I am using 4 late hours on this assignment

1a. For the first node, $p_s = 0.75$ and $|S'| = 4$:

$$Entropy = -4 * (0.75 * \log(0.75) + (0.25)\log(0.25)) = 2.249$$

For the first level:

- Package type: Bagged = {yes, yes} and Canned = {yes, no}
- Unit price >$5 = {no, yes} and <$5 = {yes, yes}
- Contains >5 grams of fat = {no, yes} and <5 grams of fat = {yes, yes}

So, we can pick any one of the three. Let us pick package type. We then know that all the bagged food will result in yes, so that node will have an entropy of zero. On the other side, we must consider either unit price or grams of fat:

- Unit price >$5 = {no} and <$5 = {yes}
- Contains >5 grams of fat = {no} and <5 grams of fat = {yes}

So, we can pick either one:

$$Entropy = -2 * (0.5 * \log(0.5) + (0.5)\log(0.5)) = 1.386$$

Thus, we can create our tree, where 1 is equivalent to healthy and 0 is equivalent to not healthy:

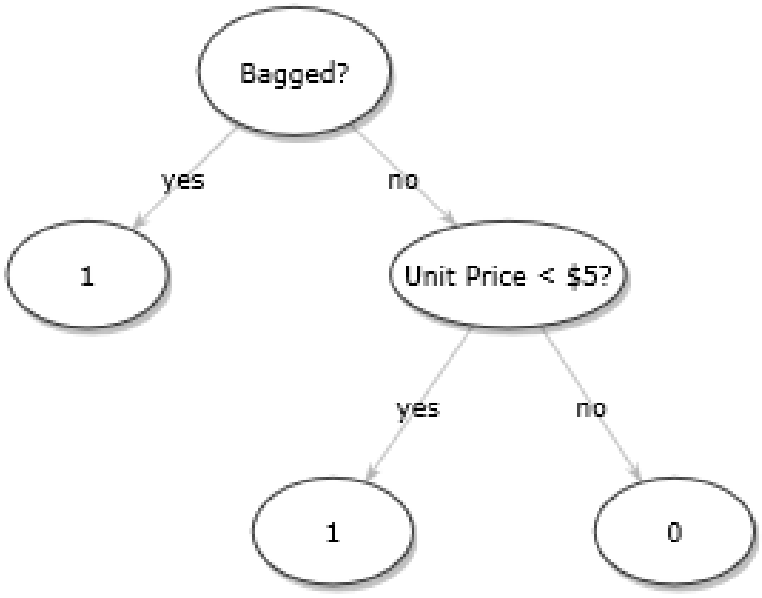
L(S') = 2.249

--- 0.863

L(S') = 0 + 1.386 = 1.386

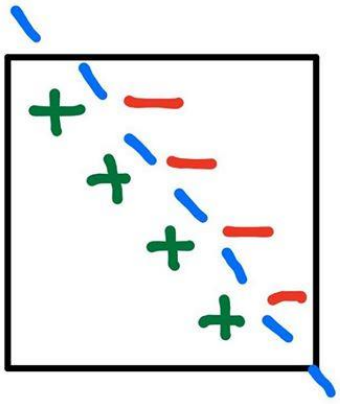
--- 1.386

L(S') = 0 + 0 = 0

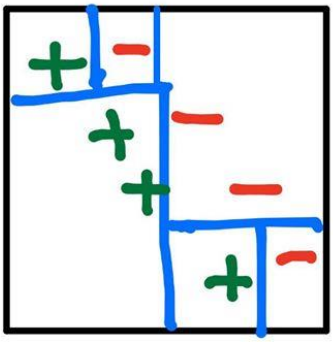


From the root to the first level, we have an impurity reduction of 0.863. From the first level to the second level, we have a reduction of 1.386.

1b. No, a decision tree is not always preferred for classification. A simple case is when the decision boundary is a diagonal line from opposite corners. A linear SVM can easily find the max margin while a decision tree requires complex partitioning:



Decision tree is a lot more complicated:



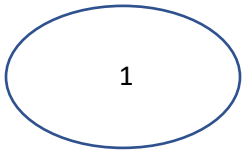
1ci. The entropy at the starting node is

$$4 * (1 - (0.5)^2 - (0.5)^2) = 2$$

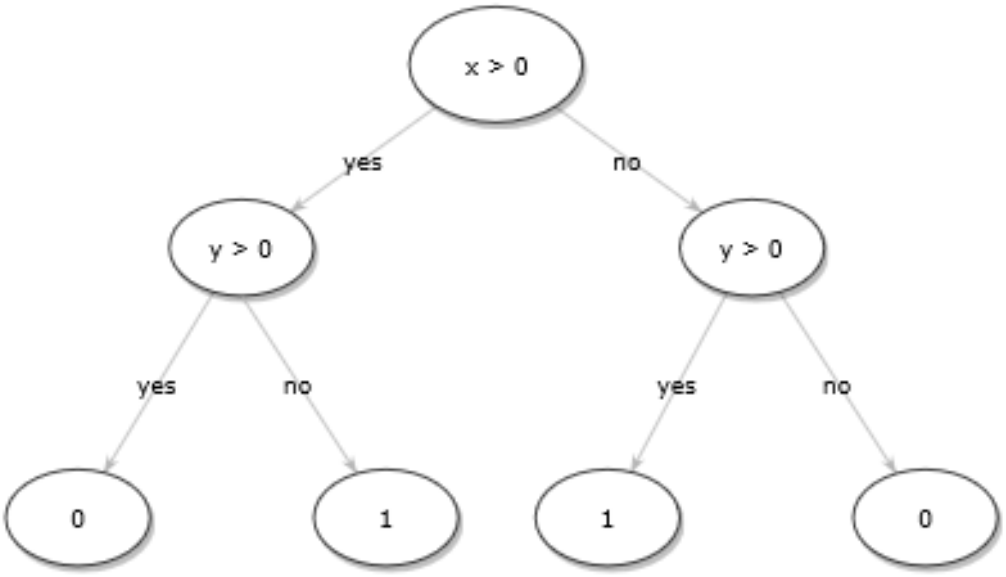
If we divide at any bisection, we will have two sections of two points where one is misclassified:

$$2\big(2 * (1 - (0.5)^2 - (0.5)^2)\big) = 2$$

By the stopping condition, we stop since there was no reduction in impurity. The resulting tree is just the root, and we have a classification error of $\frac{2}{4} = \frac{1}{2}$.



1cii. We can create a tree where 0 corresponds to a red point and 1 corresponds to a green point:



We can use an impurity measure that would match our tree:

$$L(S') = |S'|^2 p_s(1 - p_s)$$

This measure would match since the impurity would reduce between levels because of the squared term. Otherwise, we would stop after the first split.

Pros: This measure would ensure that the tree would continue to split whenever the number of correctly classified points does not change, which handles the problem of stopping early. In the case above, if we used Bernoulli's Variance, we would get $L(4) = 2$ and $2\big(L(2)\big) = 2$, and we would stop at this point. The squared root term in this measure would make sure that this is not an issue.

Cons: This measure encourages splitting, which can lead to overfitting. By continuing to split, the model fits the training data too well which is an obvious sign of overfitting. Even in situations where the number of correctly classified points does not change, we might not want splitting to improve generalization.

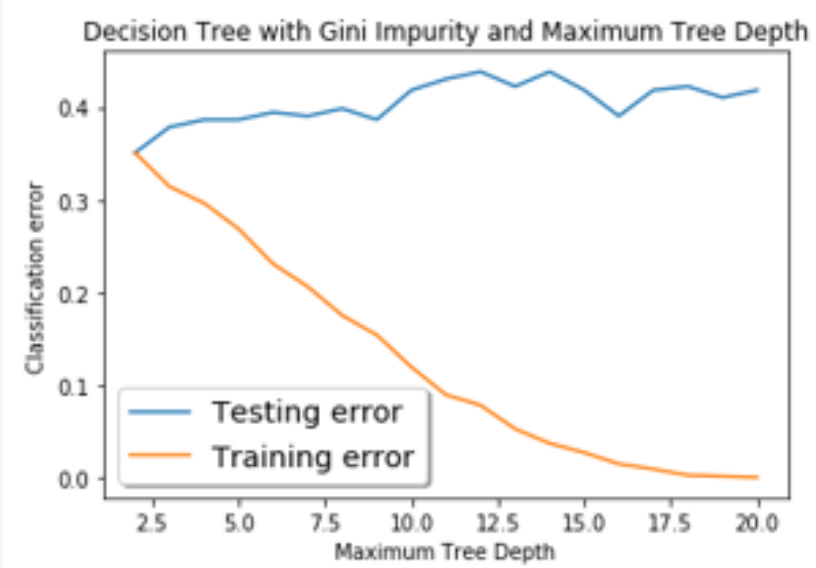
1ciii. When looking at a set of two points, we know that we will only need one split or one internal node. If we look at three points, we know that we will need at most two splits if each point has a unique case. If we have four points, we will need at most three internal nodes. In every case, we can use anywhere from zero (all points are classified the same already) to n – 1 splits (every point has a unique attribute that differentiates the point from every other point). Therefore, the largest number of unique thresholds we might need in order to achieve zero classification training error is 99.

1d. Given in the slides, we know that the number of possible queries we must consider is equal to the number of possible splits, which is equal to D * N. This makes sense because we must consider each feature on each data point in the worst-case. Thus, our worst-case complexity is O(D * N).

2a. My code in the Jupyter notebook gave this graph:


```
Test error minimized at min_samples_leaf = 12
```

2b. My code in the Jupyter notebook gave me this graph:



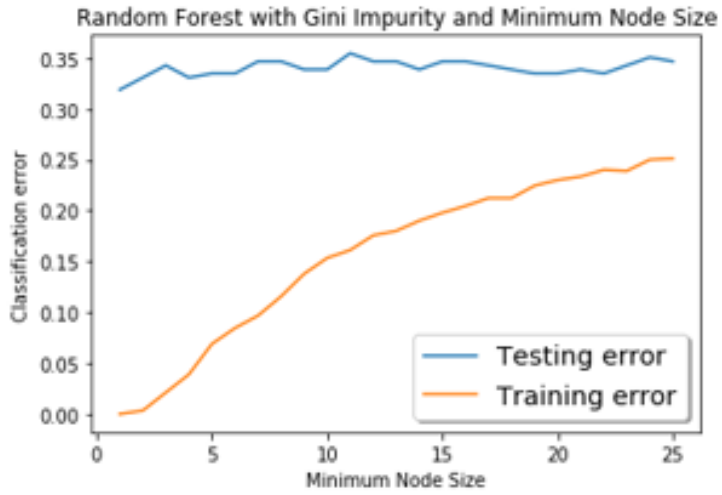

```
Test error minimized at max_depth = 2
```

2c. For the minimal leaf node size, the value that minimizes test error is 12. Stopping early has a great effect on the generalization of the model. As shown in the graph, training error starts low while the testing error is large, so the model is overfitting. This continues to be the case as the training error increases with increasing minimum node size, but the testing error spikes at different minimum node sizes. This shows that we have consequences of stopping too early. However, stopping at the right point has great effects as we obtain the lowest testing error and still have a lower training error. This gives us a good model.

For the maximum depth model, the value that minimizes test error is 2. This graph is basically a flipped version of the one before. Stopping early in this case means decreasing the maximum tree depth. At the very right of the graph, we have a large testing error and low training error, suggesting overfitting. If we stop at the stop at the right point, we achieve the lowest testing error with a normal training error.

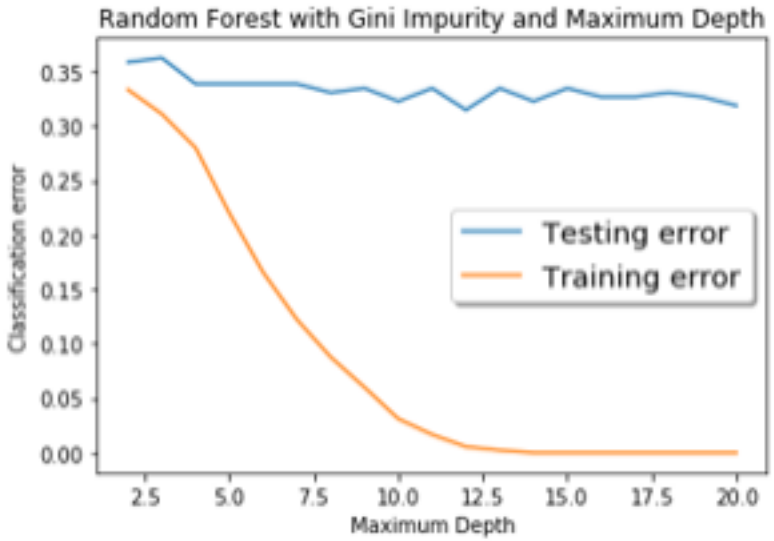
Overall, stopping early helps prevent overfitting and improves generalization, but we cannot stop too early or else the model faces consequences.

2d. My code in the Jupyter notebook gave me this graph:




```
Test error minimized at min_samples_leaf = 1
```

2e. My code in the Jupyter notebook gave me this graph:




```
Test error minimized at max_depth = 12
```

2f. For the minimal leaf node size, the value that minimized test error is 1. For the maximum depth, the value that minimized test error is 12. In both cases, stopping early prevents overfitting. In both cases, if we don't stop early, the training error will go to zero while the testing error consistently stays higher. This is a clear sign of overfitting.

2g. The training error plots are very similar for the corresponding random forest and decision tree plots. However, the testing error for the random forest plots are very consistent across both the plots, while the decision trees have variety across the values. This can be explained by the goal of random forests: reducing variance. By decorrelating trees, we keep the variance consistent overall, which keeps the testing error consistent.

3a. To show:

$$E = \frac{1}{N}\sum exp\left(-y_i f(x_i)\right) \geq \frac{1}{N}\sum 1(H(x_i) \neq y_i)$$

We just need to show

$$exp\left(-y_i f(x_i)\right) \geq 1(H(x_i) \neq y_i)$$

Thus, we consider the two cases: y_i and $f(x_i)$ are the same sign or opposite signs. First, we will consider the case where they are opposite sides:

$$\exp\left(-y_i f(x_i)\right) = \exp(nonnegative\ number) \geq 1$$

$$1(H(x_i) \neq y_i) = 1$$

For this case:

$$exp\left(-y_i f(x_i)\right) \geq 1(H(x_i) \neq y_i)$$

If y_i and $f(x_i)$ are the same sign:

$$\exp\left(-y_i f(x_i)\right) = \exp(nonpositive\ number) \geq 0$$

$$1(H(x_i) \neq y_i) = 0$$

Therefore,

$$exp\left(-y_i f(x_i)\right) \geq 1(H(x_i) \neq y_i)$$

Combining both cases, we have shown that

$$exp\left(-y_i f(x_i)\right) \geq 1(H(x_i) \neq y_i)$$

Thus,

$$E = \frac{1}{N}\sum exp\left(-y_i f(x_i)\right) \geq \frac{1}{N}\sum 1(H(x_i) \neq y_i)$$

3b. We can start with the formula provided from the lecture:

$$D_{t+1}(i) = \frac{D_{t(i)} \exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t}$$

We know that this a recursive function that starts with $D_1(x) = \frac{1}{N}$. From this point, we know that we can recursively plug in this value to get the next. We know we just continuously plug these values in, which multiplies them. We get the product:

$$D_{T+1} = \prod_{t=1}^{T} \frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t N}$$

3c. We know that

$$E = \frac{1}{N}\sum exp\left(-y_i f(x_i)\right)$$

and we have

$$H(x) = sign\big(f(x)\big) = sign\sum \alpha_t h_t(x)$$

Both equations were given. Substituting in:

$$E = \frac{1}{N}\sum exp\left(-y_i \sum \alpha_t h_t(x)\right)$$

$$E = \frac{1}{N}\sum (e^{(\Sigma(-y_i \alpha_t h_t(x_i)))})$$

Where the outer sum is from 1 to N, and the sum in the exponent is from 1 to T.

3d. We need to show:

$$E = \prod_{t=1}^{T} Z_t$$

We know that

$$Z_t = \sum (D_t(i) \exp(-\alpha_t y_i h_t(x_i)))$$

We showed that in part b

$$D_{T+1} = \prod_{t=1}^{T} \frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t N}$$

Plugging this in, we get

$$Z_T = \sum (\prod_{t=1}^{T-1} \frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t N} \exp(-\alpha_T y_i h_T(x_i)))$$

Then doing some algebra:

$$\prod_{t=1}^{T} Z_t = \frac{1}{N} \sum (\prod_{t=1}^{T} \exp(-\alpha_t y_i h_t(x_i)))$$

And the product of exponents is the same as adding the value in the exponent:

$$\prod_{t=1}^{T} Z_t = \frac{1}{N} \sum (e^{(\Sigma(-y_i \alpha_t h_t(x_i)))})$$

Plugging in our equation from part c:

$$E = \prod_{t=1}^{T} Z_t$$

3e. We want to show

$$Z_t = (1 - \epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$$

We must consider when y_i and $h_t(x_i)$ have the same or opposite signs. First, we will consider when they have opposite signs:

$$Z_t = \sum (D_t(i)\exp(-\alpha_t y_i h_t(x_i)))$$

$$Z_t = \sum (D_t(i)\exp(\alpha_t))$$

$$Z_t = (\exp(\alpha_t))$$

Using the formula we are trying to show, we can calculate the same value considering that

$$\epsilon_t = \sum D_t(i) 1(h_t(x_i) \neq y_i)$$

$$\epsilon_t = 1 \text{ since } h_t(x_i) \neq y_i$$

Therefore,

$$Z_t = (1 - 1)\exp(-\alpha_t) + 1\exp(\alpha_t)$$

$$Z_t = (\exp(\alpha_t))$$

Now, we consider when y_i and $h_t(x_i)$ have the same sign:

$$Z_t = \sum (D_t(i)\exp(-\alpha_t y_i h_t(x_i)))$$

$$Z_t = \sum (D_t(i)\exp(-\alpha_t))$$

$$Z_t = (\exp(-\alpha_t))$$

Again, checking that the behavior is the same with $\epsilon_t = 0$ by the formula above:

$$Z_t = (1 - 0)\exp(-\alpha_t) + 0\exp(\alpha_t)$$

$$Z_t = (\exp(-\alpha_t))$$

Therefore, we have shown that the normalizer behaves the same as the equation. So

$$Z_t = (1 - \epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$$

3f. We can differentiate the equation we derived in part e with respect to α_t. Then, we set the derivative equal to zero for minimization:

$$Z_t = (1 - \epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$$

$$Z_t{}' = (1 - \epsilon_t)(-\alpha_t)\exp(-\alpha_t) + \epsilon_t \alpha_t \exp(\alpha_t) = 0$$

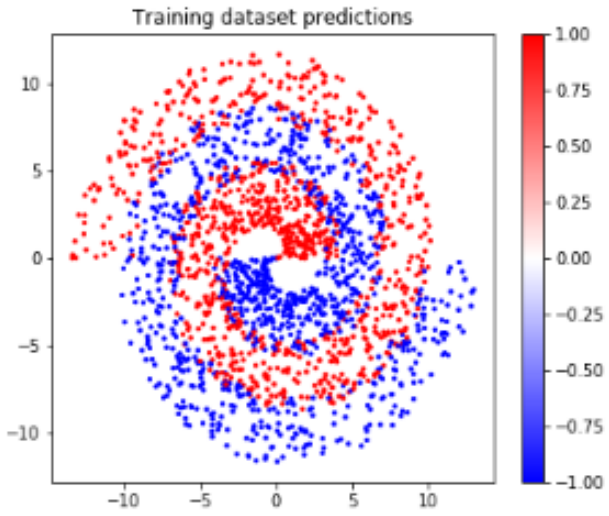
$$\epsilon_t \alpha_t \exp(\alpha_t) = (1 - \epsilon_t)(\alpha_t)\exp(-\alpha_t)$$

$$\exp(2\alpha_t) = \frac{(1 - \epsilon_t)(\alpha_t)}{\epsilon_t \alpha_t}$$

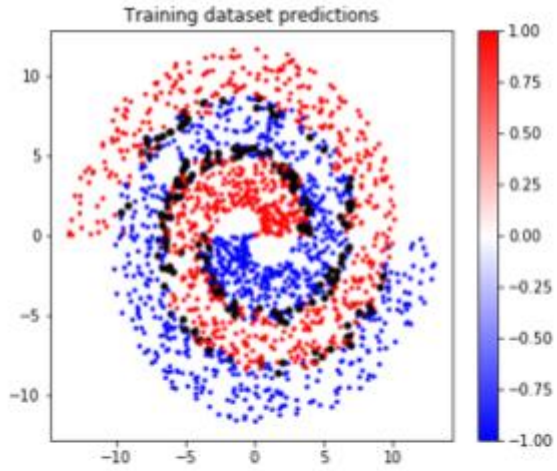
$$\exp(2\alpha_t) = \frac{(1 - \epsilon_t)}{\epsilon_t}$$

$$\alpha_t * = \frac{1}{2}\ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

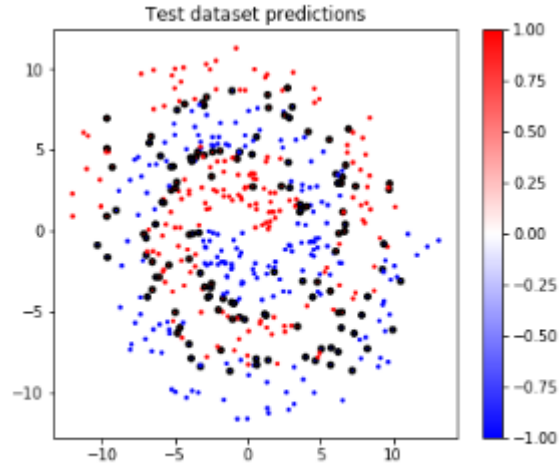
3g. The code has been updated in the notebook.



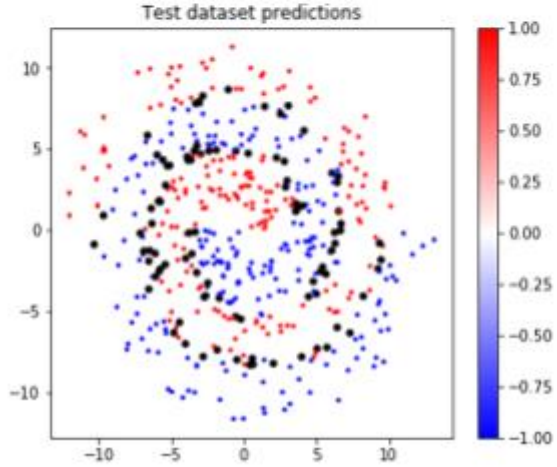

```
Training loss: 0.000000
```



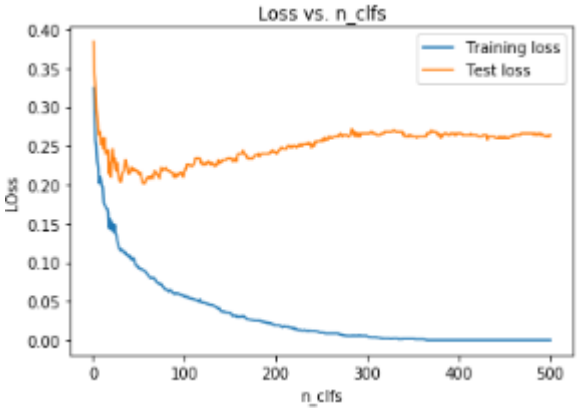

```
Training loss: 0.104500
<Figure size 432x288 with 0 Axes>
```




```
Test loss: 0.258000
```




```
Test loss: 0.186000
```



3h. The gradient boosting loss curve decreases training loss, but testing loss decreases and then increases. The graph is a lot smoother than the AdaBoost curve, and the general trend holds mostly throughout the entire set of points. The final point reached is a situation with zero training loss and around 0.26 test loss. This shows signs of overfitting at the very end.

The AdaBoost loss curve decreases training and testing loss with some jumps along the way. The graph is not smooth at all and has small jumps at almost small set of points. The trend continues through the entire graph until the final value of 0.105 training loss and 0.186 testing loss. This shows a better fit than the gradient boost.

3i. The final value for the testing error for gradient boosting was zero and the final for AdaBoost was 0.105. Obviously, the gradient boosting seems good, but we also must check the testing error. The final value for the gradient boosting was 0.26 and the final for the AdaBoost was 0.186. The gradient boosting performed better on the training dataset, and the AdaBoost performed better on the testing dataset. Overall, we would probably want to use the AdaBoost.

3j. The points that are more frequently misclassified have larger dataset weights. On the other hand, points that are less frequently misclassified have smaller dataset weights. This can see this in the definition of updating the weights, in the exponent.