**1. A brief introduction of the classification method;**

The Naïve Bayes classification method uses a set of training data as prior knowledge to predict an attribute for a set of data that is being tested. The Bayes theorem is (posterior = likelihood \* prior). The prior probability is the probability that an attribute is going to be on value or another. In the example of this project, the attribute is defined as +1 or -1. The prior probability (prior in the theorem) is calculated simply by (num\_plus / total) and (num\_minus / total) where num\_plus is the total number of +1’s in the dataset, num\_minus is the total number of -1’s in the dataset, an finally total is the total number of items in the dataset. The likelihood is defined as the probability of each attribute. In the example of the project where the dataset follows the format: ( label index1:value1 index2:value2 . . . ).

Each index is an attribute and each attribute has an index. Label is the item that we are trying to predict (+1 or -1). Given that information the likelihood is calculated as the probability that a particular attribute will have a particular value. If there were a set that consists of five records where the value of index1 was 1 in three of those five records, then the likelihood that index1 will have the value 1 is 3/5. The likelihood of each attribute is then multiplied together, then multiplied with the prior probability to result in the posterior probability.

**2. The implementation details including core data structures, algorithms, and parameters you chose during the implementation, and discuss why;**

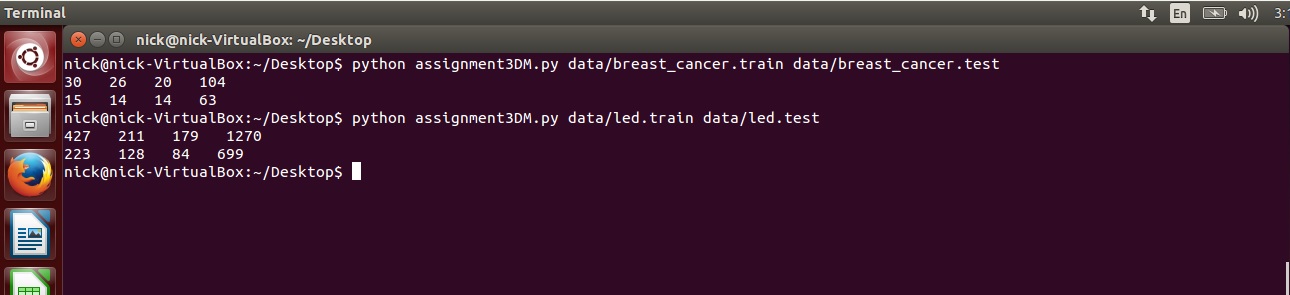
Python seemed like the perfect language to implement this classifier because I feel that it has all the tools needed to solve the problem quickly and efficiently. The main reason for Python being so appealing to use in this situation is because of its built in Dictionary data structure. A Dictionary in Python is essentially a hash table where each entry in the Dictionary has a hashable key and each key stores a value. This made storing (index1:value1 index2:value2) as easy as making a dictionary and using the index value as a key in the Dictionary.

My implementation of this classifier just reads in the files as a large object then it parses this object in the Dataset class, which becomes the stored representation of the dataset. The dataset class is also responsible for calculating the prior probability and keeping track of multiple other values that will aide in calculating likelihood later on. The data is stored as a list of records(record class) in the dataset class. The data is parsed from string form, one line at a time from the read in dataset. Each line is stored as a record object. The record class has a dictionary for storing the index’s and their corresponding values, as well as another variable for storing the label associated with this particular line of data. The record class also counts how many of each label and updates the dataset class accordingly. The final task of the record class is to count the values associated with each index, as they’re read in, in order to calculate likelihood later in the program. It counts the values and they are stored within the dataset class as a list of dictionaries. Each index in the list, [1], [2], [3], corresponds to the index of that particular attribute. Then the dictionary at that index of the list has a key for each value found at that index. The value of the key in the dictionary is the number of occurrences that have been counted. The last thing that the dataset class does before returning to main is to compensate for truncated data. If an index has the value of 0 in a dataset, then it is omitted from the file that the dataset is stored. This means that the dataset class goes over its stored list of record objects and manually inputs any missing values and indexes that were not read in from the file. While it is updating the dataset, it also updates the counted values that will be needed for calculating likelihood later on. Unfortunately this delivers a blow to execution time since the data needs to be traversed twice.

The above process happens for both the training set of data and the testing set of data. Finally, back in main, all the values needed for calculating prior probability and likelihood have already been gathered. Main then uses the training set, and all the information gathered, as prior knowledge to predict the labels of both the training set and the testing set. The program then prints two lines. The first, is the prediction of the labels in the training set using the training set as prior knowledge, and the second is the prediction of the labels in the testing set using the training set as prior knowledge. The output is in the form: true positive in training, false negative in training, false positive in training, and true negative.

The only other things that haven’t been explained are the use of the modifier and potential issues with different datasets. This program is tailored to these datasets such that it assumes that neither the index nor its corresponding value will ever exceed two digits in length. This is because of the way I parse it from string form to integers. If values are used that are greater than two digits in length I don’t think the program will crash, but it definitely will not give the correct output. Finally the modifier. Sometimes a value may only occur at one index in the entire dataset. When calculating likelihood this presents a problem since we shouldn’t multiply the other likelihood values or the probability by 0, as that would ruin our calculations. Also, simply omitting it and not multiplying a likelihood value would not be an answer either. This program uses a modifier value of .01 in the place of issues like this.

**3. All output results you calculated for different datasets;**



**4. Personal thoughts and potential improvement.**

This was a difficult project for me. Wrapping my mind around how the Naïve Bayes method worked and figuring out how to use it on a dataset like this took me a day or so. After finally being able to understand the process, all that was left was to implement it. My implementation definitely suffers from code copying, and I probably could have spent more time naming variables better. However, during this project I was more concerned with getting the correct output values than code cleanliness. The classifier I built is about 270 lines of code from beginning to end. This is about half the length that professor Zhao said his implementation in C++ was. My implementation could probably be refactored to about half the length that it is currently. There may also be a way to further improve execution time by somehow filling in missing data while parsing from input, but I there wasn’t enough time for me to figure it out. My method for parsing the input could definitely be improved to allow for more than values that are just two digits in length, but I went with an easy method I knew as opposed to finding a new, maybe more efficient, method. I am happy that I was finally able to understand the Naïve Bayes method because I was really beginning to worry. Even though I feel that I have a good grasp on how it works my output is still one off from the sample output given on the course website.