Homework 4

Hong Gan hg383@cornell.edu
Zhen Liu zl557@cornell.edu

November 29, 2016

1 Associate rule learning

As a data scientist employed by the wildly popular online retail web site Nile.com, your job is to maximize profits by bundling related items together based on your customers spending habits. On your first day of work, you put on your statistician hat and down- load a sampling of Nile.coms purchase logs for last month. Of course, Nile.com has an extensive catalog of N $\approx$ millions of conveniently packaged everyday items, all delivered to your door within two or three business days, shipping price not included. Since any combination of these items could be interesting to the marketing team, you decide to use association rule mining to decide which item correlations could be the most profitable. The table below lists sets of items that appear in customers shopping baskets along with the count of purchases that contain those items. For example, the fifth row says that 60,159 transactions contained both dog food and cat food in addition to whatever else those customers purchased.

The entire purchase log is too big to download, but you can probably already make some interesting inferences. For some of the questions below, the table does not contain enough information to give an exact answer. In these cases, give the tightest upper and lower bounds for the possible value.

Item set	# of Purchases contain at least these items
{}	927,125
{DogFood}	80,915
{CatFood}	185,279
{CatLitter}	130,122
{CatFood,DogFood}	60,159
{CatFood,CatLitter}	120,091
{Burgers}	29,751
{Burgers,VitaminCTablets}	231
{VitaminCTablets,ArtisianTapWater}	25
{Burgers,Buns,Ketchup}	15,293

a.How many purchases did Nile.com fulfill last month?
the total number of purchases is 927125.

b.How many customers purchased Vitamin C tablets in addition to their other items?

At least 231 customers and at most 927125 purchased Vitamin C tablets in addition to their other items.

c.How many customers purchased only cat food and nothing else?

185279 customers purchased only cat food and nothing else.

d.What is the (possible) value of SUPP($\{$Burgers,VitaminCTablets,Ketchup$\}$)?

the total number of purchases are 927,125, and the number of purchases contains at least Burgers and VitaminCTablets are 231, and the number of purchases contains at least Burgers, Buns, Ketchup are 15,293. So the upper bound should be $\frac{231}{927125}$, while the lower bound can be as small as 0.

e.What is the (possible) value of SUPP($\{$Burgers,Buns$\}$)? How do you know?

the support of Item set $\{$Burgers, Buns$\}$ has a upper bound that is equal to the support of Item set $\{$Burgers$\}$, which is $\frac{29751}{927125}$, and has a lower bound which is equal to the support of item set $\{$Burgers, Buns, Ketchup$\}$, $\frac{15293}{927125}$.

f.What is the (possible) value of CONF($\{$Burgers$\} \Longrightarrow \{$VitaminCTablets$\}$)? Does this seem like an interesting promotion opportunity?

$$\begin{aligned} CONF(\{Burgers\} \Longrightarrow \{VitaminCTablets\}) &= \frac{support(\{Burgers, VitaminCTablets\})}{support(\{Burgers\})} \\ &= \frac{\frac{231}{927125}}{\frac{29751}{927125}} \\ &= \frac{231}{29751} \end{aligned}$$

g.What is the (possible) value of $CONF(\{DogFood, CatFood\} \Longrightarrow \{CatLitter\})$?

$$CONF(\{DogFood, CatFood\} \Longrightarrow \{CatLitter\}) = \frac{support(\{DogFood, CatFood, CatLitter\})}{support(\{DogFood, CatFood\})}$$

so the problem turned into find out the value of $support(\{DogFood, CatFood, CatLitter\})$, $support(\{DogFood, CatFood, CatLitter\})$ can be as large as $support(\{DogFood, CatFood\})$, also can be 0;

h.What is the (possible) value of $LIFT(\{DogFood\} \Longrightarrow \{CatFood\})$?

$$\begin{aligned} L(D \Rightarrow C) &= \frac{C(D \Rightarrow C)}{T(C)} = \frac{T(D \Rightarrow C)}{T(D)T(C)} \\ &= \frac{support(D \bigcup C)}{support(D)support(C)} \\ &= \frac{\frac{60159}{927125}}{\frac{80915}{927125}\frac{185279}{927125}} \\ &= 3.72035 \end{aligned}$$

i.Suppose you wish to run the APriori algorithm with a minimum support of 0.1, 10%

of purchases. You begin by gathering the support for all item sets of length1. Which pairs of items could APriori definitely eliminate using the downward closure property. {Dogfood},{Burgers}

j.Which pairs of items could APriori probably eliminate if you knew more of the table? {VitaminCTablets},{ArtisianTapWater},{Buns},{Ketchup}

k.Which pairs of items could APriori definitely not eliminate?
{CatFood,CatLitter} is frequent, so all its subset is frequent. So {CatFood} {CatLitter} could definitely not eliminated.

2 Neural Network as function approximators

Design a feed-forward neural network to approxi- mate the 1-dimensional function given in Fig. The output should match exactly. How many hidden layers do you need? How many units are there within each layer? Show the hidden layers, units, connections, weights, and biases.
Use the ReLU nonlinearity for every unit. Every possible path from input to output must pass through the same number of layers. This means each layer should have the form

$$Y_i = \sigma(W_i Y_{i-1}^T + \beta i)$$

Where $Y_i \in \Re^{d_i - 1}$ is the output of the ith layer, $W_i \in \Re^{d_i \times d_{i-1}}$ is the weight matrix for that layer, $Y_0 = x \in \Re^{1\times 1}$ and the ReLU nonlinearity is defined as

$$\sigma(x) = \begin{cases} x & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

first write out the function

$$f(x) = \begin{cases} 0 & x \leq 1 \\ 2x - 2 & 1 < x \leq 2 \\ \frac{1}{3}x + \frac{4}{3} & 2 < x \leq 5 \\ 2x - 7 & 5 < x \leq 6 \\ -\frac{5}{3}x + 15 & 6 < x \leq 9 \\ 0 & x > 9 \end{cases} \tag{2}$$

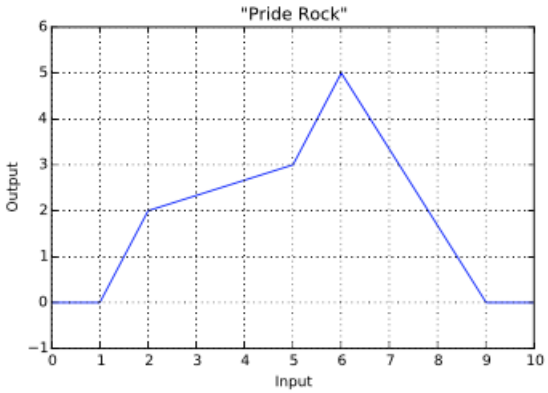
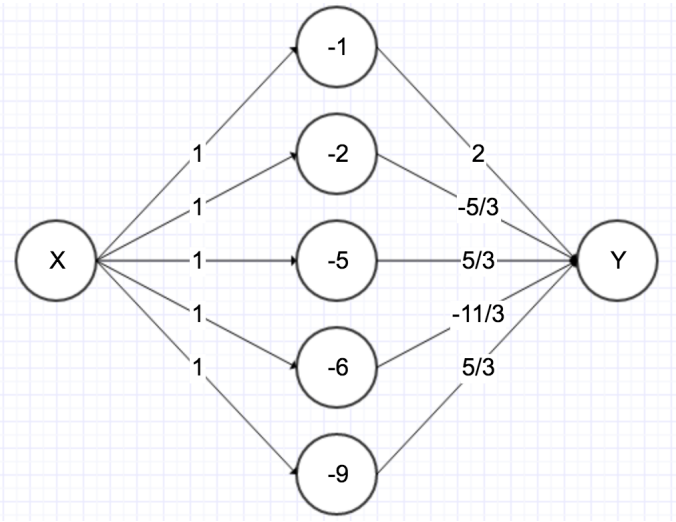


Figure 1: Example function to approximate using a neural network.

solution: express the input function as a sum of weighted and scaled ReLU units. The solution has one hidden layer.



3 Approximating Images with neural networks

3.1 The Structure of the network.

The given example in the link is a Neural Net Regression. The first layer is an input layer, at which the input data is given to the Neural network. The next 7 layers will be fully connected layers of neurons, They are called the hidden layer. Hidden layers are used for training, learning weights and bias through iterations the last layer is a regression loss layer.

3.2 Regression Loss

According to *convnetjs/src/convnet_layers_loss.js*, the Regression Layer uses L2 loss function $\sum_i ||x_i - y_i||_2$, x is its input and y is the user provided correct values.The loss function quantifies the amount by which the prediction deviates from the actual values. In this example the loss quantifies the amount by which the painted image deviates from the original image.

3.3 Plot the Loss

In this part, we download the demo code and make a little change to the javascript, we record the loss value every 50 iterations until 5,000 iterations in all. Then we plot it using matplotlib.

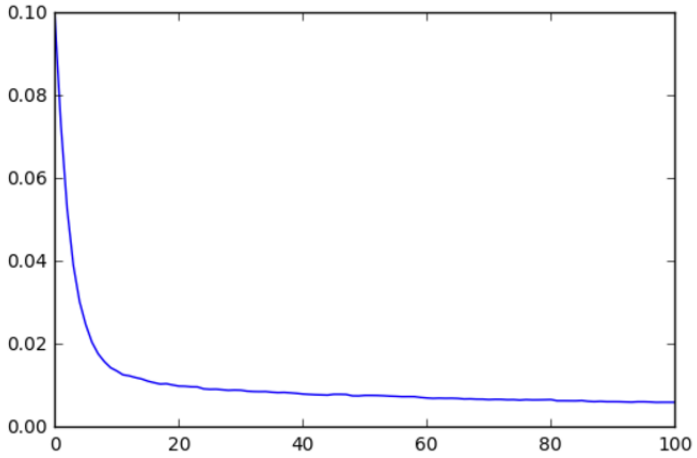


Figure 2: Example function to approximate using a neural network.

3.4 Lowering Learning Rate

Can you make the network converge to a lower loss function by lowering the learning rate every 1,000 iteration?
After experiments, if the neural network keep running until 5,000 iterations with learning rate of 0.01, the final loss at the 5000th iteration is 0.005356. Attempting to lower the learning rate every 1,000 iteration, we found that the result is better than the previous one, with loss value equals to 0.003370.

3.5 Lesion Study

The text box contains a small snippet of Javascript code that initializes the network. You can change the network structure by clicking the Reload network button, which simply eval-

uates the code. Lets perform some brain surgery: Try commenting out each layer, one by one. Report some results: How many layers can you drop before the accuracy drops below a useful value? How few hidden units can you get away with before quality drops noticeably?

num of hidden layers	loss value
1	0.0251
2	0.0125
3	0.0072
4	0.0057
5	0.0056
6	0.0054

After testing with different number of hidden layers, we found that after cutting the number of hidden layers to 3, the loss value has a sudden drop.

num of neurals	loss value
15	0.0069
17	0.0059
18	0.0054
20	0.0054

Trying to run with fewer hidden units in each layer, we found that after cutting the neuron number to 17 or below, the quality drop noticeably.

3.6 Add more hidden layers

Try adding a few layers by copy+pasting lines in the network definition. Can you noticeably increase the accuracy of the network?
After testing, we found that add a few layers in the network definition can not noticeably increase the accuracy of the network. we tested by add 4 more hidden layers into the network. The loss value after 5000 iterations is still 0.0056.

4 Coding Part

For coding part, the analysis and code are attached in the following pages as HTML.

Random Forest

November 28, 2016

```
In [1]: import os
        import numpy as np

        from sklearn.ensemble import RandomForestRegressor
        from skimage import io # pip3 install scikit-image
        from skimage import img_as_float

        from ipywidgets import IntProgress
        from IPython.display import display # pip3 install notebook ipywidgets
```

0.0.1 2a. Start with an image of the Mona Lisa. If you don't like the Mona Lisa, pick another interesting image of your choice.

We actually picked a picture in the folder named face.png. This picture is 420px x 400px. Since all commands are dynamic, you could replace this picture with other pictures and change the file name accordingly.

```
In [2]: # Read in and convert to float 0.0 - 1.0
        f = io.imread('face.png')
        fileInput = img_as_float(f)
```

0.0.2 2b. Preprocessing the input

In this step, firstly an array sized $width \times height$ is created. Then, a numpy random generator will generate an array 5000 random numbers without replace back as index numbers for the training set. After that, those data points in the raw file with matched index will be placed in an array for further usage as training set.

```
In [3]: # Insert all data into a dataset for spliting training and testing
        data_set = [None]*(fileInput.shape[0]*fileInput.shape[1])
        x = 0
        for i in range(fileInput.shape[0]):
            for j in range(fileInput.shape[1]):
                data_set[x] = [i, j, fileInput[i][j]]
                x += 1
```

```
In [4]: # Demo data entry, (x, y, [R, G, B, (alpha)])
        # Alpha channel only exist in PNG
```

```
In [5]: # 5000 Random point array
        training_X = np.random.choice(len(data_set), 5000)
        training_set = []
        for i in training_X:
            training_set.append(data_set[i])
```

0.0.3 2c. Preprocessing the output

We would like to have our output as colored pictures, so that we did not convert the image to grayscale. For each piexl, we have the Red, Green, and Blue channel. Bundle three channel value at the same time, for each regressor, the input is the location for the pixel as (x, y), and the output is the intensities for each color channel.

```
In [6]: training_set_X = []
        training_set_Y = []

        for i in range(0, len(training_set)):
            training_set_X.append([training_set[i][0], training_set[i][1]])
            training_set_Y.append([training_set[i][2][0],
                                   training_set[i][2][1],
                                   training_set[i][2][2]])
```

0.0.4 2d. Rescale the pixel intensities

We considered this problem during the input stage. In fact, we utilized the skimage lib and used a function "img_as_float". This function tansferred the input file as float from 0.0-1.0, so that we do not need to rescale it again.

0.0.5 2e. Preprocessing steps

We did not apply any preprocessing steps here. For speed optimization, we could change the size of the input, to make it a smaller image to achieve a faster processing speed.

0.0.6 2f. Build the final image

For this step, a map which has a size of picture's resolution is created, and we formatted it as an array for easier processing.

```
In [7]: # Plot a map as picture's resolution, add each [x, y] pair as input
        testing_X = []
        for i in range(fileInput.shape[0]):
            for j in range(fileInput.shape[1]):
                testing_X.append([i, j])
```

0.0.7 2g. Experimentation.

In this step, we build different random forest with the sklearn lib.

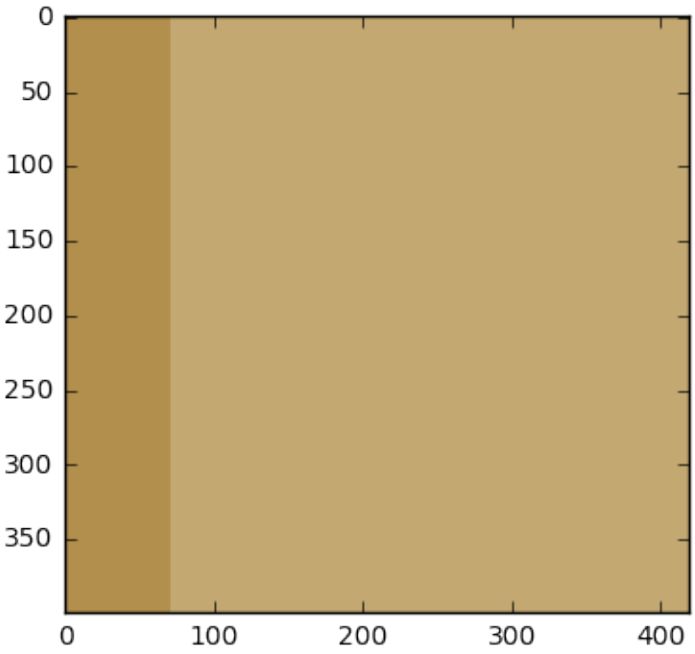
```
In [8]: q = IntProgress()
        p = IntProgress()
        r = IntProgress()
        depthList = [1,2,3,5,10,15]
        trees = [1,3,5,10,100]
```

0.0.8 2g. (a). Repeat the experiment for a random forest containing a single decision tree

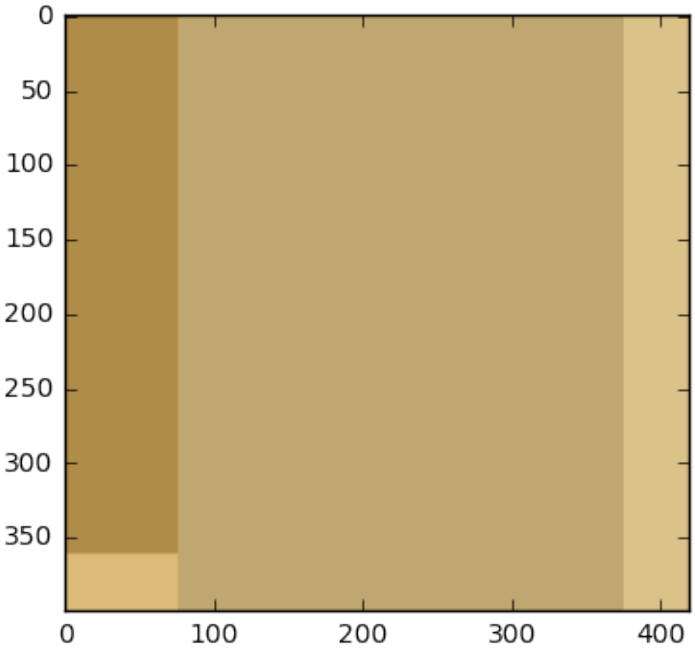
For this experiment, we created different random forest containing a single decision tree, but with different depths. So that we controlled the parameter "n_estimators" as 1, and used a list "depthList" to pass different tree depth to the regressor.

```
In [9]: display(q)
        for n in depthList:
            print('Now predict with depth '+str(n))
            q.value = (depthList.index(n)+1)/len(depthList)*100
            q.description = 'Depth ' + str(depthList.index(n)+1) \
                + '/' + str(len(depthList))
            # Create 3 Regressors for each color channel
            clf = RandomForestRegressor(n_estimators=1, n_jobs=1, max_depth=n)
            clf = clf.fit(training_set_X, training_set_Y)

            # Plot another map as picture's resolution,
            # add each [R, G, B] pair as output
            testing_Y = []
            testing_Y = np.array(clf.predict(testing_X))
            testing_Y = testing_Y.reshape(fileInput.shape[0], fileInput.shape[1], 3



            # Plot!
            import matplotlib.pyplot as plt
            plt.imshow(testing_Y)
            plt.savefig('./Output/depth_'+str(n)+'.png')
            plt.show()
```

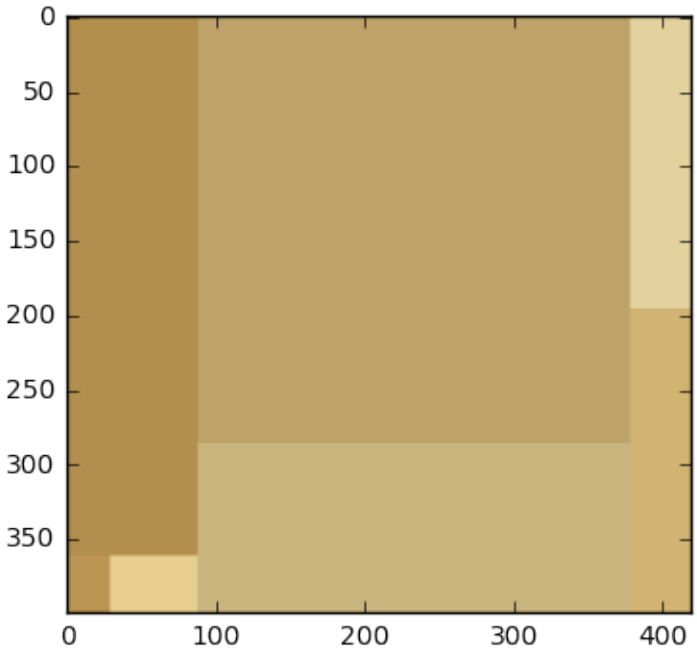
```
Widget Javascript not detected.  It may not be installed properly. Did you enable t


Now predict with depth 1
```



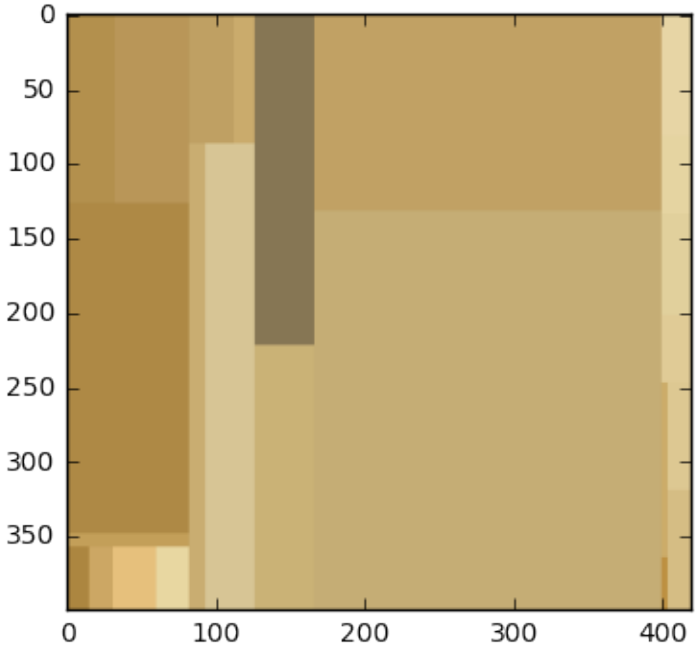
Now predict with depth 2



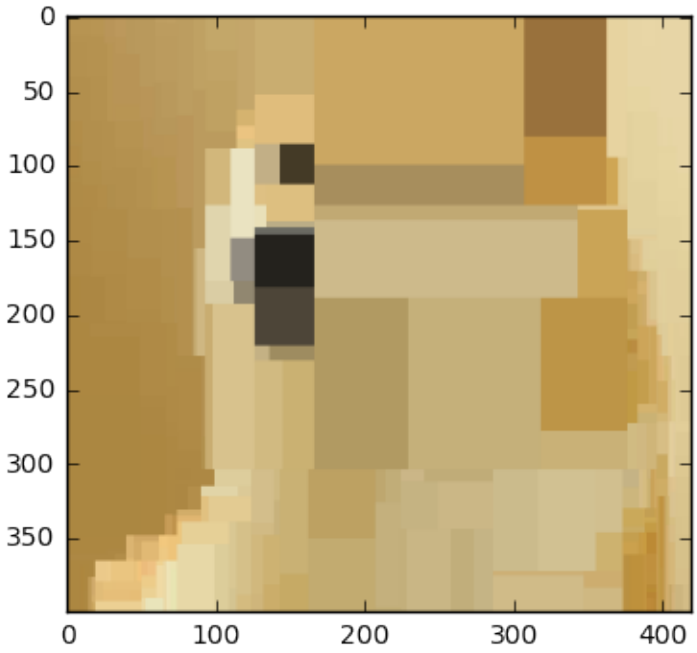
Now predict with depth 3



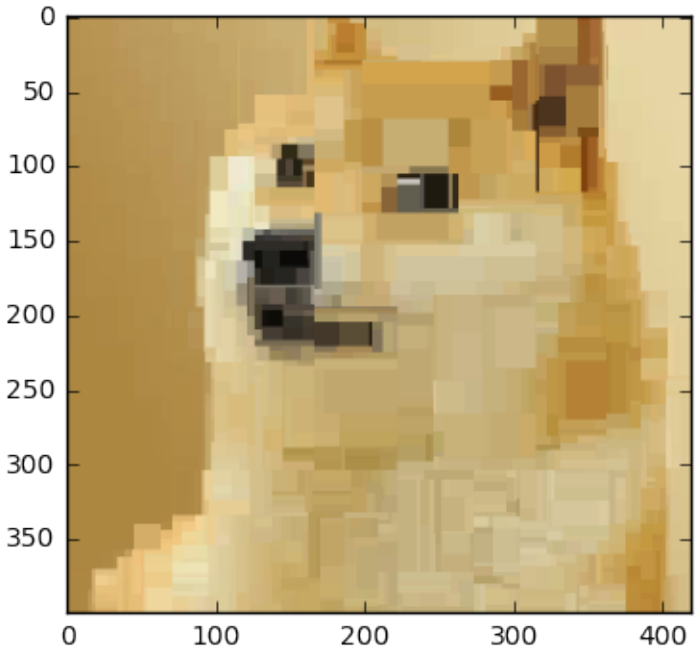
Now predict with depth 5



Now predict with depth 10



Now predict with depth 15



0.0.9 2g. (b). Repeat the experiment for a random forest containing a single decision tree

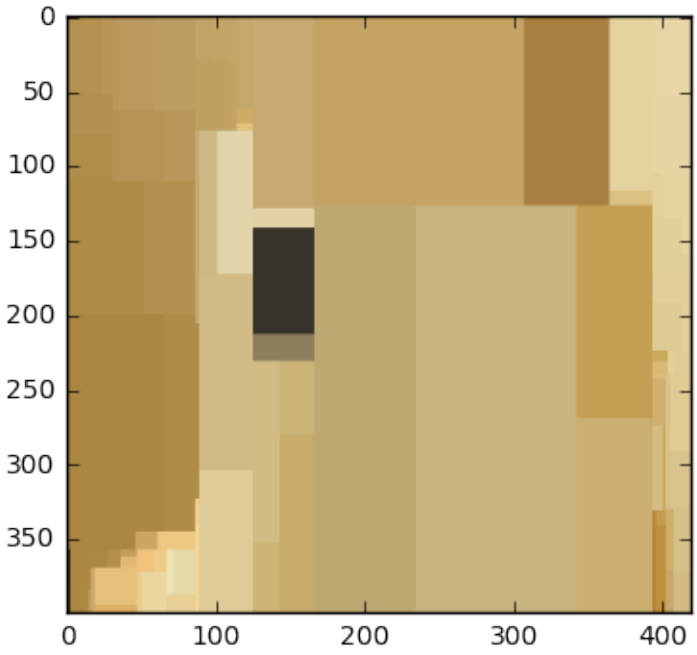
For this experiment, we created different random forest containing depths of 7, but different number of decision trees. So that we controlled the parameter "max_depth" as 7, and used a list "trees" to pass different tree numbers to the regressor. Although the picture cannot reflect the result, but adding number of decision tree could reduce the overfitting problem.

```
In [10]: display(r)
         for n in trees:
             print('Now predict with tree '+str(n))
             r.value = (trees.index(n)+1)/len(trees)*100
             r.description = 'Trees ' + str(trees.index(n)+1) \
                 + '/' + str(len(trees))
             # Create 3 Regressors for each color channel
             clf = RandomForestRegressor(n_estimators=n, n_jobs=-1, max_depth=7)
             clf = clf.fit(training_set_X, training_set_Y)

             # Plot another map as picture's resolution,
             # add each [R, G, B] pair as output
             testing_Y = []
             testing_Y = np.array(clf.predict(testing_X))
             testing_Y = testing_Y.reshape(fileInput.shape[0], fileInput.shape[1],
```

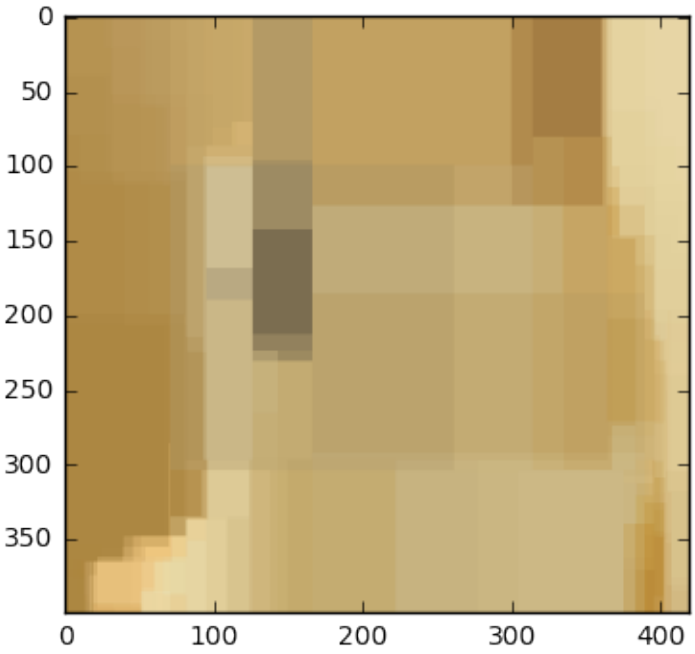
```
# Plot!
import matplotlib.pyplot as plt
plt.imshow(testing_Y)
plt.savefig('./Output/tree_'+str(n)+'.png')
plt.show()
```

```
Widget Javascript not detected.  It may not be installed properly. Did you enable t
```

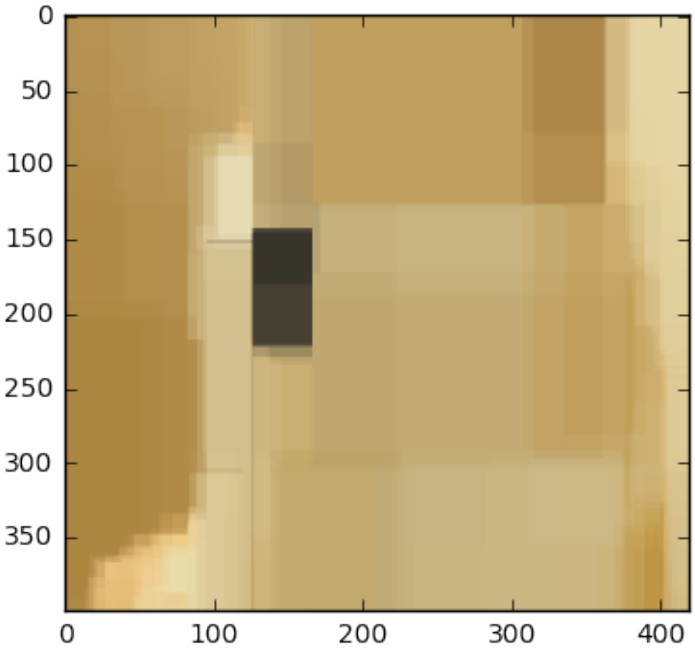
```
Now predict with tree 1
```



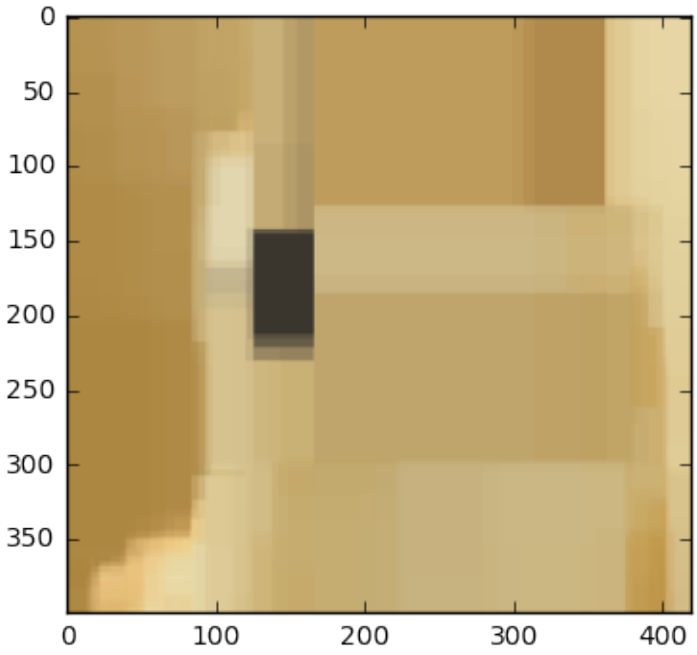

```
Now predict with tree 3
```



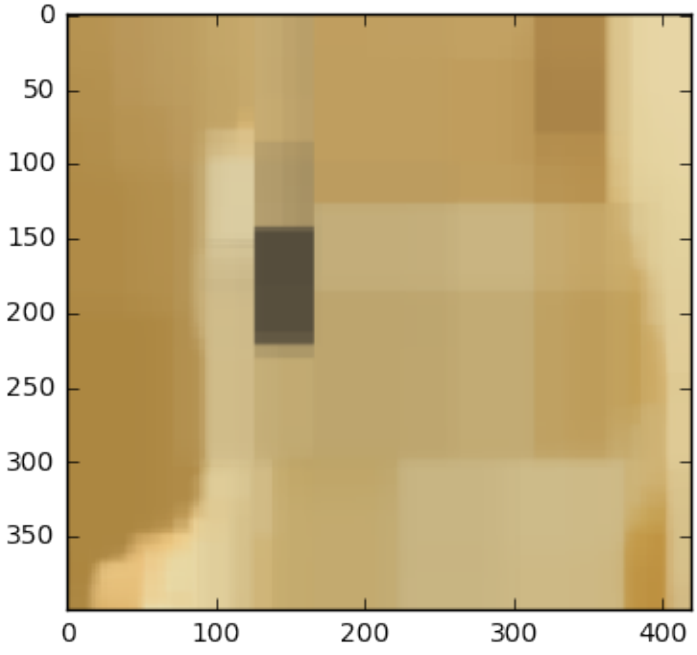
Now predict with tree 5



Now predict with tree 10



Now predict with tree 100

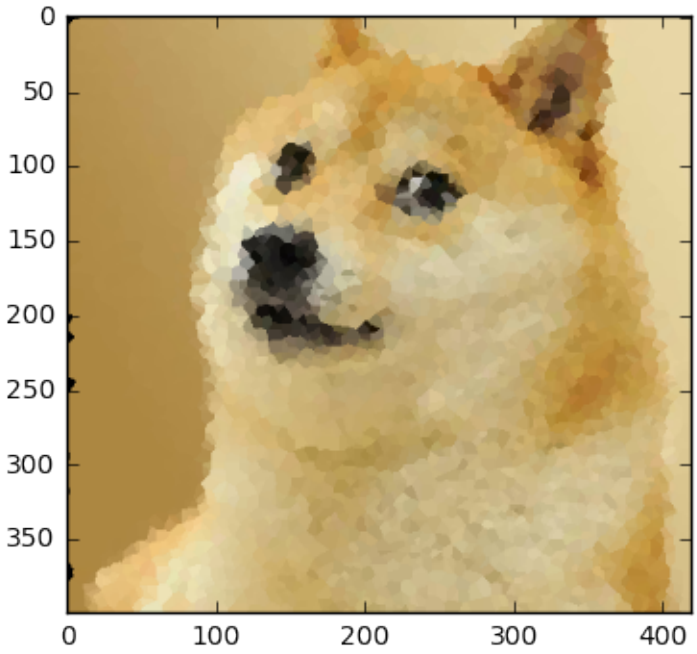


0.0.10 2g. (c). Repeat the experiment using a k-NN regressor, for k = 1.

This tree looks different from random forest regressor. Because in this algorithm, points are colored with the training point nearest to it. So that it is not nessary for it to form a rectangle area.

In [11]:
```
from sklearn.neighbors import KNeighborsRegressor

# p = IntProgress()
# display(p)

print('Now is k-NN')

clf = KNeighborsRegressor(n_neighbors=1)
clf = clf.fit(training_set_X, training_set_Y)

# Plot another map as picture's resolution,
# add each [R, G, B] pair as output
testing_Y = []
testing_Y = np.array(clf.predict(testing_X))
testing_Y = testing_Y.reshape(fileInput.shape[0], fileInput.shape[1], 3)

# Plot!
import matplotlib.pyplot as plt
plt.imshow(testing_Y)
plt.savefig('./Output/kNN_1.png')
plt.show()
```

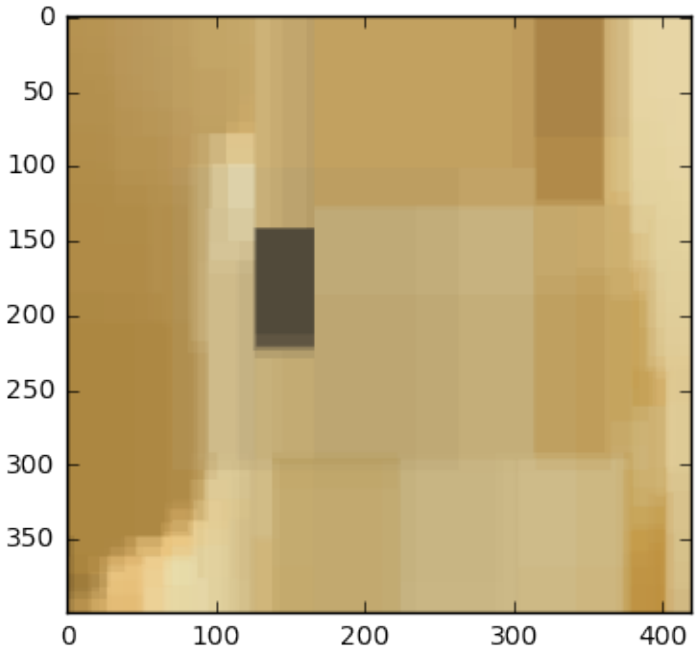
```
Now is k-NN
```



0.0.11 2g. (d). Experiment with different pruning strategie

Basically, we add two extra parameters. min_samples_split means the minimum number of samples required to split an internal node, and min_samples_leaf means the minimum number of samples required to be at a leaf node. This would reduce the possiblity that a single value create a node or a leaf, which reduced the overfit problem.

In [12]:
```
# display(q, p)

        clf = RandomForestRegressor(n_estimators=5, n_jobs=1, max_depth=7, min_sam
        clf = clf.fit(training_set_X, training_set_Y)

        # Plot another map as picture's resolution,
        # add each [R, G, B] pair as output
        testing_Y = []
        testing_Y = np.array(clf.predict(testing_X))
        testing_Y = testing_Y.reshape(fileInput.shape[0], fileInput.shape[1], 3)


        # Plot!
        import matplotlib.pyplot as plt
        plt.imshow(testing_Y)
        plt.savefig('prun.png')
        plt.show()
```



0.1 2h. Analysis.

0.1.1 2h a).

For each split point, the input is the (x, y) value and the output is either the color result, or the path to next node. For example, the root node might looks like

$$\text{root} \begin{cases} \text{node 1 if } x \text{ is greater than } \beta \\ \text{node 2 otherwise} \end{cases}$$

0.1.2 2h b).

The image looks like created by putting together different rectangle with different colors. Each patch is rectangle, and the color is different from other patches close to it. The arrangement, or the location of each patch is around the key pixel.

0.1.3 2h c).

If there is only single decision tree, then there would be at most 2^{depth} colors. In the situation, which in the training sample there are enough colors to be learn as result, the tree could have most 2^{depth} leaf nodes, thus a total of 2^{depth} different color could be produced. But the tree is not nessarily to be a fully grown tree, thus any number below could also be possible.

0.1.4 2h d).

We would say that possible color of patches would still be 2^{depth} . Because increasing number of trees do not increase the total number of leaf nodes in each sub-tree, and the forest only would select a result from exsiting sub-trees. Thus even there are n n decision trees, possible color of patches would still be 2^{depth} .