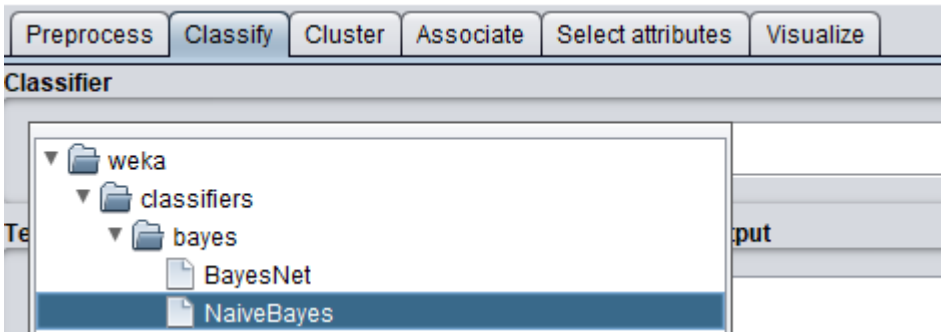
ETC HW1

Q1.

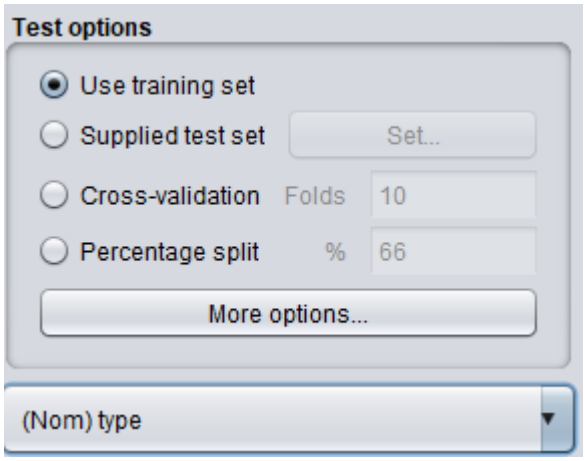
依照題意:

1. 選擇 Naïve Bayes 進行 Supervised learning



➔ 這裡代表選擇 Naïve Bayes 的方式來做分類

2. 選擇 Use training set，設定 Attribute: type 為 Output,



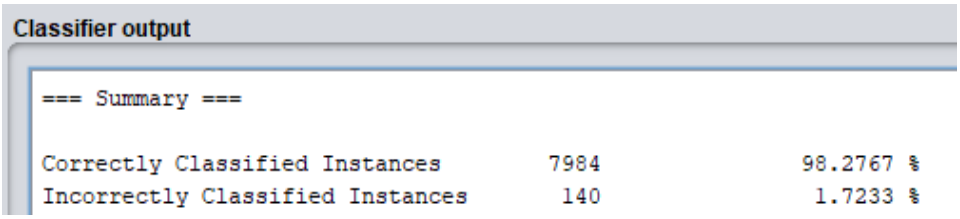
➔ 「Use training set」：代表我用這個 training set 建完 model 之後，也用這個 training set 當作 testing set 來做 testing。通常不會這樣做，而是選擇下面的 Percentage split 來決定多少%要做 training，多少%做 testing。

➔ 「Attribute: type 為 Output」: 代表我把屬性「type」當作 output 值，也就是貝式定理 P(H|E)中的 H(hypothesis)。

Ex: 假設 H = yes/no , E = sunny/rainy/cloudy, 這裡的 H 就代表我所關心的 output 是 yes/no 去做某件事情，在 E 的條件下。

(a)

1. Test data 錯誤率:

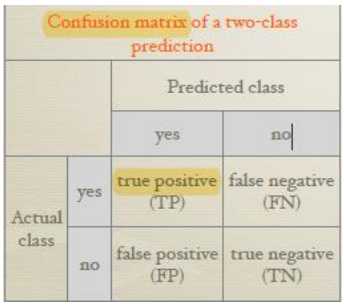


➔ 由圖可知錯誤率為 1.7233%

2. 有多少 Test dataset instances 被分類到有毒的(poisonous)但實際上屬於可食用的(edible)?用 Confusion matrix 解釋。

```
=== Confusion Matrix ===

    a    b   <-- classified as
 3822   94 |    a = p
   46 4162 |    b = e
```



如圖所示，a 代表 p(有毒的)，b 代表 e(可食用的)，配合右圖可得知以下資訊:

「預測有毒 + 實際有毒」 = 3822 → 預測正確

「預測可食用 + 實際有毒」 = 94 → 預測錯誤

「預測有毒 + 實際可食用」 = 46 → 預測錯誤

「預測可食用 + 實際可食用」 = 4162 → 預測正確

➔ 依照題意「預測有毒 + 但實際可食用」 = 46(units)

(b)

1. output prediction 中的“+”代表甚麼?

```
=== Predictions on training set ===

    inst#     actual  predicted error prediction
        1        1:p        1:p       0.985
        2        2:e        2:e       1
        3        2:e        2:e       1
        4        1:p        1:p       0.972
        5        2:e        2:e       1
        6        2:e        2:e       1
        7        2:e        2:e       1
        8        2:e        2:e       1
        9        1:p        1:p       0.973
       10        2:e        2:e       1
       11        2:e        2:e       1
       12        2:e        2:e       1
       13        2:e        2:e       1
```

依此圖可看出，第二個和第三個欄位分別為「實際值」、「預測值」

```
4097        2:e        2:e       0.994
4098        1:p        1:p       1
4099        1:p        1:p       0.937
4100        1:p        1:p       1
4101        1:p        1:p       1
4102        1:p        1:p       1
4103        1:p        1:p       1
4104        2:e        2:e       0.992
4105        1:p        1:p       1
4106        2:e        2:e       1
4107        1:p        2:e   +   1
4108        2:e        2:e       1
4109        1:p        1:p       1
4110        2:e        2:e       1
4111        1:p        1:p       1
4112        1:p        1:p       1
4113        1:p        1:p       1
4114        2:e        2:e       0.978
```

```
4266        1:p        1:p       1
4267        1:p        1:p       1
4268        1:p        1:p       1
4269        1:p        1:p       1
4270        1:p        1:p       1
4271        1:p        1:p       1
4272        1:p        1:p       1
4273        1:p        1:p       1
4274        1:p        1:p       1
4275        1:p        1:p       1
4276        1:p        1:p       1
4277        2:e        1:p   +   0.906
4278        1:p        1:p       1
4279        1:p        1:p       1
4280        1:p        1:p       1
4281        1:p        1:p       1
4282        1:p        1:p       1
4283        1:p        1:p       1
4284        2:e        2:e       1
```

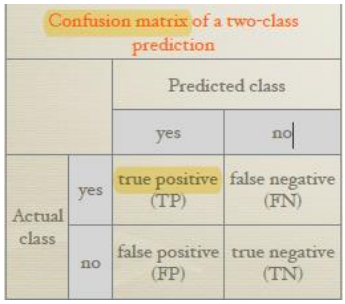
繼續觀察資料可發現，當實際值 = 預測值，並不會出現 “+” ；但實際值 ≠ 預測值的時候 “+” 就出現了，因此可推斷 “+” 代表預測錯誤。

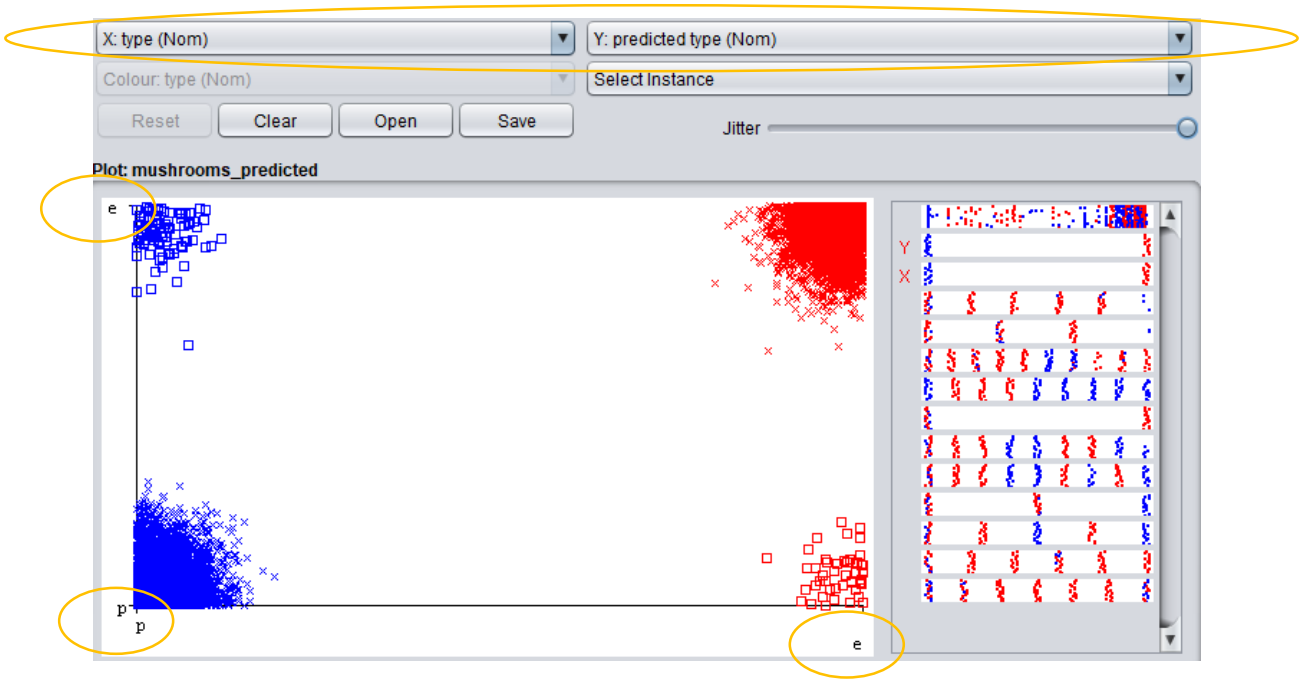
(C)

1. 使用 Visualize Classifier Errors, 解釋此圖與 Confusion matrix 之間的關係。

```
=== Confusion Matrix ===

    a    b   <-- classified as
 3822   94 |    a = p
   46 4162 |    b = e
```





首先，Visualize Classifier Errors 的圖最上方可得知，X 軸為實際值、Y 軸為預測值；再者，觀察圖的四周可發現左下角為 p，左上、右下為 e，因此可以得知左下為 pp(預測 p 且實際 p) = Confusion matrix 中的 TP(3822)，以此類推右上角為 TN(4162)、左上角為 FN(94)、右下角為 FP(46)。仔細觀察數字更可以發現與圖片上的密度分布相符。順帶一提:

Visualize Classifier Errors 圖轉 90 度就跟 Confusion matrix 一樣了。

Q2.

(a)

```
In [1]: import pandas as pd
        #讀取CSV檔案
        data = pd.read_csv('mushrooms.csv')
```

➔ 先把檔案讀進來

```
In [2]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   type                    8124 non-null   object
 1   cap_shape               8124 non-null   object
 2   cap_surface             8124 non-null   object
 3   cap_color               8124 non-null   object
 4   odor                    8124 non-null   object
 5   stalk_shape             8124 non-null   object
 6   stalk_color_above_ring  8124 non-null   object
 7   stalk_color_below_ring  8124 non-null   object
 8   ring_number             8124 non-null   object
 9   ring_type               8124 non-null   object
 10  population              8124 non-null   object
 11  habitat                 8124 non-null   object
dtypes: object(12)
memory usage: 761.8+ KB
```

➔ 然後觀察看看資料大致樣貌，可得知有 8124 個 instance、有 12 種 attribute、沒有 null 的欄位、data type 都為 object、記憶體花了多少……

```
In [4]: data
```

Out[4]:

	type	cap_shape	cap_surface	cap_color	odor	stalk_shape	stalk_color_above_ring	stalk_color_below_ring	ring_number
0	p	x	s	n	p	e	w	w	o
1	e	x	s	y	a	e	w	w	o
2	e	b	s	w	l	e	w	w	o
3	p	x	y	w	p	e	w	w	o
4	e	x	s	g	n	t	w	w	o
...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	n	e	o	o	o
8120	e	x	s	n	n	e	o	o	o
8121	e	f	s	n	n	e	o	o	o
8122	p	k	y	n	y	t	w	w	o
8123	e	x	s	n	n	e	o	o	o

8124 rows × 12 columns

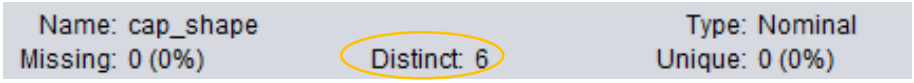
➔ 這步驟是為了觀察各 attribute 的 value 是什麼形式、有哪些可能值，由此可知為 nominal。

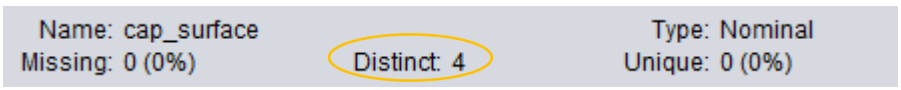
```
In [5]: data.describe()
```

Out[5]:

	type	cap_shape	cap_surface	cap_color	odor	stalk_shape	stalk_color_above_ring	stalk_color_below_ring	ring_number	ring_type	p
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	
unique	2	6	4	10	9	2	9	9	3	5	
top	e	x	y	n	n	t	w	w	o	p	
freq	4208	3656	3244	2284	3528	4608	4464	4384	7488	3968	

➔ 用 describe()來查看統計資料。由上述可知，nominal 不需要 mean、std 等參數，因此未出現這些參數是正常的。這裡有一點要注意，「unique」代表的意義相等於「Weka 中的 Distinct」而非 unique，可從以下截圖佐證:






```
In [3]: #x:input
        x=data.loc[:,"cap_shape":]
        print(x)
        #y:output
        y=data.loc[:,["type"]]
        print(y)
```

➔ loc 函數前半段處理 row，後半段處理 col，因此可得知我取得了所有的 row(instance)並切割除了「type」以外的所有屬性放入 x 中當作 input；

「type」屬性則是放入 y 當作 output。

```
In [51]: from sklearn import preprocessing
         #將屬性轉為數字label
         le = preprocessing.LabelEncoder()
```

➔ 放入 model 訓練前，先把 value 編碼成數字，所以用了一個 LableEncoder()，他可以把數字、英文這種東西按照順序編碼成 0~n，其中 a < b < … < z ; 0 < 1 < ……

```
# 儲存編碼後的值，之後作為input
X_encoded = []
# 儲存編碼前、後的對應值
X_dic = {}
for col_name in x:
    encoded_item = le.fit_transform(x[col_name]) # 將各col的value轉成數字
    X_encoded.append(encoded_item)

    # 用set排除重複值，並存入dic中
    temp_dic = {}
    key_set = sorted( set(le.inverse_transform(encoded_item)) ) # 取得原始值作為key
    value_set = set(encoded_item) # 取得編碼後的值作為val
    for i,j in zip(key_set,value_set):
        temp_dic[i] = j
    X_dic[col_name] = temp_dic
```

➔ 我把編碼後的結果存入 X_encoded，並建立一個 dictionary(X_dic)，用來方便確認各個 value 編碼後的結果為何。

```
print("x.cap_shape:\n", x.cap_shape)
print("X_encoded[0]:\n", X_encoded[0])
print("X_dic[\"cap_shape\"]:\n", X_dic["cap_shape"])
```

```
x.cap_shape:
 0       x
1       x
2       b
3       x
4       x
       ..
8119    k
8120    x
8121    f
8122    k
8123    x
Name: cap_shape, Length: 8124, dtype: object
X_encoded[0]:
 [5 5 0 ... 2 3 5]
X_dic["cap_shape"]:
 {'b': 0, 'c': 1, 'f': 2, 'k': 3, 's': 4, 'x': 5}
```

➔ 由上圖可看出，編碼後前 3 碼為[5,5,0]對應 dic 中的[x,x,b]與原始數據相同，之所以 key 為原始值是為了之後 input 方便使用，predict 時會再次說明。

```
#將type轉為數字label
#type: e:0 ,p:1
Y_dic = {0:"e", 1:"p"}
Y_type_label = le.fit_transform(y.type)
# print(Y_type_label)
```

➔ 這部分與 x 相同，只是改成 y 且只有一個「type」屬性，因此不加贅述

```
feature=list(zip(X_encoded[0], X_encoded[1], X_encoded[2], X_encoded[3], X_encoded[4], X_encoded[5],
                 X_encoded[6], X_encoded[7], X_encoded[8], X_encoded[9], X_encoded[10]))

# for i in X_encoded:
#     print(i[0], end=" ")
# print("\n" + str(feature[0]))
```

➔ 用 zip 把各個編碼過的數據壓縮起來，壓縮格式為「相同 index 的元素壓成一個元組」，最後再轉成 list 如下圖所示(只取一個元素當作範例):

```
# for i in X_encoded:
#     print(i[0], end=" ")
# print("\n" + str(feature[0]))
```

```
5 2 4 6 0 7 7 1 4 3 5
(5, 2, 4, 6, 0, 7, 7, 1, 4, 3, 5)
```

```
#轉成array
import numpy as np
features=np.asarray(feature)
# print(feature)
```

➔ 轉成 numpy array，為了給之後的 model 使用，而且也多了許多運算可用

```
#Import Gaussian Naive Bayes 模型 (高斯樸素貝氏)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()

# 訓練集訓練模型
# model.fit(x, y)
model.fit(features, Y_type_label)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

➔ Load 一個已經建好的 model，用 model.fit()把 data 丟進去訓練

```
expected = Y_type_label # 代表真正的答案
predicted = model.predict(features) # 代表預測的答案
print(expected, type(expected))
print(predicted, type(predicted))
from sklearn import metrics
print(metrics.classification_report(expected, predicted)) # 把真正的答案和預測的答案拿去做統計報告
print(metrics.confusion_matrix(expected, predicted))
```

```
[1 0 0 ... 0 1 0] <class 'numpy.ndarray'>
[0 0 0 ... 0 1 0] <class 'numpy.ndarray'>
              precision    recall  f1-score   support

           0       0.87      0.78      0.82      4208
           1       0.79      0.87      0.83      3916

    accuracy                           0.83      8124
   macro avg       0.83      0.83      0.83      8124
weighted avg       0.83      0.83      0.83      8124

[[3296  912]
 [ 493 3423]]
```

➔ Model 訓練好後，就開始預測了。Expected 為真正的答案，predicted 為預測出來的答案，metrics.classification_report(expected, predicted)可以幫我們做分析報告。Precision 代表精確度，0.87 為預測 0 正確的機率 = (3296)/(3296+493) = 0.869 ≒ 0.87。Accuracy 代表整體的準確度 = 0.87*(4208)/8124 + 0.79*(3916)/8124 = 0.83… ≒ 0.83。

```
input_data = [X_dic["cap_shape"]["b"], X_dic["cap_surface"]["f"], X_dic["cap_color"]["n"], X_dic["odor"]["a"],
              X_dic["stalk_shape"]["e"], X_dic["stalk_color_above_ring"]["n"], X_dic["stalk_color_below_ring"]["n"],
              X_dic["ring_number"]["n"], X_dic["ring_type"]["e"], X_dic["population"]["a"], X_dic["habitat"]["g"]]
# input_data = np.asarray(input_data)
predicted= model.predict([input_data]) # input不用轉成numpy array?
print ("Predicted Value:", predicted)
print ("Predicted Value:", Y_dic[predicted[0]])
```

```
Predicted Value: [0]
Predicted Value: e
```

➔ 最後我們可以實際 input 一組 data 來預測一個 output，這裡我之前特別做的 dictionary 就是為了在此方便了解自己各個 attribute 選擇了哪個 value 作為 input。把這個 input_data 放入 model.predict()中就可以得到預測的答案了。

(b)

1. mushrooms 資料集中共有多少 instance?

```
In [2]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   type                      8124 non-null   object
 1   cap_shape                 8124 non-null   object
 2   cap_surface               8124 non-null   object
 3   cap_color                 8124 non-null   object
 4   odor                      8124 non-null   object
 5   stalk_shape               8124 non-null   object
 6   stalk_color_above_ring    8124 non-null   object
 7   stalk_color_below_ring    8124 non-null   object
 8   ring_number               8124 non-null   object
 9   ring_type                 8124 non-null   object
 10  population                8124 non-null   object
 11  habitat                   8124 non-null   object
dtypes: object(12)
memory usage: 761.8+ KB
```

➔ 由此圖可以看出有 8124 個 instance

2. 是否包含空值(null)的欄位?

➔ 同樣由上圖可看出沒有空值(null)的欄位

(c)

1. stalk_color_above_ring 有幾種不同的 value?

```
# For (c)小題
print(X_dic["stalk_color_above_ring"])
print(len(X_dic["stalk_color_above_ring"]))
```

```
{'b': 0, 'c': 1, 'e': 2, 'g': 3, 'n': 4, 'o': 5, 'p': 6, 'w': 7, 'y': 8}
9
```

➔ 如圖所示，用我之前整理的 dictionary 可以找出有 9 種不同的 value

```
data.describe()
```

	type	cap_shape	cap_surface	cap_color	odor	stalk_shape	stalk_color_above_ring
count	8124	8124	8124	8124	8124	8124	8124
unique	2	6	4	10	9	2	9

➔ 也可以用內建的 describe()找出有 9 種不同的 value，但這樣無法得知這些 value 分別為何，在 input 的時候不好處理。

(d)

1. 用 metrics.confusion_matrix ( )呈現出混淆矩陣，並截圖加以說明。

```
print(metrics.classification_report(expected, predicted)) # 把真正的答案和預測的答案拿去做統計報告
print(metrics.confusion_matrix(expected, predicted))
```

```
              precision    recall  f1-score   support

           0       0.87      0.78      0.82      4208
           1       0.79      0.87      0.83      3916

    accuracy                           0.83      8124
   macro avg       0.83      0.83      0.83      8124
weighted avg       0.83      0.83      0.83      8124

[[3296  912]
 [ 493 3423]]
```

根據 report 的結果，可以推論出 confusion_matrix 的 row 代表實際值、col 代表預測值。所以 3296 代表 TP[0,0]、912 代表 TN[0,1]、493 代表 FP[1,0]、3423 代表 FN[1,1]；在此 e = 0, p = 1。舉例:

0.87 代表預測 0 實際為 0 的機率 = (3296)/(3296+493) = 0.869 ≒ 0.87

(e)

1. 請利用 metrics.classification_report 列出模型的準確率並與 Weka 的結果比較何者較高?

```
print(metrics.classification_report(expected, predicted)) # 把真正的答案和預測的答案拿去做統計報告
print(metrics.confusion_matrix(expected, predicted))
```

```
              precision    recall  f1-score   support

           0       0.87      0.78      0.82      4208
           1       0.79      0.87      0.83      3916

    accuracy                           0.83      8124
   macro avg       0.83      0.83      0.83      8124
weighted avg       0.83      0.83      0.83      8124

[[3296  912]
 [ 493 3423]]
```

➔ 準確率(這裡用加權平均) = 0.87*(4208)/8124 + 0.79*(3916)/8124 =

0.83... ≒ 0.83。

In Weka:

```
=== Detailed Accuracy By Class ===

                 TP Rate   FP Rate   Precision   Recall   F-Measure
                 0.976     0.011     0.988       0.976    0.982
                 0.989     0.024     0.978       0.989    0.983
Weighted Avg.    0.983     0.018     0.983       0.983    0.983
```

➔ 可以看出 Weka 的 0.983 > 0.83