分類法

Yi-ting Chiang

中原大學應用數學系

April 25, 2023

Supervised Learning: Classification

Classification

接下來要介紹的是分類(classification)法。分類法在機器學習領域是用來判定此資料是屬於某一種類型的方法。例如人臉辨識，垃圾郵件判斷等等。與Regression不同，輸入資料，輸出是這個資料的種類。

與Regression一樣，分類法也是屬於監督式學習，需要的資料是一組有標記好其對應類別為何的資料，由這些資料來建構往後可以用來判斷未知的新資料的模型。

Supervised Learning: Classification

要介紹的第一個分類法是KNN (K-nearest neighbor)

K-Nearest neighbor

KNN是一個在機器學習方法中常見的模型。基本的KNN訓練模型步驟很簡單，就是紀錄下所有資料，並決定一個k值即可[a]。由於這種記下所有資料來進行分類的性質，KNN被稱為是memory-based learning，或稱為instance-based learning。

訓練好之後，在使用訓練時未出現過的新資料測試(testing)KNN，或實際使用KNN進行辨識的時候，就去找出最接近新資料的k個已知的舊資料出來，根據這k個點屬於哪種，來決定新資料是屬於哪一種。

[a] k必須自己設定

K-Nearest neighbor(續)

由上頁敘述可知，KNN是利用k個最接近新資料的舊資料來"投票"決定新資料的種類。令$|\mathbb{D}|$表示已紀錄在dataset中的資料總筆數。如果$k = 1$，KNN所決定的decision boundary會在空間中形成$|\mathbb{D}|$個凸多邊形，每個多邊形中心是一個已看過的舊資料在空間中的位置。這樣的圖，稱為Voronoi diagram或Voronoi tessellation。

由於選擇$k = 1$會讓資料中的noise很容易影響KNN模型，因此一般來說會讓k至少是3以上，以降低被資料中的noise影響的機會。

Supervised Learning: K-nearest neighbor

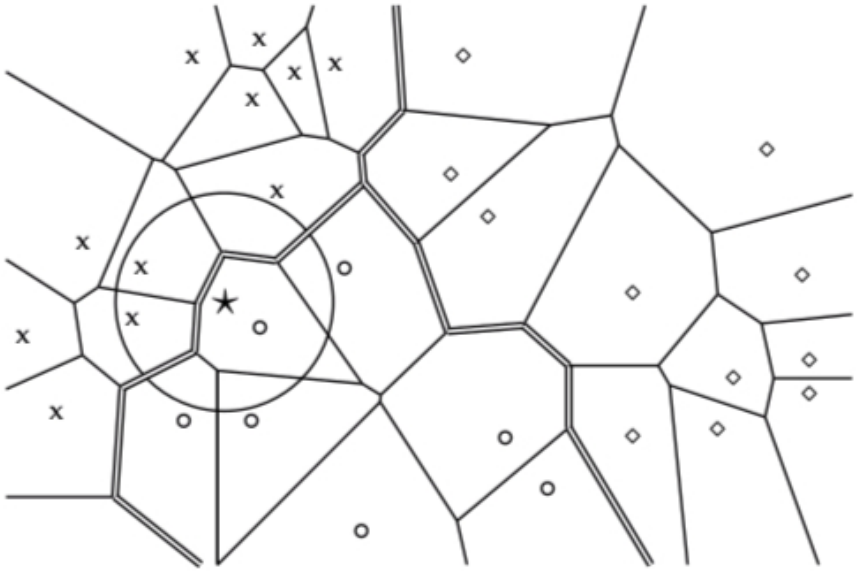


Figure: KNN在$k = 1$時，其decision boundary所形成的Voronoi diagram示意圖

K-Nearest neighbor(續)

KNN有些性質整理如下：

- KNN根據距離來判定新資料點，其有效性是成立在一個假設下：相同類型的資料會分佈在一起，形成一個連續的區域。且不同類型的資料之間所形成的區域之間沒有交集[a]
- KNN屬於一種非線性模型。意思是不需要假設資料可以在該維度被線性方程式(hyper plane)完整分開
- KNN可以不用訓練，只需要紀錄已知資料點即可
- KNN的效能不受資料可能的類別總數影響[b]
- 對於大量資料。KNN有必要將資料經過處理，以加快其testing/辨識的時候尋找最接近新資料點的舊資料的影響

[a]稱為contiguity hypothesis
[b]後面會介紹會被類別數量影響的方法

python範例：KNN模型

下面示範如何在python命令列下利用iris資料庫來建立並測試K-nearest neighbor模型。首先載入並切割資料:

```
>>> from sklearn import datasets
>>> from sklearn.model_selection import train_test_split
>>> iris_data=datasets.load_iris()
>>> x=iris_data.data
>>> y=iris_data.target
>>> x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1,shuffle=True)
>>> len(x_train)
120
>>> len(x_test)
30
```

上例中，從iris dataset取出feature部份(iris_data.data)和預測欄位(iris_data.target)，然後切成訓練與測試用資料集。由於共150筆資料，有0.2的部份拿去測試用，故訓練與資料集各為120和30筆。

python範例：KNN模型

要建立KNN模型，可以利用scikit-learn裡面內建的方法來做。會需要導入scikit-learn中的KNeighborsClassifier。導入以及使用的方法如下：

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn=KNeighborsClassifier(5)
>>> knn.fit(x_train,y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

關於上面的指令

- from sklearn.neighbors import KNeighborsClassifier：從scikit-learn導入KNN模型
- knn=KNeighborsClassifier(5)：指定要建立一個K=5的KNN模型。也可以用knn=KNeighborsClassifier(n_neighbors=5)指定變數名稱來設定數值
- knn.fit(x_train,y_train)：x_train是訓練用資料的觀察值(features)， y_train是訓練用資料的答案(label)。這個指令會根據訓練資料以及上面指定的K值建立KNN模型

python範例：KNN模型

要使用已經訓練好的KNN模型，可以用predict來給模型新的輸入讓它進行預測。方法如下：

```
>>> x_test[0:5]
array([[5.5, 2.6, 4.4, 1.2],
       [6.6, 3. , 4.4, 1.4],
       [6.7, 2.5, 5.8, 1.8],
       [5.5, 2.3, 4. , 1.3],
       [6.4, 3.1, 5.5, 1.8]])
>>> knn.predict(x_test[0:5])
array([1, 1, 2, 1, 2])
```

上面的範例先列出要測試的資料內容，然後在下面把它輸入knn.predict()來預測。由於只有出入前5筆資料，結果只會拿到5個輸出，分別代表這個KNN對這5筆資料的預測。

python範例：KNN模型

把全部資料都拿去預測：

```
>>> knn.predict(x_test)
array([1, 1, 2, 1, 2, 2, 2, 0, 0, 1, 0, 2, 2, 0, 0, 1, 2, 1, 0, 2, 1, 2,
       1, 0, 2, 0, 1, 2, 1, 2])
>>> y_test
array([1, 1, 2, 1, 2, 2, 2, 0, 0, 1, 0, 2, 2, 0, 0, 1, 2, 1, 0, 2, 1, 2,
       1, 0, 1, 0, 1, 2, 1, 2])
```

上面第一個array是預測的結果，第二個array是資料庫中真正的答案。可以看到這個KNN模型的預測和真實資料幾乎一樣。要直接拿到準確度(accuracy)可以用下面的方法：

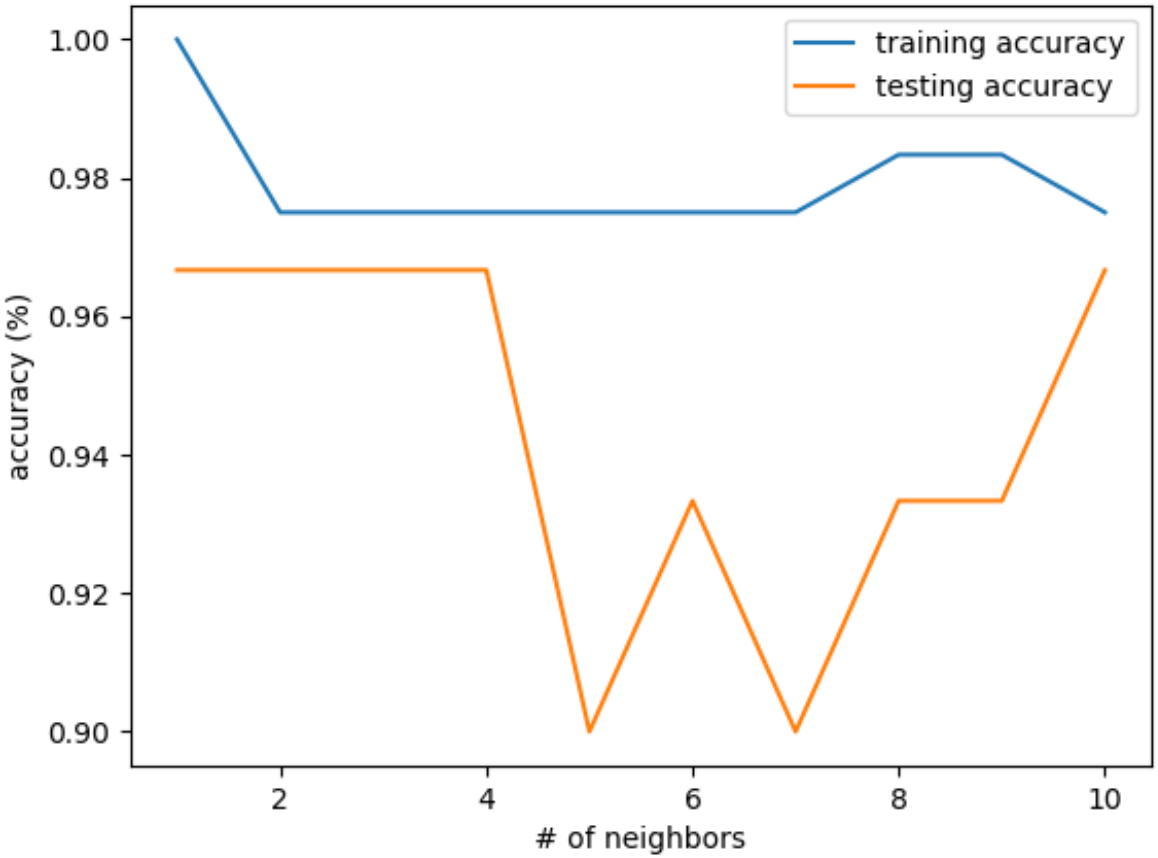
```
>>> knn.score(x_test,y_test)
0.966666666666666
```

python範例：KNN模型

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

iris_data=load_iris()
x=iris_data.data
y=iris_data.target
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=50)

train_accuracy=[]
test_accuracy=[]
k_range=range(1,11) # 測試k=1到10

for k in k_range:
  knn=KNeighborsClassifier(k)
  knn.fit(x_train,y_train)
  train_accuracy.append(knn.score(x_train,y_train))
  test_accuracy.append(knn.score(x_test,y_test))

plt.plot(k_range,train_accuracy,label="training accuracy")
plt.plot(k_range,test_accuracy,label="testing accuracy")
plt.xlabel("# of neighbors")
plt.ylabel("accuracy (%)")
plt.legend()
plt.savefig("knn—demo.png")
# 如果要展現在螢幕上，使用plt.show()
```

上面範例自動測試KNN的K值，範圍1到10，並將其結果以折線圖，輸出到knn-demo.png中。

python範例：測試KNN模型



上圖是在iris dataset上，利用上面的範例程式碼產生的實驗結果。

Supervised Learning: Decision Tree

下面要介紹的是決策樹。在介紹之前，先說明何謂樹(tree)

樹(tree)是由一些節點和邊組成的架構。若是每個節點之間相連，但不存在cycle[1]的話，它就是一個tree。

有些樹(例如決策樹)有定義根節點(root)。由根節點開始，每個和根節點相連的節點都為根節點的子節點(children)。然後每個根節點的子節點又有自己的子節點。以此類推，可以一直走到沒有任何子節點的節點。這些節點為樹的葉(leaves)節點。

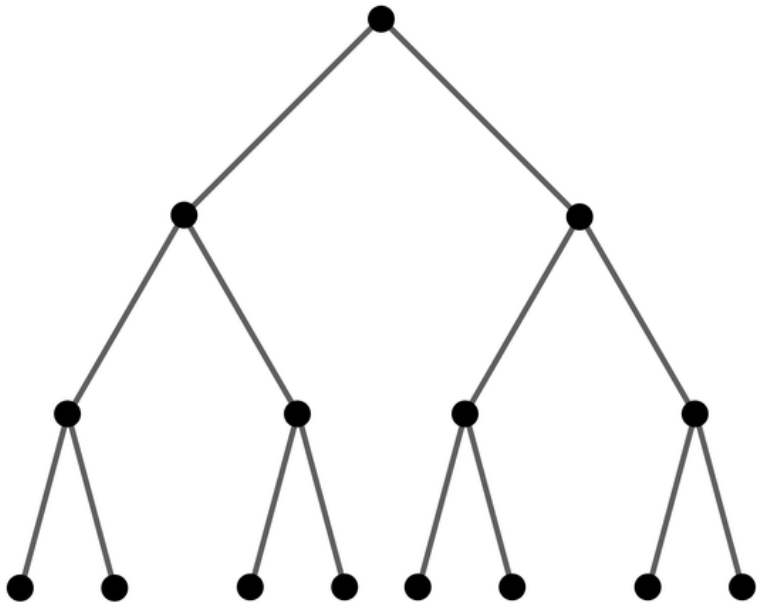


Figure: 樹的例子。此樹每個節點只有兩個子節點，稱為二元樹(binary tree)。

Supervised Learning: Decision Tree

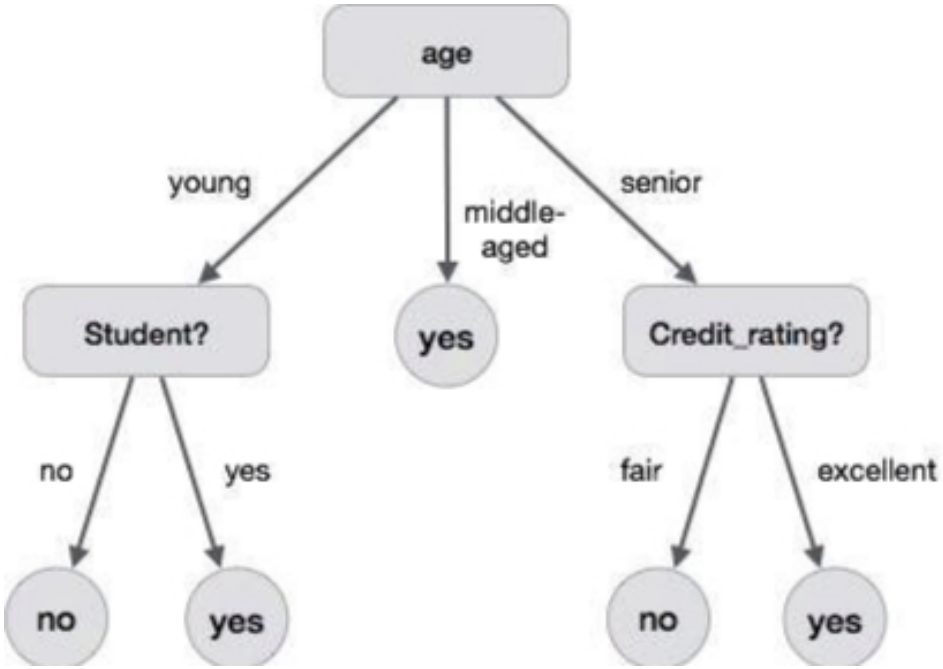


Figure: 一個決策樹的例子

在決策樹上，除了葉節點之外的每個節點都對應一個條件判斷。根據判斷結果，可決定走向這個節點的哪個子節點。用決策樹來做決策時，從根節點開始，判斷條件，往下一個節點前進。如此一直走，直到到達的此決策數樹某個葉節點

Supervised Learning: Decision Tree

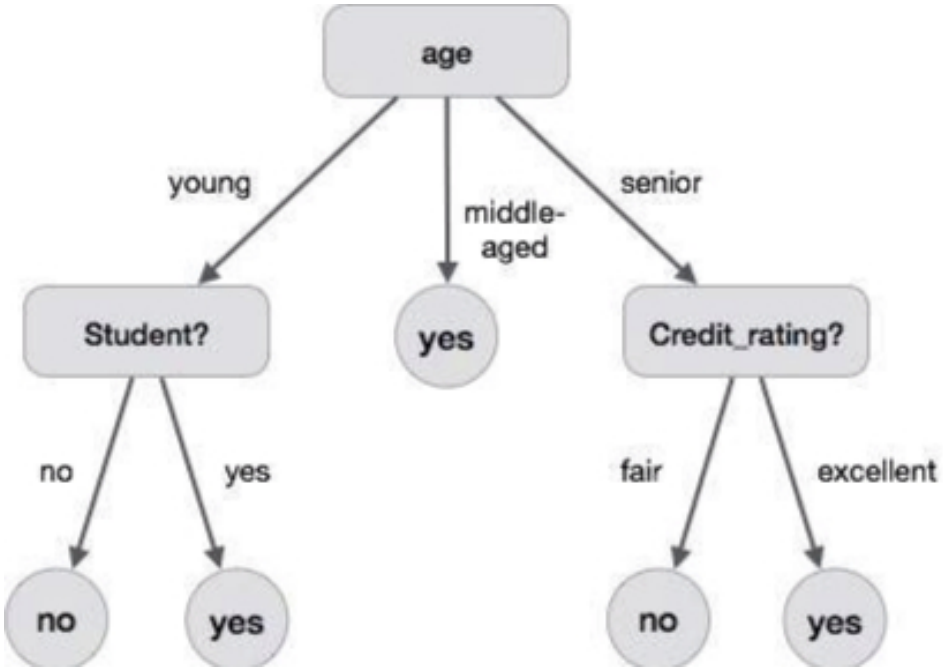


Figure: 一個決策樹的例子

觀察上面的決策樹，可以將節點分成三種：

- 根節點：決策的起點，要問的第一個問題。即上圖的“age”
- 葉節點：最後的分類結果。每個葉節點上只對應一種類別的資料。如上圖的兩個“yes”與“no”
- 中間節點：其他的節點。這時尚未完成資料分類

Supervised Learning: Decision Tree

Decision Tree

可以分析已收集的資料來建立決策樹，來幫助日後碰到未知的狀況時進行決策。假設目標是要建立一個理論上只需要問最少問題，就可以拿到答案的決策樹，亦即從根節點走到葉節點，中間所需經過的節點最少的樹。要決定的是，在決策樹的每個節點上，該針對觀察值X的哪個維度，做怎樣的條件判斷。

問題：這樣的決策樹，該如何建立？

Decision Tree(續)

上頁的概念，可以應用之前學過的entropy來實作。想法如下：

一開始在根節點，對於整個資料的不確定性是最大的，這時做一個條件判斷。如果這個條件判斷的答案可以讓整個資料的不確定性降低的程度最大[a]，當做完這個條件判斷，走到下一個節點之後，這時的資料分佈就是目前為止不確定性最低的結果。

在每個節點上要進行判斷時所的問題，應該要選擇可以讓整個資料分佈的entropy降低最多的問題。

重複上面的步驟。在下一個節點上，也選擇一個可以讓整個資料分佈entropy降低最多的問題作為此節點要做的判斷。直到最後一次問題，可以將所有種類不同的資料完全分開，製造出葉節點。

[a]這個entropy降低的狀況被稱為information gain

Supervised Learning: Decision Tree

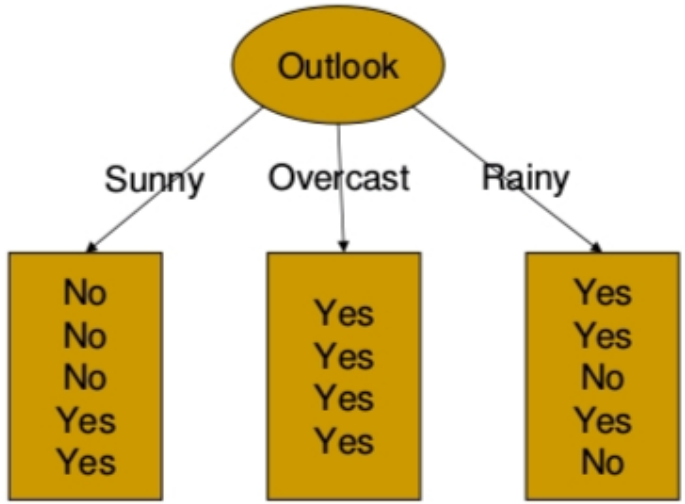
接下來是一個利用上述方法建立決策樹來進行分類的例子。下面用來建立decision tree的資料集。需要選擇一個attribute，也就是資料中的outlook, temp, humidity, 以及windy作為在每個節點上要用來將資料分開的依據。亦即，要選擇一個attribute來問問題，根據答案來分類。計算問了此問題之後，系統剩下的entropy。然後比較並選擇讓剩下的entropy最少的attribute。

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No

Supervised Learning: Decision Tree

以下是一個利用上述方法建立決策樹來進行分類的例子。下面是假設選擇Outlook進行決策，並計算系統的entropy

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No
0.693				

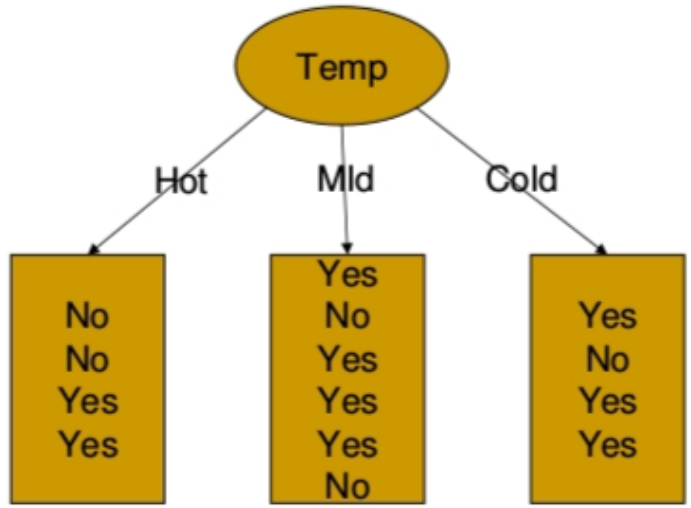


$$
\begin{aligned}
H(\text{Play}) &= -\frac{5}{14} \times \left(\frac{3}{5}\log\frac{3}{5} + \frac{2}{5}\log\frac{2}{5}\right) \\
&\quad -0 \\
&\quad -\frac{5}{14} \times \left(\frac{3}{5}\log\frac{3}{5} + \frac{2}{5}\log\frac{2}{5}\right) \\
&\approx 0.693
\end{aligned}
$$

Supervised Learning: Decision Tree

下面是選擇Temp進行決策後系統的entropy

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No
0.693	0.911			

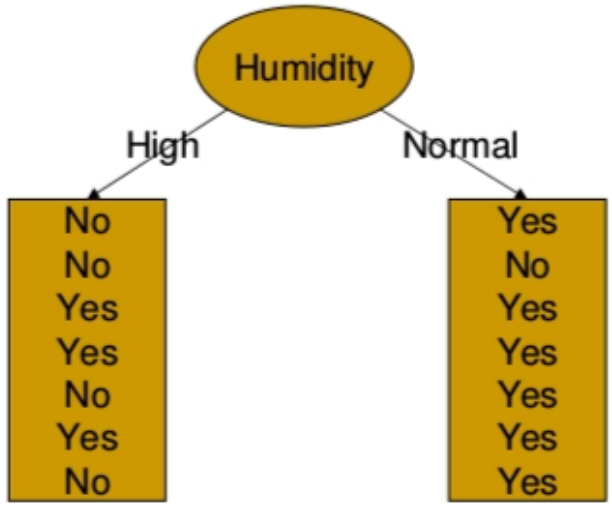


$$
\begin{aligned}
H(\text{Play}) &= -\frac{4}{14} \times \left(\frac{2}{4}\log\frac{2}{4} + \frac{2}{4}\log\frac{2}{4}\right) \\
&\quad -\frac{6}{14} \times \left(\frac{2}{6}\log\frac{2}{6} + \frac{4}{6}\log\frac{4}{6}\right) \\
&\quad -\frac{4}{14} \times \left(\frac{1}{4}\log\frac{1}{4} + \frac{3}{4}\log\frac{3}{4}\right) \\
&\approx 0.911
\end{aligned}
$$

Supervised Learning: Decision Tree

下面是選擇Humidity進行決策後系統的entropy

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No
0.693	0.911	0.788		

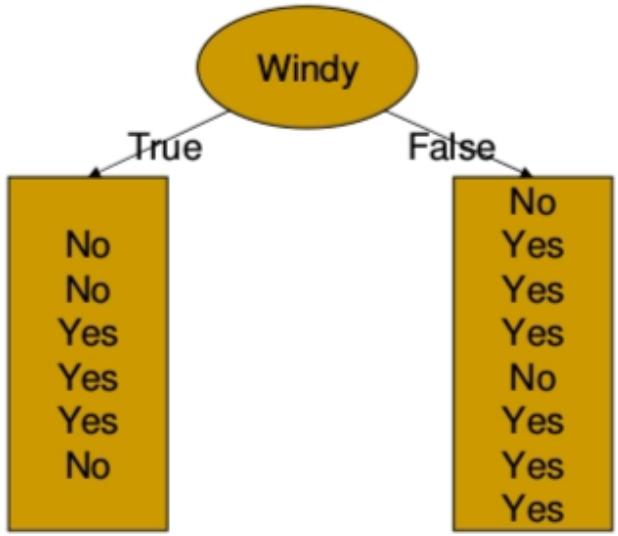


$$
\begin{aligned}
H(\text{Play}) &= -\frac{7}{14} \times \left(\frac{3}{7}\log\frac{3}{7} + \frac{4}{7}\log\frac{4}{7}\right) \\
&\quad -\frac{7}{14} \times \left(\frac{1}{7}\log\frac{1}{7} + \frac{6}{7}\log\frac{6}{7}\right) \\
&\approx 0.788
\end{aligned}
$$

Supervised Learning: Decision Tree

下面是選擇Windy進行決策後系統的entropy

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No
0.693	0.911	0.788	0.892	



$$
\begin{aligned}
H(\text{Play}) &= -\frac{6}{14} \times \left(\frac{3}{6}\log\frac{3}{6} + \frac{3}{6}\log\frac{3}{6}\right) \\
&\quad -\frac{8}{14} \times \left(\frac{2}{8}\log\frac{2}{8} + \frac{6}{8}\log\frac{6}{8}\right) \\
&\approx 0.892
\end{aligned}
$$

Supervised Learning: Decision Tree

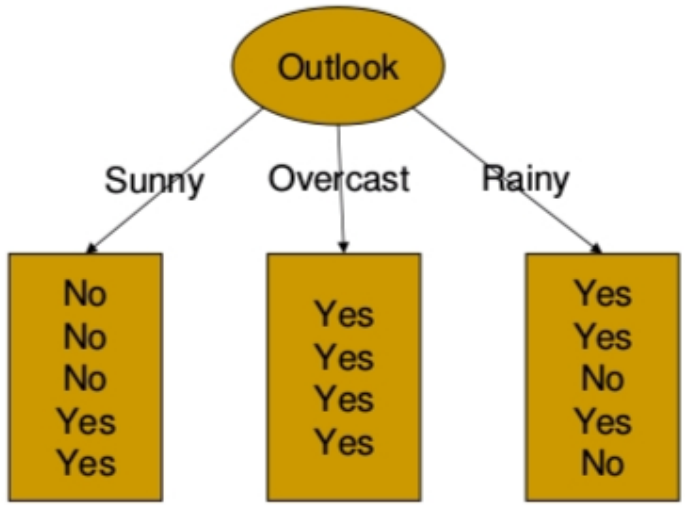
Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No
0.693	0.911	0.788	0.892	

結論：應該挑選可以在下決策之後系統entropy最低的outlook作為此節點的決策問題

Supervised Learning: Decision Tree

上面步驟完成了決定根節點應該選擇哪個決策的問題。經過根節點之後，資料會被分成三組，如下右圖。
接下來，除了已經只剩一種類型資料的節點（對應outlook=overcast）成為葉節點之外，剩下兩組形成新的節點，各自繼續上面步驟。直到相同種類資料只存在一組內為止。

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No
0.693	0.911	0.788	0.892	



Supervised Learning: Decision Tree

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No

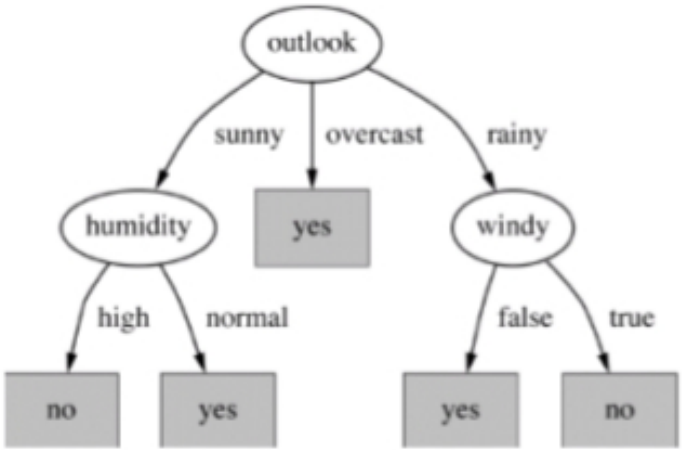


Figure: 根據左方資料，利用前述步驟最後產生的Decision Tree

Supervised Learning: Decision Tree

上面介紹的演算法稱為ID3演算法。由Ross Quinlan於1986年提出。這個演算法的目標是儘量建造較低深度的decision，亦即儘可能在問最少的問題的狀況下得到答案[2]。這個作法可以降低模型複雜度。但此演算法有下面的問題：

- 對於有noise的資料，會overfit
- 無法處理有missing data的資料
- 由於此演算法有選擇可以把資料分成比較多塊（亦即子節點數較多）的問題的傾向，如此反而增加模型複雜度
- 因上一個性質，此演算法不適合處理特徵為數值種類（尤其是實數）的資料

[2]雖然結果不保證真的會是最少的

Supervised Learning: C4.5 Decision Tree

針對上述問題，Ross Quinlan後來在1993年提出另一個也是基於Entropy (Information Gain)的改良版演算法，稱為C4.5。這是目前實作decision模型最普遍的方法。

- noise: C4.5中使用了一種稱為“剪支”(pruning)的方法，衡量是否要取消某些問題，將因此問題而分開的資料合在一起，降低模型複雜度
- missing data: C4.5訓練時只看無missing value的資料。測試或使用時，碰到有missing value的資料，以傳回分類的分佈取代傳回確切分類結果，或是根據分佈傳回機率最大者
- 分成較多組的傾向問題：以information gain ratio取代information gain，考慮分組結果會將資料分為幾種可能，對於分出許多組的結果給予懲罰，降低選擇多組結果的可能性。
- 數值種類資料: 將數值種類排序後，根據不同大小之間的資料作為決定如何根據此欄位分組的問題。

python範例：Decision Tree

下面示範如何在python下建立決策樹(decision tree)[3]。

要在python下使用decision tree，也可以使用scikit-learn。導入的指令如下：

```
from sklearn.tree import DecisionTreeClassifier
```

要建立與測試decision tree的方法與前面介紹的許多方法類似。如下例：

```
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
dt.predict(x_test)
dt.score(x_test,y_test)
```

以上指令，分別是初始化decision tree，根據資料(x_train,y_train)來訓練decision tree，拿到新的資料預測輸出，以及比對新資料的預測值與真實值，計算準確度。

建立decision tree時，可以指定某些參數來限制建立好的tree的大小。下面是一些例子：

- max_depth: 指定一個整數。限制tree的最大深度(預設為None，無限制)
- min_samples_split: 指定一個整數。繼續製造分支時節點上至少需要的資料量(預設=2)
- criterion: 指定衡量節點不確定性的方法。(預設為"gini"。可以改為"entropy")

[3]目前sklearn(版本0.24.0及之前)的決策樹僅支援數值型態資料。

python範例：Decision Tree

下面是一個利用iris dataset建立decision tree的例子：

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

iris_data=load_iris()
x=iris_data.data
y=iris_data.target
x_train,x_test,y_train,y_test=train_test_split(x,y,
test_size=0.2,random_state=50)

train_accuracy=[]
test_accuracy=[]
dt=DecisionTreeClassifier(criterion="entropy")
dt.fit(x_train,y_train)
print(f"Accuracy={dt.score(x_test,y_test)}")
```

輸出為：

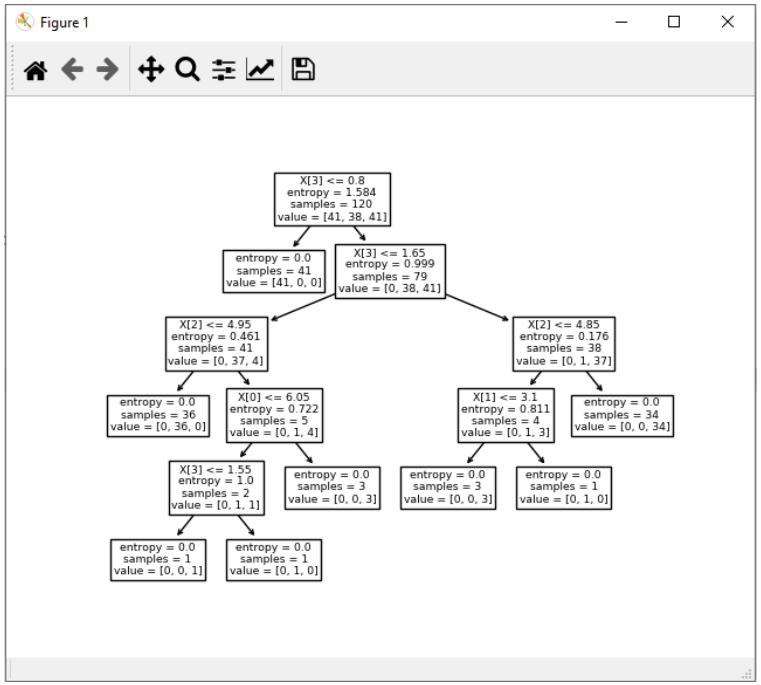
```
Accuracy=0.9666666666666667
```

python範例：Decision Tree

scikit-learn有提供可以將decision tree輸出為圖檔的功能，方法如下：[4]

```
from sklearn import tree
from matplotlib import pyplot as plt
tree.plot_tree(dt)
plt.show()
```

如此會有一個視窗，顯示出訓練好的decision tree模型dt。在此視窗上可選擇將檔案存為jpg, png, eps, pdf等許多種不同格式的檔案。



[4]需sklearn 0.23.0及之後版本

python範例：Decision Tree

也可以利用graphviz來處理已訓練好的edcision tree模型：

```
from sklearn.tree import export_graphviz
export_graphviz(dt,out_file="dt-demo.dot",feature_names=iris_data.feature_names,
class_names=iris_data.target_names)
```

上面dt是訓練好的decision tree模型；out_file指定一個字串作為檔名；feature_names和class_names是分別指定要顯示在結果上的feature名稱(這邊是花萼與花瓣的長和寬)與輸出的名稱(這邊是三種品種的iris)。

輸出的.dot檔，可以在windows或linux下安裝graphviz之後，利用dot程式轉換為png檔:

```
dot -Tpng dt-demo.dot -o dt-demo.png
```

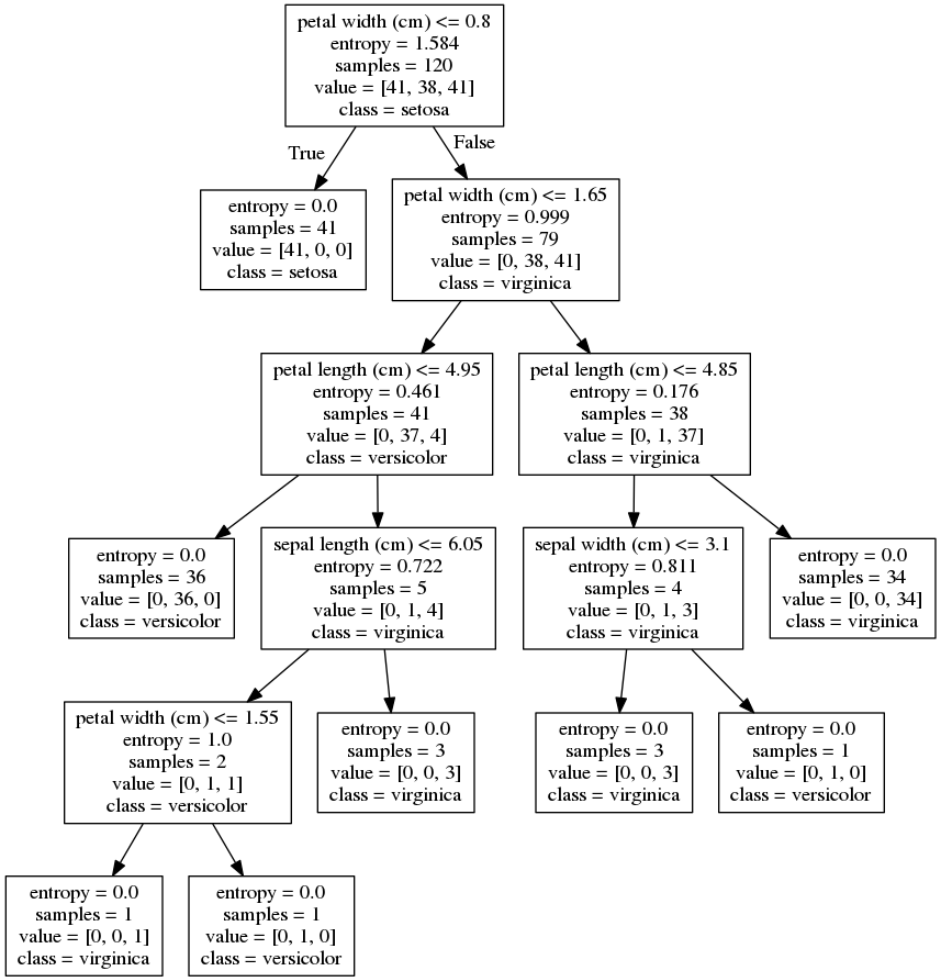
在產生出上述點dot檔之後, 日後要重新載入的話，可在python中導入graphviz套件：

```
import graphviz
f=open("dt-demo.dot")
dt=f.read()
f.close()
graphviz.Source(dt).render("dt_tree",view=True)
```

上述範例會取得一個名為dt_tree.pdf的檔案。

python範例：Decision Tree

下圖是上面利用iris dataset作為訓練資料產生的decision tree：

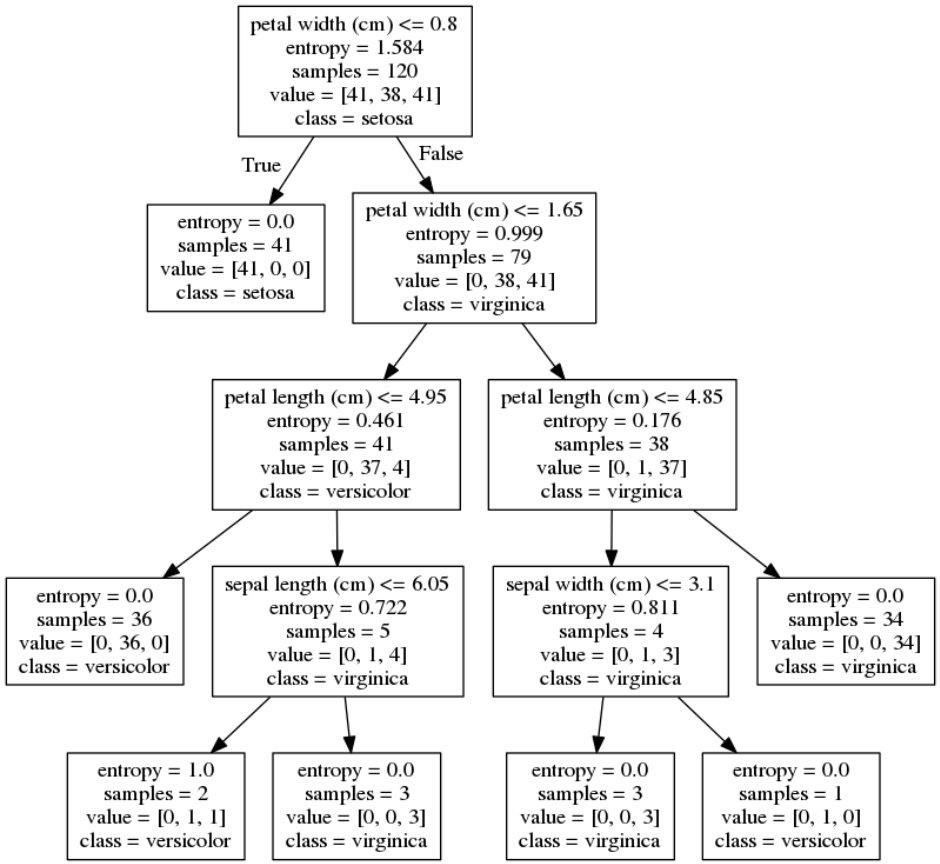


python範例：Decision Tree

從上圖可發現這個decision tree會把資料根據各類型完全分開，所以有很多層，而且有些葉節點上面只有一個資料。如果在設定時利用前面所提到的max_depth參數限制深度：

```
dt=DecisionTreeClassifier(random_state=1,criterion="entropy",max_depth=4)
```

得到的樹如下：



Supervised Learning: Naive Bayes

接下來要介紹的是另一個機器學習領域中很常見的模型，Naive Bayes。

關於Naive Bayes

Naive Bayes是基於Bayes rule的模型。給定兩個隨機變數X, Y，Bayes rule是

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

上面的式子可以根據條件機率與聯合機率函數性質導出來[a]

[a] $P(X|Y) = \frac{P(X,Y)}{P(Y)} = \frac{P(Y|X)P(X)}{P(Y)}$

Supervised Learning: Naive Bayes

可以用有向圖架構來表示Naive Bayes，如下圖。其中C代表資料屬於哪個種類，x_i代表一筆資料中的各個欄位

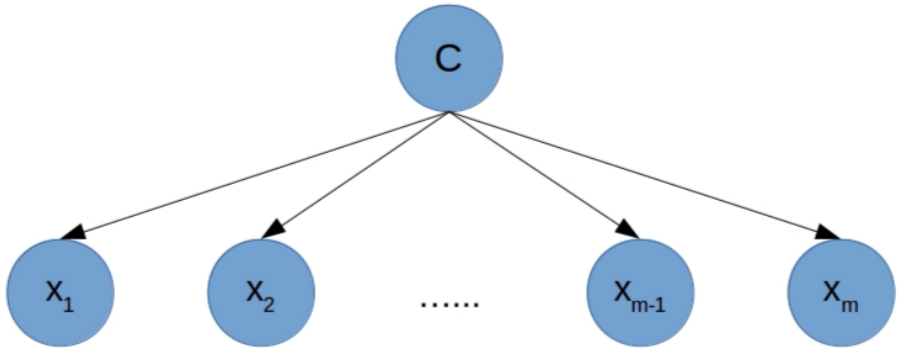


Figure: 將Naive Bayes以有向圖模型來表示

上面的有向圖架構事實上代表C和x_i的獨立與相依關係。那樣的架構表示給定C，各X_i之間互相獨立，也就是

$$P(x_1, x_2, \ldots, x_m|C) = P(x_1|C)P(x_2|C)\ldots P(x_m|C) = \prod_{i=1}^{m} P(x_i|C)$$

Supervised Learning: Naive Bayes

關於Naive Bayes(續)

由上頁，如果想要估計$P(C|x_1, x_2, \ldots, x_m)$，可以由下面的式子來計算：

$$
\begin{aligned}
P(C|x_1, x_2, \ldots, x_m) &= \frac{P(x_1, x_2, \ldots, x_m|C)P(C)}{P(x_1, x_2, \ldots, x_m)} && (1)\\
&\propto P(x_1, x_2, \ldots, x_m|C)P(C) && (2)\\
&= \prod_{i=1}^{m} P(x_i|C)P(C) && (3)
\end{aligned}
$$

其中式(1)為Bayes rule；式(1)到(2)是因為根據相同的觀察資料x_i來分類此資料，算式分母部份一樣，不影響大小比較；式(2)到(3)是前述的，給定C，各x_i後相獨立

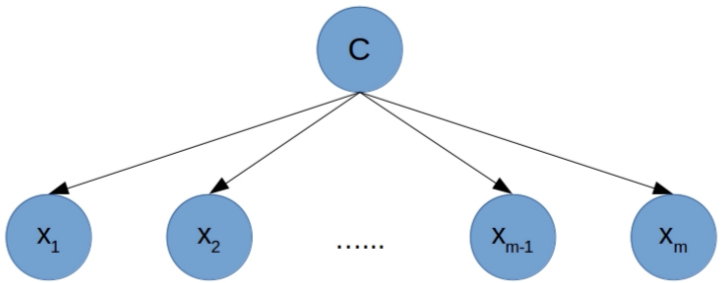


Figure: 將Naive Bayes以有向圖模型來表示

Supervised Learning: Naive Bayes

關於Naive Bayes(續)

Naive Bayes是基於

$$P(C|x_1, x_2, \ldots, x_m) \propto \prod_{i=1}^{m} P(x_i|C)P(C)$$

來判斷某個觀察結果是屬於某個類別的機率。在使用Naive Bayes時，需要

1. 從所收集的資料中去針對每個可能的種類C，計算其出現的機率$P(C)$
2. 對所有的欄位x_i，計算$P(x_i|C)$
3. 碰到新的觀察資料時，依據資料各欄位，對每個x_i及所有類別C，計算$P(C) \times P(x_i|C)$，找出最大的作為答案

Supervised Learning: Naive Bayes

範例：利用Naive Bayes判斷是否要出門去打高爾夫球

Outlook	Temp	Humidity	Windy	Play
Sunny	Cool	High	True	?

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No

Supervised Learning: Naive Bayes

Outlook	Temp	Humidity	Windy	Play
Sunny	Cool	High	True	?

$$
\begin{aligned}
& p(P = y | O = s, T = c, H = h, W = t) \\
= & \frac{p(O = s, T = c, H = h, W = t | P = y) p(P = y)}{p(O = s, T = c, H = h, W = t)} \\
= & \alpha p(O = s | P = y) p(T = c | P = y) \\
& p(H = h | P = y) p(W = t | P = y) p(P = y) \\
= & \alpha \times \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} \\
\approx & 0.0053\alpha
\end{aligned}
$$

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No

Supervised Learning: Naive Bayes

Outlook	Temp	Humidity	Windy	Play
Sunny	Cool	High	True	?

$$
\begin{aligned}
& p(P = y | O = s, T = c, H = h, W = t) \\
= & \frac{p(O = s, T = c, H = h, W = t | P = y) p(P = y)}{p(O = s, T = c, H = h, W = t)} \\
= & \alpha p(O = s | P = y) p(T = c | P = y) \\
& p(H = h | P = y) p(W = t | P = y) p(P = y) \\
= & \alpha \times \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} \approx 0.0053\alpha
\end{aligned}
$$

$$
\begin{aligned}
& p(P = n | O = s, T = c, H = h, W = t) \\
= & \frac{p(O = s, T = c, H = h, W = t | P = y) p(P = n)}{p(O = s, T = c, H = h, W = t)} \\
= & \alpha p(O = s | P = n) p(T = c | P = n) \\
& p(H = h | P = n) p(W = t | P = n) p(P = y) \\
= & \alpha \times \frac{3}{5} \times \frac{1}{5} \times \frac{3}{5} \times \frac{4}{5} \times \frac{5}{14} \approx 0.0205\alpha
\end{aligned}
$$

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mid	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mid	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mid	Normal	False	Yes
Sunny	Mid	Normal	True	Yes
Overcast	Mid	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mid	High	True	No

Supervised Learning: Naive Bayes

由下面的結果：

$$\begin{aligned} p(P=y|O=s,T=c,H=h,W=t) &\approx 0.0053\alpha \\ p(P=n|O=s,T=c,H=h,W=t) &\approx 0.0205\alpha \end{aligned}$$

可知如果Outlook是Sunny，Temp是Cool，Humidity是High，而且Windy是True的時候，根據現有資料，應該選擇Play=No。

又，由此例可以知道$\alpha = \frac{1}{p(O=s,T=c,H=h,W=t)}$並不影響結果。不過如果是因為某些原因，真的有必要計算出$p(P=y|O=s,T=c,H=h,W=t)$以及$p(P=y|O=s,T=c,H=h,W=t)$的話，因為這兩項相加等於1，可以用下面的式子來得到結果：

$$\begin{aligned} p(P=y|O=s,T=c,H=h,W=t) &= \frac{0.0053}{0.0053+0.0205} \approx 0.2054 \\ p(P=n|O=s,T=c,H=h,W=t) &= \frac{0.0205}{0.0053+0.0205} \approx 0.7946 \end{aligned}$$

Supervised Learning: Naive Bayes

關於Naive Bayes(續)

Naive bayes的假設，給定label，則各觀察值之間的機率互相獨立，這個假設在實際的問題上很少成立。不過Naive Bayes在很多應用上的表現其實非常不錯。例如垃圾郵件偵測。

雖然假設不成立，但結果卻不錯。這個狀況的一種可能的解釋是，因為Naive Bayes在判斷的時候是比較機率值大小，不需要精確的機率值。雖然錯誤的假設造成所估計的機率值和真實值有所誤差，但只要正確的種類機率值高於其他種類，Naive Bayes的結果仍然會是正確的。

python範例：Naive Bayes

在scikit-learn中有提供Naive Bayes模型。根據資料種類，有三種可以選擇：

- Multinomial Naive Bayes：假設$P(X|C)$遵守multinomial distribution。假設有n個features $X = X_1, \ldots, X_n$，此法估計$P(X_i = j|C = k) = \frac{N_{jk}+\alpha}{N_k+\alpha\times n}$。其中$N_k$是$C = k$的資料筆數，$N_{ik}$是$X_i = i$且$C = k$的資料筆數。$\alpha$是避免出現分母為0的狀況所加上的參數。
- Gaussian Naive Bayes：假設$P(X_i|C) = \frac{1}{\sqrt{2\pi\sigma_c^2}}\exp[-\frac{(x-\mu_c)^2}{2\sigma_c^2}]$。可用於feature為實數，不適合以離散隨機分佈(如multinomial distribution)來估計分佈的時候。
- Bernoulli Naive Bayes：當feature的數值只有兩種可能的狀況下，可以用Bernoulli distribution來估計$P(X|C)$

python範例：Naive Bayes

下面的程式是利用iris dataset，使用Gaussian Naive Bayes來做分類與預測的範例：

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

def run(random_seed=50,tsize=0.15):
  iris_data=load_iris()
  x=iris_data.data
  y=iris_data.target
  x_train,x_test,y_train,y_test=train_test_split(x,y,
  test_size=tsize,random_state=random_seed)

  gnb=GaussianNB().fit(x_train,y_train)
  predict=gnb.predict(x_test)
  predict_prob=gnb.predict_proba(x_test)
  print(f"Accuracy={gnb.score(x_test,y_test)}")
  return predict,predict_prob
```

可以在import此python檔之後呼叫run來執行。

python範例：Naive Bayes

Naive Bayes模型除可用predict取得模型預測，還可用pred_proba取得各預測機率值。上面的程式執行結果如下：

```
>>> import gaussian_nb_demo
>>> pred,prob=gaussian_nb_demo.run()
Accuracy=0.9565217391304348
>>> pred
array([1, 1, 0, 0, 2, 2, 2, 0, 0, 1, 0, 2, 0, 2, 1, 0, 1, 0, 1, 2, 2, 1,
       0])
>>> prob
array([[2.88821946e-074, 9.99703962e-001, 2.96037796e-004],
       [8.44569094e-122, 9.03754257e-001, 9.62457428e-002],
       [1.00000000e+000, 8.20521685e-016, 9.58861451e-022],
       [1.00000000e+000, 5.61544272e-017, 8.87780658e-023],
       [1.70463875e-169, 3.10385613e-004, 9.99689614e-001],
       [8.34730341e-226, 3.18634843e-009, 9.99999997e-001],
       [9.76057703e-156, 8.23043064e-003, 9.91769569e-001],
       [1.00000000e+000, 2.13423991e-011, 4.13825454e-018],
       [1.00000000e+000, 6.45693348e-019, 4.57868881e-025],
       [2.04465899e-103, 9.90812683e-001, 9.18731709e-003],
       [1.00000000e+000, 6.39509062e-018, 5.79175005e-024],
       [1.01149530e-222, 8.60724224e-007, 9.99999139e-001],
       [1.00000000e+000, 1.84837049e-018, 1.62779684e-024],
       [9.48599284e-196, 3.99333201e-006, 9.99996007e-001],
       [2.99851351e-091, 9.98770041e-001, 1.22995859e-003],
       [1.00000000e+000, 1.99882652e-016, 7.34871321e-023],
       [1.01041413e-063, 9.99951526e-001, 4.84742764e-005],
       [1.00000000e+000, 1.99403311e-018, 4.45054191e-024],
       [1.26387147e-048, 9.99996617e-001, 3.38337393e-006],
       [1.51122521e-139, 7.18600802e-002, 9.28139920e-001],
       [4.38360580e-257, 3.32007761e-011, 1.00000000e+000],
       [9.78519297e-056, 9.99991998e-001, 8.00248103e-006],
       [1.00000000e+000, 2.14609832e-019, 2.13096368e-024]])
```

python範例：Naive Bayes

如果是用Multinomial Naive Bayes來做分類與預測，方法與Gaussian Naive Bayes類似：

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

def run(random_seed=50,tsize=0.15):
  iris_data=load_iris()
  x=iris_data.data
  y=iris_data.target
  x_train,x_test,y_train,y_test=train_test_split(x,y,
  test_size=tsize,random_state=random_seed)

  mnb=MultinomialNB().fit(x_train,y_train)
  predict=mnb.predict(x_test)
  predict_prob=mnb.predict_proba(x_test)
  print(f"Accuracy={mnb.score(x_test,y_test)}")
  return predict,predict_prob
```

python範例：Naive Bayes

程式執行結果如下：

```
>>> import multinomial_nb_demo
>>> pred,prob=multinomial_nb_demo.run()
Accuracy=0.8695652173913043
>>> pred
array([1, 2, 0, 0, 2, 2, 2, 0, 0, 2, 0, 2, 0, 2, 1, 0, 1, 0, 1, 2, 2, 1,
       0])
>>> prob
array([[0.06543599, 0.48226697, 0.45229705],
       [0.02422008, 0.47383598, 0.50194394],
       [0.69598908, 0.19337979, 0.11063113],
       [0.75966947, 0.1554625 , 0.08486803],
       [0.01381222, 0.46207655, 0.52411124],
       [0.00654088, 0.44070536, 0.55275376],
       [0.01006596, 0.43914651, 0.55078753],
       [0.57601906, 0.25941883, 0.16456212],
       [0.69691113, 0.18981648, 0.1132724 ],
       [0.02670146, 0.47077011, 0.50252843],
       [0.71468097, 0.18116033, 0.10415871],
       [0.00860792, 0.46171994, 0.52967214],
       [0.6977903 , 0.19000194, 0.11220777],
       [0.01023648, 0.45715405, 0.53260947],
       [0.04475137, 0.48085841, 0.47439022],
       [0.6430055 , 0.22143361, 0.13556089],
       [0.0750917 , 0.48200039, 0.44290792],
       [0.75651324, 0.15739488, 0.08609188],
       [0.08881828, 0.47626398, 0.43491773],
       [0.0250446 , 0.48152996, 0.49342544],
       [0.00437141, 0.42591104, 0.56971755],
       [0.07306356, 0.47714416, 0.44979228],
       [0.83602044, 0.10962955, 0.05435001]])
```