Applied Machine Learning

Random Forest Regressor Analysis

Linna Wang

In this question, you will use random forest regression to approximate an image by learning a function, f : R2->R(3) that takes image (x, y) coordinates as input and outputs pixel brightness. This way, the function learns to approximate areas of the image that it has not seen before.

2b. Preprocessing the input. To build your "training set," uniformly sample 5,000 random (x, y) coordinate locations.

(There's no other preprocessing for input needed, decision tree with categorize the input without doing preocessing)

In [2]:
```
%matplotlib inline
import numpy as np
from scipy import misc
import random
import itertools
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor

im = misc.imread('./monaLisa.jpg')
print ("a single coordinate: (red feature)", im[1, 1, 0])
height, width, dim = im.shape
print (height, width)

# Preprocessing the input: generate 5000 random coordinates (y, x) out of 900*604 coordinates i
sample_coordinates = [[int((height-1)*random.random()), int((width-1)*random.random())] for i i
print (sample_coordinates[0], sample_coordinates[1])
print ("length of coordinates sample", len(sample_coordinates))
```

```
a single coordinate: (red feature) 104
900 604
[149, 250] [347, 180]
length of coordinates sample 5000
```

2c. Preprocessing the output. Sample pixel values at each of the given coordinate locations. Each pixel contains red, green, and blue intensity values, so decide how you want to handle this.

2d. Rescale the pixel intensities to lie between 0.0 and 1.0. (The default for pixel values may be between 0 and 255, but your image library may have different defaults.)

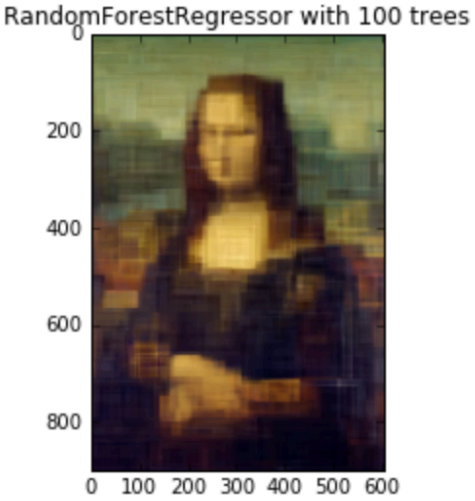
(there's no other preprocessing method for output either in the random forest regressor)

In [7]:
```
# Preprocessing the output:
# Regress all three values at once, so your function maps (x, y) coordinates to (r, g ,b) values
rgb_values = np.array([[im[y,x,0]/255.0, im[y,x,1]/255.0, im[y,x,2]/255.0]  for [y,x] in sample_

# map the coordinates to three dimensional colors
estimator = RandomForestRegressor(random_state=0, n_estimators=100)
# fitting rgb values
estimator = estimator.fit(sample_coordinates, rgb_values)
predict_coordinates = [[y,x] for y in range(height) for x in range(width)]
print (len(predict_coordinates), predict_coordinates[0], predict_coordinates[1])
predicted_rgb = estimator.predict(predict_coordinates)
predicted_im = np.array(predicted_rgb)
print ("shape of predicted image: ", predicted_im.shape)
print (predicted_im)
predicted_im = np.reshape(predicted_im, (height, width, 3))
plt.title("RandomForestRegressor with 100 trees")
plt.imshow(predicted_im)
plt.show()
```

```
543600 [0, 0] [0, 1]
shape of predicted image:  (543600, 3)
[[ 0.28560784  0.41062745  0.30815686]
 [ 0.28560784  0.41062745  0.30815686]
 [ 0.28560784  0.41062745  0.30815686]
 ...,
 [ 0.10603922  0.06984314  0.17168627]
 [ 0.10603922  0.06984314  0.17168627]
 [ 0.10603922  0.06984314  0.17168627]]
```

2f. To build the final image, for each pixel of the output, feed the pixel coordinate through the random forest and color the resulting pixel with the output prediction. You can then use

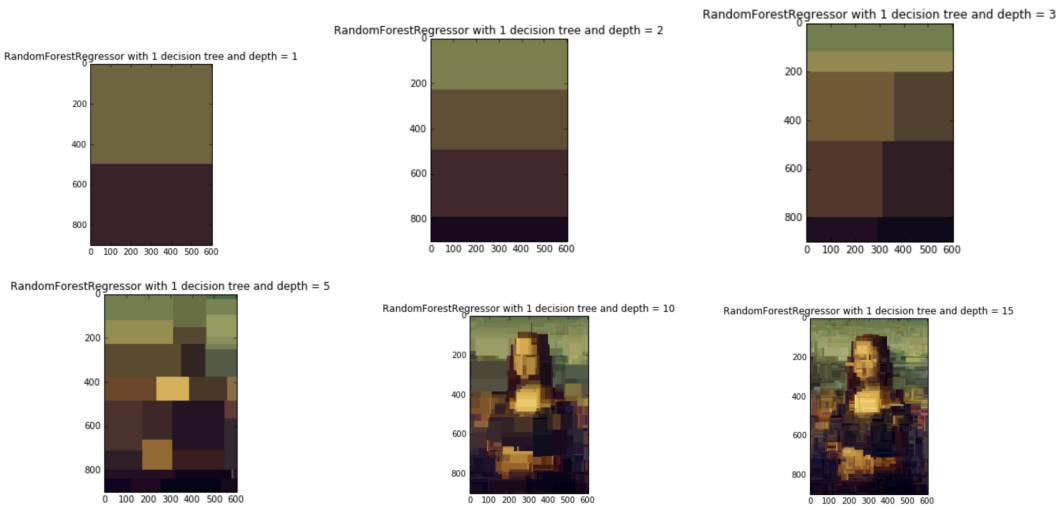
imshow to view the result. (If you are using grayscale, try imshow(Y, cmap='gray') to avoid fake-coloring). You may use any implementation of random forests, but you should understand the implementation and you must cite your sources.



2g. Experimentation.

(a) Repeat the experiment for a random forest containing a single decision tree, but with depths 1, 2, 3, 5, 10, and 15. How does depth impact the result? Describe in detail why.

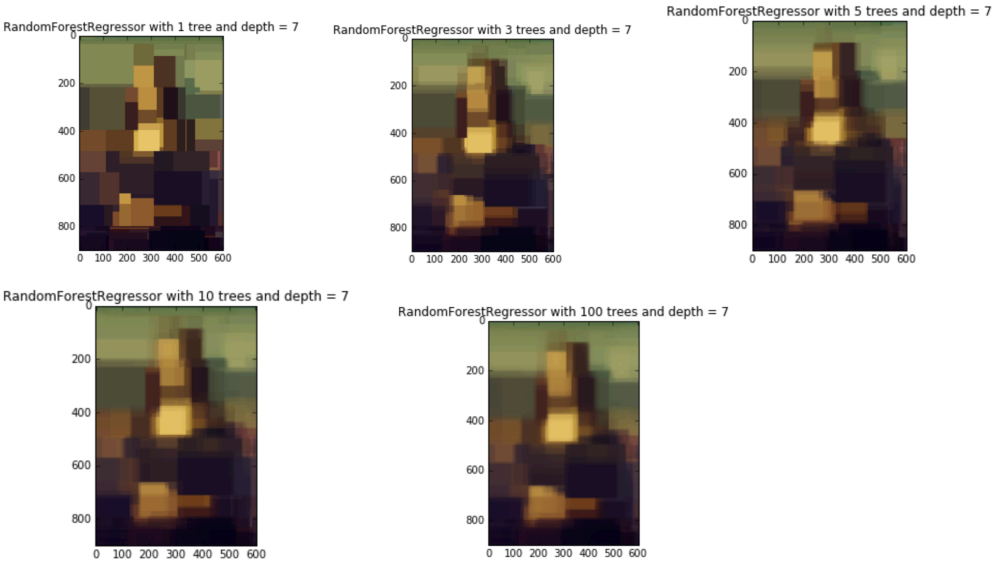
```
In [21]: # Repeat the experiment for a random forest containing a single decision tree, but with depths
         # 1, 2, 3, 5, 10, and 15. How does depth impact the result?

         # map the coordinates to three dimensional colors
         # changing the depth to 1, 2, 3, 5, 10, and 15, and compare
         estimator = RandomForestRegressor(max_depth=15, n_estimators=1)
         # fitting rgb values
         estimator = estimator.fit(sample_coordinates, rgb_values)
         predict_coordinates = [[y,x] for y in range(height) for x in range(width)]
         print (len(predict_coordinates), predict_coordinates[0], predict_coordinates[1])
         predicted_rgb = estimator.predict(predict_coordinates)
         predicted_im = np.array(predicted_rgb)
         print ("shape of predicted image: ", predicted_im.shape)
         print (predicted_im)
         predicted_im = np.reshape(predicted_im, (height, width, 3))
         plt.title("RandomForestRegressor with 1 decision tree and depth = 15")
         plt.imshow(predicted_im)
         plt.show()
```



From the above images, we find that the deeper the depth, the more patches we can get. At first we could only see vertical difference, but as depth becomes deeper, horizontal difference appear more.
A rough approximation is that: the number of color we could get is approximately: 2^depth

(b) Repeat the experiment for a random forest of depth 7, but with number of trees equal to 1, 3, 5, 10, and 100. How does the number of trees impact the result? Describe in detail why.

In [30]:
```
# Repeat the experiment for a random forest of depth 7, but with number of trees equal
# to 1, 3, 5, 10, and 100. How does the number of trees impact the result?

# map the coordinates to three dimensional colors
# changing the tree number to 1, 3, 5, 10, and 100, and compare
estimator = RandomForestRegressor(max_depth=7, n_estimators=100)
# fitting rgb values
estimator = estimator.fit(sample_coordinates, rgb_values)
predict_coordinates = [[y,x] for y in range(height) for x in range(width)]
print (len(predict_coordinates), predict_coordinates[0], predict_coordinates[1])
predicted_rgb = estimator.predict(predict_coordinates)
predicted_im = np.array(predicted_rgb)
print ("shape of predicted image: ", predicted_im.shape)
print (predicted_im)
predicted_im = np.reshape(predicted_im, (height, width, 3))
plt.title("RandomForestRegressor with 100 trees and depth = 7")
plt.imshow(predicted_im)
plt.show()
```

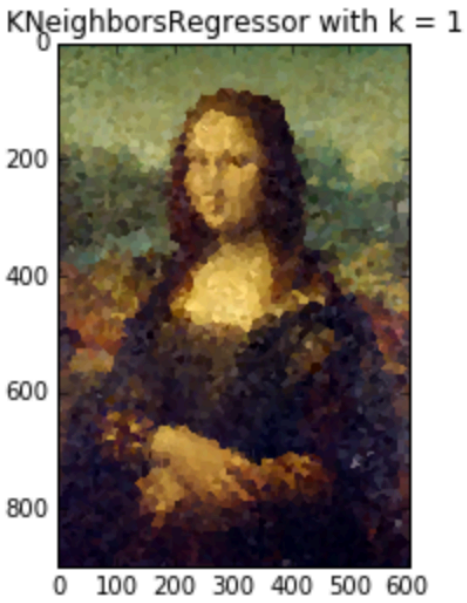


As we can see from the above images, there's no significant improvement of image clearance in changing number of decision trees. From 1 – 3 we could see a decent improvement in sharpness of separation, but from 10 there's no obvious change between images.
The number of decision trees plays a role of smoothing the border of different color patches. As the number of decision trees grow, we get more mean values from different trees, and the borderline seems vague.

(c) As a simple baseline, repeat the experiment using a k-NN regressor, for k = 1. This means that every pixel in the output will equal the nearest pixel from the "training set." Compare and contrast the outlook: why does this look the way it does?

```
In [8]: # k-NN regressor, where k = 1
from sklearn.neighbors import KNeighborsRegressor

rgb_estimator = KNeighborsRegressor(n_neighbors=1)
rgb_estimator.fit(sample_coordinates, rgb_values)
predict_coordinates = [[y,x] for y in range(height) for x in range(width)]
print (len(predict_coordinates))
predicted_rgb = rgb_estimator.predict(predict_coordinates)
print ("shape of predicted image: ", predicted_rgb.shape)
print (predicted_rgb)
predicted_rgb = np.reshape(predicted_rgb, (height, width, 3))
plt.title("KNeighborsRegressor with k = 1")
plt.imshow(predicted_rgb)
plt.show()
```

```
543600
shape of predicted image:  (543600, 3)
[[ 0.30196078  0.41960784  0.30980392]
 [ 0.30196078  0.41960784  0.30980392]
 [ 0.30196078  0.41960784  0.30980392]
 ...,
 [ 0.08627451  0.0745098   0.18039216]
 [ 0.08627451  0.0745098   0.18039216]
 [ 0.08627451  0.0745098   0.18039216]]
```

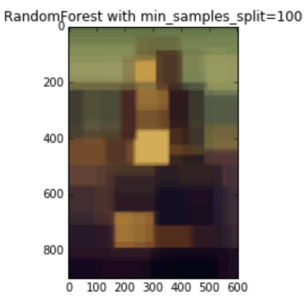


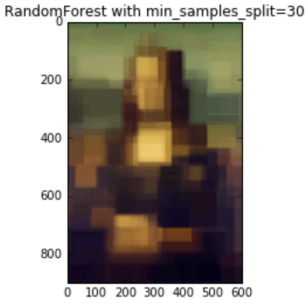
From the above image we could see that the image is not structured as patches of color, but small pieces of polygonal brick. Since the k nearest neighbor assigns the nearest value to each coordinate, so some neighbors might be assigned to the same RGB values

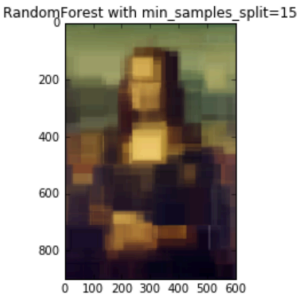
(d) Experiment with different pruning strategies of your choice.

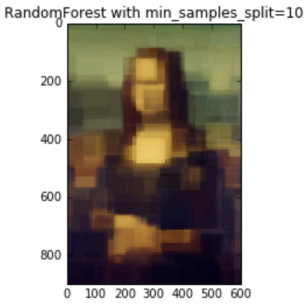
Pruning strategy: Repeat the experiment for a random forest of depth 7, but with min_samples_split equals to 100, 30, 15, 10, 5, and 2. The larger the min split required, the more efficiency we could get since we could avoid getting too small subtrees of our decision trees.

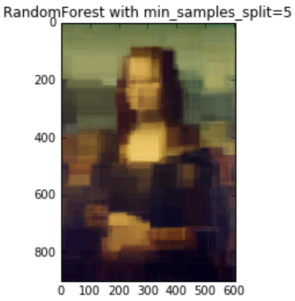
In [37]:
```
# Repeat the experiment for a random forest containing 10 decision tree with depths
# 10 and different min_samples_split values

# map the coordinates to three dimensional colors
# changing the depth to 1, 2, 3, 5, 10, and 15, and compare
estimator = RandomForestRegressor(max_depth=10, n_estimators=10, min_samples_split=2)
# fitting rgb values
estimator = estimator.fit(sample_coordinates, rgb_values)
predict_coordinates = [[y,x] for y in range(height) for x in range(width)]
print (len(predict_coordinates), predict_coordinates[0], predict_coordinates[1])
predicted_rgb = estimator.predict(predict_coordinates)
predicted_im = np.array(predicted_rgb)
print ("shape of predicted image: ", predicted_im.shape)
print (predicted_im)
predicted_im = np.reshape(predicted_im, (height, width, 3))
plt.title("RandomForest with min_samples_split=2")
plt.imshow(predicted_im)
plt.show()
```

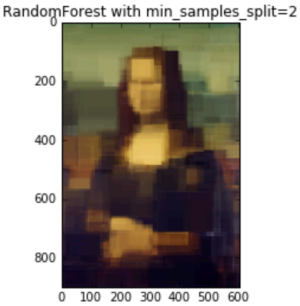












From the above images we could tell that a min split of 100 will make the color patches look bigger

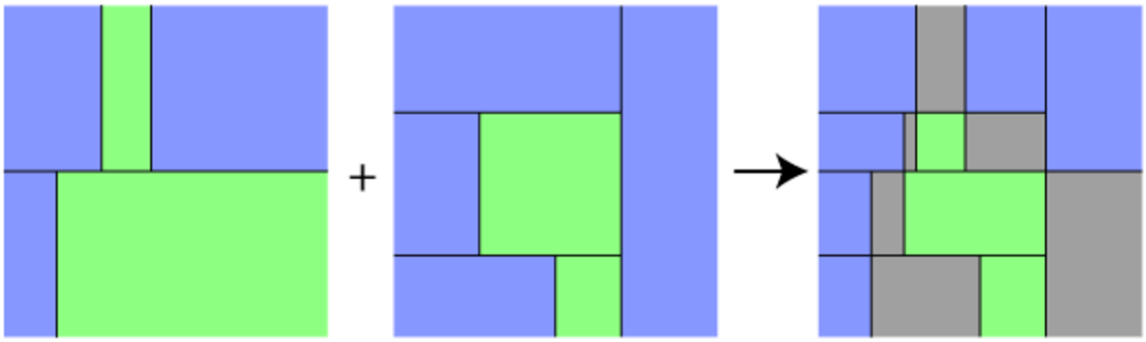
2h. Analysis.

(a) What is the decision rule at each split point? Write down the 1-line formula for the split point at the root node for one the trained decision trees inside the forest. Feel free to define any variables you need.

At each split point, suppose x is one of the feature variable,
if x < threshold, goes to the left subtree,
if x > threshold, goes to the right subtree

(b) Why does the resulting image look like the way it does? What shape are the patches of color, and how are they arranged?

Each block of the color patches is quadrilateral, An example with a random forest consisting of two decision trees is shown in the Figure below.



(c) Easy: How many patches of color may be in the resulting image if the forest contains a single decision tree? Define any variables you need. 2^depth

(d) Tricky: How many patches of color might be in the resulting image if the forest contains n decision trees? Define any variables you need. C(n, n*2^depth)

Citations

https://shapeofdata.wordpress.com/2013/07/09/random-forests/

http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html