CS 5785: APPLIED MACHINE LEARNING

HOMEWORK 4

November 27, 2017

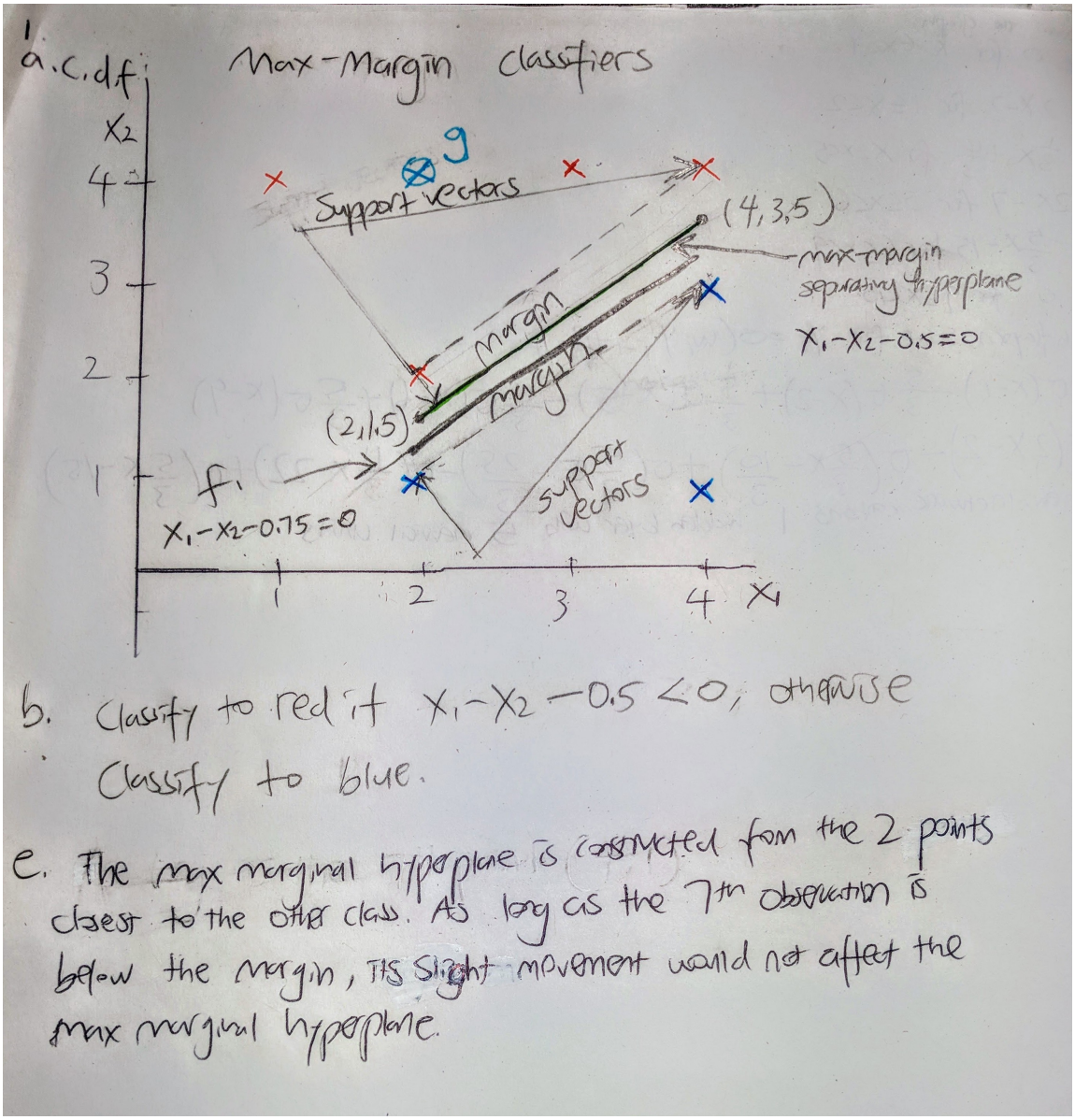
Sarah Le Cam - sdl83
Yunie Mao - ym224

Cornell Tech

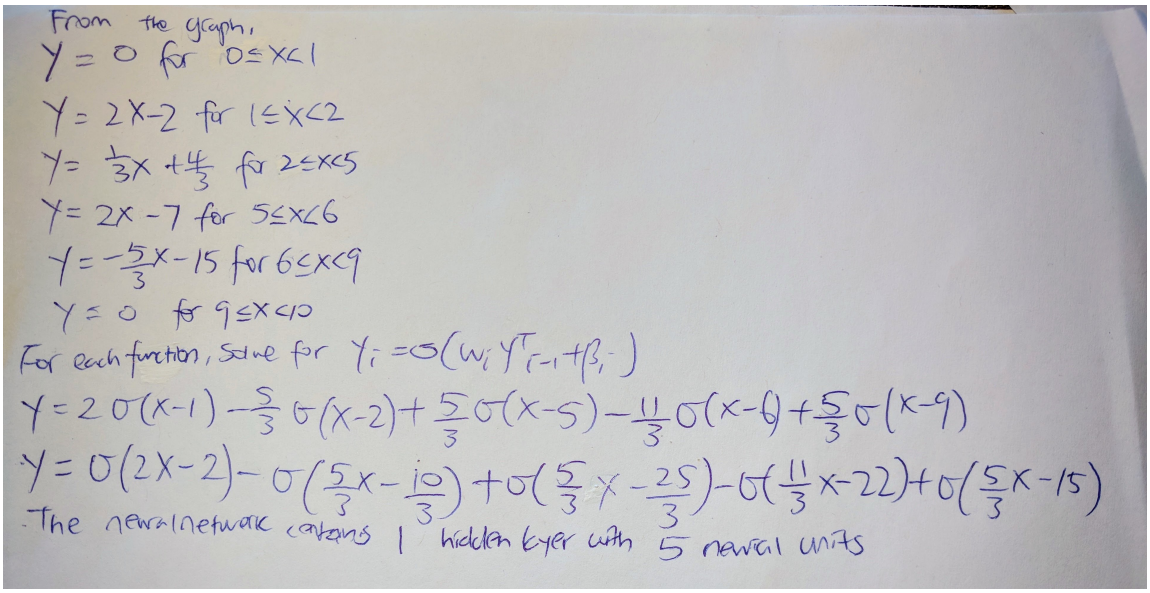
Contents

WRITTEN EXERCISES

Question 1



Question 2


From the graph,
$Y = 0$ for $0 \le x < 1$
$Y = 2x - 2$ for $1 \le x < 2$
$Y = \frac{1}{3}x + \frac{4}{3}$ for $2 \le x < 5$
$Y = 2x - 7$ for $5 \le x < 6$
$Y = -\frac{5}{3}x - 15$ for $6 \le x < 9$
$Y = 0$ for $9 \le x < 10$
For each function, solve for $Y_i = \sigma(w_i Y_{i-1}^T + \beta_i)$
$Y = 2\sigma(x-1) - \frac{5}{3}\sigma(x-2) + \frac{5}{3}\sigma(x-5) - \frac{11}{3}\sigma(x-6) + \frac{5}{3}\sigma(x-9)$
$Y = \sigma(2x-2) - \sigma(\frac{5}{3}x - \frac{10}{3}) + \sigma(\frac{5}{3}x - \frac{25}{3}) - \sigma(\frac{11}{3}x - 22) + \sigma(\frac{5}{3}x - 15)$
The neural network contains 1 hidden layer with 5 neural units

APPROXIMATING IMAGES WITH NEURAL NETWORKS

Summary

We looked at the demo provided by ConvnetJS, which contains an implementation of a convolutional neural network. This image painting demo approximates an image using a neural network of 9 layers. We plotted the loss function of the network over 5,000 iterations and analyzed the loss convergence for varying learning rates. Finally, we experimented with changing the network structure by removing and adding layers to the network and analyzed the effects of these changes on the overall image quality.

Question 1 (a)

The network has 9 layers, which follow the following structure:

- 1 input layer with 2 inputs: (x,y)
- 1 regression layer with 3 outputs: (r,g,b)
- 7 fully connected layers with 20 neurons, followed by ReLu (rectified linear unit non-linearity) activation, that are used to extract features in the image

Question 1 (b)

In the context of the neural network, loss refers to the weighted sum of the loss in the current iteration and the last iteration. According to the source code on Github, the actual loss function for a given iteration is computed using the squared errors.

Loss: $smooth\_loss = 0.99 * smooth\_loss + 0.01 * loss$

```
for (var i=0;i<this.out_depth;i++) {
        var dy = x.w[i] - y[i];
        x.dw[i] = dy;
        loss += 0.5*dy*dy;
}
```

Question 1 (c)

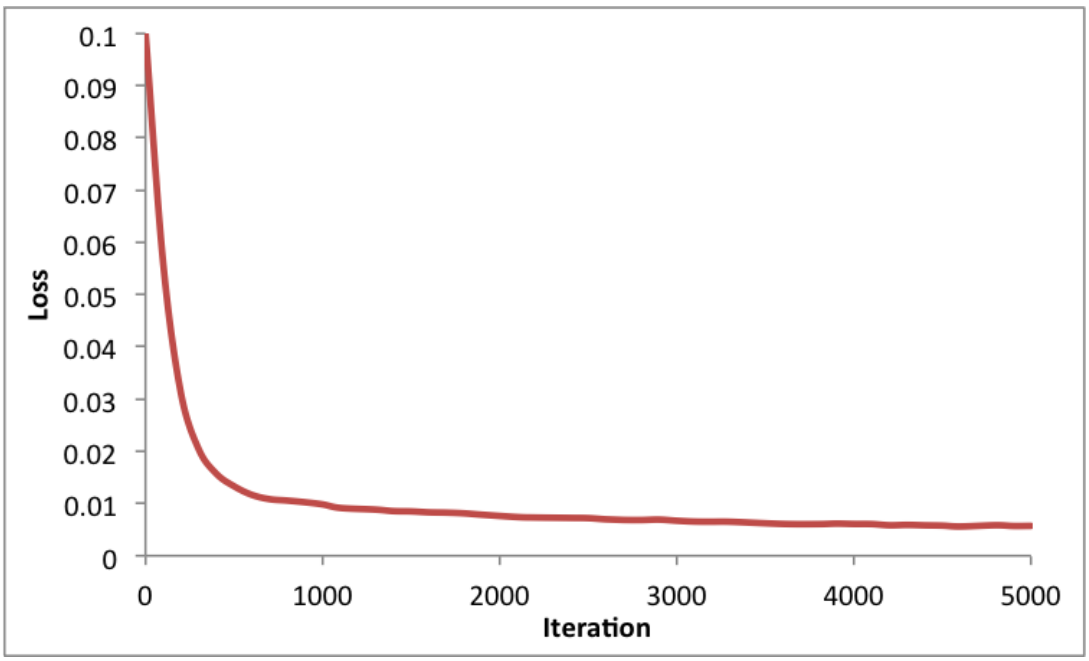


Figure 1: Loss over 5,000 Iterations

With a learning rate of 0.01, the loss function converges to approximately 0.0055 after 5,000 iterations. Using the below Javascript function in the console, we obtained the loss after each 100 iteration interval and plotted the loss function over the first 5,000 iterations. Eventually, the loss converges to 0.0026 past the first 5,000 iterations.

```
setInterval(function() {
  if(counter%100==0) {
    console.log(counter, smooth_loss)
  }
}, 1000/60);
```

Question 1 (d)

We experimented with the learning rate schedule by halving the learning rate after every 1,000 iterations. We then plotted the loss function for the neural network with the varying learning rate against the fixed learning rate over the first 5,000 iterations. We noted that, by halving the learning rate at n iterations, the network converges to the same training loss as

before, only at a faster rate. Therefore, beyond a certain number of iterations, lowering the learning rate does not lower the loss function compared to running the network with a fixed rate.

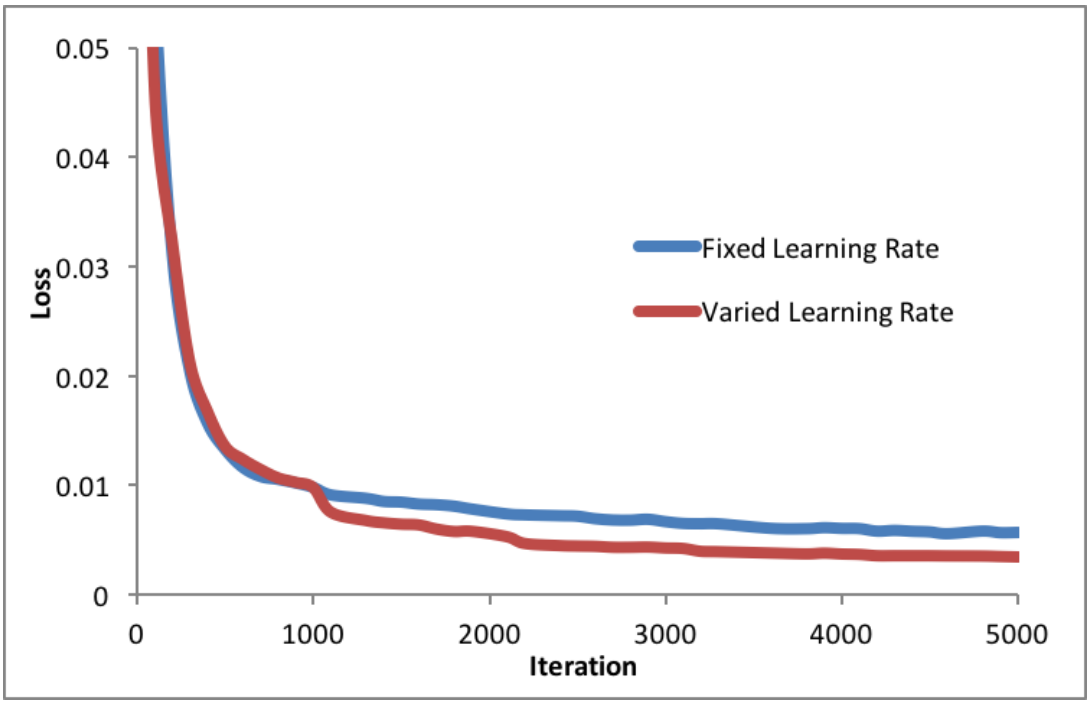


Figure 2: Loss over Time for Varied and Fixed Learning Rates

Question 1 (e)

We experimented with the network structure by commenting out each hidden layer one by one and analyzed the change in overall accuracy after reloading the network. We noticed that after dropping 3 layers, the quality of the image drops significantly. This means that you can get away with 80 hidden units (drop 60 units) before quality drops noticeably.

Question 1 (f)

Finally, we experimented with the network structure by adding more hidden layers to the network. We noted that adding layers does not help significantly increase the accuracy of the network.

RANDOM FOREST FOR IMAGE APPROXIMATION

Summary

For this problem, we used random forest regression to approximate an image of the Mona Lisa by learning a function that takes an image's (x, y) coordinates as inputs and outputs (r, g, b) pixel brightness. We created our training data by randomly selecting 5,000 samples of (x, y) coordinate locations. We then created a function that maps these coordinates to the (r, g, b) pixel intensity values. Finally, we used Scikit-learn's RandomForestRegressor to fit our training data and build a random forest regression model to approximate the Mona Lisa image.

Question 2 (b)

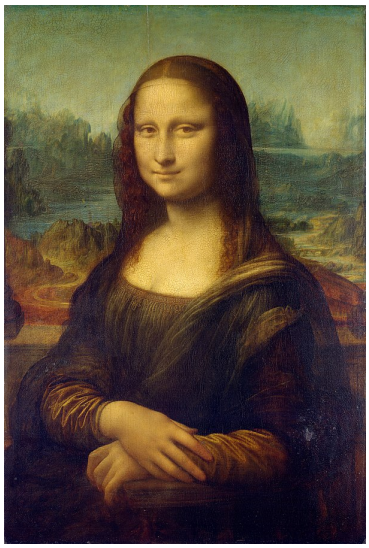


Figure 3: Base Image

To build our training set, we downloaded the small Mona Lisa image from the provided Wikipedia link. From the image, we uniformly chose 5,000 random samples of coordinate locations. We did not perform any input preprocessing such as mean subtraction, standardization, or unit-normalization since decision trees are insensitive to monotone transformations on inputs.

Question 2 (c)

Since the sample value at each given coordinate contains red, green, and blue intensity values, we regressed all three values at once to map the (x, y) coordinates to the (r, g, b) values $(f : \mathbb{R}^2 \rightarrow \mathbb{R}^3)$.

In addition, since the (r, g, b) intensity values lie between 0 and 255, we rescaled them to lie between 0 and 1. We did not perform any additional preprocessing for random regression forest outputs since decision trees are generally robust to outliers.

Question 2 (d)

Using Scikit-learn's *RandomForestRegressor* library, we fitted our training input of coordinate samples and output of pixel intensities to approximate the coordinates of the Mona Lisa image. We began with a single tree with a default depth, where the nodes are expanded until all leaves are pure.

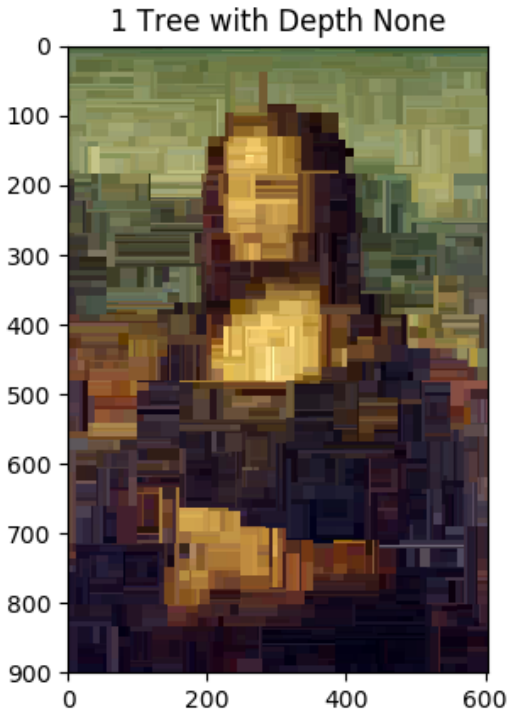


Figure 4: Image from Single Tree Regression at Default Depth

Question 2 (e)

i. We experimented with the parameters in our random tree regression model by building the image for a single tree at depths 1, 2, 3, 5, 10, and 15. We noticed that as we increase the depth of the tree, the quality of the image improves. This makes sense since depth represents the number of subspaces that the tree is divided into, and as the tree expands, we coerce the tree to become more diverse. As a result, we see a more accurate representation of the image.

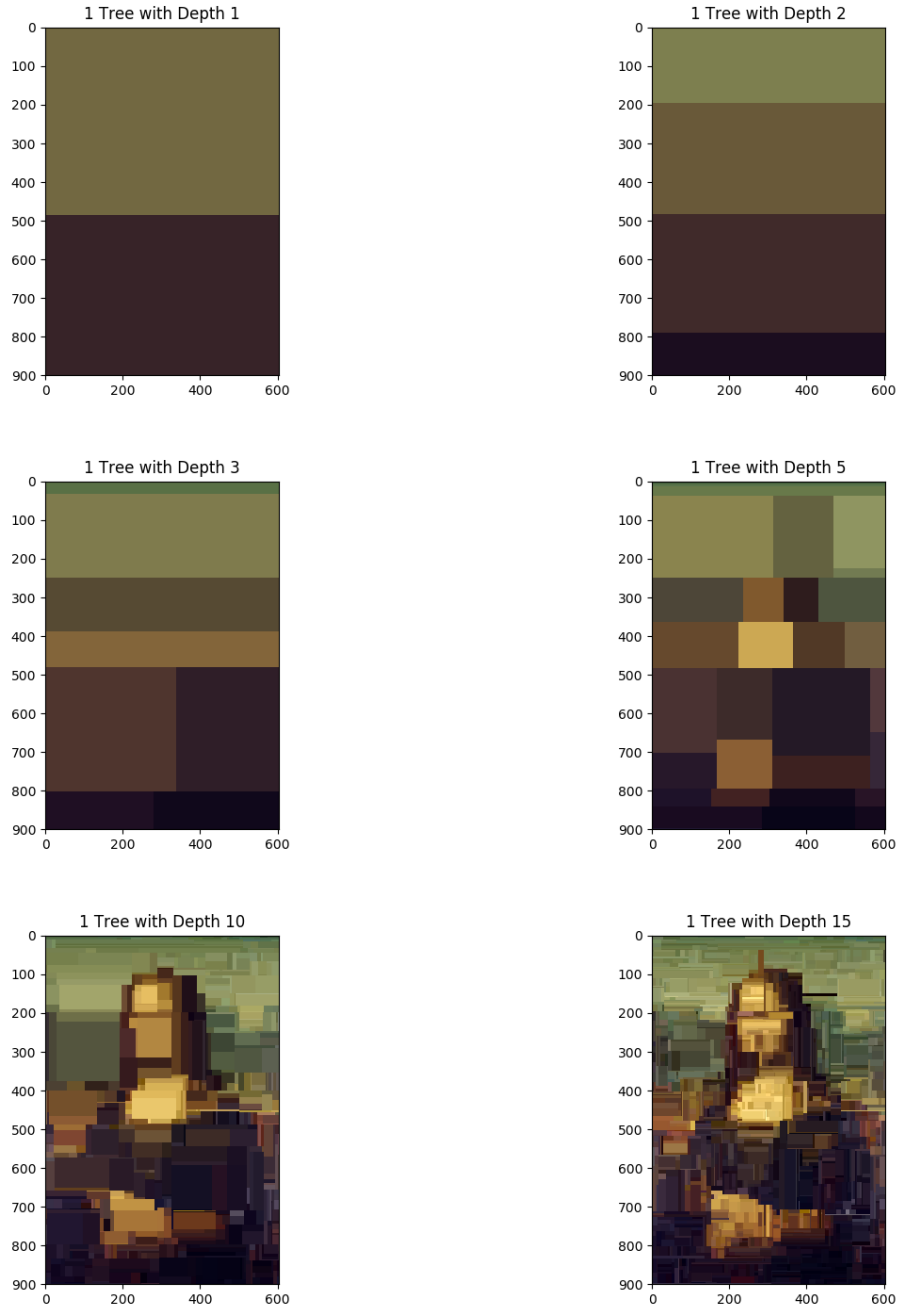


Figure 5: Images from Single Tree Regression at Varying Depths

ii. We repeated our experiments by building a random forest of depth 7 and with the number of trees equal to 1, 3, 5, 10, and 100. As the number of trees increases, we see that the image becomes smoother. Since each tree is grown on a different subset of the training data, and the overall forest averages the trees to improve the predictive accuracy, having multiple trees results in highly correlated predictors and a smoother image.

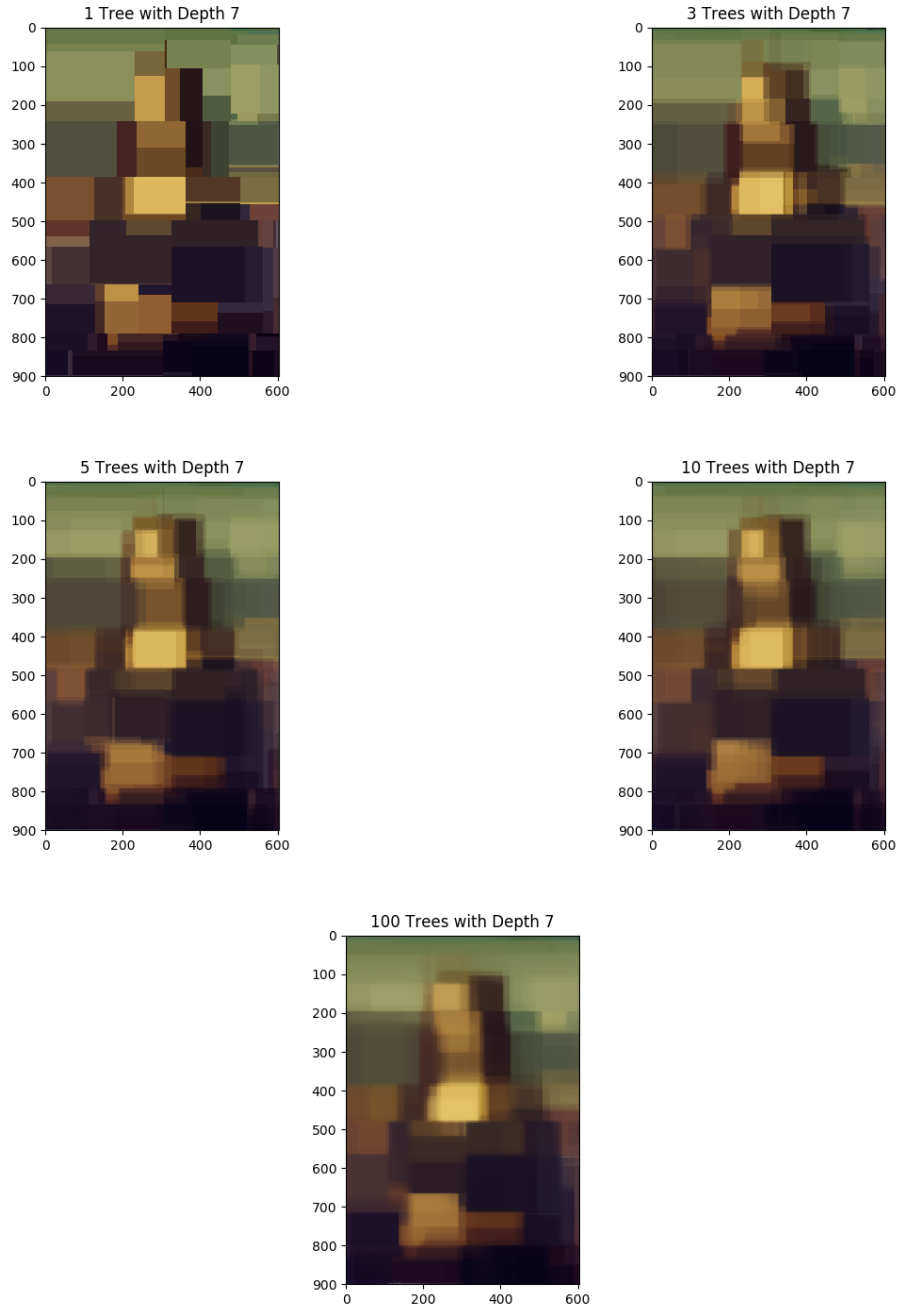


Figure 6: Images from Regression with Varying Number of Trees at Depth 7

iii. As a simple baseline, we repeated the experiment using Scikit-learn's *KNeighborsRegression* library for $k = 1$. We observed that the resulting image consists of coarse patches of colors. Since every pixel in the output is assigned to the nearest pixel from the training set, the decision boundary between each pixel is unsmooth, resulting in the various shapes of color patches in the image.

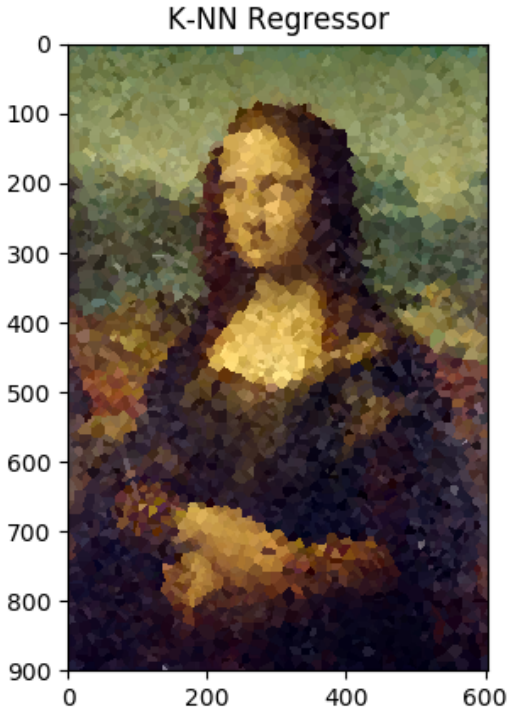


Figure 7: Image Generated Using 1-NN Regression

iv. We experimented with a pruning strategy that set the minimum samples a node can be split on. In particular, we built the image with the min_samples_split parameter set to 5, 10 and 15 for a forest with 10 trees at depth 20. We observed that the accuracy of the image decreased after adding the pruning strategy. In general, pruning strategies are used to prevent overfitting. However, since random forest already applies the bagging method to the training data in order to avoid overfitting, pruning strategies are redundant and will not result in better accuracy.

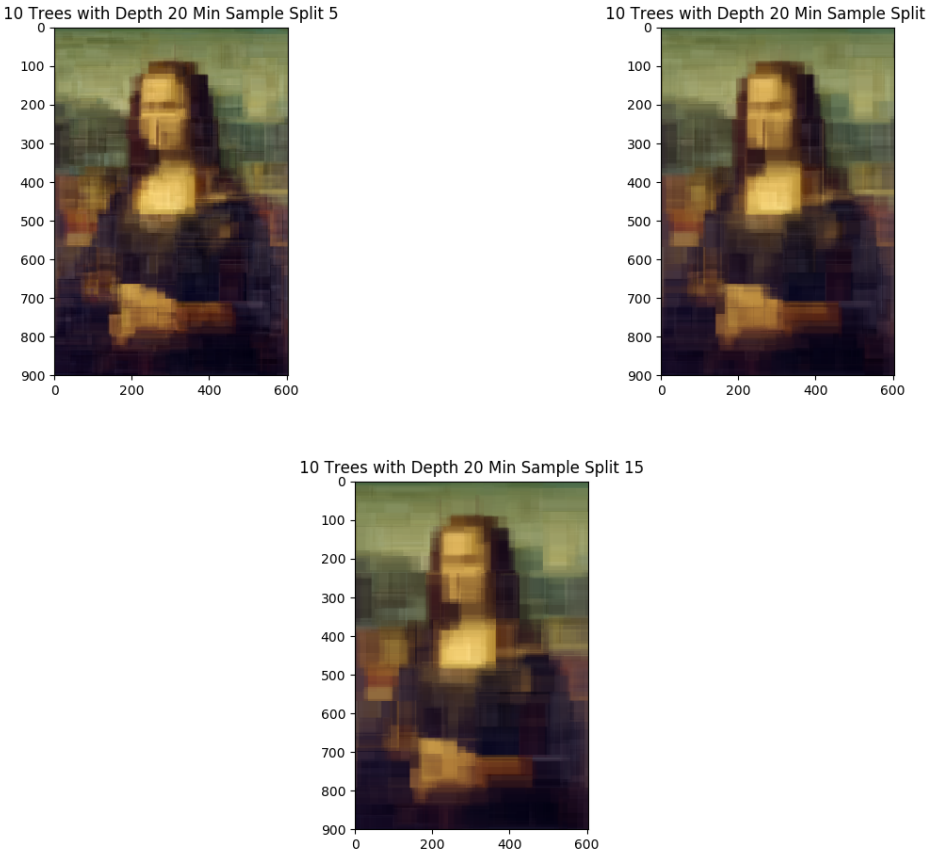


Figure 8: Images from Regression with Varying Number of Trees at Depth 7

Question 2 (f)

i. At each split point, the decision rule is to split 2 nodes based on the input coordinates (x,y). Assuming the root split decision is based on x, the formula for the split point at the root node for a trained decision tree might be: next node = left split if x >= threshold , otherwise next node = right split

ii. The resulting image consists of overlapping rectangular patches of color. For a random forest, the subspaces of its trees are put together to generate the resulting image. We observe rectangular patches of color because the subspaces are rectangles and are assigned based on the colors in our samples.

iii. If the forest contains a single decision tree, 2^{depth} patches of color are in the resulting image.

iv. For a single tree, there may be 2^depth patches of color. For n trees, the upper bound is $n * 2^depth$. There could be any number of patches from 1 to $n * 2^depth$.

SOURCES & EXTERNAL LIBRARIES

Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. *The NumPy Array: A Structure for Efficient Numerical Computation*, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

John D. Hunter. *Matplotlib: A 2D Graphics Environment*, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55

Jones E, Oliphant E, Peterson P, et al. *SciPy: Open Source Scientific Tools for Python*, 2001-, `http://www.scipy.org/`

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, 12, 2825-2830 (2011)

Wes McKinney. *Data Structures for Statistical Computing in Python*, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

Kotzias et. al,. *From Group to Individual Labels using Deep Features*, KDD 2015

Hardle, W. (1991) *Smoothing Techniques with Implementation in S.*, New York: Springer.

Azzalini, A. and Bowman, A. W. (1990). *A look at some data on the Old Faithful geyser*. Applied Statistics 39, 357-365.

"Metric Multidimensional Scaling." *15.2 Metric Multidimensional Scaling*, sfb649.wiwi.hu-berlin.de/fedc_homepage/xplore/tutorials/mvahtmlnode99.html.