CS5785
Applied Machine Learning **Siyadong Xiong**
HW4 sx225@cornell.edu
Due Date: 11/28/2016 Teammates: N/A

1 Written Exercises

1.1 Association Rule learning

a The number of total purchases can be obtained by the number of purchases that contain empty set, which is 927125.

b From 8th row, we could get the lower bound. Thus, $231 \leq x \leq 927125$.

c Remove from all transactions that contain Cat Food the transactions that also include any other else, we obtained the upper bound of purchases containing only cat food. $x = 185279 - max(120091, 60159) = 65188$

d The upper bound is given by $\frac{231}{927125}$ and the lower bound is 0 when this itemset never appear in the data set.

e The upper bound is $\frac{29751}{927125}$ while the lower bound is $\frac{15293}{927125}$.

f $CONF(Burgers \rightarrow VCT) = \frac{SUPP(Burger,VCT)}{SUPP(Burger)}$ while $SUPP(Burger, VCT) = \frac{231}{927125}$ and $SUPP(Burger) = \frac{29751}{927125}$. So the CONF is given by $\frac{231}{29751}$.

g As the $SUPP(Dogfood, Catfood) = \frac{60159}{927125}$ and
$0 \leq SUPP(Dogfood, Catfood, CatLitter) \leq \frac{60159}{927125}$, the CONF is given by $0 \leq CONF \leq 1$.

h

$$\begin{aligned} LIFT(DogFood \rightarrow CatFood) &= \frac{T(D \rightarrow C)}{T(D)T(C)} \\ &= \frac{SUPP(D \bigcup C)}{SUPP(D)SUPP(C)} \\ &= \frac{\frac{60159}{927125}}{\frac{80915}{927125}\frac{185279}{927125}}. \end{aligned} \tag{1}$$

i {Dogfood} and {Burgers}.

j {VitaminCTablets}, {ArtisianTapWater}, {Buns}, and {Ketchup}.

k {CatFood} and {CatLitter}.

1.2 Neural Networks as function approximation

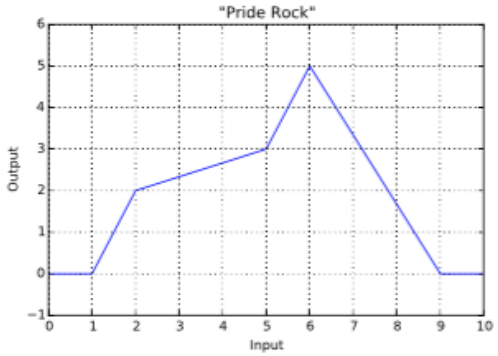

Figure 1: Function to approximate.

To approximate the given function shown in Fig.1, we firstly express it as a weighted sum of ReLU units. Recall that the ReLU activation is given by

$$\sigma(x) := \begin{cases} x, & x \geq 0, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

the given function could be written as

$$\begin{aligned} f(x) &= 2\sigma(x-1) - \frac{5}{3}\sigma(x-2) + \frac{5}{3}\sigma(x-5) - \frac{11}{3}\sigma(x-6) + \frac{5}{3}\sigma(x-9) \\ &= \sigma(2x-2) - \sigma(\frac{5}{3}x - \frac{10}{3}) + \sigma(\frac{5}{3}x - \frac{25}{3}) - \sigma(\frac{11}{3}x - 22) + \sigma(\frac{5}{3}x - 15). \end{aligned} \tag{3}$$

It is straightforward to define the structure of the neural network now. The NN should have 1 hidden layer containing 5 neural units. With input x denoted as $[1, x]^T$, the first weight matrix $\mathbf{W}^1$ is

$$\mathbf{W}^1 = \begin{bmatrix} -2 & -\frac{10}{3} & -\frac{25}{3} & -22 & -15 \\ 2 & \frac{5}{3} & \frac{5}{3} & \frac{11}{3} & \frac{5}{3} \end{bmatrix}, \tag{4}$$

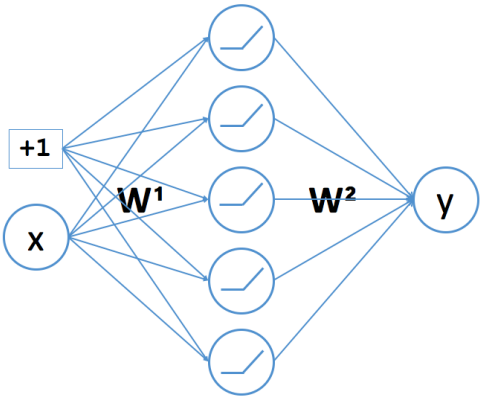

Figure 2: Neural Network structure.

and the second weight matrix $\mathbf{W}^2$ is (omitting bias here)

$$\mathbf{W}^2 = [1, -1, 1, -1, 1]^T. \tag{5}$$

The network is shown in Fig.2.

1.3 Approximating images with neural networks

```
1  layer_defs = [];
2  layer_defs.push({type:'input', out_sx:1, out_sy:1,
       out_depth:2}); // 2 inputs: x, y
3  layer_defs.push({type:'fc', num_neurons:20,
       activation:'relu'});
4  layer_defs.push({type:'fc', num_neurons:20,
       activation:'relu'});
5  layer_defs.push({type:'fc', num_neurons:20,
       activation:'relu'});
6  layer_defs.push({type:'fc', num_neurons:20,
       activation:'relu'});
7  layer_defs.push({type:'fc', num_neurons:20,
       activation:'relu'});
8  layer_defs.push({type:'fc', num_neurons:20,
       activation:'relu'});
9  layer_defs.push({type:'fc', num_neurons:20,
       activation:'relu'});
10 layer_defs.push({type:'regression', num_neurons:3}); // 3
       outputs: r,g,b
```

```
11
12 net = new convnetjs.Net();
13 net.makeLayers(layer_defs);
14
15 trainer = new convnetjs.SGDTrainer(net, {learning_rate:0.01,
      momentum:0.9, batch_size:5, l2_decay:0.0});
```

1.3.1 Neural Network structure

According to network definition, it has 1 input layer, which is basically a $1\times1\times2$ matrix and receives coordinate (x, y) as input feature; 7 full connected hidden layers that are utilized to extract features in the given image. The activation for each unit is ReLU that induces non-linearity and accelerates gradient descent. For output, the network use a regression layer with three output for (r, g, b) values. The network maps 2-dimensional coordinates to 3-dimension values.

1.3.2 Loss function

The Loss demonstrated on the website is given by $smooth\_loss = 0.9 * smooth\_loss + 0.1 * loss$, where loss refers to square errors according to the source code of ConvNetJS.

```
1 for(var i=0;i<this.out_depth;i++) {
2     var dy = x.w[i] - y[i];
3     x.dw[i] = dy;
4    loss += 0.5*dy*dy;
5 }
```

As Michael mentioned on Piazza, this smoothing filtered out high-freq noises (seems a IIR LPF) and is used to reduce the noisy property of the model's original loss.

1.3.3 Loss over time

By slightly modifying the JavaScript code, we could sample the loss values per 50 iterations until total iterations reached 5000. As Fig.3 shown, the gray line indicates the original square error loss output of the network which apparently has ripples; the blue line indicates the loss over time under the learning rate fixed at 0.01 from which we knew that the loss reduced to $6 * 10^{-2}$ level after 5000 iterations; the red line shows that if we lower the

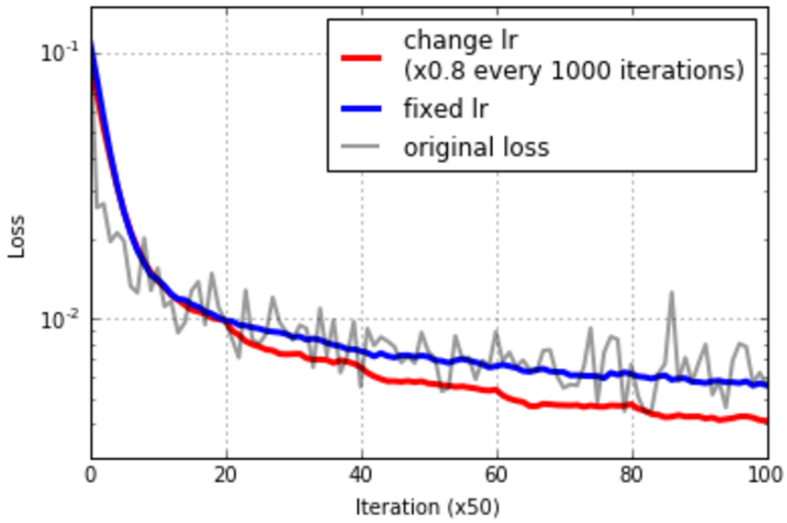

Figure 3: Loss over time and change learning rate

learning rate per 1000 iterations, for example here we reduce 20% each time, we could reach a better convergence at the training phase.

1.3.4 Lesion study

Here we conducted an experiment that analyzed the influence of the number of hidden layers. Here we used variable learning rate like Sec.1.3.2, and plotted the loss convergence lines under different number of hidden layers. Fig.4 shows the experiment results and network output for 1 hidden layer network, 3 hidden layers network, and 9 hidden layers network. It is straightforward to tell from the figure that when hidden layers are less than 5, the performance drops significantly.

2 Programming Exercises

2.1 Random Forests for Image Approximation

2.1.1 Train set and test set

We randomly chose 5000 coordinates (x, y) as input features; the corresponding (r, g, b) values, which are normalized by 255, as input values.

```
1 idx = np.arange(leng)
2 x = np.array([i / wid for i in idx])
```

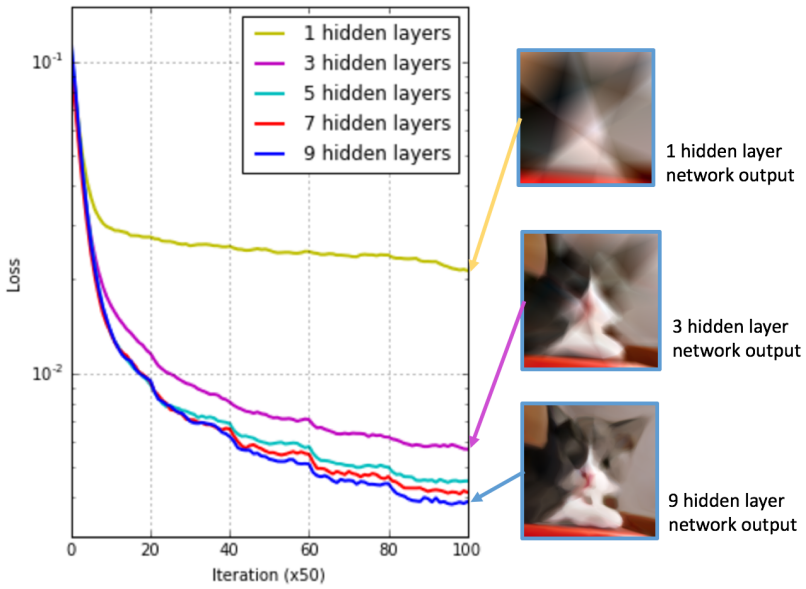

Figure 4: Loss over time comparison among different numbers of hidden layers

```
y = np.array([i % wid for i in idx])
xy = np.column_stack((x, y))

# sample shape: 5000 * 5 (x, y, r, g, b)
sample_size = 5000
sample_idx = np.random.choice(np.arange(leng),
    size=sample_size)
sample = ground_truth[sample_idx, :].astype(np.float) / 255
train = np.column_stack((xy[sample_idx, :], sample))

print train.shape

# assertion
_ = np.random.randint(0, sample_size)
np.allclose(pic[int(train[_,0]), int(train[_,1]), :] / 255.0,
    train[_, 2:])

# build data set
x_train = train[:, :2]
y_train = train[:, 2:]
```

```
x_test = xy
y_test = ground_truth / 255.0
```

2.1.2 Random Forest Regression

As input features are actually coordinates and we would not calculate distance between data points pairs, it is not necessary to perform normalization or standardization preprocesssings. For output, as we wanted to keep color information, we did not convert the image to gray scale; rather, we considered training separate random forest regression models for each channel versus training a multi-output model that maps $\mathbb{R}^2 \rightarrow \mathbb{R}^3$. By the following code snippet, we compared the difference between train separate models for each channel and train one model with multi-outputs.

```
from sklearn.ensemble import RandomForestRegressor as RF

rf_r = RF(); rf_r.fit(x_train, y_train[:,0])
rf_g = RF(); rf_g.fit(x_train, y_train[:, 1])
rf_b = RF(); rf_b.fit(x_train y_train[:, 2])
rf = RF(); rf.fit(x_train, y_train)

plt.subplot(1,2,1)
pred_pic = np.column_stack((rf_r.predict(x_test),
   rf_g.predict(x_test), rf_b.predict(x_test))).reshape(hei,
   wid, chl)
plt.imshow(pred_pic)
plt.title('Separate RF regressors')
plt.subplot(1,2,2)
pred_pic = rf.predict(x_test).reshape(hei, wid, chl)
plt.imshow(pred_pic)
plt.title('One multi-output RF regressor')
```

As indicated in Fig.5, single model is able to generate more smooth regression results since it utilize the information across different channels. In the experiment described next section, we only consider this single model.

2.1.3 Experimentation

a). single tree and variant depths With only one tree in the forest, Random Forest model degenerated towards a Decision Tree model. Using the following code snippet, we obtained different regression output images and

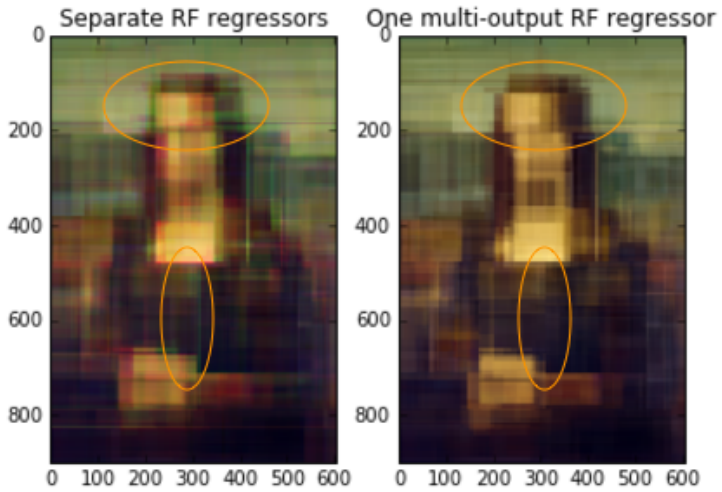

Figure 5: Comparison between separate models and one multi-ouputs model.

regression scores as shown in Fig.6. When depth is shallow, it is theoretically unable to generate as many subspace as sample size; hence significant is the training error. When depth is large enough, the tree can generate more finer granular subspaces at leaf nodes; hence easy to reduce training error. As depth keeps increasing, however, there might be risk of overfitting since there is only one tree.

```
for i, d in enumerate(depth):
    reg = RF(n_estimators=1, max_depth=d)
    reg.fit(x_train, y_train)

    _pred_pic = reg.predict(x_test).reshape(hei, wid, chl)
    plt.subplot(ROW_PLT, COL_PLT, i + 1)
    plt.imshow(_pred_pic)
    plt.axis('off')
    r2 = reg.score(x_test, y_test)
    r2_scores.append(r2)
    plt.title('Max depth = %d\nRegression score ($R^2$): %f' %
        (d, r2))
```

b). variant trees and fixed depth Similarly we obtained the results with variant number of trees in forest with a fixed tree depth. Fig.7 demonstrates the output images and regression scores. As the output of forest is the average of trees' outputs, we observed that when increasing the number of trees, the

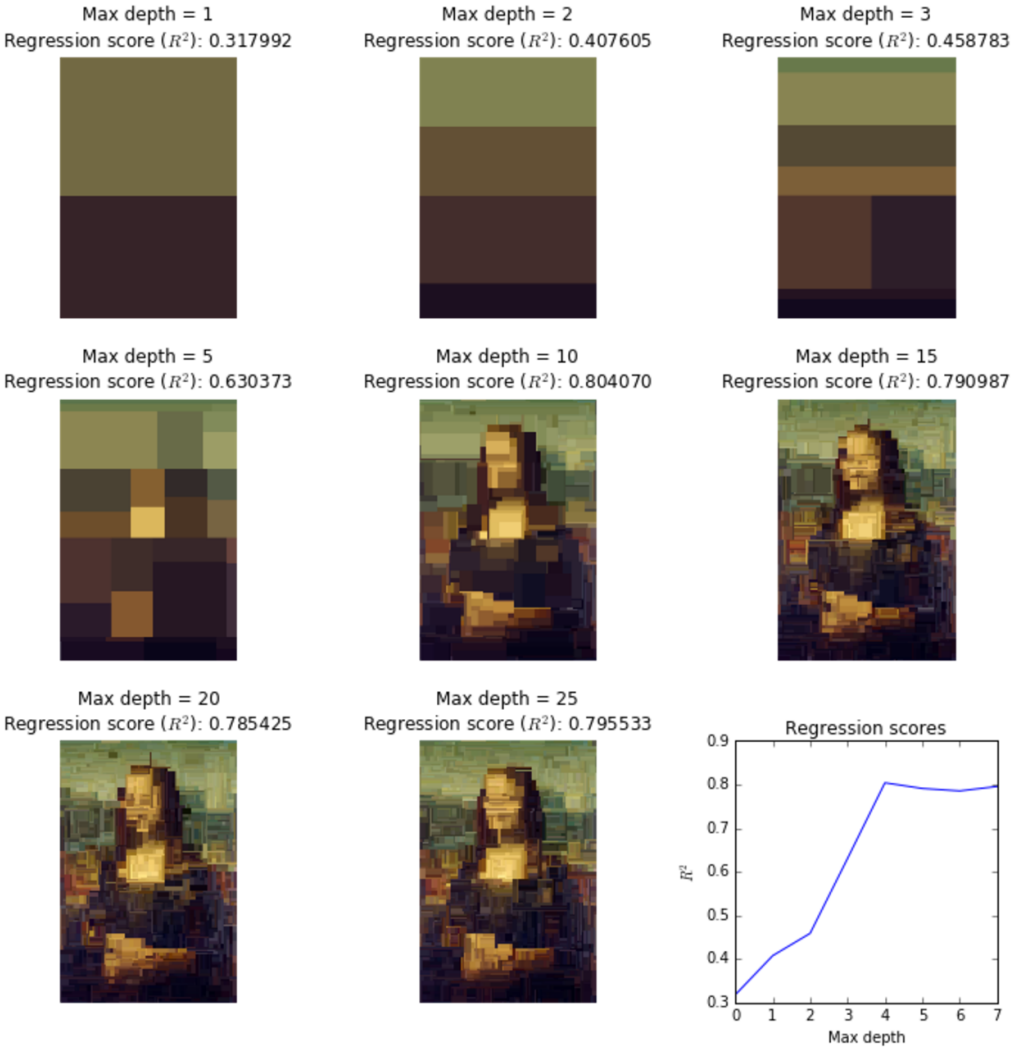

Figure 6: Comparison between different tree depths.

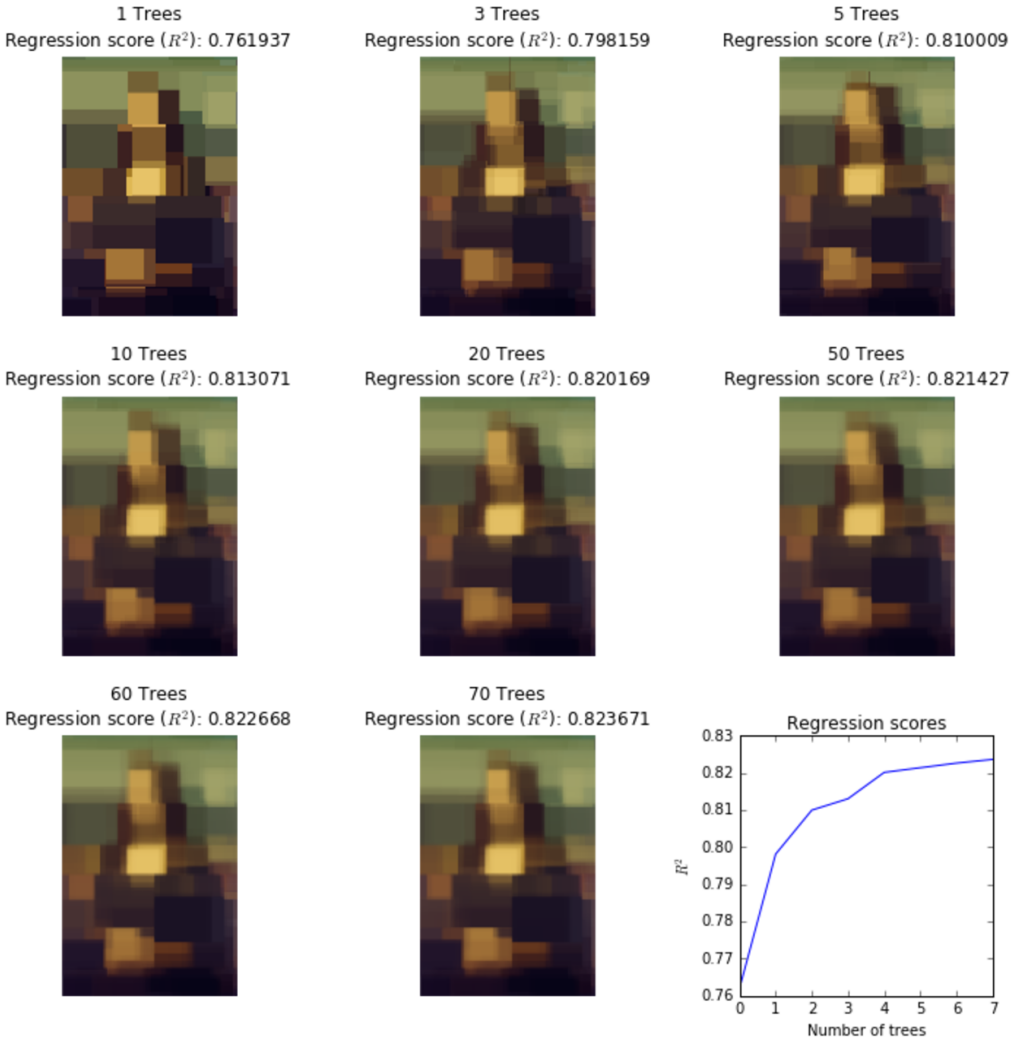

Figure 7: Comparison between different numbers of trees.

output images become more and more blurred. Contrast to variant depths situation, the subspace granularity is constant when the tree depth is fixed.

Figure 8: 1-NN regression image baseline.

c). 1-NN regressor We use the 1-NN regressor as the baseline.

```
from sklearn.neighbors import KNeighborsRegressor as kNN
knn = kNN(n_neighbors=1)
knn.fit(x_train, y_train)
_pred_pic = knn.predict(x_test).reshape(hei, wid, chl)
plt.imshow(_pred_pic)
```

As shown in Fig.8, 1-NN model has the finest granular subspace, pixel level in specific. Images are intrinsically locally high correlative, thus, for image regression this model obtained good performance with $R^2 = 0.8531$.

d). pruning strategies Setting trees to 50 and max depth to 20, we tried two pruning methods: 1). limit the lower bound of the samples on a node that can split; 2). limit the lower bound of the samples on a leaf node. As shown in Fig.9 and 10, the pruning methods decrease the performance which indicate that Random Forest is a robust model that is able resist overfitting.

2.1.4 Analysis

a Decision rule is to split into two nodes or a (r, g, b) value vector based on the input coordinate (x, y). Assuming that the root split based on only

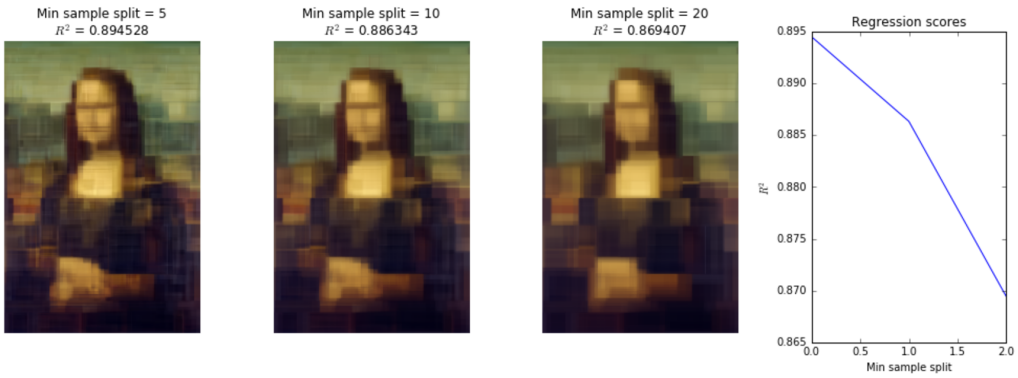

Figure 9: Impact of limit sample split.

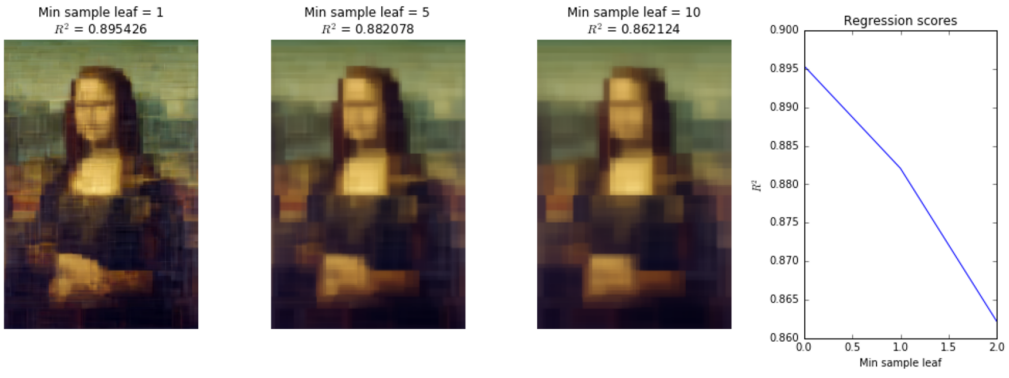

Figure 10: Impact of limit sample leaf.

x, we could write the rule as

$$next\ node = \begin{cases} node1 & x \geq \alpha, \\ node2 & \text{otherwise.} \end{cases} \tag{6}$$

b The subspaces of trees are put together to generate the output images. As the input is 2-dimensional coordinate, the subspaces are rectangles.

c The upper bound will be 2^d, where d is the depth.

d Assuming that each tree has l_i unique leaf nodes, the number of colors is determined by the sum of trees outputs. The upper bound of this sum is given by $\prod_{i=1}^{n} l_i$ and the loose lower bound should be 1.

3 Attachments Details

`hw4.ipynb` in `hw4_sidxiong.zip`.

References

[1] *http://cs231n.github.io/*