Name: Jordan Le
Date: 11/18/19
Email: jor25@pdx.edu

Hw2 Heart Anomalies Write Up:

Approaches:

- I followed the Naive Bayes algorithm to produce matching output as the spect.pdf heart anomaly study on the original spect data.
- In the implementation, I did the majority of my calculations before working with the test data. I first calculated the class probability, the probability of a heart being normal or abnormal. Then I determined the conditional probabilities using the previous class probability calculations and looping through each of the feature values. Specifically, I looked at the probabilities of features being 1's or 0's if a heart is normal or abnormal (ie. conditional probability). This resulted in four probability values for each feature:
    - P(fx = 0 | Normal), Probability of normal given that feature x is 0.
    - P(fx = 1 | Normal), Probability of normal given that feature x is 1.
    - P(fx = 0 | Abnormal), Probability of abnormal given that feature x is 0.
    - P(fx = 1 | Abnormal), Probability of abnormal given that feature x is 1.
    - NOTE that .5 was added to the numerator and denominator of each of the probability calculations, this is to ensure that no probabilities resulted in zero.
- The next step was then taking the log base 2 of each of the probabilities with their .5 numerator and denominator additions. The logging process is done to minimize the calculations needed later on in classification. Note the following log rule from ChiliMath.com:

  Rule 1: $\log_b (M \cdot N) = \log_b M + \log_b N$

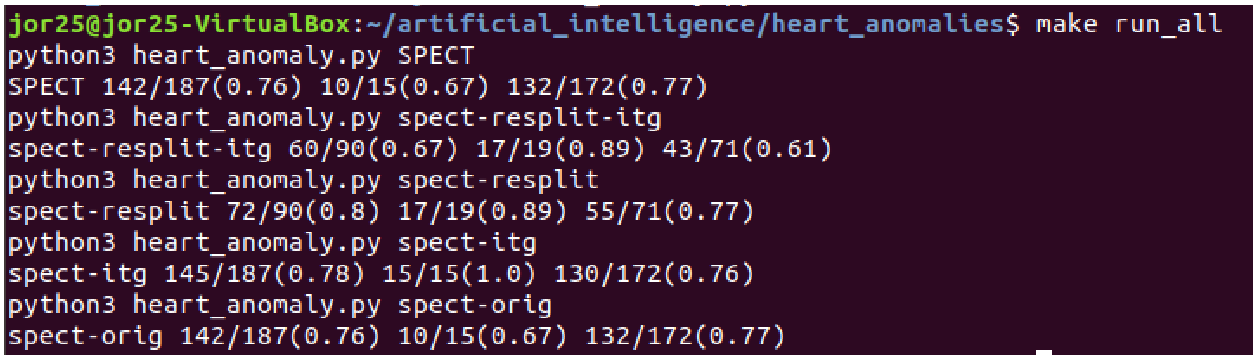
    -
    - This improves performance because addition is much faster for computers to calculate than multiplication.
- After the prework, I have a three-dimensional array of log-based 2 probabilities. I then access this 3D array with specific indices for the heart being normal=1 or abnormal=0, the feature number (0 to (len(data) - 1)), and whether or not the given feature is a 1 or 0. This is the basis for my classifier.
- Finally, I feed my test data into my classifier which uses the 3D combination array and sums up the logarithms for both a normal and an abnormal heart. Then, once it looks through all the features of the test data and finishes summing the logarithms, it selects the highest summation of the two heart classification. It then appends it to a list of predictions and goes through all the test data before comparing it to the label values of each set of features.

- This is where it calculates the accuracy (number correct/total), the true negative rate (number of abnormal hearts/total number of abnormal hearts), and the true positive rate (number of normal hearts/total number of normal hearts). Lastly, I output all this data into various text files. The outputs I receive are displayed below in the results section.

How to Run:

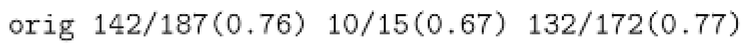
- I created a make file for running the program on all instances at once. Use the commands
    - "make run_all" - runs all the data instances.
    - Specifically, it will run each of these files in order.
    - run_all :
    - python3 heart_anomaly.py SPECT
    - python3 heart_anomaly.py spect-resplit-itg
    - python3 heart_anomaly.py spect-resplit
    - python3 heart_anomaly.py spect-itg
    - python3 heart_anomaly.py spect-orig

Results:

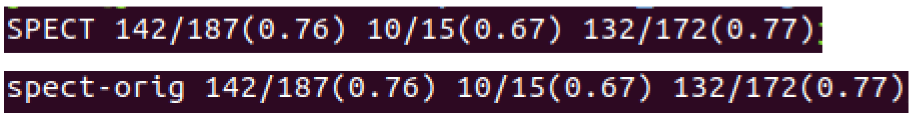
- After implementation, I ran through all the commands with the corresponding data files. I got the following results shown in the screenshot below.
- 

```
jor25@jor25-VirtualBox:~/artificial_intelligence/heart_anomalies$ make run_all
python3 heart_anomaly.py SPECT
SPECT 142/187(0.76) 10/15(0.67) 132/172(0.77)
python3 heart_anomaly.py spect-resplit-itg
spect-resplit-itg 60/90(0.67) 17/19(0.89) 43/71(0.61)
python3 heart_anomaly.py spect-resplit
spect-resplit 72/90(0.8) 17/19(0.89) 55/71(0.77)
python3 heart_anomaly.py spect-itg
spect-itg 145/187(0.78) 15/15(1.0) 130/172(0.76)
python3 heart_anomaly.py spect-orig
spect-orig 142/187(0.76) 10/15(0.67) 132/172(0.77)
```

- The above are the outputs from each of the files' test data. I validated my results by comparing the outputs of the SPECT and spect-orig files with those from the heart anomaly assignment. This comparison is displayed below.



```
orig 142/187(0.76) 10/15(0.67) 132/172(0.77)
```



```
SPECT 142/187(0.76) 10/15(0.67) 132/172(0.77)
spect-orig 142/187(0.76) 10/15(0.67) 132/172(0.77)
```

Questions:

- Which is more important in this application: accuracy of abnormal instances or accuracy on normal instances?
    - Since this is medical data of heart classifications, I want to put in a disclaimer that BOTH classifications are VERY important. However, I'd personally lean more toward the correct classification of abnormal instances (True Negatives). I think absolutely knowing that a heart is abnormal means that preventative actions can be taken to help abnormal heart patients and not harm too many normal heart individuals. Think of it as preventative care. If a person is classified with an abnormal heart, then mark them to be verified by a professional radiologist to verify the Naive Bayes' prediction. Do not let the radiologist see the prediction beforehand to prevent bias. I think it's better to know if something is wrong with the heart and get checked early than to not know something's wrong and find out when it's too late.

- Which dataset seems to give the best results?
    - In terms of accuracy, the best data set was the spect-resplit with 80% (72/90) total accuracy.
    - Best True Negative (Abnormal Hearts): spect-itg with 100% (15/15) true negative rate.
    - Best True Positive (Normal Hearts): spect-resplit with a 77.46% (55/71) true positive rate.
    - Overall though, I would say the spect-itg dataset because of its true negative rate and considering my personal opinion that preemptive care for abnormal hearts may help more individuals. Once again, I'd take the results with a grain of salt and have these results verified with a radiologist who has not seen the prediction.

Optimizations:

- The optimizations I saw were the numpy where and count operations for evaluating various numpy arrays and determining conditional probabilities. It allows me to vectorize the operation for searching two lists for specific matching conditions. Also switching all the multiplications with summations of logarithms. That was given through the pseudo-code described in the assignment document.

Hardware and Software Set up:

- For software development and experimentation set up, I continue to use a 64-bit Ubuntu virtual machine with a 2048 MB base memory. I worked on a virtual machine to continue experimenting freely without directly altering my device. It also seems to perform better than my actual device, so that's nice.

Resources:
Count the number of ones in a numpy array:
https://docs.scipy.org/doc/numpy/reference/generated/numpy.count_nonzero.html#numpy.count_nonzero

Step by step refresher on understanding Naive Bayes probability calculations:
https://www.geeksforgeeks.org/naive-bayes-classifiers/

Count the number of matches in two numpy arrays:
https://stackoverflow.com/questions/42916330/efficiently-count-zero-elements-in-numpy-array

Basic logarithm rules:
https://www.chilimath.com/lessons/advanced-algebra/logarithm-rules/