CSCI-B 551 ASSIGNMENT 5 - MACHINE LEARNING

REPORT ON PERFORMANCE OF NEAREST NEIGHBOR, ADABOOST AND NEURAL NET UNDER DIFFERENT PARAMETERS

Dec 13, 2016

Madhavan, Sarvothaman (madhavas)

Nataraj, Raghavendra (natarajr)

Srivatsva, Prateek (pratsriv)

Contents

Chapter 1

Nearest Neighbor

1.1 INTRODUCTION AND PARAMETERS:

1.1.1 Approach

During training we simply store the training rows. During testing, for each test image we find k nearest neighbors and assign the orientation which most of the neighbors were assigned to

1.1.2 Parameters:

K is the only parameter we have in this approach

1.2 PERFORMANCE RESULTS

The performance ranges between 67% to 71%, with k = 30 performing the best with 71.26% while taking one of the highest time as well. With k = 1 and 2, We get the least time and one of the least performance as well.
This is clear from the plot of rolling mean. The peak in the plot corresponds to x value of 35, which is the rolling mean of three values 30, 35 and 40. The plot also tells that that we can stop looking for further k values since the performance has started plateauing.

Table 1.1: k-nearest neighbor

k	Performance (%)	Time (s)
1	67.23	1954
2	67.34	1949
5	69.67	2007
10	70.31	2298
15	69.88	2640
20	71.26	2650
25	70.52	2756
30	71.26	2175
35	70.94	2028
40	71.05	2061
45	71.16	2056
50	70.94	2065

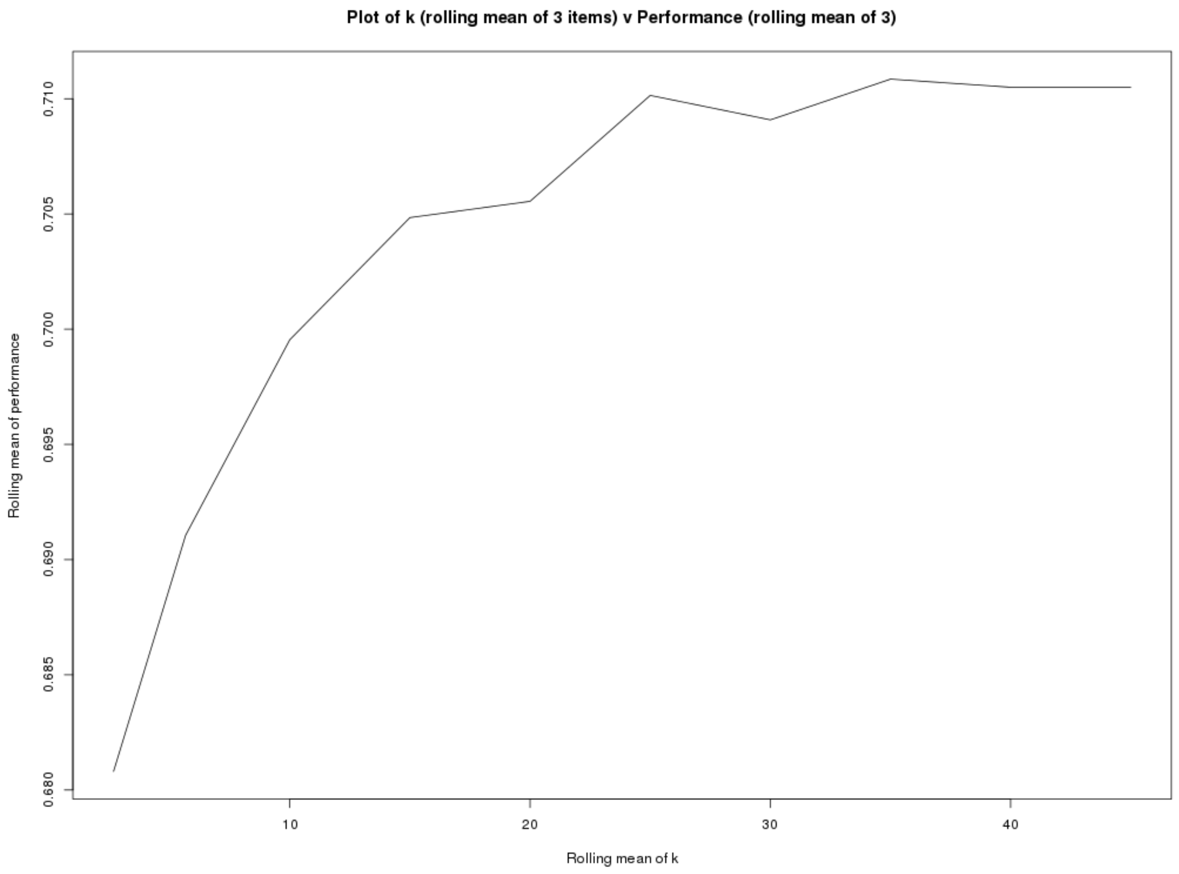

Figure 1.1: Plot of Performance

1.3 Examples

We give some examples that the nearest neighbor (k=1) incorrectly classified.
First image was misclassified as 180 (actual is 0) and second image was misclassified as 0 (actual is 180)

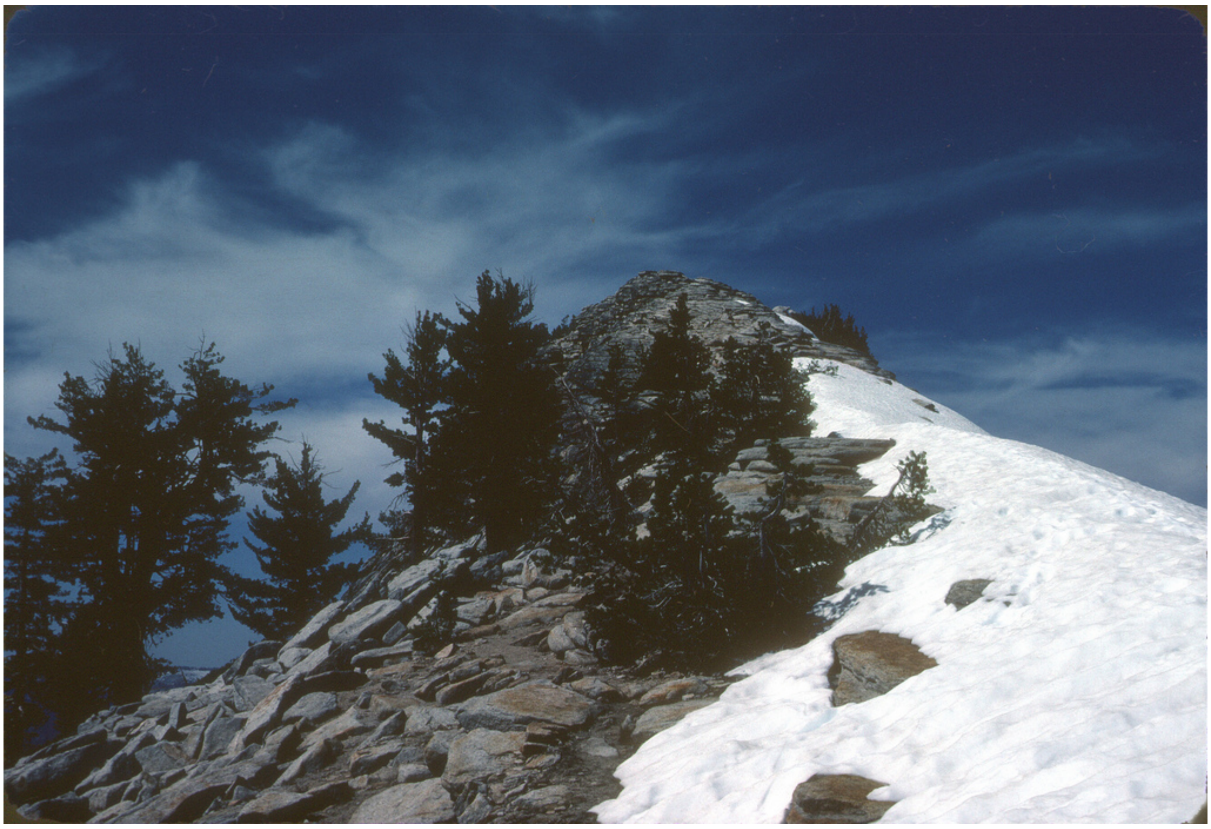

Figure 1.2: Example 1

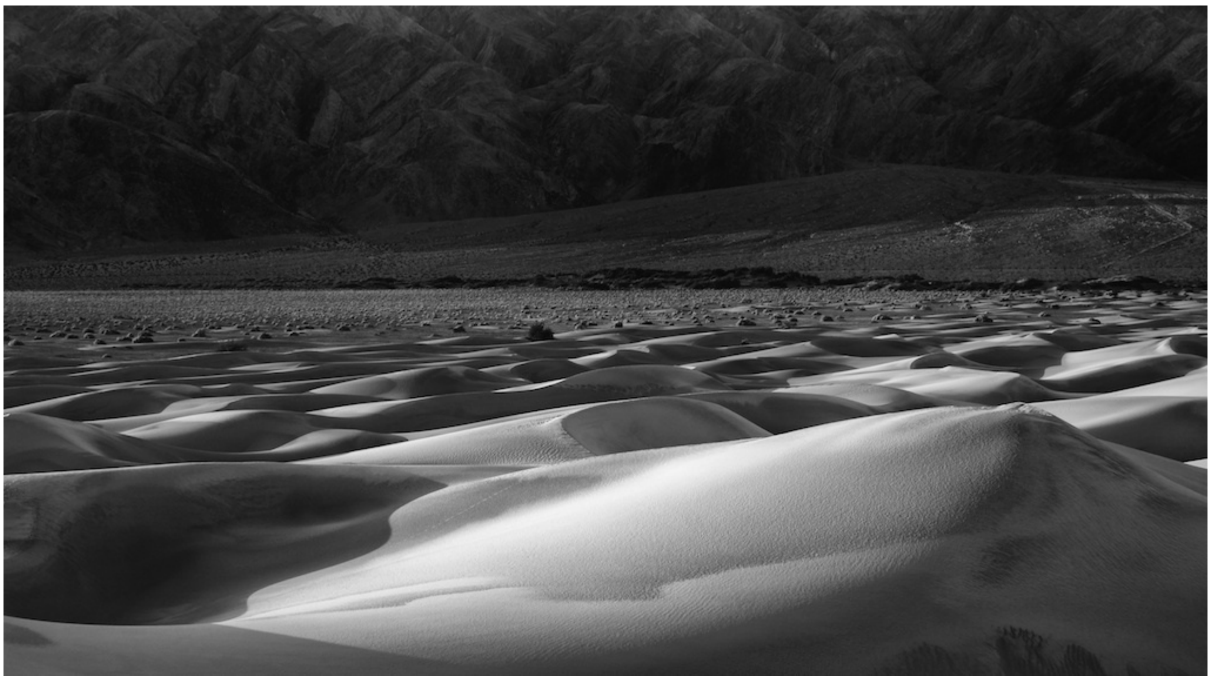

Figure 1.3: Example 2

Chapter 2

Adaboost

2.1 Introduction and Parameters:

For adaboost model, since trying out all possible combinations of vector pairs is highly time consuming, we chose to try out random pairs of vectors to use as decision stumps. For this section of the report, we call each of the stump as a model which is capable of saying Yes or No for any given orientation.

2.1.1 Approach

We randomly select 5 times *stump_count* vector pairs to compare. During the training phase, we try out all of these pairs for each model in the ensemble and choose *stump_count* number of stumps as model. The above process is performed for each of the orientation. The selection of stumps would adhere to the adaboost concept of giving preference through weights to the misclassified training data. At the end of training we are would have,

$$\text{No of stumps (or models)} = stump\_count \times \text{\# of orientations}$$

$$\text{No of stumps (or models)} = stump\_count \times 4$$

During the testing phase, each of the *stump_count* models would vote on whether or not the current test item is in the current orientation. Whichever orientation gets the maximum percent of votes would be chosen as the orientation.

2.1.2 Parameters

We measure the performance of the model in terms of the parameter *stump_count*

2.2 Performance Results

Table 2.1: Adaboost

stump count	Performance %	Time(s)
5	48.78	23
10	57.37	69
15	60.55	167
20	58.32	298
25	58.01	461
30	57.69	696
35	59.28	998
40	58.11	1222
45	60.98	1491
50	60.45	1911
55	61.51	2351

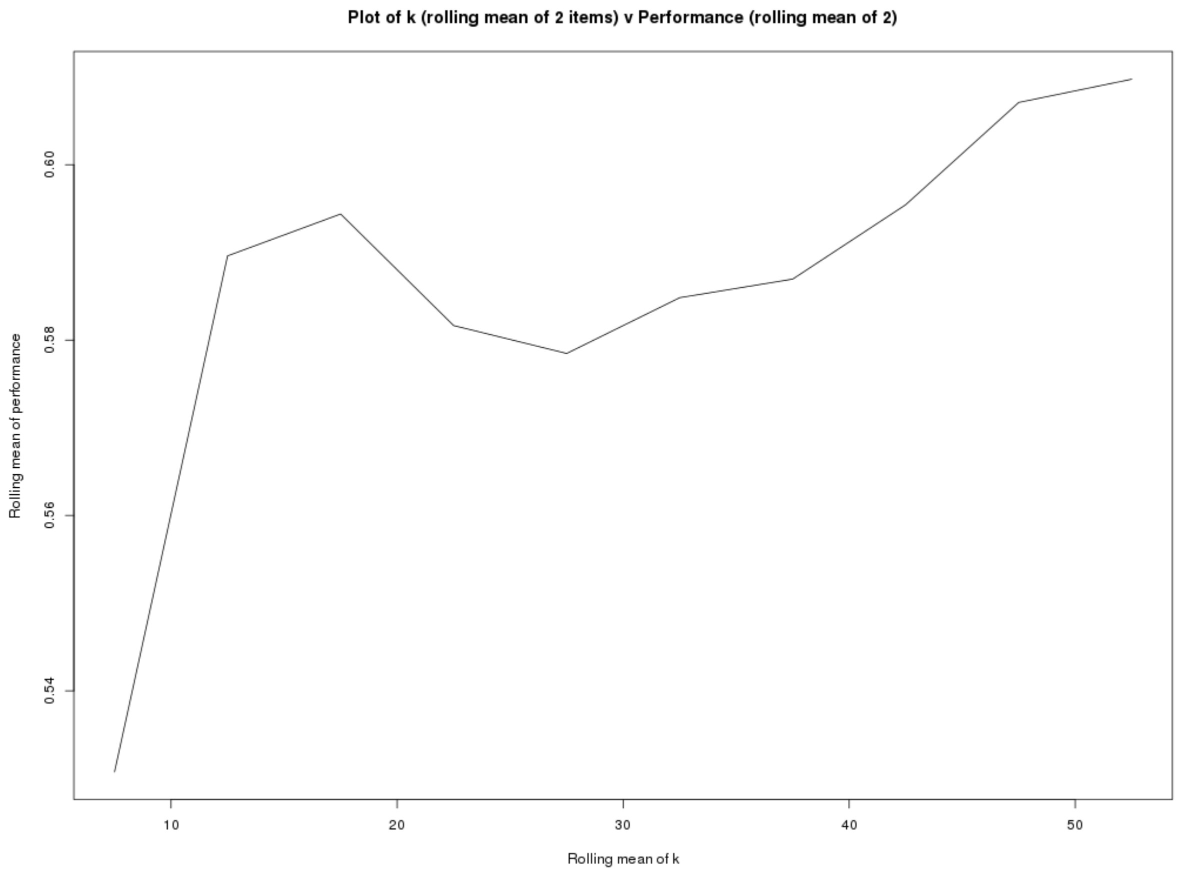

Figure 2.1: Example 1

2.3 EXAMPLES

The first image was classified as 0 (truth was its 180) and the second image was classified as 90 (actually 180)

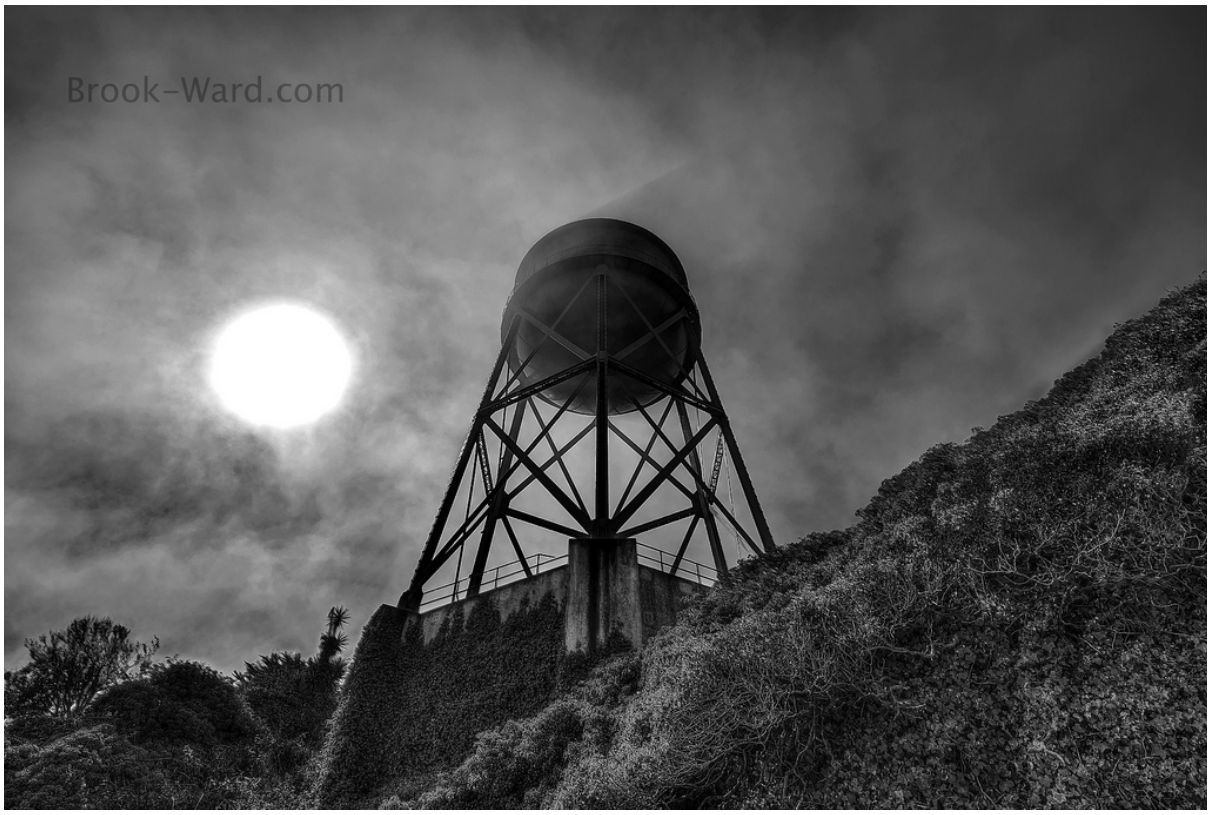

Figure 2.2: Example 1

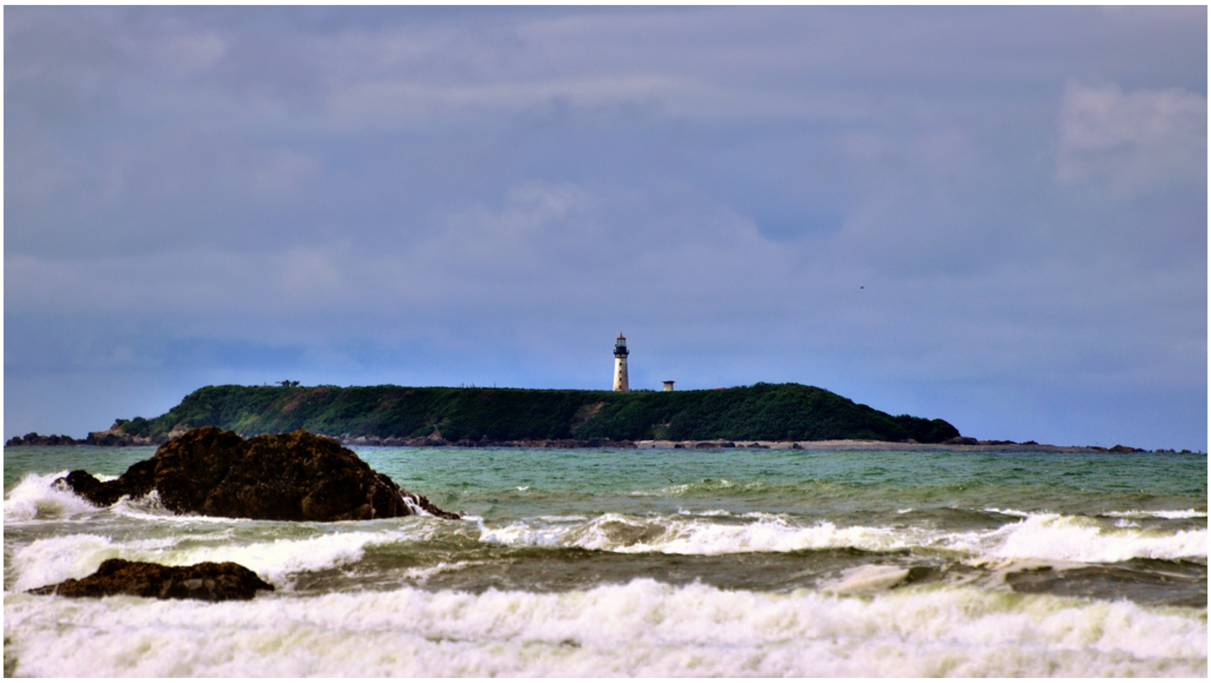

Figure 2.3: Example 2

Chapter 3

Neural Nets

3.1 Introduction and Parameters

In neural nets we have three layers. The Input, Hidden and Output layers. We generate the hidden nodes based on command line parameter. We feed the input through the layers(feed forward). Check the result with ground truth and propagate the errors back and learn the weights accordingly. Then re-train the model again. Use the learnt weights to test the test files.

3.1.1 Approach:

The primary parameter we try to vary in this method are the number of hidden nodes and the number of iterations for which we train the data. We have implemented both classic and stochastic neural nets.

The activation function we have used is leaky Rectified Linear Unit. The function is,

$$g(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & else \end{cases}$$

This gives us the g' as

$$g'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0.01 & else \end{cases}$$

3.2 Performance Results

The results in (Table 3.1) we have used the stochastic back propagation with a batch of 100 random images with a iteration of 250.

Table 3.1: Neural Nets

Hidden Nodes	Performance (%)	Time (m)
150	64.89	16.52
160	60.33	17.55
170	65.64	18.59
180	65.32	19.52
190	65.00	21.20
200	63.30	22.5
210	62.46	23.28
220	64.89	24.40
230	65.85	25.46
240	63.52	26.23
250	65.45	28.8

Table 3.2: Neural Nets - 2

Input Subset	Performance (%)	Time
5000	63.94	8.44
10000	64.58	8.4
15000	62.77	9.1
20000	62.99	8.54

Table 3.3: Neural Nets - 3

Iterations	Performance (%)	Time (m)
1	44.64	0.34
2	54.33	0.43
3	52.38	0.46
4	60.83	0.49
5	61.71	0.57
6	62.46	1.02
7	63.73	1.08
8	64.24	1.10
9	64.91	1.19
10	63.50	1.18

3.3 EXAMPLES

The first image was classified as 270 (truth was its 180) and the second image was classified as 90 (actually 180)

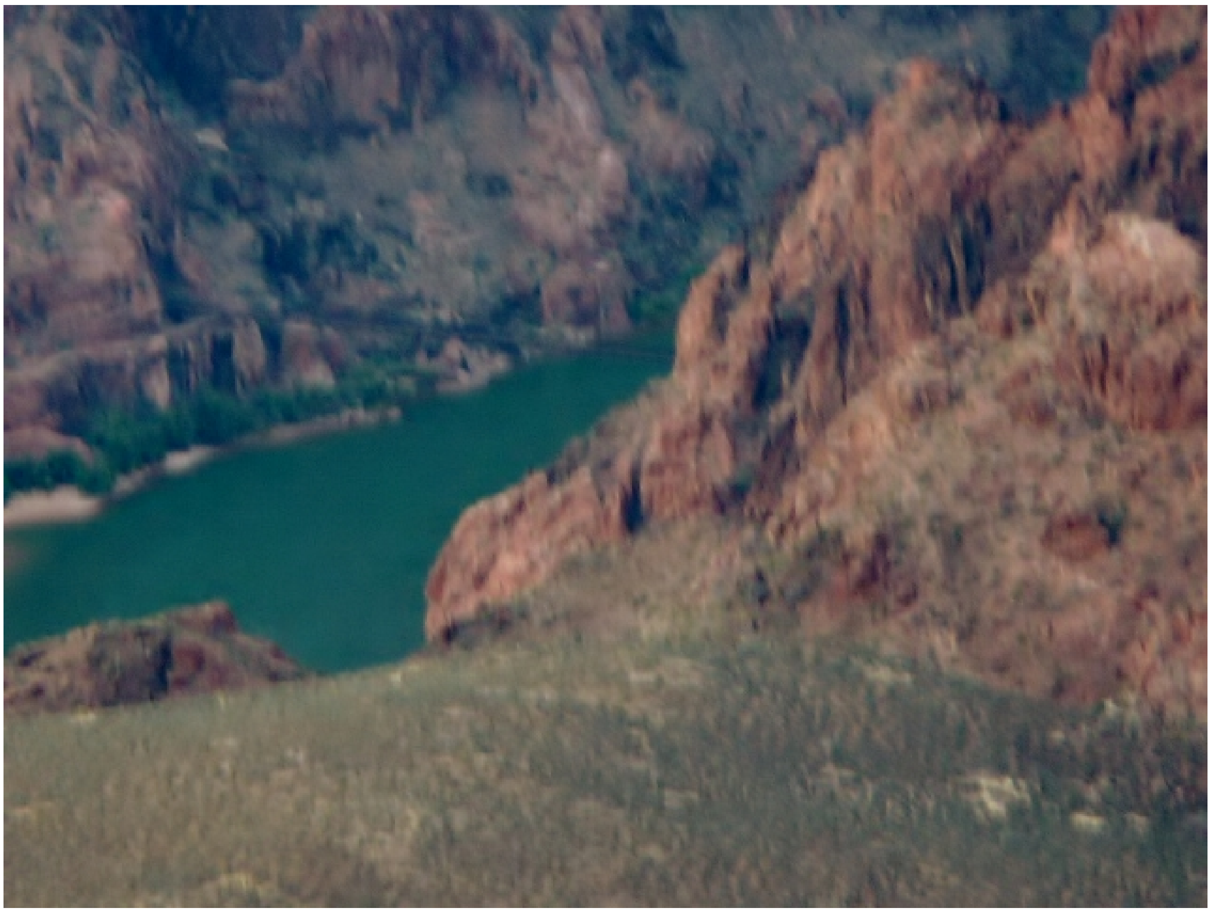

Figure 3.1: Example 1

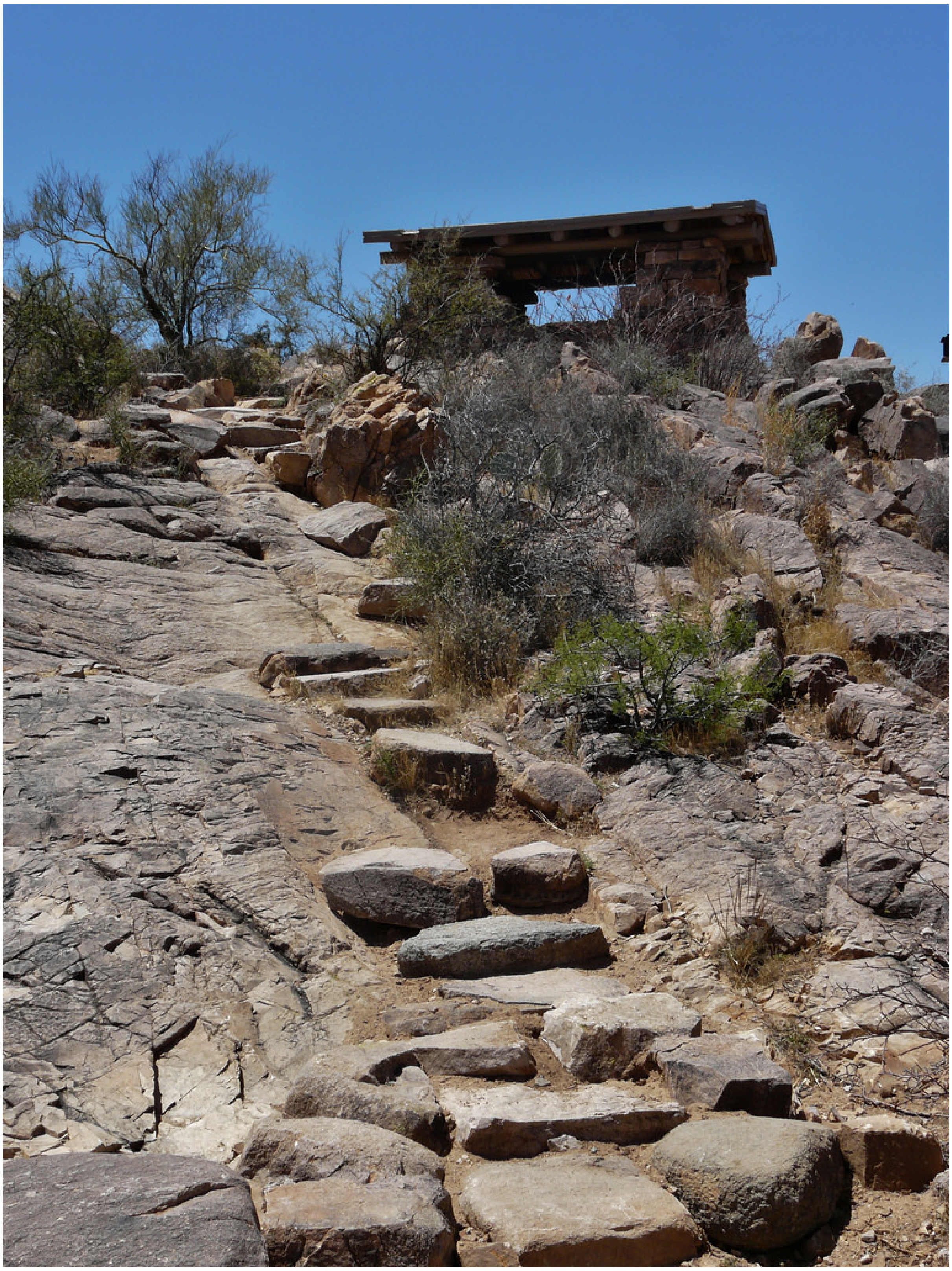

Figure 3.2: Example 2

Chapter 4

Conclusion and 'Best' Model

We find that nearest neighbor outperforms both the adaboost and the neural net model by around 5% to 10%. This better performance can be considered as a good supporting evidence for KIS(Keep it Simple) principle i.e. If you want to find orientation of an image find the image closest to it in terms of color composition.

To simplify grading, we run the simplest nearest neigbor model for 'best' option. This would be with $k = 1$ and typically takes 20 - 30 minutes