I_H(D_{left}) = -\left(\frac{2}{6} \log_2\left(\frac{2}{6}\right) + \frac{4}{6} \log_2\left(\frac{4}{6}\right)\right) = 0.92$$

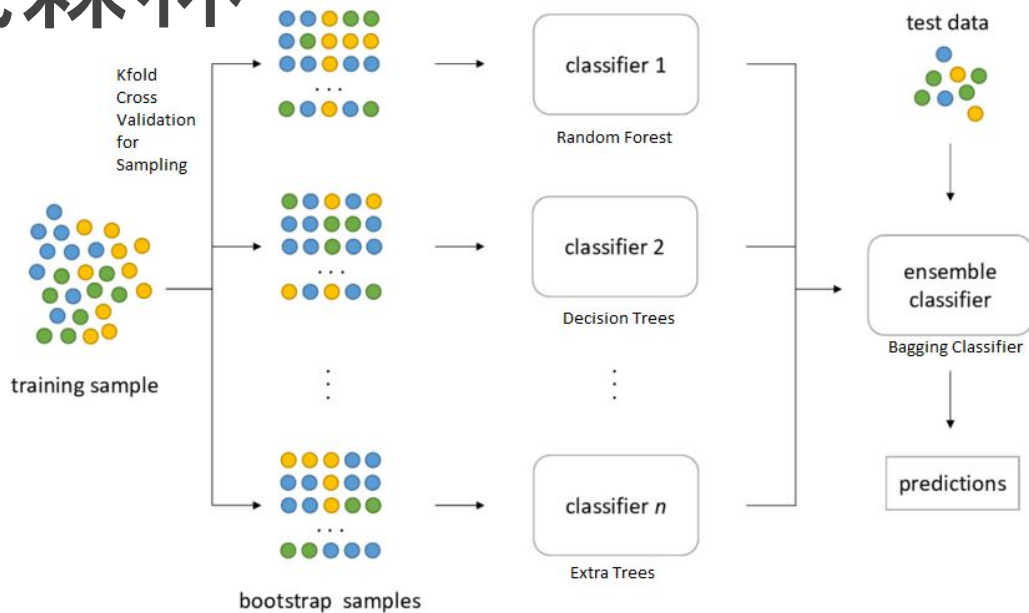
$$B: I_H(D_{right}) = 0$$

$$B: IG_H = 1 - \frac{6}{8} \cdot 0.92 - 0 = 0.31$$

決策樹Demo



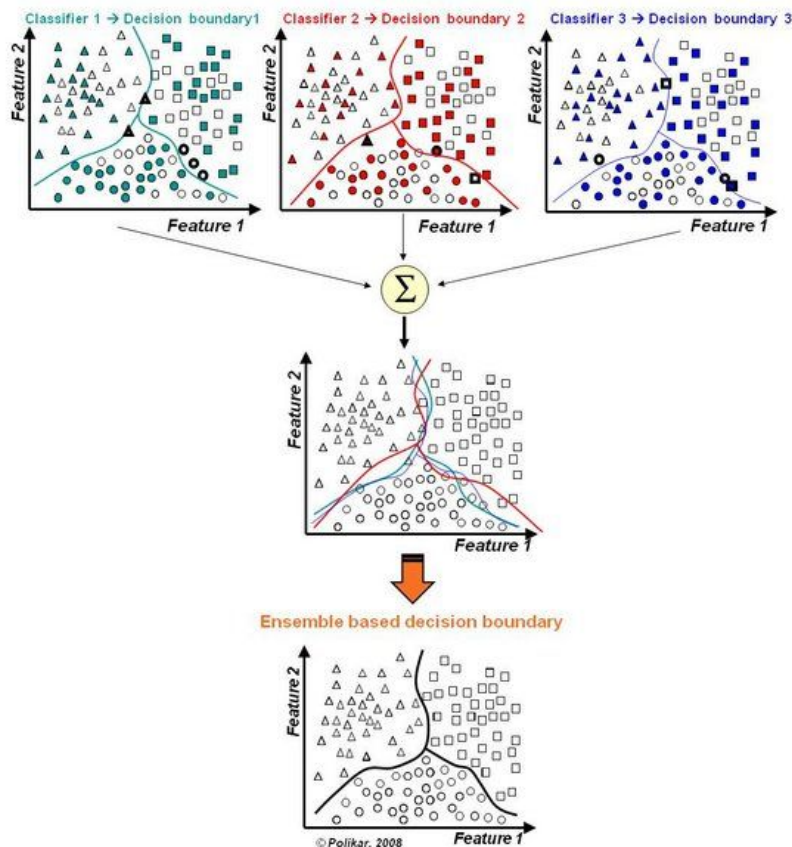
隨機森林



Bagging Classifier Process Flow

每棵樹只用到部分資料、Features

隨機森林



隨機森林 Demo

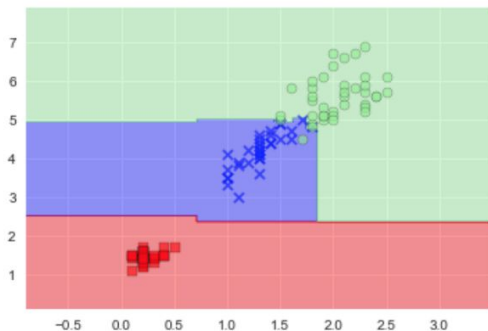
```
from sklearn.ensemble import RandomForestClassifier
```

```
forest = RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=3, n_jobs=2)
```

```
forest.fit(X_train, y_train['target'].values)
```

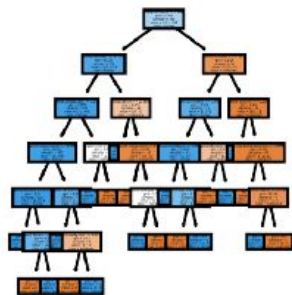
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',  
                        max_depth=None, max_features='auto', max_leaf_nodes=None,  
                        min_impurity_split=1e-07, min_samples_leaf=1,  
                        min_samples_split=2, min_weight_fraction_leaf=0.0,  
                        n_estimators=10, n_jobs=2, oob_score=False, random_state=3,  
                        verbose=0, warm_start=False)
```

```
plot_decision_regions(X_train.values, y_train['target'].values, classifier=forest)
```

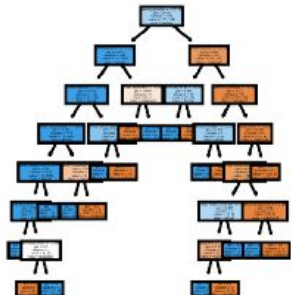


更多視覺化樹技巧

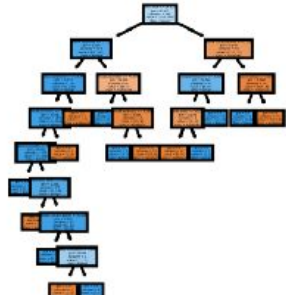
Estimator: 0



Estimator: 1



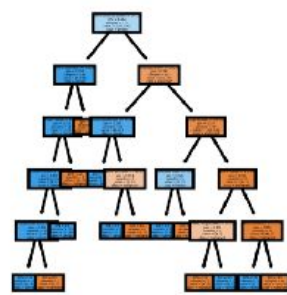
Estimator: 2



Estimator: 3



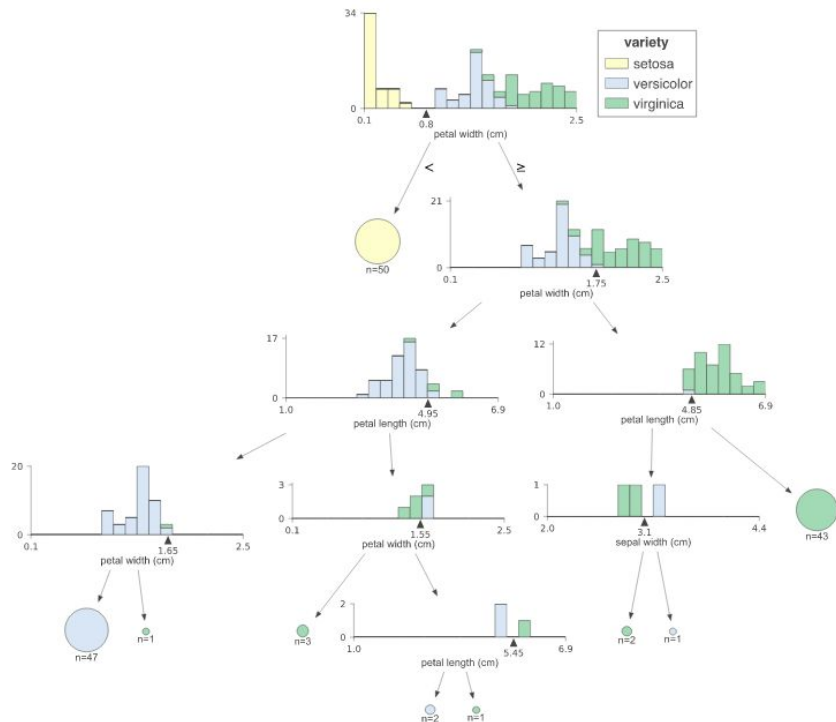
Estimator: 4



<https://scikit-learn.org/stable/modules/tree.html#tree>

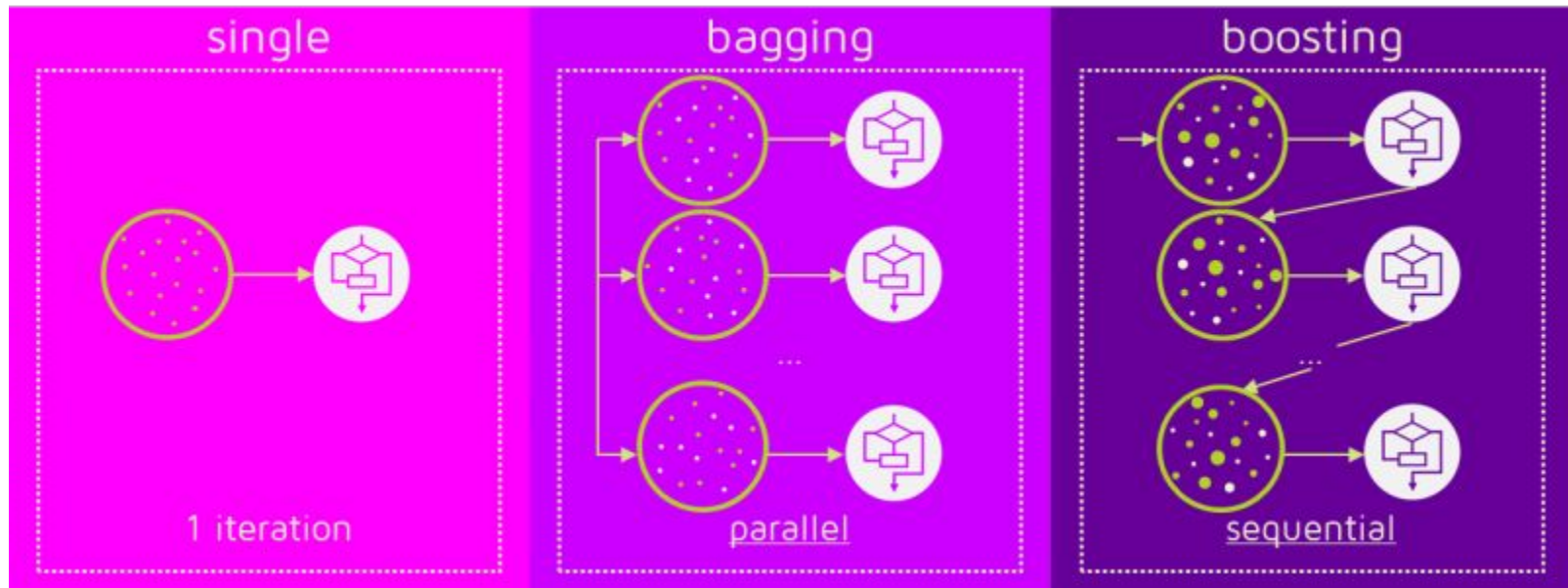
<https://towardsdatascience.com/visualizing-decision-trees-with-python-scikit-learn-graphviz-matplotlib-1c50b4aa68dc>

更多視覺化樹技巧



<https://explained.ai/decision-tree-viz/>

梯度提升決策樹(GBDT)



數學原理: <https://zhuanlan.zhihu.com/p/91652813>

梯度提升決策樹(GBDT)

概念：

假如有個人30歲，我們首先用20歲去擬合，發現損失有10歲，這時我們用6歲去擬合剩下的損失，發現差距還有4歲，第三輪我們用3歲擬合剩下的差距，差距就只有一歲了。

數學原理：<https://zhuanlan.zhihu.com/p/91652813>

隨機森林 vs 梯度提升決策樹(GBDT)

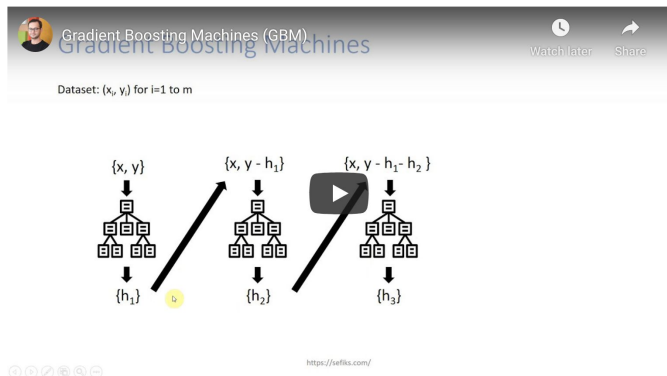
期末考時(可以分組考試的情況下):

1. 隨機森林: 每人分配部分章節同時去唸
2. 梯度提升: 每人輪流念全部內容, 並把不懂或是做錯的部分標記起來給下個人

梯度提升決策樹(GBDT)

Objective

The following video covers the idea behind GBM. Then, we will mention GBM with a step by step example.



We actually apply boosting in real world!

This is very similar to [baby step, giant step](#) method. We initially create a decision tree for the raw data set. That would be the giant step. Then, it is time to tune and boost. We will create new decision tree based on previous tree's error. We will apply this approach several times. These would be baby steps. [Terence Parr described](#) this process wonderfully in golf playing scenario as illustrated below.

[詳細數學推導過程](#)

梯度提升決策樹(GBDT)

Demo

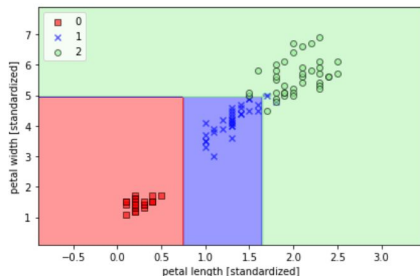
```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gbdt = GradientBoostingClassifier(n_estimators=10, random_state=3)
```

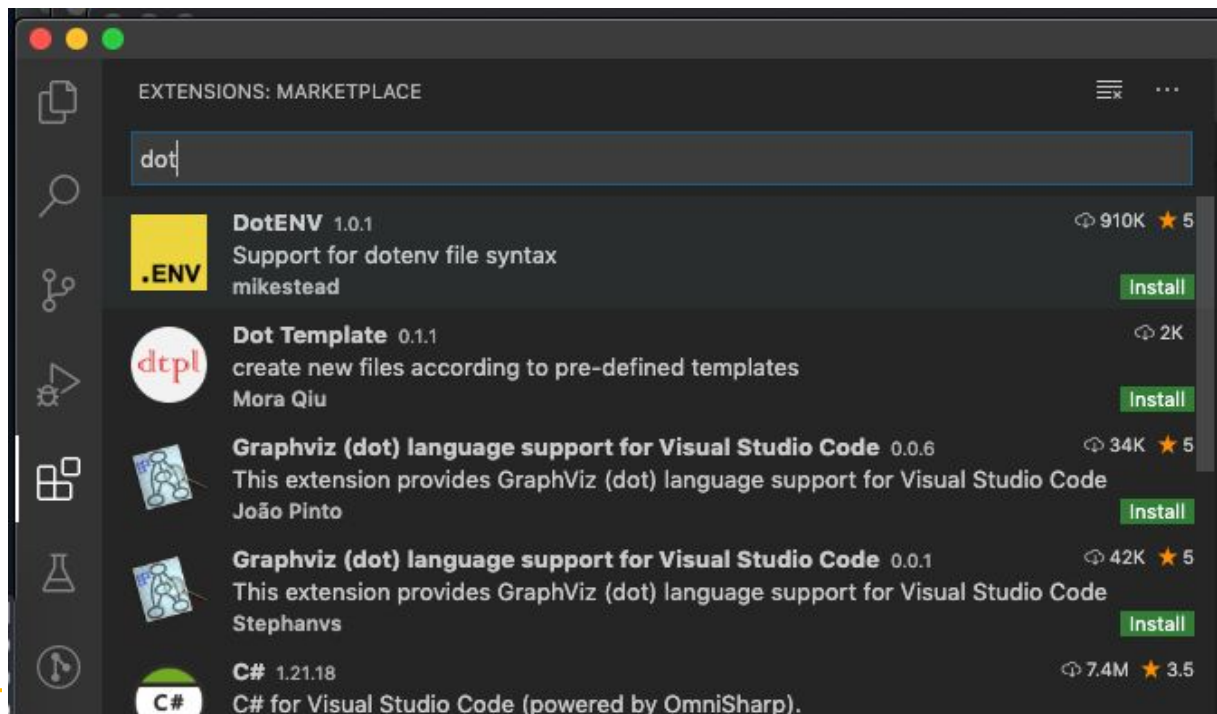
```
gbdt.fit(X_train, y_train['target'].values)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,  
                           learning_rate=0.1, loss='deviance', max_depth=3,  
                           max_features=None, max_leaf_nodes=None,  
                           min_impurity_decrease=0.0, min_impurity_split=None,  
                           min_samples_leaf=1, min_samples_split=2,  
                           min_weight_fraction_leaf=0.0, n_estimators=10,  
                           n_iter_no_change=None, presort='deprecated',  
                           random_state=3, subsample=1.0, tol=0.0001,  
                           validation_fraction=0.1, verbose=0,  
                           warm_start=False)
```

```
plot_decision_regions(X_train.values, y_train['target'].values, classifier=gbdt)  
plt.xlabel('petal length [standardized]')  
plt.ylabel('petal width [standardized]')  
plt.legend(loc='upper left')  
plt.tight_layout()  
plt.show()
```



使用vscode看樹的結構



使用vscode看樹的結構

