In [4]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

In [12]:

```
df = pd.read_csv('mushrooms.csv')
df.head(n=5)
```

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_size
0	p	x	s	n	t	p	f	c	n
1	e	x	s	y	t	a	f	c	b
2	e	b	s	w	t	l	f	c	b
3	p	x	y	w	t	p	f	c	n
4	e	x	s	g	f	n	f	w	b

5 rows × 23 columns

In [13]:

```
print(df.columns)
```

```
Index(['type', 'cap_shape', 'cap_surface', 'cap_color', 'bruises', 'odor',
       'gill_attachment', 'gill_spacing', 'gill_size', 'gill_color',
       'stalk_shape', 'stalk_root', 'stalk_surface_above_ring',
       'stalk_surface_below_ring', 'stalk_color_above_ring',
       'stalk_color_below_ring', 'veil_type', 'veil_color', 'ring_number',
       'ring_type', 'spore_print_color', 'population', 'habitat'],
      dtype='object')
```

Encode the Categorical data into Numerical Data

In [15]:

```
le = LabelEncoder()
#Applies transformation on each column
ds = df.apply(le.fit_transform)
```

In [17]:

```
ds.head(n=5)
```

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	gill_spacing	gill_size
0	1	5	2	4	1	6	1	0	1
1	0	5	2	9	1	0	1	0	0
2	0	0	2	8	1	3	1	0	0
3	1	5	3	8	1	6	1	0	1
4	0	5	2	3	0	5	1	1	0

5 rows × 23 columns

In [19]:

```
data = ds.values
print(data.shape)
print(type(data))
print(data[:5,:])

data_x = data[:,1:]
data_y = data[:,0]
```

```
(8124, 23)
<class 'numpy.ndarray'>
[[1 5 2 4 1 6 1 0 1 4 0 3 2 2 7 7 0 2 1 4 2 3 5]
 [0 5 2 9 1 0 1 0 0 4 0 2 2 2 7 7 0 2 1 4 3 2 1]
 [0 0 2 8 1 3 1 0 0 5 0 2 2 2 7 7 0 2 1 4 3 2 3]
 [1 5 3 8 1 6 1 0 1 5 0 3 2 2 7 7 0 2 1 4 2 3 5]
 [0 5 2 3 0 5 1 1 0 4 1 3 2 2 7 7 0 2 1 0 3 0 1]]
```

Break the data into Train and Test Set

```
x_train,x_test,y_train,y_test = train_test_split(data_x,data_y,test_size=0.2)
print(x_train.shape, y_train.shape)
print(x_test.shape,y_test.shape)
```

In [22]:

```
np.unique(y_train) # only 2 classes of mushroom
```

```
array([0, 1])
```

Building Classifier

In [24]:

```
def prior_prob(y_train, label):

    total_examples = y_train.shape[0]
    class_examples = np.sum(y_train==label)

    return class_examples/float(total_examples)
```

In [29]:

```
#a = np.array([0,1,1,0,1,1,0,1,0,1])
#b = np.array([10,20,30,40,50,60,70,80,90,100])
#print(a==1)
#c = b[a==1]
#print(c)
```

In [35]:

```
def cond_prob(x_train,y_train,feature_col,feature_val,label):

    x_filtered = x_train[y_train==label]
    num = np.sum(x_filtered[:,feature_col]==feature_val)
    den = np.sum(y_train==label)

    return num/float(den)
```

Compute Posterior Probability for each test example and make pr

In [36]:

```
def predict(X_train,Y_train,x_test):

    classes = np.unique(Y_train)
    features = X_train.shape[1]
    post = []
    for label in classes:

        likelihood = 1.0
        for f in range(features):
            cond = cond_prob(X_train,Y_train,f,x_test[f],label)
            likelihood *= cond

        prior = prior_prob(Y_train,label)
        posterior = likelihood * prior
        post.append(posterior)

    return np.argmax(post)
```

In [37]:

```
pred = predict(x_train,y_train,x_test[0])
print(pred)
print(y_test[0])
```

```
1
1
```

In [38]:

```
def accuracy(X_train,Y_train,X_test,Y_test):

    pred = []
    n = X_test.shape[0]
    for i in range(n):
        pred.append(predict(X_train,Y_train,X_test[i]))

    accuracy = np.sum(pred==Y_test)/float(n)
    return accuracy
```

In [40]:

```
acc = accuracy(x_train,y_train,x_test,y_test)
print("Accuracy:",acc)
```

```
Accuracy: 0.9987692307692307
```

In [ ]: