CS 5875 Applied Machine Learning

Homework 4

Weiming Zhang, Yan Jiang

I. Objective

In this homework we are going to cover random forests, boosting and CNN.

I. Problem 1: Association Rule Learning

See in the back.

II. Problem 2: Neural Networks As Function Approximators

See in the back.

III. Problem 3: Approximating Images With Neural networks

See in the back.

IV. Problem 2: Random Forest For Image Approximation

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

• What other preprocessing steps are necessary for random forests inputs? Describe them, implement them, and justify your decisions. In particular, do you need to perform mean subtraction, standardization, or unit-normalization?

We didn't use other preprocessing steps. In particular, mean subtraction, standardization or unit normalization are not necessary as decision tree does not get effected by the structure of the data.

c) *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.*

I am using the second approach in the description: Mapping Regress all three values at once, so your function maps (x, y) coordinates to (r, g ,b) values:

$$f: R2 \rightarrow R3$$

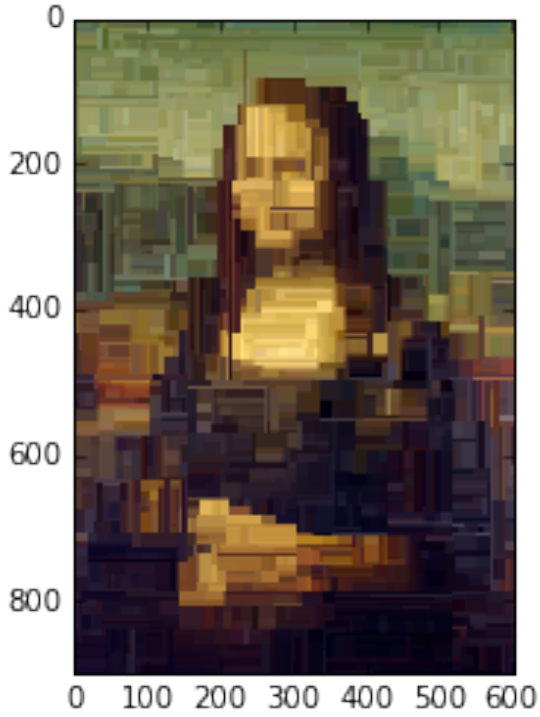
d) *Rescale the pixel intensities to lie between 0.0 and 1.0. (The default for pixel values may be between 0 and 255, but your image library may have different defaults.)*

See in code.

e) *What other preprocessing steps are necessary for random regression forest outputs? Describe them, implement them, and justify your decisions.*

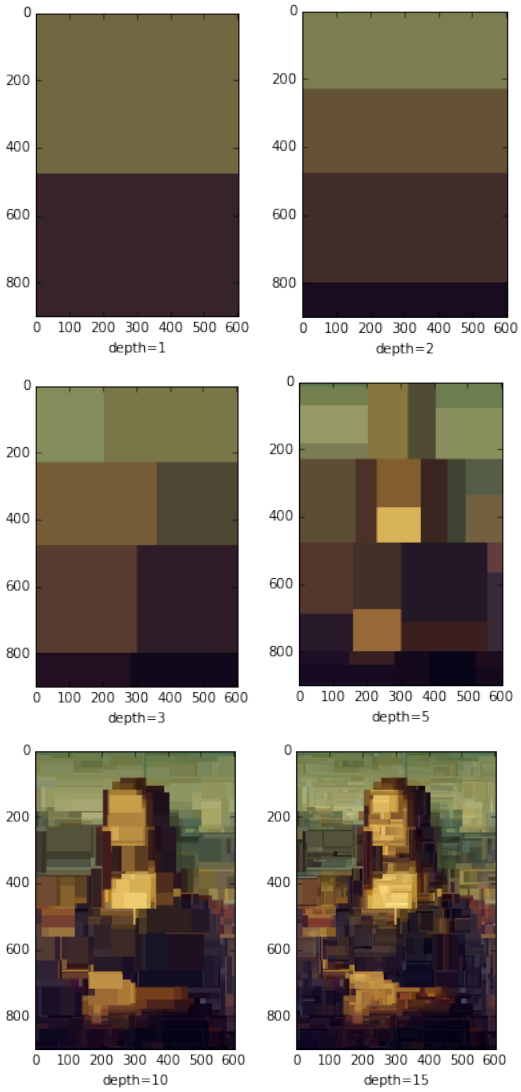
None.

f) *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.*



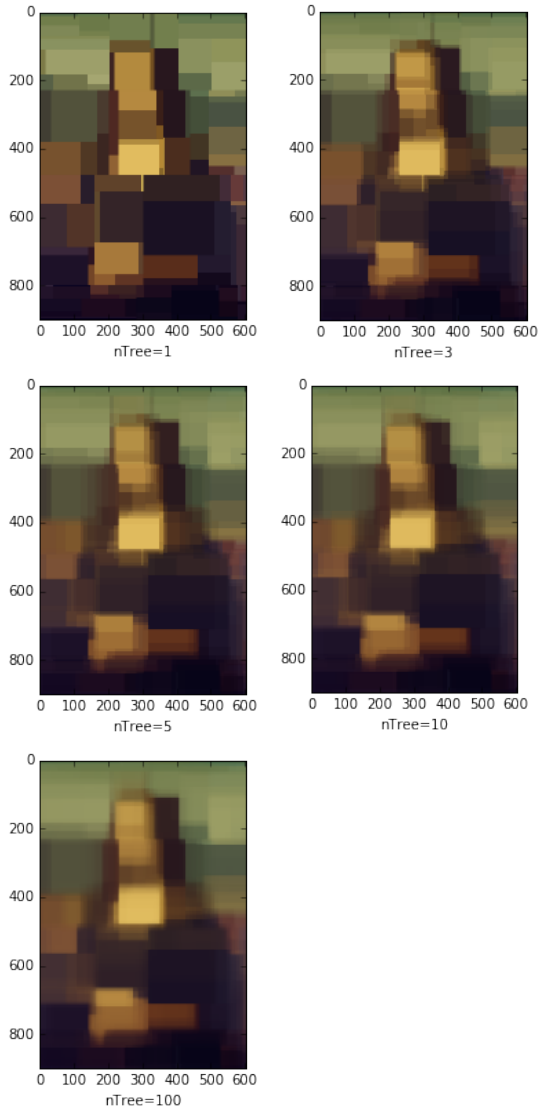
g) ***Experimentation***

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.*



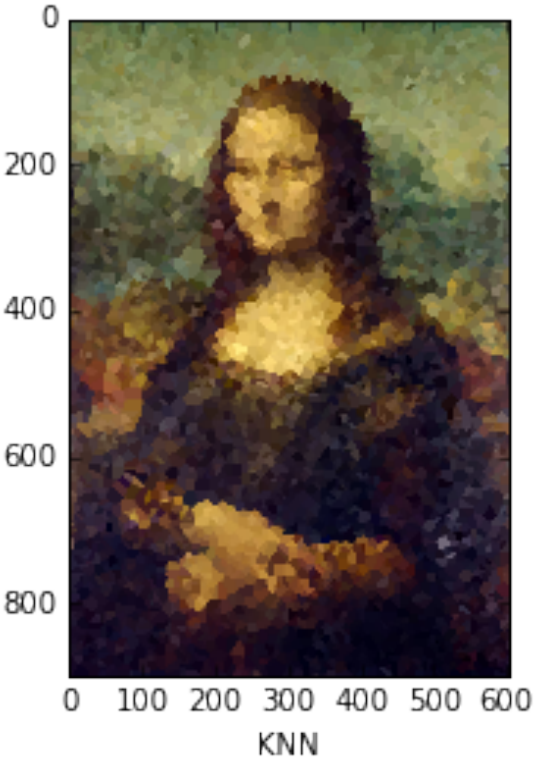
With depth=1, it became a binary case that the image is divided into two colors. As we increase the depth, we get more diversity in color and more precise of an image.

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.*



Each different tree yields a different subset of data, which can result in highly correlated predictors, limiting the amount of variance reduction. So we can see that we get smoother image as we increase the number of trees.

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 Random Forest, we split the image based on the pixel position of the image compared with the threshold of each subtree in the forest. This results in the image being divided into small rectangles of same colors. However, in the KNN case, each pixel color is assigned to the closest one in the sample. The image therefore is made of small pieces of color, but the boundary between them will not be straight lines.

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

We didn't prune the tree as in our case the sample size is large.

h) 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.*

We take the position of the image's pixel (x, y) as input and output a node on the next level of the tree. If we've reached leaf node, then the output is the actual color. The formula might look like:
next = left_sub_tree, if x >= threshold;
right_sub_tree otherwise

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

The resulting image is made of patches of color due to we have limited number of colors as our samples. In random forest, segments of the images are being assigned into these colors so we observe patches of colors. The shapes are rectangle and there are overlaps. In the KNN case, each pixel color is assigned to the closest one in the sample. The image therefore is made of small pieces of color, but the boundary between them will not be straight lines.

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
Each leaf of the tree results in a patch of color. Since there is only one tree, the number of leaves is 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, n2^d)
n*2^d choose n
For a single tree there could be 2^depth leaves. For n trees, the leaves can be at most n*2^depth. There are n2^d choose n ways of combinations to choose n subtrees out of n2^d decisions.

Applied Machine Learning

Homwork 4 Written Exercise

1. **Association 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 download a sampling of *Nile.com*'s 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.**

   a. Sanity check: How many purchases did *Nile.com* fulfill last month?

{}	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
⋮	

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

   c. Sanity check: How many customers purchased *only* cat food and *nothing else*?

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

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

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

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

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

   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 length 1. Which pairs of items could *APriori* **definitely** eliminate using the downward closure property?

   j. Which pairs of items could *APriori* **probably** eliminate if you knew more of the table?

   k. Which pairs of items could *APriori* **definitely not** eliminate?

1.

(a) 927,125

(b) Lower bound: 231

(c) Higher bound: 185,279 − 60,159 − 120,091 = 5029
Lower bound: 185,279 − 120091 = 65188 ∴ 5029 ~ 65188

(d) (possible) value of SUPP({Burgers, Vitamin C Tablets, Ketchup})?

$$SUPP(\{Burgers, Vitamin\ C\ Tablets, Ketchup\}) \leq \frac{231}{927,125}$$

∴ 0 ~ 0.02%

(e) $\frac{15293}{927125} \leq SUPP(\{Burgers, Buns\}) \leq \frac{29751}{927125}$

∴ 1.6% ~ 3.2%

(f)

$$CONF(\{Burgers\} \Rightarrow \{Vitamin\ C\ Tablets\}) = \frac{231}{29751} = 0.8\%$$

which is less than 1%. It doesn't seem like an interesting promotion opportunity.

(g)

$$CONF(\{Dog\ Food, Cat\ Food\} \Rightarrow \{Cat\ Litter\}) \leq \frac{60159}{60159}$$

∴ 0 ~ 100%

(h) $LIFT(\{DogFood\} \Rightarrow \{CatFood\})$?

$$= \frac{CONF(\{DogFood\} \Rightarrow \{CatFood\})}{T(\{CatFood\})}$$

$$= \frac{T(\{DogFood\}, \{CatFood\})}{T(\{DogFood\})\,T(\{CatFood\})}$$

$$= \frac{6015/9275}{8015/18529} = 3.12$$

(i) $10\% \times 9275 = 927.5$

∴ The item pairs which contain "Dog Food" or "Burgers" could APriori definitely eliminate.

(j) If known the number of purchases containing {Vitamin C Tablets} only, {Artisian Tap Water} only, {Buns} only, or {Ketchup} only, all item pairs which contain one of the item above could probably be eliminated.

(k) {Cat Food, Cat Litter} should not be eliminated.

2. **Neural networks as function approximators.** Design a feed-forward neural network to approximate 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

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

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

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

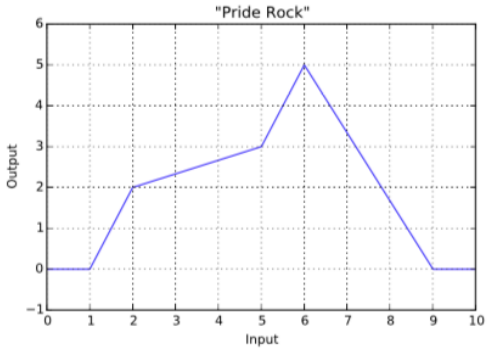


Figure 1: Example function to approximate using a neural network.

Hint 1: Try to express the input function as a sum of weighted and scaled ReLU units. Once you can do this, it should be straightforward to turn your final equation into a series of neural network layers with individual weights and biases.

Hint 2: If you're not convinced neural networks can approximate this function, see `http://neuralnetworksanddeeplearning.com/chap4.html` for an example of the flavor of solution we're looking for. (If you rely on this source, cite it!) However, keep in mind that your output must match the given input *exactly*.

2. Neural networks as function approximators.

Function in the graph:

$$y = \begin{cases} 0 & x \in [0,1) \\ 2x-2 & x \in [1,2) \\ \frac{1}{3}x+\frac{4}{3} & x \in [2,5) \\ 2x-7 & x \in [5,6) \\ -\frac{5}{3}x+15 & x \in [6,9) \\ 0 & x \in [9,10] \end{cases}$$

Calculate $a_i(b_i x + c_i)$ which add up to the target function.

$a_1(b_1x+c_1) = 2x-2$, $a_1=1$, $b_1=2$

$2x-2 + a_2(b_2x+c_2) = \frac{1}{3}x+\frac{4}{3}$, $a_2(b_2x+c_2) = -\frac{5}{3}x+\frac{10}{3}$

$a_2 = -1$, $b_2 = \frac{5}{3}$

$\frac{1}{3}x+\frac{4}{3} + a_3(b_3x+c_3) = 2x-7$, $a_3=1$, $b_3=\frac{5}{3}$

$2x-7 + a_4(b_4x+c_4) = -\frac{5}{3}x+15$, $a_4=-1$, $b_4=\frac{11}{3}$

The neural network has one layer with 4 units.

3. **Approximating images with neural networks.** In this question, you will implement your own neural network toolkit. You will be writing your own implementation from scratch, using *C++* and CUDA. You should calculate the derivatives of each layer by hand using pencil and paper. Please attach a scan of your paper notes to the homework.

Just kidding. We're not that mean. There are several good convolutional neural network packages that have done the heavy lifting for us. One of the more interesting (and well-written!) demos is called ConvNetJS. It is implemented in Javascript and runs in a modern web browser without any dependencies.

Take a look at convnet.js's "Image Painting" demo at: `http://cs.stanford.edu/people/karpathy/convnetjs/demo/image_regression.html`

a. Describe the structure of the network. How many layers does this network have? What is the purpose of each layer?

b. What does "Loss" mean here? What is the actual loss function? You may need to consult the source code, which is available on Github.

c. Plot the loss over time, after letting it run for 5,000 iterations. How good does the network eventually get?

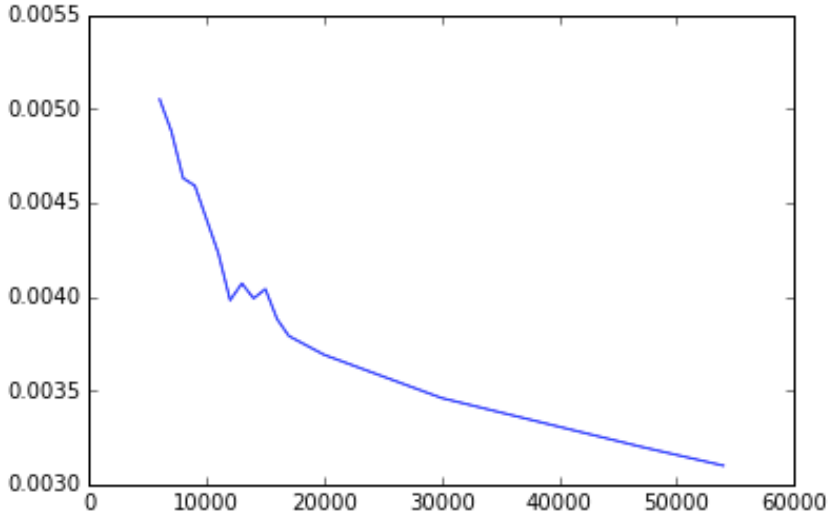
d. Can you make the network converge to a lower loss function by lowering the learning rate every 1,000 iterations? (Some *learning rate schedules*, for example, halve the learning rate every n iterations. Does this technique let the network converge to a lower training loss?)

e. 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 evaluates the code. Let's 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?

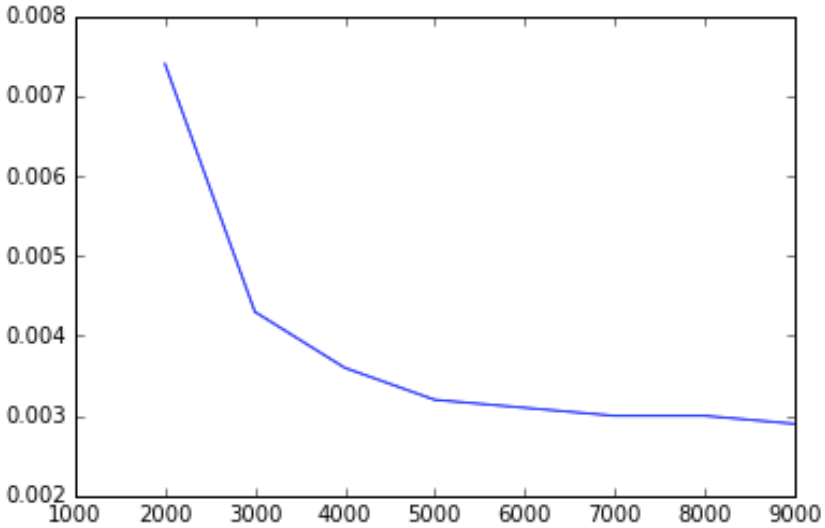
f. Try adding a few layers by copy+pasting lines in the network definition. Can you noticeably increase the accuracy of the network?

Solution:

a. 9 layers in all, including 1 input layer, 1 output layer and 7 layers each with 20 neurons.
b. The "Loss" here is the weighted sum of the current loss and the loss off the last iteration. smooth_loss = 0.99∗smooth_loss+0.01∗loss.
c. The loss converges to approx. 0.003 and gets a relatively good result.



d. The network also converges to approx. 0.003, but faster.



e. 3 layers can be dropped. 60 hidden united can be got rid of.
f. No, there's no noticeable increase.