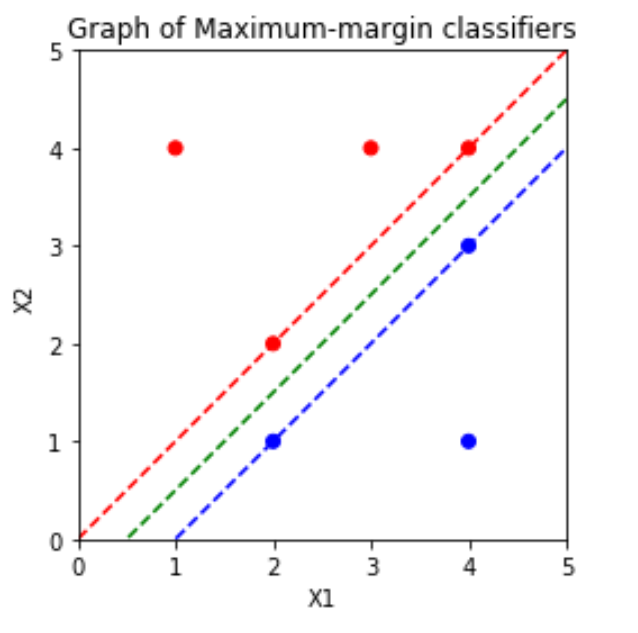
Written Exercise

Exercise 1:

(a)

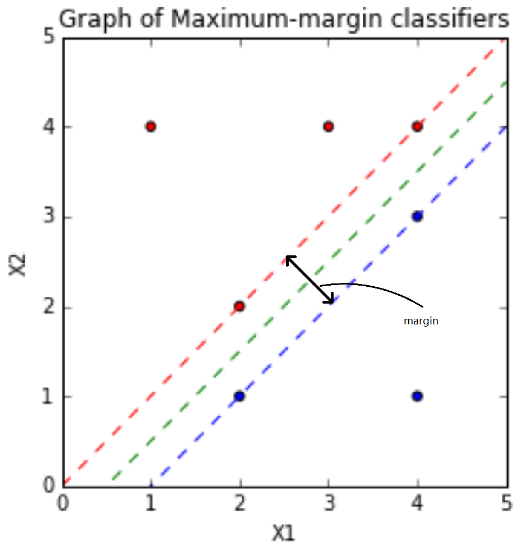


(b)

Classification rule: Classify to Red if “- X1 + X2 + 0.5 > 0”, and classify to Blue otherwise.

β1 = -1; β2 = 1; β0 = 0.5

(c)



The black double arrow line shows the margin. The margin should be .

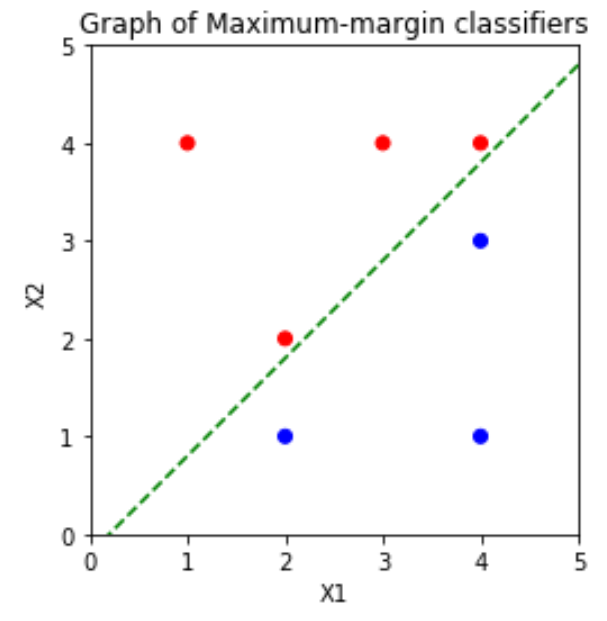
(d)

The support vectors for the maximal margin classigier are following poitns: (2,2), (4,4), (2,1), (4,3)

(e)

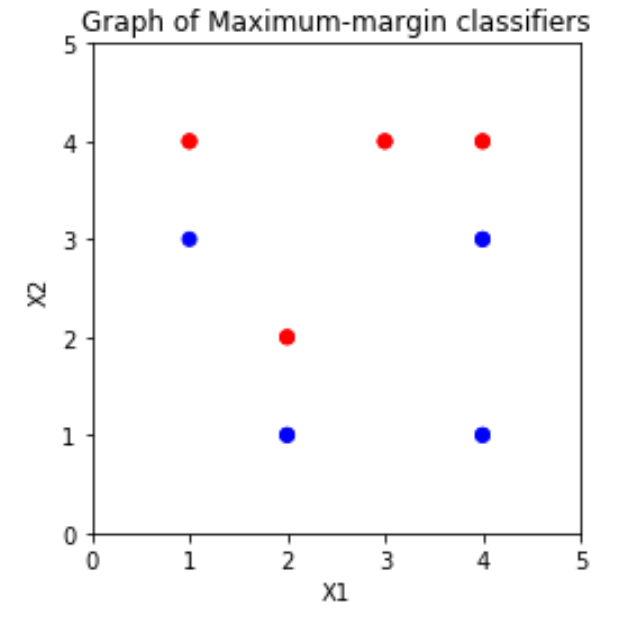
The seventh observation is not the support vector and moving slightly would not make it into support vector. Since the maximal margin hyperplane is fully depended on support vectors, moving it slightly would not affect the maximal margin hyperplane,

(f)



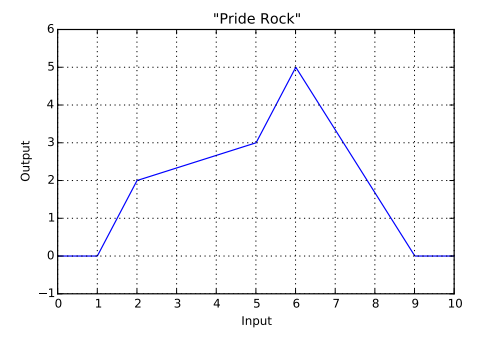
The equation for this hyperplane: X2 = X1-0.2. The margin should be , which is less than .

(g)



After adding a point (1,3), the two classes are no longer separable by a hyperplane.

Exercise 2:



1. First, we construct the whole function by combining shifted and scaled ReLU functions.

For construction, we try to superpose a shifted and scaled ReLU function each time at the next break-angle point, in order to adjust the subsequence slope.

The whole function:

=

1. After knowing this whole function, it should be straightforward for us to define the structure of the neural network now.

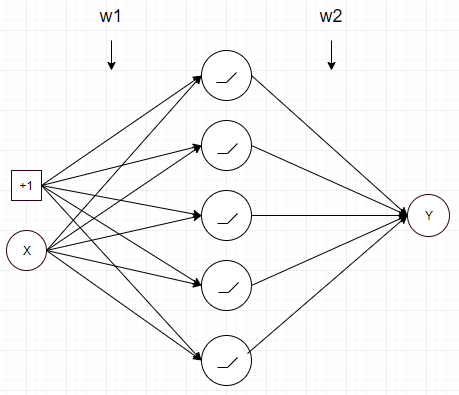
w11 = [, , , , ], for the weights of x in the first layer.

w12 = [-, , , , ], for the weights of bias in the first layer.

[w21, w22, w23, w24, w25] = [1,-1,1,-1,1], for the weights of each neurons in the second layer.

w1 = , which are weights for the first layer.

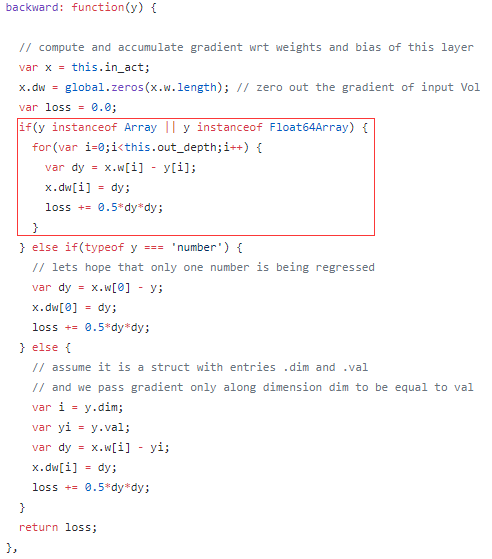
w2 = [1 -1 1 -1 1]T, which are weights for the second layer.

1. Sketch the neural network:

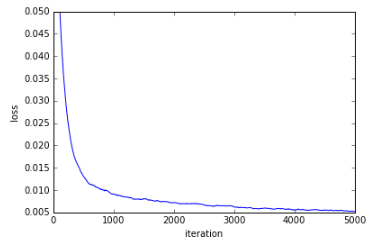
There are totally one hidden layer with 5 neurons in it.

Programming Exercise 1:

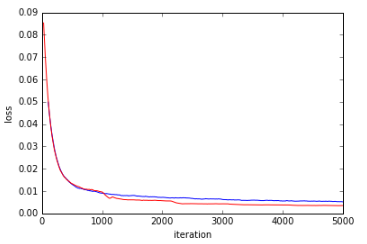
1. There are 1 input layer, 7 hidden layers, and 1 output layer. The input is a 1x1x2 matrix, representing a coordinate (x,y) in the image. Those 7 hidden layers are fully connected, with 20 neurons in each hidden layer, and using ReLU function as the activate function, which introduce non-linearity and accelerate the gradient descent. These 7 hidden layers are used for feature extraction, and there will be 20 features at the end. For the output layer, it implements a regression layer with three outputs representing (R,G,B) values. The whole network maps a 2D coordinates of the image into a 3D dimension values of RGB, which representing the color.
2. Since we can see the last layer is a regression layer, we find the source code implementing loss for regression layer as following:



As we can see from the code, the loss function is the sum of 0.5\*(y[i]’ – y[i])^2, where ‘i’ represent the ith dimension of the ouput, which here is the RGB values. y’ is the output of the neural network and y is the ground truth of the RBG value which could be knew from the training label.

1. 

Here is the plot of the loss over time, after letting it run for 5,000 iterations. The final loss is about 0.005.

1. 

The red line the loss over time when adjust the learning rate each time after 1000 iterations.

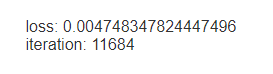
As we can see we can make the network converge to a lower loss function (or faster/earlier) by lowering the learning rate every 1000 iterations.

1. Decreasing hidden layers

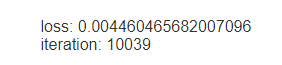
7 hidden layers:

C:\Users\AngZhou\AppData\Local\Temp\WeChat Files\da4a37e9c0107eeba07a15d84746e61.png

6 hidden layers:



5 hidden layers:



4 hidden layers:

C:\Users\AngZhou\AppData\Local\Temp\WeChat Files\0eef3691f067c0fd65a60212acf1514.png

3 hidden layers:

C:\Users\AngZhou\AppData\Local\Temp\WeChat Files\507f4b574a5f17a5c0af1bb3d49ca38.png

2 hidden layers:

C:\Users\AngZhou\AppData\Local\Temp\WeChat Files\b97a6f536b80d33ea0cb922604d2624.png

1 hidden layers:

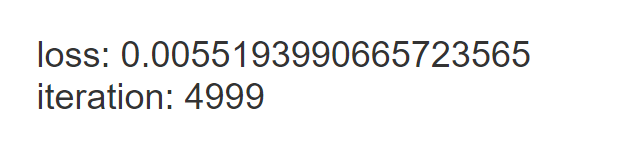
C:\Users\AngZhou\AppData\Local\Temp\WeChat Files\52d556570bda93f574975850289a059.png

As we can see:

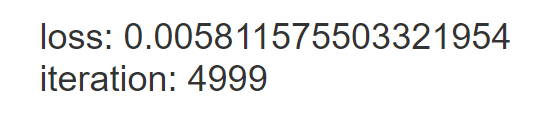
We can drop at most 3 layers before the quality drops noticeably.

1. Adding hidden layers:

8 hidden layers:



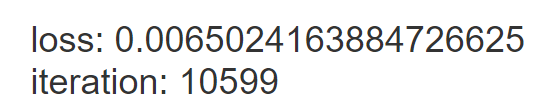
10 hidden layers:



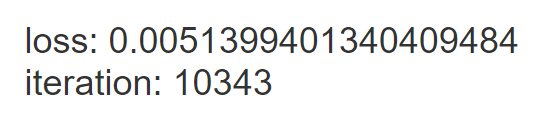
As we can see, increasing the hidden layers doesn’t obviously increase the quality.

Decreasing hidden unit:

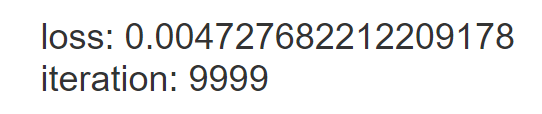
10 hidden units:



15 hidden units:



18 hidden units:



20 hidden units:

C:\Users\AngZhou\AppData\Local\Temp\WeChat Files\da4a37e9c0107eeba07a15d84746e61.png

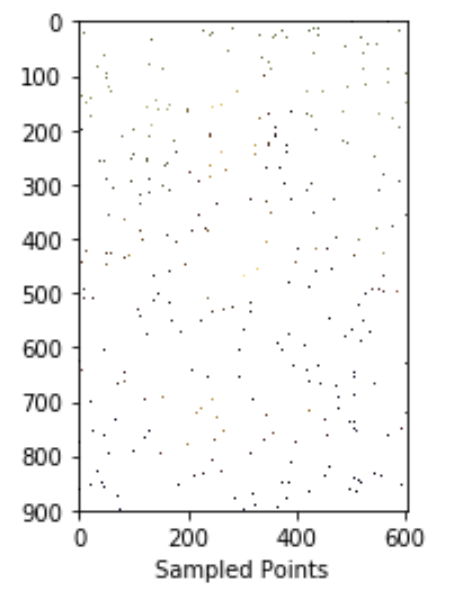
Programming Exercise 2:

(b)

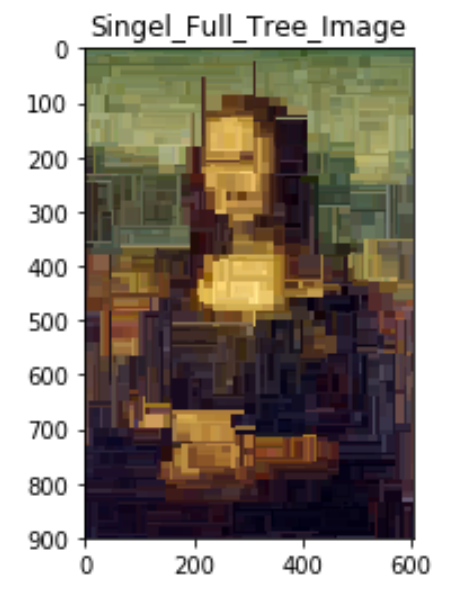
We do not need other preprocessing steps, since the input is a coordinate type data and the output is RGB values. All variables have the same scale and will be treated equally, which is already suitable for implementing random forest.

(c)

We decided to choose: “Regress all three values at once, so your function maps (x,y) coordinates to (r,g,b) values: f :R2→R3”. We think the greyscale method would losing the color and unable to recover the color. We didn’t choose “Learn a different function for each channel, fRed: R2→R, and likewise for fGreen, fBlue”, since we think three different functions (random forest) won’t share the regions of patches together, thereby causing some weird color fusion in some intersection of patches.

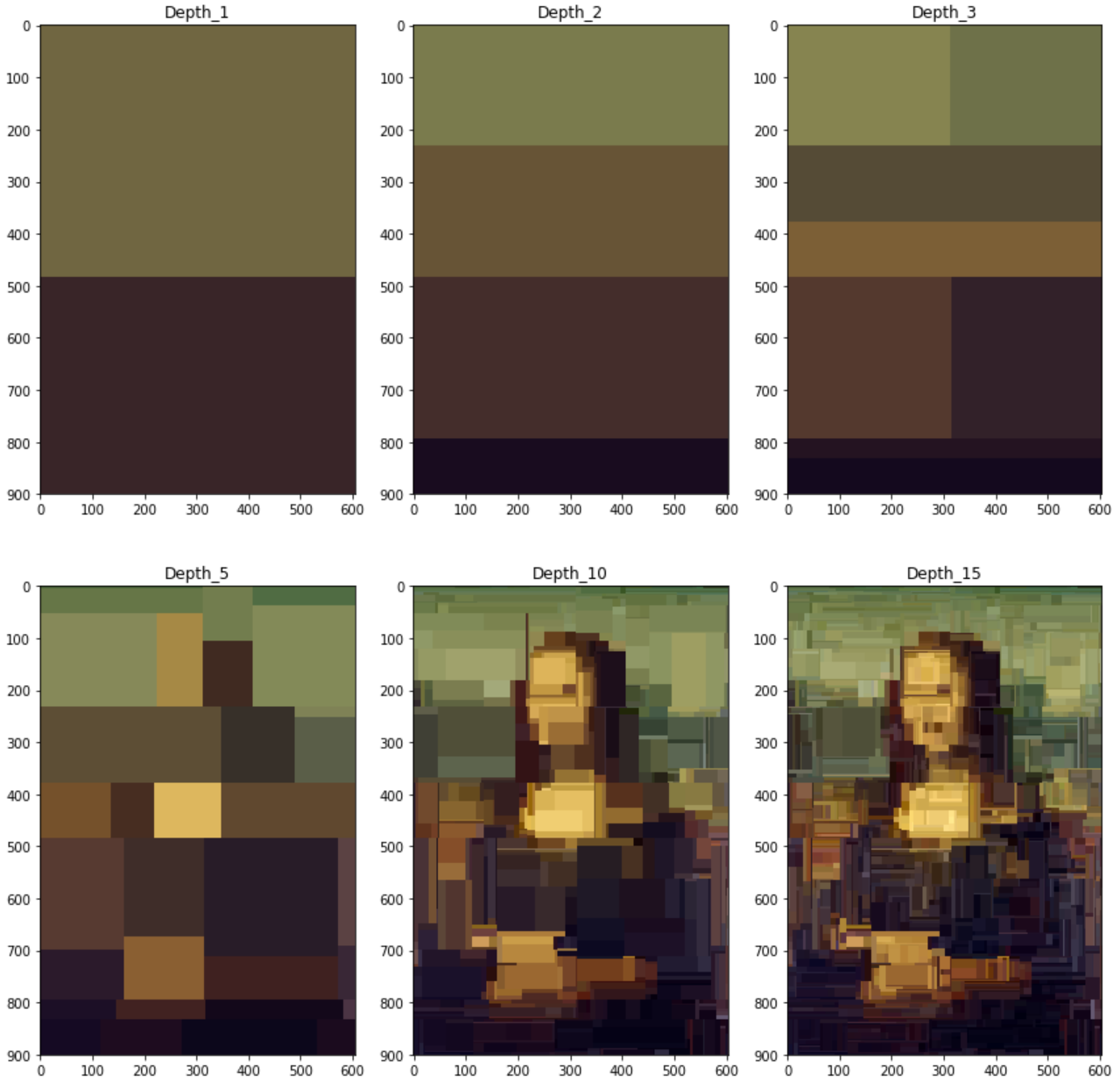


(d)

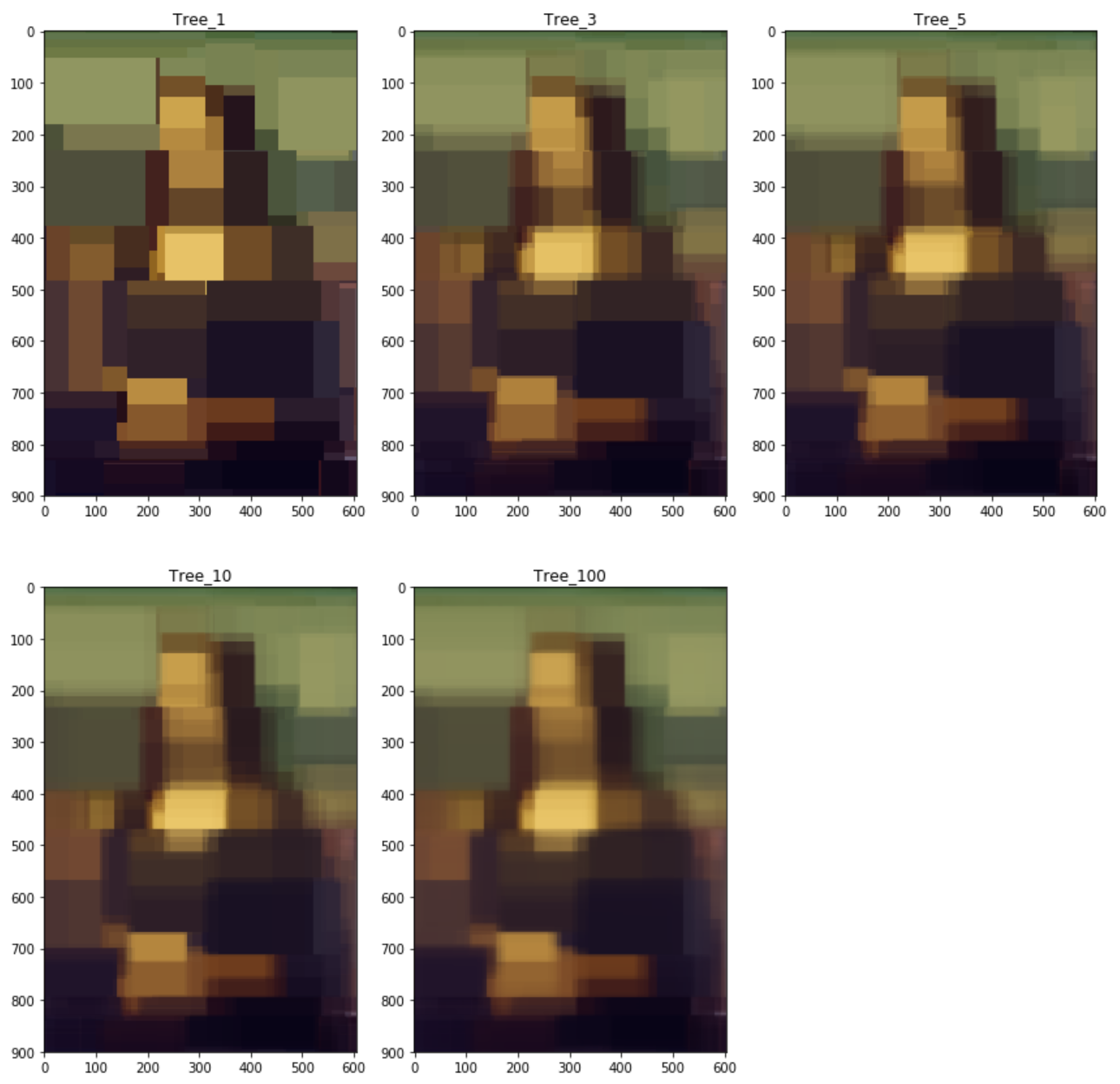


(e)

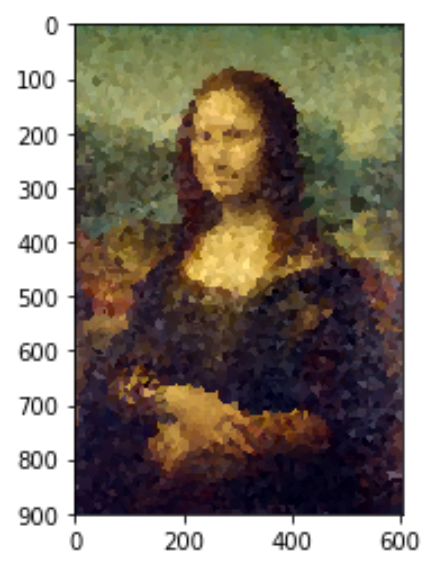
(i) As the depth increase, the patches of color become small and the number of patches increases, thereby the result is more similar the origin image. The reason is that when depth increase, the number of sample in each leaf node is decreased and the number of decision boundaries increased. (The model become more complex.) So, for each small patch, it can better approach the same patch in the origin image.



(ii) As the number of trees increase, the transition from one patch to another patch become much softer. The reason is that different trees have different patches and there would be some intersections of patches among different trees, and it takes the average RGB values, so it basically perform a smoothing process.



(iii) Since it is KNN with k=1, then for each training coordinate, it has its own color patch around it, and there are 5000 training coordinates, so there are 5000 such little color patch with each coordinate near the center of each color patches, and the boundaries are points that have equal distance to nearby training points.



(iv) We select to limit the number of samples in all leaf nodes.

We tried the following numbers as min\_leaf\_samples: 1, 2, 3, 5, 10, 20, 50, 100.

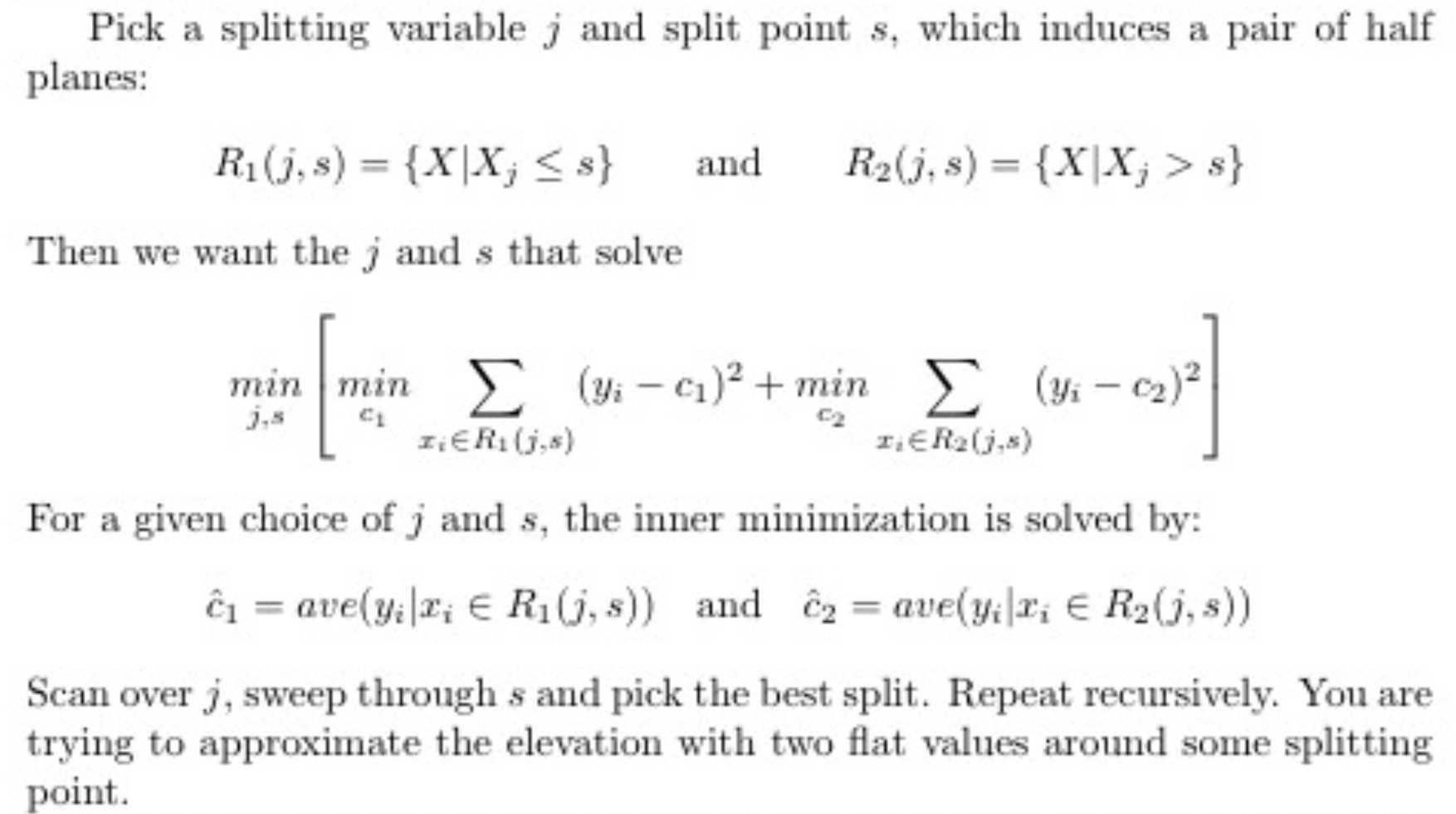
Output:

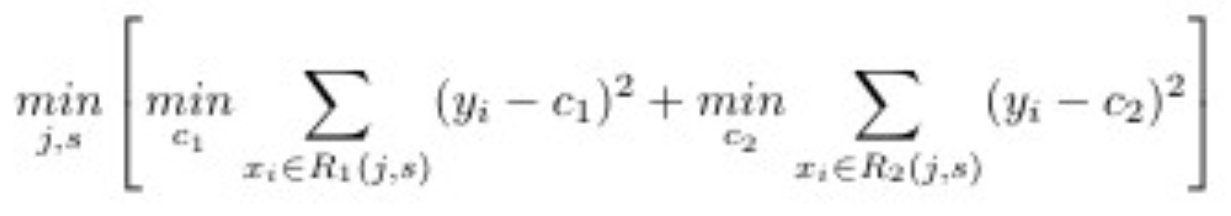


We find that the larger we limit the min\_leaf\_sample, the less color patches it generate.

(f)

(i) According to the lecture note:



The decision rule is:  So at the root, it will select the best variables with the best split which generates the minimum sum of the least square in both sides.

(ii)

Since the random forest generate multiple decision tree. In each decision tree, a single variable is picked for each splitting point, thereby already forming horizontal or vertical decision boundaries. As a result, all the patches have rectangular shapes and arranging vertically or horizontally.

(iii)

Let d be the depth of a decision tree, so a decision tree with only a root has depth 0. The number of patches of color depends on the number of leaves. Since the decision tree is a binary tree, so the number of patches can be as large as 2^d.

(iv)

Let d be the depth of a decision tree, so a decision tree with only a root has depth 0.

Case 1: all n decision tree has exactly the same decision boundaries, so the effect is just like a single tree, which generates 2^d patches.

Case 2: all n decision tree have different decision boundaries, but all these boundaries are in a same direction (horizontal or vertical), which means there is no intersection of these boundary lines. In this case, it will generates 2^d-1 boundaries (a.k.a splitting lines) for each tree, so there are totally n\*(2^d-1) boundaries (a.k.a splitting lines), and thereby generates n\*(2^d-1)+1 patches.

Case 3: We try to generate the intersection of these boundary lines as more as possible.

1. When n is even: let n = 2\*k. Half of the boundary lines are horizontal and half of the boundary lines are vertical. So there can be as many as (k\*(2^d-1)+1)^2 patches.
2. When n is odd: let n = k+k+1. So there can be as many as

(k\*(2^d-1)+1)\*( (k+1)\*(2^d-1)+1) patches.

Reference:

Read Image: <https://stackoverflow.com/questions/35286540/display-an-image-with-python/35286615>

Itertools: <https://docs.python.org/2/library/itertools.html>

Random Forest Regressor: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

KNN Regressor: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>