Q1) Implement Naïve Bay's classifier and use the data set have been used in task 1 for test your implementation ?

I attempted this question on 4 steps :
first :

1. due to the fact that some variance variables are very small resulting in NaNs in evaluating the exponential of the Gaussian distribution ,I tried to replace these small values with eps (where eps is a relatively small number which doesn't cause NaNs) ,but the algorithm turned out to be very sensitive to changing that value
2. then I thought maybe the problem is the scale of the input itself ... so I rescaled the input .. my multiplying each attribute by 10 and setting eps to 5 the algorithm became stable
3. the maximum accuracy my implementation achieved was 58% ... while the accuracy of the NN is ~71% ... that huge difference made me wonder if there was a problem with my implementation ... so I compared my my implementation's performance to that of python's famous sklearn's implementation (GaussianNB) -code included - ... the results are quite close with sklearn's peak being ~63% (only 5% more than my implementation)
4. fourth and last step … the main problem with my implementation was its low numerical stability (it couldn't handle very small values and very large values) ,I traced the code and found that the problem was the part where I evaluate the Gaussian exponential (in that part I might get values like 0/0 or inf/inf) , mathematically the way to handle this situation is to take the limit ,luckily however python has a built in function (norm.cdf) which evaluates the exponential and takes the limit when needed … the impact of this simple change was amazing and I shall explain how so on each task

Q2) Compare the Naïve Bay's classifier results with back-propagation classification.

1. Running time for each classifier
   running time NB: 0.00154304504395 seconds
   running time Backprop : 43.5755839348 seconds

2. Classification rate for each model including (TP= true positive, TN=true negative, FP=false positive, FN=false negative)

the accuracy of NN with Back propagation : 70.77%
the accuracy of NB : 63.08%
note : before I modified the NB implementation the accuracy was ~56.92% and it was unstable (jumping from 50% to 12% to 26% to 40% and so on) and some classes were never predicted … after I modified the implementation it became stable (converged in the range 48% - 65%) and a lot of times it produced the same accuracy as of that of python built-in NB and some times even beat it (in the particular run where my implementation produced 63.08% .. python's NB produced 61.54%) … of course the fluctuations are due to the random shuffling of data I perform in the beginning

below are the values of true negative ,false positive , false negative ,true positive respectively for each class
[44.61538461538462, 20.0, 7.6923076923076925, 27.692307692307693]
[69.23076923076924, 4.615384615384616, 18.461538461538463, 7.6923076923076925]
[78.46153846153847, 12.307692307692308, 6.153846153846154, 3.076923076923077]
[100.0, 0.0, 0.0, 0.0]
[93.84615384615385, 0.0, 4.615384615384616, 1.5384615384615385]
[92.3076923076923, 0.0, 0.0, 7.6923076923076925]
[84.61538461538461, 0.0, 0.0, 15.384615384615385]

3. Test sample classification results
   can be found in predictions_task_2.csv

Q3) Naïve Bay's classifier can be used for image segmentation task.

to extract the pixel values of image I wrote a script (pixel_extraction.py) which displays the image and reads the pixel values at positions where the mouse cursor hit ,created a data of 73 instances and saved it in data.csv

after that the job was straightforward and a lot of the code was reused from Q2
first I rescaled the input to be in range [0,1] and ran the classifier the predictions were saved data in saved_data.csv

running time : 0.00058 seconds
accuracy : 82.14%
note : in this instance my NB implementation and python's NB always produced exactly the same accuracy

3 sample pixels predicted as apples :
[118, 24, 25]
[108, 2, 4]
[123, 27, 31]

3 sample pixels predicted as leaf:
[143, 166, 36]
[171, 189, 81]
[136, 160, 20]

3 sample pixels predicted as background:
[219, 126, 134]
[192, 205, 117]
[255, 255, 255]