In [1]:
```python
import numpy as np
```

In [2]:
```python
def fit(X_train, Y_train):
    result = {}
    class_values = set(Y_train)
    for current_class in class_values:
        result[current_class] = {}
        result["total_data"] = len(Y_train)
        current_class_rows = (Y_train == current_class)
        X_train_current = X_train[current_class_rows]
        Y_train_current = Y_train[current_class_rows]
        num_features = X_train.shape[1]
        result[current_class]["total_count"] = len(Y_train_current)
        for j in range(1, num_features + 1):
            result[current_class][j] = {}
            all_possible_values = set(X_train[:, j - 1])
            for current_value in all_possible_values:
                result[current_class][j][current_value] = (X_train_current[:, j - 1] == current_value).sum()

    return result
```

In [3]:
```python
def probability(dictionary, x, current_class):
    output = np.log(dictionary[current_class]["total_count"]) - np.log(dictionary["total_data"])
    num_features = len(dictionary[current_class].keys()) - 1;
    for j in range(1, num_features + 1):
        xj = x[j - 1]
        count_current_class_with_value_xj = dictionary[current_class][j][xj] + 1
        count_current_class = dictionary[current_class]["total_count"] + len(dictionary[current_class][j].keys())
        current_xj_probablity = np.log(count_current_class_with_value_xj) - np.log(count_current_class)
        output = output + current_xj_probablity
    return output
```

In [4]:
```python
def predictSinglePoint(dictionary, x):
    classes = dictionary.keys()
    best_p = -1000
    best_class = -1
    first_run = True
    for current_class in classes:
        if (current_class == "total_data"):
            continue
        p_current_class = probability(dictionary, x, current_class)
        if (first_run or p_current_class > best_p):
            best_p = p_current_class
            best_class = current_class
        first_run = False
    return best_class
```

In [5]:
```python
def predict(dictionary, X_test):
    y_pred = []
    for x in X_test:
        x_class = predictSinglePoint(dictionary, x)
        y_pred.append(x_class)
    return y_pred
```

In [6]:
```python
def makeLabelled(column):
    second_limit = column.mean()
    first_limit = 0.5 * second_limit
    third_limit = 1.5*second_limit
    for i in range (0,len(column)):
        if (column[i] < first_limit):
            column[i] = 0
        elif (column[i] < second_limit):
            column[i] = 1
        elif(column[i] < third_limit):
            column[i] = 2
        else:
            column[i] = 3
    return column
```

In [7]:
```python
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
Y = iris.target
```

In [8]:
```python
for i in range(0,X.shape[-1]):
    X[:,i] = makeLabelled(X[:,i])
```

In [9]:
```python
from sklearn import model_selection
X_train,X_test,Y_train,Y_test = model_selection.train_test_split(X,Y,test_size=0.25,random_state=0)
```

In [10]:
```python
dictionary = fit(X_train,Y_train)
```

In [11]:
```python
Y_pred = predict(dictionary,X_test)
```

In [12]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        13
          1       0.94      1.00      0.97        16
          2       1.00      0.89      0.94         9

avg / total       0.98      0.97      0.97        38

[[13  0  0]
 [ 0 16  0]
 [ 0  1  8]]
```

The problem with the Naive Bayes with continous data is how are you going to calculate the $P(X^j = x^j / y = a_i)$.
Lets say i have 1.2 and 1.3 in my training data. But it doesnt mean you will never see 1.25 in the testing data. You might see it.
So the Probablity distribution that we are going to assume is that we are going to assume Gaussian Distribution.
That means we will see alot more values near the mean, as you move away from the mean, you will see lesser and lesser values coming out of that distribution.

In [15]:
```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)
print(classification_report(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

```
             precision    recall  f1-score   support

          0       1.00      0.85      0.92        13
          1       0.76      1.00      0.86        16
          2       1.00      0.67      0.80         9

avg / total       0.90      0.87      0.87        38

[[11  2  0]
 [ 0 16  0]
 [ 0  3  6]]
```

Processing math: 100%