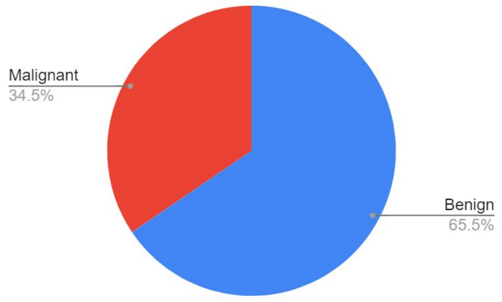
James Levy

1. DataSet detail
    a. What is the dataset
        i. The dataset is from the university of Wisconsin's hospital in Madison. Given a set of features the dataset aims to classify if a breast cancer tumor is benign or malignant. The data was collected over the course of many years to reach a total of 699 results, although 16 have missing information. Overall, the distribution shows us that there are 458 (65.5%) benign tumors and 241 (34.5) malignant tumors in the dataset.



    b. What did I use from and in it
        i. When reading in the data I used all but 1 feature. I cut out the first column as it only contained ID values, nothing that could be used for prediction. The features of importance included clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nuclei, and mitosis. The data also contains 16 incomplete rows. These rows contained a '?' instead of a value for 1 or more features. Incomplete rows were not included in the data or test sets.
    c. How did I split it
        i. In order to split between test and data sets, I used a 75% - 25% split. 75% belongs to the training data.
2. Algorithm description
    a. train()
        i. I use a standard iteration number, lambda parameter, and learning rate in my training function.
        ii. The algorithm mostly consists of nested for loops. One that goes over the number of iterations, and one that updates the weight vector and bias for every test row checked. This process acts as the gradient descent. The weights are updates based off of trying to maximize the margin in the hinge-loss function based on cost. This process is based on these formulas:
            1. $c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$ $\quad c(x, y, f(x)) = (1 - y * f(x))_+$

James Levy

$$w = w - \alpha \cdot (2\lambda w)$$

2. Gradient Update — No misclassification

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

3. Gradient Update — Misclassification

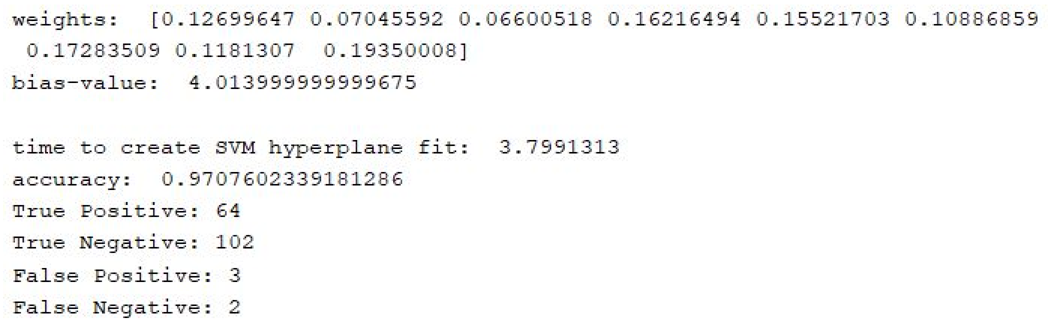
    iii. 1 fault I have in my implementation is that I do not check for convergence on the weight vector, therefore, the time and iterations may become increased as the function never "kicks out" like it may be supposed to.

  b. predict()

    i. Given the test vector x ,bias, and the weights w. A mathematical check is done to see which side of the hyperplane the vector falls on. The check is done via the formula:

      1. `np.dot(x, w) - b`

3. Algorithm results

  a. 

```
weights:  [0.12699647 0.07045592 0.06600518 0.16216494 0.15521703 0.10886859
 0.17283509 0.1181307  0.19350008]
bias-value:  4.013999999999675

time to create SVM hyperplane fit:  3.7991313
accuracy:  0.9707602339181286
True Positive: 64
True Negative: 102
False Positive: 3
False Negative: 2
```

  b. As we can see, the SVM proved to be quite efficient in producing correct answers. Accuracy was at 97% when we were checking for malignant tumors after classification. This proves that the weights and bias value were able to create a decent margin.

4. Runtime

  a. RunTime for the program was fairly efficient. As shown above, the wall clock time taken to train the SVM hyperplane was about 3.8 seconds. This was one of the faster times however, as I have also seen the algorithm take up to 10 seconds.

  b. The training function should take O(n) time, this is because you loop through the rows in the dataset and update the weights accordingly. The outer loop of the gradient descent portion is a constant 1000 iteration. Because it is constant, it does not affect the O(n) time.

  c. The predict function is O(1) time. It contains no loops and simply does a mathematical check to see which side of the hyperplane a given vector would be on.