CS5785: Applied Machine Learning **(Due: 12/02/19)**

Homework 4

Instructor: Serge Belongie *Names:* Eva Esteban Velasco, Simran Rajpal*, Netids:* epe26, sr2258

1 Written Exercises

1.1 Maximum-Margin Classifiers

(a) Sketch the observations and the maximum-margin separating hyperplane, and (c) On your sketch, indicate the margin for the maximal margin hyperplane, and (f) Sketch a hyperplane that separates the data, but is not the maximum-margin separating hyperplane. Provide the equation for this hyperplane, and (g) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

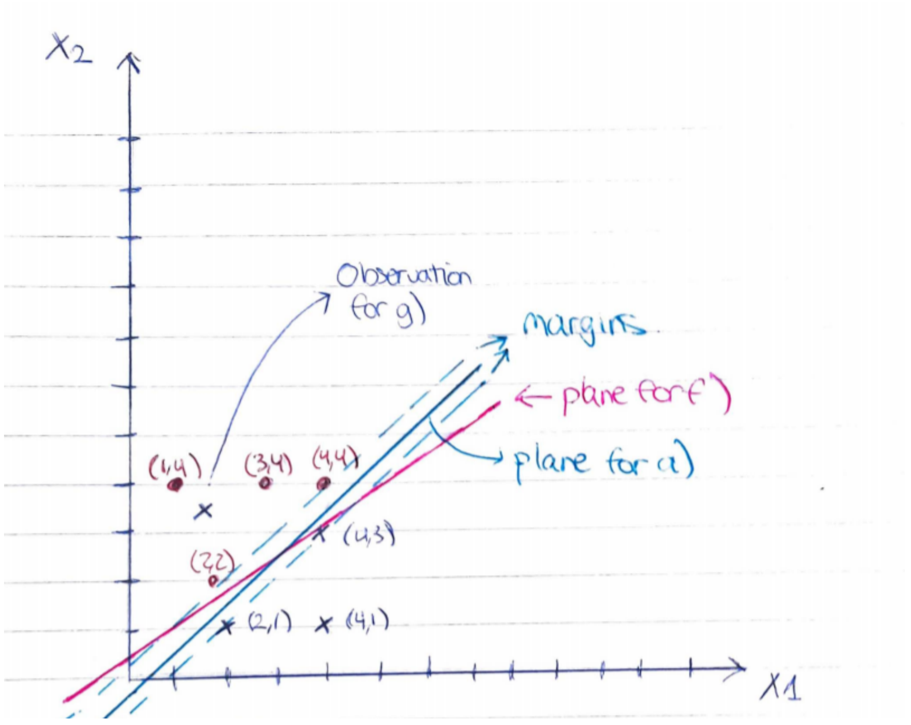


Figure 1: Maximum separation hyperplane and other parameters.

The equation for the hyperplane in part (f) is $2 + 2.5x_1 - 4x_2 = 0$.

(b) Describe the classification rule for the maximum-margin classifier

In order to solve the question, we first found two points in the hyperplane using the diagram above. These points are (2, 1.5) and (4, 3.5), in the form (X_1, X_2). We can then use these points to solve the equation for the separating hyperplane:

$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$ Eq.1

Using the point (2,1.5) in Eq. 1 we get:

$\beta_0 + 2\beta_1 + 1.5\beta_2 = 0$

$\beta_0 = -2\beta_1 - 1.5\beta_2$ Eq.2

Using the point (4,13.5) and Eq.2 in Eq. 1 we get:

$-2\beta_1 - 1.5\beta_2 + 4\beta_1 + 3.5\beta_2 = 0;$

$2\beta_1 + 2\beta_2 = 0;$

$\beta_1 = -\beta_2$ Eq.3

For a value of $\beta_2 = 1$, substituting in Eq. 3 we get $\beta_1 = -1$ and, substituting in Eq. 2 we obtain $\beta_0 = 0.5$.

Substituting these results in Eq. 1, we obtain the equation for the hyperplane as: $0.5 - X_1 + X_2 = 0$.

Taking, for example, the point (1,1), which is Red, we can test that $0.5 - X_1 + X_2 = 0.5 - 1 + 1 = 0.5 > 0$.

Therefore, the rule is: "Classify to Red if $0.5 - X_1 + X_2 > 0$, and classify to Blue otherwise."

(d) Indicate the support vectors for the mazimum margin classifier.

The support vectors are the points (2,2), (2,1), (4,3), and (4,4), in the form (X_1, X_2).

(e) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

The hyperplane maimimzes the margin between itself and the closest observations to it from each class, having only one class on each of its sides. Moving the seventh observation slightly would not affect the hyperplane because many other points are much closer to the plane to the seventh observation. The only way the hyperplane would be affected is if we were to move the seventh observation a lot and make it one of the support vectors, but this will not happen if we only move it slightly.

1.2 Neural Networks as Function Approximators

We found that we could write the function given as a sequence of ReLU functions. We found this out by realizing that we could shift, add and subtract ReLU functions to be able to get the output result. Eq. 2.1 shows the sequence of ReLU functions and Eq. 2.2 distributes the coefficients to allow us to find equations that match the Y_i function given in the problem statement.

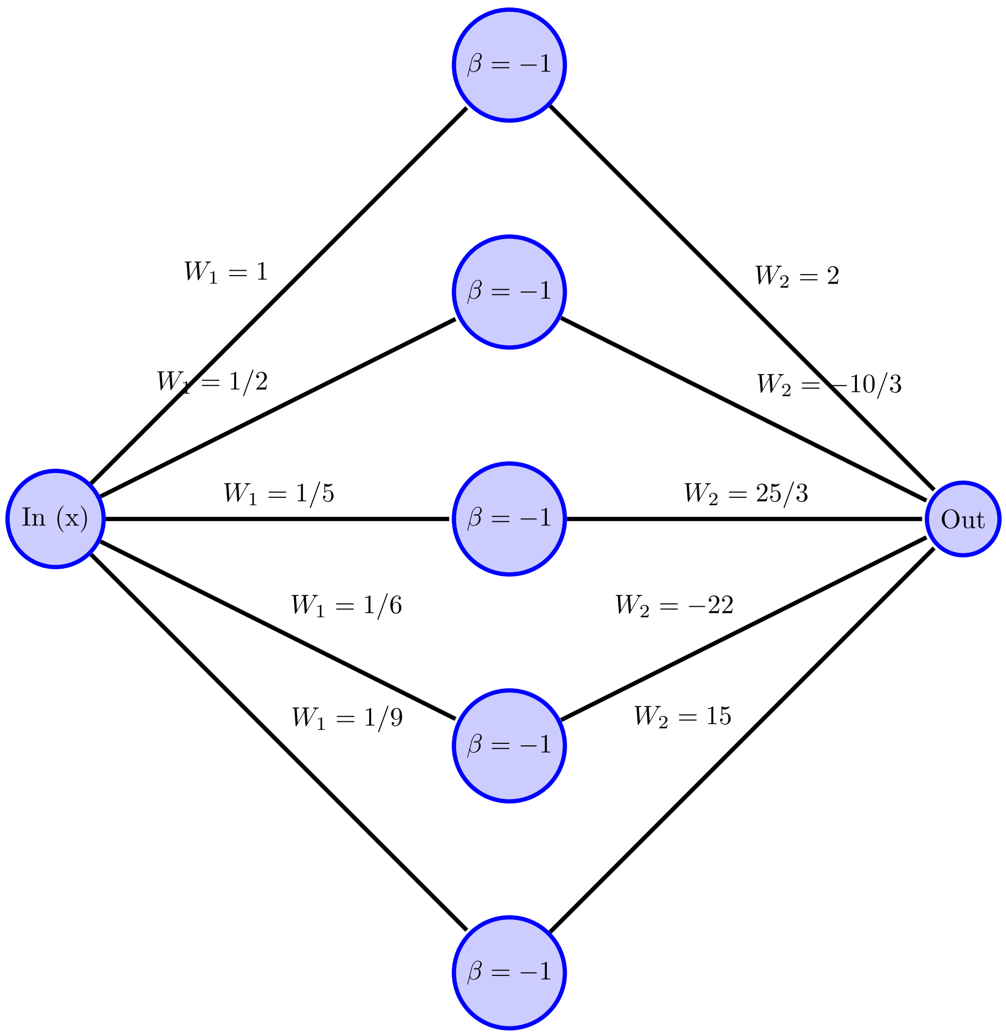
$$2ReLU(x-1) - \frac{5}{3}ReLU(x-2) + \frac{5}{3}ReLU(x-5) - \frac{11}{3}ReLU(x-6) + \frac{5}{3}ReLU(x-9) \tag{1.1}$$

$$2ReLU(x-1) - \frac{10}{3}ReLU(\frac{1}{2}x-1) + \frac{25}{3}ReLU(\frac{1}{5}x-1) - 22ReLU(\frac{1}{6}x-1) + 15ReLU(\frac{1}{9}x-1) \tag{1.2}$$

From this Eq. 2.2 we know the following:

- We need one hidden layer
- There are five units within the hidden layer
- The weights going into the hidden layer are $W_1 = [1, 1/2, 1/5, 1/6, 1/9]$
- The biases of each neuron in the hidden layer are $\beta = -1$
- The weights going out of the hidden layer are $W_2 = [2, -10/3, 25/3, -22, 15]$

We can show this network in the graph below.


$\beta = -1$
$\beta = -1$
$\beta = -1$
$\beta = -1$
$\beta = -1$
In (x)
Out
$W_1 = 1$
$W_1 = 1/2$
$W_1 = 1/5$
$W_1 = 1/6$
$W_1 = 1/9$
$W_2 = 2$
$W_2 = -10/3$
$W_2 = 25/3$
$W_2 = -22$
$W_2 = 15$

2 Programming Exercises

2.1 Approximating Images with Neural Networks

(a) Describe the structure of the network

This is a 2-layer neural network with 7 hidden layers. Each hidden layer has 20 neurons that perform weighted additions of the inputs and passes them throuh a ReLU (Rectified Linear Unit) activation function. The first layer takes the pixel's (x,y) position on the grid, and the network predicts the (r,g,b) color at that point using a regression layer as its loss layer at the end.

(b) What does "Loss" mean and what is the Loss Function?

The loss quantifies the of the predicted output image from the actual output image. In this example, the loss layer is an L2 regression layer. It penalizes $sum(\|x - y\|^2)$, where x is the input and y is the user-provided "correct" values.

(c) Plot loss for the first 5,000 iterations

For this question we used the plt() function in the matplot library [1]. To obtain the loss values, we ran $\$('\#report').bind("DOMSubtreeModified", function()\{console.log(counter, smooth\_loss)\});$ and saved the loss values for the first 5,000 iterations from the console log into a Comma Separated Values (CSV) file. We then read this CSV from the Python Notebook using the function read_csv() in the pandas library [2]. Figure 2 shows the resulting plot.

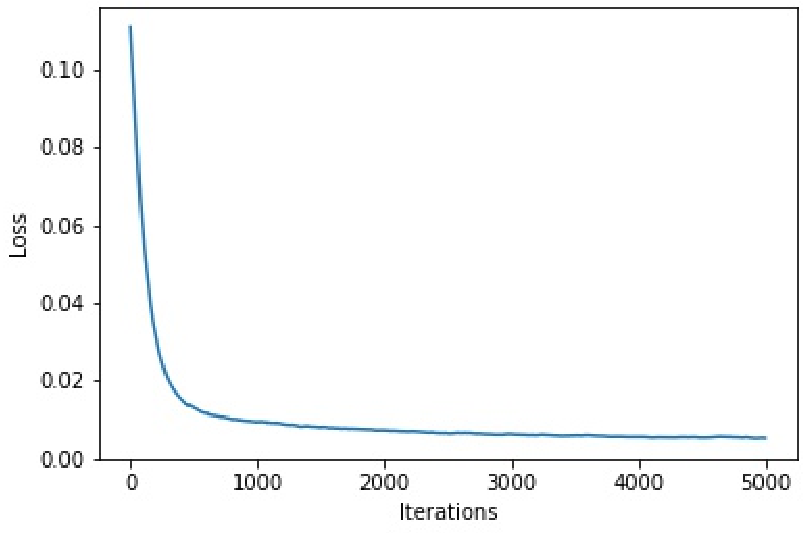


Figure 2: Loss for the first 5,000 iterations.

We can see from Figure 2 above that the loss decreases in an exponential fashion as the number of iterations increases. The loss starts at a value of 0.1109 in Iteration 1 and eventually reaches a value of 0.0051 for iteration 5,000. Therefore, the network has 21 times less loss at iteration 5,000 than at iteration 1 (0.1109/0.0051 = 21.7450980392).

(d) Lowering the learning rate every 1,000 iterations

For this question, the learning rate was reduced every 1,000 iterations. The learning rate was set to:

- 0.01 at iteration 1
- 0.004 at iteration 1,000

- 0.002 at iteration 2,000
- 0.0007 at iteration 3,000
- 0.0002 at iteration 4,000

We extracted the resulting data using the command explained in section c) of this question, and plotted it using the function plt(). The output can be found on Figure 3.

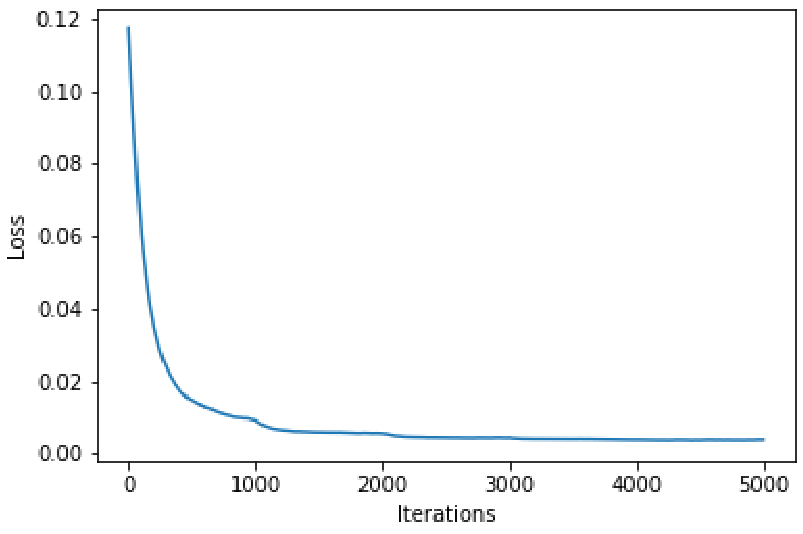


Figure 3: Loss for the first 5,000 iterations decreasing the learning rate every 1,000 iterations.

We can see from Figure 3 that reducing the learning rate every 1,000 iterations slightly reduced the loss. While the loss still follows an exponentially decreasing trend, it went from 0.117511858 at iteration 1 to 0.003637011 at iteration 5,000. Hence, the loss is 32 times smaller at iteration 5,000 than at iteration 1 (0.117511858/0.003637011 = 32.3100089607), compared to the 23 times result we obtained in c).

(e) Removing layers from the network

For this part, we commented each hidden layer of the network, going from a network of 7 hidden layers, to a network of 6 hidden layer, to a network of 5 hidden layers and so on, until the network became a 1 hidden layer network. We then plotted the results using the same techniques as the ones explained in the previous parts of this question. We also observed the reconstructed image of the cat, to determine what the minimum number of hidden layers was needed for quality not to drop noticeably. As the number of hidden layers increased, the time taken to run 5,000 iterations also increased. Figure 4 shows the resulting plots.

We concluded that at least 3 hidden layers were needed to distinguish, as a user, that the image is a black and white cat. Using 4 hidden layers gave a more accurate reconstruction, but from 5 layers and above the increase in accuracy was barely noticeable. We can also see this represented in the loss graph. Even though the loss decreases as the number of hidden layers increases, this reduction in loss is almost negligible from 4 hidden layers and above, as we can see from the graph. For this part, all iterations were ran with a learning rate of 0.01.

(f) Adding layers to the network

For this part we added 3 more hidden layers to the network one by one, going from a 7 hidden layer network, to an 8 hidden layer network, and so on, until we reached a 10 hidden layer network. We then saved and plotted the results using the same procedure as the one used in part e) of this question. The results can be found on Figure 5.

From the graph in Figure 5, we can see that adding hidden layers to the network significantly reduces the

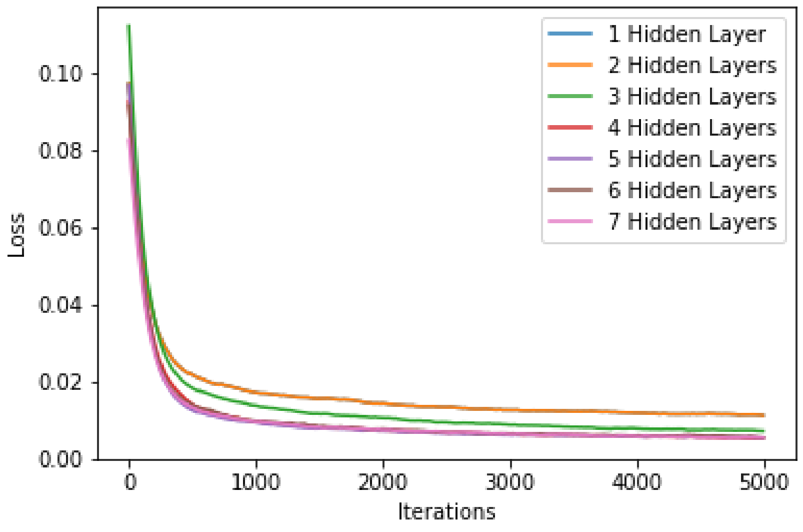


Figure 4: Loss for the first 5,000 iterations with decreasing number of hidden layers.

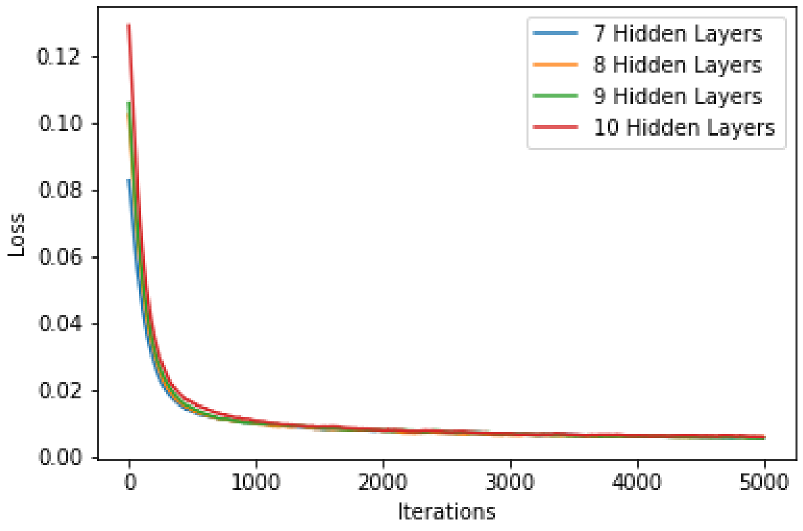


Figure 5: Loss for the first 5,000 iterations with increasing number of hidden layers.

loss i.e increases the accuracy for the first few iterations, but from iteration 500 on it does not reduce the loss noticeably.

2.2 Random Forests for Image Approximation

(a) We started with a picture of Mona Lisa [3].

(b) Preprocessing the Input

We built the training set by sampling 5000 random coordinates from the image. We did this by using numpy.randint [4] to take a uniform random sample for the x coordinate and for the y coordinate.

It would not make sense to preprocess the coordinate values because they are just positions. We decided that standardization techniques, mean subtraction, and unit normalization would not be help for with the input data because it is just coordinates.

(c) Preprocessing the Output

The only preprocessing of output data that needs to be done is unit normalization, dividing all the pixel values by 255 to get them to be between 0 and 1. Getting all the data between 0 and 1 will ensure that all of the data is within a smaller range that the random forest will be able to handle better. We decided to regress all three values of rgb at once, so we do not need to do any image flattening to grayscale.

(d) Random Forest

We fed the training coordinates and pixel values to the sklearn RandomForestRegressor function [5] with a max tree depth of 20. This function creates an appropriate number of trees in the forest to properly train the model. Then, we applied the predict function on all of the mona lisa coordinates to get the image that results in Figure 6.

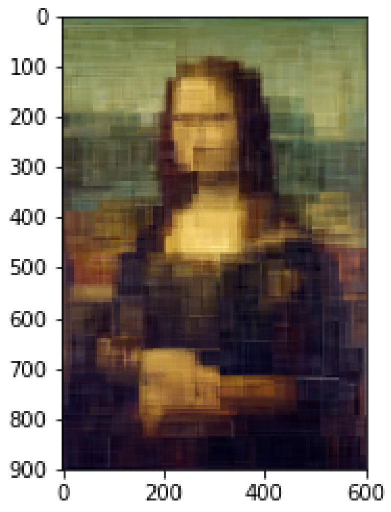


Figure 6: Random Forest Mona Lisa with max_depth = 20

(d) Experimentation

(i) Depth

We called the RandomForestRegressor on the training set and predicted the mona lisa image for a single tree with depths 1, 2, 3, 5, 10, and 15. This can be seen in Figure 7. We found that as we increased the depth of the tree the quality of the image reconstruction increases. For small changes in depth, there is a big change in quality. This is because as we increase the depth of the tree the number of weak classifiers that we are using increases. Increasing the number of weak classifiers gives us a stronger classifier.

(ii) Number of Trees

We called the RandomForestRegressor on the training set and predicted the mona lisa image for depth=7 with 1, 2, 3, 5, 10, and 15 trees. This can be seen in Figure 8. The image clarity slightly increases as the number of trees increases and then begins to blur. The change is not very noticeable. This is because with the increase in the number of trees you are not increasing the complexity of the classifier within one tree. The complexity of the classifier within one tree remains the same as before. When you increase the number of trees, the different trees might have slight differences in their classifiers and when all of the trees are combined, this is

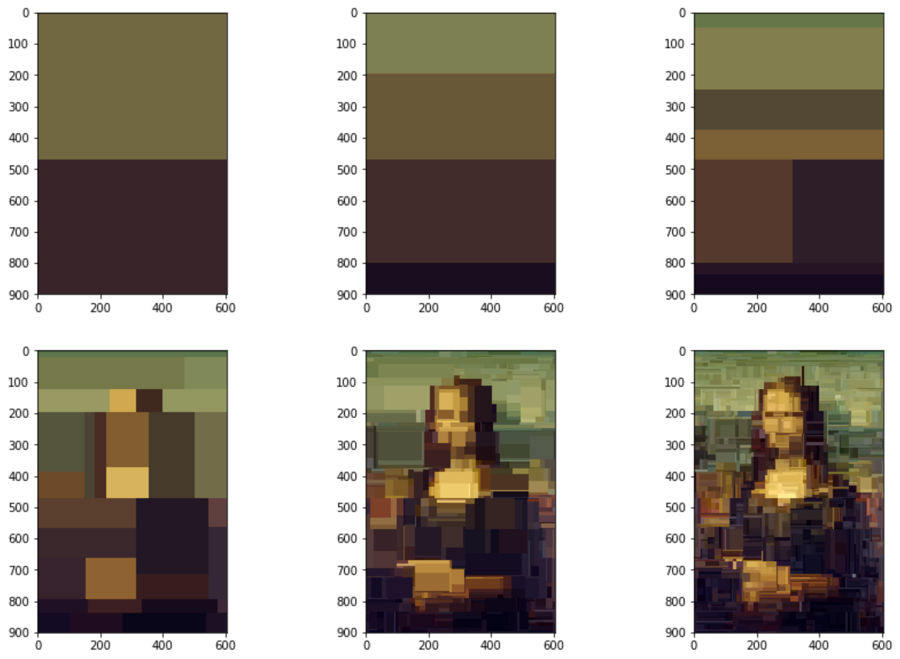


Figure 7: Random Forest Mona Lisa with One Tree and Depths 1, 2, 3, 5, 10, and 15 in Increasing Order

what we see in the images above. Thus, when there are fewer trees the images look more discrete because there is less variation because of the fact that there are fewer trees. When there is a medium number of trees you have the most discrete image because the variation in the different trees may be helping us to get more of the features represented. Finally, when there are too many trees it blurs because of the averaging of variation in many trees.

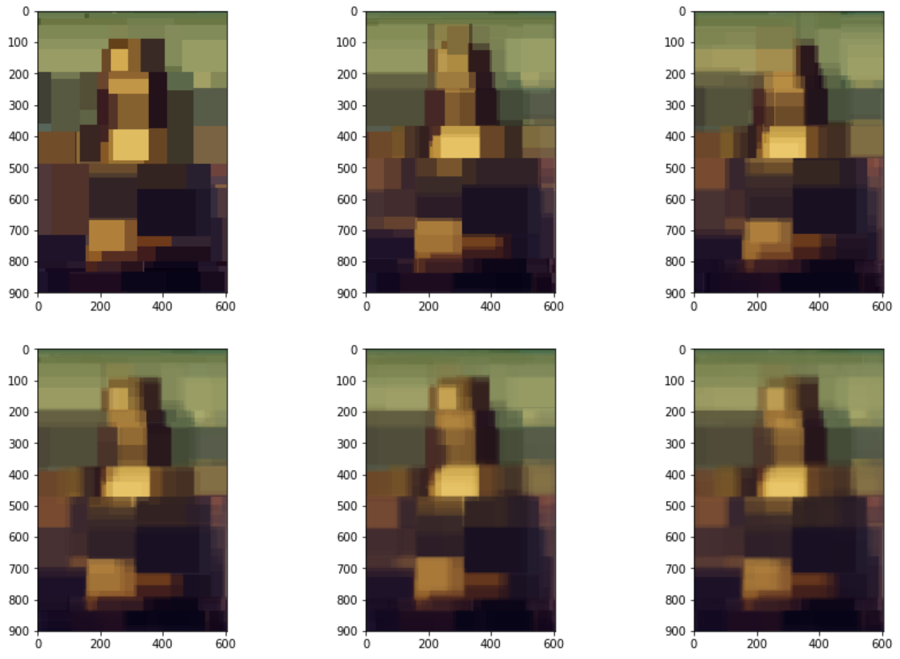


Figure 8: Random Forest Mona Lisa with Depth 7 and Number of Trees 1, 2, 3, 5, 10, and 15 Increasing

(iii) KNN

As a baseline we repeated the experiment with a k-NN regressor for k=1. We used the sklearn KNeighborsRegressor function for this [5]. This is shown in Figure 9. The KNN regressor for k=1 gives a very similar result to the Random Forest regressor with depth=20. This is because the random forest regressor is creating several very deep classification trees in order to determine what the mona lisa photo looks like, which is essentially just as complex as looking at the nearest neighbor of each point. Both images are just as boxy because that is how pixels work (in other cases KNN might be more jagged).

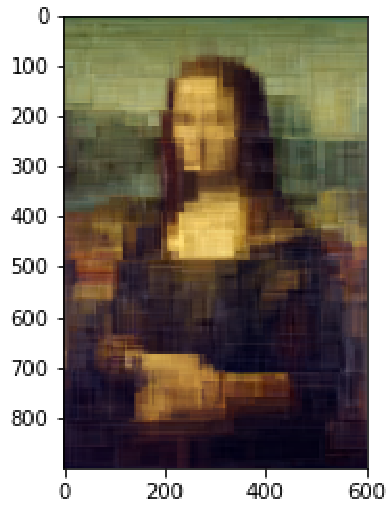


Figure 9: k-NN Regression Mona Lisa Output

(iv) Pruning

Some pruning strategies are changing the maximum depth, changing the number of leaf nodes, and changing the size of the sample leaf. We already performed the experimentation of changing the the maximum depth. However, we can perform these two other pruning techniques to see how it will will affect the results.

One technique we attempted was changing the number of maximum leaf nodes. We applied the RandomForestRegressor varying the number of maximum leaf nodes over 10, 20, 30, 50, 100, and 500. The results are showing Figure 10. We can see from that increasing the maximum number of leaf nodes leads to better accuracy of the image. This is because the more leafs the smaller the image sections are, and the more granular the reconstruction is.

Another technique we attempted was changing the size of the sample leaf. As allowed by the RandomForestRegressor library, we tried values of 30, 100, and 300. This is shown in Figure 11. We found that increasing the minimum number of samples required to be at the leaf node increases image quality. This is because by increasing the minimum number samples required to be at the leaf node we are effectively reducing the depth of the decision tree by reducing its need to grow deeper and lower the number of samples at the leaf node. This decreases the error of the leaf node but also reduces the granularity of the predicted image.

(f) Analysis

(i) Decision Rule

The decision rule for this decision tree is the splitting function for a regression tree. The decision rule is $x_j <= threshold$ to go to the left subtree and $x_j > threshold$ to go to the right subtree. Here we define j as the splitting variable and the threshold is the split point. The RandomForestRegressor decides this split point and splitting variable for each node through a minimization algorithm over different spliting variables. The splitting variables are the features and the split points are the thresholds. The splitting rule for the root node of the first tree in the forest is $x_0 <= 468.5$.

(ii) Why the figure looks the way it does?

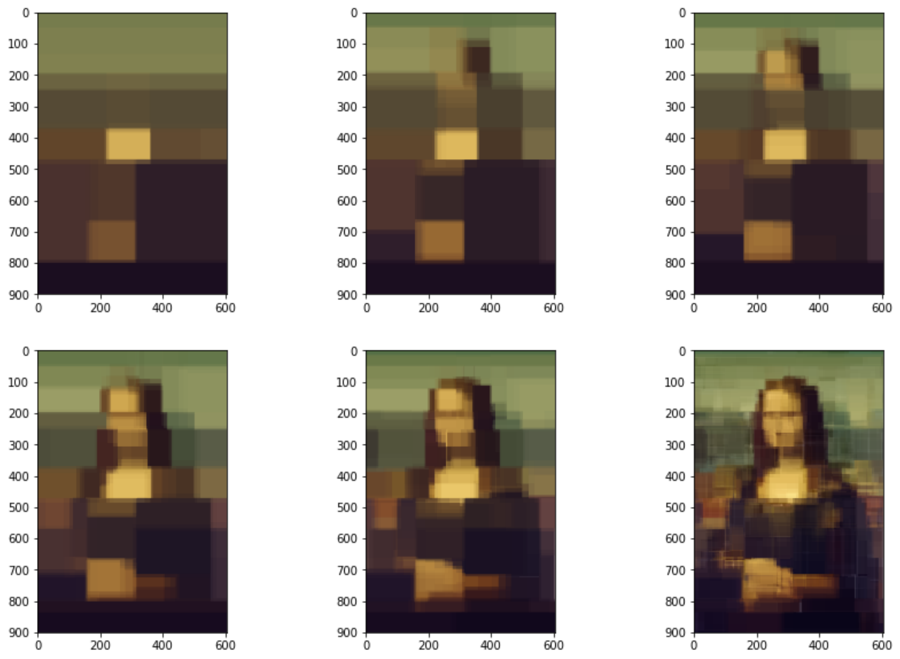


Figure 10: Maximum Leaf Node Pruning, Increasing Order of Leaves Over 10, 20, 30, 50, 100, 500

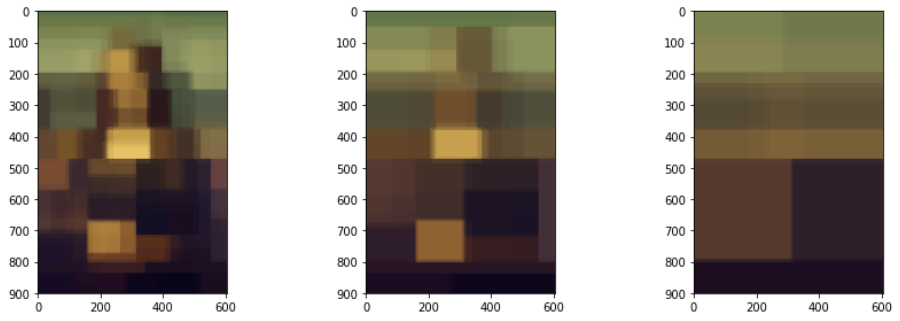


Figure 11: Maximum Samples at Leaf Node Pruning, Increasing Order of Samples Over 30, 100, 300

The resulting image looks like there are square patches of color of different sizes that are arranged in almost a circular way. The reason it looks like this is because the decision tree is made of a bunch of weak classifiers. Each of these weak classifiers creates a split in the decision tree and therefore a split in the image. This is why the patches of color are square and are arranged in this way.

(iii) Number of Square Patches

The number of square patches of color in the resulting image if there is only one tree in the forest is 2^d, where d is the depth of the tree.

(iv) Number of Square Patches for Forest

If the forest contains n decision trees then there are n trees of depth d. This means that the resulting image for each of these trees independently is 2^d. However, in a random forest we are combining the tree estimators. Thus, it is possible for there to be $n(2^d)$ square patches of color in the resulting image. However, this may not be obvious because the differences in the trees may be so subtle that many will overlap and leave us with fewer, perhaps more blurred patches of color. The patches of color that we are able to see may actually be closer to 2^d.

References

[1] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science Engineering*, 2007.

[2] W. McKinney, “Data structures for statistical computing in python,” *Proceedings of the 9th Python in Science Conference*, pp. 51–56, 2010.

[3] Wikipedia. ”https://www.wikipedia.com”[Accessed 11-18-2019].

[4] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 2006, vol. 1.

[5] A. G. e. a. Fabian Pedregosa, Gaël Varoquaux, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, 2011.