**Machine Learning HW2**

By: Roy Zohar 209896174

Roy Mezan 319042800

**Note**:

We implemented the entire project in python since we are more comfortable in it than in MATLAB. We implemented our own methods for reading the data.

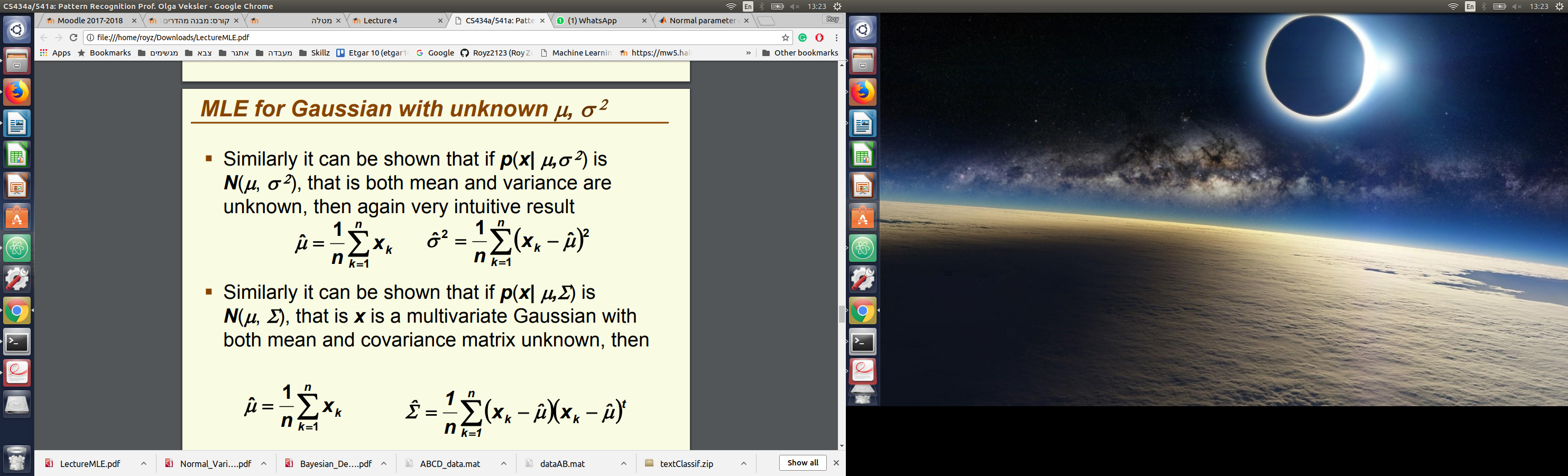
**Question 1:**

a)

We created the 2\*300 normally distributed samples using **gen\_rand.py**. The program gets as in input the mean and standard deviation for a normal distribution and creates a file containing 300 samples of that distribution.

b)

As we have seen in class, the mean parameter of a normal distribution can be found by averaging all of the samples, and similarly the standard deviation can be found by averaging the squared distance from the mean:



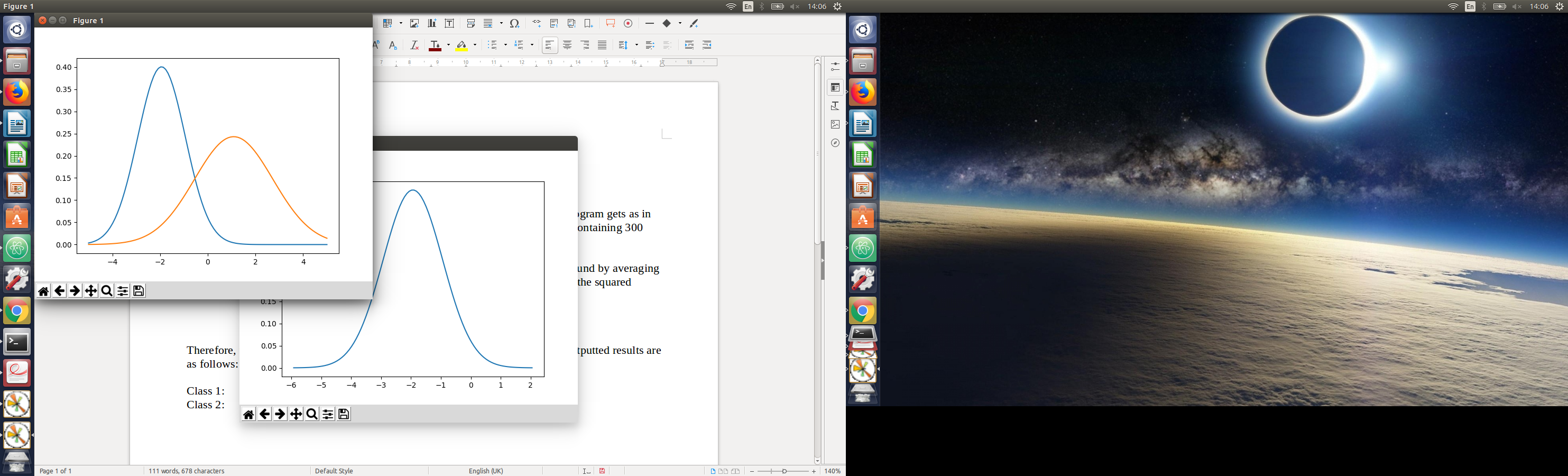
Therefore, we wrote **est\_parms.py**, which found both parameters. The outputted results are

as follows:

Class 1: mean = -1.938 standard deviation = 0.996

Class 2: mean = 1.088 standard deviation = 1.642

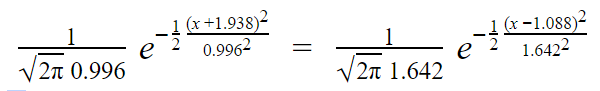
Graphically:



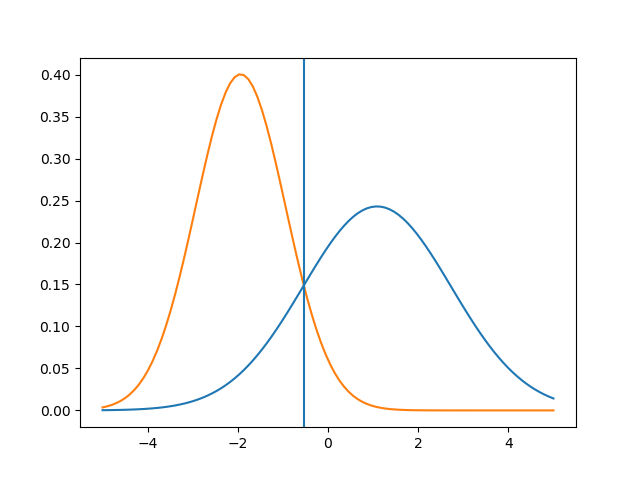
These estimations are fairly similar to the original parameters, with an error of around 0.1. We played around with a larger sample set and saw the error rate go dramatically down afterwards.

c)

We can easily find the decision rule for the plots we found. As we have seen in class, the decision rule can be found at the intersection of the two functions. Algebrically, it can be found like so:



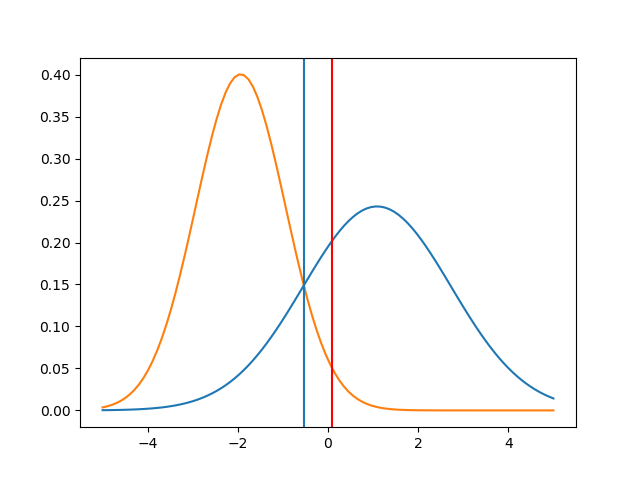
After plugging into a calculator, we get the decision boundary at x =, such that for every value larger then x, we classify the sample as Class 2, otherwise we classify as class 1. Graphically, the boundary appears like so:



Note: We "hardcoded" the boundary into our code and didn’t calculate it from within since we found it easier than dealing with an equation solver in python, but it could have been done.

d)

As we have seen in class, if we increase the weight of misclassifying a sample the boundary will lean towards that class. Using the technique we saw in class, using λ, we can calculate the new decision boundary to be x = 0.0892. Graphically this comes out to be (Can be seen as the new vertical line in red):

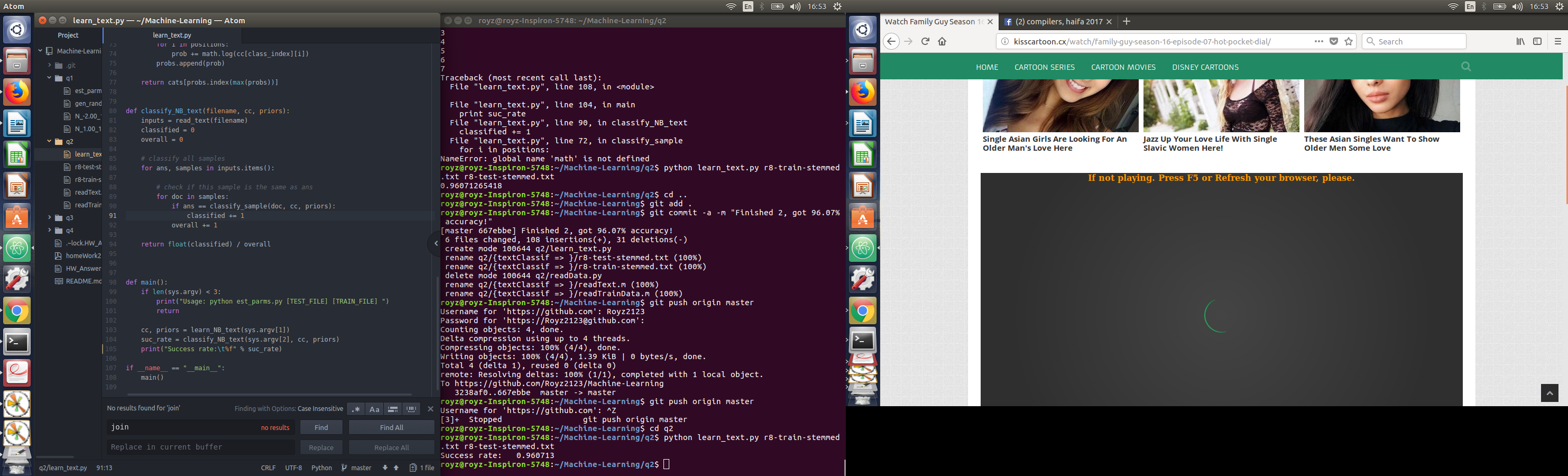


**Question 2:**

a) + b)

The two methods are implemented in the file **learn\_text.py.** We also wrote our own “read data” method since we prefer working in python. To run the program, the training file and the testing file need to be specified as arguments. Note that the learning takes a considerable amount of time.

c)

Our classification rate in the end was 96.07%:

**Question 3:**

a)

We implemented our own PNN in python. We copied the training, validating and testing data into csv files so that we can read them from python. We implemented two modules, **pnn.py** and **classify.py.** After some validation, we found that the optimal parzen window was somewhere between 0.5 to 2.0. For this window size, we got a classification rate of 93.65%:



**Question 4:**

a)

We implemented the KNN in python too. We again copied the training and testing samples to csv format so that our python script could read them. We implemented the "KNN-search" method by ourselves (Not the most efficient from the ways we have seen in class but still runs in reasonable time). After some validation, we found that the optimal K for this data set was K=5. For this K we got a classification rate of 99.5%:

