Naive Bayes Classifier Script Documentation

Requirements:

1. Python 2.7
2. numpy
3. matplotlib

Procedure:

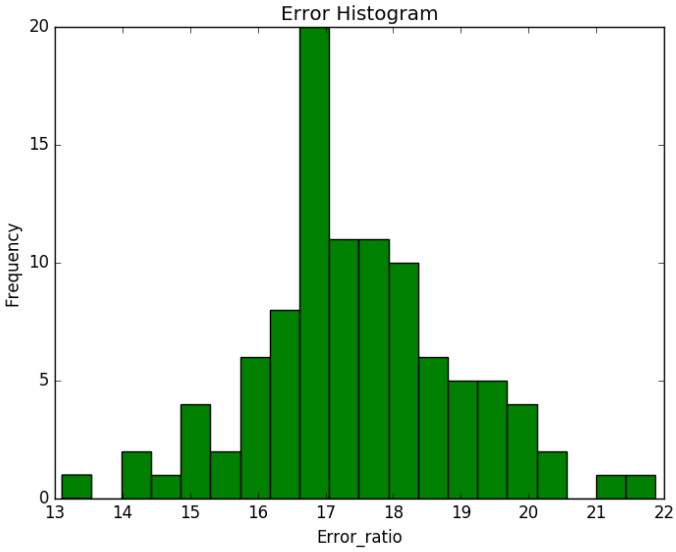
1. navigate to the script directory
2. type `python NBC.py` and press Enter

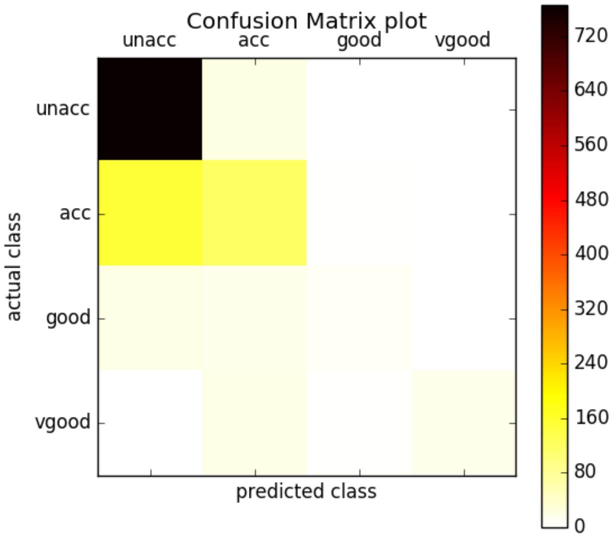
```
amr@amr-Laptop:~/machine_learning/cardaten/NBC$ python
NBC.py
Doing Experiment 100

Mean error after 100 random experiments is 17.46 %

Printing confusion Matrix
=========================
predicted unacc acc good vgood
actual
unacc        766    21    0    0
acc        153    122    4    1
good        19    15    11    0
vgood        0    20    3    17
```

You get an output on the command like that. A counter for the experiments, the mean error rate for all the experiments and a confusion matrix. You also get two plots like the following: a histogram for the error ratios of the different experiments and a color map for the confusion matrix. We discuss theses plots later in the discussion part.





Code structure

The code is structured mainly in 7 functions:-

sample_the_data(data)

Takes the data array as an argument and returns randomly third of the data as a training sample and the rest two thirds as a testing sample.

```
In [5]: len(cardata)
Out[5]: 1728
In [6]: training_sample, testing_sample = sample_the_data(cardata)
In [7]: len(training_sample)
Out[7]: 576
In [8]: len(testing_sample)
Out[8]: 1152
```

NBC_train(training_sample, attributes, classes)

The training function takes as arguments the training sample, the attributes names and the classes and returns the model of attributes values frequencies and the classes count. The first return value is a list of lists(nested list). Each list represent a class and it contains a list of dictionaries. Each dictionary represent an attribute whose keys are the values of this attribute and the values of the the keys are the frequency of this attribute value within this class.
The second return value is just the overall frequencies of the classes.

```
In [9]: model = NBC_train(training_sample, attributes, classes)
In [10]: model[0]
Out[10]:
[[{'high': 105, 'low': 86, 'med': 91, 'vhigh': 117},
{'high': 101, 'low': 94, 'med': 93, 'vhigh': 111},
{'2': 110, '3': 104, '4': 100, '5more': 85},
{'2': 203, '4': 91, 'more': 105},
{'big': 112, 'med': 133, 'small': 154},
{'high': 93, 'low': 184, 'med': 122}],
[{'high': 41, 'low': 26, 'med': 48, 'vhigh': 25},
{'high': 38, 'low': 30, 'med': 47, 'vhigh': 25},
{'2': 28, '3': 33, '4': 41, '5more': 38},
{'4': 68, 'more': 72},
{'big': 49, 'med': 53, 'small': 38},
{'high': 76, 'med': 64}],
[{'low': 14, 'med': 6},
{'low': 14, 'med': 6},
{'2': 6, '3': 8, '4': 2, '5more': 4},
{'4': 12, 'more': 8},
{'big': 10, 'med': 6, 'small': 4},
{'high': 8, 'med': 12}],
[{'low': 9, 'med': 8},
{'high': 3, 'low': 6, 'med': 8},
{'2': 2, '3': 4, '4': 6, '5more': 5},
{'4': 10, 'more': 7},
{'big': 8, 'med': 9},
{'high': 17}]]
In [11]: model[1]
Out[11]: [399, 140, 20, 17]
```

NBC_classify(instance, model, classes)

The classify function takes as argument one new instance, the model and the classes list and returns the most probable class classification for that instance.

```
In [12]: NBC_classify(testing_sample[0], model, classes)
Out[12]: 'unacc'
```

error(testing_sample, model, classes)

The error function takes as an argument the testing sample, the model and the classes values. it classify every instance and compare the result with the actual classification. It returns an error ratio, a list of the actual classifications and a list of the predicted classifications.

```
In [13]: error_ratio, actual, predicted = error(testing_sample, model, classes)
In [14]: error_ratio
Out[14]: 17.795138888888889
In [17]: actual[500:510]
Out[17]:
['unacc',
'acc',
'unacc',
'unacc',
'acc',
'unacc',
'acc',
'unacc',
'unacc',
'unacc']
In [18]: predicted[500:510]
Out[18]:
['unacc',
'acc',
'unacc',
'acc',
'acc',
'unacc',
'acc',
'unacc',
'unacc',
'unacc']
```

test(K)

The test function takes an integer K and do K experiments. in each experiment it takes a new random training and testing samples and do the training, classification and error calculation. it returns a list of all error ratios and the mean error ratio.

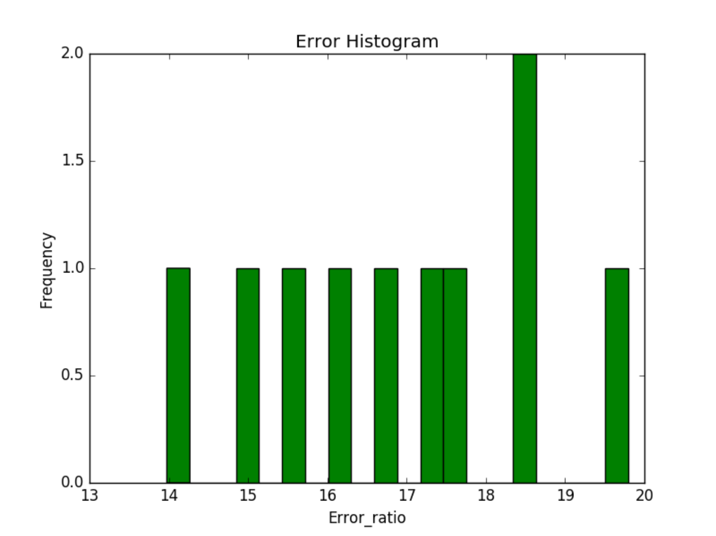
```
In [20]: K = 10
In [21]: mean_error, error_ratios = test(K)
Doing Experiment 10

In [22]: mean_error
Out[22]: 16.883680555555554
In [23]: error_ratios
Out[23]:
[13.975694444444445,
15.01736111111111,
18.40277777777778,
17.36111111111111,
18.489583333333336,
16.05902777777778,
16.753472222222222,
17.534722222222222,
19.791666666666664,
15.451388888888889]
```

plot_error_histogram(error_ratios)

It takes a list of error ratios and plots a histogram of the frequency of each error ratio.

```
In [24]: plot_error_histogram(error_ratios)
In [25]: plt.show()
```

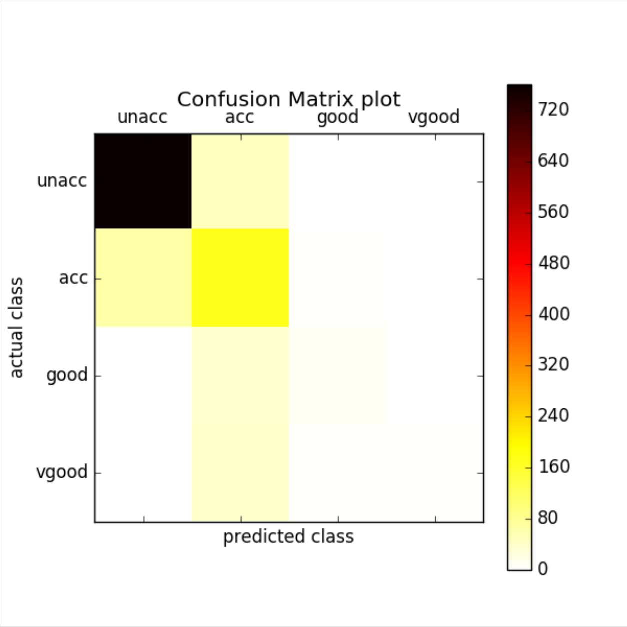


confusion_matrix(actual, predicted, classes)

This function takes a list of actual classifications, another list of predicted classifications and a list of the classes. It calculates the confusion matrix, prints it to the screen, plots it as a color map and returns it as a list of lists.

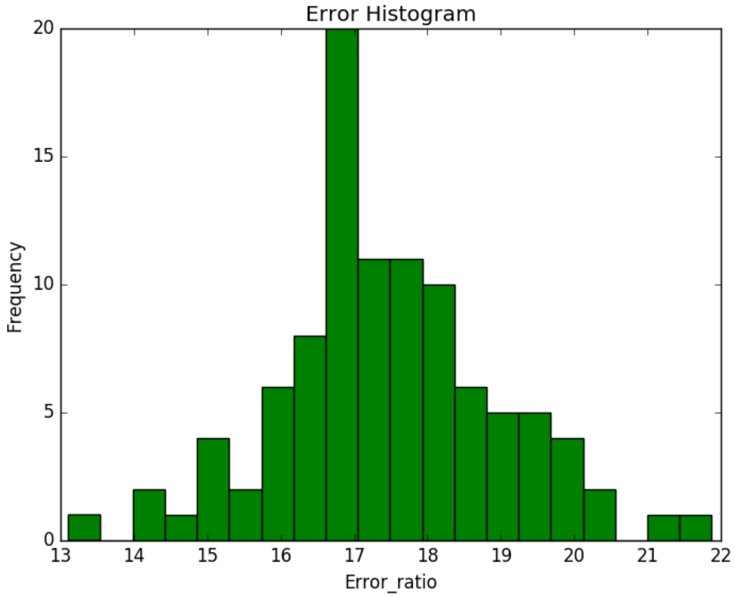
```
In [26]: m = confusion_matrix(actual, predicted, classes)
Printing confusion Matrix
=========================
predicted unacc acc good vgood
actual
unacc 761 50 0 0
acc 67 174 3 0
good 2 38 9 0
```

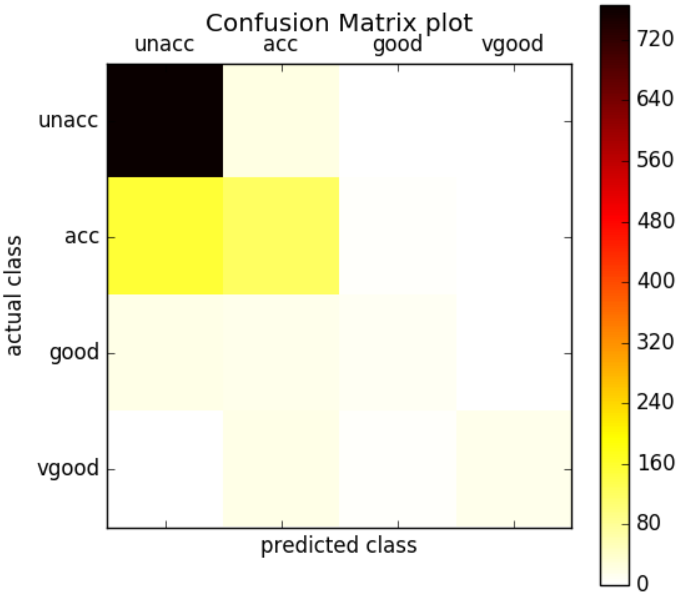
```
vgood 0 41 4 3
In [27]: m
Out[27]: [[761, 50, 0, 0], [67, 174, 3, 0], [2, 38, 9, 0], [0, 41,
4, 3]]
```



Discussion

Now we return to the original results we had.





After 100 random experiments, we got a mean error ratio of about 17 % which is better than what we got before when we used k means clustering algorithm as a way of classification. We got an error of about 30 %.
we can see from the histogram that the error varied from 13% to 22% with different probabilities. This naïve bayes classification algorithm is based on probabilities so its accuracy will be dependent on the training sample choice and how well it represents the population from which it was drawn. That explains the variability of error among different experiments. Sometimes the sample represents the probabilities in the population and hence in the testing samples better and sometimes worse.
From the confusion matrix we can see that the off diagonals of the matrix are mostly white (low numbers) which is good as they represent the misclassification (the diagonal cells represent the correct classification). Only in one case which when the actual class is "acc" and the algorithm predicts mostly "unacc". This can be explained as the prior probability of the "unacc"class (***p(class="unacc")***) is very high relative to "acc" class (the class is dominant in the data set) so it wins in the competition even if the conditional probabilities of the attributes values vote for favor of "acc" class. If we look at the actual numbers not the plot, we realize that it's

actually a trend on the left side of the diagonal. For example also “good” to be classified as “acc”. This is also can be explained because of the higher prior probability of class “acc” than “good”. That’s why aslo we do not see much errors on the right side of the diagonal.